Out-of-Core Columnar Datasets

Introducing **bcolz**, an In-Memory/On-Disk Columnar, Chunked and Compressed Data Container

Francesc Alted <francesc@blosc.io>
Freelance Trainer And Consultant

EuroPython 2014, July 25, Berlin

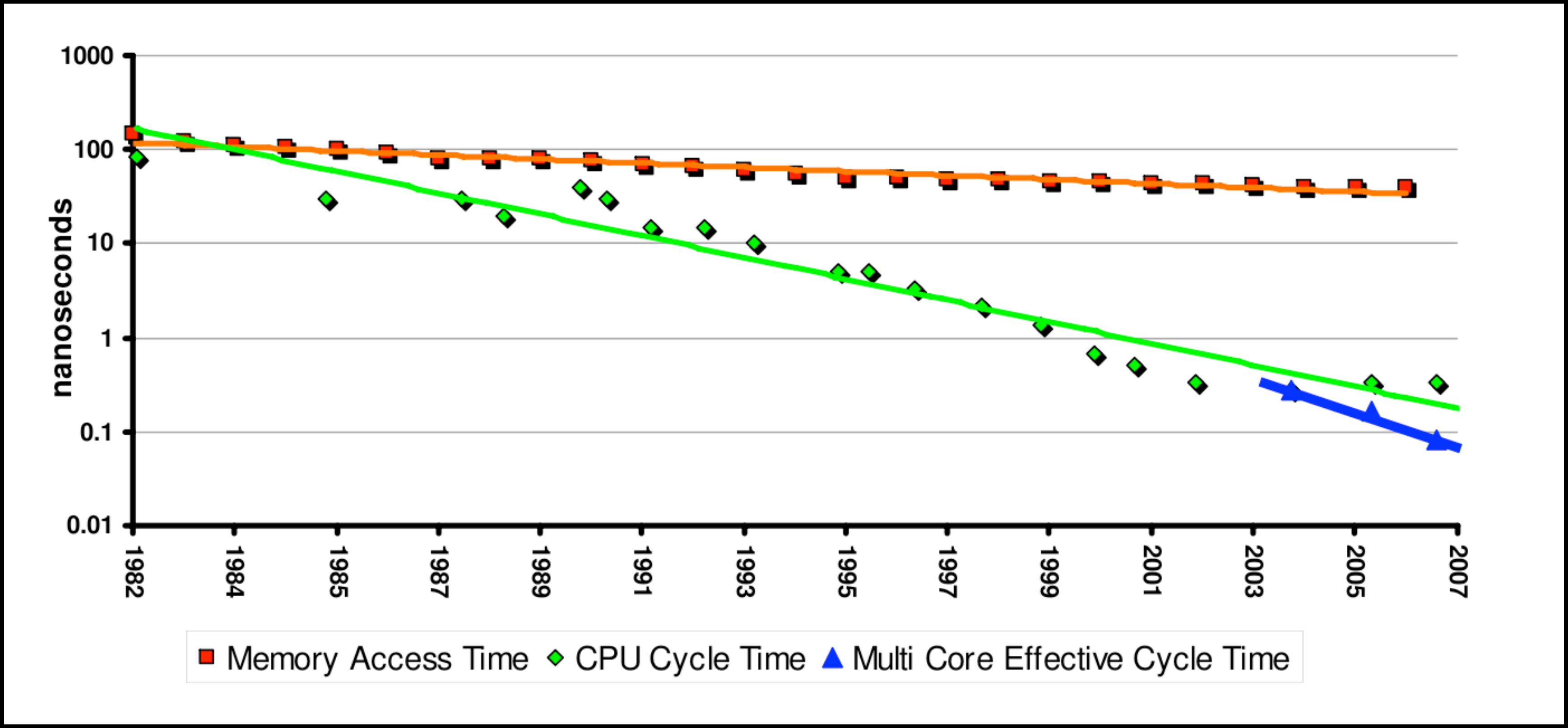
About Me

- I am the creator of tools like **PyTables**, **Blosc**, **bcolz**, and a long-term maintainer of **Numexpr**
- I am an experienced **developer and trainer** in:
  - Python (almost 15 years of experience)
  - High Performance Computing and Storage
- Also available for **consulting**

What? Yet Another Data Container?

- We are bound to live in a world of wildly different instances of data containers
- The NoSQL movement is an example of that
- Why? Mainly because the increasing gap between CPU and memory speeds

CPU vs Memory Speed



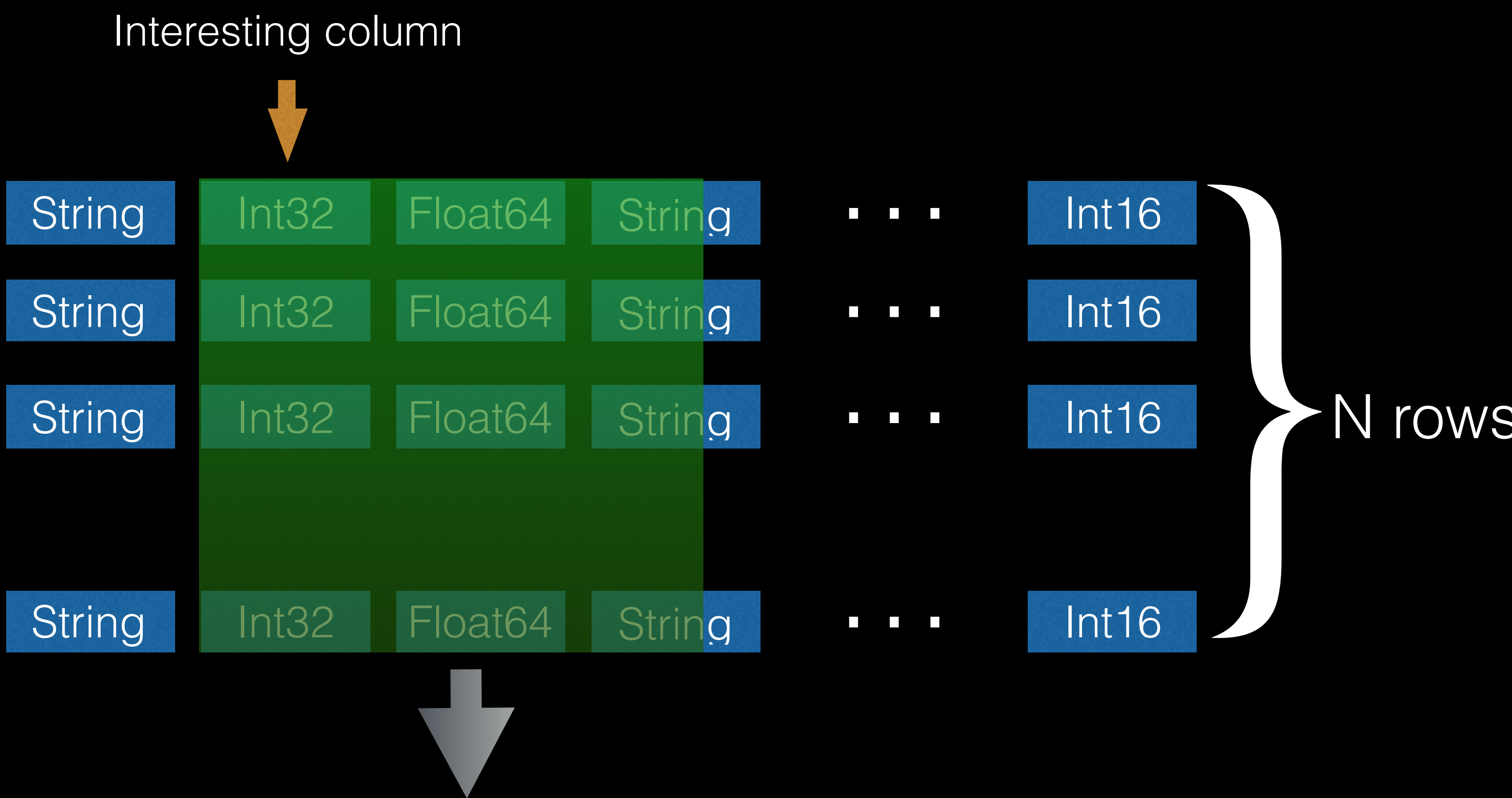
See my article:

