PyTables

An on-disk binary data container

Francesc Alted

May 9th 2012, Austin Python meetup

Overview

- What PyTables is?
- Data structures in PyTables
- The one million song dataset
- Advanced capabilities in PyTables

What it is

- A binary data container for on-disk, structured data
- Based on the standard de-facto HDF5 format
- Free software (BSD license)
- Distinctive capabilities:
  - NumPy way to select data
  - Data can be compressed using many different compressors (and filters)
  - Out-of-core calculations
  - Powerful search in Table objects (including column indexing)

What it is not

- Not a relational database replacement
- Not a distributed database
- Not extremely secure or safe
- Not a mere HDF5 wrapper

Design goals

- Allow to structure your data in a **hierarchical** way.
- **Easy to use**. It implements the Natural Naming scheme for allowing convenient access to the data.
- **All the cells** in datasets can be **multidimensional** entities.
- Most of the I/O operations **speed** should be **only limited by the underlying I/O subsystem**, be it disk or memory.
- Enable the end user to save and deal with large datasets with **minimum overhead**, i.e. each single byte of data on disk has to be represented by one byte plus a small fraction when loaded into memory.

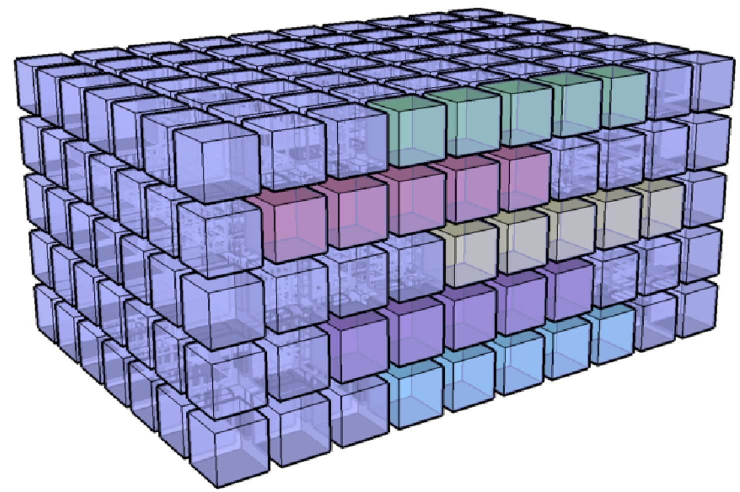
About HDF5 (Hierarchical Data File version 5)

- A **versatile data model** that can represent very complex data objects and a wide variety of metadata.
- A completely portable file format with **no limit** on the number or size of data objects in the collection.
- Implements a high-level API with C, C++, Fortran 90, and Java interfaces.
- A rich set of integrated **performance features** that allow for **access time and storage space optimizations**.
- Free software (BSD, MIT kind of license).

LEVERAGING NUMPY

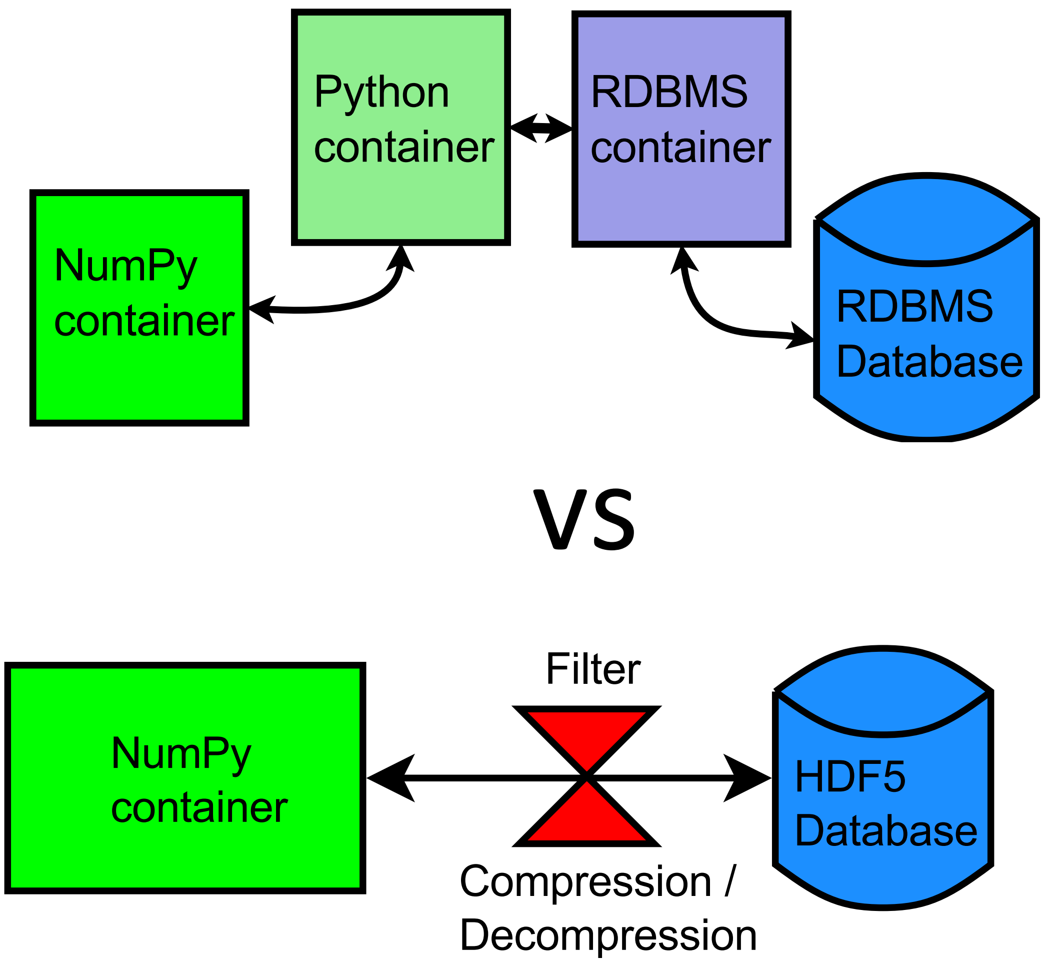
Easing disk access via NumPy paradigm

- Retrieving a data set portion
  - array[1]
  - array[2:3,2:100:2, ..., :10]
  - array[[3,10,30,1000]]
  - array[array2 > 0]
- Out of core operations
  - (array1**3 / array2) - sin(array3)



You don't need to learn other paradigms!

