New Trends In Storing And Analyzing Large Data Silos With Python

Francesc Alted
Freelancer
(*ÜberResearch, University of Oslo*)

April 3rd, 2015

About ÜberResearch

- Team's 10+ years experience delivering solutions and services for funding and research institutions
- Over 20 development partners, and clients globally, from smallest non-profits to large government agencies
- Portfolio company of Digital Science (Macmillan Publishers), the younger sibling of the Nature Publishing Group

http://www.uberresearch.com/

Application layer	Dimensions FOR FUNDERS				Integration in customer application and scenarios
Functional layer	NLP / noun phrase extraction	Thesaurus support for indexing			Customer functionalities
	Search	Clustering / topic modelling	Research classification support	Visualisation support	APIs
Data Enrichment	Data model mapping / cleansing	Institution disambiguation	Person disambiguation		Customer services and APIs
Data layer	Global grant database	Publications, trials, patents			Internal data, databases

About Me

- Physicist by training
- Computer scientist by passion
- Open Source enthusiast by philosophy
  - PyTables (2002 - 2011)
  - Blosc (2009 - now)
  - bcolz (2010 - now)

Why Free/Libre Projects?

"*The art is in the execution of an idea. Not in the idea. There is not much left just from an idea.*"

–Manuel Oltra, music composer

"*Real artists ship*"

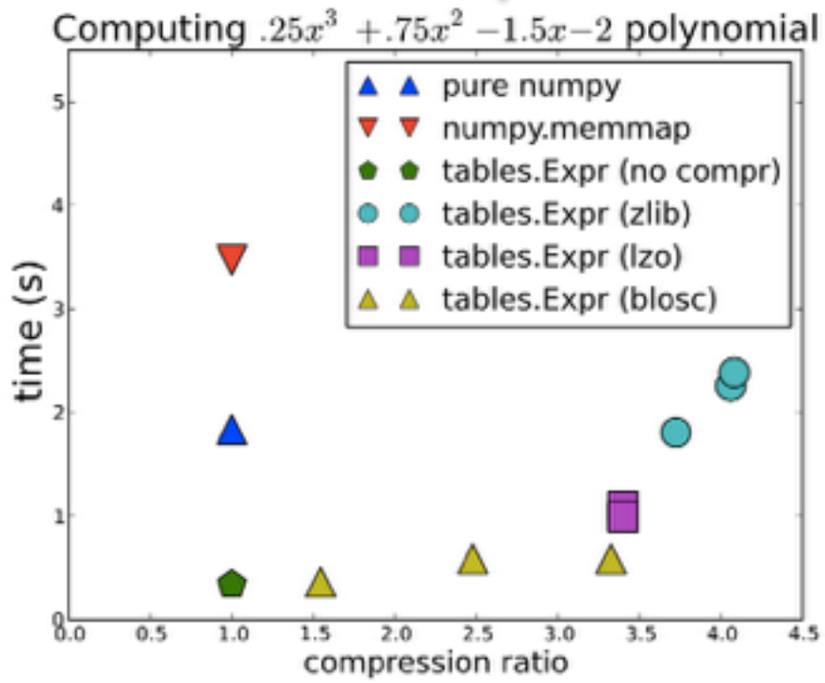
–Seth Godin, writer

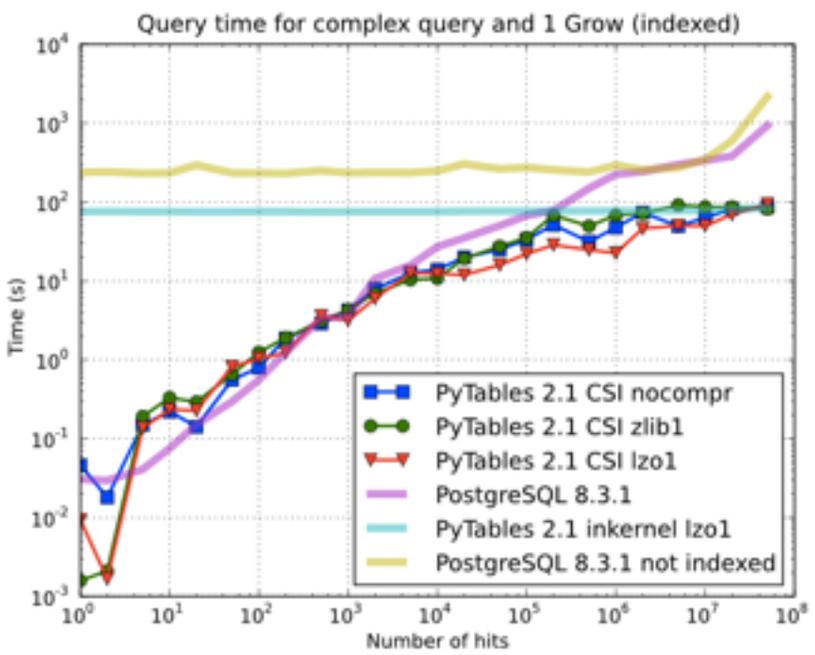
- Nice way to realize yourself while helping others

The technology platform to make a difference in your relationship with large and complex data

HDF5 + a Twist

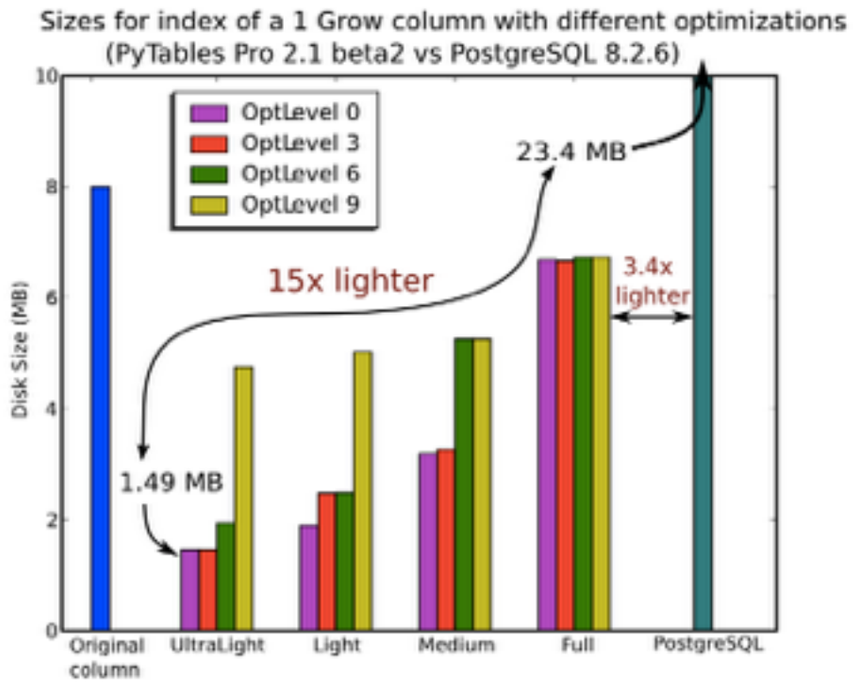
Out-of-core Expressions





OPSI

Indexed Queries



Overview

- The need for speed: fitting and analyzing as much data as possible with your existing resources

- Recent trends in computer hardware

- **bcolz**: an example of data container for large datasets following the principles of newer computer architectures

The Need For Speed

Don’t Forget Python’s Real Strengths

- *Interactivity*
- Data-oriented libraries (NumPy, Pandas, Scikit-Learn…)
- *Interactivity*
- Performance (thanks to Cython, SWIG, f2py…)
- **Interactivity** (did I mentioned that already?)

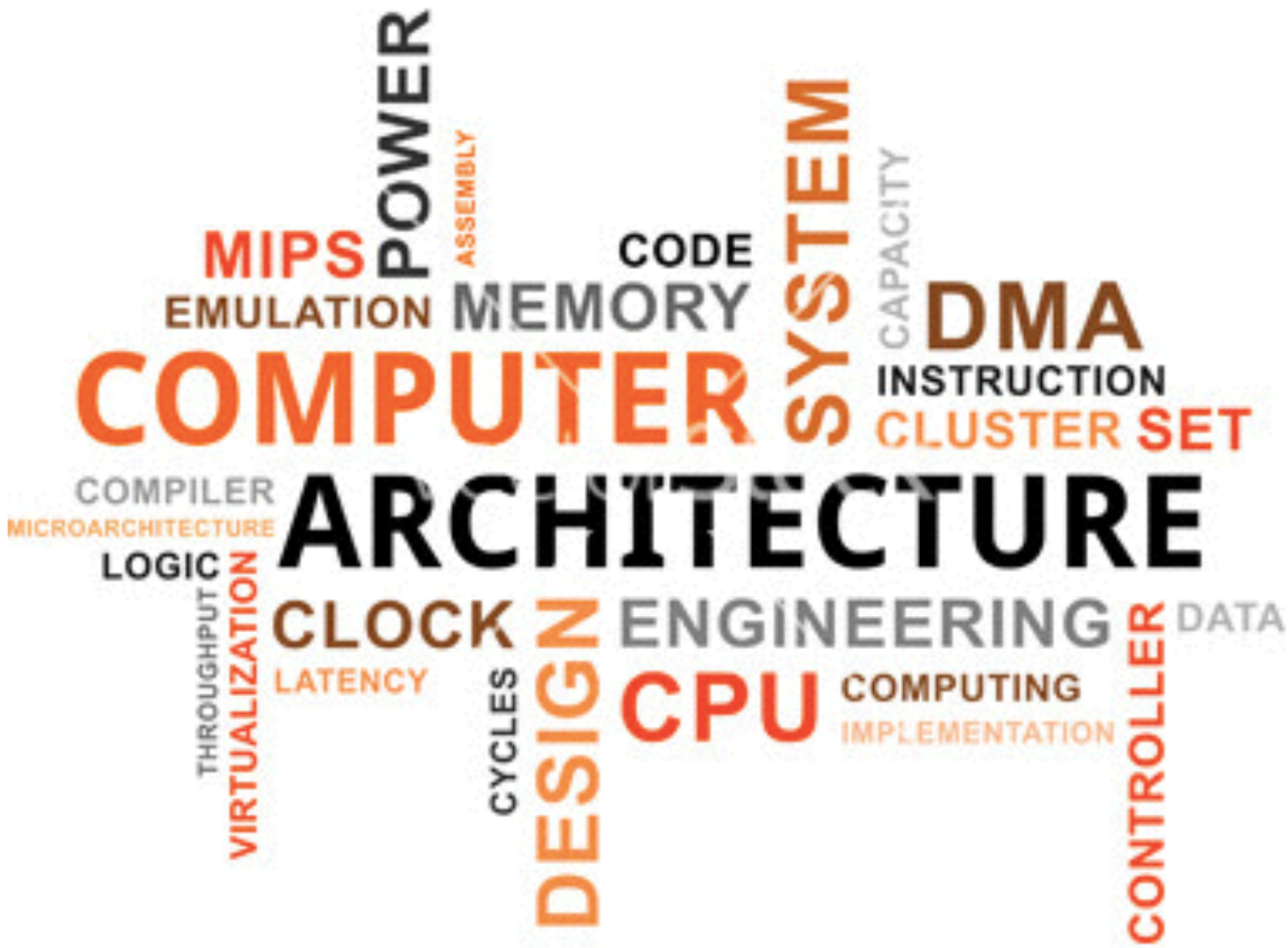
The Need For Speed

- But interactivity without performance in Big Data is a no go
- Designing code for data storage performance depends very much on computer architecture
- IMO, existing Python libraries need more effort in getting the most out of existing and future computer architectures

The Daily Python Working Scenario

Quiz: which computer is best for interactivity?

