# Как класть Parquet

**Федор Лаврентьев @ Front Tier**

**t.me/fediq**

# План

- Что за паркет?
- Зачем всё это?
- Сильные и слабые стороны
- Что можно покрутить
- Как делать нельзя
- А есть что-нибудь получше?

# Apache Parquet

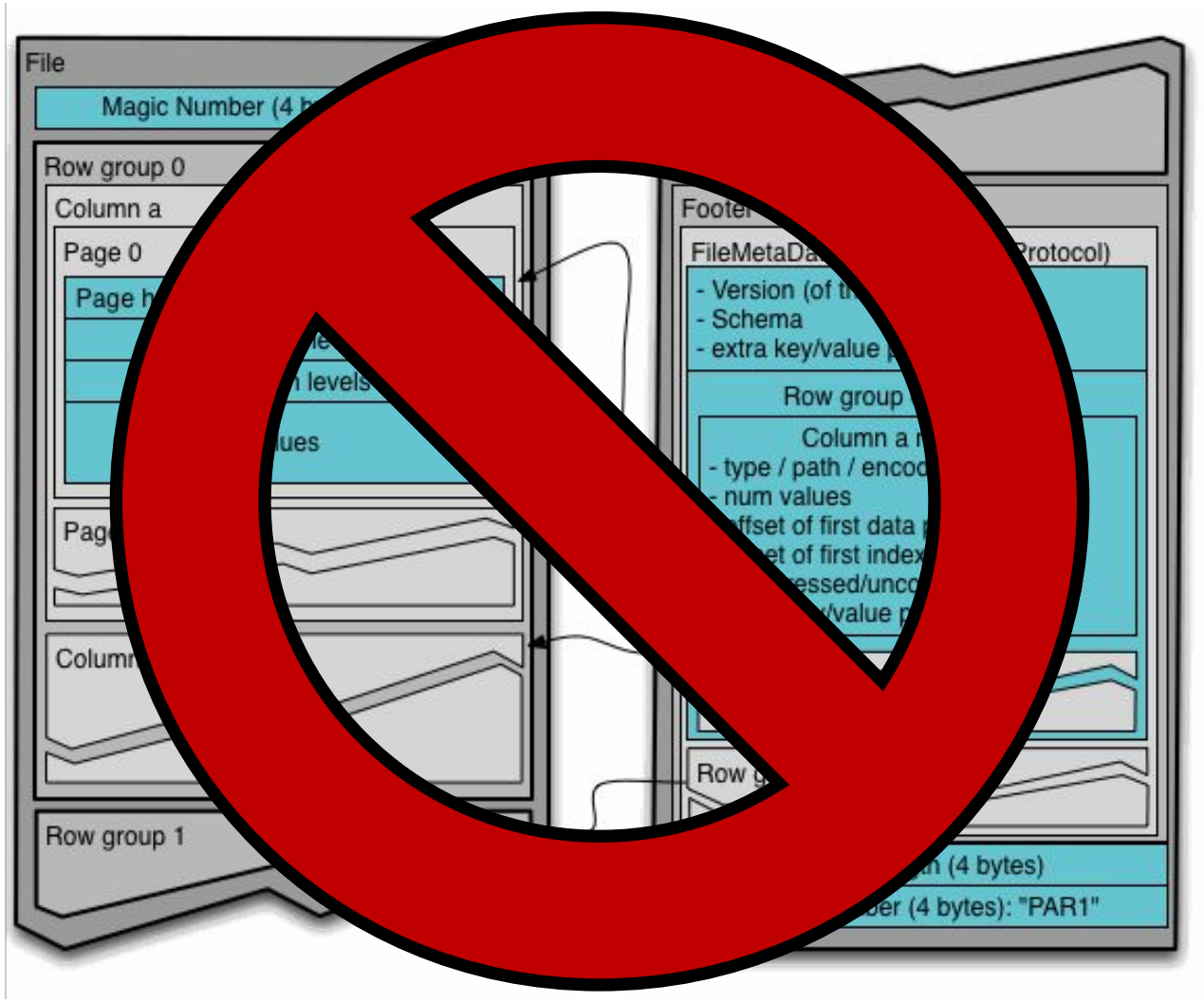
is a **columnar storage format**

available to any project in the Hadoop ecosystem

regardless of the choice of data processing framework, data model or programming language.

