



# ML Advanced

Практическое занятие - Оптимизация  
кода, parallelization, multiprocessing,  
ускорение pandas, Modin для Pandas



Проверить, идет ли  
запись

# Меня хорошо видно && слышно?



Ставим “+”, если все хорошо “-”,  
если есть проблемы

Тема вебинара

# Практическое занятие - Оптимизация parallelization, multiprocessing Modin для Pandas

Поменять фотографию и текст описания



**Сизов Александр Александрович, PhD**

**Lead Data Scientist**

15 years of software development and research experience;  
8 years of experience in Machine Learning/Deep Learning projects  
development management.

<https://www.linkedin.com/in/aleksandr-sizov-550593b8/>

# Правила вебинара



Активно  
участвуем



Off-topic обсуждаем  
в учебной группе



Задаем вопрос  
в чат или голосом



Вопросы вижу в чате,  
могу ответить не сразу

## Условные обозначени



Индивидуально



Время, необходимое  
на активность



Пишем в чат



Говорим голосом

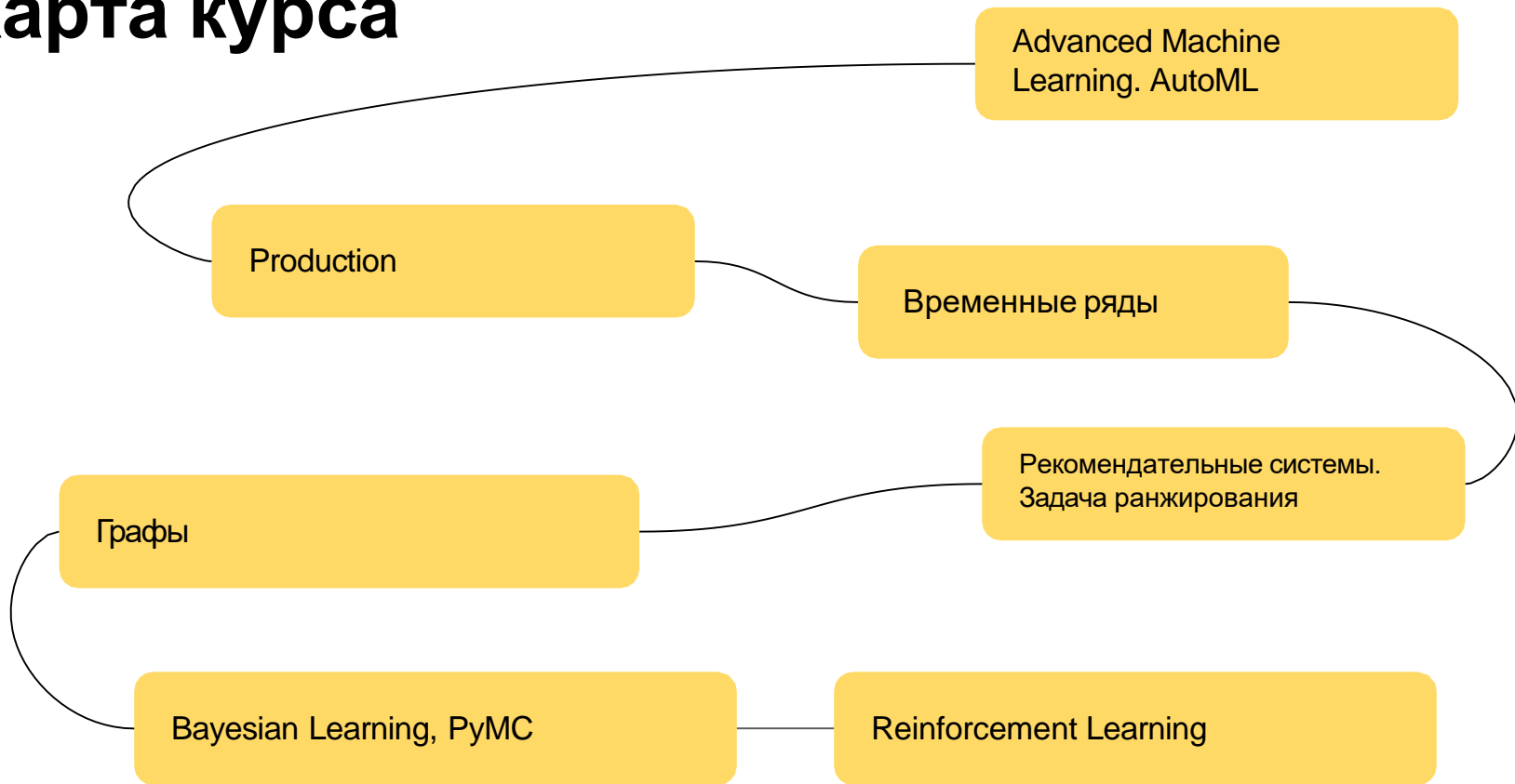


Документ



Ответьте себе или  
задайте вопрос

# Карта курса



# Маршрут вебинара



Профайлинг

Параллелизуем: Threading & multiprocessing

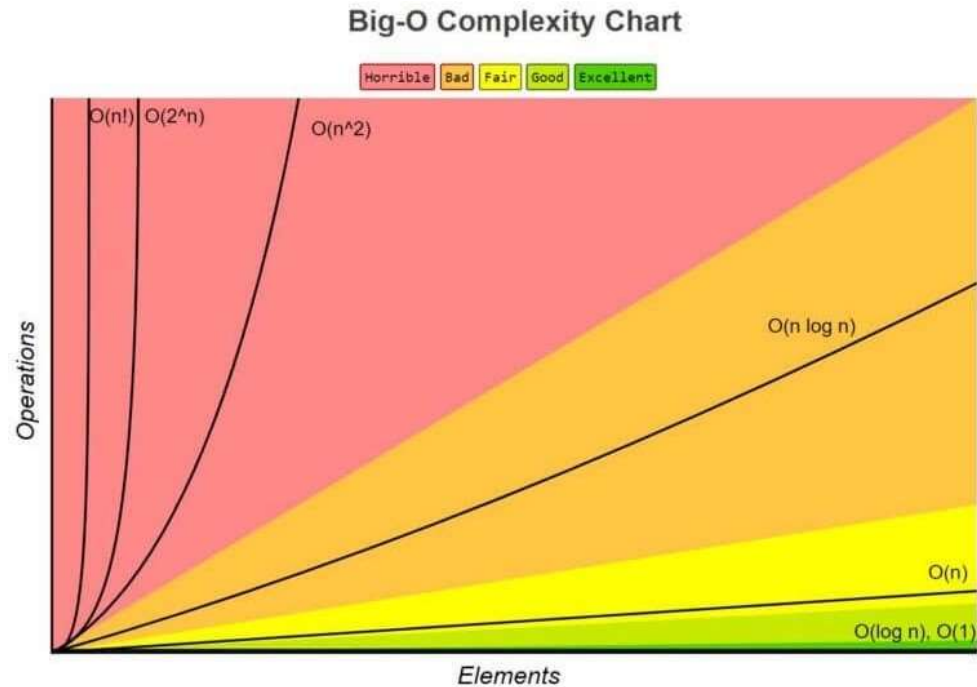
Ускоряем Python: Cython, numba

Pandas на кластере: Dask, Ray, Modin

Практика


Рефлексия

# Нотация $O(n)$



<https://www.bigocheatsheet.com/>

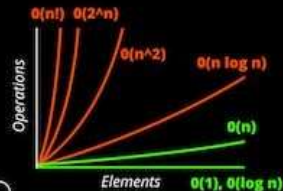
# LEGEND

TIME Complexity  VS.  SPACE Complexity

 Good  Fair  Bad  
 Good  Fair  Bad

## <BIG-O-CHEATSHEET>

[www.bigocheatsheet.com](http://www.bigocheatsheet.com)



### DATA STRUCTURE Operations

### ARRAY SORTING Algorithms

DATA Structure	TIME Complexity				SPACE Complexity			
	Average				Worst			
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion
Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
Stack	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Queue	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Singly-Linked List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Doubly-Linked List	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$
Skip List	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n \log(n))$
Hash Table	N/A	$O(1)$	$O(1)$	$O(1)$	N/A	$O(n)$	$O(n)$	$O(n)$
Binary Search Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Cartesian Tree	N/A	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	N/A	$O(n)$	$O(n)$	$O(n)$
B-Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$
Red-Black Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$