Although Modern Servers/Laptops Can Be Very Complex Beasts



We need to know them better so as to get the most out of them

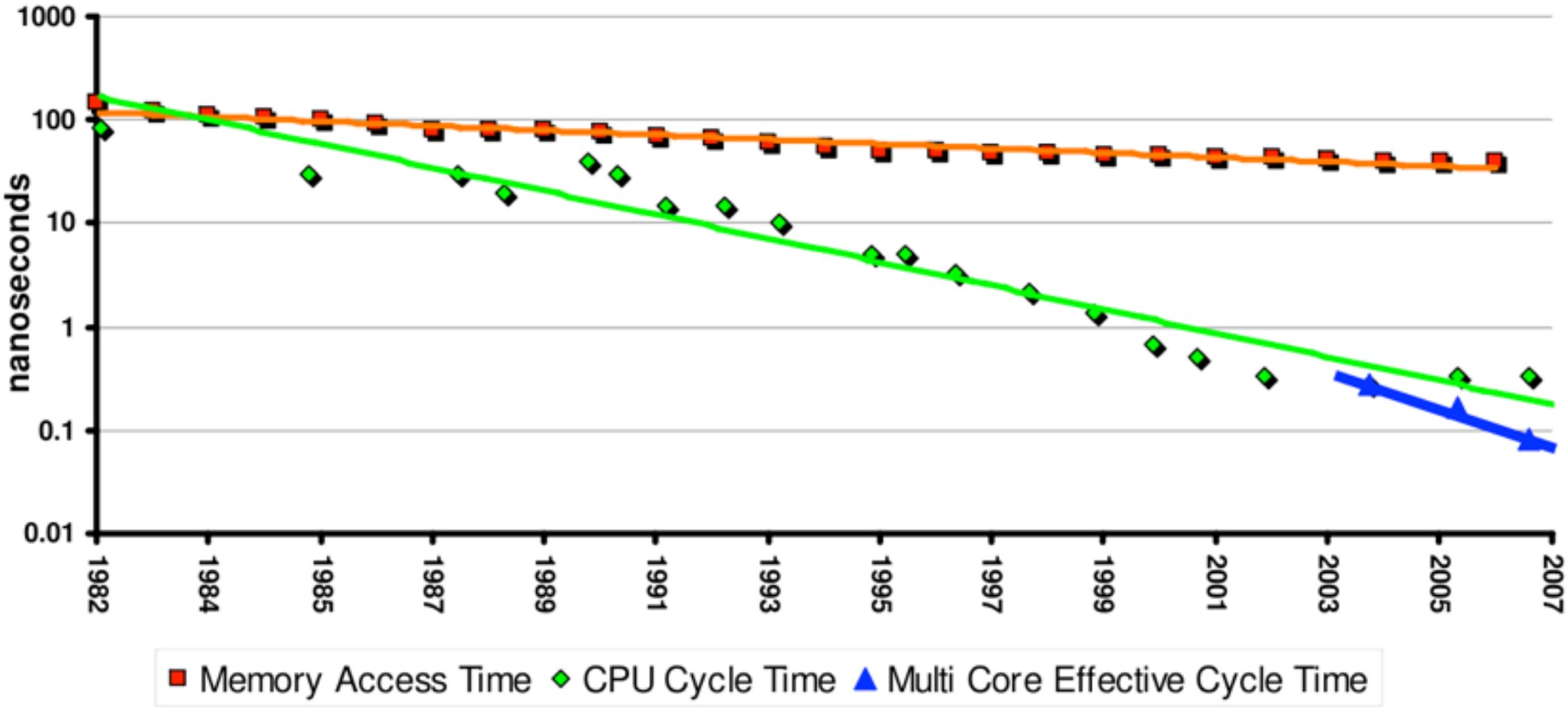
“There's Plenty of Room at the Bottom”

An Invitation to Enter a New Field of Physics

—Talk by Richard Feynman at Caltech, 1959

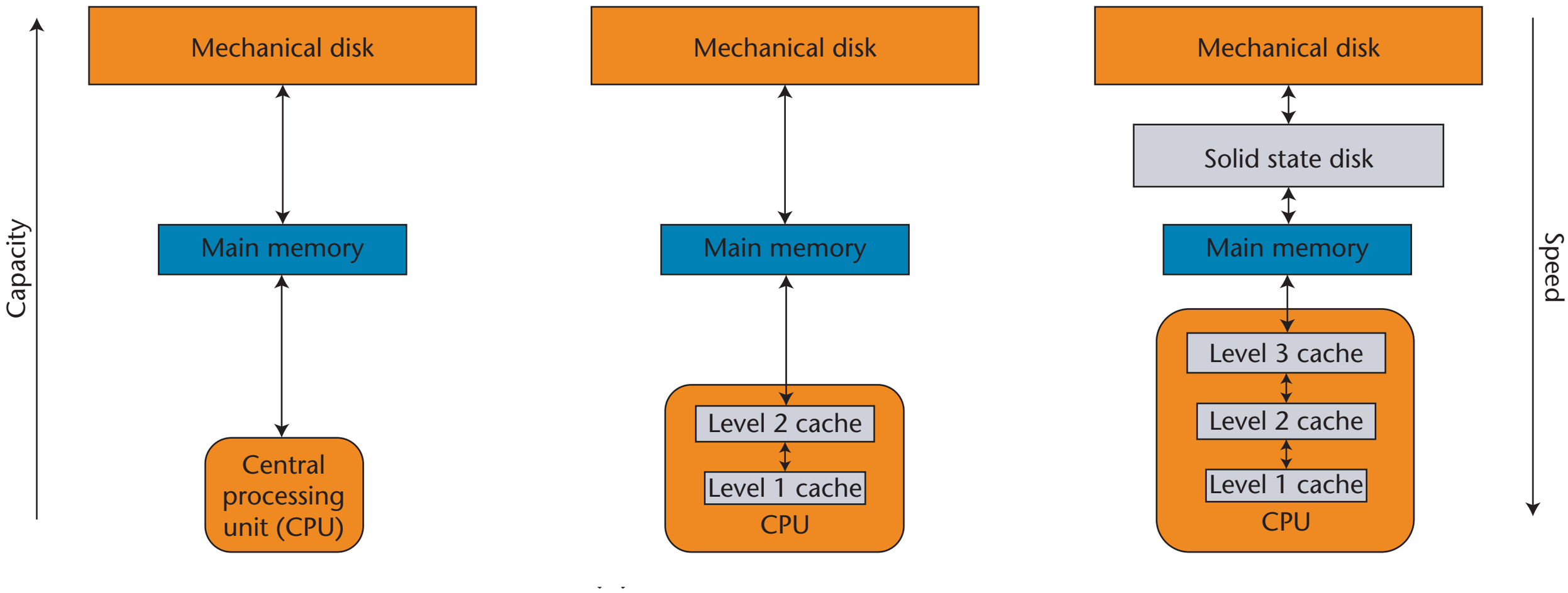
Recent Trends In Computer Hardware

Memory Access Time vs CPU Cycle Time



The gap is wide and still opening!

Computer Architecture Evolution



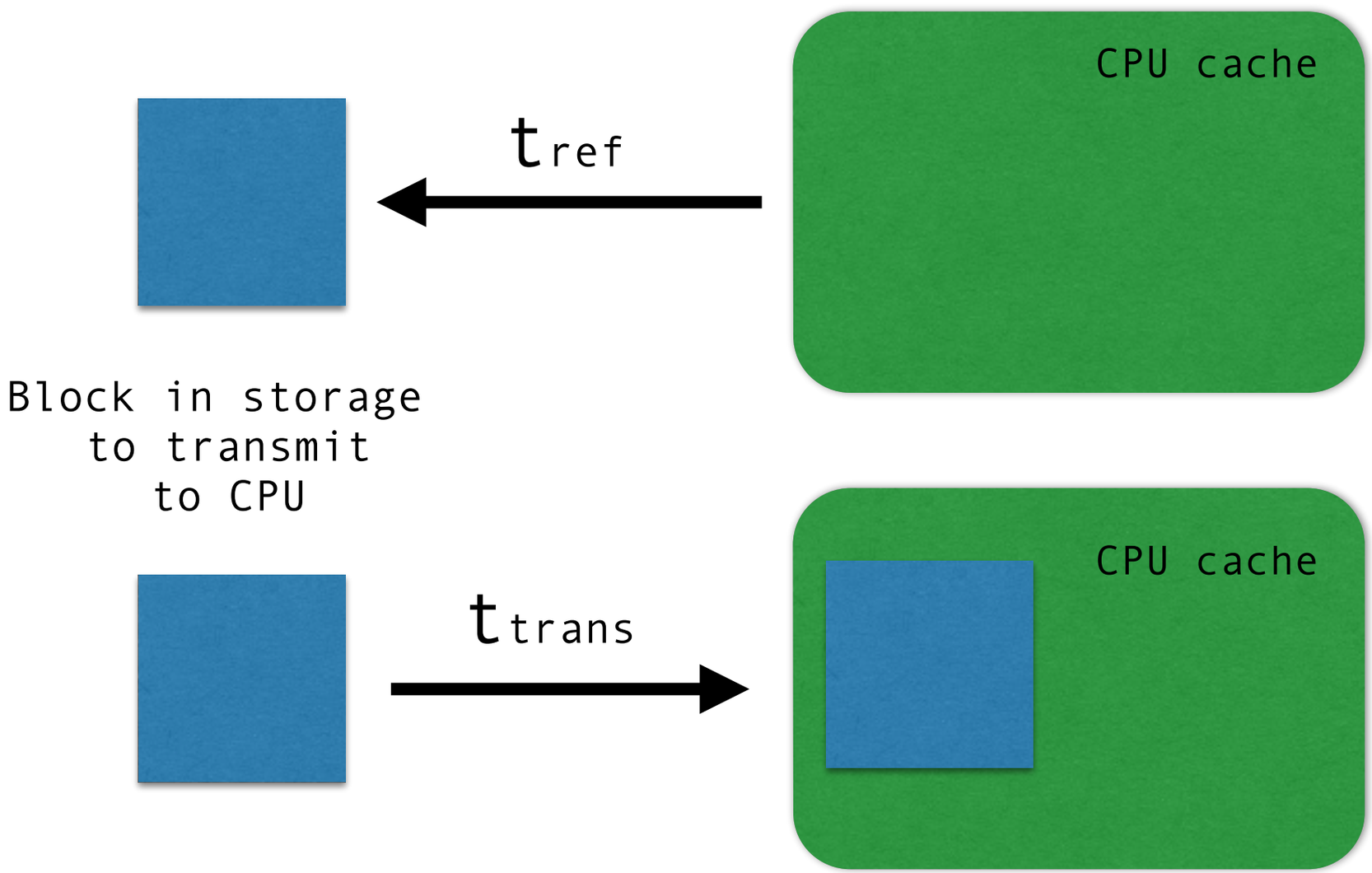
Latency Numbers Every Programmer Should Know

```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                     20x L2 cache, 200x L1 cache
Read 4K randomly from memory             1,000   ns    0.001 ms
Compress 1K bytes with Zippy             3,000   ns
Send 1K bytes over 1 Gbps network       10,000   ns    0.01 ms
Read 4K randomly from SSD*             150,000   ns    0.15 ms
Read 1 MB sequentially from memory     250,000   ns    0.25 ms
Round trip within same datacenter      500,000   ns    0.5  ms
Read 1 MB sequentially from SSD*     1,000,000   ns    1    ms  4X memory
Disk seek                           10,000,000   ns   10    ms  20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20    ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150    ms
```

