NYU | Center for Data Science

Week 06.3: Column storage

DS-GA 1004: Big Data

Lab 2 ...

Announcements

- Lab 3 (Spark) starts Thursday
- No class next week!

Previously...

- Spark and RDDs
- Delayed computation ⇒ optimization

This week

1. Column-oriented storage

2. Dremel & Parquet

vs.

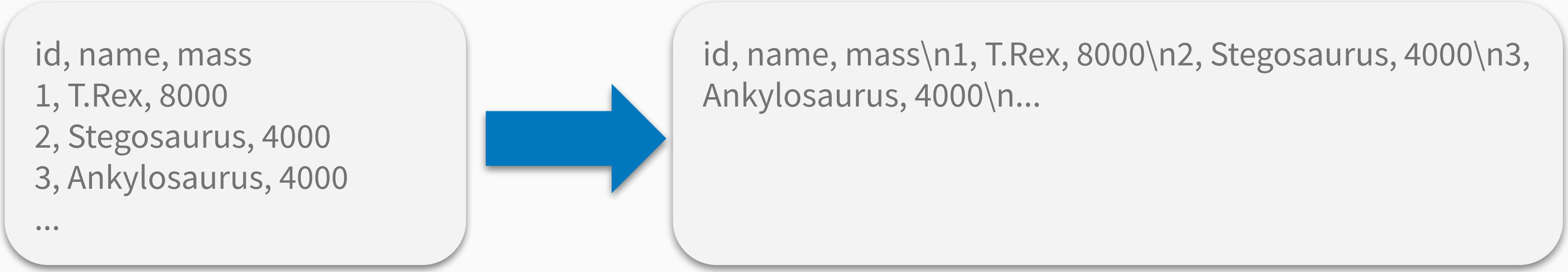
Row-oriented storage: CSV files?

- Imagine you have data stored as rows of text in the usual way

```
id, name, mass
1, T.Rex, 8000
2, Stegosaurus, 4000
3, Ankylosaurus, 4000
...
```

Row-oriented storage: CSV files?

- Imagine you have data stored as rows of text in the usual way

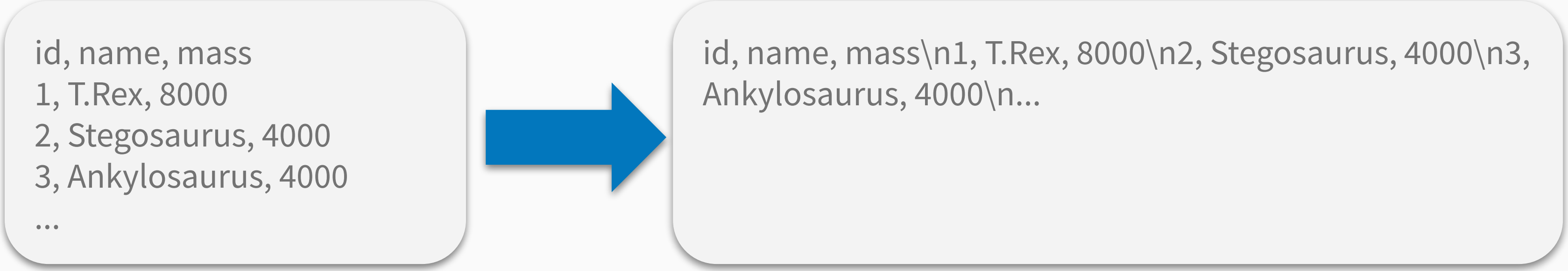


```
id, name, mass
1, T.Rex, 8000
2, Stegosaurus, 4000
3, Ankylosaurus, 4000
...
```

```
id, name, mass\n1, T.Rex, 8000\n2, Stegosaurus, 4000\n3,
Ankylosaurus, 4000\n...
```

Row-oriented storage: CSV files?