Using NumPy as memory container

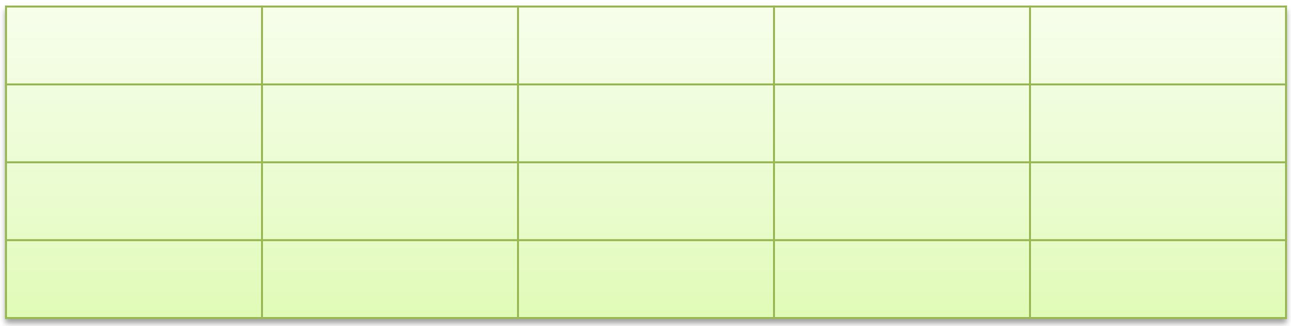


DATA STRUCTURES

Data structures

- High level of flexibility for structuring your data:
  - Datatypes: scalars (numerical & strings), records, enumerated, time…
  - Tables support multidimensional cells and nested records
  - Mutidimensional arrays
  - Variable length arrays

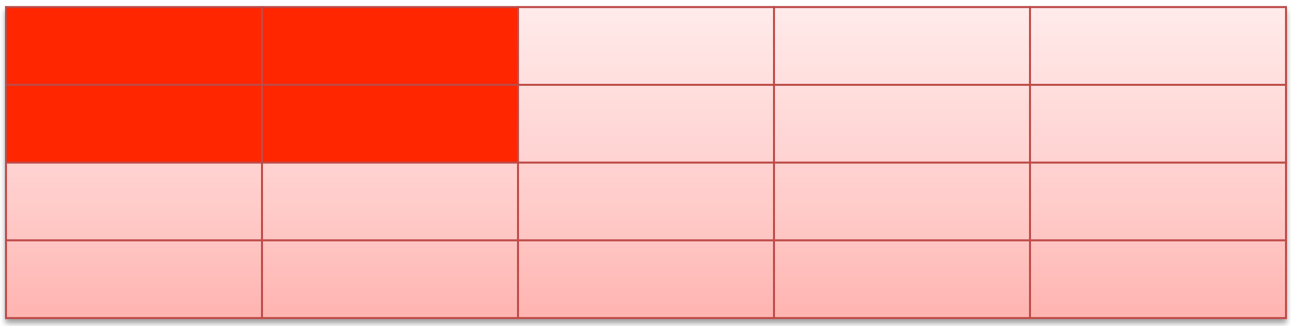
The Array object



- Easy to create:
  - file.createArray(mygroup, ‘array’, numpy_arr)

- Shape cannot change
- Cannot be compressed

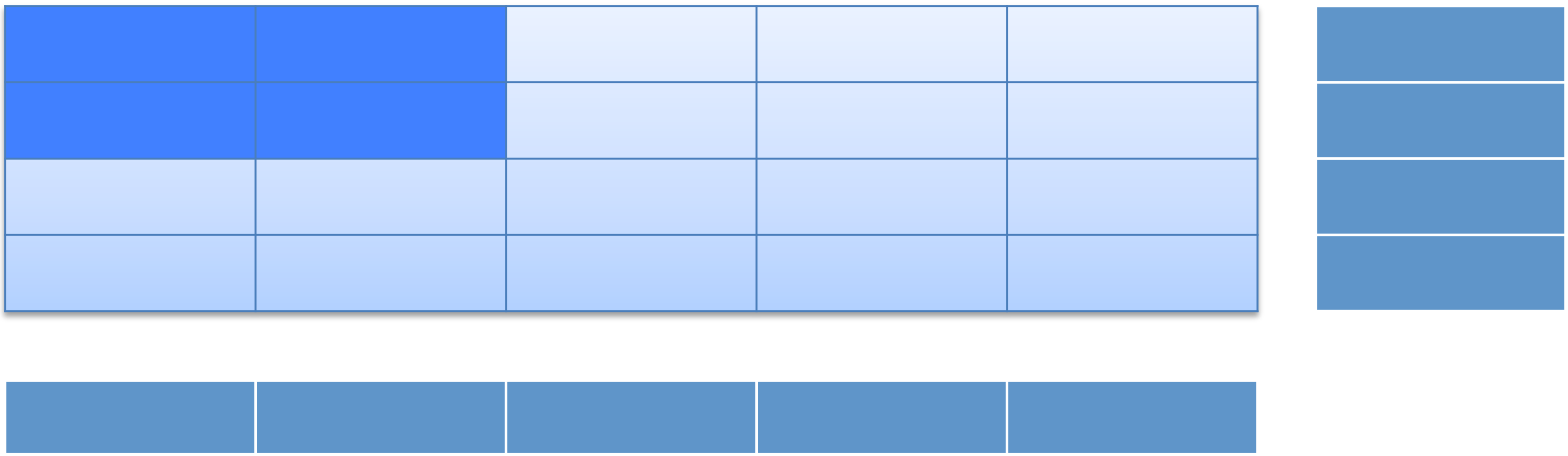
The CArray object



- Data is stored in chunks
- Each chunk can be compressed independently

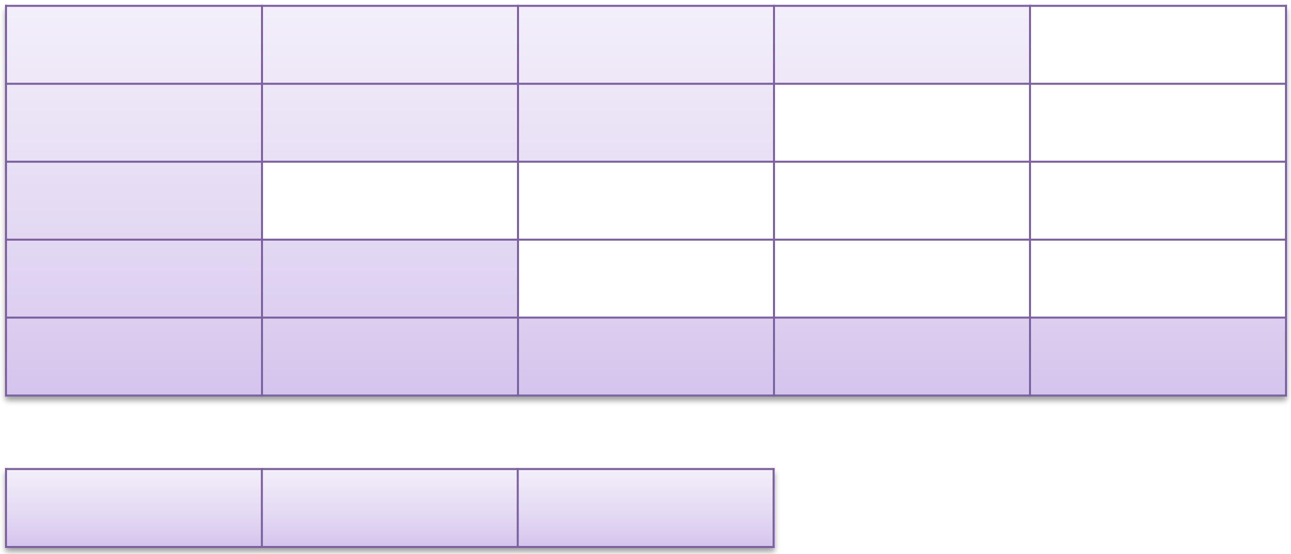
- Shape cannot change

The EArray object



- Data is stored in chunks
- Can be compressed
- Shape can change (either enlarged or shrunk)
- Shape must be kept regular

