AQUAQ
Analytics

Vaex, Arrow, Parquet

Experts in fast data solutions

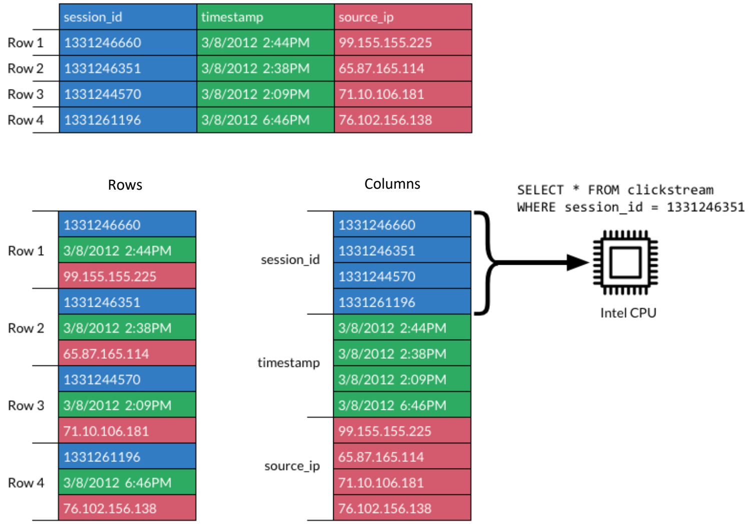
for demanding environments

Focus on two things

- Open columnar data formats and tools:
  - Arrow, Parquet and others
  - Vaex
- How can we use these tools for similar applications to kdb+?
  - similarities
  - differences
  - some interesting possibilities…

Columnar data formats

- Apache ORC
- Apache Parquet
- Apache Arrow
- and of course kdb+



	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

Rows

	1331246660
Row 1	3/8/2012 2:44PM
	99.155.155.225
	1331246351
Row 2	3/8/2012 2:38PM
	65.87.165.114
	1331244570
Row 3	3/8/2012 2:09PM
	71.10.106.181
	1331261196
Row 4	3/8/2012 6:46PM
	76.102.156.138

Columns

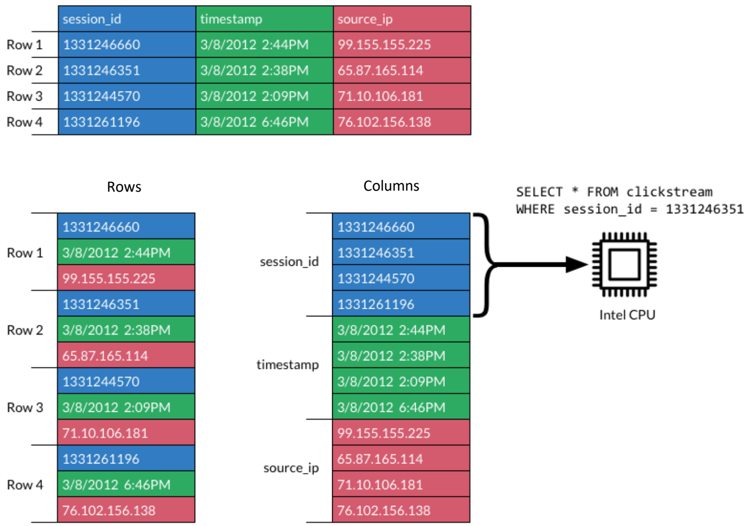
	1331246660
session_id	1331246351
	1331244570
	1331261196
	3/8/2012 2:44PM
timestamp	3/8/2012 2:38PM
	3/8/2012 2:09PM
	3/8/2012 6:46PM
	99.155.155.225
source_ip	65.87.165.114
	71.10.106.181
	76.102.156.138

```
SELECT * FROM clickstream
WHERE session_id = 1331246351
```

- Advantages:
  - Only read required columns
  - Data adjacency for sequential access (scans)
  - O(1) (constant-time) random access
  - SIMD optimization and vectorization-friendly
  - Compression friendly

Columnar data formats

- Apache ORC
- Apache Parquet
- **Apache Arrow**
- **and of course kdb+**



	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

Rows

Row 1	1331246660 3/8/2012 2:44PM 99.155.155.225
Row 2	1331246351 3/8/2012 2:38PM 65.87.165.114
Row 3	1331244570 3/8/2012 2:09PM 71.10.106.181
Row 4	1331261196 3/8/2012 6:46PM 76.102.156.138

Columns

session_id	1331246660 1331246351 1331244570 1331261196
timestamp	3/8/2012 2:44PM 3/8/2012 2:38PM 3/8/2012 2:09PM 3/8/2012 6:46PM
source_ip	99.155.155.225 65.87.165.114 71.10.106.181 76.102.156.138

```
SELECT * FROM clickstream
WHERE session_id = 1331246351
```

- Advantages:
  - Only read required columns
  - Data adjacency for sequential access (scans)
  - O(1) (constant-time) random access
  - SIMD optimization and vectorization-friendly
  - Compression friendly

APACHE ARROW

- Started in 2016 by Wes McKinney
- The “Arrow Columnar Format” includes a language-agnostic in-memory data structure specification, metadata serialization, and a protocol for serialization and generic data transport.
  - Columns can be either fixed size or variable size
  - Nulls are supported via validity maps, and variable width records via offset calculations
  - Built in types are what you might expect:
    - int8/16/32/64
    - float16/32/64
    - time/date/timestamp/duration/interval
    - Binary
    - utf8
    - dictionaries (enums)

- Arrays are defined by a few pieces of metadata and data:
  - A logical data type.
  - A sequence of buffers.
  - A length as a 64-bit signed integer. Implementations are permitted to be limited to 32-bit lengths, see more on this below.
  - A null count as a 64-bit signed integer.
  - An optional dictionary, for dictionary-encoded arrays.

AQUAQ Analytics

APACHE ARROW

For example a primitive array of int32s:

```
[1, null, 2, 4, 8]
```

Would look something like:

```
* Length: 5, Null count: 1
* Validity bitmap buffer:

  |Byte 0 (validity bitmap) | Bytes 1-63            |
  |-------------------------|-----------------------|
  | 00011101                | 0 (padding)           |

* Value Buffer:

  |Bytes 0-3   | Bytes 4-7   | Bytes 8-11  | Bytes 12-15 | Bytes 16-19 | Bytes 20-63 |
  |------------|-------------|-------------|-------------|-------------|-------------|
  | 1          | unspecified | 2           | 4           | 8           | unspecified |
```

https://arrow.apache.org/docs/format/Columnar.html#

AQUAQ Analytics

APACHE ARROW

For example a primitive array of int32s:

```
[1, null, 2, 4, 8]
```

Would look something like:

metadata/header

```
* Length: 5, Null count: 1
* Validity bitmap buffer:

  |Byte 0 (validity bitmap) | Bytes 1-63             |
  |-------------------------|------------------------|
  | 00011101                | 0 (padding)            |
```

```
* Value Buffer:

  |Bytes 0-3   | Bytes 4-7   | Bytes 8-11  | Bytes 12-15 | Bytes 16-19 | Bytes 20-63 |
  |------------|-------------|-------------|-------------|-------------|-------------|
  | 1          | unspecified | 2           | 4           | 8           | unspecified |
```

https://arrow.apache.org/docs/format/Columnar.html#

For example a primitive array of int32s:

```
[1, null, 2, 4, 8]
```

Would look something like:

metadata/header

```
* Length: 5, Null count: 1
* Validity bitmap buffer:

  |Byte 0 (validity bitmap) | Bytes 1-63            |
  |-------------------------|-----------------------|
  | 00011101                | 0 (padding)           |
```

data vector

```
* Value Buffer:

  |Bytes 0-3   | Bytes 4-7   | Bytes 8-11  | Bytes 12-15 | Bytes 16-19 | Bytes 20-63 |
  |------------|-------------|-------------|-------------|-------------|-------------|
  | 1          | unspecified | 2           | 4           | 8           | unspecified |
```

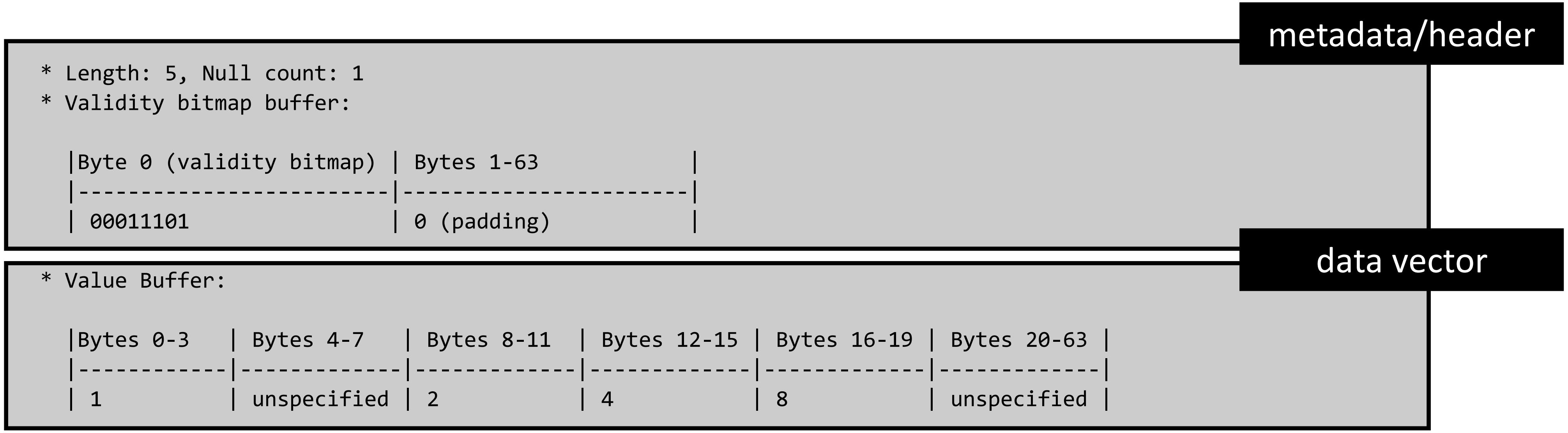
https://arrow.apache.org/docs/format/Columnar.html#

For example a primitive array of int32s:

```
[1, null, 2, 4, 8]
```

"The recommendation for 64 byte alignment comes from the [Intel performance guide](#) that recommends alignment of memory to match SIMD register width. The specific padding length was chosen because it matches the largest SIMD instruction registers available on widely deployed x86 architecture (Intel AVX-512)."

Would look something like:



metadata/header

```
* Length: 5, Null count: 1
* Validity bitmap buffer:

  |Byte 0 (validity bitmap) | Bytes 1-63            |
  |-------------------------|-----------------------|
  | 00011101                | 0 (padding)           |
```

data vector

```
* Value Buffer:

  |Bytes 0-3   | Bytes 4-7   | Bytes 8-11  | Bytes 12-15 | Bytes 16-19 | Bytes 20-63 |
  |------------|-------------|-------------|-------------|-------------|-------------|
  | 1          | unspecified | 2           | 4           | 8           | unspecified |
```

https://arrow.apache.org/docs/format/Columnar.html#

kdb+

- The kdb data format is very similar, it stores data as columns and includes the types you might expect

```
q)l:1 0N 2 4 8i                  / simple int32 vector

q)0 1 4 8 9 10 cut 14#-8!l
,0x01
0x000000
0x22000000
,0x06
,0x00
0x05000000

q)0N 8#14_-8!l
0x01000000
0x00000080
0x02000000
0x04000000
0x08000000
```

https://code.kx.com/q/kb/serialization/

kdb+

- The kdb data format is very similar, it stores data as columns and includes the types you might expect

```
q)l:1 0N 2 4 8i                 / simple int32 vector
```

metadata/header

```
q)0 1 4 8 9 10 cut 14#-8!l
,0x01                           / little endian
0x000000
0x22000000                      / message length
,0x06                           / type
,0x00                           / attributes
0x05000000                      / vector length
```

```
q)0N 8#14_-8!l
0x01000000                      / 1
0x00000080                      / null
0x02000000                      / 2
0x04000000                      / 4
0x08000000                      / 8
```

https://code.kx.com/q/kb/serialization/

kdb+

- The kdb data format is very similar, it stores data as columns and includes the types you might expect

```
q)l:1 0N 2 4 8i                / simple int32 vector
```

metadata/header

```
q)0 1 4 8 9 10 cut 14#-8!l
,0x01                          / little endian
0x000000
0x22000000                     / message length
,0x06                          / type
,0x00                          / attributes
0x05000000                     / vector length
```

data vector

```
q)0N 8#14_-8!l
0x01000000                     / 1
0x00000080                     / null
0x02000000                     / 2
0x04000000                     / 4
0x08000000                     / 8
```

