Datalake File Format: Parquet

arnaud.nauwynck@gmail.com

Course Esilv 2024

This document:
https://github.com/Arnaud-Nauwynck/presentations/pres-bigdata/Datalake-File-Format-Parquet

Outline

- Parquet Caracteristics
  - Structured, with Schema
  - Data – Metadata (footer)
  - Columnar ... Column Pruning
  - Splittable ... spark Dataset Partitions
  - Compression
  - Encoding
  - Statistics, Bloom ... spark Predicate-Push-Down
  - Optimize write once, read many
    ... spark dataset.repartition().sortWithinPartition().write

Parquet is a Structured Format
Strongly Typed (Schema)

File Formats

Unstructured

Text

```
text line 1 \n
text line 2 \n
```

Csv

```
Col1;Col2;Col3\n
a1;b1;c1\n
a2;b2;c2\n
```

Semi-Structured

Json

```
{"a":"a1","b":"b1"} \n
{"a":"a2","b":"b2"} \n
```

Xml

```
<elt>
  <a>a1</a>
  <b>b1</b>
```

Structured

Serialization Structured

Avro,Thrift,Protobuf

```
schema:XX,
value:0101010101
```

Columnar Structured

Orc, Parquet

Structured : struct<>, array<>, map<>

Scalar Value

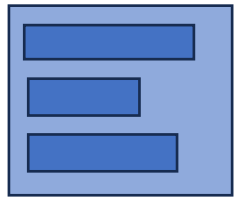
(= terminal element in grammar)

= primitive data-type

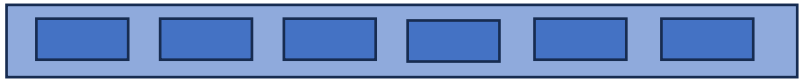
String
boolean
int
double
Date
…

Composite Value

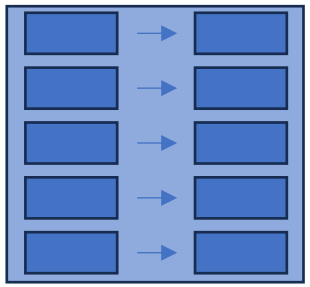
struct<a:Type1, b:Type2, ...>



array<ElementType>



map<KeyType,ValueType>



Type constraint = Schema

semi-structured
schema is optional

{ JSON } → JSON Schema (optional)

<xml> → XSD Schema (optional)

structured
schema is strict

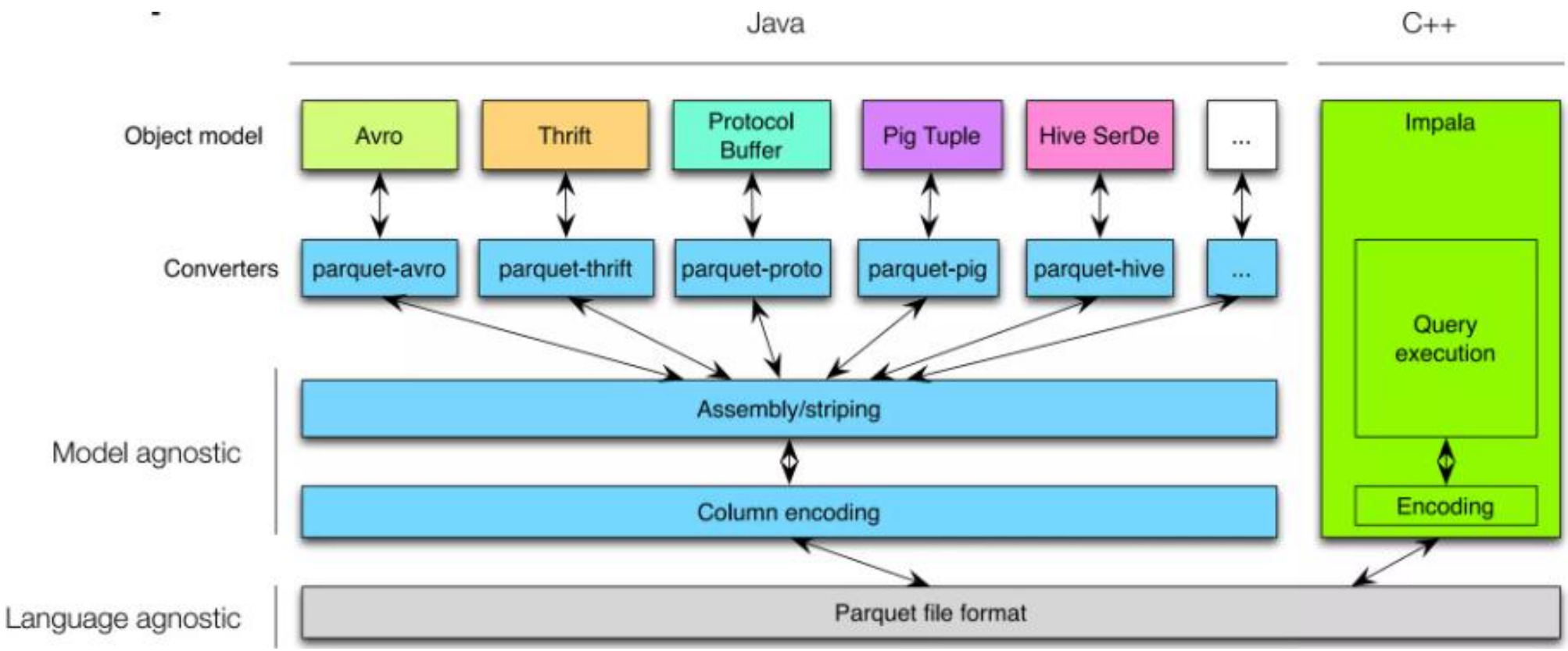
.avro → .avsc = avro schema
included in avro datafile format header
implicit in avro client-server exchanges

thrift

protobuf

.parquet → included in parquet file footer (= metadata)

Parquet SDK / Converter / ObjectModel

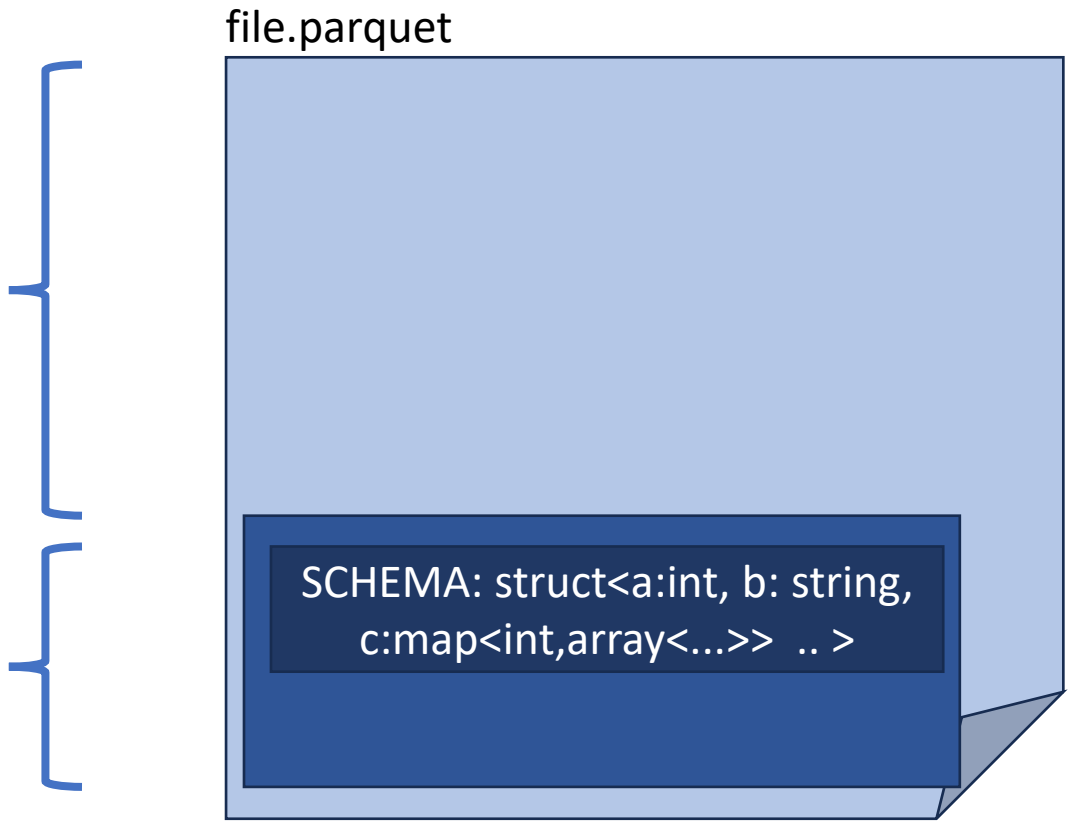


Parquet separates Data and MetaData

Parquet Schema : in metadata footer

data part
(usually N x blocks of 128 Mo)

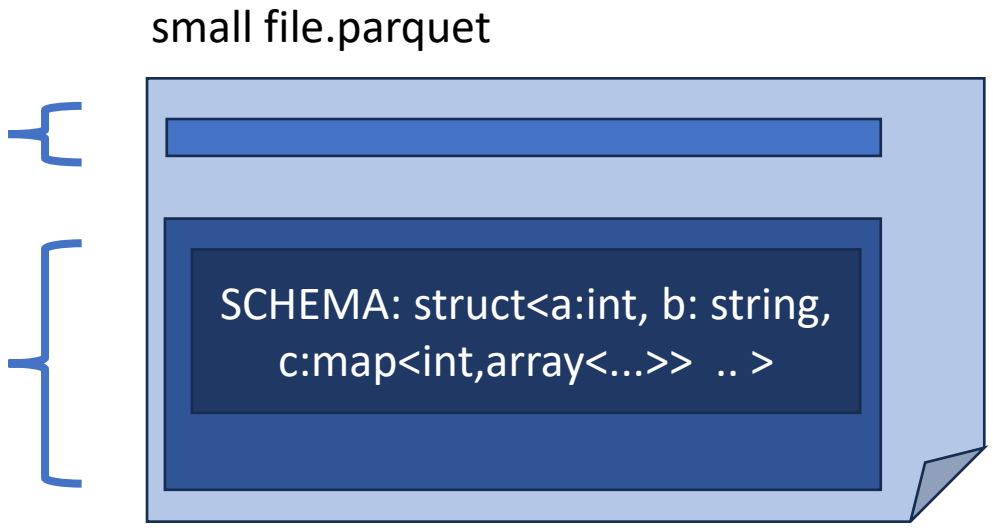
footer: metadata part
(usually <= 50 ko)



Parquet for Small Files ?

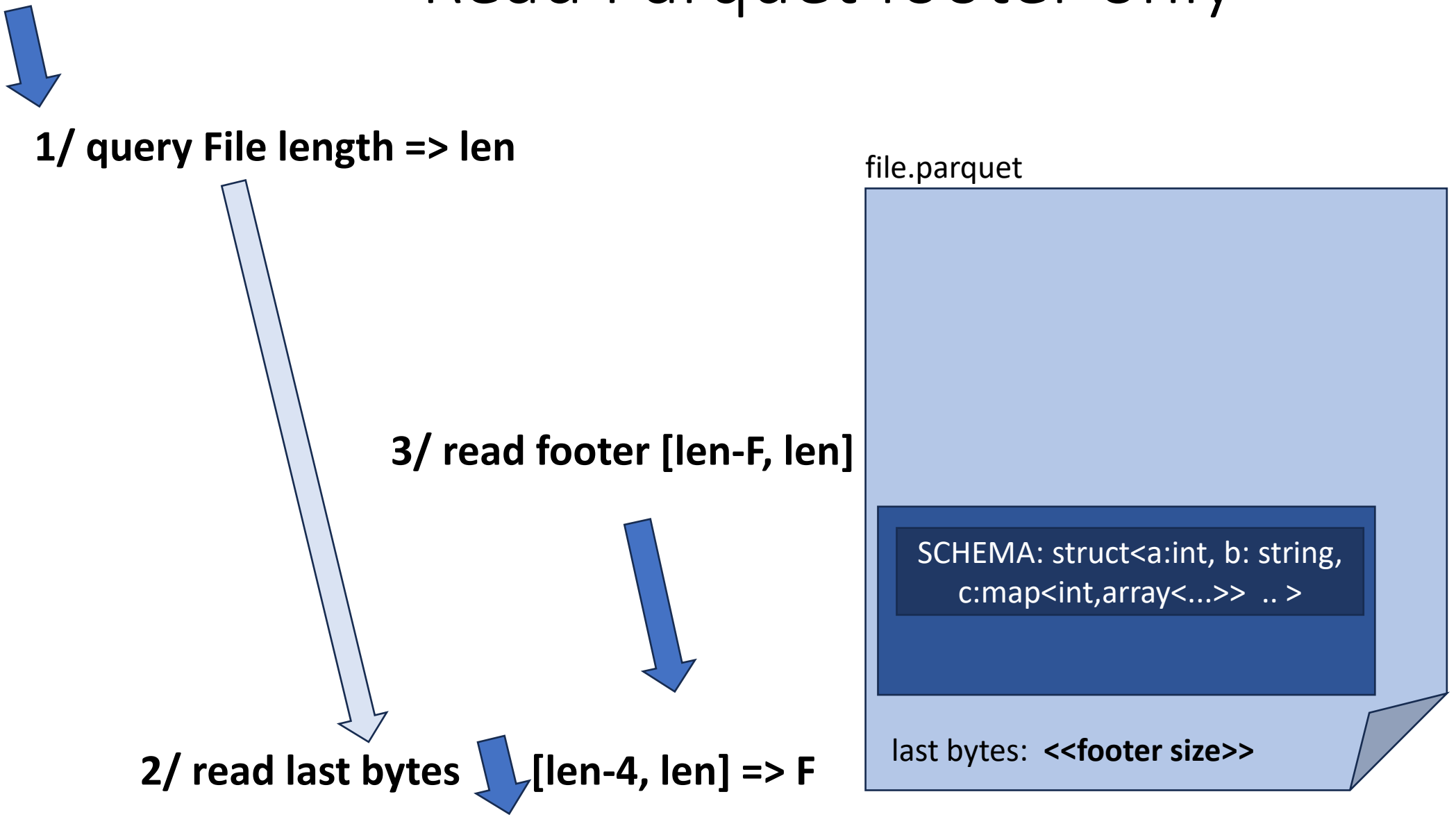
data part can be small
(even empty)

parquet files
ALWAYS have an
uncompressable footer,
containing SCHEMA



Parquet for small data is NOT efficient
bad ratio of "data / metadata" size

Read Parquet footer only



Why Footer instead of Header?