Source: Jeff Dean and Peter Norvig (Google), with some additions

https://gist.github.com/hellerbarde/2843375

Reference Time vs Transmission Time



t_{ref} ~= t_{trans} => optimizes memory access

Not All Storage Layers Are Created Equal

Memory: t_{ref}: 100 ns / t_{trans} (**1 KB**): ~100 ns

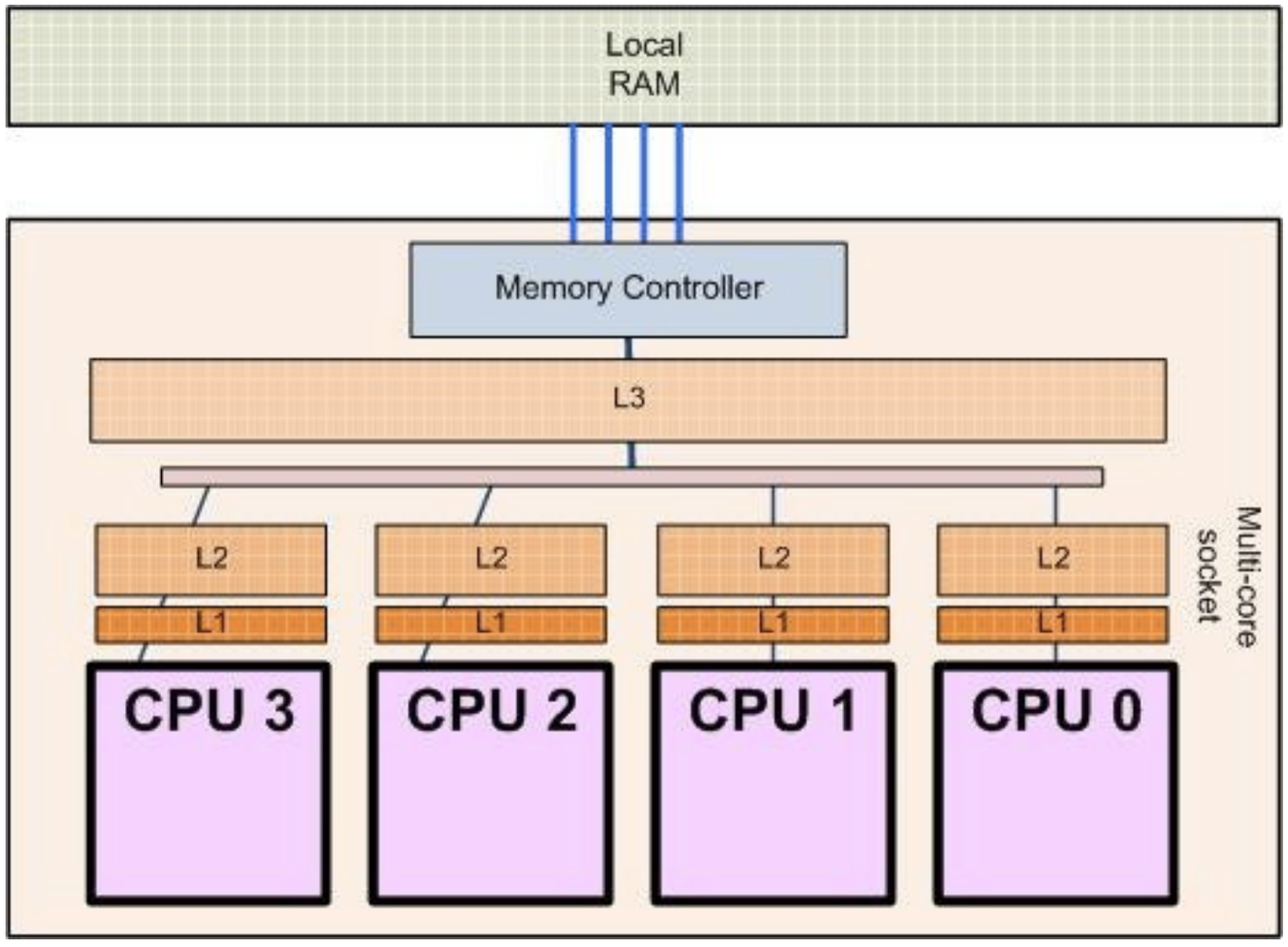
Solid State Disk: t_{ref}: 10 us / t_{trans} (**4 KB**): ~10 us

Mechanical Disk: t_{ref}: 10 ms / t_{trans} (**1 MB**): ~10 ms

The slower the media, the larger the block that is worth to transmit

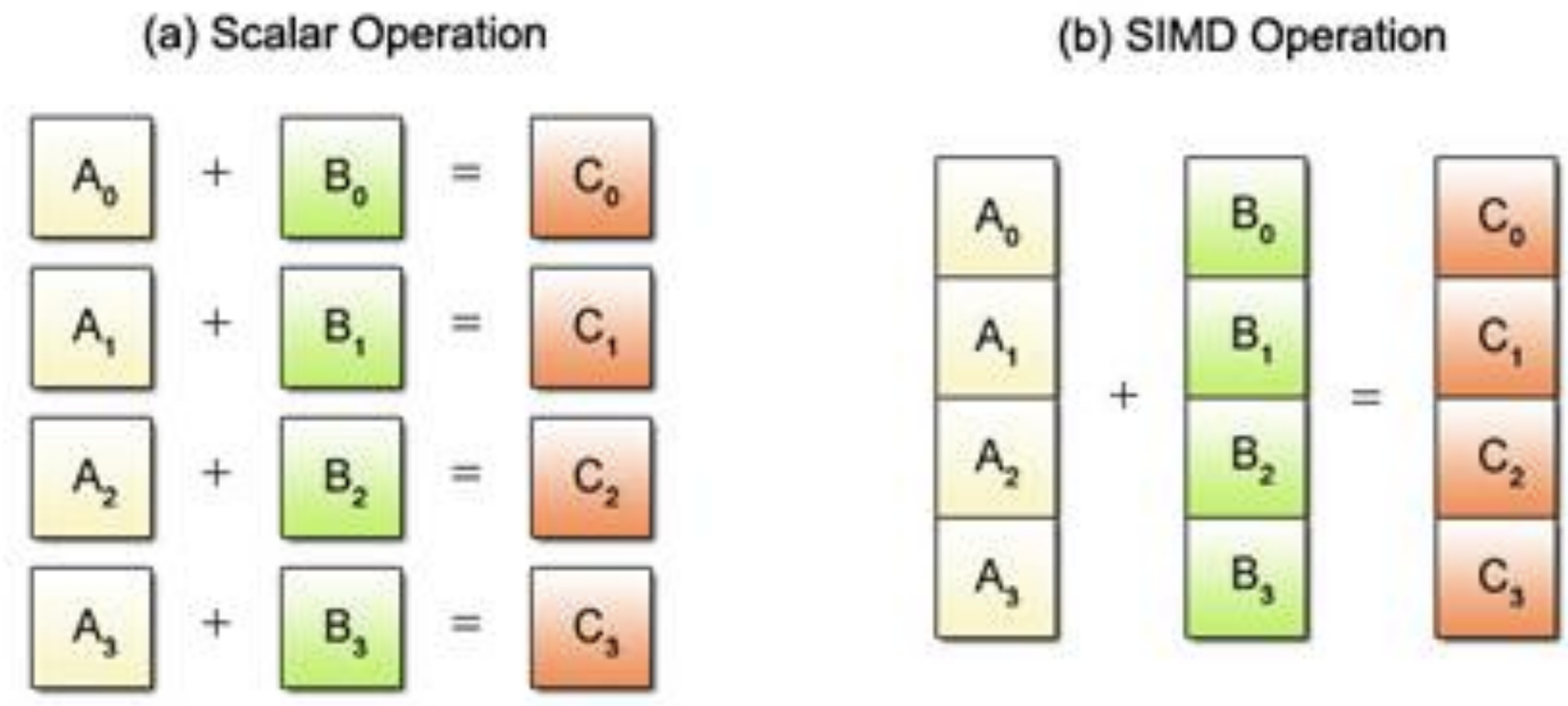
This has profound implications on how you access storage!

We Are In A Multicore Age



- This requires special programming measures to leverage all its potential: threads, multiprocessing

