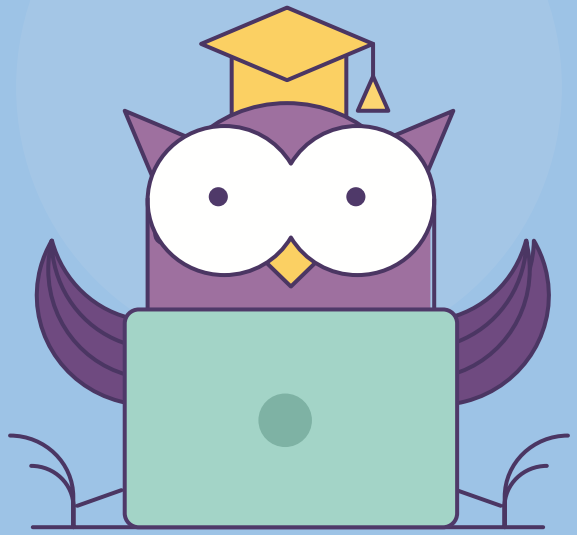
OTUS

Инженер Данных.
Форматы данных.

Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте + если все хорошо
Ставьте - если есть проблемы

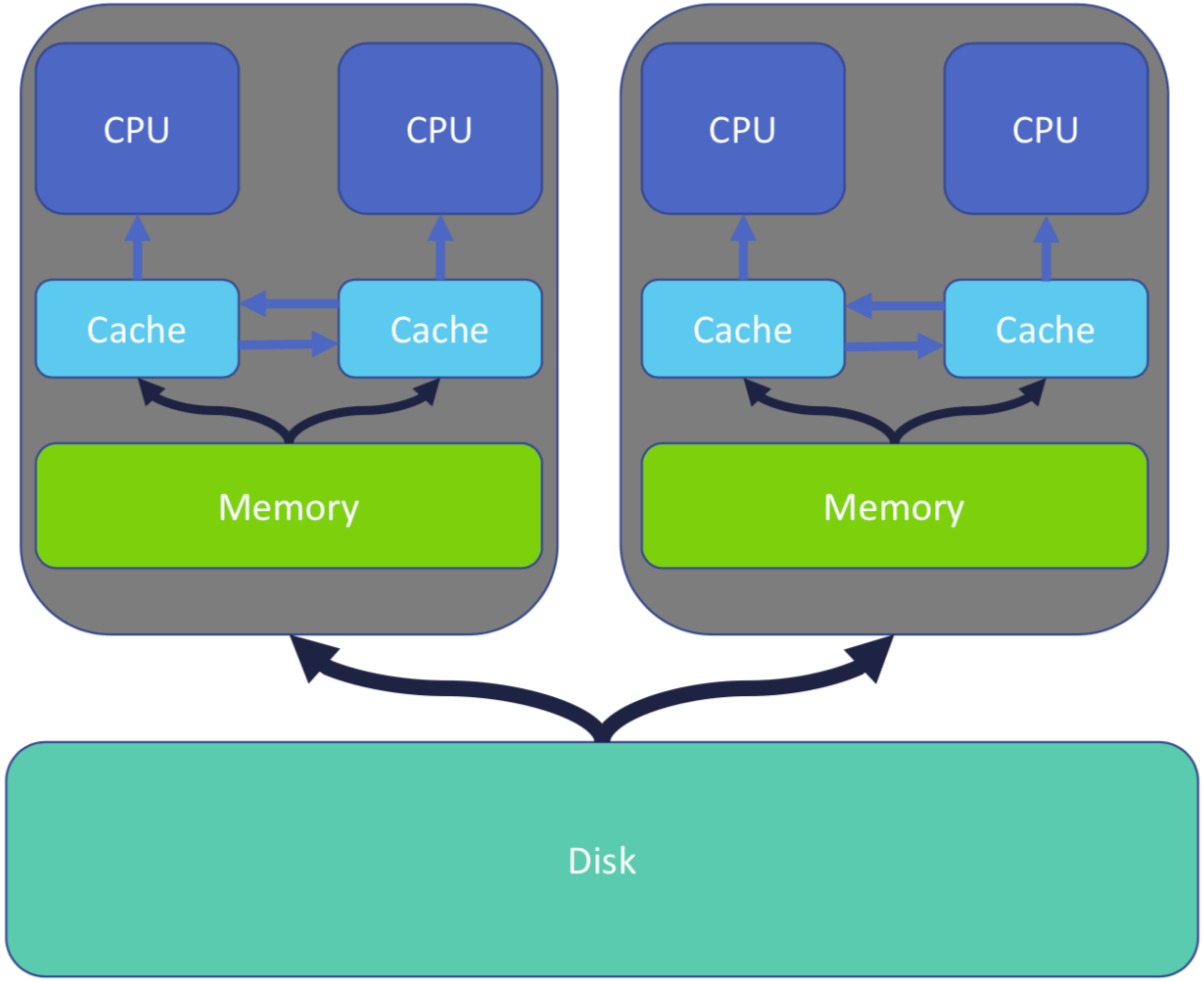
План занятия

- Требования к форматам
- Факторы выбора
- Текстовые форматы
- AVRO
- ORC
- Parquet

Специфика задачи

- Большие данные (100+ GB, TB, PB)
- Распределенные вычисления
- Write once, read many
- ETL, Analytics, Machine Learning, IoT, и т.д.

Слабые места – Disk & Network IO



Слабые места – Disk & Network IO

Action	Computer Time	"Human Scale" Time
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	**50-150 µs**	**2-6 days**
Rotational disk I/O	**1-10 ms**	**1-12 months**
Internet: SF to NYC	**40 ms**	**4 years**
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years

* The Infinite Space Between Words

Требования к форматам хранения

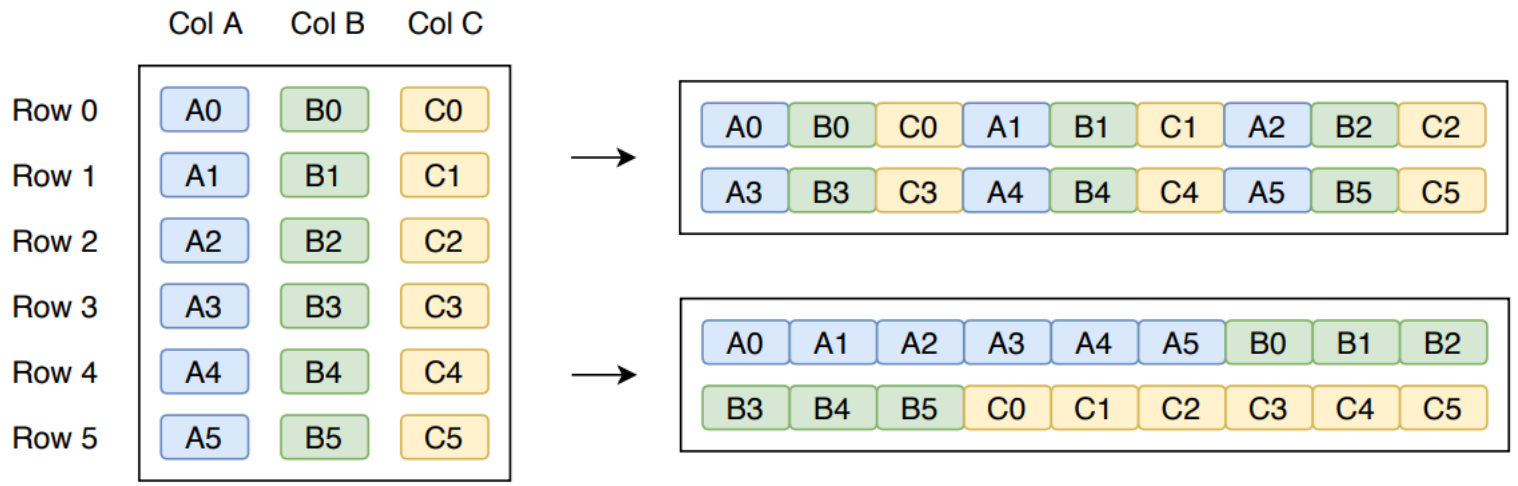
- **Быстрые запросы**
- Эффективность ресурсов
  - Место на дисках
  - Disk IO
  - Network IO
- Практичность
  - Интеграция с существующими инструментами
  - Минимальные требования к обучению
  - Schema Evolution
- Затраты
  - Разработка и сопровождение

Факторы выбора формата

- Инструменты для обработки и запросов
- Эволюция модели данных (Schema Evolution)
- Расщепление (splitability) файлов с данными
- Компрессия данных
- Время отклика (latency)
- Тип нагрузки (Write / Read)

Row vs Column

- Доступ к подмножеству колонок
- Доступ ко всем колонкам для каждой записи



Расщепление данных - splitability

- Разделение на кусочки
- Параллельная независимая обработка
- Пример неразделимых данных: большой XML-документ или JSON-запись

Сжатие - compression

- Эффективность сжатия
- Затраты на компрессию - декомпрессию

Schema evolution

- Обновление схемы
  - Добавить, Удалить, Переименовать, Типы
- Обратная совместимость
- Человекочитаемый формат
- Как быстро можно прочитать схему
- Как схема влияет на общий объем данных

Текстовые данные

Текстовые форматы

- CSV, TSV, JSON, ...
- Удобный формат: импорт, экспорт, обмен сообщениями
- Человекочитаемые данные
- Монолит, неэффективные запросы
- Нет сжатия на уровне блоков