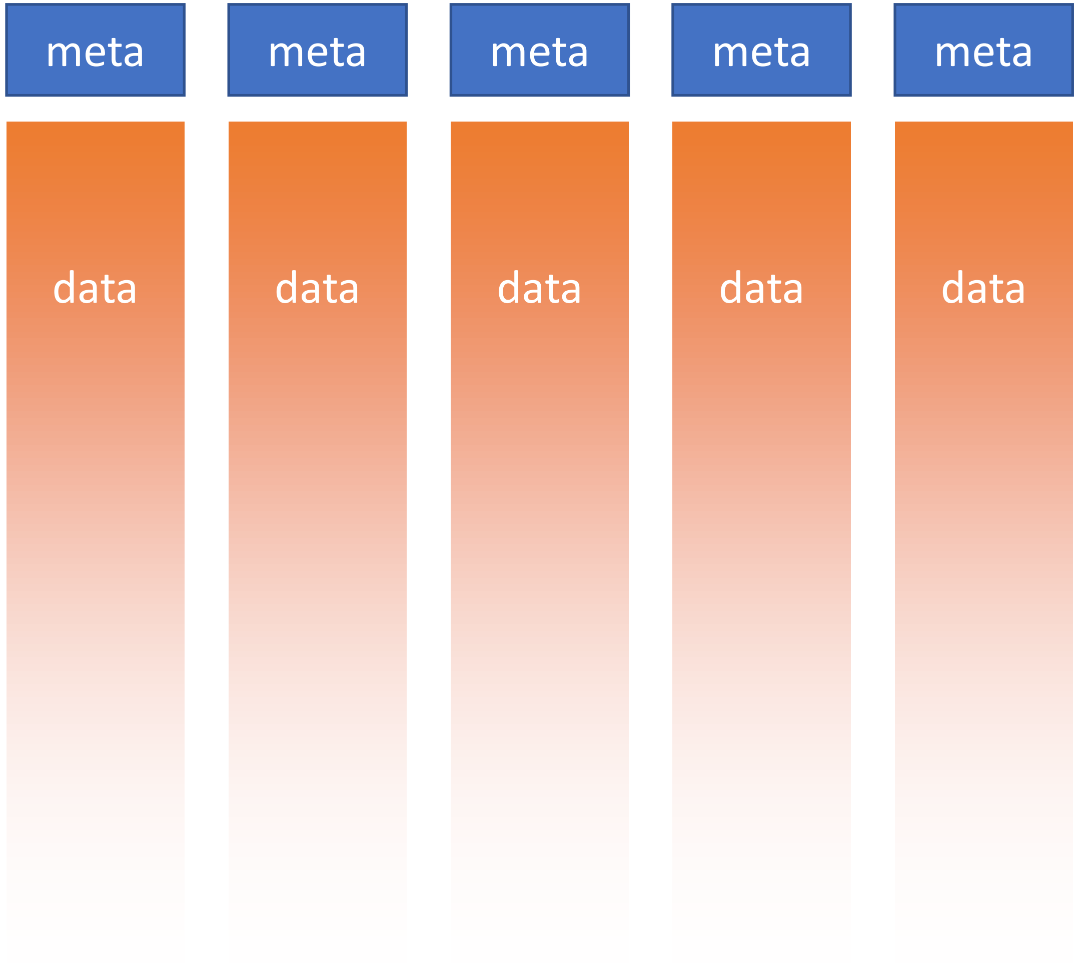
https://code.kx.com/q/kb/serialization/

Arrow <-> kdb

meta

data

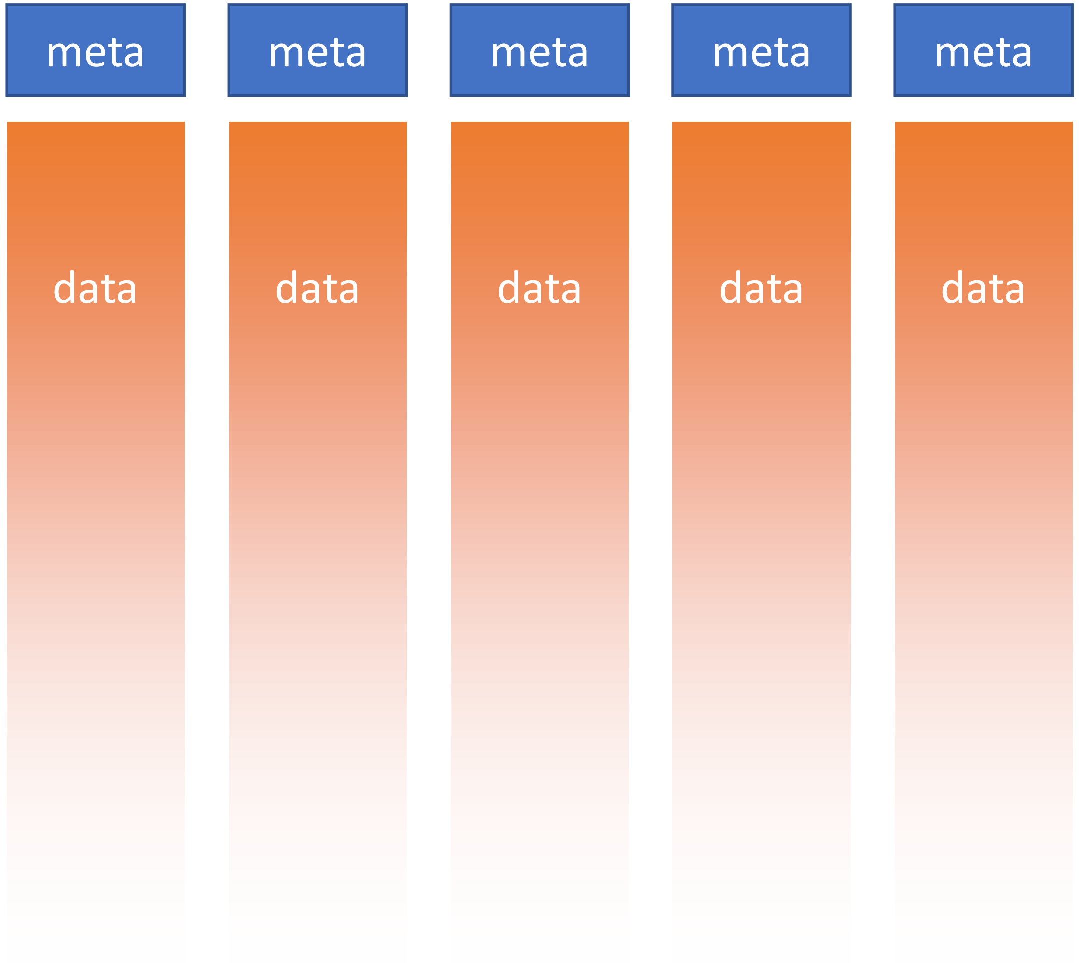
Arrow <-> kdb



meta meta meta meta meta

data data data data data

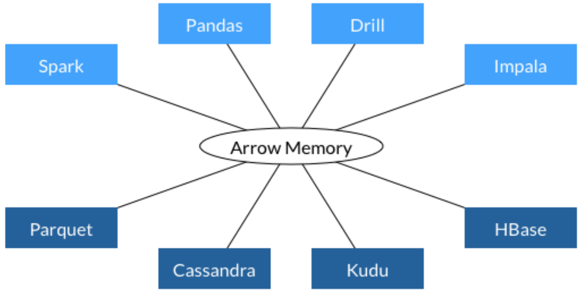
Arrow <-> kdb



Arrow this is a struct

Kdb+ this is a table

Arrow <-> kdb



- Broadly similar:
  - Types
  - General structure
  - **General philosophy that serialization format == in-memory format. Zero copy!**

(kx had this insight several decades ago)

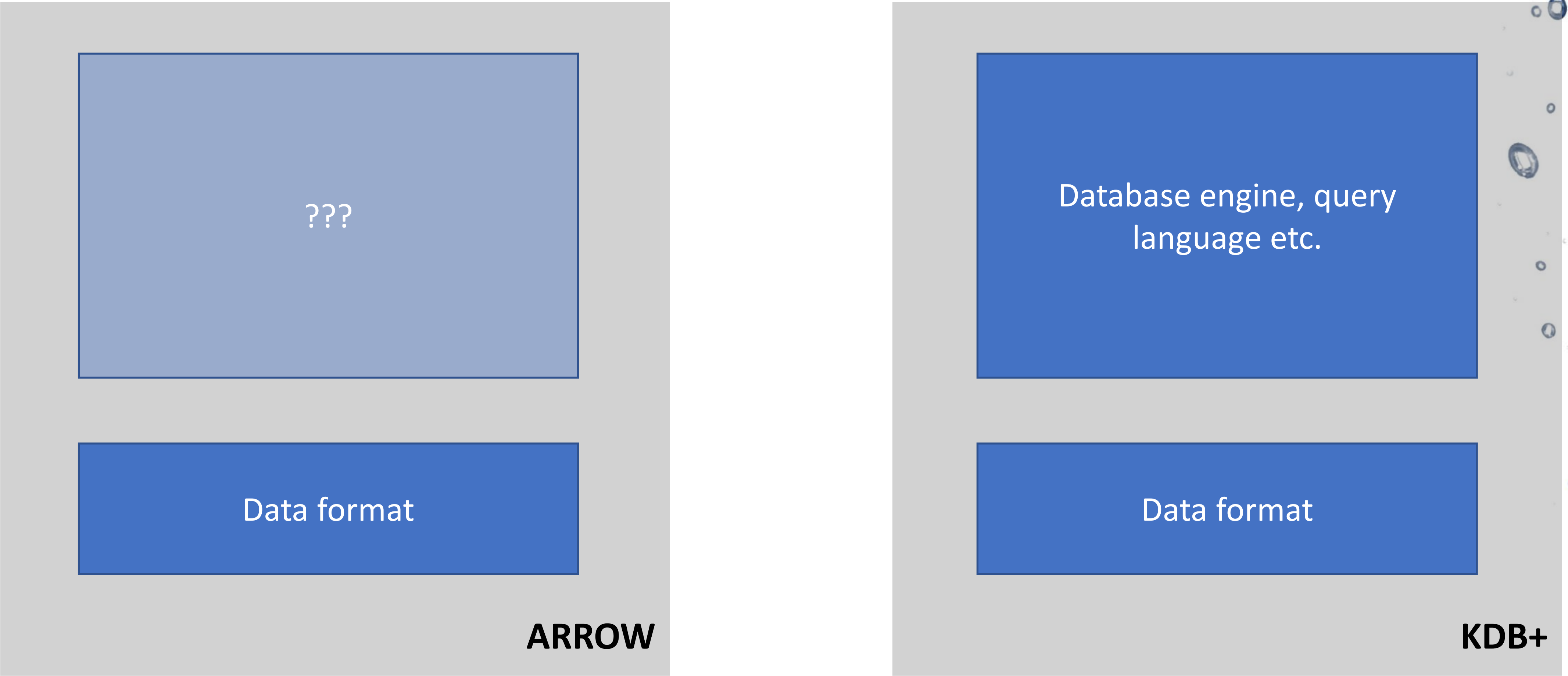
- Except:
  - Arrow doesn't splay
  - Arrow batches records into buffers
  - Null handling and some other minor differences in meta data

Arrow <-> kdb

So the data formats are fairly similar, but the structure of the projects are quite different

- **Arrow**:
  - Format and implementations in a bunch of languages
  - Tools to read into and out of arrow
  - A few other bits and pieces (arrow flight RPC, plasma in-memory store, gandiva expression optimizer)
  - Main points are columns and zero-copy
- **KDB**:
  - Format (in memory and on disk)
  - Full database engine (q-SQL) and programming language

Arrow <-> kdb



???

Data format

ARROW

Database engine, query language etc.

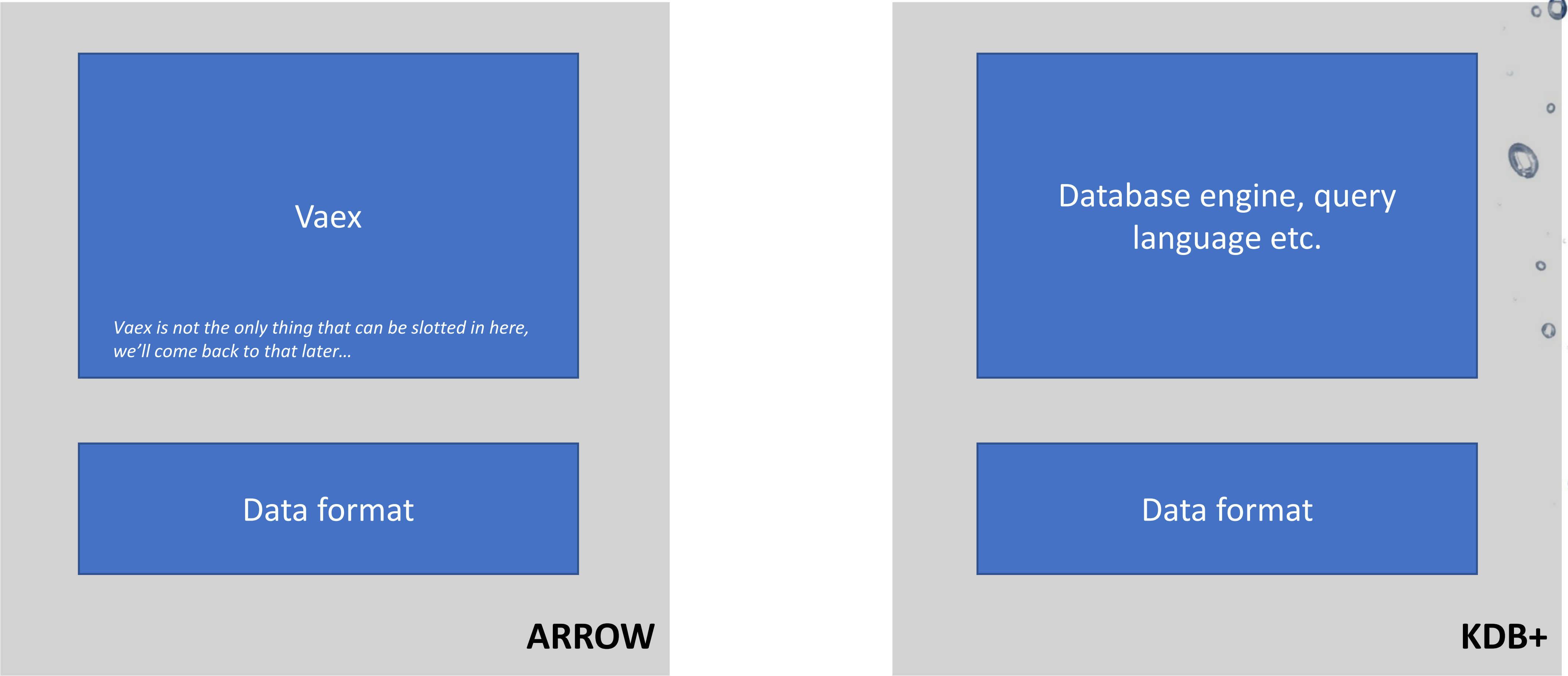
Data format

KDB+

Vaex

- Stands for *Visualize and explore*
- Vaex uses memory mapping on top of arrow files, a zero memory copy policy, and lazy computations
- And provides an API that looks like pandas
- Just released version 4.0 two days ago!

Vaex, Arrow and kdb+



Vaex

Vaex is not the only thing that can be slotted in here, we'll come back to that later…

Data format

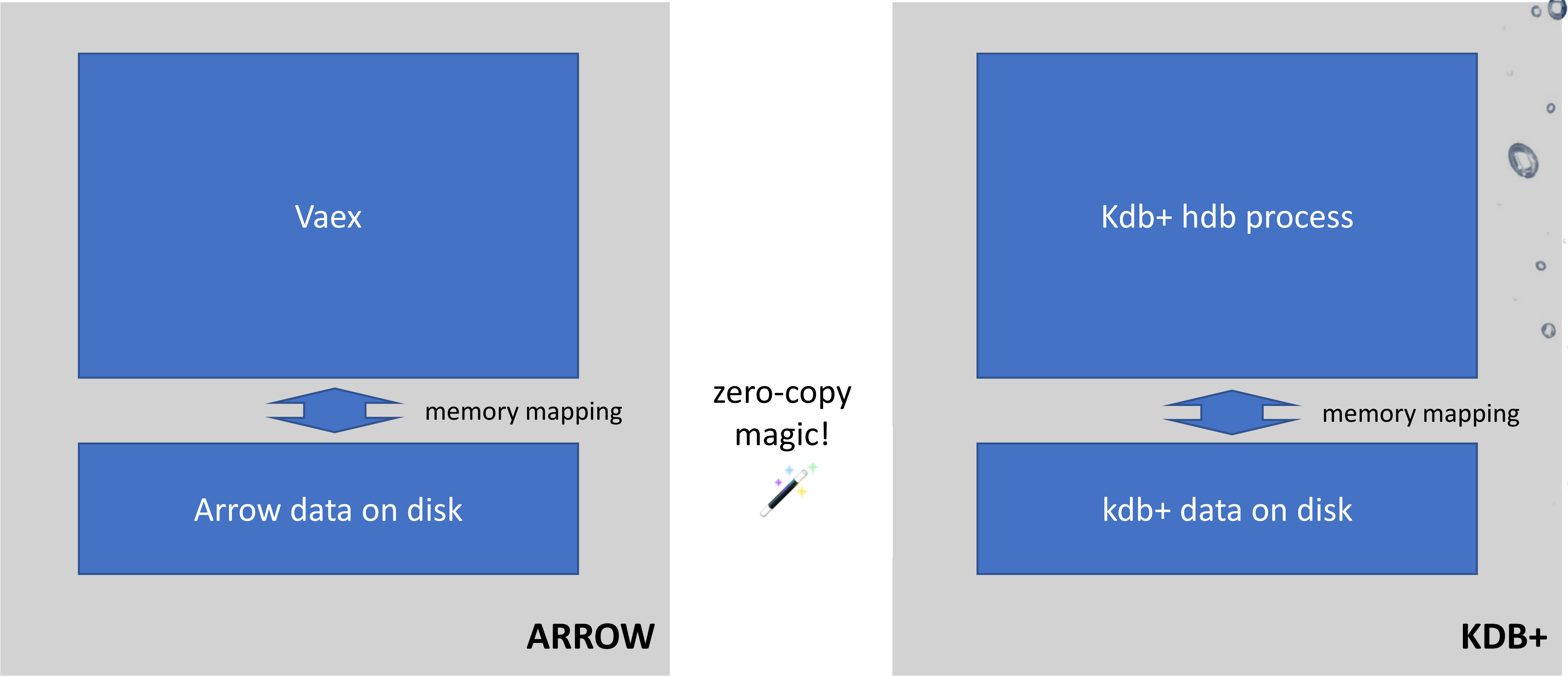
ARROW

Database engine, query language etc.

Data format

KDB+

Vaex, Arrow and kdb+



Vaex

memory mapping

Arrow data on disk

ARROW

zero-copy magic!

