It´s The Memory, Stupid!

Understanding the Memory Hierarchy for Better Data Management

Francesc Alted
Software Architect

CodeJam 2014, Jülich Supercomputing Centre
January 26, 2014

About Continuum Analytics

- Develop **new ways** on how data is stored, computed, and visualized.
- Provide open technologies for **data integration** on a **massive scale**.
- Provide **software tools**, **training**, and **integration/consulting services** to corporate, government, and educational clients worldwide.

Main Software Projects

- **Anaconda**: Easy deployment of Python packages
- **Bokeh**: Interactive visualization and analysis for the web and in Python
- **Blaze**: Out-of-core and distributed data computation & store (NG NumPy)
- **Wakari**: Sharing Python environments in the web (Python for teams)

Overview

- The Era of 'Big Data'
- A small exercise on query performance
- The Starving CPU problem
- Efficient computing: numexpr and Numba
- Optimal Containers for Big Data

"There were 5 exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days, and the pace is increasing."

— Erich Schmidt, Google CEO,
Techonomy Conference, August 4, 2010

The Dawn of 'Big Data'

Interactivity and Big Data

- Interactivity is crucial for handling data
- Interactivity **and performance** are crucial for handling **Big Data**

Python and 'Big Data'

- Python is an interpreted language and hence, it offers interactivity
- Myth: "Python is slow, so why on hell are you going to use it for Big Data?"
- Answer: Python has access to an incredibly powerful range of libraries that boost its performance far beyond your expectations
- ...and during this talk I will prove it…

NumPy: A Standard 'De Facto' Container

PyOpenCL

NVIDIA CUDA

PyCUDA

scikits-image
image processing in python

SM StatsModels
Statistics in Python

pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

matplotlib

sage

NumPy

OpenCV

RPy

PyTables

SciPy

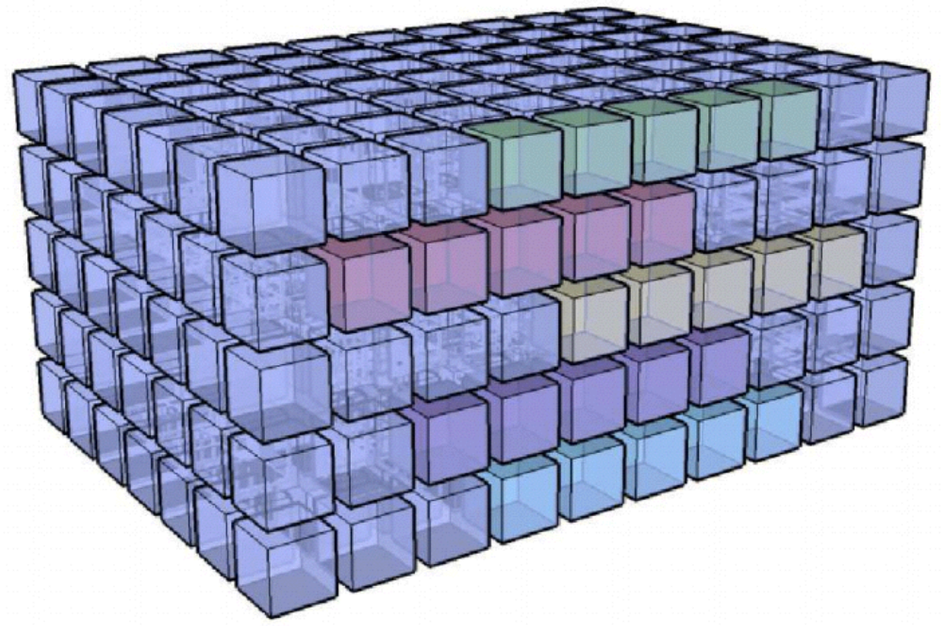
Mayavi
an ETS project

scikits
learn
machine learning in Python

spyder

CONTINUUM
ANALYTICS

Operating with NumPy



- array[2]; array[1,1:5, :]; array[[3,6,10]]
- (array1**3 / array2) - sin(array3)
- numpy.dot(array1, array2): access to optimized BLAS (*GEMM) functions
- and much more...

Interlude:
Querying Big Tables

Exercise

- We are starting to see datasets with a huge number of rows, and more importantly, **columns**
- Let's do an experiment querying such a large table by using several classic methods (and a new one :)

The Table & The Query

- Suppose that our table is something like one million rows and 100 columns (yes, we are talking about big data)
- The query is like:

```
(f2>.9) and ((f8>.3) and (f8<.4))
```

- Selectivity is about 1%

Times for querying (seconds)

Database (in-memory)	(create)	(query)
NumPy	23.4	1.97
sqlite3	454	28.9
sqlite3 (index)	140	2.87
BLZ	22.0	0.081

