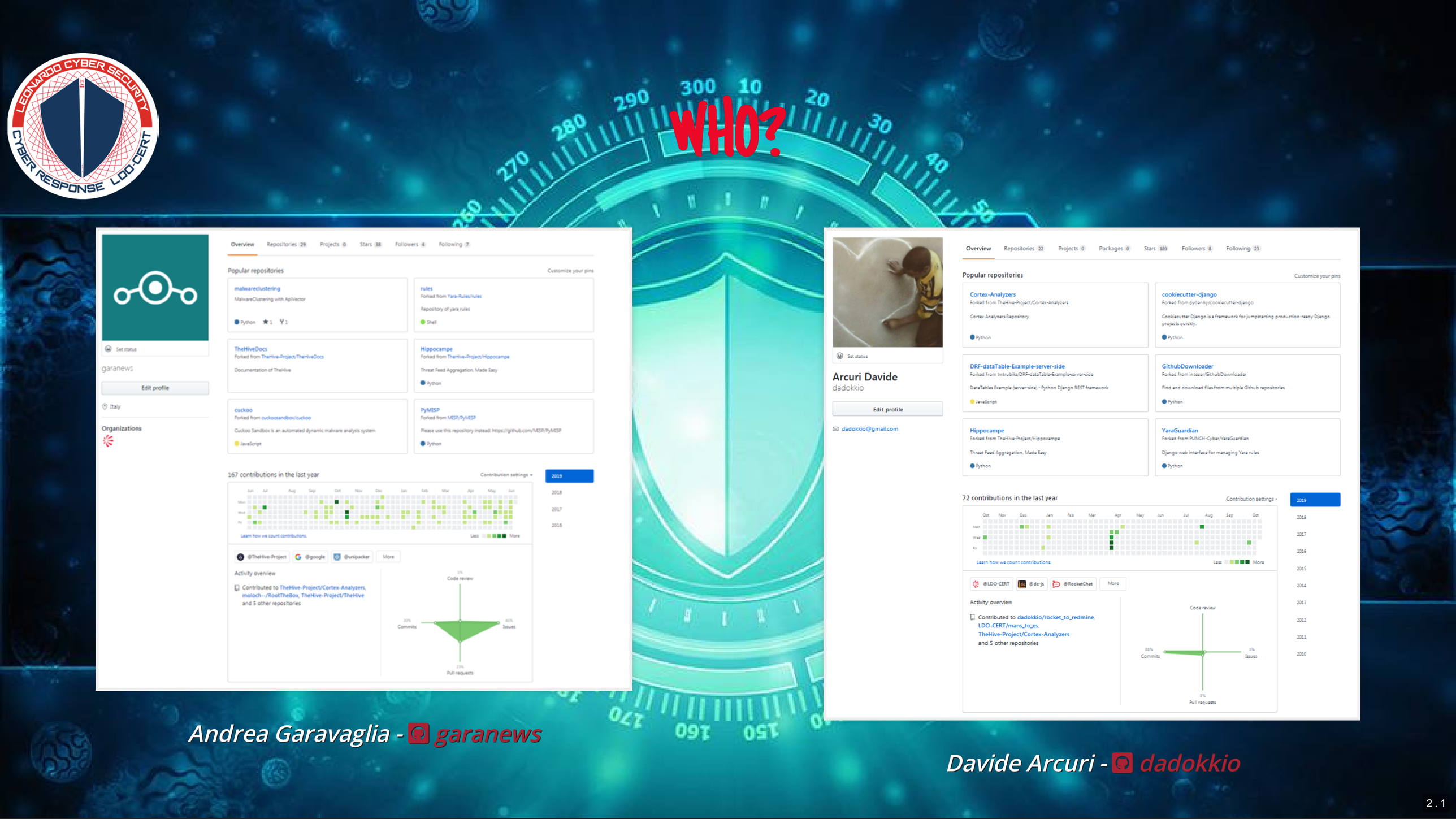

...AND
PERFORMANCE
FOR ALL



garanews

Set status

Edit profile

Italy

Organizations

Overview Repositories 29 Projects 0 Stars 38 Followers 4 Following 7

Popular repositories

Customize your pins

malwareclustering
MalwareClustering with ApiVector
Python ★1 ⑂1

rules
Forked from Yara-Rules/rules
Repository of yara rules
Shell

TheHiveDocs
Forked from TheHive-Project/TheHiveDocs
Documentation of TheHive

Hippocampe
Forked from TheHive-Project/Hippocampe
Threat Feed Aggregation, Made Easy
Python

cuckoo
Forked from cuckoosandbox/cuckoo
Cuckoo Sandbox is an automated dynamic malware analysis system
JavaScript

PyMISP
Forked from MISP/PyMISP
Please use this repository instead: https://github.com/MISP/PyMISP
Python

167 contributions in the last year

Contribution settings

2019 2018 2017 2016

Learn how we count contributions.

Less More

@TheHive-Project @google @unipacker More

Activity overview

Contributed to TheHive-Project/Cortex-Analyzers, maloch--/RootTheBox, TheHive-Project/TheHive and 5 other repositories

Code review 1%
Commits 30%
Issues 46%
Pull requests 23%

Andrea Garavaglia - garanews

Set status

Arcuri Davide

dadokkio

Edit profile

dadokkio@gmail.com

Overview Repositories 22 Projects 0 Packages 0 Stars 189 Followers 8 Following 23

Popular repositories

Customize your pins

Cortex-Analyzers
Forked from TheHive-Project/Cortex-Analyzers
Cortex Analyzers Repository
Python

cookiecutter-django
Forked from pydanny/cookiecutter-django
Cookiecutter Django is a framework for jumpstarting production-ready Django projects quickly.
Python

DRF-dataTable-Example-server-side
Forked from twtrubiks/DRF-dataTable-Example-server-side
DataTables Example (server-side) - Python Django REST framework
JavaScript

GithubDownloader
Forked from intezer/GithubDownloader
Find and download files from multiple Github repositories
Python

Hippocampe
Forked from TheHive-Project/Hippocampe
Threat Feed Aggregation, Made Easy
Python

YaraGuardian
Forked from PUNCH-Cyber/YaraGuardian
Django web interface for managing Yara rules
Python

72 contributions in the last year

Contribution settings

2019 2018 2017 2016 2015 2014 2013 2012 2011 2010

Learn how we count contributions.

Less More

@LDO-CERT @dc-js @RocketChat More

Activity overview

Contributed to dadokkio/rocket_to_redmine, LDO-CERT/mans_to_es, TheHive-Project/Cortex-Analyzers and 5 other repositories

Code review
88% Commits
3% Issues
9% Pull requests

Davide Arcuri - dadokkio

WHY?

The number of malicious samples that appear every day makes manual analysis impractical.

Although these samples belong to a limited number of malware families, it is difficult to categorize them automatically...

WHY?

The number of malicious samples that appear every day makes manual analysis impractical.

Although these samples belong to a limited number of malware families, it is difficult to categorize them automatically...

..AND QUICKLY!!

MISP 2.4.88 released (aka Fuzzy hashing correlation, STIX 1.1 import and many API improvements)

Posted 21 Feb 2018

A new version of MISP 2.4.88 has been released including fuzzy hashing correlation (ssdeep), STIX 1.1 import functionality, various API improvements and many bug fixes

Fuzzy hashing (e.g ssdeep or tlsh) is a commonly used technique used to classify malware,

MYSQL SEARCH OPTIMIZATIONS

Only data with same or double chunksize.

```php
// app/Model/FuzzyCorrelateSsdeep.php

public function query_ssdeep_chunks($hash, $attribute_id){
  $chunks = $this->ssdeep_prepare($hash);
  $result = $this->find('list', array(
    'conditions' => array(
      'AND' => array(
        'OR' => array('FuzzyCorrelateSsdeep.chunk_size' => $chunks[0], 'FuzzyCorrelateSsdeep.chunk_size' => $chunks[0] * 2,),
        'OR' => array('FuzzyCorrelateSsdeep.chunk' => $chunks[1], 'FuzzyCorrelateSsdeep.chunk' => $chunks[2])
    )),
    'fields' => array('FuzzyCorrelateSsdeep.attribute_id', 'FuzzyCorrelateSsdeep.attribute_id')
  ));
  $to_save = array();
  foreach (array(1, 2) as $type) {
    foreach ($chunks[$type] as $chunk) {
      $to_save[] = array('attribute_id' => $attribute_id, 'chunk' => $chunk);
  }}
  $this->saveAll($to_save);
  return $result;
}
```

LEONARDO

MYSQL SEARCH OPTIMIZATIONS

Only data with any seven-character string in common.

```php
// app/Model/FuzzyCorrelateSsdeep.php

public function get_all_7_char_chunks($hash){
  $result = '';
  $results = array();
  for ($i = 0; $i < strlen($hash) - 6; $i++) {
    $current = substr($hash, $i, 7);
    $temp = $current . '=';
    $temp = base64_decode($temp);
    $temp = $temp . "\x00\x00\x00";
    $temp = base64_encode($temp);
    if (!in_array($temp, $results)) {
      $results[] = $temp;
    }
  }
  return $results;
}
```

```
mysql> select count(*) from attributes where type="ssdeep";
     count(*) : 4270

mysql> select count(*) from fuzzy_correlate_ssdeep;
     count(*) : 246285

mysql> select ssdeep from attributes where id=291630;
     ssdeep  :  49152:F0Mnrnb04mvy6e4L ... hxq0v

mysql> select * from fuzzy_correlate_ssdeep;
    +----+--------------+--------------+
    | id | chunk        | attribute_id |
    +----+--------------+--------------+
    |  1 | F0MnrnYAAAA= |       291630 |
    |  2 | 0Mnrnb0AAAA= |       291630 |
    |  3 | Mnrnb04AAAA= |       291630 |
    |  4 | nrnb04kAAAA= |       291630 |
    |  5 | rnb04msAAAA= |       291630 |
    |  6 | nb04mvwAAAA= |       291630 |
    |  7 | b04mvy4AAAA= |       291630 |
    |  8 | 04mvy6cAAAA= |       291630 |
    +----+--------------+--------------+
```

LEONARDO

SSDEEP IMPLEMENTATION

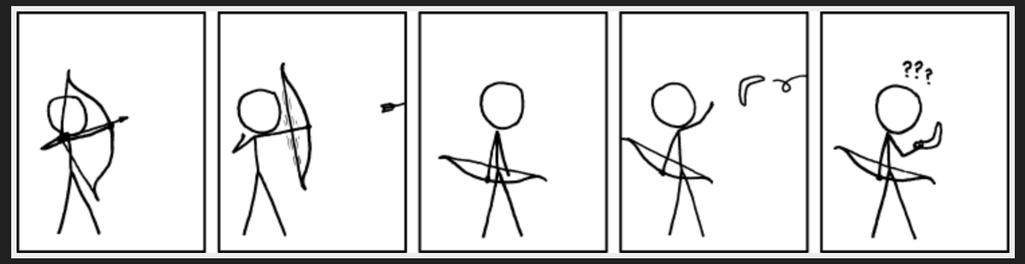
Standard php code used for string similarity

```
// app/Model/Attribute.php

if($a['type'] == 'ssdeep'){
 if(function_exists('ssdeep_fuzzy_compare')){
    $this->FuzzyCorrelateSsdeep = ClassRegistry::init('FuzzyCorrelateSsdeep');
    $fuzzyIds = $this->FuzzyCorrelateSsdeep->query_ssdeep_chunks($a['value'], $a['id']);
    if(!empty($fuzzyIds)){
     $ssdeepIds = $this->find('list', array(
        'recursive' => -1,
        'conditions' => array('Attribute.type' => 'ssdeep', 'Attribute.id' => $fuzzyIds),
        'fields' => array('Attribute.id', 'Attribute.value1')
     ));
     $extraConditions = array('Attribute.id' => array());
     $threshold = !empty(Configure::read('MISP.ssdeep_correlation_threshold'))?
                  Configure::read('MISP.ssdeep_correlation_threshold'):40;
     foreach($ssdeepIds as $k => $v){
        $ssdeep_value = ssdeep_fuzzy_compare($a['value'], $v);
        if($ssdeep_value >= $threshold){$extraConditions['Attribute.id'][] = $k;}
}}}}
```

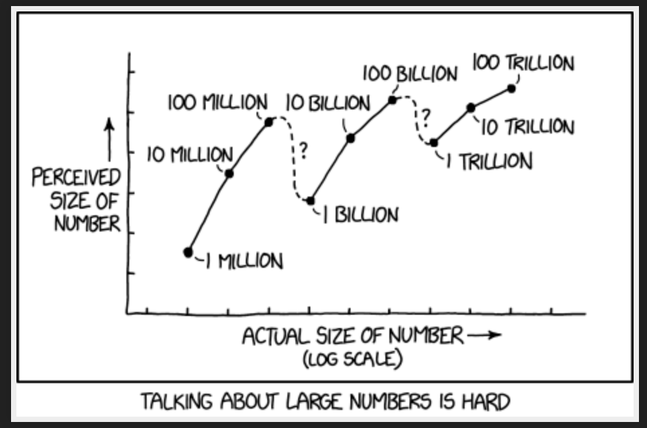
LEONARDO

MAIN ISSUES

Accuracy



Performance



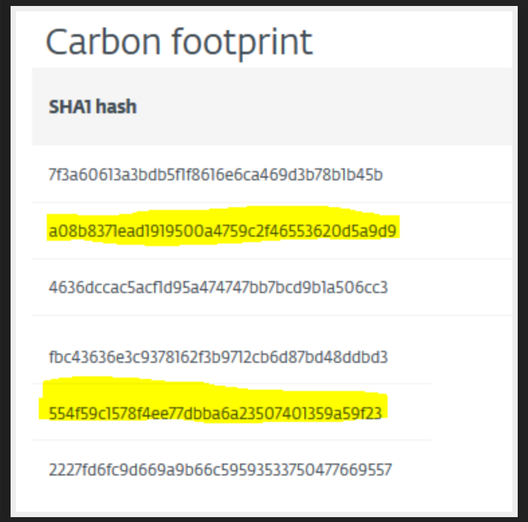
WHERE WE LEFT

https://www.youtube.com/watch?v=nt2l5m8AJpQ

EXAMPLE [COBRA]

Taking couple of malware of same family with different compilation date:

Compilation date	Orchestrator version	Injected library version
2014-02-26	3.71	3.62
2016-02-02	3.77	4.00
2016-03-17	3.79	4.01
2016-03-24	3.79	4.01
2016-04-01	3.79	4.03
2016-08-30	3.81	????
2016-10-05	3.81	????
2016-10-21	3.81	????



