File System

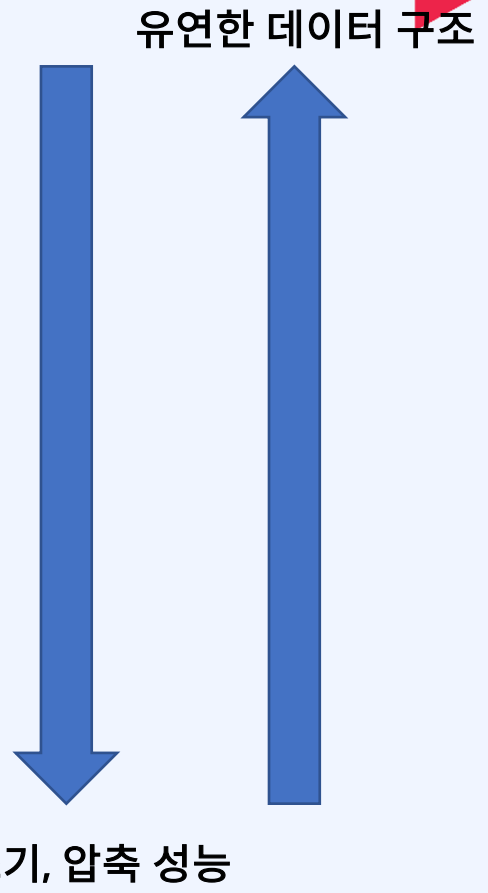
1. 컬럼 기반의 파일 포맷 - Apache Parquet

1. Apache Parquet 개요

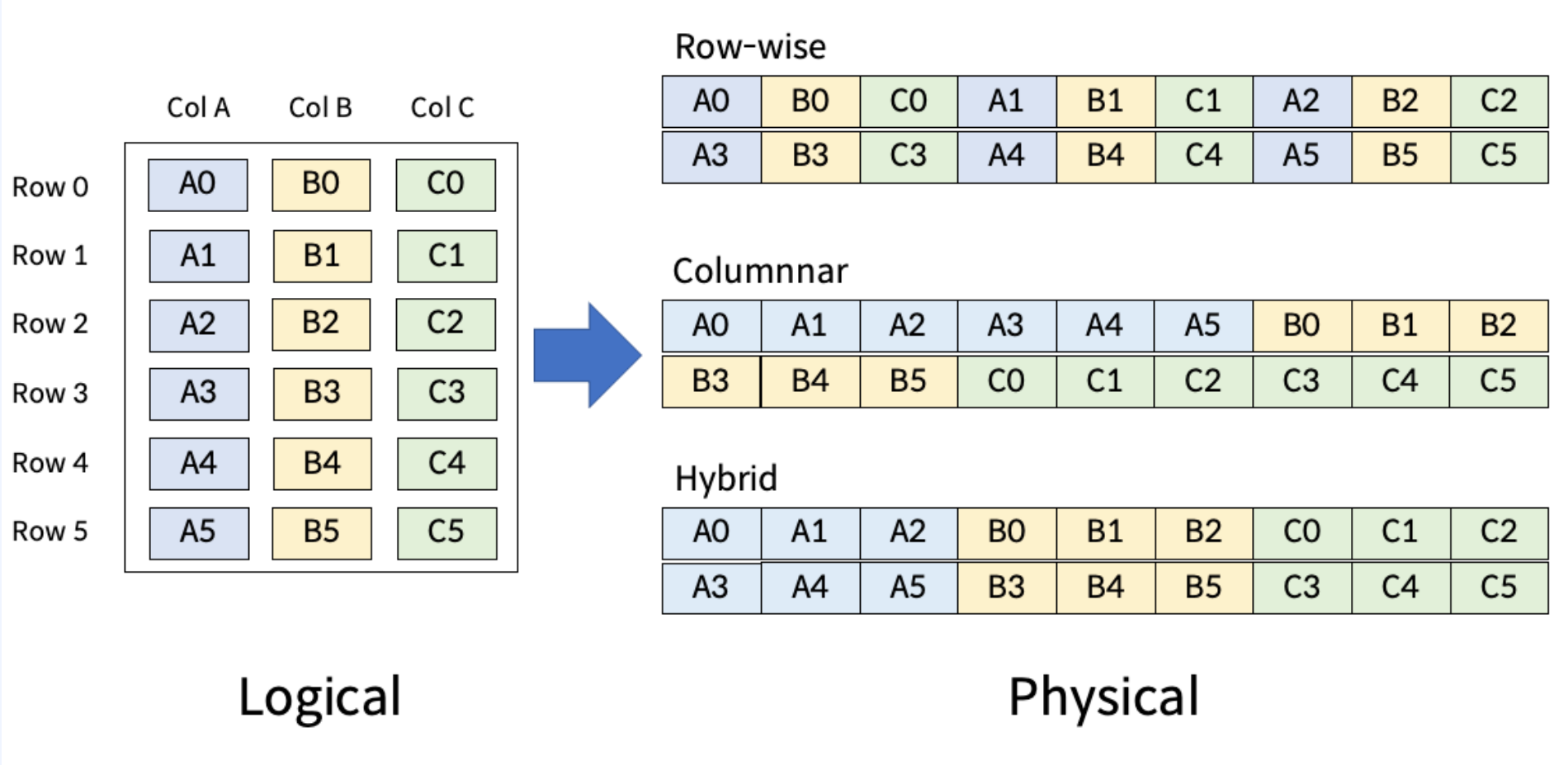
- Columnar storage format.
- 데이터 처리 프레임워크, 프로그래밍 언어와는 독립적
- Spark, Hadoop ecosystem에서 널리 사용 중.