Kdb+ hdb process

memory mapping

kdb+ data on disk

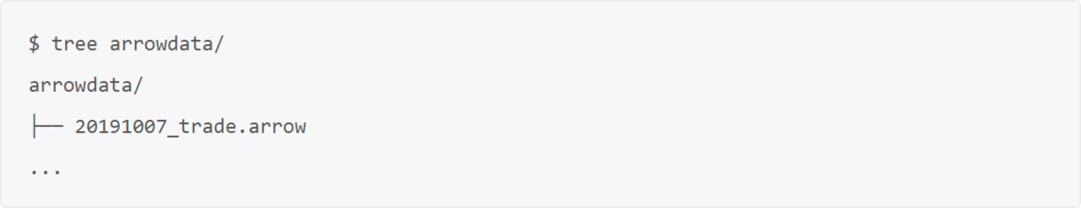
KDB+

Vaex, Arrow and kdb+

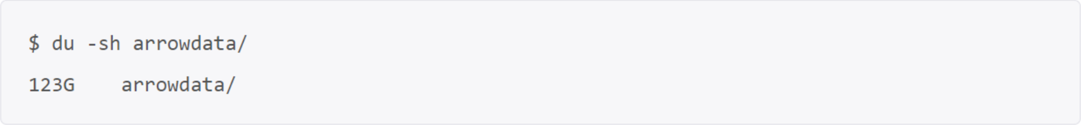
- Let’s have a look at a side by side demo
- a single day of NYSE TAQ (all the trades and quotes for securities listed on US regulated exchanges)
- Full details at https://www.aquaq.co.uk/q/comparing-columnar-data-formats-arrow-vaex-and-kdb/
- Xeon Gold, 128GB, HDD

Vaex, Arrow and kdb+ - data on disk

vaex/arrow

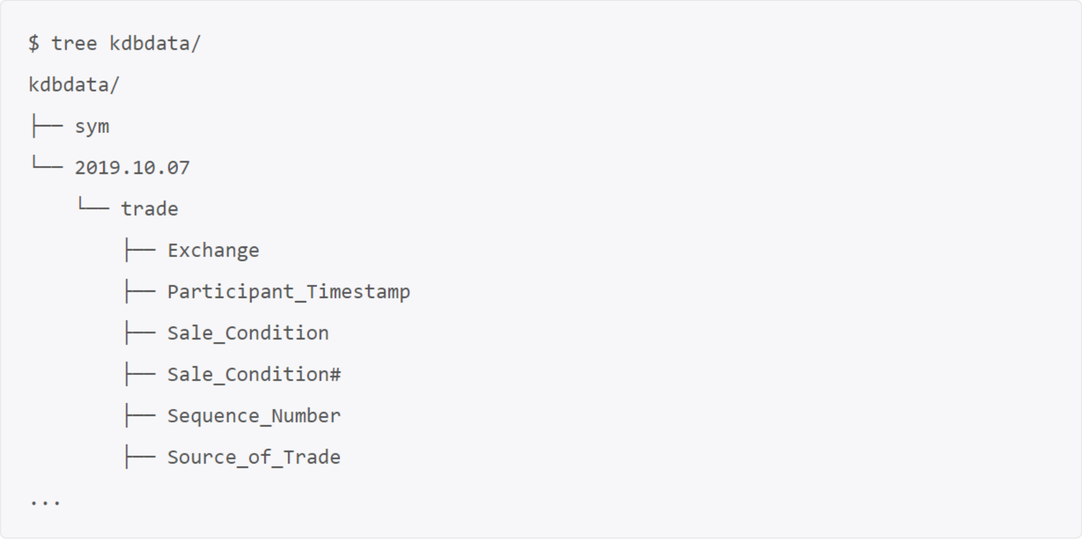


```
$ tree arrowdata/
arrowdata/
├── 20191007_trade.arrow
...
```

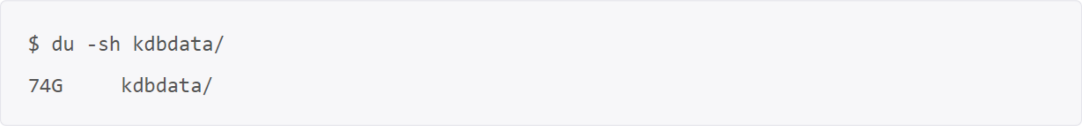


```
$ du -sh arrowdata/
123G    arrowdata/
```

kdb+



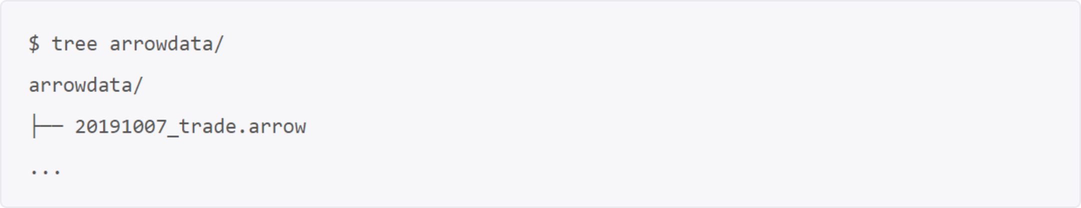
```
$ tree kdbdata/
kdbdata/
├── sym
└── 2019.10.07
    └── trade
        ├── Exchange
        ├── Participant_Timestamp
        ├── Sale_Condition
        ├── Sale_Condition#
        ├── Sequence_Number
        ├── Source_of_Trade
...
```



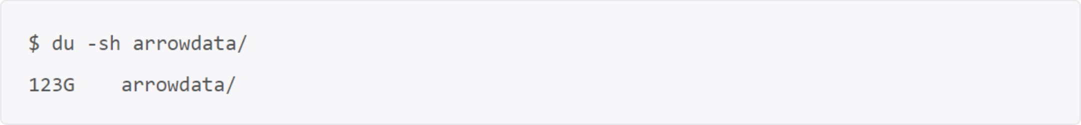
```
$ du -sh kdbdata/
74G     kdbdata/
```

Vaex, Arrow and kdb+ - data on disk

vaex/arrow

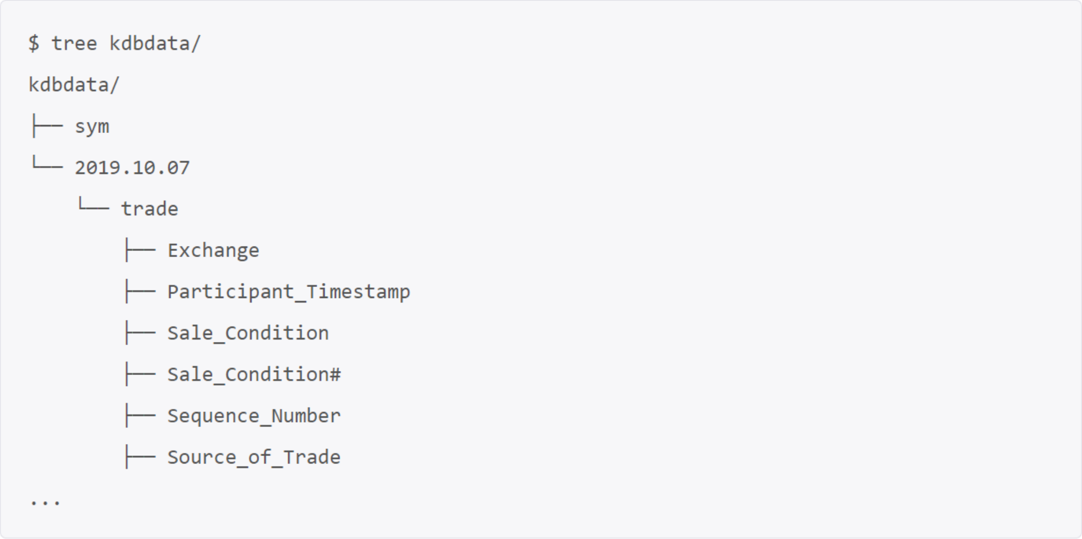


```
$ tree arrowdata/
arrowdata/
├── 20191007_trade.arrow
...
```

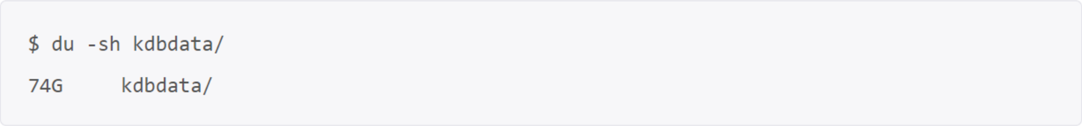


```
$ du -sh arrowdata/
123G    arrowdata/
```

kdb+



```
$ tree kdbdata/
kdbdata/
├── sym
└── 2019.10.07
    └── trade
        ├── Exchange
        ├── Participant_Timestamp
        ├── Sale_Condition
        ├── Sale_Condition#
        ├── Sequence_Number
        ├── Source_of_Trade
...
```

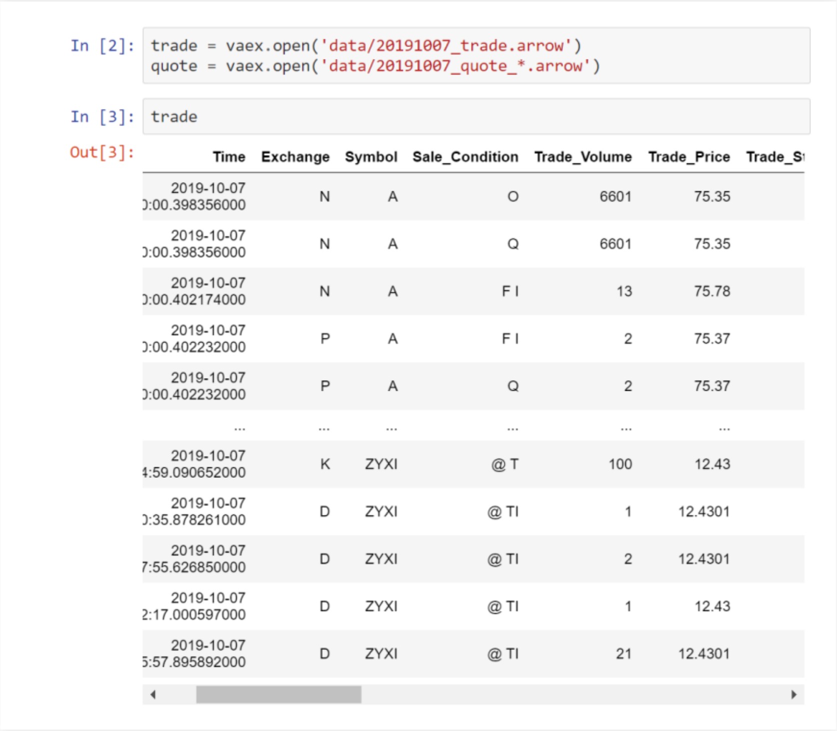


```
$ du -sh kdbdata/
74G     kdbdata/
```

- We aren’t using dictionaries with arrow here (size difference)
- One file vs. splay
- “p” attribute on Symbol for kdb+

Vaex, Arrow and kdb+ - loading data

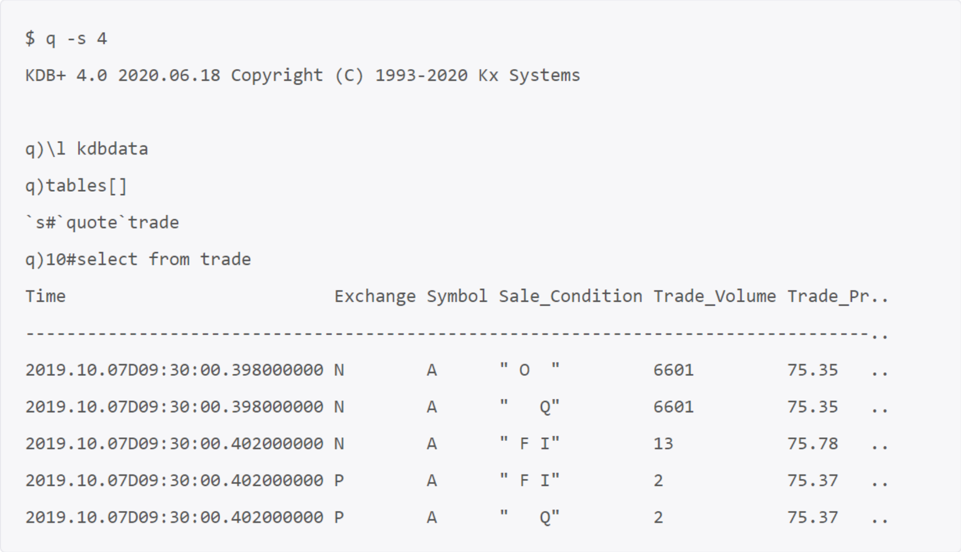
vaex/arrow



```
In [2]: trade = vaex.open('data/20191007_trade.arrow')
        quote = vaex.open('data/20191007_quote_*.arrow')

In [3]: trade
Out[3]:
```

Time	Exchange	Symbol	Sale_Condition	Trade_Volume	Trade_Price	Trade_S
2019-10-07 :00.398356000	N	A	O	6601	75.35	
2019-10-07 :00.398356000	N	A	Q	6601	75.35	
2019-10-07 :00.402174000	N	A	F I	13	75.78	
2019-10-07 :00.402232000	P	A	F I	2	75.37	
2019-10-07 :00.402232000	P	A	Q	2	75.37	
...	...	...	...	...	...	
2019-10-07 :59.090652000	K	ZYXI	@ T	100	12.43	
2019-10-07 :35.878261000	D	ZYXI	@ TI	1	12.4301	
2019-10-07 :55.626850000	D	ZYXI	@ TI	2	12.4301	
2019-10-07 :17.000597000	D	ZYXI	@ TI	1	12.43	
2019-10-07 :57.895892000	D	ZYXI	@ TI	21	12.4301	

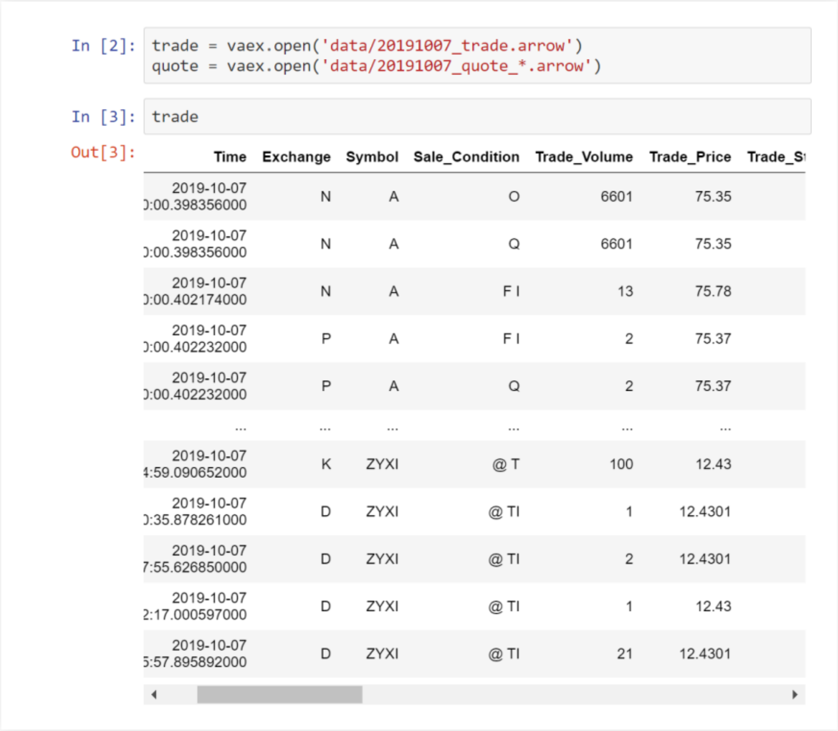
kdb+



```
$ q -s 4
KDB+ 4.0 2020.06.18 Copyright (C) 1993-2020 Kx Systems

q)\l kdbdata
q)tables[]
`s#`quote`trade
q)10#select from trade
Time                          Exchange Symbol Sale_Condition Trade_Volume Trade_Pr..
----------------------------------------------------------------------------------..
2019.10.07D09:30:00.398000000 N        A      " O  "         6601         75.35   ..
2019.10.07D09:30:00.398000000 N        A      "   Q"         6601         75.35   ..
2019.10.07D09:30:00.402000000 N        A      " F I"         13           75.78   ..
2019.10.07D09:30:00.402000000 P        A      " F I"         2            75.37   ..
2019.10.07D09:30:00.402000000 P        A      "   Q"         2            75.37   ..
```