The VLArray object



- Data is stored in variable length rows
- Can be enlarged or shrunk

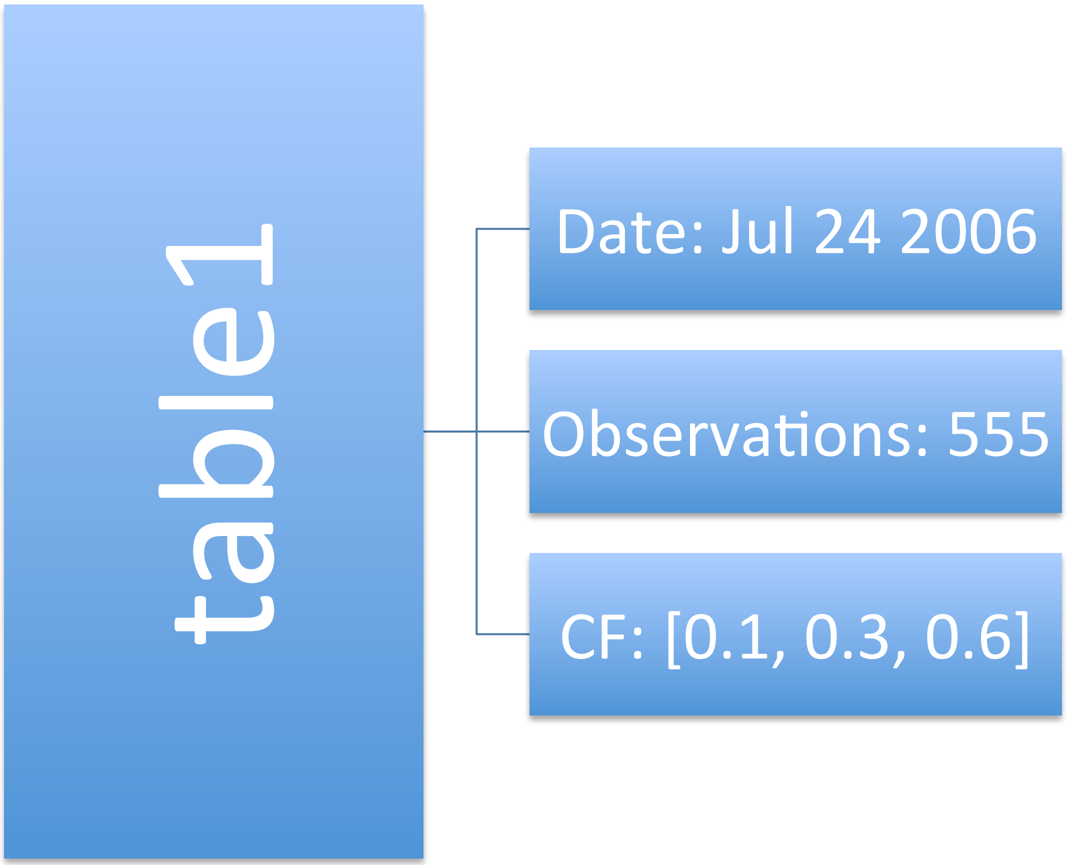
- Data cannot be compressed

The Table object

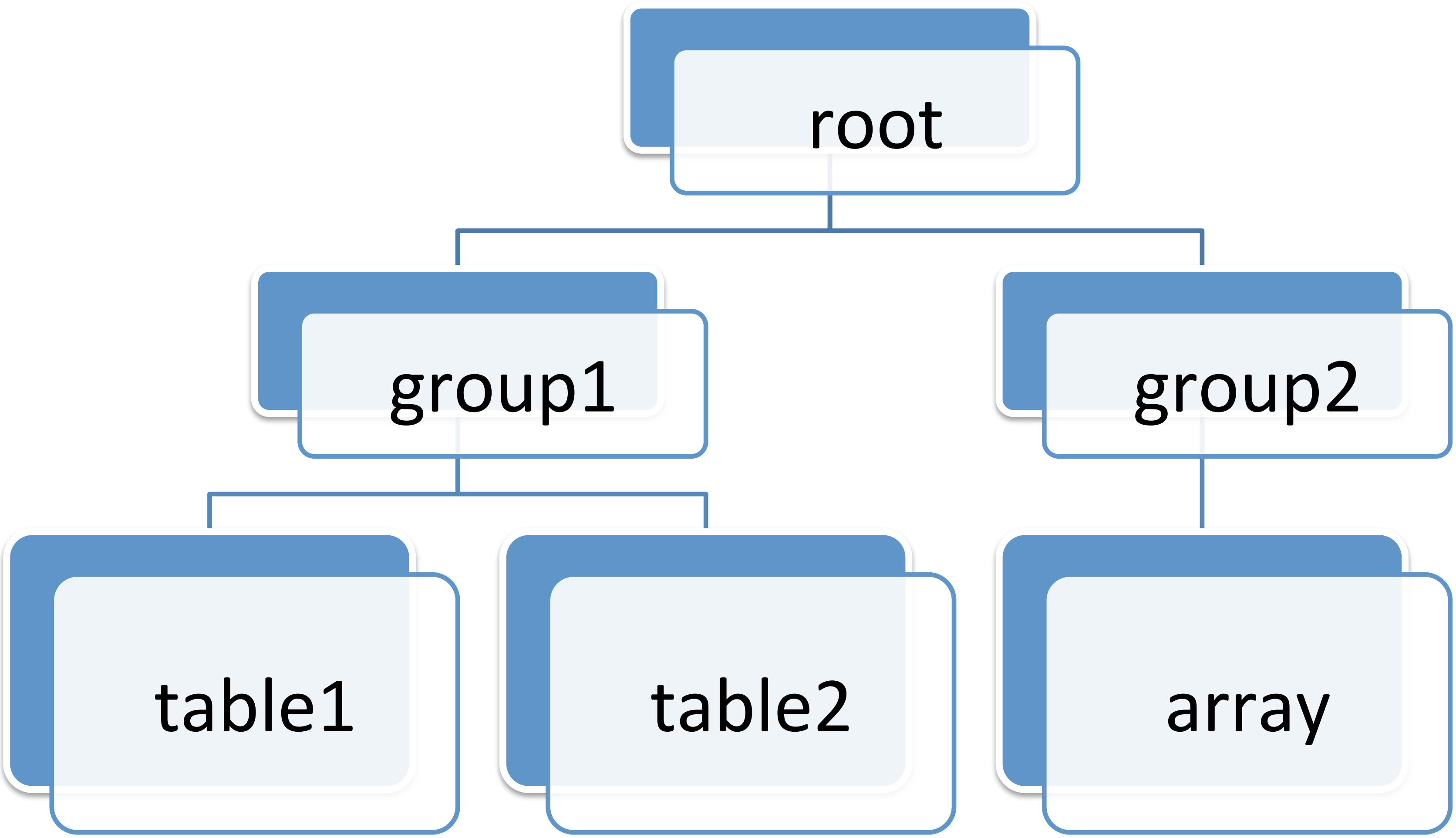
Col1 (int32)	Col2 (string 10)	Col3 (bool)	Col4 (complex64)	Col5 (float32)