2.0 Background) 다양한 데이터 파일 포맷

1. Unstructured
   - ex) TXT, CSV
   - 정의된 형식, 스키마가 없음.
2. Semi-Structured
   - ex) XML, JSON
   - 잘 정의된 스키마는 없지만, 어느 정도 일정한 포맷을 가지고 있고, nested 형태의 데이터를 표현할 수 있음.
3. Structured
   - ex) Parquet, ORC, Avro
   - 잘 정의된 스키마 존재.



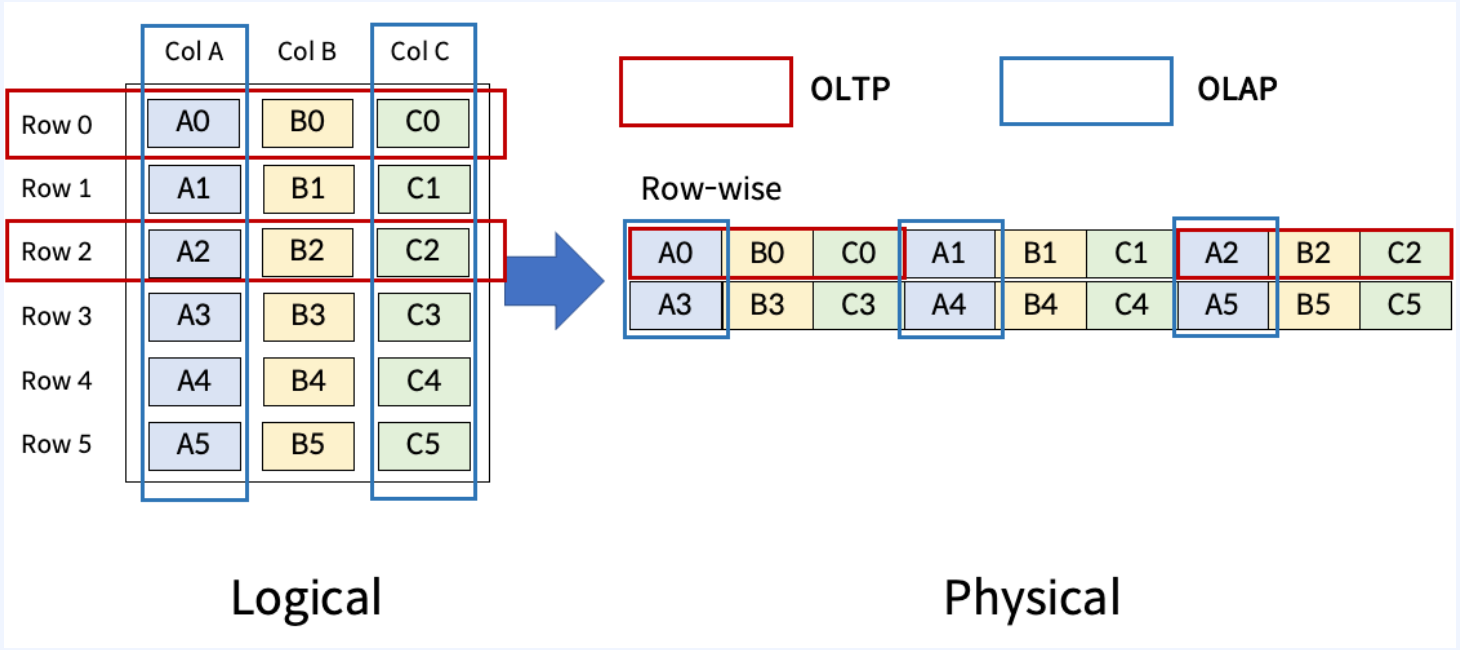
2.1 Background) 스토리지에 데이터를 저장하는 방식 비교



2.2 Background) Workload 방식

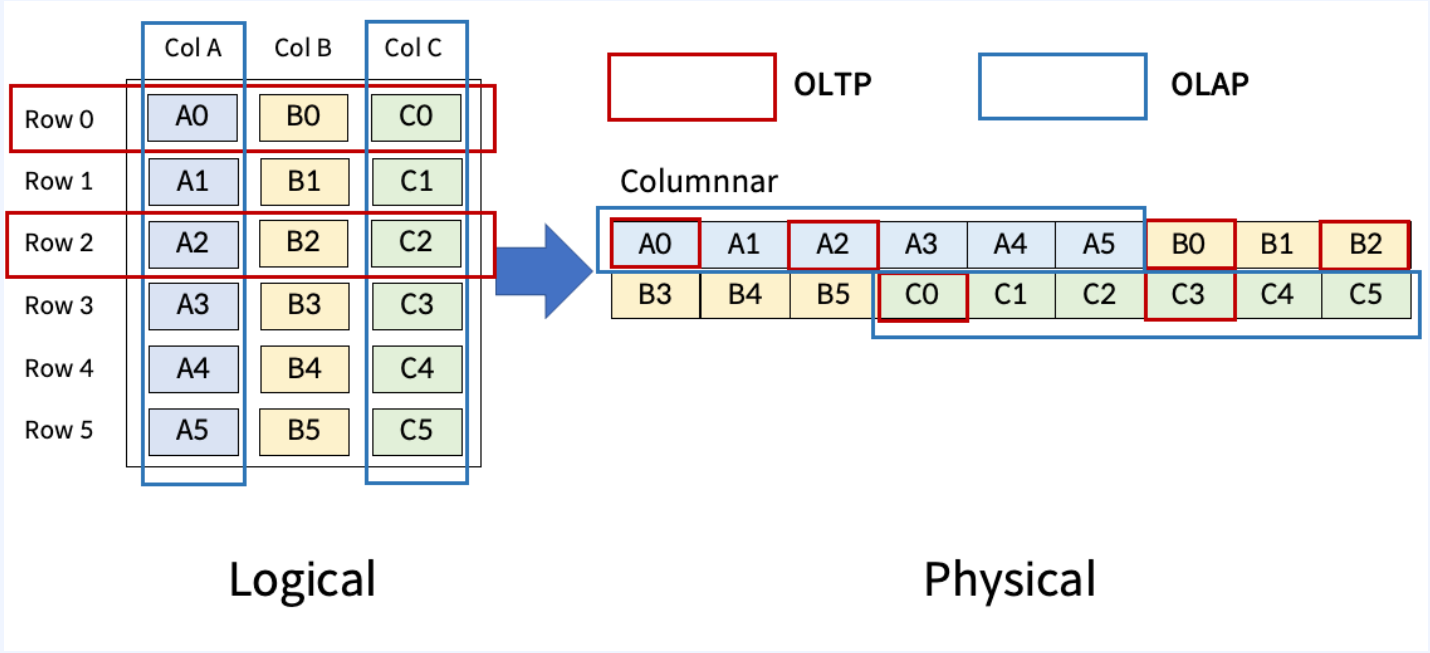
1. OLTP
    - Online transaction processing.
    - OLTP 데이터베이스 : Row-wise 방식으로 데이터를 배치 -> Why?
    - 전체 Row들에 대해 작은 연산들을 많이 수행.
    - ex) SNS의 글 쓰기, 읽기 - 하나하나는 작은 연산, 그러나 초당 수천, 수만 번 이상 수행될 수 있음
2. OLAP
    - Online analytic processing.
    - OLAP 데이터베이스 : Columnar 방식으로 데이터를 배치. -> Why?
    - 일부 Column 집합에 대해 아주 큰 연산을 조금 수행.
    - ex) 통계 정보 추출 - 큰 Group BY 연산, 그러나 이 작업을 초당 수천, 수만 번하지는 않음.