- Imagine you have data stored as rows of text in the usual way



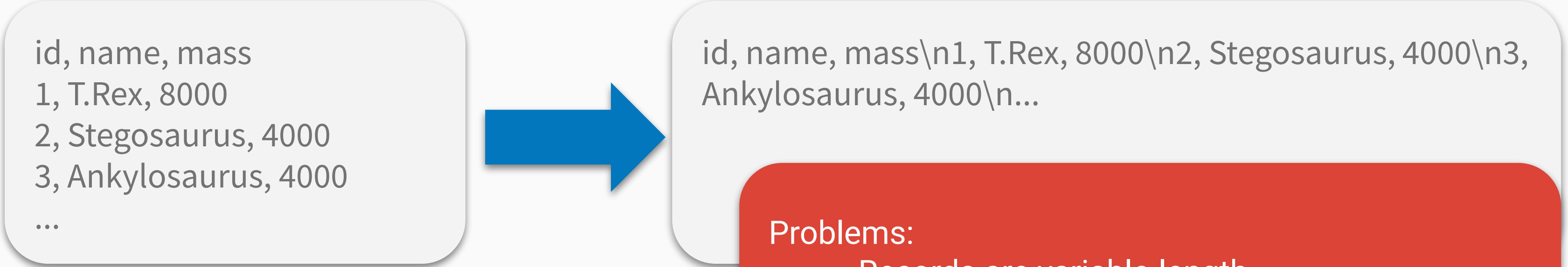
```
id, name, mass
1, T.Rex, 8000
2, Stegosaurus, 4000
3, Ankylosaurus, 4000
...
```

```
id, name, mass\n1, T.Rex, 8000\n2, Stegosaurus, 4000\n3,
Ankylosaurus, 4000\n...
```

- How would you access the 1000th record?
- How would you access just the third column?

Row-oriented storage: CSV files?

- Imagine you have data stored as rows of text in the usual way



id, name, mass
1, T.Rex, 8000
2, Stegosaurus, 4000
3, Ankylosaurus, 4000
...

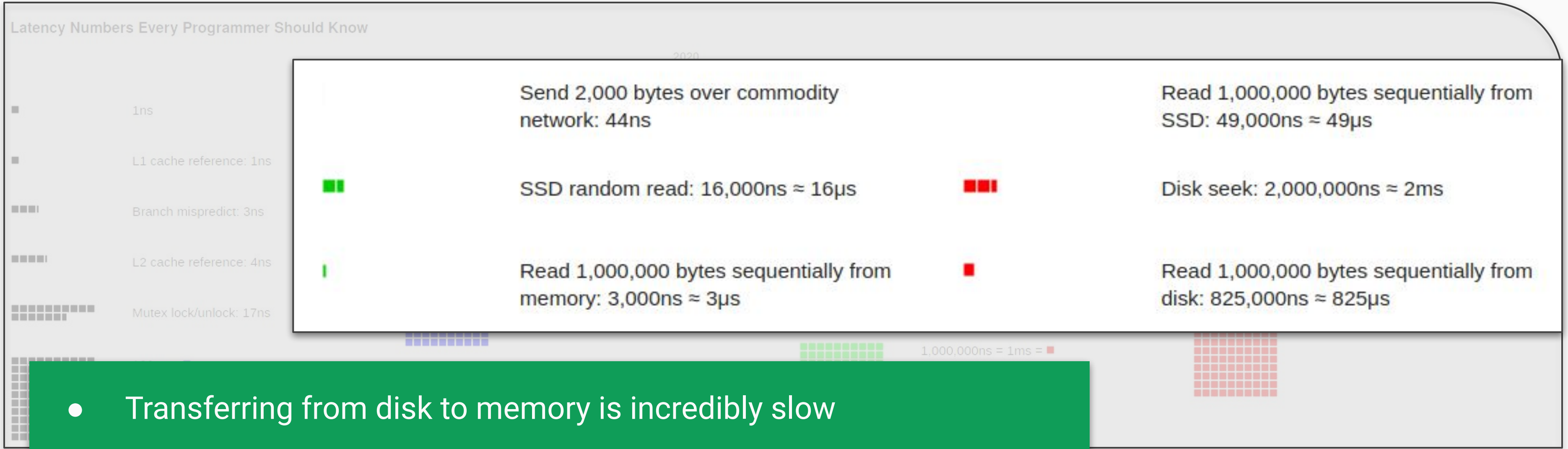
id, name, mass\n1, T.Rex, 8000\n2, Stegosaurus, 4000\n3, Ankylosaurus, 4000\n...

