Improving the Presto Parquet Reader

Venki Korukanti, Interactive Analytics Team

Uber

Uber's mission is to **ignite opportunity** by setting the world in **motion.**

600+
Cities

75M
Monthly Riders

15M
Trips/Day

Data informs **every decision** at the company

Marketplace Pricing

Growth Marketing

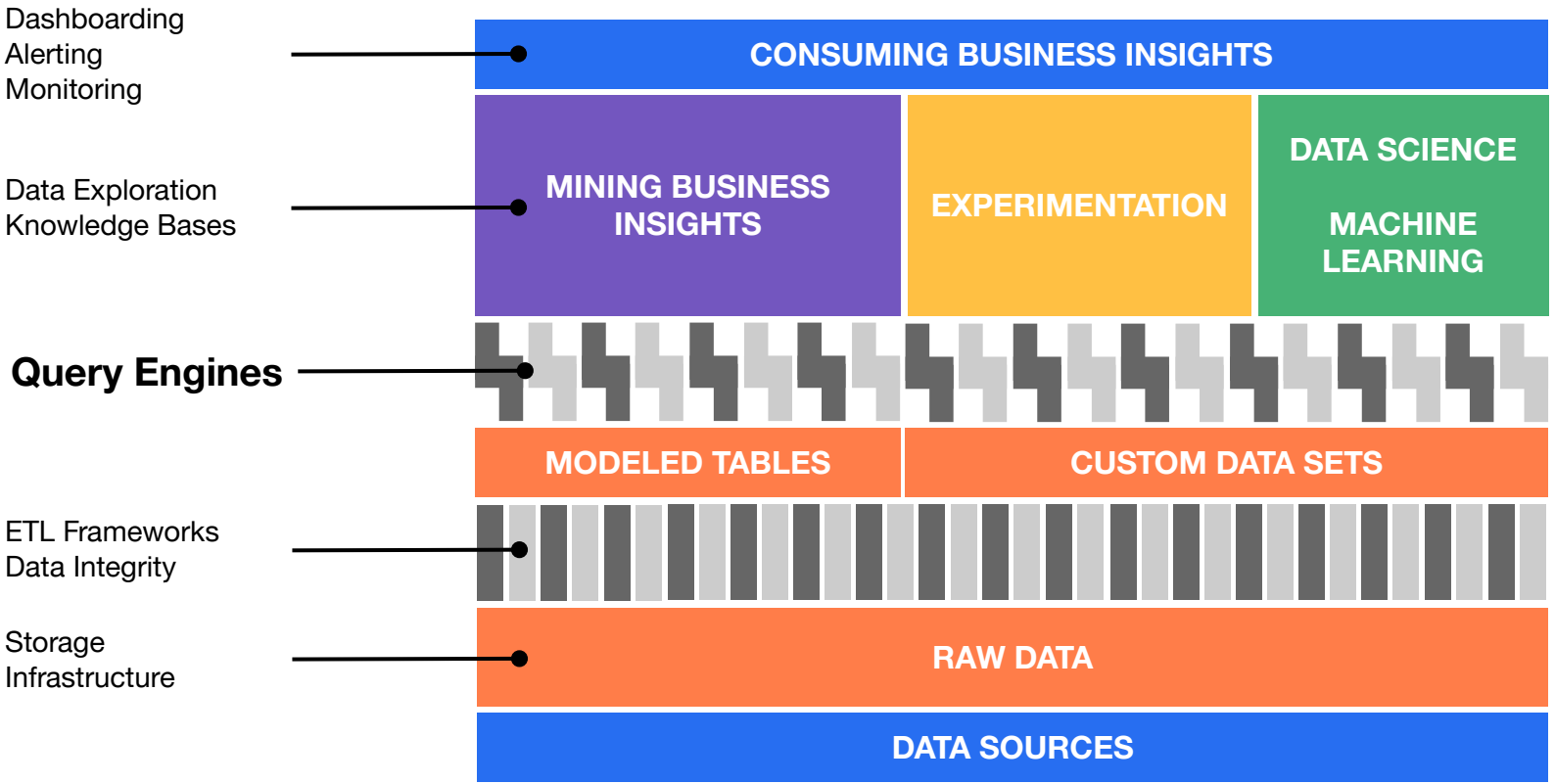