2.3 Background) Row-wise



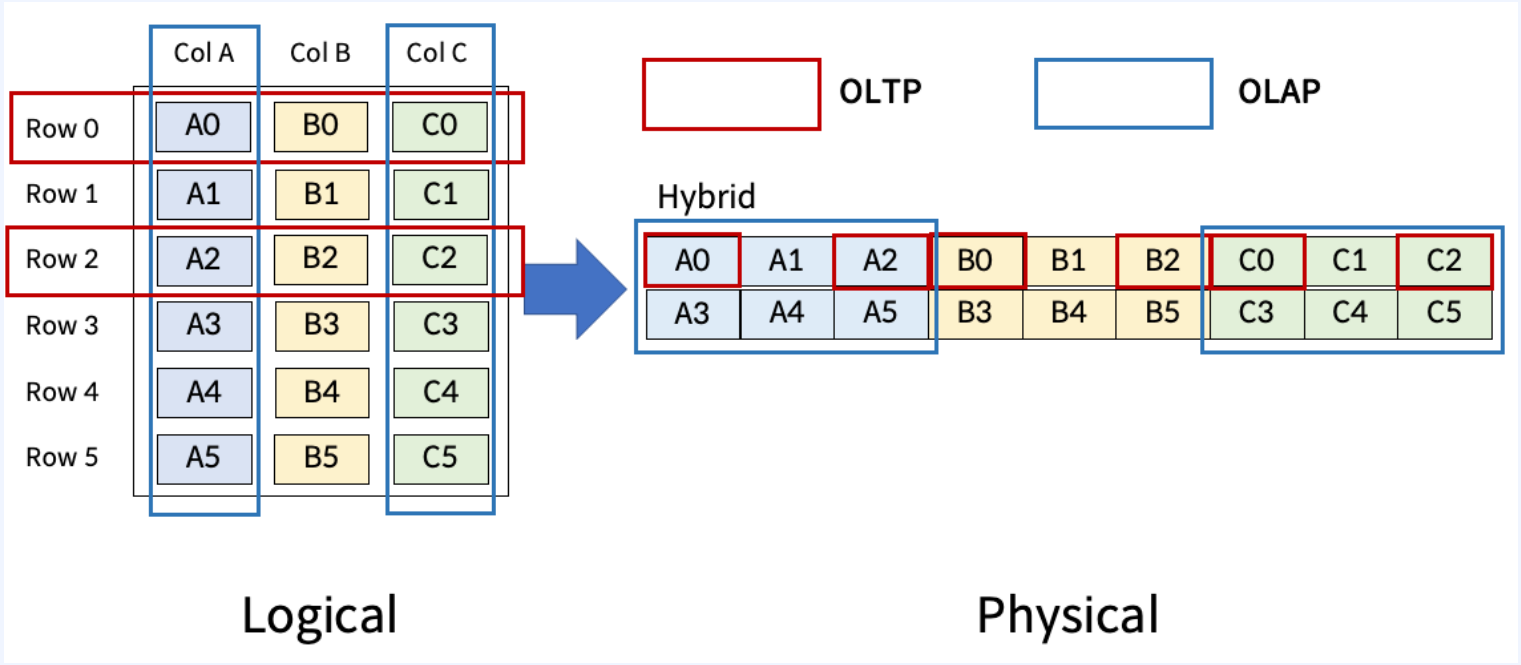
- Horizontal partitioning
- **OLTP에 적합**
- I/O 비용이 비쌀 때, OLAP에는 적합 X (OLAP에서는 특정 column만 가져오는 유즈 케이스가 많은데, Row-wise에서는 전체 Row를 우선 저장소로부터 가져와야 함. -> 낭비)

