

A large, glowing orange sphere with a blue ring, set against a black background. The sphere has a soft, ethereal glow and a thin blue ring around its equator. The background is solid black.

Archivos

Mauricio Avilés

# Archivos

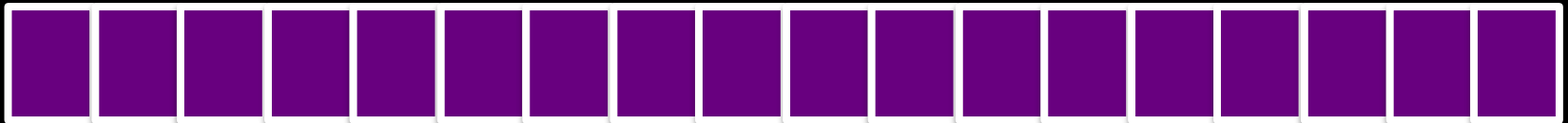
- Principios
  - Archivos
  - Directorios
  - Operaciones
- Archivos en C++
- Archivos secuenciales
- Archivos de acceso aleatorio
- Archivos indexados
  - Indexación lineal
  - Indexación con árboles
  - B-Tree, B+Tree, B\*Tree

# Principios

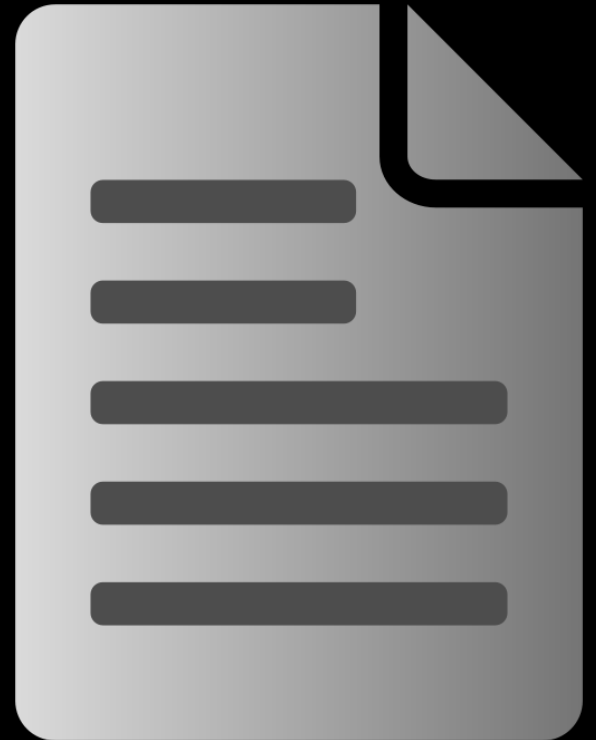
- Un **archivo** es un conjunto de datos
- Colección de archivos organizados de forma que:
  - El SO los puede **administrar**
  - Un programa pueda **acceder** los archivos
- Rol del sistema operativo:
  - **Administrar** y **controlar** el sistema de archivos jerárquico y los archivos mismos

# Archivo

- Conjunto de datos que tiene alguna relación y es **persistente**
- Colección de registros **lógicamente** ordenados

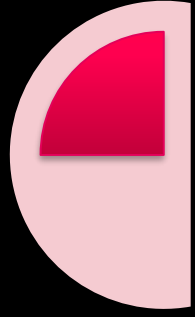


- Son una **unidad** o entidad
- Por eso deben tener un **nombre**, para poder referirse a ellos
- El SO provee el mecanismo para poder nombrar archivos
- El usuario asigna los nombres a los archivos



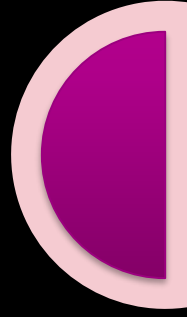
La mayoría de los archivos tienen una **organización** predeterminada

Los archivos **agrupan** sus registros con algún **objetivo**, esto les da valor semántico



## Archivo plano

Serie de bytes o caracteres almacenados uno después de otro



## Archivo estructurado

Archivo con una estructura determinada, como registros con algún formato (video, audio, documentos, etc.)

Almacenan datos independientemente de los programas que se ejecutan

Almacena gran volumen comparado con la memoria principal

La cantidad de datos está limitada por la capacidad del dispositivo de almacenamiento



# Sistema de Archivos

- Es la forma en que el SO hace que los datos estén **disponibles**
- Controla el **acceso** a los archivos
- Los SO manejan carpetas, directorio, unidades, etc.
- Algunas veces están orientados a satisfacer ciertas **necesidades específicas**
- File System (FS)

- Responsabilidades

- Mantener un **directorio** para identificar y localizar la información
- Establecer **rut**as para los flujos entre memoria principal y los dispositivos de almacenamiento
- Administrar eficientemente al **CPU** para que no desperdicie tiempo en operaciones de I/O
- Mantener la **integridad** de los datos

- El sistema de archivos existe a través de un medio de almacenamiento
- El SO interactúa con el controlador de los dispositivos de almacenamiento para escribir datos dentro del medio
  - Disco duro
  - Memoria USB
  - CD
  - DVD
  - Etc.

- Existen muchos **tipos** de sistemas de archivos
  - EXT2, EXT3 (Linux)
  - NTFS (Windows)
  - FAT, FAT32 (DOS, Windows)
  - CDFS (Discos Compactos)
  - HPFS (OS/2)
  - Otros

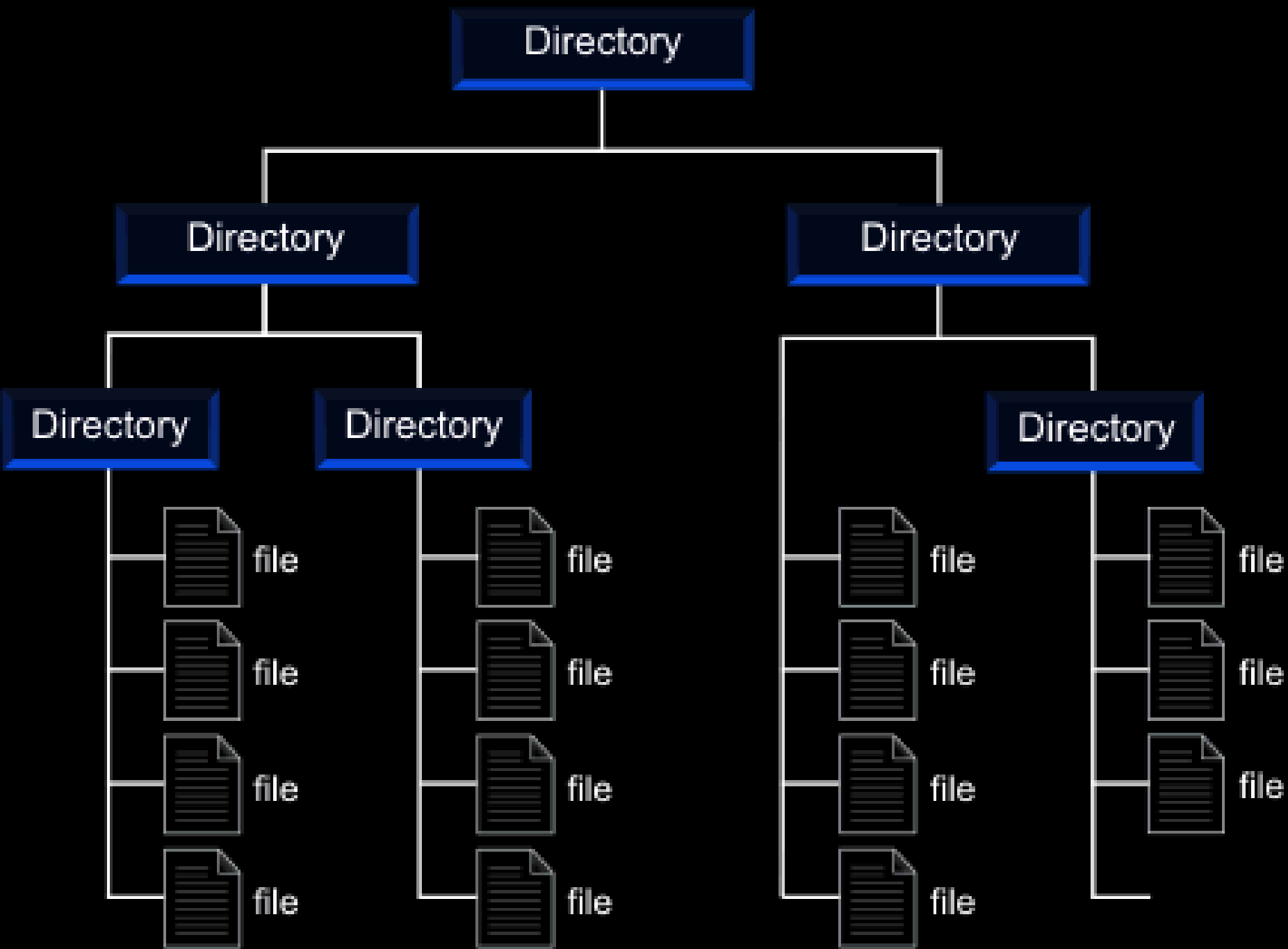
# API (Application Programming Interface)

- El SO **oculta particularidades** de las operaciones de I/O con los archivos
- Ofrece a los **programadores** una forma abstracta y limpia para manipularlos

# Organización jerárquica

- La mayoría utiliza el concepto de **directorio** como **agrupación** de archivos
- Directorios y archivos se organizan **jerárquicamente** como un **árbol**
- Todo archivo tiene una **ruta** y un **nombre**
- La ruta es la **lista de directorios** que deben ser recorridos desde la raíz hasta el elemento

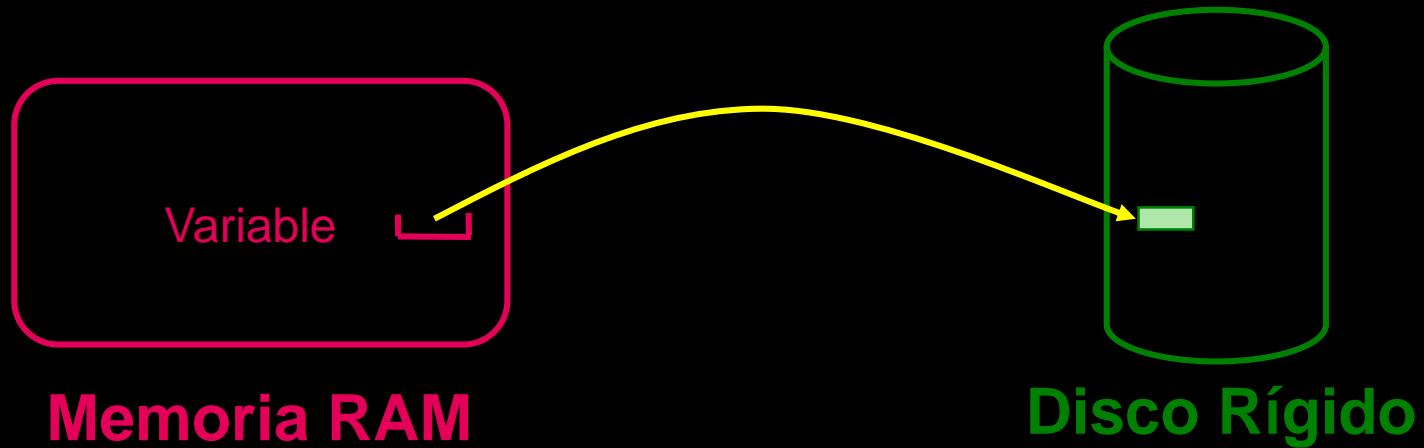
C:\Games\Q3A\Baseq3\pak0.pk3



# Acceso a un archivo

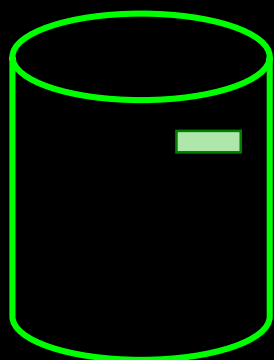
- Para escribir o leer de un archivo, primero debe **abrirse**
- Se revisan los **permisos** de acceso
- Si es permitido, se retorna un número entero que es el **descriptor** o **manejador** del archivo que se utiliza para realizar las siguientes **operaciones** (*file descriptor, file handle*)