- How would you access the 1000th re

Problems:

- Records are variable-length
- Row and column offsets are hard to predict
- Basically requires full serial scan

- How would you access just the third column?



Latency Numbers Every Programmer Should Know

2020

1ns

L1 cache reference: 1ns

Branch mispredict: 3ns

L2 cache reference: 4ns

Mutex lock/unlock: 17ns

Send 2,000 bytes over commodity network: 44ns

SSD random read: 16,000ns ≈ 16µs

Read 1,000,000 bytes sequentially from memory: 3,000ns ≈ 3µs

Read 1,000,000 bytes sequentially from SSD: 49,000ns ≈ 49µs

Disk seek: 2,000,000ns ≈ 2ms

Read 1,000,000 bytes sequentially from disk: 825,000ns ≈ 825µs

1,000,000ns = 1ms = 🟥

- Transferring from disk to memory is incredibly slow
- Sequential memory reads are faster due to cache pre-fetching
- Strategies:
  - Transfer fewer bytes
  - Use predictable and contiguous memory access patterns

https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

Compression

id	Species	Era	Diet	Awesome	Mass
1	T. Rex	Cretaceous	Carnivore	True	8000
2	Stegosaurus	Jurassic	Herbivore	True	4000
3	Ankylosaurus	Cretaceous	Herbivore	False	4000

- **Records** have **heterogeneous** types
- A **single column** only has **one type**
- **Low entropy** in a column ⇒ **compression**
  - Compressed columns take less space
  - Compressed columns are **cheaper to load**
  - Sometimes we can **compute directly on compressed columns**!

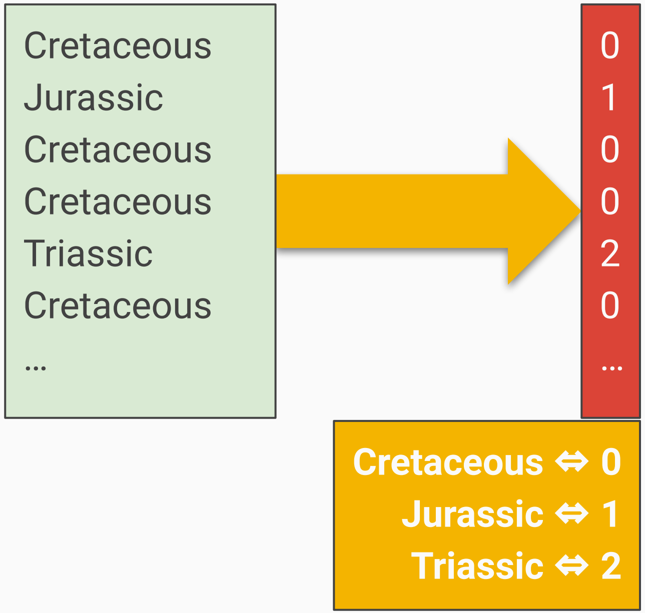
Dictionary encoding

id	Species	Era	Diet	Awesome	Mass
1	T. Rex	Cretaceous	Carnivore	True	8000
2	Stegosaurus	Jurassic	Herbivore	True	4000
3	Ankylosaurus	Cretaceous	Herbivore	False	4000

- Useful when you have an attribute which takes **few distinct values**

- Replace **string values** by **string identifiers**

- Column now has **uniform data width**
  ⇒ better cache locality!



Cretaceous
Jurassic
Cretaceous
Cretaceous
Triassic
Cretaceous
…

0
1
0
0
2
0
…

Cretaceous ⇔ 0
Jurassic ⇔ 1
Triassic ⇔ 2

slido

When using dictionary coding, do we need to decompress the data to do partial matching?ex: "SELECT * FROM Table WHERE name LIKE 'Sue%'"

ⓘ Start presenting to display the poll results on this slide.

slido

[2] When using dictionary coding, do we need to decompress the data to do partial matching?ex: "SELECT * FROM Table WHERE name LIKE 'Sue%'"

ⓘ Start presenting to display the poll results on this slide.

