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

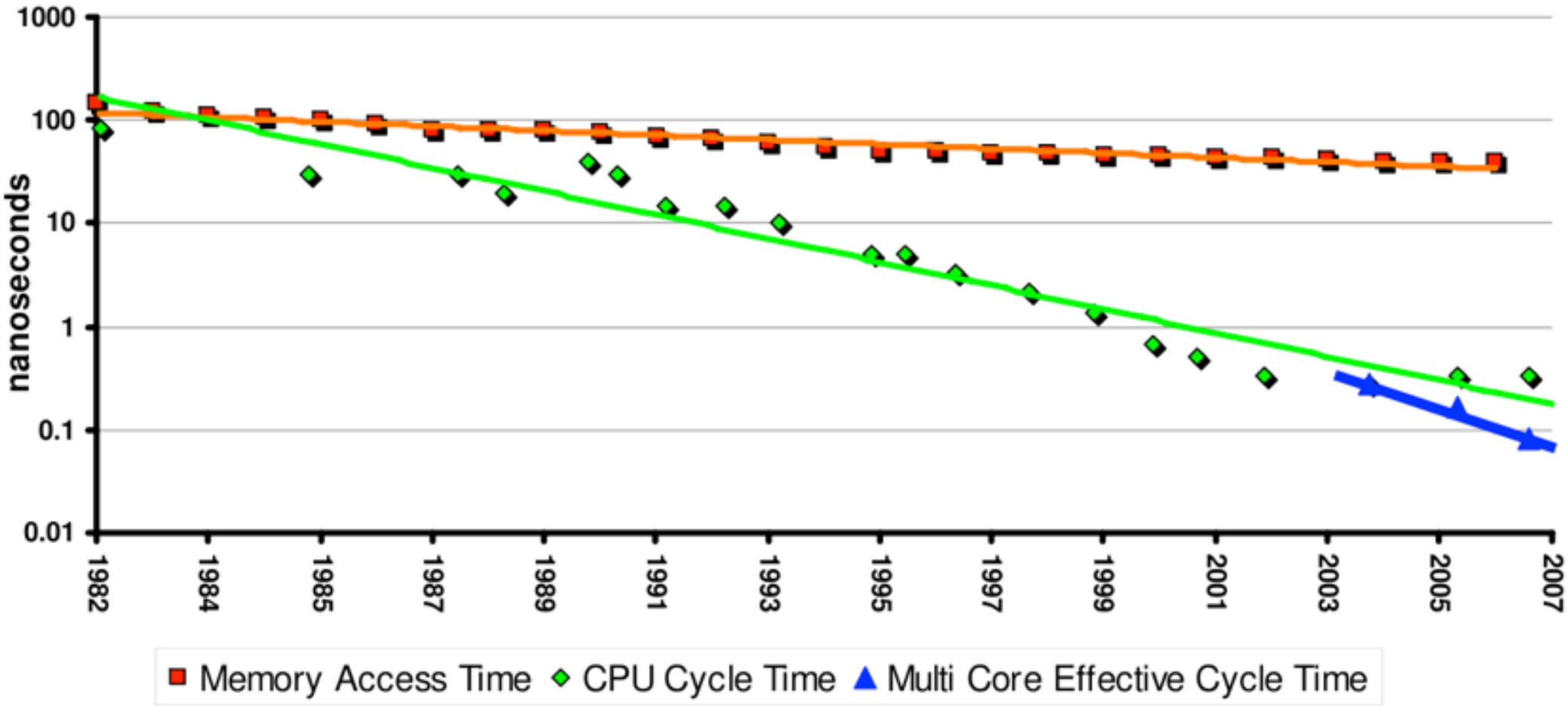
- The Starving CPU problem
- Recent trends in computer architecture
- Handling Big Data: storing and processing as much data as possible with your existing resources

