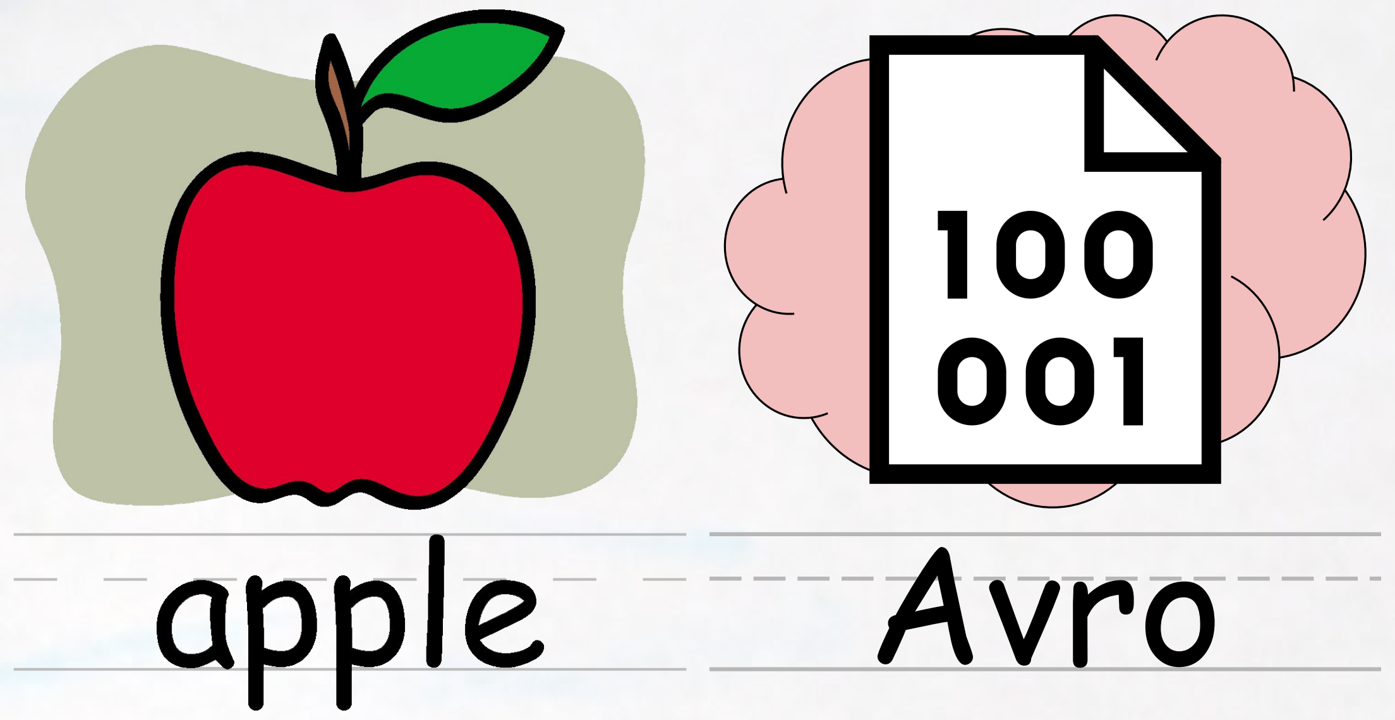
2-1-2018

Avro: Overview and Implications for Metadata Processing

Graham S. Hukill
Wayne State University, ej2929@wayne.edu

Cole Hudson
Wayne State University, fi1806@wayne.edu

Avro: Overview & Implications for Metadata Processing



Code4Lib Annual 2018

Cole Hudson & Graham Hukill
Wayne State University

What is Avro?

Originally developed for Apache Hadoop, Avro is a system that provides standardized ways to package and send data in big data clusters. For our use, we found Avro to be a useful tool in a metadata processing engine we use that runs on Apache Hadoop and Apache Spark.

