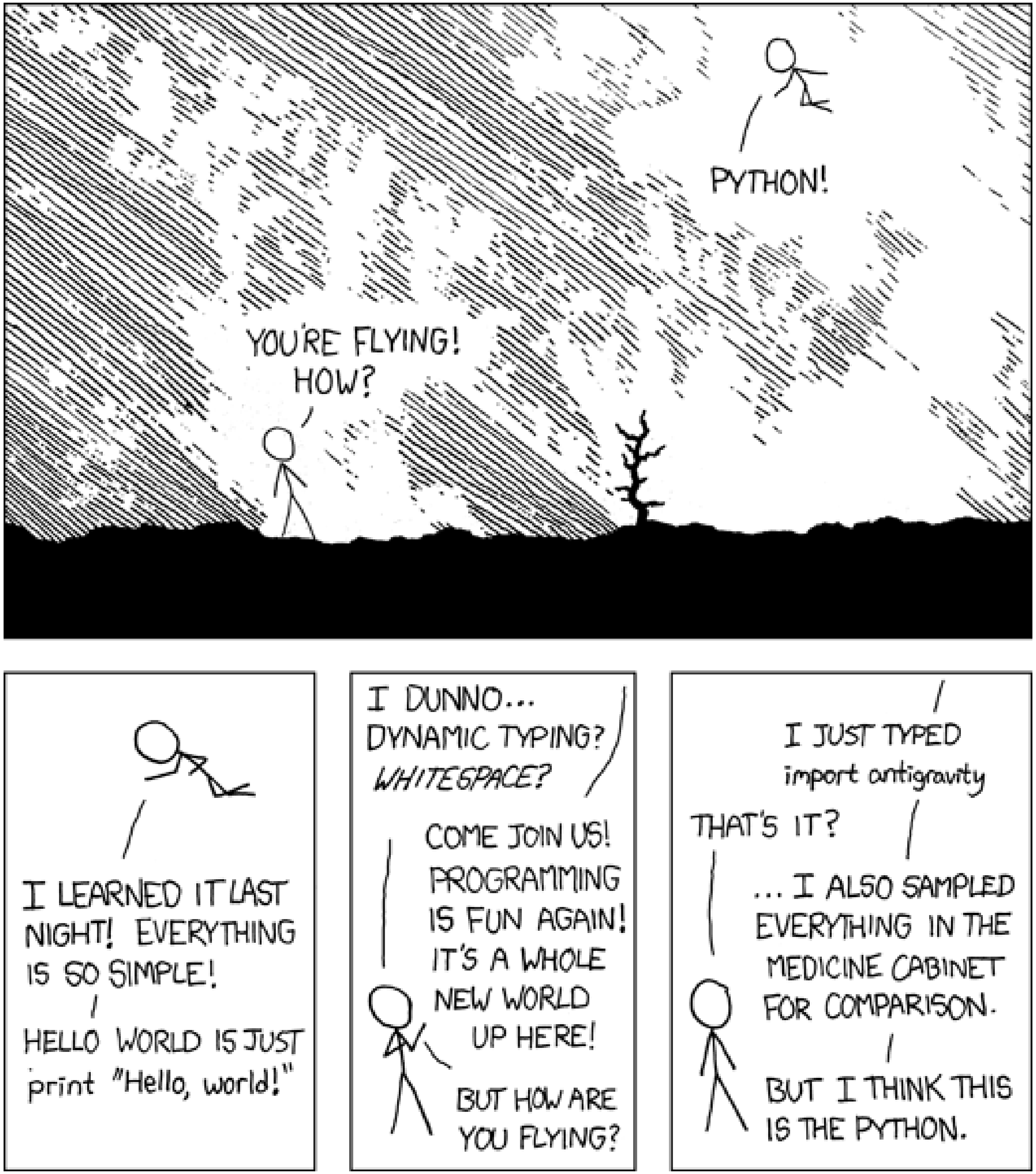
Centro de Modelamiento Matemático
Universidad de Chile

Python para Data Science

Paralelismo y Distribución

Juan Carlos Maureira B.
<*jcm@dim.uchile.cl*>

Taller CMM-MOP – 08/09/2016


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

xkcd #353

Facil de usar:

- Lenguaje no tipificado
- Interpretado
- Muchas librerias (modulos)
- Capacidades graficas
- Estadisticas
- Manejo numerico
- Facilmente portable

- Programar se hace “entretenido”

Conceptos Básicos

(una pasada a lo pintor)

Python 2

- Lenguaje interpretado (no compilado)
  - CPython VM
- Debilmente tipificado

```
[jcm@leftraru3 ~]$ module load python/2.7.10
[jcm@leftraru3 ~]$ python
Python 2.7.10 (default, Nov 10 2015, 18:09:20)
[GCC Intel(R) C++ gcc 4.8 mode] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> A = “Hola mundo”
>>> print A
Hola mundo
>>> A = 123
>>> print A
123
>>>
```

✓

- Facilmente extensible
  - Modulos son basicamente un directorio con un archivo llamado __init__.py
- Corre en windows, linux, mac, android, etc… (portable)
- Pip es tu amigo

Python 2 : snippets de lo clásico

```
# Bucles (loops)

for i in [1, 2, 3]:
    for j in [1, 2, 3]:
        print i, j
print "End"
x = 0
while x < 4:
    print x
    x += 1
```

```
# Funciones

def f(x):
    return x**2
def applyf(f, x=6):
    return f(x)
print applyf(f, 2)
print applyf(f)
```

```
# Modulos

import re
import matplotlib.pyplot as plt
import numpy as np
from __future__ import division
```

```
# asignaciones

x, y = [1, 2]
print x, y

x, y = 1, 2
print x, y
```

```
# Diccionarios (mapas asociativos)

mydict = {}
mydict2 = dict()
codes = {"AR": 54, "BR": 55, "CL": 56, "UR": 598}
print codes
print codes["AR"]
```

```
# Sets

s = set()
s.add(1)
s.add(2)
print s
print 1 in s
print 3 in s
```

```
# Argumentos variables

def g(**kwargs):
    x = kwargs["x"]
    y = kwargs["y"]
    return x**y
g(x = 2, y = 3)
```

```
# Listas y Tuplas

integers = [1, 2, 3]
heterogeneous = ["string", 0.1, True]
lists = [integers, heterogeneous, [ ] ]
mylist = [1, 2]
mytuple = (1, 2)
mytuple2 = 1, 2
print mylist, mytuple, mytuple2
```

```
# Rangos

numbers = range(5)
print numbers
[0, 1, 2, 3, 4, 5]

print numbers[0:2], numbers[:2]
print 2 in numbers
print 15 in numbers
```

```
# Condicionales

if 1 > 2:
    print "Nope"
elif 1 > 3:
    print "Nope"
else:
    print "Here"
```

Python 2: Un poco mas avanzado

```
# List comprehensions

>>> even = [x for x in range(5) if x % 2 == 0]
>>> print even
[0, 2, 4]
>>> squared = [x * x for x in range(5)]
>>> print squared
[0, 1, 4, 9, 16]
```

```
# Map Reduce

>>> import re
>>> words = document.split()
>>> p = re.compile("imaginari\S")
>>> matches =  map(lambda x: bool(re.match(p, x)), words)
>>> print matches
[False, False, True, False,  …. ]
>>> reduce(lambda x,y: x**y, [2,1,3,2])
64
>>> reduce(max, [1, 7, 3, 4, 5, 6])
7
```

```
# Generadores e Iteradores

>>> mylist = [x*x for x in range(3)]
>>> for i in mylist:
...     print(i)
0
1
4
>>> mygenerator = (x*x for x in range(3))
>>> for i in mygenerator:
...     print(i)
0
1
4
>>> def createGenerator():
...     mylist = range(3)
...     for i in mylist:
...         yield i*i
...
>>> mygenerator = createGenerator()
>>> print(mygenerator) # mygenerator is an object!
<generator object createGenerator at 0xb7555c34>
>>> for i in mygenerator:
...      print(i)
0
1
4
```

Pythoneando

- Paths y librerias

```
$ echo $PATH
/home/apps/python/2.7.10/bin/:/home/apps/astro/sbin:/home/apps/astro/bin:
/home/jcm/src/omnetpp-4.6/bin: ….  /usr/bin:/usr/local/sbin:/usr/sbin
$ echo $LD_LIBRARY_PATH
/home/apps/python/2.7.10/lib:/home/apps/astro/lib: …. /home/apps/intel/2016/lib/intel64
```

- Module environments

```
$ module list
Currently Loaded Modulefiles:
  1) intel/2016       2) impi/5.1.3.181
$ module load python/2.7.10
$ module list
Currently Loaded Modulefiles:
  1) intel/2016       2) impi/5.1.3.181     3) python/2.7.10
```

Pythoneando

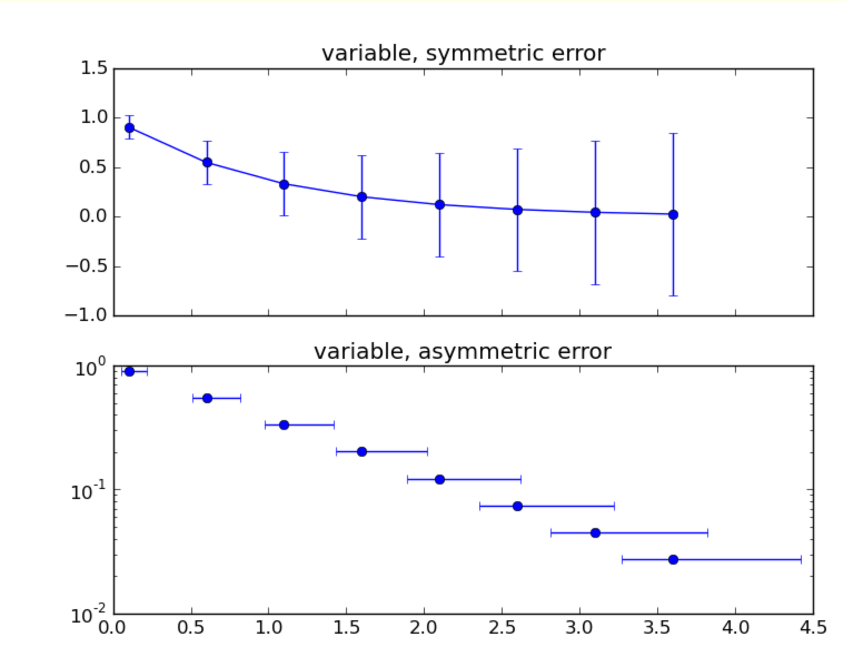
- Instalando Modulos como usuario

```
$ python
>>> import dispy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named dispy

[Control-D]

$ pip install --user dispy
Collecting dispy
  Downloading dispy-4.6.16.tar.gz (281kB)
    100% |████████████████████████████████| 286kB 1.3MB/s
Collecting asyncoro>=4.2.2 (from dispy)
  Downloading asyncoro-4.2.2.tar.gz (356kB)
    100% |████████████████████████████████| 358kB 1.3MB/s
Installing collected packages: asyncoro, dispy
  Running setup.py install for asyncoro ... done
  Running setup.py install for dispy ... done
Successfully installed asyncoro-4.2.2 dispy-4.6.16
$ python
>>> import dispy
>>>
```

Ojo con las dependencias!!