Compliance

Community Operations

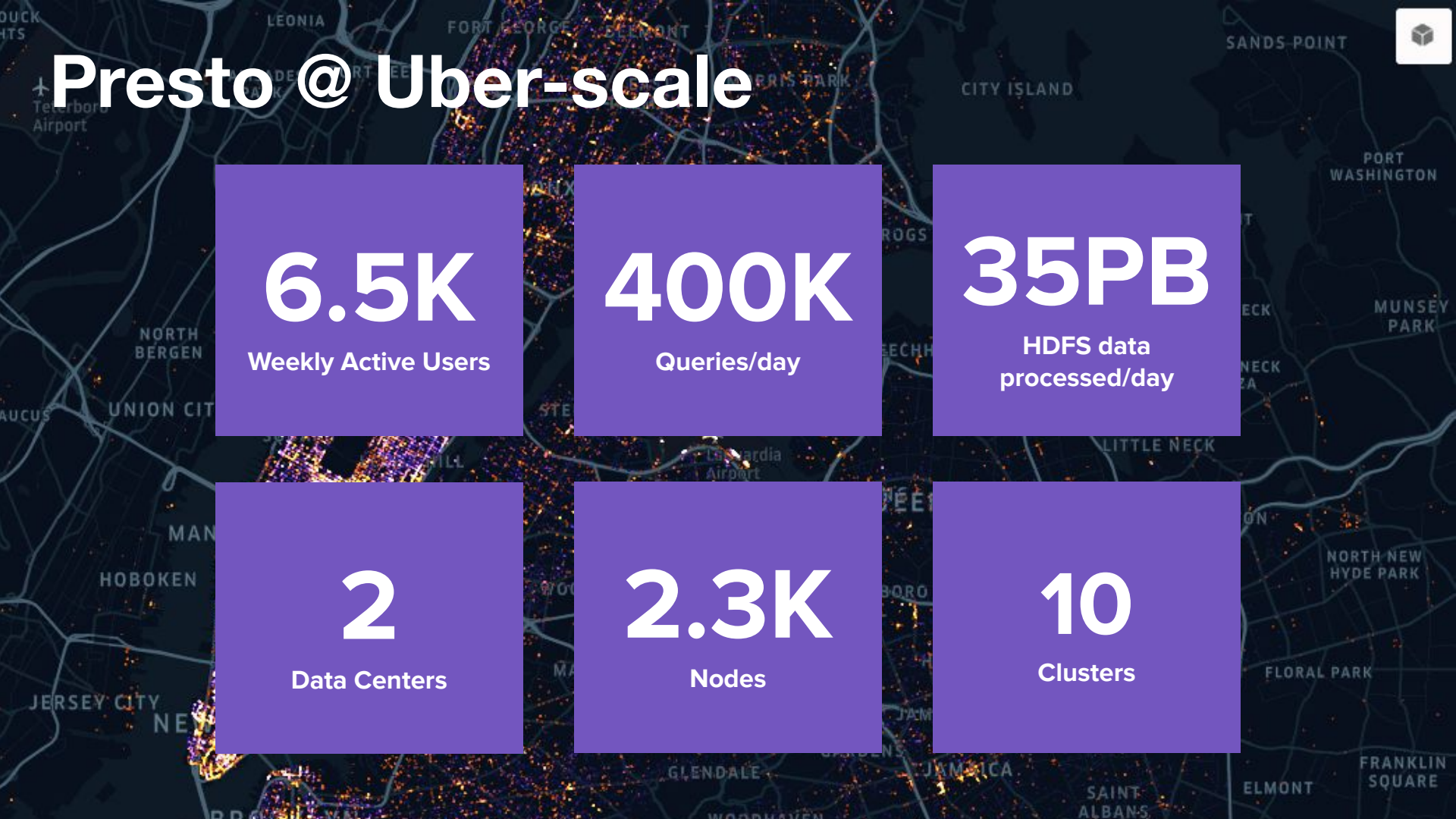
Data Science

Ad-hoc Querying

Overview of Uber's Data Platform



Presto @ Uber-scale



Presto Usage Growth

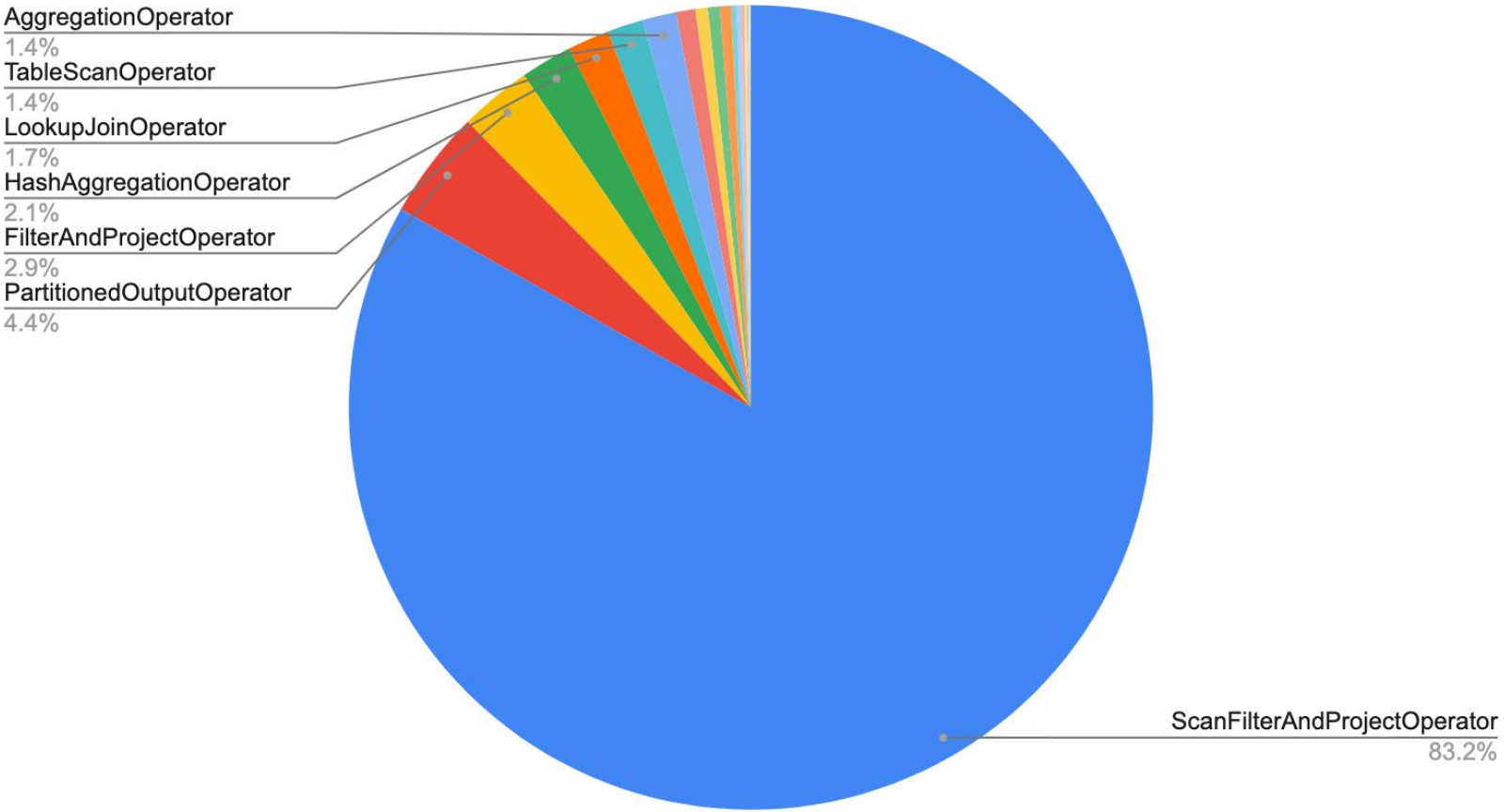
In last 6 months

- Weekly active users up by 25%
- Weekly queries up by 150%
- Weekly data read up by 70%
- Query latency P90 remained the same