Формат CSV

- Формат с разделителями
- Все данные хранятся как строки
- Универсальный формат

Строки на диске

Value	Binary
1	00000001
"1"	00110001
18	00010010
"18"	0011000100111000
511	111111111
"511"	001101010011000100110001
65535	1111111111111111
"65535"	00110110001101010011010100110011001100110101

Пример

Sentence Start,Descriptor,Number,\nMy dataset is,great,6234,\nMy dataset is,cool,8679,\nMy dataset is,nice,3,\nMy dataset is,,2857,\nMy dataset is,full of nulls,,\nMy dataset is,whatever,3758,\n

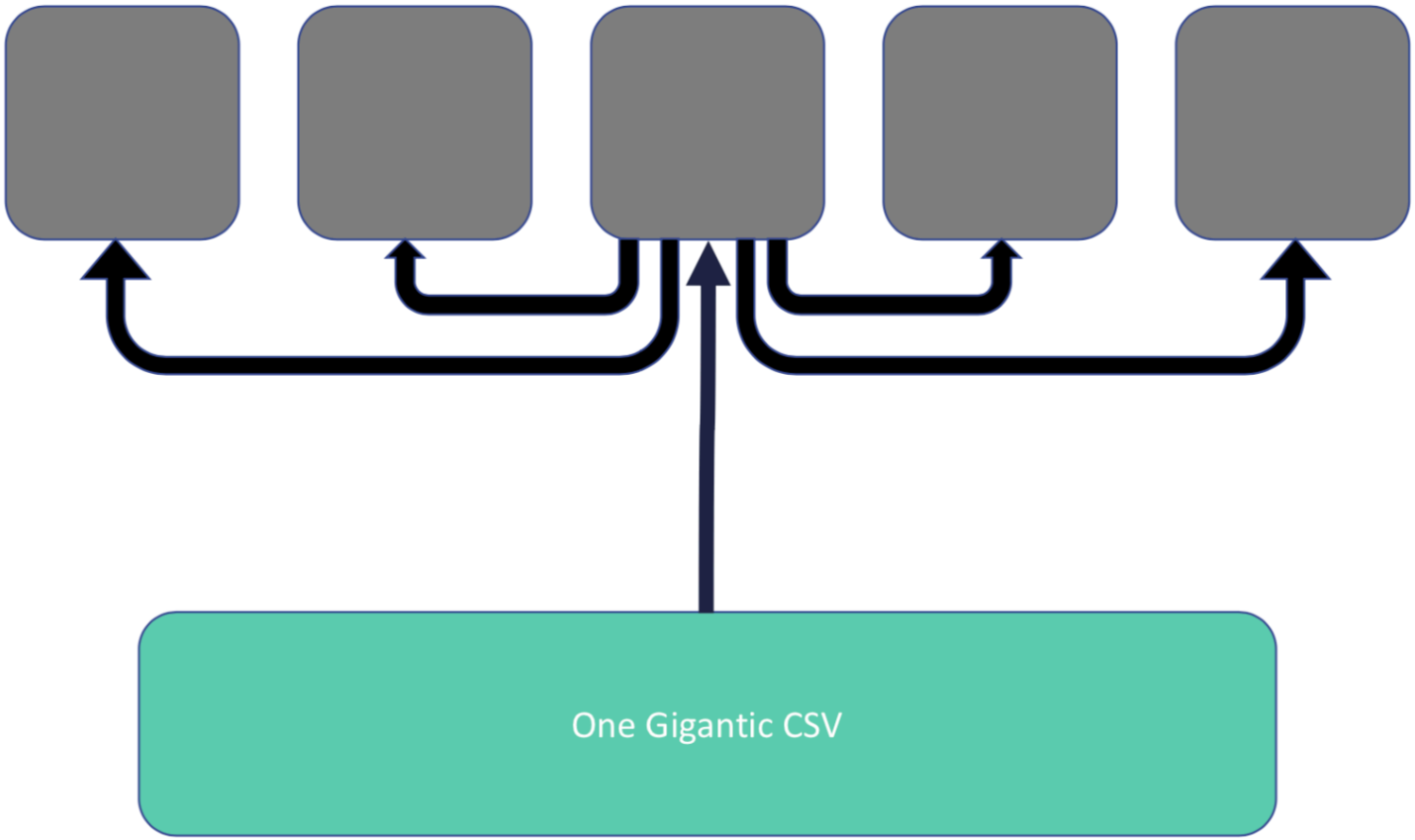
...

Недостатки

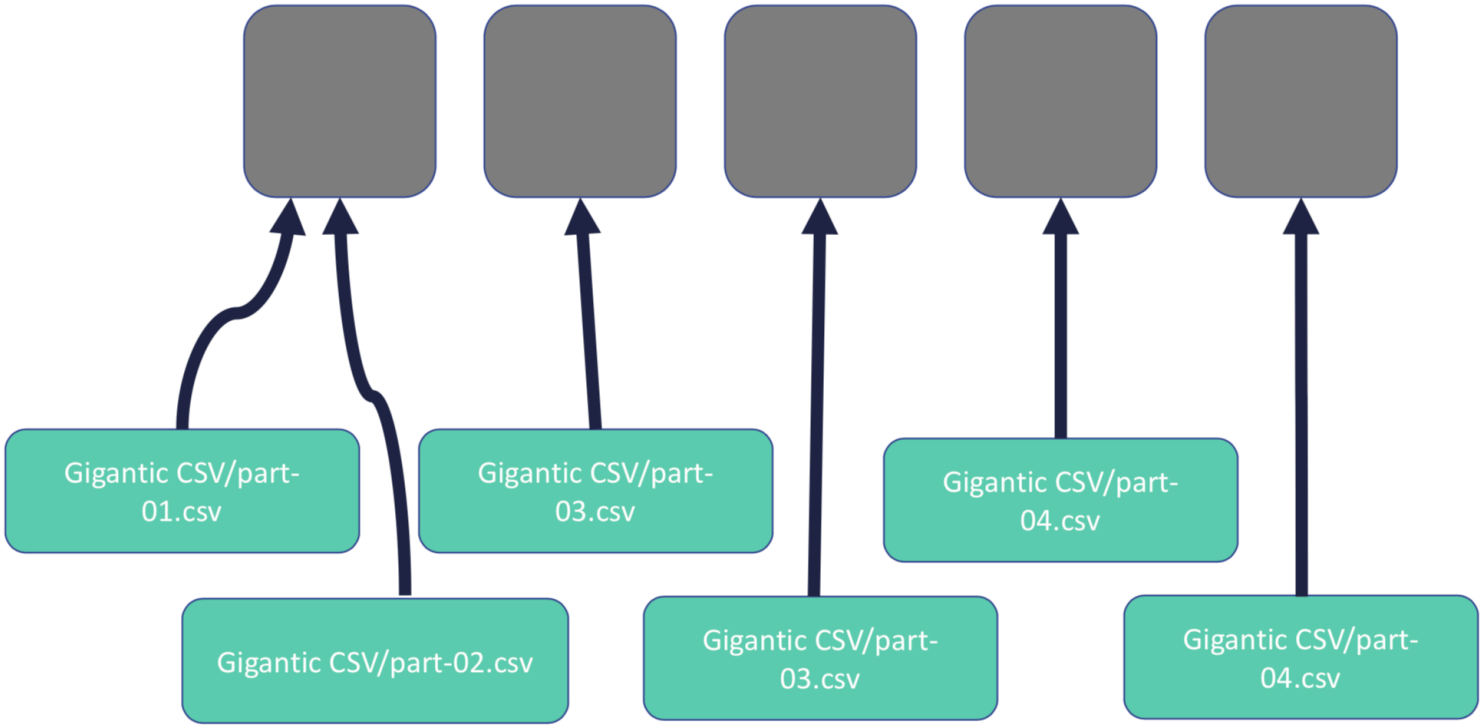
- Огромный объем данных
- Пост-обработка данных (преобразование типов)
- Линейное чтение
- Нет расщепления файлов. Что если мне нужна малая часть данных?

Как улучшить производительность любого формата данных?

