Estrutura e tipo de dados
Índices, tipos de árvores B
Tipos de arquivos

Enzo @EnzoTMorente

Armazenamento em disco

Armazenamento em disco

É o termo que define o ato de armazenar dados em hard disks em sistemas computacionais, ou seja a memória secundária

- A informação não cabe na memória principal
- Queremos que a informação seja permanente

Hard disk(HD)

- -São utilizados para armazenamento de grande quantidade de dados
- -Não voláteis
- -Na unidade de disco estão presentes: cabeçote de leitura/escrita, braço mecânico, atuador e controladora de disco
- -Divisão física e divisão lógica

Divisão física

- Disco: conjunto de ‘pratos’ empilhados, com duas faces, dados são gravados nas superfícies desses pratos, a partir de cabeças de leitura(1 para cada face)
- Superfícies: são organizadas em trilhas
- Trilhas: são círculos que vão do exterior do disco ao interior, as trilhas são organizadas em setores
- Setores:menores unidades físicas de armazenamento em um disco rígido.
- Cilindro: conjunto de trilhas na mesma posição

Trilhas e setores:

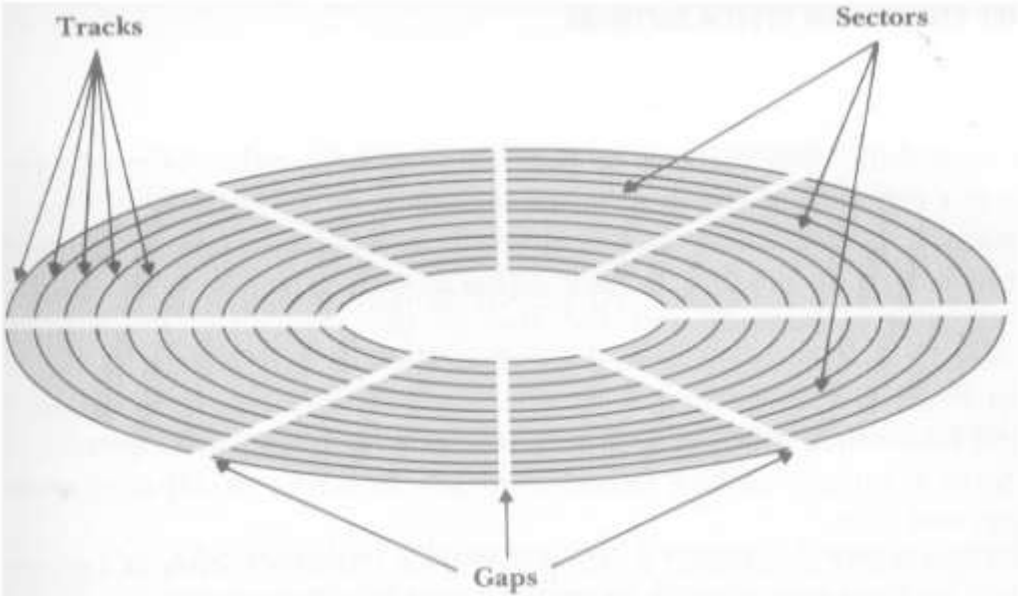


FIGURE 3.2 Surface of disk showing tracks and sectors.