Some useful tidbits to know:

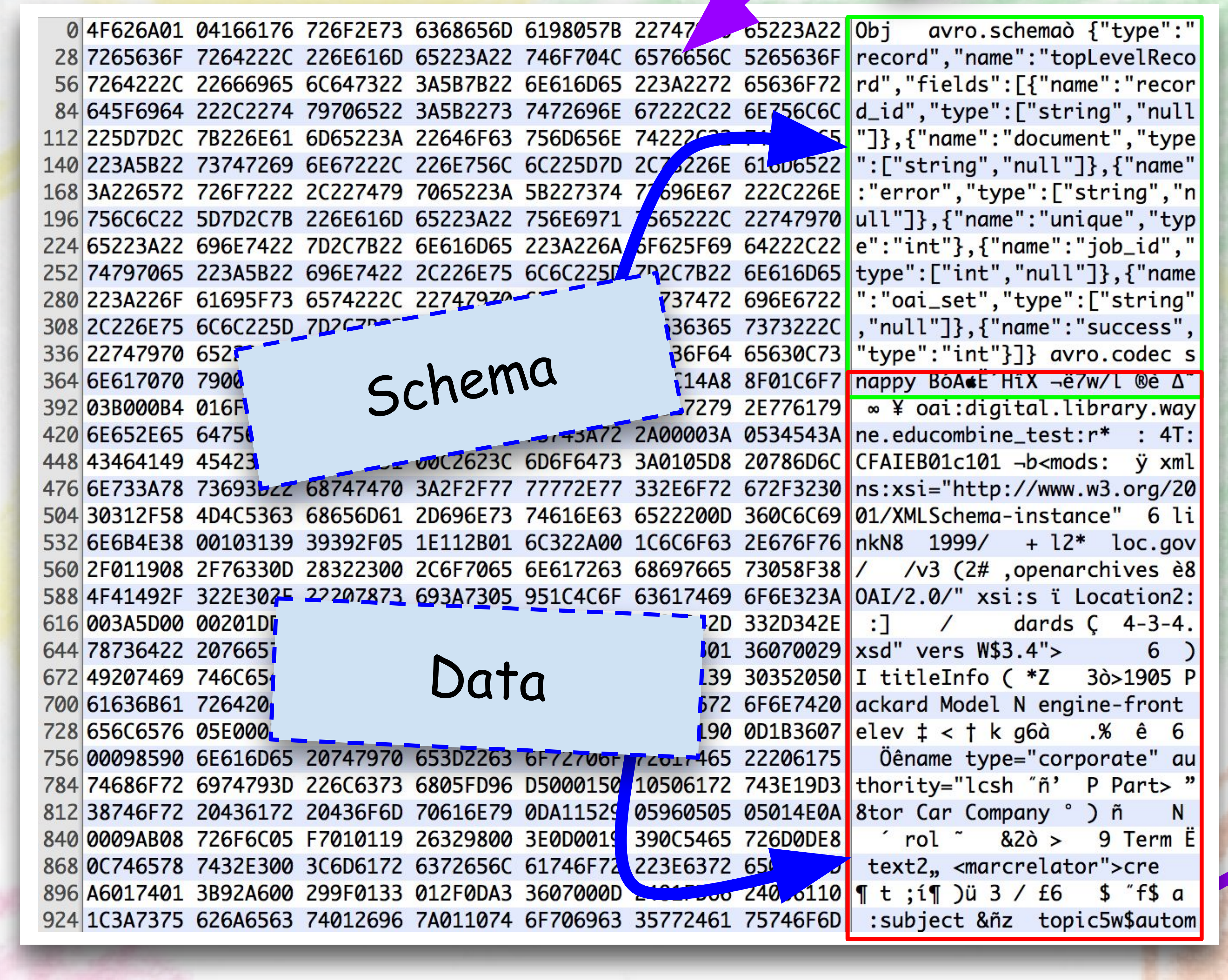
- Avro creates compact files (known as Avro data files) in a process known as serialization.
- Avro occupies a similar space as Binary JSON, Parquet, Thrift, and others.
- Avro serialization includes both the data as well as a schema--a JSON data structure that describes column names and data types.
- Avro is fast. In a big data environment with millions of records, Avro has to work quickly and create files that can be parsed easily by machine and human.
- Avro is robust. As schemas have changed over the spec's evolution, old versions of Avro files can still be read and used.