Vaex, Arrow and kdb+ - loading data

vaex/arrow

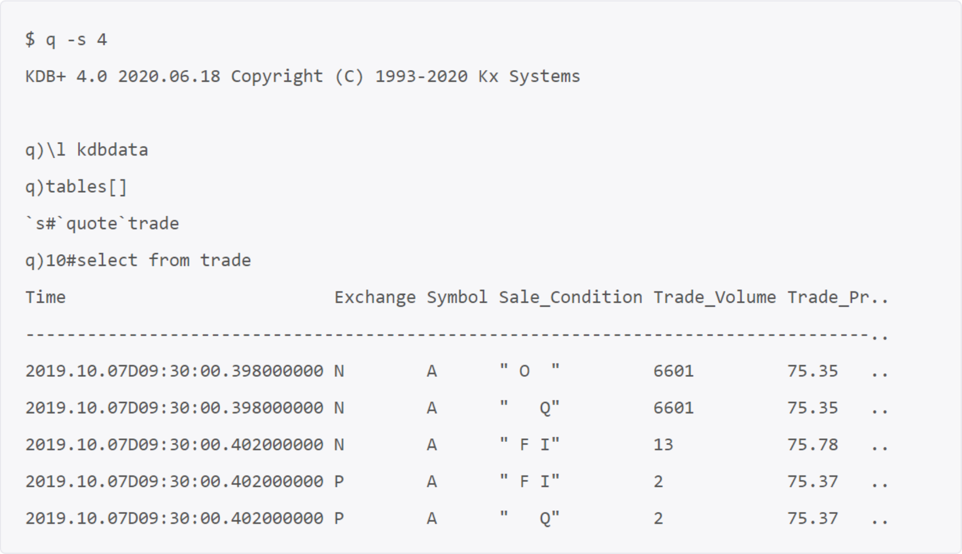


```
In [2]: trade = vaex.open('data/20191007_trade.arrow')
        quote = vaex.open('data/20191007_quote_*.arrow')

In [3]: trade
```

Out[3]:

Time	Exchange	Symbol	Sale_Condition	Trade_Volume	Trade_Price	Trade_S
2019-10-07):00.398356000	N	A	O	6601	75.35	
2019-10-07):00.398356000	N	A	Q	6601	75.35	
2019-10-07):00.402174000	N	A	F I	13	75.78	
2019-10-07):00.402232000	P	A	F I	2	75.37	
2019-10-07):00.402232000	P	A	Q	2	75.37	
...	...	...	...	...	...	
2019-10-07 4:59.090652000	K	ZYXI	@ T	100	12.43	
2019-10-07):35.878261000	D	ZYXI	@ TI	1	12.4301	
2019-10-07 7:55.626850000	D	ZYXI	@ TI	2	12.4301	
2019-10-07 2:17.000597000	D	ZYXI	@ TI	1	12.43	
2019-10-07 5:57.895892000	D	ZYXI	@ TI	21	12.4301	

kdb+

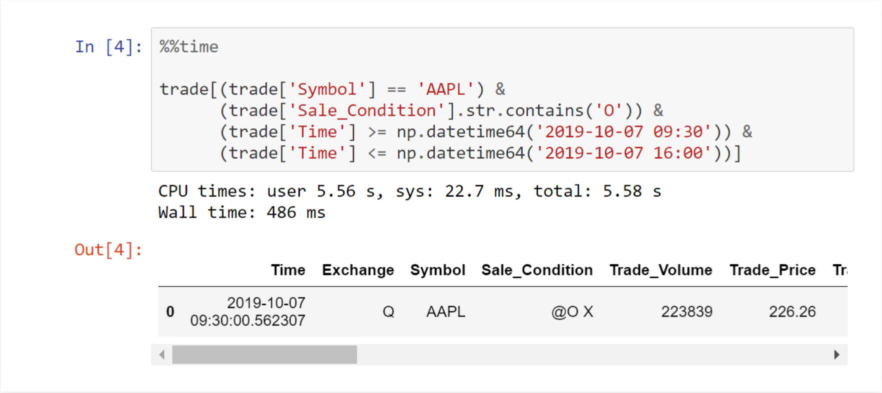


```
$ q -s 4
KDB+ 4.0 2020.06.18 Copyright (C) 1993-2020 Kx Systems

q)\l kdbdata
q)tables[]
`s#`quote`trade
q)10#select from trade
Time                          Exchange Symbol Sale_Condition Trade_Volume Trade_Pr..
-----------------------------------------------------------------------------------..
2019.10.07D09:30:00.398000000 N        A      " O  "         6601         75.35    ..
2019.10.07D09:30:00.398000000 N        A      "    Q"        6601         75.35    ..
2019.10.07D09:30:00.402000000 N        A      " F I"         13           75.78    ..
2019.10.07D09:30:00.402000000 P        A      " F I"         2            75.37    ..
2019.10.07D09:30:00.402000000 P        A      "    Q"        2            75.37    ..
```

- In both cases this is very fast (zero-copy, memory mapping magic) 🪄

Vaex, Arrow and kdb+ - querying data (filter)

vaex/arrow

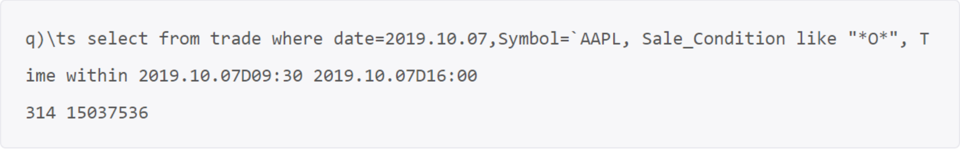


```
In [4]: %%time

        trade[(trade['Symbol'] == 'AAPL') &
              (trade['Sale_Condition'].str.contains('O')) &
              (trade['Time'] >= np.datetime64('2019-10-07 09:30')) &
              (trade['Time'] <= np.datetime64('2019-10-07 16:00'))]

        CPU times: user 5.56 s, sys: 22.7 ms, total: 5.58 s
        Wall time: 486 ms
```

Out[4]:

	Time	Exchange	Symbol	Sale_Condition	Trade_Volume	Trade_Price	Tr
0	2019-10-07 09:30:00.562307	Q	AAPL	@O X	223839	226.26	

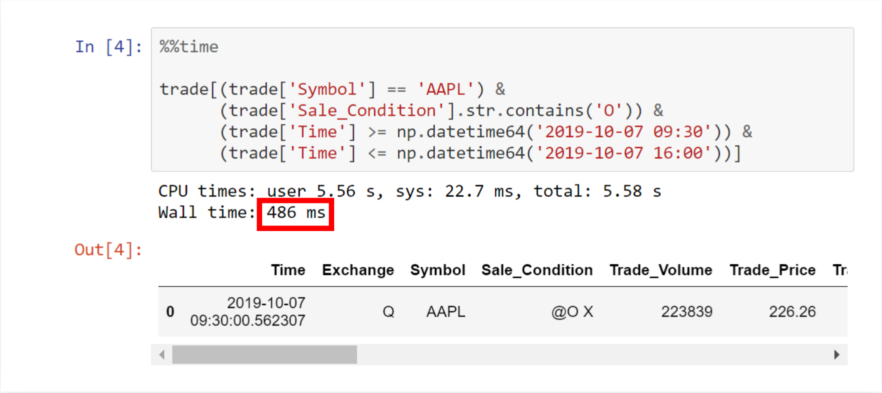
kdb+



```
q)\ts select from trade where date=2019.10.07,Symbol=`AAPL, Sale_Condition like "*O*", T
ime within 2019.10.07D09:30 2019.10.07D16:00
314 15037536
```

Vaex, Arrow and kdb+ - querying data (filter)

vaex/arrow



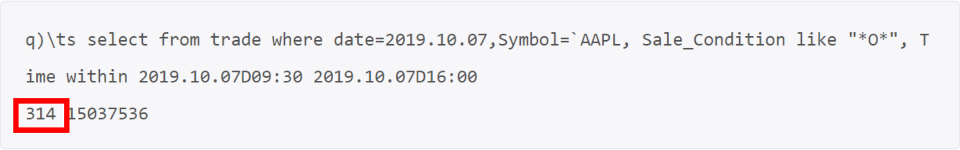
In [4]:
```
%%time

trade[(trade['Symbol'] == 'AAPL') &
      (trade['Sale_Condition'].str.contains('O')) &
      (trade['Time'] >= np.datetime64('2019-10-07 09:30')) &
      (trade['Time'] <= np.datetime64('2019-10-07 16:00'))]
```

```
CPU times: user 5.56 s, sys: 22.7 ms, total: 5.58 s
Wall time: 486 ms
```

Out[4]:

	Time	Exchange	Symbol	Sale_Condition	Trade_Volume	Trade_Price	Tr
0	2019-10-07 09:30:00.562307	Q	AAPL	@O X	223839	226.26	

kdb+



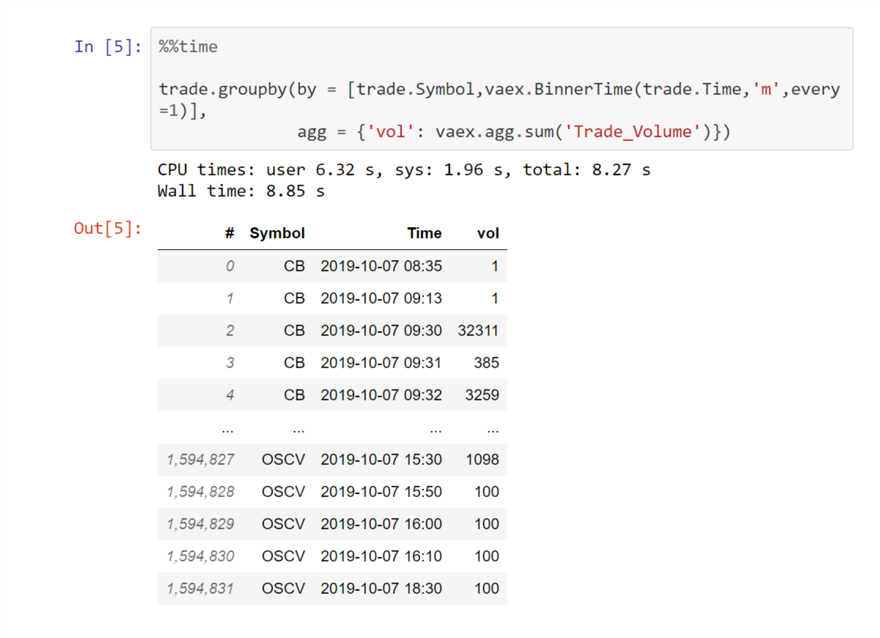
```
q)\ts select from trade where date=2019.10.07,Symbol=`AAPL, Sale_Condition like "*O*", T
ime within 2019.10.07D09:30 2019.10.07D16:00
314 15037536
```

- In both cases this is similarly fast. We're able to scan and filter, reading only the columns and part of columns we need

Vaex, Arrow and kdb+ - querying data (group)

vaex/arrow

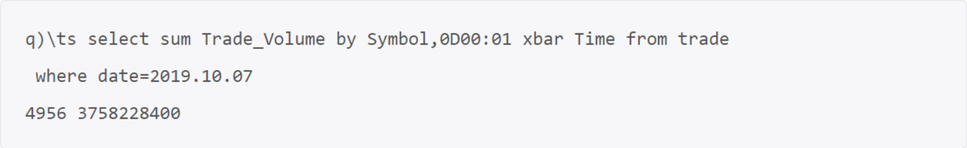
kdb+



```
In [5]: %%time

        trade.groupby(by = [trade.Symbol,vaex.BinnerTime(trade.Time,'m',every
        =1)],
                      agg = {'vol': vaex.agg.sum('Trade_Volume')})

        CPU times: user 6.32 s, sys: 1.96 s, total: 8.27 s
        Wall time: 8.85 s
```

Out[5]:

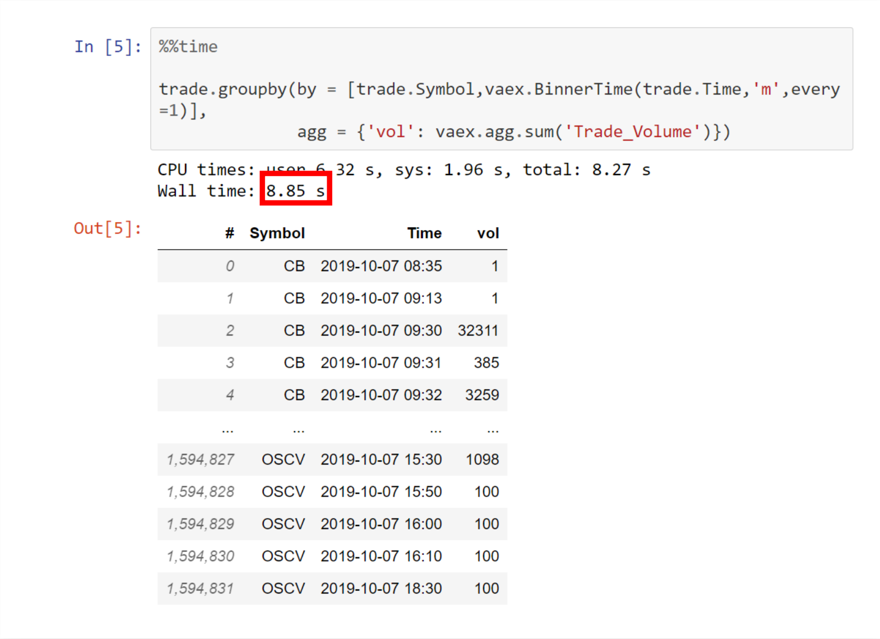
#	Symbol	Time	vol
0	CB	2019-10-07 08:35	1
1	CB	2019-10-07 09:13	1
2	CB	2019-10-07 09:30	32311
3	CB	2019-10-07 09:31	385
4	CB	2019-10-07 09:32	3259
...	...	...	...
1,594,827	OSCV	2019-10-07 15:30	1098
1,594,828	OSCV	2019-10-07 15:50	100
1,594,829	OSCV	2019-10-07 16:00	100
1,594,830	OSCV	2019-10-07 16:10	100
1,594,831	OSCV	2019-10-07 18:30	100



```
q)\ts select sum Trade_Volume by Symbol,0D00:01 xbar Time from trade
 where date=2019.10.07
4956 3758228400
```

Vaex, Arrow and kdb+ - querying data (group)

vaex/arrow

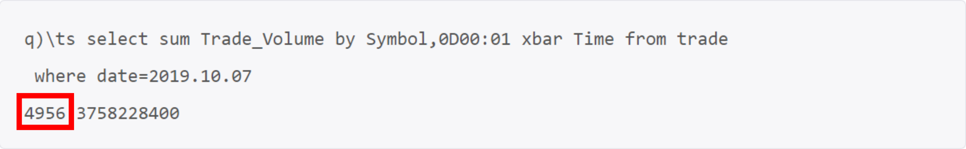


```
In [5]: %%time

        trade.groupby(by = [trade.Symbol,vaex.BinnerTime(trade.Time,'m',every
        =1)],
                      agg = {'vol': vaex.agg.sum('Trade_Volume')})

        CPU times: user 6.32 s, sys: 1.96 s, total: 8.27 s
        Wall time: 8.85 s
```

Out[5]:

#	Symbol	Time	vol
0	CB	2019-10-07 08:35	1
1	CB	2019-10-07 09:13	1
2	CB	2019-10-07 09:30	32311
3	CB	2019-10-07 09:31	385
4	CB	2019-10-07 09:32	3259
...	...	...	...
1,594,827	OSCV	2019-10-07 15:30	1098
1,594,828	OSCV	2019-10-07 15:50	100
1,594,829	OSCV	2019-10-07 16:00	100
1,594,830	OSCV	2019-10-07 16:10	100
1,594,831	OSCV	2019-10-07 18:30	100

kdb+



```
q)\ts select sum Trade_Volume by Symbol,0D00:01 xbar Time from trade
 where date=2019.10.07
4956 3758228400
```