Dictionary coding + partial matching

- We don’t need to decompress!
- Do the partial match on the dictionary first
    - ⇒ find all (if any) matching indices
- Then search the compressed table for the **matching indices**

Bit-packing

- Integers usually consume 4, or 8 bytes (32 or 64 bits)

- **Bit-packing** squeezes **small integers** together

Values	0	1	0	2	1	1
8-bit (binary)	`0000 0000`	`0000 0001`	`0000 0000`	`0000 0010`	`0000 0001`	`0000 0001`
Compressed	`0001 0010`	`0101 ....`				

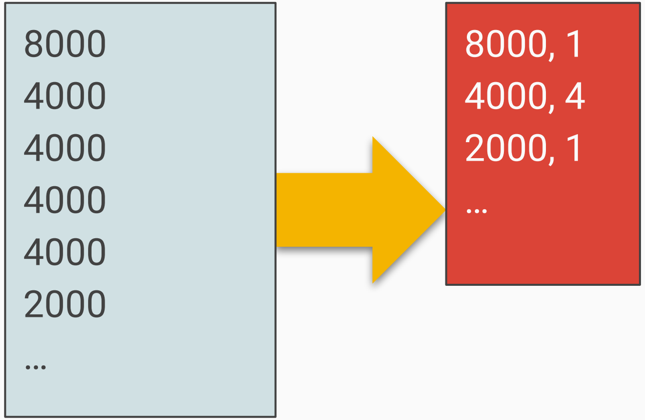
- Matching and comparing can be done on **compressed values**

Run-length encoding

id	Species	Era	Diet	Awesome	Mass
1	T. Rex	Cretaceous	Carnivore	True	8000
2	Stegosaurus	Jurassic	Herbivore	True	4000
3	Ankylosaurus	Cretaceous	Herbivore	False	4000

- Useful when you have long runs of a constant value

- Convert **sequence of values** to tuples **(value, # repetitions)**



8000
4000
4000
4000
4000
2000
…

8000, 1
4000, 4
2000, 1
…

- Sums, averages, counts, etc can all be done on **compressed values**

Compression schemes abound...

- Frame of reference coding
  - 1004, 1005, 1006 ⇒ **1000** | 4, 5, 6
- Delta coding
  - 1004, 1005, 1006 ⇒ **1004** | +0, +1, +1
- Lempel-Ziv-Welch (LZW) compression

Compression schemes can be **combined**!
Delta + bit packing
Dictionary + Run-length encoding

Main trade-off is ***space efficiency*** vs. ***complexity of querying/processing***.

slido

Which of the following compression schemes are sensitive to the order of values within a column?

ⓘ Start presenting to display the poll results on this slide.

slido

[2] Which of the following compression schemes are sensitive to the order of values within a column?

ⓘ Start presenting to display the poll results on this slide.

Compression and ordering

- **Dictionary coding** doesn't depend on order
  - (Arguably, isn't even truly *compression*)
- **Bit-packing** depends only on the maximum value in a column
- **RLE**, **FoR**, **Delta** all depend critically on order!
  - Compare RLE on a,b,a,b,a,b,a,b vs. a,a,a,a,b,b,b,b
- **Take-away message: it can pay off to sort your data!**

Nested and structured data

- Not everything fits nicely in relations / tables / dataframes
- Variable-length and variable-depth data can be difficult to deal with
- Record-oriented storage is relatively straightforward

How can we get all the benefits of column stores but for structured data?