Run on Mac OSX, Core2 Duo @ 2.13 GHz

Open Questions

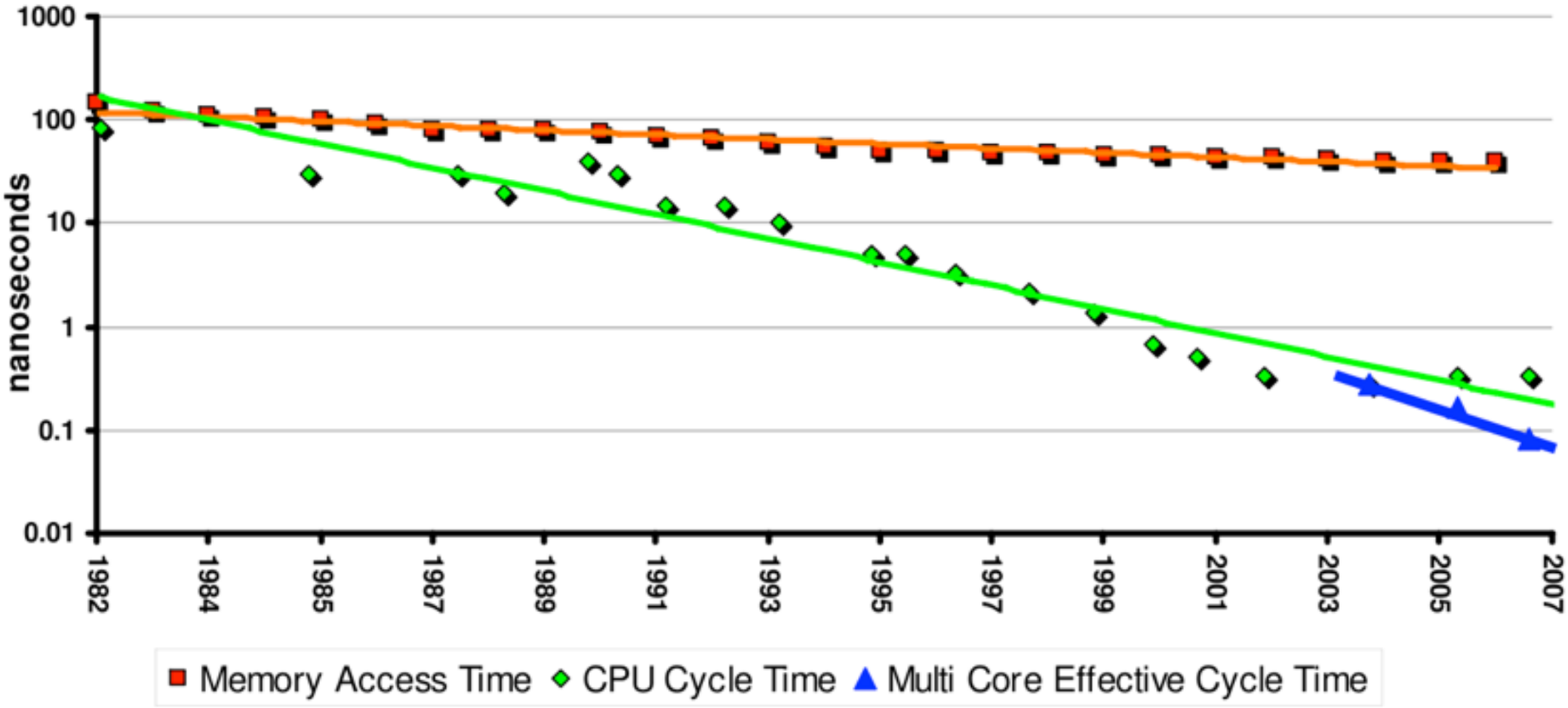
- If all the query engines store their data in-memory, and written in C, why they have such a large difference in performance?
- Why SQLite3 performance is so poor, even when using indexing?
- Why BLZ (essentially a data container on steroids) can be up to 25x faster than NumPy for querying?

"Across the industry, today's chips are largely able to execute code faster than we can feed them with instructions and data."

– Richard Sites, after his article
"It's The Memory, Stupid!",
Microprocessor Report, 10(10),1996

The Starving CPU Problem

Memory Access Time vs CPU Cycle Time



Book in 2009

MORGAN & CLAYPOOL PUBLISHERS

The Memory System

You Can't Avoid It,
You Can't Ignore It,
You Can't Fake It

Bruce Jacob

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE

Mark D. Hill, *Series Editor*

The Status of CPU Starvation in 2014

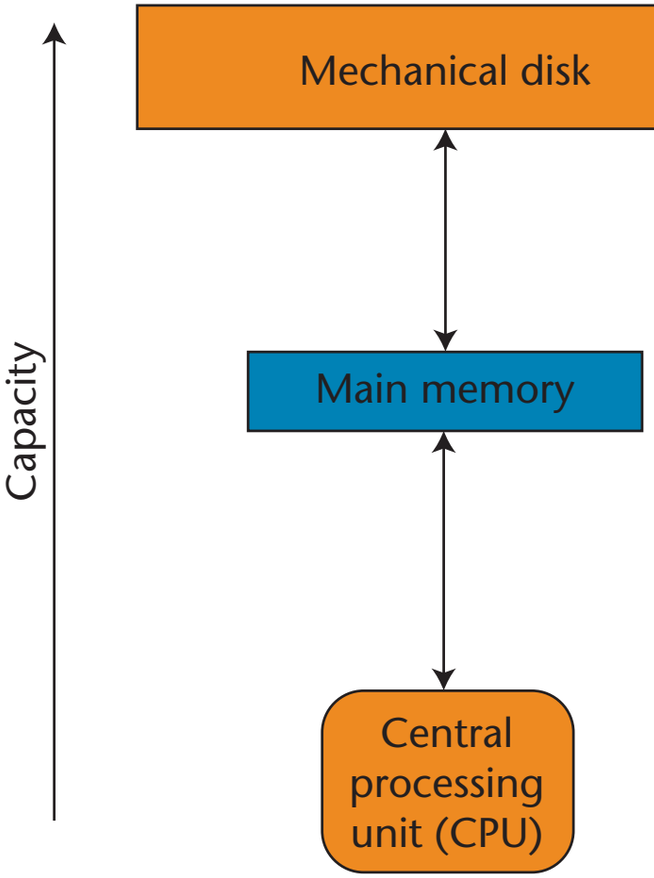
- Memory latency is much slower (between 250x and 1000x) than processors.
- Memory bandwidth is improving at a better rate than memory latency, but it is also slower than processors (between 30x and 100x).

