Memory Bound Computing

Francesc Alted

Freelance Consultant

http://www.blosc.org/professional-services.html

Advanced Scientific Programming in Python
Reading, UK
September, 2016

"No sensible decision can be made any longer without taking into account not only the world as it is, but the world as it will be."

— Isaac Asimov

"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."

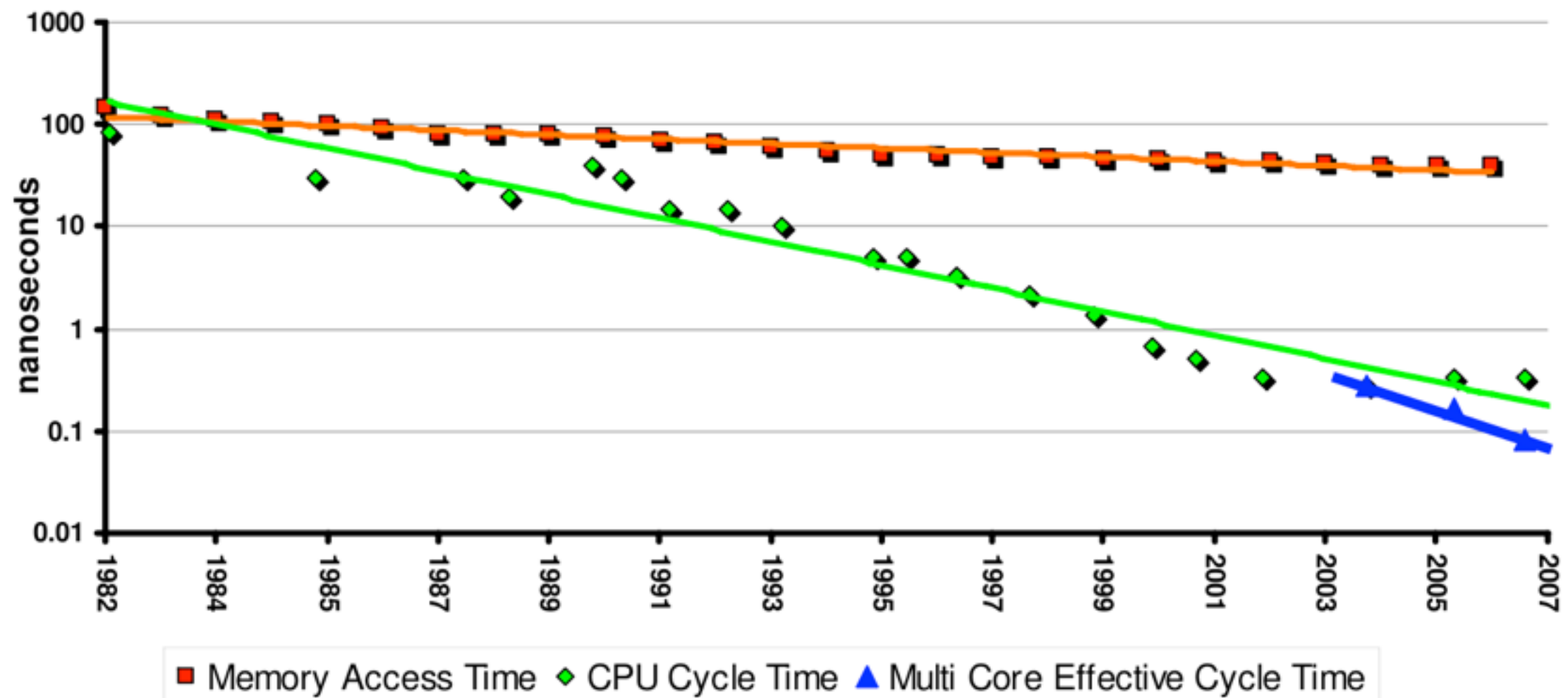— My own rephrasing

Overview

- The Starving CPU problem

- Recent trends in computer architecture

- Handling Big Data: storing and processing as much data as possible with your existing resources

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

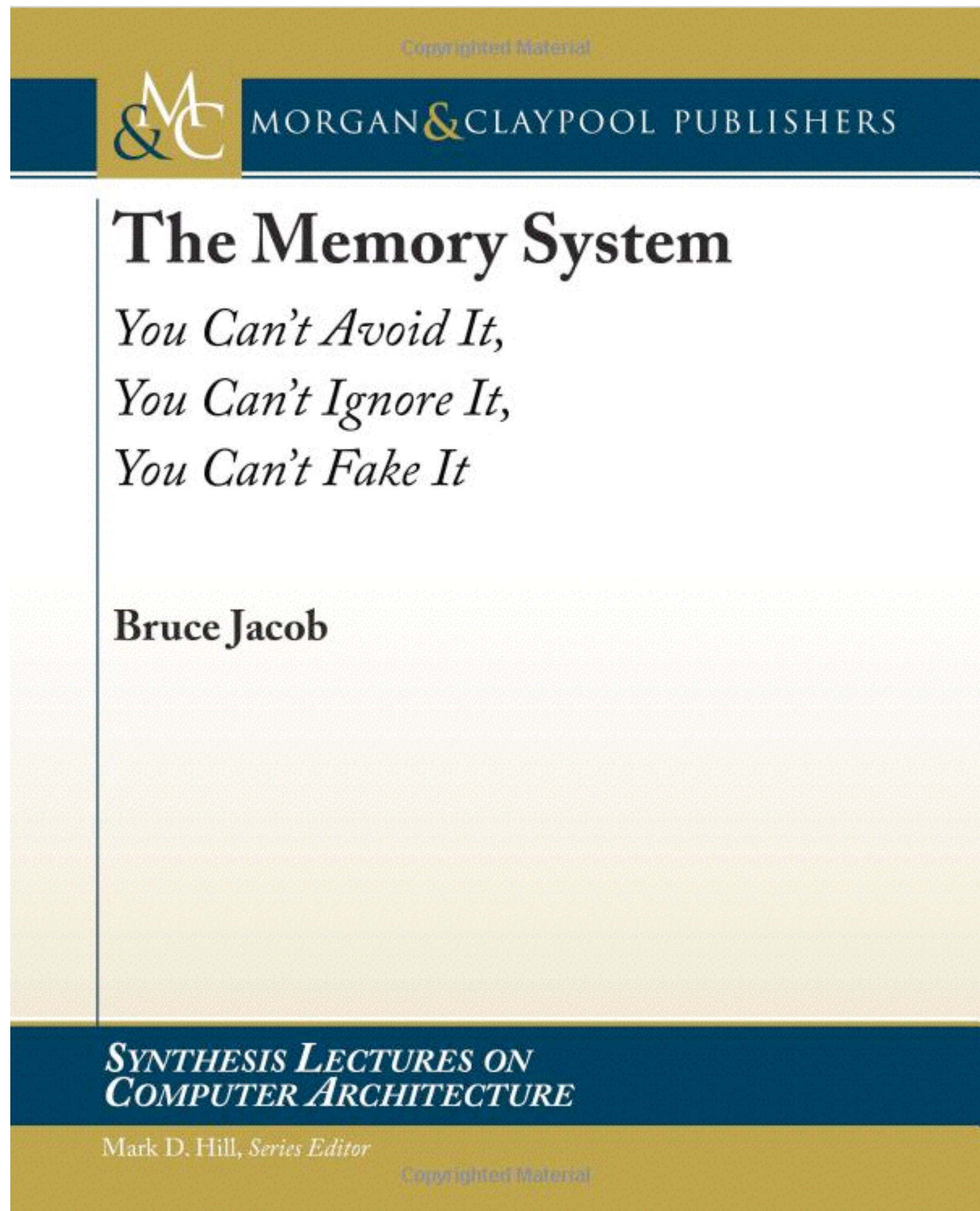— Richard Sites, after his article "It's The Memory, Stupid!", Microprocessor Report, 10(10),1996

The Starving CPU Problem

Memory Access Time vs CPU Cycle Time

The gap is wide and still opening!

