Efficient Data Storage for Analytics

with Apache Parquet 2.0

Julien Le Dem @J_

Processing tools tech lead, Data Platform at Twitter

Nong Li nong@cloudera.com

Software engineer, Cloudera Impala

@ApacheParquet

Outline

- Why we need efficiency
- Properties of efficient algorithms
- Enabling efficiency
- Efficiency in Apache Parquet

Why we need efficiency

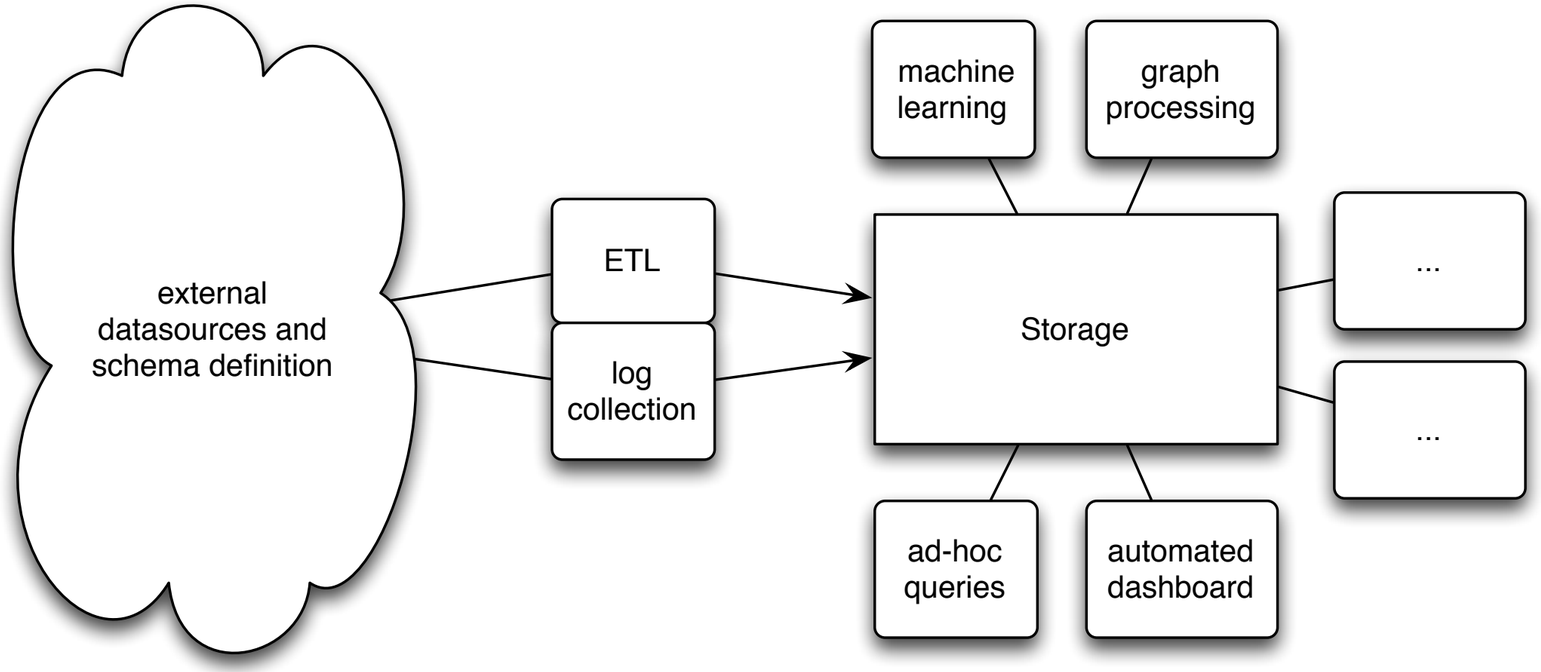
Producing a lot of data is easy

Producing a lot of derived data is even easier.
Solution: Compress all the things!

Scanning a lot of data is easy

… but not necessarily fast.
Waiting is not productive. We want faster turnaround.
Compression but not at the cost of reading speed.

Trying new tools is easy



We need a storage format interoperable with all the tools we use
and keep our options open for the next big thing.

Enter Apache Parquet

Parquet design goals

- Interoperability

- Space efficiency

- Query efficiency

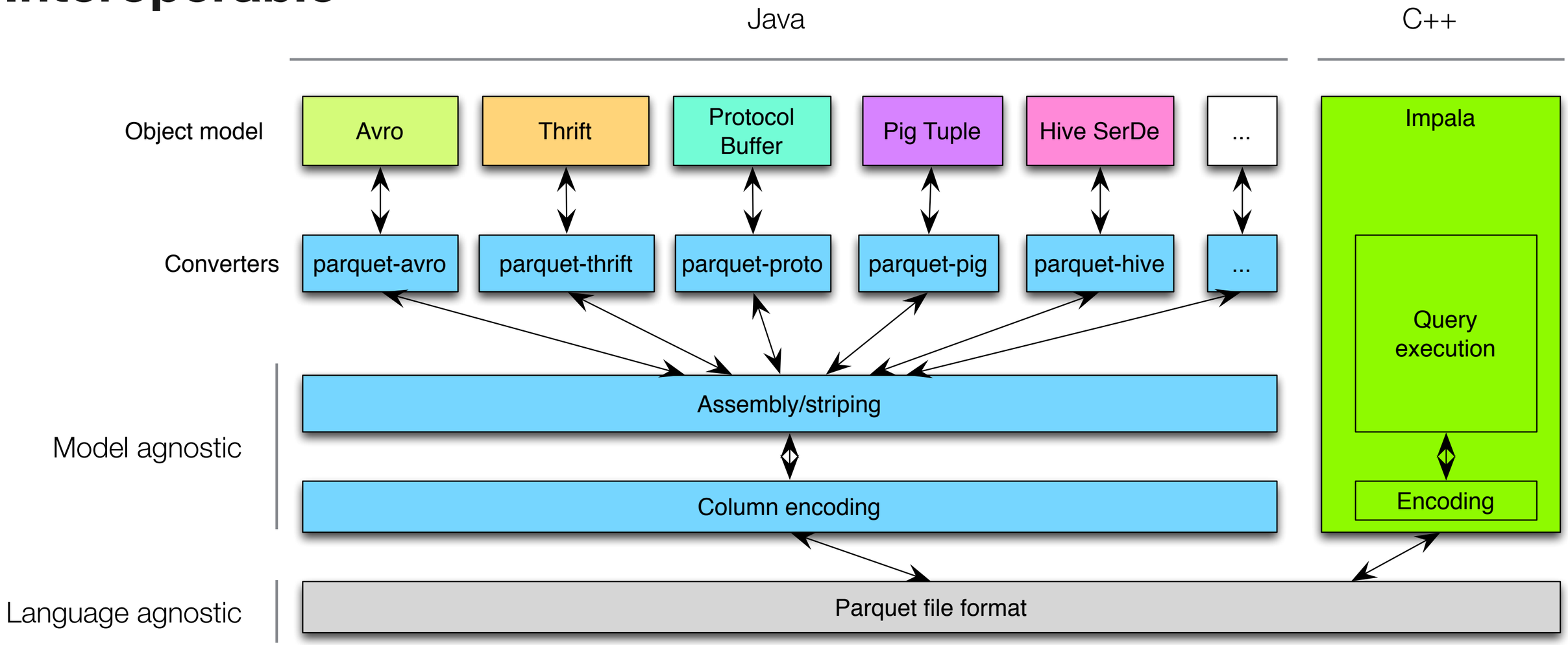
Parquet timeline

- Fall 2012: Twitter & Cloudera merge efforts to develop columnar formats
- March 2013: OSS announcement; Criteo signs on for Hive integration
- July 2013: 1.0 release. 18 contributors from more than 5 organizations.
- May 2014: Apache Incubator. 40+ contributors, 18 with 1000+ LOC. 26 incremental releases.
- Parquet 2.0 coming as Apache release