"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

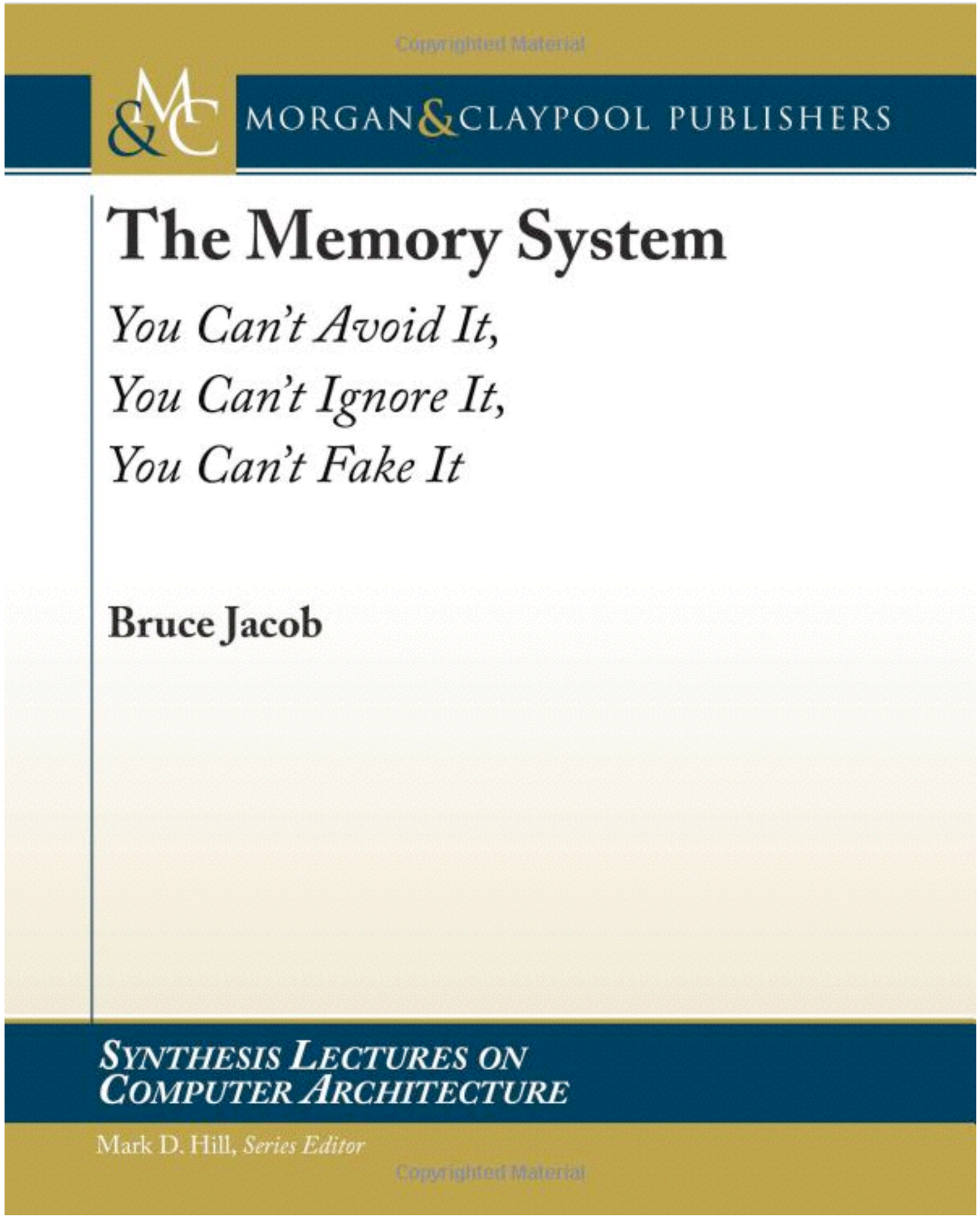
Memory Access Time vs CPU Cycle Time



The gap is wide and still opening!