Cilindros

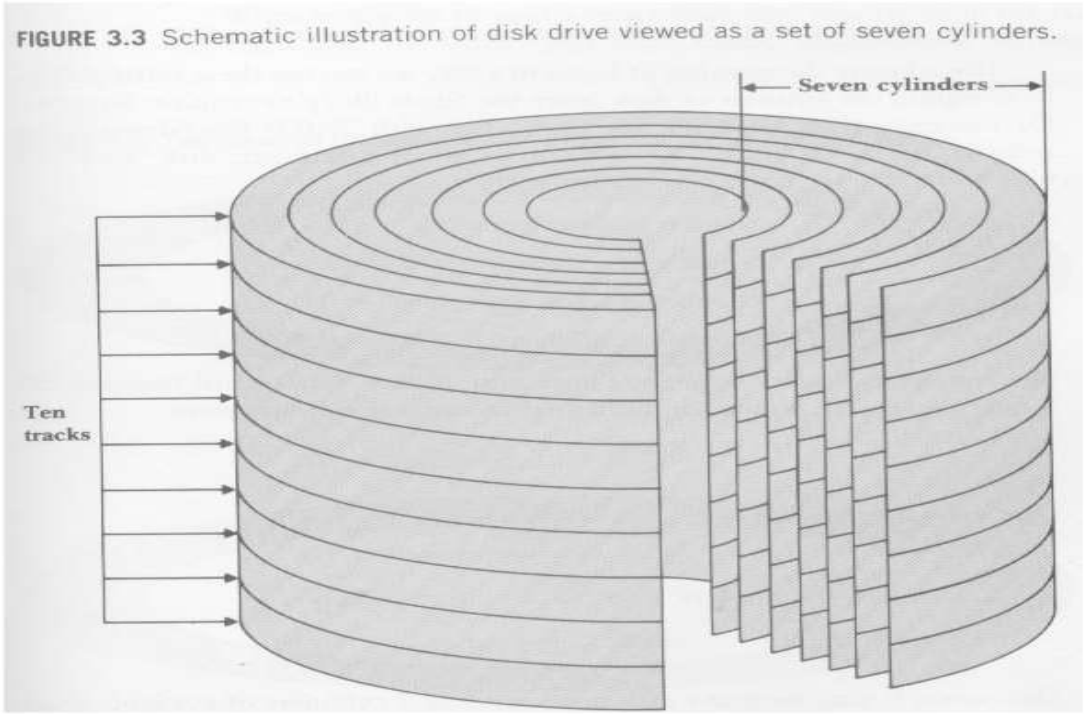
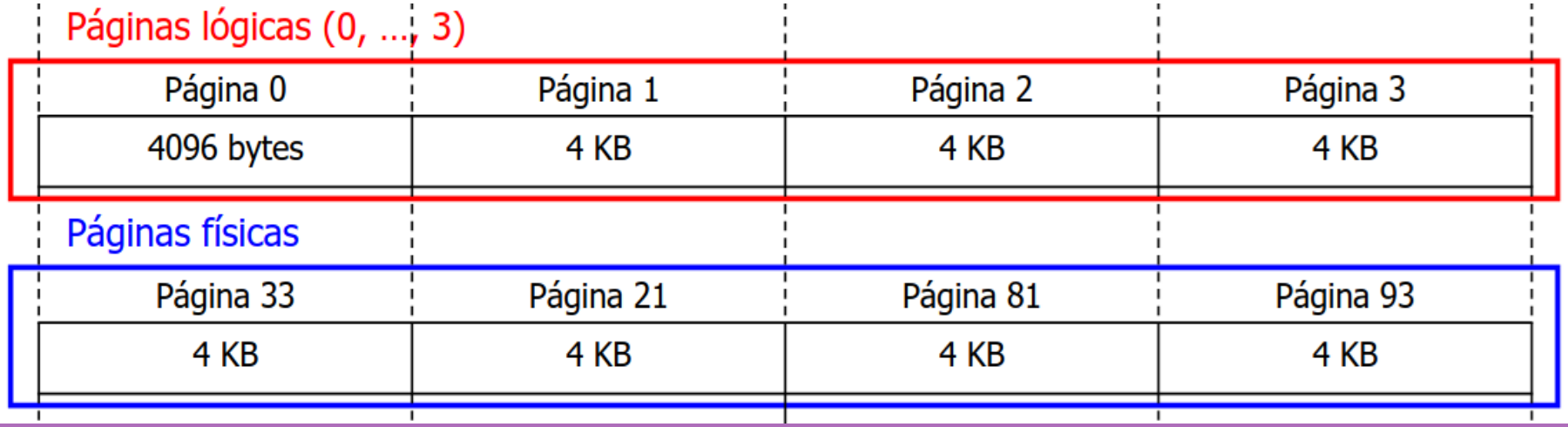


FIGURE 3.3 Schematic illustration of disk drive viewed as a set of seven cylinders.
Seven cylinders
Ten tracks

Divisão lógica

- Blocos: são a unidade de transferência entre o disco e a memória principal



Estrutura de dados em disco

Estrutura de dados em disco

Diferente das estruturas de dados na memória RAM, que são voláteis e usadas para acesso rápido, as estruturas de dados em disco são projetadas para armazenamento persistente, garantindo que os dados sejam mantidos mesmo quando o sistema é desligado.