For reader : not a big overhead, ONLY 3 calls to read

For writer : MUCH more practical to "stream" write Nx rows,
keep in-memory only few metadata,
to flush write at end

Parquet is Columnar

Parquet : Columnar File Format

Logical view: rows - columns

a1	b1	c1	d1
a2	b2	c2	d2
a3	b3	c3	d3
a4	b4	c4	d4

On Disk Row Serialization: like CSV, JSON, AVRO,

On Disk: **Columnar**

Columnar
=> better memory aligned,
Vectorized CPU pipeline

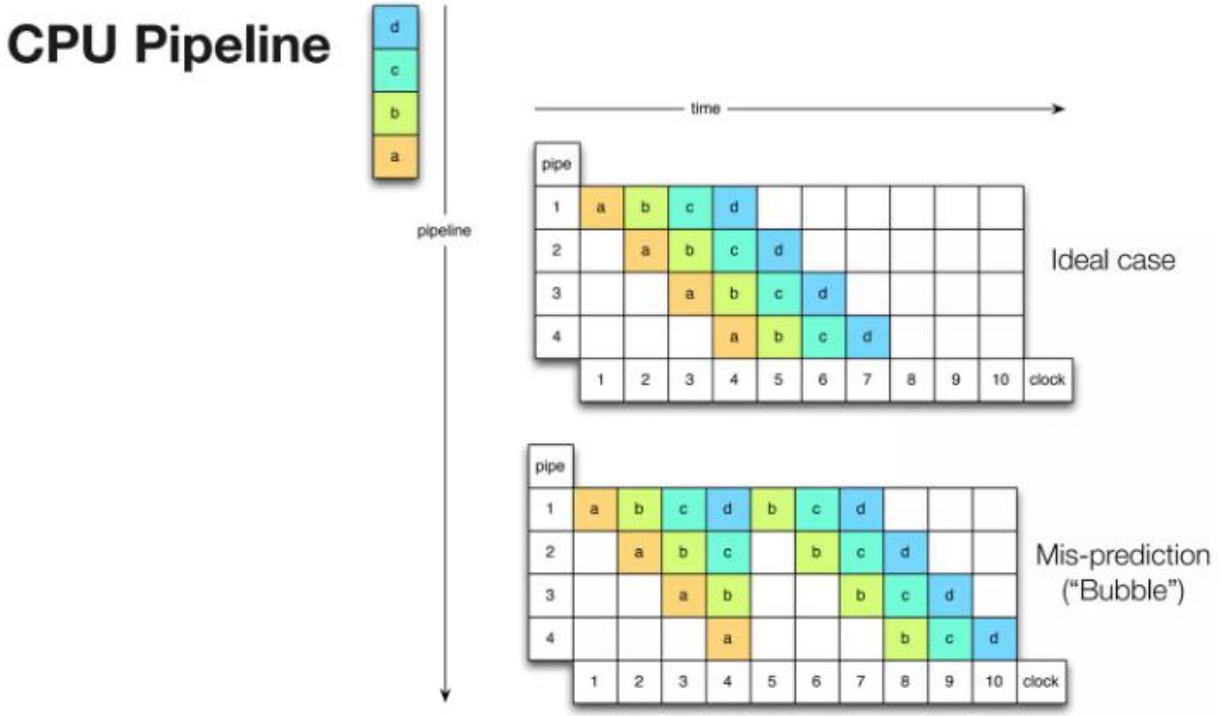
```
struct A {

  boolean f1;   // <= 1 bit (on 1 byte)
                // in-memory padding 3 bytes
  int     f2;  // 4 bytes, aligned on multiple of 4
               //  in-memory padding 4 bytes
  long    f3  // 8 bytes
}
```

array[struct<..>] <=> array[boolean]
array[int]
array[long]

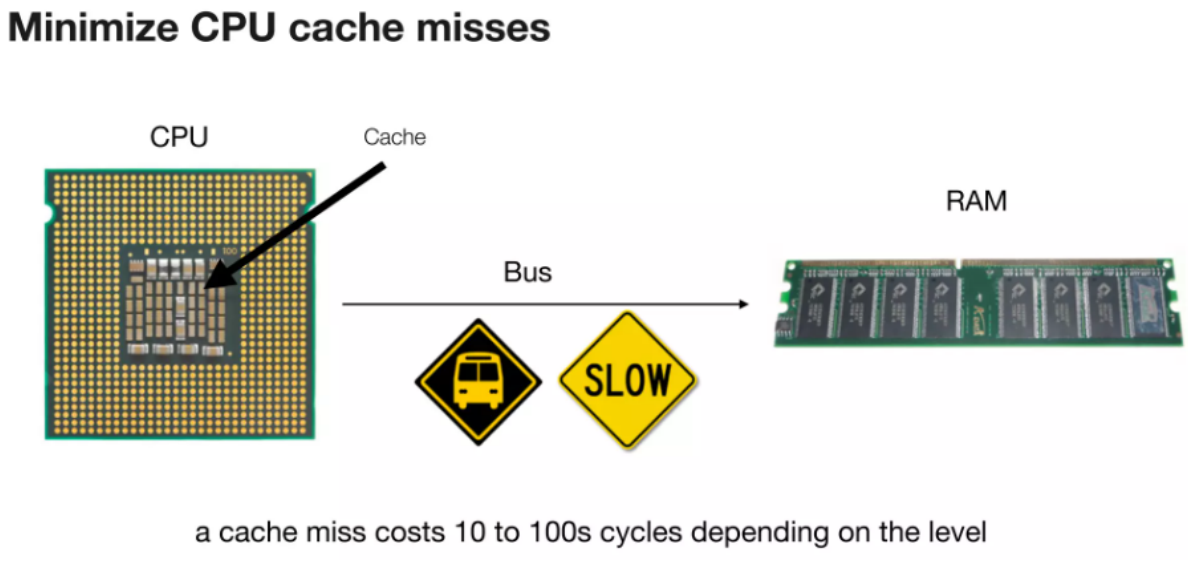
Vectorized Reader **~9x Faster**

Vectorized code ... fewer "If", "Loop", "calls"

Better CPU Pipeline



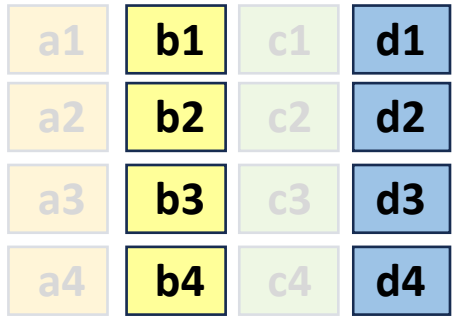
Better Memory Cache



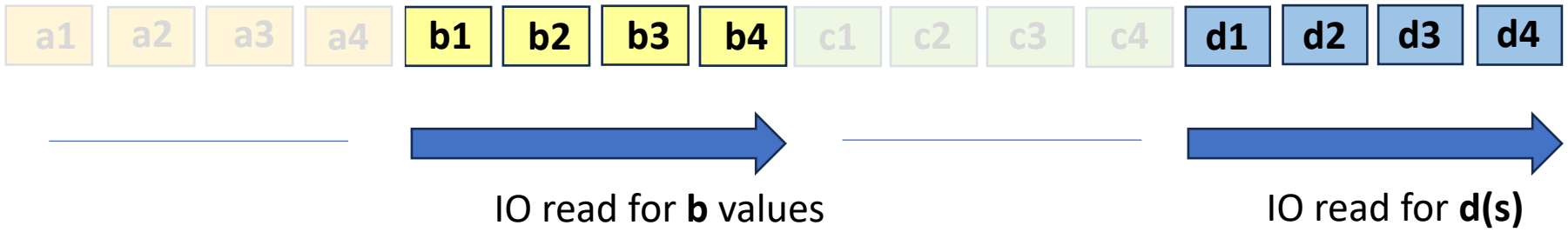
Column Pruning Optimization

SELECT b , d -- ONLY 2 columns
FROM table
WHERE ..

Logical view: rows - columns

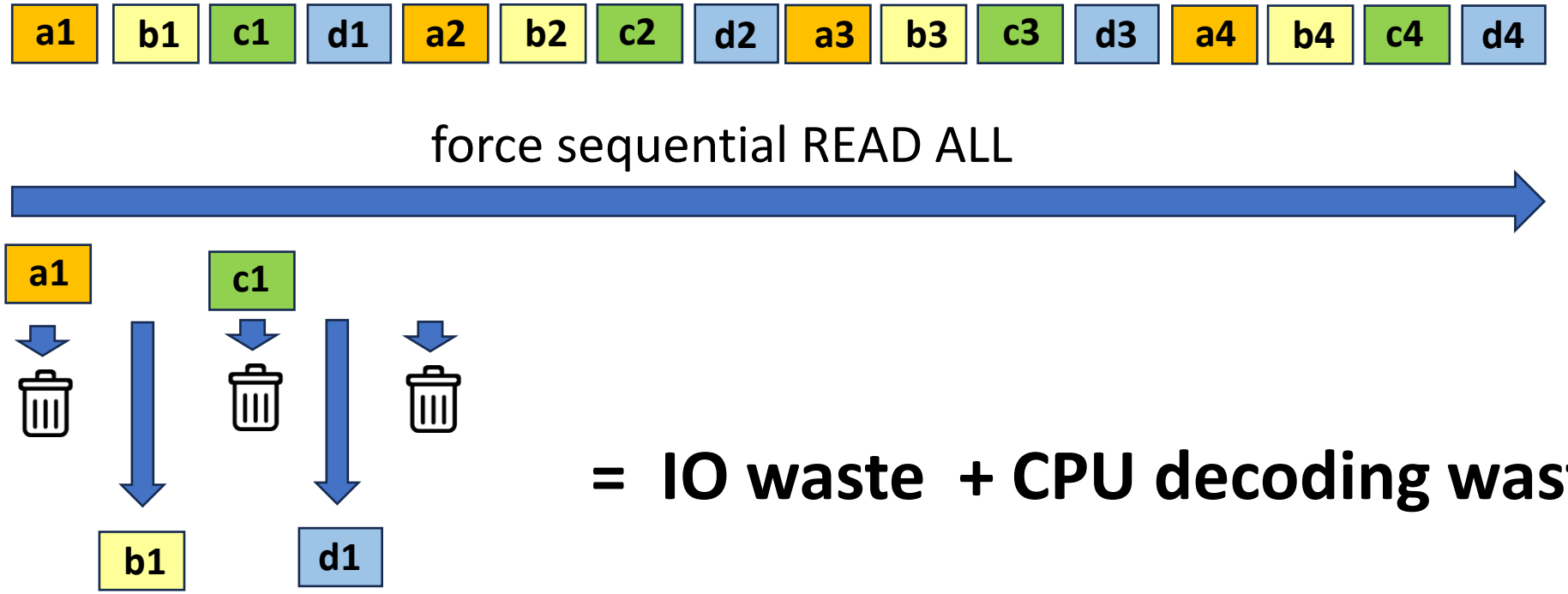


On Disk READ: columnar



Compare IO Reads with JSON, CSV, Avro, ..

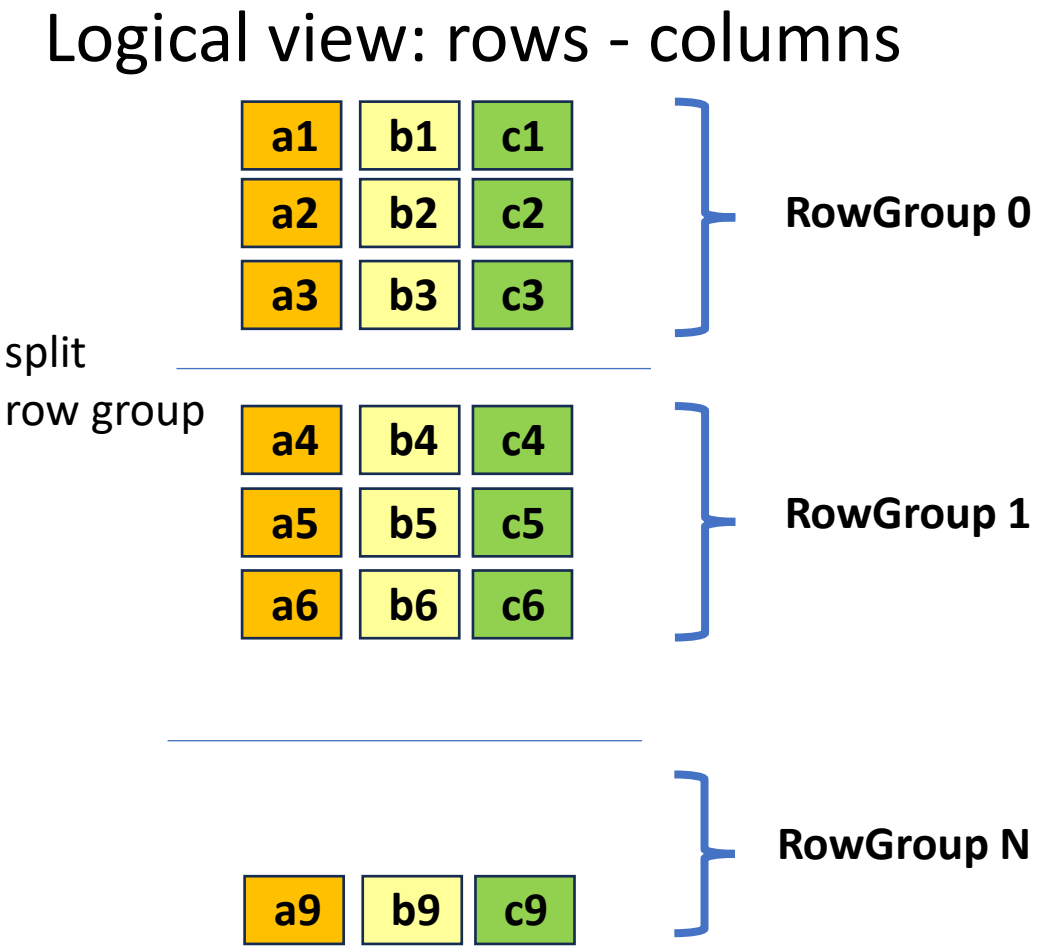
SELECT b, d -- ONLY 2 columns
FROM table
WHERE ..