"Why Modern CPUs Are Starving And What You Can Do About It"

Why Columnar?

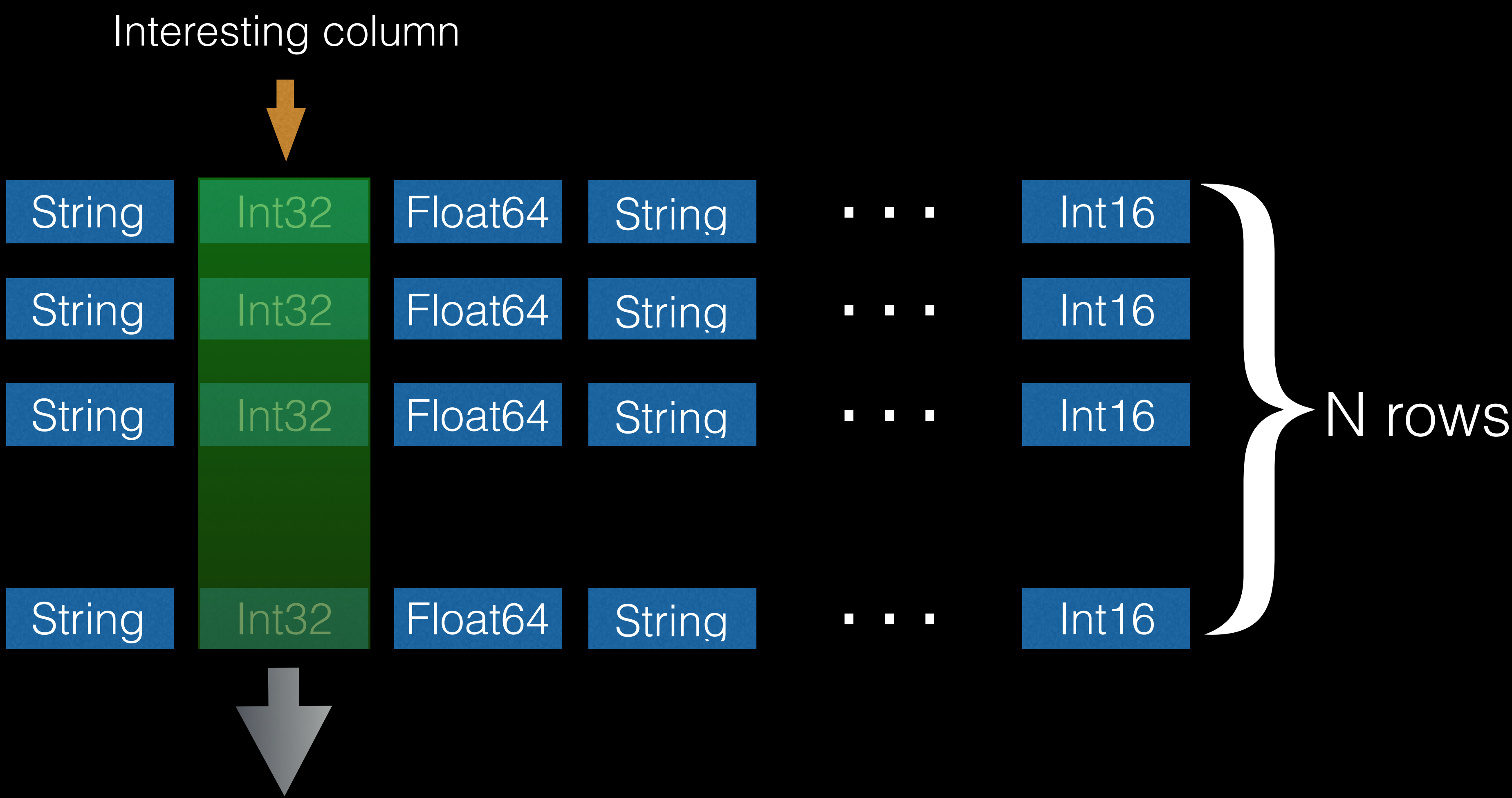
- When querying tabular data, only the interesting data is accessed
- Less I/O required

In-memory Row-Wise Table



Interesting Data: N * 4 bytes (Int32)
Actual Data Read: N * 64 bytes (cache line)