Splay Tree	N/A	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	N/A	$O(\log(n))$	$O(\log(n))$	$O(n)$
AVL Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
KD Tree	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

ARRAY Algorithms	TIME Complexity				SPACE Complexity			
	Best				Worst			
	Best	Average	Worst	Worst	Best	Average	Worst	Worst
Quicksort	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	$O(\log(n))$				
Mergesort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$				
Timsort	$O(n)$	$O(n \log(n))$	$O(n \log(n))$	$O(1)$				
Heapsort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$	$O(1)$				
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$				
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$				
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$				
Tree Sort	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$	$O(n)$				
Shell Sort	$O(n \log(n))$	$O(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$				
Bucket Sort	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n)$				
Radix Sort	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$				
Counting Sort	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(k)$				
Cubesort	$O(n)$	$O(n \log(n))$	$O(n \log(n))$	$O(n)$				



# Python complexities

Lists [a, b, c, ...]	
<code>mylist.append(val)</code>	$O(1)$
<code>mylist[i]</code>	$O(1)$
<code>val in mylist</code>	$O(N)$
<code>for val in mylist:</code>	$O(N)$
<code>mylist.sort()</code>	$O(N \log N)$



Pro-tip: replace lists with sets

Dicts {k:v, ...}	
<code>mydict[key] = val</code>	$O(1)$
<code>mydict[key]</code>	$O(1)$
<code>key in mydict</code>	$O(1)$
<code>for key in mydict:</code>	$O(N)$

Sets {a, b, c, ...}	
<code>myset.add(val)</code>	$O(1)$
<code>val in myset</code>	$O(1)$
<code>for val in myset:</code>	$O(N)$

```
# 0.162 s (with 10000 people [1,100] years old)
def count_living_per_year(population: list[tuple[int, int]]) -> dict[int, int]:
    living_per_year = {}
    for birth, death in population:
        if birth in living_per_year:
            living_per_year[birth] += 1
        else:
            living_per_year[birth] = 1
        if death in living_per_year:
            living_per_year[death] -= 1
        else:
            living_per_year[death] = -1
    for year in range(min(living_per_year), max(living_per_year)):
        living_per_year[year] += living_per_year.get(year - 1, 0)
    living_per_year.pop(max(living_per_year))
    return living_per_year
```