CPU Caches to the Rescue

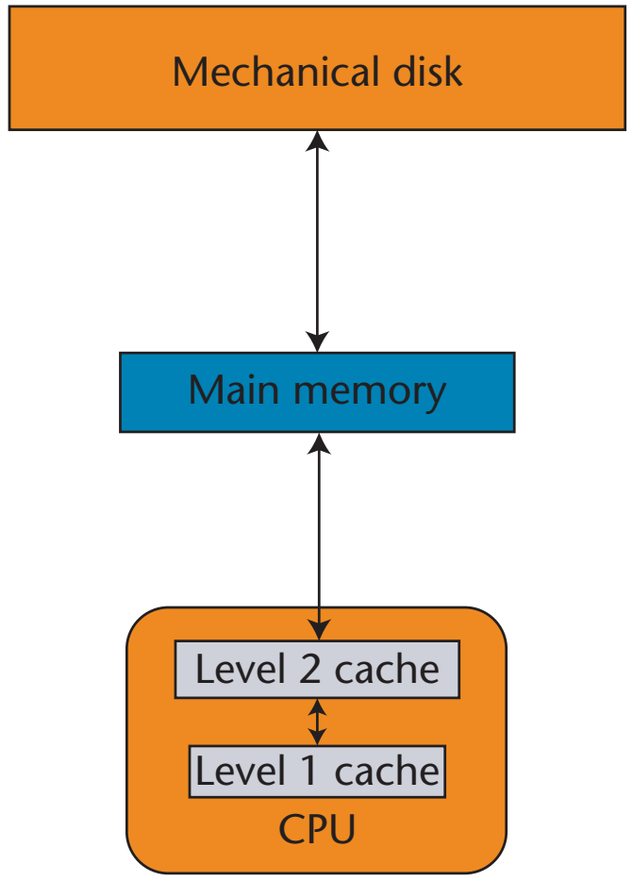
- CPU cache latency and throughput are much better than memory
- However: the faster they run the smaller they must be (because of heat dissipation problems)

