

# Лекция 5: Форматы данных (часть 2)

Автор: Сергей Вячеславович Макрушин e-mail: [SVMakrushin@fa.ru](mailto:SVMakrushin@fa.ru) (<mailto:SVMakrushin@fa.ru>)

Финансовый университет, 2020 г.

При подготовке лекции использованы материалы:

- Документация к рассмотренным пакетам

V 0.2 04.10.2021

## Разделы:

- [Хранение табличных данных](#)
- [CSV](#)
- [NPY](#)
- [HDF](#)
- [Apache Parquet](#) -
- [к оглавлению](#)

In [1]:

```
1 # загружаем стиль для оформления презентации
2 from IPython.display import HTML
3 from urllib.request import urlopen
4 html = urlopen("file:./lec_v1.css")
5 HTML(html.read().decode('utf-8'))
```

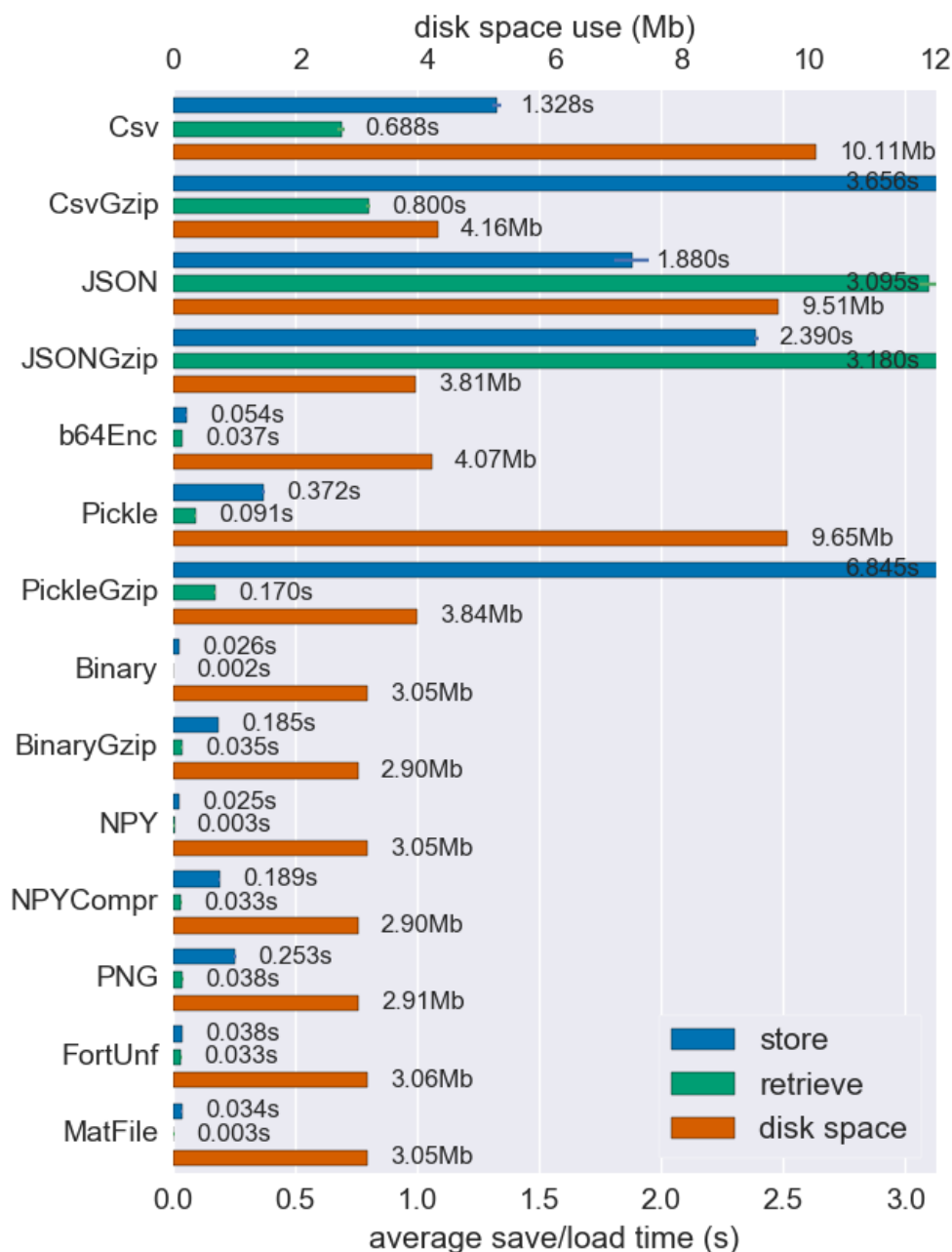
Out[1]:

## Хранение табличных данных

- [к оглавлению](#)

Разнообразие форматов хранения табличных данных

## Random array storage performance (1000x400, avg of 50x)



### Сравнение производительности при работе с различными форматами файлов

Сравнение:

- Plain-text CSV — a good old friend of a data scientist
- Pickle — a Python's way to serialize things
- MessagePack — it's like JSON but fast and small
- HDF5 — a file format designed to store and organize large amounts of data
- Feather — a fast, lightweight, and easy-to-use binary file format for storing data frames
- Parquet — an Apache Hadoop's columnar storage format

Результаты сравнения: <https://towardsdatascience.com/the-best-format-to-save-pandas-data-414dca023e0d>  
<https://towardsdatascience.com/the-best-format-to-save-pandas-data-414dca023e0d>

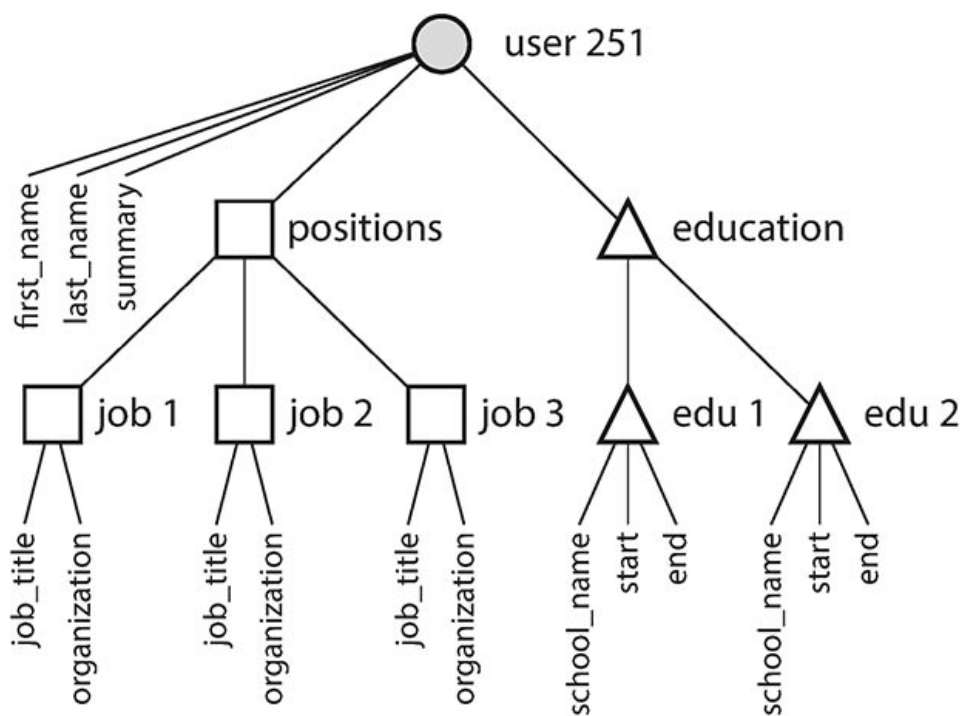
### Сложности представления иерархических данных в табличном виде

Пример описания в профессиональной соцсети в формате JSON:

```

{
  "user_id": 251,
  "first_name": "Bill",
  "last_name": "Gates",
  "summary": "Co-chair of the Bill & Melinda Gates... Active blogger.",
  "region_id": "us:91",
  "industry_id": 131,
  "photo_url": "/p/7/000/253/05b/308dd6e.jpg",
  "positions": [
    {"job_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},
    {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
  ],
  "education": [
    {"school_name": "Harvard University", "start": 1973, "end": 1975},
    {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
  ],
  "contact_info": {
    "blog": "http://thegatesnotes.com",
    "twitter": "http://twitter.com/BillGates"
  }
}

```



**Иерархическая структура документа**



**Bill Gates**

Greater Seattle Area | Philanthropy

**Summary**

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

**Experience**

Co-chair • Bill & Melinda Gates Foundation  
2000 – Present

Co-founder, Chairman • Microsoft  
1975 – Present

**Education**

Harvard University  
1973 – 1975

Lakeside School, Seattle

**Contact Info**

Blog: [thegatesnotes.com](http://thegatesnotes.com)  
Twitter: @BillGates

users table			
user_id	first_name	last_name	summary
251	Bill	Gates	Co-chair of ... blogger.
	region_id	industry_id	photo_id
	us:91	131	57817532

regions table	
id	region_name
us:7	Greater Boston Area
us:91	Greater Seattle Area

industries table	
id	industry_name
43	Financial Services
48	Construction
131	Philanthropy

positions table			
id	user_id	job_title	organization
458	251	Co-chair	Bill & Melinda Gates F...
457	251	Co-founder, Chairman	Microsoft