New capacity addition every few months

Queries are constrained by CPU resources

Query CPU Time by Operator

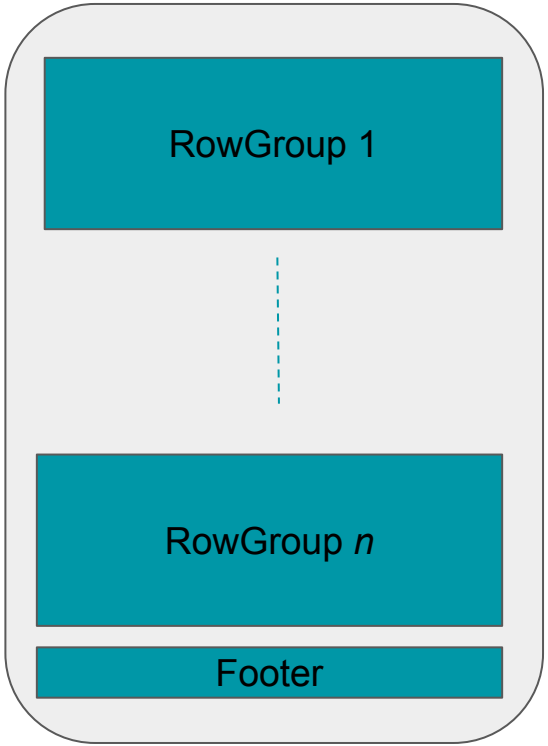


Efforts to Reduce the Scan-Filter-Project CPU Time

- Better data modelling
- Push Filter completely into reader
- Even finer filter pruning using stats
- Improve reader decoding