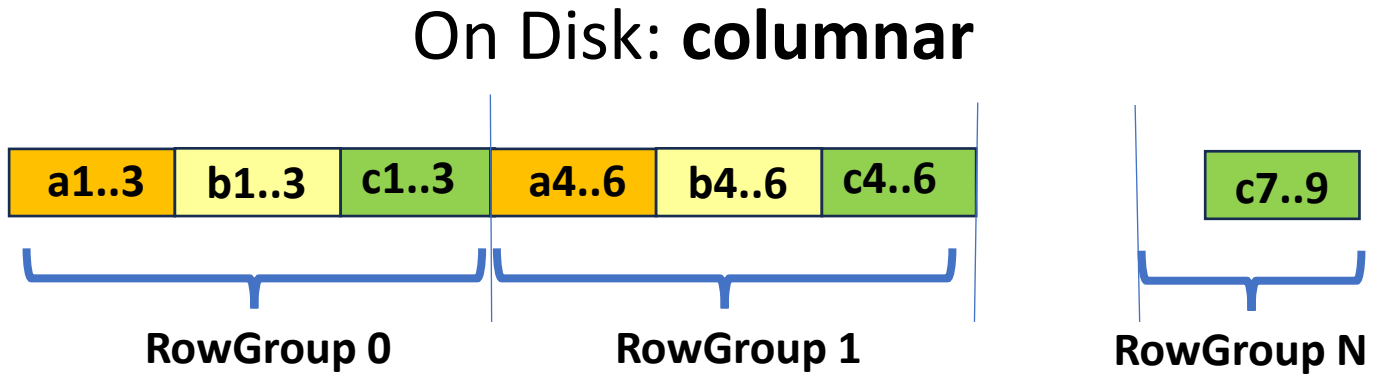


= IO waste + CPU decoding waste

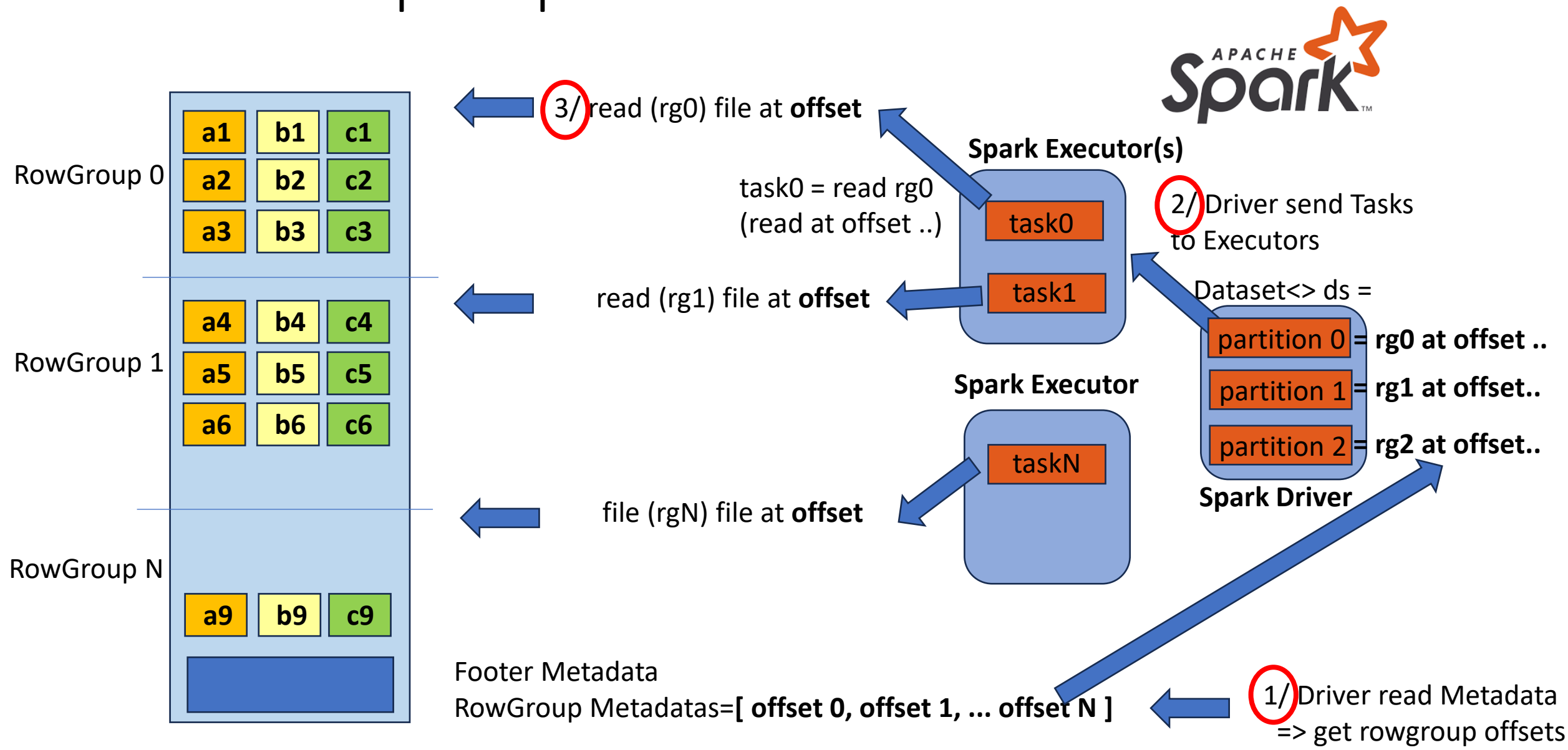
Parquet is Splitteable

Parquet split rows by RowGroups





RowGroup ~ Spark Partition ~ Executor Thread



1 Parquet RowGroup(s)
-> default to 1 Spark DataSet partition

by default,

parquet.block.size = 128 Mo

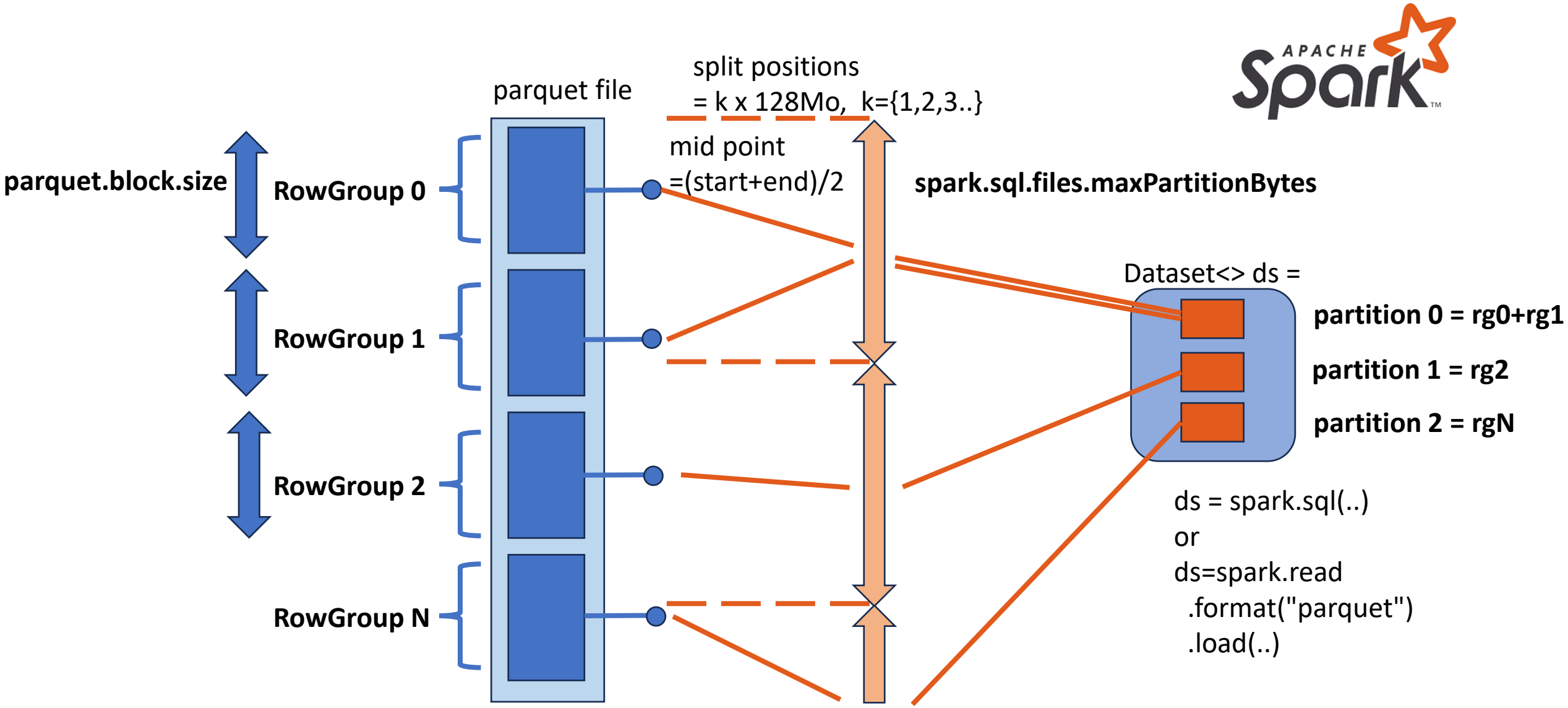
=

spark.files.maxPartitionBytes = 128 Mo

=

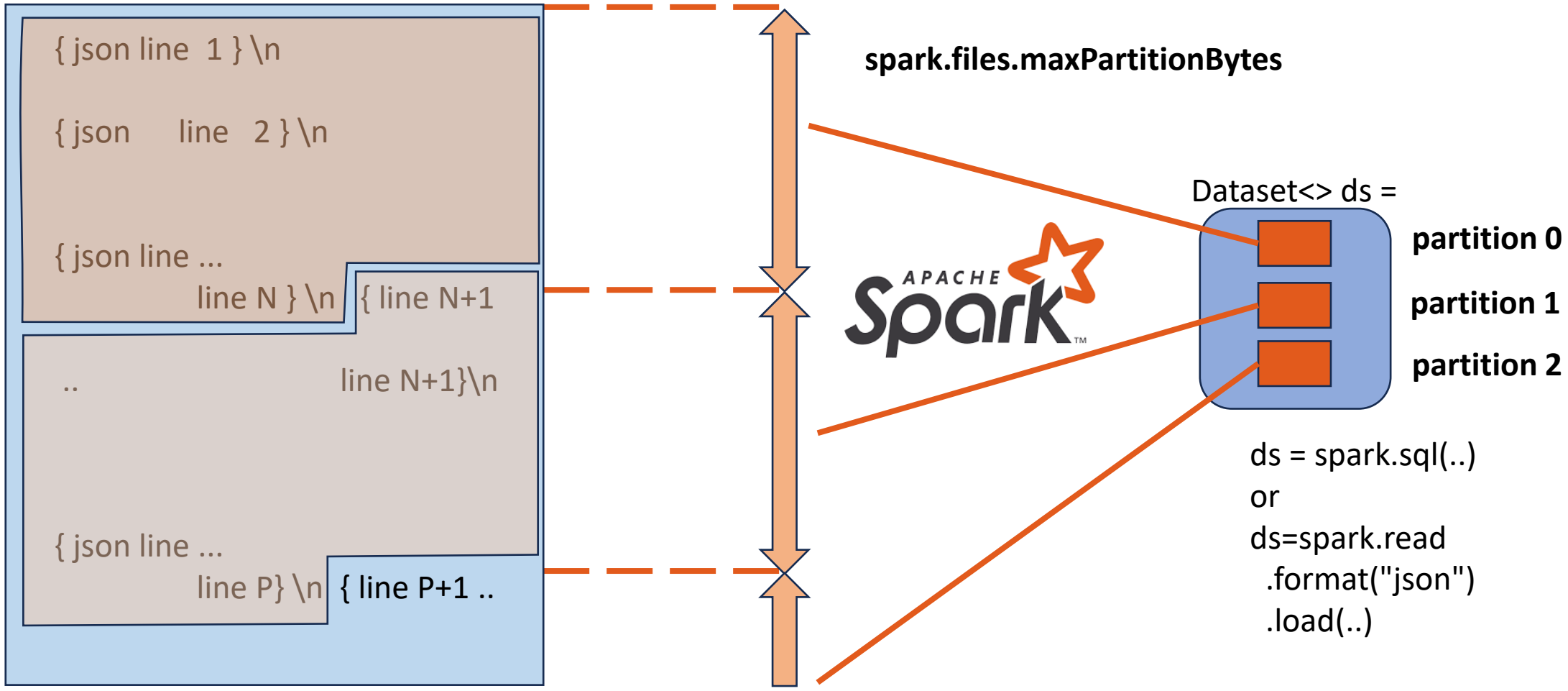
spark.sql.files.maxPartitionBytes = 128 Mo

N Parquet RowGroups
-> FileSplit to P (<=N) Spark Partitions

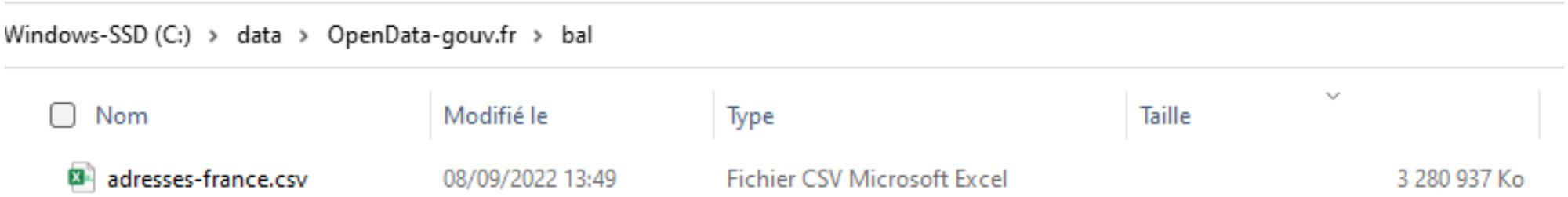


{Texts,CSV,Json} formats FileSplit

each spark executor reader Thread
  at split start => ignore first chars until '\n'
  at split end => read extra chars until '\n



Example : Reading 1 CSV of 3.2 Go
=> 26 splits = 25 (~128 Mo) + 1 very small

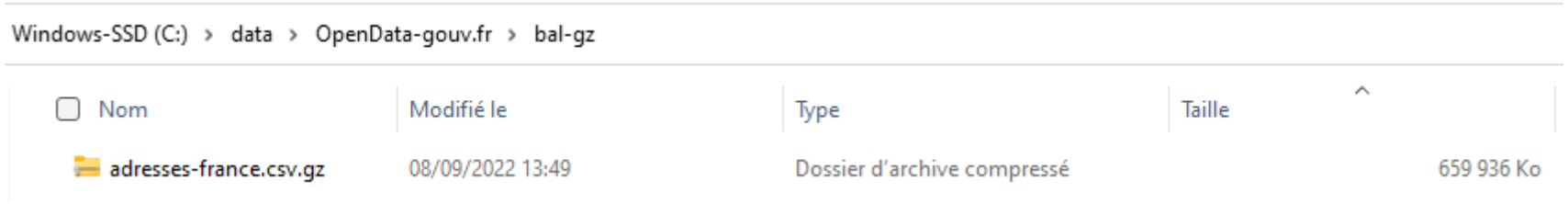


3 280 937 / (128*1024) = 25.03

```
scala> val ds = spark.read.format("csv").option("delimiter",";").load("C:/data/OpenData-gouv.fr/bal")
val ds: org.apache.spark.sql.DataFrame = [_c0: string, _c1: string ... 17 more fields]

scala> ds.toJavaRDD.getNumPartitions
val res0: Int = 26
```

Parquet is Compressable { snappy | gz }

*.gz is NOT Splitteable !



CSV file was 3.2 Go, now 660 Mo in .gz

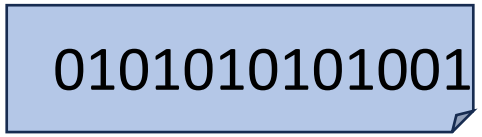
but NOT Splitteable => 1 spark partition, reading (CPU intensive) by 1 Thread only !

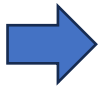
~8 times slower on a 8 cores PC

```
scala> val csvGzDs = spark.read.format("csv").option("delimiter",";").load("C:/data/OpenData-gouv.fr/bal-gz")
23/11/05 20:17:12 WARN ZlibFactory: Failed to load/initialize native-zlib library
val csvGzDs: org.apache.spark.sql.DataFrame = [_c0: string, _c1: string ... 17 more fields]

scala> csvGzDs.count()
[Stage 2:>                                                          (0 + 1) / 1]
```

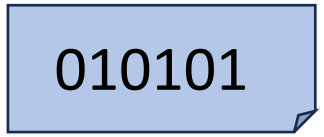
Compressions Algorithm

data



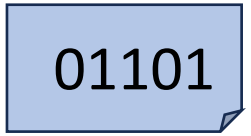


.snappy



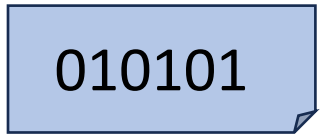
fast compression/decompression
(focus on speed)

.gz (gzip)



slower-compression,
better compression ratio
(focus on size)

.lz4



compromise between
fast / compression ratio

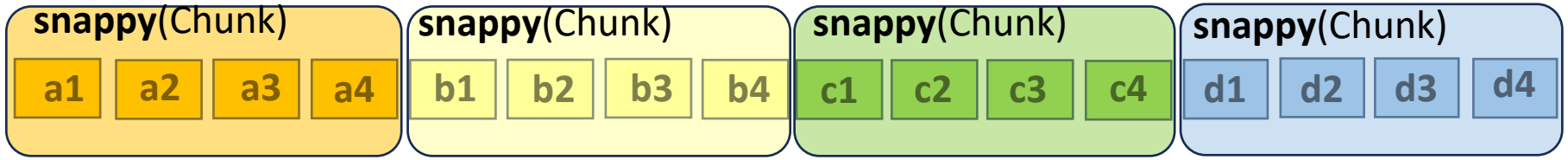
PARQUET.{snappy | gz}

Logical view: rows - columns

a1	b1	c1	d1
a2	b2	c2	d2
a3	b3	c3	d3
a4	b4	c4	d4

On Disk: **Columnar NO compression**

On Disk: **Columnar parquet.{snappy | gz }**



every "Chunk" of column data are compressed INDEPENDENTLY
=> file is STILL Splitteable

Parquet use Encodings

Run-length encoding (RLE)

a
a
a
bb
bb
bb
bb
bb
bb
c
c
c
c

repeated **x3** times

repeated **x6** times

repeated **x4** times

Var Length Encoder:

3 a
6 bb
4 c

Size of Adding "Constant" Column

adding a column with only "0" for Billions of rows

=> take only ~100 bytes per RowGroup

Dictionary Encoding

Manchester City,
Arsenal,
Manchester City,
FC Barcelone,
Arsenal,
Newcastle,
Manchester United,
Newcastle,
FC Barcelone,
Arsenal,
Manchester United,
…

Dictionary Encoder:

Distinct Dictionary Values:

1=Manchester City
2=Arsenal
3=FC Barcelone
4=Newcastle
5..

Value Indexes:

1, 2, 1, 3, 4, 5 ..