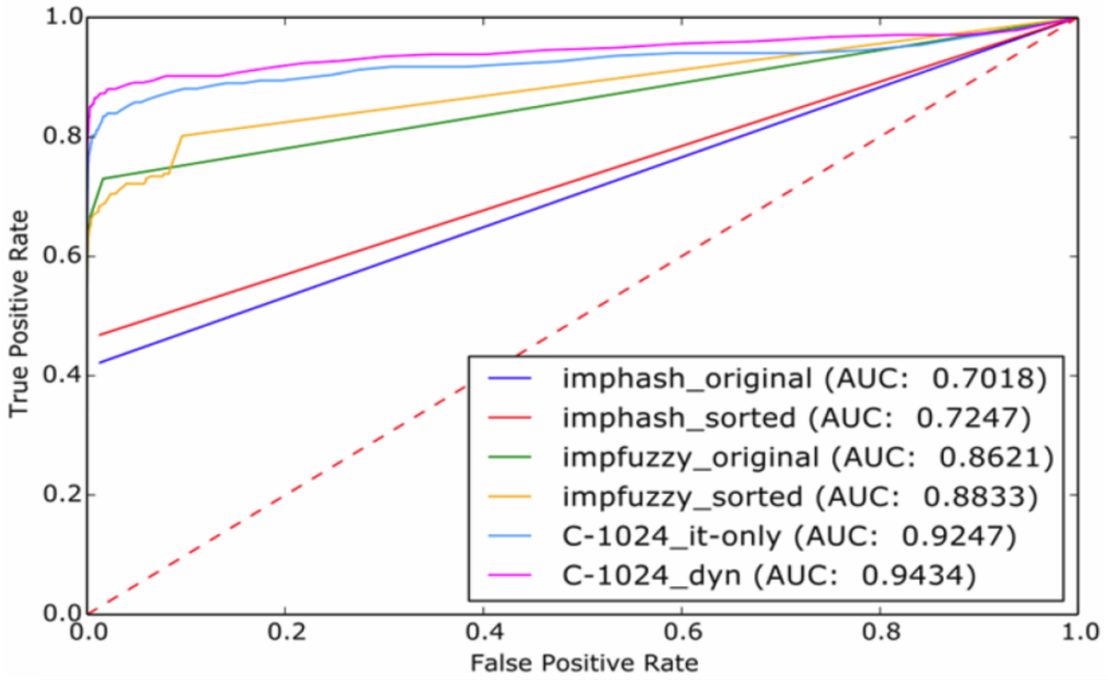
welivesecurity

EXAMPLE [COBRA]

ssdeep vs impfuzzy vs apivector

```
$ ssdeep_compare cobra1 cobra2
        Ssdeep cobra1: 3072:dSjh5NkoeU+KmSrNja7l5jIJeUd9f3bs/uIeUx:dQReUm57l9IcUd9Uu
        Ssdeep cobra2: 6144:N0aBhxDgpfGxzudQwDYm6WSN4Tq+9UjT:77DgRGxzMQwDxY43
        Ssdeep Compare: 0

$ impfuzzy_compare cobra1 cobra2
        ImpFuzzy cobra1: 96:cA8yytCJq5gugxLRz0I3qYCJWMoZpcf+Po4pzdStK3BK:cAh05ULRz0IbC2rSQQ
        ImpFuzzy cobra2: 96:k5JFIEZeVeXxLdaM+fgclikn4xtS23K5bnvaqch2sz:kxheVCjan4xMX59cgsz
        Impfuzzy Compare: 0

$ apivector_compare cobra1 cobra2
        ApiVector cobra1: A23IA13CA7CMA8QgMABAABIA4EIkIIAFAAQ ... oAGAogiMIUDACiBUAAHAkDBahUwwaEBHJMDuGggCYGGINxW?
        ApiVector cobra2: A23IA3+A9CA3BA3CA9QgMABAABIA4EIEIIA ... AEAoACMAQCACiBQAAHAgDBShUwwaEBHJMDuEAgCYGGINxWW?
        ApiVector Compare: 73.0656996953
```

APIVECTORS ACCURACY

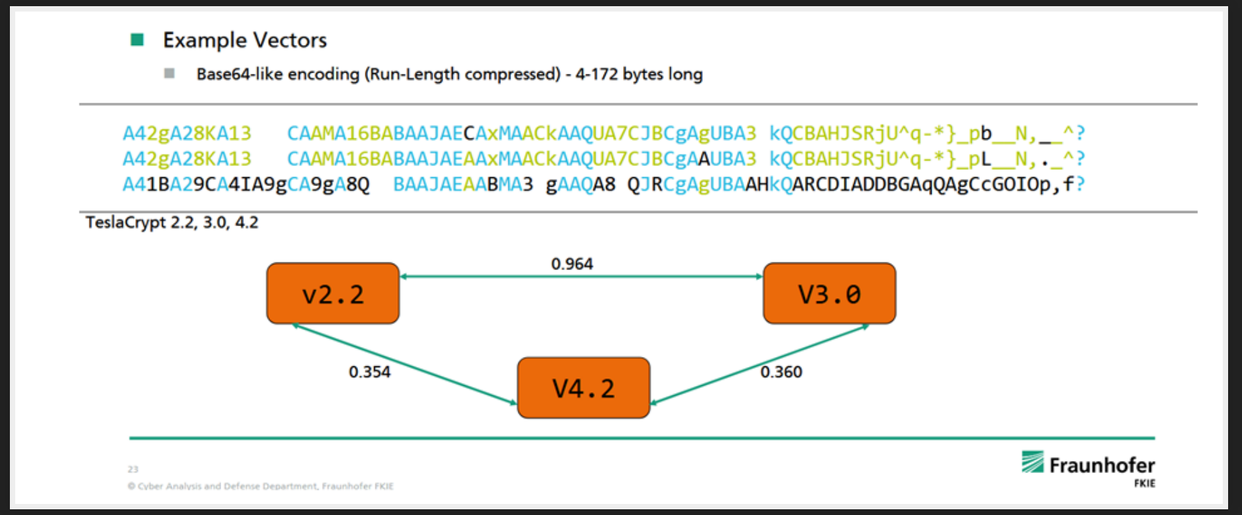


- Data set: Malpedia (2018-05-17)
  - 673 families, 1854 samples
- Comparison with ImpHash, ImpFuzzy
  - Mean Fingerprint sizes:
    - ImpHash: 32 bytes
    - ImpFuzzy: 54.4 bytes
    - ApiVector: 74.3 bytes
  - ApiVector: recoverable info
- Performance @ Thresholds
  - T: 0.18 – 90.18% TPR, 9.45% FPR
  - T: 0.22 – 89.10% TPR, 4.74% FPR
  - T: 0.32 – 86.55% TPR, 0.99% FPR
  - T: 0.55 – 80.72% TPR, 0.09% FPR

24

Fraunhofer FKIE

APIVECTOR?



Apivector are 1024 bit vectors encoding the presence of interesting Windows API functions chosen by semantic context and prevalence.

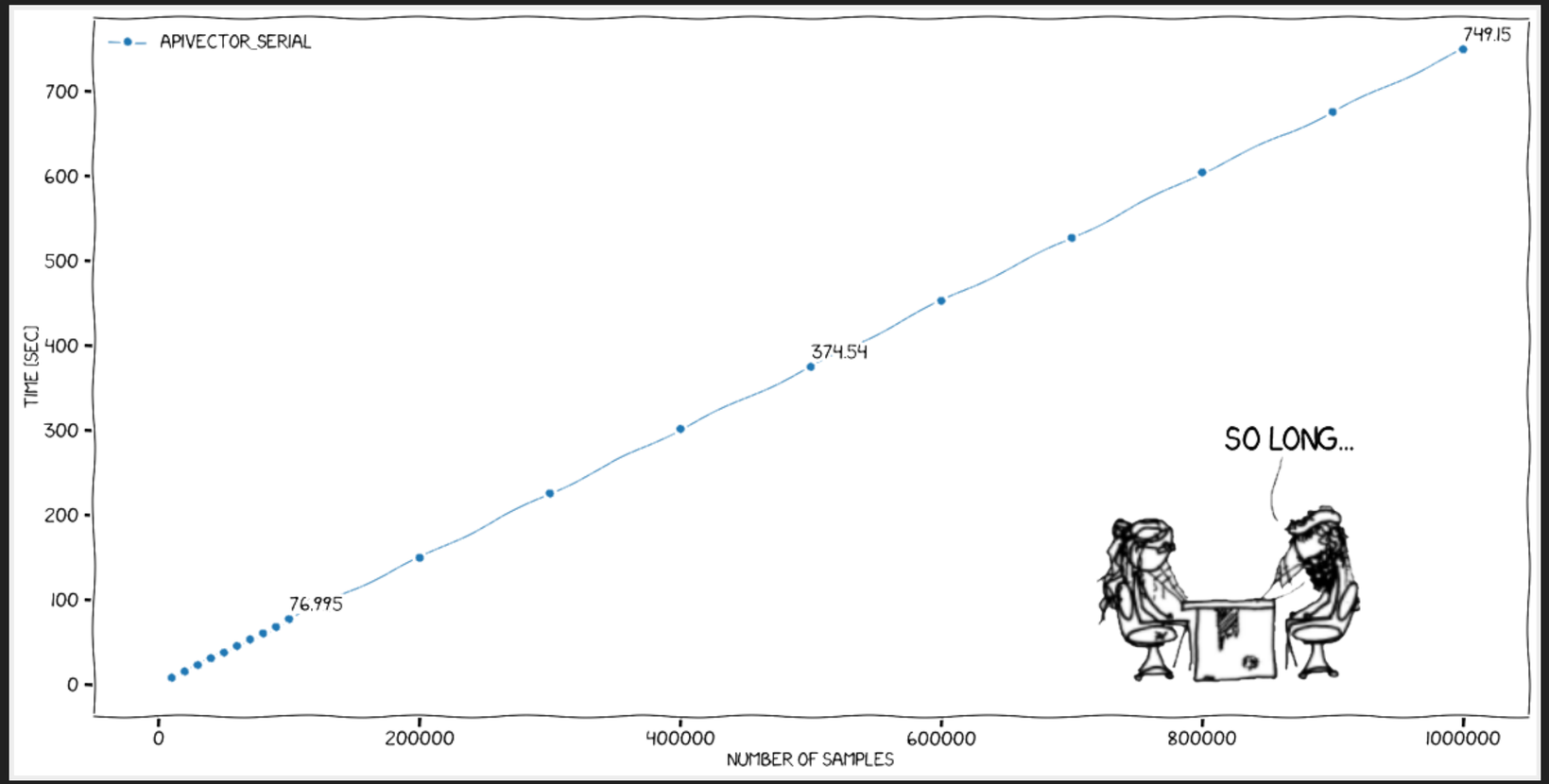
Daniel Plohmann slides

APIVECTORS PERFORMANCES

ApiVectors are very useful for matching samples.
But: does the matching engine scale well?

```
from apiscout.ApiScout import ApiScout

apiscout = ApiScout()
apiscout.setBaseAddress(0)
apiscout.loadWinApi1024('data/winapi1024v1.txt')
vector_a = "A22gA5BA35QA17gACA3Q ... fpT}/Mp-.n?"
vector_b = "A64IA13CA5RA13wAABA5 ... ACIECmth.n?"

range_vals =  1000000

for number in range(range_vals):
    apiscout.matchVectors(vector_a, vector_b)
```

PERFORMANCE PT.1



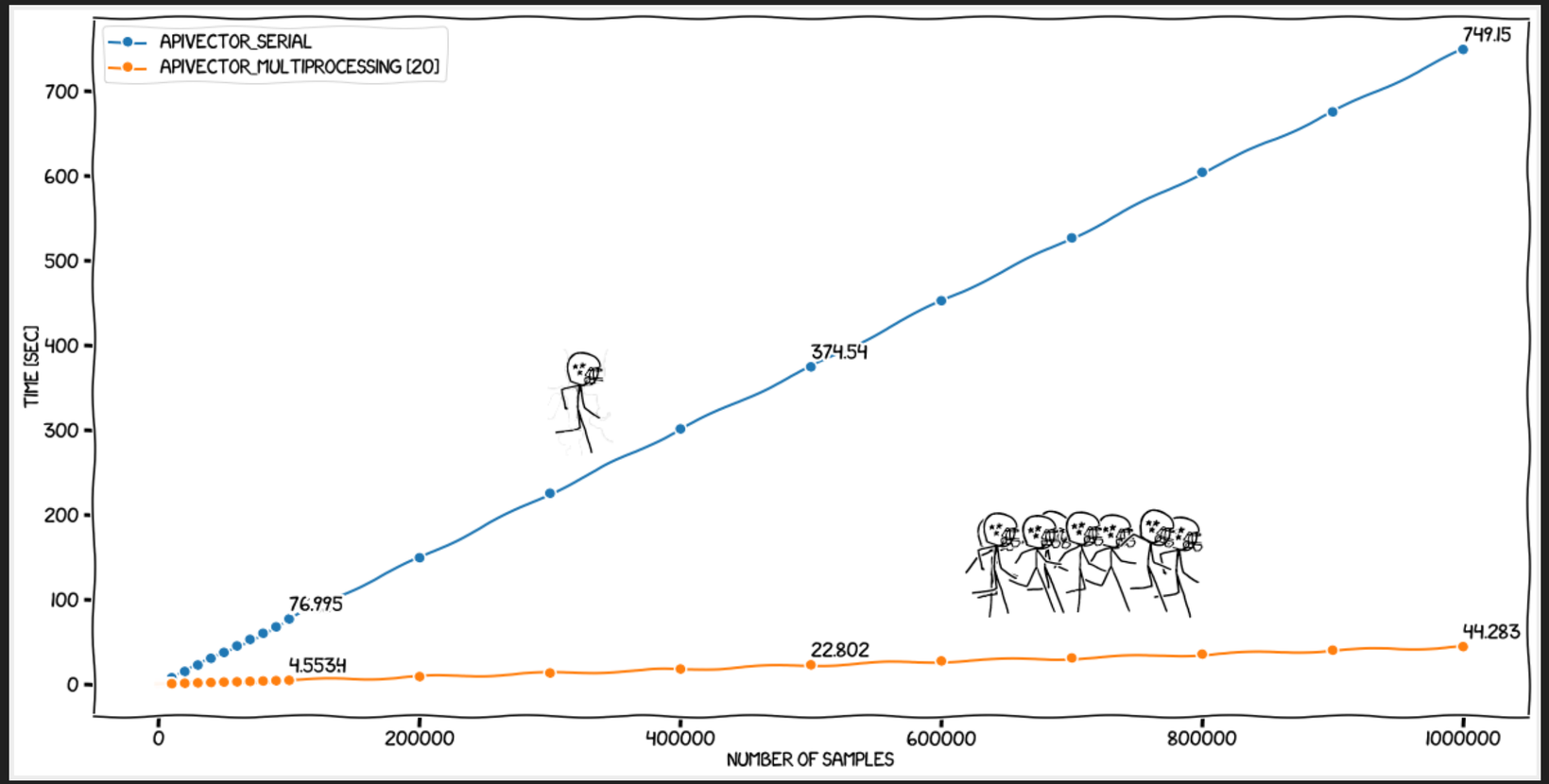
LEONARDO

IMPROVE PERFORMANCE

If I have multiple CPUs why not use them together?

```python
from apiscout.ApiScout import ApiScout
from multiprocessing import Pool, cpu_count

apiscout = ApiScout()
apiscout.setBaseAddress(0)
apiscout.loadWinApi1024('data/winapi1024v1.txt')
vector_a = "A22gA5BA35QA17gACA3Q ... fpT}/Mp-.n?"
vector_b = "A64IA13CA5RA13wAABA5 ... ACIECmth.n?"

range_vals =  1000000

def parallel_apiscout(Q):
    (vector_a, vector_b) = Q
    return apiscout.matchVectors(vector_a, vector_b)

with Pool(cpu_count()) as p:
    p.map(parallel_apiscout, [(vector_a, vector_b) for x in range(range_vals)])
```

PERFORMANCE PT.2



LEONARDO

Medium

IMPROVE PERFORMANCE

To increase the speed of the code is necessary to understand how it works:

```python
def matchVectors(self, vector_a, vector_b):
    # ensure binary representation and apply weights
    if len(vector_a) != 1024:
        vector_a = self.decompress(vector_a)
    vector_a = self._apply_weights(vector_a)
    if len(vector_b) != 1024:
        vector_b = self.decompress(vector_b)
    vector_b = self._apply_weights(vector_b)
    # calculate Jaccard index
    intersection_score = 0
    union_score = 0
    jaccard_index = 0
    for offset in range(len(vector_a)):
        intersection_score += vector_a[offset] & vector_b[offset]
        union_score += vector_a[offset] | vector_b[offset]
    if union_score > 0:
        jaccard_index = 1.0 * intersection_score / union_score
    return jaccard_index
```

W Jaccard Index

The Jaccard index, also known as Intersection over Union and the Jaccard similarity coefficient (originally given the French name coefficient de communauté by Paul Jaccard), is a statistic used for gauging the similarity and diversity of sample sets.

