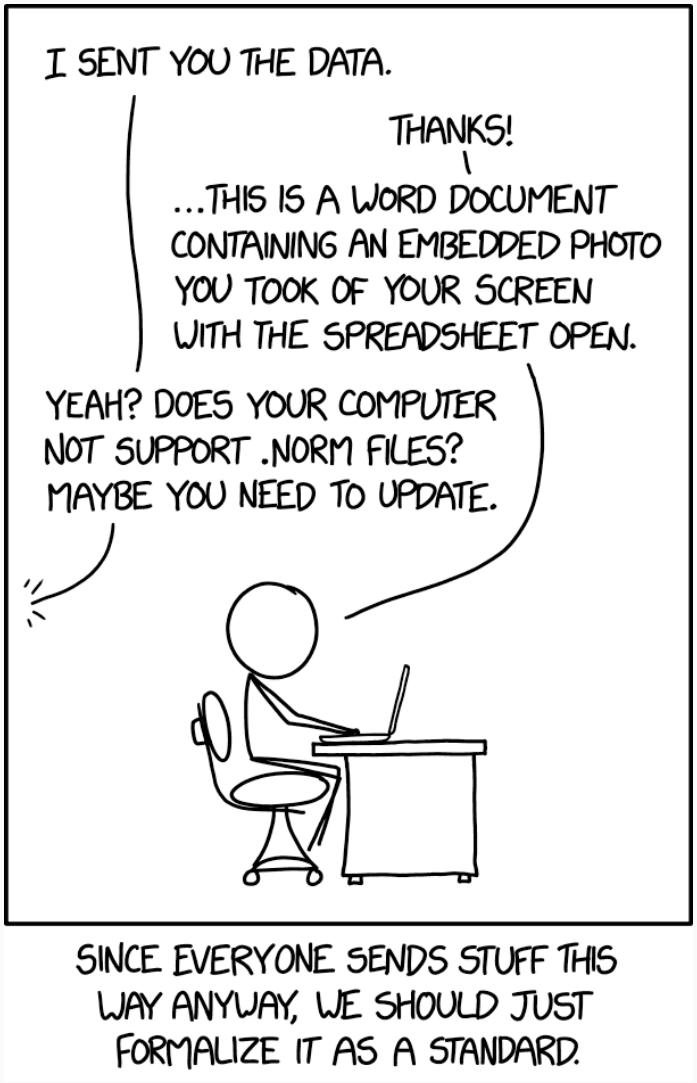
Data formats

Subhasis Ray

2023-01-27

Why do we need data formats?