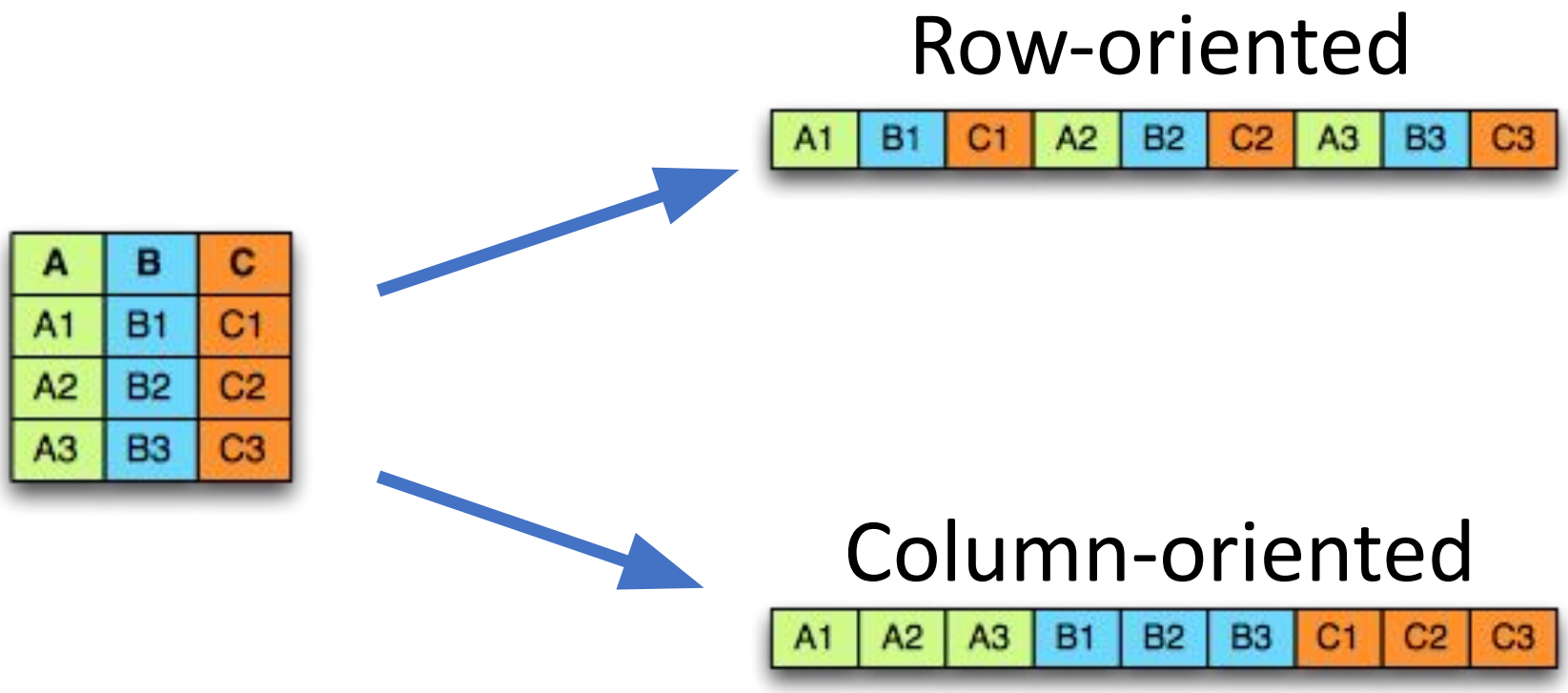
https://parquet.apache.org/

# Как устроен




https://github.com/apache/parquet-format

# Колоночное хранение




https://habr.com/company/wrike/blog/279797/

https://en.wikipedia.org/wiki/Column-oriented_DBMS

# Вложенные структуры

```
DocId: 10                    r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

**DocId**

| value | r | d |
|---|---|---|
| 10 | 0 | 0 |
| 20 | 0 | 0 |

**Name.Url**

| value | r | d |
|---|---|---|
| http://A | 0 | 2 |
| http://B | 1 | 2 |
| NULL | 1 | 1 |
| http://C | 0 | 2 |

**Links.Forward**

| value | r | d |
|---|---|---|
| 20 | 0 | 2 |
| 40 | 1 | 2 |
| 60 | 1 | 2 |
| 80 | 0 | 2 |