The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

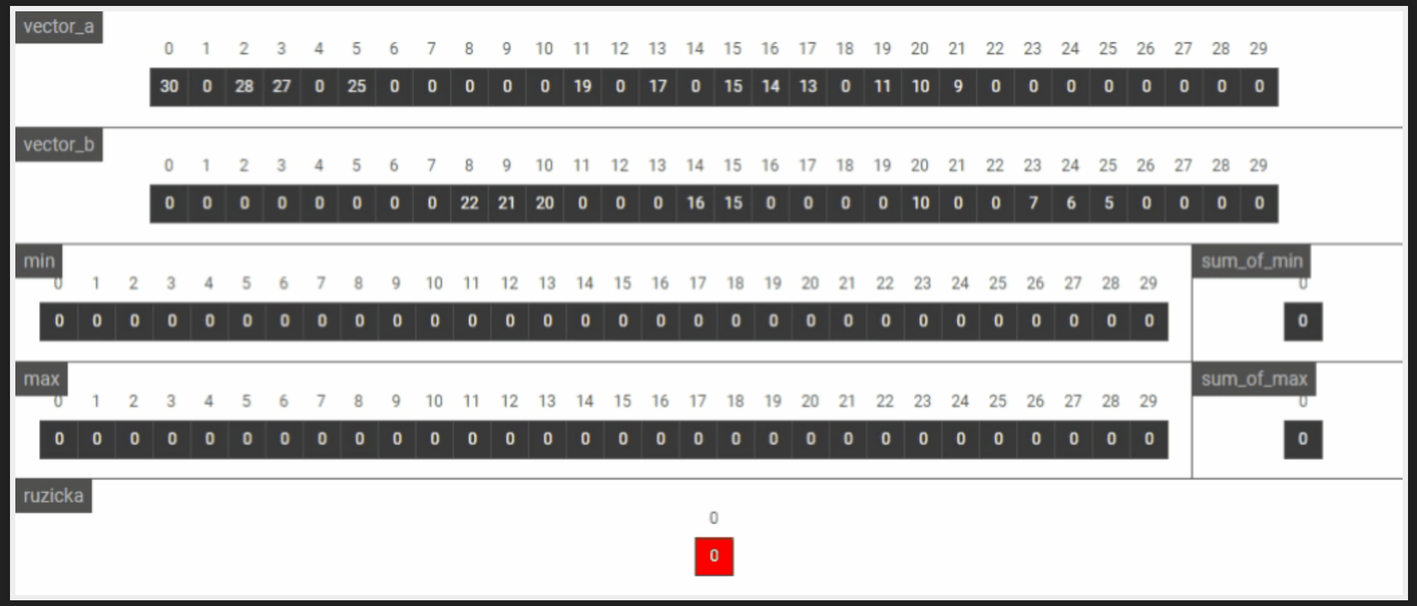
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

If x and y are two vectors with all real > 0, then their Jaccard similarity coefficient (also known then as Ruzicka similarity) is defined as:

$$J_W(\mathbf{x},\mathbf{y}) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)},$$

LEONARDO

Ruzicka similarity

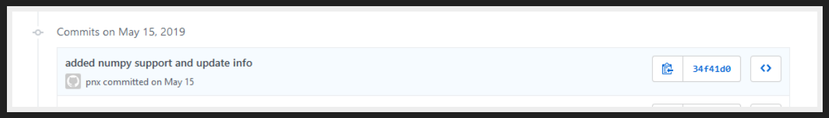


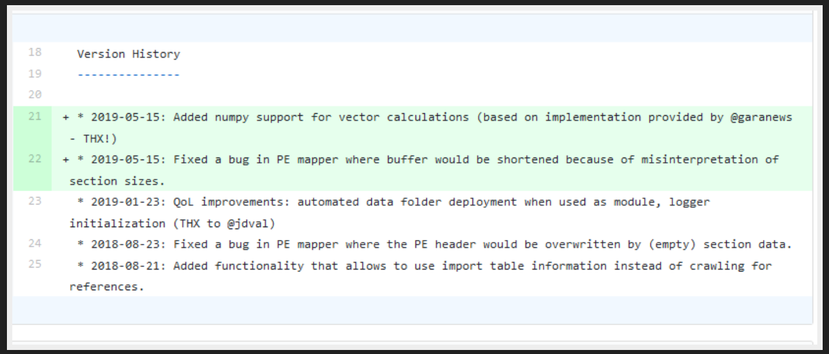
INTRODUCING NUMPY

```python
def matchVectors(self, vector_a, vector_b):
    # ensure binary representation and apply weights
    if not self._isDecompressed(vector_a):
        vector_a = self.decompress(vector_a)
    vector_a = self._apply_weights(vector_a)
    if not self._isDecompressed(vector_b):
        vector_b = self.decompress(vector_b)
    vector_b = self._apply_weights(vector_b)
    # calculate Jaccard index
    if NUMPY_AVAILABLE:
        maxPQ = np.sum(np.maximum(vector_a, vector_b))
        return np.sum(np.minimum(vector_a, vector_b)) / maxPQ
    intersection_score = 0
    union_score = 0
    jaccard_index = 0
    for offset in range(len(vector_a)):
        intersection_score += vector_a[offset] & vector_b[offset]
        union_score += vector_a[offset] | vector_b[offset]
    if union_score > 0:
        jaccard_index = 1.0 * intersection_score / union_score
    return jaccard_index
```

Our implementation has been accepted and merged in apivector source code:

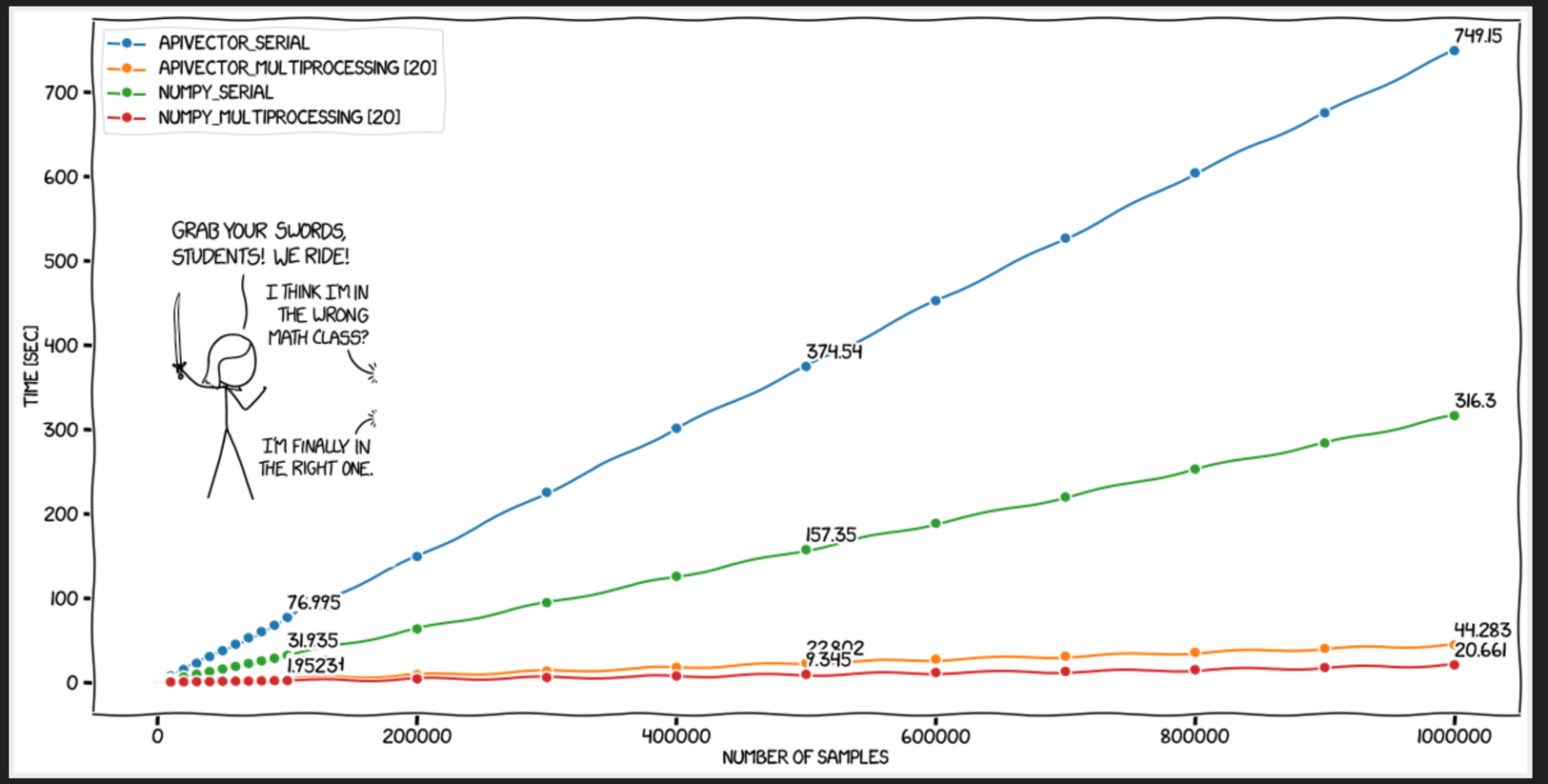
Pull Request merged





LEONARDO

PERFORMANCE PT.3

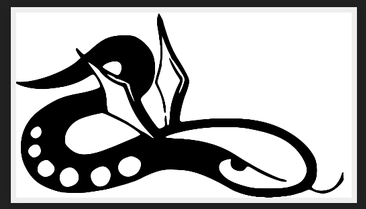


Hard

CROSSING THE NATIVE CODE FRONTIER

The numexpr package supplies routines for the fast evaluation of array expressions elementwise by using a vector-based virtual machine.

Cython is an optimising static compiler for both the Python programming language and the extended Cython programming language.



Pythran is a Python to c++ compiler for a subset of the Python language, with a focus on scientific computing.

pybind11 is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code.

Numba is an open source JIT compiler that translates a subset of Python and NumPy code into fast machine code.

LEONARDO

INTRODUCING CYTHON

```python
import numpy as np
cimport numpy as np
from multiprocessing import Pool, cpu_count

# n_decompress & _n_apply_weights are the same function used by apivector

cdef double ruzicka(x, y):
    cdef int[:] ix, iy, vector_a, vector_b
    if len(x) != 1024:
        ix = n_decompress(x)
    vector_a = _n_apply_weights(ix)
    if len(y) != 1024:
        iy = n_decompress(y)
    vector_b = _n_apply_weights(iy)
    cdef double maxPQ = np.sum(np.maximum(vector_a, vector_b))
    return np.sum(np.minimum(vector_a, vector_b))/maxPQ

def run_cython(Q):
    (vector_a, vector_b) = Q
    return ruzicka(vector_a, vector_b)

with Pool(cpu_count()) as p:
    p.map(run_cython, [(vector_a, vector_b) for x in range(range_vals)])
```

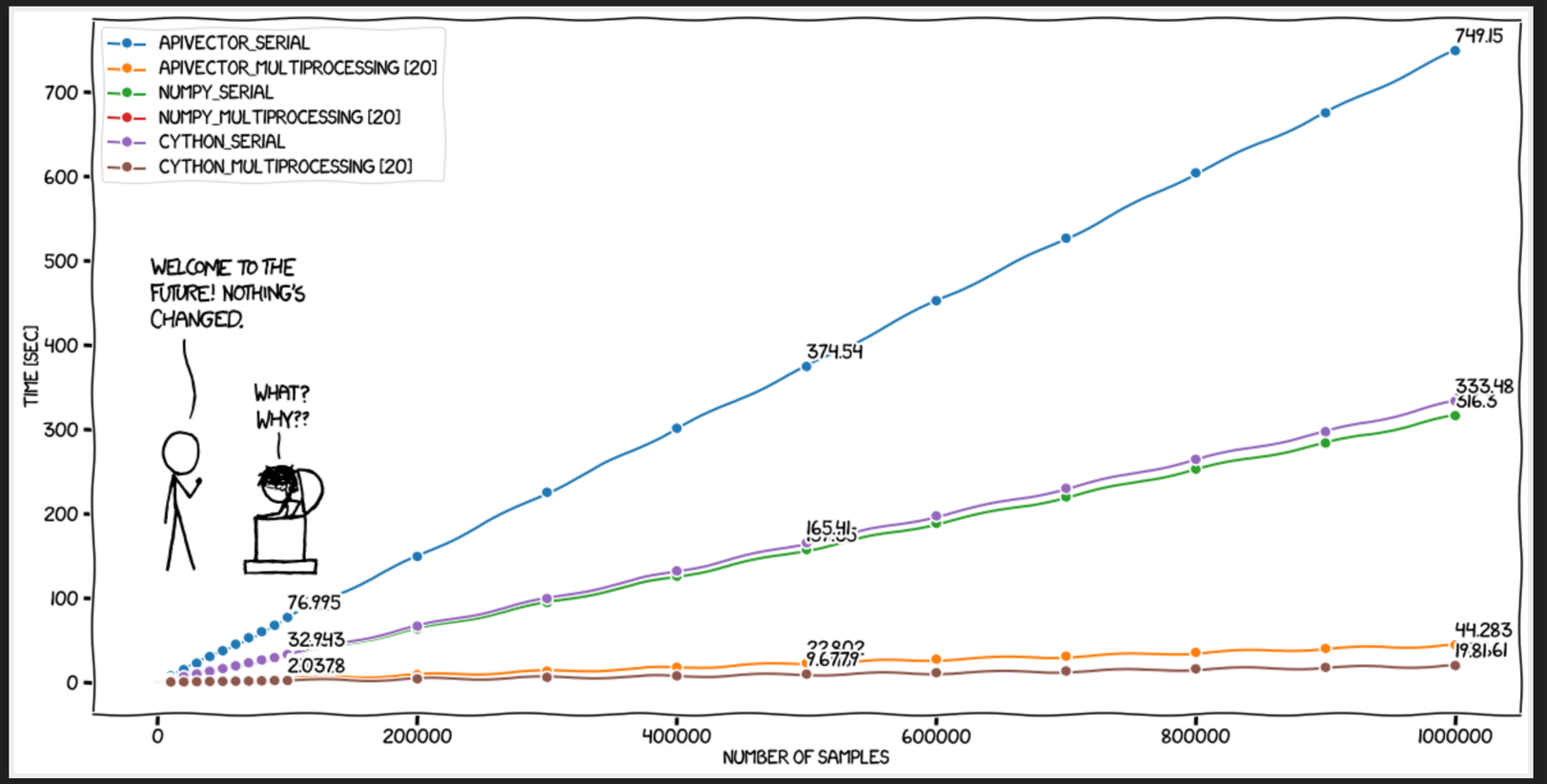
Generated by Cython 0.28.5

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

Raw output: ruzicka.c

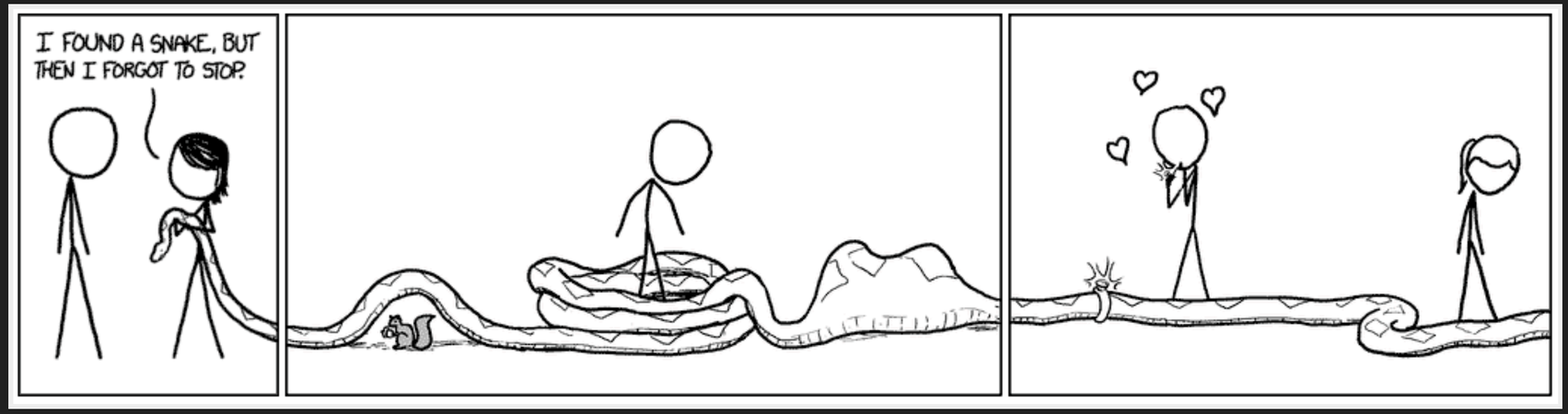
```
+001: # cython: profile=True
 002:
+003: import numpy as np
+004: import re
 005: cimport numpy as np
 006: cimport cython
 007: from cython.parallel import prange
 008: cimport openmp
 009:
 010: from cython.view cimport array as cvarray
 011:
 012:
+013: bc = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz@}]^+-*/?,._'
 014:
+015: cdef int[:] _n_vector_ranks_only  = np.arange(1023, -1, -1, dtype=np.int32)
+016: _bin2base64 = {"{:06b}".format(i): base64char for i, base64char in enumerate(bc)}
+017: _base642bin = {v: k for k, v in _bin2base64.items()}
 018:
+019: cdef int[:] _n_apply_weights(int[:] vector):
+020:     return np.multiply(vector, _n_vector_ranks_only)
 021:
 022: @cython.boundscheck(False)
 023: @cython.wraparound(False)
+024: cdef double ruzicka_nc(int[:] x, int[:] y):
 025:     cdef int[:] vector_a, vector_b
+026:     vector_a = _n_apply_weights(x)
+027:     vector_b = _n_apply_weights(y)
+028:     cdef double maxPQ = np.sum(np.maximum(vector_a, vector_b))
+029:     return np.sum(np.minimum(vector_a, vector_b))/maxPQ
 030:
+031: def _decompress_get(data):
+032:     for match in re.finditer(r"(?P<char>.)((?P<count>\d+))?", data):
+033:         if not match.group("count"):
+034:             yield match.group("char")
 035:         else:
+036:             yield match.group("char") * int(match.group("count"))
 037:
 038:
+039: cdef int[:]  n_decompress(compressed_vector):
 040: #    cdef char* decompressed_b64, vectorized
+041:     decompressed_b64 = "".join(_decompress_get(compressed_vector))
+042:     vectorized = "".join(_base642bin[c] for c in decompressed_b64)[:-2]
+043:     cdef int[:] as_binary = np.fromiter(vectorized, np.int32)
+044:     return as_binary
```

PERFORMANCE PT.4



NEW BOTTLENECKS

- string: it's necessary to store data as string and decompress them?
- Storage: we spend more time retrieving data than processing them
- RAM: what if my apivector data don't fit in memory?