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 2016

- 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).
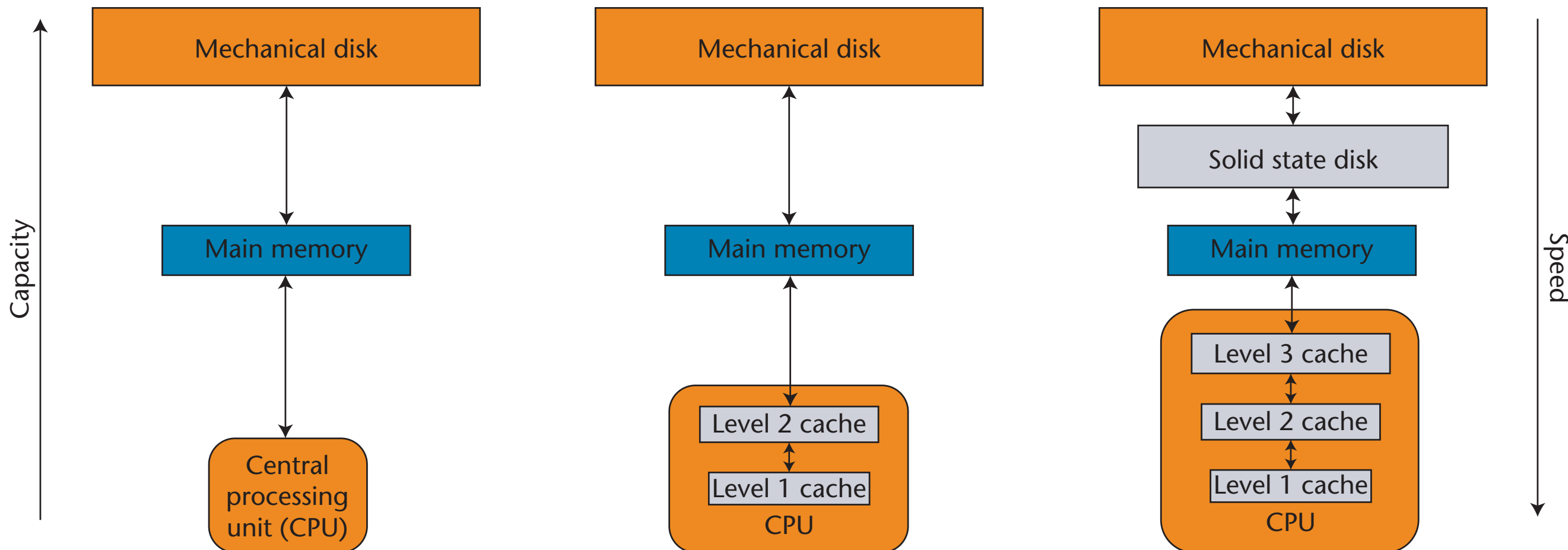
CPU Caches to the Rescue

- 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)

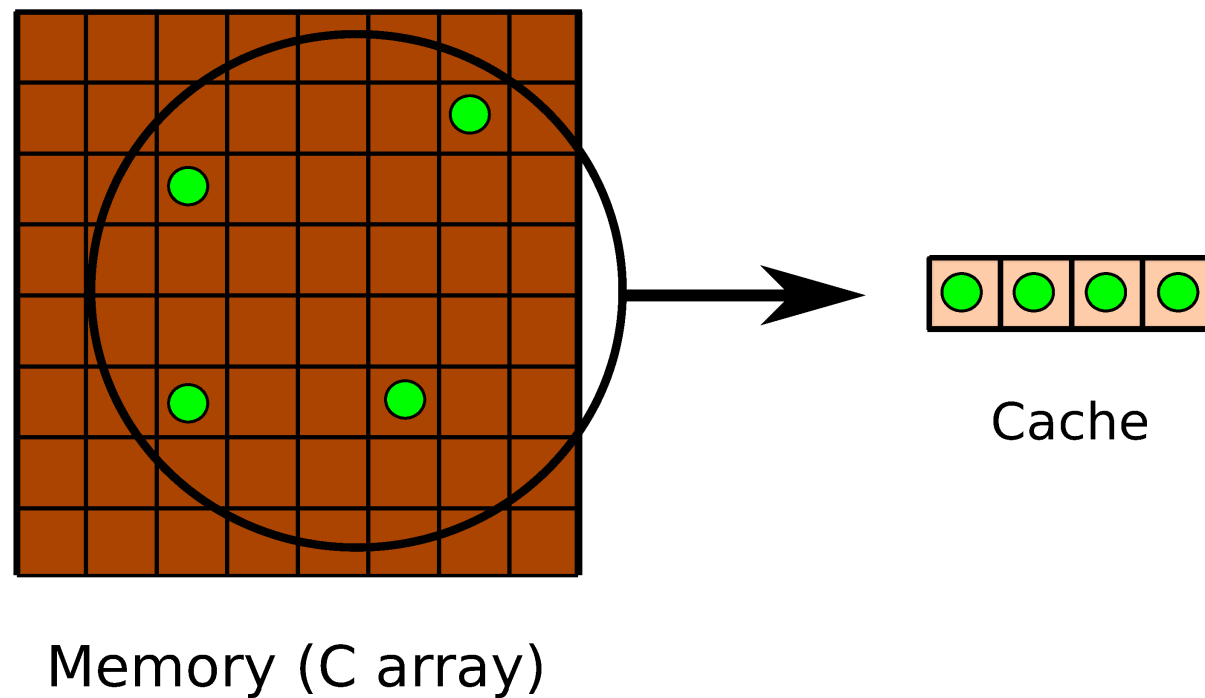# Computer Architecture Evolution

When CPU Caches Are Effective?

Mainly in a couple of scenarios:

- Time locality: when the dataset is reused

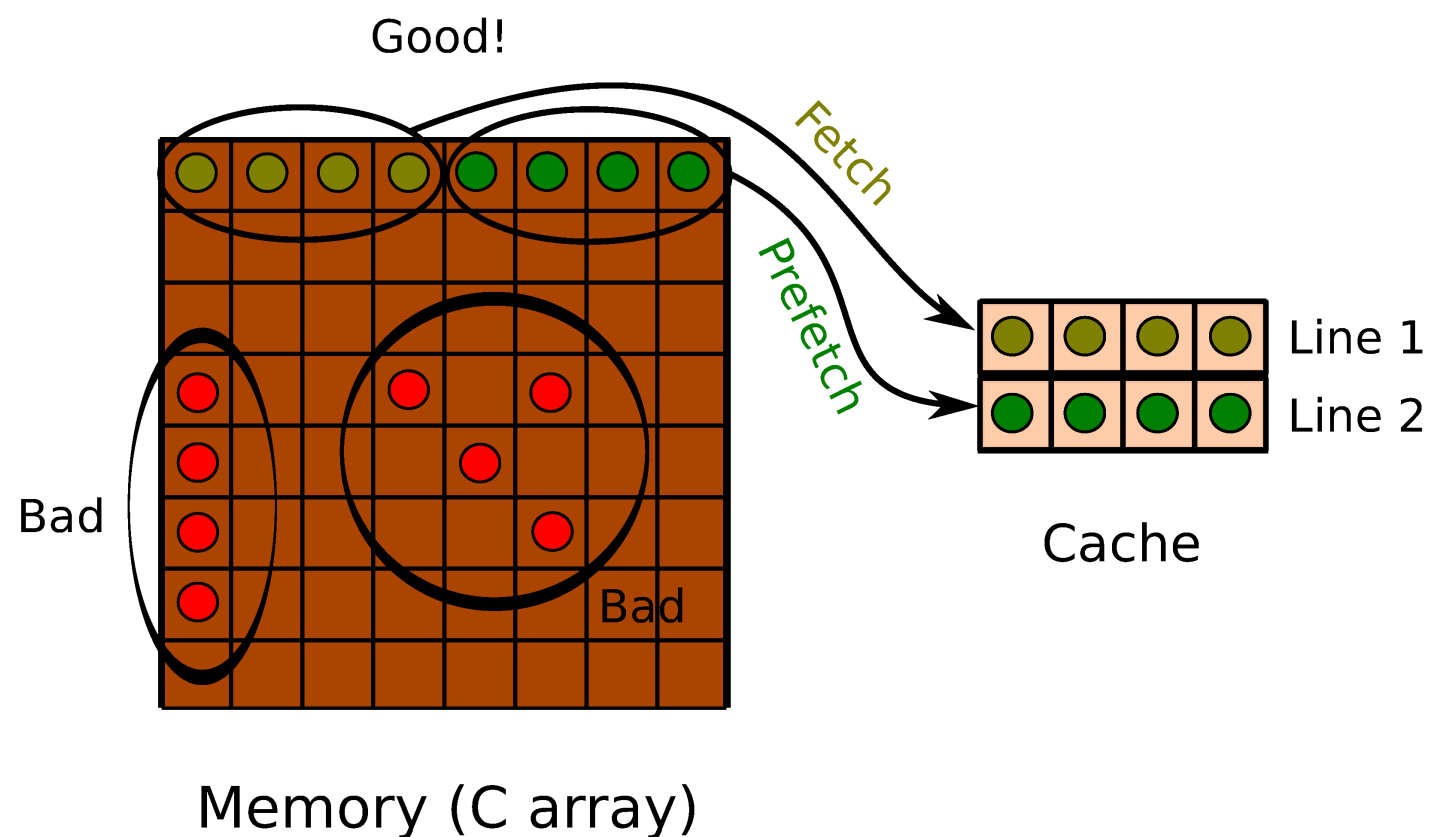- Spatial locality: when the dataset is accessed sequentially
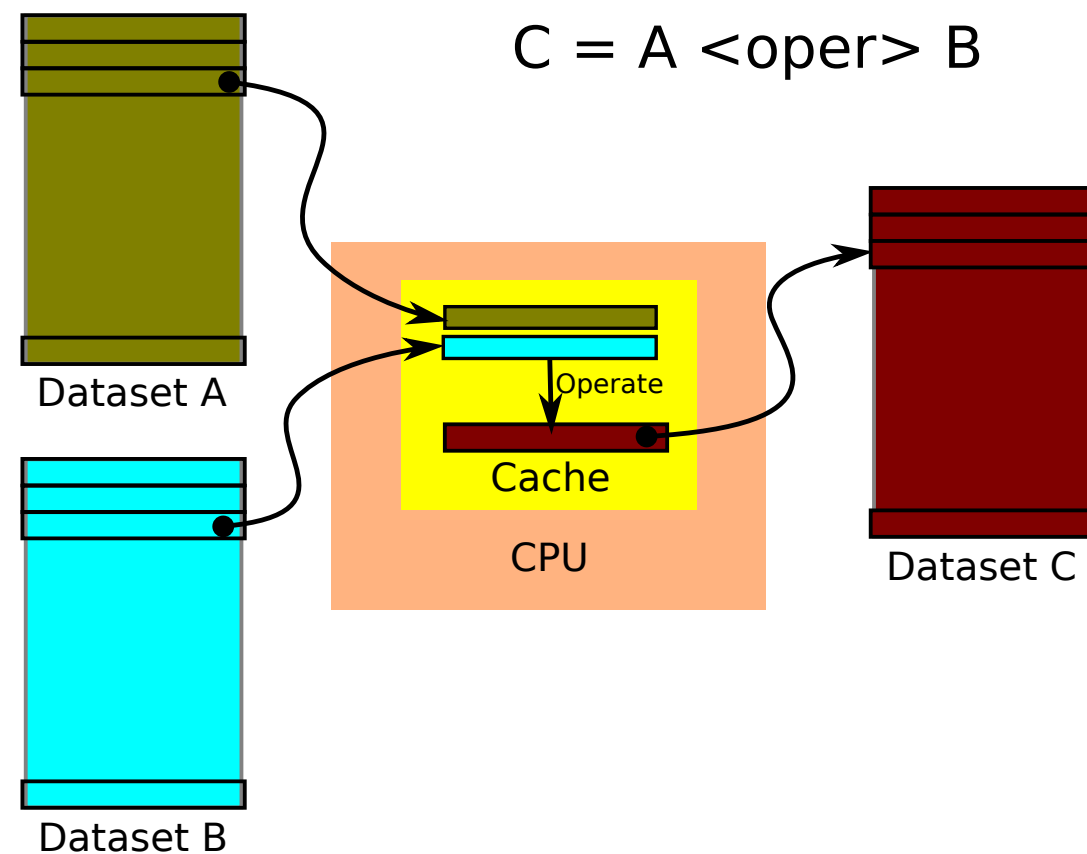
Parts of the dataset are reused