**Links.Backward**

| value | r | d |
|---|---|---|
| NULL | 0 | 1 |
| 10 | 0 | 2 |
| 30 | 1 | 2 |

**Name.Language.Code**

| value | r | d |
|---|---|---|
| en-us | 0 | 2 |
| en | 2 | 2 |
| NULL | 1 | 1 |
| en-gb | 1 | 2 |
| NULL | 0 | 1 |

**Name.Language.Country**

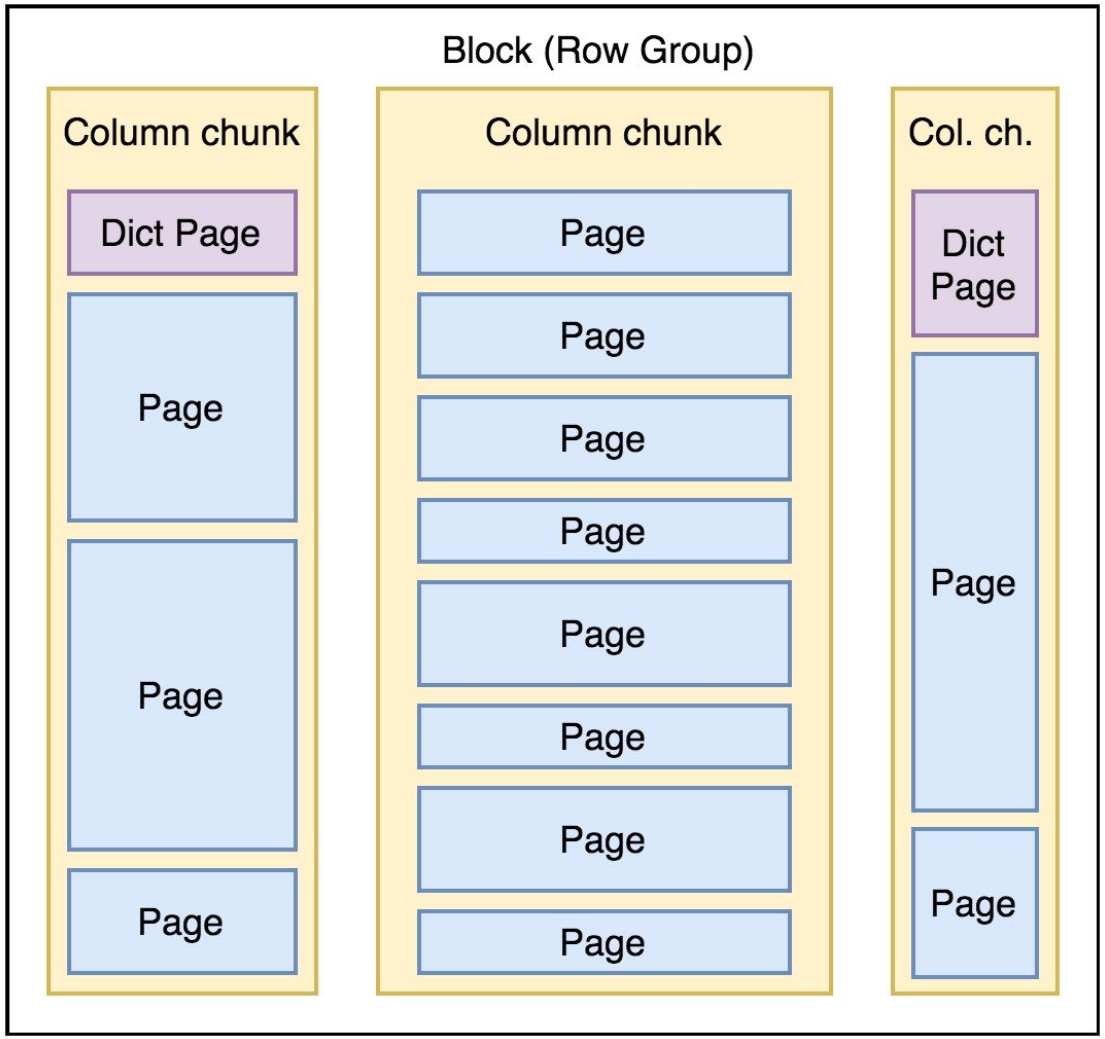
| value | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| NULL | 0 | 1 |

https://ai.google/research/pubs/pub36632

https://habr.com/post/207234/

https://stackoverflow.com/questions/43568132/

# Колоночное хранение в Parquet



# И что?

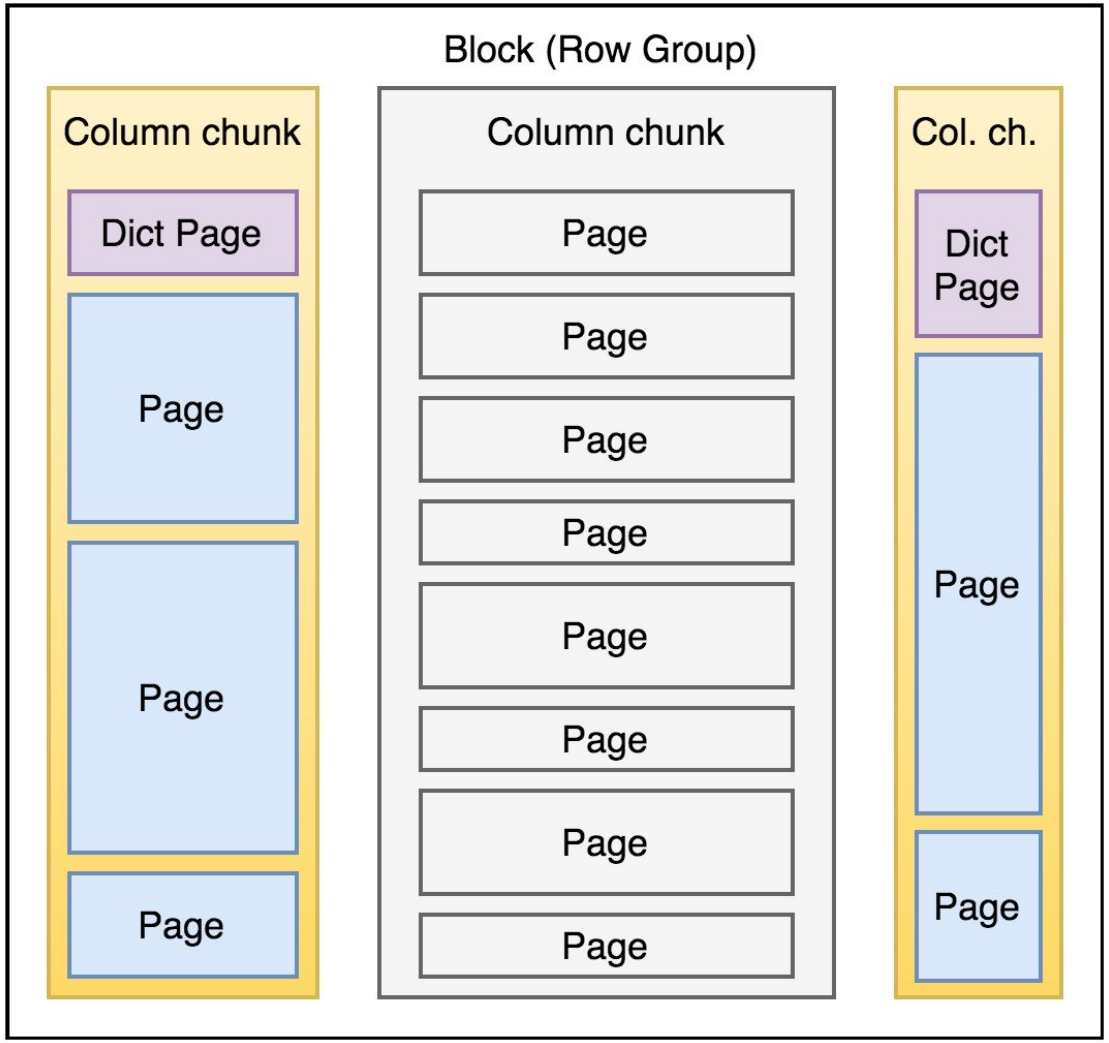
# Pages можно кодировать!

- Bit packing
- Run-length encoding (RLE)