In-memory Column-Wise Table

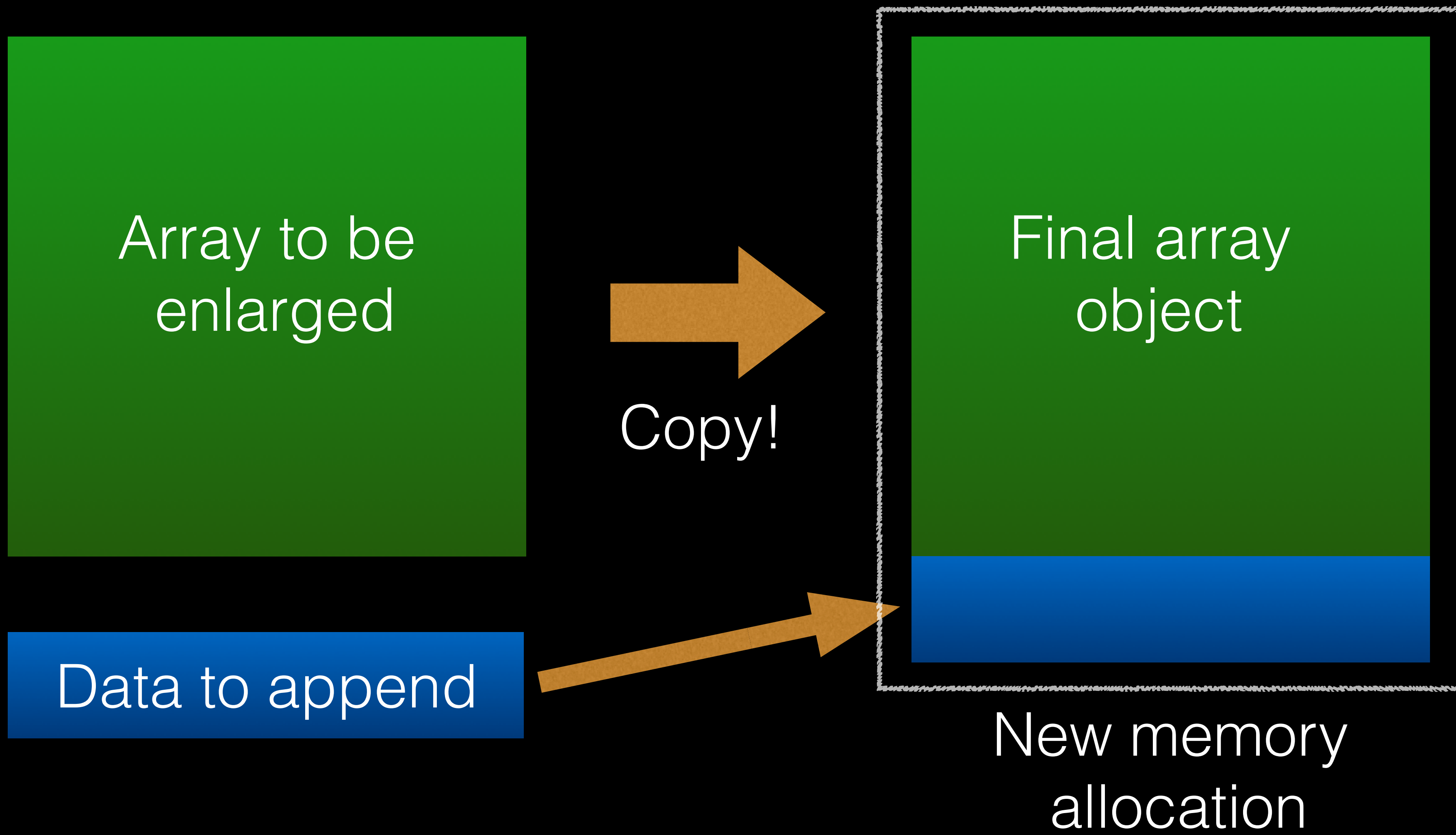


Interesting Data: N * 4 bytes (Int32)
Actual Data Read: N * 4 bytes (Int32)

Why Chunking?

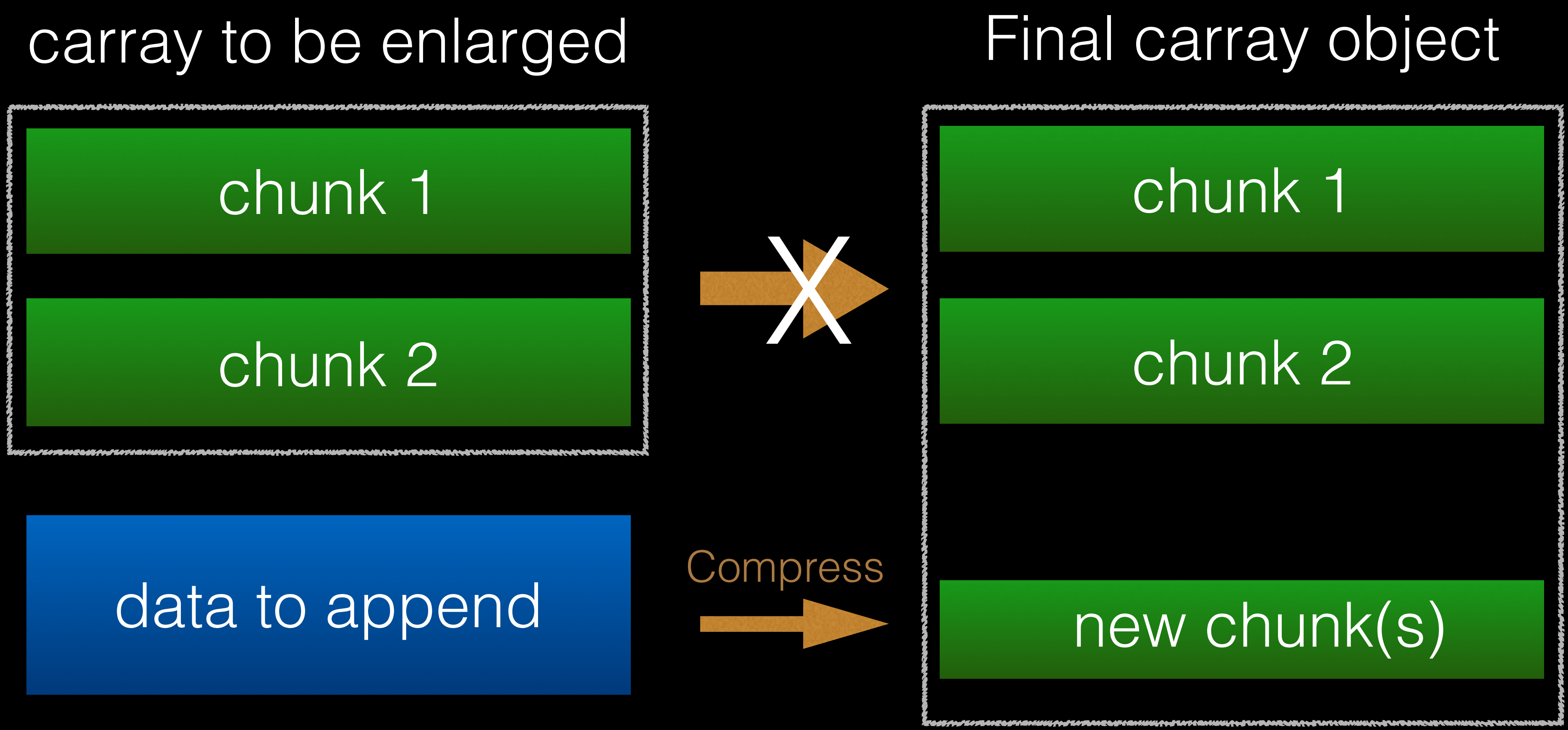
- Chunking means more difficulty handling data, so why bother?
  - Efficient enlarging and shrinking
  - On-flight compression possible