WHY STRINGS?

```
>>> # STEP 1 - THE ORIGINAL COMPRESSED VECTOR
...
>>> compressed_vector = "A8gAgAFAIA3gA7IA4EAACA7CQA4QA8QABA3EA6FAEA5CA3IA69BAEAABAABA10"
>>> print(compressed_vector)
A8gAgAFAIA3gA7IA4EAACA7CQA4QA8QABA3EA6FAEA5CA3IA69BAEAABAABA10
>>>
```

asciinema

LEONARDO

HELP NEEDED!

Looking for some inspiration on stackoverflow, good result but better avoid string

1 Answer — active | oldest | votes

2

It looks like you're struggling with getting the nth bit of an array (essentially doing what `np.unpackbits` does).

The nth bit is contained within the `n//8` byte (I'm using the `//` divide-and-round-down operator). You can access an individual bit in a byte doing a "bitwise and" (`&`) with `1<<m` (one bitshifted by `m`). That will give you the number `2**(m-1)` , and you really just care if it's 0 or not.

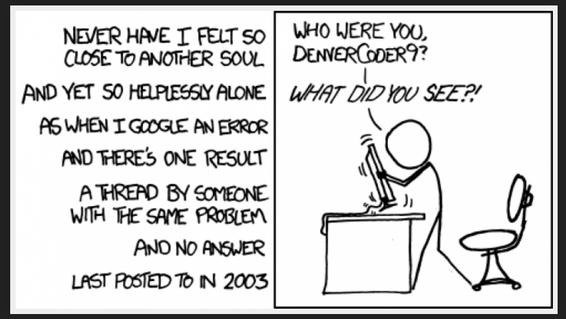
So assuming that `vector_b` is a `np.int8_t` memoryview, you can do:

```
byte_idx = n//8
bit_idx = n%8 # remainder operator
bitmask = 1<<bit_idx
bit_is_true = 1 if (vector_b[byte_idx]&bitmask) else 0
```

You need to put that in a loop and `cdef` the types of the variables.

share edit — edited Feb 10 at 22:31 — answered Feb 9 at 9:39

DavidW
15.6k ● 1 ● 24 ● 44



1 Answer — active | oldest | votes

0

You can get a pretty good speed-up on the second part of the problem (which you're doing through a dictionary lookup) using Numpy alone. I've replaced the dictionary lookup by indexing into a Numpy array.

I generate the Numpy array at the start. One trick is to realise that letters can be converted into the underlying number that represents them using `ord` . For an ASCII string this is always between 0 and 127:

```
_base642bin_array = np.zeros((128,),dtype=np.uint8)
for i in range(len(_base64chars)):
    _base642bin_array[ord(_base64chars[i])] = i
```

I do the conversion into 1s and 0s in the `n_decompress` function, using a built-in numpy function.

```
def n_decompress2(compressed_vector):
    # encode is for Python 3: str -> bytes
    decompressed_b64 = "".join(_decompress_get(compressed_vector)).encode()
    # byte string into the underlying numeric data
    decompressed_b64 = np.fromstring(decompressed_b64,dtype=np.uint8)
    # conversion done by numpy indexing rather than dictionary lookup
    vectorized = _base642bin_array[decompressed_b64]
    # convert to a 2D array of 1s and 0s
    as_binary = np.unpackbits(vectorized[:,np.newaxis],axis=1)
    # remove the two digits you don't care about (always 0) from binary array
    as_binary = as_binary[:,2:]
    # reshape to 1D (and chop off two at the end)
    return as_binary.ravel()[:-2]
```

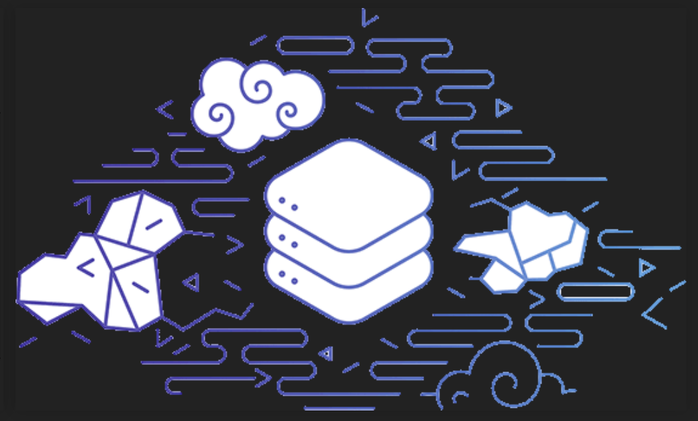
This gives me a 2.4x speed over your version (note that I haven't changed `_decompress_get` at all, so both timings include your `_decompress_get`) just from using Numpy (no Cython/Numba, and I suspect they won't help too much). I think the main advantage is that indexing into an array with numbers is fast compared to a dictionary lookup.

`_decompress_get` probably could be improved using Cython but it's a significantly harder problem...

LEONARDO

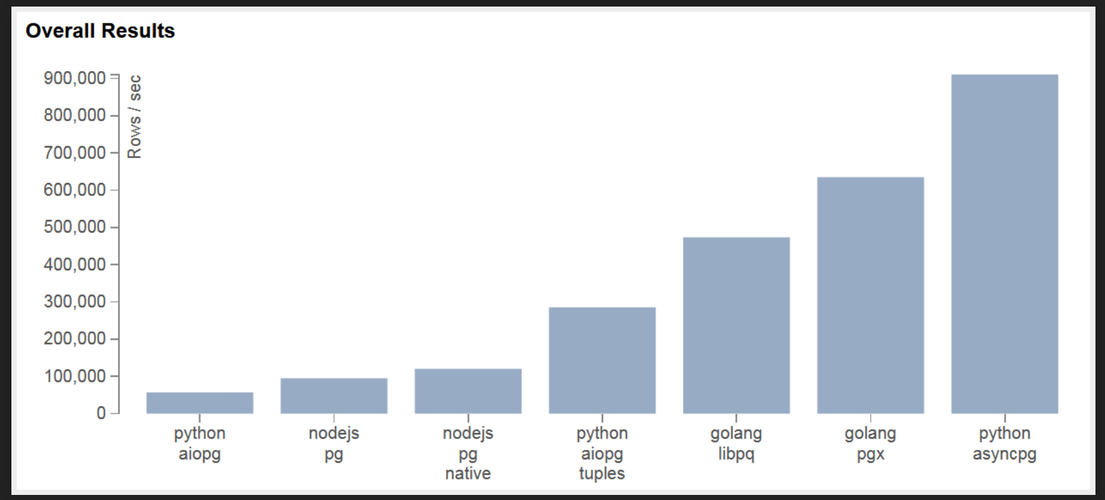
STORAGE

At the beginning we stored data in Neo4j DB,
good for graph generation but not for retrieve whole data for compare.
After start store binaries instead strings in DB we evaluated MongoDB and PostgreSQL.



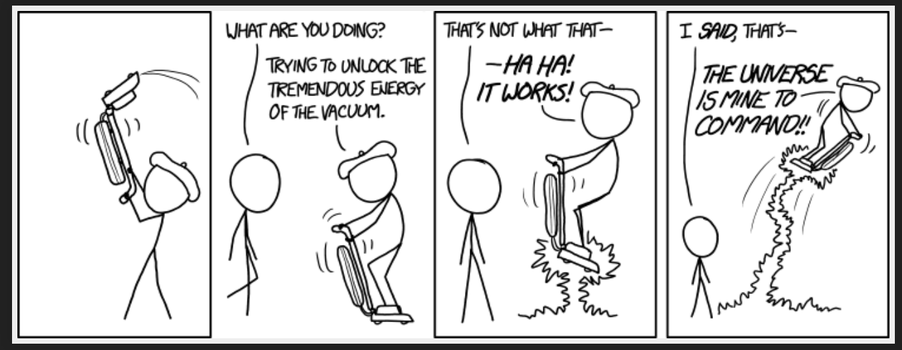
LEONARDO

AND THE WINNER IS...

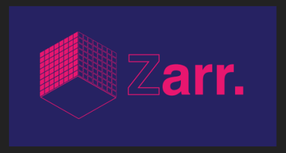


asyncpg is a new fully-featured open-source Python client library for PostgreSQL.
It is built specifically for asyncio and Python 3.5 async / await.
asyncpg is the fastest driver among common Python, NodeJS and Go implementations.

PostgreSQL with asyncpg has good results: ~ 1M rows/sec



LEONARDO



CAN WE DO MORE?

Lets't try saving binaries directly on disk: ZARR

Zarr Highlights:

- Create N-dimensional arrays with any NumPy dtype.
- Chunk arrays along any dimension.
- Compress and/or filter chunks using any NumCodecs codec.
- Store arrays in memory, on disk, inside a Zip file, on S3, …
- Read an array concurrently from multiple threads or processes.
- Write to an array concurrently from multiple threads or processes.
- Organize arrays into hierarchies via groups.

DASK

RAM

If number of samples grows to millions or billions? Say Hello to DASK!

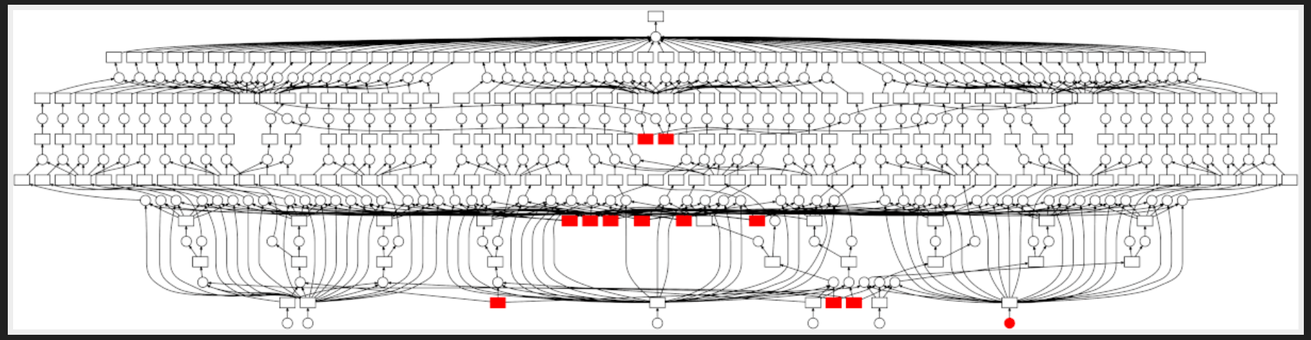
- Parallelized NumPy array and Pandas DataFrame object
- Task scheduling interface for custom workloads and integration with other projects
- Enables distributed computing in pure Python with access to the PyData stack
- Operates with low overhead, low latency, and minimal serialization
- Runs resiliently on clusters with 1000s of cores
- Trivial to set up and run on a laptop in a single process
- Designed with interactive computing in mind providing rapid feedback and diagnostics

DASK

PROS & CONS

CONS:

- scheduling overhead (worst for small data)
- additional complexity (scheduler, workers)



PROS:

- code runs on different machines
- memory managment
- data are splitted and processed in parallel

LEONARDO

Putting things all together pt.1

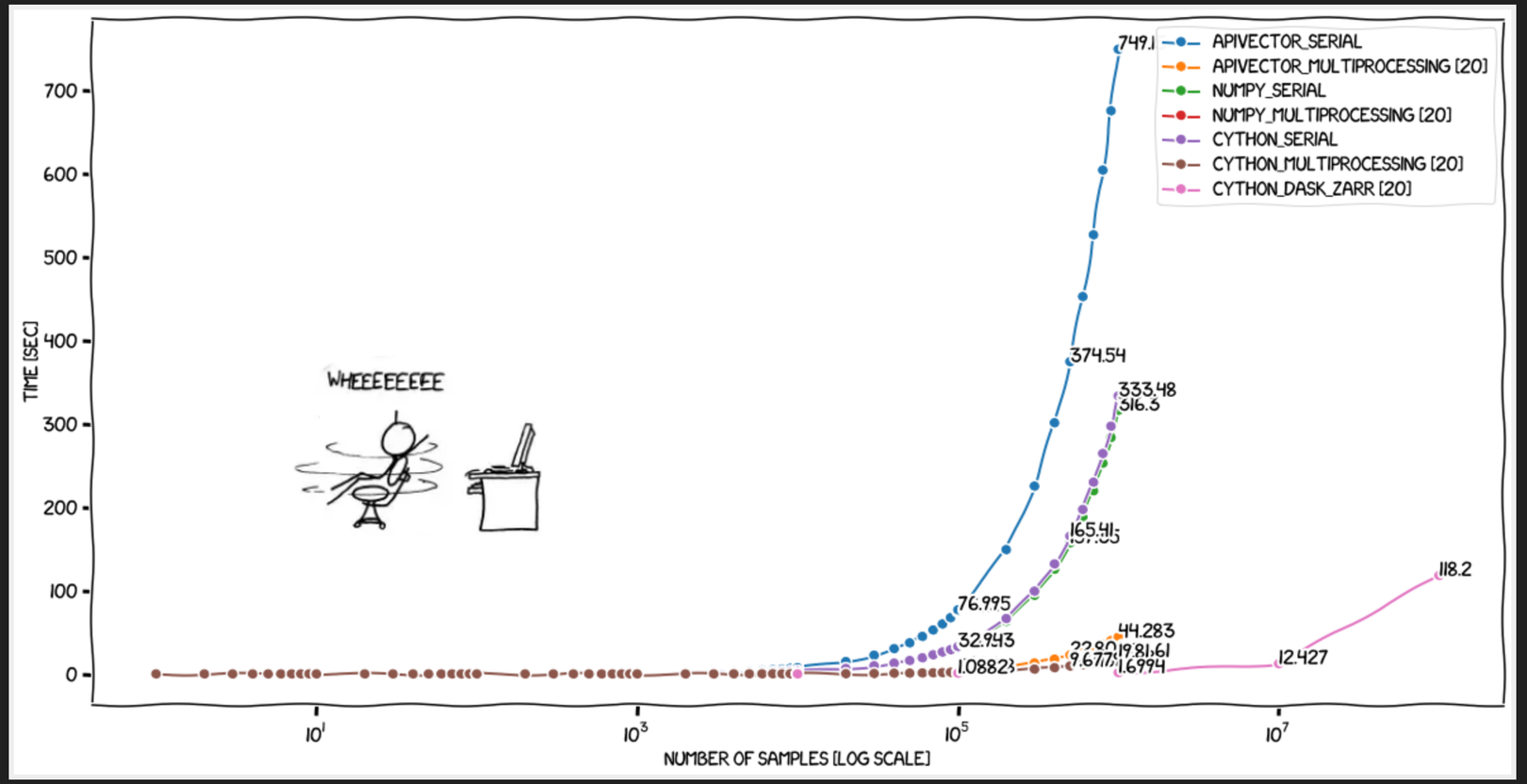
```c
// C_RUZICKA.C
#include <stdint.h>
double c_ruzicka (uint16_t* array_a, uint16_t* array_b) {
    double max_ab = min_ab =  0.0;
    int tmp = index = 0;
    for (index = 0; index < 1024; index++) {
        tmp = array_b[index] * (1023-index);
        if(array_a[index] > tmp){
            max_ab = max_ab + array_a[index];
            min_ab = min_ab + tmp;
        }else{
            min_ab = min_ab + array_a[index];
            max_ab = max_ab + tmp;
        }
    }
    return min_ab/max_ab;
}
```


```python
# RUZICKA.PY
import numpy as np
cimport numpy as np
cimport cython
# IMPORTING EXTERNAL C CODE FROM C_RUZICKA
ctypedef np.uint16_t data_type_16
cdef extern double c_ruzicka (data_type_16* array_a, data_type_16* array_b)
@cython.boundscheck(False)
@cython.wraparound(False)
def ruzicka(np.ndarray[data_type_16, ndim=1, mode="c"] vector_a, np.ndarray[data_type_16, ndim=1, mode="c"] vector_b):
    res = c_ruzicka(&vector_a[0], &vector_b[0])
    return res
```