Interoperability

Interoperable



Frameworks and libraries integrated with Parquet

Query engines:

Hive, Impala, HAWQ,
IBM Big SQL, Drill, Tajo,
Pig, Presto

Frameworks:

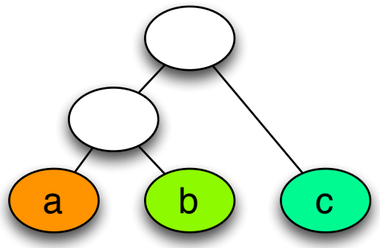
Spark, MapReduce, Cascading,
Crunch, Scalding, Kite

Data Models:

Avro, Thrift, ProtocolBuffers,
POJOs

Enabling efficiency

Columnar storage

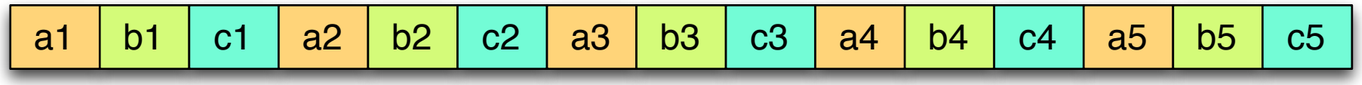


Nested schema

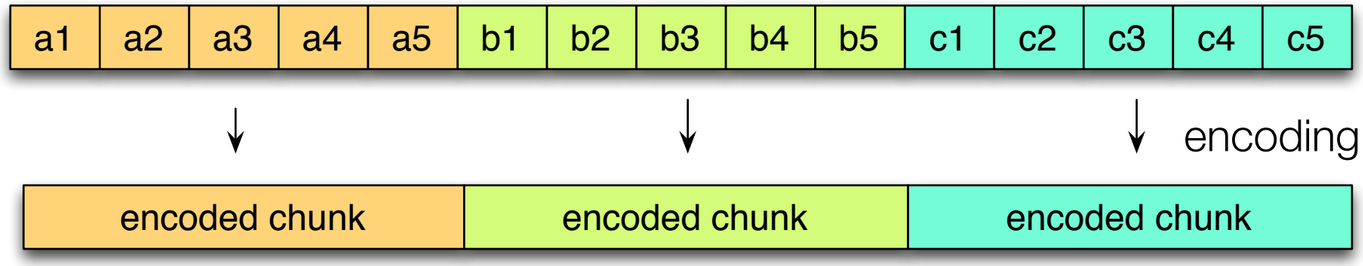
Logical table representation

a	b	c
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Row layout



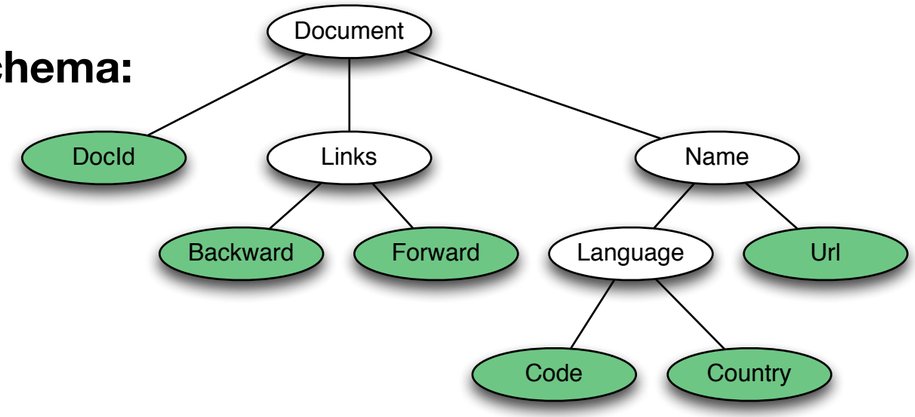
Column layout



Parquet nested representation

Borrowed from the Google Dremel paper

Schema:



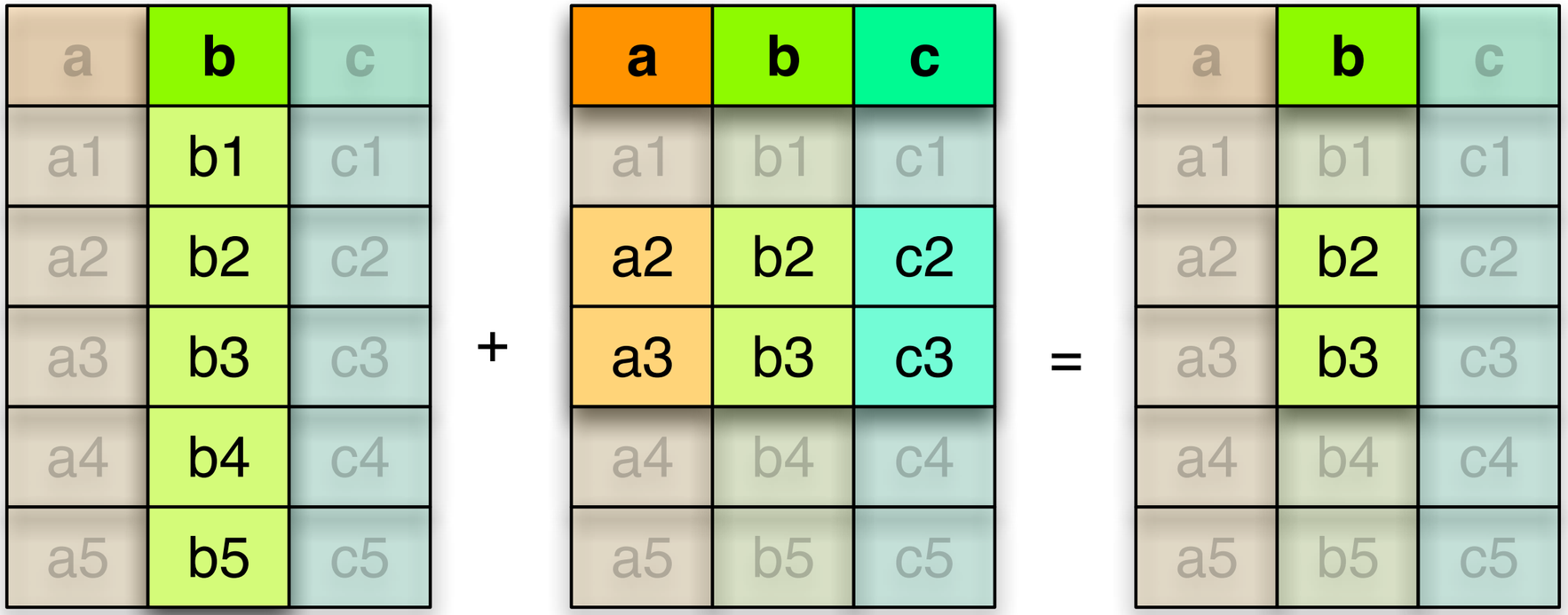
Columns:

docid
links.backward
links.forward
name.language.code
name.language.country
name.url

https://blog.twitter.com/2013/dremel-made-simple-with-parquet

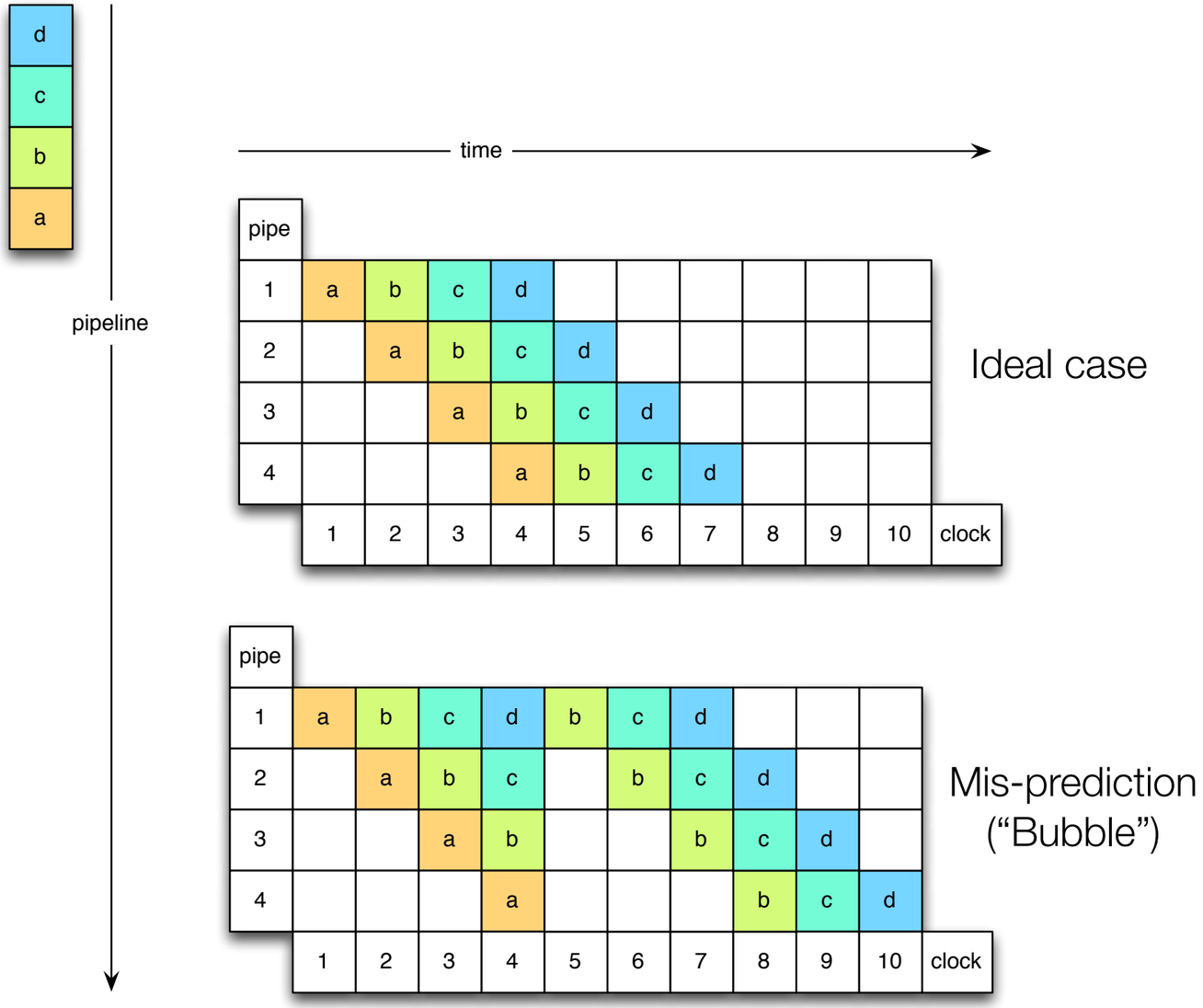
Statistics for filter and query optimization

Vertical partitioning (projection push down) + Horizontal partitioning (predicate push down) = Read only the data you need!