Appending Data in NumPy



- Both memory areas have to exist **simultaneously**

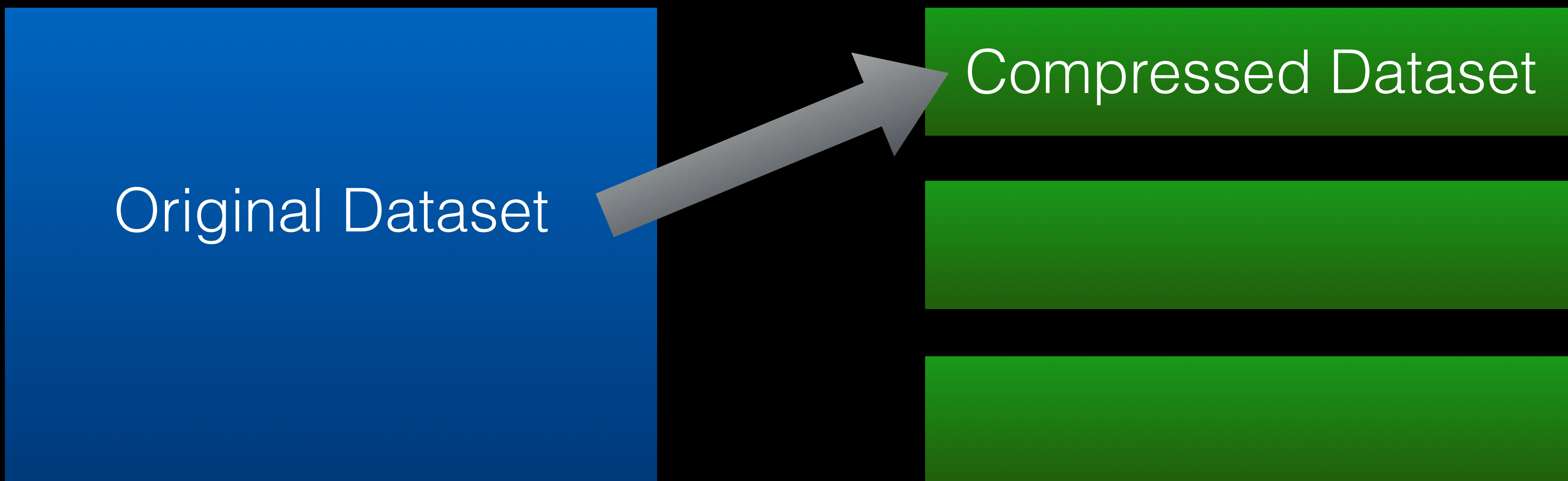
Appending Data in **bcolz**



- Only a compression operation on new data is required

Why Compression (I)?

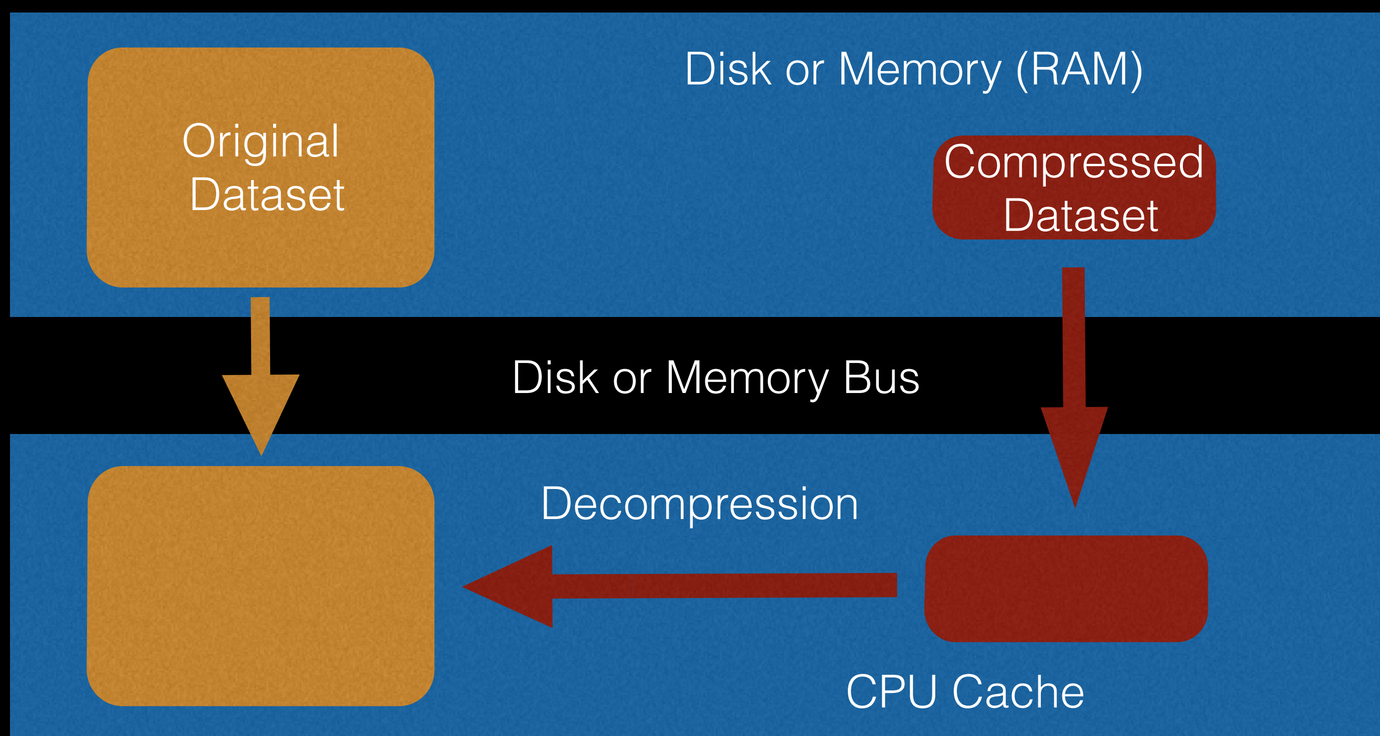
More data can be stored in the same amount of media



