PyTables

Procesando y analizando enormes cantidades de datos en Python

Francesc Alted
falted@openlc.org

iParty 5, 13 de Marzo de 2003, Castelló

Esquema

- ¿Qué es PyTables, qué ofrece y porqué existe?
- Introducción a las bases de datos jerárquicas
- Demostración interactiva
- Algunos "benchmarks"
- Notas finales

Motivación

- Muchas aplicaciones necesitan guardar y leer grandes cantidades de datos --> desafío

- Los ordernadores actuales son suficientemente potentes para manejar esas cantidades de datos. El problema es:

  podemos procesarlos los humanos?

- Requisitos:
  - El análisis es un proceso iterativo: interactividad
  - Releer múltiples veces los datos: eficiencia
  - Buen entorno para dotar a los datos de una estructura
  - Fácil manejo

- PyTables es un paquete Python diseñado con estos requisitos en mente.

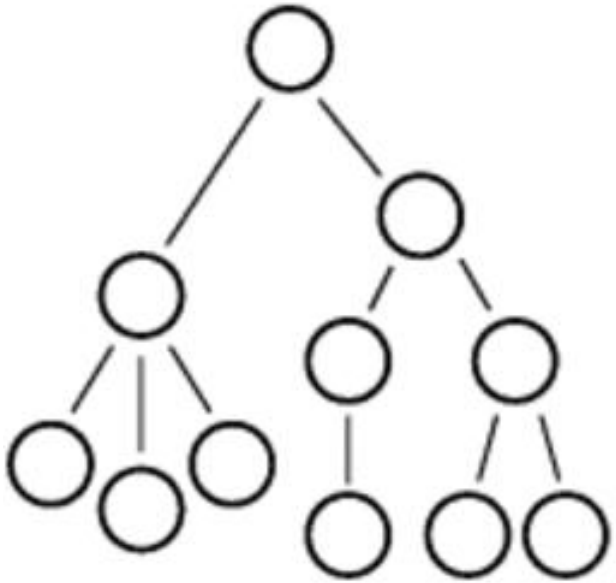
¿Qué es PyTables?

- Es una base de datos jerárquica para Python
- Permite trabajar con tablas y matrices multimensionales
- Pensada para que sea fácil de usar
- Optimizada para trabajar con grandes cantidades de datos

¿Qué ofrece PyTables?

- Interactividad
  - El usuario puede tomar acciones inmediatas dependiendo del resultado de las operaciones anteriores
  - Esto acelera enormemente el proceso de extración de información ("data mining")
- Eficiencia
  - Mejora la productividad
  - Muy importante cuando la interactividad es necesaria
- Estructura jerárquica de los datos
  - Permite organizar los datos en grupos pequeños y relacionados y en un sólo fichero
  - Ofrece una forma intuitiva y compacta de clasificar los datos
- Interface orientada al objeto
  - Los conjuntos de datos son objetos fácilemente manipulables
  - En una estructura jerárquica, facilita el acceso a los datos