CPU Cache Evolution

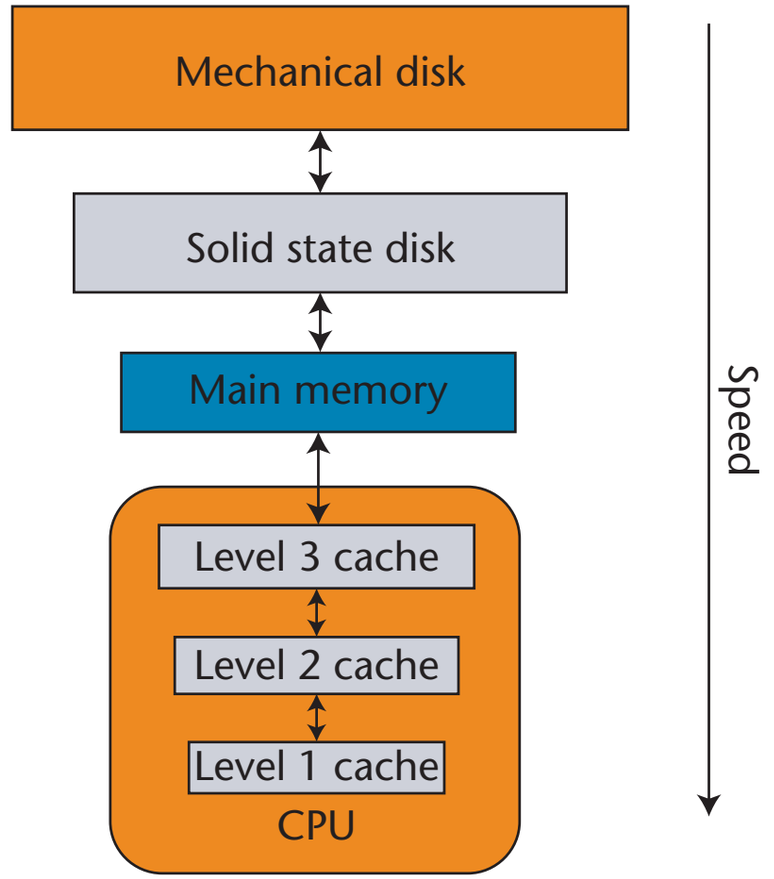
Up to end 80's



90's and 2000's



2010's

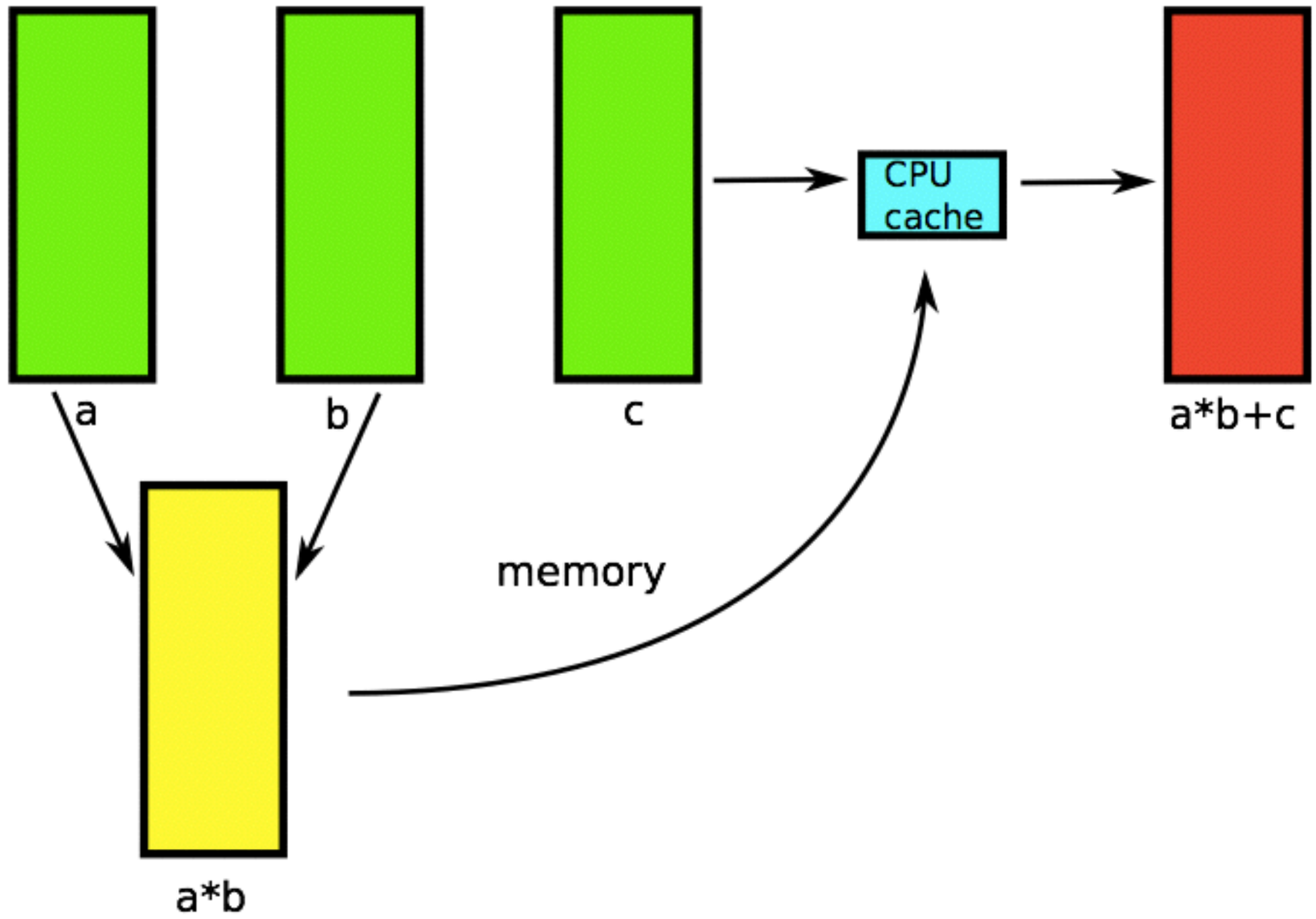


numexpr

- numexpr is a computational engine for NumPy that makes a sensible use of the memory hierarchy for better performance
- Other libraries normally use numexpr when it is time to accelerate large computations
- PyTables, pandas and BLZ can all leverage numexpr