Dictionary Size Limit

By default, Dictionary size = max 1 Mega (per RowGroup - Column Chunk)

When using

Huge RowGroup (> 128Mo) => less Dictionaries used

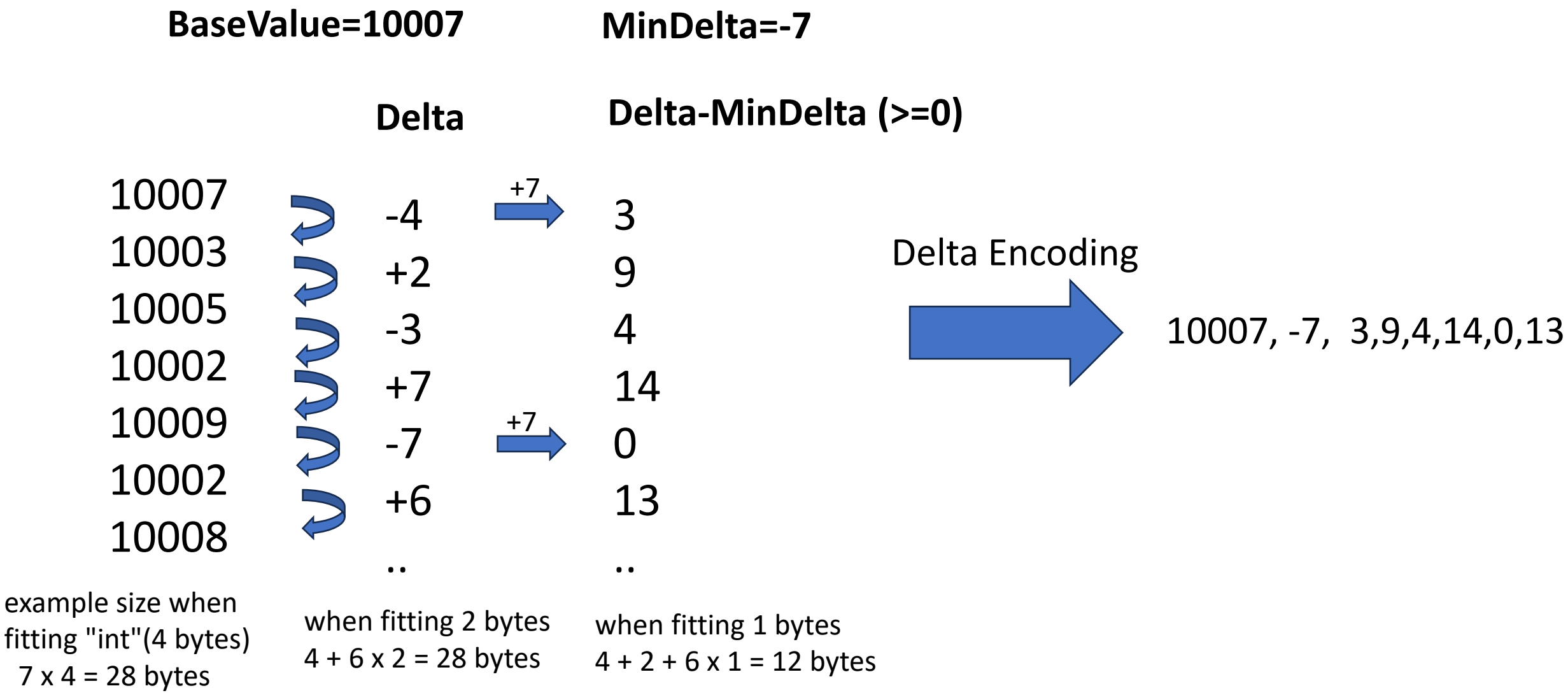
Small RowGroup (32M, 64Mo) => more Dictionaries

Parameters:

parquet.enable.dictionary=true (default)

parquet.dictionary.page.size=1M

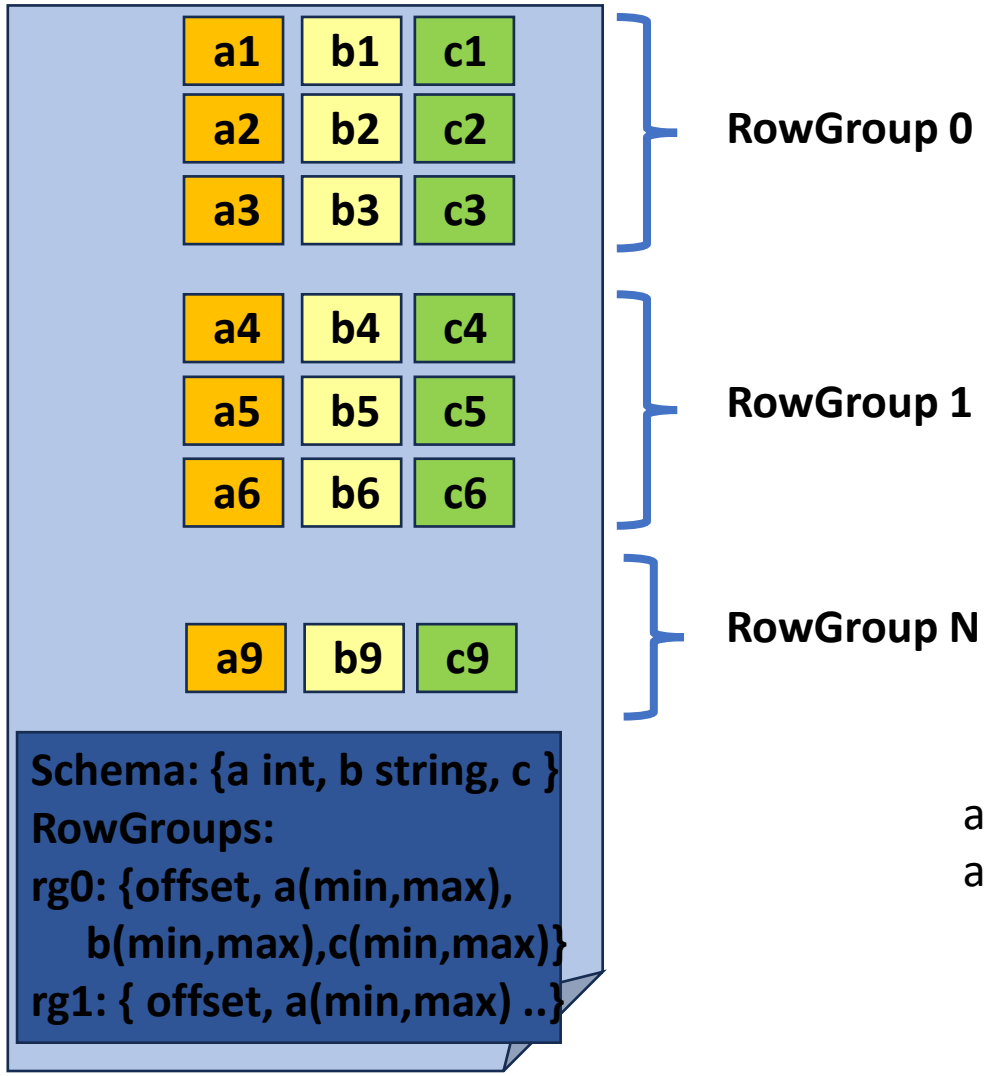
Delta Encoding



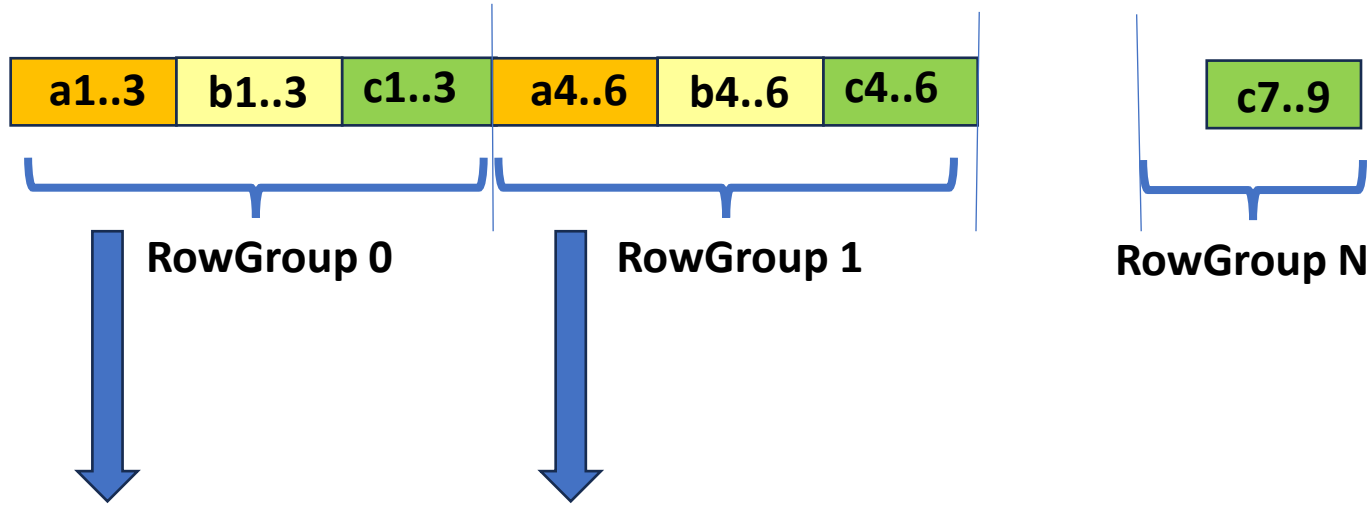
Parquet use Statistics

Column Statistics: min/max Value per RowGroup

Logical view: rows - columns



On Disk: **columnar**



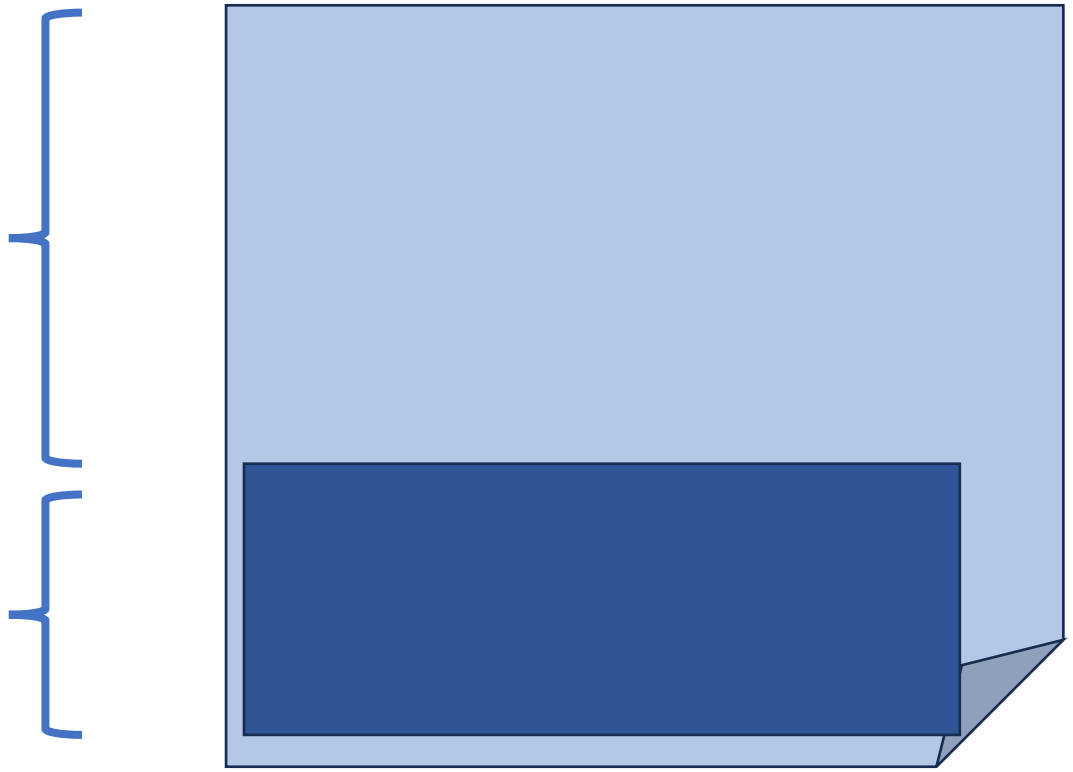
a min = min (a1,a2,..a3)
a max = max(a1,a2,..a3)

a min = min (a4,a5,..a6)
a max = max(a4,a5,..a6)

Reading Metadata
=> read schema + offset + statistics

data part
(usually N x blocks of 128 Mo)

footer: metadata part
(usually <= 50 ko)

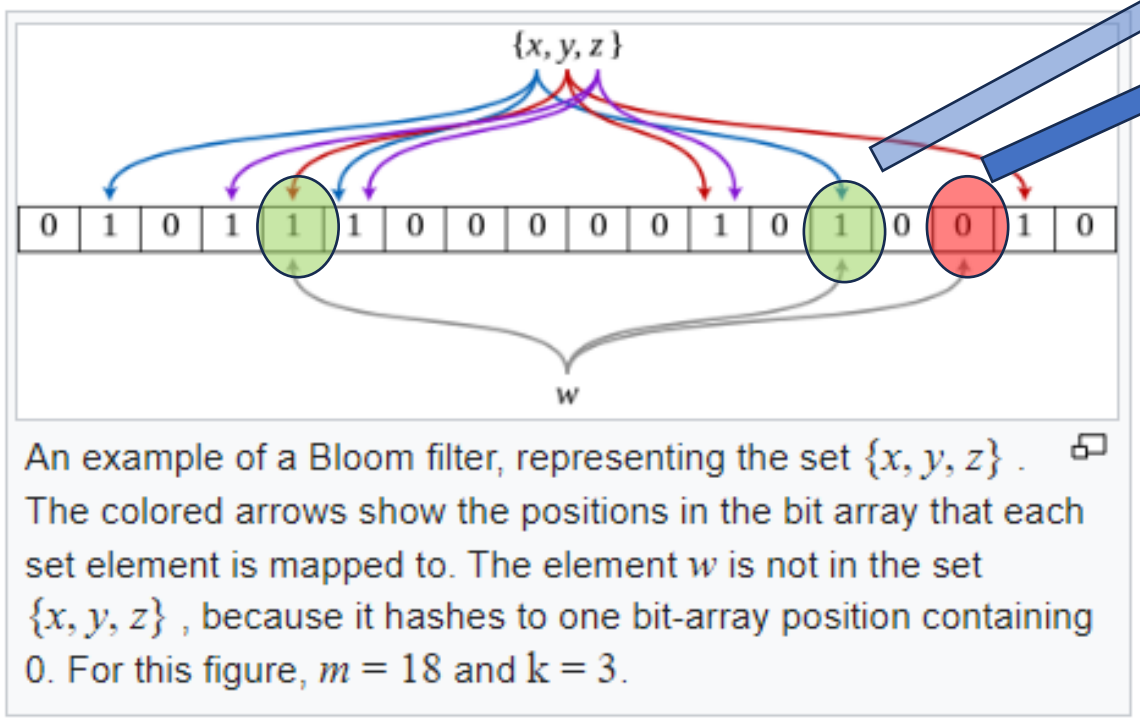


Parquet use Bloom Filter

Bloom Filter

is W in the set {x,y,z} ?

hash(W) = 0000100...10100



An example of a Bloom filter, representing the set $\{x, y, z\}$. The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set $\{x, y, z\}$, because it hashes to one bit-array position containing 0. For this figure, $m = 18$ and $k = 3$.