Junior

```
# 0.118 s (with 10000 people [1,100] years old)
def count_living_per_year(population: list[tuple[int, int]]) -> dict[int, int]:
    living_per_year = collections.defaultdict(int)
    for birth, death in population:
        living_per_year[birth] += 1
        living_per_year[death] -= 1
    min_, max_ = min(living_per_year), max(living_per_year)
    for year in range(min_, max_):
        living_per_year[year] += living_per_year[year - 1]
    del living_per_year[min_ + 1], living_per_year[max_]
    return living_per_year
```

Intermediate

```
# 0.110 s (with 10000 people [1,100] years old)
def count_living_per_year(population: list[tuple[int, int]]) -> dict[int, int]:
    births = collections.Counter(birth for birth, _ in population)
    deaths = collections.Counter(death for _, death in population)
    living_per_year = births
    for year in range(min(births), max(deaths)):
        living_per_year[year] += living_per_year[year - 1] - deaths[year]
    return living_per_year
```

Senior

```
# 0.093 s (with 10000 people [1,100] years old)
def count_living_per_year(population: list[tuple[int, int]]) -> dict[int, int]:
    births, deaths = zip(*population)
    births_counter = collections.Counter(births)
    deaths_counter = collections.Counter(deaths)
    living_per_year = collections.Counter()
    for year in range(min(births_counter), max(deaths_counter)):
        delta_living = births_counter[year] - deaths_counter[year]
        living_per_year[year] = living_per_year[year - 1] + delta_living
    return living_per_year
```

Expert

# Работа со списками

```
numbers = []
with open("file.txt") as f:
    group = []
    for line in f:
        if line == "\n":
            numbers.append(group)
            group = []
        else:
            group.append(int(line.rstrip()))
    # append the last group because if
    # line == "\n" will not be True for
    # the last group
    numbers.append(group)
```

```
with open("file.txt") as f:
    # split input into groups based on
    # empty lines
    groups =
    f.read().rstrip().split("\n\n")
    # convert all the values in the
    # groups into integers
    nums = [list(map(int,
    (group.split()))) for group in
    groups]
```

# Структуры данных для работы с последовательностями

- **lists** для упорядоченных и изменяемых последовательностей.
- **sets** для неупорядоченных, уникальных и изменяемых последовательностей.

Эффективны для проверки вхождений.

- **tuples** для упорядоченных и неизменяемых последовательностей. Кортежи быстрее и требуют меньше памяти, чем списки.

# Enum вместо if-elif-else

```
def points_per_shape(shape: str) ->
int:
    if shape == 'X':
        return 1
    elif shape == 'Y':
        return 2
    elif shape == 'Z':
        return 3
    else:
        raise ValueError('Invalid
shape')
```

```
from enum import Enum

class ShapePoints(Enum):
    X = 1
    Y = 2
    Z = 3

def points_per_shape(shape: str) ->
int:
    return ShapePoints[shape].value
```

# Вопросы?



Ставим “+”,  
если вопросы есть



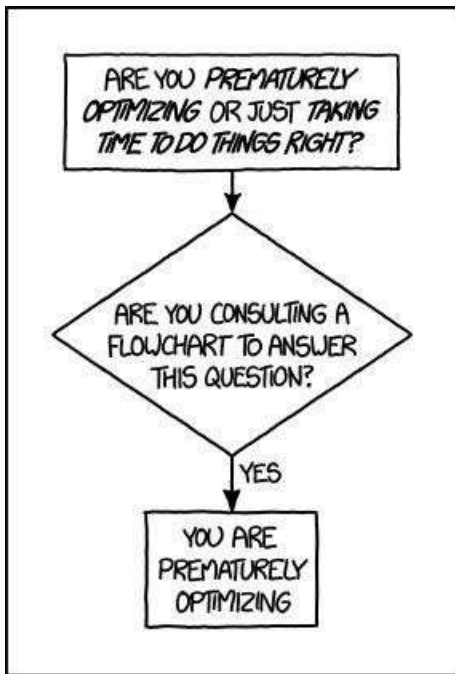
Ставим “-”,  
если вопросов нет



# Профайлинг

# Premature optimization is the roof of all evil

Зачем считать что-то час, если можно десять часов пытаться это ускорить?





# Profiling

```
[20]: !pip install snakeviz

Requirement already satisfied: snakeviz in /home/boris/.local/lib/python3.8/site-packages (2.1.0)
Requirement already satisfied: tornado==2.0 in /home/boris/.local/lib/python3.8/site-packages (from snakeviz) (6.0.4)

[30]: %load_ext snakeviz

The snakeviz extension is already loaded. To reload it, use:
%reload_ext snakeviz

[31]: %snakeviz

times = 100
for i in range(times):
    compute_pawise_products(df)

print(open('prof0', 'r').read())

21941748 function calls (23797648 primitive calls) in 9.810 seconds

Ordered by: cumulative time
List reduced from 210 to 10 due to restriction <10>

ncalls  tottime  perrall  ctime  pcall  filename:lineno(function)
1      0.000    0.000    9.810    9.810 {built-in method builtins.exec}
1      0.020    0.020    9.810    9.810 <string>:1=<module>
188    0.138    0.881    9.782    8.898 <ipython-input-20-5b362972ac1f>:1(compute_pawise_products)
46500  0.570    0.880    7.389    0.880 common.py:49(new method)
46500  0.126    0.000    7.222    0.000 init_.py:495(wrapper)
46500  0.105    0.000    3.890    0.000 init_.py:440(construct_result)
46500  0.303    0.000    3.538    0.000 series.py:183( init_)
46500  0.211    0.000    2.251    0.000 array_ops.py:156(arithmetic_op)
93100  0.215    0.000    1.786    0.000 blocks.py:2907(get_block_type)
100    0.001    0.000    1.567    0.010 frame.py:414( init_)
```