Modelo de datos jerárquico

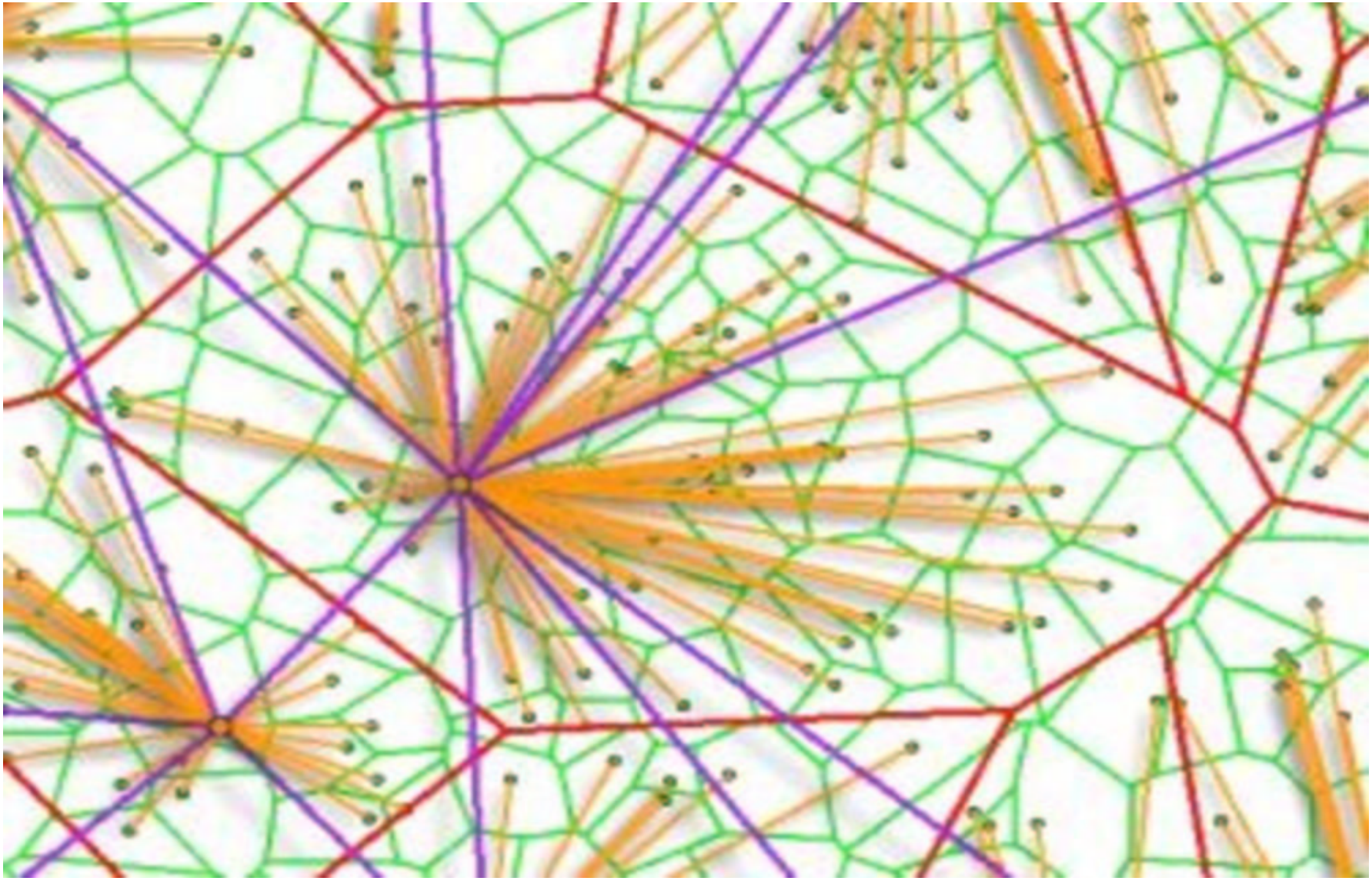


- Los datos se organizan en una estructura arbórea
- En la cúspide de la estructura se encuentra la raíz
- Cada nivel (excepto la raíz) tiene un sólo nivel por encima (padre), pero puede tener muchos niveles por debajo (hijos)

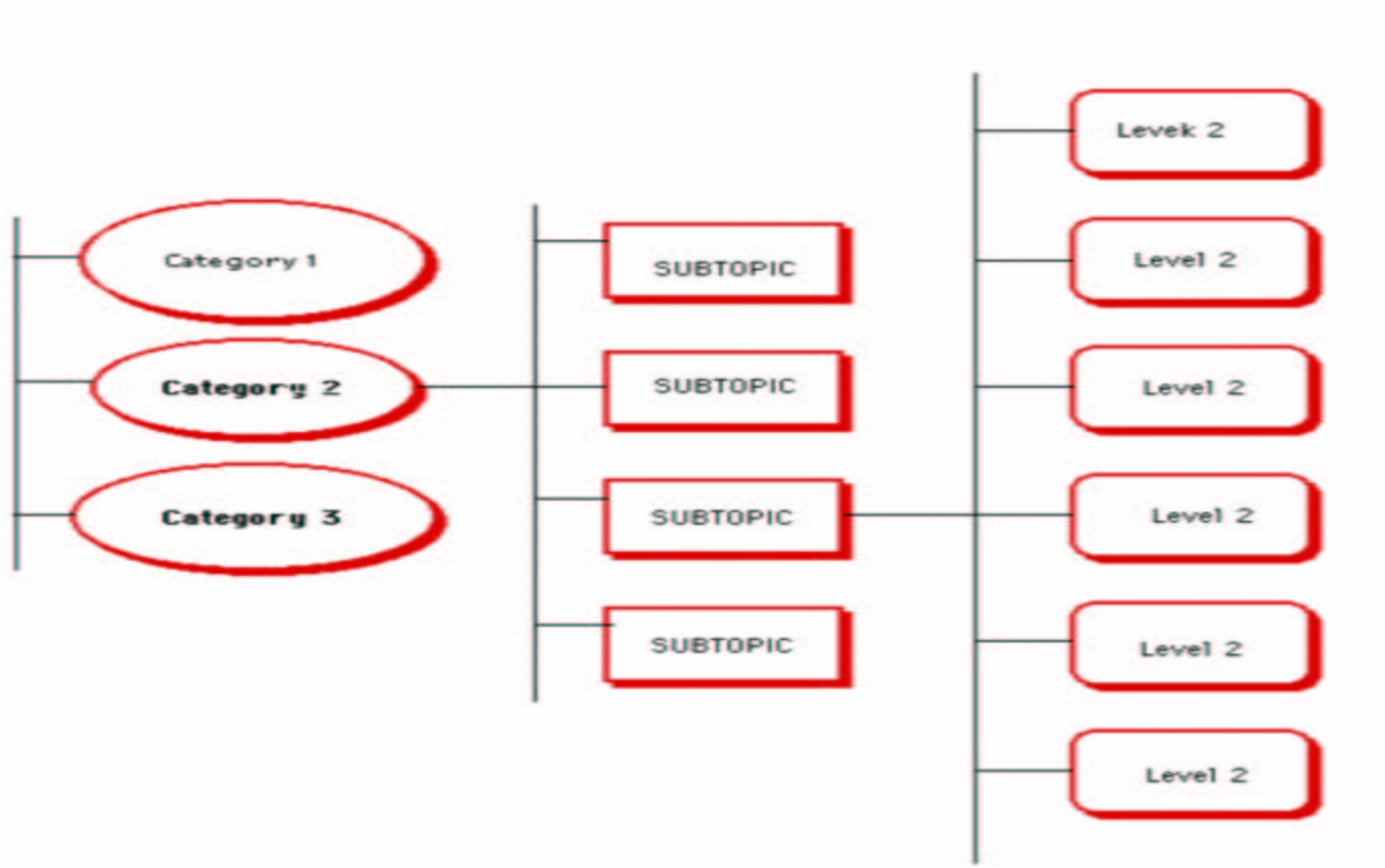
¿Por qué jerarquizar los datos?