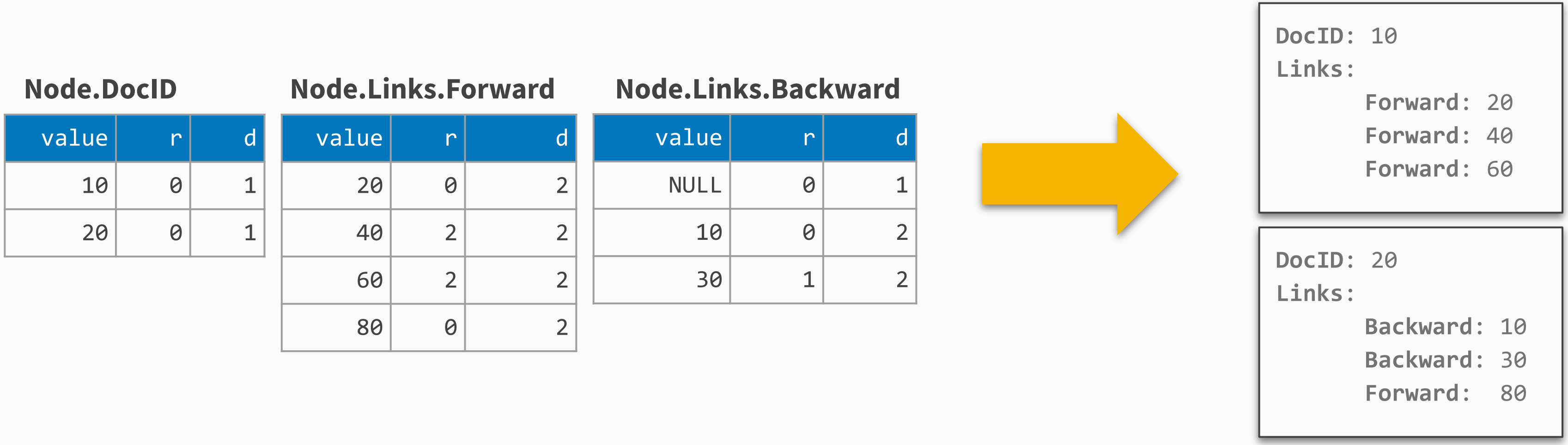
Record flattening

- **Key idea**: track repetitions of fields within a record
- ***Repetition level (r)***: which level repeated most recently?
- ***Definition level (d)***: how many optional fields in the path are present?
- **Required fields ⇒ Same levels as parent**
- **Optional fields ⇒ Same r-level as parent, d-level increments**
- **Repeated fields ⇒ r-level and d-level both increment from parent**

```
DocID: 10
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’
```

Partial record assembly

- Dremel can rebuild **partial views** (projections) of the data easily
- Unused attributes can be ignored!
- But decoding is **inherently sequential** ⇒ difficult to parallelize



Node.DocID

value	r	d
10	0	1
20	0	1

Node.Links.Forward

value	r	d
20	0	2
40	2	2
60	2	2
80	0	2

Node.Links.Backward

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

```
DocID: 10
Links:
        Forward: 20
        Forward: 40
        Forward: 60
```

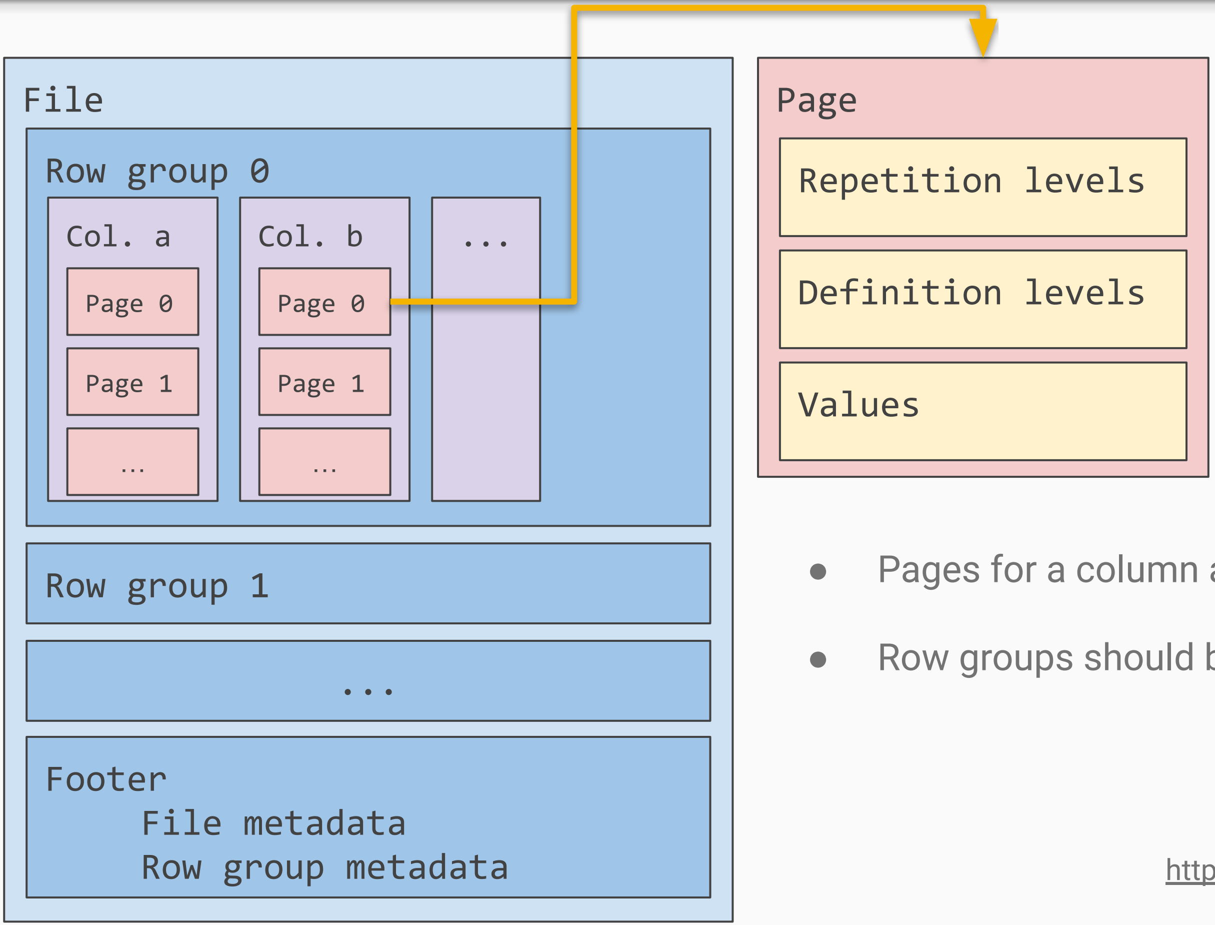
```
DocID: 20
Links:
        Backward: 10
        Backward: 30
        Forward:  80
```