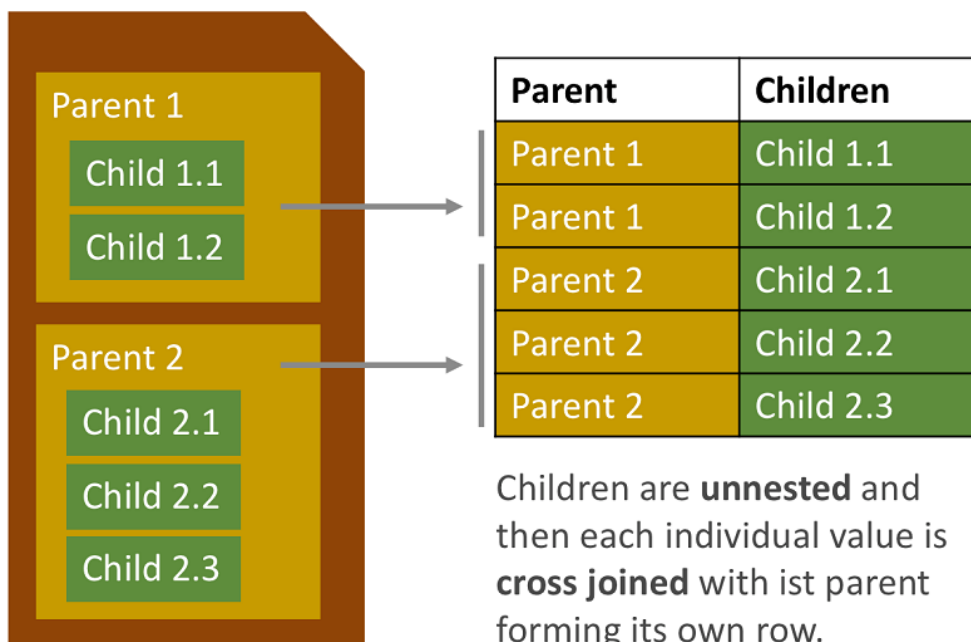
  

education table				
id	user_id	school_name	start	end
807	251	Harvard University	1973	1975
806	251	Lakeside School, Seattle	NULL	NULL

contact_info table			
id	user_id	type	url
155	251	blog	<a href="http://thegatesnotes.com">http://thegatesnotes.com</a>
156	251	twitter	<a href="http://twitter.com/BillGates">http://twitter.com/BillGates</a>

### Сравнение представления документа и реляционной модели



### Вариант решения проблемы представления иерархических данных в табличном виде

Другая потенциальная проблема:

- Динамическая схема документа.

CSV, numpy-native, parquet, xlsx, sqlite

# CSV

- [к оглавлению](#)

**CSV** (Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделённых запятыми.

- Формат CSV стандартизирован не полностью.
- Ключевые проблема: в табличных данных могут иметься запятые или переводы строк
- Популярным решением проблемы запятых и переносов строк является **заключение данных в кавычки**, однако исходные данные могут содержать кавычки.
- Термином CSV могут обозначаться похожие форматы, в которых разделителем является символ табуляции ( TSV ) или точка с запятой.
- Многие приложения, которые работают с форматом CSV, позволяют выбирать символ разделителя и символ кавычек.

Спецификация:

- Каждая **строка файла** — это одна строка таблицы.
- **Разделителем** (delimiter) значений колонок является символ запятой , . Однако на практике часто используются другие разделители, то есть формат путают с DSVruen и TSV (см. ниже).
- Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками ( " ). Если в значении встречаются кавычки — они представляются в файле в виде двух кавычек подряд ( "" ).

## Примеры

Пример 1:

```
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture «Extended Edition»","",4900.00
1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof, loaded",4799.00
```

1997	Ford	E350	ac, abs, moon	3000
1999	Chevy	Venture «Extended Edition»		4900
1996	Jeep	Grand Cherokee	MUST SELL! air, moon roof, loaded	4799

Результат обработки примера 1

Пример 2:

```
1965;Пиксель;E240 - формальдегид (опасный консервант)!;"красный, зелёный, биты
й";"3000,00"
1965;Мышка;"А правильней использовать ""Ёлочки""";;"4900,00"
"Н/д";Кнопка;Сочетания клавиш;"MUST USE! Ctrl, Alt, Shift";"4799,00"
```

1965	Пиксель	E240 — формальдегид (опасный консервант)!	красный, зелёный, битый	3000
1965	Мышка	А правильней использовать "Ёлочки"		4900
Н/д	Кнопка	Сочетания клавиш	MUST USE! Ctrl, Alt, Shift	4799

## Результат обработки примера 2

In [2]:

```
1 import csv
2 import pandas as pd
```

In [3]:

```
1 df0 = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]},
2                     index=['a', 'b', 'c'])
3
4 df0
```

Out[3]:

	A	B
a	1	4
b	2	5
c	3	6

```
csv.reader(csvfile, dialect='excel', **fmtparams)
```

- fmtparams : <https://docs.python.org/3/library/csv.html#csv-fmt-params>  
(<https://docs.python.org/3/library/csv.html#csv-fmt-params>)

In [81]:

```
1 with open('participants.csv') as csv_file:
2     csv_reader = csv.reader(csv_file, delimiter=';')
3     header = next(csv_reader)
4     print(f'Заголовок: {"| ".join(header)}')
5     for line_count, row in enumerate(csv_reader, 1):
6         print(f'Строка {line_count}: {"| ".join(row)}')
7     print(f'Обработано {line_count} строк.')
```

Заголовок: Last Name| First Name| Company Name| Company Department| Assigned Classifications| City| State| Country  
Строка 1: AALTONEN| Wanida| University| Graduate student| School or university (student)| London| | United Kingdom  
Строка 2: ABDELHAMID| Mohamed| alexandria university| student| School or university (student)| alexandria| | Egypt  
Строка 3: ABDELMEGUID| Sheref| IBM EGYPT BRANCH| Software Engineer| School or university (staff)| Alexandria| Esatern| Egypt  
Строка 4: ABDOU| Lahadji| TIFAKI HAZI| MANAGER| Association| MAMOUDZOU| | France  
Строка 5: ABEJIDE| Oluwaseun Adeyemi| Individual| Project Officer| National government (members + staff)| Ido-Osi| Ekiti State| Nigeria  
Строка 6: ABTAHIFOROUSHANI| Seyedehasieh| Student| Student| School or university (student)| ROMAINVILLE| | France  
Строка 7: ABUELALA| Sherif| AGAP company| Architect| Other company| Alexandria| | Egypt  
Строка 8: ABUNEMEH| Omar| omar kamal| social worker| Association| tulkarem| west bank| Palestinian Territory, Occupied  
Строка 9: ACHARYA| Arjun Bahadur| Wave Express Youth Club| Program Officer|

Прямое чтение CSV в словарь

```
class csv.DictReader(f, fieldnames=None, restkey=None, restval=None, dialect='excel', *args, **kws)
```

The `fieldnames` parameter is a sequence.

- If `fieldnames` is omitted, the values in the first row of file `f` will be used as the `fieldnames`.
- Regardless of how the `fieldnames` are determined, the dictionary preserves their original ordering.
- If a row has more fields than `fieldnames`, the remaining data is put in a list and stored with the `fieldname` specified by `restkey` (which defaults to `None`).
- If a non-blank row has fewer fields than `fieldnames`, the missing values are filled-in with the value of `restval` (which defaults to `None`).

In [82]:

```
1 with open('participants.csv') as csv_file:
2     csv_reader = csv.DictReader(csv_file, delimiter=';')
3     header = next(csv_reader)
4     print(f'Заголовок: {"| ".join(header)}')
5
6     for line_count, row in enumerate(csv_reader, 1):
7         # Last Name| First Name| Company Name
8         print(f'\t{row["First Name"]} {row["Last Name"]} работает в {row["Company Name"]}')
9         print(f'Обработано {line_count} строка.')
```

Заголовок: Last Name| First Name| Company Name| Company Department| Assigned Classifications| City| State| Country

Mohamed ABDELHAMID работает в alexandria university.  
Sheref ABDELMEGUID работает в IBM EGYPT BRANCH.  
Lahadji ABDOU работает в TIFAKI HAZI.  
Oluwaseun Adeyemi ABEJIDE работает в Individual.  
Seyedehasieh ABTAHIFOROUSANI работает в Student.  
Sherif ABUELALA работает в AGAP company.  
Omar ABUNEMEH работает в omar kamal.  
Arjun Bahadur ACHARYA работает в Wave Express Youth Club.  
Sarah ACHERMANN работает в cewas.  
Anne-Laure ACKERMANN работает в ADAPEI du BAS RHIN.  
Raoul ACKERMANN работает в -.  
Jose ACOSTA VALDEZ работает в 012-0100922-0.  
Filippo ADDARII работает в Young Foundation.  
Michael ADESANWO работает в T..  
Shree Ram ADHIKARI работает в N/A.  
Livia ADINOLFI работает в Une.  
J?romine ADLER работает в Sciences Po Strasbourg.

Чтение CSV в Pandas:

- Документация: [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)  
([https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html))



In [83]:

```
1 df = pd.read_csv('participants_.csv', delimiter=';')
2 df[:10]
```

Out[83]:

	Last Name	First Name	Company Name	Company Department	Assigned Classifications	City	State	Cour
0	AALTONEN	Wanida	University	Graduate student	School or university (student)	London	NaN	Un Kingc
1	ABDELHAMID	Mohamed	alexandria university	student	School or university (student)	alexandria	NaN	Eg
2	ABDELMEGUID	Sheref	IBM EGYPT BRANCH	Software Engineer	School or university (staff)	Alexandria	Esatern	Eg
3	ABDOU	Lahadji	TIFAKI HAZI	MANAGER	Association	MAMOUDZOU	NaN	Fra
4	ABEJIDE	Oluwaseun Adeyemi	Individual	Project Officer	National government (members +	Ido-Osi	Ekiti State	Nig

## Запись в CSV

In [84]:

```
1 with open('employee_file.csv', mode='w') as employee_file:
2     employee_writer = csv.writer(employee_file, delimiter=',',
3                                   quotechar='"', quoting=csv.QUOTE_MINIMAL)
4
5     employee_writer.writerow(['John Smith', 'Accounting', 'November'])
6     employee_writer.writerow(['Erica Meyers', 'IT', 'March'])
```

In [85]:

```
1 with open('employee_file.csv', mode='r') as employee_file:
2     for line in employee_file:
3         print(line, end="")
```