- Aparece de manera natural en muchos casos:
  - Dispersión geográfica
  - Dispersión temporal
  - Categorización de la información (LDAP)
- La misma estructura arbórea da mucha información
  /eu/es/uji/exp
- El acceso a la información se hace más intuitivo
- La velocidad de acceso a datos es normalmente mayor que en bases de datos relacionales (tiempo de acceso de log(N) con una base de datos de tamaño N)
  - Caché (Intersystems)
  - Google
  - Web semántica (XML)

Ejemplo de dispersión geográfica



Ejemplo de categorización de materias



Maquinaria detrás de PyTables

PyTables aprovecha diferente software para conseguir sus objetivos:

- Python -- lenguaje interpretado (fundamental para lograr la interactividad)
- HDF5 -- libreria y formato de propósito general para almacenamiento de datos de carácter científico
- numarray -- la siguiente generación del paquete Numeric
- Pyrex -- herramienta que permite hacer extensiones Python con una sintaxis muy parecida a la de Python

¿Qué es HDF5?

Es una libreria y un formato de fichero diseñado para guardar datos de carácter científico de una manera jerárquica. Está desarrollado y mantenido por el NCSA en EEUU.

- Puede guardar dos objectos principales: "datasets" y grupos
  - "Dataset" -- array multidimensional de datos
  - Grupo -- estructura para organizar los objectos
- Muy flexible y bien testeado en entornos científicos
- API’s soportadas oficialmente: C, Fortran y Java
- Se usa en: Metereología, Oceanografía, Astronomía, Astrofísica, Simulación Numérica, ...

¿Qué es numarray?

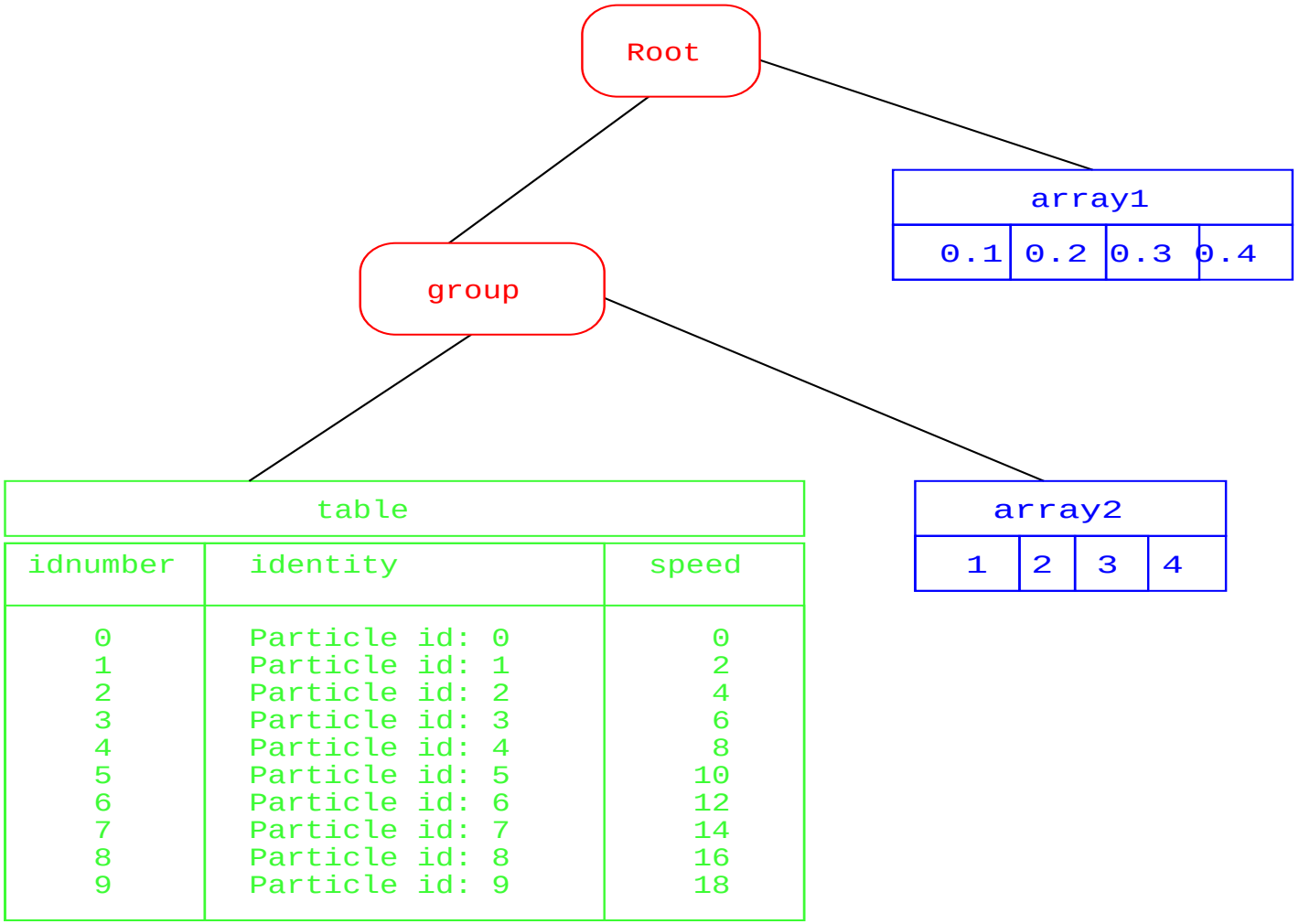
Es la próxima generación del paquete Numeric Python