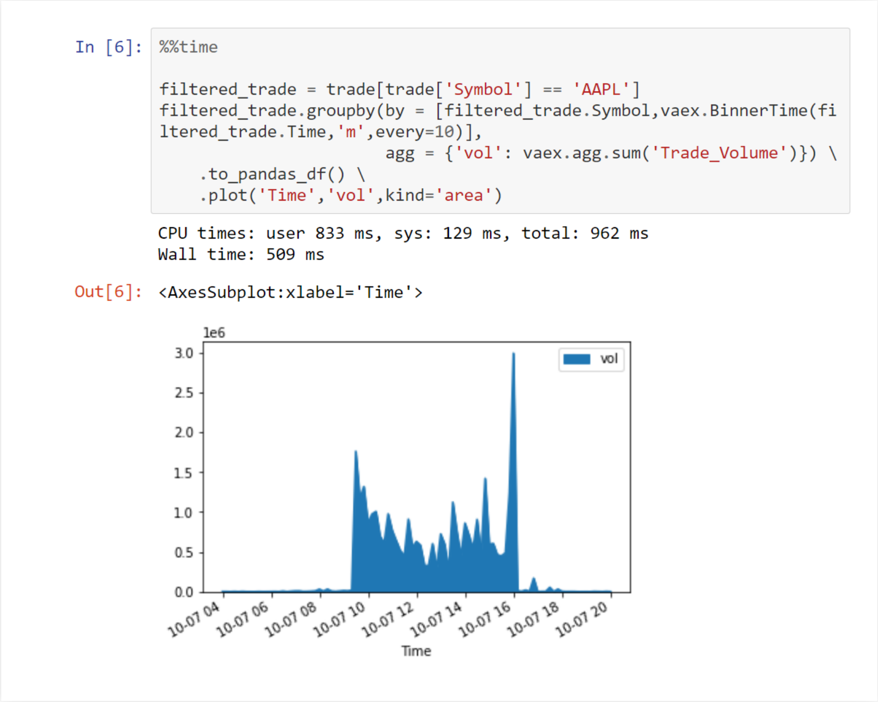
- Again performance is relatively similar

AQUAQ Analytics

Vaex, Arrow and kdb+ - querying data

vaex/arrow

kdb+



```
In [6]: %%time

        filtered_trade = trade[trade['Symbol'] == 'AAPL']
        filtered_trade.groupby(by = [filtered_trade.Symbol,vaex.BinnerTime(fi
        ltered_trade.Time,'m',every=10)],
                               agg = {'vol': vaex.agg.sum('Trade_Volume')}) \
            .to_pandas_df() \
            .plot('Time','vol',kind='area')

        CPU times: user 833 ms, sys: 129 ms, total: 962 ms
        Wall time: 509 ms

Out[6]: <AxesSubplot:xlabel='Time'>
```

- Comes with some other nice stuff out of the box

Comparison and possibilities

- Like any benchmark this should be taken with a pinch of salt
- So headline takeaway -> Vaex/arrow is closer to kdb+ hdb than you might think.
- Kdb+ is ~30 years old at this point so it’s obviously much more mature
- However the fundamental advantages of columnar storage and zero-copy are the same, so this gap will probably continue to close

Comparison and possibilities

Vaex/Arrow is closer than you might think

- **Shortcomings**:
  - Dictionaries/enums not fully supported yet
- **Possibilities/advantages**:

worth noting that in the comparisons vaex was using strings, while kdb+ used enums

Comparison and possibilities

Vaex/Arrow is closer than you might think

- **Shortcomings**:
  - Dictionaries/enums not fully supported yet
  - Partitioning support not great yet

Only fully supports Parquet partitioning for now e.g.

```
dataset_name/
  year=2007/
    month=01/
       data0.parquet
       data1.parquet
       ...
    month=02/
       data0.parquet
       data1.parquet
       ...
    month=03/
    ...
  year=2008/
    month=01/
    ...
```

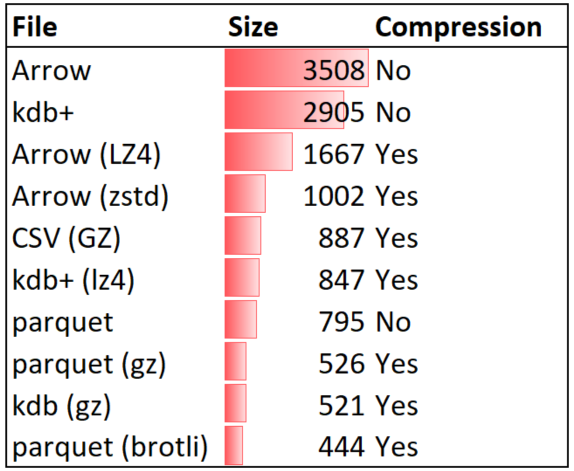
- **Possibilities/advantages**:

Comparison and possibilities

Vaex/Arrow is closer than you might think

- **Shortcomings**:
  - Dictionaries/enums not fully supported yet
  - Partitioning support not great yet
  - Compression (incoming AquaQ blog by Michael Turkington on this topic!)

- **Possibilities/advantages**:

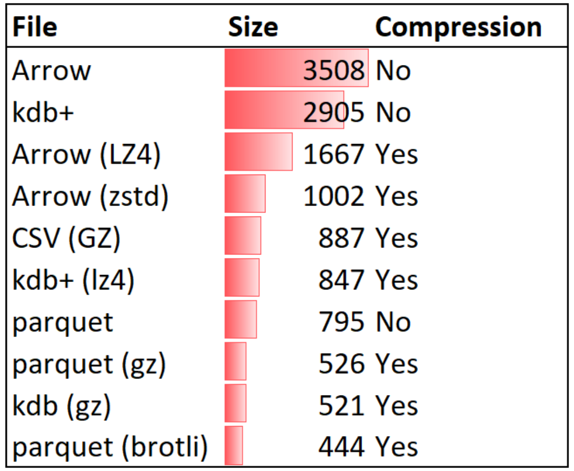


File	Size	Compression
Arrow	3508	No
kdb+	2905	No
Arrow (LZ4)	1667	Yes
Arrow (zstd)	1002	Yes
CSV (GZ)	887	Yes
kdb+ (lz4)	847	Yes
parquet	795	No
parquet (gz)	526	Yes
kdb (gz)	521	Yes
parquet (brotli)	444	Yes

Comparison and possibilities

Vaex/Arrow is closer than you might think

- **Shortcomings**:
  - Dictionaries/enums not fully supported yet
  - Partitioning support not great yet
  - ~~Compression~~ https://github.com/vaexio/vaex/pull/1078
- **Possibilities/advantages**:



File	Size	Compression
Arrow	3508	No
kdb+	2905	No
Arrow (LZ4)	1667	Yes
Arrow (zstd)	1002	Yes
CSV (GZ)	887	Yes
kdb+ (lz4)	847	Yes
parquet	795	No
parquet (gz)	526	Yes
kdb (gz)	521	Yes
parquet (brotli)	444	Yes

Comparison and possibilities

Vaex/Arrow is closer than you might think

- **Shortcomings**:
  - Dictionaries/enums not fully supported yet
  - Partitioning support not great yet
  - ~~Compression~~
  - Lack of maturity (in comparison to kdb+)
- **Possibilities/advantages**:

Comparison and possibilities

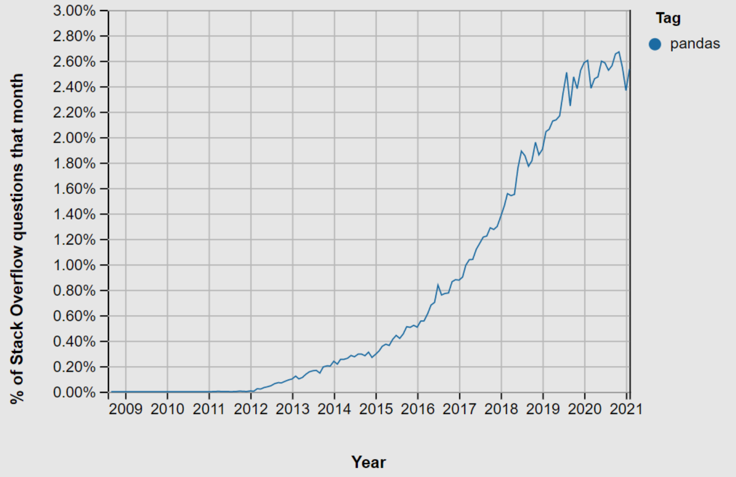
Vaex/Arrow is closer than you might think

- **Shortcomings**:
  - Dictionaries/enums not fully supported yet
  - Partitioning support not great yet
  - ~~Compression~~
  - Lack of maturity (in comparison to kdb+)

- **Possibilities/advantages**:
  - The API looks like pandas

stackoverflow

- ***kdb*** *– 1,760 questions*
- ***pandas*** *– 192,362 questions*



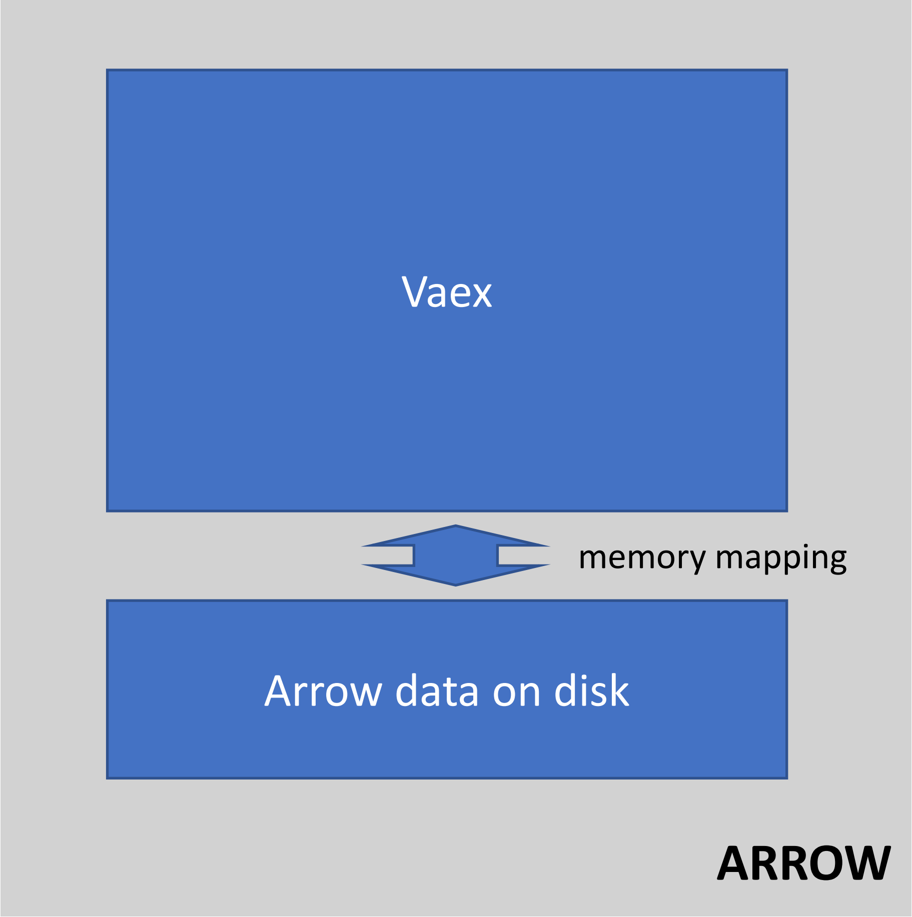
Comparison and possibilities

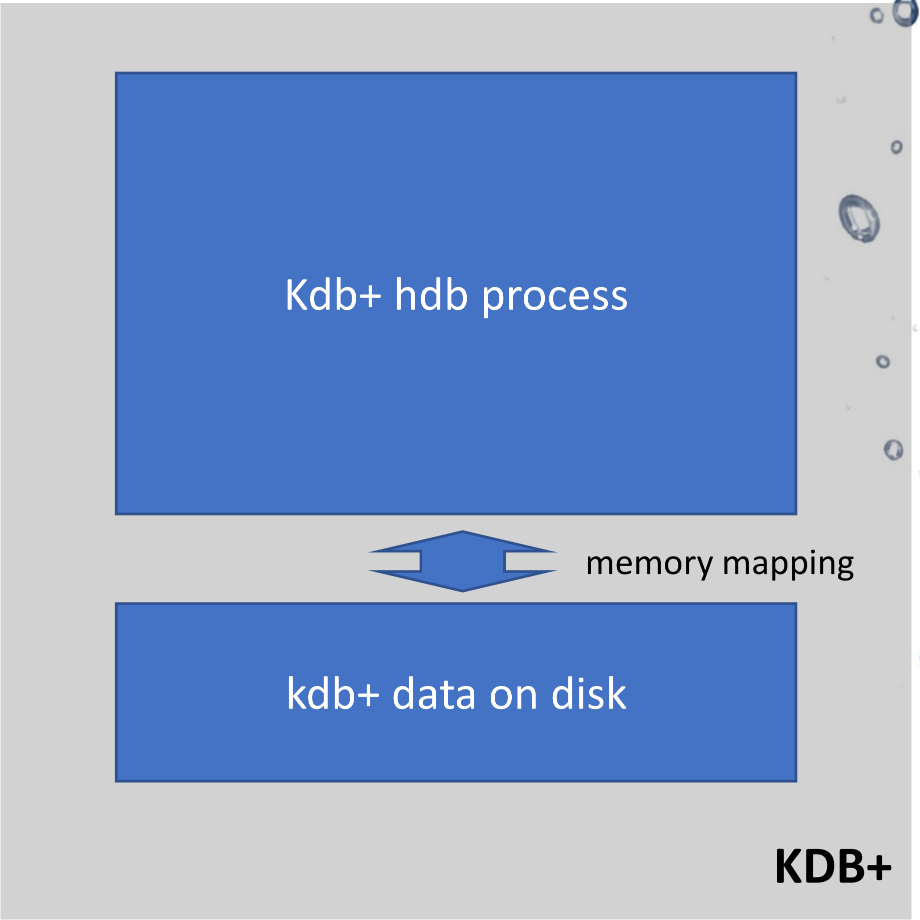
Vaex/Arrow is closer than you might think

- **Shortcomings**:
  - Dictionaries/enums not fully supported yet
  - Partitioning support not great yet
  - ~~Compression~~
  - Lack of maturity (in comparison to kdb+)