Índices

índices

- Recuperação de informação de forma rápida
- Eficiência na busca
- Facilitar acesso a elementos de um grande conjunto de dados sem precisar percorrê-lo completamente

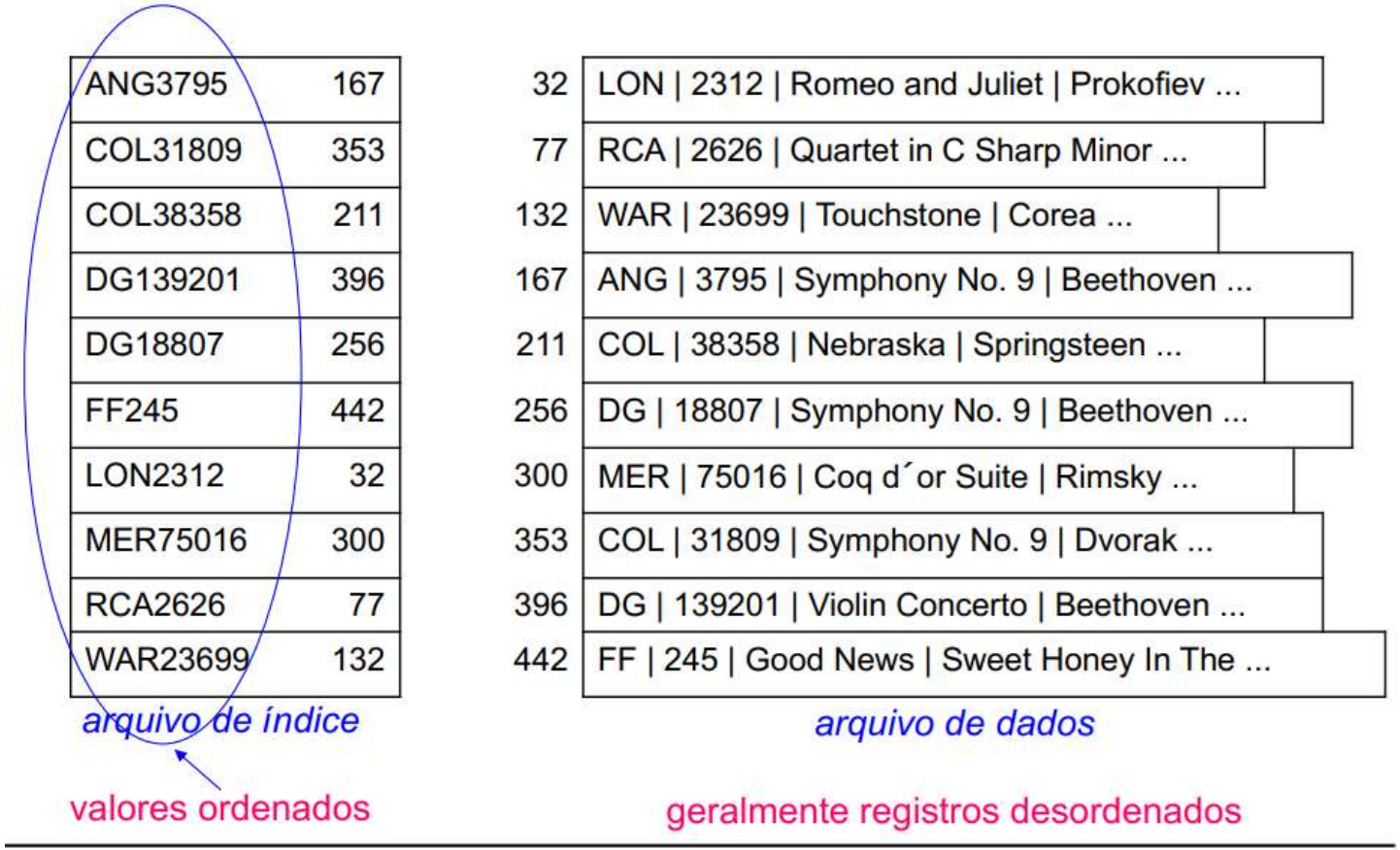
0	1 2 3 4	5 6 7 8 9 10 11 12	13 14 15 16
removido	tamanhoRegistro	Prox	id
1 byte	4 bytes	8 bytes	4 bytes