- 1) Перед тем, как оптимизировать, поймите где проблема: сделайте профайлинг.
- 2) Оптимизируйте только самое узкое место.

<https://docs.python.org/3/library/profile.html> <https://jiffyclub.github.io/snakeviz/>

<https://github.com/nschloe/tuna>

# Вопросы?

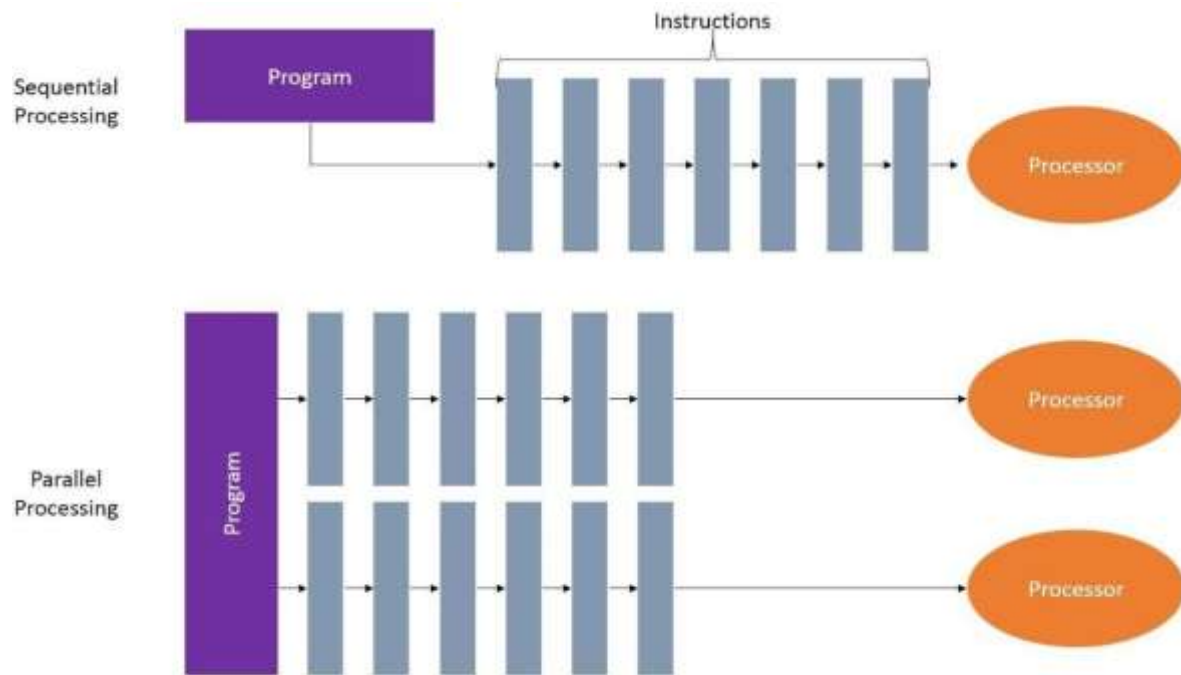


Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет

# Параллельные вычисления



<https://towardsdatascience.com/an-intro-to-parallel-computing-with-ray-d8503629485>

# А что же GIL?

- GIL это глобальный мьютекс на память в Python.
- Только один поток может использовать память одновременно, а значит любая CPU-bound Python программа по факту однопоточная.
- Но GIL не ограничивает IO операции.

Вывод: **multithreading бесполезен для ускорения вычислений, но не бесполезен в случае IO задержек.**

```
Python
# single_threaded.py
import time
from threading import Thread

COUNT = 50000000

def countdown(n):
    while n > 0:
        n -= 1

start = time.time()
countdown(COUNT)
end = time.time()

print('Time taken in seconds -', end - start)
```

Running this code on my system with 4 cores gave the following output:

```
Shell
$ python single_threaded.py
Time taken in seconds - 8.09024037381146
```

```
Python
# multi_threaded.py
import time
from threading import Thread

COUNT = 50000000

def countdown(n):
    while n > 0:
        n -= 1

t1 = Thread(target=countdown, args=(COUNT//2,))
t2 = Thread(target=countdown, args=(COUNT//2,))

start = time.time()
t1.start()
t2.start()
t1.join()
t2.join()
end = time.time()

print('Time taken in seconds -', end - start)
```

And when I ran it again:

```
Shell
$ python multi_threaded.py
Time taken in seconds - 0.934543032293781
```