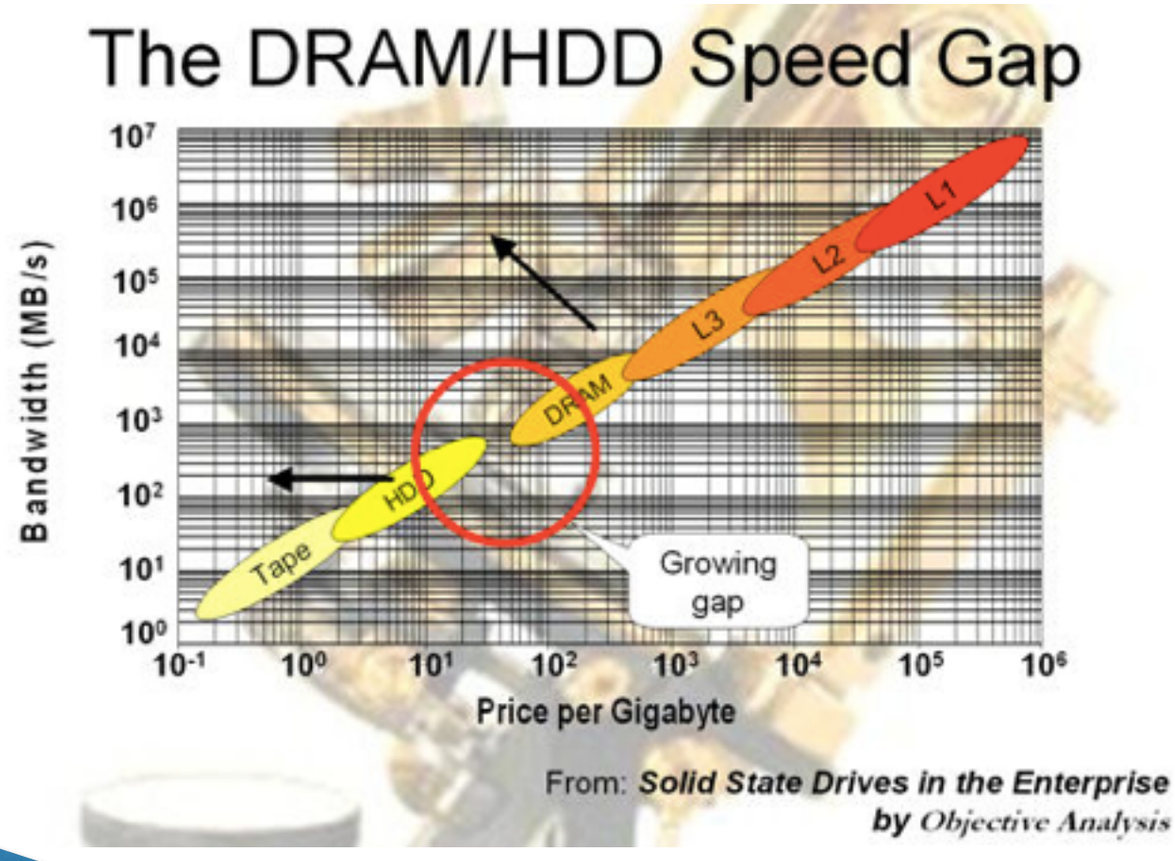
SIMD: Single Instruction, Multiple Data



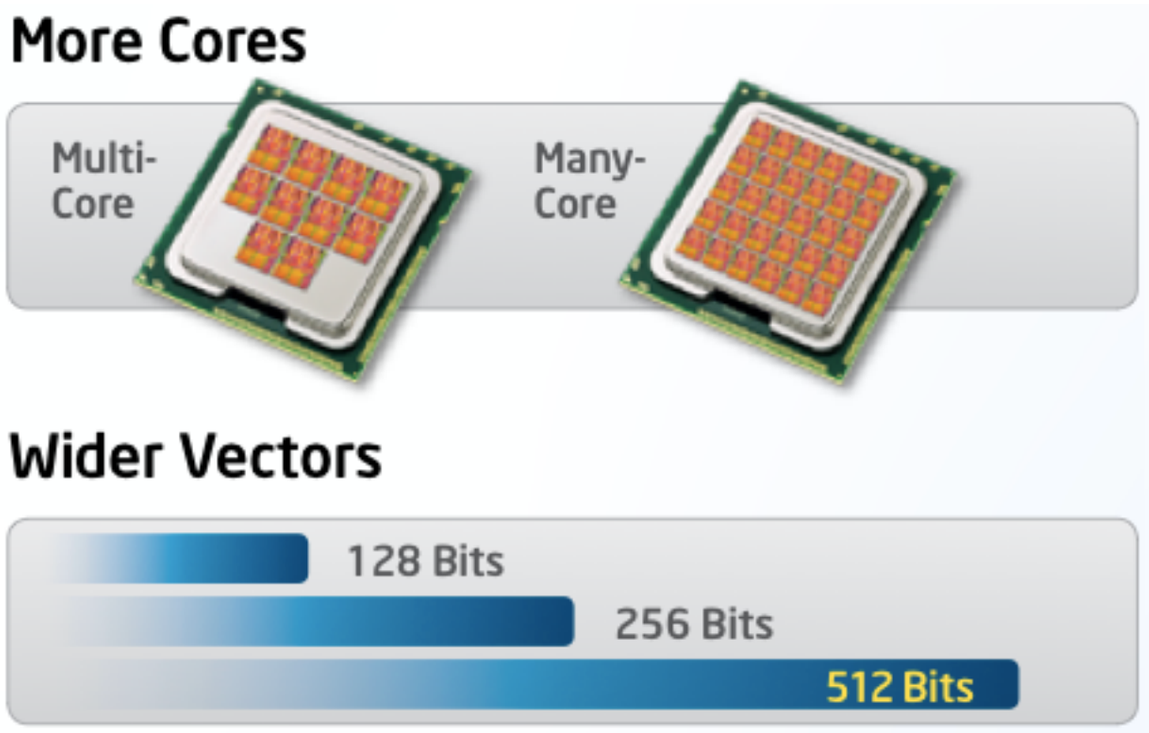
More operations in the same CPU clock

Forthcoming Trends (I)

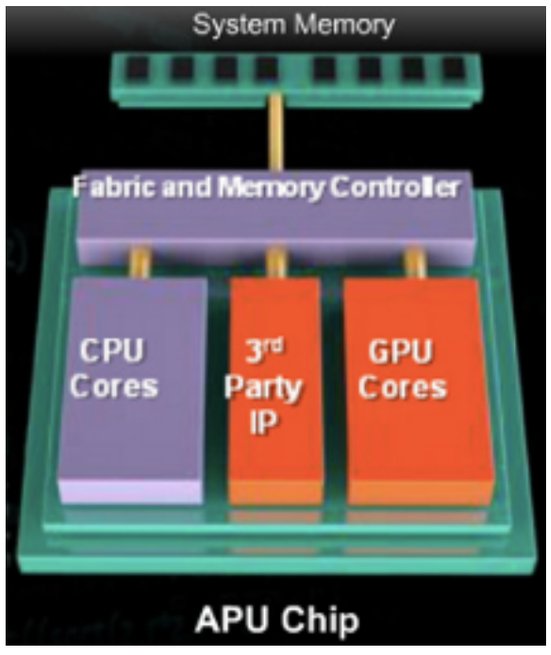
The growing gap between DRAM and HDD is facilitating the introduction of new SDD devices



Forthcoming Trends (II)



CPU+GPU Integration



Bcolz: An Example Of Data Containers Applying The Principles Of New Hardware

What is bcolz?

- bcolz provides data containers that can be used in a similar way than the ones in NumPy, Pandas

- The main difference is that data storage is **chunked**, not **contiguous**

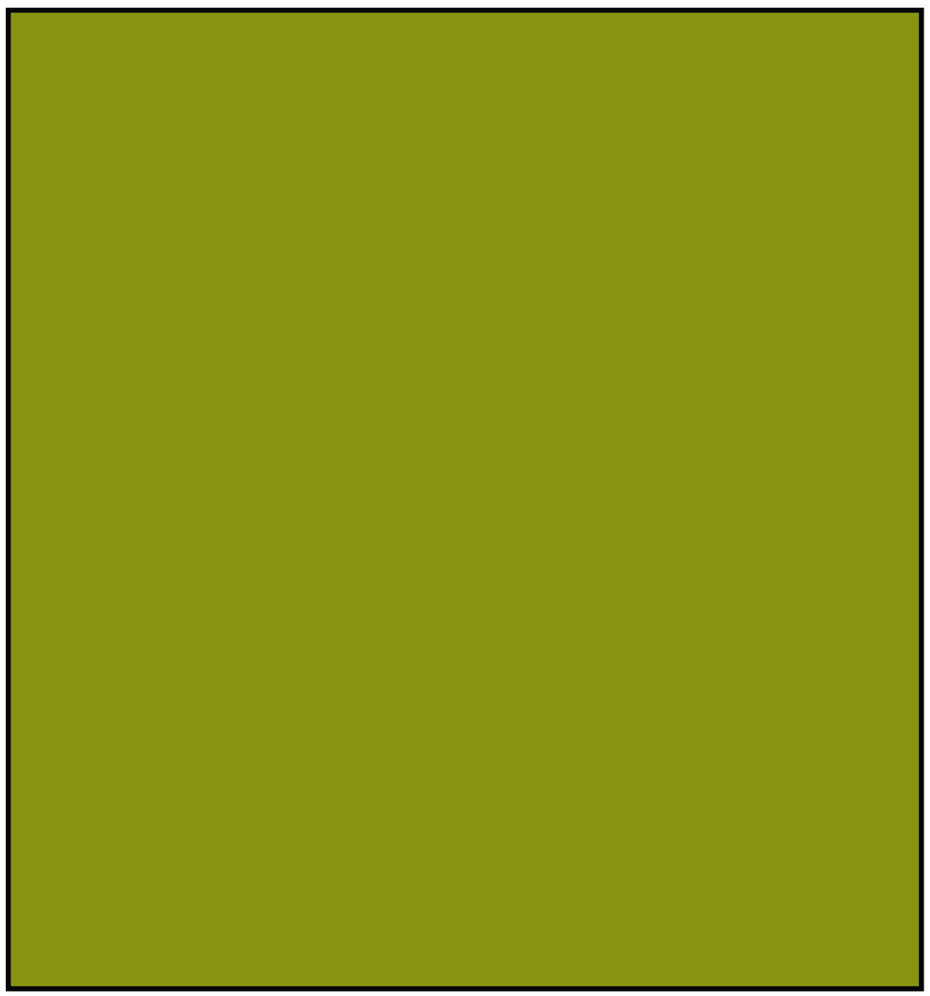
- Two flavors:

    - **carray**: homogenous, n-dim data types

    - **ctable**: heterogeneous types, columnar

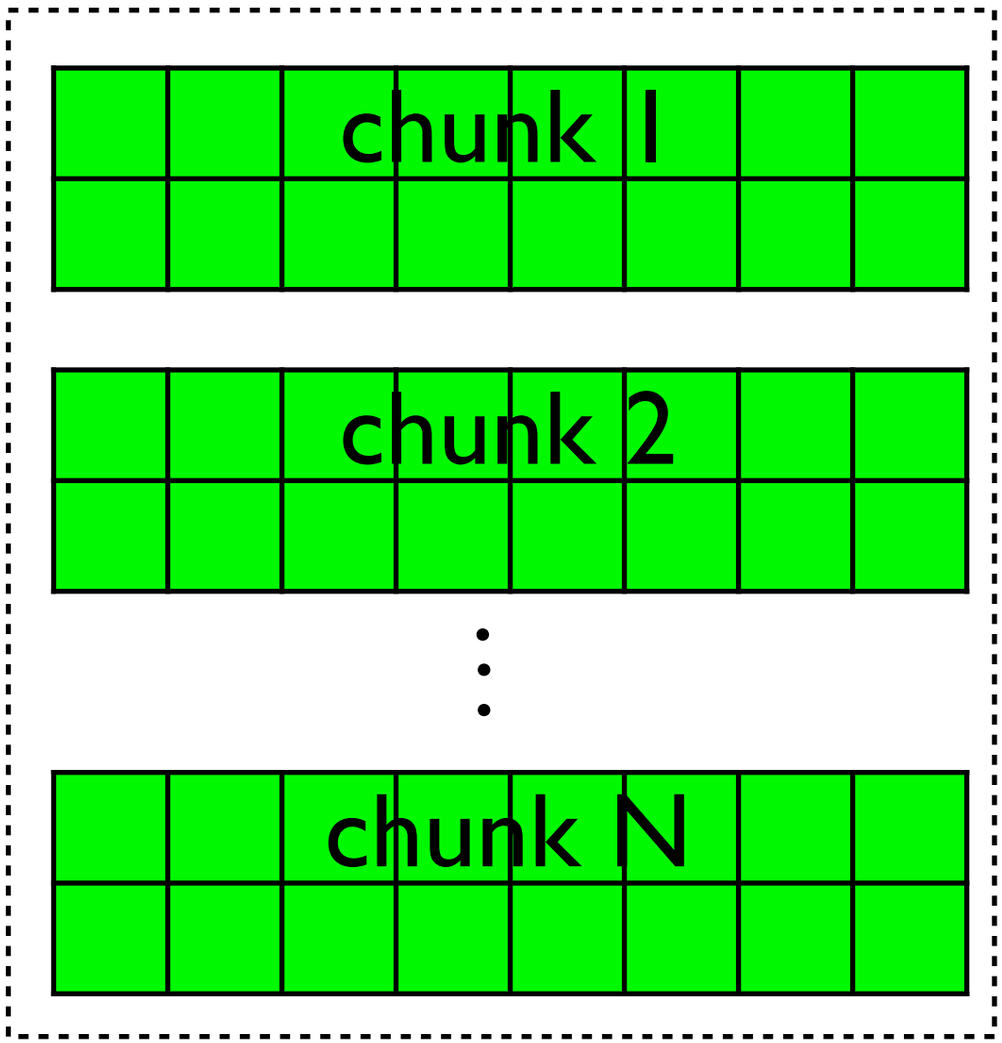
Contiguous vs Chunked

NumPy container



Contiguous memory

carray container

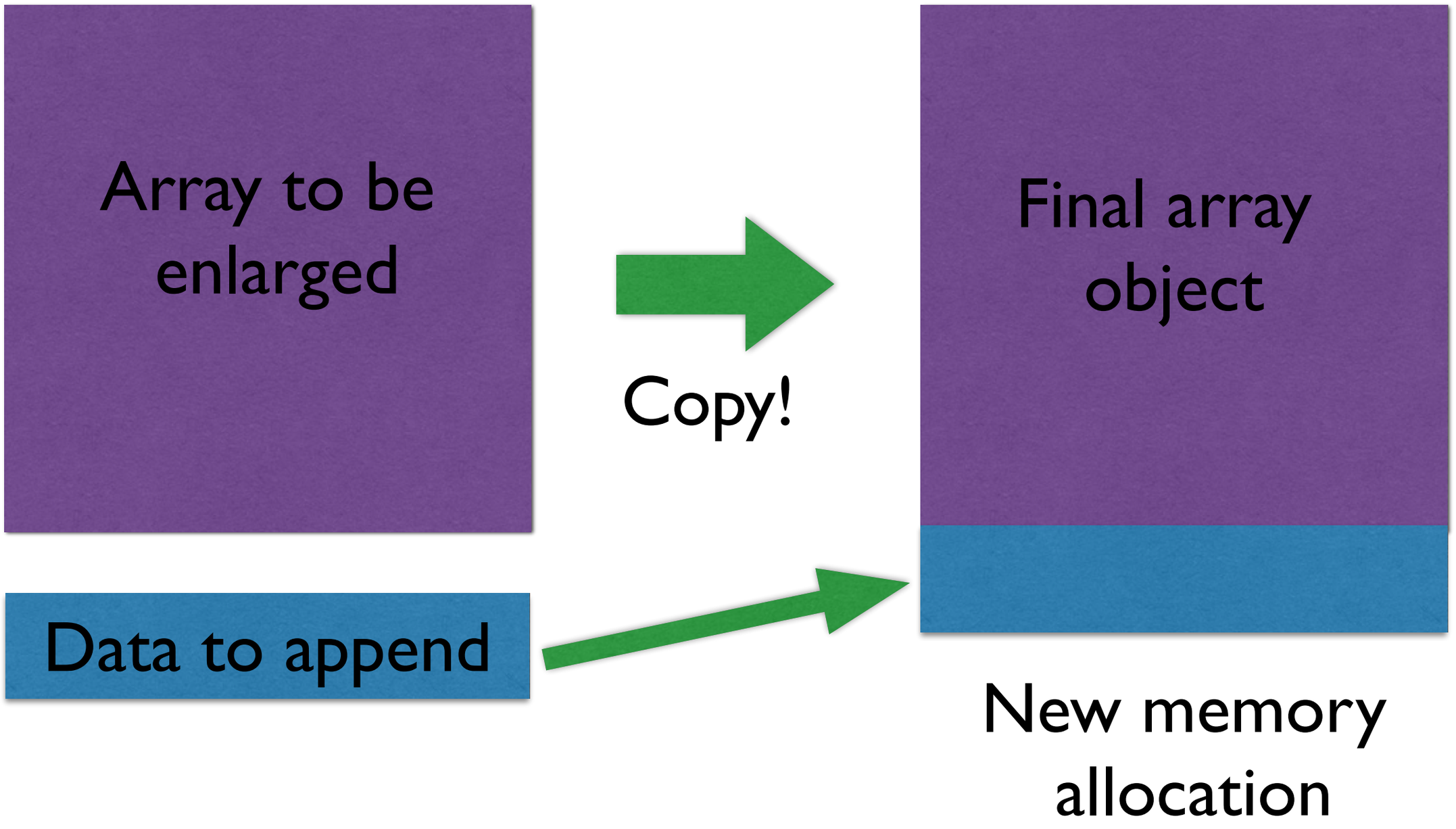


Discontiguous memory

Why Chunking?

- Chunking means more difficulty handling data, so why bother?
  - Efficient enlarging and shrinking
  - Compression is possible
  - Chunk size can be adapted to the storage layer (memory, SSD, mechanical disk)

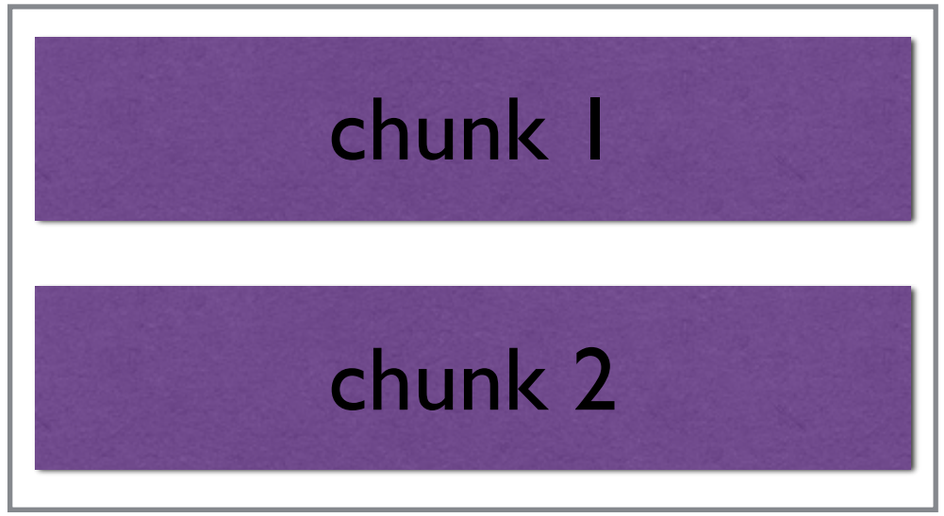
Appending Data in NumPy



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

Appending Data in bcolz

carray to be enlarged

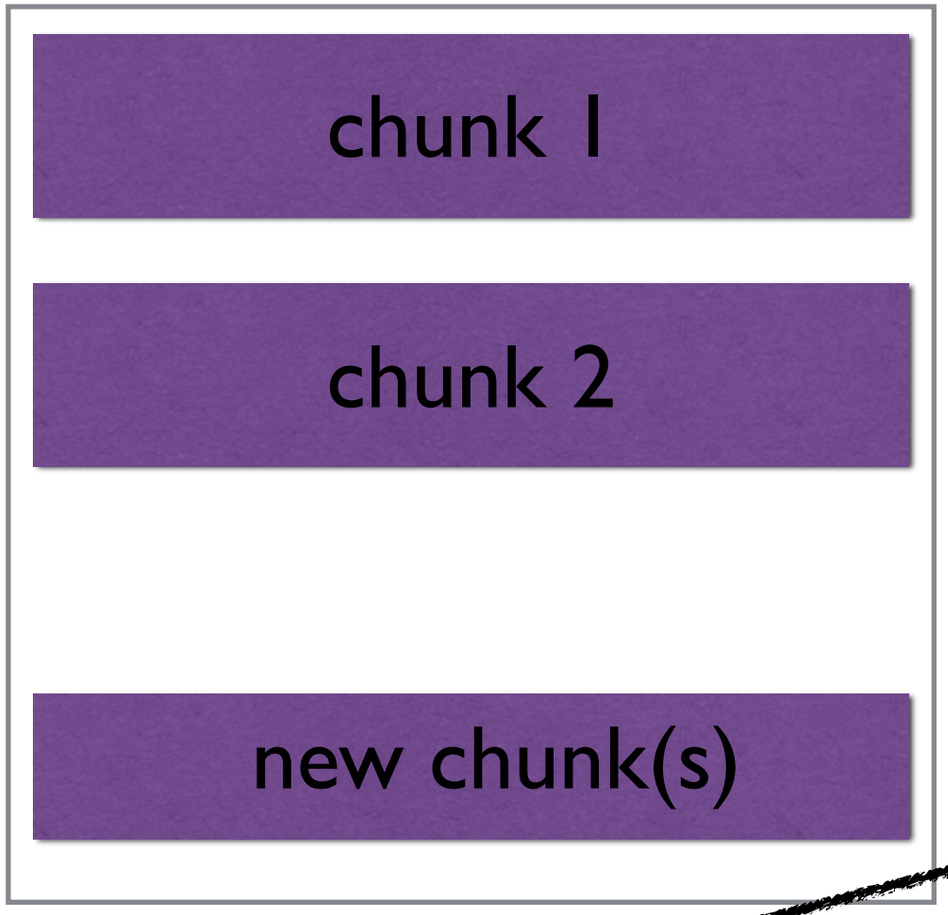


data to append

Blosc

compression

final carray object



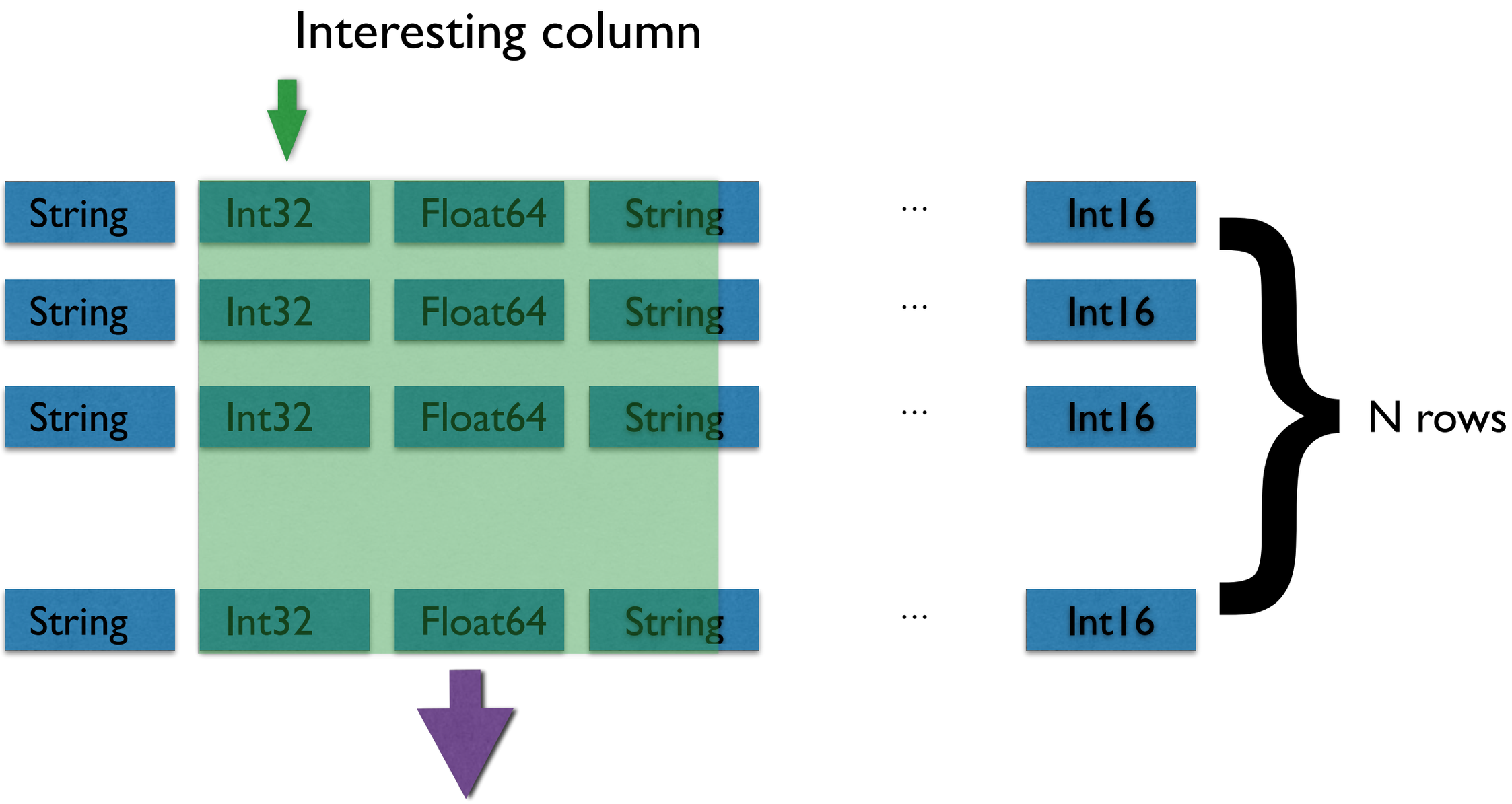
Only compression on new data is required!