Максимально плохо: один большой файл



Расщепить файл на части



Добавить сжатие

- Меньший объем на диске / по сети
- Необходима декомпрессия

JSON

JSON

- Формат сериализации для HTTP & Javascript
- Строчный формат
- Схема интегрирована с данными
- Множество парсеров

Пример

```
{
   "Sentence_Start" : "My dataset is",
   "Descriptor" : "great",
   "Number" : 6234
}
{
   "Sentence_Start" : "My dataset is",
   "Descriptor" : "cool",
   "Number" : 4367
}
{
   "Sentence_Start": "My dataset is",
   "Descriptor" : "",
   "Number" : 4367
}
```

JSON недостатки

- Огромный объем данных
- Пост-обработка данных (преобразование типов)
- Линейное чтение
- Нет расщепление файлов. Что если мне нужна малая часть данных?
- **Схема с данными. Снова и снова.**

Avro

AVRO

- Платформа сериализации данных сравнимая с Thrift and Protobuf
- Cross-language
- Строковый формат
- Схема отдельно от данных (JSON)
- Данные в бинарном формате
- Расщепление файлов

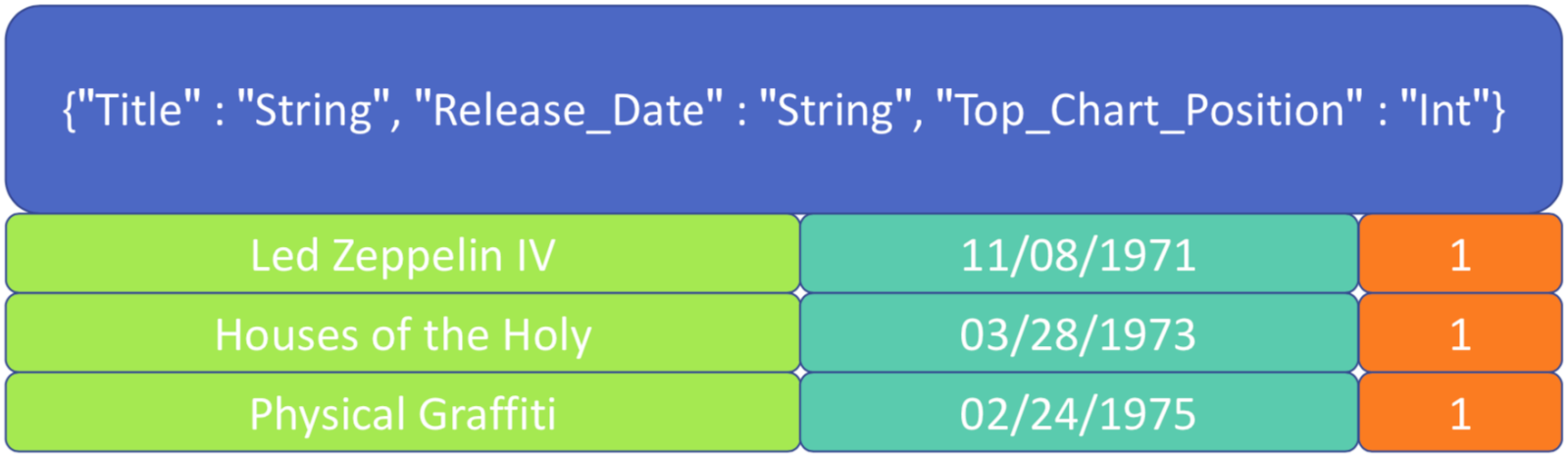
AVRO

- Schema evolution - изначальный приоритет
- Поддержка вложенных схем (это JSON)
- Сжатие на уровне блоков
- Сжатие: gzip, deflate, snappy
- Supports primitive types and wide array of complex types (Map, array, record, enum, etc)

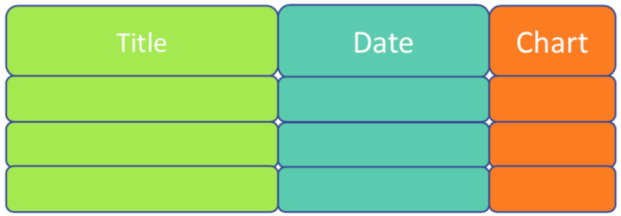
AVRO типы данных

- *primitive*
  - string
  - bytes
  - int & long
  - float & double
  - boolean
  - Null (enables optional fields)
- *complex*
  - record
  - array
  - map: string -> Type
  - union (aka “choice”)
  - fixed<N> (byte arrays, eg uuid)
  - enum: [“larry”|“moe”|“curly”]

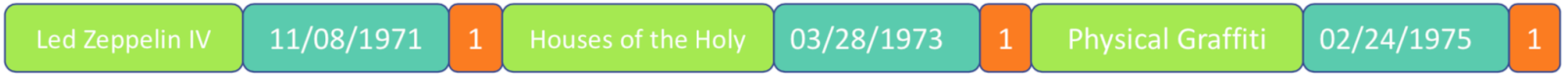
Avro: JSON Schema + Binary Data



Avro is Row Oriented



Row-Oriented data on disk



Column-Oriented data on disk

AVRO пример

```
{  "name": "loggerName",
   "type": "string“,
   "default": “unknown”,
   "doc": "The name of the logger class that
       generated the message.”},
{  "name": "message",
   "type": ["null", "string“],
   "default": null,
   "doc": "The formatted message of the log event."}
```

AVRO - хорошо 👍

- Баланс Read / Write
- Частое изменение схемы
- Всегда читаем всю строку (все колонки данных)

AVRO - плохо 👎

- Везде, где есть выборка колонок
  - Аналитика, BI
  - ML, ...
- Быстрое чтение - приоритет

Форматы с поддержкой Schema Evolution

- AVRO
- ORC
- Parquet

Пример Schema Evolution - AVRO

```
{"namespace": "drwho.avro",
 "type": "record",
 "name": "drwho",
 "fields": [
    {"name": "doctor_who_season", "type": "string"}, {"name": "doctor_actor", "type": "string"},
    {"name": "episode_no", "type": "string"}, {"name": "episode_title", "type": "string"},
    {"name": "date_from", "type": "string"}, {"name": "date_to", "type": "string"},
    {"name": "estimated", "type": "string"}, {"name": "planet", "type": "string"},
    {"name": "sub_location", "type": "string"},  {"name": "main_location", "type": "string"}
]}
```

Пример Schema Evolution - AVRO

```
{"namespace": "drwho.avro",
 "type": "record",
 "name": "drwho",
 "fields": [
    {"name": "drwho_season", "type": ["null","string"], "aliases": ["doctor_who_season"]},
    {"name": "drwho_actor", "type": ["null","string"], "aliases": ["doctor_actor"]},
    {"name": "episode_no", "type": ["null","string"]}, {"name": "episode_title", "type": ["null","string"]},
    {"name": "date_from", "type": ["null","string"]},  {"name": "date_to", "type": ["null","string"]},
    {"name": "estimated", "type": "string"}, {"name": "planet", "type": ["null","string"]},
    {"name": "sub_location", "type": ["null","string"]}, {"name": "main_location", "type": ["null","string"]},
    {"name": "hd", "type": "string", "default": "no"}
]}
```

ORC: Optimized Row Columnar

ORC - особенности

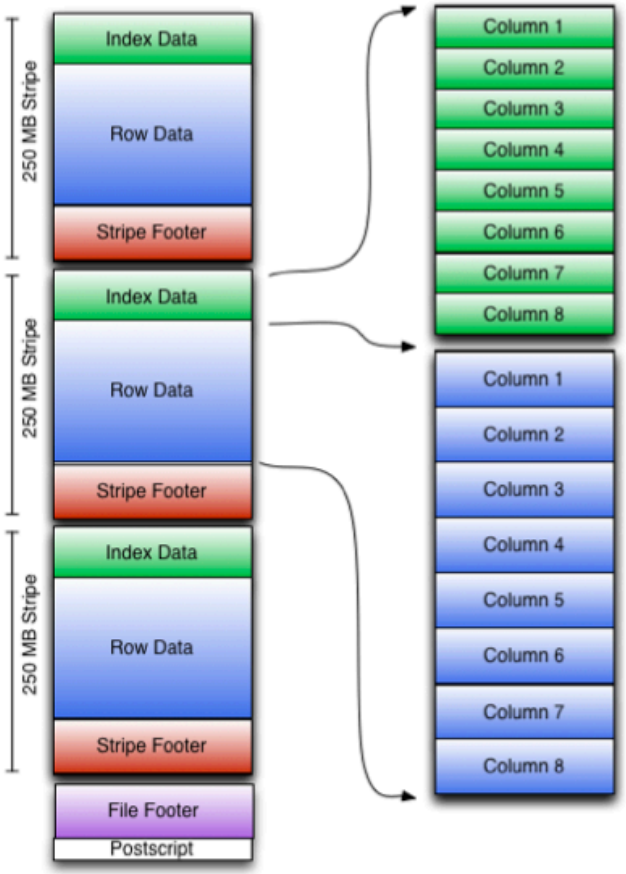
- Проект HIVE (Hortonworks) по замене RCFile
- Колоночный формат
- Хранение строк в Stripes (группы по 10К) – расщепление и параллельная обработка
- Схема хранится в Footer
- Различные типы данных
- Встроенные сжатие, индексы, статистики