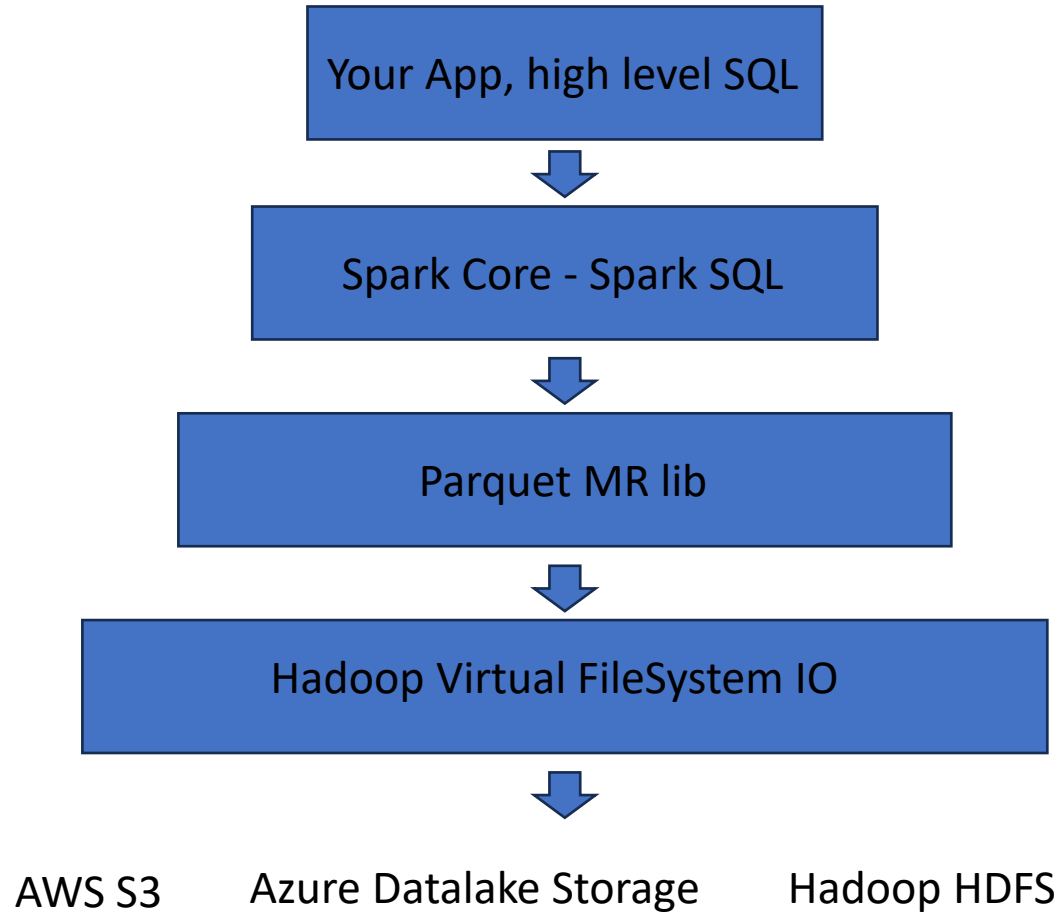
Probably Yes ...

Firm NO

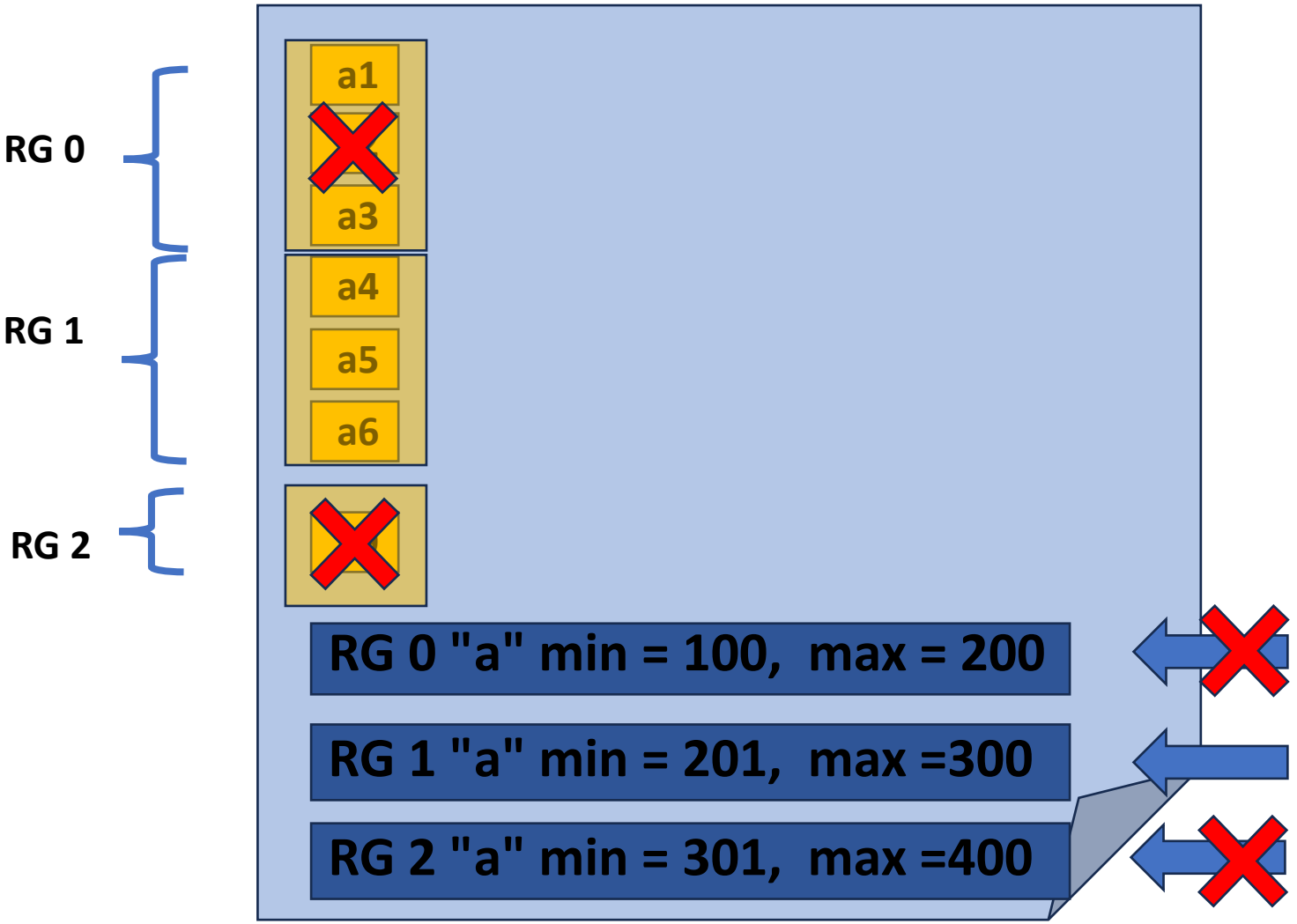
https://en.wikipedia.org/wiki/Bloom_filter

Parquet Predicate-Push-Down

Push-Down :
from Spark -> Parquet Lib -> IO Storage



Statistics for skip read RowGroup
SELECT .. WHERE column=value



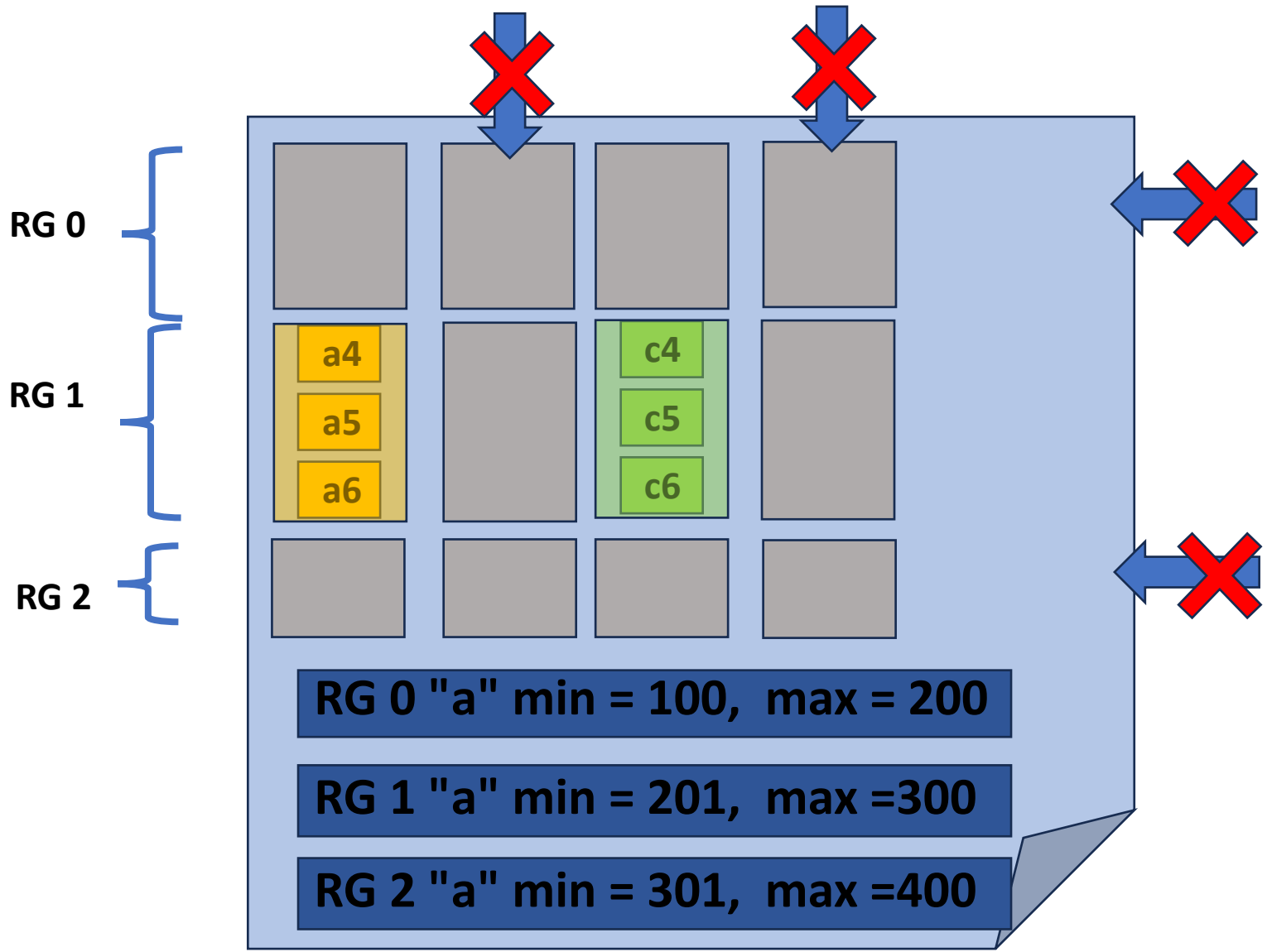
SELECT ...
WHERE a = 250

NO match max < a

OK min <= a <= max (maybe false positive)

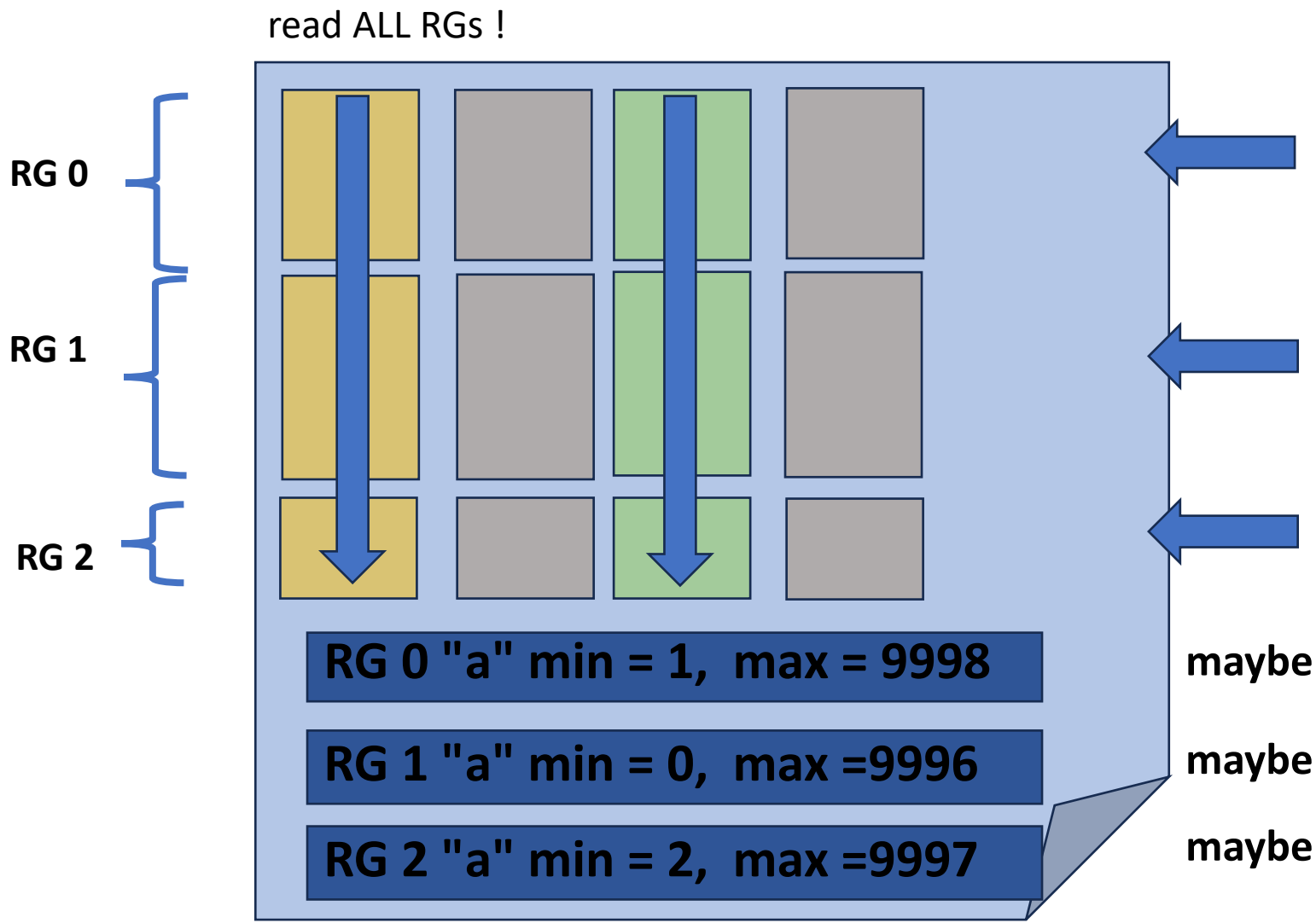
NO match a < min

Column Pruning
+ RowGroup Pruning(PPD) = minimal Reads



SELECT c
WHERE a = 250

Badly sorted files
=> Bad min/max statistics => False positives



SELECT c
WHERE a = 250

maybe false positive min <= a <= max

maybe false positive min <= a <= max

maybe false positive min <= a <= max

Optim Write Once / Read Many

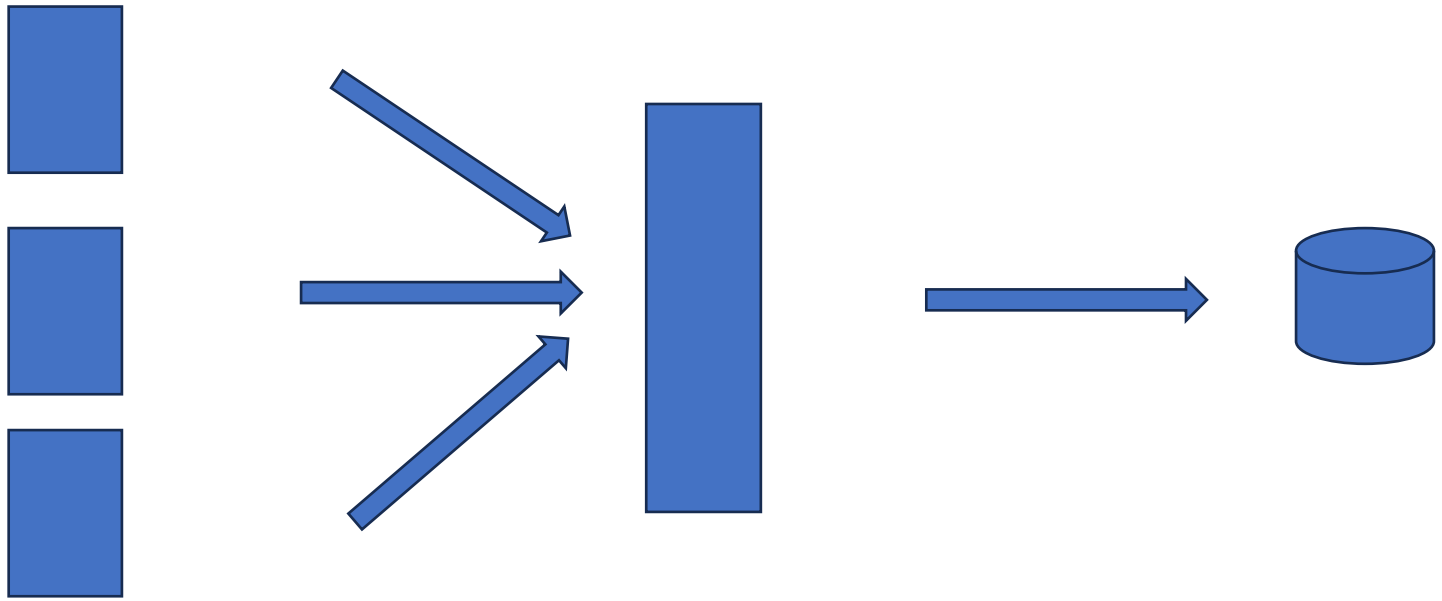
when writing Parquet files ... think how file might be read later !