After flattening...

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

- Repetition and definition columns are highly compressible
  - Not even needed for complete, tabular data!
- Value fields are now columnar
  - May also be compressed
- Columns are broken into **blocks** and compressed independently
  - This alleviates some decoding complexity and improves parallelism

Parquet format



File

Row group 0

Col. a

Page 0

Page 1

...

Col. b

Page 0

Page 1

...

...

Row group 1

...

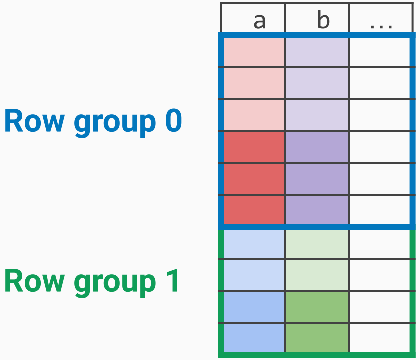
Footer
File metadata
Row group metadata

Page

Repetition levels

Definition levels

Values



- Pages for a column are compressed independently
- Row groups should be large, but fit in one HDFS block

https://parquet.apache.org/documentation/latest/

slido

What are some benefits and drawbacks of using large pages?

ⓘ Start presenting to display the poll results on this slide.

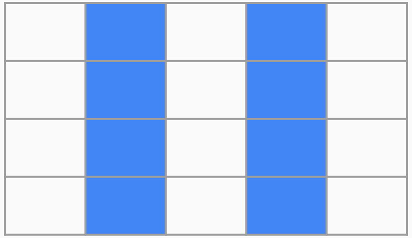
Page size in parquet

- Larger pages
  - Potentially better compression rate
  - But worse overhead for serial decoding
- Smaller pages
  - Faster decoding: fewer records to scan through before reconstructing
  - Worse compression rate due to less context

Column storage take-aways

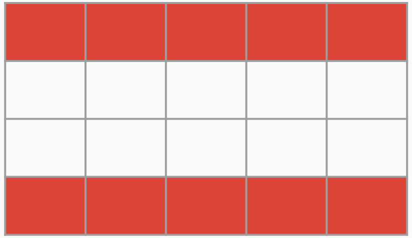
Pros:

- Can be much faster when you only want a **subset of attributes**
- Higher **storage efficiency** and **throughput**
- Collecting **data of the same type** enables compression and better access patterns



Cons:

- Reconstructing full tuples can be slow
  - Not great for **record-oriented** jobs
- Writes / deletion can be slow
- Handling non-tabular data is tricky
  - Dremel → Parquet to the rescue



After spring break...