2.4 Background) Columnar



- Vertical partitioning
- **OLAP 에 적합** (Column별로 순차적으로 모여있어 특정 Column 값만 불러 오기 용이)
  - + Compression (압축) 용이
- OLTP 적합 X (특정 Row를 불러오기 위해 메모리의 여러 지점들을 확인해야 함)

2.5 Background) Hybrid

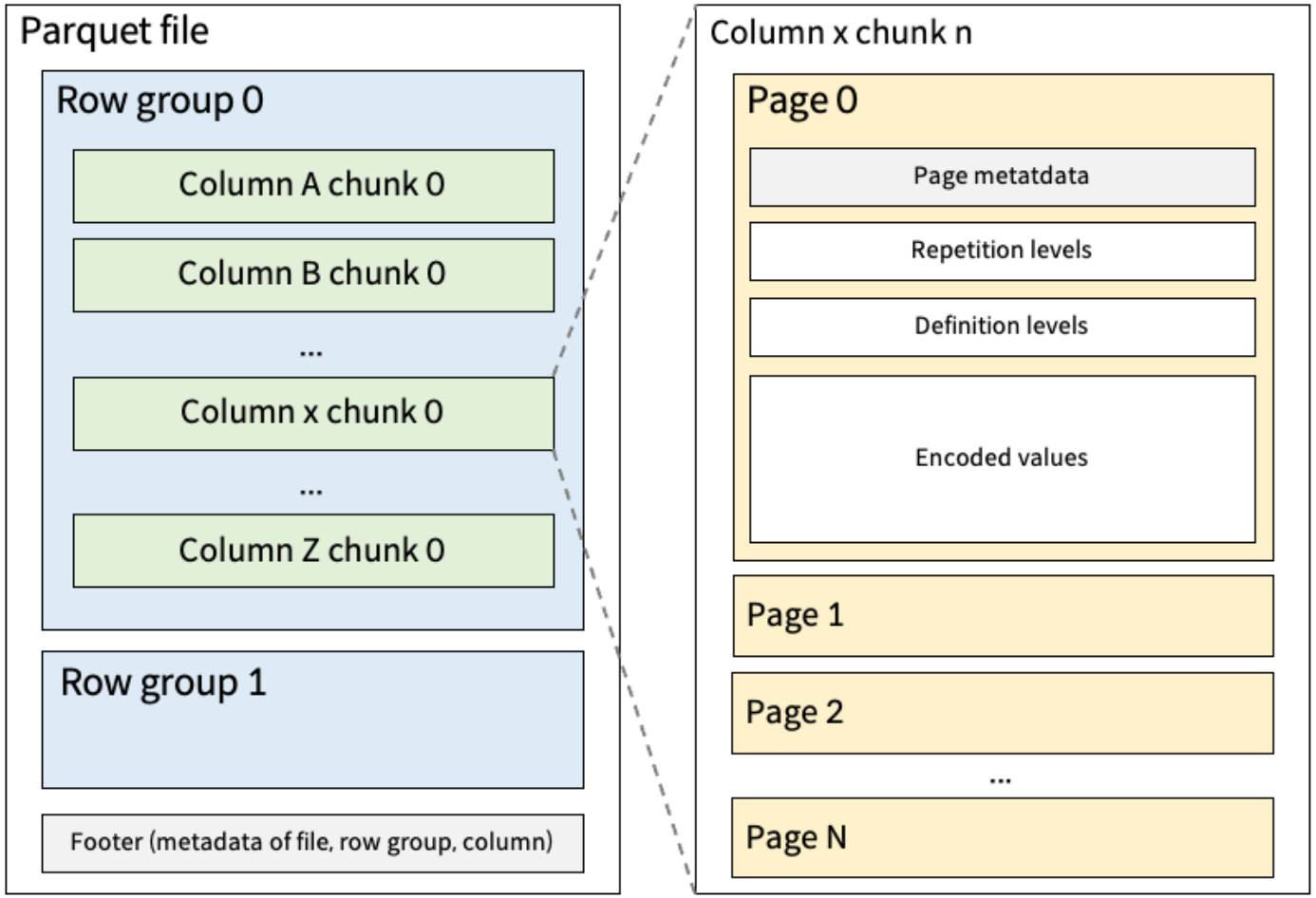


- Horizontal & Vertical partitioning
- Parquet, ORC 파일 포맷에서 사용하는 방식.