- Data is stored in chunks
- Can be compressed
- Can be enlarged or shrunk
- Fields cannot be of variable length

Attributes: Metadata about data



Dataset hierarchy



INTERACTIVE SESSION

The 1 million song dataset

- The **Million Song Dataset** is a freely-available collection of audio features and metadata for a million contemporary popular music tracks
- 300 GB !
- Created using PyTables

http://labrosa.ee.columbia.edu/millionsong/

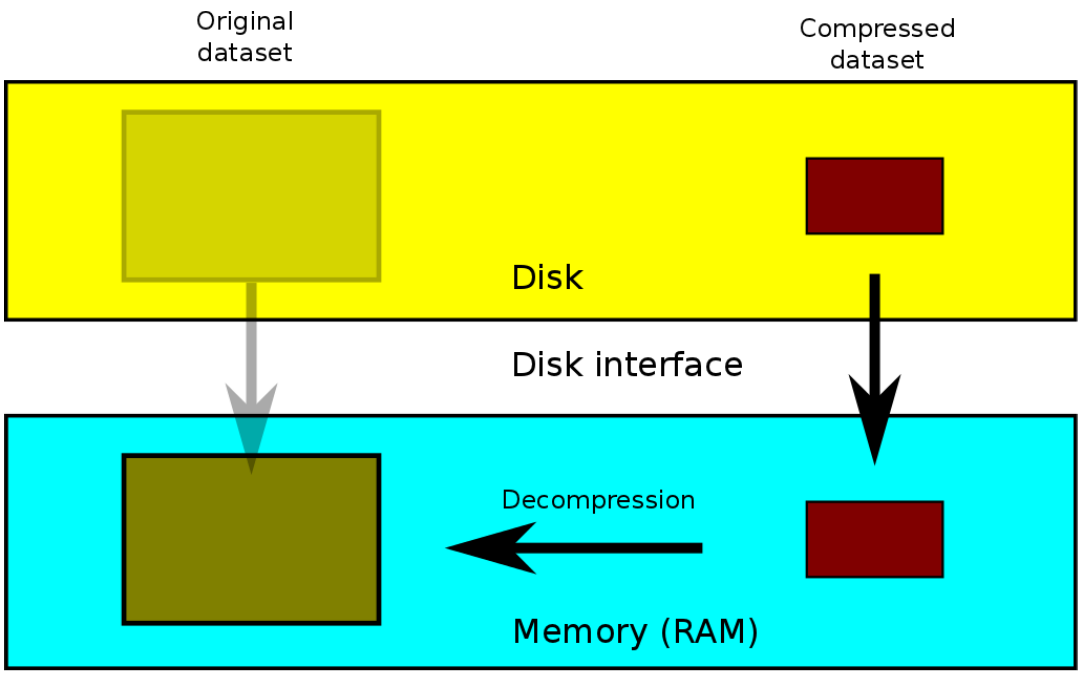
PyTables distinctive features

- Supports a range of compressors: zlib, bzip2, lzo and blosc
- Can do out-of-core operations
- Powerful search capabilities for Table objects, including column indexing

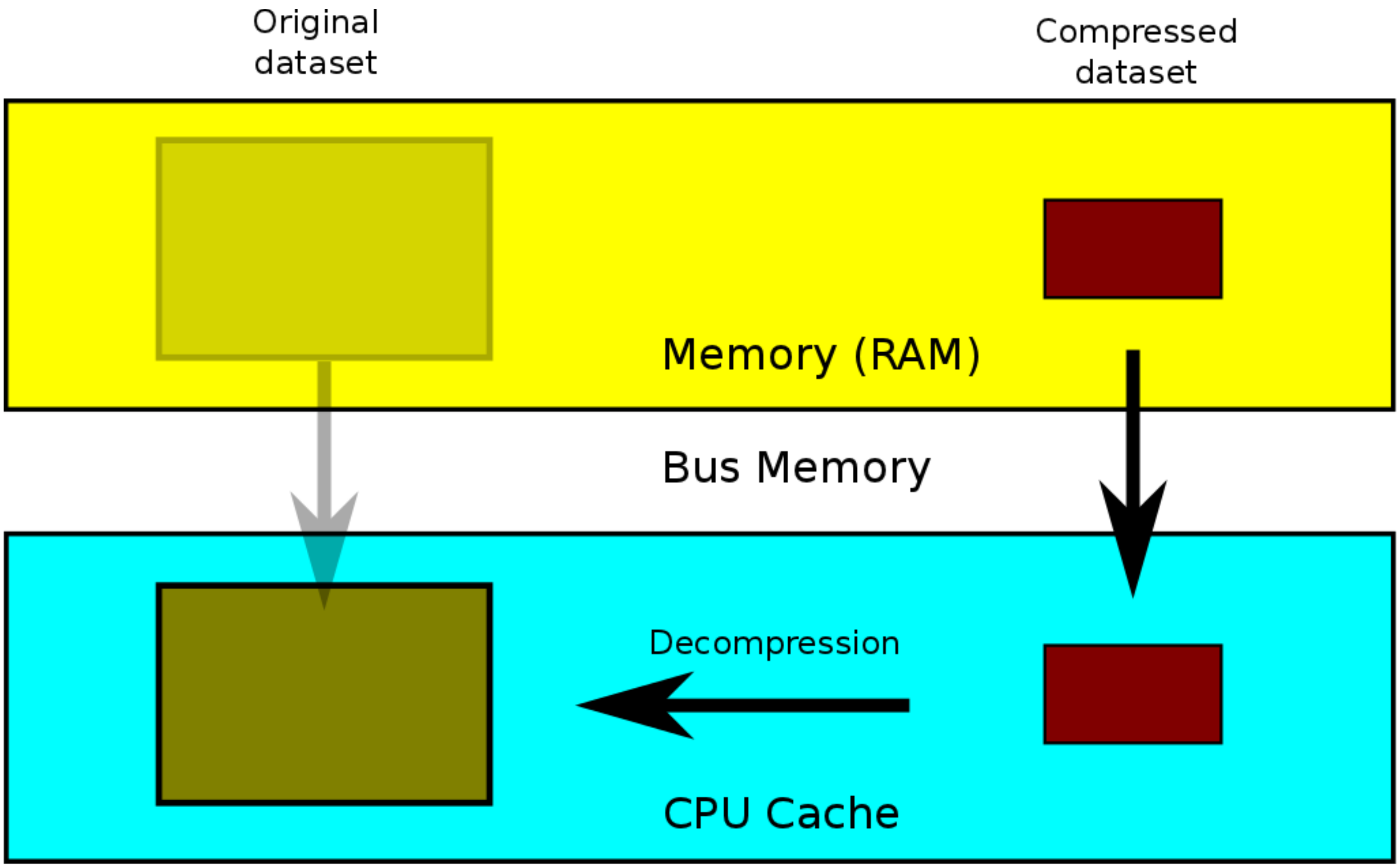
COMPRESSION CAPABILITIES

Why compression?