Book in

2009


Copyrighted Material
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
Copyrighted Material

The Status of CPU Starvation in 2016

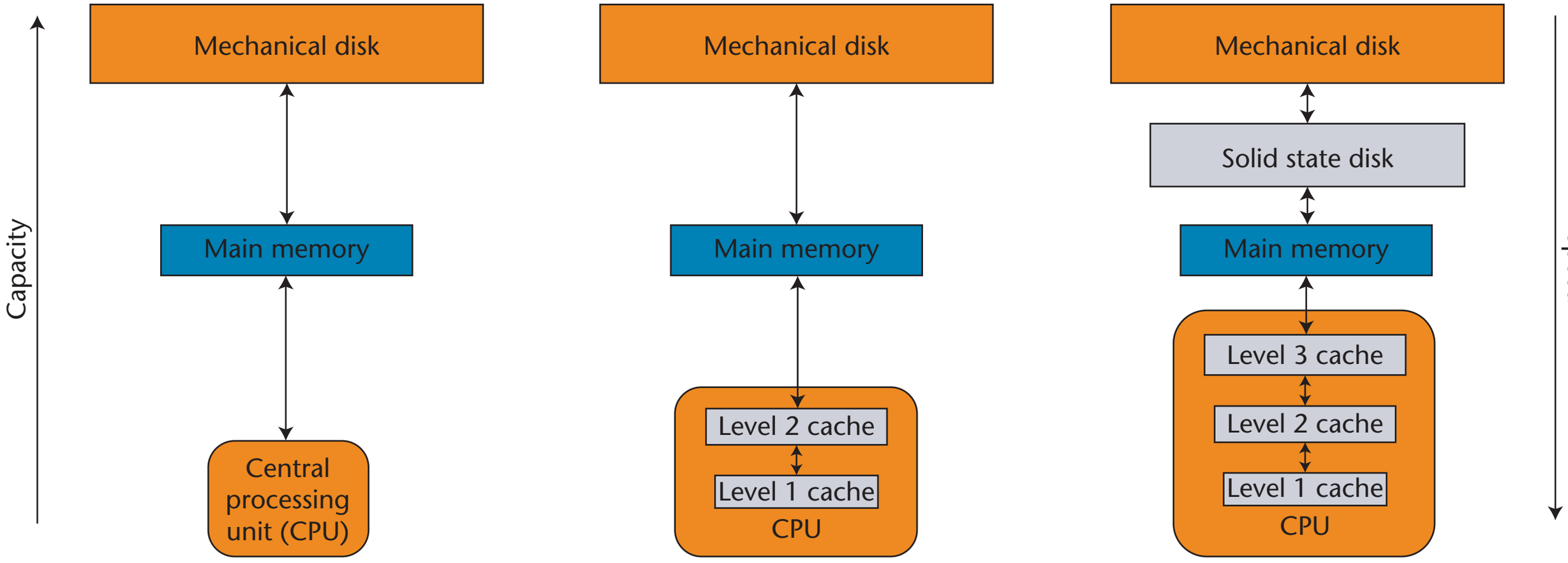
- Memory latency is much slower (between 250x and 1000x) than processors.