Numpy & matplotlib

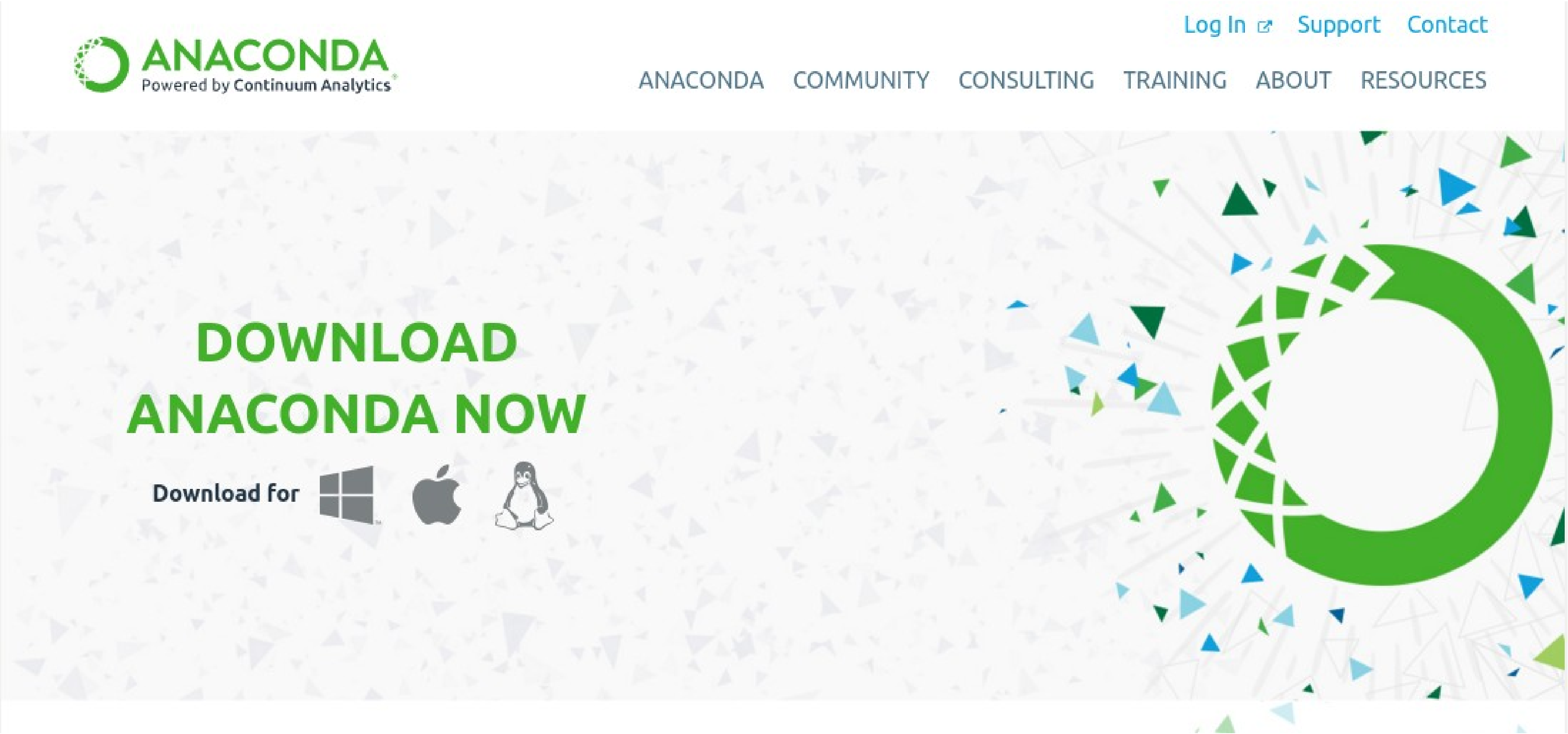
```
>>> import numpy as np
>>> x = np.array(range(10))
>>> y = np.array(range(10))**2
>>> print x, y
[0 1 2 3 4 5 6 7 8 9] [ 0  1  4  9 16 25 36 49 64 81]

>>> print np.shape(x)
(10,)

>>> z1 = np.hstack([x, y])
>>> print np.shape(z1)
(20,)

>>> pri
[ 0  1  2

>>> np
[ 0.  1.

>>> z =
>>> pri
[0 1 2 3
0.7764
0.3342
```



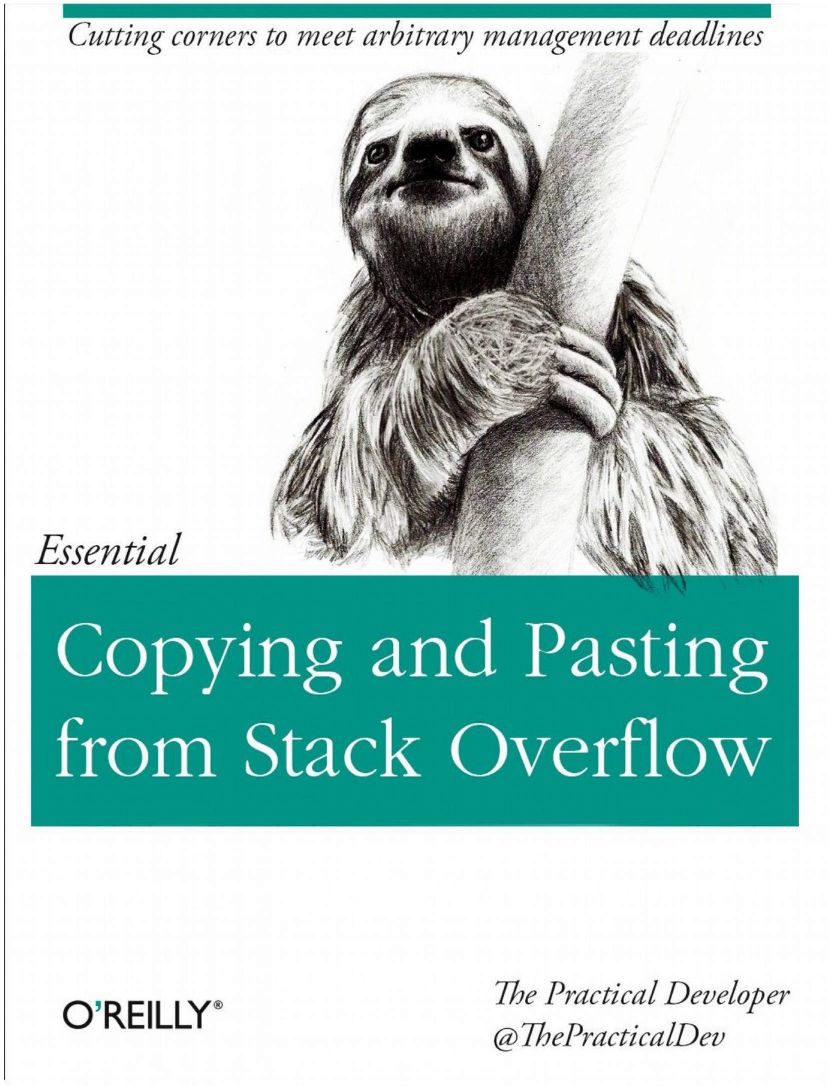
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0.1, 4, 0.5)
y = np.exp(-x)

error = 0.1 + 0.2 * x

lower_error = 0.4 * error
upper_error = error
asymmetric_error = [lower_error, upper_error]

fig, (ax0, ax1) = plt.subplots(nrows=2, sharex=True)
ax0.errorbar(x, y, yerr=error, fmt='-o')
ax0.set_title('variable, symmetric error')

ax1.errorbar(x, y, xerr=asymmetric_error, fmt='o')
ax1.set_title('variable, asymmetric error')
ax1.set_yscale('log')
plt.show()