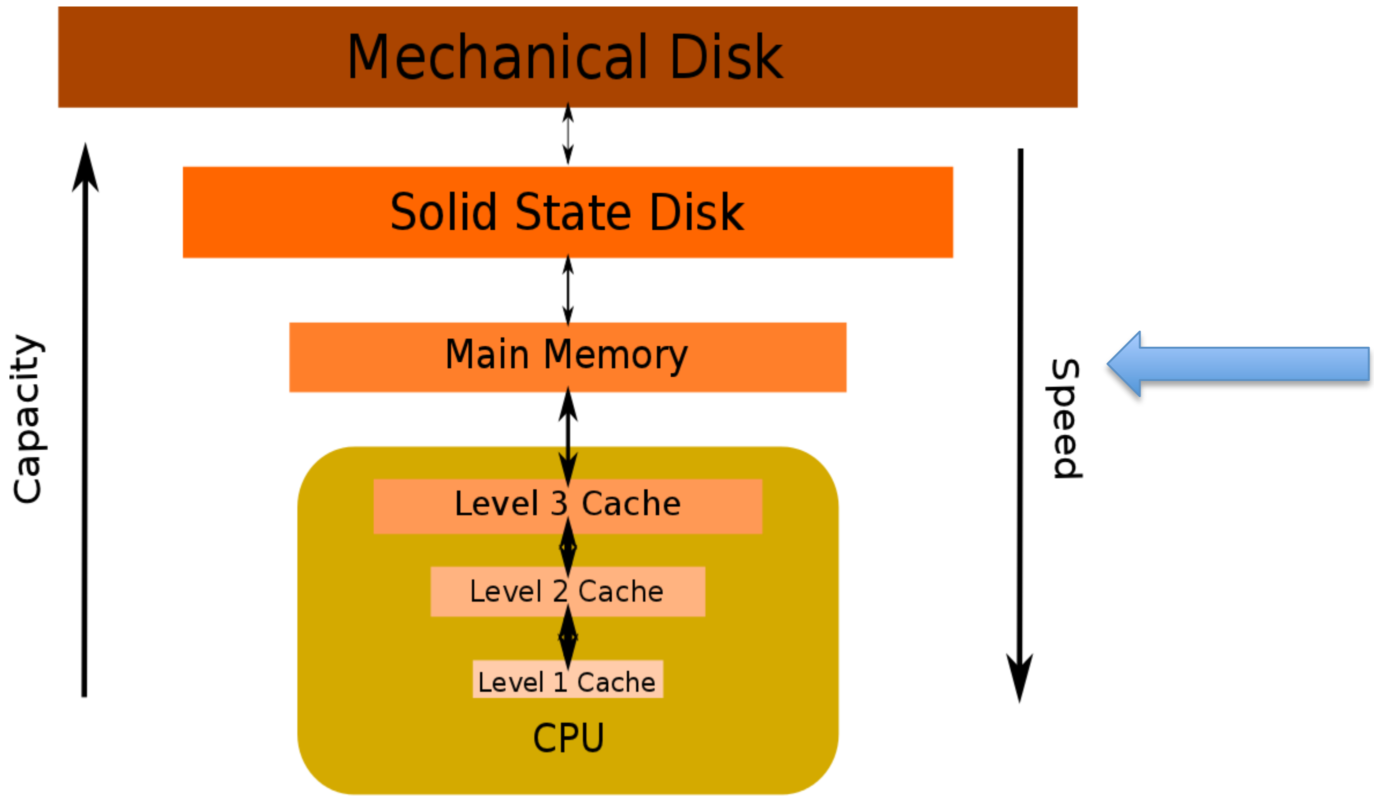
- Lets you store more data using the same space
- Uses more CPU, but CPU time is cheap compared with disk access
- Different compressors for different uses: bzip2, zlib, lzo, blosc



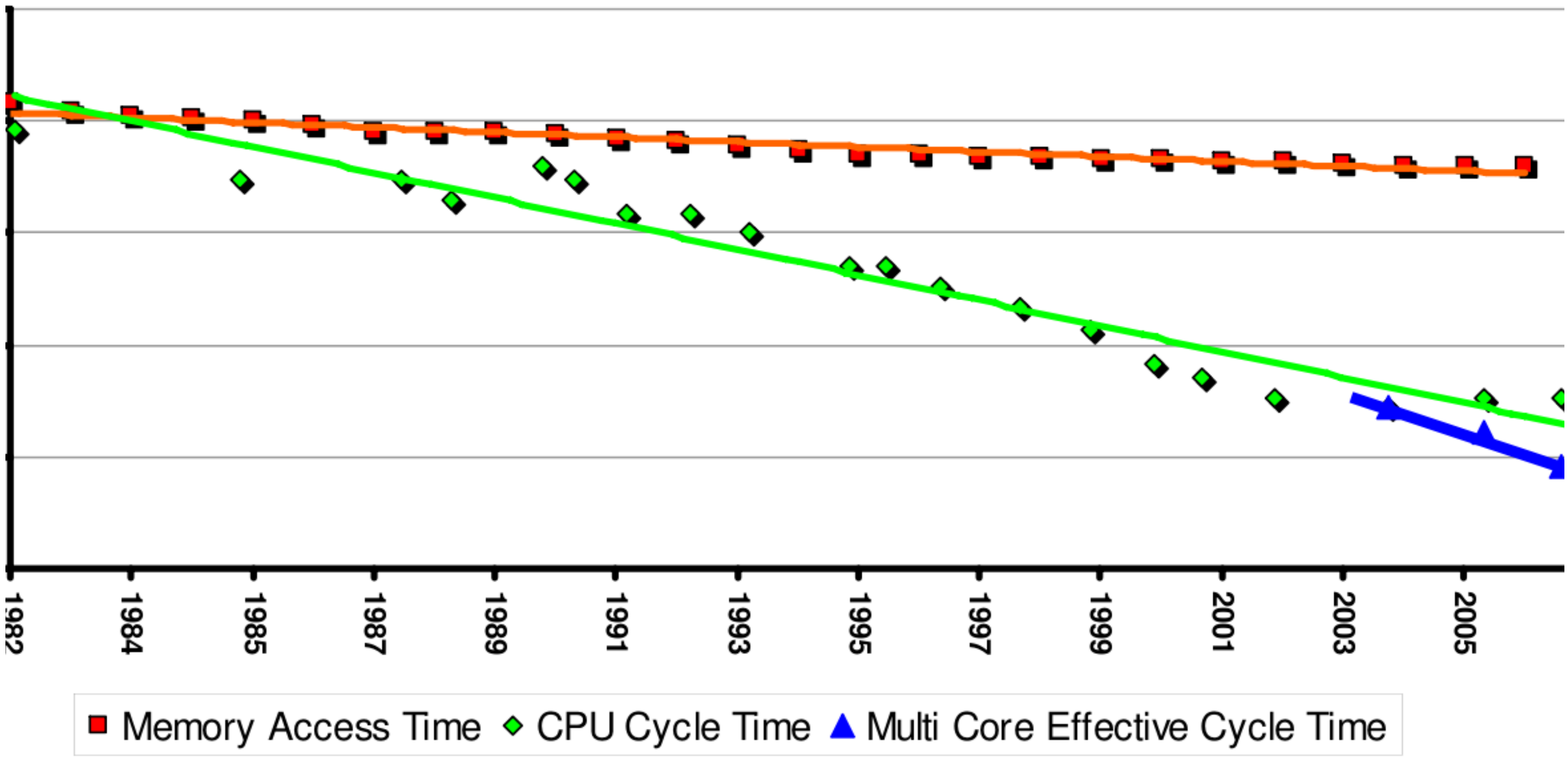
Why Blosc?