- Dota de un potente lenguaje de manejo de matrices a Python
- Los cálculos con las matrices són muy rápidos
- Las matrices se guardan en memoria de manera eficiente
- El número de dimensiones de la matrices es prácticamente ilimitado (32)

Principales caracterísiticas de PyTables

- Soporte de objectos Numeric Python y numarray
- Puede leer ficheros genéricos HDF5 y trabajar con ellos
- Soporta compresión de datos de manera transparente (y eficiente)
- Puede trabajar con objectos que no caben en memoria
- Soporta ficheros más grandes que 2 GB
- Independiente de arquitectura (big/little endian)

Un primer ejemplo



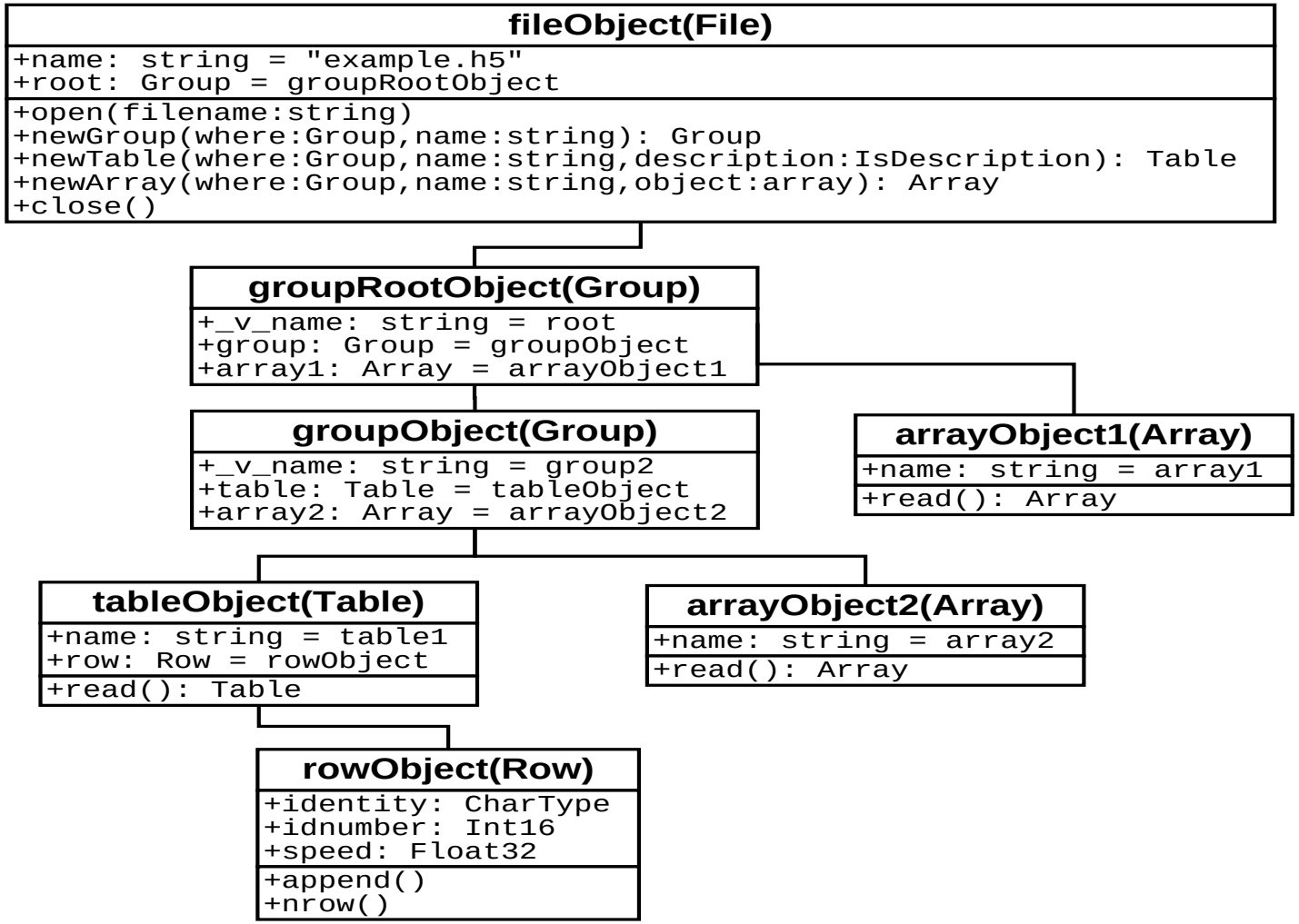
El código PyTables

```
from tables import *

class Particula(IsDescription):
    identity = Col("CharType", 16, " ", pos = 0)  # cadena de caracteres
    speed = Col("Float32", 1, pos = 2)  # precisión simple
    idnumber = Col("Int16", 1, pos = 1)  # entero corto

fileh = openFile("example.h5", mode = "w")
group = fileh.createGroup(fileh.root, "group", "un grupo")
array = fileh.createArray(fileh.root, "array1", [.1,.2,.3,.4], "array de floats")
array = fileh.createArray(group, "array2", [1,2,3,4], "array de enteros")
table = fileh.createTable(group, "table", Particula, "Tabla con 3 campos")
row = table.row
for i in xrange(10):
    row['identity']  = 'Particle id: %3d' % (i)
    row['idnumber'] = i
    row['speed']  = i * 2.
    row.append()

fileh.close()
```