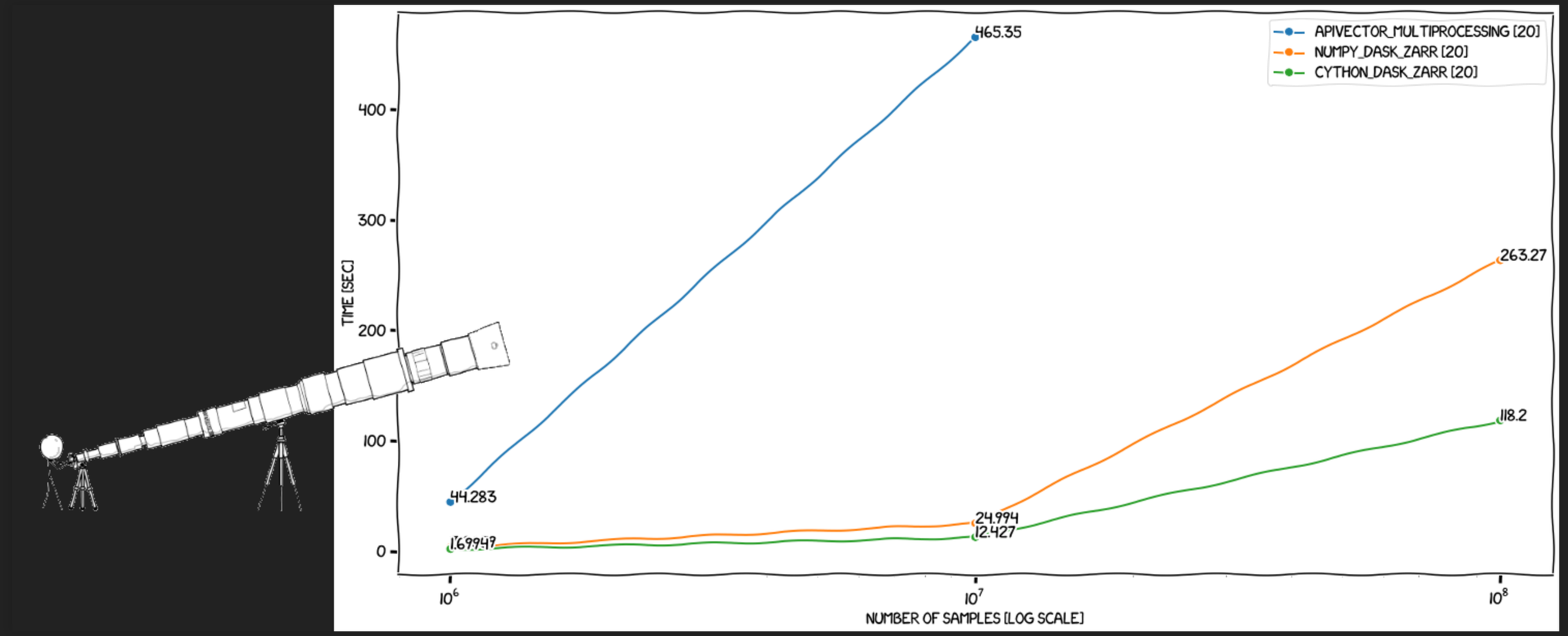
Putting things all together pt.2

```python
# MAIN.PY
import zarr
import numpy as np
import dask.array as da

def ruzicka_func(vector_old, vector_new):
    from ruzi_cython import ruzicka
    return ruzicka.ruzicka(vector_new, vector_old)

if __name__ == "__main__":
    vector_new = np.array(np.random.choice(a=[0, 1], size=(1, 1024)), dtype=np.uint16)
    vector_weighted = vector_new[0] * np.arange(1023, -1, -1, dtype=np.uint16)
    df = da.from_zarr('/tmp/zarr100M_uint16_test.zarr', chunks=(100000, 1024))
    res = da.apply_along_axis(lambda x: ruzicka_func(x, vector_weighted), 1, df).compute()
```

PERFORMANCES PT.5



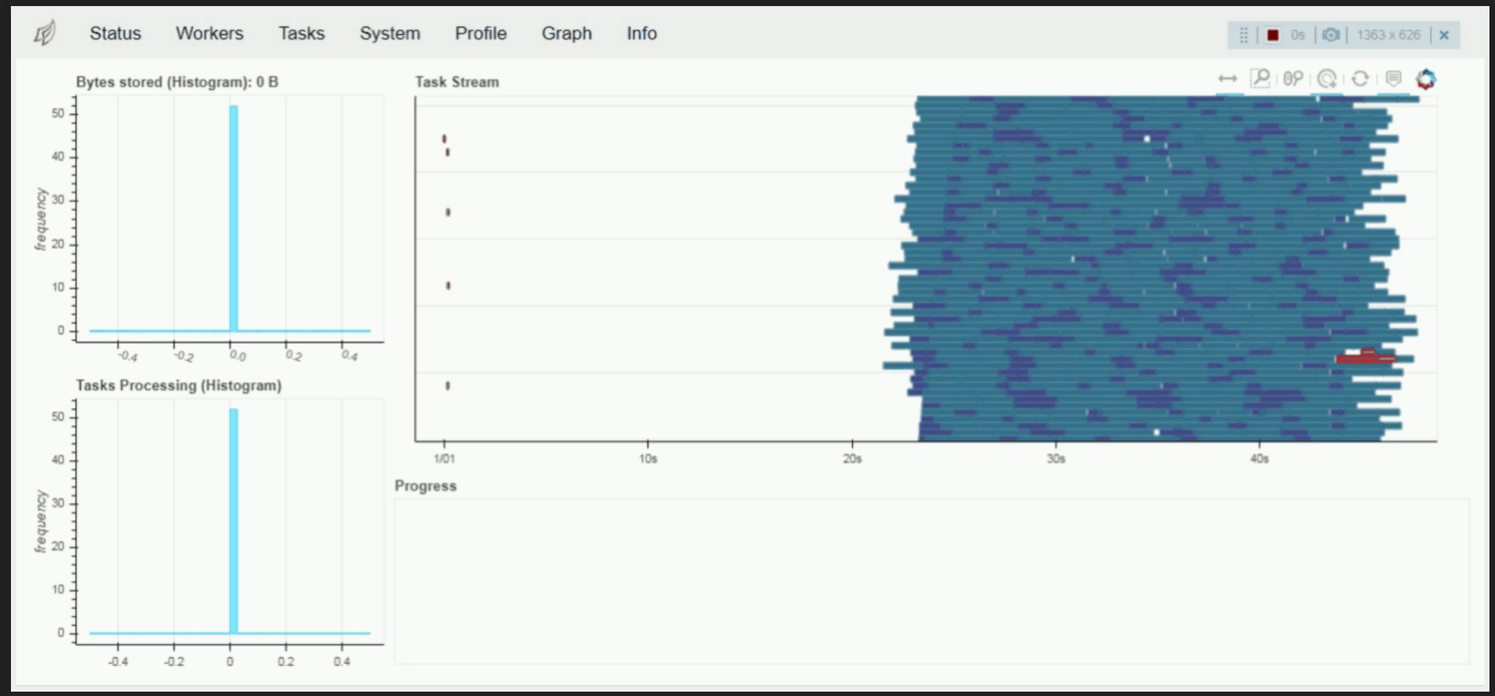
PERFORMANCES PT.5 - ZOOM



DASK IN ACTION

3 heterogeneous server, 20cpu + 16cpu + 16cpu

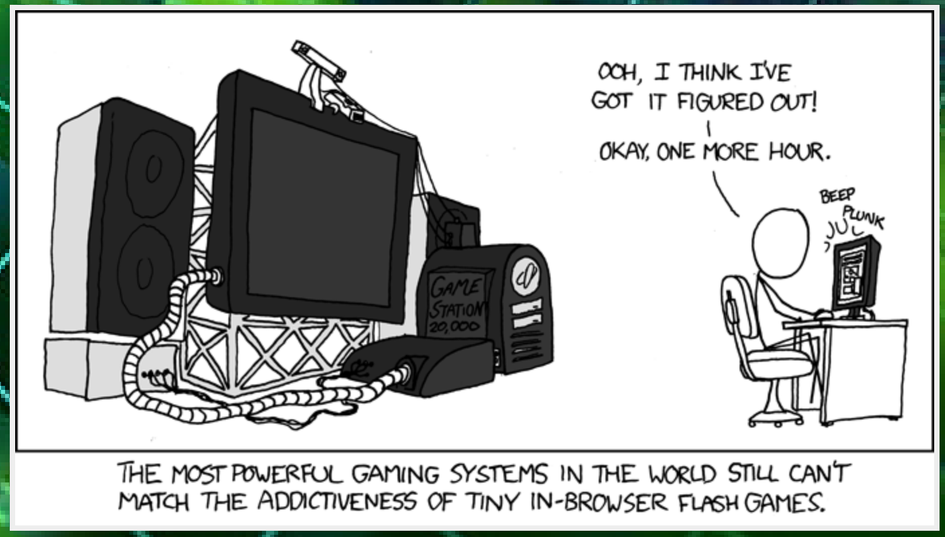
```
' MAIN NODE '
[dask@scheduler]# dask-scheduler --interface eth1 --port 8786 &
[dask@scheduler]# dask-worker scheduler:8786 --interface eth1 --nprocs 20 --nthreads 1 &

' WORKER NODE 1 '
[dask@worker1]# dask-worker scheduler:8786 --interface eth1 --nprocs 16 --nthreads 1 &

' WORKER NODE 2 '
[dask@worker2]# dask-worker scheduler:8786 --interface eth1 --nprocs 16 --nthreads 1 &

' MAIN NODE '
[dask@scheduler]# python run_ruzicka.py
```



Up to 1M records 1 or 3 machine have similar time.
For 100M we obtain a 2.5x speedup.

CUDA GPUs

CUDA™ is a revolutionary parallel computing architecture from NVIDIA.
CUDA makes it possible to use the many computing cores in a graphics processor to perform general-purpose mathematical calculations, achieving dramatic speedups in computing performance.

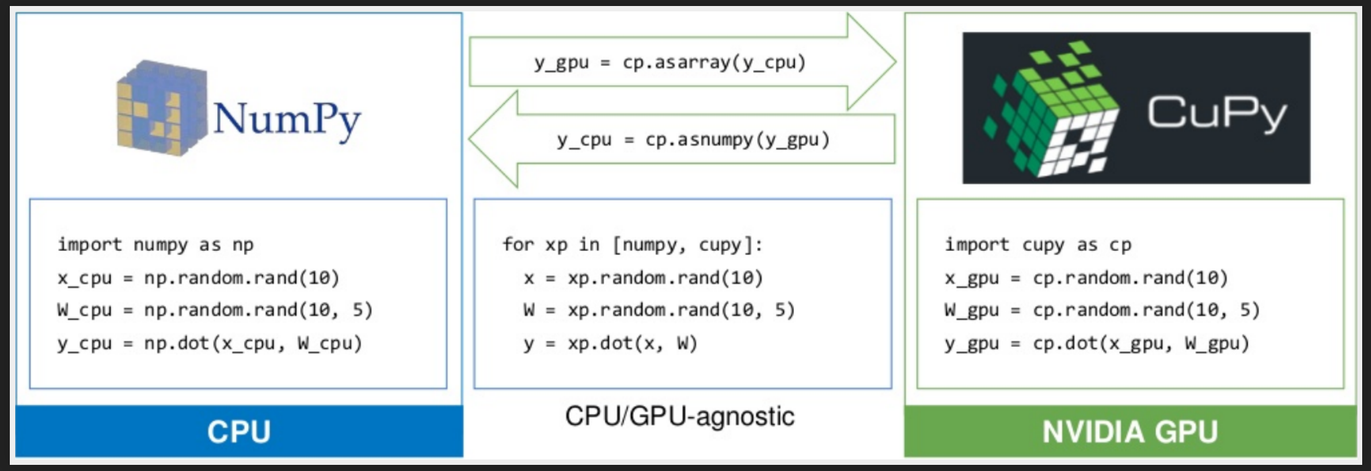


☑EASY =)

CuPy: NumPy-compatible lib. for GPU

Numpy is extensively used in python but GPU is not supported

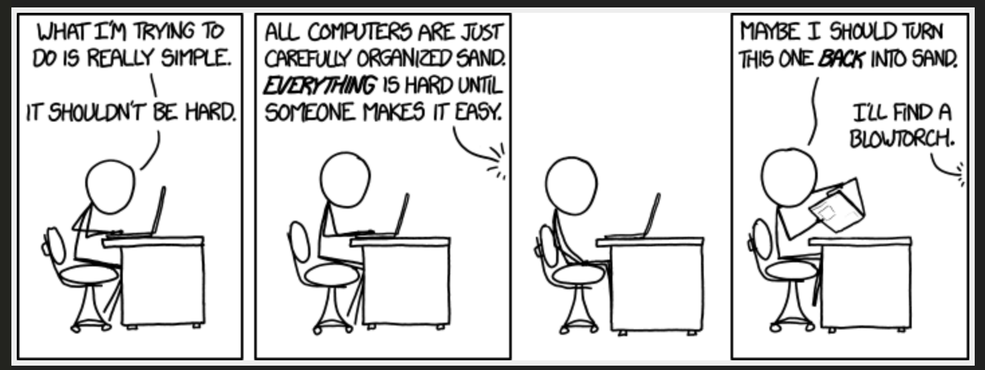
GPU is getting faster and more important for scientific computing