3x more data

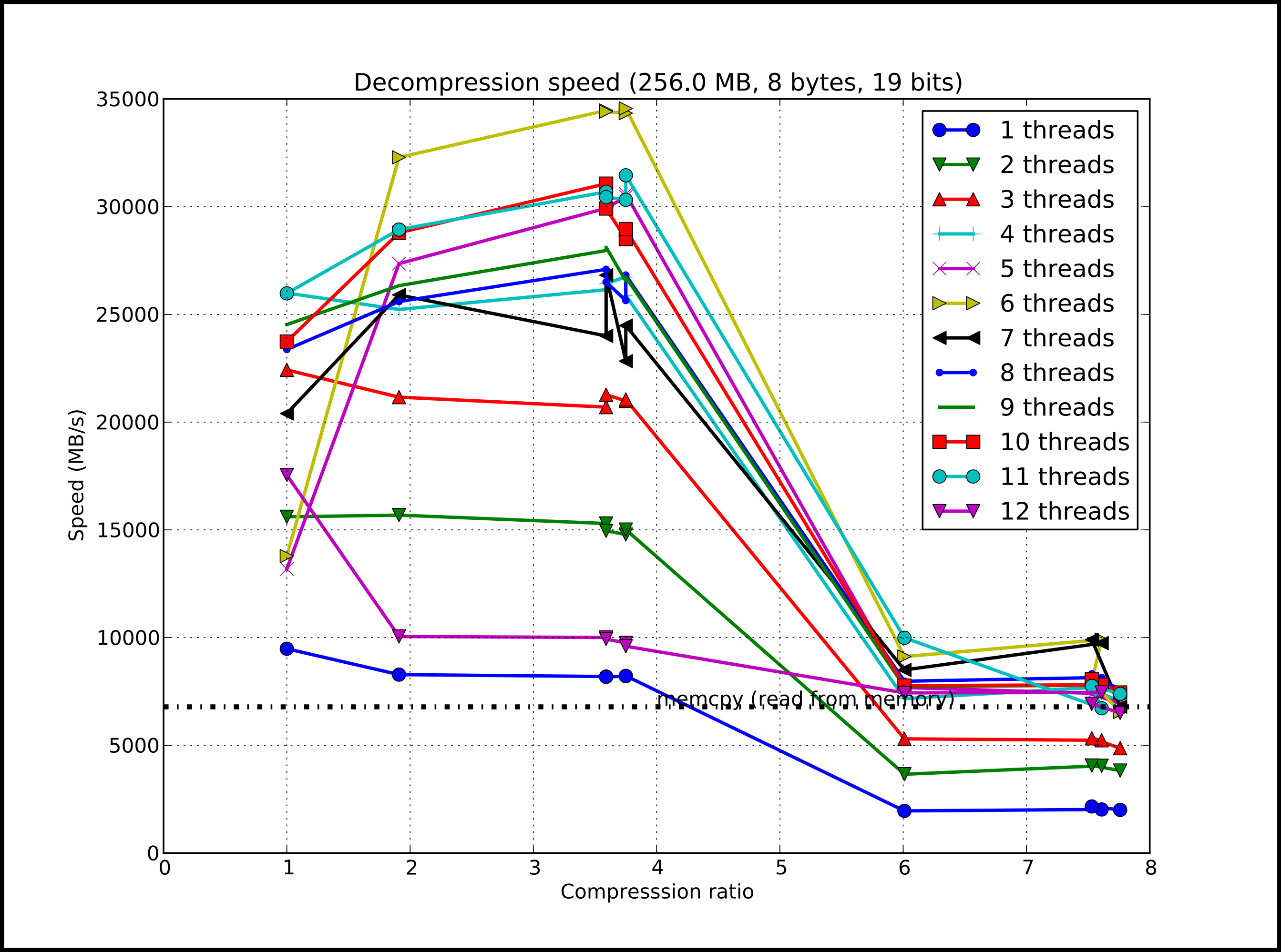
Why Compression (II)?

Less data needs to be transmitted to the CPU



Transmission + decompression faster than direct transfer?

Blosc: Compressing Faster Than Memory Speed



bcolz: Goals and Implementation

Feature inclusion driven by the:

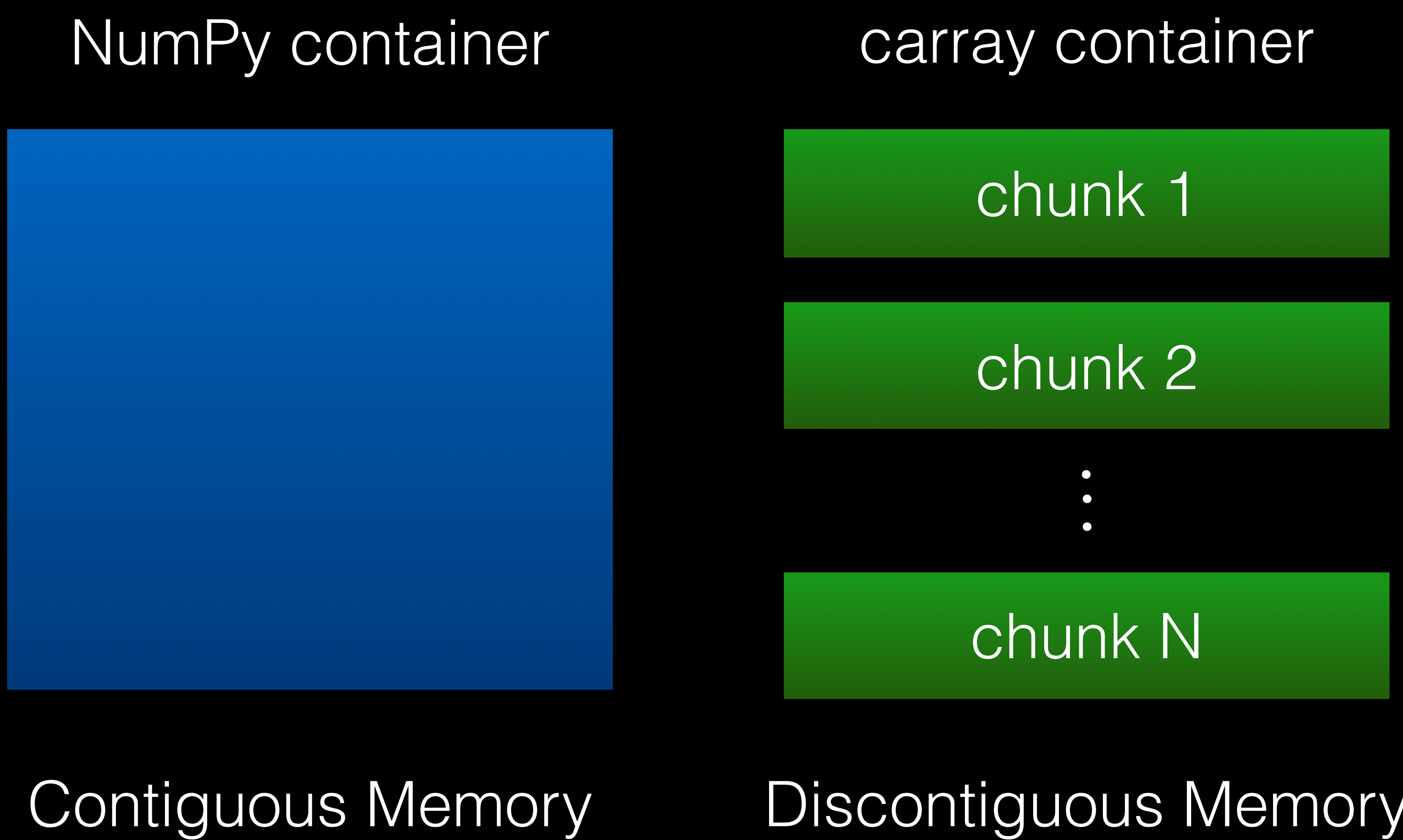
“Keep It Simple, Stupid”