Properties of efficient algorithms

CPU Pipeline

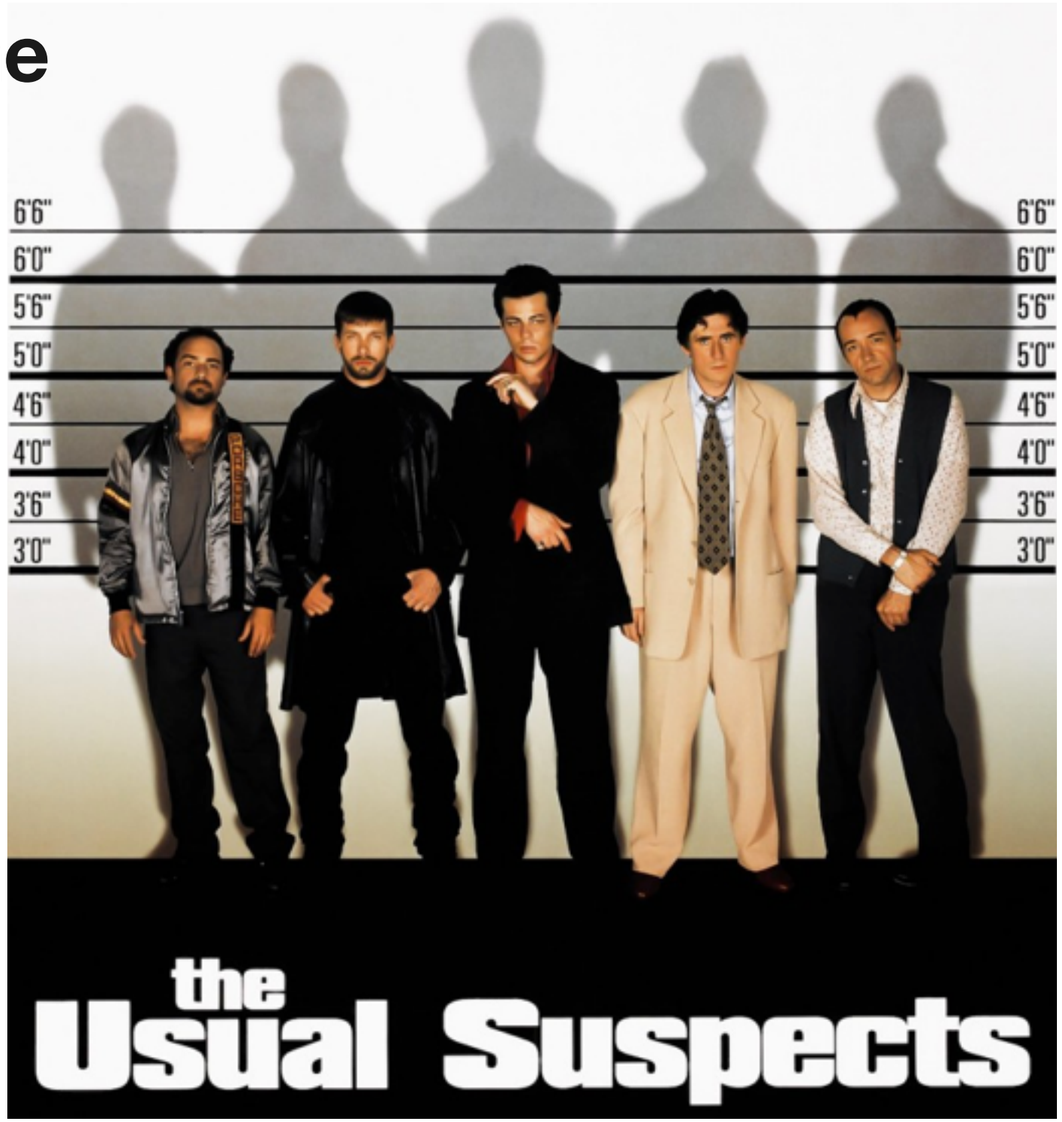


Optimize for the processor pipeline

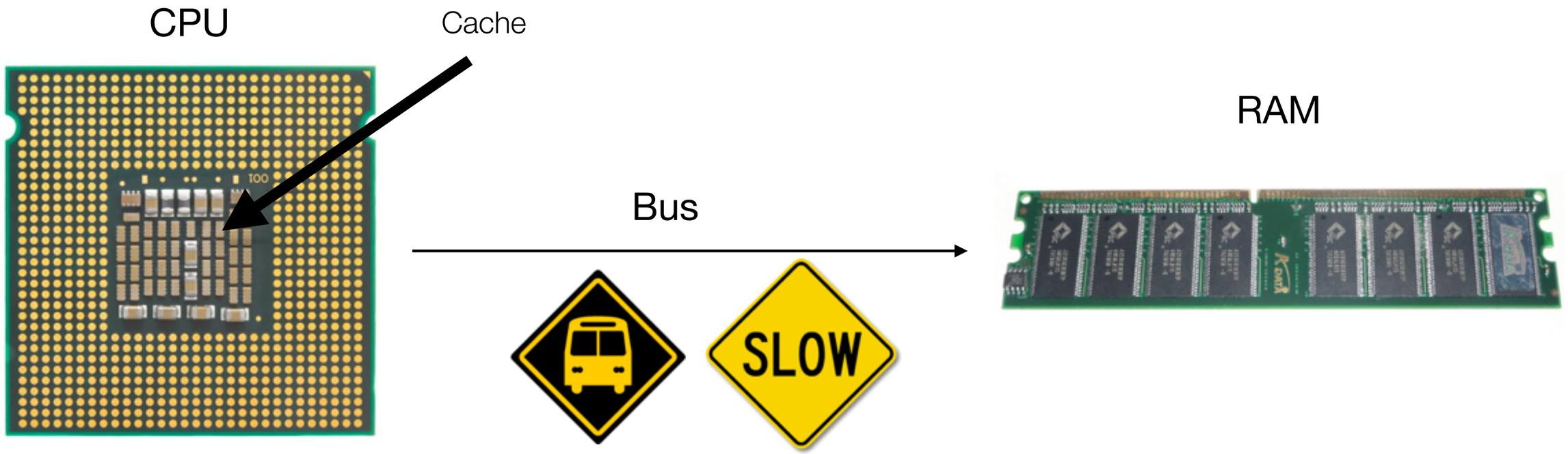
“Bubbles” can be caused by:

		virtual	data
ifs	loops	calls	dependency

cost ~ 12 cycles



Minimize CPU cache misses



a cache miss costs 10 to 100s cycles depending on the level

Encodings in Apache Parquet 2.0

The right encoding for the right job

- Delta encodings:

for sorted datasets or signals where the variation is less important than the absolute value. (timestamp, auto-generated ids, metrics, …) Focuses on avoiding branching.

- Prefix coding (delta encoding for strings)

When dictionary encoding does not work.

- Dictionary encoding:

small (60K) set of values (server IP, experiment id, …)

- Run Length Encoding:

repetitive data.

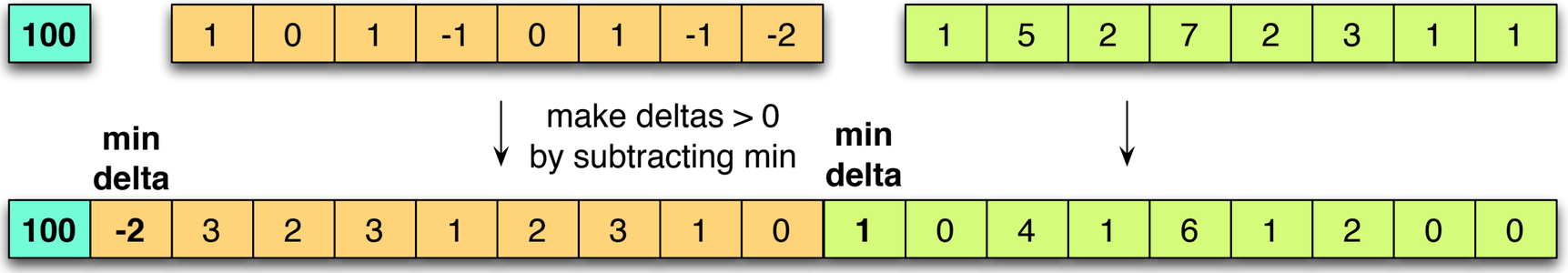
Delta encoding

values:	100	101	101	102	101	101	102	101	99	100	105	107	114	116	119	120	121

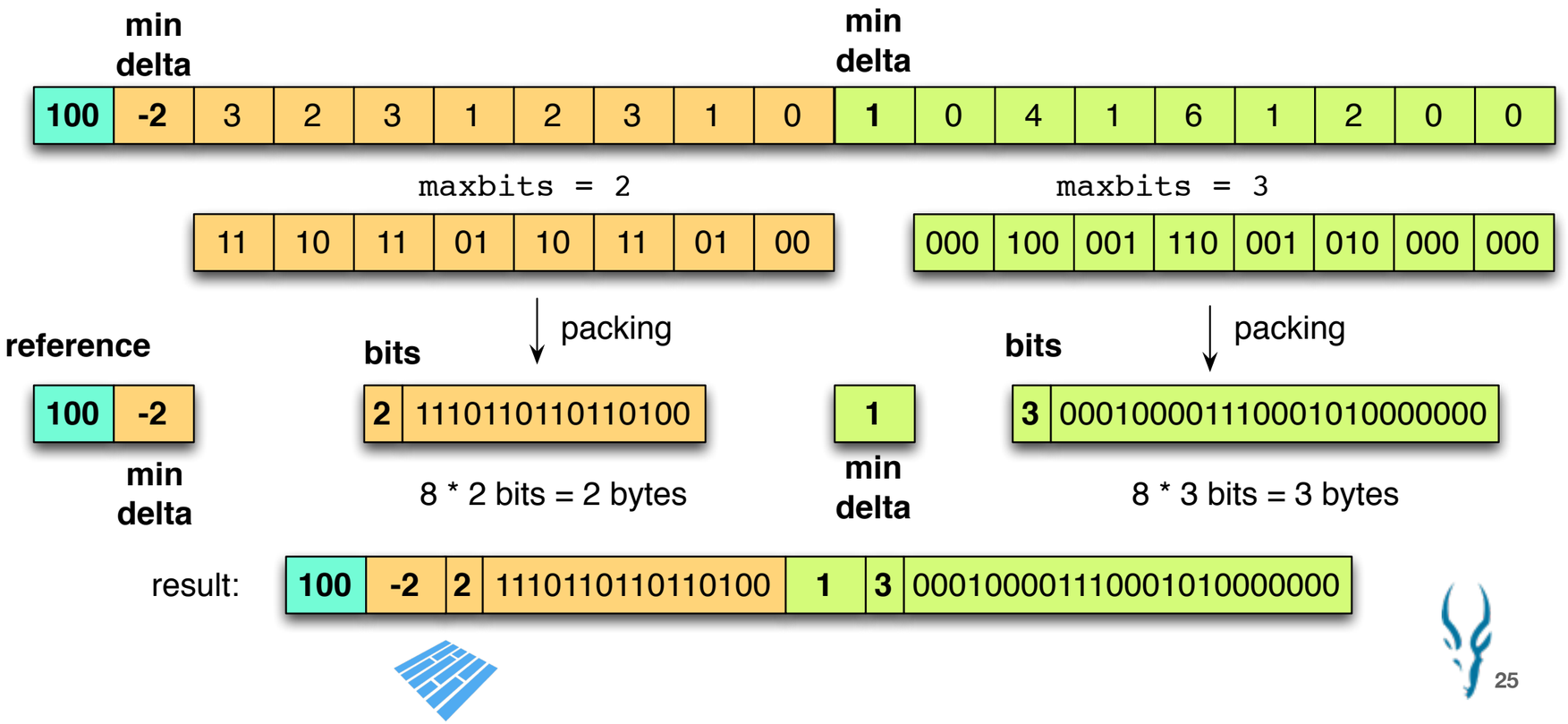
reference	block 1								block 2							
100	101	101	102	101	101	102	101	99	100	105	107	114	116	119	120	121
	8 * 64bits values = 64 bytes								8 * 64bits values = 64 bytes							
100	1	0	1	-1	0	1	-1	-2	1	5	2	7	2	3	1	1