<https://realpython.com/python-gil/>

# Multiprocessing vs Multithreading

<b>multiprocessing</b>	<b>multithreading</b>
Задействует несколько ядер CPU	Задействует одно ядро CPU
Все процессы имеют собственную память	Все потоки делят общую память
Не ограничивается GIL	Ограничивается GIL
Позволяет производить вычисления параллельно	Не позволяет производить параллельных вычислений
Выгодно использовать, когда производительность ограничена возможностями CPU	Выгодно использовать, когда производительность ограничена задержками ввода/вывода

Полезные ссылки:

- <https://timber.io/blog/multiprocessing-vs-multithreading-in-python-what-you-need-to-know/>
- <https://stackoverflow.com/questions/3044580/multiprocessing-vs-threading-python>
- <https://medium.com/contentsquare-engineering-blog/multithreading-vs-multiprocessing-in-python-ece023ad55a>

# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “—”,  
если вопросов нет

# Ускоряем Python



# Cython

```
1 # python_functions.py
2 # -----
3 import math
4
5 def f(x):
6     return math.exp(-(x ** 2))
7
8 def integrate_f(a, b, N):
9     s = 0
10    dx = (b - a) / N
11    for i in range(N):
12        s += f(a + i * dx)
13    return s * dx
14
```

```
1 # cython_functions.pyx
2 # -----
3 from libc cimport math
4
5 cdef double f(double x):
6     return math.exp(-(x ** 2))
7
8 def integrate_f(double a, double b, int N):
9     cdef double s = 0
10    cdef double dx = (b - a) / N
11    cdef int i
12    for i in range(N):
13        s += f(a + i * dx)
14    return s * dx
15
```