Memory (C array)

Cache

Spatial Locality

Dataset is accessed sequentially
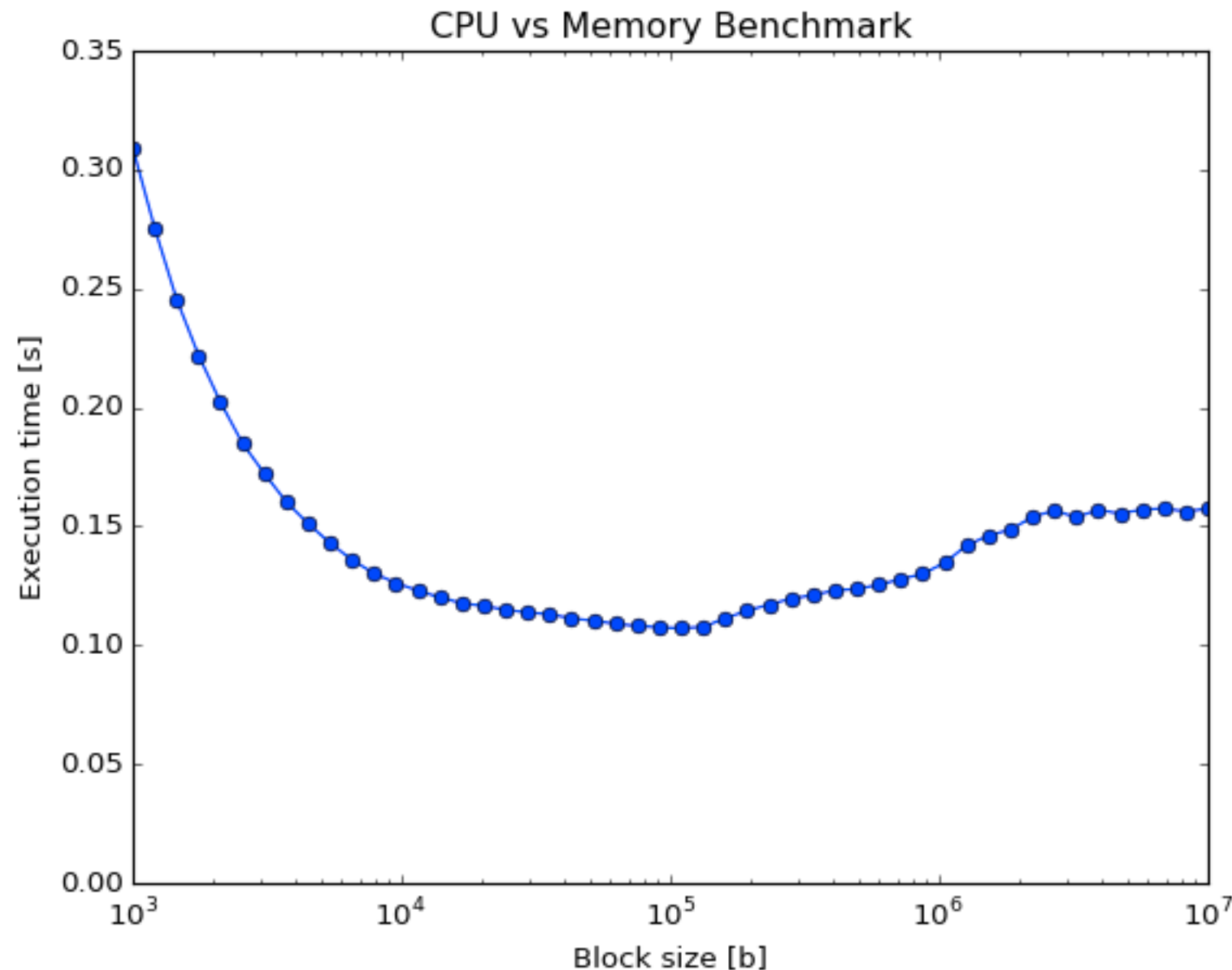

Good!

Fetch

Prefetch

Line 1

Line 2

Bad

Bad

Cache

Memory (C array)

ue

When accessing disk or memory, get a contiguous block that fits in CPU cache, operate upon it and reuse it as much as possible.