John Smith,Accounting,November

Erica Meyers,IT,March

In [86]:

```
1 with open('employee_file2.csv', mode='w') as csv_file:
2     fieldnames = ['emp_name', 'dept', 'birth_month']
3     writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
4
5     writer.writeheader()
6     writer.writerow({'emp_name': 'John Smith', 'dept': 'Accounting', 'birth_month': 'November'})
7     writer.writerow({'emp_name': 'Erica Meyers', 'dept': 'IT', 'birth_month': 'March'})
```

In [87]:

```
1 with open('employee_file2.csv', mode='r') as employee_file:
2     for line in employee_file:
3         print(line, end="")
```

emp\_name,dept,birth\_month

John Smith,Accounting,November

Erica Meyers,IT,March

Работа с CSV в Pandas: Документация:

- чтение: [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)  
([https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html))
- запись: [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html#pandas.read\\_csv](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html#pandas.read_csv) ([https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html#pandas.read\\_csv](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html#pandas.read_csv))

In [88]:

```
1 participants_df = pd.read_csv('participants_.csv', delimiter=';')
2 participants_df[:10]
```

Out[88]:

	Last Name	First Name	Company Name	Company Department	Assigned Classifications	City
0	AALTONEN	Wanida	University	Graduate student	School or university (student)	London
1	ABDELHAMID	Mohamed	alexandria university	student	School or university (student)	alexandria
2	ABDELMEGUID	Sheref	IBM EGYPT BRANCH	Software Engineer	School or university (staff)	Alexandria
3	ABDOU	Lahadji	TIFAKI HAZI	MANAGER	Association	MAMOUDZOU
4	ABEJIDE	Oluwaseun Adeyemi	Individual	Project Officer	National government (members + staff)	Ido-Osi
5	ABTAHIFOROUSANI	Seyedehasieh	Student	Student	School or university (student)	ROMAINVILLE
6	ABUELALA	Sherif	AGAP company	Architect	Other company	Alexandria
7	ABUNEMEH	Omar	omar kamal	social worker	Association	tulkarem
8	ACHARYA	Arjun Bahadur	Wave Express Youth Club	Program Officer	Other	Kathmandu
9	ACHERMANN	Sarah	cewas	Project Manager	Association	Willisau



In [89]:

```
1 hrdata_df = pd.read_csv('hrdata.csv')
2 hrdata_df
```

Out[89]:

	Name	Hire Date	Salary	Sick Days remaining
0	Graham Chapman	03/15/14	50000.0	10
1	John Cleese	06/01/15	65000.0	8
2	Eric Idle	05/12/14	45000.0	10
3	Terry Jones	11/01/13	70000.0	3
4	Terry Gilliam	08/12/14	48000.0	7
5	Michael Palin	05/23/13	66000.0	8

In [90]:

```
1 hrdata_df = pd.read_csv('hrdata.csv', index_col='Name')
2 print(hrdata_df)
```

	Hire Date	Salary	Sick Days remaining
Name			
Graham Chapman	03/15/14	50000.0	10
John Cleese	06/01/15	65000.0	8
Eric Idle	05/12/14	45000.0	10
Terry Jones	11/01/13	70000.0	3
Terry Gilliam	08/12/14	48000.0	7
Michael Palin	05/23/13	66000.0	8

In [91]:

```
1 print(type(hrdata_df['Hire Date'][0]))
```

<class 'str'>

In [93]:

```
1 hrdata_df = pd.read_csv('hrdata.csv',
2                           index_col='Emp',
3                           parse_dates=['Hired'],
4                           header=0,
5                           names=['Emp', 'Hired', 'Salary', 'Sick Days'])
6
7 hrdata_df
```

Out[93]:

	Hired	Salary	Sick Days
Emp			
Graham Chapman	2014-03-15	50000.0	10
John Cleese	2015-06-01	65000.0	8
Eric Idle	2014-05-12	45000.0	10
Terry Jones	2013-11-01	70000.0	3
Terry Gilliam	2014-08-12	48000.0	7
Michael Palin	2013-05-23	66000.0	8

## Формат для ndarray - пры

- [к оглавлению](#)

**Формат пры** - двоичный формат для сохранения (сериализации) произвольных единичных массивов ndarray на диск.

- Формат хранит информацию о:
  - форме (shape)
  - типе элементов (dtype)
- Формат позволяет корректно восстановить массив на компьютере с другой архитектурой.
- Формат задуман как максимально простой, достигающий поставленной цели.
- Поддержка работы с форматом распространяется как часть основного пакета numpy.
- Реализация предназначена для чистого Python и распространяется как часть основного пакета numpy.

### Сравнение с альтернативными подходами

Сравнение с использованием pickle:

- Использование pickle приводит к дублированию данных в памяти. Для очень больших массивов это может быть очень затратно или неприемлемо из-за нехватки памяти на компьютере.
- пры использует механизм memoizing который позволяет не загружать большой объем данных тогда, когда требуется лишь небольшой фрагмент данных.

In [1]:

```
1 import numpy as np
2 import pandas as pd
3 import pickle
4
5 from sys import getsizeof
6 import os
```

In [2]:

```
1 a1 = np.arange(100)
2 a2 = np.arange(10000, step=10)
```

In [3]:

```
1 getsizeof(a1)
```

Out[3]:

496

```
numpy.save(file, arr, allow_pickle=True, fix_imports=True)
```

**Save an array to a binary file in NumPy .npy format.**

### Parameters

- *file* : file, str, or pathlib.Path

Файл или имя файла, в котором сохраняются данные. Если файл является объектом-файлом, то имя файла не изменяется. Если файл представляет собой строку или путь, к имени файла будет добавлено расширение `.npy`, если этого не было сделано ранее.

- *arr* : array\_like

Массив для сохранения

- *allow\_pickle* : bool, optional

Разрешить сохранение массивов объектов с помощью Python pickles. Причины запрета на использование pickle включают безопасность (загрузка данных в формате pickle может выполнять произвольный код) и переносимость (объекты pickle могут не загружаться на разных инсталляциях Python, например, если для сохраненных объектов требуются библиотеки, которые недоступны, и не все данные совместимы при переносе между Python 2 и Python 3). **По умолчанию: True**

- *fix\_imports* : bool, optional

(Необходим для совместимости между Python 3 и Python 2)

In [4]:

```
1 with open('a1.npy', 'wb') as f:
2     np.save(f, a1)
```

In [5]:

```
1 os.path.getsize('a1.npy')
```

Out[5]:

528

In [6]:

```
1 with open('a1.pickle', 'wb') as f:
2     pickle.dump(a1, f)
```

In [7]:

```
1 os.path.getsize('a1.pickle')
```

Out[7]:

558

In [8]:

```
1 # загрузка ndarray из файла npy:
2 with open('a1.npy', 'rb') as f:
3     a1_l = np.load(f)
```

In [9]:

```
1 a1_l
```

Out[9]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

In [10]:

```
1 with open('a1_2.npy', 'wb') as f:
2     np.save(f, a1)
3     np.save(f, a2)
```

In [11]:

```
1 os.path.getsize('a1_2.npy')
```

Out[11]:

4656

In [12]:

```
1 with open('a1_2.npy', 'rb') as f:
2     a1_l = np.load(f)
3     a2_l = np.load(f)
```

In [13]:

```
1 len(a1_1), len(a2_1)
```

Out[13]:

```
(100, 1000)
```

In [14]:

```
1 a2_1[:10], a2_1[-10:]
```

Out[14]:

```
(array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90]),  
 array([9900, 9910, 9920, 9930, 9940, 9950, 9960, 9970, 9980, 9990]))
```

## Использование формата npz

Формат `.npz` — это архив `zip` с файлами имеющими имена переменных, которые в нем сохранены. Файл не сжат и каждый файл в нем это файл содержащий `ndarray` в формате `.npz`. При открытии файла `.npz` возвращается объект типа `NpzFile`. Это объект имеющий интерфейс словаря из которого можно получить список массивов и содержимое самих массивов. При сохранение `ndarray` необходимо использовать имена, которые могут быть корректными именами файлов (в т.ч. не содержат `/` или `..`).

```
numpy.savez(file, *args, **kws)
```

### Save several arrays into a single file in uncompressed `.npz` format.

If arguments are passed in with no keywords, the corresponding variable names, in the `.npz` file, are `'arr_0'`, `'arr_1'`, etc. If keyword arguments are given, the corresponding variable names, in the `.npz` file will match the keyword names.

#### Parameters

- `file` : str or file

Either the filename (string) or an open file (file-like object) where the data will be saved. If file is a string or a Path, the `.npz` extension will be appended to the filename if it is not already there.

- `args` : Arguments, optional

Arrays to save to the file. Since it is not possible for Python to know the names of the arrays outside `savez`, the arrays will be saved with names `"arr_0"`, `"arr_1"`, and so on. These arguments can be any expression.

- `kws` : Keyword arguments, optional

Arrays to save to the file. Arrays will be saved in the file with the keyword names.