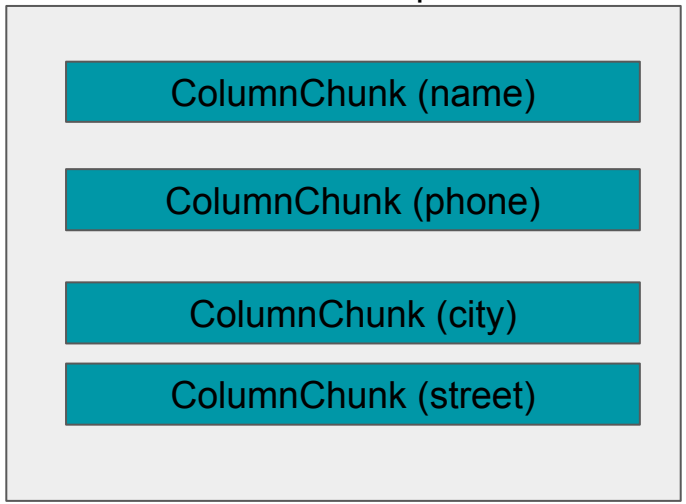
Parquet Reader Improvements

Parquet Format Overview



File

RowGroup

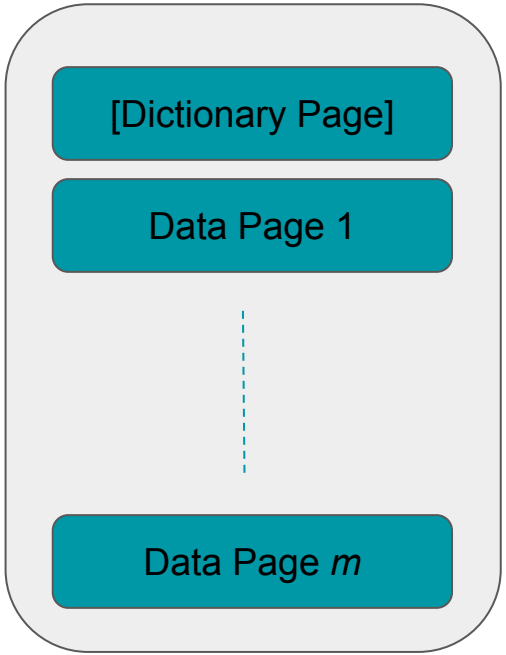


Schema:

1. name string,
2. phone array<string>,
3. address struct(city string, street string)

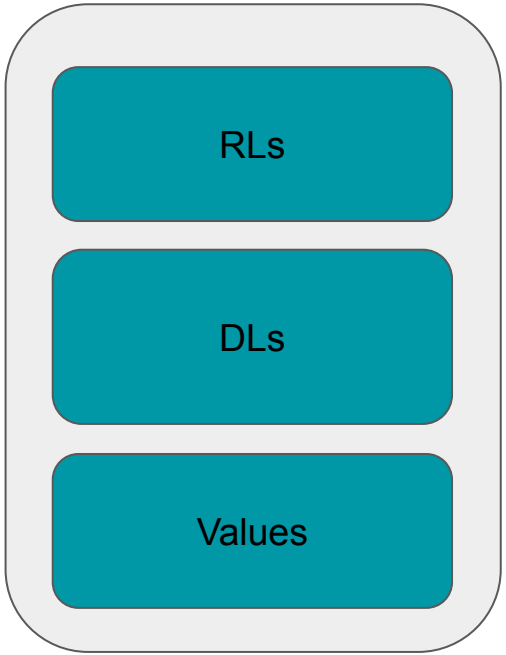
ColumnChunk

- Consist of one or more pages
- Page types
  - Dictionary
    - Optional and depends on the data
  - Data page
    - dictionary page exists → dictionary ids
    - no dictionary page → contains the actual values



Data Page

- Repetition Levels (RLs)
  - Encodes list starting
  - Values: ([1, 2], [3, 4]) written as (1, 2, 3, 4)
  - RLs: [0, 1, 0, 1]
  - More details here
- Definition Levels (DLs)
  - Nullability. [1, null, 2] → DLs: [1, 0, 1]
- Values: Only contains the non-null values
- Encoding Types
  - RLE/BitPacked/Plain/Delta encoding