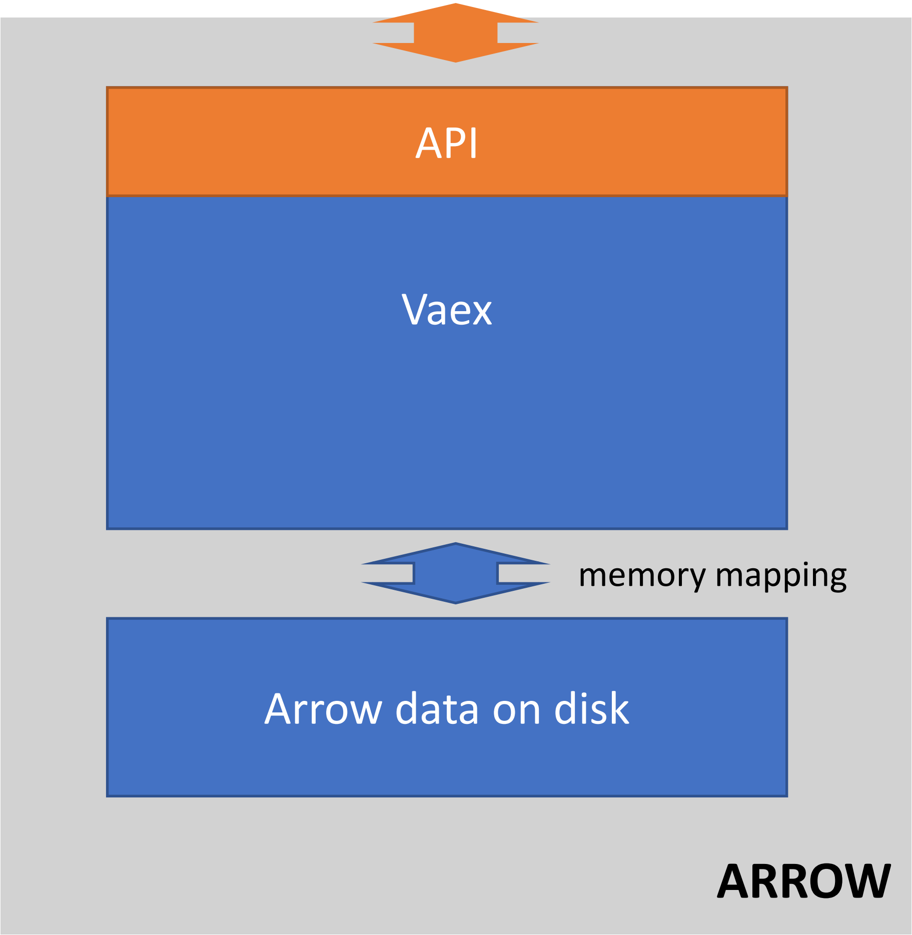
- **Possibilities/advantages**:
  - The API looks like pandas
  - Open and free! (Apache 2.0 and MIT licence) Let's talk about this one a little more…