C = A <oper> B
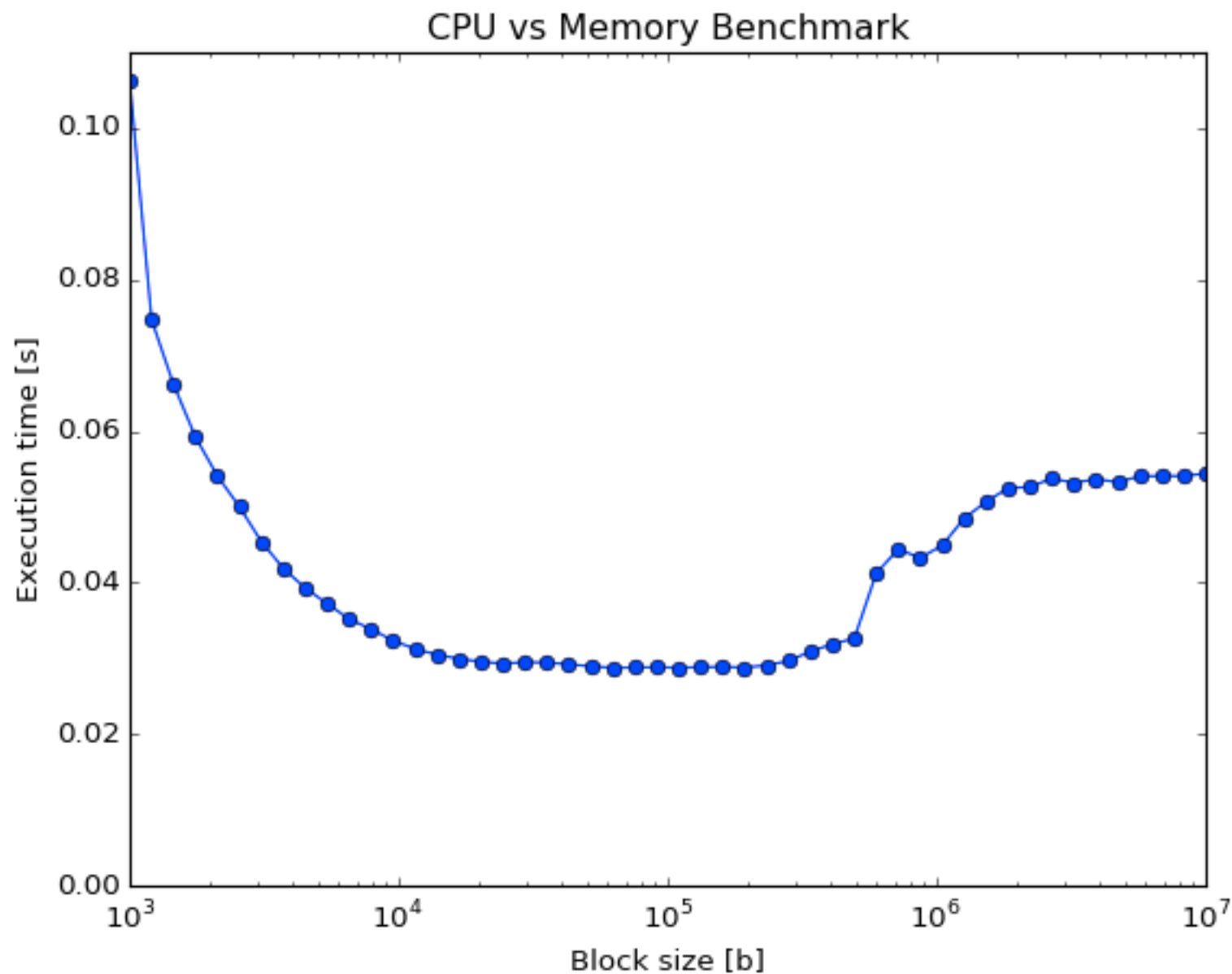
Dataset A

Operate

Cache

CPU

Dataset B

Dataset C

Use this extensively to leverage spatial and temporal localities

The Blocking Technique in Action (Exercise 0)

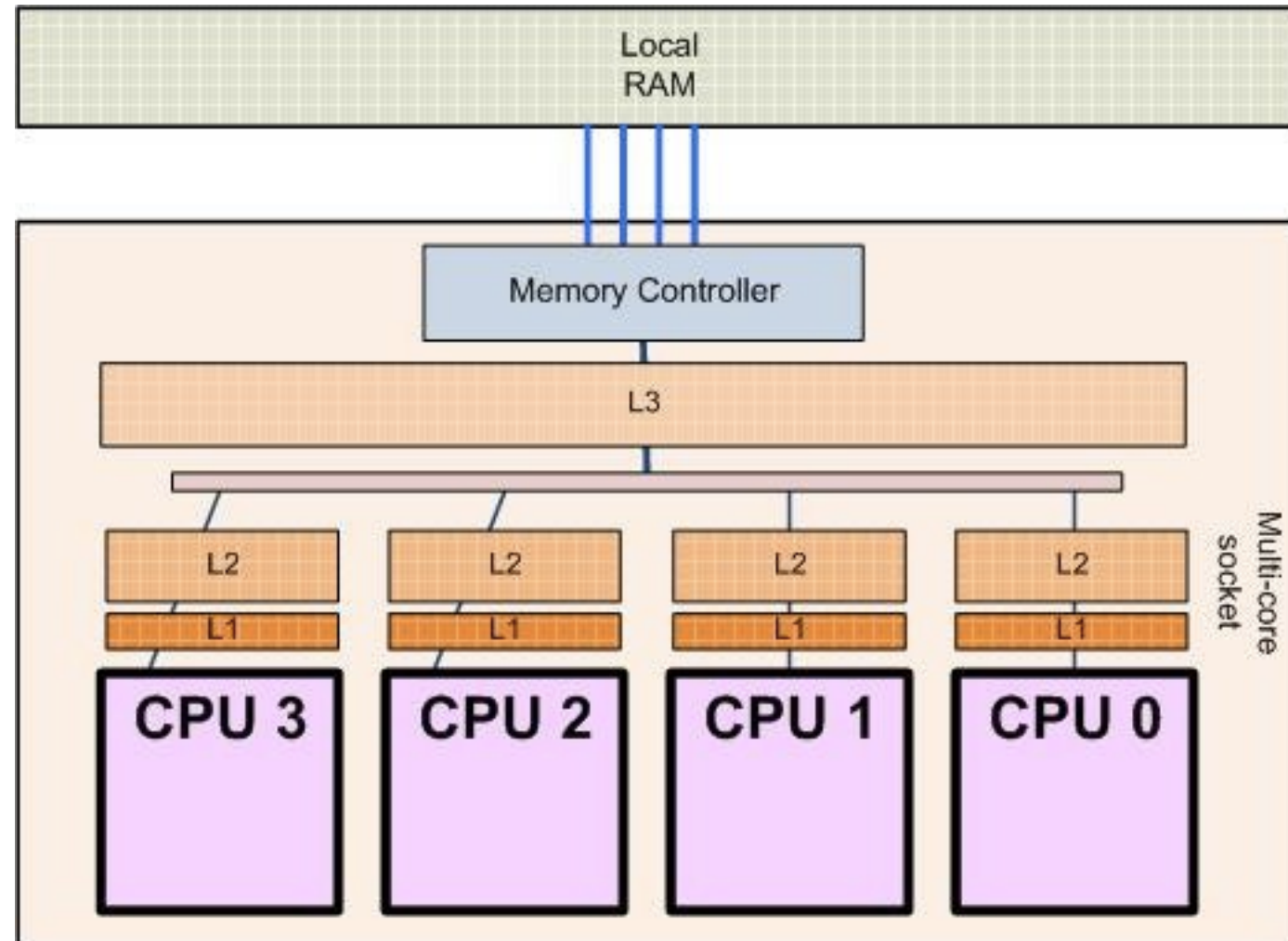CPU: Intel Xeon E312xx @ 2.00GHz (Sandy Bridge)

The Blocking Technique in Action (Exercise 0)