print x
[ 0.1  0.6  1.1  1.6  2.1  2.6  3.1  3.6]
print y
[ 0.90483742  0.54881164  0.33287108  0.20189652
0.12245643  0.07427358  0.0450492   0.02732372]
print error
[ 0.12  0.22  0.32  0.42  0.52  0.62  0.72  0.82]
```

Pythoneando

- Python para el “ciudadano de a pie”



Literatura ?


Cutting corners to meet arbitrary management deadlines
Essential
Copying and Pasting from Stack Overflow
O'REILLY®
The Practical Developer
@ThePracticalDev

Es muy dificil ser original al momento de preguntar algo en la web

(siempre hay alguien que ya la hizo)

The internet will make those bad words go away
Essential
Googling the Error Message
O RLY?
The Practical Developer
@ThePracticalDev

Conceptos Básicos

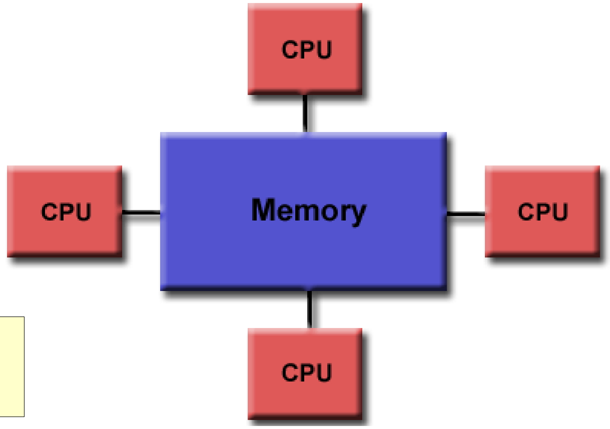
(segunda parte)

(sacando tornillos con un cuchillo de mesa)

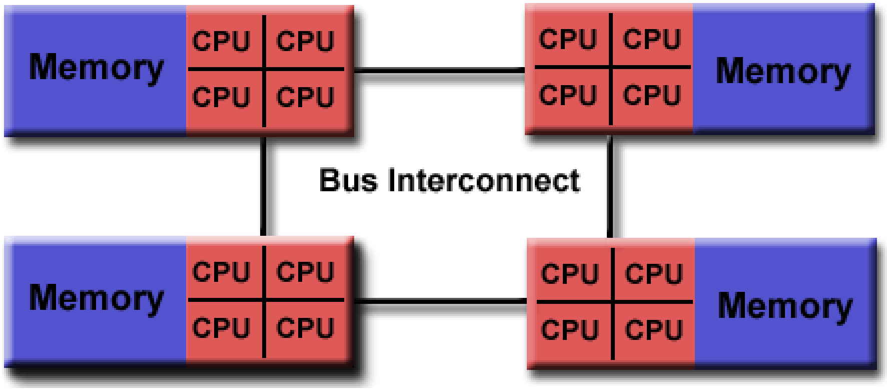
Conceptos Básicos

Memoria Compartida/Distribuida

- Un solo computador

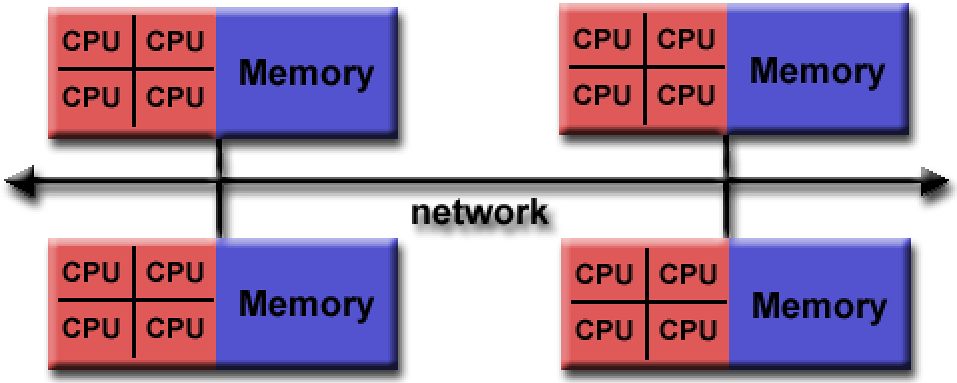


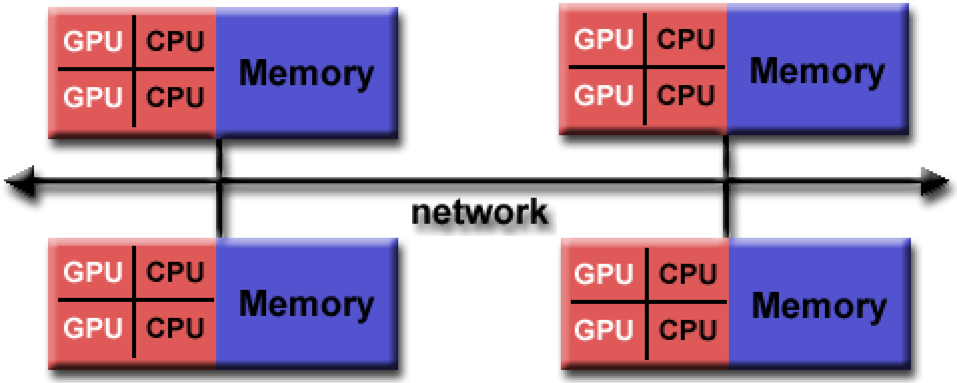
Single Core (UMA)



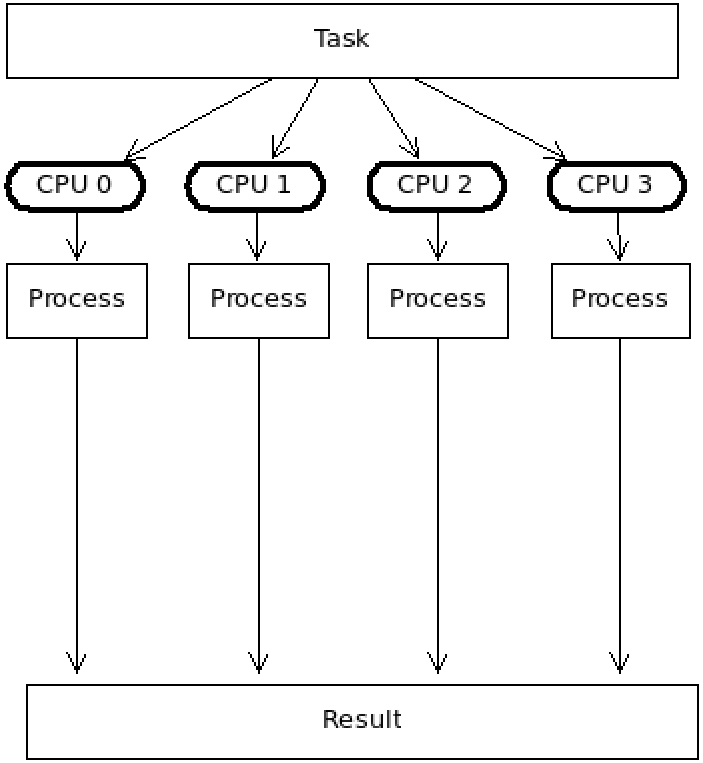
Multi Core (NUMA)

- Muchos computadores

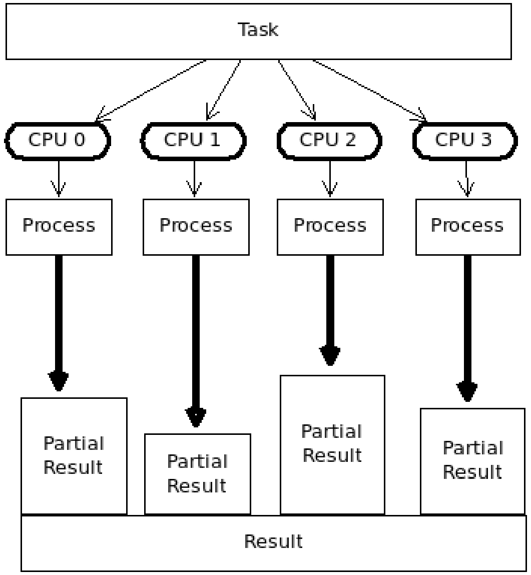




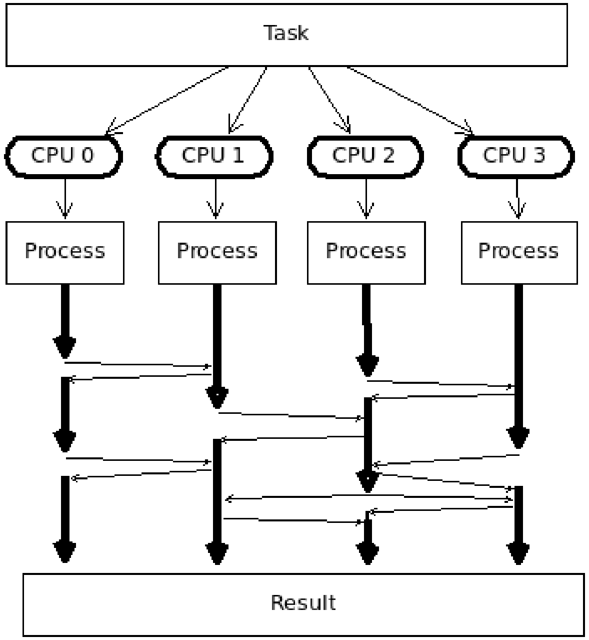
Conceptos Básicos
Paralelo / Distribuido



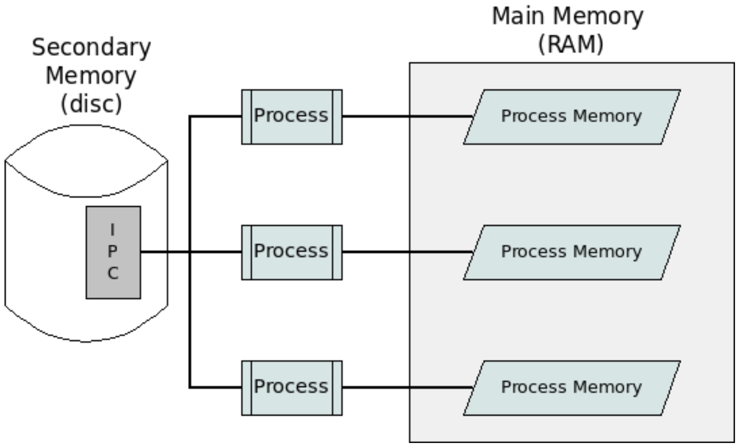
Distribuido

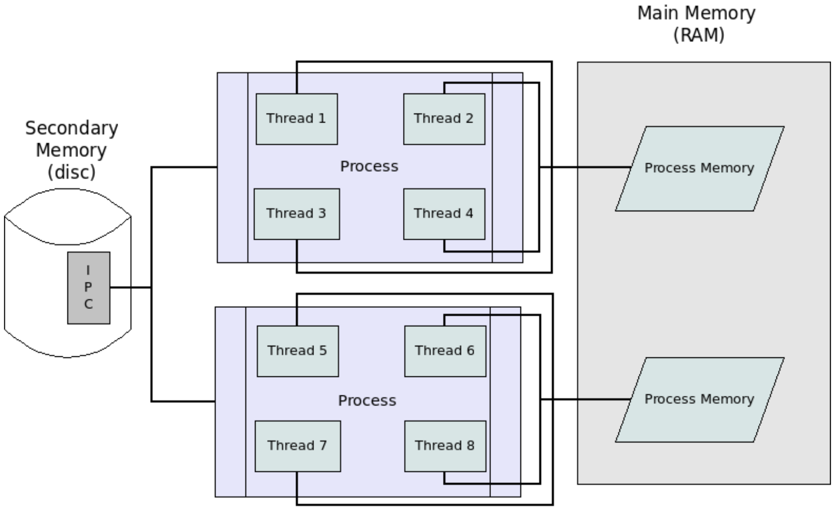


Paralelo



Conceptos Básicos
Proceso / Thread

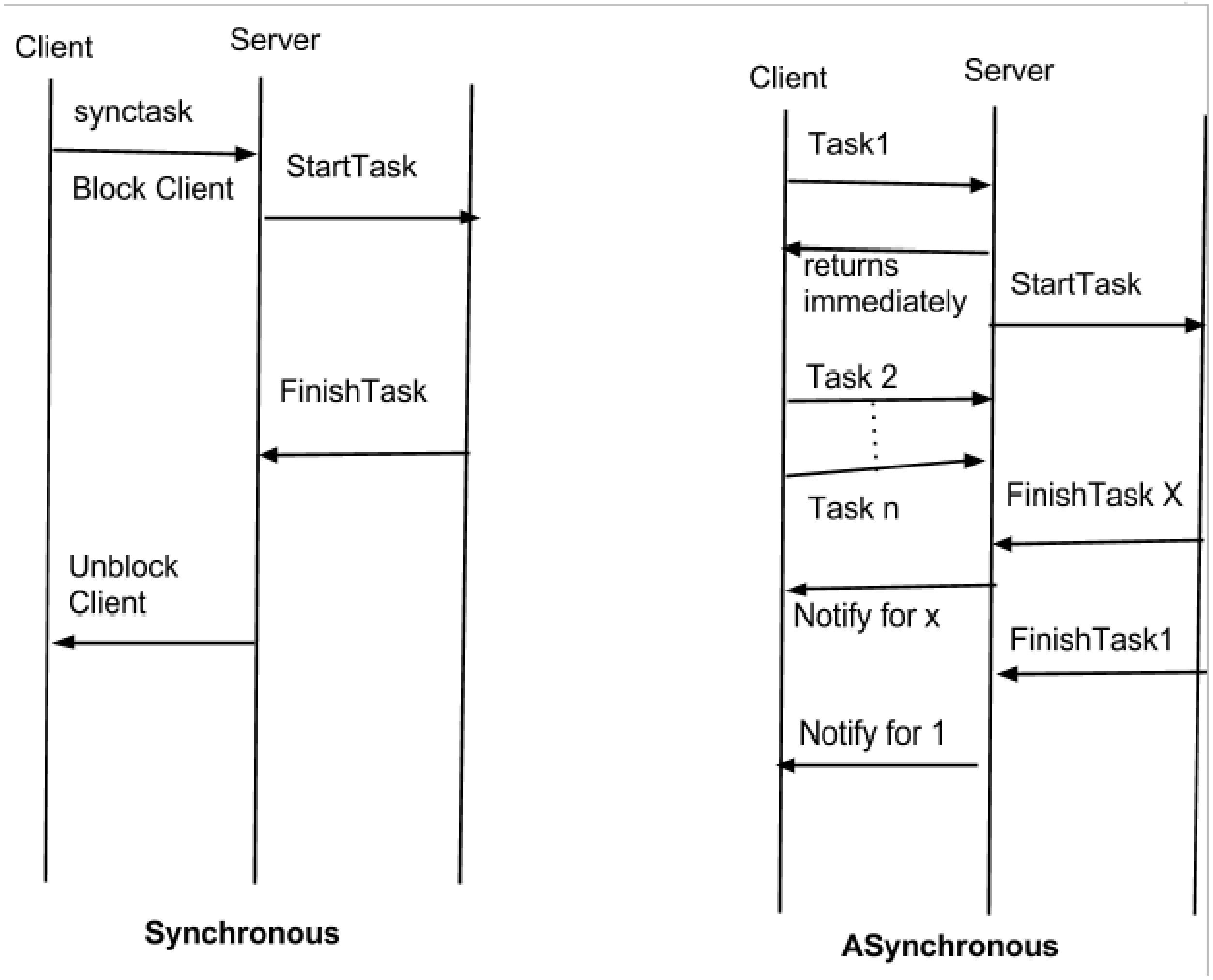




- Tareas pesadas e independientes.
- **Diferentes espacios de direcciones de memoria**, descriptores de archivos, stack, etc.
- Solo un control de ejecucion (la rutina main).
- Cada proceso tiene una c**opia del espacio de memoria del proceso padre**.
- Se comunican via *Inter Process Communication*
- No implementa locks u otra semantica de acceso exclusivo de memoria (no lo require)

- Tareas livianas y cooperativas.
- **El mismo espacio de direcciones de memoria,** descriptores de archivos, stack, etc.
- Multiples controles de ejecucion (uno por thread)
- Cada thread tiene **acceso al mismo espacio de direcciones del proceso padre**.
- Se comunican directamente.
- Implementa locks y otras semanticas para acceso exclusivo de memoria y codigo.

Conceptos Básicos
Sincrono / Asincrono

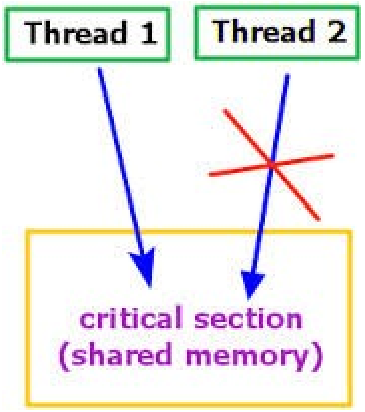


- Bloqueo del proceso que llama (calling thread)
- Facil determinar el estado de ejecucion
- Dificil para explotar multicore.

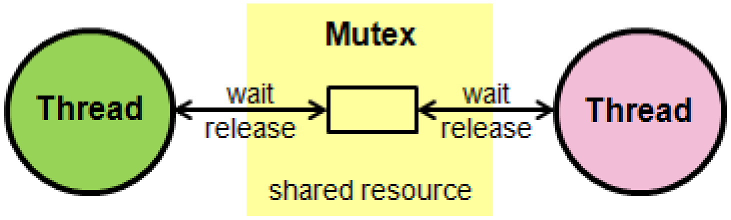
- El calling thread sigue su curso.
- Dificil determinar el estado de ejecucion (empieza el paralelismo!!!!)
- Lazy Evaluation
- Future / Promise
- Wait / Notify

Conceptos Básicos
Locks / Mutex / Semaforos

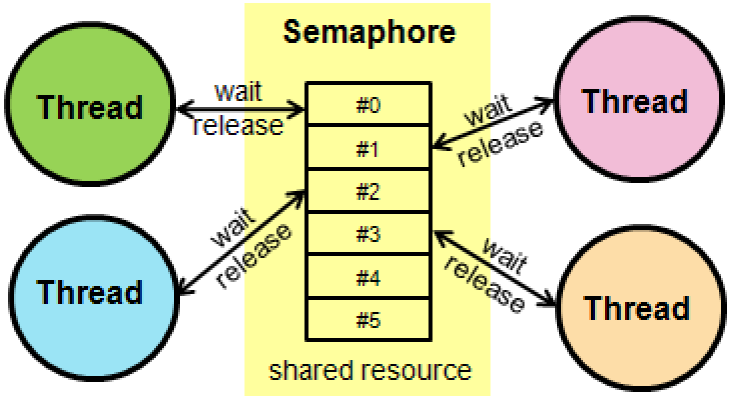
- Concurrencia
  - Lock
    (aka seccion critica).



  - Semaforo Mutex
    (aka mutex).



  - Semaforo de conteo
    (aka semaforo).



Frameworks para computación Paralela y distribuida en Python

(la punta del iceberg)

Frameworks para procesamiento paralelo/distribuido

Multiprocessing (mp)

- Basado en procesos
- Funciona en *nix y Windows
- Local o “remoto” (hacerlo remoto a mano)
- Permite intercambiar objetos entre procesos
- Sincronizacion (locks)
- Pools, managers, queues, locks, pipes, events.