- 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



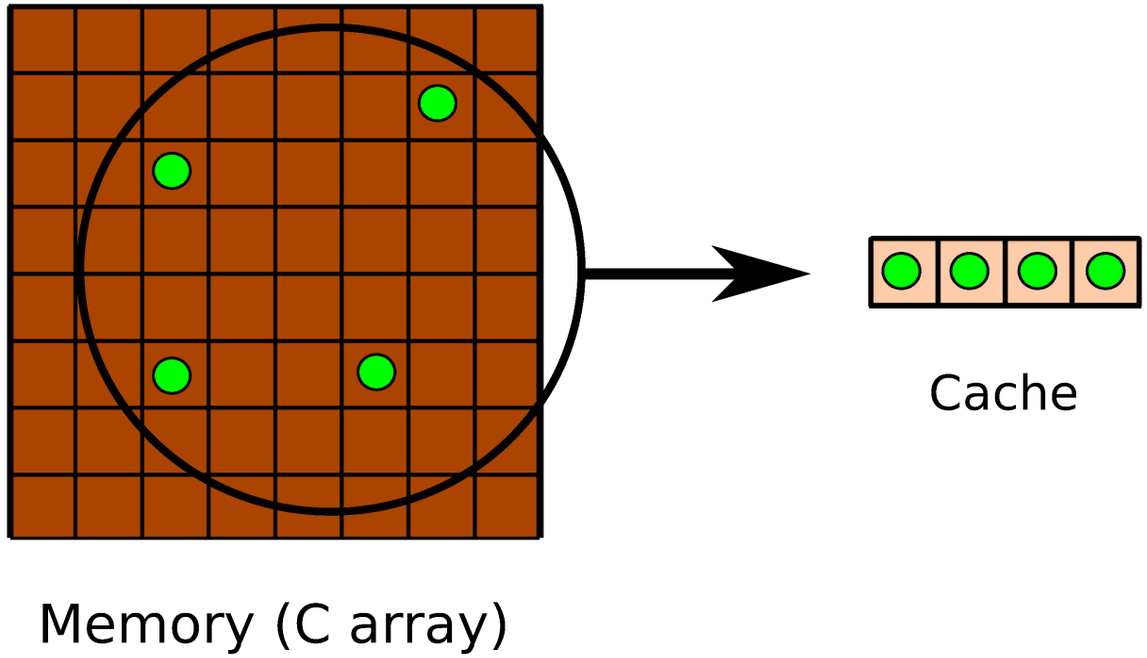
When CPU Caches Are Effective?

Mainly in a couple of scenarios:

- Time locality: when the dataset is reused
- Spatial locality: when the dataset is accessed sequentially