Fichero resultante del primer ejemplo

```
$ h5ls -rd example.h5
/array1                  Dataset {4}
    Data:
        (0) 0.1, 0.2, 0.3, 0.4
/group                   Group
/group/array2            Dataset {4}
    Data:
        (0) 1, 2, 3, 4
/group/table             Dataset {10/Inf}
    Data:
        (0) {0, "Particle id:   0", 0}, {1, "Particle id:   1", 2},
        (2) {2, "Particle id:   2", 4}, {3, "Particle id:   3", 6},
        (4) {4, "Particle id:   4", 8}, {5, "Particle id:   5", 10},
        (6) {6, "Particle id:   6", 12}, {7, "Particle id:   7", 14},
        (8) {8, "Particle id:   8", 16}, {9, "Particle id:   9", 18}
```

El árbol de objectos



¿PyTables es rápido, pero cuánto?

- Se han efectuado varios benchmarks para saber si PyTables es competitivo con herramientas ya existentes para guardar datos en disco
- Se han hecho comparaciones cPickle, struct, shelve y SQLite
- Se ha testeado la escritura y la lectura (con filtros) de datos
- Los parámetros básicos que se han cambiado en cada test son:
  - El tamaño de fila
  - El número de filas en cada tabla

Las descripciones de las filas

Los tamaños de fila utilizados son de dos longitudes diferentes:

- 16 Bytes

```
class Small(IsDescription):
    var1 = Col("CharType", 4, "")
    var2 = Col("Int32", 1, 0)
    var3 = Col("Float64", 1, 0)
```

- 56 bytes

```
class Medium(IsDescription):
    name        = Col("CharType", 16, "")
    float1      = Col("Float64", 2, NA.arange(2))
    ADCcount    = Col("Int32", 1, 0)
    grid_i      = Col("Int32", 1, 0)
    grid_j      = Col("Int32", 1, 0)
    pressure    = Col("Float32", 1, 0)
    energy      = Col("Float64", 1, 0)
```

El mecanismo de selección

- PyTables:
  - e = [p[’var1’] for p in table.iterrows()
    if p[’var2’] < 20]
- cPickle:
  - while rec:
    record = cPickle.loads(rec[1])
    if record[’var2’] < 20:
    e.append(record[’var1’])
- struct:
  - while rec:
    record = struct.unpack(isrec._v_fmt, rec[1])
    if record[1] < 20:
    e.append(record[0])
- SQLite:
  - cursor.execute("select var1 from table where var2 < 20")

Nota: los tests con cPickle y struct usan una BBDD Berkeley (4.1.3) en su variedad RECNO, con el fin de emular la fila de manera eficiente.

Descripción de la plataforma de benchmark

- Portátil con Pentium IV @ 2 GHz and 256 MB RAM
- Disco IDE @ 4200 RPM
- PyTables 0.4
- HDF5 1.4.5
- numarray 0.4
- Linux Debian 3.0
- GCC 2.95

Comparación con cPickle y struct

Package	Record length	Krows/s		MB/s		total Krows	file size	memory used	%CPU	
		write	read	write	read		(MB)	(MB)	write	read
cPickle	small	23.0	4.3	0.65	0.12	30	2.3	6.0	100	100
cPickle	small	22.0	4.3	0.60	0.12	300	24	7.0	100	100
cPickle	medium	12.3	2.0	0.68	0.11	30	5.8	6.2	100	100
cPickle	medium	8.8	2.0	0.44	0.11	300	61	6.2	100	100
struct	small	61	71	1.6	1.9	30	1.0	5.0	100	100
struct	small	56	65	1.5	1.8	300	10	5.8	100	100
struct	medium	51	52	2.7	2.8	30	1.8	5.8	100	100
struct	medium	18	50	1.0	2.7	300	18	6.2	100	100
PyTables	small	434	469	6.8	7.3	30	0.49	6.5	100	100
PyTables	small (c)	326	435	5.1	6.8	30	0.12	6.5	100	100
PyTables	small	663	728	10.4	11.4	300	4.7	7.0	99	100
PyTables	medium	194	340	10.6	18.6	30	1.7	7.2	100	100
PyTables	medium (c)	142	306	7.8	16.6	30	0.3	7.2	100	100
PyTables	medium	274	589	14.8	32.2	300	16.0	9.0	100	100

Table 1: Comparing PyTables performance with cPickle and struct serializer modules in Standard Library. (c) means that a compression is used.