spend CPU when writing to save CPU / IO later reading

```
dataset
    .repartition(nRepartitionCount)
    // or  .repartition("col1", nRepartitionHash)

    .sortWithinPartitions("colA", "colB")

    .write
    .option("parquet.block.size", 32*MEGA)

    .format("parquet")
    .save("file://some-dir")
    // or
    // .format("hive").insertInto("hiveDb.hiveTableName")
```

avoid many small files
dataset.repartition(1)
or .coalesce(1)

dataset
.repartition(nRepartitionCount) // or .coalesce(nRepartitionCount)



N spark in-memory Partitions
distributed over N executors

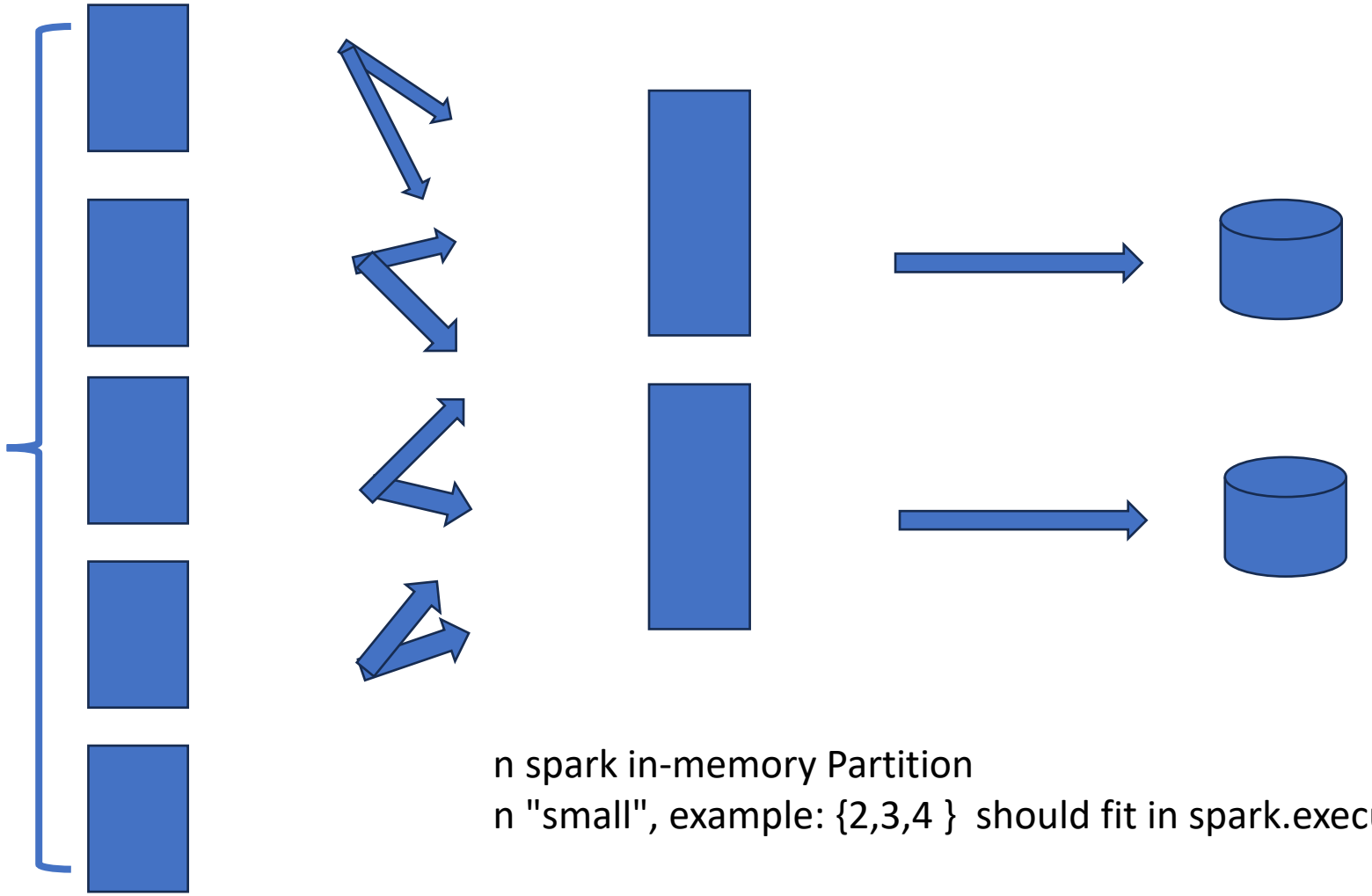
1 spark in-memory Partition
(should fit in spark.executor.memory !!)

write 1 parquet file

Does not fit in memory.. compromise to .repartition(smallN)

typically 200 partitions

default
spark.shuffle.size=200

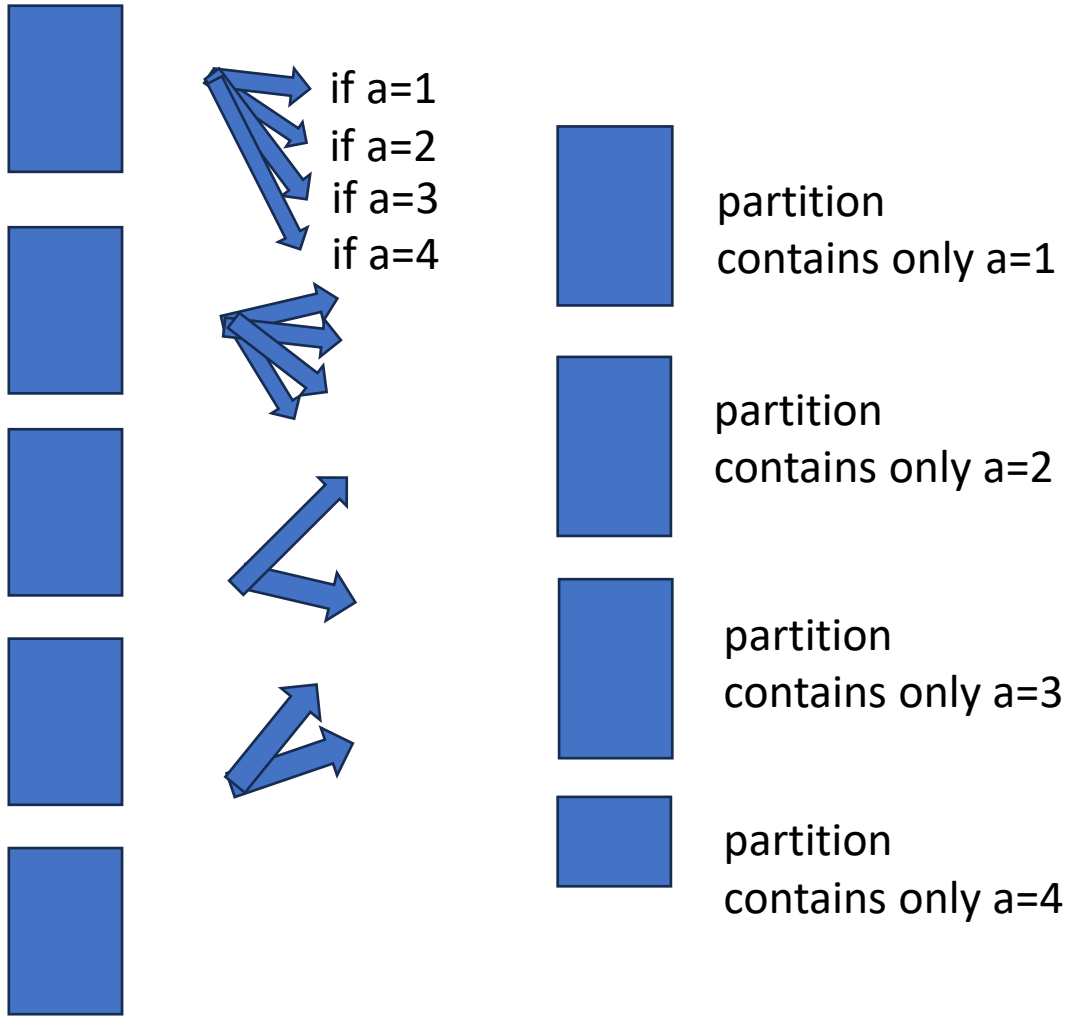


n spark in-memory Partition
n "small", example: {2,3,4 } should fit in spark.executor.memory !!

.repartition("column")

typical usage:
column "a" has FEW distinct values { 1, 2, 3, 4 }

ds = dataset.repartition("a")

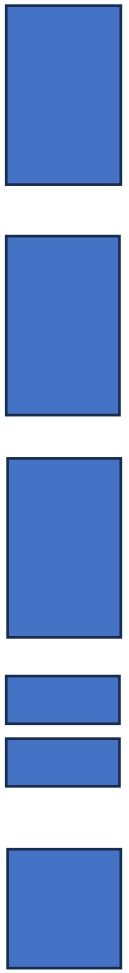


.repartition("column", nHashCount)

example ... having many distinct values

col "a" values in [1, 2, 3, ... 500000]

=> h = hash(a) % 15 in [0, 1, .. 14]



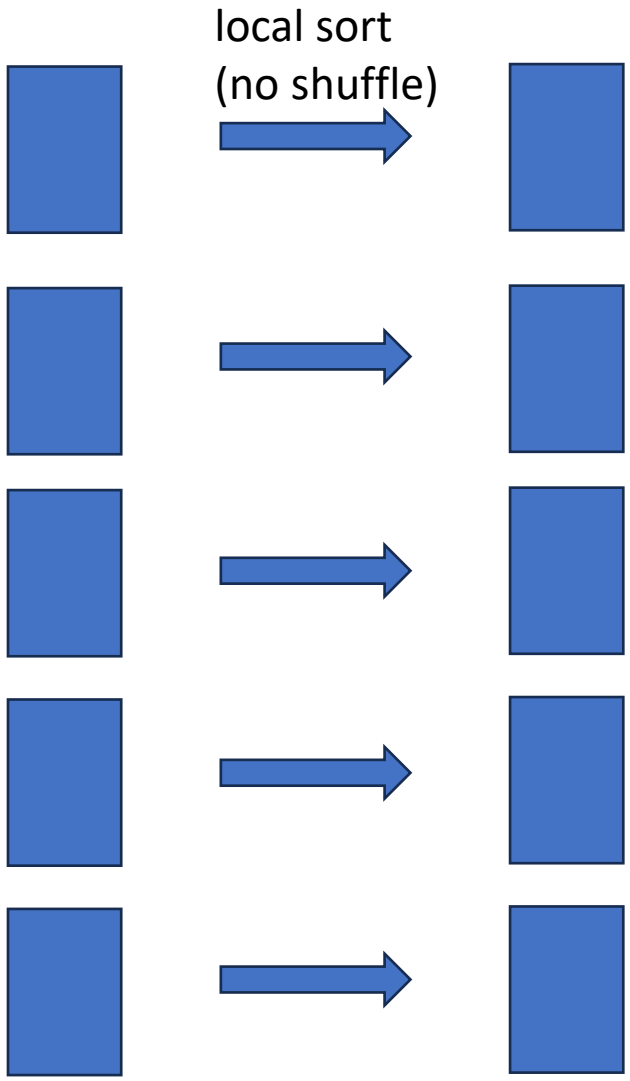
partition
contains only a%15==0 i.e. {0, 15, 30, 45, ..}

partition
contains only a%15==1 i.e. {1, 16, 31, 46, ..}

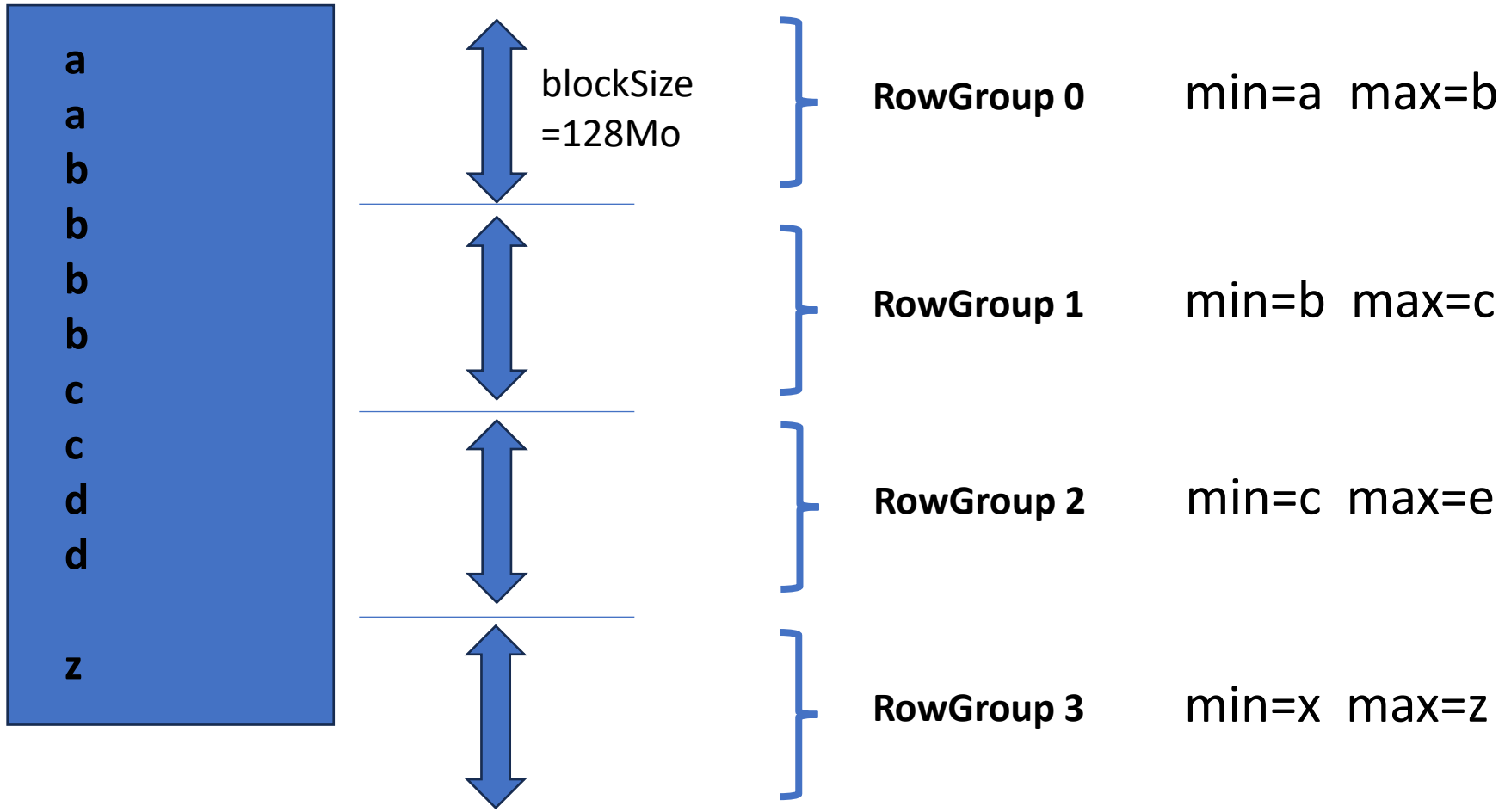
partition
contains only a%15==2 i.e. {2, 12, 32, 47, ..}

partition
contains only a%15==14 i.e. {14, 29, 44, ..}

.sortWithinPartitions("a", "b", ...)