- Delta encoding
- Dictionary encoding
- ТОДО картинки

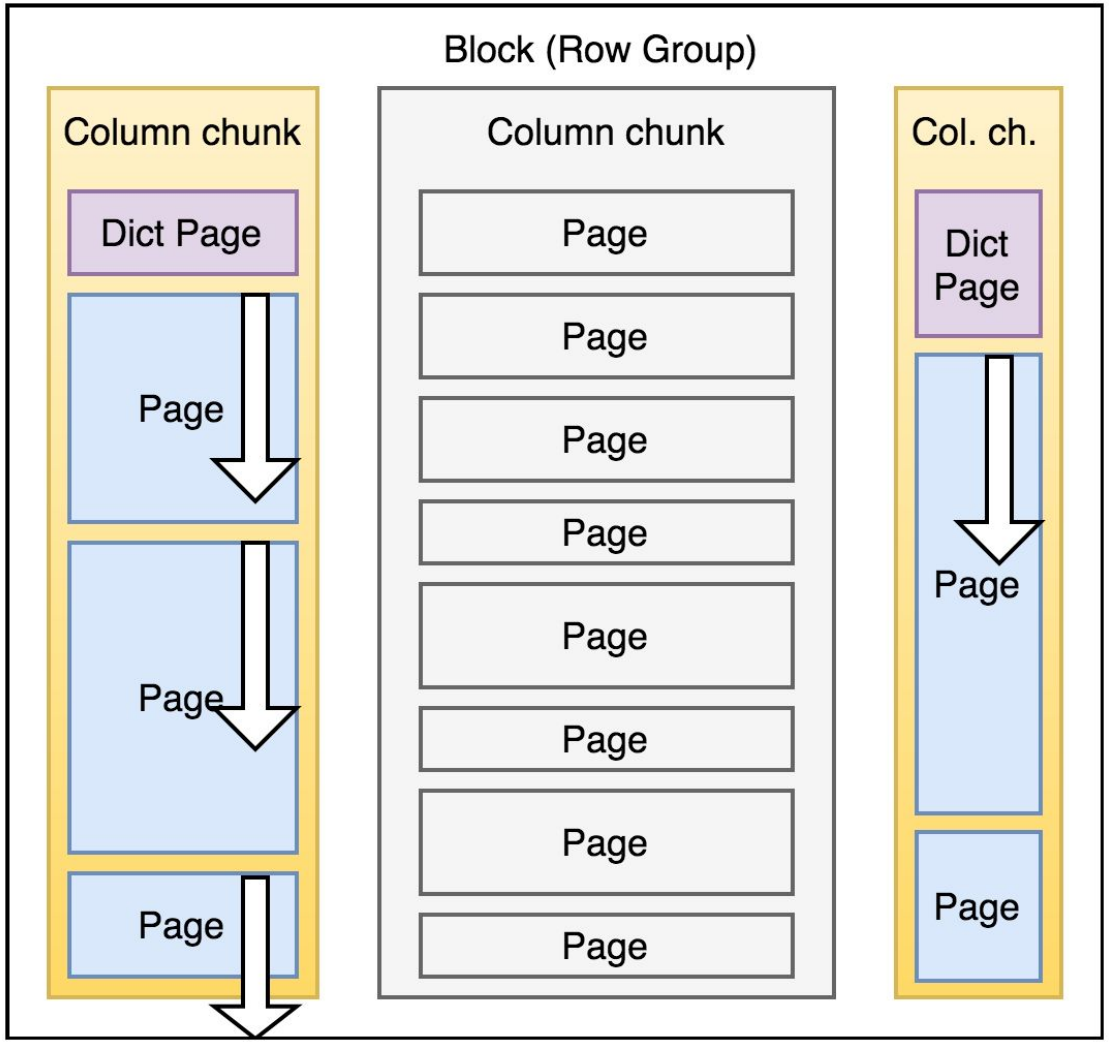
# Кодированные Pages можно сжимать!

- GZIP
- LZO
- Snappy

# Ненужные колонки можно не читать!



# Нужные колонки можно читать параллельно!



# Срочно в продакшен!..

# Выбрасываем легаси!..

- Postgres VS Parquet
- MongoDB VS Parquet
- Cassandra VS Parquet
- Excel VS Parquet

# Write once!

- Append only
- Read only
- Только батчевая запись

- Нет транзакций. Никак. Совсем.

# Подходит для аналитики

- Только чтение
- Большие range scan’ы
- Сложные фильтры
- Группирующие запросы

# Большие Range Scan’ы

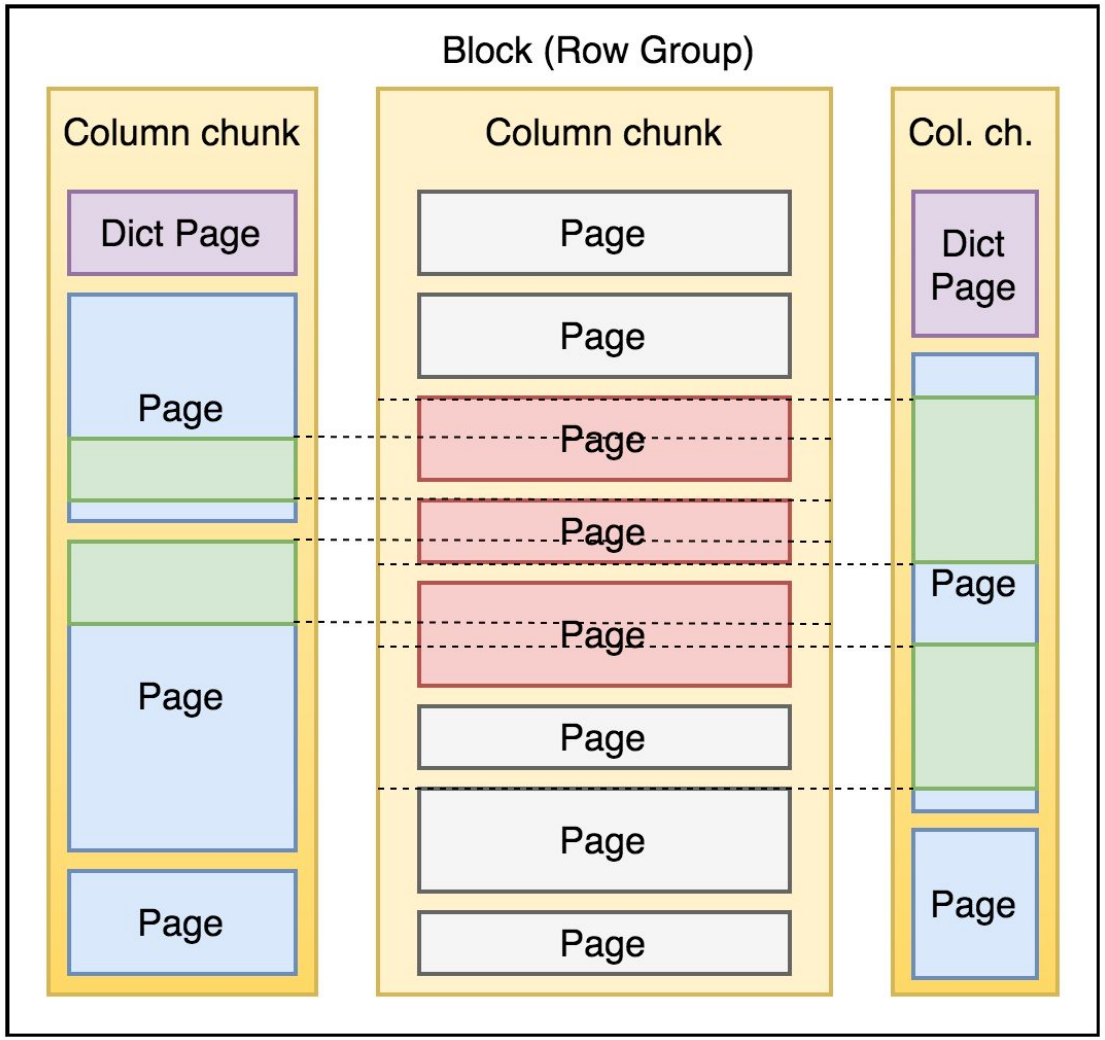
⇒ Партиционируйте данные!