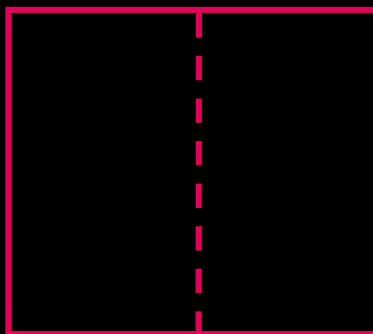
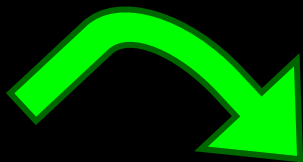


Esquema de la conexión lógica a un archivo ubicado en disco generada luego de la apertura del mismo. Notar que el archivo sigue estando ubicado físicamente en el disco.

**Entrada** a procesamiento

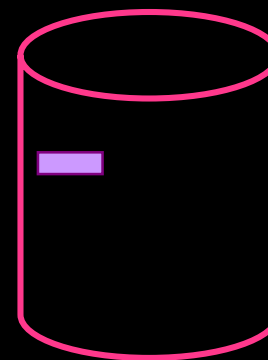


Archivo en Disco



Programa

**Salida** a disco



Archivo en Disco

Lectura

Escritura

# Operaciones

- Aunque la forma en que se manipulan los archivos depende del SO y del FS, existen operaciones comunes
  - Crear un archivo con un nombre
  - Cambiar atributos de un archivo
  - Abrir un archivo para usarlo
  - Leer o actualizar los contenidos
  - Guardar los cambios al dispositivo de almacenamiento
  - Cerrar el archivo y perder el acceso

# Archivos en C++

- Clases proveídas
  - `ofstream`: escribir en archivos
  - `ifstream`: leer de archivos
  - `fstream`: leer y escribir en archivos
- Estas clases se pueden utilizar de la misma forma que `cin` y `cout`

# Abrir un archivo

- Asocia al objeto con un archivo real

```
open( filename, mode);
```

ios::in          Abrir para input

ios::out        Abrir para output

ios::binary     Modo binario

ios::ate        Pone la posición inicial al final del archivo

ios::app        Todas las operaciones se llevan a cabo al final del archivo.

ios::trunc      Si el archivo abierto para output ya existe, entonces lo borra y hace uno nuevo.

```
ofstream myfile; myfile.open ("example.bin",  
ios::out | ios::app | ios::binary);
```

- El método `open()` de las clases tiene un modo por defecto:

Clase	Modo por defecto
<code>ofstream</code>	<code>ios::out</code>
<code>ifstream</code>	<code>ios::in</code>
<code>fstream</code>	<code>ios::in   ios::out</code>

- Se incluye constructor con los parámetros de `open`
- Ver si un archivo se abrió satisfactoriamente

```
if (myfile.is_open()) {  
    /* ok, proceed with output */  
}
```

# Cerrar un archivo

- Cuando ya se terminó de leer o escribir de un archivo, debe cerrarse para que esté **disponible** de nuevo
- Luego de cerrar un archivo, el objeto stream se puede usar para abrir otro

```
myfile.close();
```

- Ejemplo de escritura en archivo de texto

# Archivos de texto

- No incluyen el modo `ios::binary`
- Diseñados para almacenar texto
- Ejemplo de lectura de archivo



# Banderas de estado

- `bad()`
  - Retorna verdadero si una operación de lectura o escritura falla. Como escribir en un archivo cerrado, o espacio insuficiente.
- `fail()`
  - Igual que `bad()` pero también retorna verdadero cuando ocurre un error de formato, como leer una letra cuando se espera un número.
- `eof()`
  - Verdadero si un archivo abierto para lectura ya llegó al final.
- `good()`
  - Retorna verdadero cuando `bad()`, `fail()` y `eof()` retornan falso.

# Archivos binarios

- Almacenan los datos en binario, no en formato de texto
- Pueden leer y escribir

```
write ( memory_block, size );  
read  ( memory_block, size );
```

- Recibe un puntero a un arreglo de bytes de donde se van a leer los datos para escribirlos en el archivo, y un entero que indica la cantidad de bytes que se van a leer

```
ifstream::pos_type size;
char * memblock;

int main () {
    ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
    if (file.is_open())
    {
        size = file.tellg();
        memblock = new char [size];
        file.seekg (0, ios::beg);
        file.read (memblock, size);
        file.close();
        cout << "the complete file content is in memory";
        delete[] memblock;
    }
    else cout << "Unable to open file";
    return 0;
}
```

```
ifstream::pos_type size;  
char * memblock;
```

Puntero al final del archivo



```
int main () {  
    ifstream file ("example.bin", ios::in|ios::binary|ios::ate);  
    if (file.is_open())  
    {  
        size = file.tellg();  
        memblock = new char [size];  
        file.seekg (0, ios::beg);  
        file.read (memblock, size);  
        file.close();  
        cout << "the complete file content is in memory";  
        delete[] memblock;  
    }  
    else cout << "Unable to open file";  
    return 0;  
}
```

```
ifstream::pos_type size;
char * memblock;

int main () {
    ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
    if (file.is_open())
    {
        size = file.tellg();
        memblock = new char [size];
        file.seekg (0, ios::beg);
        file.read (memblock, size);
        file.close();
        cout << "the complete file content is in memory";
        delete[] memblock;
    }
    else cout << "Unable to open file";
    return 0;
}
```

**Obtiene el tamaño del archivo**

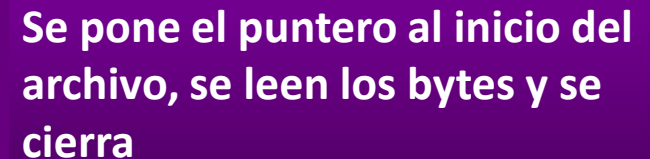
```
ifstream::pos_type size;
char * memblock;

int main () {
    ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
    if (file.is_open())
    {
        size = file.tellg();
        memblock = new char [size];
        file.seekg (0, ios::beg);
        file.read (memblock, size);
        file.close();
        cout << "the complete file content is in memory";
        delete[] memblock;
    }
    else cout << "Unable to open file";
    return 0;
}
```

**Solicitar memoria para datos**

```
ifstream::pos_type size;
char * memblock;

int main () {
    ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
    if (file.is_open())
    {
        size = file.tellg();
        memblock = new char [size];
        file.seekg (0, ios::beg);
        file.read (memblock, size);
        file.close();
        cout << "the complete file content is in memory";
        delete[] memblock;
    }
    else cout << "Unable to open file";
    return 0;
}
```



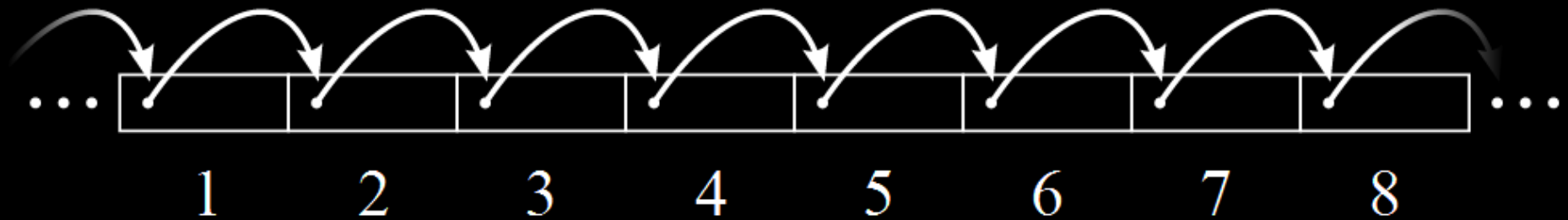
**Se pone el puntero al inicio del archivo, se leen los bytes y se cierra**

```
ifstream::pos_type size;
char * memblock;

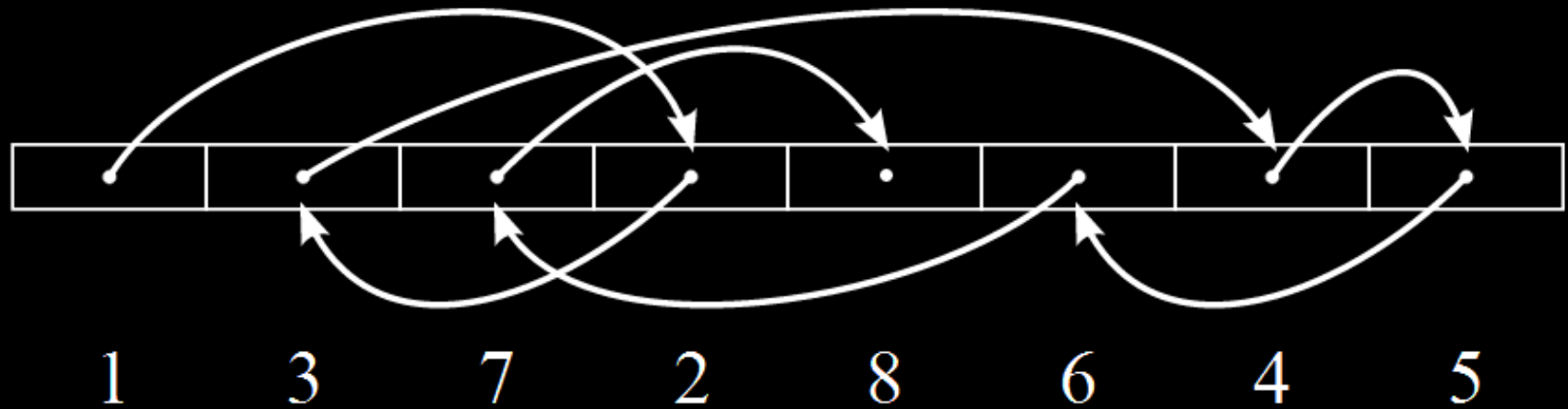
int main () {
    ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
    if (file.is_open())
    {
        size = file.tellg();
        memblock = new char [size];
        file.seekg (0, ios::beg);
        file.read (memblock, size);
        file.close();
        cout << "the complete file content is in memory";
        delete[] memblock;
    }
    else cout << "Unable to open file";
    return 0;
}
```