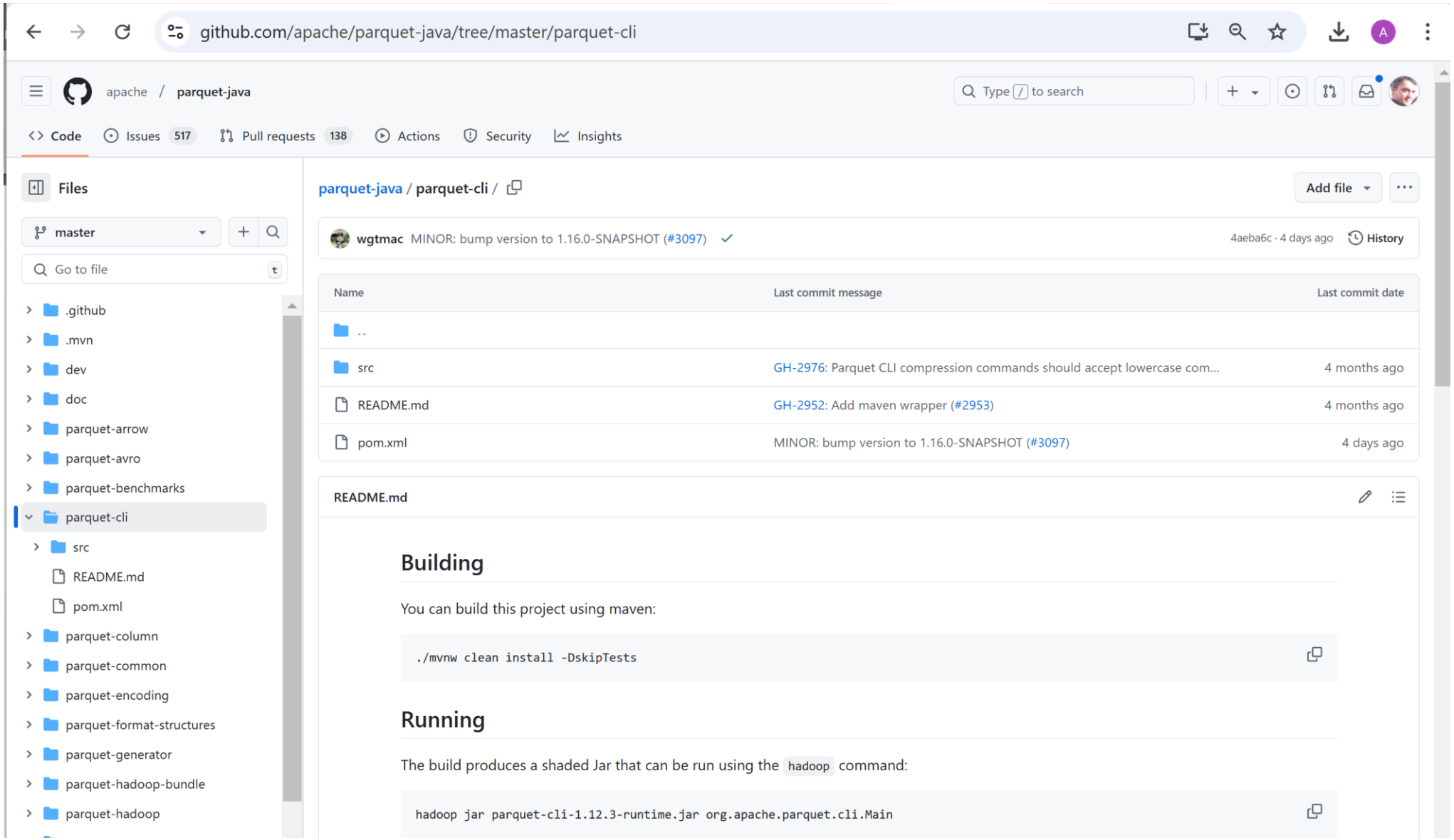


.sortWithinPartitions
=> RowGroups stats more "compact"

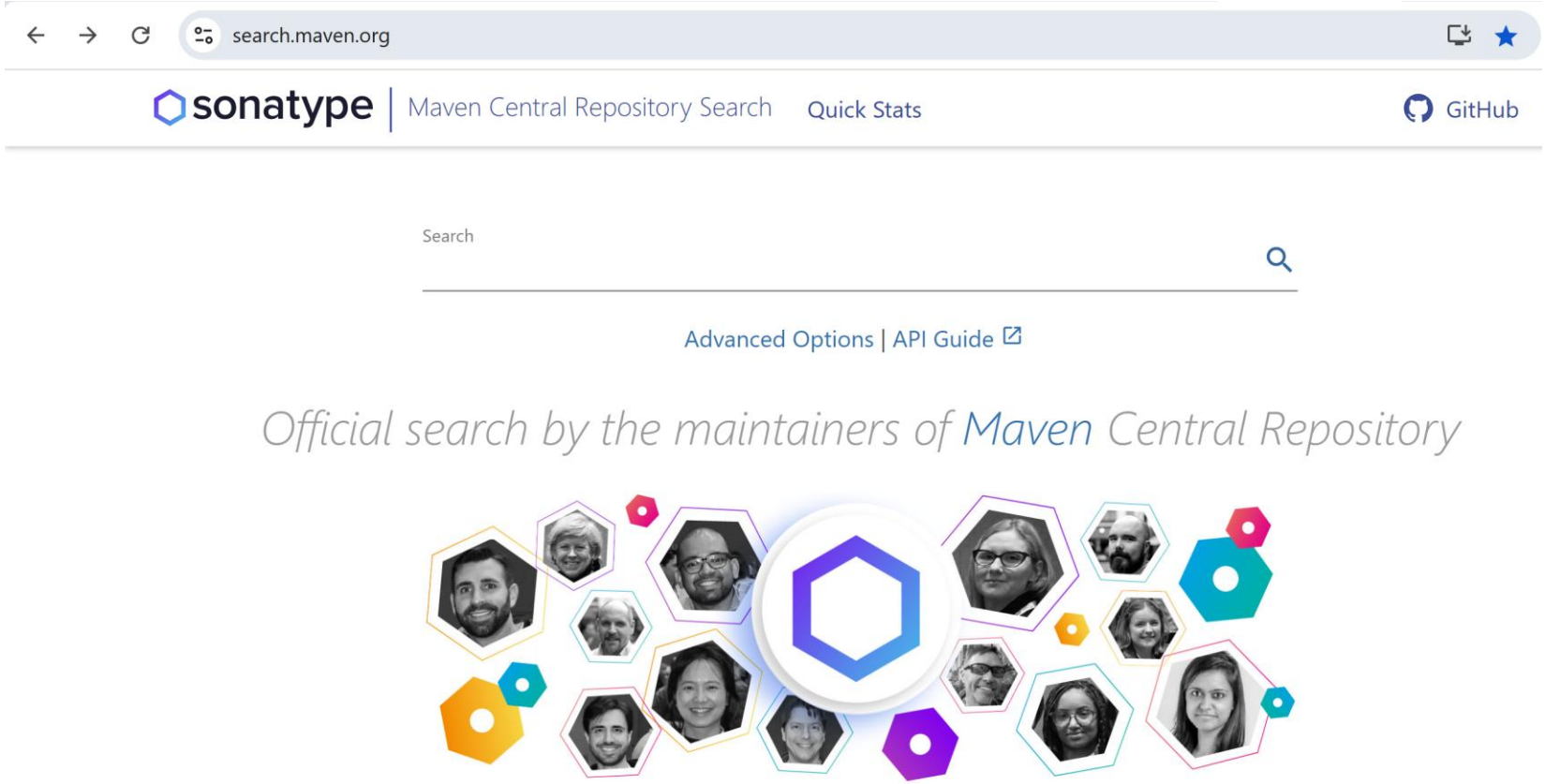


Analyzing Parquet Files, using "parquet-cli"

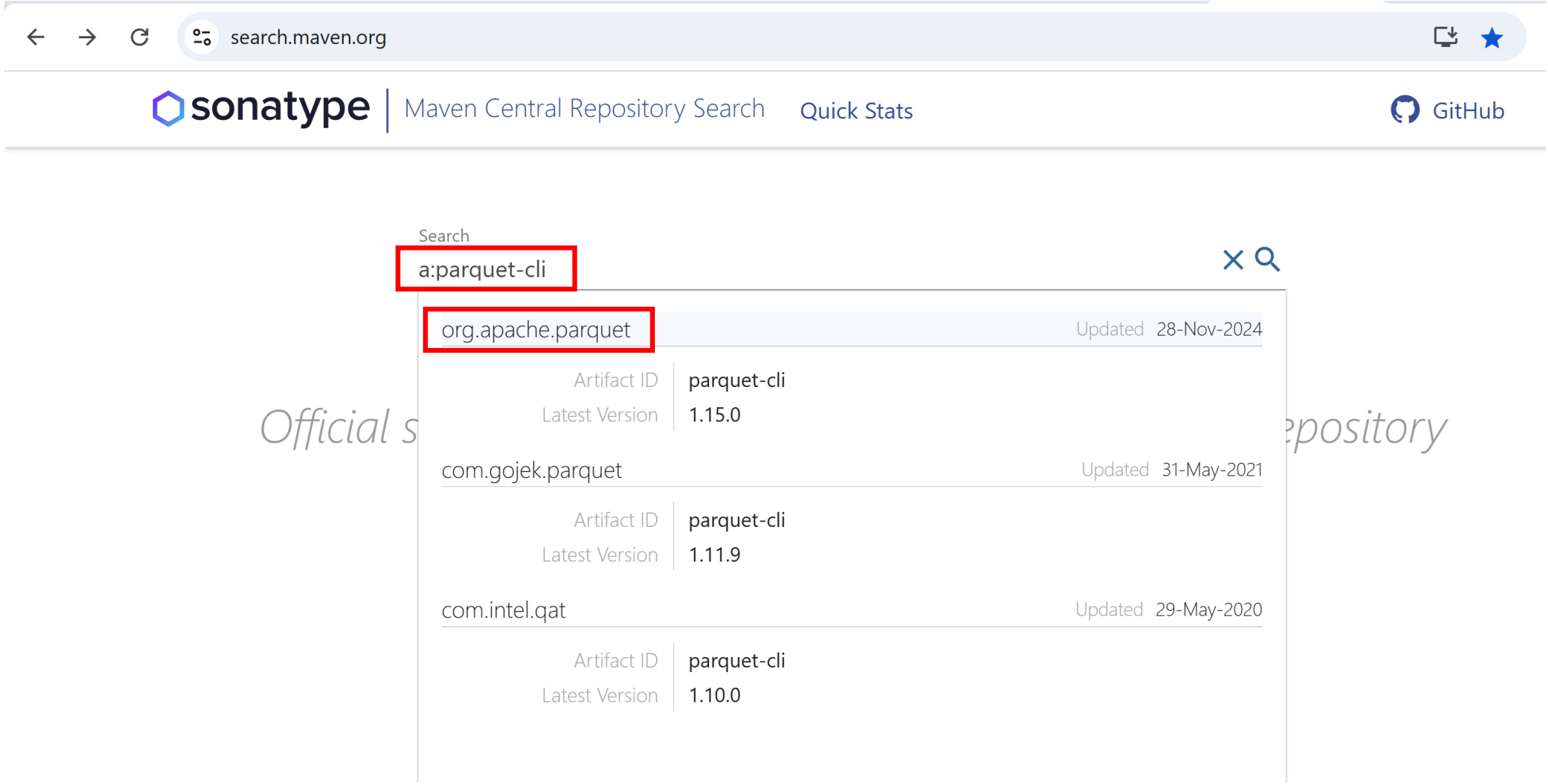
Analysing Parquet File, using "parquet-cli"



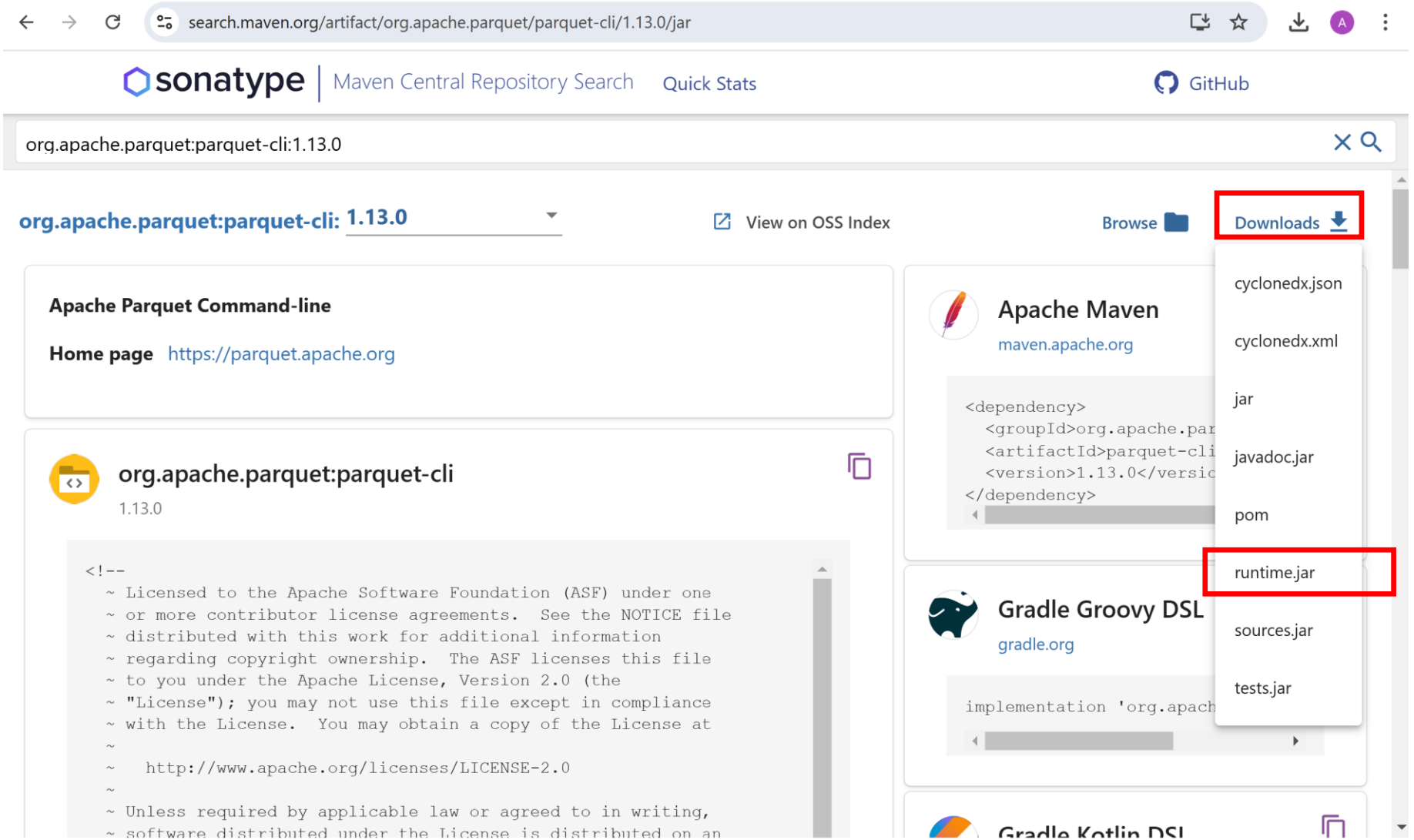
Search jar in maven repo: https://search.maven.org



type search "a:parquet-cli" g:org.apache.parquet



Downloading "parquet-cli" runtime.jar for same version 1.13.0 (same as spark/jars/*.jar)