Less memory travels to CPU!

Why Columnar?

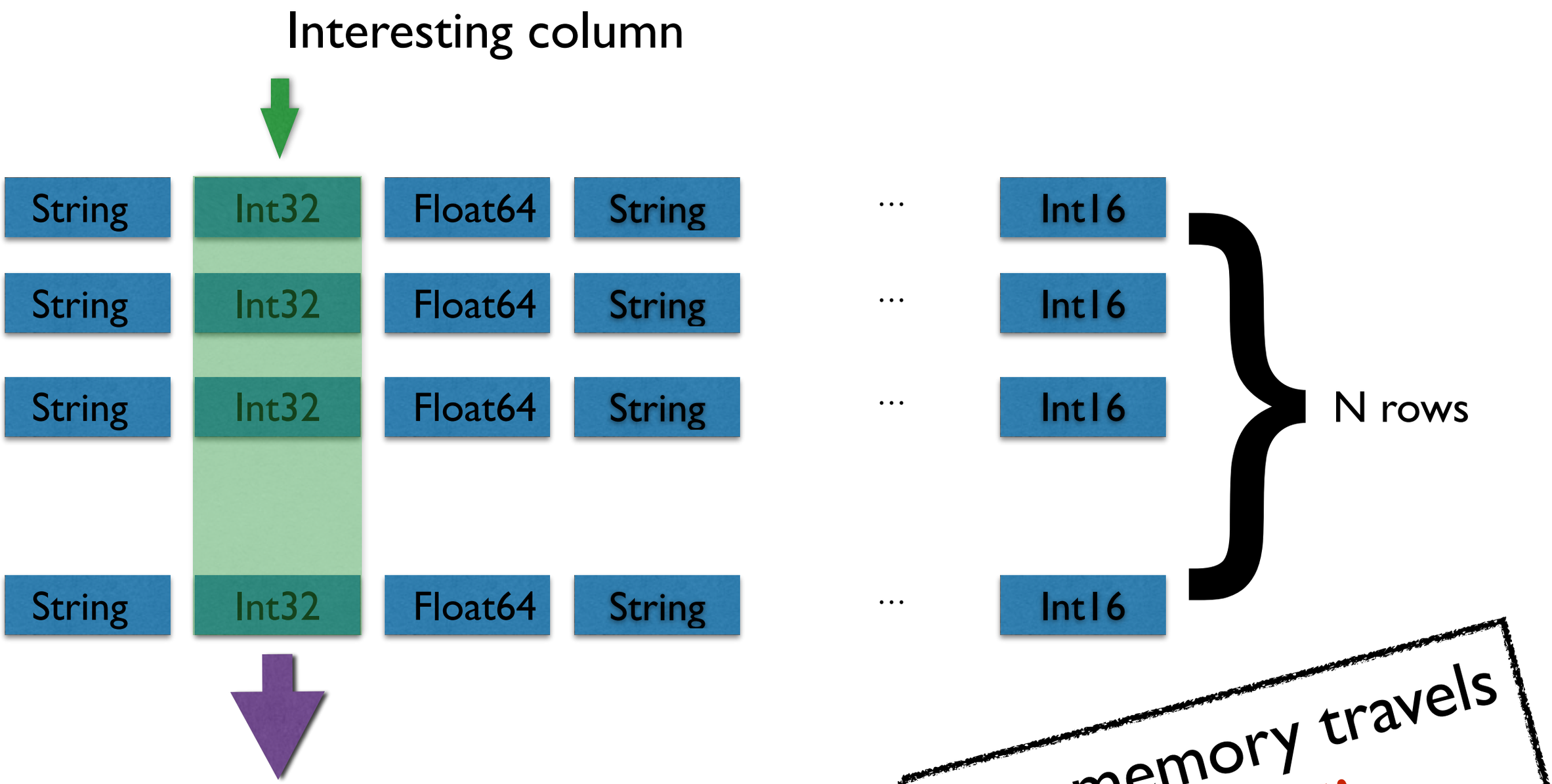
- Because it adapts better to newer computer architectures

In-Memory Row-Wise Table (Structured NumPy array)



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

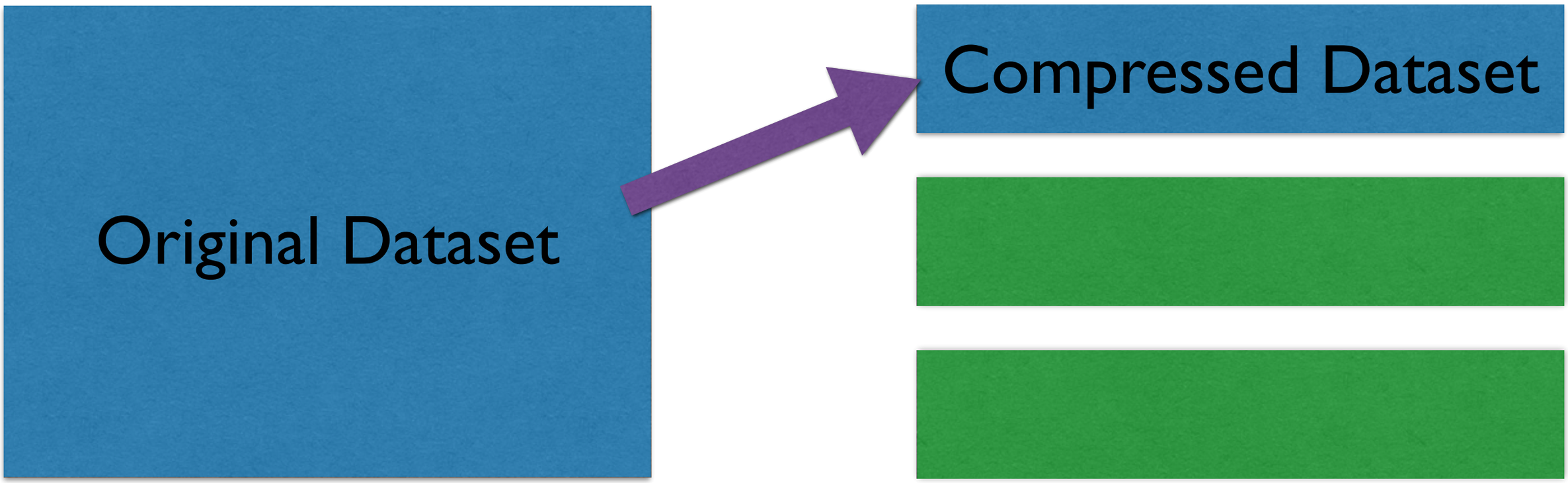
In-Memory Column-Wise Table (bcolz *ctable*)



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

Less memory travels **to CPU!**

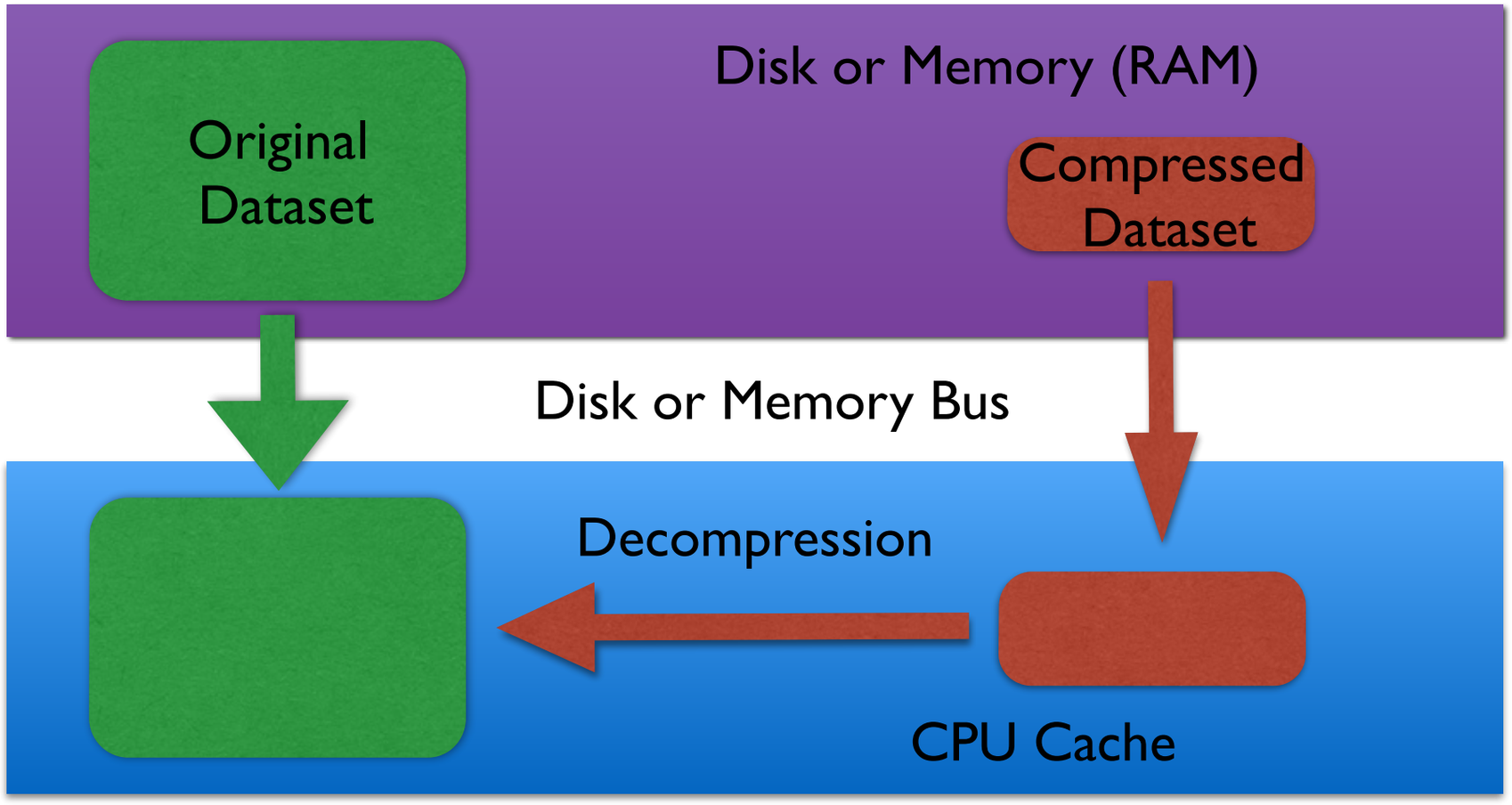
Why Compression (I)?