# Sequential access



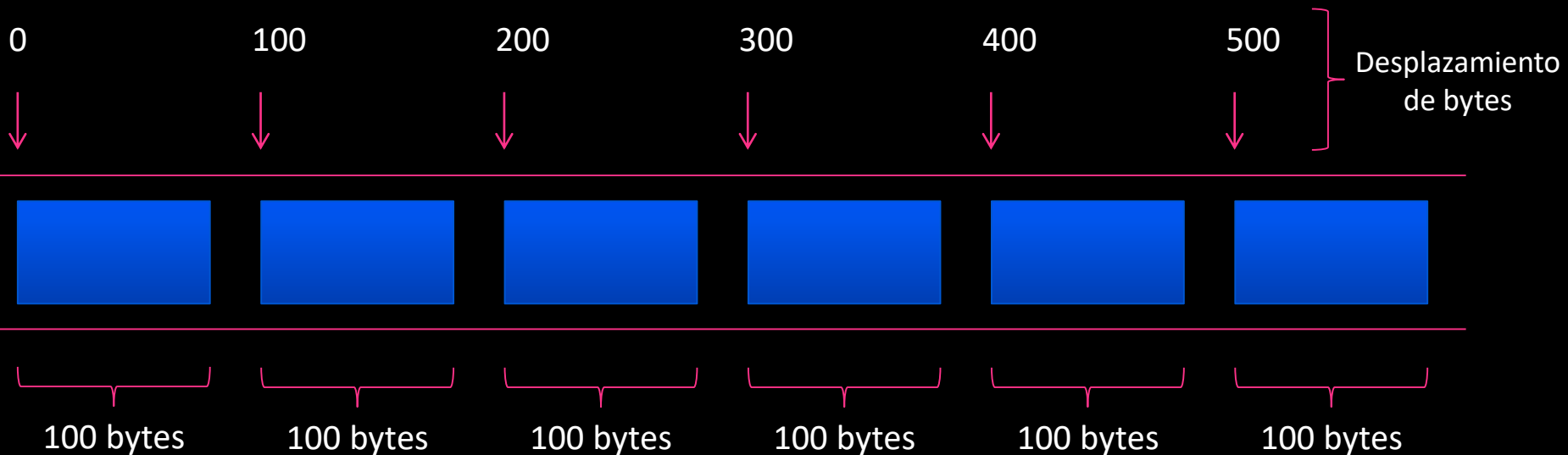
# Random access



# Archivos de acceso aleatorio

- Los archivos de acceso secuencial son inapropiados para acceso instantáneo
- Los archivos de acceso aleatorio permiten acceder **cualquier parte** del archivo sin tener que leer el resto
- C++ no impone estructuras de archivos, por lo que hay que **implementarlo** si se quiere usar

- Lo más sencillo: dividir el archivo en registros de **longitud fija**
- Es fácil calcular la ubicación exacta de cualquier registro respecto del inicio del archivo
  - Función del tamaño del registro y la clave del registro




# Ejemplo

1. Clase base para escribir registros
2. Programa que inicializa el archivo
3. Programa que escribe registros
4. Programa que lee e imprime los registros

# Ejemplo

1. Clase base para escribir registros
2. Programa que inicializa el archivo
3. Programa que escribe registros
4. Programa que lee e imprime los registros

```
class DatosCliente {  
public:  
    DatosCliente( int = 0, string = "", string = "", double  
= 0.0 );  
    void setNumeroCuenta( int );  
    int getNumeroCuenta() const;  
    void setApellidoPaterno( string );  
    string getApellidoPaterno() const;  
    void setPrimerNombre( string );  
    string getPrimerNombre() const;  
    void setSaldo( double );  
    double getSaldo() const;  
  
private:  
    int numeroCuenta;  
    char apellidoPaterno[ 15 ];  
    char primerNombre[ 10 ];  
    double saldo;  
};
```



**Clase que representa la información de crédito de un cliente**

# Ejemplo

1. Clase base para escribir registros
2. Programa que inicializa el archivo
3. Programa que escribe registros
4. Programa que lee e imprime los registros

```
int main() {
    ofstream creditoSalida( "credito.dat",
                           ios::out | ios::binary );
    if ( !creditoSalida.is_open() )
    {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    DatosCliente clienteEnBlanco;
    for ( int i = 0; i < 100; i++ )
        creditoSalida.write(
            reinterpret_cast<const char *>( &clienteEnBlanco ),
            sizeof( DatosCliente ) );
    return 0;
}
```



```
int main() {  
    ofstream creditoSalida( "credito.dat",  
                            ios::out | ios::binary );  
    if ( !creditoSalida.is_open() )  
    {  
        cerr << "No se pudo abrir el archivo." << endl;  
        exit( 1 );  
    }  
    DatosCliente clienteEnBlanco;  
    for ( int i = 0; i < 100; i++ )  
        creditoSalida.write(  
            reinterpret_cast<const char *>( &clienteEnBlanco ),  
            sizeof( DatosCliente ) );  
    return 0;  
}
```

Se crea el archivo de salida en binario


```
int main() {  
    ofstream creditoSalida( "credito.dat",  
                            ios::out | ios::binary );  
    if ( !creditoSalida.is_open() )  
    {  
        cerr << "No se pudo abrir el archivo." << endl;  
        exit( 1 );  
    }  
    DatosCliente clienteEnBlanco;  
    for ( int i = 0; i < 100; i++ )  
        creditoSalida.write(  
            reinterpret_cast<const char *>( &clienteEnBlanco ),  
            sizeof( DatosCliente ) );  
    return 0;  
}
```

**Chequear si se abrió bien el archivo**

```
int main() {
    ofstream creditoSalida( "credito.dat",
                           ios::out | ios::binary );
    if ( !creditoSalida.is_open() )
    {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    DatosCliente clienteEnBlanco;
    for ( int i = 0; i < 100; i++ )
        creditoSalida.write(
            reinterpret_cast<const char *>( &clienteEnBlanco ),
            sizeof( DatosCliente ) );
    return 0;
}
```

**Escribir 100 registros en blanco**

```
int main() {  
    ofstream creditoSalida( "credito.dat",  
                            ios::out | ios::binary );  
    if ( !creditoSalida.is_open() )  
    {  
        cerr << "No se pudo abrir el archivo." << endl;  
        exit( 1 );  
    }  
    DatosCliente clienteEnBlanco;  
    for ( int i = 0; i < 100; i++ )  
        creditoSalida.write(  
            reinterpret_cast<const char *>( &clienteEnBlanco ),  
            sizeof( DatosCliente ) );  
    return 0;  
}
```



**Necesario para interpretar el  
objeto como una tira de char**

```
int main() {
    ofstream creditoSalida( "credito.dat",
                           ios::out | ios::binary );
    if ( !creditoSalida.is_open() )
    {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    DatosCliente clienteEnBlanco;
    for ( int i = 0; i < 100; i++ )
        creditoSalida.write(
            reinterpret_cast<const char *>( &clienteEnBlanco ),
            sizeof( DatosCliente ) );
    return 0;
}
```



Cantidad de bytes a escribir

# Ejemplo

1. Clase base para escribir registros
2. Programa que inicializa el archivo
3. Programa que escribe registros
4. Programa que lee e imprime los registros

```
int main()
{
    int numeroCuenta;
    char apellidoPaterno[ 15 ];
    char primerNombre[ 10 ];
    double saldo;

    fstream creditoSalida( "credito.dat", ios::in | ios::out |
ios::binary );
    if ( !creditoSalida.is_open() ) {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    cout << "Escriba el numero de cuenta (de 1 a 100, 0 para
terminar la entrada)\n? ";
    DatosCliente cliente;
    cin >> numeroCuenta;
```

```
int main()
```

```
{
```

```
    int numeroCuenta;
```

```
    char apellidoPaterno[ 15 ];
```

```
    char primerNombre[ 10 ];
```

```
    double saldo;
```

```
    fstream creditoSalida( "credito.dat", ios::in | ios::out |  
ios::binary );
```

```
    if ( !creditoSalida.is_open() ) {
```

```
        cerr << "No se pudo abrir el archivo." << endl;
```

```
        exit( 1 );
```

```
    }
```

```
    cout << "Escriba el numero de cuenta (de 1 a 100, 0 para  
terminar la entrada)\n? ";
```

```
    DatosCliente cliente;
```

```
    cin >> numeroCuenta;
```

**Variables a utilizar para leer los  
datos del cliente**



```
int main()
{
    int numeroCuenta;
    char apellidoPaterno[ 15 ];
    char primerNombre[ 10 ];
    double saldo;


    fstream creditoSalida( "credito.dat", ios::in | ios::out |
ios::binary );
    if ( !creditoSalida.is_open() ) {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    cout << "Escriba el numero de cuenta (de 1 a 100, 0 para
terminar la entrada)\n? ";
    DatosCliente cliente;
    cin >> numeroCuenta;
```



Se abre el archivo como  
entrada/salida

```
int main()
{
    int numeroCuenta;
    char apellidoPaterno[ 15 ];
    char primerNombre[ 10 ];
    double saldo;

    fstream creditoSalida( "credito.dat", ios::in | ios::out |
ios::binary );
    if ( !creditoSalida.is_open() ) {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    cout << "Escriba el numero de cuenta (de 1 a 100, 0 para
terminar la entrada)\n? ";
    DatosCliente cliente;
    cin >> numeroCuenta;
```



Leer el número de registro que se  
va a sobrescribir

```
while ( numeroCuenta > 0 && numeroCuenta <= 100 ) {  
    cout << "Escriba apellido paterno, primer nombre y  
saldo\n? ";  
    cin >> setw( 15 ) >> apellidoPaterno;  
    cin >> setw( 10 ) >> primerNombre;  
    cin >> saldo;  
    cliente.setNumeroCuenta( numeroCuenta );  
    cliente.setApellidoPaterno( apellidoPaterno );  
    cliente.setPrimerNombre( primerNombre );  
    cliente.setSaldo( saldo );  
    creditoSalida.seekp( ( cliente.getNumeroCuenta() - 1 ) *  
sizeof( DatosCliente ) );  
    creditoSalida.write( reinterpret_cast< const char * >(  
&cliente ), sizeof( DatosCliente ) );  
    cout << "Escriba el numero de cuenta\n? ";  
    cin >> numeroCuenta;  
}  
return 0;  
}
```

**Se leen los datos a escribir y se  
copian al objeto cliente**

```
while ( numeroCuenta > 0 && numeroCuenta <= 100 ) {  
    cout << "Escriba apellido paterno, primer nombre y  
saldo\n? ";  
    cin >> setw( 15 ) >> apellidoPaterno;  
    cin >> setw( 10 ) >> primerNombre;  
    cin >> saldo;  
    cliente.setNumeroCuenta( numeroCuenta );  
    cliente.setApellidoPaterno( apellidoPaterno );  
    cliente.setPrimerNombre( primerNombre );  
    cliente.setSaldo( saldo );  
    creditoSalida.seekp( ( cliente.getNumeroCuenta() - 1 ) *  
sizeof( DatosCliente ) );  
    creditoSalida.write( reinterpret_cast< const char * >(  
&cliente ), sizeof( DatosCliente ) );  
    cout << "Escriba el numero de cuenta\n? ";  
    cin >> numeroCuenta;  
}  
return 0;  
}
```