# Numba

```
[45]: import numba
      from numba import jit, prange

[46]: X = np.random.randn(1000) * 8.342

[50]: @jit(nopython=True) # Do not set parallel=True here or it will crash
      def scalar_sum_numba(X, y):
          running_sum = 0
          for i in range(len(X)):
              running_sum += y * X[i]
          return running_sum

      @jit(nopython=True, parallel=True)
      def pairwise_sum_numba(X):
          running_sum = 0
          for i in prange(len(X)): # Parallelized!
              running_sum += scalar_sum_numba(X, X[i])
          return running_sum

[51]: pairwise_sum_numba(X)

[51]: 69412062.18552035

[52]: %timeit pairwise_sum_numba(X)

566 µs ± 54.2 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

<https://hub.gke2.mybinder.org/user/numba-numba-examples-v1ni0xtg/notebooks/notebooks/basics.ipynb>

# Вопросы?



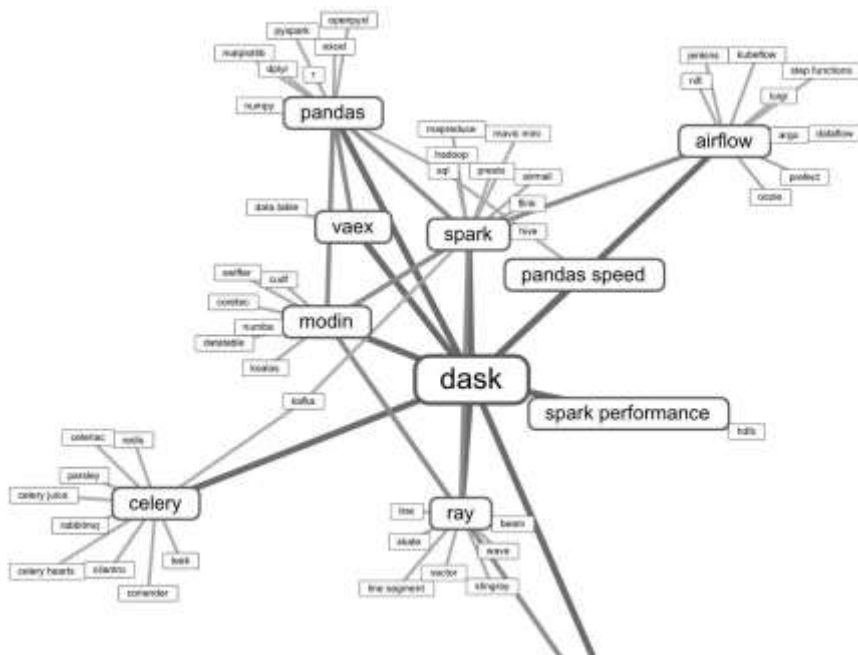
Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет

# Pandas на кластере

# Альтернативы ускорения обработки данных в python



## Native capabilities



- Collections (RDD)
- DataFrames
- SQL
- Streaming
- Graph
- Machine learning



- Collections (Bag)
- DataFrames
- Arrays
- Machine learning
- Custom
  - Tasks - Delayed, Futures