17 18 19 20	21 22 23 24	25 … … … …	… … … …
idade	tamNomeJog	nomeJogador	tamNacionalidade
4 bytes	4 bytes	variável	4 bytes

… … … …	… … … …	… … … … …
nacionalidade	tamNomeClube	nomeClube
variável	4 bytes	variável

índices

- Vários registros contidos na memória um do lado do outro
- Necessário percorrer todos os registros para achar o desejado
  - Ineficiente
- Guardar o byte offset e o id de cada jogador
  - Fazer uma busca binária dado um id para achar o byte offset


ANG3795 167
COL31809 353
COL38358 211
DG139201 396
DG18807 256
FF245 442
LON2312 32
MER75016 300
RCA2626 77
WAR23699 132
arquivo de índice
valores ordenados
32 LON | 2312 | Romeo and Juliet | Prokofiev ...
77 RCA | 2626 | Quartet in C Sharp Minor ...
132 WAR | 23699 | Touchstone | Corea ...
167 ANG | 3795 | Symphony No. 9 | Beethoven ...
211 COL | 38358 | Nebraska | Springsteen ...
256 DG | 18807 | Symphony No. 9 | Beethoven ...
300 MER | 75016 | Coq d´or Suite | Rimsky ...
353 COL | 31809 | Symphony No. 9 | Dvorak ...
396 DG | 139201 | Violin Concerto | Beethoven ...
442 FF | 245 | Good News | Sweet Honey In The ...
arquivo de dados
geralmente registros desordenados

Índice

- Muito boa para poucos dados
- Muitos dados
  - Não tem como manter todo o arquivo de índices na memória principal
  - Ou vai ter que fazer acesso em locais diferentes muito rapidamente

Árvore B

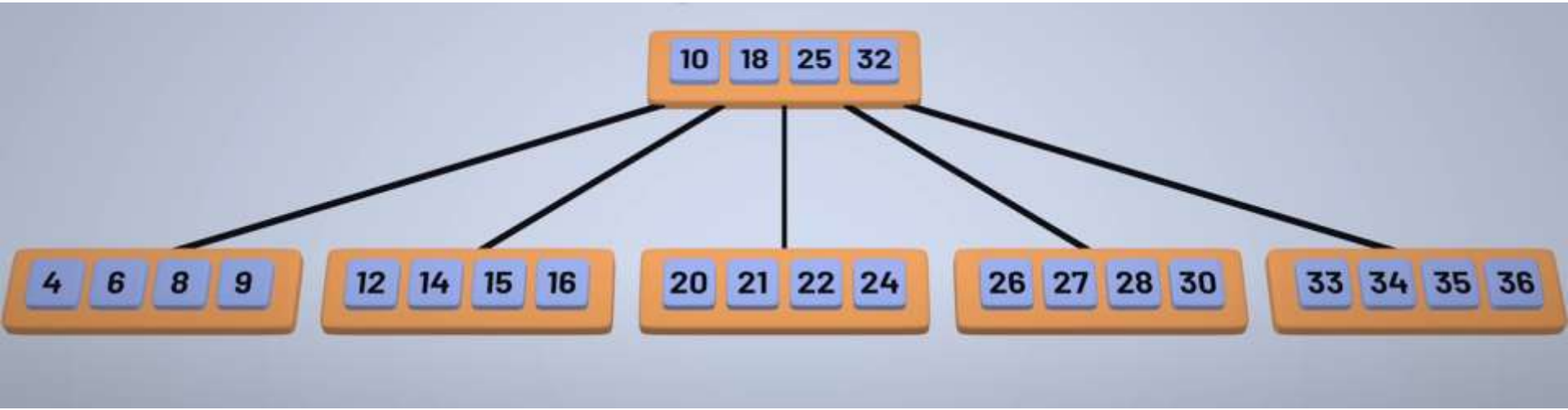
Árvore B

- Muito utilizada quando se tem grandes números de dados
- A Árvore B nunca será completamente carregada na memória principal
  - Só se carrega no máximo um número fixo de nós por vez na memória

Árvore B

- Que nem uma árvore binária
  - Todos os filhos a esquerda são menores
  - Todos os filhos a direita são maiores
- Grande diferença
  - Cada nó da árvore vai conter mais de 1 item
  - Vai conter múltiplos filhos