**Buscar posición donde se va a  
escribir**

```
while ( numeroCuenta > 0 && numeroCuenta <= 100 ) {  
    cout << "Escriba apellido paterno, primer nombre y  
saldo\n? ";  
    cin >> setw( 15 ) >> apellidoPaterno;  
    cin >> setw( 10 ) >> primerNombre;  
    cin >> saldo;  
    cliente.setNumeroCuenta( numeroCuenta );  
    cliente.setApellidoPaterno( apellidoPaterno );  
    cliente.setPrimerNombre( primerNombre );  
    cliente.setSaldo( saldo );  
    creditoSalida.seekp( ( cliente.getNumeroCuenta() - 1 ) *  
sizeof( DatosCliente ) );  
    creditoSalida.write( reinterpret_cast< const char * >(  
&cliente ), sizeof( DatosCliente ) );  
    cout << "Escriba el numero de cuenta\n? ";  
    cin >> numeroCuenta;  
}  
return 0;  
}
```




**Escribir los bytes del objeto con los  
datos del cliente**

# Ejemplo

1. Clase base para escribir registros
2. Programa que inicializa el archivo
3. Programa que escribe registros
4. Programa que lee e imprime los registros

```
int main(){
    ifstream creditoEntrada( "credito.dat", ios::in |
ios::binary );
    if ( !creditoEntrada.is_open() ) {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    DatosCliente cliente;
    creditoEntrada.read( reinterpret_cast< char * >( &cliente ),
        sizeof( DatosCliente ) );
    while ( creditoEntrada.good() ) {
        if ( cliente.getNumeroCuenta() != 0 )
            imprimirLinea( cout, cliente );
        creditoEntrada.read( reinterpret_cast< char * >( &cliente
),
            sizeof( DatosCliente ) );
    }
    return 0;
}}
```

```
int main(){
    ifstream creditoEntrada( "credito.dat", ios::in |
ios::binary );
    if ( !creditoEntrada.is_open() ) {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    DatosCliente cliente;
    creditoEntrada.read( reinterpret_cast< char * >( &cliente ),
        sizeof( DatosCliente ) );
    while ( creditoEntrada.good() ) {
        if ( cliente.getNumeroCuenta() != 0 )
            imprimirLinea( cout, cliente );
        creditoEntrada.read( reinterpret_cast< char * >( &cliente
),
            sizeof( DatosCliente ) );
    }
    return 0;
}}
```



Se abre como archivo de entrada



```
int main(){
    ifstream creditoEntrada( "credito.dat", ios::in |
ios::binary );
    if ( !creditoEntrada.is_open() ) {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    DatosCliente cliente;
    creditoEntrada.read( reinterpret_cast< char * >( &cliente ),
        sizeof( DatosCliente ) );
    while ( creditoEntrada.good() ) {
        if ( cliente.getNumeroCuenta() != 0 )
            imprimirLinea( cout, cliente );
        creditoEntrada.read( reinterpret_cast< char * >( &cliente
),
            sizeof( DatosCliente ) );
    }
    return 0;
}}
```

Leer los bytes del tamaño del  
objeto



```
int main(){
    ifstream creditoEntrada( "credito.dat", ios::in |
ios::binary );
    if ( !creditoEntrada.is_open() ) {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    DatosCliente cliente;
    creditoEntrada.read( reinterpret_cast< char * >( &cliente ),
        sizeof( DatosCliente ) );
    while ( creditoEntrada.good() ) {
        if ( cliente.getNumeroCuenta() != 0 )
            imprimirLinea( cout, cliente );
        creditoEntrada.read( reinterpret_cast< char * >( &cliente
),
            sizeof( DatosCliente ) );
    }
    return 0;
}}
```

Si el número de registro es  
diferente de cero, se imprime

```
int main(){
    ifstream creditoEntrada( "credito.dat", ios::in |
ios::binary );
    if ( !creditoEntrada.is_open() ) {
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    }
    DatosCliente cliente;
    creditoEntrada.read( reinterpret_cast< char * >( &cliente ),
        sizeof( DatosCliente ) );
    while ( creditoEntrada.good() ) {
        if ( cliente.getNumeroCuenta() != 0 )
            imprimirLinea( cout, cliente );
        creditoEntrada.read( reinterpret_cast< char * >( &cliente
),
            sizeof( DatosCliente ) );
    }
    return 0;
}}
```



Leer siguiente hasta EOF

# Archivos indizados

- En un archivo secuencial los registros están en el mismo **orden** en que se **insertan**
- Esto no permite una **búsqueda** eficiente
- Lo natural sería **ordenar** los registros pero esto es costoso
- También se requiere buscar por diferentes **criterios** de búsqueda