```
$ cat shmem.py
from multiprocessing import Process, Value, Array

def f(n, a):
    n.value = 3.1415927
    for i in range(len(a)):
        a[i] = -a[i]

if __name__ == '__main__':
    num = Value('d', 0.0)
    arr = Array('i', range(10))

    p = Process(target=f, args=(num, arr))
    p.start()
    p.join()

    print num.value
    print arr[:]

$ python shmem.py
3.1415927
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
$
```

Frameworks para procesamiento paralelo/distribuido

Threads

- API compatible with multiprocessing
- Compatible con *nix y Windows
- **GIL: solo un thread puede ser ejecutado a la vez!!!**
- Comparten la memoria del proceso padre
  - Tambien pueden tener datos locales
- Conditions, Locks (Rlocks), Semaforos, Events, Timers.

```
$ cat threads.py
import threading, logging, time, random

logging.basicConfig(level=logging.DEBUG,
                    format='(%(threadName)-10s) %(message)s',
                    )

class MyThread(threading.Thread):

    def run(self):
        wt = random.randint(1,10)
        logging.debug('running for %d',wt)
        time.sleep(wt)
        logging.debug('done')
        return

threads = []
for i in range(5):
    threads.insert(i,MyThread())
    threads[i].start()

[t.join() for t in threads]

$
```

Frameworks para procesamiento paralelo/distribuido

Parallel Python (pp)

- Orientado a computacion distribuida en varios nodos (tambien sirve para multicore).
- Similar a multiprocessing remoto, pero mas simple
- Basado en el paradigma Job->Submit->results.
- Numero de workers dinamico.
- Balance de carga dinamico
- Multiplataforma.
- Worker (ppserver.py)

```
import math, sys, time, pp

def worker(n):
    """a dummy worker that compute n*n"""
    response = n*n
    return response

# tuple of all parallel python servers to connect with
ppservers = ()

ncpus = int(sys.argv[1])
job_server = pp.Server(ncpus, ppservers=ppservers)

print "Starting pp with", job_server.get_ncpus(), "workers"

start_time = time.time()

# The following submits one job per element in the list
inputs = range(1,100000)
jobs = [(input, job_server.submit(worker,(input,))) for input in inputs]
for input, job in jobs:
    print "result: ", input, "is", job()

print "Time elapsed: ", time.time() - start_time, "s"
job_server.print_stats()
```

Frameworks para procesamiento paralelo/distribuido

Distributed Python (dispy)

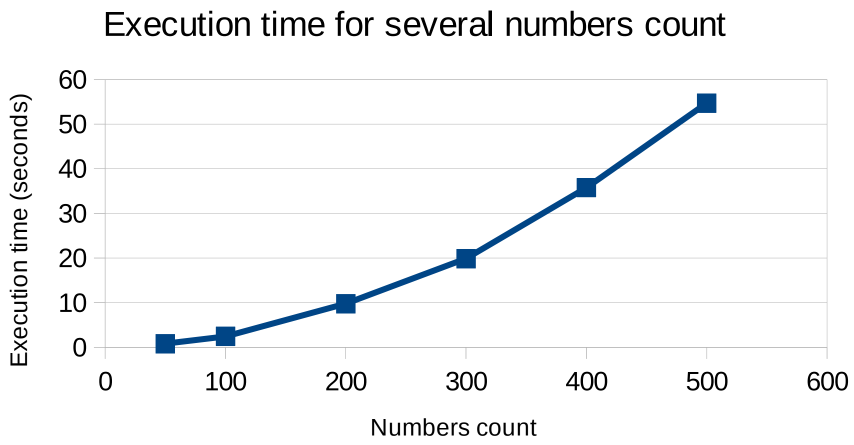
- Similar a parallel python.
- Worker explicito: distnode.py
- Se puede tener un nodo de scheduler
- No comunicacion entre workers
- Transferencia de archivos

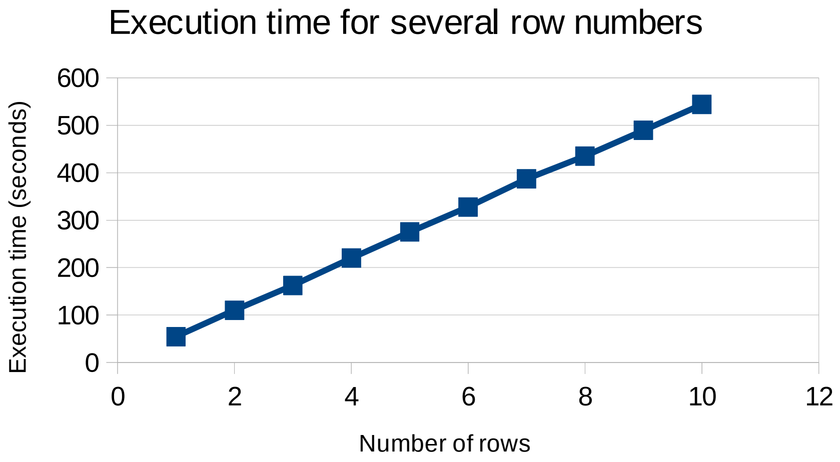
```
def compute(n):
    import time, socket
    print("woker sleeping for %d",n)
    time.sleep(n)
    host = socket.gethostname()
    return (host, n)

if __name__ == '__main__':
    import dispy, random
    cluster = dispy.JobCluster(compute)
    jobs = []
    for i in range(10):
        job = cluster.submit(random.randint(5,20))
        job.id = i
        jobs.append(job)
    for job in jobs:
        host, n = job() # waits for job to finish and returns results
        print('%s executed job %s at %s with
                        %s' % (host, job.id, job.start_time, n))
    cluster.print_status()
```

Ejemplo Practico

PowerCouples

Caso Figura: PowerCouples
Codigo Base


Execution time for several numbers count


Execution time for several row numbers