ORC типы данных

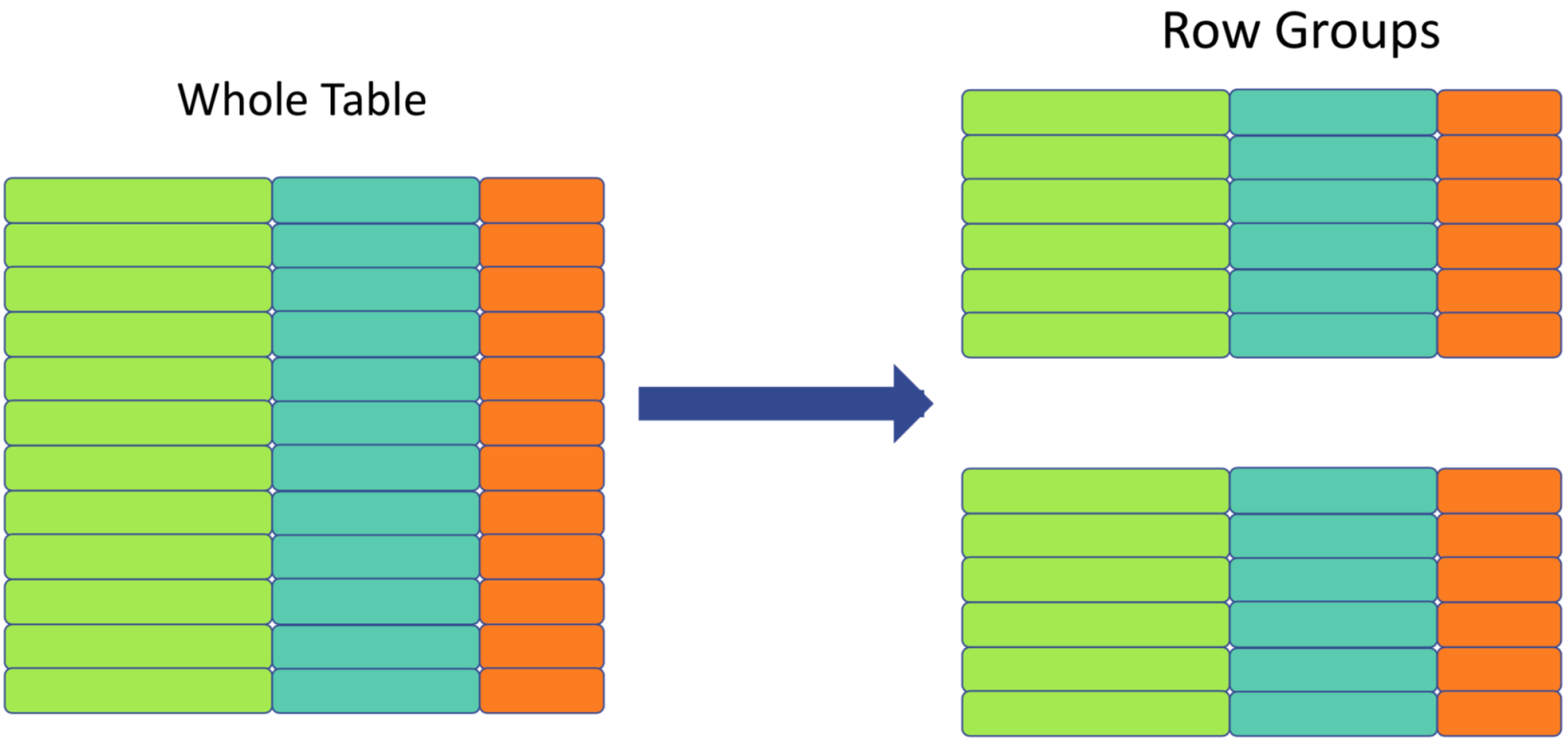
- Integer
  - boolean (1 bit)
  - tinyint (8 bit)
  - smallint (16 bit)
  - int (32 bit)
  - bigint (64 bit)
- Floating point
  - float
  - double
- String types
  - string
  - char
  - varchar
- Binary blobs
  - binary
- Date/time
  - timestamp
  - date
- Compound types
  - struct
  - list
  - map
  - union

ORC – структура файла

- Файл состоит из групп строк называемых **Stripes**
- Метаданные хранятся в **Footer**
- В конце файле есть **Postscript**, котором указаны параметры сжатия и размер **Footer**
- Размер **Stripe** по умолчанию 250M
- Footer содержит список **Stripes** количество строк в каждом **Stripe**, Схему, типы данных, статистики уровня файла

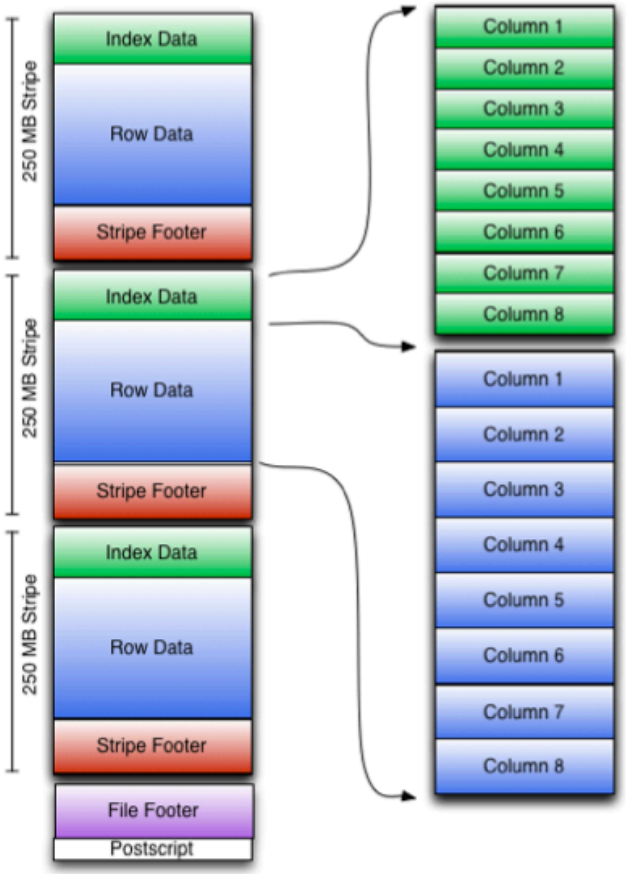


Storage

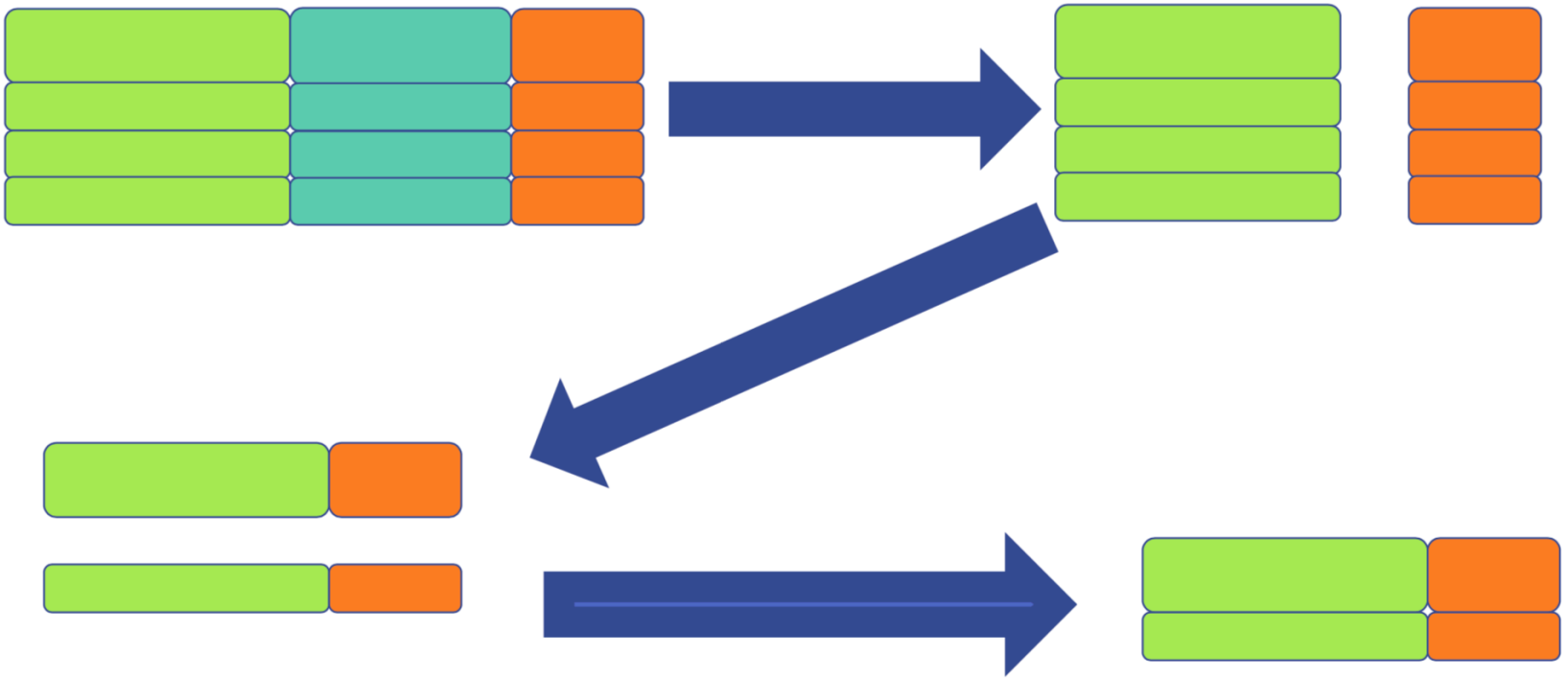


ORC – структура Stripe

- Каждый Stripe содержит индекс, строки и Footer
- Footer содержит указатели на данные
- **Индексы** включают min / max / bloom filters
- Индексы используются только для выборки Stripes, Row Groups
- Наличие индексов позволяет быстро читать только запрашиваемые диапазоны данных.
- По умолчанию каждые 10К строк могут быть отфильтрованы