2



2	8	3	5	10
---	---	---	---	----







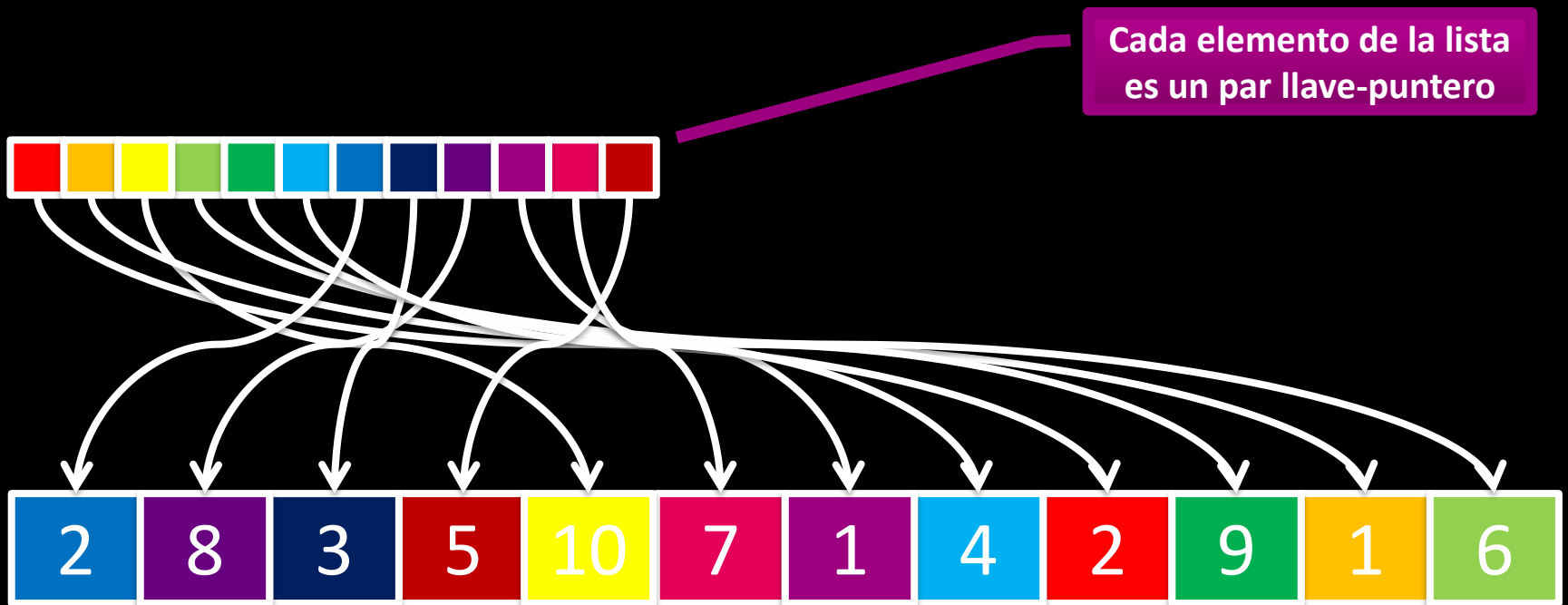


# Índice



Se guardan las llaves  
necesarias, ordenadas

2	8	3	5	10	7	1	4	2	9	1	6
---	---	---	---	----	---	---	---	---	---	---	---

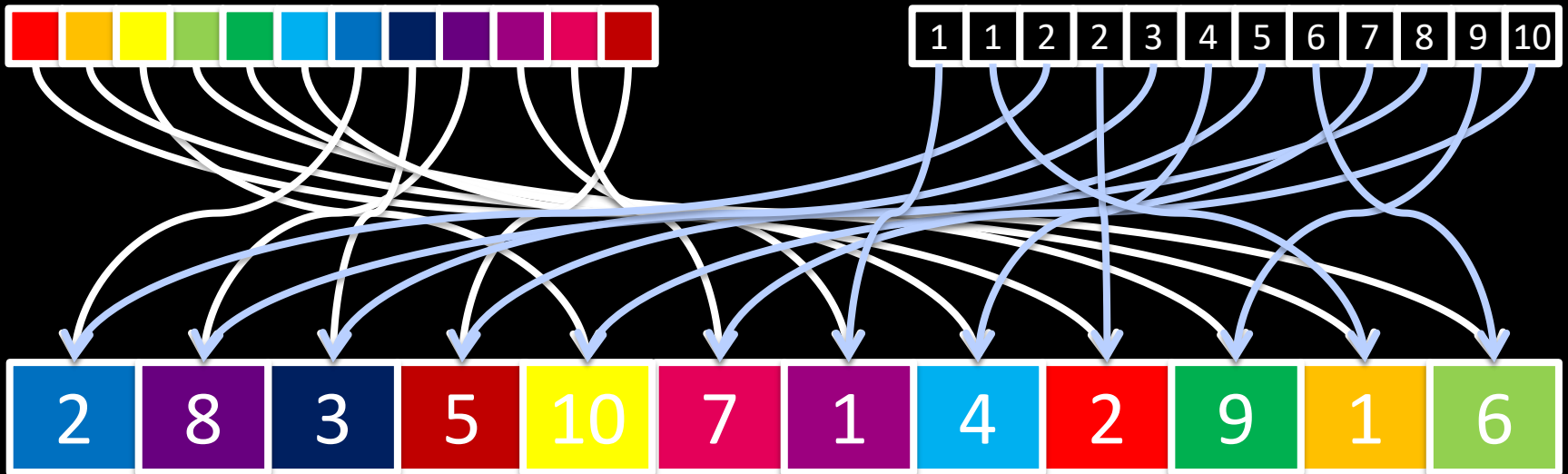




También puede ser necesario  
ordenar por algún otro criterio

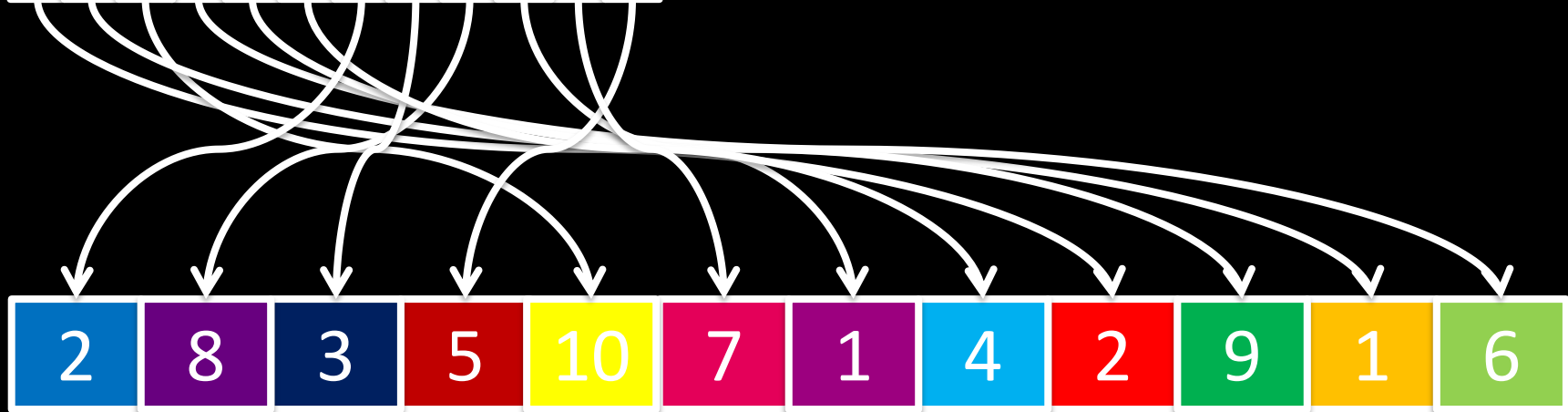


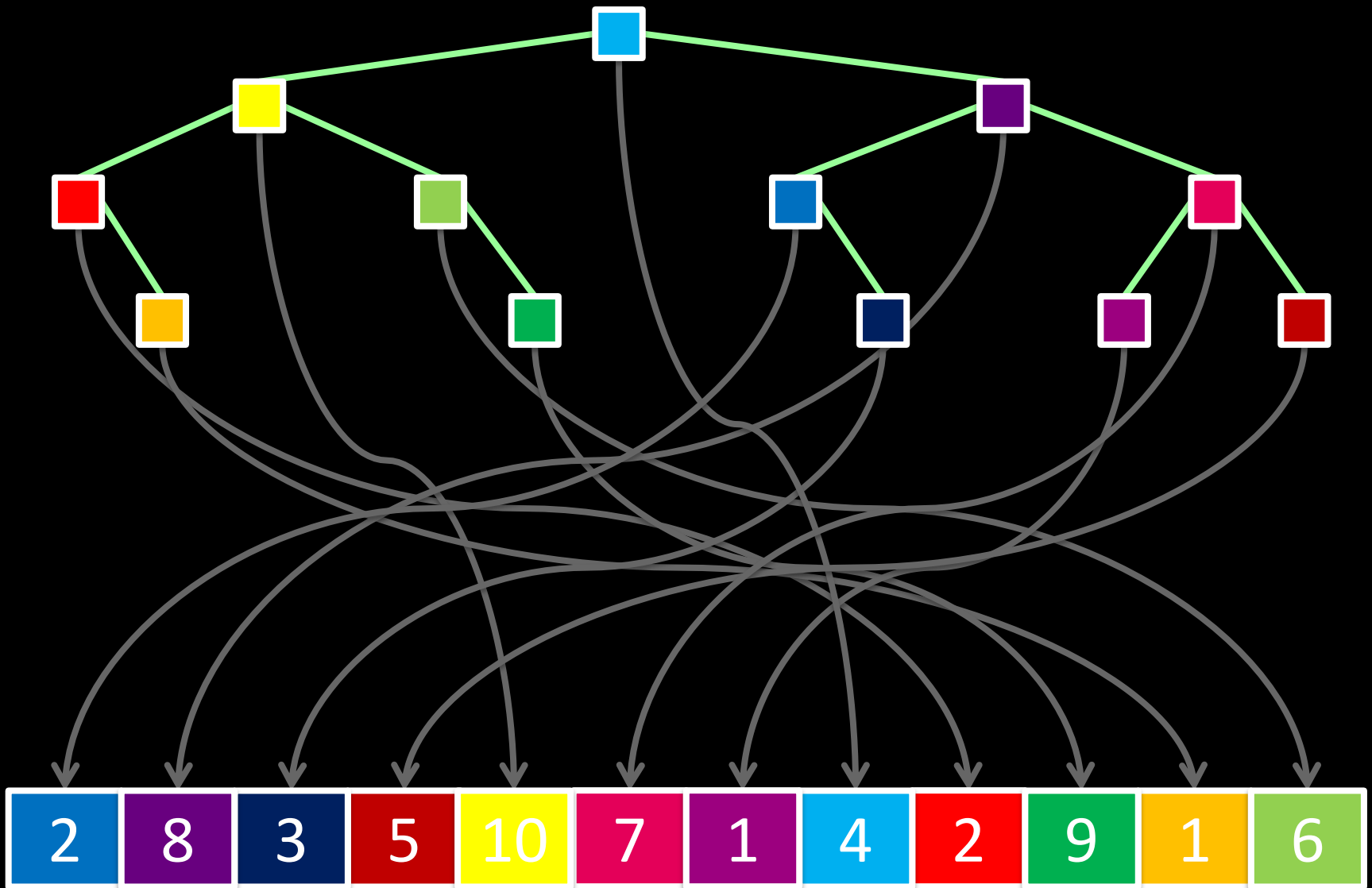
# Índice primario, índice secundario





El índice secundario normalmente apunta al primario, no hacia los registros en disco





La estructura del índice puede ser cualquiera que permita guardar pares llave-valor



# Indexación lineal

- Un índice lineal es una **secuencia** (lista) de pares **llave-puntero**
- Las llaves se encuentran **ordenadas**
- Los punteros
  1. Posición del registro en **disco**
  2. Posición de la llave primaria en el **índice primario**
  3. **Valor** de la llave primaria

# Posición del registro en disco

Persona = 120 bytes

- int cedula; (PK)
- char nombre[30];
- char apellido[30];
- int diaNac;
- int mesNac;
- int anoNac;
- char domElec[40];
- char sexo;

## Índice

102340255 480	205551345 240	306560229 0	410220545 120	701540234 360
------------------	------------------	----------------	------------------	------------------

306560229 Pablo Hernández 4 2 1984 Oreamuno, Cartago M	410220545 Julio Rojas 23 3 1988 Flores, Heredia M	205551345 Marta Sánchez 29 2 1992 Atenas, Alajuela F	701540234 Jaime Vindas 25 10 1989 Pococí, Limón M	102340255 María Solano 12 10 1993 Montes de Oca, San José F
---	--	---	--	--