Page

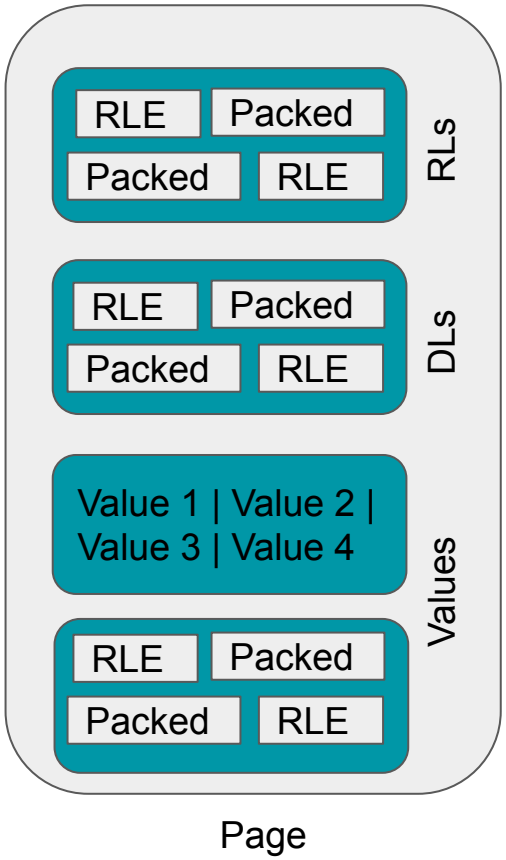
RLE/Bit Packed Encoding

RLE (Run Length Encoding)

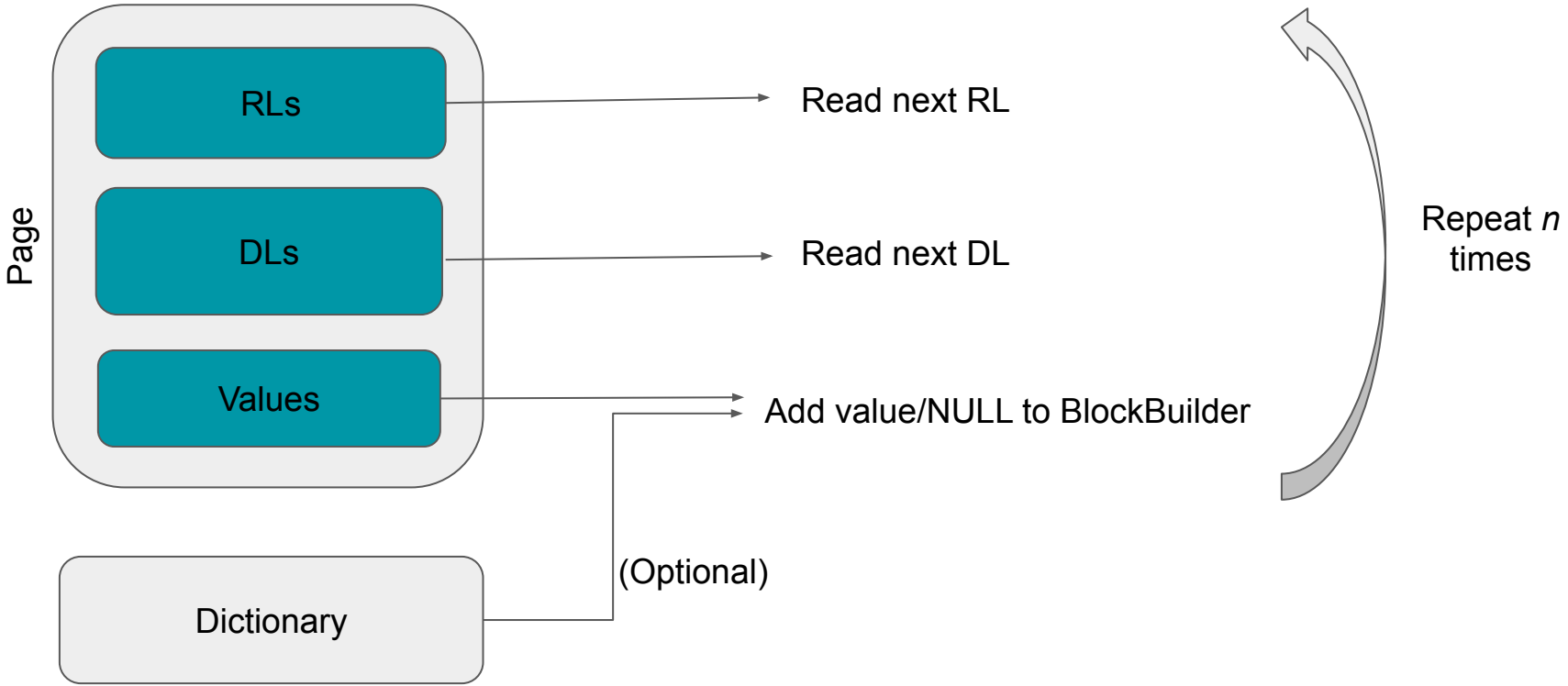
1. Consecutively repeating values
2. Example: CA, CA, CA, CA → (4, CA)