Columnar - Slice and Dice



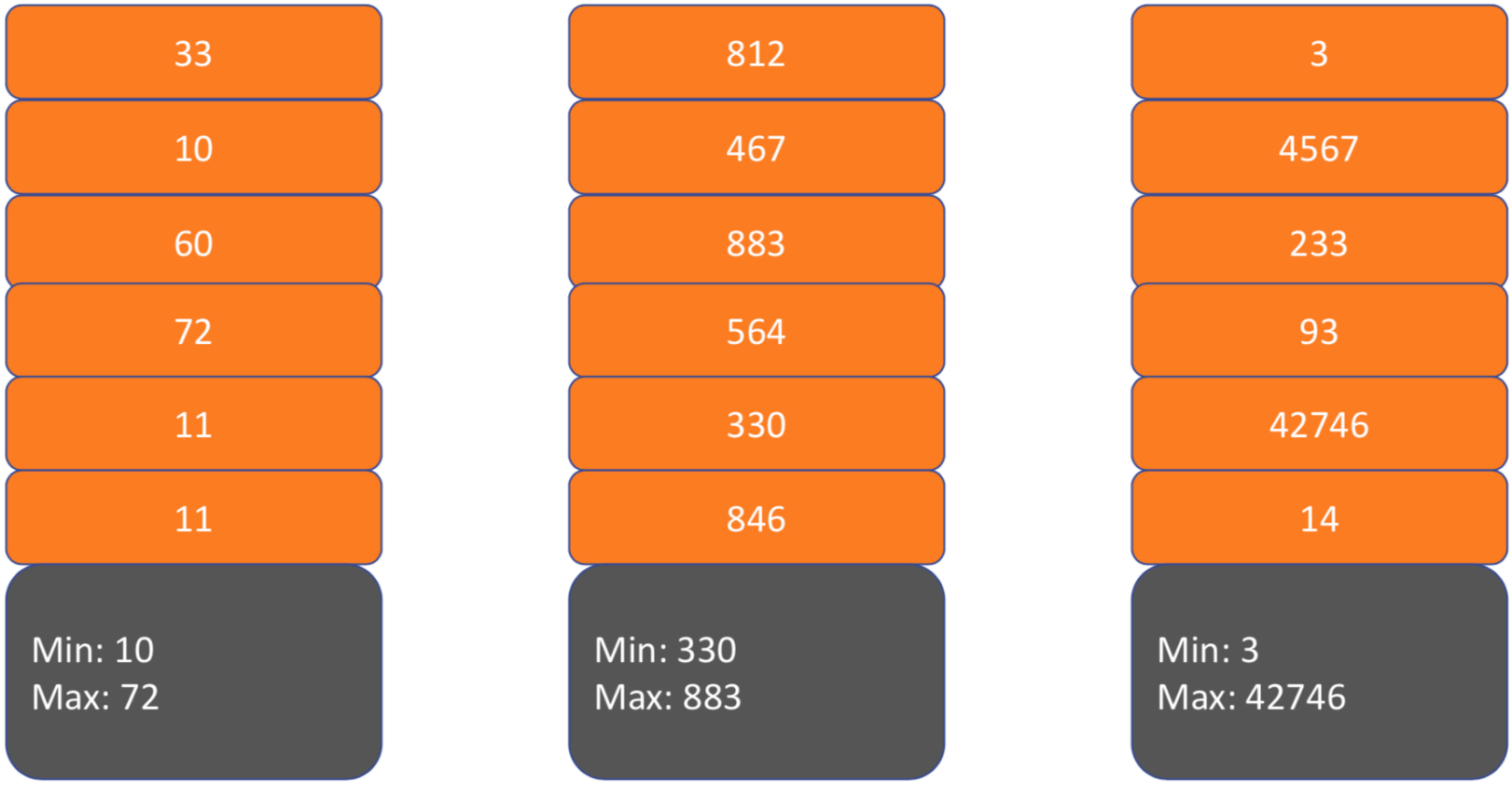
ORC

- Преимущества
  - Колоночный
  - Индексы
  - Сжатие: ZLIB, Snappy
  - Гибкие настройки индексов / Bloom filters
- Недостатки
  - Нет полноценной поддержки в Spark
  - Только плоские схемы (без вложенности)

Indexing

- Статистики
- Числовые данные
- Оптимально для предварительно отсортированных данных

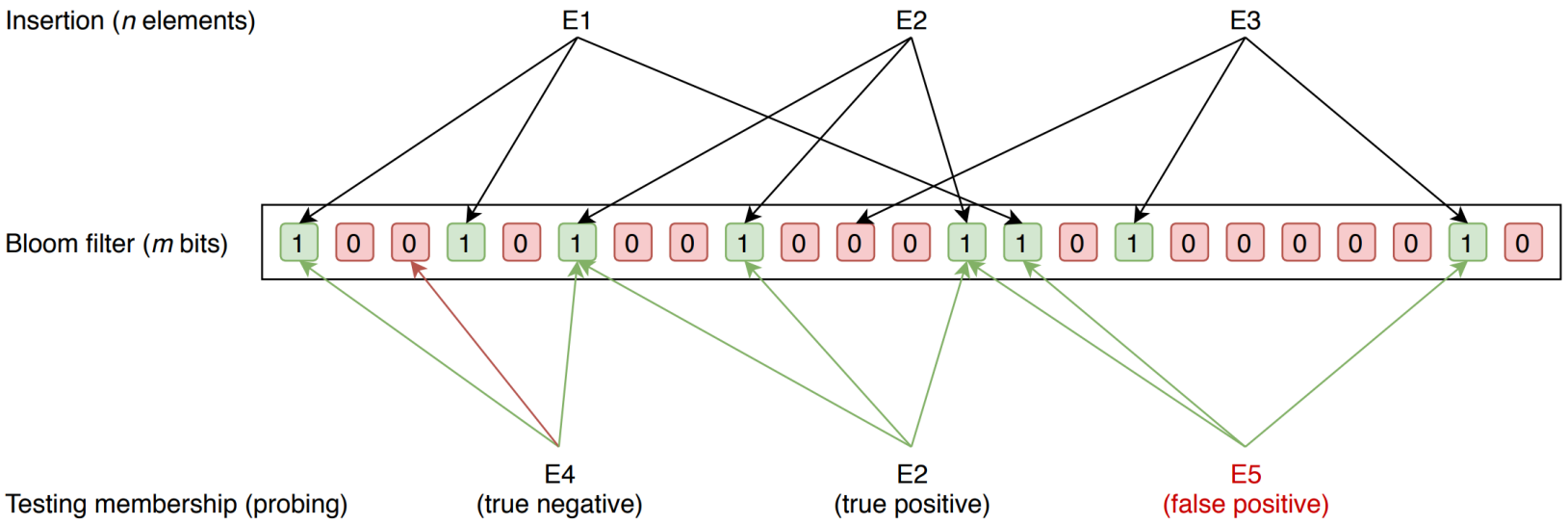
Indexing


33
10
60
72
11
11
Min: 10
Max: 72
812
467
883
564
330
846
Min: 330
Max: 883
3
4567
233
93
42746
14
Min: 3
Max: 42746

Bloom Filtering

- Вероятностная структура
- Вопрос: Присутствует ли элемент в выборке
- Ответ: «Вероятно да» / «Точно нет»
- Включается для определенных колонок
- Полезен для нечисловых колонок с небольшим количеством значений

Bloom Filter with m = 23, n = 3 and k = 3



ORC Hive Parameters

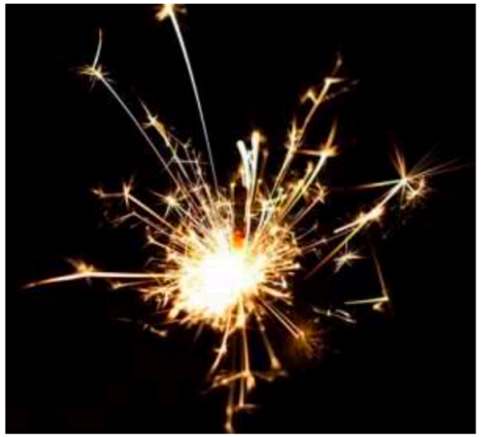
Key	Default	Notes
orc.compress	ZLIB	high level compression (one of NONE, ZLIB, SNAPPY)
orc.compress.size	262,144	number of bytes in each compression chunk
orc.stripe.size	67,108,864	number of bytes in each stripe
orc.row.index.stride	10,000	number of rows between index entries (must be >= 1000)
orc.create.index	true	whether to create row indexes
orc.bloom.filter.columns	""	comma separated list of column names for which bloom filter should be created
orc.bloom.filter.fpp	0.05	false positive probability for bloom filter (must >0.0 and <1.0)