3. Apache Parquet의 구성

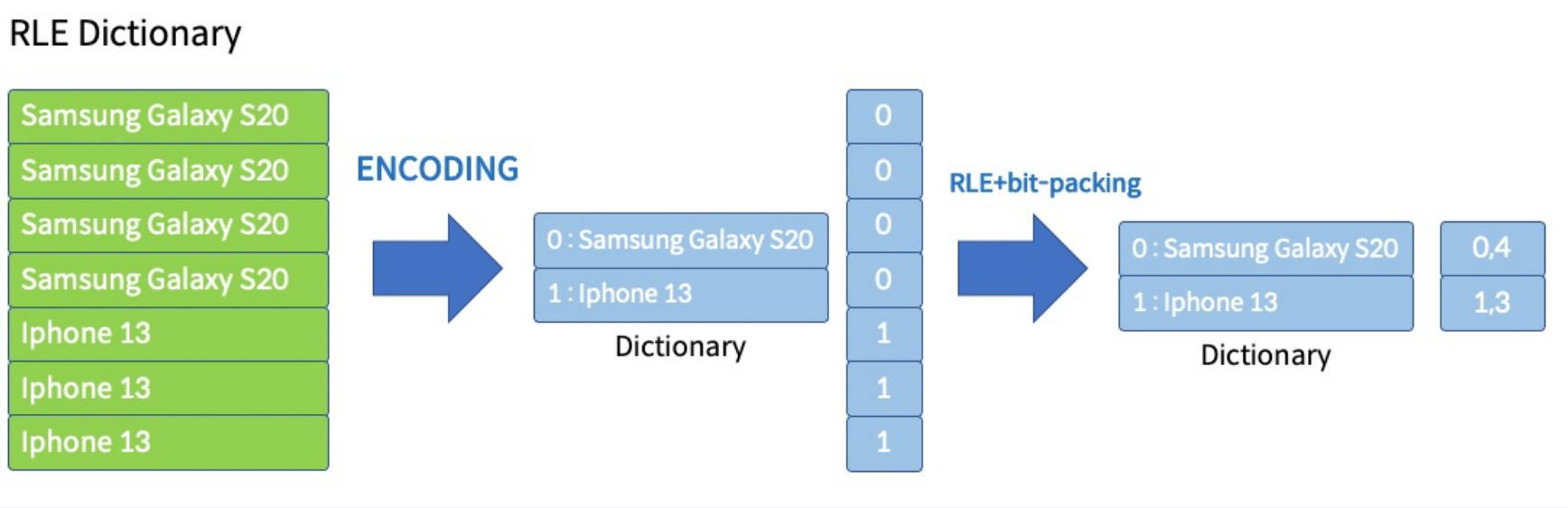
- Row groups
- Column chunks
- Pages (default : 1MB)
  - Metadata
    - Min, Max, Count, Sum etc)
  - Rep/Dep levels (nested column을 인코딩 하기 위해 필요)
  - Encoded values