Conclusiones del primer benchmark (cPickle y struct)

PyTables resulta en:

Escritura

- Entre 20 y 30 veces más rápido que cPickle
- Entre 3 y 10 veces más rápido que struct

Lectura

- Alrededor de 100 veces más rápido que cPickle
- Alrededor de 10 veces más rápido que struct

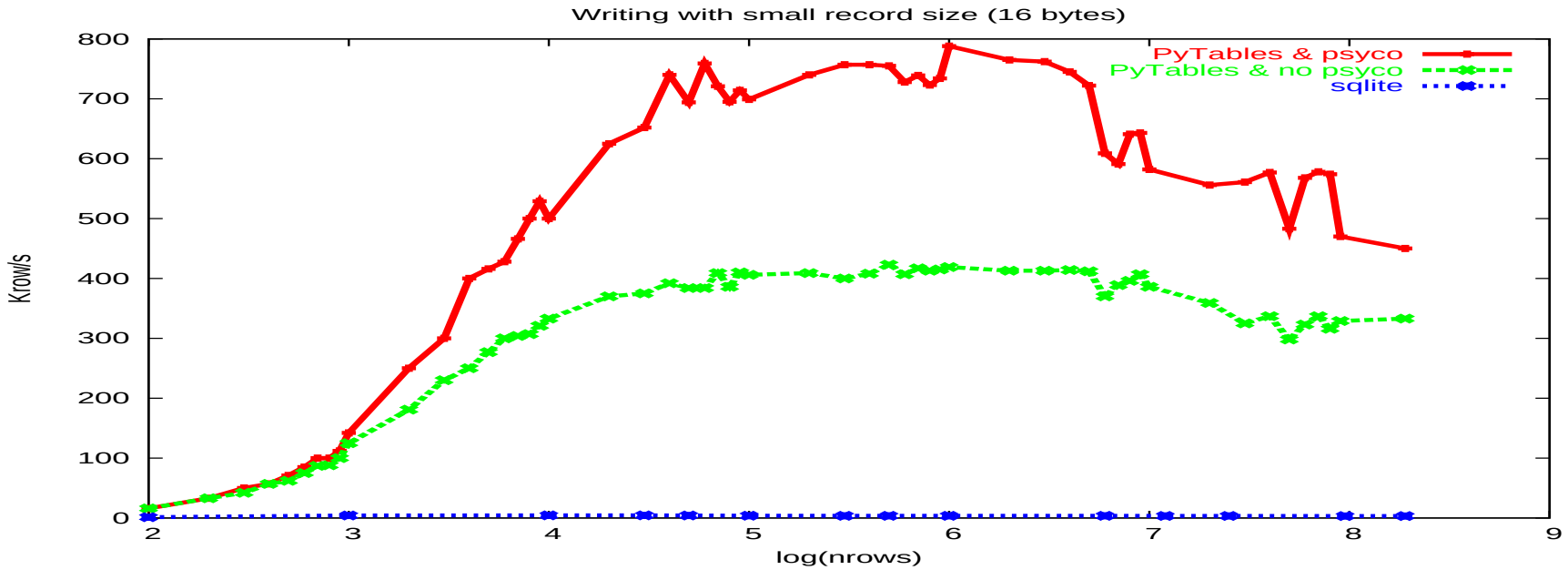
PyTables es muy superior a cPickle y struct para cualquier cantidad de datos (!)