```
import os, sys, argparse, csv, itertools

def pow(x):
    return x[0]*x[1]

def find_powerCouple(numbers):
    tuples = itertools.permutations(numbers,2)
    return max(tuples, key=pow)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(
         description='PowerCouples Serial native version')
    parser.add_argument('-i','--input',
         dest="input_csv", help="input file in csv format",
         required=True, type=argparse.FileType('r'))
    parser.add_argument('-o','--output',
         dest="output_csv", help="output file in csv format",
         default=sys.stdout, type=argparse.FileType('w'))

    args = parser.parse_args()
    out = csv.writer(args.output_csv)
    for row in csv.reader(args.input_csv):
        name = row[0]
        numbers = [int(i) for i in row[1:] ]
        pc = find_powerCouple(numbers)
        out.writerow( (name, pc[0], pc[1]) )
```

Caso Figura: PowerCouples

Multiprocessing

```
import os, sys, argparse as ap, csv, itertools
import pew.pew as pw
import multiprocessing as mp

def pow(x):
    return x[0]**x[1]

def pewT(x):
    return pw.pew(x[0],x[1])

def find_powerCouple(numbers):
    tuples = itertools.permutations(numbers,2)
    return max(tuples, key=pewT)

def worker(infile,out_q):
    try:
        results = []
        print "processing %s",infile
        for row in csv.reader(infile):
            name = row[0]
            numbers = [int(i) for i in row[1:] ]
            pc = find_powerCouple(numbers)
            results.append( (name, pc[0], pc[1]) )
        out_q.put(results)
    except:
        print "worker failed"
    finally:
        print "done"
```

```
if __name__ == "__main__":
    parser = ap.ArgumentParser()
    parser.add_argument('-i','--inputs',nargs='+',
         dest="inputs_csv", help="list of input files",
         required=True, type=ap.FileType('r'))
    parser.add_argument('-o','--output', dest="output_csv",
         help="output file in csv format", default=sys.stdout,
         type=ap.FileType('w'))

    args = parser.parse_args()
    out = csv.writer(args.output_csv)
    m = mp.Manager()
    result_queue = m.Queue()

    jobs = []
    for infile in args.inputs_csv:
        jobs.append( mp.Process(target=worker,
                  args=(infile,result_queue)))
        jobs[-1].start()

    for p in jobs:
        p.join()

    num_res=result_queue.qsize()
    while num_res>0:
        out.writerows(result_queue.get())
        num_res -= 1
```

Caso Figura: PowerCouples

Threads

```
import os
import sys
import argparse
import csv
import itertools
import pew.pew as pw
import threading

def pow(x):
    return x[0]**x[1]

def pewT(x):
    return pw.pew(x[0],x[1])

def find_powerCouple(numbers):
    tuples = itertools.permutations(numbers,2)
    return max(tuples, key=pewT)

def worker(infile,out,lock):
    for row in csv.reader(infile):
        name = row[0]
        numbers = [int(i) for i in row[1:] ]
        pc = find_powerCouple(numbers)
        with lock:
            out.writerow( (name, pc[0], pc[1]) )

    return True
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-i','--inputs',nargs='+',
        dest="inputs_csv", help="list of input file in csv format",
        required=True, type=argparse.FileType('r'))

    parser.add_argument('-o','--output', dest="output_csv",
        help="output file in csv format", default=sys.stdout,
        type=argparse.FileType('w'))

    args = parser.parse_args()
    out = csv.writer(args.output_csv)

    out_lck = threading.Lock()
    threads = []
    for infile in args.inputs_csv:
        t = threading.Thread(target=worker,
                  args=(infile,out,out_lck))
        threads.append(t)
        t.start()

    print "waiting for termination"
    for t in threads:
        t.join()

    print "done"
```

Caso Figura: PowerCouples

Parallel Python

```
import os, sys, argparse, csv, itertools
import pew.pew as pw
import pp

def pow(x):
    return x[0]**x[1]

def pewT(x):
    return pw.pew(x[0],x[1])

def find_powerCouple(numbers):
    tuples = itertools.permutations(numbers,2)
    return max(tuples, key=pewT)

def worker(infile):
    results = []
    for row in csv.reader(infile):
        name = row[0]
        numbers = [int(i) for i in row[1:] ]
        pc = find_powerCouple(numbers)
        results.append( (name, pc[0], pc[1]) )

    return results
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-i','--inputs',nargs='+',
         dest="inputs_csv",
        help="list of input file in csv format", required=True,
        type=argparse.FileType('r'))
    parser.add_argument('-o','--output', dest="output_csv",
        help="output file in csv format", default=sys.stdout,
        type=argparse.FileType('w'))

    args = parser.parse_args()
    out = csv.writer(args.output_csv)
    ncpus = 10
    jobs = []
    ppservers = ()
    job_server = pp.Server(ncpus, ppservers=ppservers)

    for infile in args.inputs_csv:
        f = list(infile,)
        jobs.append(job_server.submit(worker,(f,),
                  (find_powerCouple,pewT,pow),
                  ("csv","itertools","pew.pew as pw")))

    for job in jobs:
        out.writerows(job())

    job_server.print_stats()
```

Caso Figura: PowerCouples

Parallel Python + Slurm

```
#!/bin/bash
#
# PowerCouples Parallel Python version
#
# starting script
# 2016 (c) Juan Carlos Maureira, CMM - Uchile

IN_FILES=($@)
NUM_FILES=${#IN_FILES[@]}
CORES=20
NUM_WORKERS=`echo "scale=1; \
         ($NUM_FILES / $CORES) + 0.5" | bc | cut -f 1 -d"."`
PORT=5000
SECRET="my_secret"

module load python/2.7.10

function deploy_workers() {
    let NODES=$1

    RESOURCES=""

    if [ $NODES -le 1 ]; then
        CORES=$NUM_FILES
        RESOURCES="-n1 -c $CORES"
    else
        RESOURCES="-N $NODES -c $CORES"
    fi
```

```
 echo "running for $1 workers"

 srun --exclusive –reservation=cursomop \
     $RESOURCES -J ppserver ~/.local/bin/ppserver.py \
    -w $CORES -a -p $PORT -s $SECRET

    echo "closing workers..."
}

if [ $NUM_WORKERS -eq 0 ]; then
    echo "No input files given"
    exit 1
fi

deploy_workers $NUM_WORKERS &

sleep 1
python ./powercouples-pp.py -i ${IN_FILES[@]}
sleep 1

scancel --name ppserver -s INT

wait

echo "done"
```

Caso Figura: PowerCouples

Implementacion en C++

Implementar la funcion findPowerCouple(list) en C++ usando las bondades que nos ofrece la STL (Standard Template Library).

Estrategia 1: Optimizar el calculo de las permutaciones

Estrategia 2: Paralelizar la evaluacion de las permutaciones

Estrategia 3: Usar mejor Aritmetica (GMP)

powercouple.py

```
import os, sys, argparse, csv, itertools
import pc.find as pc

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='...')
    parser.add_argument('-i','--input', dest="input_csv",
             help="input file in csv format",
             required=True, type=argparse.FileType('r'))
    parser.add_argument('-o','--output', dest="output_csv",
             help="output file in csv format",
             default=sys.stdout, type=argparse.FileType('w'))

    args = parser.parse_args()

    out = csv.writer(args.output_csv)

    for row in csv.reader(args.input_csv):
        name = row[0]
        numbers = [int(i) for i in row[1:] ]
        t = pc.find(numbers)
        out.writerow( (name, t[0], t[1]) )
```

find.h

```
#include <vector>

std::vector<int> find(std::vector<int> numbers);
std::vector<int> find_gmp(std::vector<int> numbers);
std::vector<int> find_omp(std::vector<int> numbers);
```

Swig File

```
%module find
%{
#include "find.h"
%}

%include "std_vector.i"
namespace std {
   %template(IntVector) vector<int>;
}

%include "find.h"
```

Caso Figura: PowerCouples

findPowerCouples en C++ (find)

```
std::vector<int> find(std::vector<int> numbers) {
    std::vector<int> resp(2);
    resp[0] = 0;
    resp[1] = 0;

    // compute permutations

    const int r = 2;
    const int n = numbers.size();

    PowerCouple pc(numbers);

    std::vector<int> v(n);
    std::iota(v.begin(), v.end(), 0);


    std::uint64_t count = for_each_reversible_permutation(v.begin(),
                                v.begin() + r,
                                v.end(),
                                EvalPowerCouple(v.size(), &pc));

    // copy the power couple to the response vector
    resp[0] = pc.x;
    resp[1] = pc.y;

    // WARNING copy the vector!!
    return resp;
}
```

Respuesta: vector
De dos posiciones (tupla)

Parametros
Para calcular la permutacion

Estructura para ir
Guardando la tupla
Que tiene el max(x^y)

Vector de posiciones
Del arreglo de numeros
Inicializado en 0

Calculo de Permutaciones
Reversibles, instanciando
En cada una la clase
EvalPowerCouple

Caso Figura: PowerCouples

findPowerCouples en C++ (find_gmp)

```
std::vector<int> find_gmp(std::vector<int> numbers) {
    std::vector<int> resp(2);
    resp[0] = 0;
    Resp[1] = 0;

    mpz_t pow, p1, p2;
    mpz_init(pow);  mpz_init(p1);  mpz_init(p2);

    int x = 0;
    int y = 0;

    std::vector<std::pair<int,int>> perms = getPermutations(numbers);

    for(auto it=perms.begin();it!=perms.end();it++) {
        x = (*it).first;
        y = (*it).second;

        mpz_ui_pow_ui(p1,numbers[x],numbers[y]);
        mpz_ui_pow_ui(p2,numbers[y],numbers[x]);