In [15]:

```
1 np.savez('a.npz', a1, a2)
```



In [16]:

```
1 npzfile = np.load('a.npz')
```

In [17]:

```
1 list(npzfile)
```

Out[17]:

```
['arr_0', 'arr_1']
```

In [18]:

```
1 npzfile['arr_0']
```

Out[18]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
        68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
        85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])
```

In [19]:

```
1 arrs = list(npzfile.values())
2 len(arrs)
```

Out[19]:

```
2
```

In [20]:

```
1 arrs[0][-10:], arrs[1][-10:]
```

Out[20]:

```
(array([90, 91, 92, 93, 94, 95, 96, 97, 98, 99]),
 array([9900, 9910, 9920, 9930, 9940, 9950, 9960, 9970, 9980, 9990]))
```

In [21]:

```
1 # NpzFile нужно закрывать:
2 npzfile.close()
```

In [22]:

```
1 npzfile['arr_0']
```

```
-----  
AttributeError                                Traceback (most recent call last)  
<ipython-input-22-a744a10672d6> in <module>  
----> 1 npzfile['arr_0']  
  
~\conda\envs\pyTorch_1_5v2\lib\site-packages\numpy\lib\npyio.py in __getitem__  
__(self, key)  
    253         key += '.npy'  
    254         if member:  
--> 255             bytes = self.zip.open(key)  
    256             magic = bytes.read(len(format.MAGIC_PREFIX))  
    257             bytes.close()
```

AttributeError: 'NoneType' object has no attribute 'open'

**Альтернативные способы работы с .npz :**

In [30]:

```
1 M = 1000_000  
2 S = 10  
3 sz = S * M  
4 a3 = np.arange(sz)
```

In [31]:

```
1 # сохранение в npz массивов с заданными именами:  
2 np.savez('a_new.npz', a1=a1, a3=a3)
```

In [44]:

```
1 size = os.path.getsize('a_new.npz')  
2 print(f'uncompressed size: {size:,}')
```

uncompressed size: 40,000,894

In [34]:

```
1 # другой стиль чтения из с npz:  
2 with np.load('a_new.npz') as data:  
3     a3_1 = data['a3']  
4  
5 a2_1[-10:]
```

Out[34]:

array([9900, 9910, 9920, 9930, 9940, 9950, 9960, 9970, 9980, 9990])

In [37]:

```
1 # np.savez('a_compr.npz', a1=a1, a2=a2)
2
3 # сохранение в сжатый файл .npz "_compressed":
4 np.savez_compressed('a_compr.npz', a1=a1, a3=a3)
```

In [45]:

```
1 size = os.path.getsize('a_compr.npz')
2 print(f'compressed size: {size:,}')
```

compressed size: 13,831,101

In [46]:

```
1 # чтение из сжатого файла не отличается:
2 with np.load('a_compr.npz') as data:
3     a2_1 = data['a2']
4
5 a2_1[-10:]
```

```
-----
KeyError                                Traceback (most recent call last)
<ipython-input-46-c1f0cd3220db> in <module>
      1 # чтение из сжатого файла не отличается:
      2 with np.load('a_compr.npz') as data:
----> 3     a2_1 = data['a2']
      4
      5 a2_1[-10:]

~\.conda\envs\pyTorch_1_5v2\lib\site-packages\numpy\lib\ndarray.py in __getitem__(self, key)
    264         return self.zip.read(key)
    265     else:
--> 266         raise KeyError("%s is not a file in the archive" % key)
    267
    268
```

**KeyError:** 'a2 is not a file in the archive'

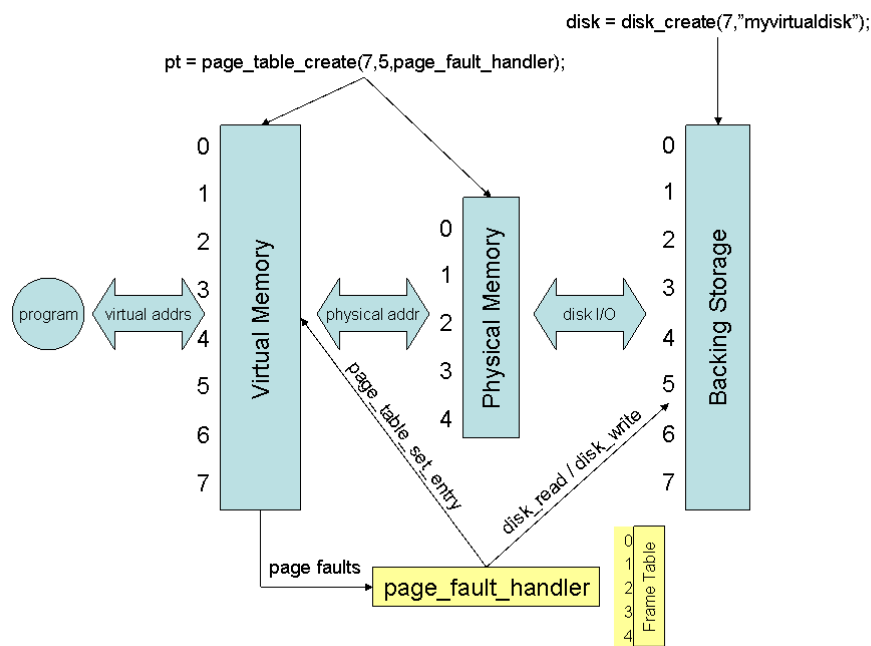
## Работа с использованием mmap

- <https://numpy.org/doc/stable/reference/generated/numpy.load.html#numpy.load>  
(<https://numpy.org/doc/stable/reference/generated/numpy.load.html#numpy.load>)
- <https://numpy.org/doc/stable/reference/generated/numpy.memmap.html#numpy.memmap>  
(<https://numpy.org/doc/stable/reference/generated/numpy.memmap.html#numpy.memmap>)

**Виртуальная память** (virtual memory) — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем (например, жёстким диском).

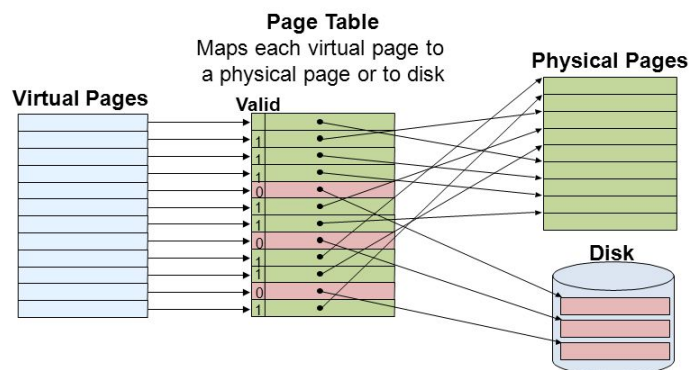
- Для программы **применение виртуальной памяти полностью прозрачно** и не требует дополнительных усилий со стороны программиста.

- Реализация виртуальной памяти требует как **аппаратной поддержки**, так и **поддержки со стороны операционной системы**.



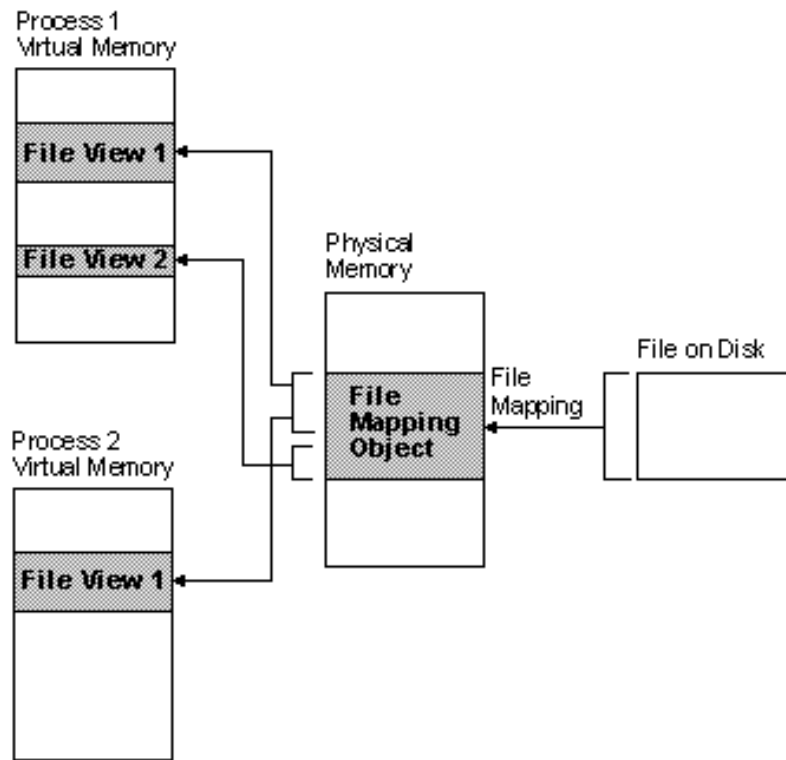
### Концепция виртуальной памяти

**Таблица страниц** (page table)- это структура данных, используемая системой виртуальной памяти в операционной системе компьютера для хранения сопоставления между виртуальным адресом и физическим адресом. Виртуальные адреса используются выполняющимся процессом, в то время как физические адреса используются аппаратным обеспечением, или, более конкретно, подсистемой ОЗУ. Таблица страниц является ключевым компонентом преобразования виртуальных адресов, который необходим для доступа к данным в памяти.



- **The OS manages a page table for each process**
  - The OS maps the process virtual pages to physical pages or to disk
  - Only the OS writes into the page tables
  - The page tables reside in the physical memory (DRAM)

### Принцип работы с таблицей страниц



**Memory-mapped files**

Модуль Python `mmap` предоставляет возможность работать вводом-выводом (I/O) с помощью memory-mapped file. Это позволяет использовать преимущества низкоуровневых возможностей операционных систем для представления файла в виде одной большой строки или массива. Это может дать значительный прирост производительности для кода, требующего большого объема ввода-вывода.

- Подробнее про меммап <https://docs.python.org/3.0/library/mmap.html> (<https://docs.python.org/3.0/library/mmap.html>) , <https://realpython.com/python-mmap> (<https://realpython.com/python-mmap>).

In [48]:

```

1 M = 1000000
2 size = 100 * M # 100 M
3 # size = S * M # 10 M
4 a_big_1 = np.arange(size)

```

In [49]:

```

1 size = getsizeof(a_big_1)
2 print(f'size of a_big_1 in memory: {size:,}')

```

size of a\_big\_1 in memory: 400,000,096

Пишем и читаем фрагмент с помощью pickle:

In [50]:

```

1 with open('a_big_1.pickle', 'wb') as f:
2     pickle.dump(a_big_1, f)

```

In [53]:

```
1 size = os.path.getsize('a_big_1.pickle')
2
3 print(f'a_big_1.pickle size: {size:,}')
```

a\_big\_1.pickle size: 400,000,161

In [56]:

```
1 %%time
2 with open('a_big_1.pickle', 'rb') as f:
3     a_big_1_lp = pickle.load(f)
4
5 # чтение из середины массива:
6 my_data = a_big_1_lp[5 * M : 5 * M + 10000]
7 my_data[:10], my_data[-10:]
```

Wall time: 271 ms

Out[56]:

```
(array([5000000, 5000001, 5000002, 5000003, 5000004, 5000005, 5000006,
        5000007, 5000008, 5000009]),
 array([5009990, 5009991, 5009992, 5009993, 5009994, 5009995, 5009996,
        5009997, 5009998, 5009999]))
```

In [54]:

```
1 with open('a_big_1.npy', 'wb') as f:
2     np.save(f, a_big_1)
```

In [57]:

```
1 size = os.path.getsize('a_big_1.npy')
2
3 print(f'a_big_1.npy: {size:,}')
```

a\_big\_1.npy: 400,000,128

**mmap\_mode** = {None, 'r+', 'r', 'w+', 'c'}, optional

Если параметр `memory-map` установлен не в `None`, то используется выбранный режим.

- Массив с отображением памяти хранится на диске
- но к нему можно получить доступ и выполнить срезы как для любого массива `ndarray`
- использование `mmap` особенно полезно для доступа к небольшим фрагментам больших файлов без чтения всего файла в память.

Подробнее о работе с `mmap`: <https://numpy.org/doc/stable/reference/generated/numpy.memmap.html>  
(<https://numpy.org/doc/stable/reference/generated/numpy.memmap.html>)

In [76]:

```
1 %%time
2 a_big_mmap = np.load('a_big_1.npy', mmap_mode='r')
3
4 # чтение из середины массива:
5 my_data = a_big_mmap[5 * M : 5 * M + 10000]
6 my_data[:10], my_data[-10:]
```

Wall time: 965 µs

Out[76]:

```
(mmap([5000000, 5000001, 5000002, 5000003, 5000004, 5000005, 5000006,
      5000007, 5000008, 5000009]),
 mmap([5009990, 5009991, 5009992, 5009993, 5009994, 5009995, 5009996,
      5009997, 5009998, 5009999]))
```

In [77]:

```
1 type(a_big_mmap), type(my_data)
```

Out[77]:

```
(numpy.memmap, numpy.memmap)
```

In [78]:

```
1 # если необходимо скопировать данные из мемтар в обычный ndarray:
2 my_data_c = np.copy(my_data)
3 my_data_c[:10], my_data_c[-10:]
```

Out[78]:

```
(array([5000000, 5000001, 5000002, 5000003, 5000004, 5000005, 5000006,
      5000007, 5000008, 5000009]),
 array([5009990, 5009991, 5009992, 5009993, 5009994, 5009995, 5009996,
      5009997, 5009998, 5009999]))
```

In [79]:

```
1 np.info(my_data)
```

```
class: memmap
shape: (10000,)
strides: (4,)
itemsize: 4
aligned: True
contiguous: True
fortran: True
data pointer: 0x1d1ba1a2d80
byteorder: little
byteswap: False
type: int32
```

In [80]:

```
1 np.version.version
```

Out[80]:

```
'1.18.1'
```

In [81]:

```
1 my_data._mmap
```

Out[81]:

```
<mmap.mmap at 0x1d13a866ab0>
```

In [82]:

```
1 np.info(my_data_c)
```

```
class: ndarray
shape: (10000,)
strides: (4,)
itemsize: 4
aligned: True
contiguous: True
fortran: True
data pointer: 0x1d13918b530
byteorder: little
byteswap: False
type: int32
```

In [83]:

```
1 my_data_c._mmap
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-83-1df436867b16> in <module>
----> 1 my_data_c._mmap
```

```
AttributeError: 'numpy.ndarray' object has no attribute '_mmap'
```

Закрытие файла mmap произойдет либо после того, как сборщик мусора удалит все объекты ссылающиеся на файл. Это можно ускорить удалив объекты:

```
del a_big_mmap
del my_data
```

или явно:

```
a_big_mmap._mmap.close()
```

но тогда любое обращение использование этого mmap (в том числе и в других объектах) приведет к ошибке. Чтобы избежать проблем, перед закрытием скопируйте необходимые данные в другие ndarray.



In [84]:

```
1 a_big_mmap._mmap.close()
```

In [85]:

```
1 # ошибка: mmap уже закрыт (НЕ ВЫПОЛНЯТЬ!)
2 # my_data2 = a_big_mmap[50 * M : 50 * M + 10000]
3 # my_data2[:10]
```

In [86]:

```
1 # со скопированными данными все в порядке:
2 my_data_c[:10], my_data_c[-10:]
```

Out[86]:

```
(array([5000000, 5000001, 5000002, 5000003, 5000004, 5000005, 5000006,
        5000007, 5000008, 5000009]),
 array([5009990, 5009991, 5009992, 5009993, 5009994, 5009995, 5009996,
        5009997, 5009998, 5009999]))
```

## Формат HDF

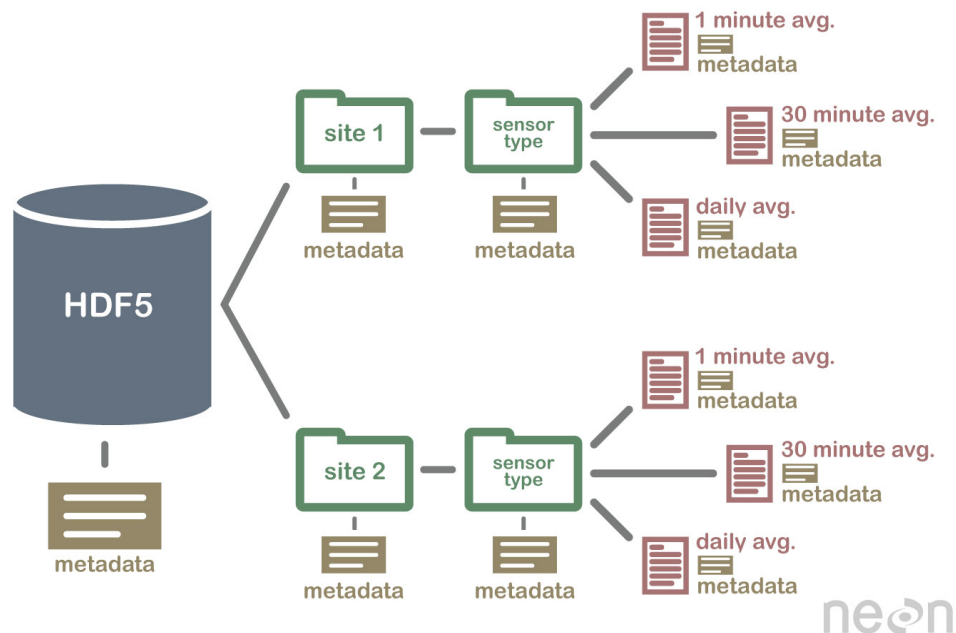
- [к оглавлению](#)

**Hierarchical Data Format, HDF** (Иерархический формат данных) — название формата файлов, разработанного для хранения большого объема цифровой информации.

- Поддерживается некоммерческой организацией HDF Group.
- Библиотеки для работы с форматом и связанные с ним утилиты доступны для использования под свободной лицензией.
- Имеются библиотеки для работы с HDF для большинства языков программирования, активно используемых для обработки числовых данных (в частности: Java, Matlab, Scilab, Octave, Mathematica, IDL, Python, R и Julia)
- Существуют отдельные инструменты для работы с HDF, например:  
<https://support.hdfgroup.org/products/java/hdfview/> (<https://support.hdfgroup.org/products/java/hdfview/>).

На данный момент в ходу следующие варианты HDF:

- **HDF4** — старая версия формата, однако все еще активно поддерживаемая HDF Group.
  - Формат поддерживает различные модели данных, включая многомерные массивы, растровые изображения и таблицы.
  - Использует 32-битные целые числа, поэтому имеет проблемы с хранением больших объемов информации (более нескольких гигабайт).
- **HDF5** — современная версия формата. Содержит иерархию из двух основных типов объектов:
  - **Datasets** — наборы данных, многомерные массивы объектов одного типа
  - **Groups** — группы, являются контейнерами для наборов данных и других групп
  - Содержимое файлов HDF5 организовано подобно иерархической файловой системе, и для доступа к данным применяются пути, сходные с POSIX-синтаксисом, например, `/path/to/resource`.
  - Метаданные хранятся в виде набора именованных атрибутов объектов.



### Логика организации данных в HDF5

Популярный модуль для работы с HDF5 на Python: `h5py`.

- Документация по `h5py` : <https://docs.h5py.org/en/stable/> (<https://docs.h5py.org/en/stable/>)
- Рекомендованное руководство для изучения работы с HDF5 в python:  
<https://www.pythonforthelab.com/blog/how-to-use-hdf5-files-in-python/>  
(<https://www.pythonforthelab.com/blog/how-to-use-hdf5-files-in-python/>)
- Перевод руководства: <https://habr.com/ru/company/otus/blog/416309/>  
(<https://habr.com/ru/company/otus/blog/416309/>)

In [21]:

```
1 # import numpy as np
2
3 # модуль для работы с HDF5:
4 import h5py
5
6 import time
7 import os
```

In [22]:

```
1 len(a1), a1[:10], a1[-10:]
```

Out[22]:

```
(100,
 array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([90, 91, 92, 93, 94, 95, 96, 97, 98, 99]))
```

In [23]:

```
1 with h5py.File('data1.hdf5', 'w') as f:
2     dset = f.create_dataset("a1", data=a1)
```

In [24]:

```
1 os.path.getsize('data1.hdf5')
```

Out[24]:

2448

In [25]:

```
1 with h5py.File('data1.hdf5', 'r') as f:
2     a1_l = f['a1']
3     print(len(a1), a1[:10], a1[-10:])
```

100 [0 1 2 3 4 5 6 7 8 9] [90 91 92 93 94 95 96 97 98 99]

In [26]:

```
1 # hdf5 позволяет обойти сохраненные в файле датасеты:
2 with h5py.File('data1.hdf5', 'r') as f:
3     for key in f.keys():
4         print(key)
```

a1

In [27]:

```
1 f = h5py.File('data1.hdf5', 'r')
2 a1_l = f['a1']
```

In [28]:

```
1 # пока файл не закрыт, мы можем пользоваться датасетом, в т.ч. с использованием срезов:
2 a1_l[:10], a1_l[-10:]
```

Out[28]:

(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),  
 array([90, 91, 92, 93, 94, 95, 96, 97, 98, 99]))

In [29]:

```
1 # mun a1:
2 type(a1_l)
```

Out[29]:

h5py.\_hl.dataset.Dataset

In [30]:

```
1 f.close()
```

In [187]:

```
1 # ошибка!
2 # a1_l[:10]
```

In [31]:

```
1 with h5py.File('data1.hdf5', 'r') as f:
2     a1_l = f['a1']
3     data = a1_l[:10]
```

In [32]:

```
1 a1_l[1]
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-32-e531e1b40df7> in <module>
----> 1 a1_l[1]
```

```
h5py\_objects.pyx in h5py._objects.with_phil.wrapper()
```

```
h5py\_objects.pyx in h5py._objects.with_phil.wrapper()
```

```
C:\ProgramData\Anaconda3\envs\pyTorch_1_6\lib\site-packages\h5py\_hl\dataset.
py in __getitem__(self, args)
487     """
488     args = args if isinstance(args, tuple) else (args,)
--> 489     if is_empty_dataspace(self.id):
490         if not (args == tuple() or args == (Ellipsis,)):
491             raise ValueError("Empty datasets cannot be sliced")
```

```
C:\ProgramData\Anaconda3\envs\pyTorch_1_6\lib\site-packages\h5py\_hl\base.py
in is_empty_dataspace(obj)
85 def is_empty_dataspace(obj):
86     """ Check if an object's dataspace is empty """
--> 87     if obj.get_space().get_simple_extent_type() == h5s.NULL:
88         return True
89     return False
```

```
h5py\_objects.pyx in h5py._objects.with_phil.wrapper()
```

```
h5py\_objects.pyx in h5py._objects.with_phil.wrapper()
```

```
h5py\h5d.pyx in h5py.h5d.DatasetID.get_space()
```

```
ValueError: Not a dataset (not a dataset)
```

In [33]:

```
1 # Ошибки нет, т.к. при выполнении среза a1_l[:10] данные были прочитаны и сохранены в с
2 data[1]
```

Out[33]:

1

In [34]:

```
1 type(data)
```

Out[34]:

numpy.ndarray

Запись в HDF5

In [36]:

```
1 a_rnd1 = np.random.randn(100)
2
3 with h5py.File('random1.hdf5', 'w') as f:
4     dset = f.create_dataset("default", (1000,))
5     dset[10:20] = a_rnd1[50:60] # записываем только часть датасета!
```

In [37]:

```
1 with h5py.File('random1.hdf5', 'r') as f:
2     print(f['default'][:30])
```

```
[ 0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        -0.320671 -0.57363313
  1.195517   1.5308056   0.37572408  0.23483108 -0.3134993   0.11133709
  0.9928594   0.46926144  0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.]
```

In [38]:

```
1 a_rnd2 = np.random.randn(1000)
```

In [40]:

```
1 with h5py.File('random2.hdf5', 'w') as f:
2     dset = f.create_dataset("default", (1000,))
3     dset = a_rnd2 # ошибка!
```

In [41]:

```
1 with h5py.File('random2.hdf5', 'r') as f:
2     print(f['default'][:10])
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

In [42]:

```
1 with h5py.File('random2.hdf5', 'w') as f:
2     dset = f.create_dataset("default", (1000,))
3     dset[:] = a_rnd2 # копирование производится при обходе среза!
```

In [43]:

```
1 with h5py.File('random2.hdf5', 'r') as f:
2     print(f['default'][:10])
```

```
[-1.7904351 -1.0703183 -0.44565916 -0.04734383 -1.4929967 -0.85322
 1.3765613  2.318192 -1.2199734 -0.6433627 ]
```

Типизация датасетов в HDF5

- документация: <https://docs.h5py.org/en/latest/faq.html> (<https://docs.h5py.org/en/latest/faq.html>)

In [44]:

```
1 with h5py.File('several_datasets.hdf5', 'w') as f:
2     # объявление нескольких типизированных датасетов в одном файле:
3     dset_int_1 = f.create_dataset('integers', (10, ), dtype='i1')
4     dset_int_8 = f.create_dataset('integers8', (10, ), dtype='i8')
5     dset_complex = f.create_dataset('complex', (10, ), dtype='c16')
6
7     # помещение данных в датасеты:
8     dset_int_1[0] = 1200
9     dset_int_8[0] = 1200.1
10    dset_complex[0] = 3 + 4j
```

In [46]:

```
1 arr = np.random.randn(100000)
2
3 f = h5py.File('integer_1.hdf5', 'w')
4 d = f.create_dataset('dataset', (100000, ), dtype='i1')
5 d[:] = arr
6 print(type(d[0]))
7 f.close()
8 print(os.path.getsize('integer_1.hdf5'))
9
10 f = h5py.File('integer_8.hdf5', 'w')
11 d = f.create_dataset('dataset', (100000, ), dtype='i8')
12 d[:] = arr
13 print(type(d[0]))
14 f.close()
15 print(os.path.getsize('integer_8.hdf5'))
16
17 f = h5py.File('float_2.hdf5', 'w')
18 d = f.create_dataset('dataset', (100000, ), dtype='f2')
19 d[:] = arr
20 print(type(d[0]))
21 f.close()
22 print(os.path.getsize('float_2.hdf5'))
```

```
<class 'numpy.int8'>
102048
<class 'numpy.int64'>
802048
<class 'numpy.float16'>
202048
```

Создание сжатых файлов:

In [47]:

```
1 with h5py.File('integer_1_compr.hdf5', 'w') as f:
2     d = f.create_dataset('dataset', (100000,), dtype='i1',
3                             compression="gzip", compression_opts=9)
4     d[:] = arr
5 print(os.path.getsize('integer_1.hdf5'))
6 print(os.path.getsize('integer_1_compr.hdf5'))
7
8 with h5py.File('integer_8_compr.hdf5', 'w') as f:
9     d = f.create_dataset('dataset', (100000,), dtype='i8', compression="gzip", compress
10     d[:] = arr
11 print(os.path.getsize('integer_8.hdf5'))
12 print(os.path.getsize('integer_8_compr.hdf5'))
13
14 with h5py.File('float_2_compr.hdf5', 'w') as f:
15     d = f.create_dataset('dataset', (100000,), dtype='f2', compression="gzip", compress
16     d[:] = arr
17 print(os.path.getsize('float_2_compr.hdf5'))
18 print(os.path.getsize('float_2.hdf5'))
```

102048  
27969  
802048  
43231  
188337  
202048

Изменение размера датасета

In [48]:

```
1 with h5py.File('resize_dataset.hdf5', 'w') as f:
2     d = f.create_dataset('dataset', (100, ), maxshape=(500, ))
3     d[:100] = np.random.randn(100)
4     d.resize((200,))
5     d[100:200] = np.random.randn(100)
6
7 with h5py.File('resize_dataset.hdf5', 'r') as f:
8     dset = f['dataset']
9     print(dset[99])
10    print(dset[199])
```

0.60942894  
-0.2867902

In [50]:

```
1 # изменение размера датасета в уже существовавшем файле:
2 with h5py.File('resize_dataset.hdf5', 'a') as f:
3     dset = f['dataset']
4     dset.resize((300,))
5     dset[:200] = 0
6     dset[200:300] = np.random.randn(100)
7
8 with h5py.File('resize_dataset.hdf5', 'r') as f:
9     dset = f['dataset']
10    print(dset[99])
11    print(dset[199])
12    print(dset[299])
```

0.0

0.0

-1.5227135

### Сохранение данных блоками (Chunks)

Чтобы оптимизировать хранение данных, их можно хранить блоками (chunk).

- Каждый блок (chunk) будет смежным на жестком диске и будет храниться единым фрагментом, т.е. весь фрагмент будет записан сразу.
- Когда вы читаете блок, произойдет то же самое - он будет загружен целиком. Чтобы создать кусочный набор данных.

In [54]:

```
1 with h5py.File('chunked_dataset.hdf5', 'a') as f:
2     dset1 = f.create_dataset("chunked2", (1000, 1000),
3                                     chunks=(100, 100))
4     dset2 = f.create_dataset("autochunk2", (1000, 1000),
5                                     chunks=True)
```

### Организация данных группами (Groups)

Группы (Groups) позволяют организовать информацию в файле. Наборы данных могут быть размещены внутри групп, которые ведут себя аналогично тому, как работают каталоги в файловой системе. Сначала мы можем создать группу, а затем добавить в нее набор данных.