Launching parquet-cli

java -jar parquet-cli-runtime.jar help

(using shaded jar, otherwise may need to add hadoop classpath)

```
C:\apps\spark>java -jar parquet-cli-1.15.0-SNAPSHOT-runtime.jar help

Usage: parquet [options] [command] [command options]

  Options:

    -v, --verbose, --debug
        Print extra debugging information

  Commands:

    help
        Retrieves details on the functions of other commands
    meta
        Print a Parquet file's metadata
    pages
        Print page summaries for a Parquet file
    dictionary
        Print dictionaries for a Parquet column
    check-stats
        Check Parquet files for corrupt page and column stats (PARQUET-251)
    schema
        Print the Avro schema for a file
    csv-schema
        Build a schema from a CSV data sample
    convert-csv
```

parquet-cli meta [1/2]

java -jar parquet-cli-runtime.jar meta file.parquet

```
C:\apps\spark>java -jar parquet-cli-1.15.0-SNAPSHOT-runtime.jar meta  C:\apps\spark\spark
t

File path:  C:\apps\spark\spark-warehouse\db1.db\addr\part-c000.snappy.parquet
Created by: parquet-mr version 1.13.1 (build db4183109d5b734ec5930d870cdae161e408ddba)
Properties:
                   org.apache.spark.version: 3.5.0
  org.apache.spark.sql.parquet.row.metadata: {"type":"struct","fields":[{"name":"uid_adre
e":"cle_interop","type":"string","nullable":true,"metadata":{}},{"name":"commune_insee","
mmune_nom","type":"string","nullable":true,"metadata":{}},{"name":"commune_deleguee_insee
:"commune_deleguee_nom","type":"string","nullable":true,"metadata":{}},{"name":"voie_nom"
lieudit_complement_nom","type":"string","nullable":true,"metadata":{}},{"name":"numero","
uffixe","type":"string","nullable":true,"metadata":{}},{"name":"position","type":"string"
e","nullable":true,"metadata":{}},{"name":"y","type":"double","nullable":true,"metadata":
ata":{}},{"name":"lat","type":"double","nullable":true,"metadata":{}},{"name":"cad_parcel
e":"source","type":"string","nullable":true,"metadata":{}},{"name":"certification_commune
:"date_der_maj","type":"date","nullable":false,"metadata":{}}]}
Schema:
message spark_schema {
  optional binary uid_adresse (STRING);
  optional binary cle_interop (STRING);
  optional binary commune_insee (STRING);
  optional binary commune_nom (STRING);
  optional int32 commune_deleguee_insee;
  optional binary commune_deleguee_nom (STRING);
  optional binary voie_nom (STRING);
  optional binary lieudit_complement_nom (STRING);
  optional int32 numero;
  optional binary suffixe (STRING);
  optional binary position (STRING);
  optional double x;
```

parquet-cli meta [2/2]

```
Row group 0:  count: 219962  71,92 B records  start: 4  total(compressed): 15,086 MB total(uncompressed):33,177 MB
--------------------------------------------------------------------------------
                         type      encodings count     avg size   nulls   min / max
uid_adresse              BINARY    S _ R_ F  219962    34,79 B    47399   " @a:00003b17-45be-48c1-aa..." / "Argence""
cle_interop              BINARY    S   _     219962    5,47 B     0       "01001_0005_00026" / "54"
commune_insee            BINARY    S _ R     219962    0,01 B     1       "01001" / "01350"
commune_nom              BINARY    S _ R     219962    0,02 B     0       "Ambléon" / "Évosges"
commune_deleguee_insee   INT32     S _ R     219962    0,02 B     202278  "1015" / "1442"
commune_deleguee_nom     BINARY    S _ R     219962    0,02 B     202277  "6518380.18" / "Étrez"
voie_nom                 BINARY    S _ R     219962    1,44 B     0       """le château""" / "Îlot Grammont"
lieudit_complement_nom   BINARY    S _ R     219962    0,09 B     197279  """le château""" / "Étrez"
numero                   INT32     S _ R     219962    1,45 B     1       "0" / "99999"
suffixe                  BINARY    S _ R     219962    0,11 B     206335  "1" / "z"
position                 BINARY    S _ R     219962    0,25 B     15376   "bâtiment" / "service technique"
x                        DOUBLE    S   _     219962    5,06 B     226     "-0.0" / "943078.82"
y                        DOUBLE    S   _     219962    5,37 B     227     "6505390.74" / "6603083.59"
long                     DOUBLE    S   _     219962    7,99 B     227     "4.730471" / "6.163086"
lat                      DOUBLE    S   _     219962    7,64 B     227     "45.617249" / "46.506743"
cad_parcelles            BINARY    S _ R     219962    1,87 B     167172  "010010000A1033" / "01380000_E0291"
source                   BINARY    S _ R     219962    0,06 B     2266    "arcep" / "inconnue"
certification_commune    INT32     S _ R     219962    0,03 B     1       "0" / "1"
date_der_maj             INT32     S _ R     219962    0,23 B     0       "1901-01-01" / "2024-12-05"

Row group 1:  count: 215938  72,72 B records  start: 15819187  total(compressed): 14,976 MB total(uncompressed):33,489 MB
--------------------------------------------------------------------------------
                         type      encodings count     avg size   nulls   min / max
uid_adresse              BINARY    S   _     215938    36,79 B    39793   " @a:00003774-58fc-46a7-b1..." / " @v:ffe769c6-a8b
cle_interop              BINARY    S   _     215938    5,19 B     0       "01350_0005_00707" / "02546_rcnpc4_00054"
commune_insee            BINARY    S _ R     215938    0,02 B     0       "01350" / "02546"
commune_nom              BINARY    S _ R     215938    0,04 B     0       "Abbécourt" / "Évergnicourt"
commune_deleguee_insee   INT32     S   _     215938    0,01 B     208894  "1059" / "2811"
commune_deleguee_nom     BINARY    S   _     215938    0,03 B     208894  "Anizy-le-Château" / "Virieu-le-Petit"
```

parquet-cli meta | grep "Row group"

java -jar parquet-cli-runtime.jar meta file.parquet > meta.txt

grep "Row group" meta.txt | head

Row group 0: count: 219962 71,92 B records start: 4 total(compressed): 15,086 MB total(uncompressed):33,177 MB
Row group 1: count: 215938 72,72 B records start: 15819187 total(compressed): 14,976 MB total(uncompressed):33,489 MB
Row group 2: count: 213992 73,68 B records start: 31523114 total(compressed): 15,036 MB total(uncompressed):33,821 MB
Row group 3: count: 233199 70,29 B records start: 47289138 total(compressed): 15,631 MB total(uncompressed):34,232 MB
Row group 4: count: 231558 69,04 B records start: 63679653 total(compressed): 15,246 MB total(uncompressed):33,383 MB
Row group 5: count: 219962 74,89 B records start: 79665824 total(compressed): 15,710 MB total(uncompressed):34,531 MB
Row group 6: count: 218409 73,91 B records start: 96138854 total(compressed): 15,394 MB total(uncompressed):33,957 MB
Row group 7: count: 199907 75,02 B records start: 112281114 total(compressed): 14,301 MB total(uncompressed):31,914 MB
Row group 8: count: 218409 69,87 B records start: 127277278 total(compressed): 14,552 MB total(uncompressed):34,174 MB
Row group 9: count: 272568 65,80 B records start: 142536439 total(compressed): 17,104 MB total(uncompressed):35,616 MB

grep "Row group" meta.txt | tail

Row group 108: count: 252150 64,55 B records start: 1719335249 total(compressed): 15,522 MB total(uncompressed):34,027 MB
Row group 109: count: 215240 71,68 B records start: 1735611436 total(compressed): 14,715 MB total(uncompressed):34,014 MB
Row group 110: count: 219962 68,48 B records start: 1751040857 total(compressed): 14,365 MB total(uncompressed):32,978 MB
Row group 111: count: 219962 68,18 B records start: 1766103960 total(compressed): 14,302 MB total(uncompressed):32,914 MB
Row group 112: count: 234887 64,52 B records start: 1781100645 total(compressed): 14,454 MB total(uncompressed):33,355 MB
Row group 113: count: 236624 66,44 B records start: 1796256273 total(compressed): 14,993 MB total(uncompressed):37,495 MB
Row group 114: count: 234063 65,95 B records start: 1811977242 total(compressed): 14,722 MB total(uncompressed):33,626 MB
Row group 115: count: 213992 73,42 B records start: 1827414634 total(compressed): 14,983 MB total(uncompressed):34,526 MB
Row group 116: count: 230100 69,58 B records start: 1843125866 total(compressed): 15,269 MB total(uncompressed):36,039 MB
Row group 117: count: 152925 67,36 B records start: 1859136956 total(compressed): 9,823 MB total(uncompressed):26,622 MB

parquet-cli column-size

java -jar parquet-cli-runtime.jar column-size file.parquet

```
C:\apps\spark>java -jar parquet-cli-1.15.0-SNAPSHOT-runtime.jar column-size  C:\apps\spark\spa
commune_deleguee_insee-> Size In Bytes: 431641 Size In Ratio: 2.3089352E-4
commune_insee-> Size In Bytes: 322609 Size In Ratio: 1.725701E-4
numero-> Size In Bytes: 26951190 Size In Ratio: 0.014416738
voie_nom-> Size In Bytes: 35412828 Size In Ratio: 0.01894304
uid_adresse-> Size In Bytes: 882820344 Size In Ratio: 0.4722385
date_der_maj-> Size In Bytes: 7501158 Size In Ratio: 0.0040125214
cle_interop-> Size In Bytes: 139359234 Size In Ratio: 0.07454608
source-> Size In Bytes: 1142012 Size In Ratio: 6.108854E-4
long-> Size In Bytes: 213866608 Size In Ratio: 0.11440159
commune_deleguee_nom-> Size In Bytes: 836453 Size In Ratio: 4.4743568E-4
suffixe-> Size In Bytes: 3721170 Size In Ratio: 0.0019905292
cad_parcelles-> Size In Bytes: 66126109 Size In Ratio: 0.035372198
certification_commune-> Size In Bytes: 673339 Size In Ratio: 3.601827E-4
x-> Size In Bytes: 133896306 Size In Ratio: 0.07162385
lieudit_complement_nom-> Size In Bytes: 2886263 Size In Ratio: 0.0015439206
y-> Size In Bytes: 143779166 Size In Ratio: 0.07691039
commune_nom-> Size In Bytes: 549711 Size In Ratio: 2.9405157E-4
position-> Size In Bytes: 4716951 Size In Ratio: 0.0025231927
lat-> Size In Bytes: 204444396 Size In Ratio: 0.109361455
```

Column Sizes ... sorting

cat column-size
 | sed 's/-> Size In Bytes//g' | sed 's/ Size In Ratio//g' | sed 's/: /;/g' > column-size.csv

Column	Size	Ratio	Size in Mo
uid_adresse	882820344	47,22%	841,9
long	213866608	11,44%	204,0
lat	204444396	10,94%	195,0
y	143779166	7,69%	137,1
cle_interop	139359234	7,45%	132,9
x	133896306	7,16%	127,7
cad_parcelles	66126109	3,54%	63,1
voie_nom	35412828	1,89%	33,8
numero	26951190	1,44%	25,7
date_der_maj	7501158	0,40%	7,2
position	4716951	0,25%	4,5
suffixe	3721170	0,20%	3,5
lieudit_complement_nom	2886263	0,15%	2,8
source	1142012	0,06%	1,1
commune_deleguee_nom	836453	0,04%	0,8
certification_commune	673339	0,04%	0,6
commune_nom	549711	0,03%	0,5
commune_deleguee_insee	431641	0,02%	0,4
commune_insee	322609	0,02%	0,3
total	1869437488	100,00%	1782,8

Parquet Column Sizes Summary

zipcode ("commun_insee") : 0.3 Mo
+
city name ("commun_nom") : 0.5 Mo
+
street name ("voie_nom") : 33.8 Mo
+
numero : 25.7 Mo

=> 60 Mega (3% of file)

+ longitude, latitude : 204 Mo + 195 Mo
=> 459 Mo

most of the column size is

uid_address : 840 Mo (47.2%)
+
x, y ... redundant with longitude,lattitude
+
cle_interop : 132 Mo (7.4%)

... we will see parquet "column-pruning" in action : skipping unnecessary columns !

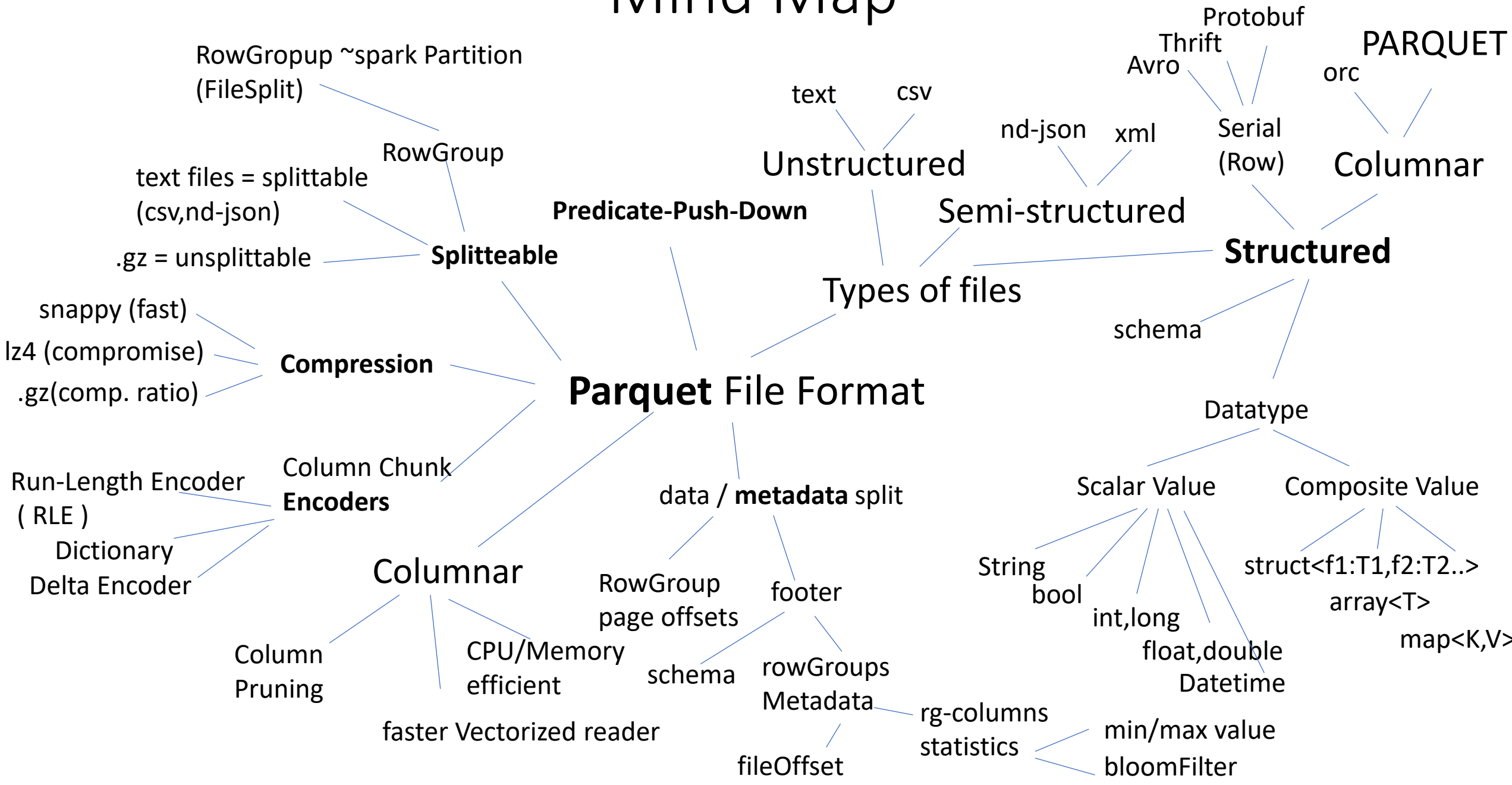
Conclusion

Parquet File Format is AMAZING

Spark is great using Parquet

Doing BigData processing = doing Spark + Parquet
with good .repartition().sortWithinPartition() ...

Mind Map


RowGropup ~spark Partition
(FileSplit)
RowGroup
text files = splittable
(csv,nd-json)
.gz = unsplittable
Splitteable
Predicate-Push-Down
text
csv
Unstructured
nd-json
xml
Semi-structured
Protobuf
Thrift
Avro
Serial
(Row)
PARQUET
orc
Columnar
Types of files
Structured
schema
snappy (fast)
lz4 (compromise)
.gz(comp. ratio)
Compression
Parquet File Format
Run-Length Encoder
(RLE)
Dictionary
Delta Encoder
Column Chunk
Encoders
Columnar
data / metadata split
RowGroup
page offsets
footer
schema
rowGroups
Metadata
fileOffset
rg-columns
statistics
min/max value
bloomFilter
Column
Pruning
CPU/Memory
efficient
faster Vectorized reader
Datatype
Scalar Value
Composite Value
String
bool
int,long
float,double
Datetime
struct<f1:T1,f2:T2..>
array<T>
map<K,V>

Questions?

arnaud.nauwynck@gmail.com