–KISS Principle

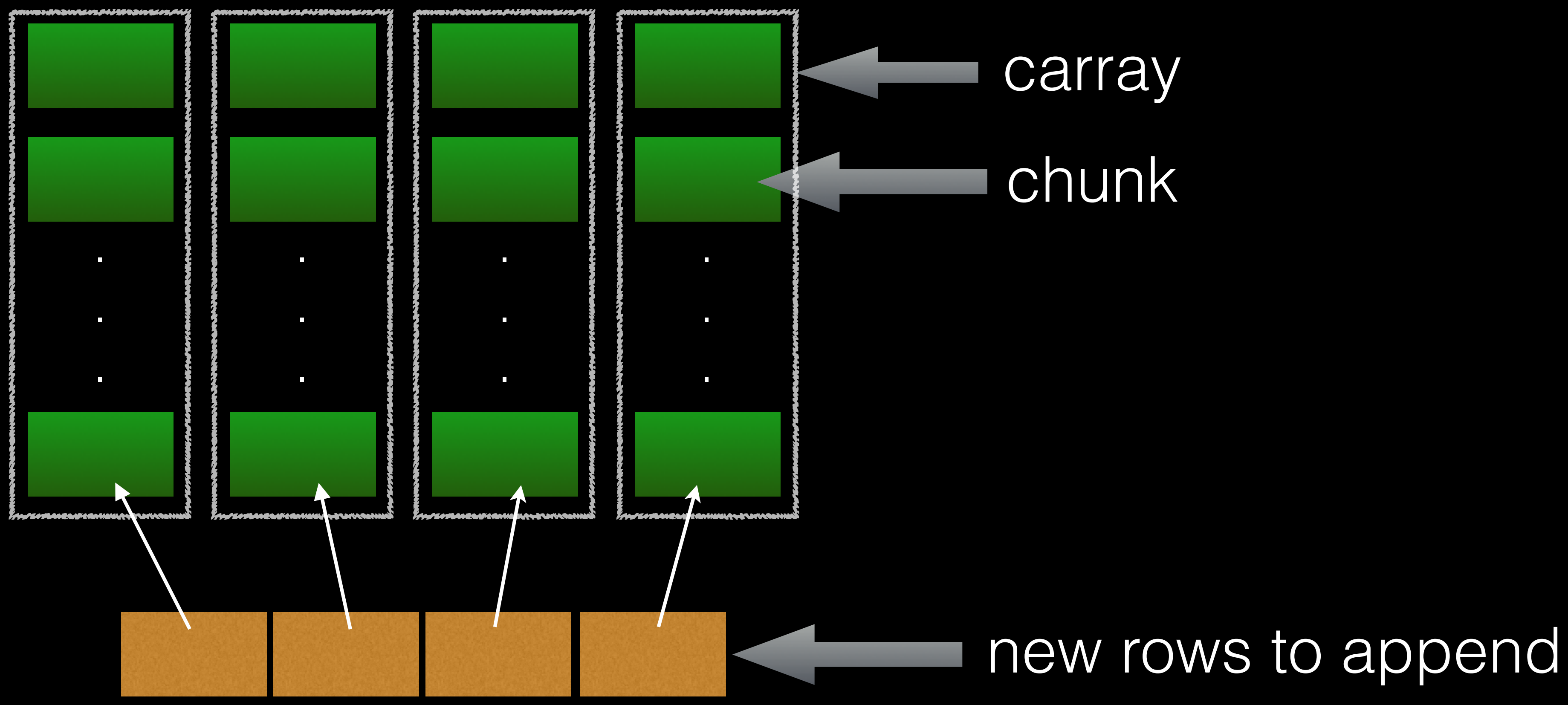
What **bcolz** Is?

- **Columnar, chunked, compressed** data containers for Python
- Offers `***carray***` and `***ctable***` container flavors
- Uses the powerful Blosc compression library for on-the-flight compression/decompression
- 100% written in Python/Cython

carray: Multidimensional Container for Homogeneous Data



The **ctable** Object



- Chunks follow column order
- Very efficient for querying
- Adding or removing columns is cheap too

Persistency

- carray and ctable objects can live **on disk**, not only in memory

- The format for persistency is heavily based in **bloscpack**, a nascent library for compressing large datasets

- bcolz allows every operation to be executed entirely **on-disk** (out-of-core operations)

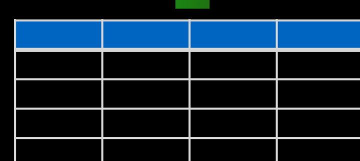
Streaming Analytics With **bcolz**

map(), filter(),
groupby(), sortby(),
reduceby(),
join()