Bit Packed

1. Max value bitwidth < 8, 16, 24 or 48 bits
2. Encode the bits back to back
3. To encode: 2, 3, 5, 7, 2:
   a. Write bit width (3) in first byte
   b. Followed by values: <010 011 10> | <1 111 010 0>
   c. Encoded in three bytes, rather than five



Current Column Reader



Improvements

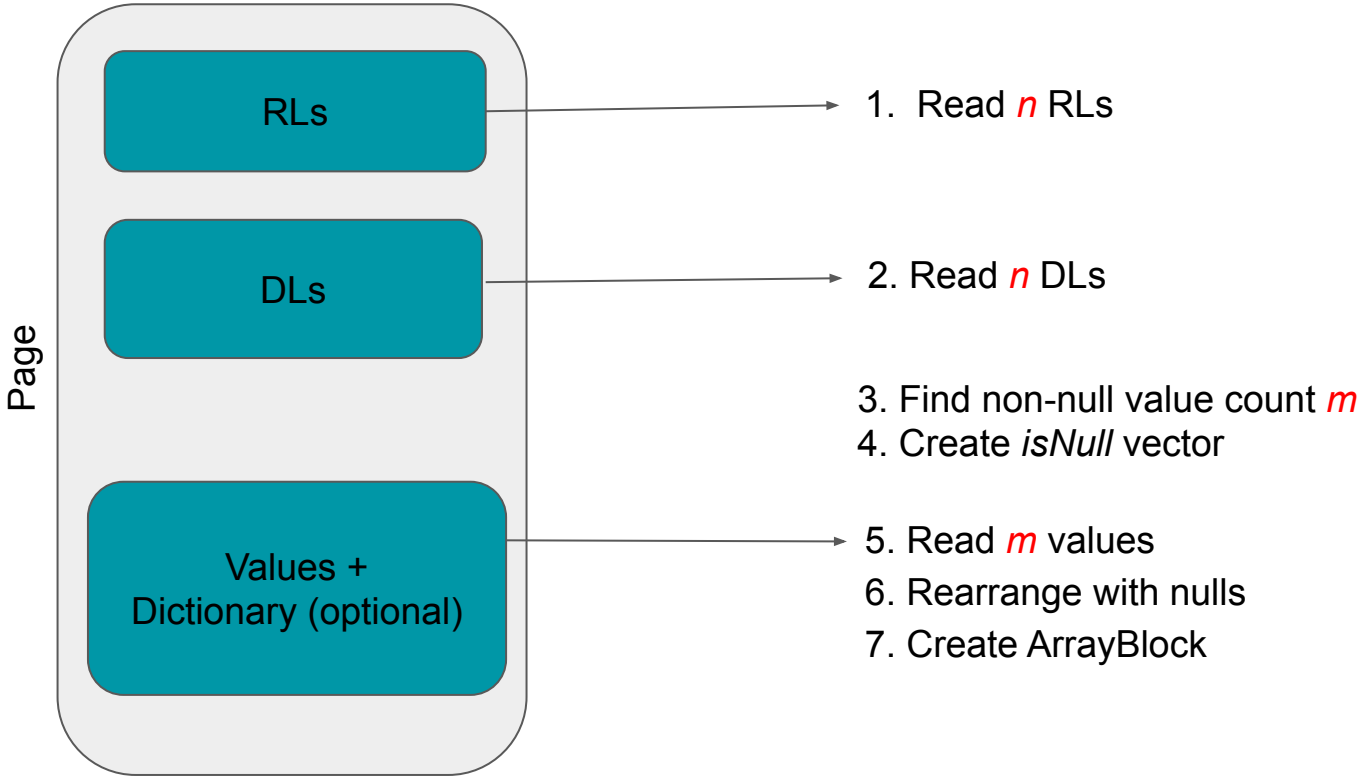
- Batch reads from Decoders
  - One end-of-stream check
  - One status update
  - Decoder state can be kept in registers

- ArrayBlock implementations instead of BlockBuilder
  - LongArrayBlock/IntegerArrayBlock/ByteArrayBlock/VariableWidthArrayBlock
  - ArrayBlocks take an array of values and array of *isNull* flags
  - Avoids function call to BlockBuilder