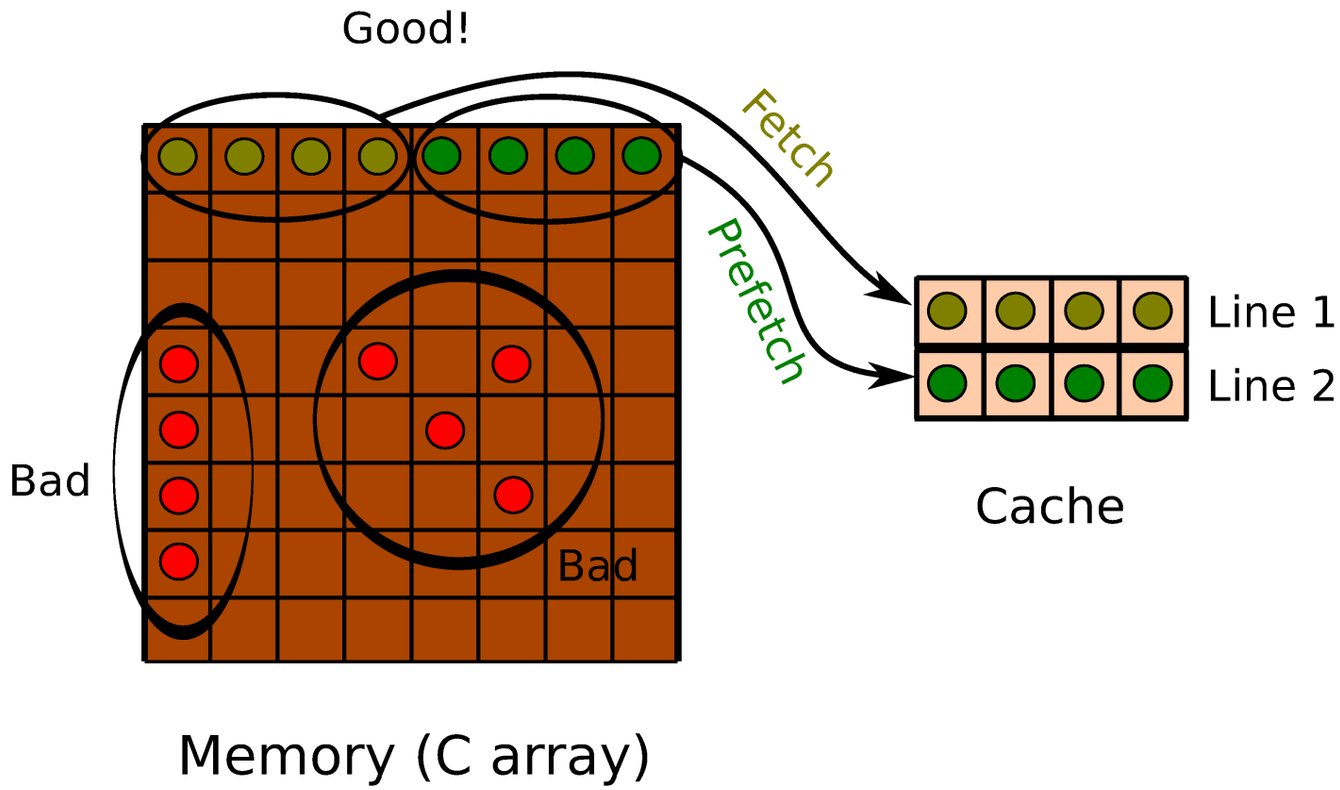
Time Locality

Parts of the dataset are reused



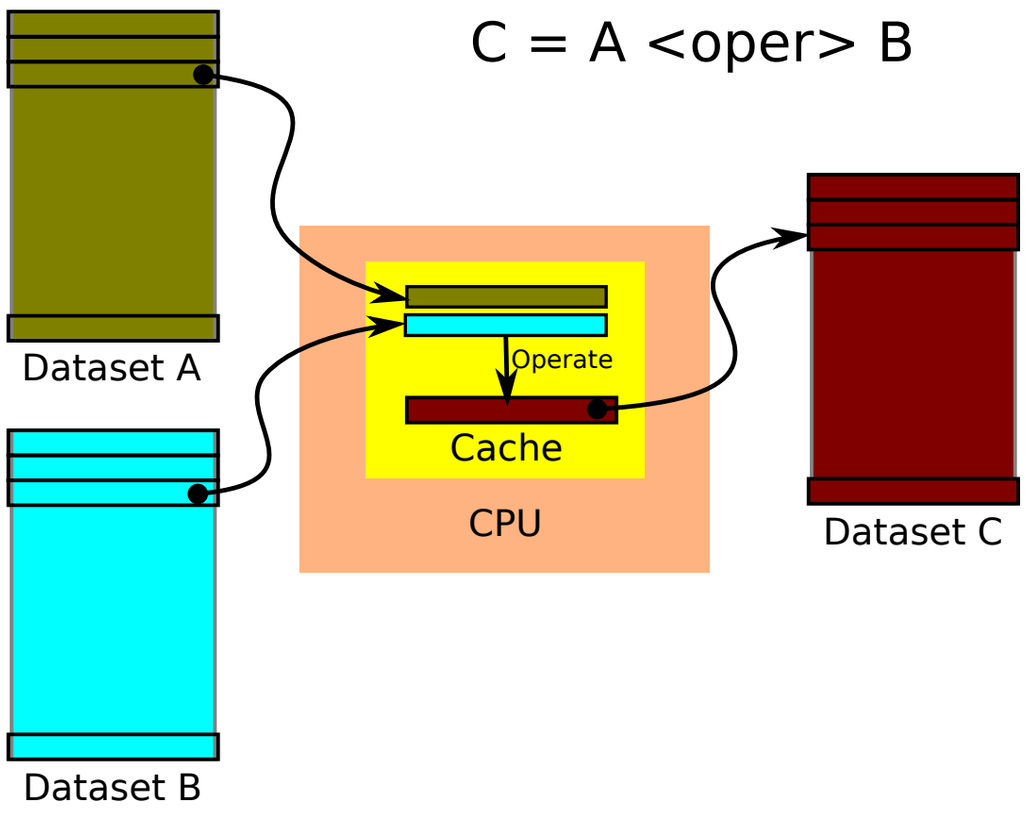
Spatial Locality