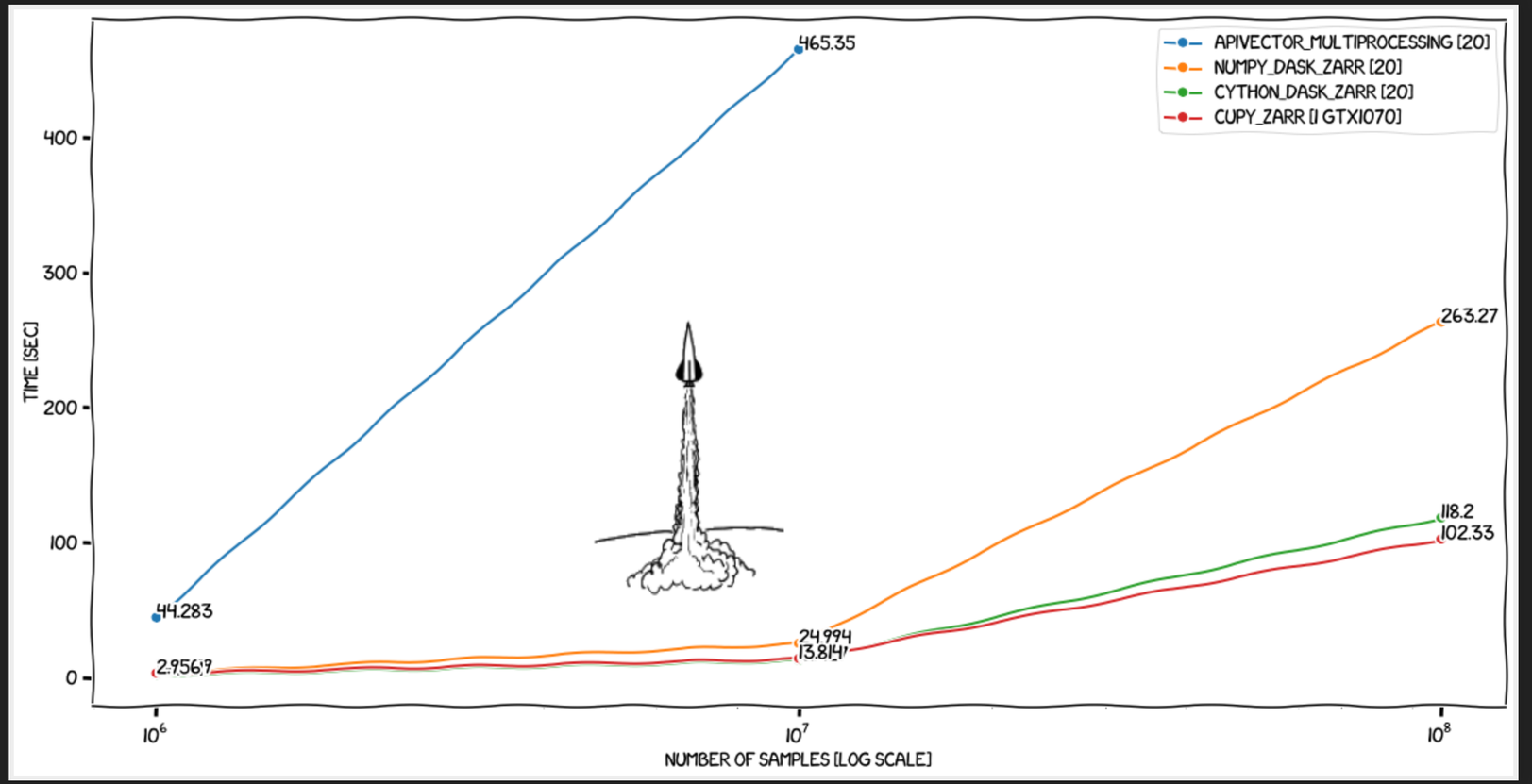
Shohei Hido slides

IS CUPY REALLY EASY?

```python
import numpy as np
import cupy as cp

def ruzicka_numpy(vector_old, vector_new):
    vector_old =  vector_old * np.arange(1023, -1, -1, dtype=np.uint16)
    min_up = np.minimum(vector_old, vector_new)
    max_down = np.maximum(vector_old, vector_new)
    numerator = np.sum(min_up)
    denominator = np.sum(max_down)
    return np.divide(numerator, denominator)

def ruzicka_cupy(vector_old, vector_new):
    vector_old = vector_old * cp.arange(1023, -1, -1, dtype=cp.uint16)
    min_up = cp.minimum(cp.array(vector_old), vector_new)
    max_down = cp.maximum(cp.array(vector_old), vector_new)
    numerator = cp.sum(min_up, axis=1)
    denominator = cp.sum(max_down, axis=1)
    return cp.asnumpy(cp.divide(numerator, denominator))
```



PERFORMANCE PT.6



LEONARDO

Medium

NEW OLD PROBLEMS!

Cupy is limited to GPU ram.

Cupy uses only one GPU.

LEONARDO

Medium

NEW OLD PROBLEMS!

Cupy is limited to GPU ram.

Cupy uses only one GPU.

RE-INTRODUCING DASK[-CUDA]

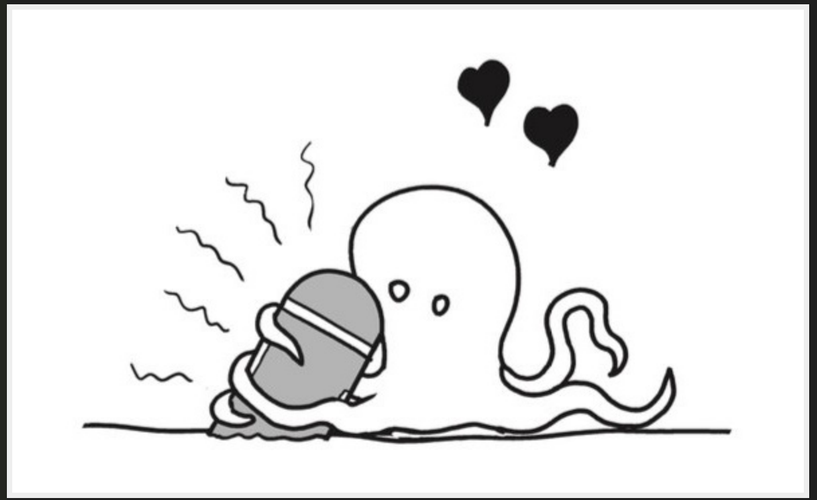
Utilities for DASK and CUDA interactions.

RAPIDS

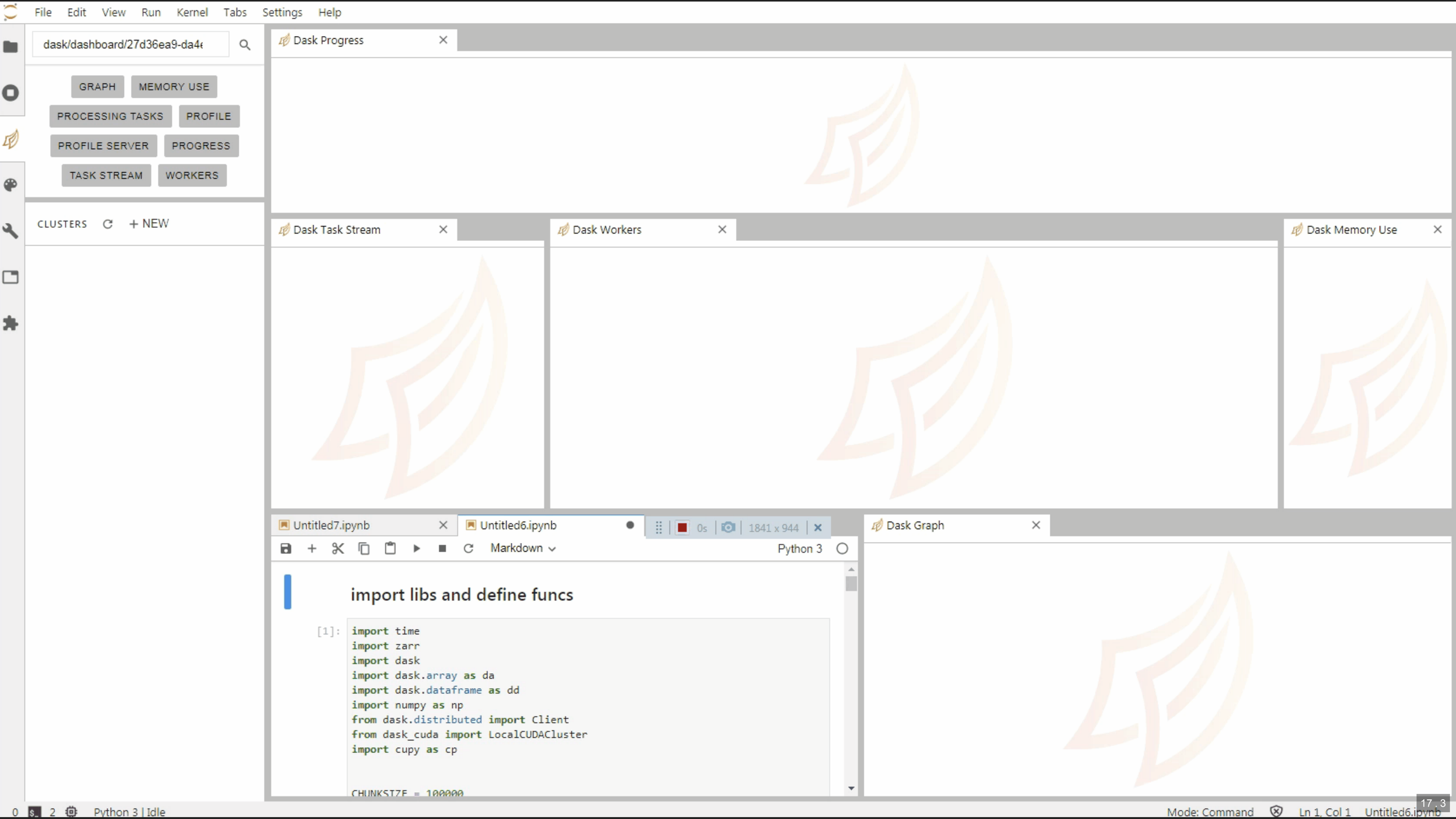
LEONARDO

OUR CODE

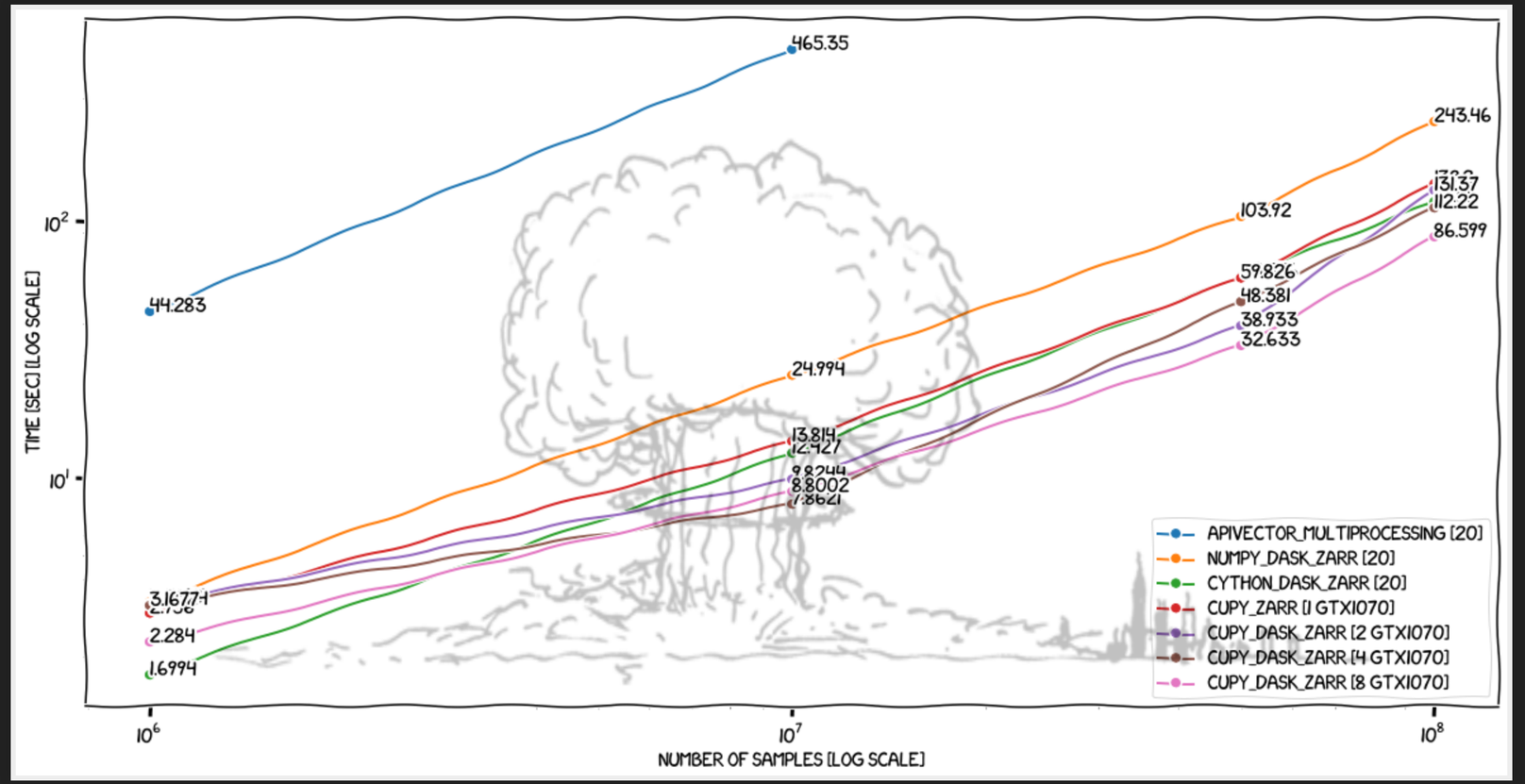
```python
import zarr
import dask
import dask.array as da
import dask.dataframe as dd
from dask.distributed import Client
from dask_cuda import LocalCUDACluster
import cupy as cp

CHUNKSIZE = 100000

def ruzicka_cupy(vector_old, vector_new):
    vector_old_cp = cp.array(vector_old) * cp.arange(1023, -1, -1, dtype=cp.uint16)
    min_up = cp.minimum(vector_old_cp, vector_new)
    max_down = cp.maximum(vector_old_cp, vector_new)
    numerator = cp.sum(min_up, axis=1)
    denominator = cp.sum(max_down, axis=1)
    return cp.asnumpy(cp.divide(numerator, denominator))

if __name__ == "__main__":
    cluster = LocalCUDACluster(n_workers=8, threads_per_worker=8)
    client = Client(cluster)

    # GENERATE ONE RANDOM SAMPLE TO IDENTIFY
    vector_new = cp.array(cp.random.choice([0, 1], 1024), dtype=cp.uint16) * cp.arange(
        1023, -1, -1, dtype=cp.uint16
    )

    # DATABASE TO COMPARE
    d_da = da.from_zarr('malware_database.zarr', chunks=(CHUNKSIZE,1024))
    d_dd = dd.from_dask_array(d_da)
    res = d_dd.map_partitions(
        lambda df: ruzicka_cupy(cp.array(df), vector_new),
            meta=("result", cp.float32),
    ).compute()
    client.close()
```



LEONARDO


File Edit View Run Kernel Tabs Settings Help
dask/dashboard/27d36ea9-da4
GRAPH
MEMORY USE
PROCESSING TASKS
PROFILE
PROFILE SERVER
PROGRESS
TASK STREAM
WORKERS
CLUSTERS
+ NEW
Dask Progress
Dask Task Stream
Dask Workers
Dask Memory Use
Untitled7.ipynb
Untitled6.ipynb
0s
1841 x 944
Markdown
Python 3
import libs and define funcs
[1]:
import time
import zarr
import dask
import dask.array as da
import dask.dataframe as dd
import numpy as np
from dask.distributed import Client
from dask_cuda import LocalCUDACluster
import cupy as cp
CHUNKSIZE = 100000
Dask Graph
0 2 Python 3 | Idle
Mode: Command
Ln 1, Col 1
Untitled6.ipynb
17 : 3

PERFORMANCE PT.1



LEONARDO

DISK BOTTLENECK!

With 8 GPUs the bottleneck is Disk

```
Total DISK READ :     486.20 M/s | Total DISK WRITE :      14.37 K/s
Actual DISK READ:     488.07 M/s | Actual DISK WRITE:     388.08 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN      IO>    COMMAND
 7475 be/4 dask       10.33 M/s    0.00 B/s  0.00 % 99.99 % python3.6 -c from multiprocessing.forks
 7421 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 99.41 % python3.6 -c from multiprocessing.forks
 7409 be/4 dask        9.66 M/s    0.00 B/s  0.00 % 99.19 % python3.6 -c from multiprocessing.forks
 7444 be/4 dask        8.76 M/s    0.00 B/s  0.00 % 98.90 % python3.6 -c from multiprocessing.forks
 7441 be/4 dask       10.22 M/s    0.00 B/s  0.00 % 98.27 % python3.6 -c from multiprocessing.forks
 7447 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.25 % python3.6 -c from multiprocessing.forks
 7432 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.25 % python3.6 -c from multiprocessing.forks
 7460 be/4 dask        9.88 M/s    0.00 B/s  0.00 % 98.19 % python3.6 -c from multiprocessing.forks
 7422 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.16 % python3.6 -c from multiprocessing.forks
 7437 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.15 % python3.6 -c from multiprocessing.forks
 7480 be/4 dask        9.31 M/s    0.00 B/s  0.00 % 98.07 % python3.6 -c from multiprocessing.forks
 7438 be/4 dask        9.77 M/s    0.00 B/s  0.00 % 97.87 % python3.6 -c from multiprocessing.forks
 7435 be/4 dask        9.62 M/s    0.00 B/s  0.00 % 97.87 % python3.6 -c from multiprocessing.forks
 7412 be/4 dask        9.88 M/s    0.00 B/s  0.00 % 97.31 % python3.6 -c from multiprocessing.forks
 7410 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 97.26 % python3.6 -c from multiprocessing.forks
```

LEONARDO

DISK BOTTLENECK!