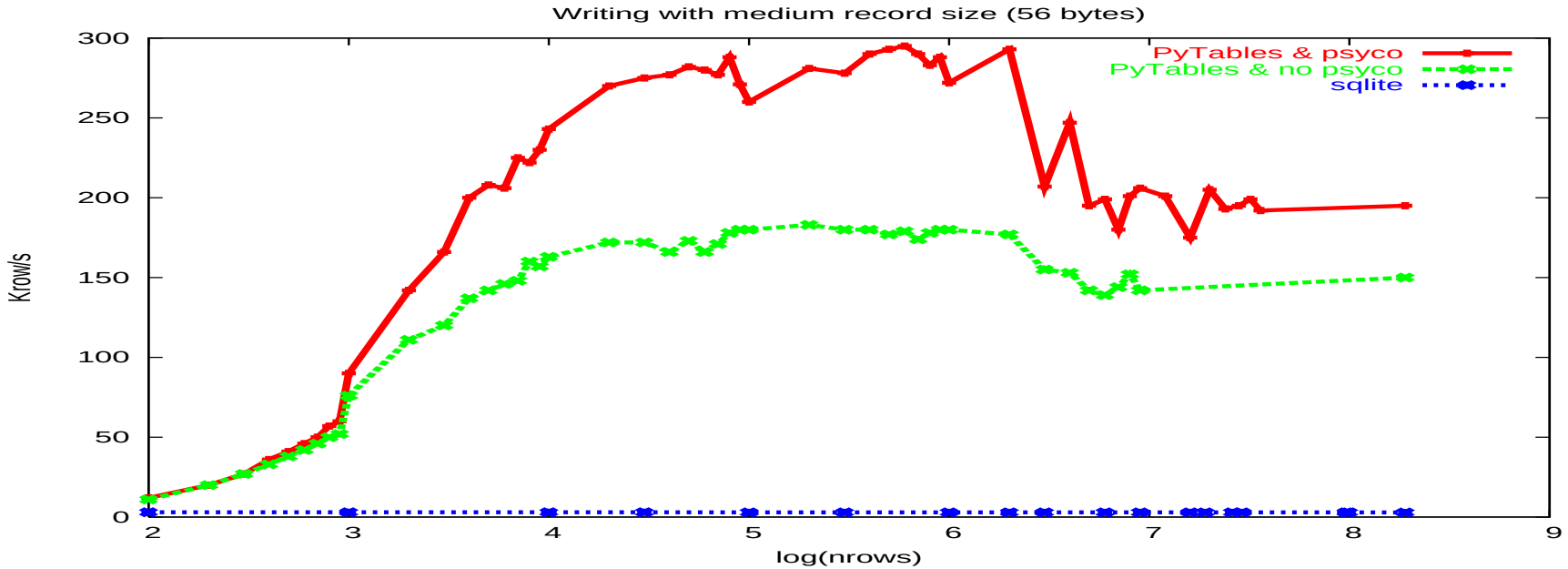
Comparación entre SQLite y PyTables

Package	Record length	Krows/s		MB/s		total Krows	file size	memory used	%CPU	
		write	read	write	read		(MB)	(MB)	write	read
SQLite (ic)	small	3.66	1030	0.18	51.0	300	15.0	5.0	100	100
SQLite (oc)	small	3.58	230	0.19	12.4	6000	322.0	5.0	100	25
SQLite (ic)	medium	2.90	857	0.27	80.0	300	28.0	5.0	100	100
SQLite (oc)	medium	2.90	120	0.30	13.1	3000	314.0	5.0	100	11
PyTables (ic)	small	679	797	10.6	12.5	3000	46.0	10.0	98	98
PyTables (oc)	small	576	590	9.0	9.2	30000	458.0	11.0	78	76
PyTables (ic)	medium	273	589	14.8	32.2	300	16.0	9.0	100	100
PyTables (oc)	medium	184	229	10.0	12.5	10000	534.0	17.0	56	40

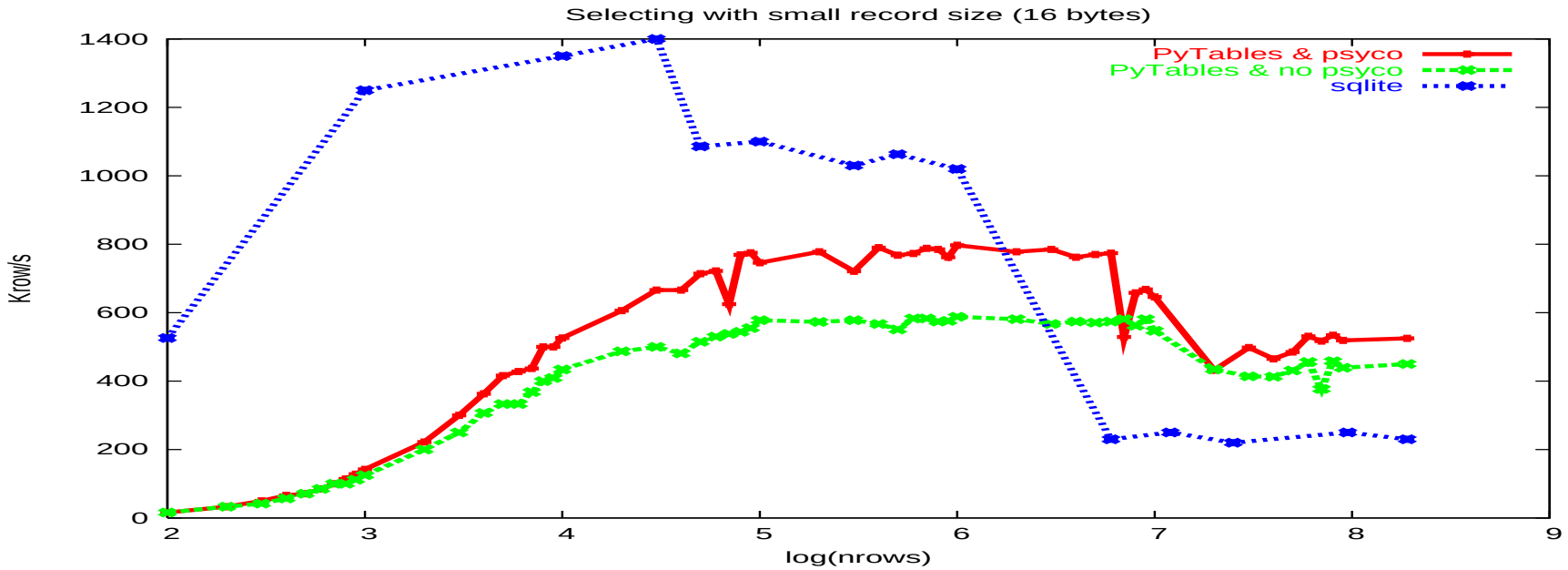
Table 2: Effect of different record sizes and table lengths in SQLite performance. (ic) means that the test ran in-core while (oc) means out-of-core.