0

120

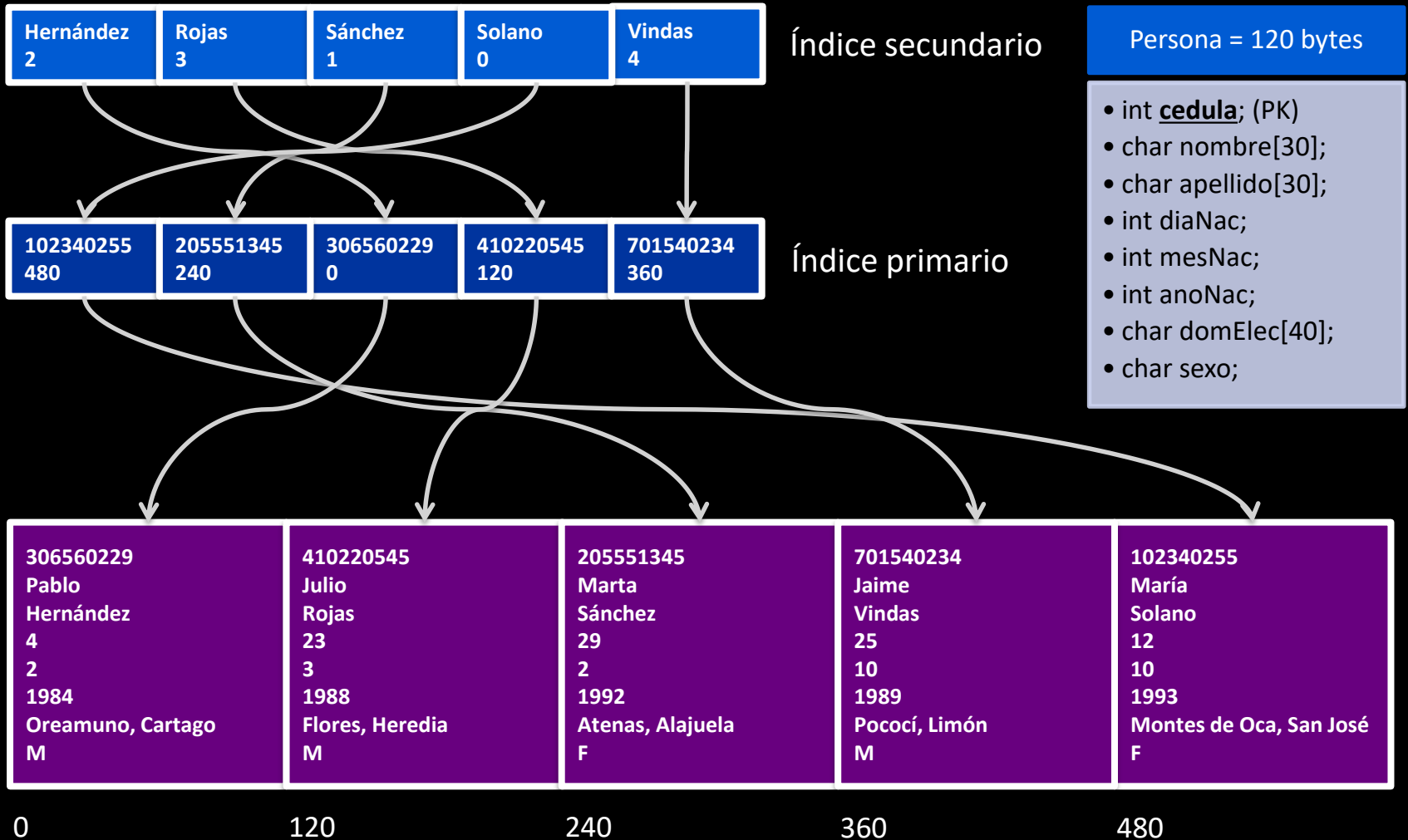
240

360

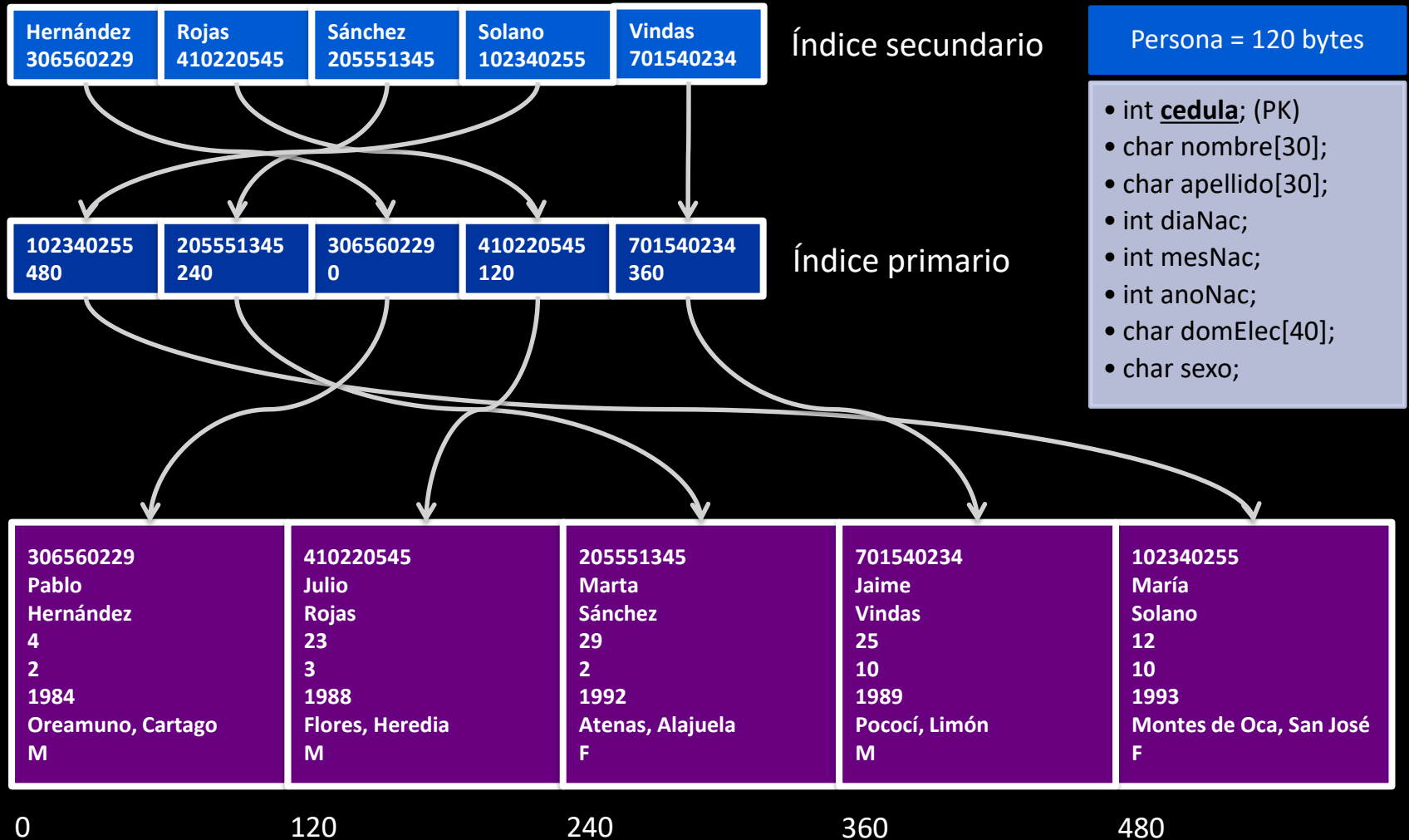
480

## Archivo en disco

# Posición de la llave primaria en el índice primario



# Valor de la llave primaria



- Cómo los índices están ordenados es posible utilizar **búsqueda binaria** sobre ellos
- Los índices también deben almacenarse en **disco**, en un archivo aparte
- Al utilizarlo, lo ideal es cargarlo en **memoria**
- **Actualizaciones** en el índice son caras, deben moverse muchos elementos

- En algunas situaciones la cantidad de registros puede ser tan grande que incluso el **índice principal no se puede** cargar en **memoria**
- Este problema puede solucionarse creando un **segundo nivel** para el índice
- El índice de segundo nivel indica el **bloque** del índice principal donde se encuentra un **rango** de llaves

1	2003	5894	10528
---	------	------	-------

Second Level Index

1	2001	2003	5688	5894	9942	10528	10984
---	------	------	------	------	------	-------	-------

Linear Index: Disk Blocks

# Problemas con índices secundarios

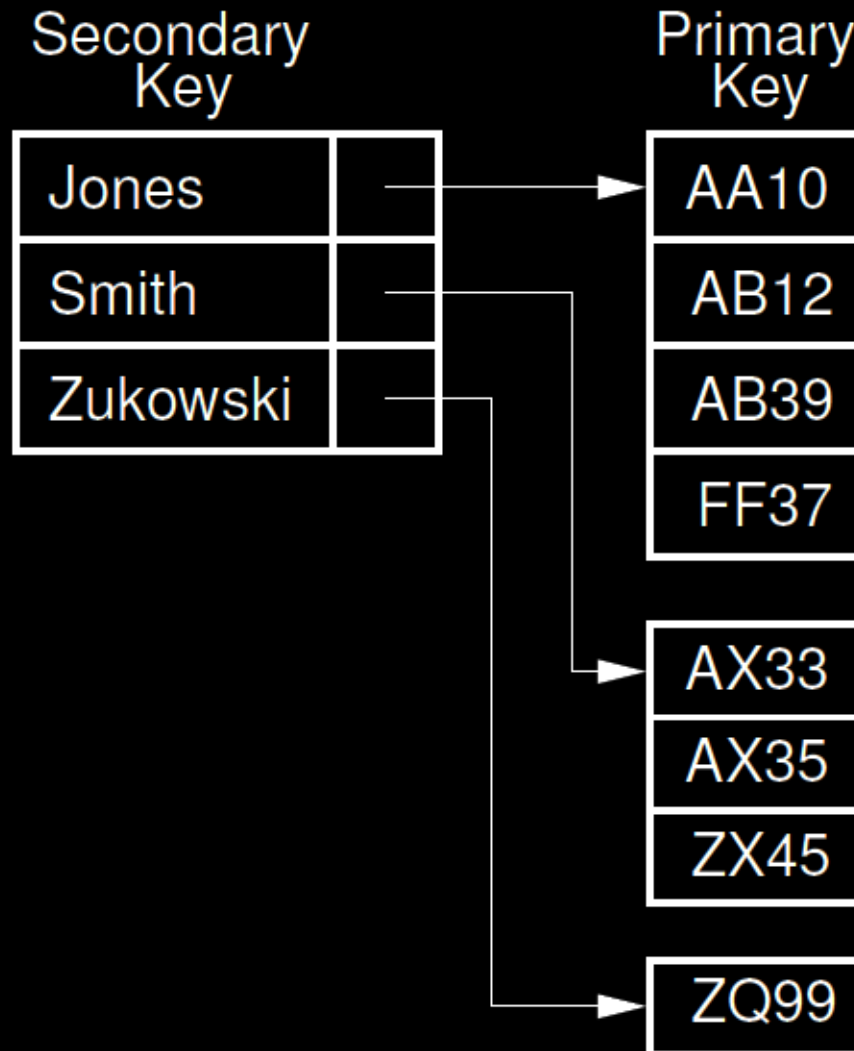
- Si muchos valores del índice **secundario** se **repiten**, entonces se **desperdicia espacio**
- Por ejemplo, índice secundario por **provincia** resultaría en pocas llaves (las 7 provincias) repetidas muchas veces por cada registro
- Estos casos se pueden **optimizar** con:
  - Índices bidimensionales
  - Listas invertidas

# Índices bidimensionales

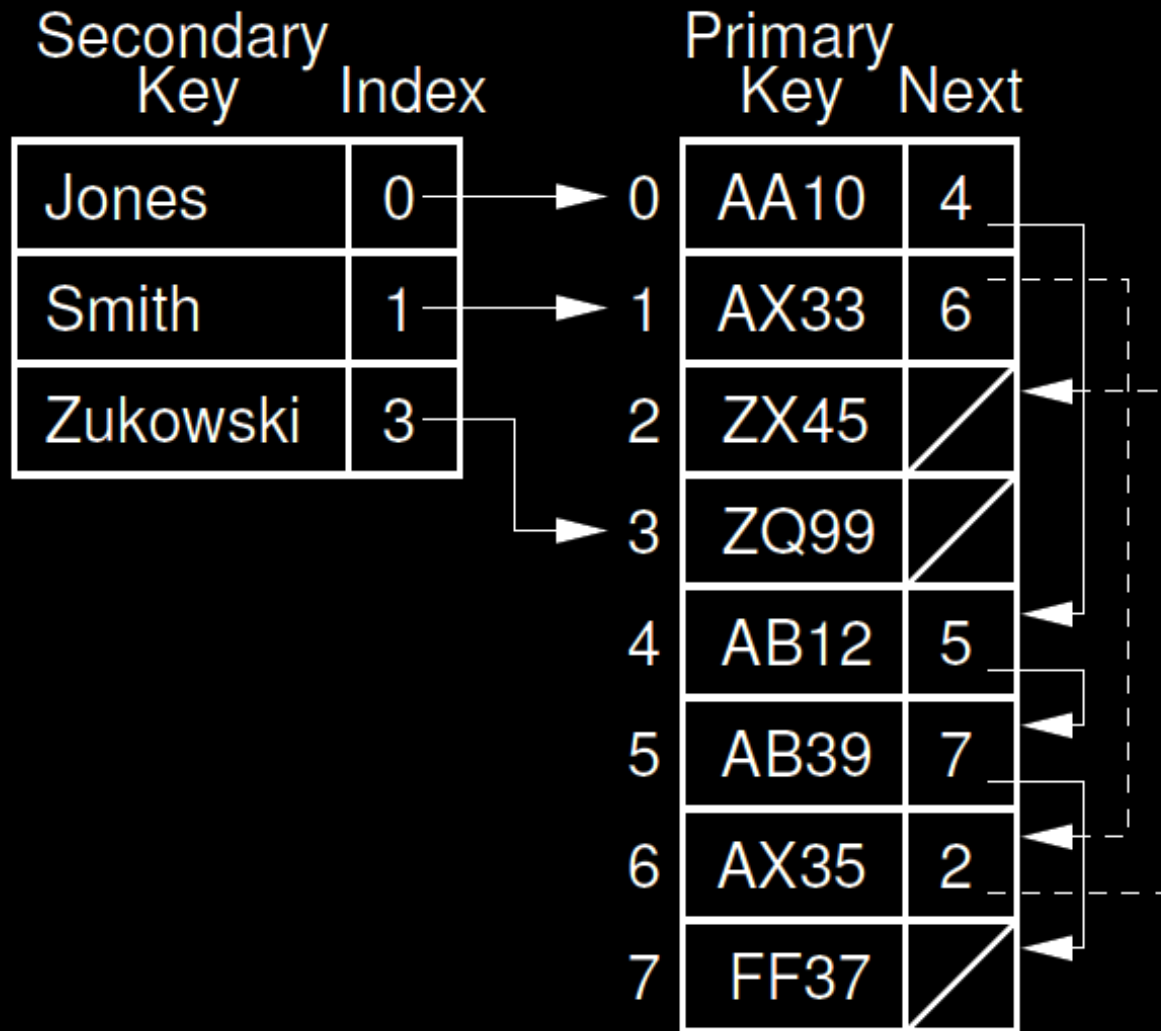
Jones	AA10	AB12	AB39	FF37
Smith	AX33	AX35	ZX45	
Zukowski	ZQ99			



# Lista invertida



# Lista invertida (evitar múltiples archivos)



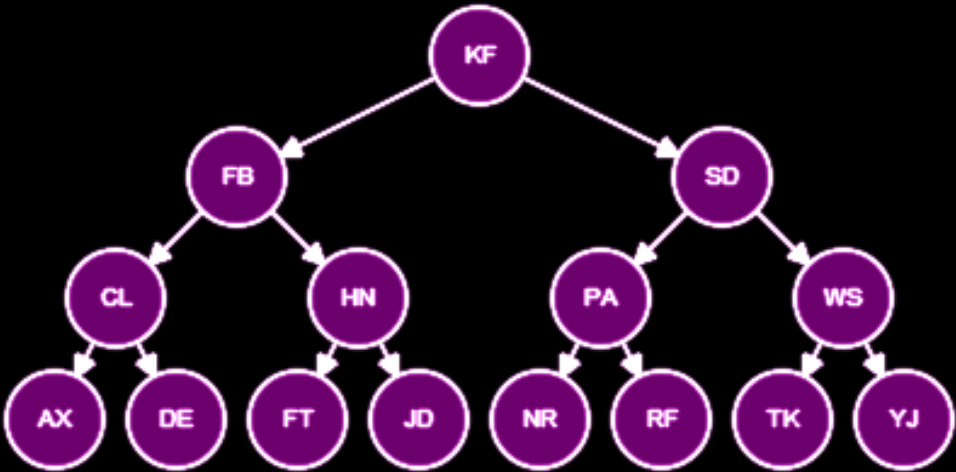
# Indexación con árboles

- El principal problema con archivos es que acceder al almacenamiento en disco es lento
- Los índices lineales → búsqueda binaria
- Búsqueda binaria → requiere muchas comparaciones y accesos a disco
  - 9.5 accesos a disco en promedio para buscar en 1000 elementos