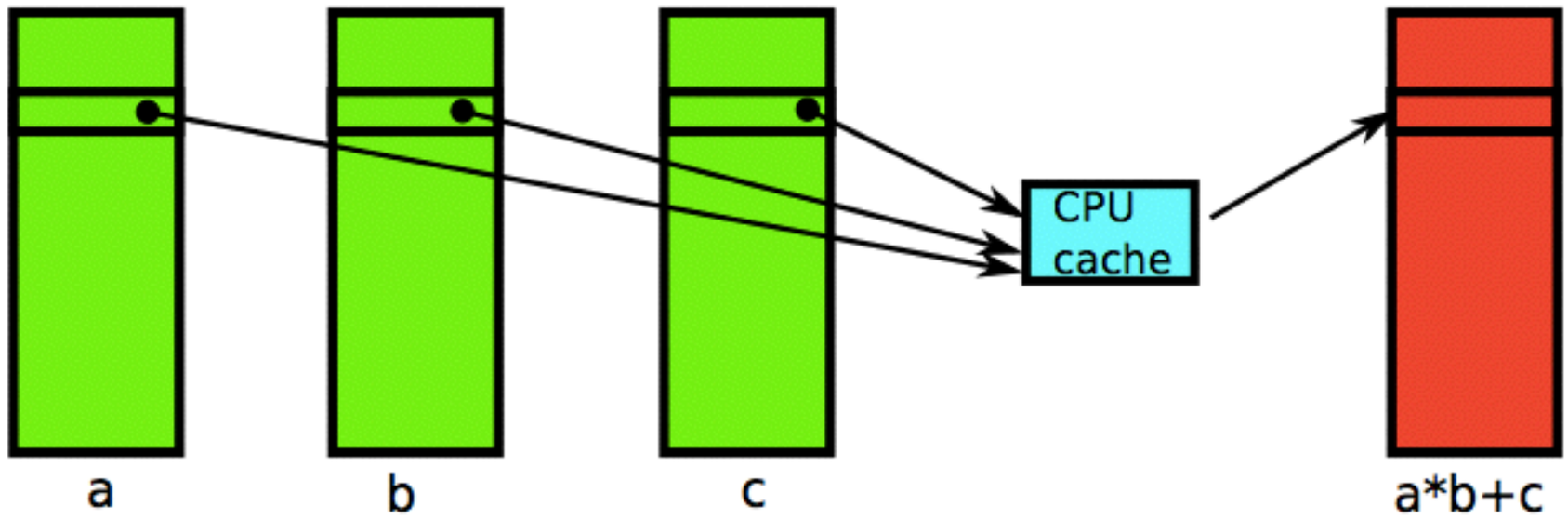
Computing with NumPy

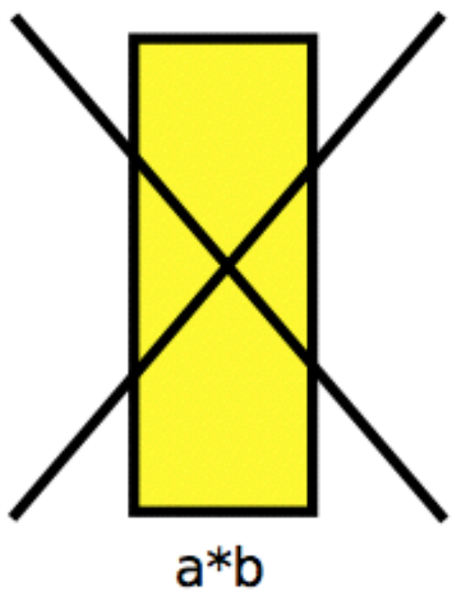
Temporaries go to memory



Computing with Numexpr

Temporaries stay in cache





memory

First Hint for Open Questions

- BLZ is using numexpr behind the scenes so as to do the computations needed for the query
- It creates less in-memory temporaries than NumPy, and CPUs can do more work

Second Interlude
Beyond numexpr: Numba

Numexpr Limitations

- Numexpr only implements element-wise operations, i.e. 'a*b' is evaluated as:

```
for i in range(N):
    c[i] = a[i] * b[i]
```

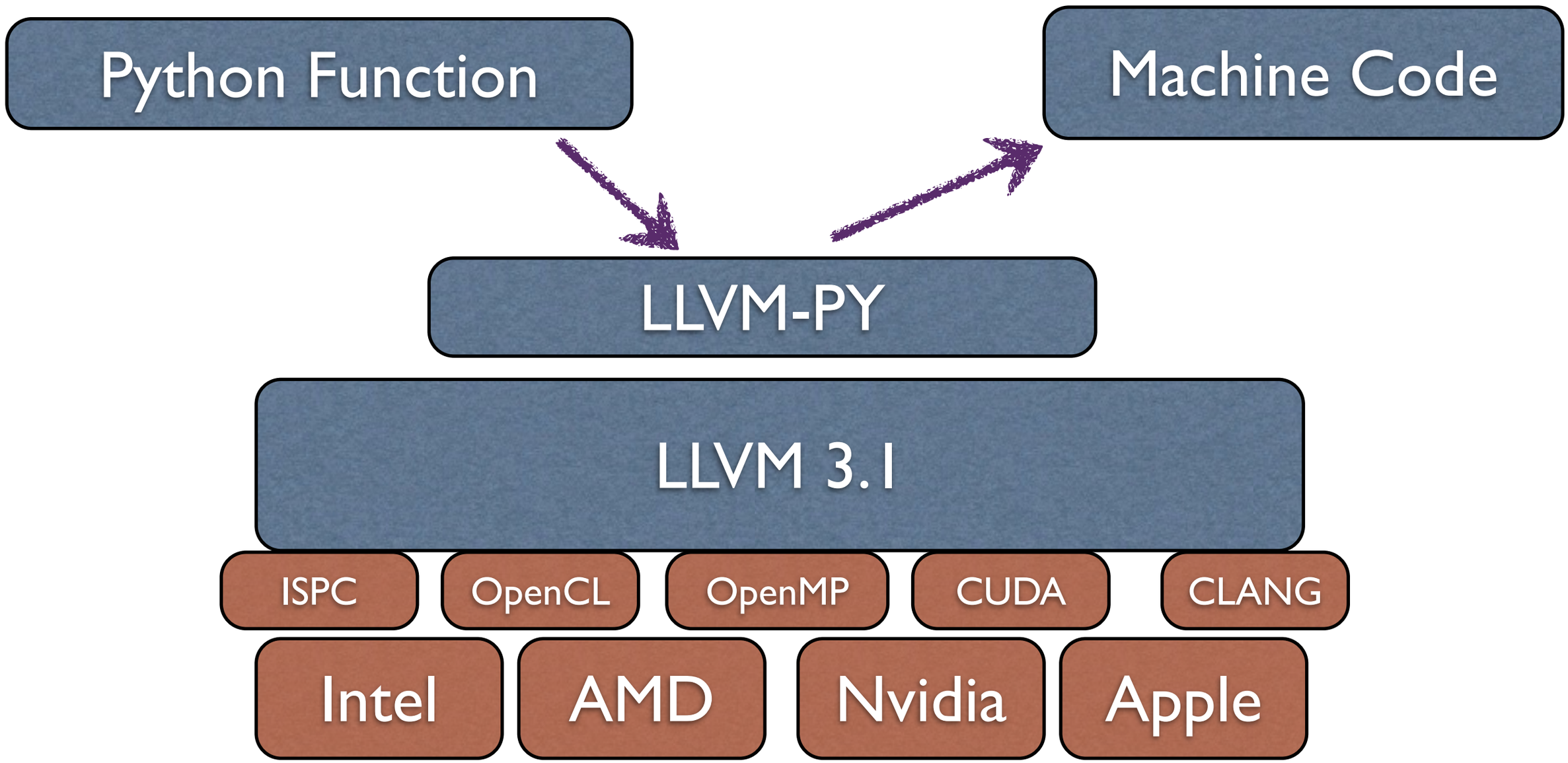
- In particular, it cannot deal with things like:

```
for i in range(N):
    c[i] = a[i-1] + a[i] * b[i]
```