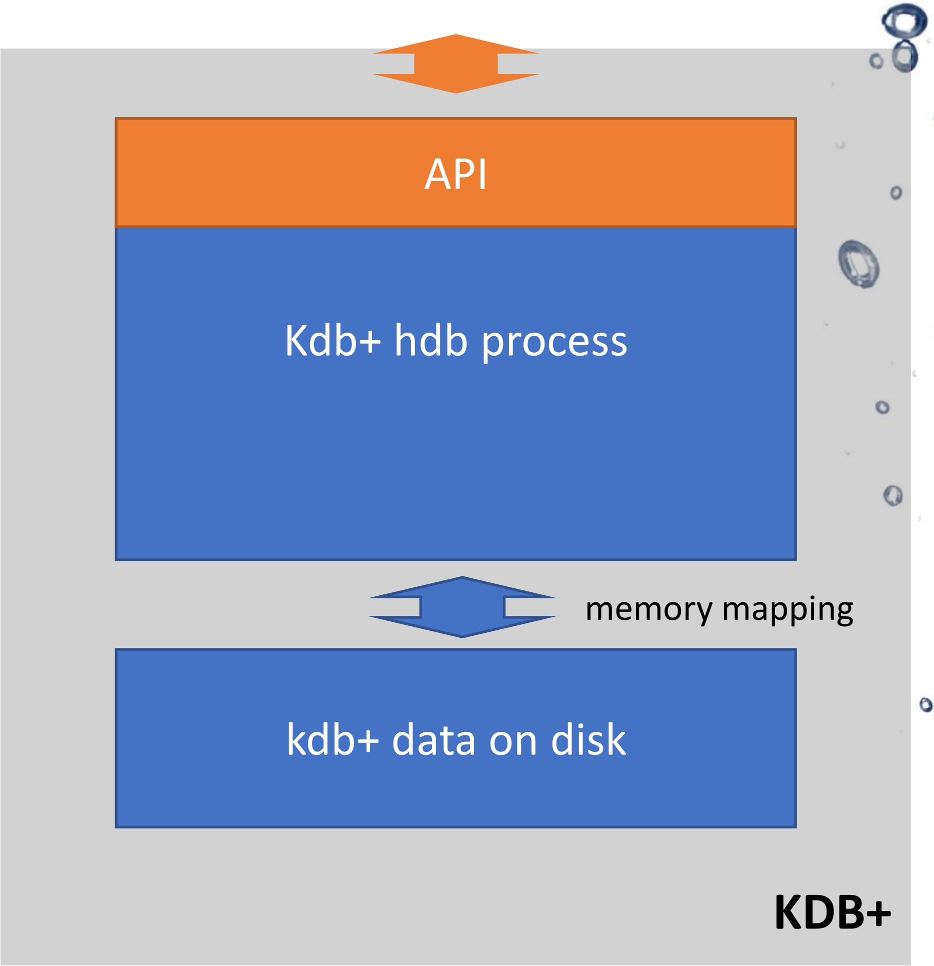
Comparison and possibilities



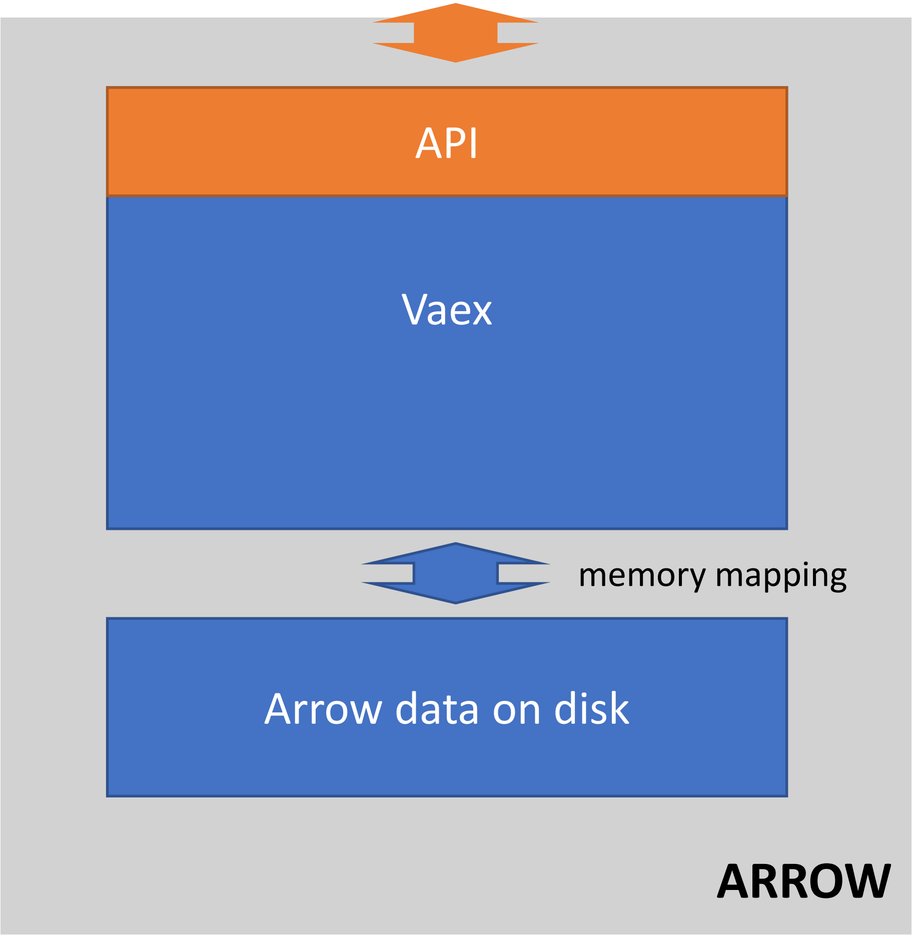


Comparison and possibilities

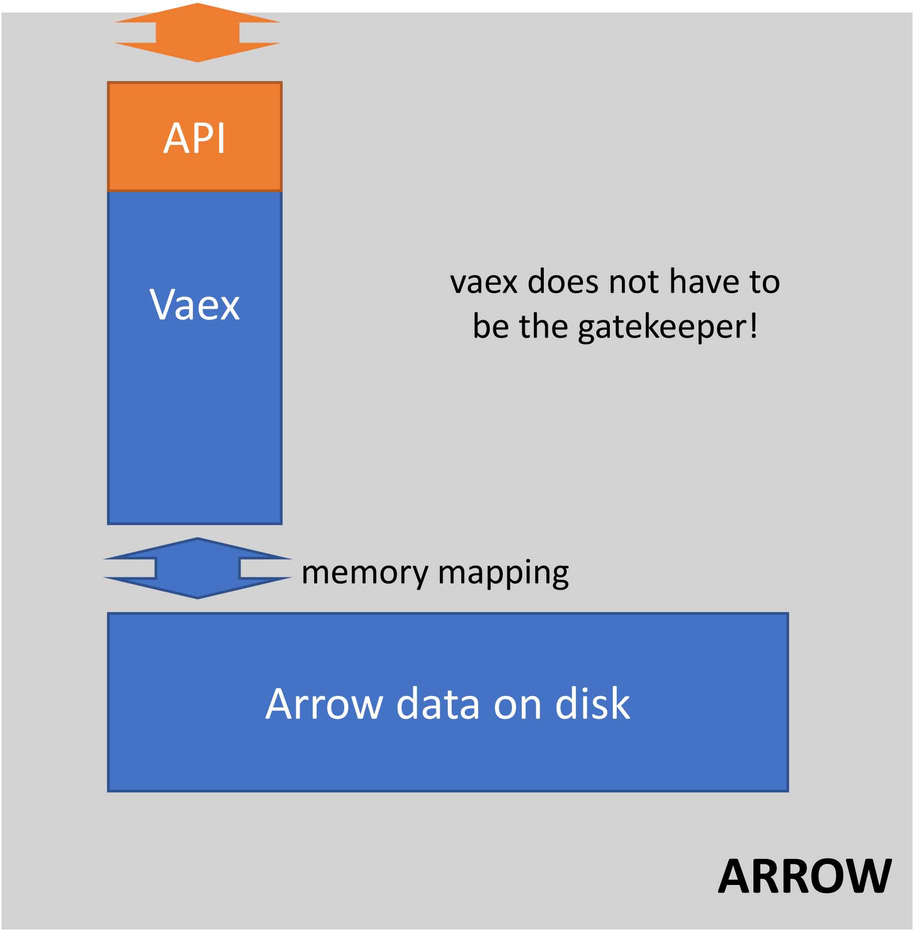




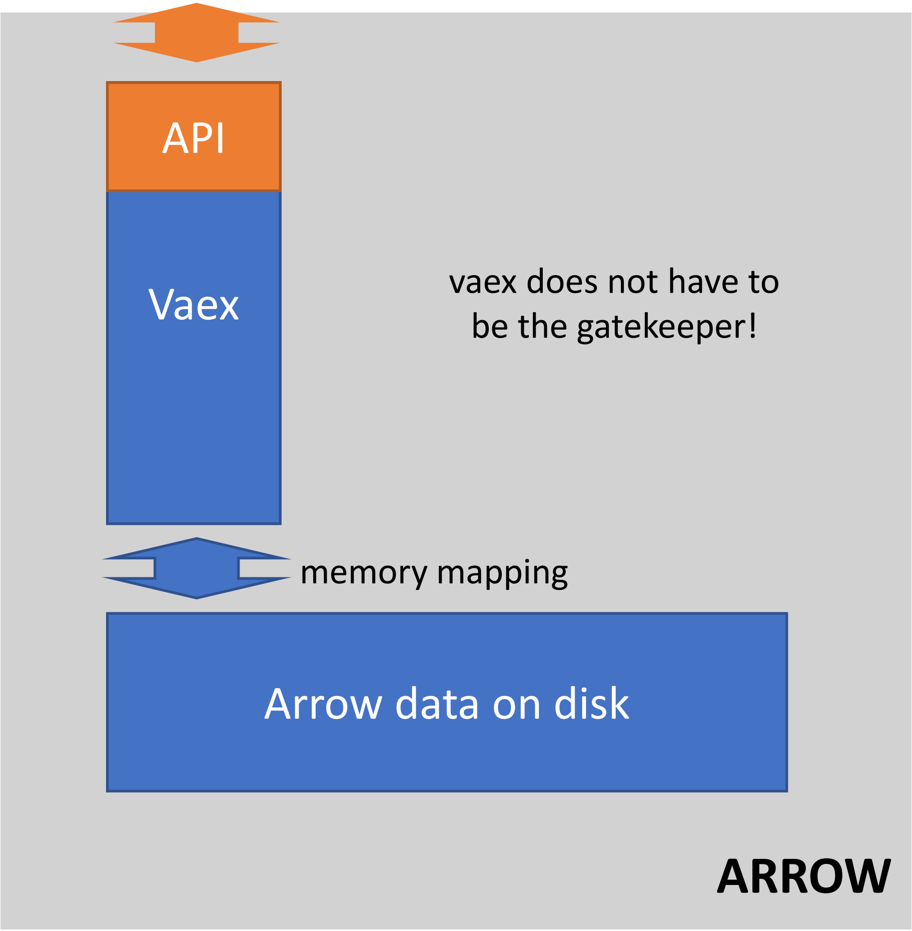
Comparison and possibilities



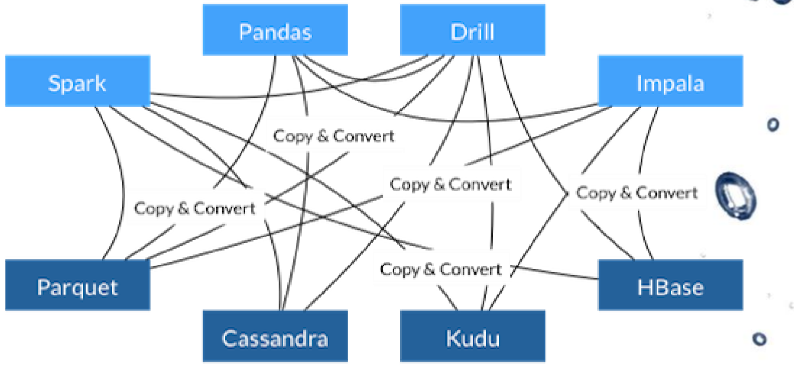
Comparison and possibilities

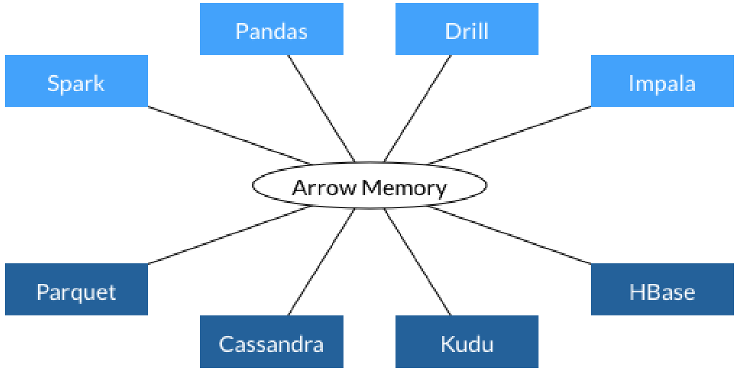


Comparison and possibilities

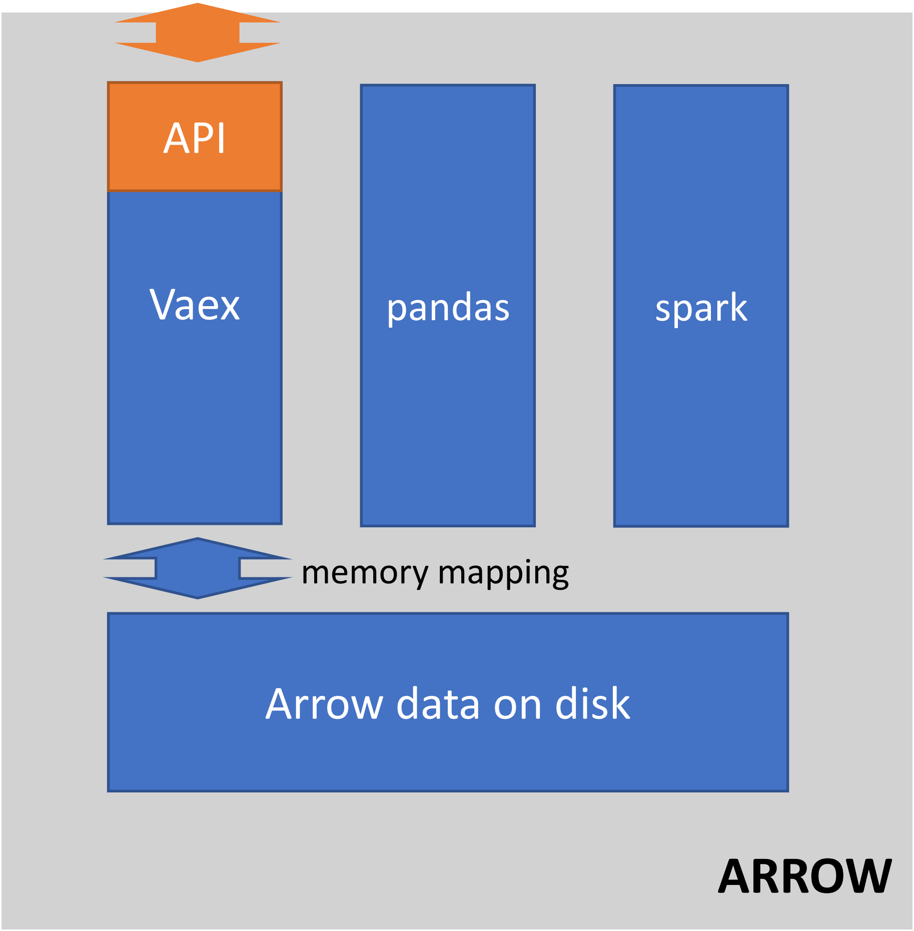


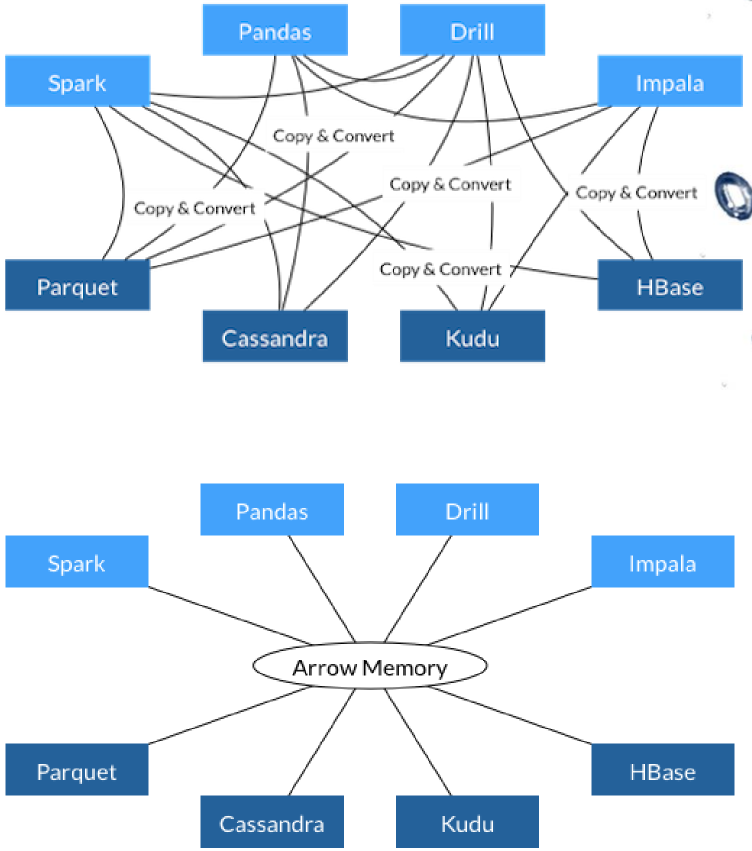
vaex does not have to be the gatekeeper!



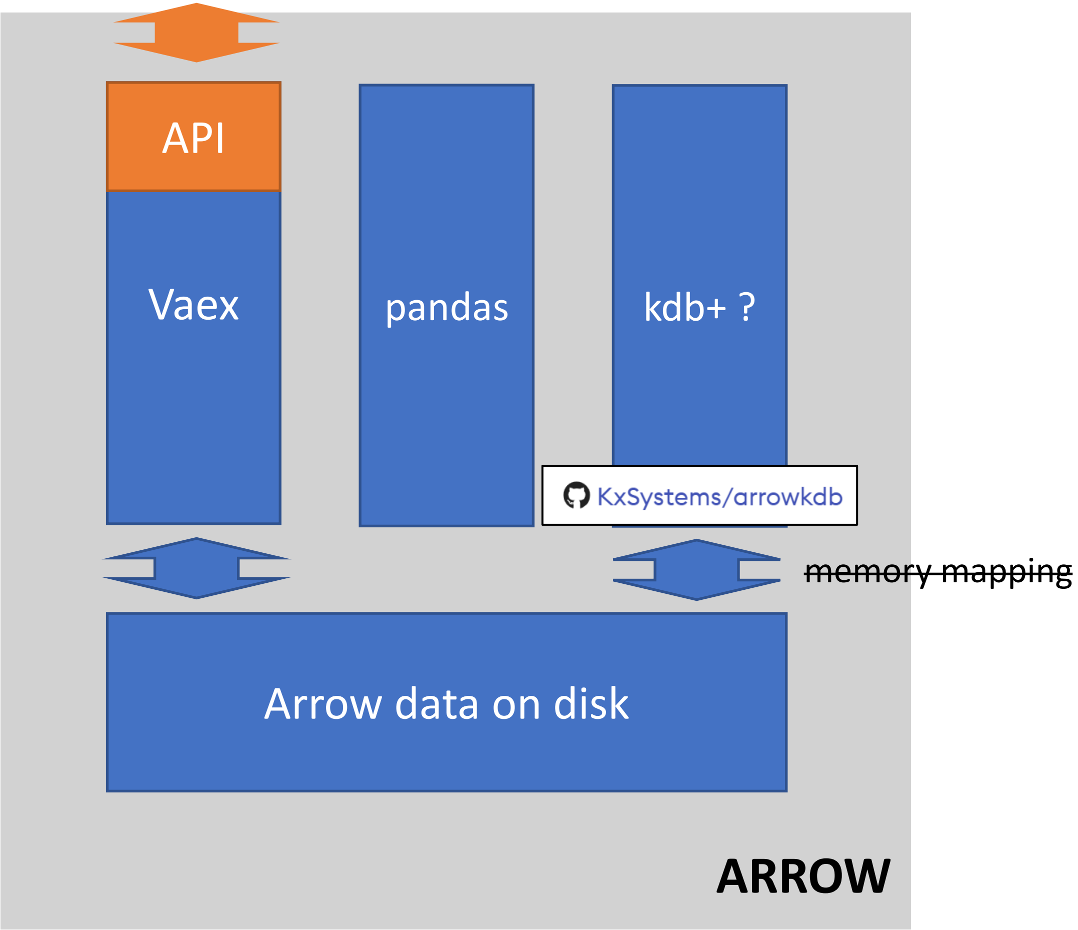


Comparison and possibilities





Comparison and possibilities



API

Vaex

pandas

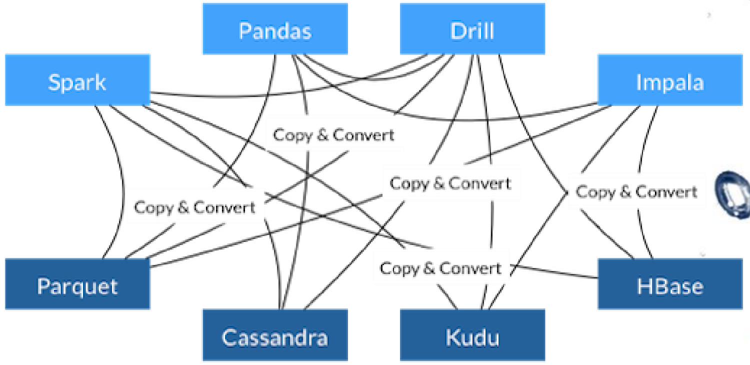
kdb+ ?

KxSystems/arrowkdb

~~memory mapping~~

Arrow data on disk

ARROW



Pandas

Drill

Spark

Impala

Copy & Convert

Copy & Convert

Copy & Convert

Copy & Convert

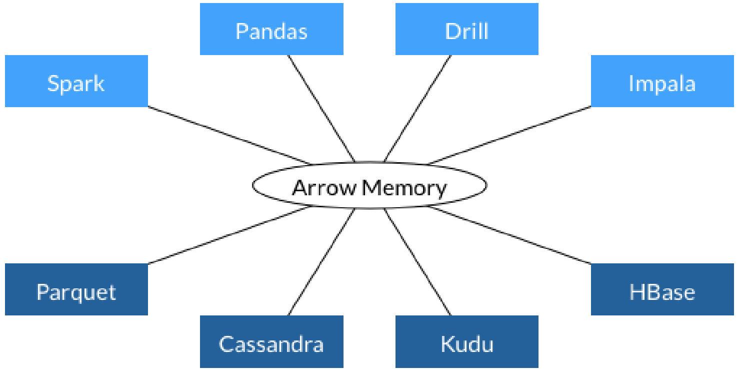
Copy & Convert

Parquet

Cassandra

Kudu

HBase



Pandas

Drill

Spark

Impala

Arrow Memory

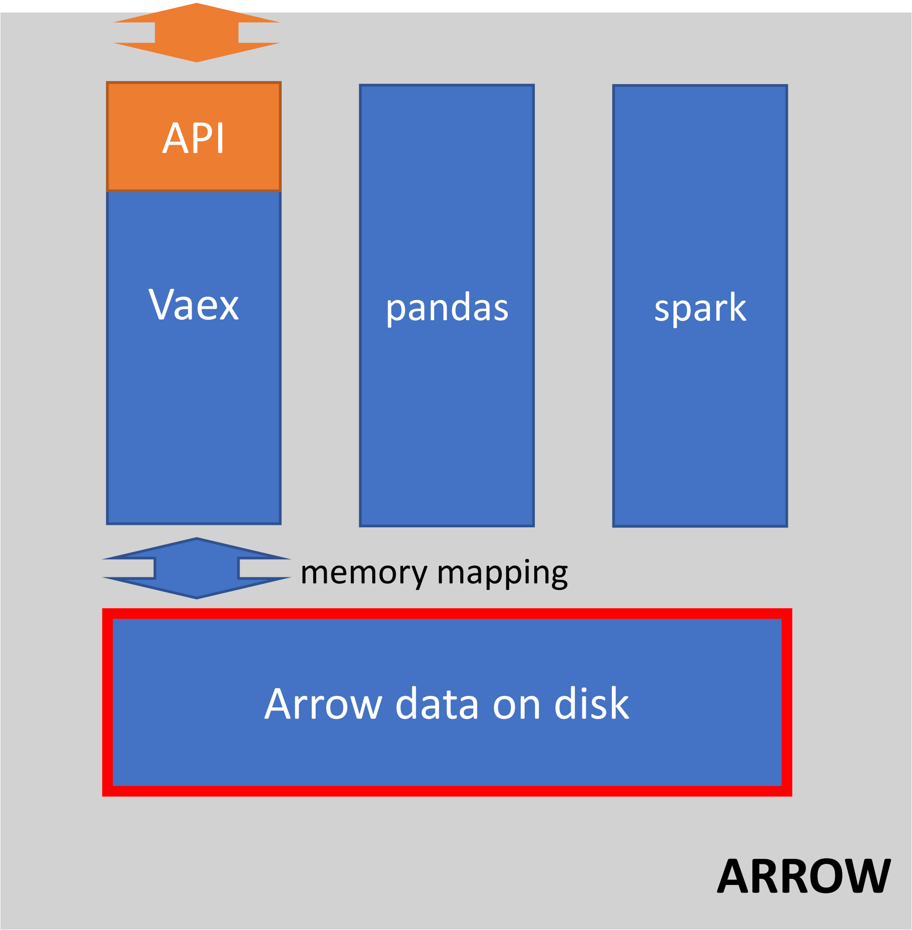
Parquet

Cassandra

Kudu

HBase

Comparison and possibilities

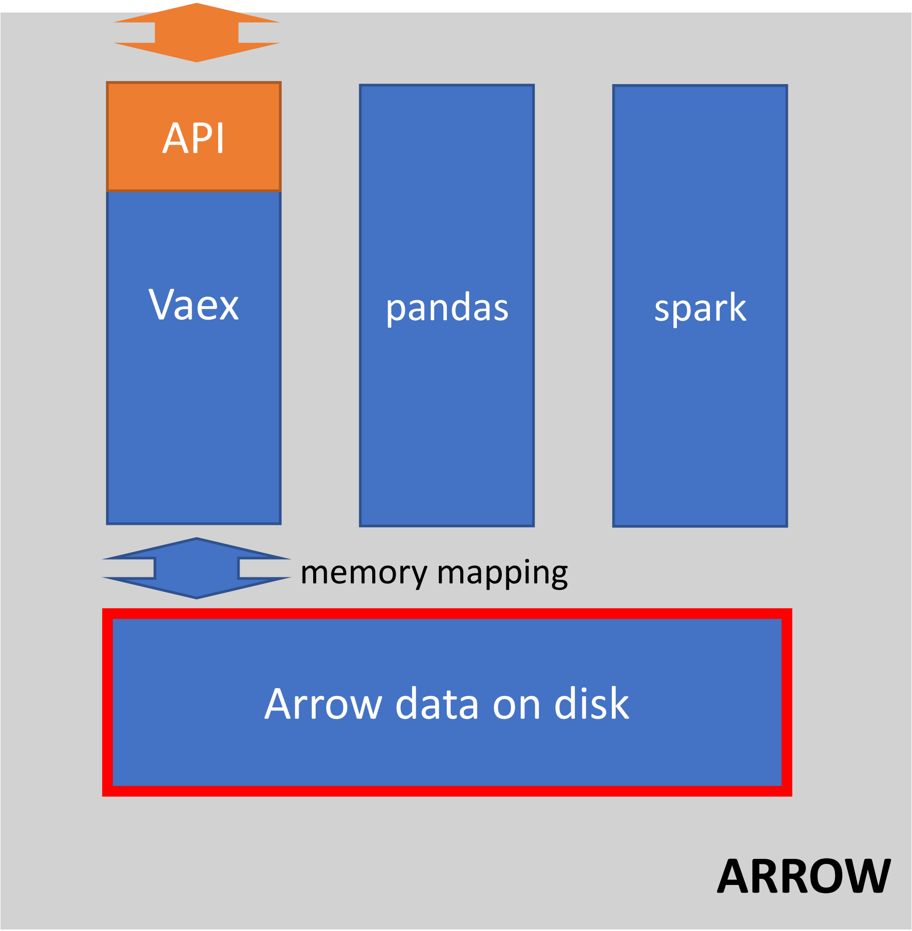


But wait, isn’t arrow an in-memory format?

Here (and in kdb+) we’re really using disk and memory mapping as a way to share memory locally

But doesn’t that mean it’s slow?