  - Classes - Actors (experimental)



- Model serving (Serve)
- Hyperparameter tuning (Tune)
- Reinforcement learning (RLlib)
- Distributed deep learning training (SGD)
- Custom
  - Tasks - Remote
  - Classes - Actors

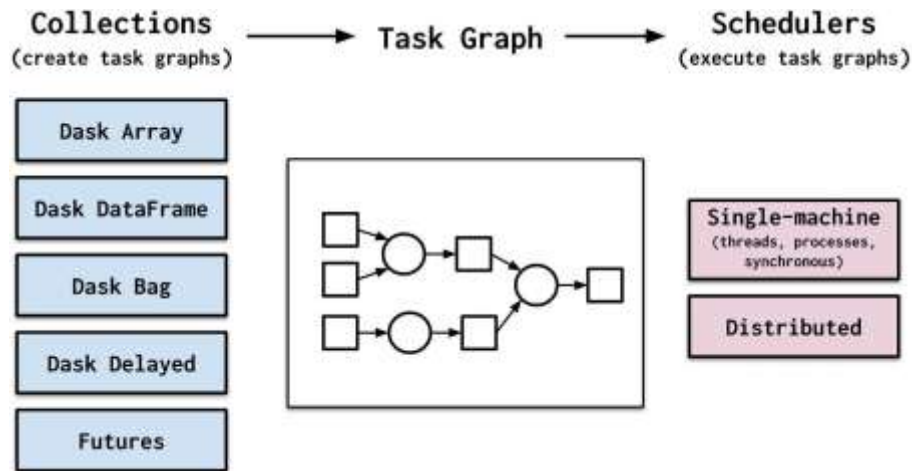
<https://c-nemri.medium.com/spark-dask-ray-choose-the-right-distributed-computing-framework-6b766e177728>

# Сравнение популярных инструментов

	Maturity	Popularity	Ease of Adoption	Scaling ability	Use cases	Main scaling strategy
Dask	A	B	B	1 TB+	Data science / General	Clusters
Vaex	C	C	A	100 GB+	Data science	Lazy loading
RAPIDS (cuDF)	B	C	C	100 GB+	Data science	GPUs
Modin	C	C	A	10 GB+	DataFrame	Clusters
Ray	A	A	B	1 TB+	General	Clusters

# Dask

- Движок для параллельных вычислений, в том числе на кластере.
- Поддерживает важнейшие части pandas API, но не все.
- Питоничный и расширяемый.
- Позволяет работать с датафреймом, который не влезает в оперативную память.



<https://tutorial.dask.org/>



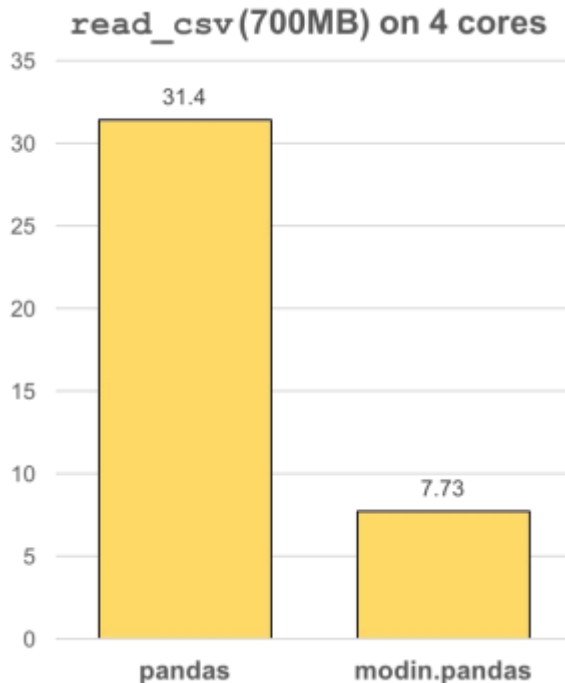
# Modin

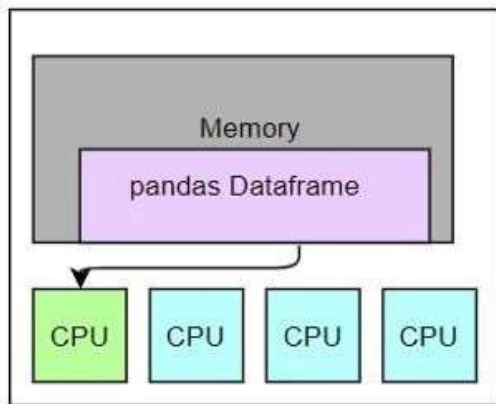
- Pandas-подобный интерфейс/адаптер к Ray или Dask.
- Полностью\* реализует pandas API.
- “Под капотом” использует Dask или Ray для распределения вычислений.