deltas ↓

Delta encoding



Delta encoding



Delta encoding

values: 100 101 101 102 101 101 102 101 99 100 105 107 114 116 119 120 121

reference: 100

block 1: 101 101 102 101 101 102 101 99

8 * 64bits values = 64 bytes

block 2: 100 105 107 114 116 119 120 121

8 * 64bits values = 64 bytes

deltas

100 | 1 0 1 -1 0 1 -1 -2 | 1 5 2 7 2 3 1 1

make deltas > 0 by subtracting min

min delta: -2 | min delta: 1

100 | -2 3 2 3 1 2 3 1 0 | 1 0 4 1 6 1 2 0 0

maxbits = 2

11 10 11 01 10 11 01 00

maxbits = 3

000 100 001 110 001 010 000 000

packing

reference: 100 | -2 (min delta)

bits: 2 | 1110110110110100

8 * 2 bits = 2 bytes

packing

1 (min delta)

bits: 3 | 000100001110001010000000

8 * 3 bits = 3 bytes

result: 100 -2 2 1110110110110100 1 3 000100001110001010000000

Binary packing designed for CPU efficiency

naive maxbit:

```
max = 0
for (int i = 0; i<values.length; ++i) {
  current = maxbit(values[i])
  if (current > max) max = current
}
```

Unpredictable branch!

better:

```
orvalues = 0
for (int i = 0; i<values.length; ++i) {
  orvalues |= values[i]
}
max = maxbit(orvalues)
```

Loop => Very predictable branch

even better:

```
orvalues = 0
orvalues |= values[0]
…
orvalues |= values[32]
max = maxbit(orvalues)
```

no branching at all!

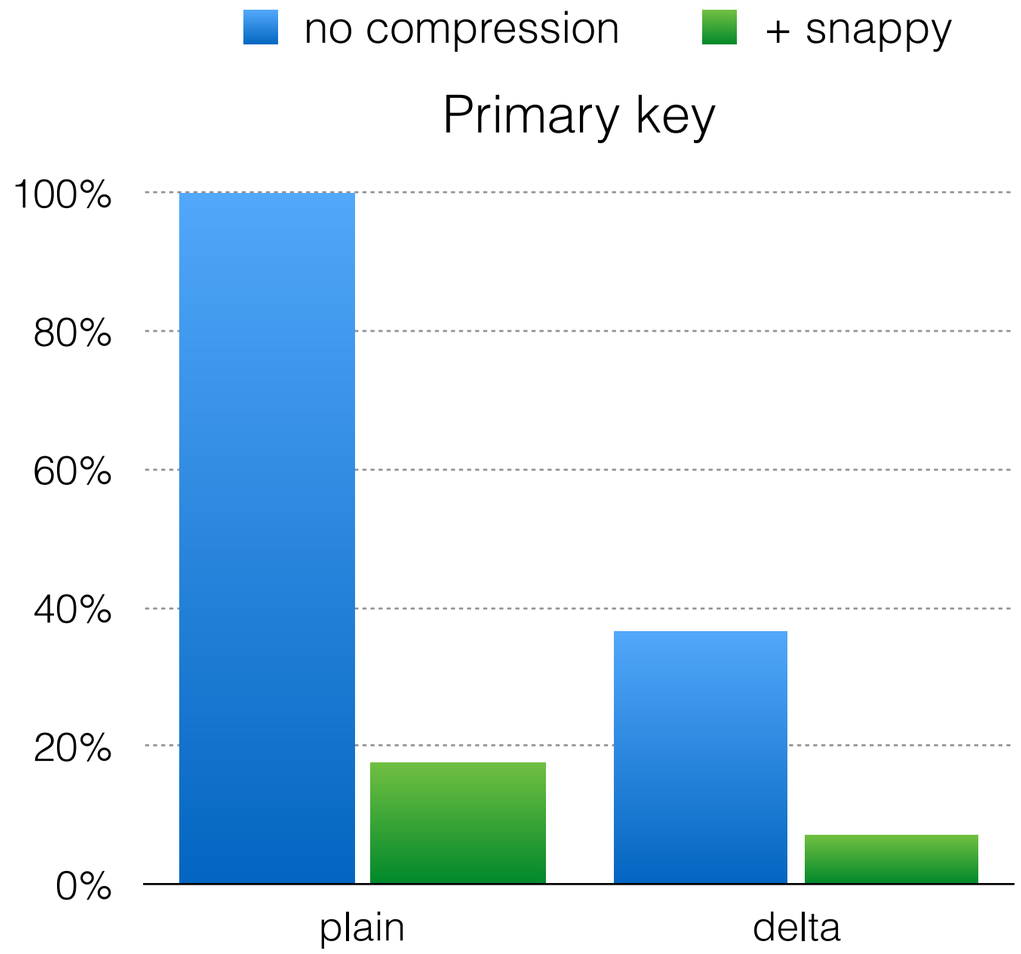
see paper:
“Decoding billions of integers per second through vectorization”
by Daniel Lemire and Leonid Boytsov

Binary unpacking designed for CPU efficiency

```
int j = 0
while (int i = 0; i < output.length; i += 32) {
  maxbit = input[j]
  unpack_32_values(values, i, out, j + 1, maxbit);
  j += 1 + maxbit
}
```