        int cmp_1 = mpz_cmp(p1,p2);
        if (cmp_1 > 0) {
            int cmp_2 = mpz_cmp(p1,pow);
            if (cmp_2>0) {
                resp[0] = numbers[x];
                resp[1] = numbers[y];
                mpz_set(pow,p1);
            }
        } else {
            int cmp_2 = mpz_cmp(p2,pow);
            if (cmp_2>0) {
                resp[1] = numbers[x];
                resp[0] = numbers[y];
                mpz_set(pow, p2);
            }
        }
    }
    return resp;
}
```

Declaracion de enteros con aritmetica
De precision de GMP (mpz_t): p1 y p2 para
Calcular x^y e y^x, y pow para guardar
El maximo encontrado

Obtener un vector de tuplas con
Todas las permutaciones reversibles
del vector numbers

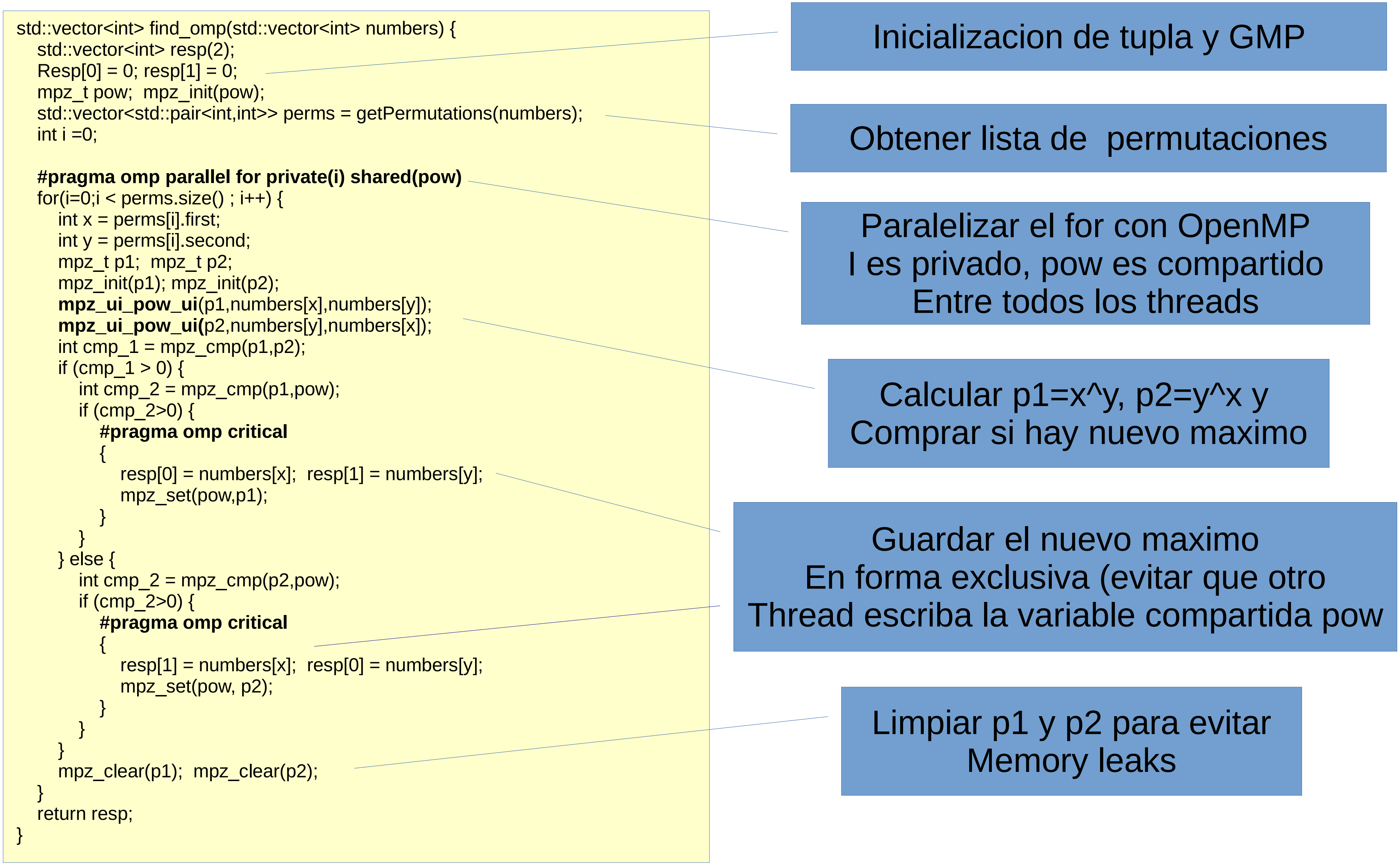
Iterarar por cada permutacion
(x,y : indices del vector numbers)

Calcular p1 = x^y, p2 = y^x
Con artimetica de numeros grandes

Si p1>p2, comparar p1 con pow y
Guardar la tupla si es nuevo maximo.
Proceso analogo para p2 y pow

Caso Figura: PowerCouples

findPowerCouples en C++ (find_omp)



```
std::vector<int> find_omp(std::vector<int> numbers) {
    std::vector<int> resp(2);
    Resp[0] = 0; resp[1] = 0;
    mpz_t pow;  mpz_init(pow);
    std::vector<std::pair<int,int>> perms = getPermutations(numbers);
    int i =0;

    #pragma omp parallel for private(i) shared(pow)
    for(i=0;i < perms.size() ; i++) {
        int x = perms[i].first;
        int y = perms[i].second;
        mpz_t p1;  mpz_t p2;
        mpz_init(p1); mpz_init(p2);
        mpz_ui_pow_ui(p1,numbers[x],numbers[y]);
        mpz_ui_pow_ui(p2,numbers[y],numbers[x]);
        int cmp_1 = mpz_cmp(p1,p2);
        if (cmp_1 > 0) {
            int cmp_2 = mpz_cmp(p1,pow);
            if (cmp_2>0) {
                #pragma omp critical
                {
                    resp[0] = numbers[x];  resp[1] = numbers[y];
                    mpz_set(pow,p1);
                }
            }
        } else {
            int cmp_2 = mpz_cmp(p2,pow);
            if (cmp_2>0) {
                #pragma omp critical
                {
                    resp[1] = numbers[x];  resp[0] = numbers[y];
                    mpz_set(pow, p2);
                }
            }
        }
        mpz_clear(p1);  mpz_clear(p2);
    }
    return resp;
}
```

Inicializacion de tupla y GMP

Obtener lista de permutaciones

Paralelizar el for con OpenMP
I es privado, pow es compartido
Entre todos los threads

Calcular p1=x^y, p2=y^x y
Comprar si hay nuevo maximo

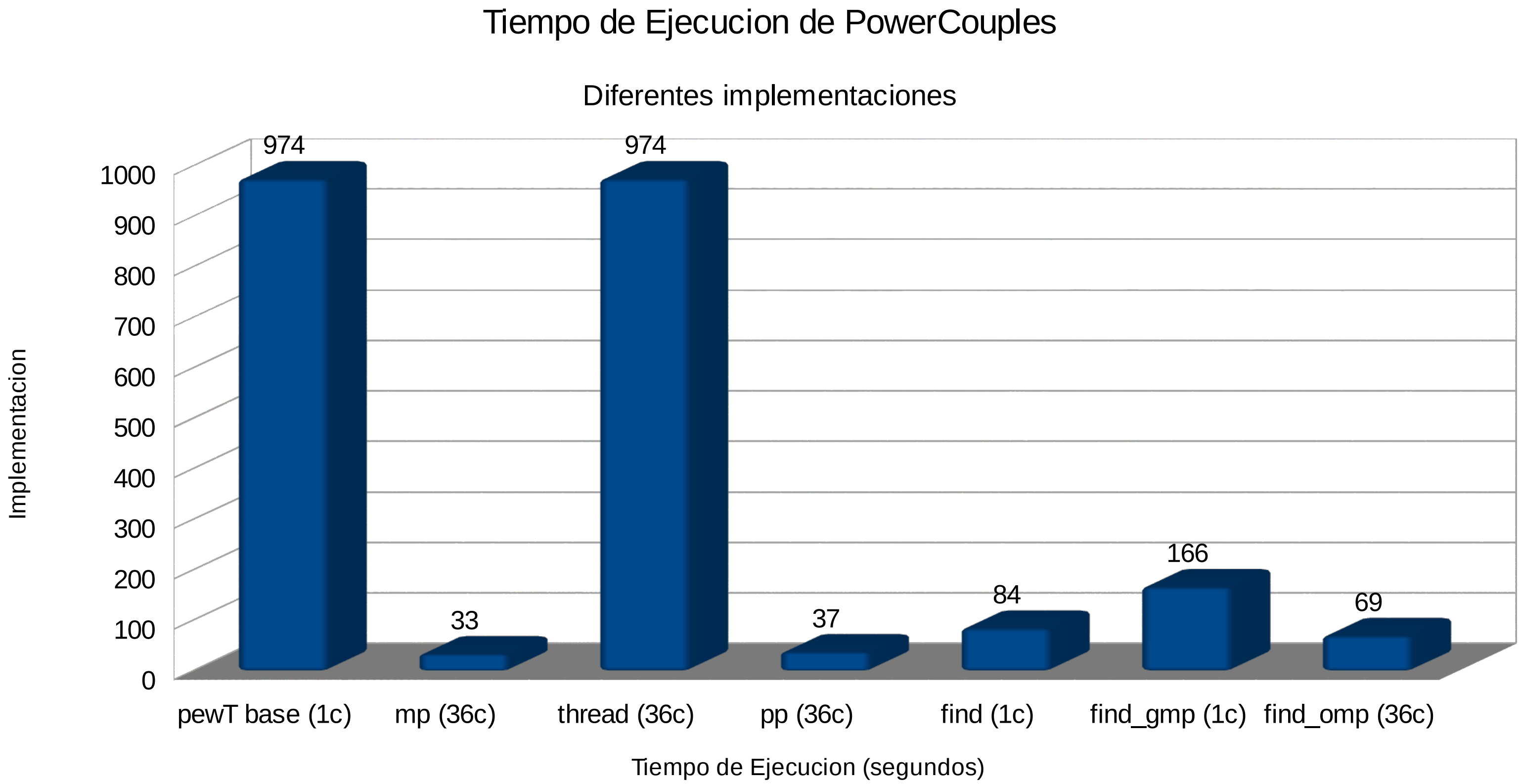
Guardar el nuevo maximo
En forma exclusiva (evitar que otro
Thread escriba la variable compartida pow

Limpiar p1 y p2 para evitar
Memory leaks

Caso Figura: PowerCouples

Comparacion de Rendimiento

- 10k filas
- 500 numeros por fila
- x ~ uniform[1-500]
- 1 core
- 36 cores



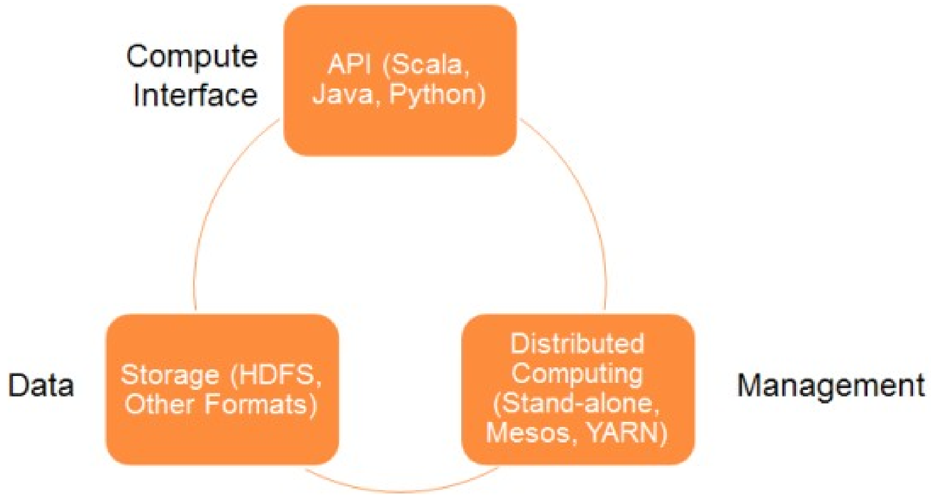
Reducción de datos con Spark

(una pincelada solamente)

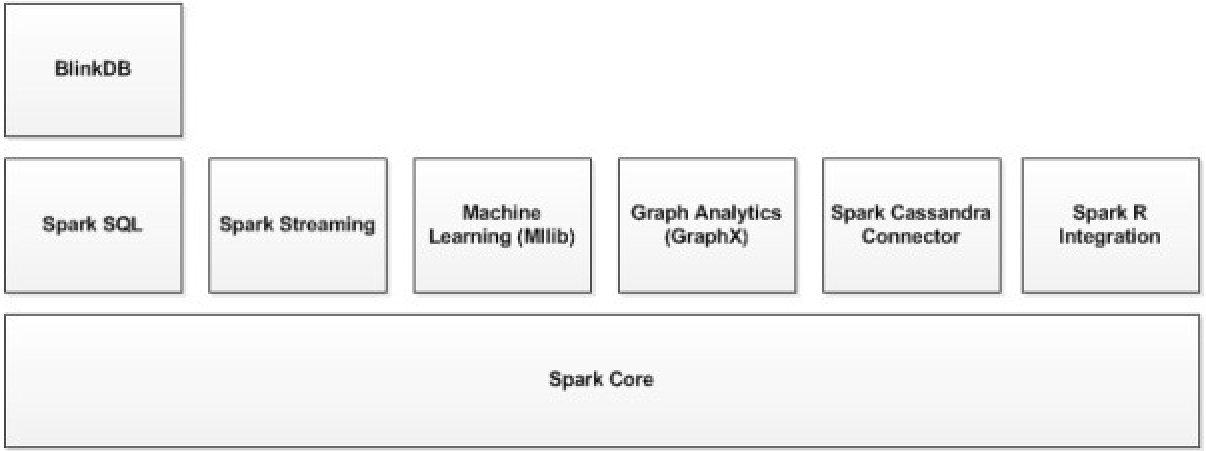
Reducción de Datos con Spark

Conceptos básicos de Spark

- Corre sobre Hadoop (si, sobre java)
- Workflow de Jobs a traves de DAGs
- Scheduler: YARN (yet another resource negotiator)
  - Esto era uno de los puntos debiles de Hadoop
- No solo MapReduce y HDFS
- Lazy Evaluation, Broadcast, shared vars
- Scala / Java / Python!!!!
- Muchos conectores para datos (no binarios)
- Hay que desplegar los workers.
  - O sea, requiere un deployment no menor



Compute Interface
API (Scala, Java, Python)
Data
Storage (HDFS, Other Formats)
Distributed Computing (Stand-alone, Mesos, YARN)
Management



Caso Figura: PowerCouples
Spark