ORC + Hive

+

=

+



=

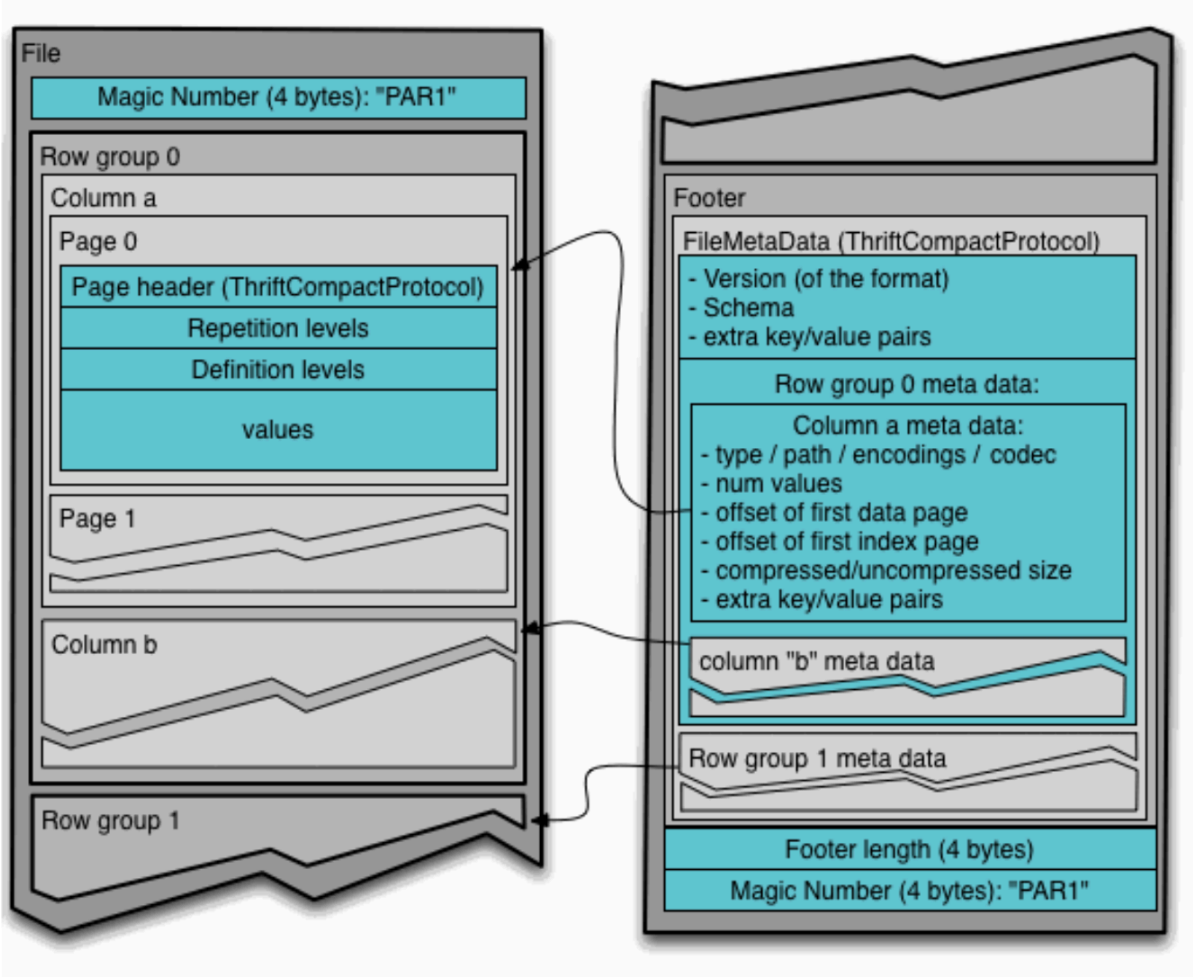
¯_(ツ)_/¯

Parquet

Parquet

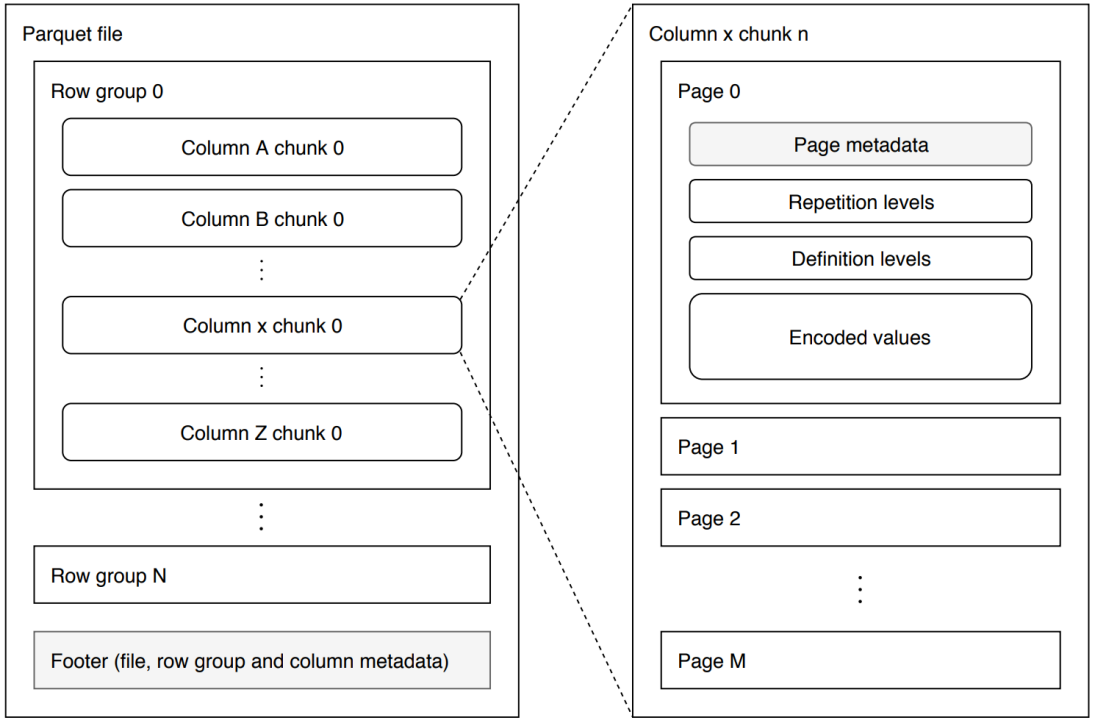
- Колоночный формат
- Проект Cloudera & Twitter
- Файлы делятся на Row Groups
- Схема и метаданные хранятся в Footer
- Поддержка вложенных структур
- Поддержка Schema Evolution

Parquet – структура Footer



Parquet - структура файла

- **ROW GROUPS**
  - Объединяет данные строк
  - Состоит из **COLUMN CHUNKS**
- **COLUMN CHUNKS**
  - Последовательные данные одной колонки
  - Состоит из **DATA PAGES** и опционально **DICTIONARY PAGE**
- **DATA PAGES**
  - Сжатые данные
  - Данные по ~8KB

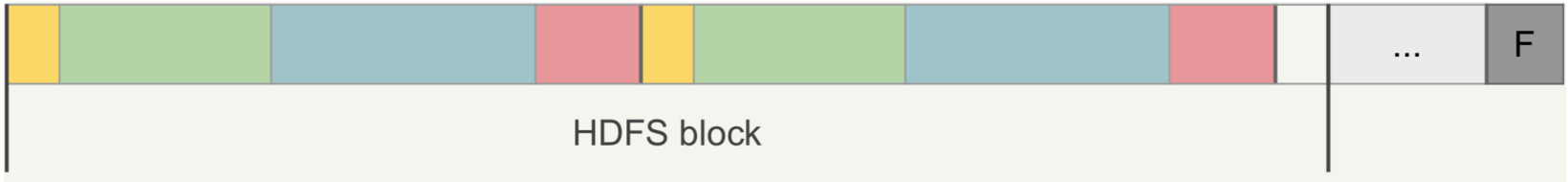


Единицы параллелизации

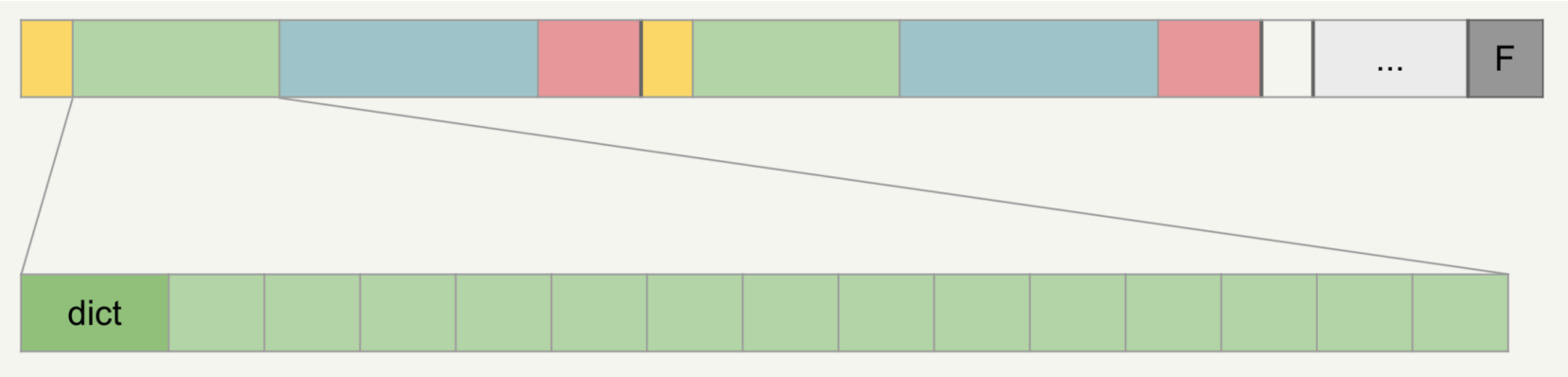
- MR task - File/Row Group
- IO - Column chunk
- Encoding/Compression - Page

Parquet - Row Groups

A	B	C	D
a1	b1	c1	d1
...	...	...	...
aN	bN	cN	dN
...	...	...	...