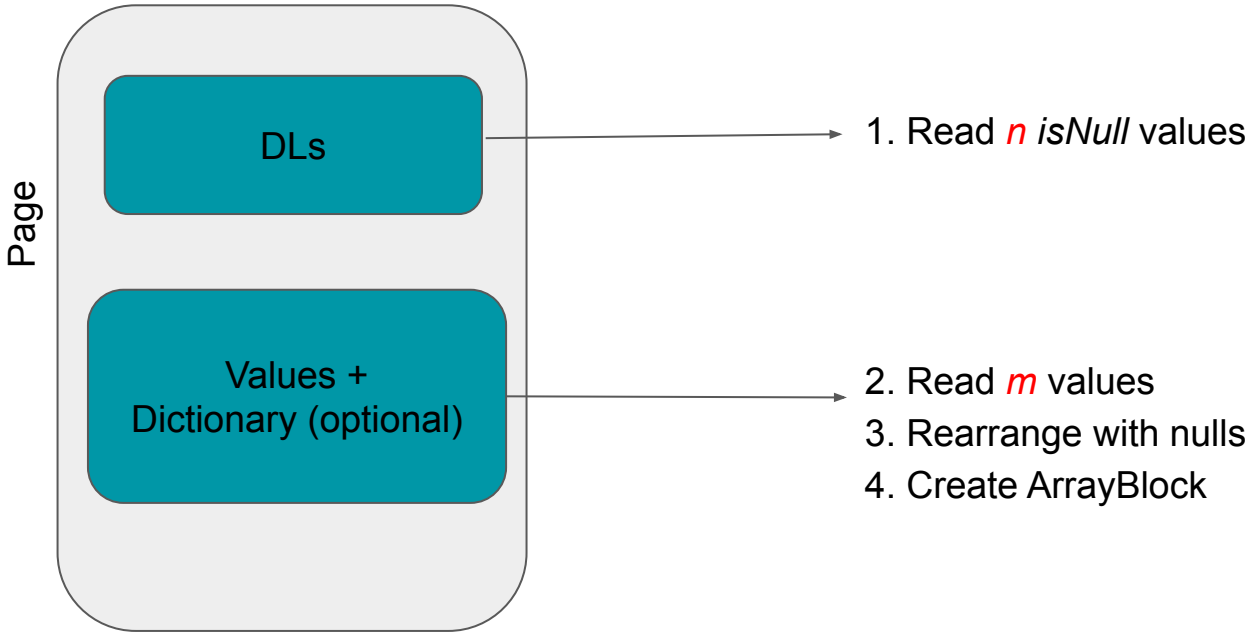
Improvements [2]

- RLE Block, Dictionary decoding
  - If the RLE block says 300 values of dictionary id 27 → 300 dictionary lookups
  - New Decoder that contains both the dictionary and values decoder
    - lookup only once for the RLE block
- Avoid reading RL or DL if not needed
  - non-Nested columns don’t need RL
  - non-Nested and non-nullable columns don’t need to read DL
- Use *System.arraycopy* wherever possible
  - Avoid generating byte arrays and copying
- Skip values without decoding
  - Ex. Interested in reading from 100th value in a Page.
  - Update the current offset in page value buffer

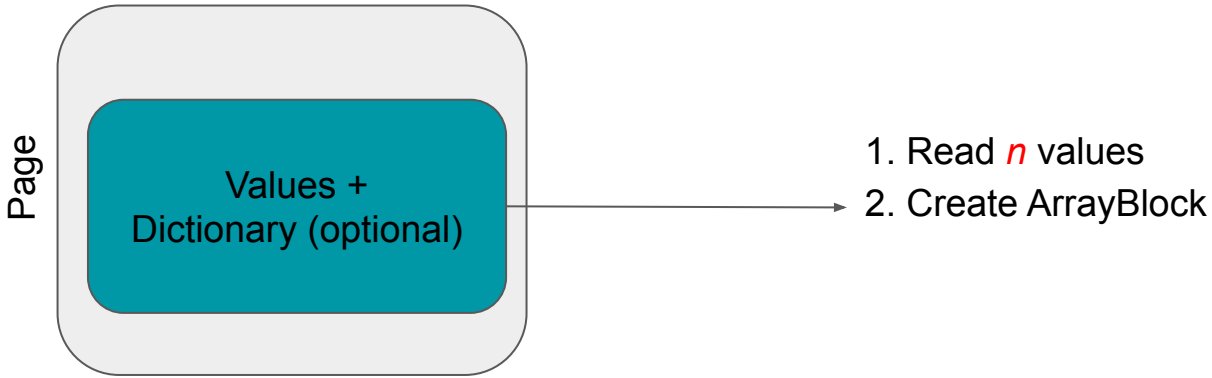
New Nested Column Reader



New non-Nested Nullable Column Reader



New non-Nested non-Nullable Column Reader



JMH Benchmarks

- Based on ORC benchmark tests
- 10m rows, 30 warmup iterations, 20 test iterations