/dataset/2018/07/30/

/dataset/2018/08/01/

/dataset/2018/08/02/

# Сложные фильтры



# Predicate Pushdown

- Только простые условия (<, >, ==, IN, null)
- Заранее заданные константы
- Можно комбинировать логически (OR, AND, NOT)

+ Специальные условия для строк (StartsWith)

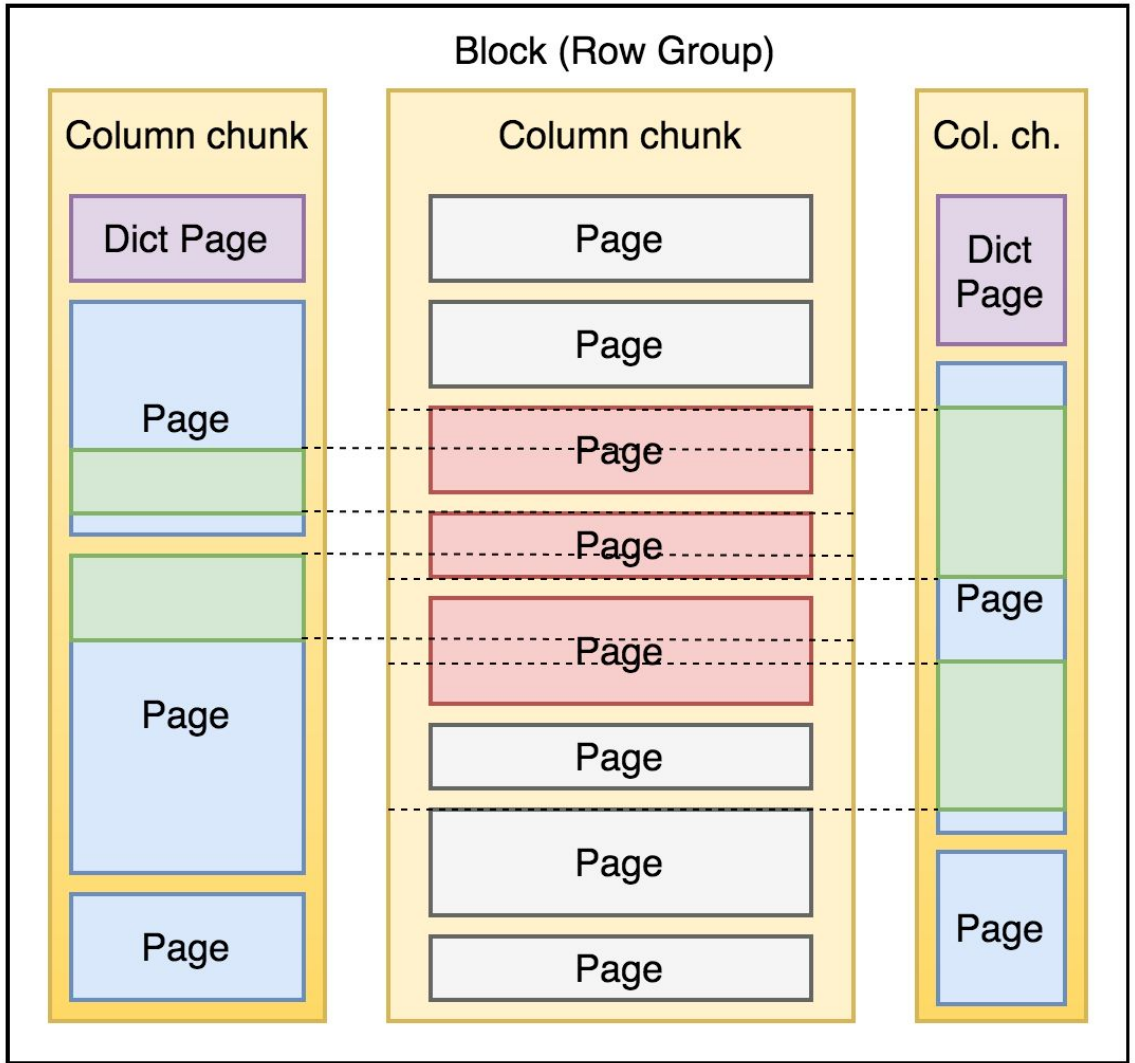
https://db-blog.web.cern.ch/blog/luca-canali/2017-06-diving-spark-and-parquet-workloads-example

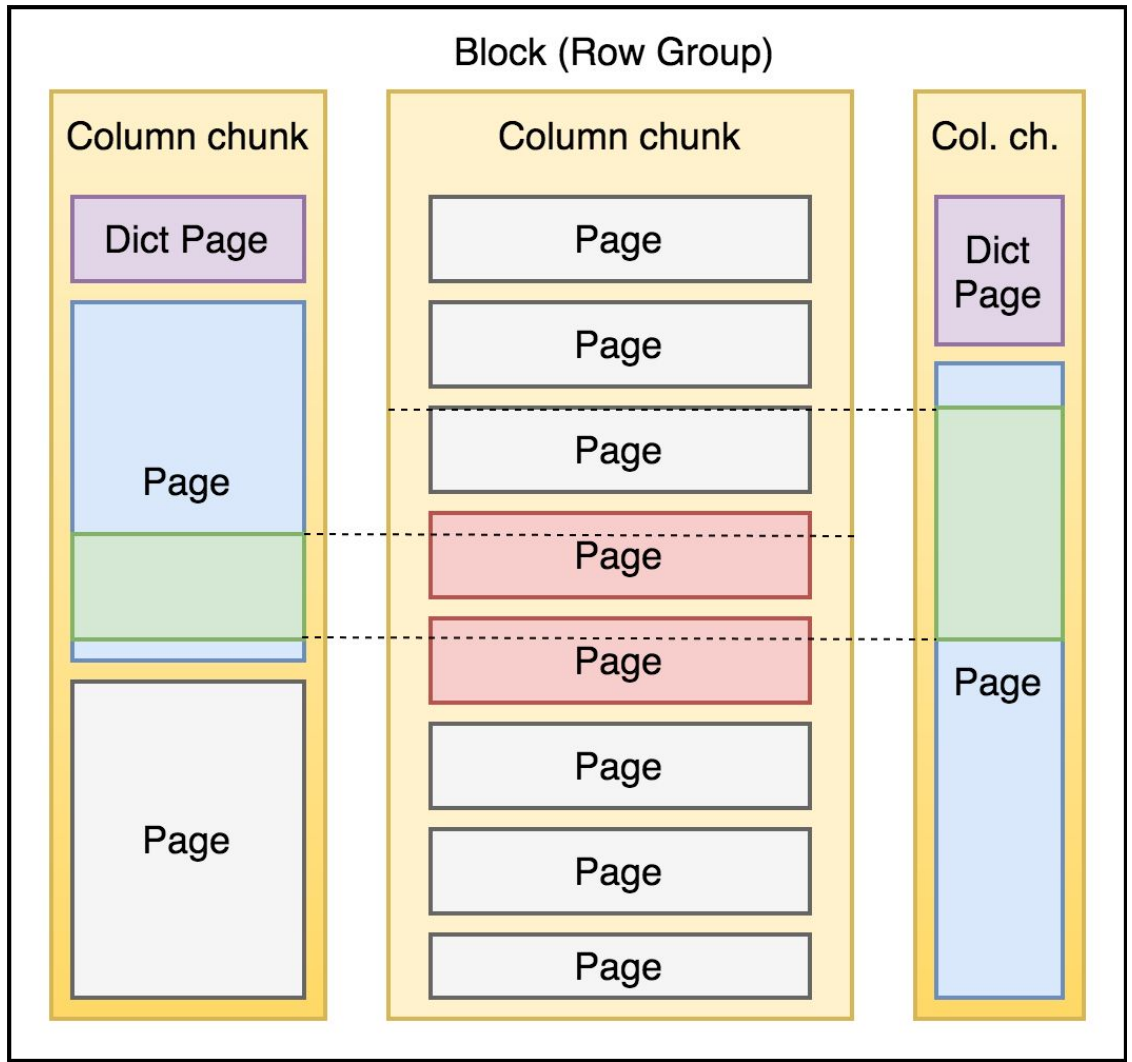
# Predicate Pushdown

Не эффективен для чтения одной строки!

select * from table where id = 1234

# Pred. Pushdown + Encoding + Sorting



# А теперь можно в продакшен?

# Не всё так просто!

Приходится:

- Сортировать и партиционировать
- Оптимизировать типы
- Контролировать размеры

# Оптимизация типов

- Чем меньше тип – тем лучше

- XML, JSON -> Infer schema -> Struct
- Plain text -> Parsed struct
- float -> int (?)

# parquet.block.size

- Больше => лучше сжатие
- Больше памяти при записи
  - Требует x2-x3 памяти
- Должен умещаться в HDFS блоке

64 МБ – 1 ГБ (128 МБ)

https://www.slideshare.net/RyanBlue3/parquet-performance-tuning-the-missing-guide

# Несколько блоков в файле?

- Формат позволяет
- Но нарушается граница HDFS блоков

# Один файл – один блок!

***parquet.block.size == dfs.block.size***

- Делайте repartition перед записью
- Держите 10-20% запас

# Repartition - до