- 공식 문서 링크)
  - https://parquet.apache.org/docs/



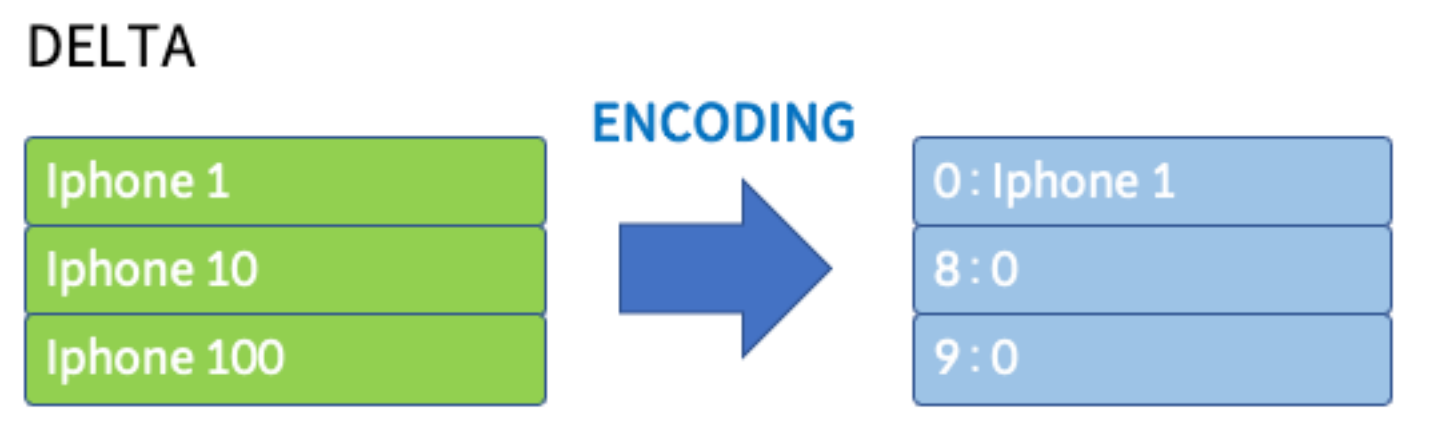
4. Encoding in Parquet

- **반복되는 data를 찾은 후, 더 작은 것으로 변환하는 과정.**
- Parquet에서는 여러 가지 방법의 Encoding 방법 제공.
  - https://parquet.apache.org/docs/file-format/data-pages/encodings/
  - Ex) RLE Dictionary Encoding (RLE : run-length encoding)



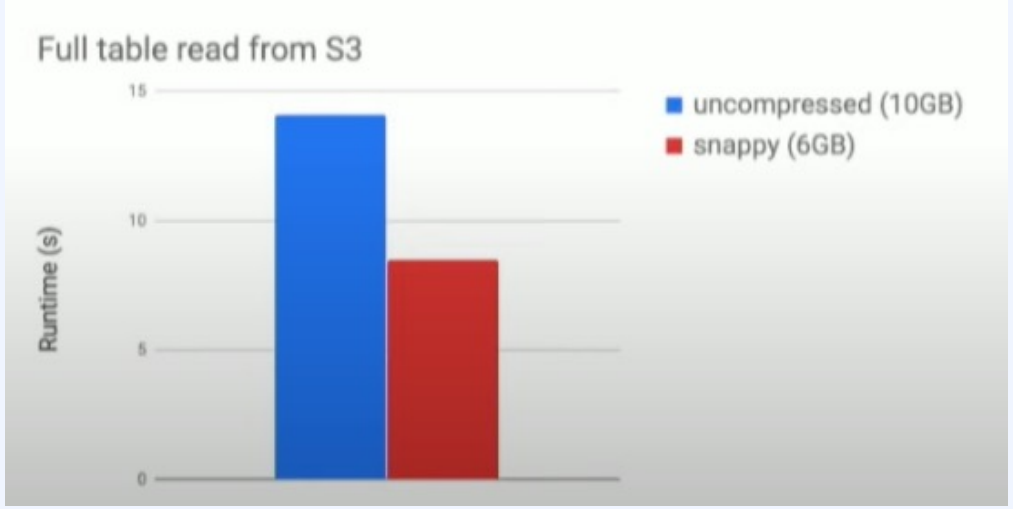
4. Encoding in Parquet

- Delta Encoding.



5. Compression (압축)

- snappy, gzip, lzo 등 여러 가지 압축 방법 존재.
  - spark 에서는 spark.sql.parquet.compression.codec 파라미터로 세팅 가능.
- Decompression speed vs I/O savings trade-off
- 일반적으로, 압축되지 않은 데이터 대비 압축된 데이터에서 Full Read 속도가 더 빠름.