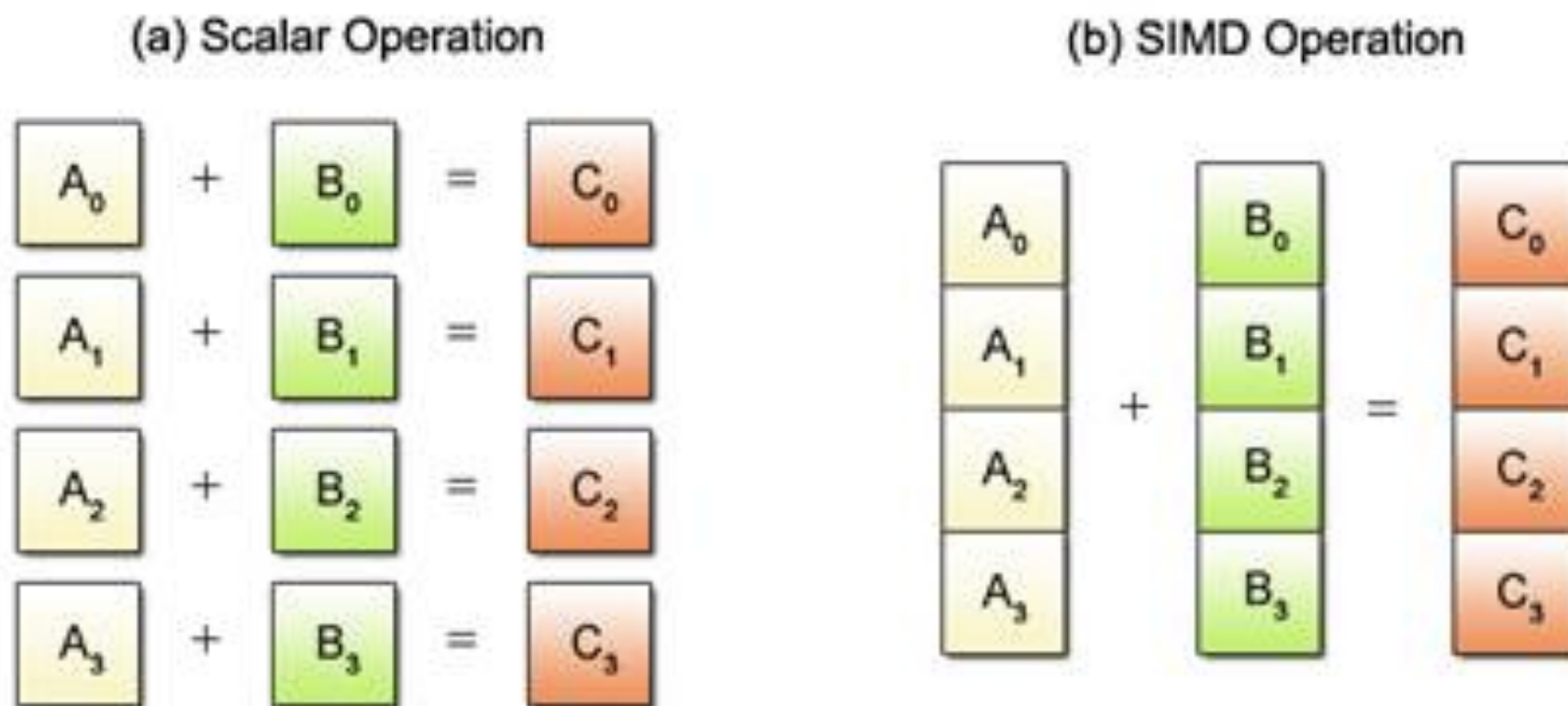CPU: Intel Xeon(R) CPU E3-1245 v5 @ 3.50GHz (Skylake)

Trends in Computer Architecture

We Are In A Multicore Age

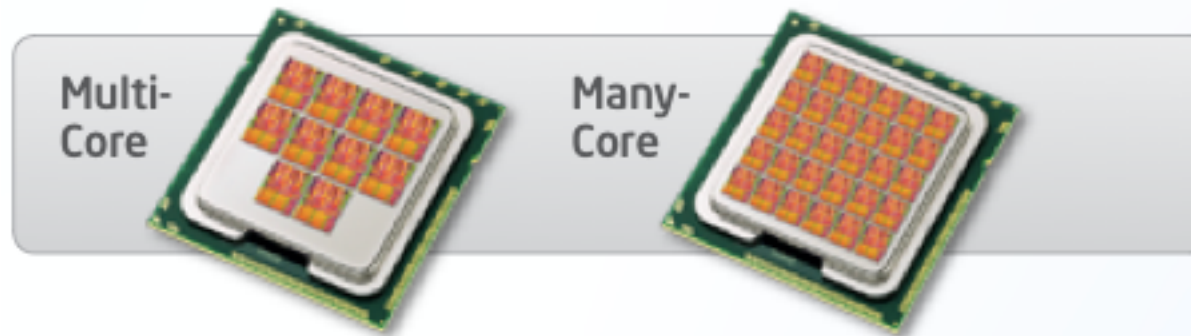- This requires special programming measures to leverage all its potential: threads, multiprocessing
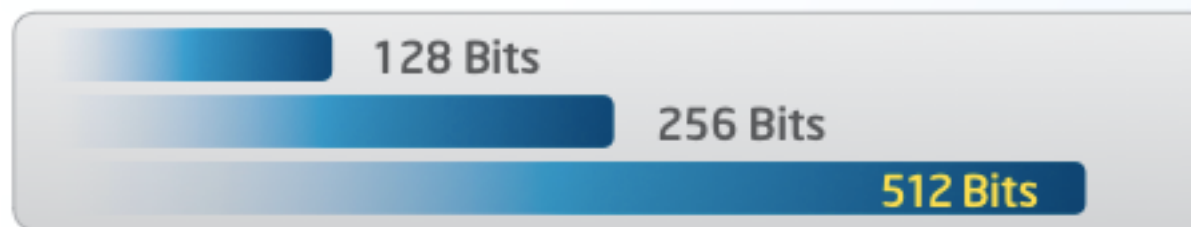
SIMD: Single Instruction, Multiple Data

(a) Scalar Operation

(b) SIMD Operation

More operations in the same CPU clock

Forthcoming Trends in CPU

More Cores

Multi-Core · Many-Core

Wider Vectors

128 Bits · 256 Bits · 512 Bits

CPU+GPU Integration

System Memory · Fabric and Memory Controller · CPU Cores · 3rd Party IP · GPU Cores · APU Chip
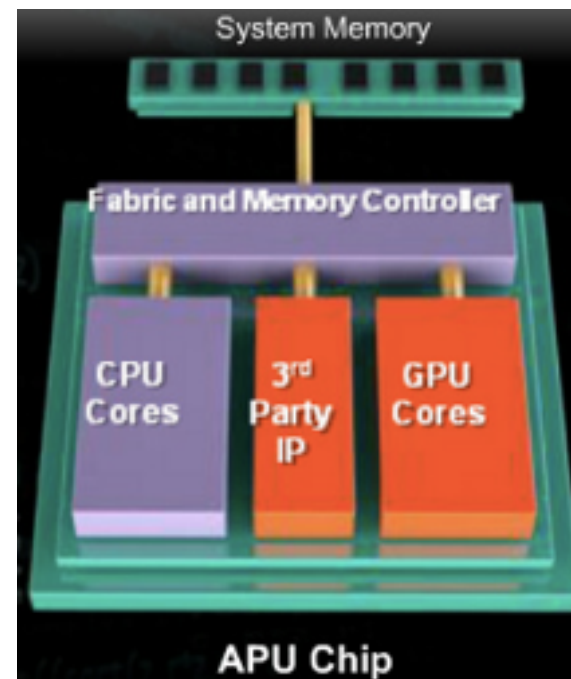
Hierarchy of Memory
By 2018 (Educated Guess)

HDD (persistent)

SSD SATA (persistent)

SSD PCIe (persistent)

XPoint (persistent)

RAM (addressable)