```
df
  // 200 tasks
  .write
  .parquet(path)

!hdfs dfs —ls —h $path | tail -1
180.9M
/dataset/part-00199-hash.parquet
```

# Repartition – после

```
df
  .repartition(320)
  .write
  .parquet(path)

!hdfs dfs —ls —h $path | tail -1
118.3M
/dataset/part-00319-hash.parquet
```

# parquet.page.size

- Больше => лучше сжатие
- Меньше => эффективнее Predicate Pushdown
- Читается целиком в память

8 кб – 1 МБ (1 МБ)

https://www.slideshare.net/RyanBlue3/parquet-performance-tuning-the-missing-guide

# parquet.dictionary.page.size

- Одна страница dictionary на колонку
- Держится в памяти целиком при чтении
- Увеличивайте при работе с повторяющимся строками

(1 МБ)

https://www.slideshare.net/RyanBlue3/parquet-performance-tuning-the-missing-guide

# Теперь-то всё будет хорошо!..

# Spark Streaming – Append

```
stream
  .write
  .mode(Append)
  .parquet(path)
```

# Много маленьких файликов

⇒ Много HDFS блоков
- Неэффективное использование DataNode
- Высокая нагрузка на NameNode

⇒ Много Spark Tasks
- Большой оверхед на старт
- Нагрузка на мастер

# 1. Убирайте партиции

```
stream
  .coalesce(1)
  .write
  .mode(Append)
  .parquet(landPath)
```

https://evoeftimov.wordpress.com/2017/08/29/spark-streaming-parquet-and-too-many-small-output-files/

# 2. Перекладывайте потоки