Árvore B



Source: https://www.youtube.com/watch?v=K1a2Bk8NrYQ

Árvore B

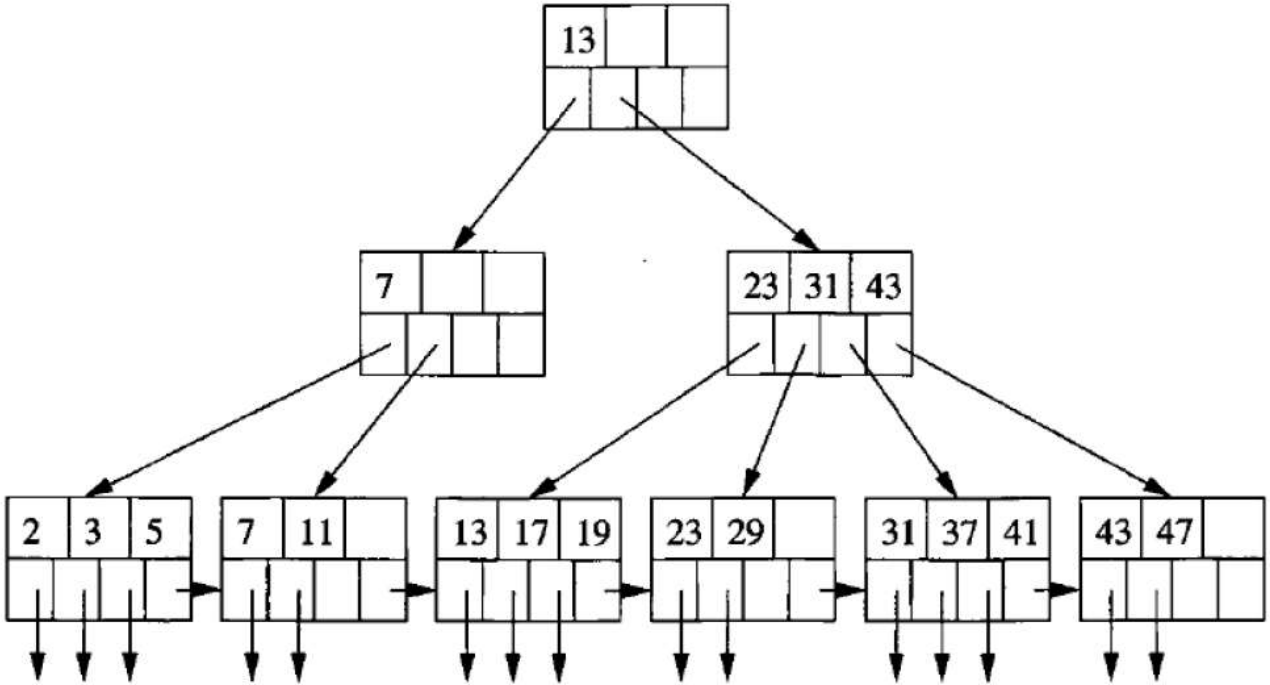
- Ter vários nó em um filho só aumenta a complexidade
  - No fim você vai acabar fazendo mais comparações
- Então porque a árvore b é utilizada?
  - A operação mais custosa é a de acessar a memória
  - Acessar a memória 1 vez vai trazer vários itens
  - Nó tamanho da página de disco

Árvore B+

Árvore B+

- Todos os Byte Offsets estão armazenados somente nos nós folhas
- Os nós intermediários contém valores somente para ajudar na navegação
- Os nós folhas estão ligados entre si
    - Busca linear muito rápida

EXAMPLE: B+-TREE



Árvore B*

Árvore B*

- O algoritmo de balanceamento
  - Diminui a altura da árvore
  - Maior número de itens nos nós internos
- Árvore mais compacta
- Melhor desempenho

Comparações

Árvore B

- Desvantagens
  - Uso ineficiente dos nós internos
  - Mais demorada na busca
- Vantagens
  - Implementação mais simples

Árvore B+

- Desvantagens
  - Duplicação de chaves
  - Maior complexidade de inserção
- Vantagens
  - Uso eficiente dos nós internos
  - Busca sequencial rápida

Árvore B*

- Desvantagens
  - Mais complexa
- Vantagens
  - Uso eficiente dos nós internos
  - Mais rápida para buscar um elemento

Select Where

- Select jogador where id = 1920
  - B* seria melhor
- Select jogadores where id >= 1920
  - B+ seria melhor

Qual é a ideal para implementar?