Library Connection

Using Avro opens the door to some interesting metadata processing implications. Have you ever had 500k records take down your computer as you processed them? Avro and Spark may be a way to perform powerful analysis using underpowered machines. Here is why:

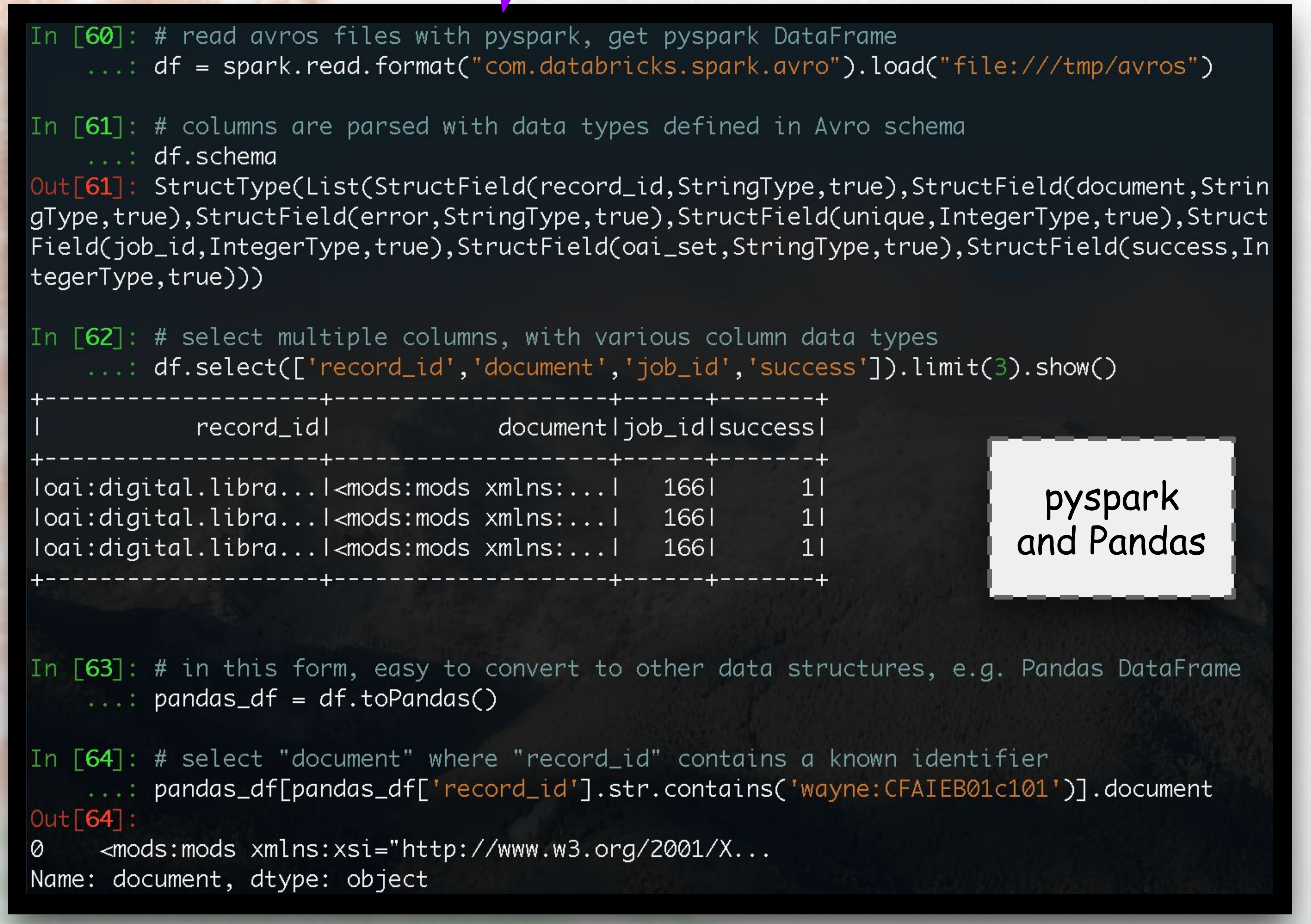
- Spark can efficiently handle files (like Avro) and split them into multiple segments that are reconstituted as needed.
- Avro files, because of their efficient serialization and integration into the Spark environment, can easily be parsed and queried in Spark.