With 8 GPUs the bottleneck is Disk

```
Total DISK READ :     486.20 M/s | Total DISK WRITE :      14.37 K/s
Actual DISK READ:     488.07 M/s | Actual DISK WRITE:     388.08 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 7475 be/4 dask       10.33 M/s    0.00 B/s  0.00 % 99.99 % python3.6 -c from multiprocessing.forks
 7421 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 99.41 % python3.6 -c from multiprocessing.forks
 7409 be/4 dask        9.66 M/s    0.00 B/s  0.00 % 99.19 % python3.6 -c from multiprocessing.forks
 7444 be/4 dask        8.76 M/s    0.00 B/s  0.00 % 98.90 % python3.6 -c from multiprocessing.forks
 7441 be/4 dask       10.22 M/s    0.00 B/s  0.00 % 98.27 % python3.6 -c from multiprocessing.forks
 7447 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.25 % python3.6 -c from multiprocessing.forks
 7432 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.25 % python3.6 -c from multiprocessing.forks
 7460 be/4 dask        9.88 M/s    0.00 B/s  0.00 % 98.19 % python3.6 -c from multiprocessing.forks
 7422 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.16 % python3.6 -c from multiprocessing.forks
 7437 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.15 % python3.6 -c from multiprocessing.forks
 7480 be/4 dask        9.31 M/s    0.00 B/s  0.00 % 98.07 % python3.6 -c from multiprocessing.forks
 7438 be/4 dask        9.77 M/s    0.00 B/s  0.00 % 97.87 % python3.6 -c from multiprocessing.forks
 7435 be/4 dask        9.62 M/s    0.00 B/s  0.00 % 97.87 % python3.6 -c from multiprocessing.forks
 7412 be/4 dask        9.88 M/s    0.00 B/s  0.00 % 97.31 % python3.6 -c from multiprocessing.forks
 7410 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 97.26 % python3.6 -c from multiprocessing.forks
```

Trying with RAMDISK

```
sagitta@brutalis:/mnt$ sudo dd if=/dev/zero of=./largefile bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 0.733735 s, 1.5 GB/s
sagitta@brutalis:/mnt$ sudo sh -c "sync && echo 3 > /proc/sys/vm/drop_caches"
sagitta@brutalis:/mnt$ dd if=./largefile of=/dev/null bs=4k
262144+0 records in
262144+0 records out
1073741824 bytes (1.1 GB) copied, 2.12231 s, 506 MB/s
sagitta@brutalis:/mnt$ cd ramdisk/
sagitta@brutalis:/mnt/ramdisk$ sudo dd if=/dev/zero of=./largefile bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 0.501162 s, 2.1 GB/s
sagitta@brutalis:/mnt/ramdisk$ sudo sh -c "sync && echo 3 > /proc/sys/vm/drop_caches"
sagitta@brutalis:/mnt/ramdisk$ dd if=./largefile of=/dev/null bs=4k
262144+0 records in
262144+0 records out
1073741824 bytes (1.1 GB) copied, 0.432493 s, 2.5 GB/s
```

DISK BOTTLENECK!

With 8 GPUs the bottleneck is Disk

```
Total DISK READ :     486.20 M/s | Total DISK WRITE :      14.37 K/s
Actual DISK READ:     488.07 M/s | Actual DISK WRITE:     388.08 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 7475 be/4 dask       10.33 M/s    0.00 B/s  0.00 % 99.99 % python3.6 -c from multiprocessing.forks
 7421 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 99.41 % python3.6 -c from multiprocessing.forks
 7409 be/4 dask        9.66 M/s    0.00 B/s  0.00 % 99.19 % python3.6 -c from multiprocessing.forks
 7444 be/4 dask        8.76 M/s    0.00 B/s  0.00 % 98.90 % python3.6 -c from multiprocessing.forks
 7441 be/4 dask       10.22 M/s    0.00 B/s  0.00 % 98.27 % python3.6 -c from multiprocessing.forks
 7447 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.25 % python3.6 -c from multiprocessing.forks
 7432 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.25 % python3.6 -c from multiprocessing.forks
 7460 be/4 dask        9.88 M/s    0.00 B/s  0.00 % 98.19 % python3.6 -c from multiprocessing.forks
 7422 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.16 % python3.6 -c from multiprocessing.forks
 7437 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 98.15 % python3.6 -c from multiprocessing.forks
 7480 be/4 dask        9.31 M/s    0.00 B/s  0.00 % 98.07 % python3.6 -c from multiprocessing.forks
 7438 be/4 dask        9.77 M/s    0.00 B/s  0.00 % 97.87 % python3.6 -c from multiprocessing.forks
 7435 be/4 dask        9.62 M/s    0.00 B/s  0.00 % 97.87 % python3.6 -c from multiprocessing.forks
 7412 be/4 dask        9.88 M/s    0.00 B/s  0.00 % 97.31 % python3.6 -c from multiprocessing.forks
 7410 be/4 dask       10.11 M/s    0.00 B/s  0.00 % 97.26 % python3.6 -c from multiprocessing.forks
```

Trying with RAMDISK

```
sagitta@brutalis:/mnt$ sudo dd if=/dev/zero of=./largefile bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 0.733735 s, 1.5 GB/s
sagitta@brutalis:/mnt$ sudo sh -c "sync && echo 3 > /proc/sys/vm/drop_caches"
sagitta@brutalis:/mnt$ dd if=./largefile of=/dev/null bs=4k
262144+0 records in
262144+0 records out
1073741824 bytes (1.1 GB) copied, 2.12231 s, 506 MB/s
sagitta@brutalis:/mnt$ cd ramdisk/
sagitta@brutalis:/mnt/ramdisk$ sudo dd if=/dev/zero of=./largefile bs=1M count=1024
1024+0 records in
1024+0 records out
1073741824 bytes (1.1 GB) copied, 0.501162 s, 2.1 GB/s
sagitta@brutalis:/mnt/ramdisk$ sudo sh -c "sync && echo 3 > /proc/sys/vm/drop_caches"
sagitta@brutalis:/mnt/ramdisk$ dd if=./largefile of=/dev/null bs=4k
262144+0 records in
262144+0 records out
1073741824 bytes (1.1 GB) copied, 0.432493 s, 2.5 GB/s
```

we see the improvement but we are limited to ram size.

LEONARDO

WANNA INVEST "SOME" MONEY?

SPECIFICATIONS


	Tesla V100 PCIe	Tesla V100 SXM2
GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS
GPU Memory	32GB /16GB HBM2	
Memory Bandwidth	900GB/sec	
ECC	Yes	
Interconnect Bandwidth	32GB/sec	300GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power Comsumption	250 W	300 W
Thermal Solution	Passive	
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC	

LEONARDO

WANNA INVEST "SOME" MONEY?

SPECIFICATIONS


	Tesla V100 PCIe	Tesla V100 SXM2
GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS
GPU Memory	32GB /16GB HBM2	
Memory Bandwidth	900GB/sec	
ECC	Yes	
Interconnect Bandwidth	32GB/sec	300GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power Comsumption	250 W	300 W
Thermal Solution	Passive	
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC	

Shopping Cart

Price

NVIDIA Tesla V100 Volta GPU Accelerator 32GB Graphics Card $8,978.00

Only 4 left in stock - order soon.

Shipped from: Professional Graphics Solutions

Gift options not available. Learn more

Qty: 1 | Delete | Save for later

Compare with similar items

Subtotal (1 item): $8,978.00

Subtotal (1 item): $8,978.00

Proceed to checkout

LEONARDO

WANNA INVEST "SOME" MONEY?

SPECIFICATIONS


	Tesla V100 PCIe	Tesla V100 SXM2
GPU Architecture	NVIDIA Volta	
NVIDIA Tensor Cores	640	
NVIDIA CUDA® Cores	5,120	
Double-Precision Performance	7 TFLOPS	7.8 TFLOPS
Single-Precision Performance	14 TFLOPS	15.7 TFLOPS
Tensor Performance	112 TFLOPS	125 TFLOPS
GPU Memory	32GB /16GB HBM2	
Memory Bandwidth	900GB/sec	
ECC	Yes	
Interconnect Bandwidth	32GB/sec	300GB/sec
System Interface	PCIe Gen3	NVIDIA NVLink
Form Factor	PCIe Full Height/Length	SXM2
Max Power Comsumption	250 W	300 W
Thermal Solution	Passive	
Compute APIs	CUDA, DirectCompute, OpenCL™, OpenACC	

Shopping Cart

Price

NVIDIA Tesla V100 Volta GPU Accelerator 32GB Graphics Card — $8,978.00

Only 4 left in stock - order soon.

Shipped from: Professional Graphics Solutions

Gift options not available. Learn more

Qty: 1 | Delete | Save for later

Compare with similar items

Subtotal (1 item): $8,978.00

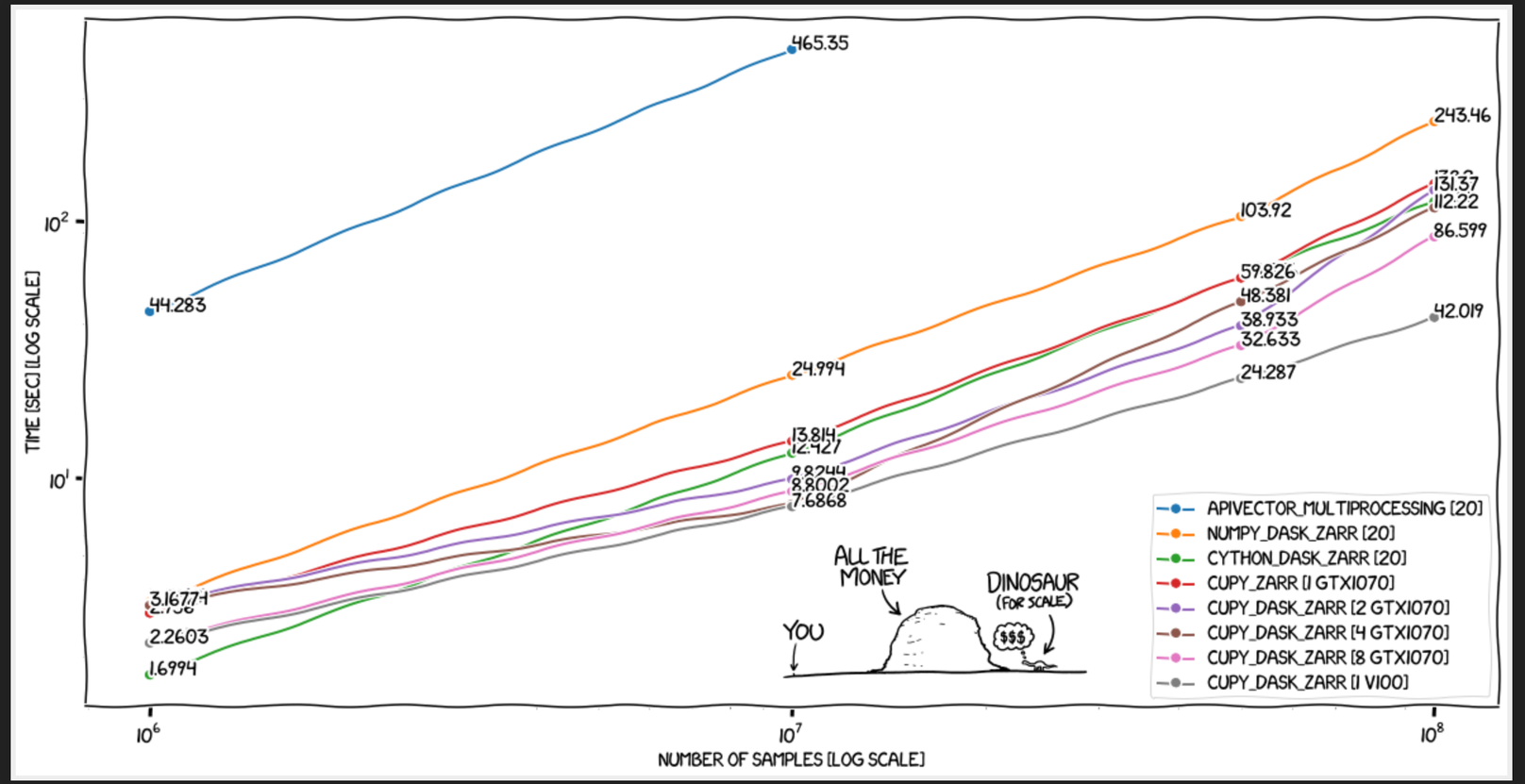
Subtotal (1 item): $8,978.00

Proceed to checkout

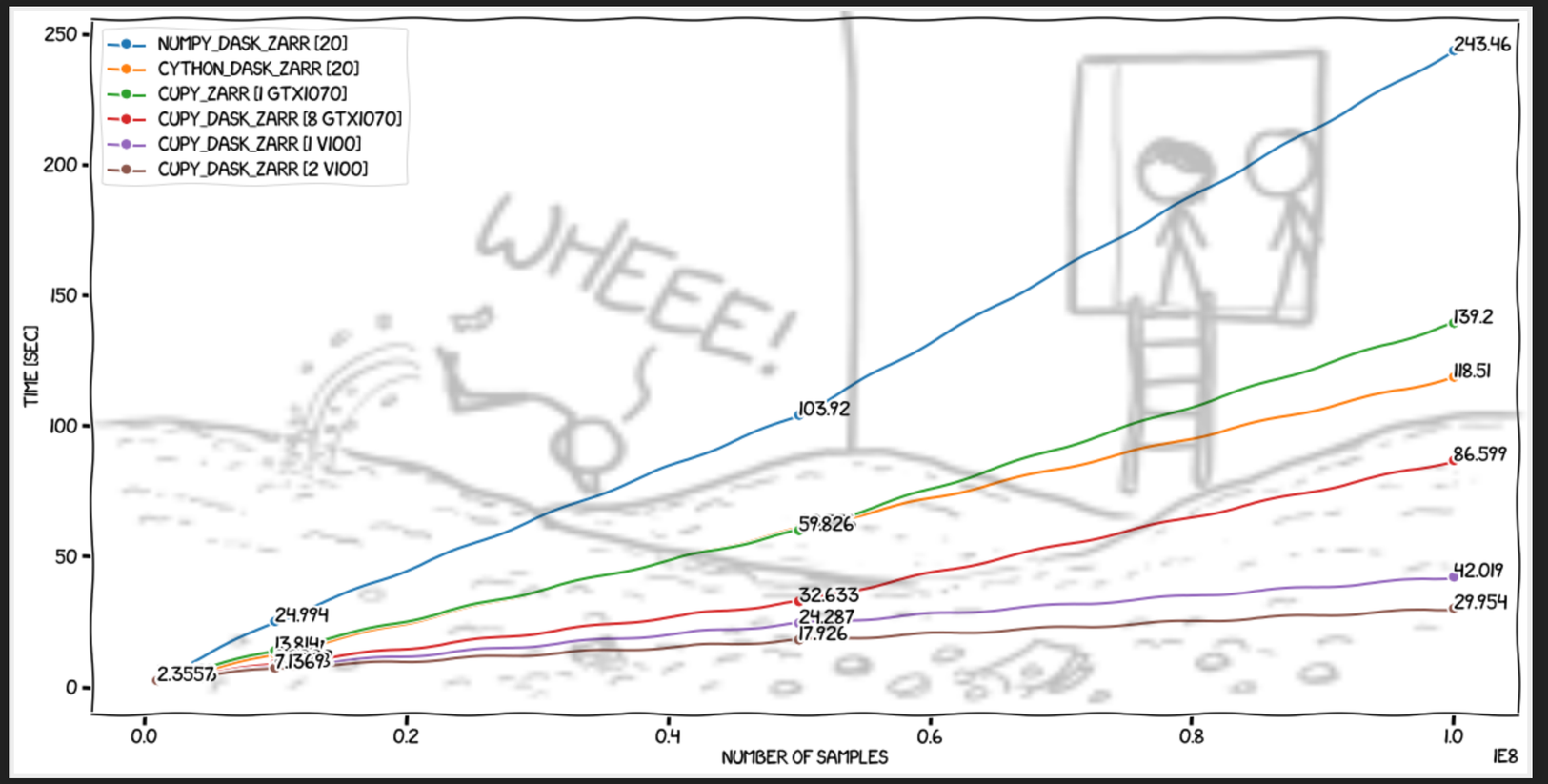
```
Every 1.0s: nvidia-smi

Fri Oct 18 17:24:08 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.87.01    Driver Version: 418.87.01    CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-PCIE...  Off  | 00000000:86:00.0 Off |                    0 |
| N/A   80C    P0    62W / 250W |  10168MiB / 32480MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     30133      C   /opt/miniconda3/envs/test/bin/python         921MiB |
|    0     30204      C   /opt/miniconda3/envs/test/bin/python        2285MiB |
|    0     30344      C   /opt/miniconda3/envs/test/bin/python        6791MiB |
+-----------------------------------------------------------------------------+
```