L4

L3

L2

L1   9 levels will be common!
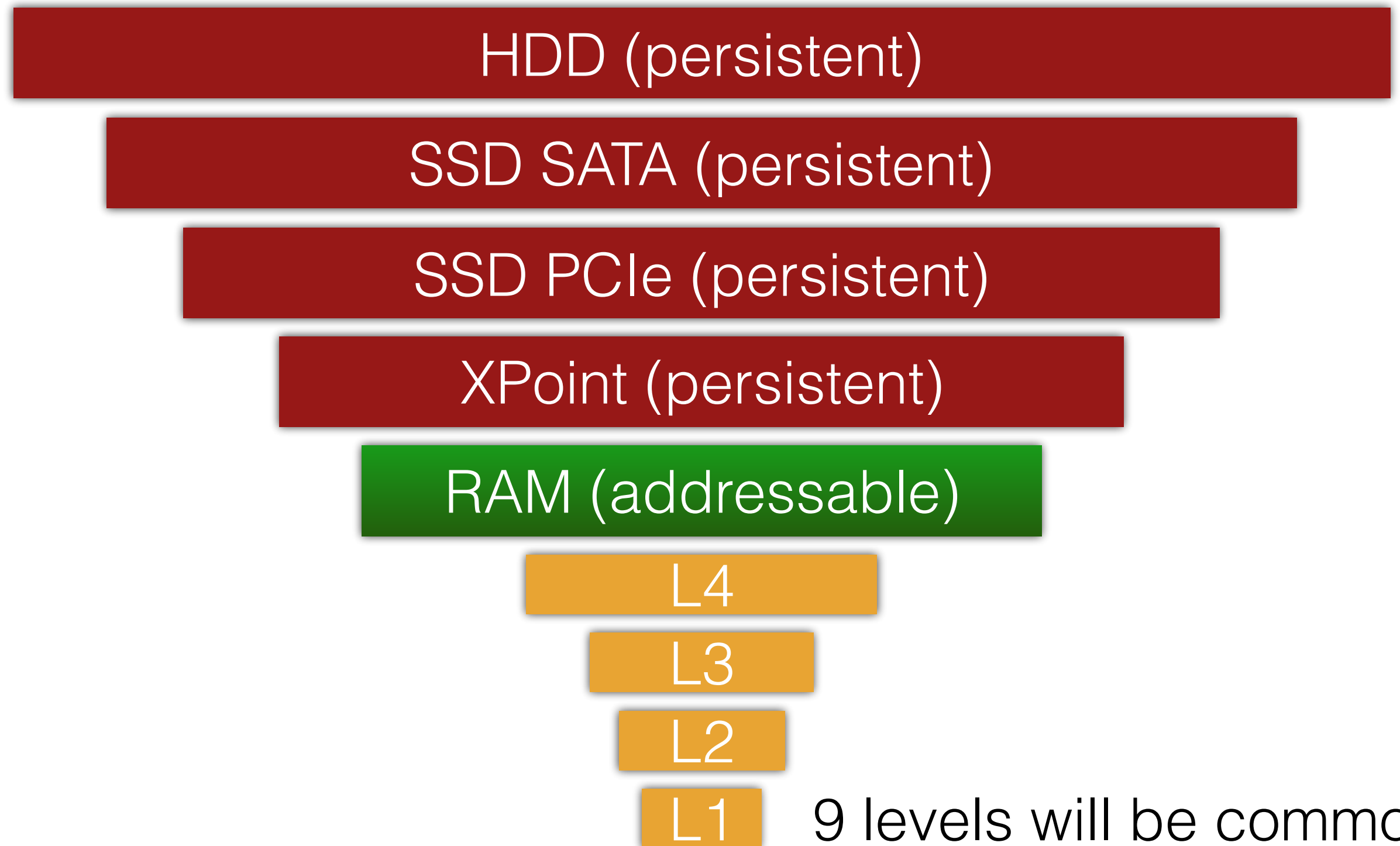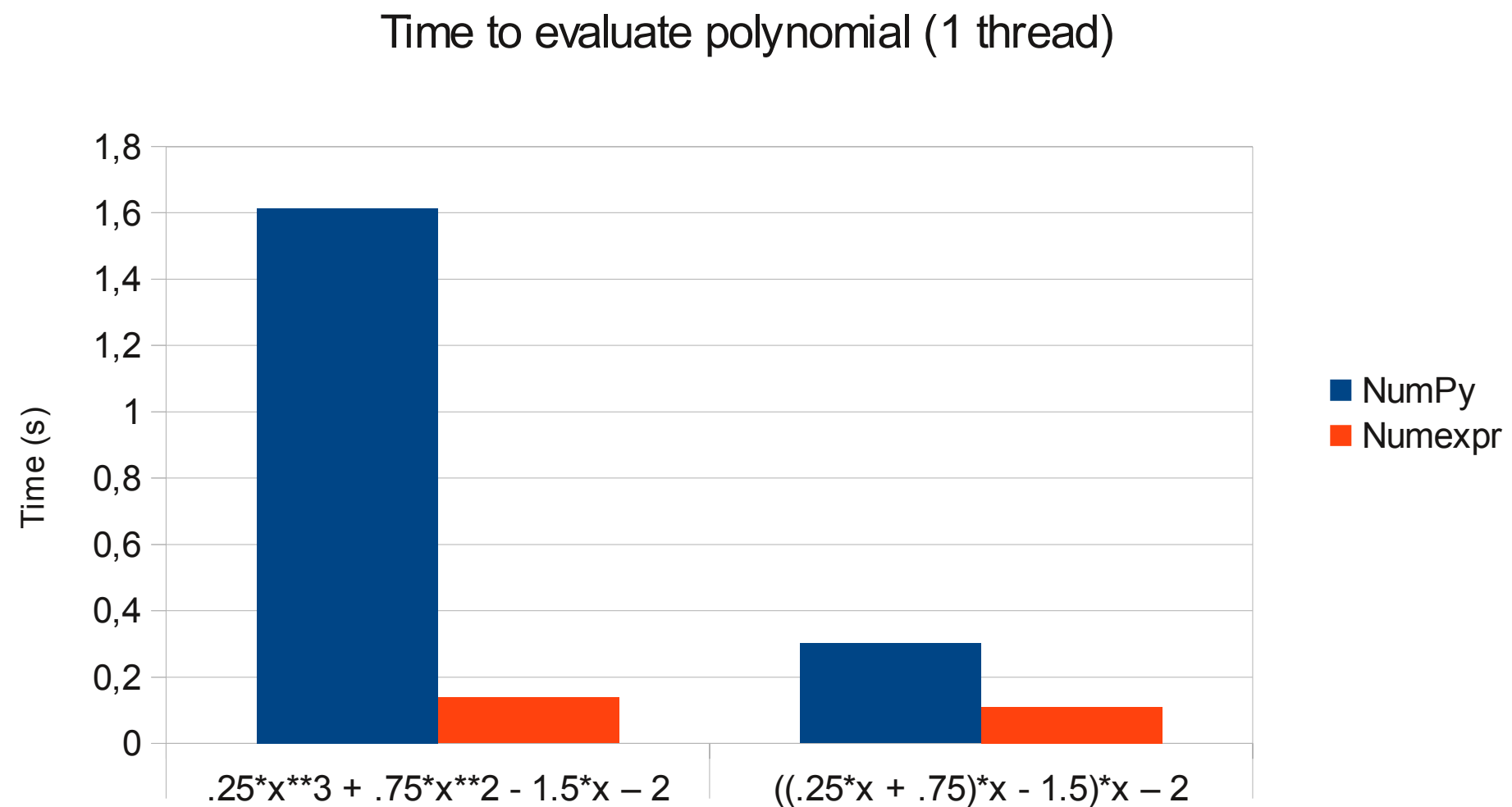
numexpr

- It is a computational engine for NumPy that makes a sensible use of the memory hierarchy for better performance

- It can use multi-core (via multi-threading) and SIMD (via Intel's MKL) for better CPU usage.

- PyTables, pandas and bcolz (among others) can all leverage numexpr automatically if installed

Computing with numexpr

Time to evaluate polynomial (1 thread)

Power Expansion

Numexpr expands expression:
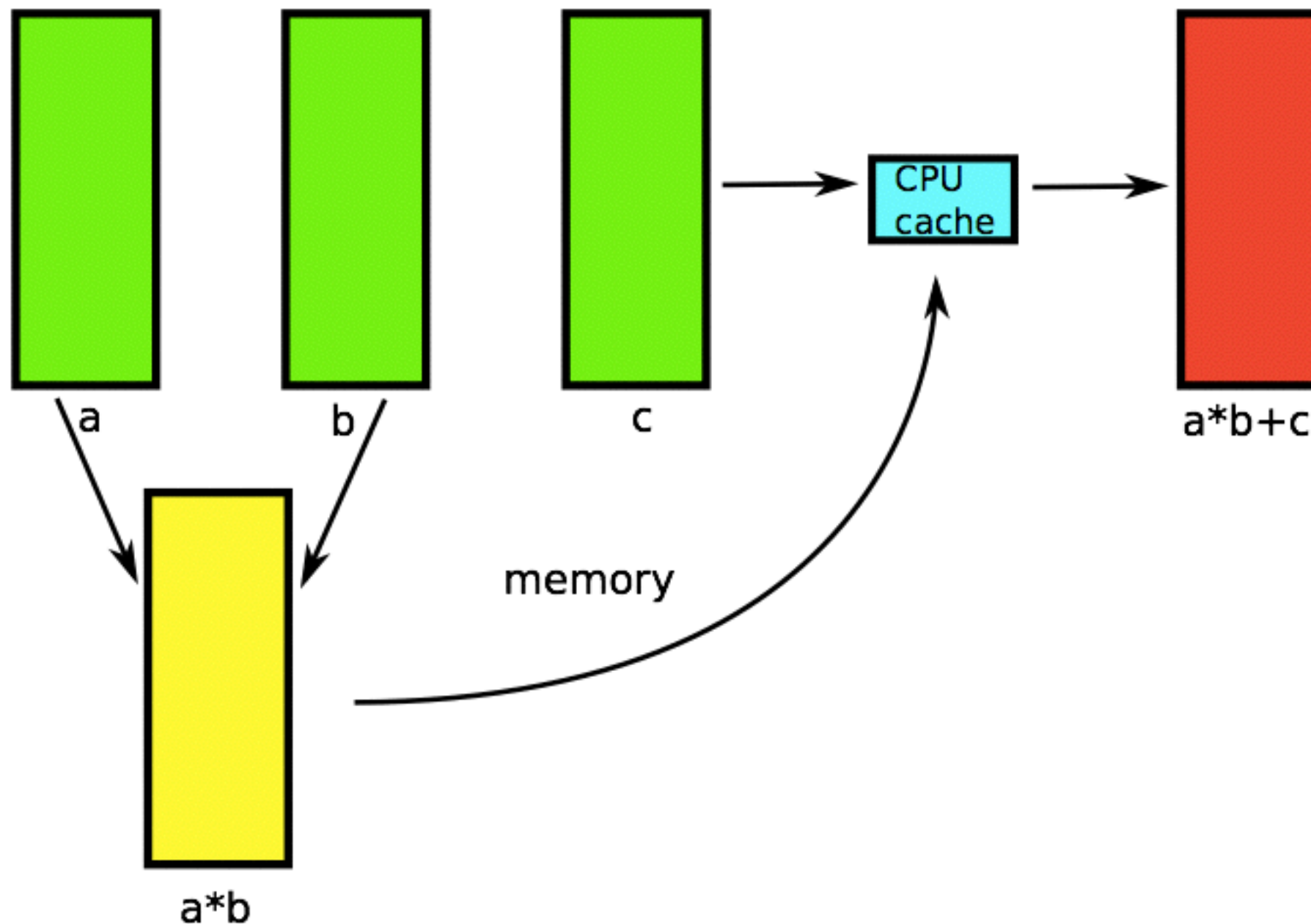
0.25*x3 + 0.75*x**2 + 1.5*x − 2**

to:

0.25*x*x*x + 0.75*x*x + 1.5*x − 2

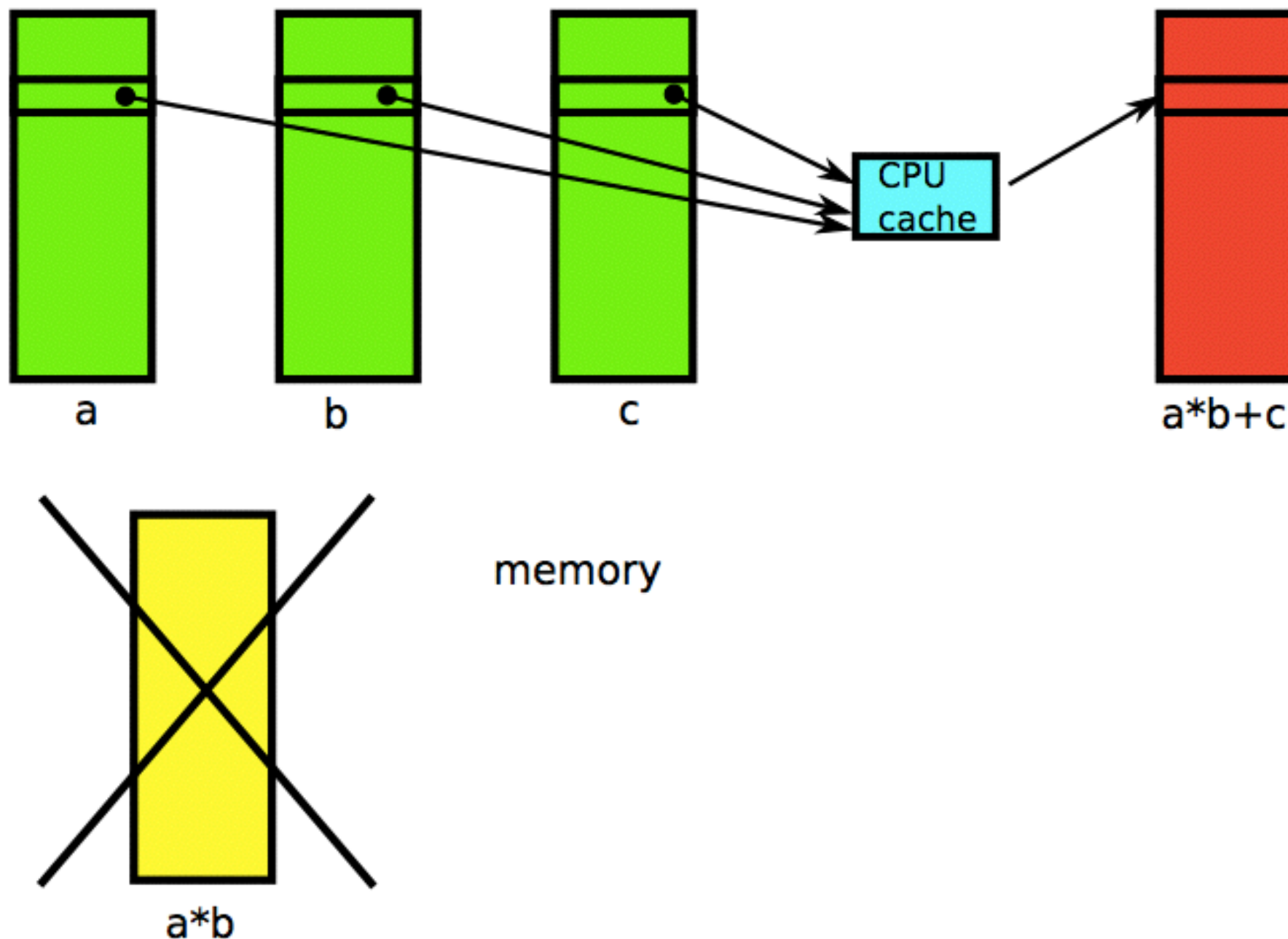so, no need to use the expensive **pow()**

Computing with NumPy
Temporaries go to memory

Computing with numexpr
Temporaries stay in cache

a   b   c   CPU cache   a*b+c

memory

a*b

Multithreaded numexpr and Beyond: Numba

numexpr with 16 (logical) cores

Legend:
- .25*x**3 + .75*x**2 - 1.5*x − 2
- ((.25*x + .75)*x - 1.5)*x − 2
- x

Memory bounded!

((.25*x + .75)*x - 1.5)*x − 2
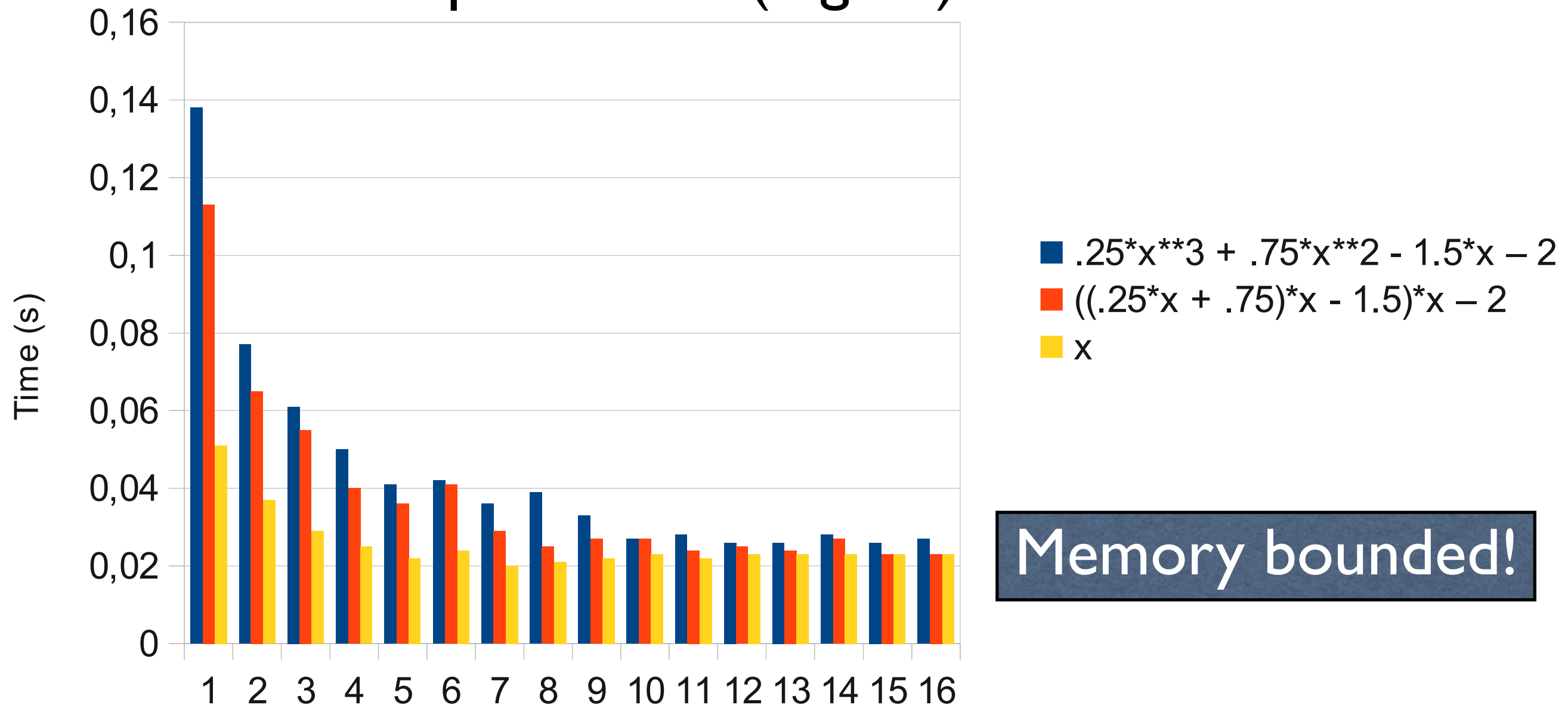
Transcendental Functions