Numba: Overcoming numexpr Limitations

- Numba is a JIT that can translate a subset of the Python language into machine code
- It uses LLVM infrastructure behind the scenes
- Can achieve similar or better performance than numexpr, but with more flexibility

How Numba Works



Numba Example: Computing a Polynomial

```
import numpy as np
import numba as nb

N = 10*1000*1000

x = np.linspace(-1, 1, N)
y = np.empty(N, dtype=np.float64)

@nb.autojit
def poly(x, y):
    for i in range(N):
        # y[i] = 0.25*x[i]**3 + 0.75*x[i]**2 + 1.5*x[i] - 2
        y[i] = ((0.25*x[i] + 0.75)*x[i] + 1.5)*x[i] - 2

poly(x, y)  # run through Numba!
```

Times for Computing (sec.)

Poly version	(I)	(II)
Numpy	1.086	0.505
numexpr	0.108	0.096
Numba	0.055	0.054
Pure C, OpenMP	0.215	0.054

Run on Mac OSX, Core2 Duo @ 2.13 GHz

Numba: LLVM for Python

Python code can reach C speed without having to program in C itself

(and without losing interactivity!)

Numba is Powering Blaze

- Blaze is Continuum Analytics' on-going effort to provide efficient computation for out-of-core and distributed data.

- Blaze is using a series of technologies for this: DyND, Numba, BLZ

- Numba is at the computational core of Blaze

FlyPy Has Born (Numba in 2014)

- Complete reimplementation of Numba (i.e. different computing engine)
- More powerful type system than Numba
- High Performance Computing for High Level Language (Python)

FlyPy Highlights

- SIMD support
- Fused generators (restricted use)
- Garbage collection
- Efficient user-defined abstraction
- Shared nothing architecture with optional safe sharing cache latency

If a datastore requires all data to fit in memory, it isn't big data

-- Alex Gaynor (in twitter)