LEONARDO

PERFORMANCE PT.8



PERFORMANCE PT.9

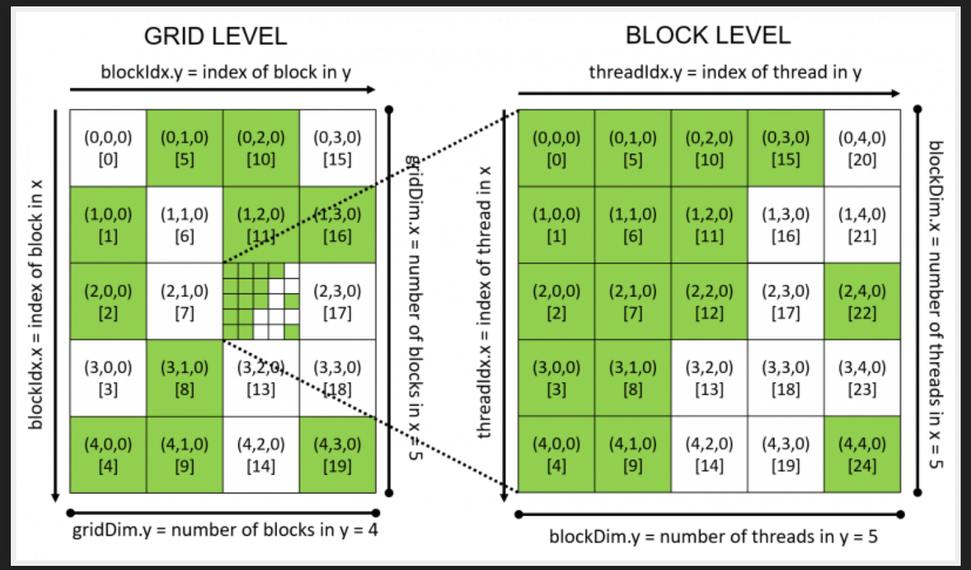


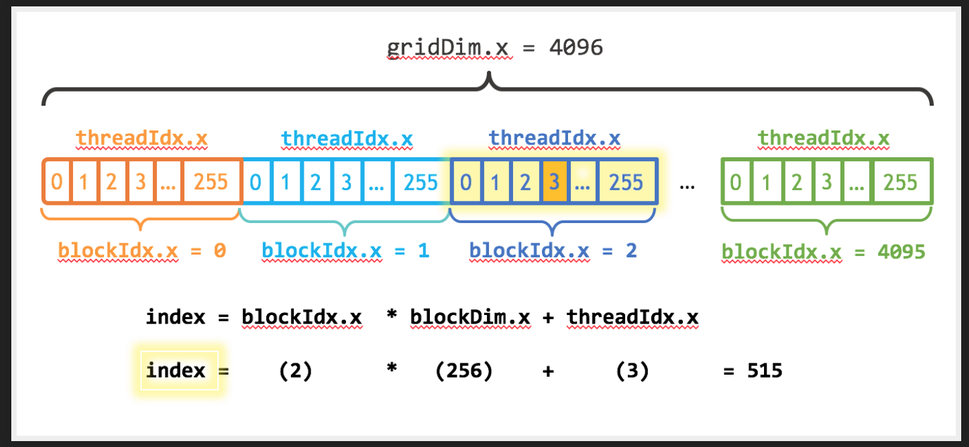
☑Hard

CUDA KERNEL

CuPy provides easy ways to define three types of CUDA kernels: elementwise kernels, reduction kernels and raw kernels.

Raw Kernels permits you to work directly with Grids, Blocks and Threads.

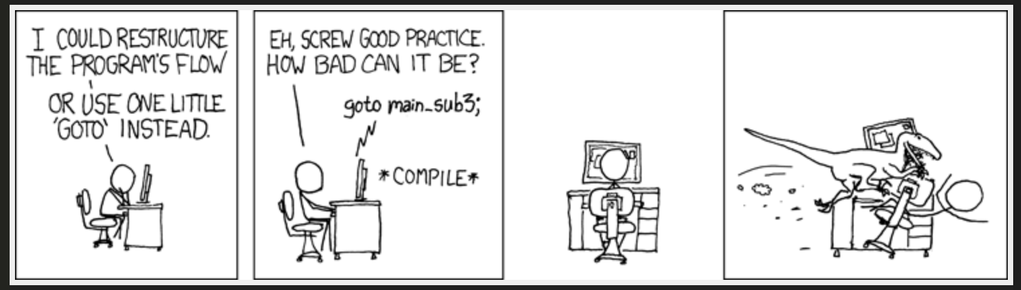




LEONARDO

OUR CODE

```
import cupy as cp

ruzicka_shared_kernel = cp.RawKernel(r"""
    extern "C" __global__
    #define TILE_WIDTH 32
    void shared_ruzicka(const unsigned short* x1, const unsigned short* x2,
                        float* y, int width, int height) {
        __shared__ unsigned short x1s[TILE_WIDTH][TILE_WIDTH];
        __shared__ unsigned short x2s[TILE_WIDTH];
        int by = blockIdx.y; int tx = threadIdx.x; int ty = threadIdx.y;
        int row = by * TILE_WIDTH + ty;
        int idx = 0;
        unsigned short tmp = 0;
        float maxab = minab = 0.0;

        for (int m = 0; m < 1024/TILE_WIDTH; ++m){
            idx = row * width + m * TILE_WIDTH + tx;
            if (idx < width * height){
                x1s[ty][tx] = x1[idx];
                x2s[tx] = x2[m*TILE_WIDTH + tx];
                __syncthreads();
                for (int k=0; k < TILE_WIDTH; ++k){
                    tmp = x1s[ty][k] * (1023 - (m*TILE_WIDTH+k));
                    if(x2s[k] > tmp){ maxab += x2s[k]; minab += tmp; }
                    else{ minab += x2s[k]; maxab += tmp; }
                }
                __syncthreads();
            }}
        y[row] = minab / maxab;
    }""", "shared_ruzicka",
)

def ruzicka_shared_retval(a, b):
    y = cp.zeros(CHUNKSIZE, dtype=cp.float32).reshape(1, CHUNKSIZE)
    ruzicka_shared_kernel((32, (len(a) + 31) // 32), (32, 32), (a, b, y, 1024, len(a)))
    return y

dda = da.from_zarr('malware_database.zarr', chunks=(100000,1024))
res = dda.map_blocks(lambda df: ruzicka_shared_retval(cp.array(df), vector_new),
                     dtype=cp.float32).compute()
```



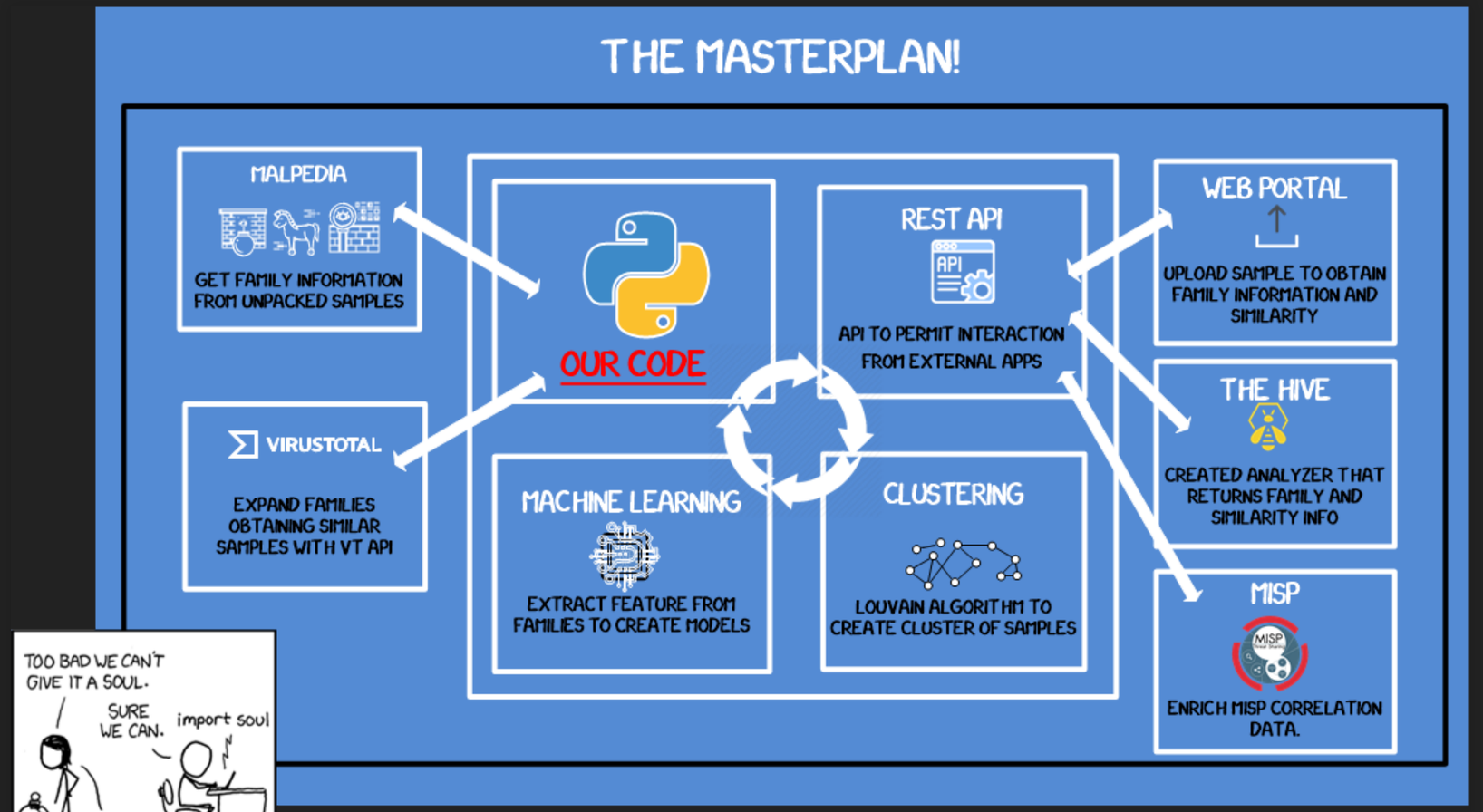
LEONARDO

FINAL WORDS:

From the beginning we obtained a ~1800x speedup.

We learned how to scale up and out.

ECOSYSTEM



LEONARDO

REFERENCES

FIND US AT:

- all benchmarks
- presentation
- LDO-CERT

THANKS TO:

- pictures: XKCD
- cover: Rita

LINKS:

- apiscout & apivector
- malpedia
- MISP
- asyncpg
- jupyter
- numpy
- Zarr
- dask-cuda
- Cython
- pybind11
- numexpr
- Pythran
- numba
- DASK
- dask-cuda
- dask-labextension
- CuPy

LEONARDO

QUESTIONS

FOUND IN GOOGLE AUTOCOMPLETE

WHY IS SEA SALT BETTER
WHY ARE THERE TREES IN THE MIDDLE OF FIELDS
WHY IS THERE NOT A POKEMON MMO
WHY IS THERE LAUGHING IN TV SHOWS
WHY ARE THERE DOORS ON THE FREEWAY
WHY ARE THERE SO MANY SVCHOST.EXE RUNNING
WHY AREN'T THERE ANY COUNTRIES IN ANTARCTICA
WHY ARE THERE SCARY SOUNDS IN MINECRAFT
WHY IS THERE KICKING IN MY STOMACH
WHY ARE THERE TWO SLASHES AFTER HTTP
WHY ARE THERE CELEBRITIES
WHY DO SNAKES EXIST
WHY DO OYSTERS HAVE PEARLS
WHY ARE DUCKS CALLED DUCKS
WHY DO THEY CALL IT THE CLAP
WHY ARE KYLE AND CARTMAN FRIENDS
WHY IS THERE AN ARROW ON AANG'S HEAD
WHY ARE TEXT MESSAGES BLUE
WHY ARE THERE MUSTACHES ON CLOTHES
WHY ARE THERE MUSTACHES ON CARS
WHY ARE THERE MUSTACHES EVERYWHERE
WHY ARE THERE SO MANY BIRDS IN OHIO
WHY IS THERE SO MUCH RAIN IN OHIO
WHY IS OHIO WEATHER SO WEIRD
WHY ARE THERE MALE AND FEMALE BIKES
WHY ARE THERE BRIDESMAIDS
WHY DO DYING PEOPLE REACH UP
WHY AREN'T THERE VARIOUS ARTISTS
WHY ARE OLD KLINGONS DIFFERENT
WHY ARE THERE SQUIRRELS
WHY IS PROGRAMMING SO HARD
WHY ARE THERE TINY SPIDERS IN MY HOUSE
WHY DO SPIDERS COME INSIDE
WHY ARE THERE HUGE SPIDERS IN MY HOUSE
WHY ARE THERE LOTS OF SPIDERS IN MY HOUSE
WHY ARE THERE SPIDERS IN MY ROOM
WHY ARE THERE SO MANY SPIDERS IN MY ROOM
WHY DO SPIDER BITES ITCH
WHY IS DYING SO SCARY
WHY IS THERE NO GPS IN LAPTOPS
WHY DO KNEES CLICK
WHY AREN'T THERE E GRADES
WHY IS SEX SO IMPORTANT
WHY AREN'T ECONOMISTS RICH
WHY DO AMERICANS CALL IT SOCCER
WHY ARE MY EARS RINGING
WHY ARE THERE SO MANY AVENGERS
WHY ARE THE AVENGERS FIGHTING THE X MEN
WHY IS WOLVERINE NOT IN THE AVENGERS
WHY ARE THERE ANTS IN MY LAPTOP
WHY IS EARTH TILTED
WHY IS SPACE BLACK
WHY IS OUTER SPACE SO COLD
WHY ARE THERE PYRAMIDS ON THE MOON
WHY IS NASA SHUTTING DOWN
WHY ARE THERE GHOSTS
WHY IS THERE AN OWL IN MY BACKYARD
WHY IS THERE AN OWL OUTSIDE MY WINDOW
WHY IS THERE AN OWL ON THE DOLLAR BILL
WHY DO OWLS ATTACK PEOPLE
WHY ARE AK 47S SO EXPENSIVE
WHY ARE THERE HELICOPTERS CIRCLING MY HOUSE
WHY ARE THERE GODS
WHY ARE THERE TWO SPOCKS
WHY IS MT VESUVIUS THERE
WHY DO THEY SAY T MINUS
WHY ARE THERE OBELISKS
WHY ARE WRESTLERS ALWAYS WET
WHY ARE OCEANS BECOMING MORE ACIDIC
WHY ARE THERE SHARKS IN LAKES
WHY IS THERE PHLEGM
WHY ARE THERE SO MANY CROWS IN ROCHESTER, MN
WHY IS PSYCHIC WEAK TO BUG
WHY DO CHILDREN GET CANCER
WHY IS POSEIDON ANGRY WITH ODYSSEUS
WHY IS THERE ICE IN SPACE
WHY ARE DOGS AFRAID OF FIRE
WHY IS THERE NO KING IN ENG
WHY ARE MY BOOBS ITCHY
WHY ARE CIGARETTES LEGAL
WHY ARE THERE DUCKS IN MY POOL
WHY IS JESUS WHITE
WHY IS THERE LIQUID IN MY EAR
WHY DO Q TIPS FEEL GOOD
WHY DO GOOD PEOPLE DIE
WHY AREN'T THERE GUNS IN HARRY POTTER
WHY DO I FEEL DIZZY
WHY ARE THERE WEEKS
WHY DO IGUANAS DIE
WHY AREN'T THERE DINOSAUR GHOSTS
WHY IS THERE LAVA
WHY ARE THERE FEMALE M
WHY IS THERE HELL IF GO
WHY IS LIFE SO
WHY AREN'T MY ARMS GROWING
OK ON ALL ZIPPERS