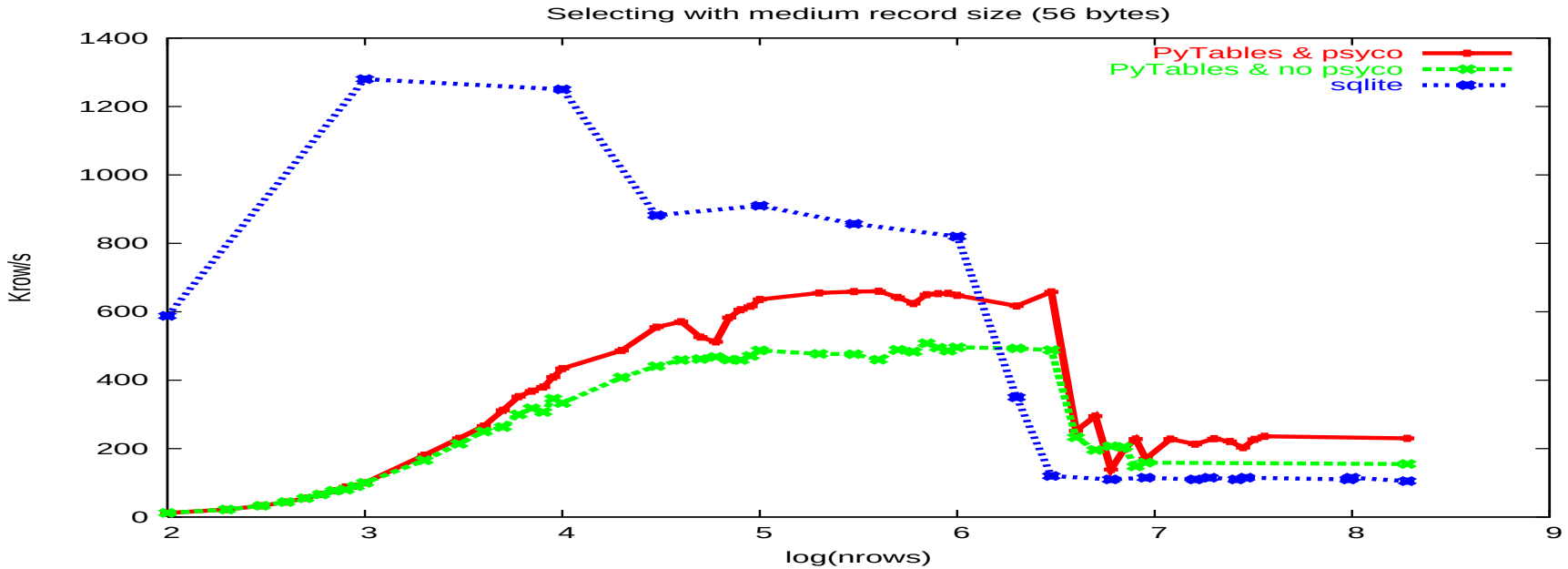
Comparación con SQLite (escritura)





Comparación con SQLite (lectura)





Conclusiones segundo benchmark (SQLite)

Escritura (SQLite no optimizado para los "inserts")

- PyTables es aprox. 100 veces más rápido que SQLite

Lectura (depende del tamaño de fila y si psyco esta activo o no)

- Caso de que el tamaño de fichero cabe en memoria
  - PyTables consigue entre un 60% y un 80% de la velocidad de SQLite
- Caso de que el tamaño de fichero no cabe en memoria
  - PyTables es entre 1.3 y 2 veces más rápido que SQLite

PyTables supera a SQLite cuando se procesan grandes cantidades de datos

(al tiempo que se mantiene bastante cerca en el otro caso)

Limitaciones actuales y planes de futuro

- Los elementos en las columnas no pueden tener más de una dimensión
- Los atributos en los nodos sólo soportan cadenas (pero tenemos cPickle!)
- Los arrays ilimitados no estan soportados (seguramente se implementarán en la versión 0.5)
- La compresión para arrays no está disponible
- Los objectos en el árbol no se pueden relacionar con otros objectos

Notas finales

- PyTables permite procesar datos de manera interactiva y rápida
- Con grandes cantidades de datos, PyTables demuestra que un leguaje interpretado como Python és suficiente para obtener má ximas prestaciones: PyTables (+ Psyco) sólo estan limitados por la velocidad de I/O a disco
- PyTables se ha diseñado para ser muy rápido durante la recuperación y selección de datos. Sin embargo, ha resultado ser también muy eficiente durante el proceso de escritura.

¡PyTables está preparado para el trabajo duro!

- En la versión 0.4 se han incorporado más de 200 tests que chequean virtualmente casi todas sus características
- PyTables ya está en beta y su API es estable
- Completa documentación tanto on-line como en un documento PDF de 40 páginas. Sin embargo, está en inglés.
- Podeis descargar la última versión (0.4, liberada el pasado marzo) y usarla de manera gratuita (licencia BSD) de:

http://pytables.sourceforge.net