Original
dataset
Compressed
dataset
Memory (RAM)
Bus Memory
Decompression
CPU Cache

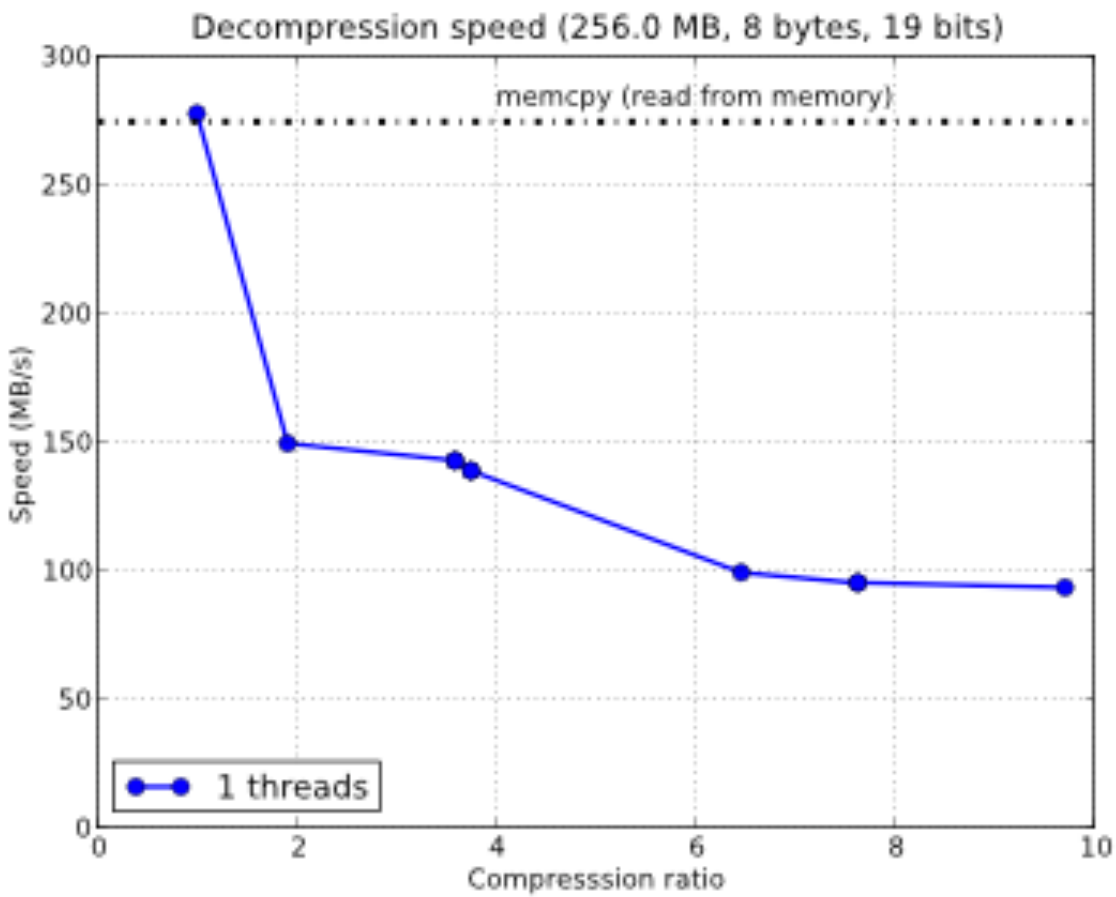
OS memory buffers



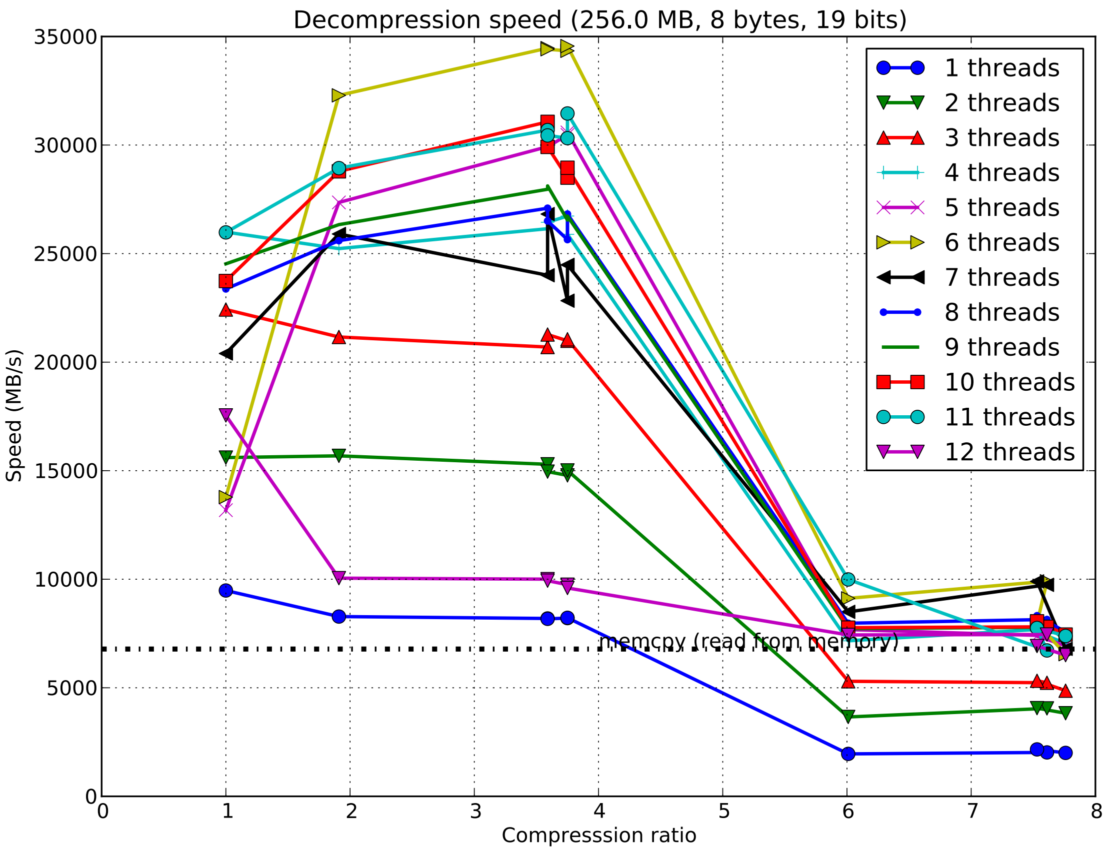
Memory access vs CPU cycle time



Laptop computer back in 2005



State of the art computer in 2012 (single node)