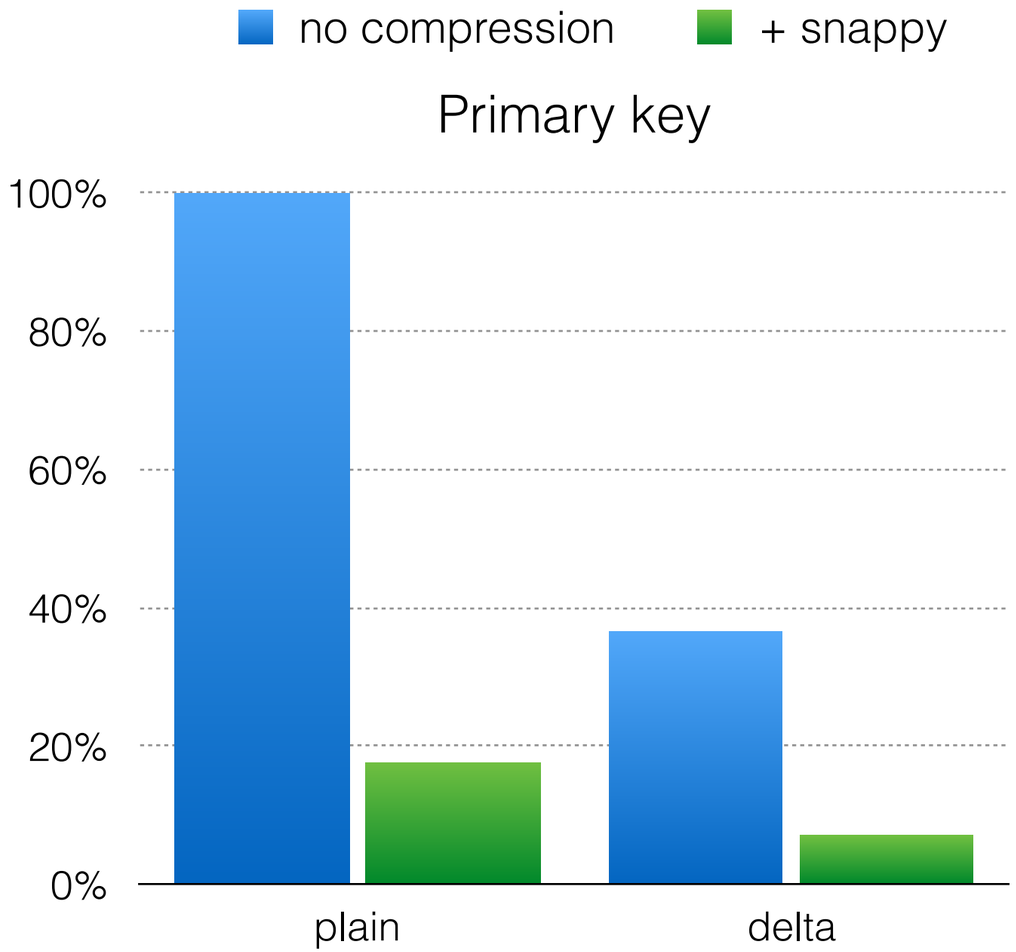
Compression comparison

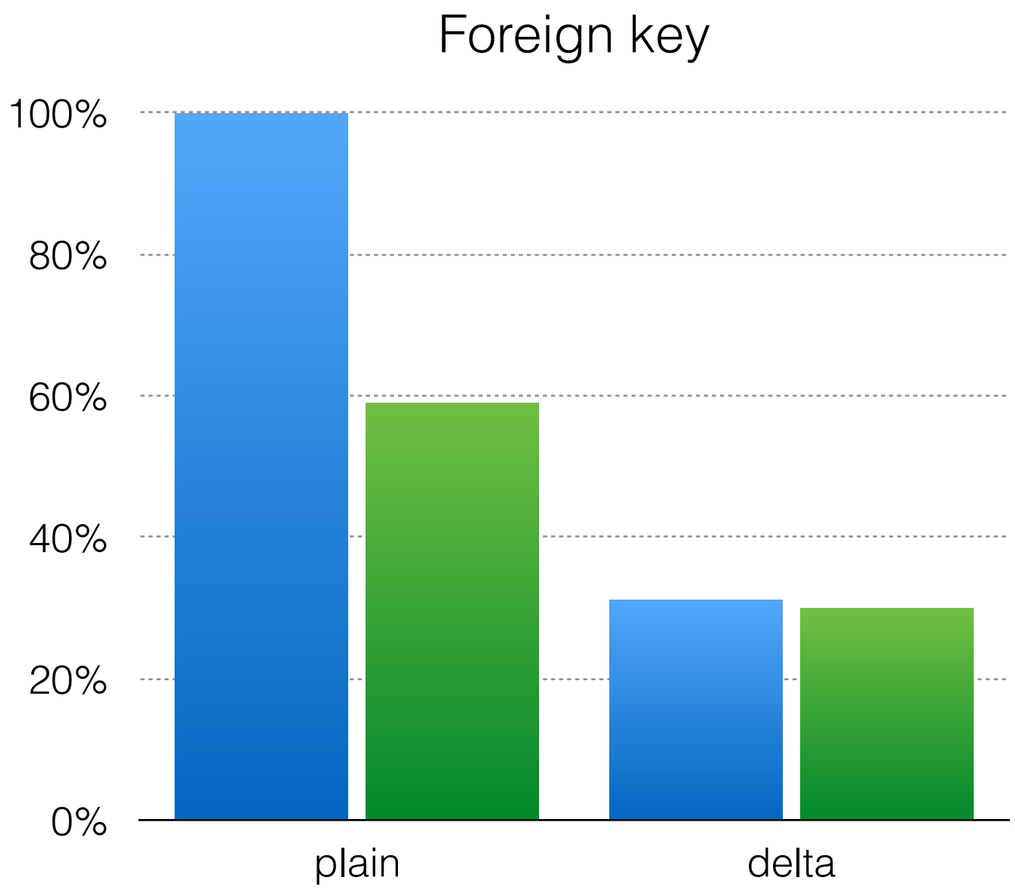
TPCH: compression of two 64 bits id columns with delta encoding