This means you could take 10GB of records with a system that could handle at most 2GB at a time, and, with a SELECT query (see far chart), you could find all records with a certain field. If you were to write this type of query manually, you would need to do a fair amount of programming on your own: loop through records, keep track of all instances of the field you want, and, oh yeah, don't run out of memory or take an untenable amount of time.



```
  0 4F626A01 04166176 726F2E73 6368656D 6198057B 22747    65223A22 Obj   avro.schemaò {"type":"
 28 7265636F 7264222C 226E616D 65223A22 746F704C 6576656C 5265636F record","name":"topLevelReco
 56 7264222C 22666965 6C647322 3A5B7B22 6E616D65 223A2272 65636F72 rd","fields":[{"name":"recor
 84 645F6964 222C2274 79706522 3A5B2273 7472696E 67222C22 6E756C6C d_id","type":["string","null
112 225D7D2C 7B226E61 6D65223A 22646F63 756D656E 7422           65 "]},{"name":"document","type
140 223A5B22 73747269 6E67222C 226E756C 6C225D7D 2C 226E  61 D6522 ":["string","null"]},{"name"
168 3A226572 726F7222 2C227479 7065223A 5B227374 7 696E67 222C226E :"error","type":["string","n
196 756C6C22 5D7D2C7B 226E616D 65223A22 756E6971  565222C 22747970 ull"]},{"name":"unique","typ
224 65223A22 696E7422 7D2C7B22 6E616D65 223A226A  F625F69 64222C22 e":"int"},{"name":"job_id","
252 74797065 223A5B22 696E7422 2C226E75 6C6C225D 7D2C7B22 6E616D65 type":["int","null"]},{"name
280 223A226F 61695F73 6574222C 2274797        737472 696E6722 ":"oai_set","type":["string"
308 2C226E75 6C6C225D 7D2C                                 36365 7373222C ,"null"]},{"name":"success",
336 22747970 652                                            6F64 65630C73 "type":"int"}]} avro.codec s
364 6E617070 7900                                         14A8 8F01C6F7 nappy BòA●È HïX ¬ë7w/L ®è Δ
392 03B000B4 016F                                         7279 2E776179  ∞ ¥ oai:digital.library.way
420 6E652E65 6475                              3A72 2A00003A 0534543A ne.educombine_test:r*   : 4T:
448 43464149 45423              00C2623C 6D6F6473 3A0105D8 20786D6C CFAIEB01c101 ¬b<mods:  ÿ xml
476 6E733A78 73693    68747470 3A2F2F77 77772E77 332E6F72 672F3230 ns:xsi="http://www.w3.org/20
504 30312F58 4D4C5363 68656D61 2D696E73 74616E63 6522200D 360C6C69 01/XMLSchema-instance"  6 li
532 6E6B4E38 00103139 39392F05 1E112B01 6C322A00 1C6C6F63 2E676F76 nkN8  1999/   + l2*  loc.gov
560 2F011908 2F76330D 28322300 2C6F7065 6E617263 68697665 73058F38 /   /v3 (2# ,openarchives è8
588 4F41492F 322E302F 22207873 693A7305 951C4C6F 63617469 6F6E323A OAI/2.0/" xsi:s ï Location2:
616 003A5D00 00201D                                    2D 332D342E  :]    /     dards Ç  4-3-4.
644 78736422 207665                                    01 36070029 xsd" vers W$3.4">      6  )
672 49207469 746C65                                    39 30352050 I titleInfo ( *Z   3ò>1905 P
700 61636B61 726420                                    72 6F6E7420 ackard Model N engine-front
728 656C6576 05E000                                    90 0D1B3607 elev ‡ < † k g6à   .%  ê  6
756 00098590 6E616D65 20747970 653D2263 6F72706F   465 22206175  Öêname type="corporate" au
784 74686F72 6974793D 226C6373 6805FD96 D5000150 10506172 743E19D3 thority="lcsh ˝ñ'  P Part> "
812 38746F72 20436172 20436F6D 70616E79 0DA11529 05960505 05014E0A 8tor Car Company ˚ ) ñ    N
840 0009AB08 726F6C05 F7010119 26329800 3E0D0019 390C5465 726D0DE8  ´ rol ˜   &2ò >   9 Term Ë
868 0C746578 7432E300 3C6D6172 6372656C 61746F72 223E6372 65        text2„ <marcrelator">cre
896 A6017401 3B92A600 299F0133 012F0DA3 3607000D          240 6110 ¶ t ;í¶ )ü 3 / £6   $ ˝f$ a
924 1C3A7375 626A6563 74012696 7A011074 6F706963 35772461 75746F6D  :subject &ñz  topic5w$autom
```