More data can be packed using the same storage

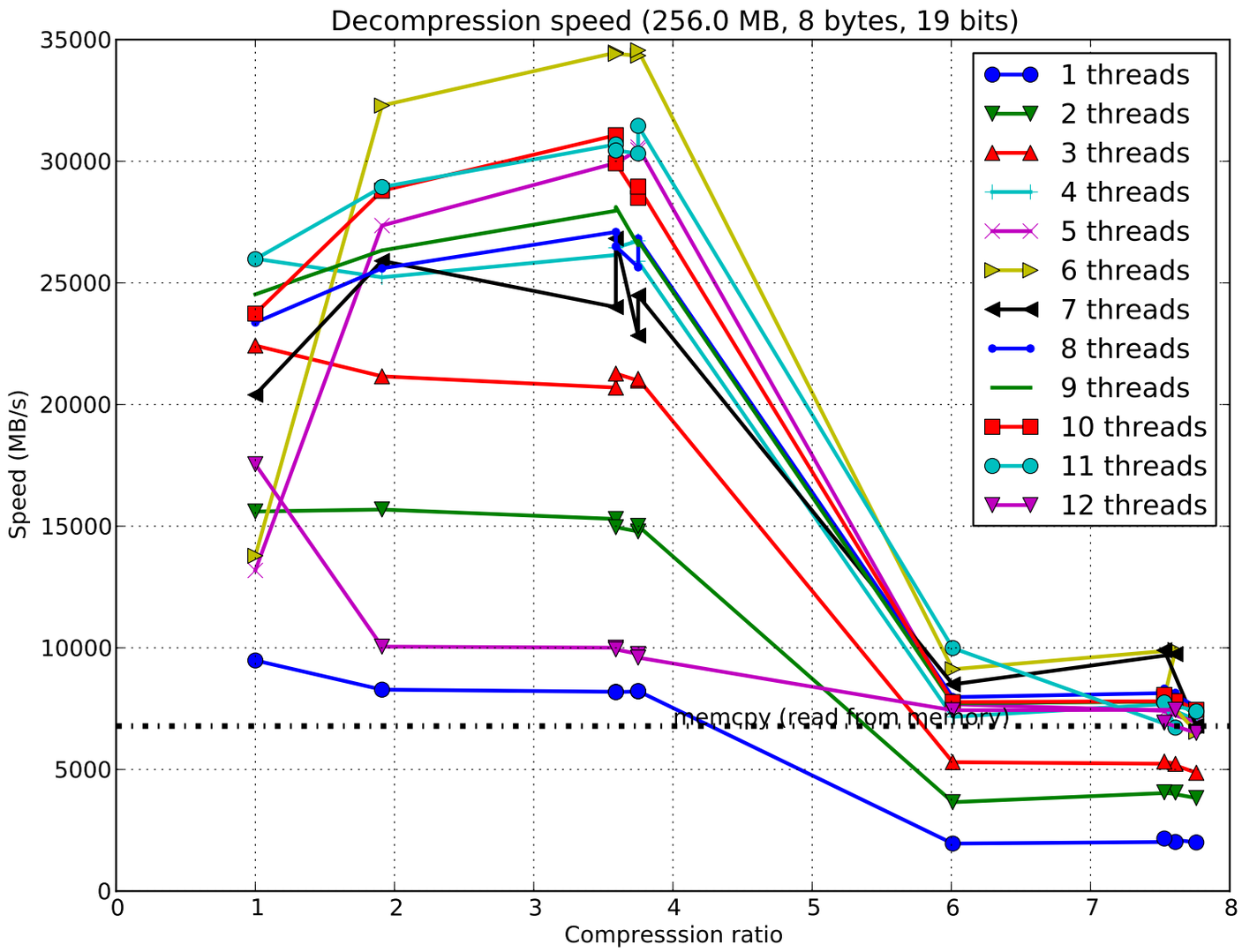
Why Compression (II)?

Less data needs to be transmitted to the CPU



Transmission + decompression faster than direct transfer?

Blosc: Compressing Faster Than *memcpy()*



How Blosc Works

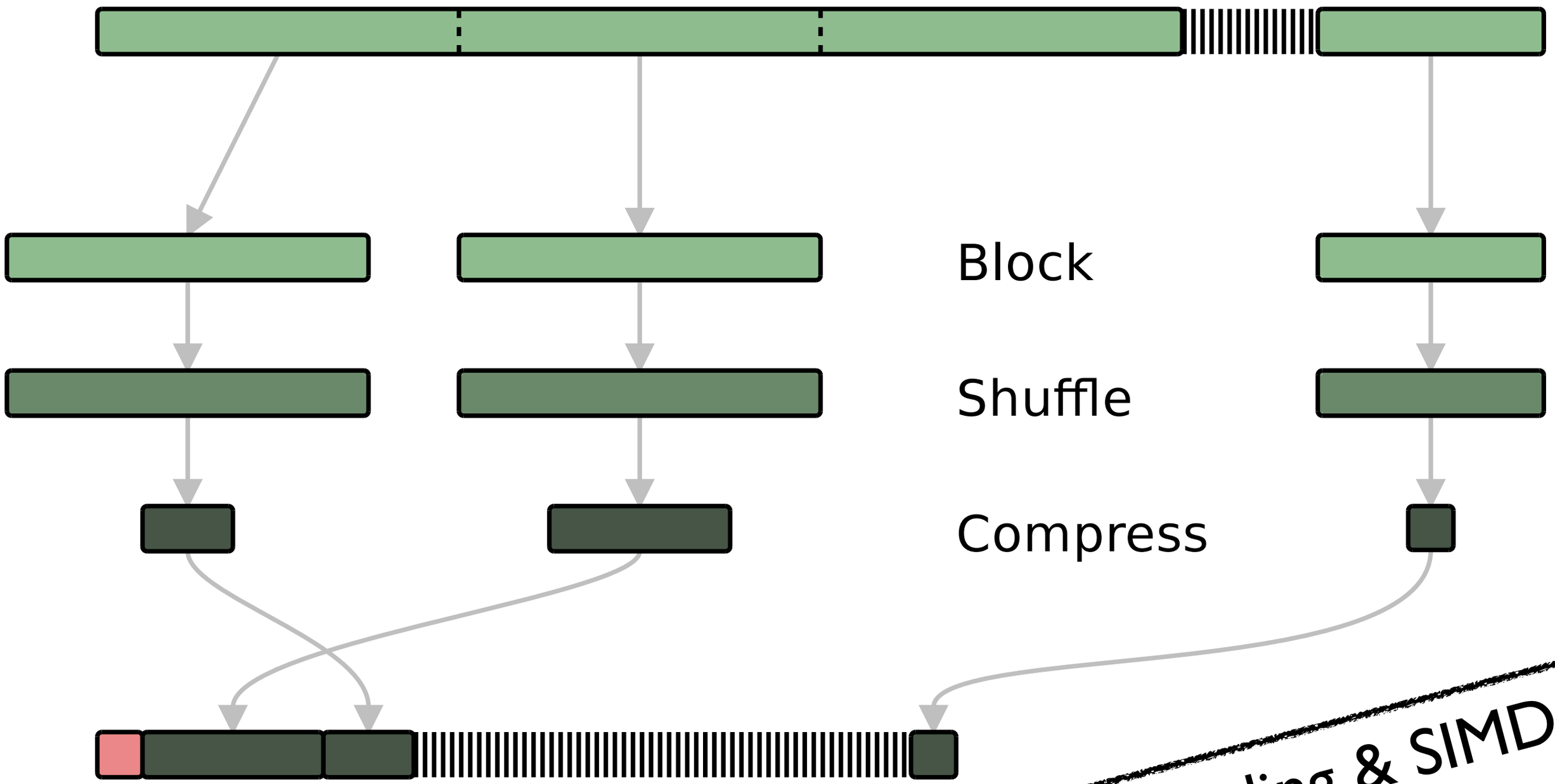
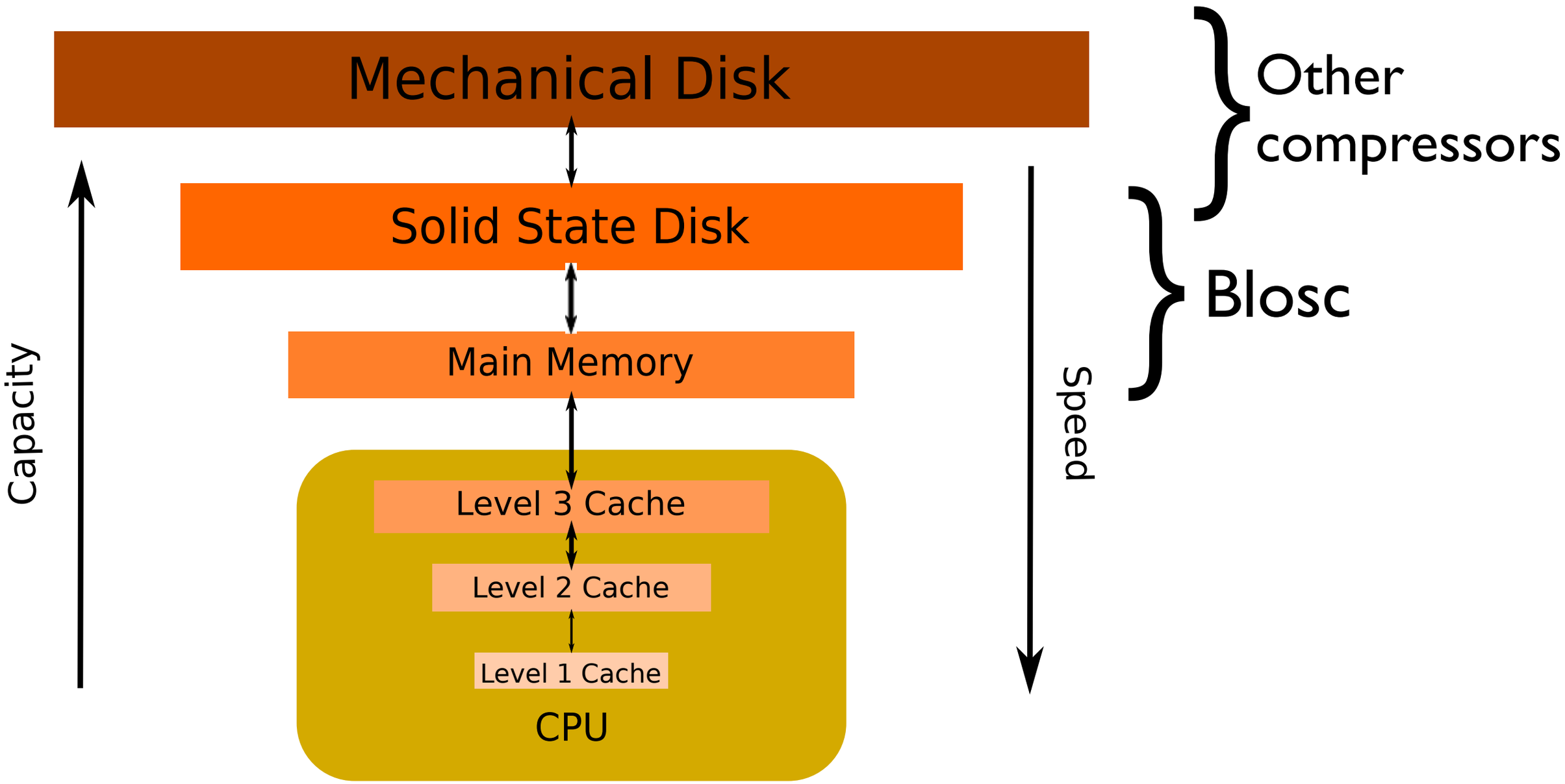