```
spark
  .read.parquet(landPath)
  .repartition(partitions, key)
  .sortWithinPartitions(keys)
  .write.parquet(path)
```

https://evoeftimov.wordpress.com/2017/08/29/spark-streaming-parquet-and-too-many-small-output-files/

# Spark VS Impala VS Hive

- Кто быстрее?
- Кто совместим?!

# Имплементации Parquet

1. apache/parquet-mr (Java)
2. apache/parquet-cpp (C++)
3. Spark Catalyst (Scala)
4. Dask/fastparquet (Python)

# Decimal

- Не читается

**spark.sql.parquet.writeLegacyFormat**

https://issues.apache.org/jira/browse/IMPALA-2494
https://issues.apache.org/jira/browse/SPARK-10400
https://issues.apache.org/jira/browse/SPARK-6777
https://issues.apache.org/jira/browse/SPARK-20937
https://issues.apache.org/jira/browse/SPARK-20297

https://stackoverflow.com/questions/44279870/

# Timestamp

- Не читается
- Теряет таймзону

**spark.sql.parquet.writeLegacyFormat**
**spark.sql.parquet.int96AsTimestamp**
**spark.sql.parquet.outputTimestampType**
**spark.sql.parquet.int64AsTimestampMillis**

https://github.com/apache/spark/blob/master/sql/catalyst/src/main/scala/org/apache/spark/sql/internal/SQLConf.scala

https://issues.apache.org/jira/browse/HIVE-12767
https://issues.apache.org/jira/browse/HIVE-13534
https://issues.apache.org/jira/browse/SPARK-12297

# JSON/BSON

- Не нужен
- Но там всё равно что-то не работает

https://gist.github.com/squito/f348508ca7903ec2e1a64f4233e7aa70 https://issues.apache.org/jira/browse/SPARK-16216

# Spark Legacy Format

- Был дефолтным до Spark 1.5.9
  - Примерно 2015ый
  - Примерно Parquet 1.6.x
- Не задокументирован
  - SPARK-20937 «Может, всё-таки задокументируем?»

***spark.sql.parquet.writeLegacyFormat***

**==**

***!spark.sql.parquet.followParquetFormatSpec***

https://issues.apache.org/jira/browse/SPARK-20937

# Impala?

- Ищет колонки по номерам, а не по именам
- Не поддерживает LZO
- Не поддерживает  binary
- Проблемы с Decimal и Timestamp

***spark.sql.parquet.writeLegacyFormat***
***spark.sql.parquet.binaryAsString***

https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_parquet.html

# Sqoop?

**Не используйте Parquet в Sqoop!**

- Не умеет repartitioning
- OOM’ы

# ColumnWriter.writePage()

- Проверки, что пора писать очередной Page:
  - initial-page-run check
  - next-page-size check
- Когда можно ошибиться:
  - Строки большие с самого начала
  - Строки маленькие, но есть несколько больших

https://issues.apache.org/jira/browse/PARQUET-99
https://www.cloudera.com/documentation/enterprise/release-notes/topics/cdh_rn_sqoop_ki.html

# Если совсем нельзя обойти

***parquet.page.size.row.check.min***
***parquet.page.size.row.check.max***
***parquet.page.size.check.estimate***

https://github.com/apache/parquet-mr/blob/master/parquet-column/src/main/java/org/apache/parquet/column/impl/ColumnWriterV1.java

# Schema Merging

- _metadata, _common_metadata
- Можно просто отключить

***parquet.enable.summary-metadata=false***

***spark.sql.parquet.mergeSchema=false***

https://stackoverflow.com/questions/36739940/

# А теперь – дьявольщина!

- Parquet буферизируется в памяти
- Контрольные суммы не предусмотрены
- Память может биться

⇒ Могут биться файлы

https://lists.apache.org/thread.html/495804dc92af580486c146fdb8e58d89d06d204b611bcc1e911c4fdc@%3Cdev.spark.apache.or

# А нормального ничего нет?

# Напомните, зачем нам Parquet?

- Экономия по месту
- Быстрая фильтрация
- Чтение по частям
- HDFS-native

- Очень дорогая запись

# Может, CSV?

- Человекочитаемый
- Нет оверхеда для текстов
- Поддерживает append

- Бейзлайн по ужасности
- Плоская структура

# Может, JSON?

- Человекочитаемый
- Schema-free

- Еще более медленный и жирный

# А XML?

# Avro?

- Поддерживает append
- HDFS-native
- Продвинутая эволюция схем

- Менее производителен, чем Parquet

# Parquet VS ORC – всё сложно

- По объему и скорости однозначного лидера нет

- Hive отстаёт в поддержке Parquet
- Spark отстаёт в поддержке ORC
- Impala игнорирует существование ORC

https://medium.com/@denisgabaydulin/choosing-a-columnar-format-e49333395f92
https://www.slideshare.net/HadoopSummit/file-format-benchmark-avro-json-orc-parquet

# MPP?

Для структурированных данных
специализированные MPP-системы
на порядок быстрее Spark + HDFS + Parquet.

# Ясно, понятно…

# Parquet

- Прекрасный формат для исторических данных
- Для Spark, особенно на CDH – альтернатив нет
- Имеет массу тонкостей
  - Эффективность варьируется на порядок

# Главные тонкости

- Пиши один раз, читай много
- Структурируй данные
- Партиционируй и перекладывай потоки
- Оптимизируй размеры
- Не верь в интероперабельность

# Спасибо!

Федор Лаврентьев

t.me/fediq