Tipos de dados

Tipos de dados

Tipos de arquivos

CSV

```
name, id, favorite food
quincy, 1, hot dogs
beau, 2, cereal
abbey, 3, pizza
mrugesh, 4, ice cream
```

	A	B	C
1	name	id	favorite food
2	quincy	1	hot dogs
3	beau	2	cereal
4	abbey	3	pizza
5	mrugesh	4	ice cream

Parquet

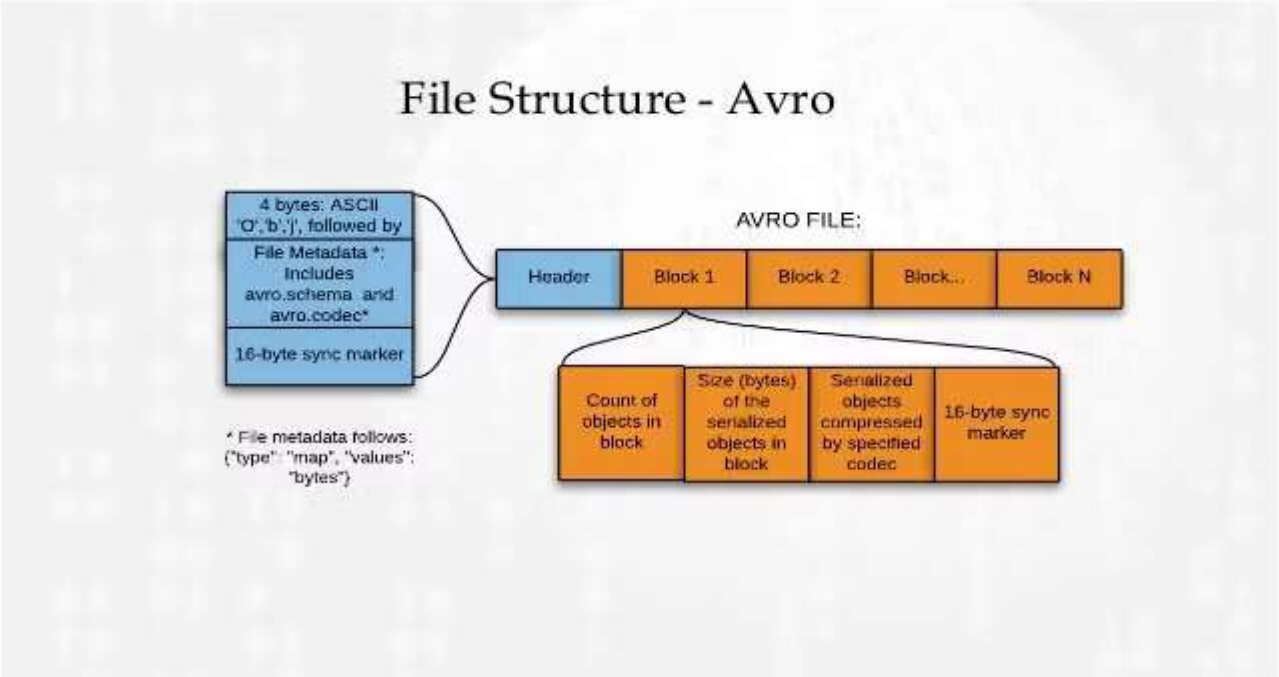
ID	Nome	Idade
1	Alice	25
2	Bob	30
3	Carol	22

```
+-----------------------------------+
| $1                                |
|-----------------------------------|
| {                                 |
|   "CITY": "Paris",                |
|   "CONTINENT": "Europe",          |
|   "COUNTRY": "France"             |
| }                                 |
|-----------------------------------|
| {                                 |
|   "CITY": "Nice",                 |
|   "CONTINENT": "Europe",          |
|   "COUNTRY": "France"             |
| }                                 |
|-----------------------------------|
| {                                 |
|   "CITY": "Marseilles",           |
|   "CONTINENT": "Europe",          |
|   "COUNTRY": "France"             |
| }                                 |
+-----------------------------------+
```