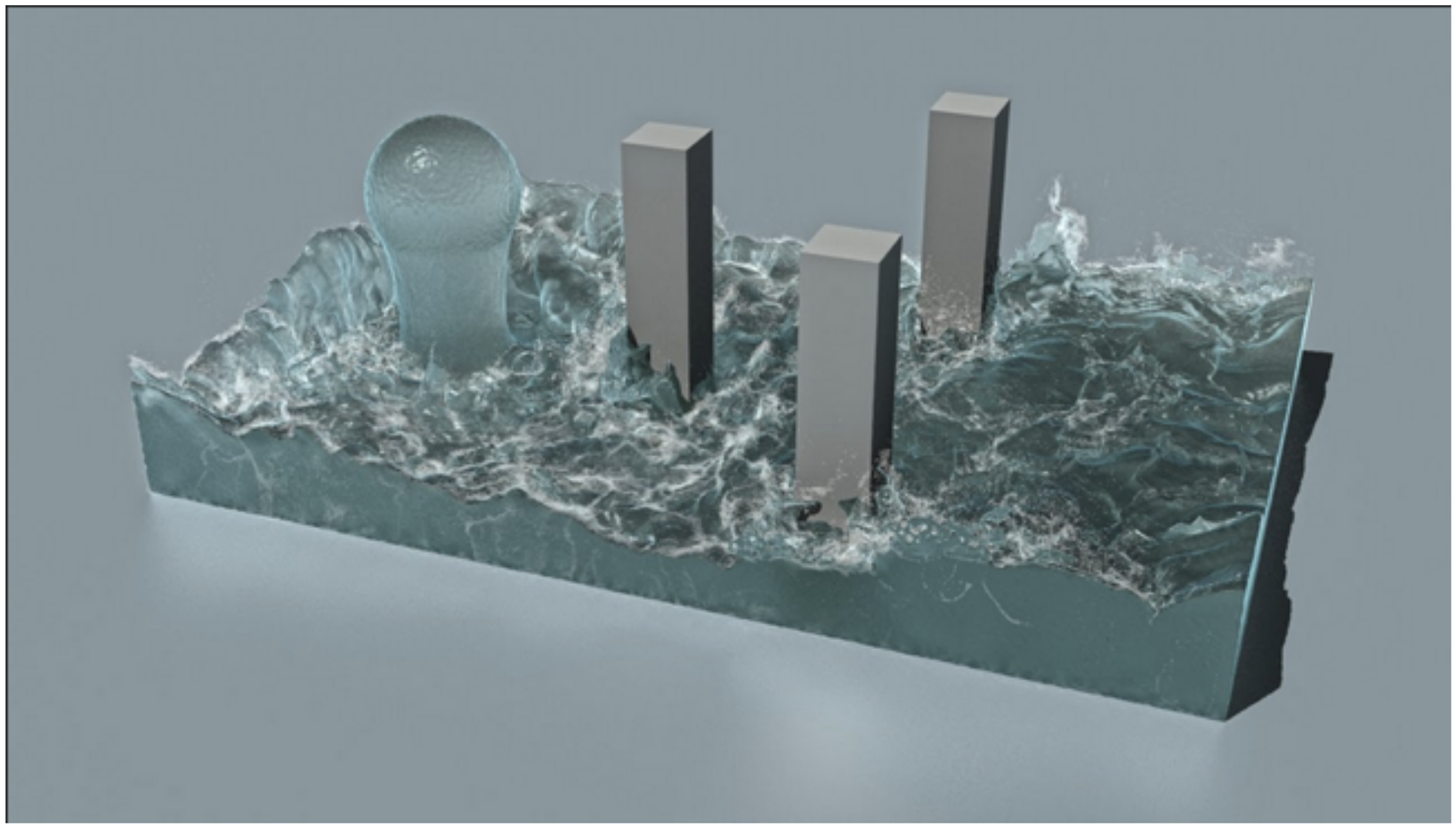
Figure attr: Valentin Haenel

Multithreading & SIMD
at work!

Accelerating I/O With Blosc



Blosc In OpenVDB And Houdini



"Blosc compresses almost as well as ZLIB, but it is much faster"

–Release Notes for OpenVDB 3.0, maintained by DreamWorks Animation

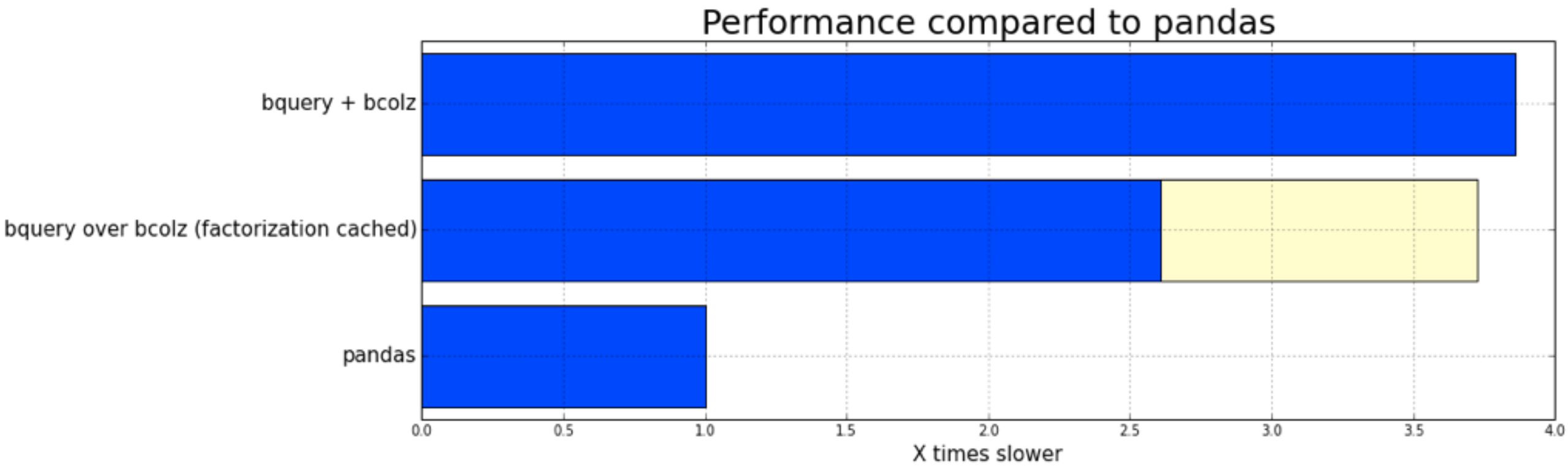
Some Projects Using bcolz

- Visualfabriq's bquery (out-of-core groupby's):
  https://github.com/visualfabriq/bquery

- Continuum's Blaze:
  http://blaze.pydata.org/

- Quantopian:
  http://quantopian.github.io/talks/NeedForSpeed/slides#/

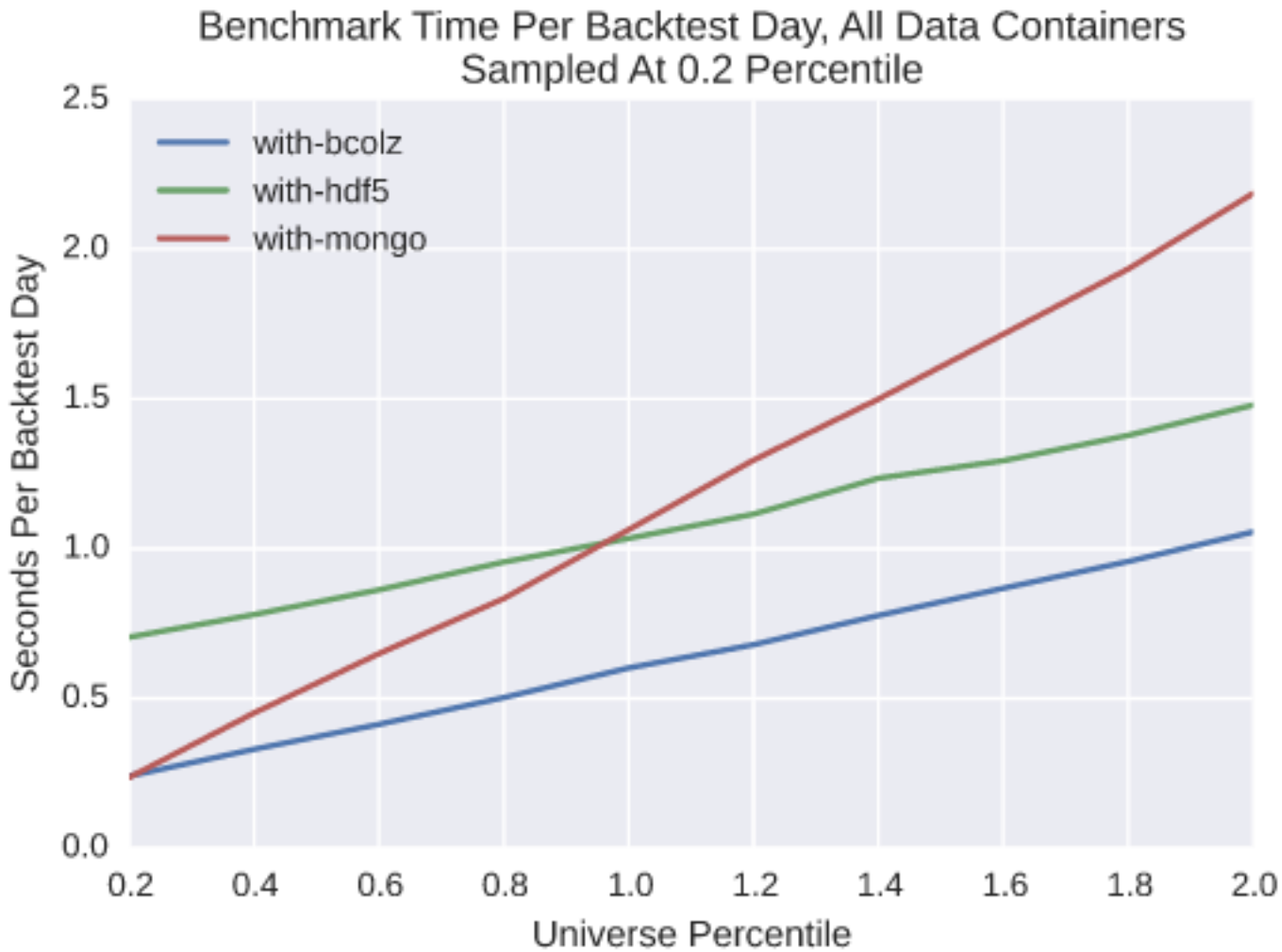
bquery - On-Disk GroupBy



In-memory (pandas) vs on-disk (bquery+bcolz) groupby

“Switching to bcolz enabled us to have a much better scalable architecture yet with near in-memory performance”
— Carst Vaartjes, co-founder visualfabriq

Quantopian's Use Case



"We set up a project to convert Quantopian's production and development infrastructure to use bcolz" — Eddie Herbert

Closing Notes

- If you need a data container that fits your needs, **look for already nice libraries out there** (NumPy, DyND, Pandas, PyTables, bcolz…)

- Pay **attention to hardware and software trends** and make informed decisions in your current developments (which, btw, will be deployed in the future :)

- **Performance is needed** for improving interactivity, so do not hesitate to optimize the hot spots in C if needed (via Cython or other means)

"It is change, continuing change, inevitable change, that is the dominant factor in Computer Sciences today. No sensible decision can be made any longer without taking into account not only the computer as it is, but the computer as it will be."

— Based on a quote by *Isaac Asimov*

Thank You!