itertools,
PyToolz,
CyToolz

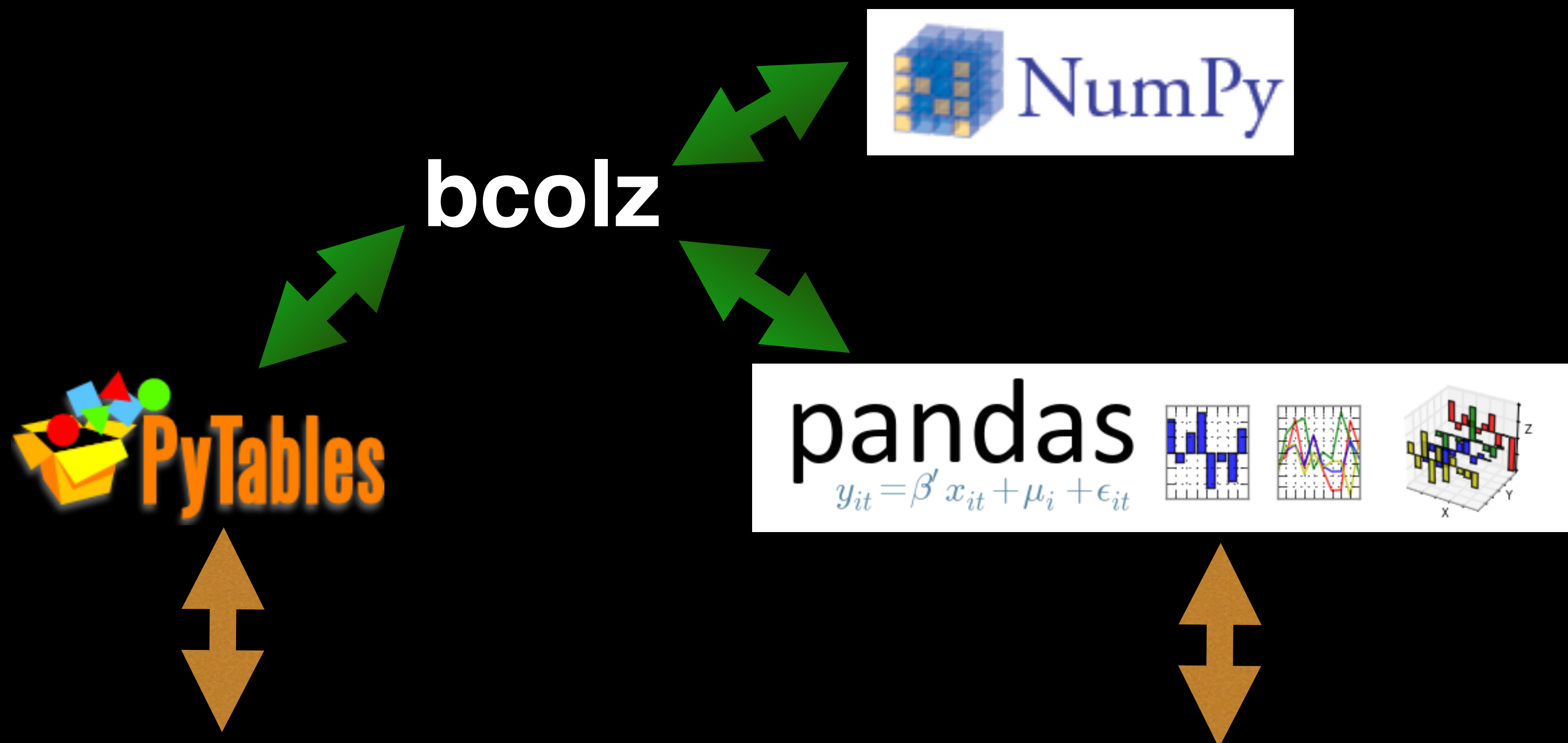
iter(), iterblocks(),
where(), whereblocks(),
__getitem__()

bcolz
iterators/filters
with blocking



bcolz container
(disk or memory)

Interacting with Neighbors



- HDF5 format
- Indexed queries
- Long term storage
- Blosc support

- Relational Databases
- CSV files
- HDF5/PyTables
- Excel

Some Benchmarks With Real Data: The MovieLens Dataset

Materials in:
https://github.com/Blosc/movielens-bench

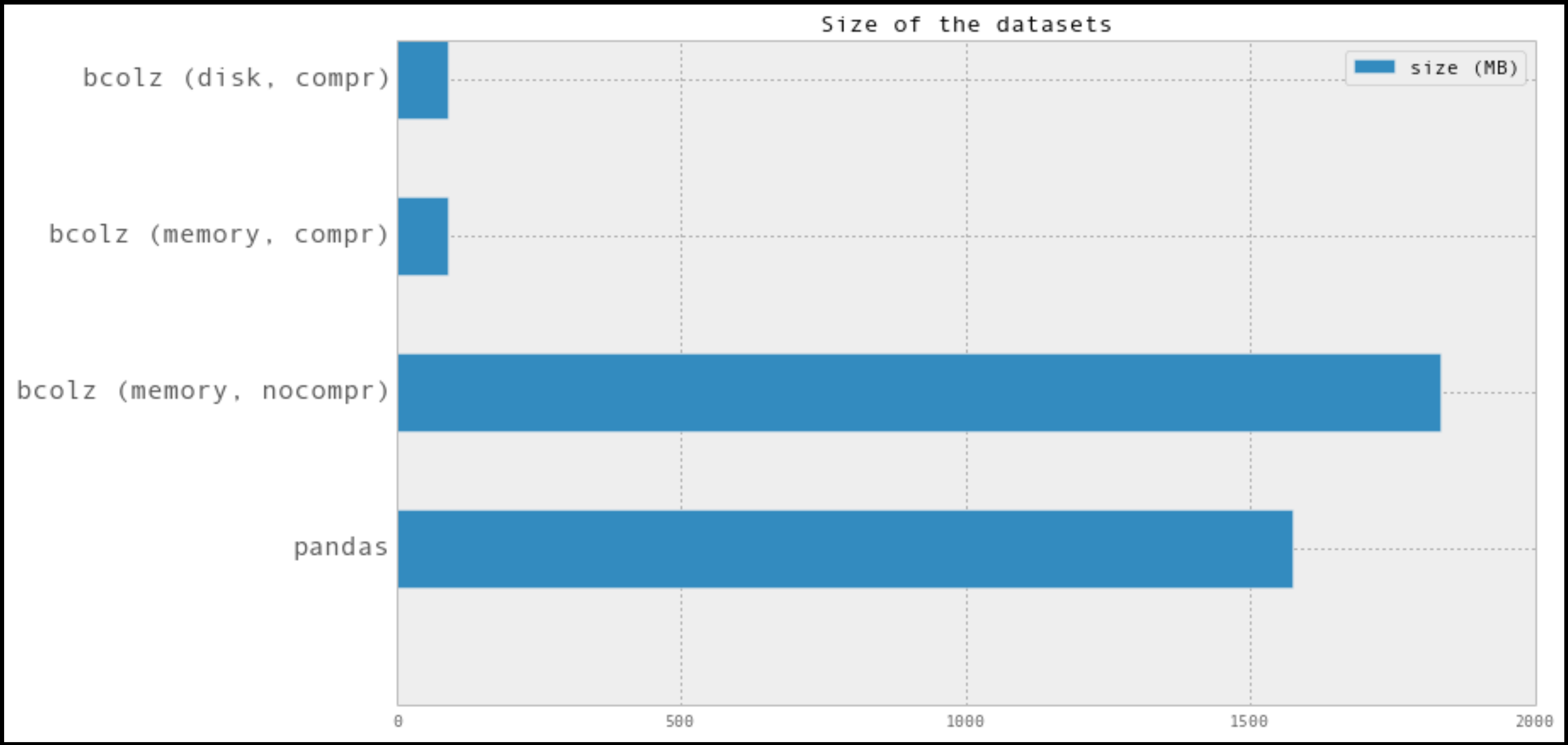
The MovieLens Dataset

- Datasets for movie ratings
- Different sizes: 100K, 1M, 10M ratings (the 10M will be used in benchmarks ahead)
- The datasets were collected over various periods of time