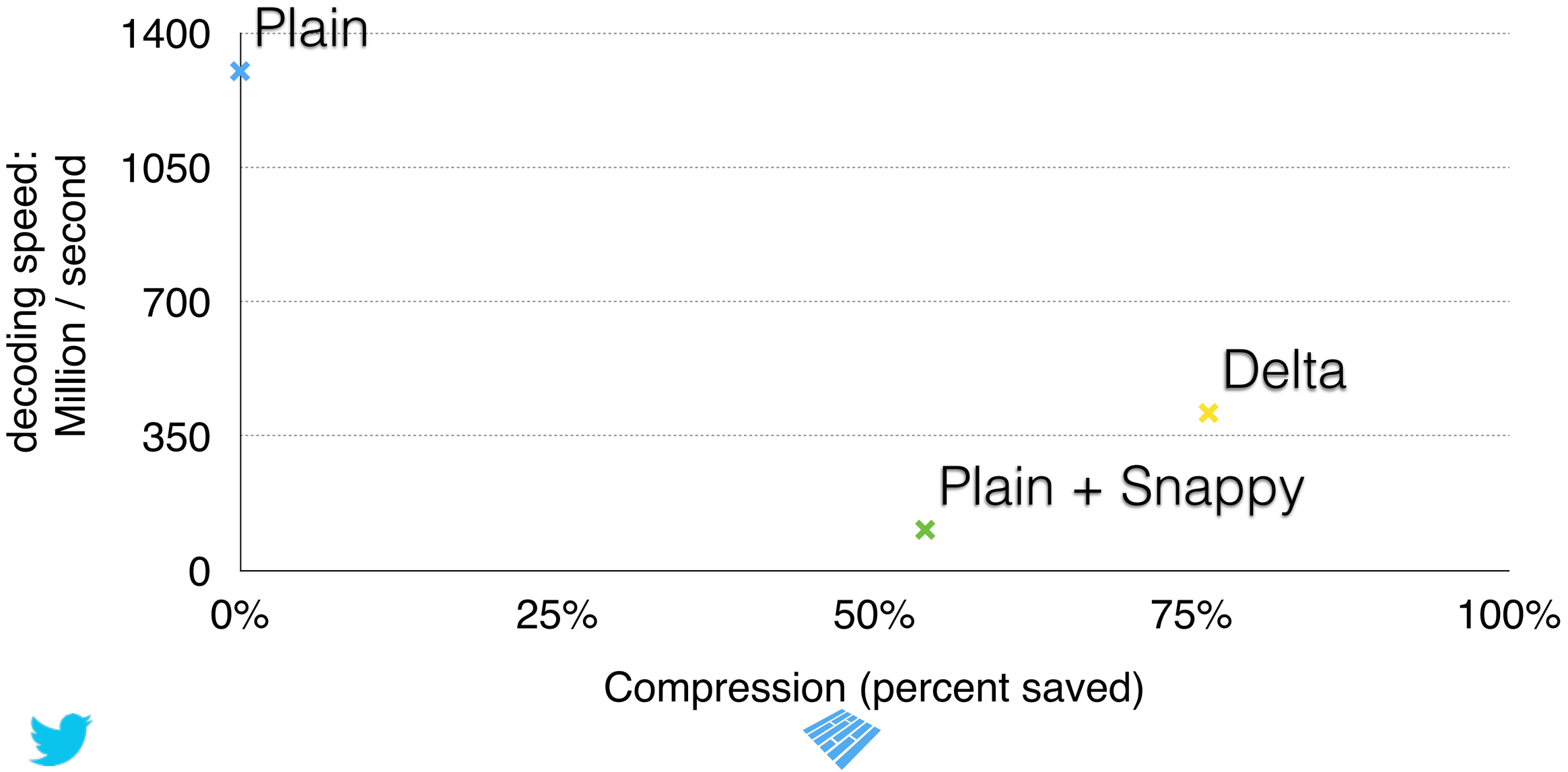
Compression comparison

TPCH: compression of two 64 bits id columns with delta encoding





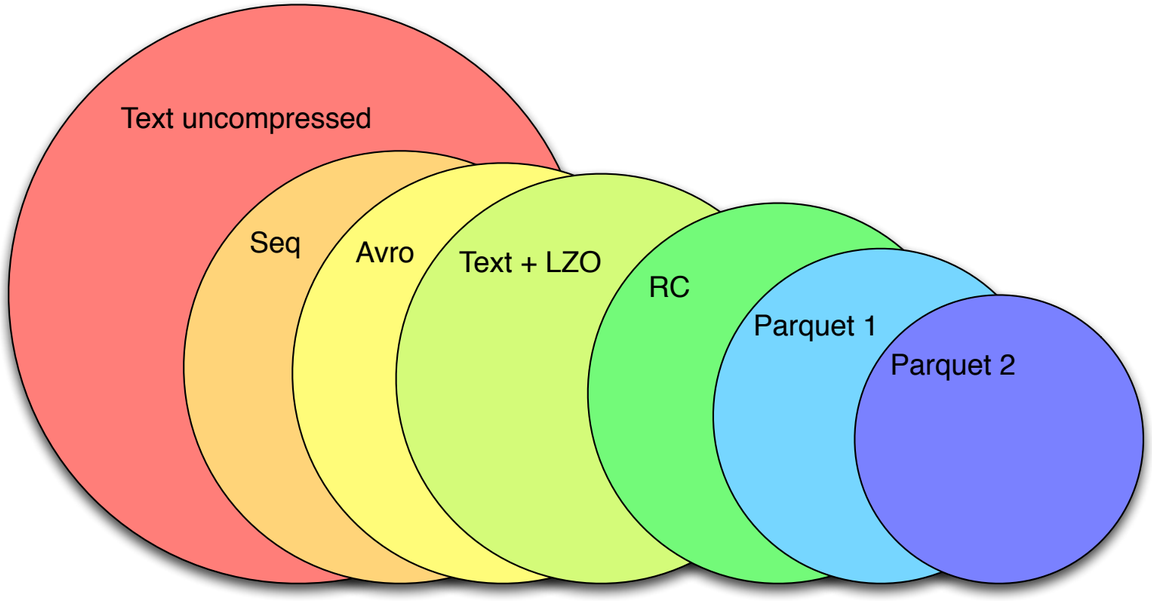
Decoding time vs Compression



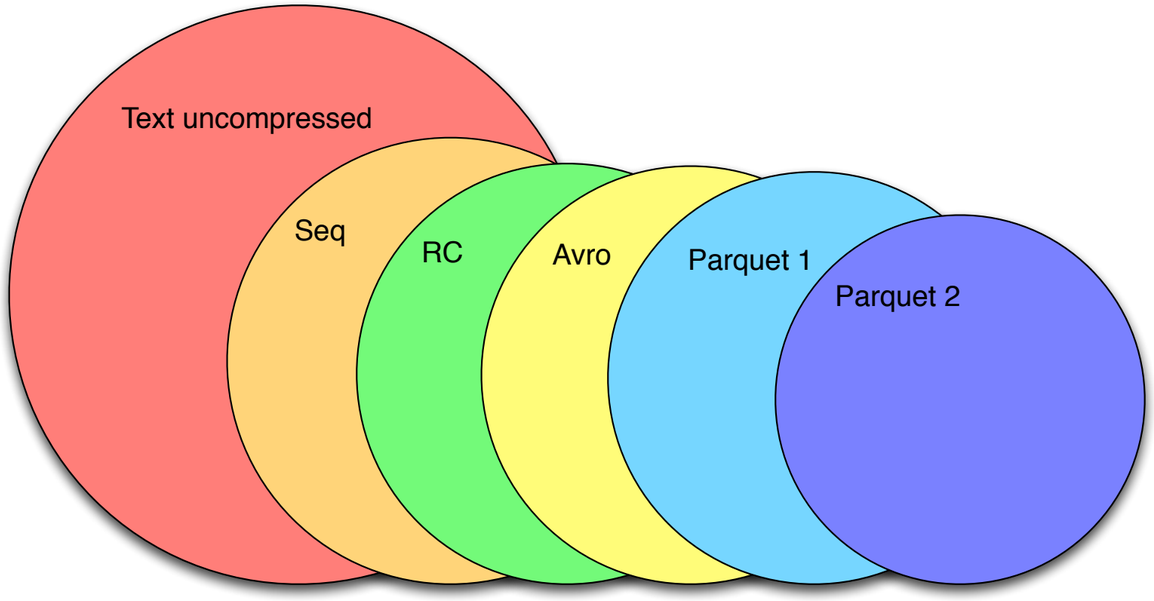
Performance

Size comparison

TPCDS 100GB scale factor (+ Snappy unless otherwise specified)