- Para un **rendimiento** óptimo se busca lo siguiente
  - Buscar en el índice debe ser más rápido que la búsqueda binaria
  - Inserción y borrado deben ser tan rápidos como la búsqueda

AX	CL	DE	FB	FT	HN	JD	KF	NR	PA	RF	SD	TK	WS	YJ
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Root → 9



0	FB	10	8
1	JD		
2	RF		
3	SD	6	13
4	AX		
5	YJ		
6	PA	11	2
7	FT		
8	HN	7	1
9	KF	0	3
10	CL	4	12
11	NR		
12	DE		
13	WS	14	15
14	TK		

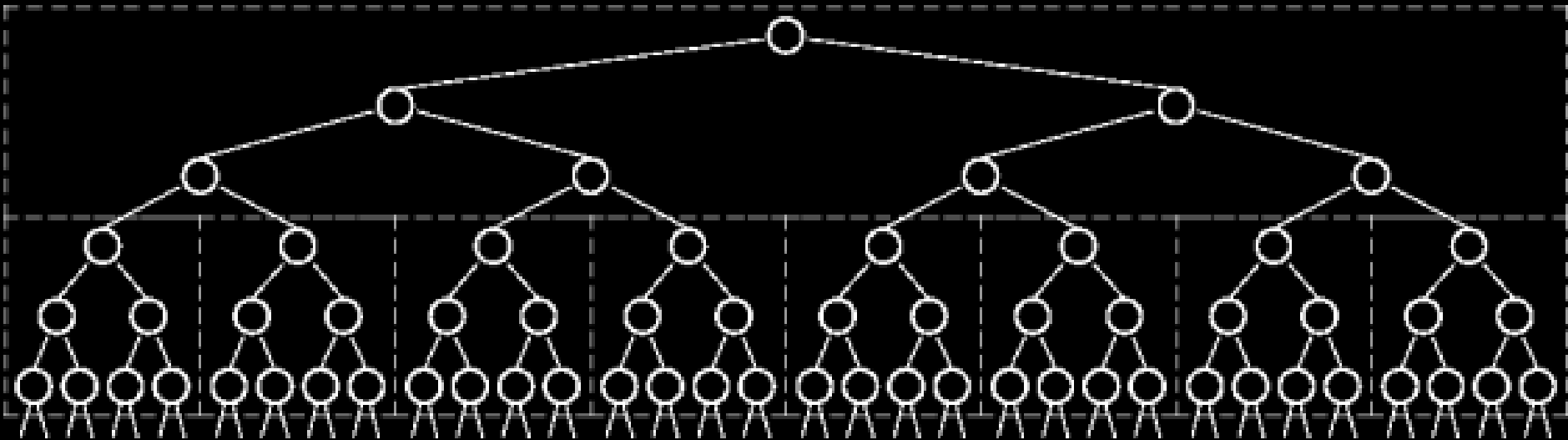
- Dado que la estructura de un árbol binario depende del orden de inserción, un árbol **desbalanceado** puede requerir muchos accesos a disco
- Utilizar **AVL** como opción para minimizar los accesos a disco
- Estudios del comportamiento de un AVL han demostrado que el árbol se reorganiza casi para **cada inserción** y para cada **cuatro borrados**
- Esto limita la utilidad de los AVL sólo para índices en **memoria**

- Un índice implementado con AVL
  - No es más rápido que la búsqueda binaria
  - Sí realiza inserción y borrado eficientes
- El problema es que, en caso de estar en disco, sigue requiriendo múltiples accesos a disco para llegar a un valor

# Indexación con árboles paginados

- Traer información de disco por cada nodo que se desea acceder puede **optimizarse**
- Una opción es distribuir los nodos del árbol de índices en **páginas**
- Esto **disminuye** la cantidad de **accesos** a disco porque se cargan en memoria los bloques que se van necesitando
- Las páginas se ubican en bloques **contiguos** para disminuir la cantidad de accesos





- ¿Cuántas lecturas de disco máximo se necesitan para localizar uno de los 63 nodos?
- ¿Cuántas lecturas de disco máximo se necesitan para localizar uno de 511 nodos?

# Problemas con los árboles paginados

- Se sigue usando **mucho espacio** en disco para almacenar referencias
- La utilización de un árbol **no** está generando tanto **beneficio** en contraste a utilizar una estructura lineal
- **Construir el árbol** a partir de una lista de índices también representa un problema complejo

- Debe intentarse mantener el árbol lo más **balanceado** posible, por medio de rotaciones dentro de las páginas
- Ejemplo:
  - Crear un árbol paginado con tres nodos por página
  - Los nodos de cada página pueden rotarse para mejorar el balanceo
  - Ins.: 3, 19, 4, 20, 1, 13, 16, 9, 2, 23, 14, 7, 21, 18, 11, 5, 8, 15, 12, 10, 25, 17, 26, 6, 24, 22

Ins.: 3, 19, 4, 20, 1, 13, 16, 9, 2, 23, 14, 7, 21, 18, 11, 5, 8, 15, 12, 10,  
25, 17, 26, 6, 24, 22

- Aunque paginar un árbol es una buena idea para minimizar accesos a disco, se siguen dando problemas
  - ¿Cómo asegurar que las llaves en la página raíz **dividen** de forma **adecuada** el resto del árbol?
  - ¿Cómo evitar que llaves **contiguas** queden juntas en la misma página?
  - ¿Cómo evitar que toda una página tenga **pocas llaves**?

# B-trees

Rudolf Bayer (1939- Alemania)

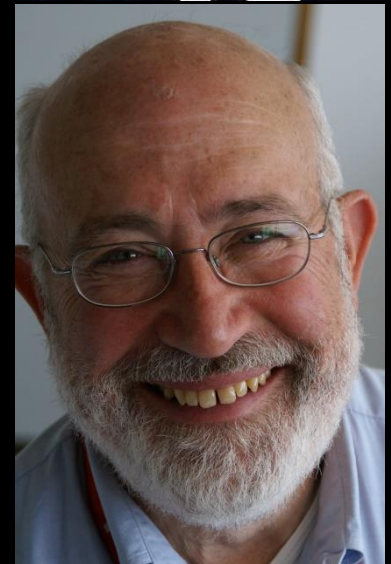
Edward M. McCreight (EE. UU)

Laboratorio de Matemáticas y Ciencias de la Información de  
**Boeing**

- 1971: Organization and maintenance of large ordered indexes
- 1979: The Ubiquitous B-Tree
- Durante los 80s, varias empresas lo utilizaron para **DBMS**
- Durante los 90s, tercera era de los B-Trees, **FS**
- En la actualidad se continúa mejorándolos y aplicándolos a diferentes escenarios

¿Qué **significa** la B?

Balanced, broad, bushy, Boeing, Bayer



# B-trees

- Estructura de **árbol** que mantiene los datos ordenados → **Generalización** de los árboles de búsqueda binaria
- Permite búsquedas, acceso secuencial, inserciones y borrados de forma eficiente
- Optimizados para sistemas que leen y escriben **grandes bloques** de datos
- Utilizados para la creación de **bases de datos**, **sistemas de archivos** y en compresión de datos

1. Balanceados por altura, todas las **hojas** están siempre al **mismo nivel**
2. Actualización y búsqueda afectan **pocos** bloques de disco
3. Registros **similares** se encuentran en el mismo bloque
4. Nodos llenos hasta cierto porcentaje mínimo

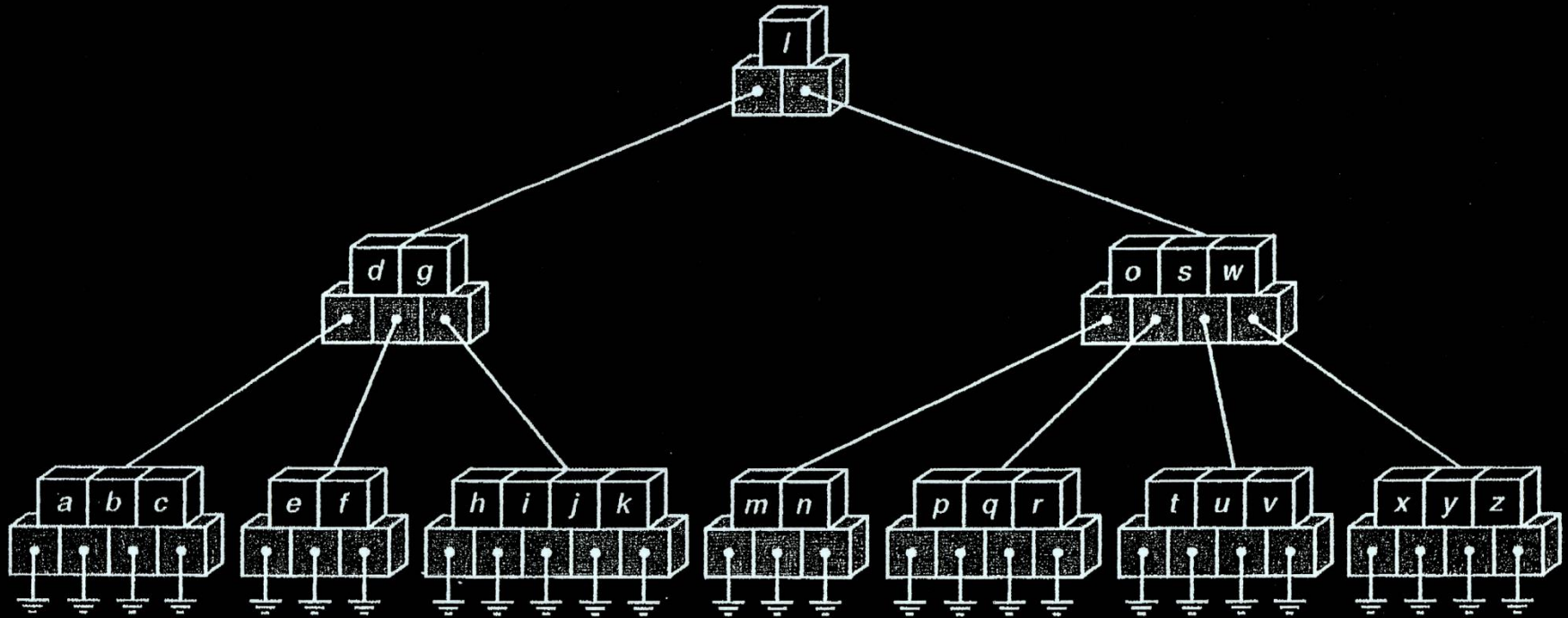


# Definiciones

- Nodo → **Página**
  - Se acceden en bloques
- En un B-Tree de **orden  $m$** :
  - La raíz es una hoja o tiene al menos dos hijos
  - Las páginas internas (menos la raíz) tienen mínimo  $\lceil m/2 \rceil$  hijos y máximo  $m$  hijos
  - Cada página tiene **varias llaves** (1 menos que el número de hijos) que dividen sus hijos igual que un árbol de búsqueda
  - La raíz tiene como máximo  $m$  ramas