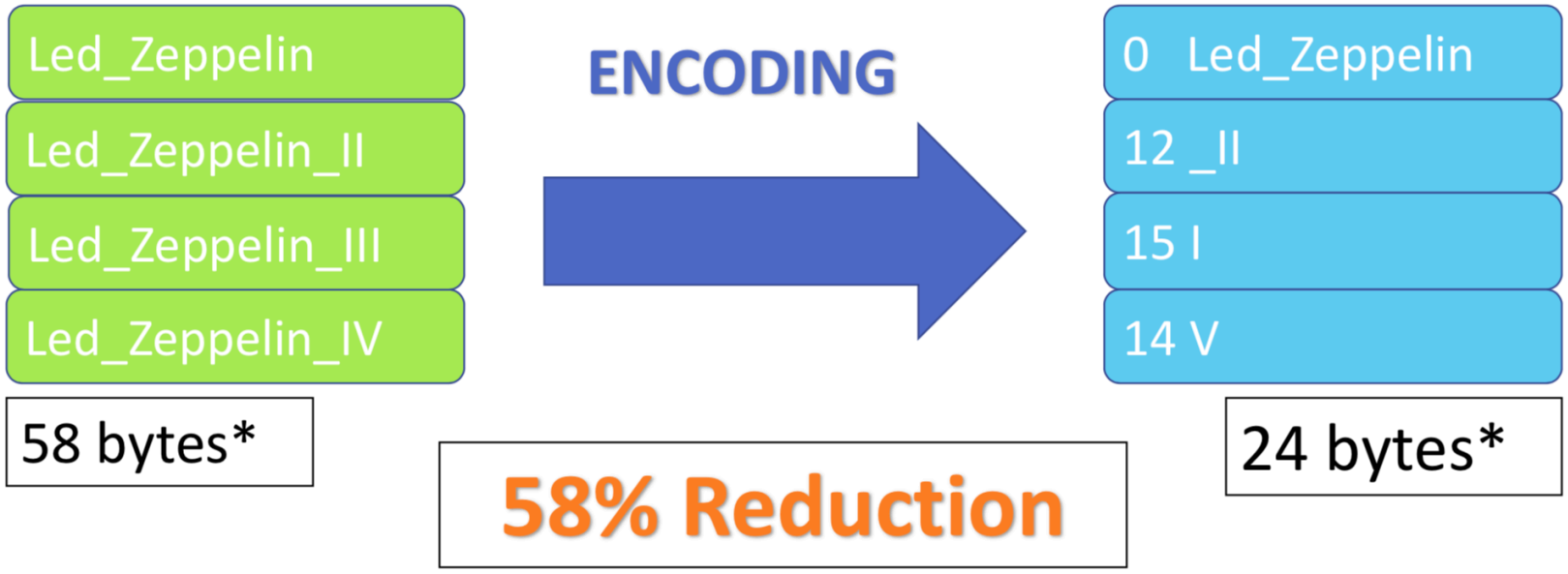
Column chunks and pages



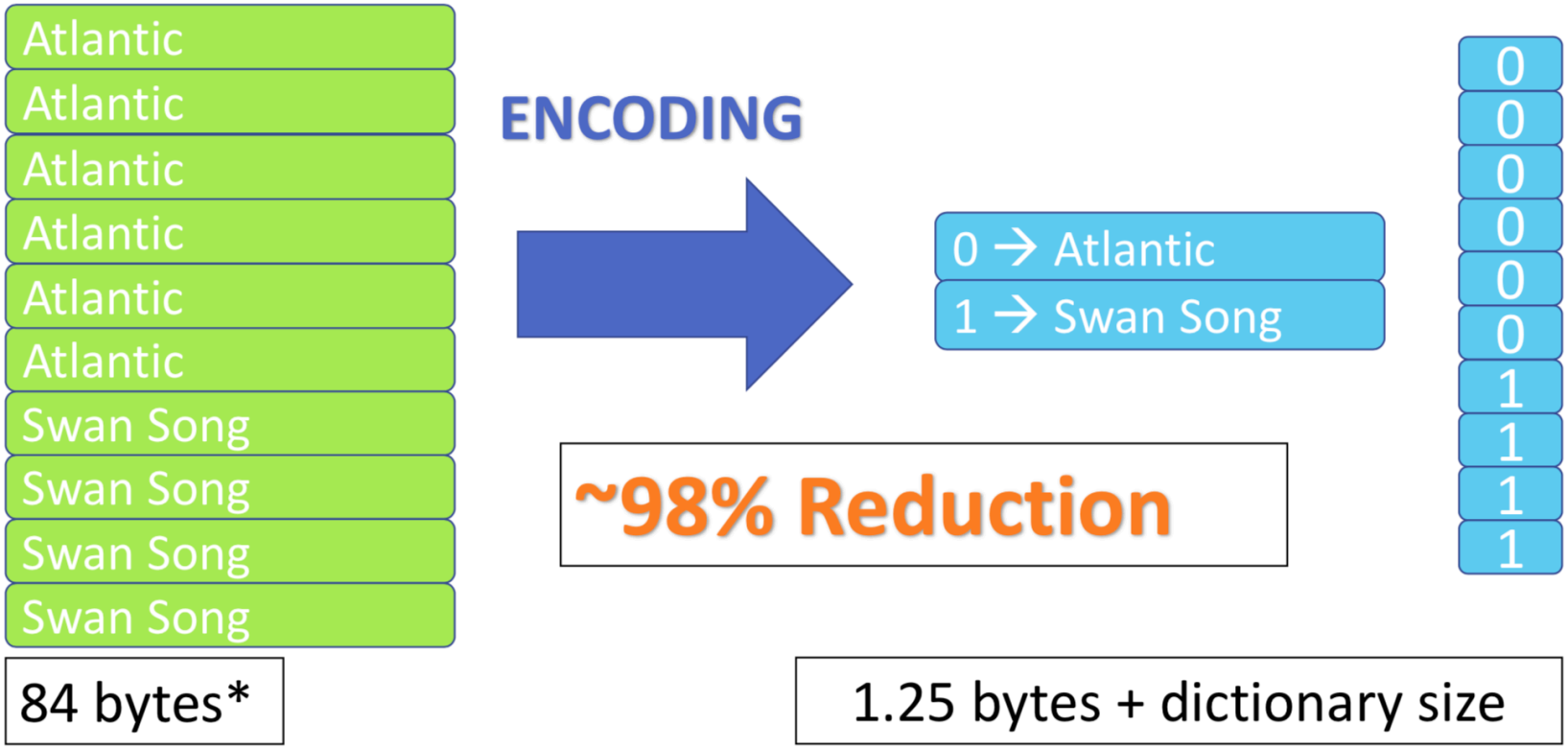
Parquet – преимущества

- Колоночный формат
  - Эффективное сжатие и кодирование
  - Column projection: читаем запрашиваемые колонки
- Фильтрация на уровне Row group
  - Footer статистики с целью исключить Row groups
  - Dictionary pages с целью исключить Row groups
- Фильтрация на уровне Page
  - Используем статистики на уровне Page