Lineitem

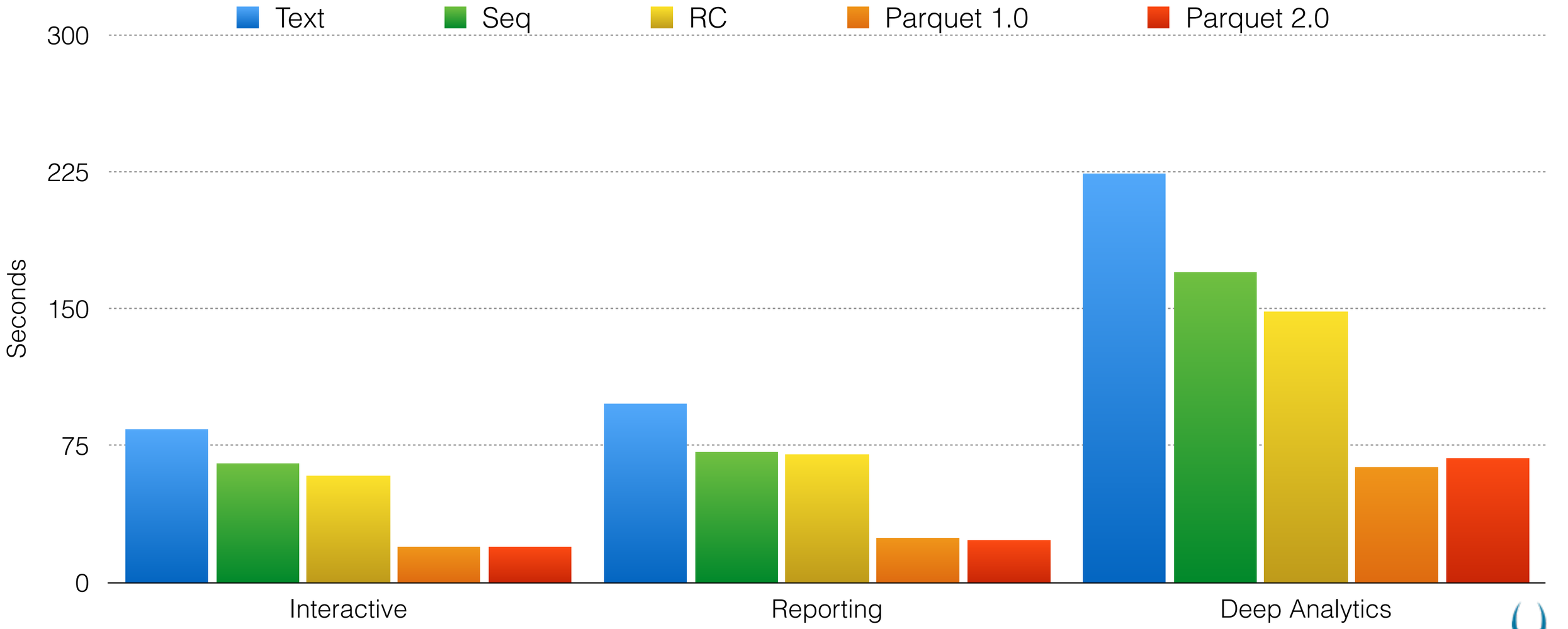


Store sales

The area of the circle is proportional to the file size

Impala query performance

10 machines:
8 cores
48 GB of RAM
12 Disks
OS buffer cache flushed between every query



TPCDS geometric mean per query category

Roadmap 2.x

Roadmap 2.x

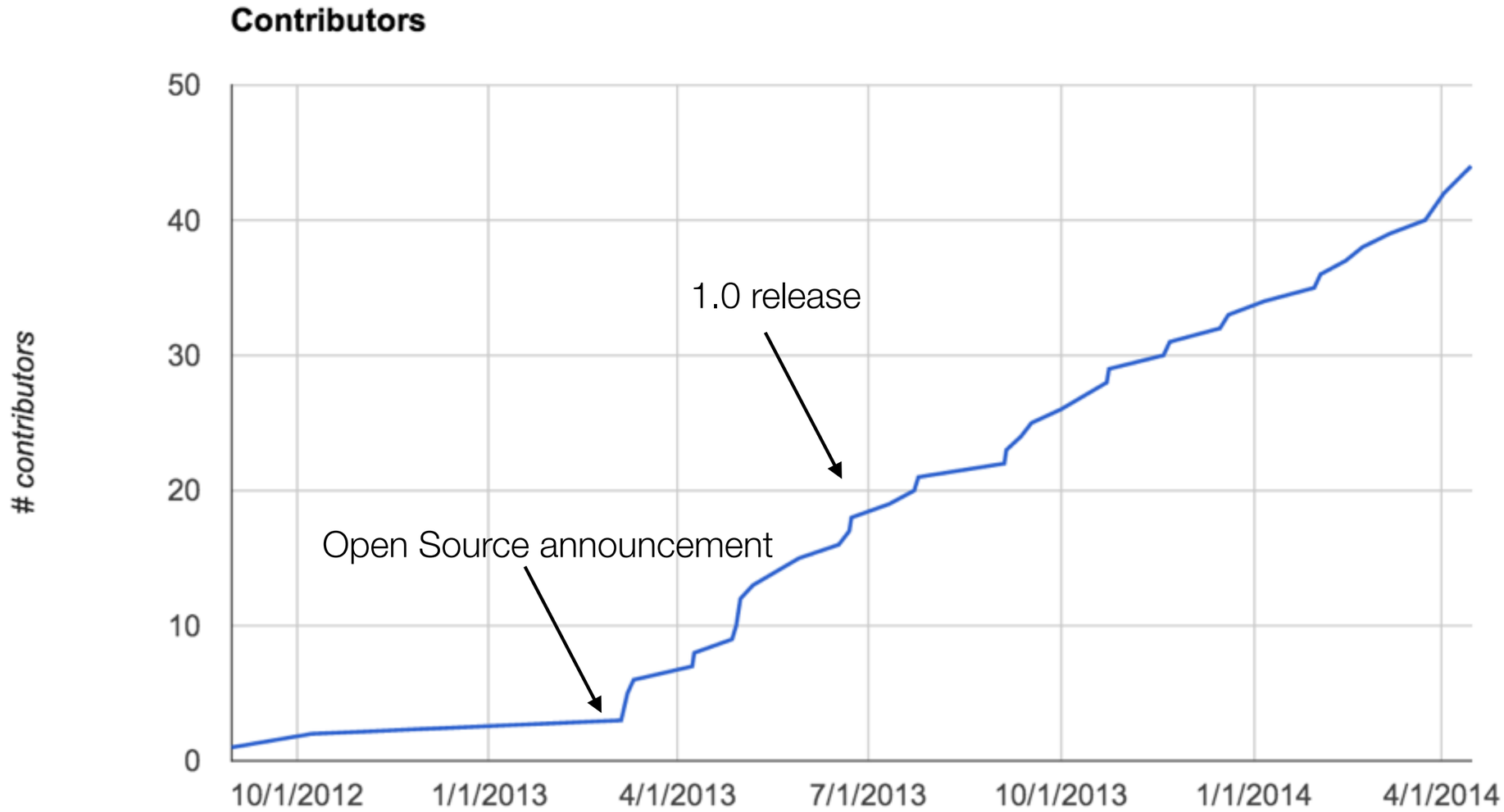
C++ library: implementation of encodings

Predicate push down:
  use statistics to implement filters at the metadata level

Decimal, Timestamp logical types

Community

Thank you to our contributors



Get involved

Mailing lists:

- dev@parquet.incubator.apache.org

Parquet sync ups:

- Regular meetings on google hangout

Questions

@ApacheParquet

Questions.foreach(answer(_))