- Dask (last software/framework)
- Then on to other things:
  - Data structures
  - Algorithms
  - Applications

Flattening example

Node.DocID

value	r	d

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d

Node.Links.Backward

value	r	d

Node.Name.Language.Code

value	r	d

Node.Name.Language.Country

value	r	d

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d

Node.Links.Backward

value	r	d

Node.Name.Language.Code

value	r	d

Node.Name.Language.Country

value	r	d

DocID is required

r=0, d=0

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

value	r	d

Node.Name.Language.Country

value	r	d

Links is **optional** (but present)
Links.Backward is **repeated** (but absent)

r=0, d=1

No value in this record, so fill a NULL

```
DocID: 10
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
Links:
      Backward: 10
      Backward: 30
      Forward:  80
Name:
      URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d
20	0	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

value	r	d

Node.Name.Language.Country

value	r	d

...Forward ⇒ d=2

No repetitions: r=0

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d
20	0	2
40	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

value	r	d

Node.Name.Language.Country

value	r	d

...Forward ⇒ d=2

Repetition in level r=1

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

value	r	d

Node.Name.Language.Country

value	r	d

```
DocID: 10
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’
```

...Forward ⇒ d=2

Repetition in level r=1

```
DocID: 20
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

value	r	d
en-us	0	2

Node.Name.Language.Country

value	r	d

Name.Language.Code required

First occurrence (r=0)
Full definition path (d=2)

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

value	r	d
en-us	0	2

Node.Name.Language.Country

value	r	d
us	0	3

...Country is optional ⇒ d=3

First occurrence (r=0)
Full definition path (d=3)

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

value	r	d
en-us	0	2
en	2	2

Node.Name.Language.Country

value	r	d
us	0	3

...Code is required

Repetition at r=2 (Name.Language)

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

value	r	d
en-us	0	2
en	2	2

Node.Name.Language.Country

value	r	d
us	0	3
NULL	2	2

...Language.Country optional

Repeated at Language level
r=2, d=2

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d
http://A	0	2

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

value	r	d
en-us	0	2
en	2	2

Node.Name.Language.Country

value	r	d
us	0	3
NULL	2	2

Node.Name.URL is optional
⇒ d=2

No repetitions: r=0

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d
http://A	0	2

Node.Links.Forward

value	r	d
20	0	2
40		
60		

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name ⇒ d=1

But no Language.* data...

Node.Name.Language.Code

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

Node.Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NULL	1	1

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d
http://A	0	2
http://B	1	2

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

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

Node.Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NU		

Node.Name.URL ⇒ d=2

Repetition at r=1 (Node.Name)

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d
http://A	0	2
http://B	1	2

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

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

Node.Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NU		

...Language.Code ⇒ d=2

Repetition at r=1 (Node.Name)

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

value	r	d
http://A	0	2
http://B	1	2

Node.Links.Forward

value	r	d
20	0	2
40		
60		

Node.Links.Backward

value	r	d
NULL	0	1

...Language.Country ⇒ d=3

Repetition at r=1 (Node.Name)

Node.Name.Language.Code

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

Node.Name.Language.Country

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

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0

Node.Name.URL

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

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

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

Node.Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NU		

Node.Name ⇒ d=1

No URL data

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0
20	0	0

Node.Name.URL

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

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1

Node.Name.Language.Code

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

Node.Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NU		

Node.DocID ⇒ d=0

Required field, new document (r=0)

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: 'http://C'
```

Flattening example

Node.DocID

value	r	d
10	0	0
20	0	0

Node.Name.URL

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

Node.Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2

Node.Links.Backward

value	r	d
NULL	0	1
10	0	2

Node.Name.Language.Code

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

Node.Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NU		

Node.Links.Backward ⇒ d=2

```
DocID: 10
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
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```

Flattening example

Node.DocID

value	r	d
10	0	0
20	0	0

Node.Name.URL

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

Node.Links.Forward

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

Node.Links.Backward

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

Node.Name.Language.Code

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

Node.Name.Language.Country

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

```
DocID: 10
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’
```

... and all the rest

```
DocID: 20
Links:
        Backward: 10
        Backward: 30
        Forward:  80
Name:
        URL: ‘http://C’
```