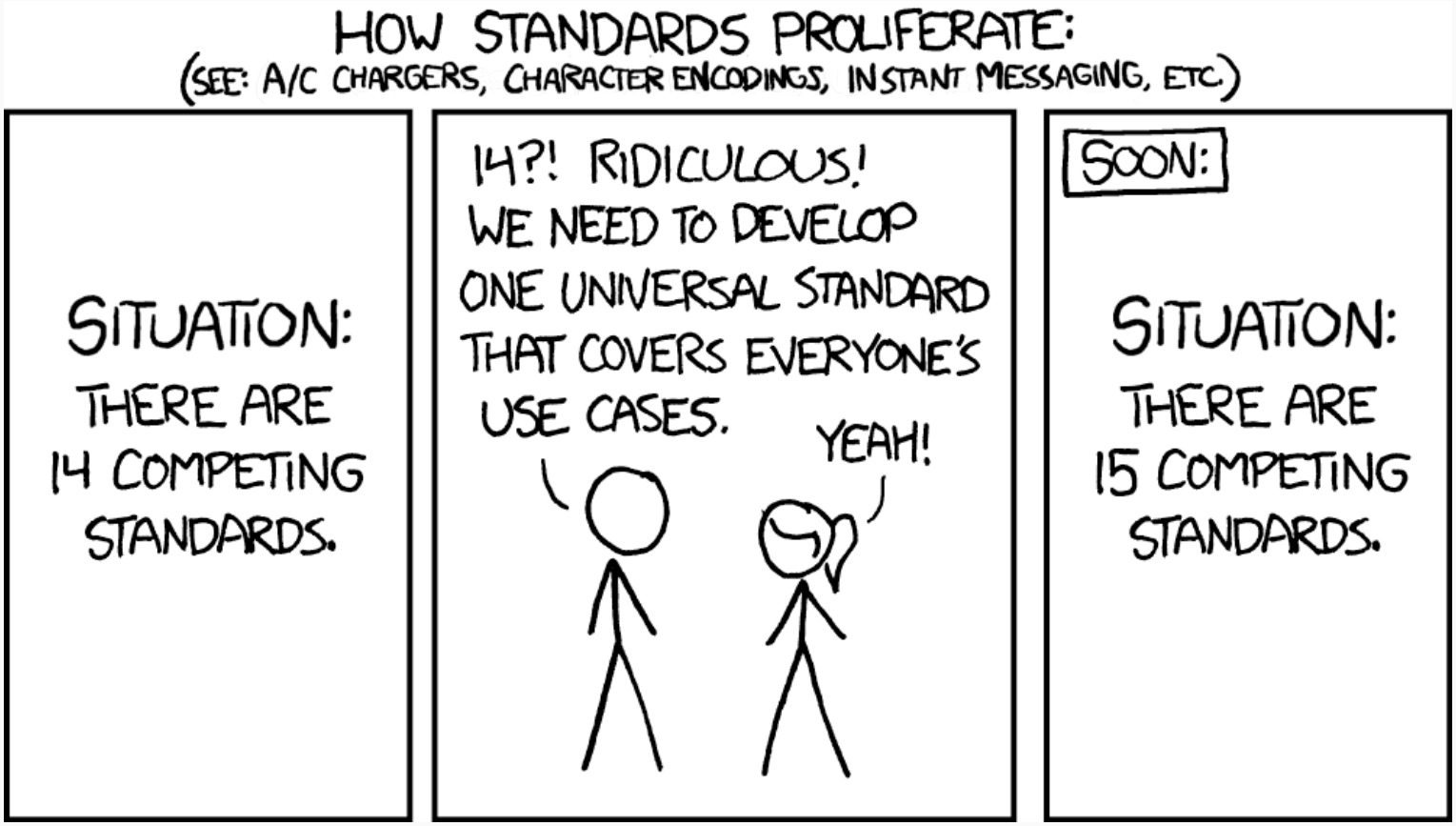


.NORM Normal File Format [https://xkcd.com/2116/]

Why do we need data formats?

- Sharing with others
- Access on different platforms (portability)
- Access with different programs
- Better design from expertise and experience
- Efficiency (space/time)
- There is no silver bullet
    - Domain specific formats

Standards



https://xkcd.com/927/

Common data formats

- Text: CSV, XML, TXT, JSON, YAML,
- Data Matrix: CSV, Excel (XLS), OpenDocument (ODS), STATA (.DTA)?, Feather, Parquet, Arrow*, ORC, SPSS (SAV),
- Structured/hierarchical: HDF5, netCDF, TXT, JSON, YAML, MAT, PICKLE,
- Binary: HDF5, XLS, netCDF, STATA (.DTA), MAT, PICKLE, FITS, DICOM, Feather, Parquet, Arrow*, ORC, SPSS (SAV),
- Big Data: HDF5, netCDF, Feather, Parquet, ORC

Text format: CSV (comma separated values), TSV (tab separated values)

- Simple data matrix
- Each line/row is a data record
- Each column represents one attribute
- Every record has same sequence of fields
- Single error or omission can mess up everything
- Meta-information must passed separately
- Commas, linebreaks in the data?
  - Some use quotations around fields
  - What if quotes are also present in data?
- RFC 4180