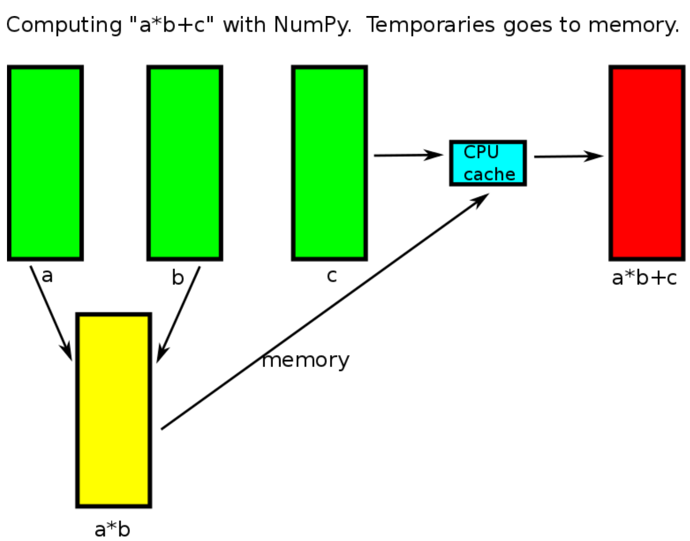


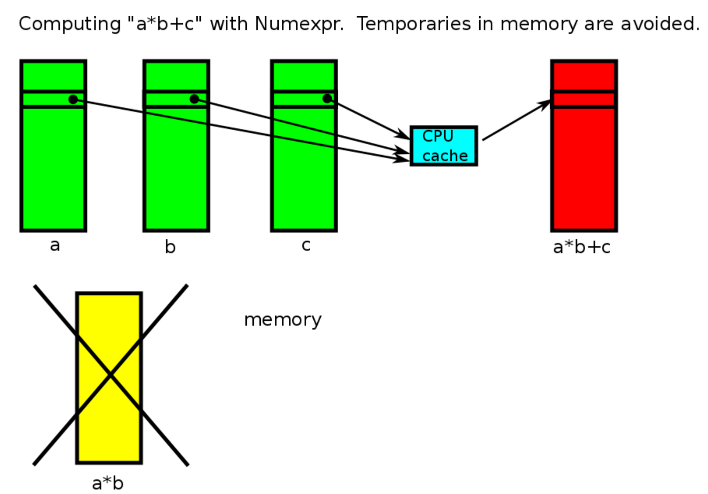
OUT-OF-CORE OPERATIONS

Operating with disk-based arrays

- tables.Expr is an optimized evaluator for expressions of disk-based arrays.
- It is a combination of the Numexpr advanced computing capabilities with the high I/O performance of PyTables.
- Similarly to Numexpr, disk-temporaries are avoided, and multi-threaded operation is preserved.

Avoiding temporaries with Numexpr





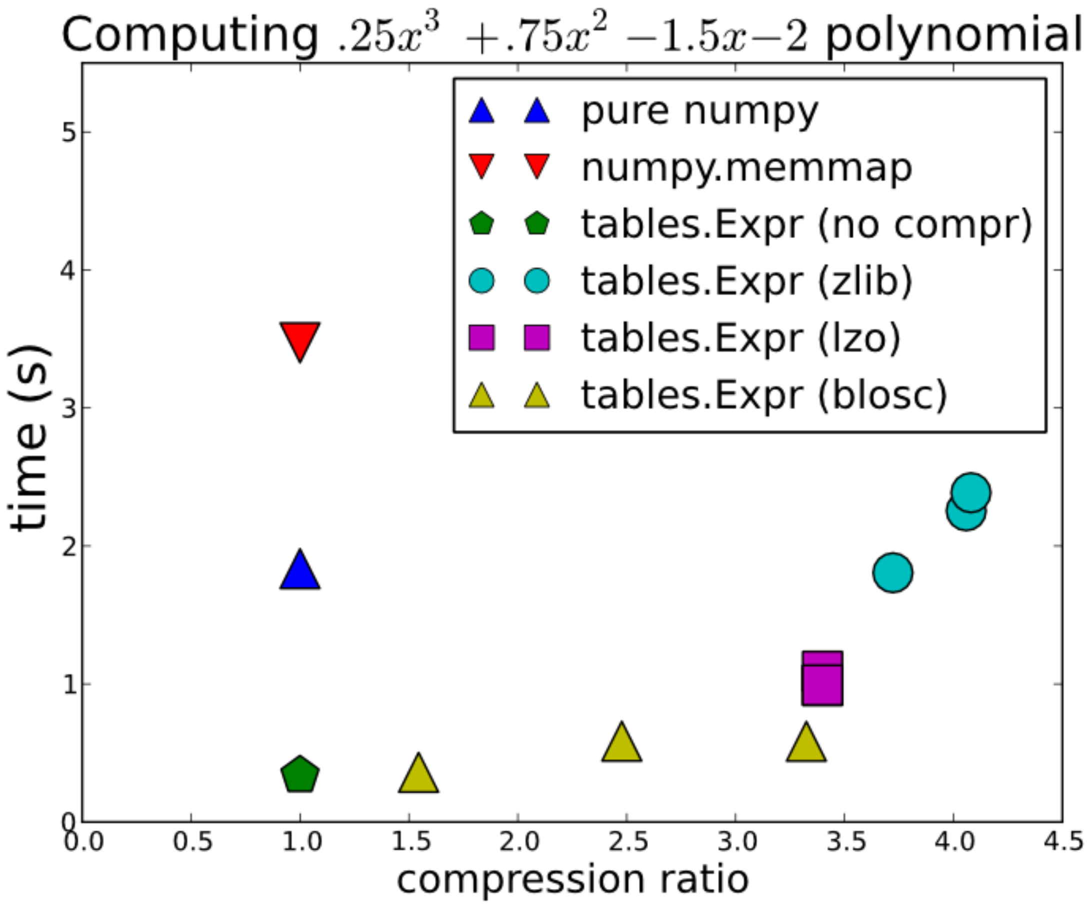
Tables.Expr follows the same approach, but with disk and memory instead

Tables.Expr in action

- Evaluating .25*x**3 + .75*x**2 - 1.5*x - 2

```
import tables as tb
f = tb.openFile(h5fname, "a")
x = f.root.x # get the x input
r = f.createCArray(f.root, "r", atom=x.atom, shape=x.shape)
ex = tb.Expr('.25*x**3 + .75*x**2 - 1.5*x - 2')
ex.setOutput(r) # output will got to the CArray on disk
ex.eval() # evaluate!
f.close()
```

Example of out-of-core operation



ADVANCED QUERY CAPABILITIES

Different query modes

Regular query:

- ```
  [ r['c1'] for r in table
              if r['c2'] > 2.1 and r['c3'] == True)) ]
  ```