In [55]:

```
1 with h5py.File('groups.hdf5', 'w') as f:
2     g = f.create_group('Base_Group')
3     gg = g.create_group('Sub_Group')
4
5     d = g.create_dataset('default', data=arr)
6     dd = gg.create_dataset('default', data=arr)
```



In [56]:

```
1 with h5py.File('groups.hdf5', 'r') as f:
2     d = f['Base_Group/default']
3     dd = f['Base_Group/Sub_Group/default']
4     print(d[1])
5     print(dd[1])
```

0.540710342299558

0.540710342299558

In [57]:

```
1 with h5py.File('groups.hdf5', 'r') as f:
2     for k in f.keys():
3         print(k)
```

Base\_Group

In [59]:

```
1 # cnoco6 o6oŷmu
2 def get_all(name):
3     print(name)
4
5 with h5py.File('groups.hdf5', 'r') as f:
6     f.visit(get_all)
```

Base\_Group

Base\_Group/Sub\_Group

Base\_Group/Sub\_Group/default

Base\_Group/default

In [60]:

```
1 def get_all(name):
2     if 'Sub_Group' in name:
3         return name
4
5 with h5py.File('groups.hdf5', 'r') as f:
6     g = f.visit(get_all)
7     print(g)
```

Base\_Group/Sub\_Group

In [61]:

```
1 with h5py.File('groups.hdf5', 'r') as f:
2     g_name = f.visit(get_all)
3     group = f[g_name]
4     for k, ds in group.items():
5         print(k)
6         print(ds[0])
7     # print(group[0])
```

default

-0.4059243035864696

In [62]:

```
1 def get_objects(name, obj):
2     if 'Sub_Group' in name:
3         return obj
4
5 with h5py.File('groups.hdf5', 'r') as f:
6     group = f.visititems(get_objects)
7     data = group['default']
8     print('First data element: {}'.format(data[0]))
```

First data element: -0.4059243035864696

## Хранение метаданных в HDF5

Одной из важных возможностей HDF5 является возможность хранения метаданных, прикрепленных к любой группе или набору данных. Метаданные позволяют понять, например, источник данных, каковы параметры, используемые для измерения или моделирования, и т.д. Метаданные делают файл самоописательным.

In [64]:

```
1 with h5py.File('groups.hdf5', 'w') as f:
2     g = f.create_group('Base_Group')
3     d = g.create_dataset('default', data=arr)
4
5     g.attrs['Date'] = time.time()
6     g.attrs['User'] = 'Me'
7
8     d.attrs['OS'] = os.name
9
10    for k in g.attrs.keys():
11        print('{} => {}'.format(k, g.attrs[k]))
12
13    for j in d.attrs.keys():
14        print('{} => {}'.format(j, d.attrs[j]))
```

Date => 1632730138.2346385

User => Me

OS => nt

In [212]:

```
1 with h5py.File('groups.hdf5', 'w') as f:
2     g = f.create_group('Base_Group')
3     d = g.create_dataset('default', data=arr)
4
5     metadata = {'Date': time.time(),
6                 'User': 'Me',
7                 'OS': os.name,}
8
9     f.attrs.update(metadata)
10
11     for m in f.attrs.keys():
12         print('{ } => { }'.format(m, f.attrs[m]))
```

Date => 1601647108.3349116

OS => nt

User => Me

## Использование HDF5 для работы с Pandas

```
DataFrame.to_hdf(path_or_buf, key, mode='a', complevel=None, complib=None, append=False, format=None, index=True, min_itemsize=None, nan_rep=None, dropna=None, data_columns=None, errors='strict', encoding='UTF-8')
```

Документация: [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to\\_hdf.html#pandas.DataFrame.to\\_hdf](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_hdf.html#pandas.DataFrame.to_hdf)  
([https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to\\_hdf.html#pandas.DataFrame.to\\_hdf](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_hdf.html#pandas.DataFrame.to_hdf))

Некоторые параметры:

- mode : {'a', 'w', 'r+'}, default 'a' .
- complevel : {0-9}, optional, Specifies a compression level for data. A value of 0 disables compression.
- complib : {'zlib', 'lzo', 'bzip2', 'blosc'}, default 'zlib'
- format : {'fixed', 'table', None}, default 'fixed'

Possible values:

- 'fixed': Fixed format. Fast writing/reading. Not-appendable, nor searchable.
- 'table': Table format. Write as a PyTables Table structure which may perform worse but allow more flexible operations like searching / selecting subsets of the data.
  - If None, `pd.get_option('io.hdf.default_format')` is checked, followed by fallback to "fixed"
- min\_itemsize : dict or int, optional. Map column names to minimum string sizes for columns.
- nan\_rep : Any, optional. How to represent null values as str.
- data\_columns : list of columns or True, optional

```
DataFrame.to_hdf(path_or_buf, key, mode='a', complevel=None, complib=None, append=False, format=None, index=True, min_itemsize=None, nan_rep=None, dropna=None, data_columns=None, errors='strict', encoding='UTF-8')
```

Документация: [https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_hdf.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_hdf.html)  
([https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_hdf.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_hdf.html))

In [65]:

```
1 df1 = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]},
2                     index=['a', 'b', 'c'])
3 df1
```

Out[65]:

	A	B
a	1	4
b	2	5
c	3	6

In [66]:

```
1 df1.to_hdf('pandas.hdf5', key='df', mode='w')
```

In [67]:

```
1 # как все хранится внутри:
2 with h5py.File('pandas.hdf5', 'r') as f:
3     data = f['df']
4     print(data)
5     for k, ds in data.items():
6         print(k)
7         print(ds[:])
```

```
<HDF5 group "/df" (4 members)>
axis0
[b'A' b'B']
axis1
[b'a' b'b' b'c']
block0_items
[b'A' b'B']
block0_values
[[1 4]
 [2 5]
 [3 6]]
```

In [68]:

```
1 df_1 = pd.read_hdf('data.h5', 'df')
2 df_1
```

Out[68]:

	A	B
a	1	4
b	2	5
c	3	6

In [69]:

```
1 # Добавляем серию в тот же файл:
2 s = pd.Series([1, 2, 3, 4])
3 s.to_hdf('data.h5', key='s')
```

In [218]:

```
1 pd.read_hdf('data.h5', 's')
```

Out[218]:

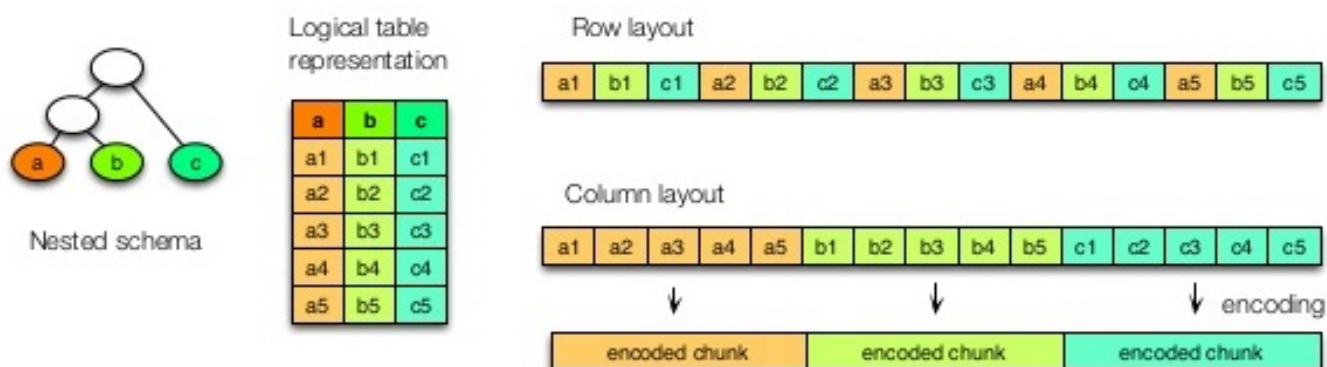
```
0    1
1    2
2    3
3    4
dtype: int64
```

## Apache Parquet

- [к оглавлению](#)

Apache Parquet хранит данные способом, при котором значения в каждом столбце физически хранятся в смежных ячейках памяти. Благодаря колоночно-ориентированному хранилищу Apache Parquet обеспечивает следующие преимущества:

- Сжатие по столбцам выполняется эффективно и экономит место для хранения данных.
  - т.к. могут применяться методы сжатия, специфичные для определенного типа, поскольку значения столбцов, как правило, имеют один и тот же тип;
  - к разным столбцам можно применять разные методы кодирования.
- Можно читать и десериализовать только те столбцы, которые необходимы.
  - Запросы, которые извлекают значения из определенного столбца, не должны считывать данные всей строки, что повышает производительность.



Принцип колоночного хранилища

Сравнение с другими популярными форматами

Spark Format Showdown		File Format		
		<u>CSV</u>	<u>JSON</u>	<u>Parquet</u>
A t t r i b u t e	Columnar	No	No	Yes
	Compressable	Yes	Yes	Yes
	Splittable	Yes*	Yes**	Yes
	Human Readable	Yes	Yes	No
	Nestable	No	Yes	Yes
	Complex Data Structures	No	Yes	Yes
	Default Schema: Named columns	Manual	Automatic (full read)	Automatic (instant)
	Default Schema: Data Types	Manual (full read)	Automatic (full read)	Automatic (instant)










#### CSV vs JSON vs Parquet

- \* CSV is splittable when it is a raw, uncompressed file or using a splittable compression format such as BZIP2 or LZO (note: LZO needs to be indexed to be splittable!)
- \*\* JSON has the same conditions about splittability when compressed as CSV with one extra difference. When “wholeFile” option is set to true (re: SPARK-18352), JSON is NOT splittable.

Основные преимущества каждого из форматов:

- CSV should generally be the fastest to write
  - CSV is the defacto standard of a lot of data and for fair reasons;
  - it's (relatively) easy to comprehend for both users and computers and made more accessible via Microsoft Excel.
- JSON the easiest for a human to understand
  - JSON is the standard for communicating on the web.
  - APIs and websites are constantly communicating using JSON because of its usability properties such as well-defined schemas.
- Parquet the fastest to read.
  - Parquet is optimized for the **Write Once Read Many (WORM)** paradigm.
  - It's slow to write, but incredibly fast to read, especially when you're only accessing a subset of the total columns.
  - For use cases **requiring operating on entire rows of data**, a format like **CSV, JSON or even AVRO should be used**.