Optimal Containers for High Performance Computing

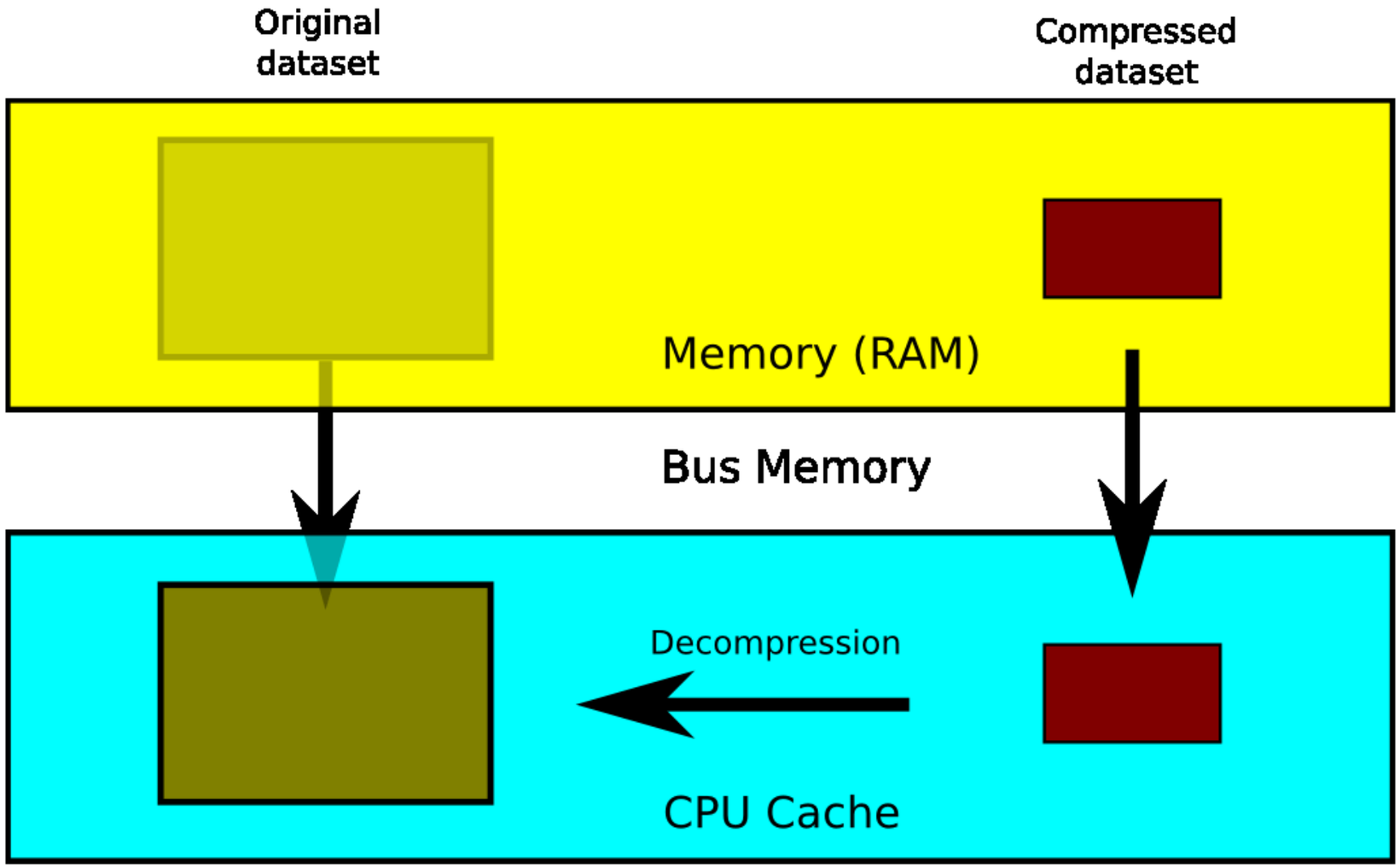
The Need for a Good Data Container

- Too many times we are too focused on computing as fast as possible
- But we have seen how important data access is
- BLZ is data container for in-memory or on-disk data that can be **compressed**

The Need for Compression

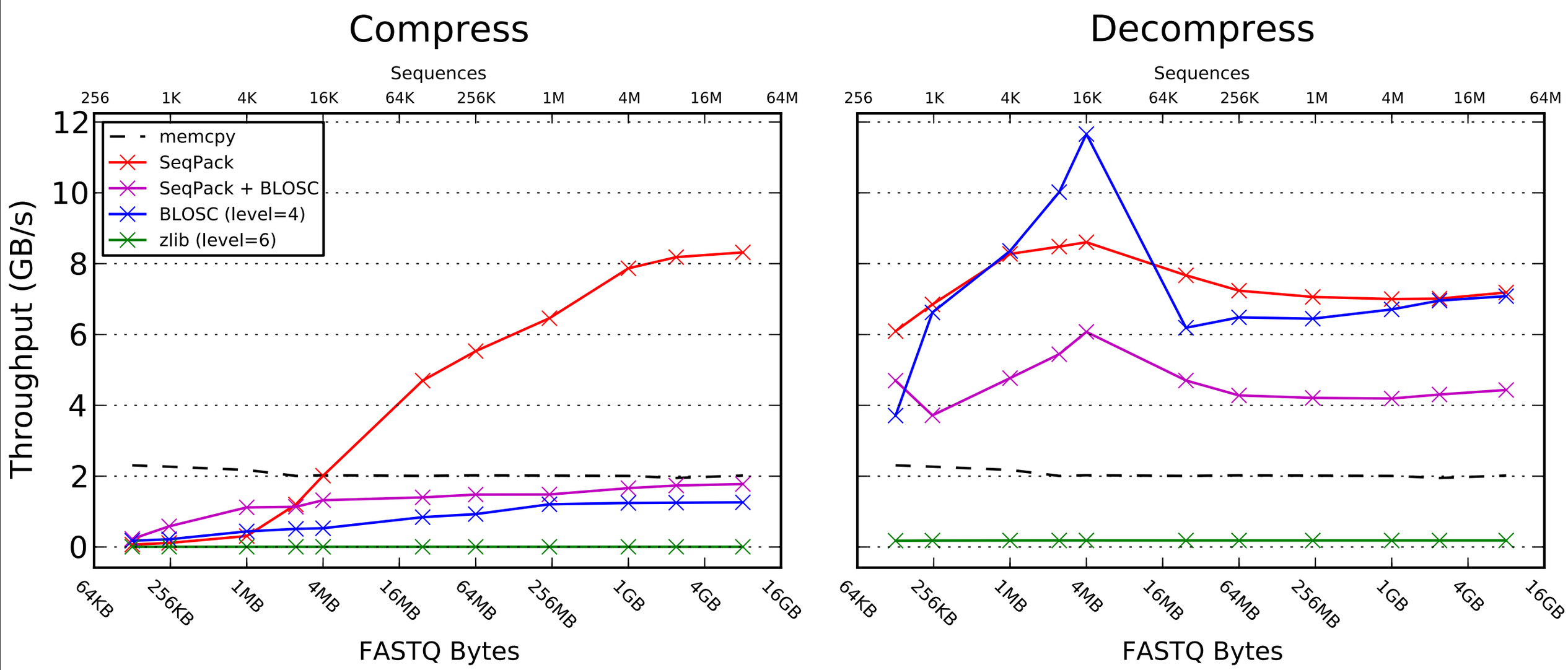
- Compression allows to store more data using the same storage capacity
- Sure, it uses more CPU time to compress/ decompress data
- **But, that actually means using more wall clock time?**

Blosc: (de)compressing faster than memcpy()



Transmission + decompression faster than direct transfer?