numexpr with 16 (logical) cores

- ■ .25*x**3 + .75*x**2 - 1.5*x − 2
- ■ ((.25*x + .75)*x - 1.5)*x − 2
- ■ x
- ■ sin(x)**2+cos(x)**2
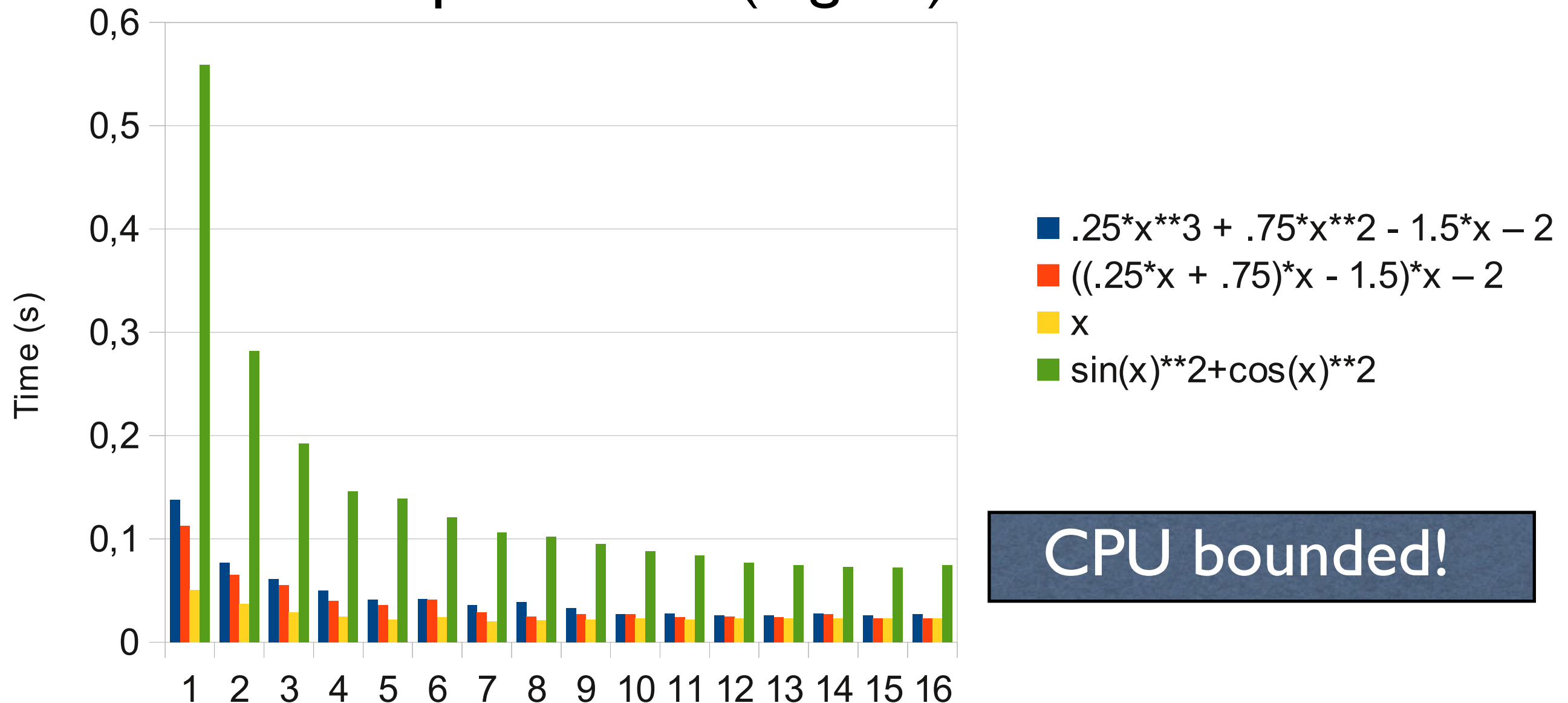
CPU bounded!

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]
```
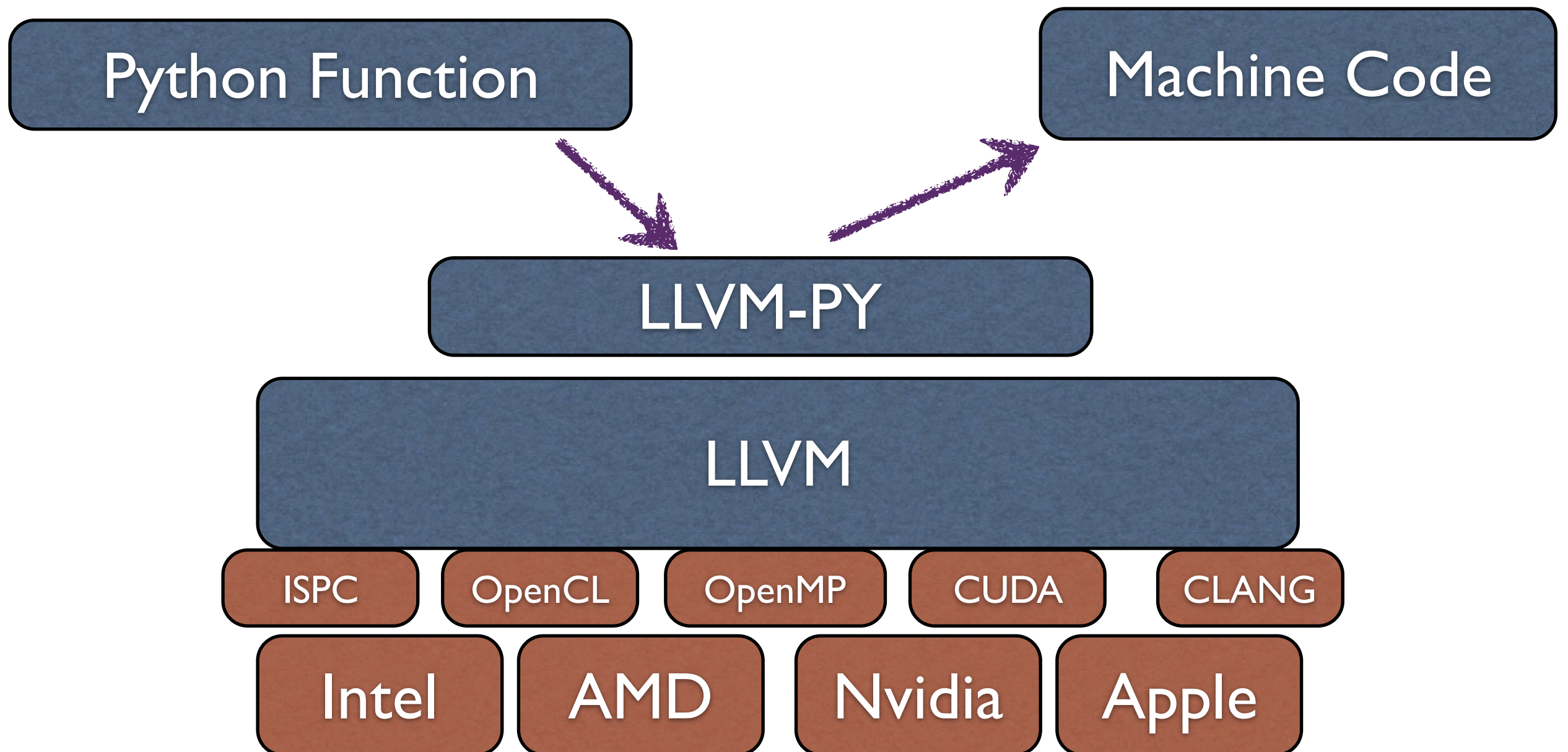
- In particular, it cannot deal with things like:

```
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

Python Function

Machine Code

LLVM-PY

LLVM

ISPC | OpenCL | OpenMP | CUDA | CLANG

Intel | AMD | Nvidia | Apple

Take-away Messages

- When you have to optimize, have in mind the starving CPU problem.

- Do not always try to parallelize blindly. Give optimization a try first.

- Use proper tools when you need speed. Using one single tool for everything is not going to work well.

What's Next

- Parallel Computing tomorrow (Eilif)

- Efficient Data Containers (in-memory and on-disk)

- Social event this night!