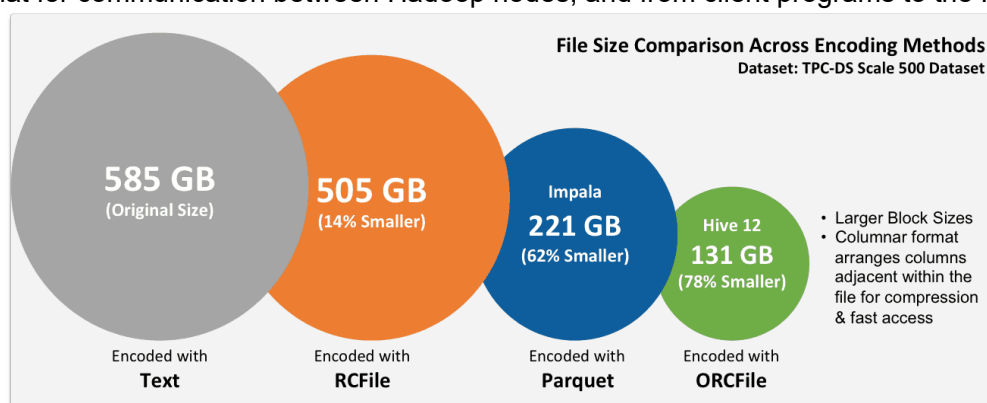
Сравнение форматов для хранения больших данных

	Avro	Parquet	ORC
Schema Evolution Support			
Compression			
Splitability			
Most Compatible Platforms	Kafka, Druid	Impala, Arrow Drill, Spark	Hive, Presto
Row or Column	Row	Column	Column
Read or Write	Write	Read	Read

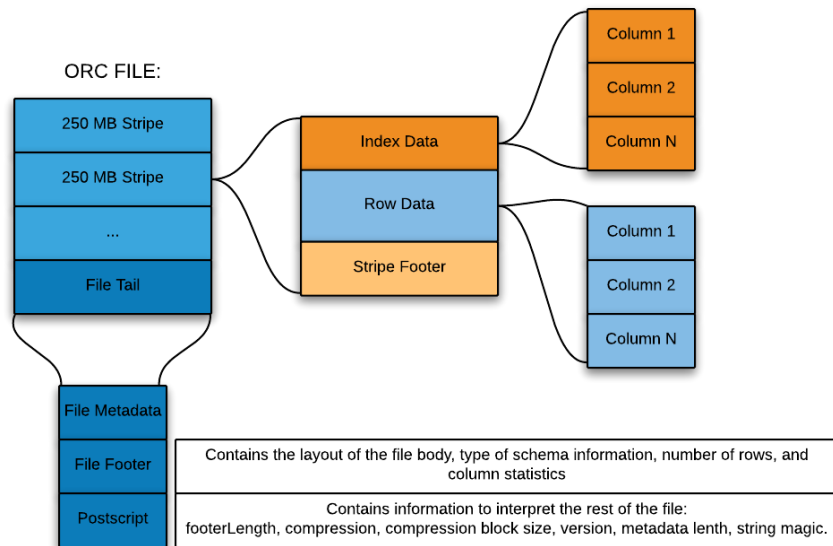
### Сравнение форматов для хранения больших данных

**Avro** - это ориентированная на строки структура удаленного вызова процедур и сериализации данных, разработанная в рамках проекта Apache Hadoop. Он использует JSON для определения типов данных и протоколов и сериализует данные в компактном двоичном формате. Его основное применение - в Apache Hadoop, где он может предоставлять как формат сериализации для постоянных данных, так и формат проводной связи для связи между узлами Hadoop и от клиентских программ к службам Hadoop.

**Avro** is a row-oriented remote procedure call and data serialization framework developed within Apache's Hadoop project. It uses JSON for defining data types and protocols, and serializes data in a compact binary format. Its primary use is in Apache Hadoop, where it can provide both a serialization format for persistent data, and a wire format for communication between Hadoop nodes, and from client programs to the Hadoop services.



### Сравнение форматов для хранения больших данных



### Организация хранения в файле формата ORC

In [219]:

```
1 import sys
```

### Apache Arrow

**Apache Arrow** - это не зависящая от языка программная среда для разработки приложений анализа данных, которые **обрабатывают колоночно-ориентированные данные**. Он содержит стандартизированный формат памяти с ориентацией на столбцы, который может представлять плоские и иерархические данные для эффективных аналитических операций на современном аппаратном обеспечении CPU и GPU. Это снижает или устраняет факторы, ограничивающие возможность работы с большими наборами данных, такие как стоимость, непостоянство хранения или физические ограничения динамической памяти с произвольным доступом.

PyArrow это API для Apache Arrow на Python. Реализация ядра Apache Arrow выполнена на C++, а PyArrow реализует интерфейс к этому ядру, реализуя принцип "Python as a 'glue language' ". PyArrow поддерживает полноценную интеграцию с:

- NumPy
- Pandas
- встроенными объектами Python

Работу с файлами parquet будем организовывать с помощью **PyArrow** .

- документация: <https://enpiar.com/arrow-site/docs/python/index.html> (<https://enpiar.com/arrow-site/docs/python/index.html>)
- работа с Pandas: <https://enpiar.com/arrow-site/docs/python/parquet.html> (<https://enpiar.com/arrow-site/docs/python/parquet.html>)

Установка и импорт модуля pyarrow:



In [5]:

```
1 !conda install --yes --prefix {sys.prefix} pyarrow arrow-cpp parquet-cpp -c conda-forge
```

Collecting package metadata (current\_repodata.json): ...working... done  
Solving environment: ...working... done

## Package Plan ##

environment location: C:\ProgramData\Anaconda3\envs\pyTorch\_1\_6

added / updated specs:

- arrow-cpp
- parquet-cpp
- pyarrow

The following packages will be downloaded:

package	build	
-----	-----	
arrow-cpp-0.15.1	py37h47fa567_6	2.9 MB
boost-cpp-1.68.0	h6a4c333_1000	31.1 MB conda-fo

In [3]:

```
1 # альтернативный способ:  
2 # %conda install pyarrow arrow-cpp parquet-cpp -c conda-forge
```

Note: you may need to restart the kernel to use updated packages.

КГ' гѣ Гвбп ўлї®«Ёвм гЄ § го ĩa®Ја →→г.

In [70]:

```
1 import pyarrow.parquet as pq
```

In [71]:

```
1 df1
```

Out[71]:

	A	B
a	1	4
b	2	5
c	3	6

Запись Pandas DataFrame в parquet

DataFrame.to\_parquet(\*\*kwargs)

Write a DataFrame to the binary parquet format.

This function writes the dataframe as a parquet file. You can choose different parquet backends, and have the option of compression.

## Parameters

- path: str or file-like object
- engine: {'auto', 'pyarrow', 'fastparquet'}, default 'auto'
- compression: {'snappy', 'gzip', 'brotli', None}, default 'snappy'
- index: bool, default None If True, include the dataframe's index(es) in the file output.
- partition\_cols: list, optional, default None

In [72]:

```
1 df1.to_parquet('df1.parquet.gzip', compression='gzip')
```

Загрузка Pandas DataFrame данных из parquet:

```
pandas.read_parquet(path, engine='auto', columns=None, **kwargs)
```

Load a parquet object from the file path, returning a DataFrame.

- path : str, path object or file-like object

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. A local file could be:

```
``file://localhost/path/to/table.parquet``.
```

A file URL can also be a path to a directory that contains multiple partitioned parquet files. Both pyarrow and fastparquet support paths to directories as well as file URLs. A directory path could be: ```file://localhost/path/to/tables``` or ```s3://bucket/partition_dir```

If you want to pass in a path object, pandas accepts any ```os.PathLike```.

By file-like object, we refer to objects with a ```read()``` method, such as a file handler (e.g. via builtin ```open``` function) or ```StringIO```.

- engine : {'auto', 'pyarrow', 'fastparquet'}, default 'auto'

Parquet library to use. If 'auto', then the option ```io.parquet.engine``` is used. The default ```io.parquet.engine``` behavior is to try 'pyarrow', falling back to 'fastparquet' if 'pyarrow' is unavailable.

- columns : list, default=None

If not None, only these columns will be read from the file.

In [73]:

```
1 df1_1 = pd.read_parquet('df1.parquet.gzip')
2
3 df1_1
```

Out[73]:

	A	B
a	1	4
b	2	5
c	3	6

Разделение содержимого parquet на файлы (Partitioned Datasets):

- <https://enpiar.com/arrow-site/docs/python/parquet.html#partitioned-datasets-multiple-files>  
(<https://enpiar.com/arrow-site/docs/python/parquet.html#partitioned-datasets-multiple-files>)

In [74]:

```
1 df1.to_parquet('df1_prt.parquet.gzip', partition_cols = ['A'], compression='gzip')
```

In [75]:

```
1 def list_files(startpath):
2     for root, dirs, files in os.walk(startpath):
3         level = root.replace(startpath, '').count(os.sep)
4         indent = ' ' * 4 * (level)
5         print('{}{}/'.format(indent, os.path.basename(root)))
6         subindent = ' ' * 4 * (level + 1)
7         for f in files:
8             print('{}{}'.format(subindent, f))
```

In [76]:

```
1 # скрипты директории:
2 list_files('./df1_prt.parquet.gzip')
```

```
df1_prt.parquet.gzip/
A=1/
  96cda9032f074ee0a299cf39eb9f7cb6.parquet
  b6db53d9a02047c6926a94d7097c10eb.parquet
  f6f7f03e22ab44ff843ca3b38f97d3d8.parquet
A=2/
  1362e6fefaeb4fa3aa83bc865edc4e39.parquet
  4db82abfce4b48718caf741b5228c7f7.parquet
  5b52e72a86754b629a32e2b08f4d59e5.parquet
A=3/
  40df716c3ff5416cade748f6c58941a2.parquet
  6023123d6df243d6a0b9befc9c37e887.parquet
  7c32c070003c4594b4e03bbbee36e980.parquet
```

In [77]:

```
1 df1_lp = pd.read_parquet('df1_prt.parquet.gzip')
2 df1_lp
```

Out[77]:

	B	__index_level_0__	A
0	4	a	1
1	4	a	1
2	4	a	1
3	5	b	2
4	5	b	2
5	5	b	2
6	6	c	3
7	6	c	3
8	6	c	3

In [ ]:

```
1
```