https://www.youtube.com/watch?v=1j8SdS7s_NY

5. Parquet 사용 시 가능한 최적화

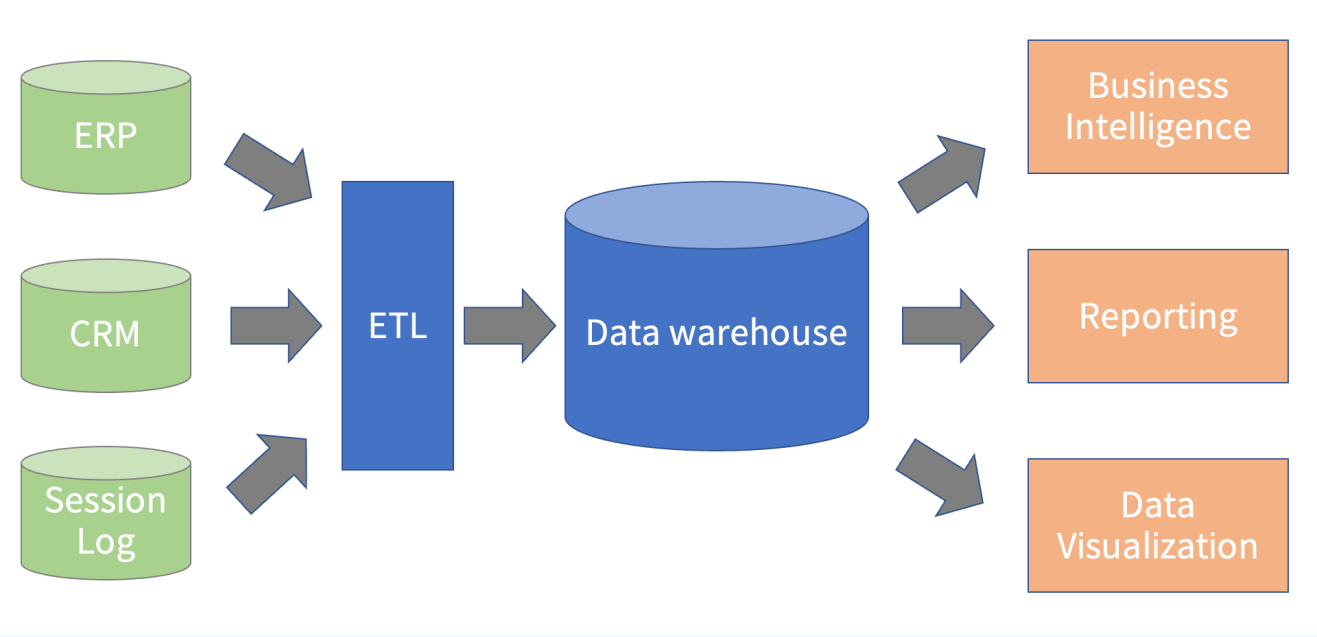
- Predicate Pushdown
- Partitioning output file

(세부 내용 : Part 9 Spark : DPP - Dynamic Partition Pruning에 대한 이해)

2. Data warehouse, lake, lakehouse 개념 소개 및 비교

1. Data warehouse 정의, 특징

- 한 조직 내의 다양한 데이터 소스로부터 기원하는 대량의 데이터를 저장하는 통합 스토리지.
- BI (Business Intelligence), 데이터 시각화, Reporting 등의 목적으로 사용.
- OLAP
- Ex) Amazon Redshift, BigQuery



1.1 Data warehouse의 한계

- Data Flexibility (유연성)의 부족
  - Structured Data에 대해서는 잘 처리하지만, 점점 더 더양하지는 Semi-Structured, UnStructured Data 소스들에 대응하는데는 어려움이 있음.

- 높은 구현, 유지 비용 (뒤에 나오는 Data lake 대비)

2. Data Lake 정의, 특징, 장단점

- ETL 프로세스를 통해 어느 정도 정제된 데이터를 적재하는 warehouse와는 달리, 다양한 종류의 데이터 (Structured, Semi Structured, Unstructured)를 raw, original, unformatted 상태로 저장하는 스토리지
- ex) AWS S3, Glacier, Google Cloud Storage, Azure Data Lake Storage.
- Data warehouse와 장단점이 서로 반대라고 이해하면 편하다.
  - 장점
    - Data Flexibility (유연성)
    - Data warehouse 대비 낮은 비용.
    - 더 다양한 목적 (data science, machine learning 등)에 활용 가능.
  - 단점
    - BI, 분석 목적으로 쓰기에는 성능이 좋지 않고, 데이터 일관성, ACID transcation 보장 X
    - 보안에 더 취약.

3. Data Lakehouse

- Data Warehouse와 Data Lake의 장점만을 취하고자 만들어진 새로운 스토리지.
  - Data Flexibilty 확보 (Unstrucured, Semi-Structured, Structured)
  - ACID transaction 보장
  - 낮은 비용
  - 높은 읽기, 쓰기 성능

- ex) Apache Hudi, Delta Lake, Apache Iceberg.

- 위의 장점들을 갖추었다고 해도 Database를 대체할 수는 없음. 둘은 서로 다른 목적으로 사용!
  (읽기, 쓰기 성능이 좋아도 Database와 비교할 수는 X)
  - OLTP vs OLAP