Querying the MovieLens Dataset

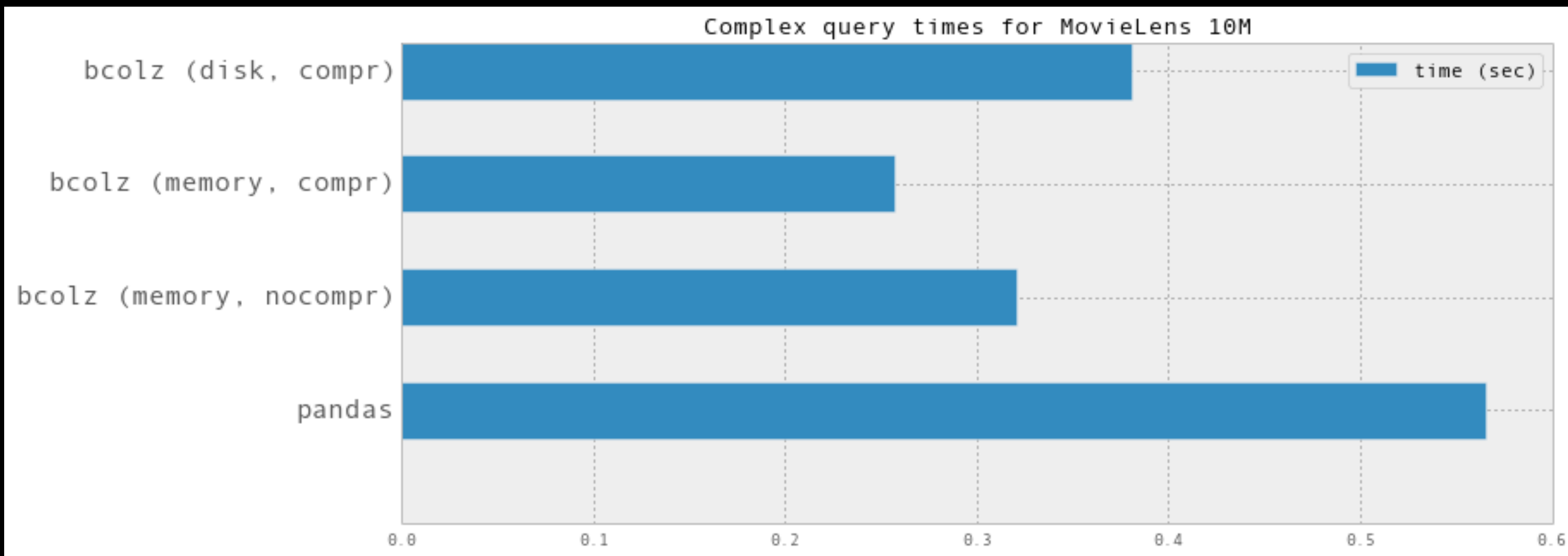
```
import pandas as pd
import bcolz


# Parse and load CSV files using pandas


# Merge some files in a single dataframe
lens = pd.merge(movies, ratings)


# The pandas way of querying
result = lens.query("(title == 'Tom and Huck (1995)') & (rating == 5)”)['user_id']


zlens = bcolz.ctable.fromdataframe(lens)


# The bcolz way of querying (notice the use of the `where` iterator)
result = [r.user_id for r in dblens.where(
    "(title == 'Tom and Huck (1995)') & (rating == 5)", outcols=['user_id'])]
```

Sizes of Datasets



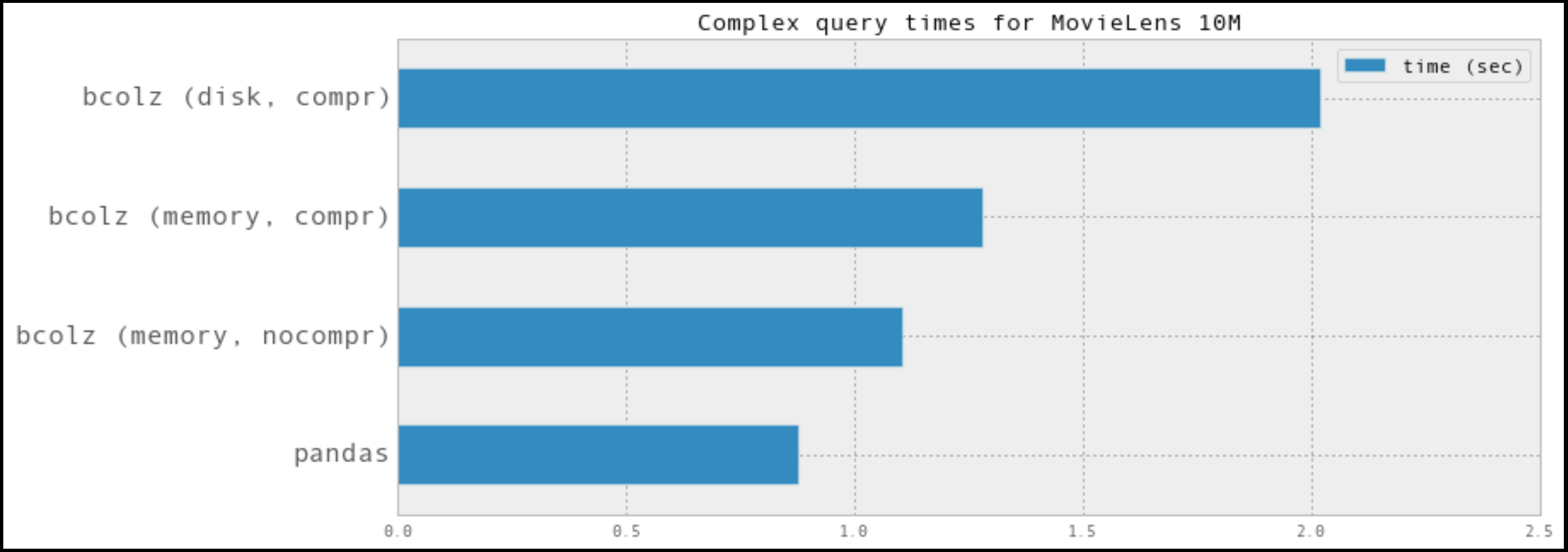
- Compression means ~20x less space
- The uncompressed ctable is larger than pandas

Query Times
(laptop 1-year old)



- Compression leads to better query speeds (15% faster)
- Querying a disk-based ctable is fast!

Query Times (laptop 3-year old, Core2)



- Compression still makes things slower on old boxes (15% slower)
- So, expect better improvements in the future

Status and Overview

- Version 0.7.0 released this week. Check it out!
- Focus on refining the API and tweaking knobs for making things even faster
- Better integration with bloscpack (super-chunks)
- bcolz main goal is to demonstrate that compression can help performance, even using **in-memory** data containers

Tell Us About Your Experience!

- Which is your scenario?
- You are not getting the expected speed or compression ratio?
- Mailing list: http://groups.google.com/group/bcolz
- Bugs/patches, please file them at: http://github.com/Blosc/bcolz

References

- Manual: http://bcolz.blosc.org/
- Bloscpack: https://github.com/Blosc/bloscpack
- The Blosc ecosystem: http://blosc.org/

Thank you!

Questions?