Кодирование: Incremental Encoding



Кодирование: Dictionary Encoding

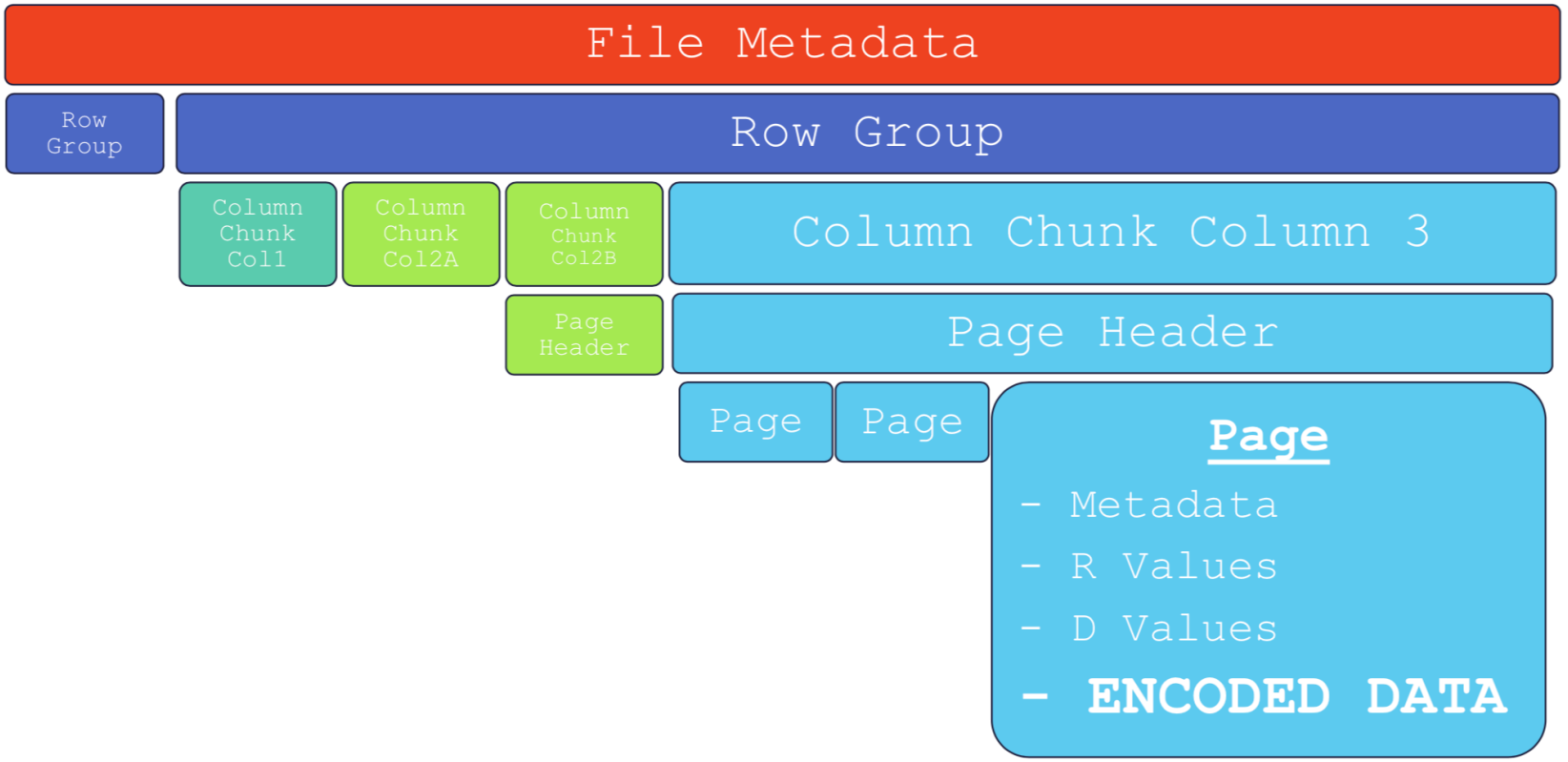


Все схемы кодирования

- Plain (bit-packed, little endian, etc)
- Dictionary Encoding
- Run Length Encoding/Bit Packing Hybrid
- Delta Encoding
- Delta-Length Byte Array
- Delta Strings (incremental Encoding)

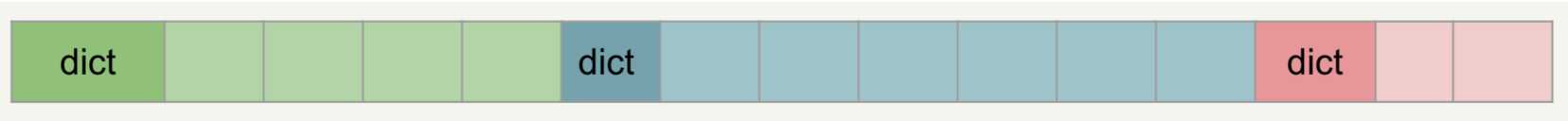
* https://github.com/apache/parquet-format/blob/master/Encodings.md

Parquet – Slice & Dice



Фильтры на уровне Dictionary

- **Dictionary (словарь)** – список всех встречающихся значений
  - Значение отсутствует - пропустить Row Group
  - Как Bloom filter, но без false positives
- **Когда это полезно:**
  - Колонка отсортирована на уровне файла, но не глобально
  - Небольшое количество значений



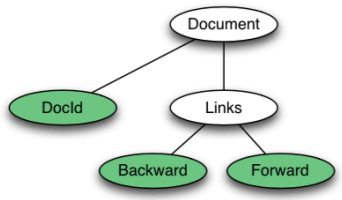
Вложенные структуры

Nested record shredding/assembly

- Algorithm borrowed from Google Dremel's column IO
- Each cell is encoded as a triplet: **repetition level, definition level, value**.
- Level values are bound by the depth of the schema: **stored in a compact form.**

Schema:

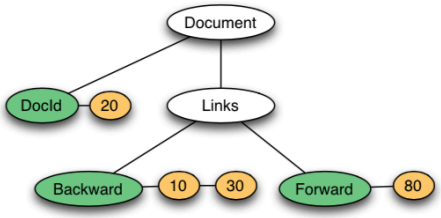
```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward;
  }
}
```



Columns	Max rep. level	Max def. level
DocId	0	0
Links.Backward	1	2
Links.Forward	1	2

Record:

```
DocId: 20
Links
  Backward: 10
  Backward: 30
  Forward:  80
```



Column	Value	R	D
DocId	20	0	0
Links.Backward	10	0	2
Links.Backward	30	1	2
Links.Forward	80	0	2

Dremel made simple with Parquet

Repetition level

Schema:

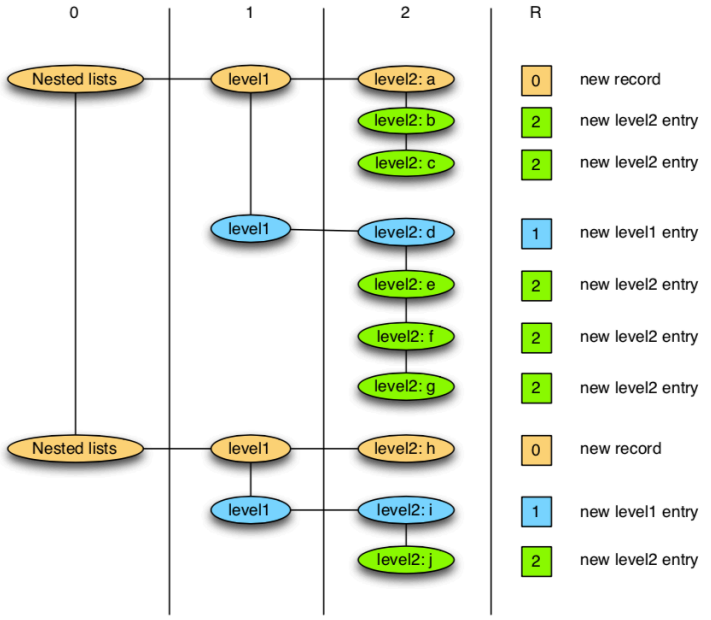
```
message nestedLists {
   repeated group level1 {
      repeated string level2;
   }
}
```

Records:

```
[[a, b, c], [d, e, f, g]]
[[h], [i, j]]
```

Columns:

```
Level: 0,2,2,1,2,2,2,0,1,2
Data:  a,b,c,d,e,f,g,h,i,j
```



Пример

Title	Released	Label	PeakChart.UK	Certification.BVMI	Certification.RIAA	(omitted for space…)
Led Zeppelin	01/12/1969	Atlantic	6		8x Platinum	…
Led Zeppelin II	10/22/1969	Atlantic	1	Platinum	Diamond	…
Led Zeppelin III	10/05/1970	Atlantic	1	Gold	6x Platinum	…
Led Zeppelin IV	11/08/1971	Atlantic	1	3x Gold	Diamond	…
Houses of the Holy	03/28/1973	Atlantic	1	Gold	Diamond	…
Physical Graffiti	02/24/1975	Swan Song	1	Gold	Diamond	…
Presence	03/31/1976	Swan Song	1		3x Platinum	…
In Through The Out Door	08/15/1979	Swan Song	1		6x Platinum	…
Coda	11/19/1982	Swan Song	4		Platinum	…

Пример – схема

COLUMN NAME	Title
OPTIONAL / REQUIRED / REPEATED	OPTIONAL
DATA TYPE	BINARY
ENCODING INFO FOR BINARY	0:UTF8
REPETITION VALUE	R:0
DEFINITION VALUE	D:0

FLAT SCHEMA

```
TITLE:              OPTIONAL BINARY O:UTF8 R:0 D:1
RELEASED:           OPTIONAL BINARY O:UTF8 R:0 D:1
LABEL:              OPTIONAL BINARY O:UTF8 R:0 D:1
PEAKCHART.UK:       REQUIRED INT32 R:0 D:0
```

Пример – строка данных

```
TITLE = LED ZEPPELIN IV
RELEASED = 11/8/1971
LABEL = ATLANTIC
PEAKCHART.UK = 1
PEAKCHART.AUS = 2
PEAKCHART.US = 2
CERTIFICATION.ARIA = 9X PLATINUM
CERTIFICATION.BPI = 6X PLATINUM
CERTIFICATION.BVMI = 3X GOLD
CERTIFICATION.CRIA = 2X DIAMOND
CERTIFICATION.IFPI = 2X PLATINUM
CERTIFICATION.NVPI = PLATINUM
CERTIFICATION.RIAA = DIAMOND
CERTIFICATION.SNEP = 2X PLATINUM
```

```
Title = Led Zeppelin IV
Released = 11/8/1971
Label = Atlantic
PeakChart:
.AUS = 2
.UK = 1
.US = 2
Certification:
.ARIA = 9x Platinum
.BPI = 6x Platinum
.BVMI = 3x Gold
.CRIA = 2x Diamond
.IFPI = 2x Platinum
.NVPI = Platinum
.RIAA = Diamond
.SNEP = 2x Platinum
```

Факторы выбора формата

- Инструменты для обработки и запросов
- Эволюция модели данных (Schema Evolution)
- Расщепление (splitability) файлов с данными
- Компрессия данных
- Время отклика (latency)
- Тип нагрузки (Write / Read)

Сравнение форматов данных

	Avro	Parquet	ORC
Schema Evolution Support			
Compression			
Splitability			
Most Compatible Platforms	Kafka, Druid	Impala, Arrow Drill, Spark	Hive, Presto
Row or Column	Row	Column	Column
Read or Write	Write	Read	Read

Source: Nexla analysis, April 2018

Интересные материалы

- Faster Performance for Selective Queries
- Parquet performance tuning: the missing guide
- 1.1 Billion Taxi Rides: Spark 2.4.0 versus Presto 0.214
- Benchmarking Apache Parquet: The Allstate Experience
- Hadoop File Formats: It's not just CSV anymore
- Dremel made simple with Parquet
- Fast Access To Your Complex Data - Avro, JSON, ORC, and Parquet

Рефлексия

- Что вам запомнилось больше всего
- Пройти опрос

Ваши вопросы?