Example of How Blosc Accelerates Genomics I/O: SeqPack (backed by Blosc)

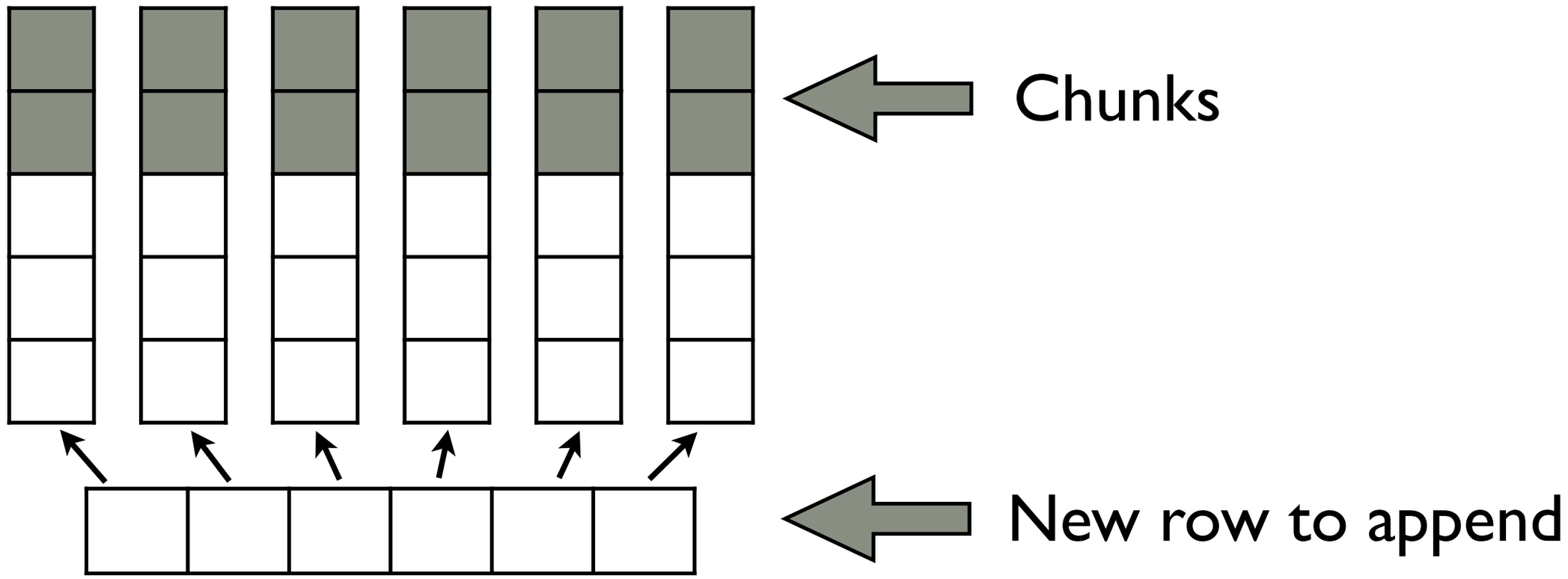


Source: Howison, M. (in press). High-throughput compression of FASTQ data with SeqDB. IEEE Transactions on Computational Biology and Bioinformatics.

Second Hint for Open Questions

- BLZ does make use of Blosc for dealing with compressed data transparently
- That means not only better storage usage, but frequently **better access time** to the data

The btable object in BLZ



- Columns are **contiguous** in memory
- Chunks follow **column** order
- Very efficient for **querying** (specially with a large number of columns)

And a Third Hint…

- BLZ, differently from NumPy or SQLite3, stores columns contiguously in-memory or on-disk
- Again, BLZ is making use of the principles of time and spatial locality for optimizing the access to memory
- Blaze can leverage BLZ for storage and improved I/O capabilities

Summary

- Python is a perfect match for Big Data
- Nowadays you should be aware of the memory system for getting good performance
- Choosing appropriate data containers is of the utmost importance when dealing with Big Data

References

- **Blaze**: http://blaze.pydata.org
- **BLZ**: http://blz.pydata.org
- **Blosc**: http://www.blosc.org
- **Numba**: http://numba.pydata.org
- **FlyPy**: https://github.com/ContinuumIO/flypy

"Across the industry, today's chips are largely able to execute code faster than we can feed them with instructions and data. There are no longer performance bottlenecks in the floating-point multiplier or in having only a single integer unit. The real design action is in memory subsystems— caches, buses, bandwidth, and latency."

"Over the coming decade, memory subsystem design will be the only important design issue for microprocessors."

– Richard Sites, after his article "It's The Memory, Stupid!", Microprocessor Report, 10(10),1996

Thank you!

CONTINUUM

ANALYTICS