Text format: JSON/YAML

- JSON: Java Script Object Notation
- Derived from JavaScript, popular with JavaScript developers, but independent of JavaScript
- Plain text file with structure
- Collection of name-value pairs
- Ordered list of values
- YAML: Yet Another Markup Language (superset of JSON)

```
{
    "firstName": "John",
    "lastName": "Doe",
    "age": 20,
    "address": {
        "streetAddress": "Alpha, Sector 101",
        "city": "Mohali",
        "state": "Punjab",
        "postalCode": "140603"
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "8910 234-567"
        },
        {
            "type": "office",
            "number": "160 1234 5678"
        }
    ],
}
```

Text format: XML (eXtensible Markup Language)

- Extremely versatile format (too much?)
- Originally designed for documents on the internet
- Tags define structure and metadata (you can define your own tags)
- Data within tags
- Binary data can be encoded as string using Base64
- Many formats based on XML: RSS, SVG, Open Document, newer MS Office formats
- Parsing can be hard/slow

XML Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
<firstName>John</firstName>
<lastName>Doe</lastName>
<age>20</age>
<address>
  <streetAddress>Alpha, Sector 101</streetAddress>
  <city>Mohali</city>
  <state>Punjab</state>
  <postalCode>140603</postalCode>
</address>
<phoneNumbers>
  <type>home</type>
  <number>8910 234-567</number>
</phoneNumbers>
<phoneNumbers>
  <type>office</type>
  <number>160 1234 5678</number>
</phoneNumbers>
</root>
```

XSD: XML Schema

Describes the structure of an XML document (alternative to Document Type Definition or DTD)

- Written in XML itself
- Restrict what is allowed in an XML doc
- Specify data type
- Define default or fixed values
- Set multiplicity and order of elements

XSL - eXtensible Stylesheet Language

- Based on CSS (cascading style sheets used for styling HTML pages)
- XSLT: XSL Transformation - rules to transform data from XML for rendering (or converting, e.g., into XHTML)
  - An XSLT processor applies the rules on the XML document
- XPath: Standard for addressing nodes in an XML tree