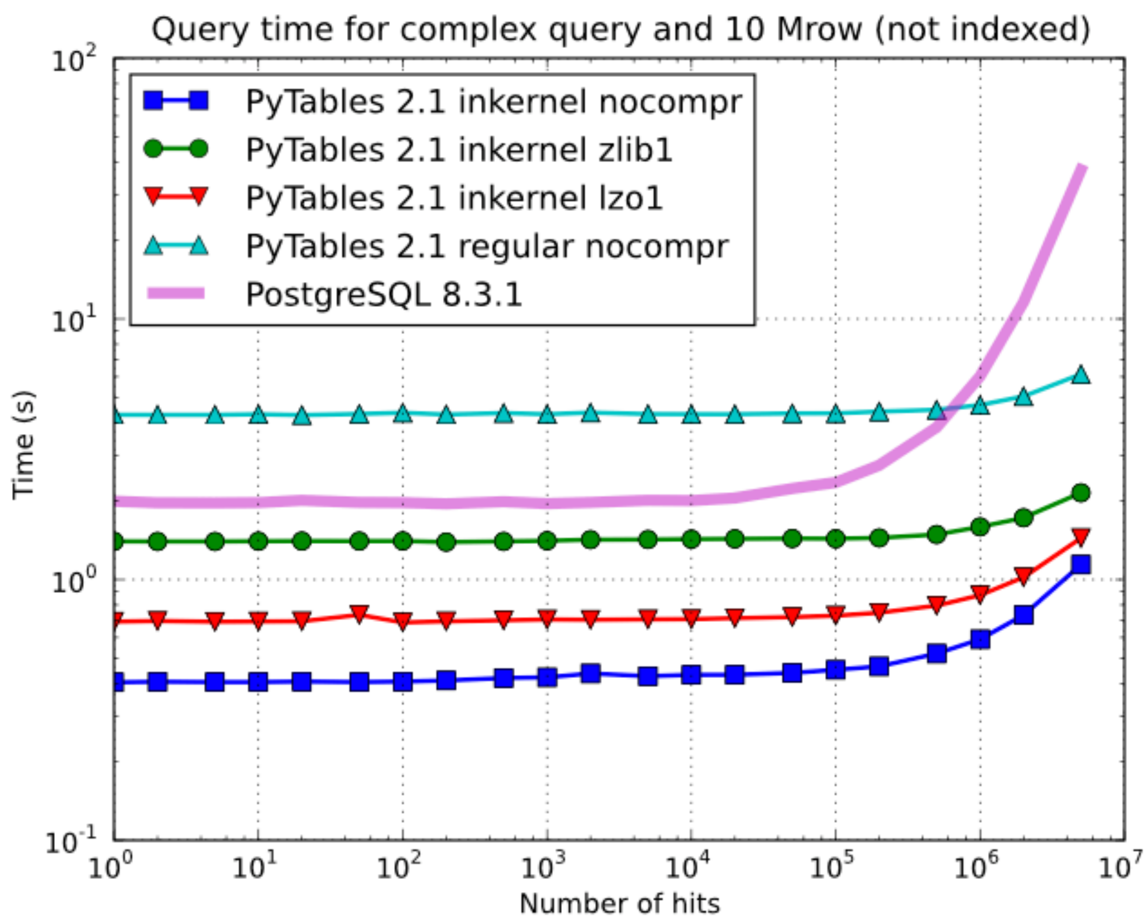
In-kernel query:

- ```
  [ r['c1'] for r in table.where('(c2>2.1)&(c3==True)') ]
  ```

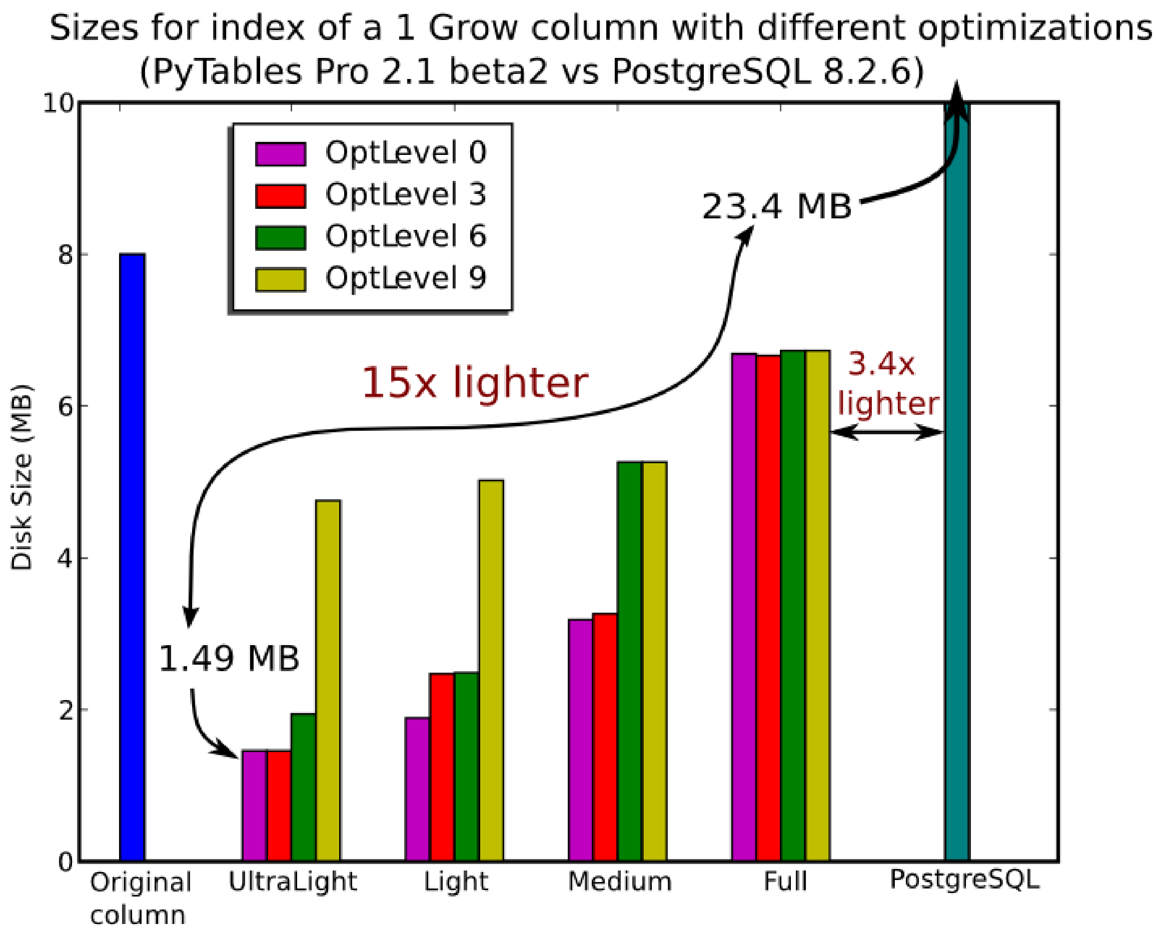
Indexed query:

- `table.cols.c2.createIndex()`
- `table.cols.c3.createIndex()`
- `[ r['c1'] for r in table.where('(c2>2.1)&(c3==True)') ]`

Regular and in-kernel queries



Customizable indexes



Indexed query performance