- Пока не очень взрослый.



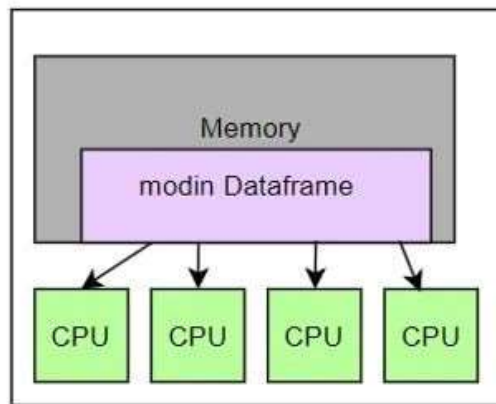
# MODIN

<https://modin.readthedocs.io/en/latest/#>



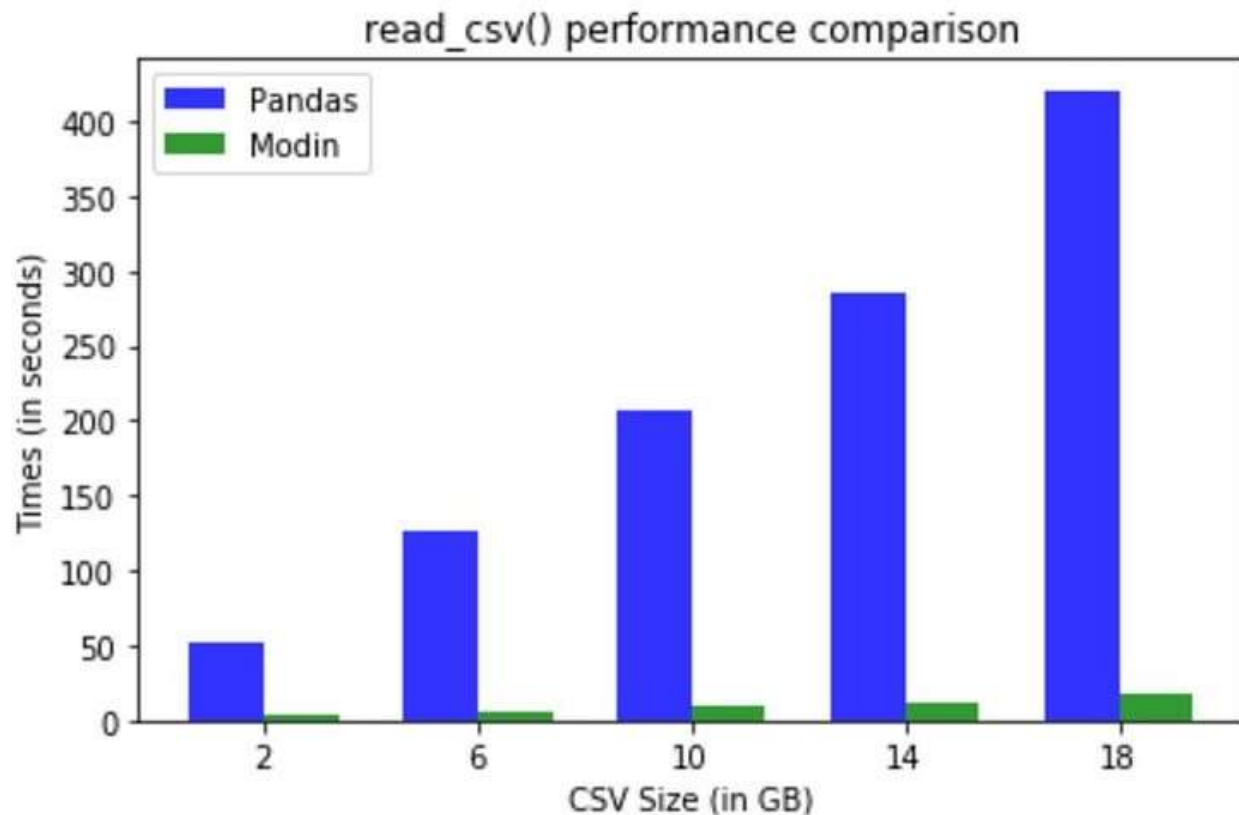


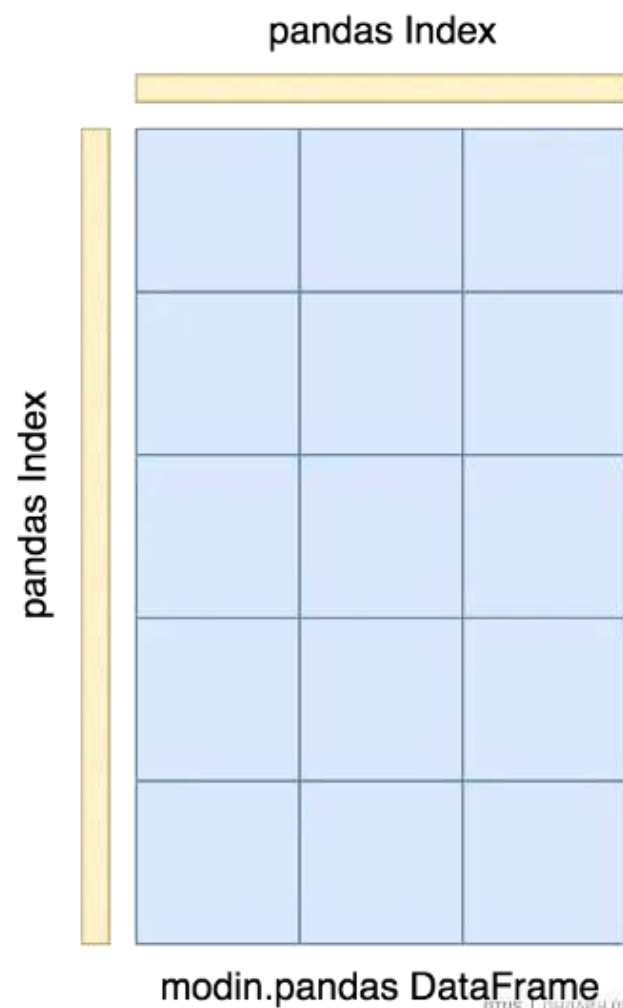
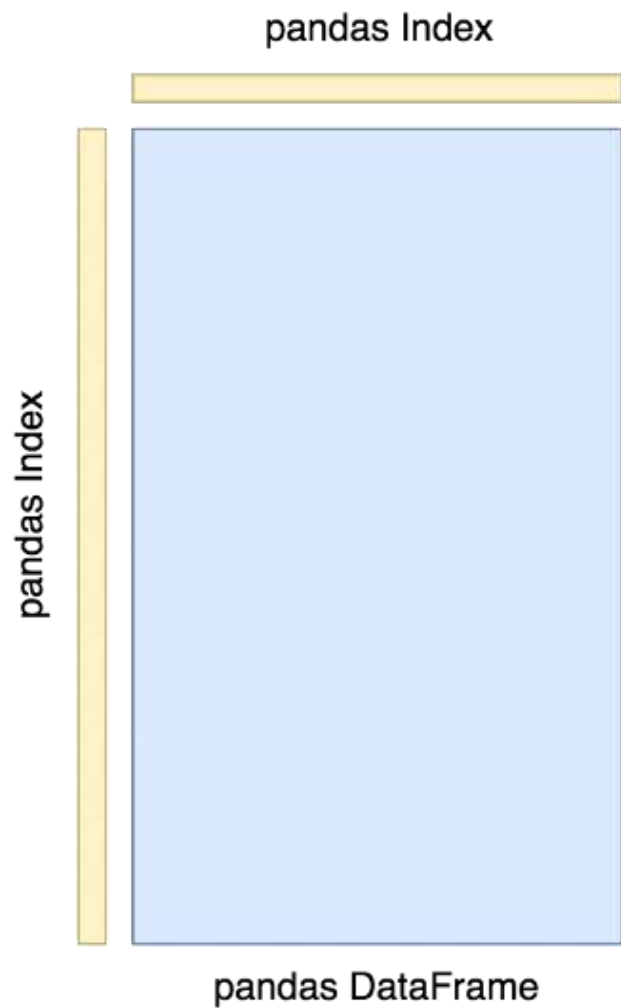
Cores which are  
idle



Full utilisation of  
Cores

зависимость времени работы команды `read_csv` от размера файла на 144-ядерном компьютере с использованием **modin**

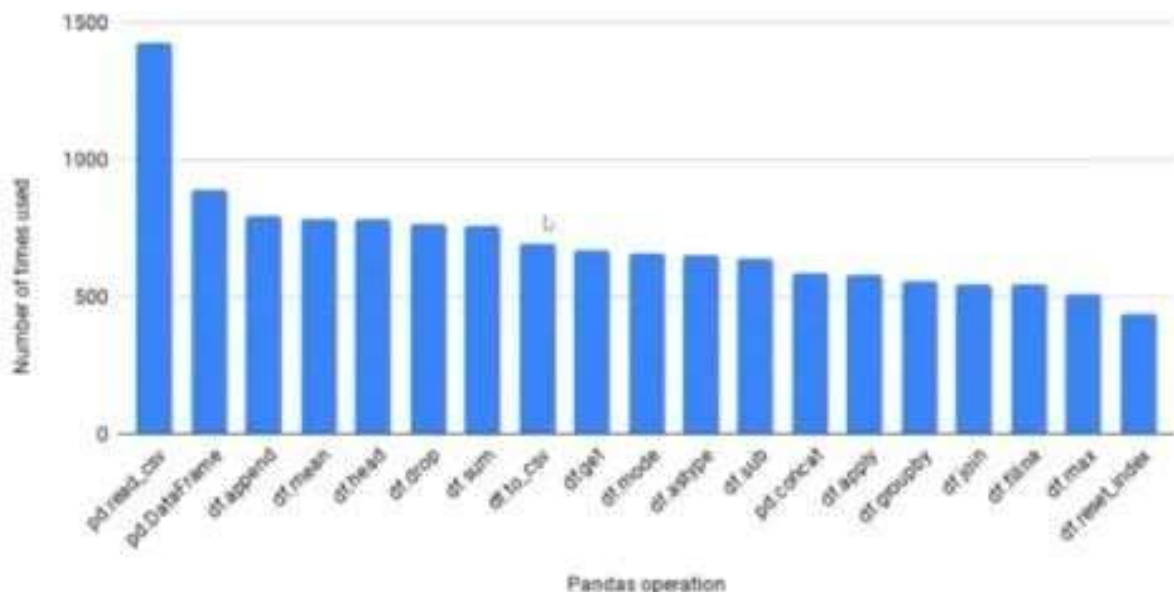




# What do people use in the pandas API?

## Top 20 Most Used Pandas methods in Kaggle

From the top 1800 upvoted scripts and notebooks in Kaggle



# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет

# LIVE

# Рефлексия



# Рефлексия



С какими впечатлениями уходите с вебинара?



Как будете применять на практике то, что узнали на вебинаре?

**Заполните, пожалуйста,  
опрос о занятии  
по ссылке в чате**

Спасибо за внимание!

# Приходите на следующую

Поменять фотографию и текст описания



**Сизов Александр Александрович, PhD**

**Lead Data Scientist**

15 years of software development and research experience;  
8 years of experience in Machine Learning/Deep Learning projects  
development management.

<https://www.linkedin.com/in/aleksandr-sizov-550593b8/>