  - Used by XSLT
  - Very powerful - can be used by a program to access specific nodes
- XQuery: Originally a query language for XML documents
  - Extract and manipulate data from XML
  - A full functional programming language
  - Overlaps with XSLT

File formats for Big Data

Designed for efficient access of large datasets

- For analysis, we often want to access a given field of all data at once. Thus column-oriented access is preferable.
- Compression saves space, direct memory access saves time

Serialization
Process of translating a data structure or object state for storage or transmission, that can be reconstructed later (deserialized).

Formats for Big-Data

Parquet (Apache)

- Column oriented file format
- Decompress and decode to access, sequentially
- Saves space for storage on disk
- Archival purpose (the specs are stable, so can be used years later)

Formats for Big-Data

Arrow (Apache)

- Column oriented serialization protocol
- Runtime, in-memory representation
- No deserialization required
- Used for streaming data between processes
- Can be memory-mapped from disk
- Can read files in batches (no need to load the whole data at once)

Formats for big data

Feather (Apache)

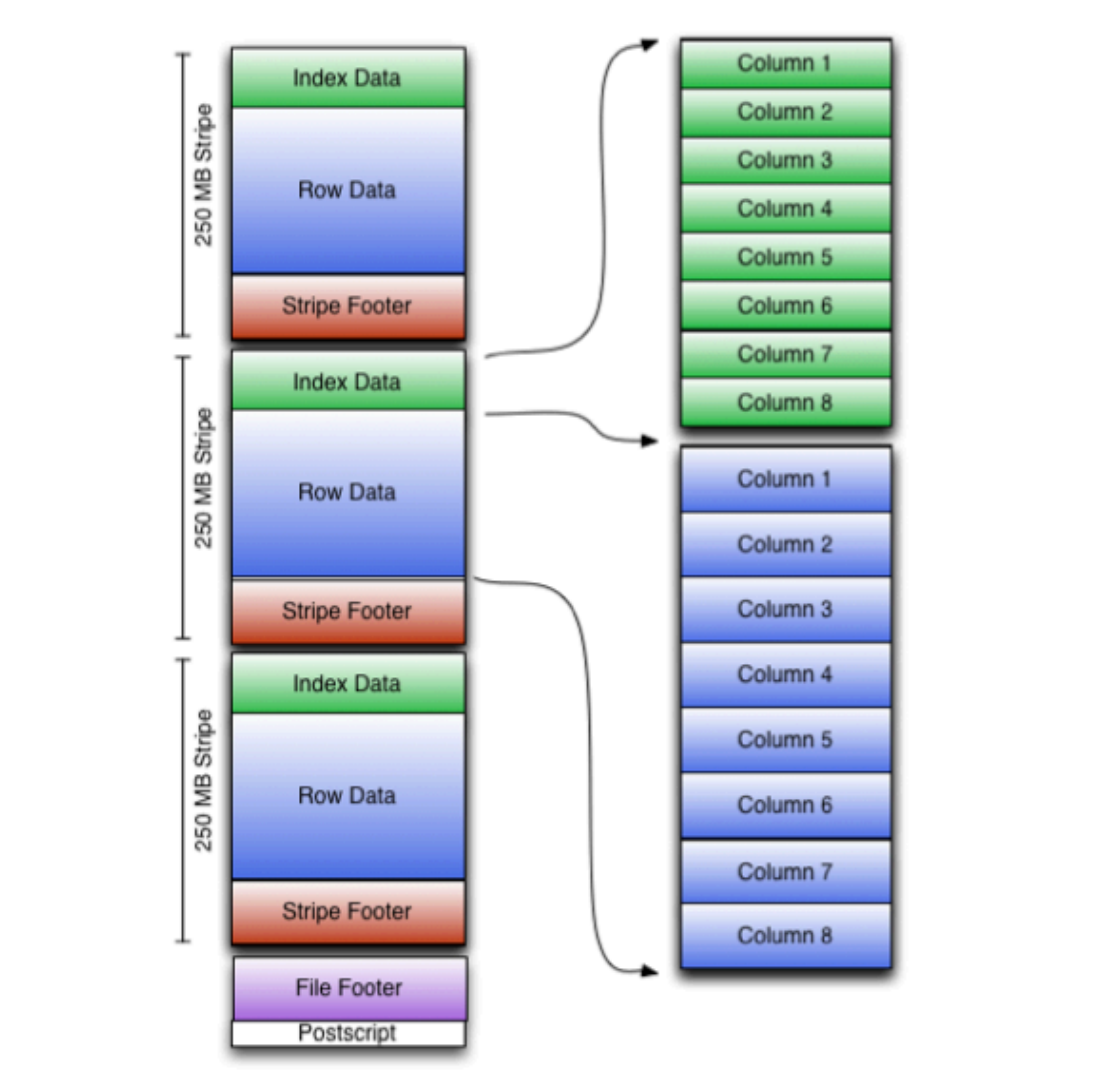
- Apache Arrow dumped on disk!
- Data can be split into multiple files - efficient parallel processing
- Fast read and write
- Allows compression, too!

Formats for big data

ORC (Optimized Row Columnar)

- Originally aimed at Apache Hive data (Hive is a data warehouse software)
- Supports complex data structures (all these big-data formats do!) like struct, list, map, etc.
- Row data grouped in column-oriented "stripes"
- Footer contains column-level aggregates
- Allows skipping rows - hence fast random access
- However, not well supported on Windows/Mac

ORC file layout



https://orc.apache.org/specification/ORCv1/

Further reading

- Pandas documentation
- All XML related standards are defined by W3C (but better not try to read them).
- Feather: `https://arrow.apache.org/docs/python/feather.html`
- ORC: `https://orc.apache.org/specification/ORCv1/`