```
# PowerCouples
# Spark (v2.0) Version
# Juan Carlos Maureira

import os
import sys
import argparse
import csv
import itertools
import pew.pew as p

from pyspark import SparkContext,SparkConf

def pow(x):
    return x[0]**x[1]

def pewT(x):
    return p.pew(x[0],x[1])

def find_powerCouple(raw_row):

    row = raw_row.split(",")
    name = str(row[0])
    numbers = [int(i) for i in row[1:] ]
    tuples = itertools.permutations(numbers,2)
    pc =  max(tuples, key=pewT)
    return [name, pc[0], pc[1]]
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-i','--input', dest="input_csv",
         help="input file in csv format", required=True)
    parser.add_argument('-o','--output', dest="output_csv",
         help="output file in csv format", default=sys.stdout,
         type=argparse.FileType('w'))

    args = parser.parse_args()

    # set the spark context
    conf = SparkConf()
    conf.setMaster("local[4]")
    conf.setAppName("PowerCouples")
    sc = SparkContext(conf=conf)

    # compute power couples
    infile = sc.textFile(args.input_csv,4)
    result = infile.map(find_powerCouple)
         .map(lambda elem: elem[0]+","
                  +str(elem[1])+","+str(elem[2])).collect()

    # write results
    out = csv.writer(args.output_csv)
    for row in result:
        out.writerow([row])
```

Caso Figura: PowerCouples

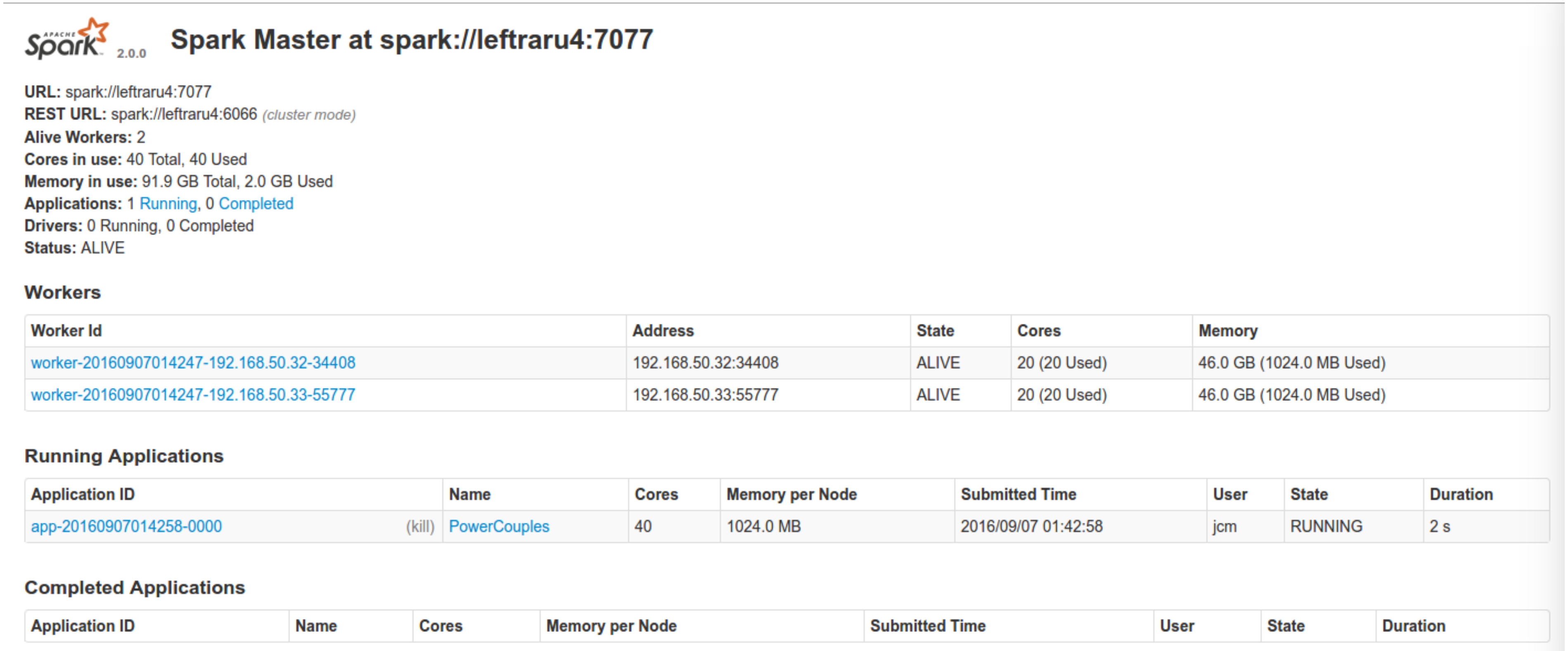
Spark + Slurm

```
#!/bin/bash
#
# Spark 2.0 Slurm submision script
# Deploy master and workers, then submit the python script
#
# 2016 (c) Juan Carlos Maureira
# Center for Mathematical Modeling
# University of Chile

# make the spark module available
module use -a $HOME/modulefiles
module load spark
Module load python/2.7.10

NUM_WORKERS=2
CORES_PER_WORKER=20

if [ "$1" == "deploy" ]; then
    # deploy spark workers on nodes

    MASTER=$2
    HOST=`hostname -a`
    echo "starting slave at $HOST"
    $SPARK_HOME/sbin/start-slave.sh \
         --cores $CORES_PER_WORKER \
         spark://$MASTER:7077

    tail -f /dev/null
```

```
else
    # main routine

    MASTER=`hostname -a`
    echo "Using as master $MASTER"
    $SPARK_HOME/sbin/start-master.sh

    srun --exclusive -n $NUM_WORKERS \
         --reservation=cursomop \
         -c $CORES_PER_WORKER \
         -J spark $0 deploy $MASTER &