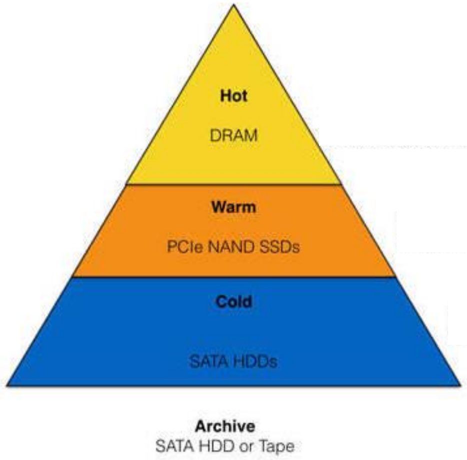
Comparison and possibilities



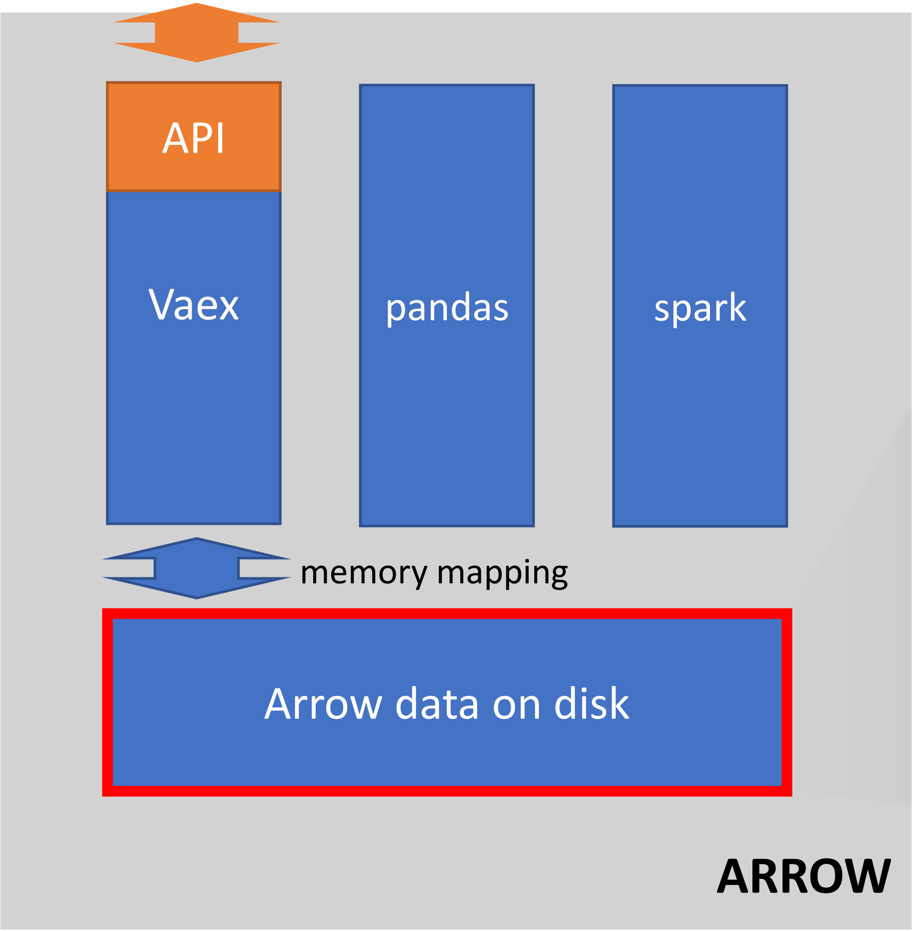
But wait, isn't arrow an in-memory format?

Here (and in kdb+) we're really using disk and memory mapping as a way to share memory locally

But doesn't that mean it's slow?

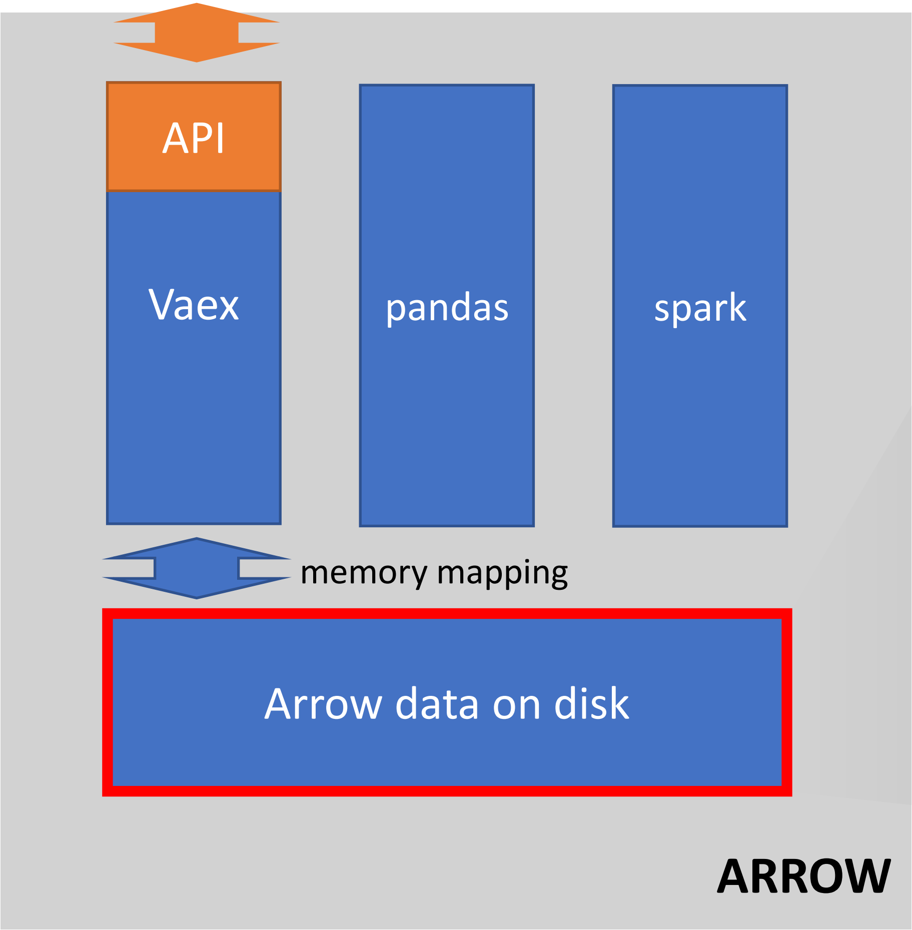


Comparison and possibilities



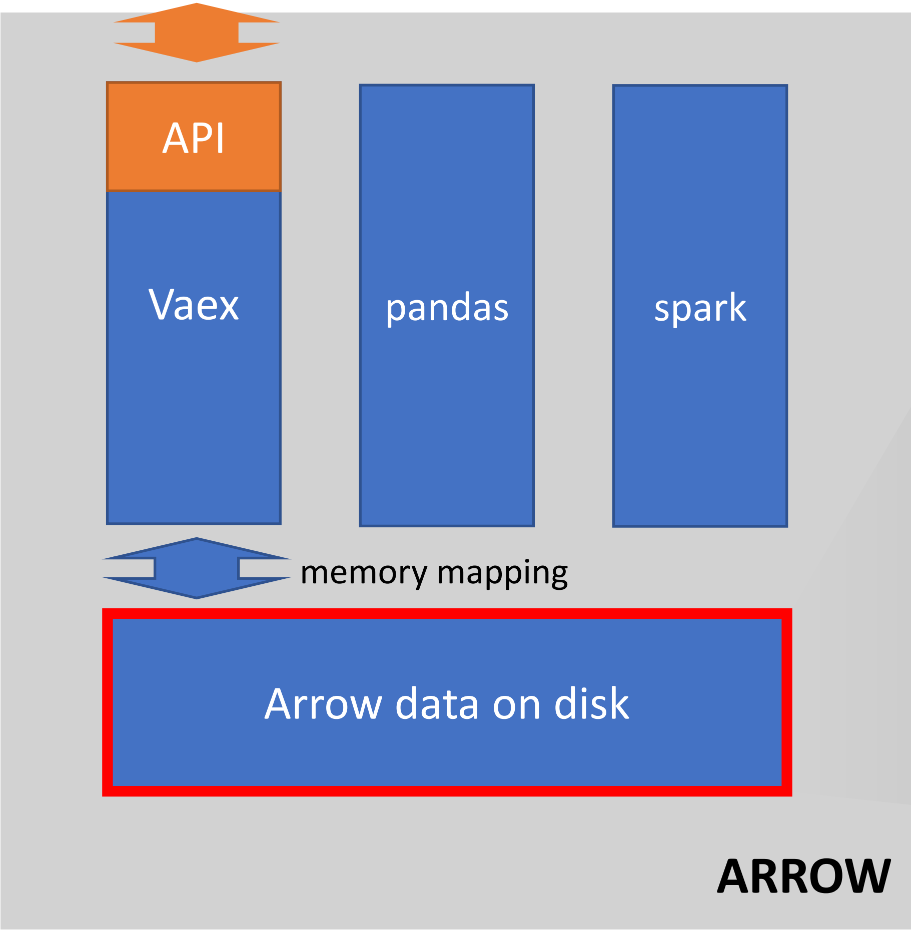
```
$ ls data/
…
2021.02.23
2021.02.24
2021.02.25
2021.02.26
2021.02.27
2021.02.28
2021.03.01
2021.03.02
2021.03.03
2021.03.04
2021.03.05
2021.03.06
2021.03.07
2021.03.08
2021.03.09
2021.03.10
2021.03.11
```

Comparison and possibilities

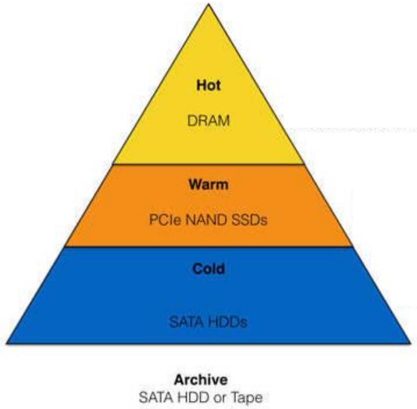


```
$ ls data/
…
2021.02.23
2021.02.24
2021.02.25
2021.02.26
2021.02.27
2021.02.28
2021.03.01
2021.03.02
2021.03.03
2021.03.04
2021.03.05
2021.03.06
2021.03.07
2021.03.08
2021.03.09
2021.03.10
2021.03.11
   ├── quote
   ├── trade
   └── etc.
```

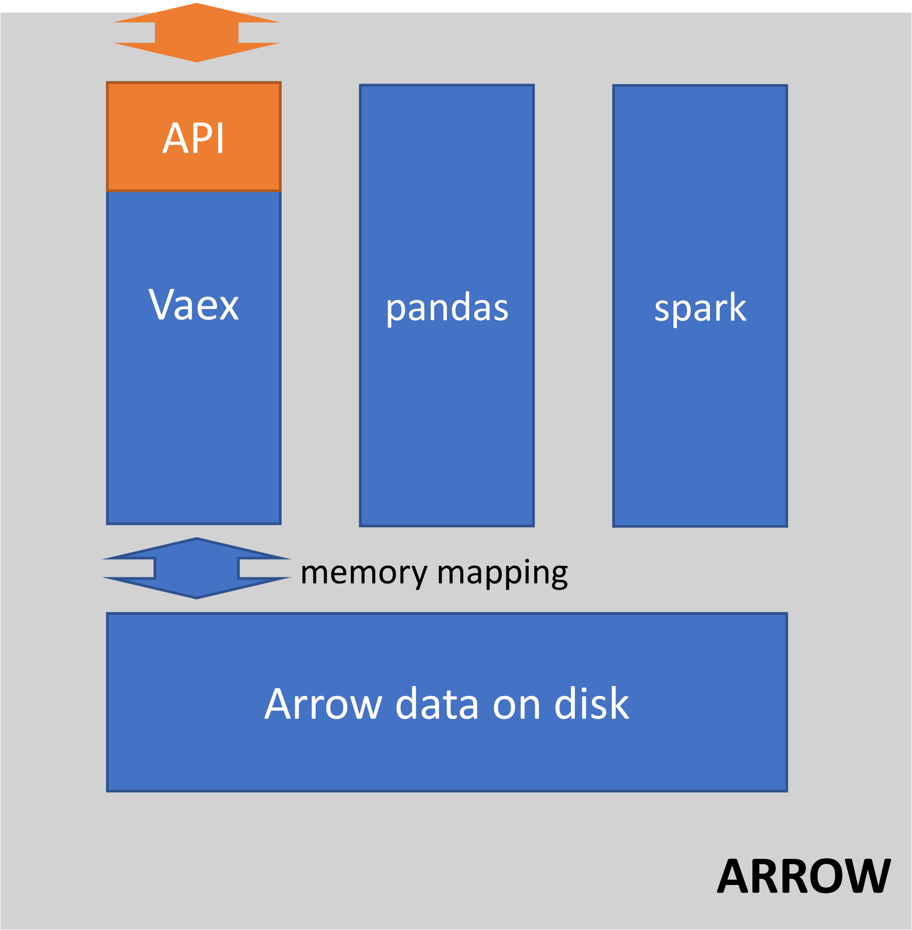
Comparison and possibilities



```
$ ls data/
…
2021.02.23
2021.02.24
2021.02.25
2021.02.26
2021.02.27
2021.02.28
2021.03.01
2021.03.02
2021.03.03
2021.03.04
2021.03.05
2021.03.06
2021.03.07
2021.03.08
2021.03.09
2021.03.10
2021.03.11 (RAMdisk)
```



Comparison and possibilities

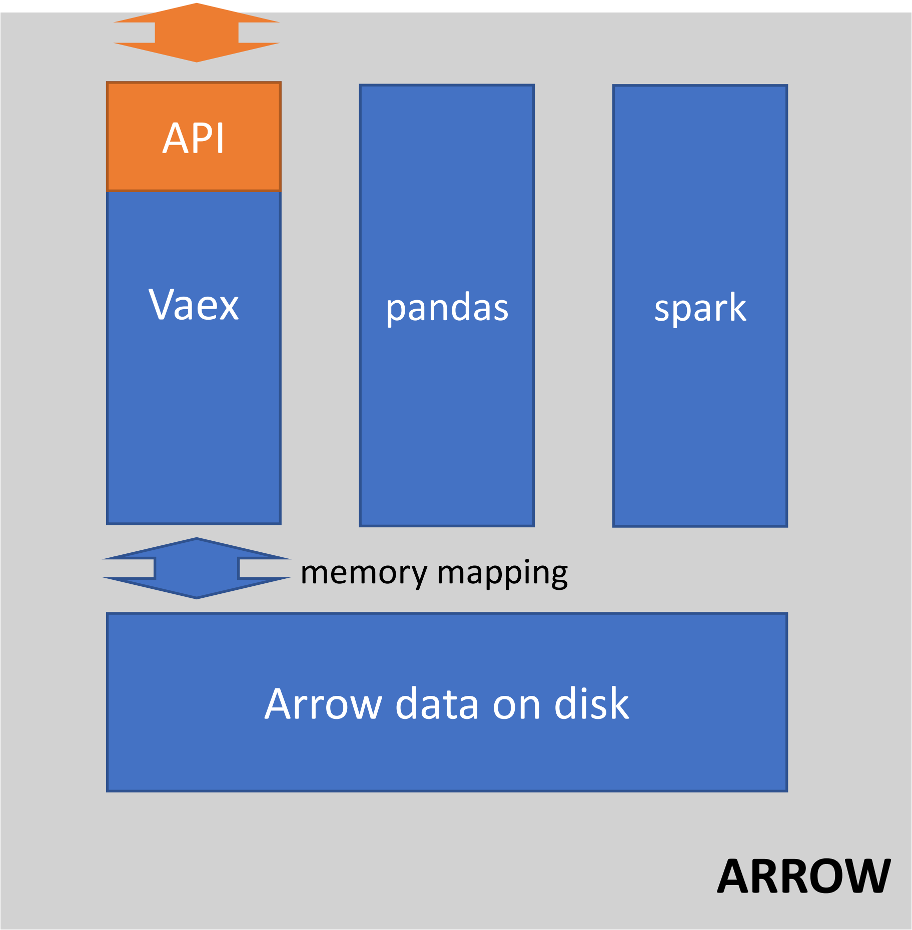


The Amazon API Mandate (2002)

> *1) All teams will henceforth expose their data and functionality through service interfaces.*
>
> *2) Teams must communicate with each other through these interfaces.*
>
> *3) There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.*
>
> *4) It doesn't matter what technology is used. HTTP, Corba, Pubsub, custom protocols — doesn't matter.*
>
> *5) All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.*
>
> *6) Anyone who doesn't do this will be fired.*

— JEFF BEZOS

Comparison and possibilities



“Data Lake” architecture rather than a database

Processes to create/manage these files, separate from users

This is just one suggestion, other architectures are available…

Takeaways

- Vaex and arrow combo is pretty nice:
    - Obviously doesn’t match 30 years of KDB maturity, but it’s closer than you might think
    - Performance difference measured in factors, rather than orders of magnitude
- Open nature and zero-serialization creates possibilities!
- Questions?