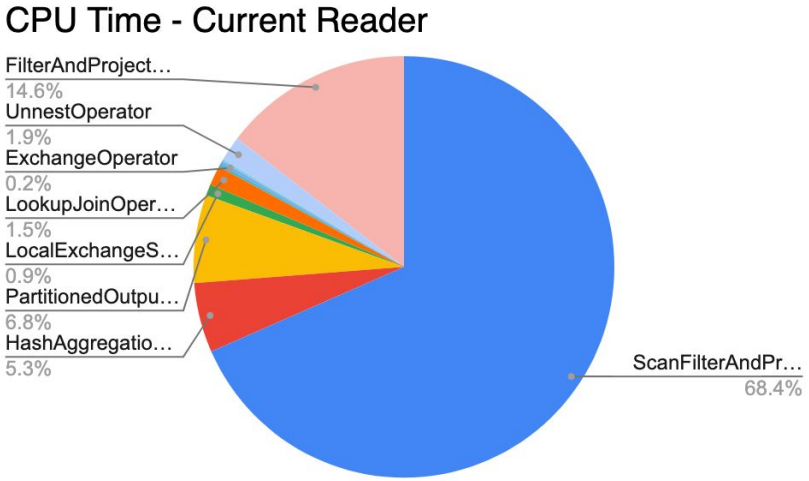
Type (non-nested)	Speed up
Boolean	4x
Float	3.5x
Integer	3.5x
Double	3x
Long	3x
VarChar	3.5x

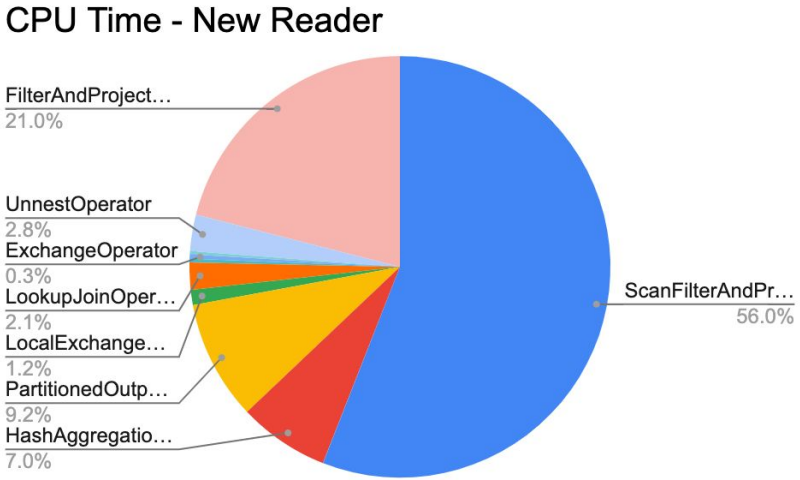
Type (inside Struct)	Speed up
Boolean	50%
Float	40%
Integer	40%
Double	45%
Long	45%
VarChar	47%

Type (List Type)	Speed up
Boolean	30%
Float	35%
Integer	35%
Double	30%
Long	30%
VarChar	40%

Results in Production

- Shadowed traffic of one of the dedicated customer cluster (50 nodes)
- ~25K queries per day
- ScanFilterAndProject CPU time decreased by **~40%**
- Total CPU time saved is **~28%**, Latency improved by **~15%**





Thank you

Parquet Format Overview - Dictionary

- The encoding is PLAIN
- For fixed length types, values are written one after the other
- For variable length types:
  - Length is written in first four bytes
  - Followed by the value

Current Reader’s method of decoding

ColumnReader (reads values in one ColumnChunk)

1. Initialize the BlockBuilder
2. (Optional) Initialize the dictionary page
3. Read the next page and initialize
    a. RL decoder
    b. DL decoder
    c. Values decoder
4. Read one value from RL and one value from DL
5. Based on the RL and DL values
    a. read value from values decoder (may involve dictionary lookup)
    b. write it to BlockBuilder

New Reader

- Two separate ColumnChunk readers for nested and non-nested columns
  a. Nested column reading adds complexity with RLs
  b. Restrict the RL decoding logic only to nested column readers
- In each reader separate path for
  a. Nullable column
  b. Non-nullable column (skips reading DLs)
- Values Decoders for each physical data type
  a. Read *n* values at time
  b. PLAIN encoding
  c. RLE/BitPacked encoding
- DL/RL decoders
  - Read *n* values at a time
  - RLE/BitPacked encoding