JSon

```
{
  "user": {
    "id": "11",
    "animal": "Bob",
    "idade": “2”
  }
}
```

Avro


File Structure - Avro
4 bytes: ASCII 'O','b','j', followed by
File Metadata *: Includes avro.schema and avro.codec*
16-byte sync marker
* File metadata follows: {"type": "map", "values": "bytes"}
AVRO FILE:
Header
Block 1
Block 2
Block...
Block N
Count of objects in block
Size (bytes) of the serialized objects in block
Serialized objects compressed by specified codec
16-byte sync marker

Avro vs CSV

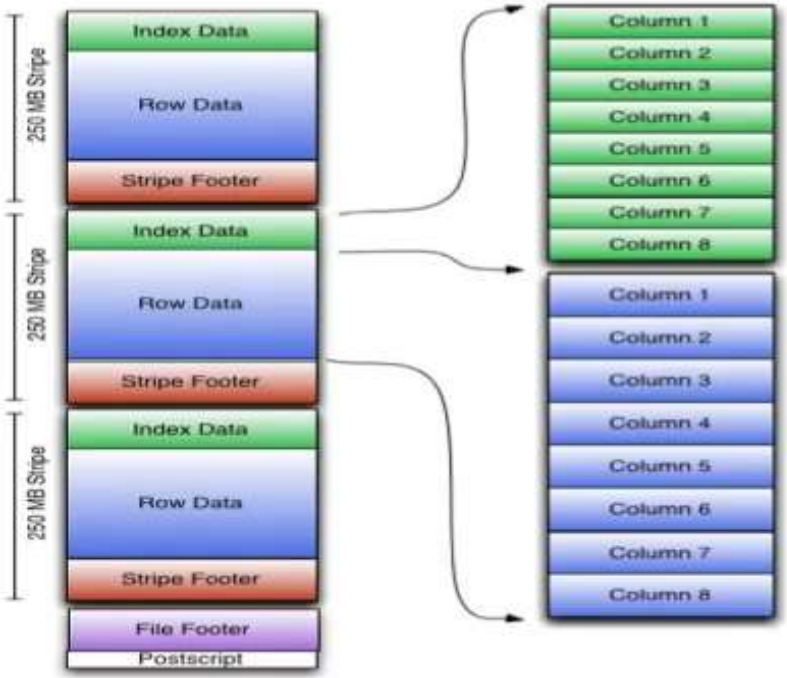
```
{
"id": 1,
"nome": "Alice",
"idade": 25,
"endereco": {
"rua": "Rua A",
"cidade": "São Paulo",
"pais": "Brasil"
}
}
```

Avro: possuem dados aninhados

```
{
"id": 1,
"nome": "Alice",
"idade": 25
}
```

CSV: possuem apenas dados simples,

ORC


250 MB Stripe
Index Data
Row Data
Stripe Footer
250 MB Stripe
Index Data
Row Data
Stripe Footer
250 MB Stripe
Index Data
Row Data
Stripe Footer
File Footer
Postscript
Column 1
Column 2
Column 3
Column 4
Column 5
Column 6
Column 7
Column 8
Column 1
Column 2
Column 3
Column 4
Column 5
Column 6
Column 7
Column 8

ORC vs Parquet

Característica	ORC	Parquet
Origem	Desenvolvido para Apache Hive	Desenvolvido para Hadoop/Spark
Organização	Estrutura mais complexa com stripes e metadados detalhados	Organizado em row groups com chunks de colunas
Metadados	Muito detalhados (índices, estatísticas)	Menos detalhados que ORC
Compressão	Melhor compressão, especialmente para dados numéricos e repetidos	Boa compressão, mas geralmente inferior ao ORC
Tamanho do arquivo	Menor devido à compressão eficiente	Tende a ser maior que ORC
Performance de leitura	Excelente para operações agregadas e scan de grandes volumes	Excelente para consultas que acessam colunas específicas
Casos de uso principais	Consultas analíticas no Hive, agregações em grandes volumes	Consultas analíticas em Spark, Presto, Athena, data lakes
Suporte à evolução de esquema	Limitado	Limitado, mas um pouco mais flexível