Dataset is accessed sequentially



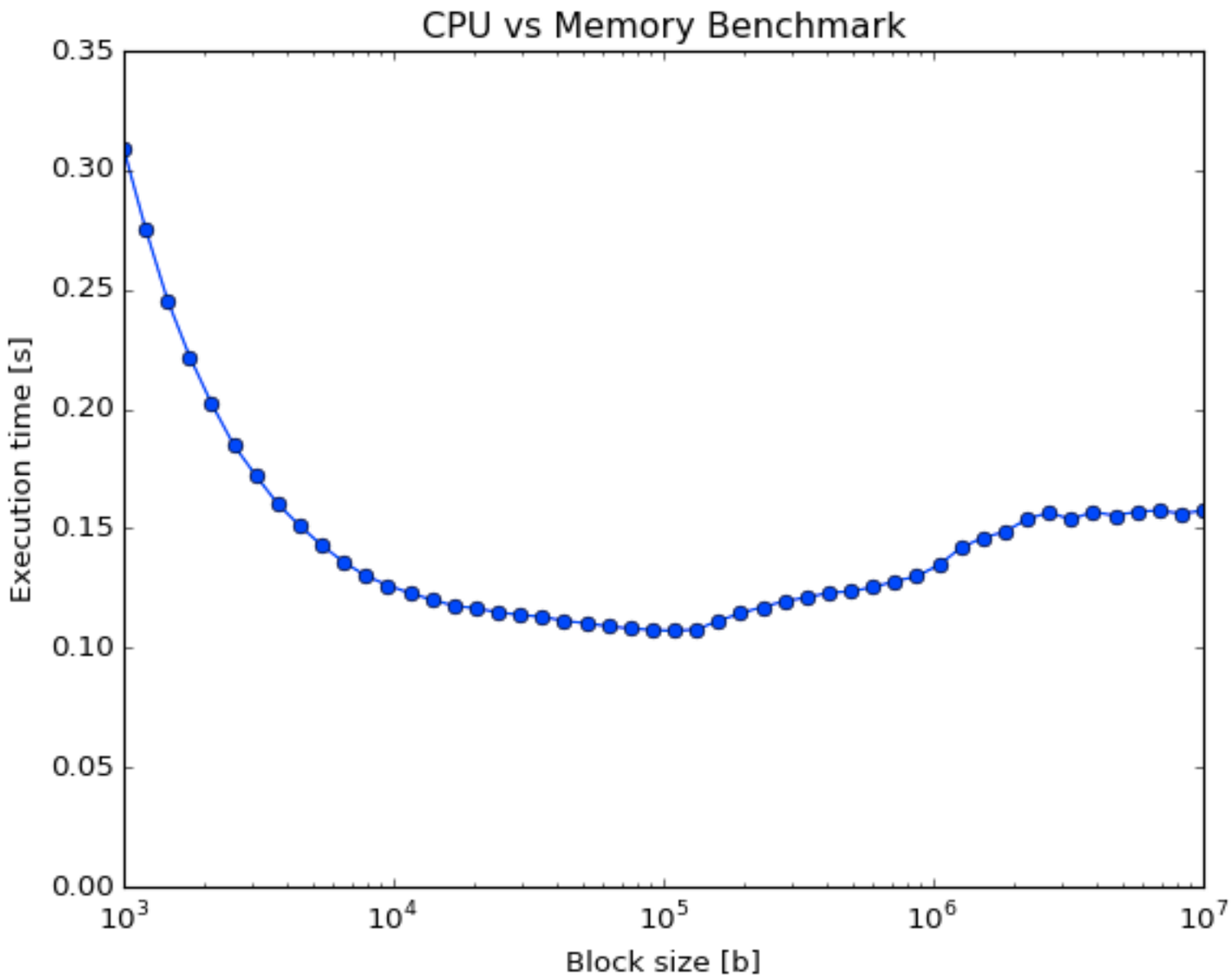
The Blocking Technique

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