- Los B-Trees han sustituido básicamente todos los métodos de acceso a **archivos grandes**
- Se utilizan para implementar la mayoría de los sistemas de archivos modernos
- El tamaño de la página se escoge para que sea el mismo que un **bloque en disco**
- Una implementación típica permite 100 o más hijos

# B-Tree de orden 5



# Representación de un B-Tree de orden $m$

- Almacenar **claves** y almacenar los **hijos**
- Se utilizan dos **vectores** y un campo que indica cuántas claves tiene la página

# Inserción

- Como todas las hojas deben estar en el mismo nivel, los B-Trees crecen “**hacia arriba**”
- Algoritmo de inserción:
  - Buscar si la llave ya existe en el árbol
  - Si no está, se intenta insertar en el último nodo encontrado
  - Si no está lleno el nodo, se inserta
  - Si está lleno el nodo se **divide** en dos nodos (incluyendo la nueva llave) y la llave del medio se “promueve” y se inserta recursivamente en el nodo padre

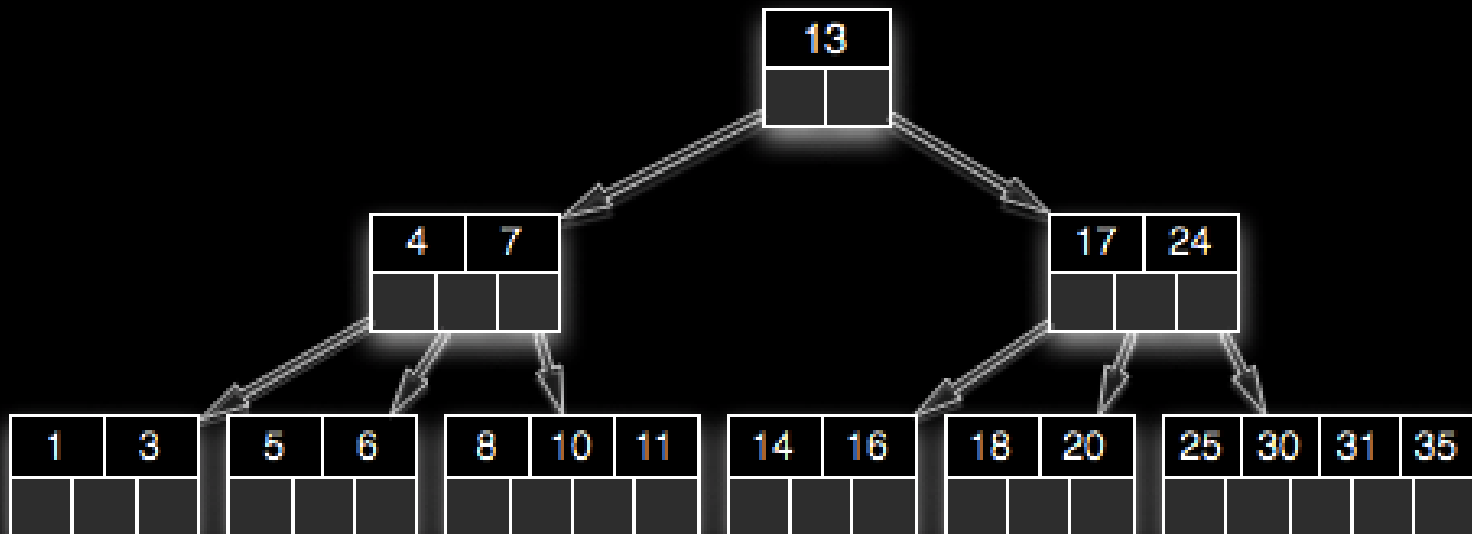
- [6, 11, 5, 4, 8, 9, 12, 21, 14, 10, 19, 28, 3, 17, 32, 15, 16, 26, 27]
- [53, 47, 51, 70, 38, 48, 13, 82, 49, 39, 28, 42, 72, 41, 8, 40, 54, 91, 16, 55, 7, 64, 12, 67, 75]
- [67, 6, 62, 91, 89, 63, 69, 61, 8, 10, 2, 56, 87, 65, 72, 31, 7, 60, 97, 68, 59, 20, 85, 80, 74]

# Búsqueda de una llave

- Es necesario para la operación de **inserción**
- Misma estrategia que la búsqueda en un árbol binario
- Pasos:
  - Se realiza **búsqueda binaria** en el nodo actual
  - Si se encuentra la llave, se retorna el nodo actual
  - Si no se encuentra y el nodo es una hoja, se retorna "no encontrado"
  - Si no es una hoja, repetir la búsqueda en el hijo correspondiente

# Recorridos

- Al igual que con los árboles binarios se pueden realizar recorridos **preorden**, **inorden** y **postorden**
- Ejercicio: escriba la salida de los tres recorridos en el siguiente B-Tree





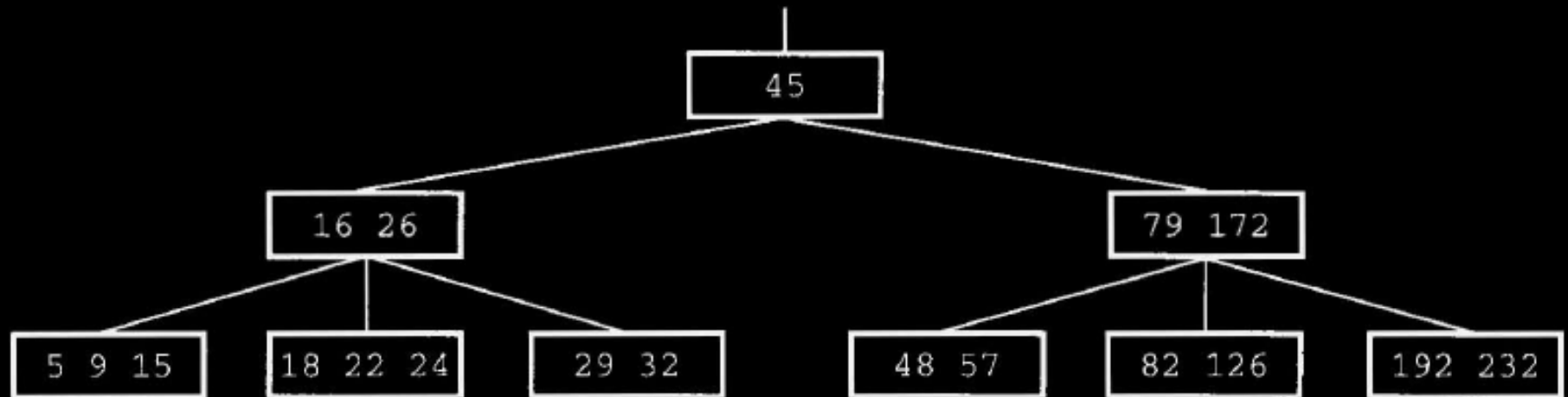
# Eliminación de un B-Tree

- Casos:
  - La llave es un separador de dos nodos (**interno**)
    - Se busca un **reemplazo** para esa llave, el valor más pequeño del subárbol derecho o el más grande del subárbol izquierdo
    - Se **intercambian** y ahora la llave a eliminar está en una hoja
  - La llave está en una **hoja**
    - Se busca la llave y se **borra** (no hay hijos por los cuales preocuparse)
    - Si el nodo queda con menos de  $m/2$  elementos, debe **rebalancearse**

# Rebalanceo después de un borrado

- Si algún nodo hermano tiene **más del mínimo** de nodos
  - Se agrega el separador al nodo deficiente
  - Se toma el mayor (hermano derecho) o el menor (hermano izquierdo) y se usa como separador
- Si **ningún** hermano tiene más del mínimo de nodos
  - Se crea un nodo nuevo con las claves del nodo deficiente, las claves de uno de sus hermanos y la clave separadora
  - Se elimina la clave separadora y sus dos ramas se sustituyen sólo por el nuevo nodo
- Se repite el rebalanceo necesario en nodos superiores hasta llegar a la raíz

Eliminar (orden 5):  
16, 24, 22



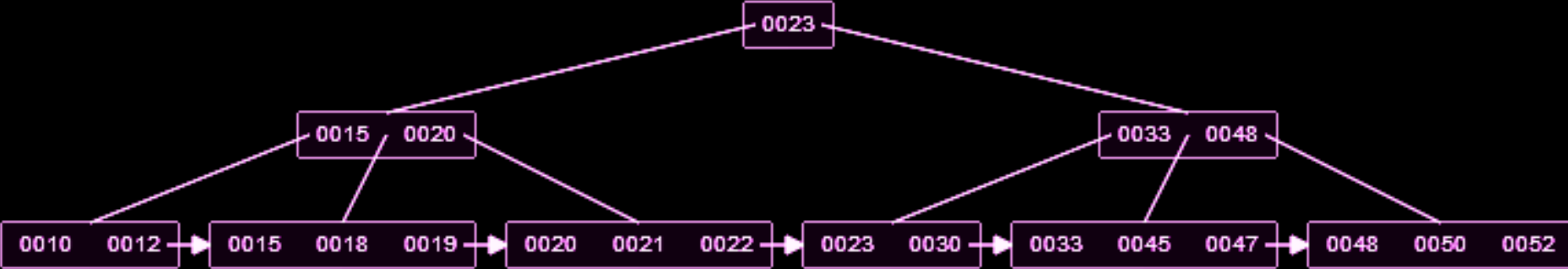
# B+Tree

- **Modificación** el B-Tree
- Se puede ver como un B-Tree que:
  - Cada nodo sólo contiene **llaves**, no pares llave-valor
  - Tiene un **nivel** adicional con **hojas enlazadas**
- Almacena datos de forma que es fácil accederlos **secuencialmente** y en **bloques**

- Sistemas de **archivos**
  - NTFS (Windows)
  - ReiserFS (Linux)
  - NFS (Novell)
  - XFS (Linux, FreeBSD)
  - JFS (Linux, AIX, OS/2)
  - ReFS (Windows Server)
- Motores de **BD**
  - IBM DB2
  - Informix
  - MS SQL Server
  - Oracle 8
  - Sybase ASE
  - SQLite

- Las **páginas** pueden ser internas u hojas
- Las **internas** contienen punteros a otras páginas internas o a las hojas
- Las **hojas** pueden contener
  - Los datos reales
  - Punteros a registros en otro archivo
- La estructura de ambos es diferente

- Las hojas del B+Tree están **enlazadas** en una lista (simple o doble)
- Esto permite recorrer los registros en orden **secuencial**
- Los nodos internos no contienen datos ni punteros a los datos reales, por lo que una **llave** encontrada en un nodo interno **también** va a estar en una **hoja**
- Los nodos internos se utilizan sólo para guiar la **búsqueda**





# Inserción

- Se **busca** la hoja donde debe insertarse
- Si la hoja **no está llena**, se inserta
- Si está llena, entonces **partir** el nodo y **promover** una llave
  - La llave que se promueve también **permanece** en el nodo mayor

# Búsqueda

- Similar a un B-Tree
- Si se encuentra la llave en un nodo interno, se continúa buscando por el hijo mayor de esa llave hasta llegar a la hoja respectiva

# Borrado

- **Buscar** la hoja que contiene el elemento a eliminar
- Si tiene **más** que la cantidad **mínima**, se borra
- Si no, se elimina y **rebalancea** similar a un B-Tree

A large, glowing yellow-orange sphere with a blue ring, set against a black background. The sphere has a soft, ethereal glow and a thin blue ring around its equator. The background is solid black.

Archivos

Mauricio Avilés