Allows for



```
In [60]: # read avros files with pyspark, get pyspark DataFrame
    ...: df = spark.read.format("com.databricks.spark.avro").load("file:///tmp/avros")

In [61]: # columns are parsed with data types defined in Avro schema
    ...: df.schema
Out[61]: StructType(List(StructField(record_id,StringType,true),StructField(document,Strin
gType,true),StructField(error,StringType,true),StructField(unique,IntegerType,true),Struct
Field(job_id,IntegerType,true),StructField(oai_set,StringType,true),StructField(success,In
tegerType,true)))

In [62]: # select multiple columns, with various column data types
    ...: df.select(['record_id','document','job_id','success']).limit(3).show()
+--------------------+--------------------+------+-------+
|           record_id|            document|job_id|success|
+--------------------+--------------------+------+-------+
|oai:digital.libra...|<mods:mods xmlns:...|   166|      1|
|oai:digital.libra...|<mods:mods xmlns:...|   166|      1|
|oai:digital.libra...|<mods:mods xmlns:...|   166|      1|
+--------------------+--------------------+------+-------+


In [63]: # in this form, easy to convert to other data structures, e.g. Pandas DataFrame
    ...: pandas_df = df.toPandas()

In [64]: # select "document" where "record_id" contains a known identifier
    ...: pandas_df[pandas_df['record_id'].str.contains('wayne:CFAIEB01c101')].document
Out[64]:
0    <mods:mods xmlns:xsi="http://www.w3.org/2001/X...
Name: document, dtype: object
```

A portable database

Avro files alone do not make a "portable database," but they get you closer! A SQL dump may contain the same schema support that Avro does, but require a spinning database instance to interact with. Other formats might offer compressible, binary serializations, but might also be difficult to reconstitute and inspect.

Avro **strikes a balance** between:

- efficient storage and transmission
- defined data types
- ease of analysis through readily available tools...

...without requiring extensive infrastructure. In the same way that MySQL allows querying over a million rows, so too can the combination of Avro and environments like Spark, all while efficiently utilizing computer resources if available, or working within the constraints of what's at hand.