    sleep 10

    spark-submit --master spark://$MASTER:7077 $@

    # clean up
    scancel --name spark
    $SPARK_HOME/sbin/stop-master.sh
    echo "done"
fi
```

Caso Figura: PowerCouples

Spark Web UI



Spark Master at spark://leftraru4:7077

URL: spark://leftraru4:7077
REST URL: spark://leftraru4:6066 *(cluster mode)*
Alive Workers: 2
Cores in use: 40 Total, 40 Used
Memory in use: 91.9 GB Total, 2.0 GB Used
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20160907014247-192.168.50.32-34408	192.168.50.32:34408	ALIVE	20 (20 Used)	46.0 GB (1024.0 MB Used)
worker-20160907014247-192.168.50.33-55777	192.168.50.33:55777	ALIVE	20 (20 Used)	46.0 GB (1024.0 MB Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160907014258-0000 (kill)	PowerCouples	40	1024.0 MB	2016/09/07 01:42:58	jcm	RUNNING	2 s

Completed Applications

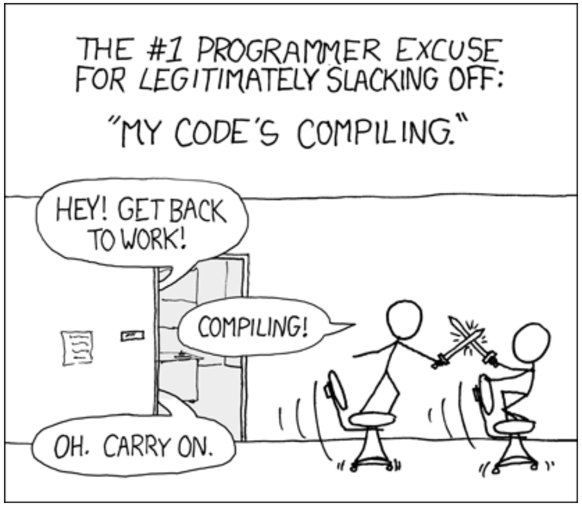
Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Finalizando...

(para llevarse a casa)

Optimización de Python et al.
Compila! Compila! Compila!

- Como hacer que mi python corra mas rápido sin tener que reimplemetar todo yo mismo?
  - Numpy/Scipy y otros módulos utilizan mucho librerías base de calculo científico
    - Blas, Lapack, fftw, TBB, GMP, etc.
  - Usar versiones optimizadas de estas librerías requiere **recompilar python** y todos los módulos que se utilicen
    - MKL de Intel → blas, fftw, tbb, gmp todo en uno!!
    - CuBLAS de Cuda → blas optimizado que usa GPU!!!
    - Usar **compiladores optimizados** para el procesador que se esta usando (intel básicamente)
      - Fp model → strict, precise, source, fast
      - Flags de compilación : xHost, O3,
  - Construir ***toolchains*** donde este todo armado, bien amarrado y optimizado para mis requerimientos.
  - **No es tarea fácil**, pero es el camino mas corto para un “free ride” en performance



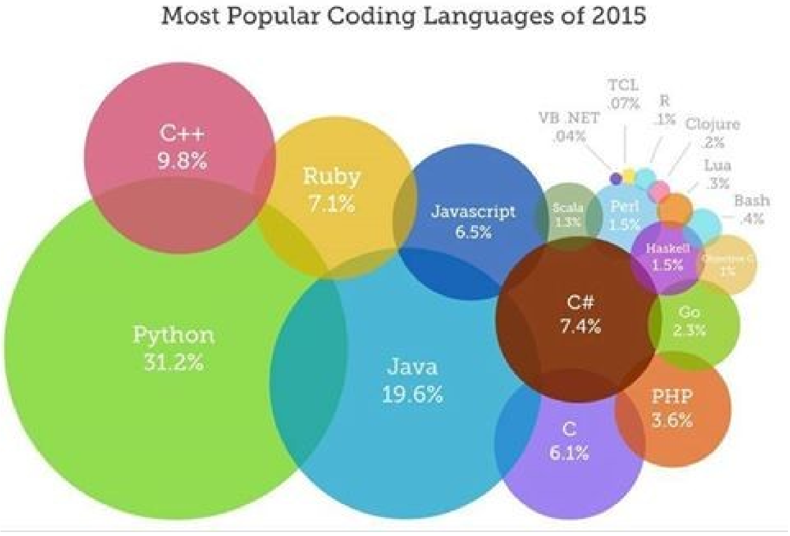
Consejos Prácticos
Desmitificando MPI

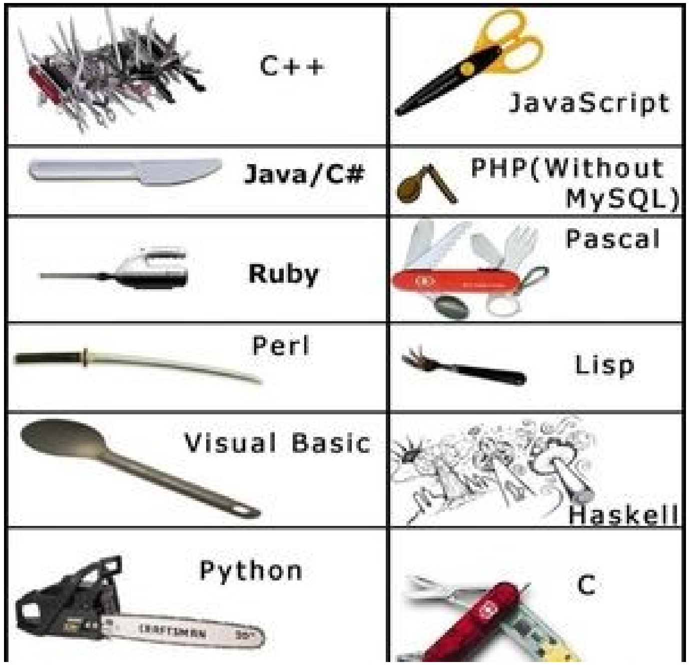
- MPI es una libreria para pasar mensajes de un proceso a otro (localmente o entre maquinas)
- Todos los frameworks vistos aca implementan la comunicacion remota en forma interna! No hay necesidad de reinventar la rueda
- MPI convine cuando se tiene un interconect rapido (como Infiniband), el cual se salta todo el kernel para enviar o recibir bloques de memoria.
- El rendimiento de una apliacion paralela (que use MPI) depende de:
  - lo intensivo de la comunicacion,
  - La latencia del interconnect,
  - La forma de implementacion.
- MPI no es bueno para
  - Enviar procesos a diversas maquinas / cores → user un sistema de colas para eso (SLURM / SGE / TORQUE)
  - Sincronizar procesos independientes → use una libreria de RMI para ello.

Consejos Prácticos
Lenguajes

“Escoja sabiamente su arma para ir a la batalla.“

“No pretenda pelear todas las batallas con la misma arma.”

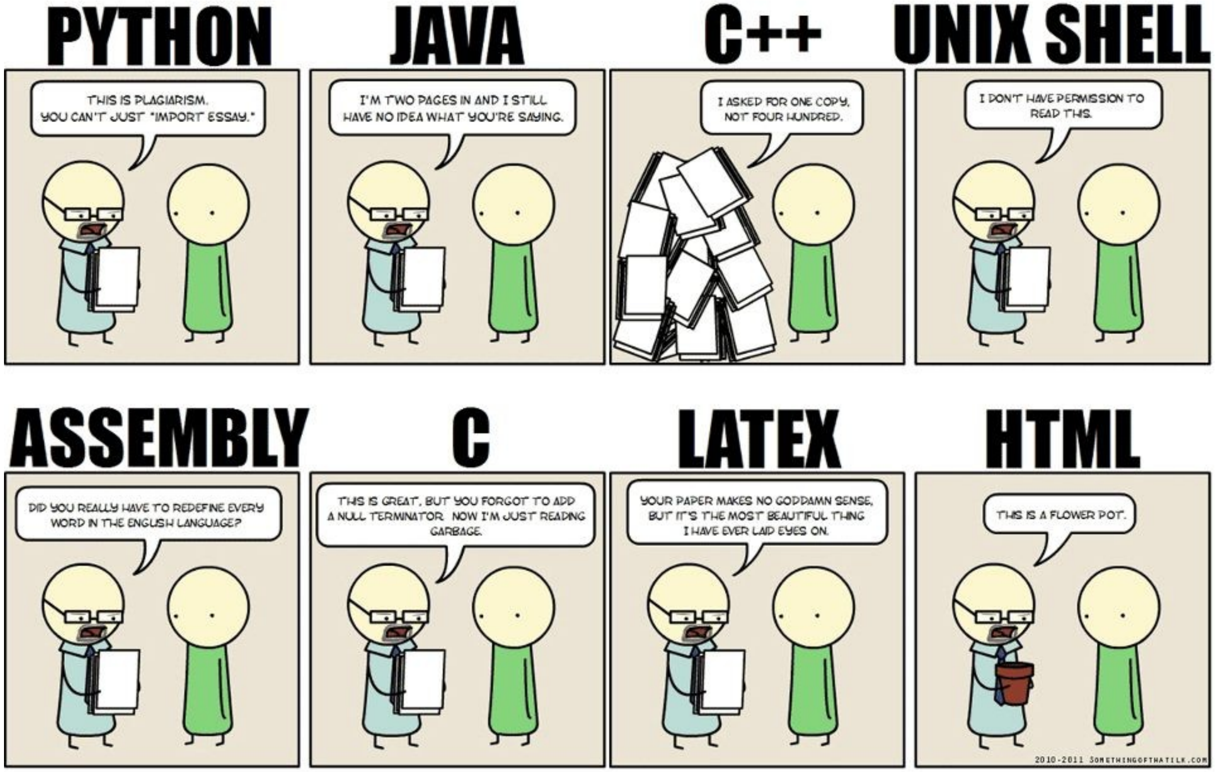


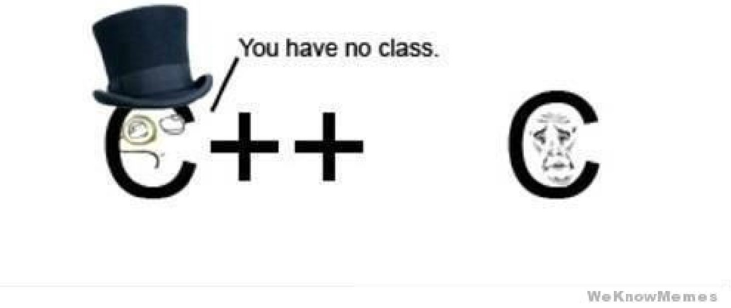


Consejos Prácticos

Poliglota

"Aprenda varios lenguajes"


PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.
C++
I ASKED FOR ONE COPY, NOT FOUR HUNDRED.
UNIX SHELL
I DON'T HAVE PERMISSION TO READ THIS.
ASSEMBLY
DID YOU REALLY HAVE TO REDEFINE EVERY WORD IN THE ENGLISH LANGUAGE?
C
THIS IS GREAT, BUT YOU FORGOT TO ADD A NULL TERMINATOR. NOW I'M JUST READING GARBAGE.
LATEX
YOUR PAPER MAKES NO GODDAMN SENSE, BUT IT'S THE MOST BEAUTIFUL THING I HAVE EVER LAID EYES ON.
HTML
THIS IS A FLOWER POT.
2010-2011 SOMETHINGOFTHATILK.COM

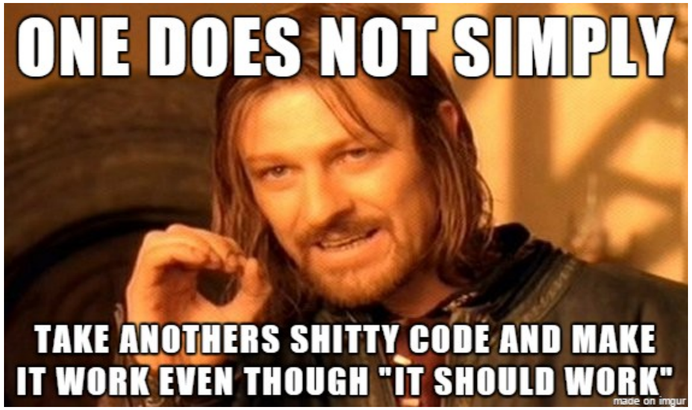


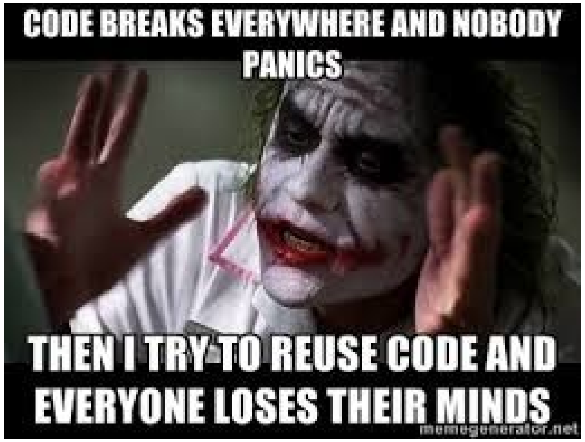
Consejos Prácticos

Reuse codigo

“La frase clásica de : si total lo usare una vez nomas y despues lo boto”

Ya no es válida hoy en día


ONE DOES NOT SIMPLY
TAKE ANOTHERS SHITTY CODE AND MAKE IT WORK EVEN THOUGH "IT SHOULD WORK"
made on imgur


CODE BREAKS EVERYWHERE AND NOBODY PANICS
THEN I TRY TO REUSE CODE AND EVERYONE LOSES THEIR MINDS
memegenerator.net

Centro de Modelamiento Matemático
Universidad de Chile

CMM
Center for
Mathematical
Modeling

Python para Data Science

Paralelismo y Distribución

Muchas Gracias
Por su atención!

Juan Carlos Maureira B.
<*jcm@dim.uchile.cl*>

Taller CMM-MOP – 08/09/2016