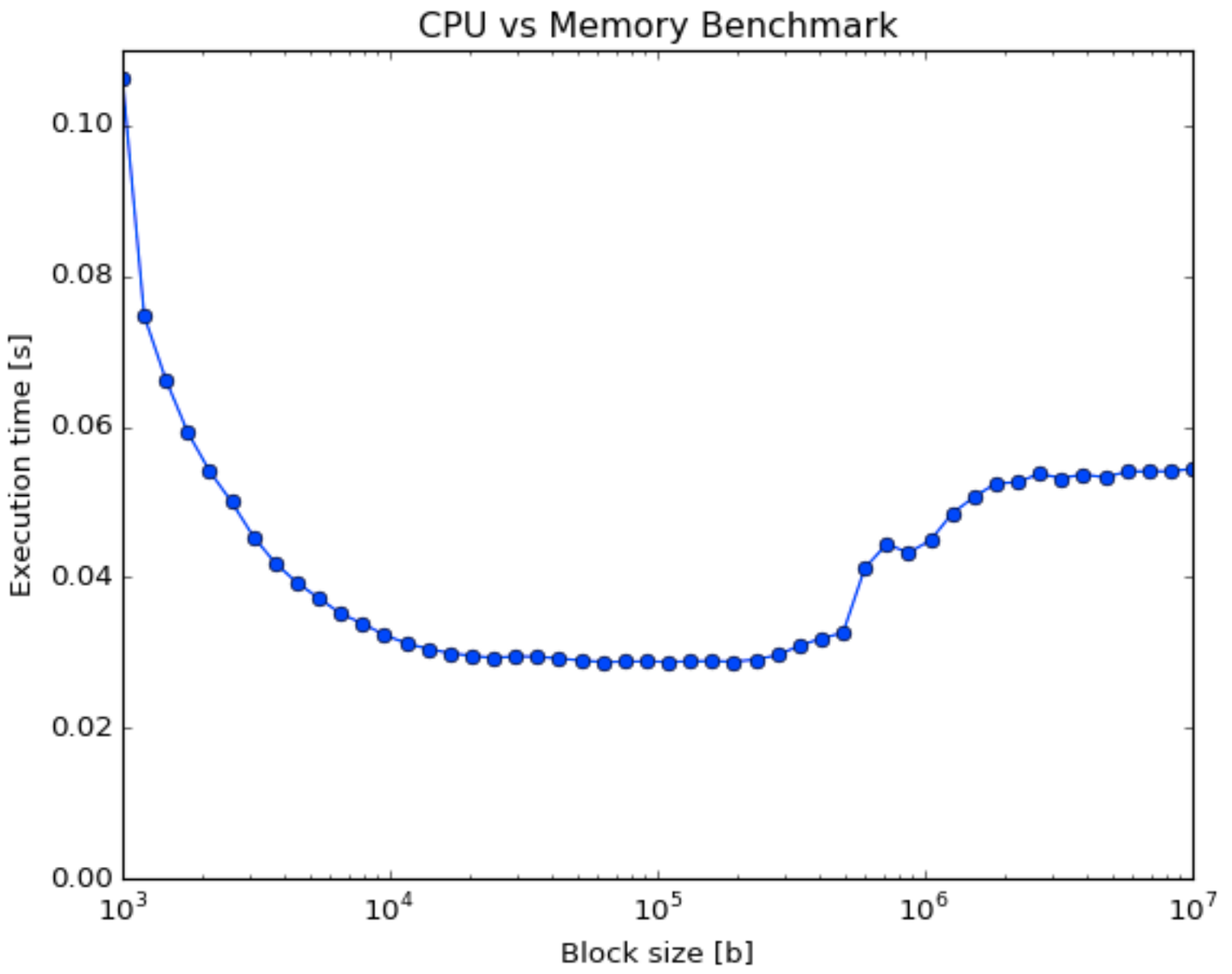
Use this extensively to leverage spatial and temporal localities

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)

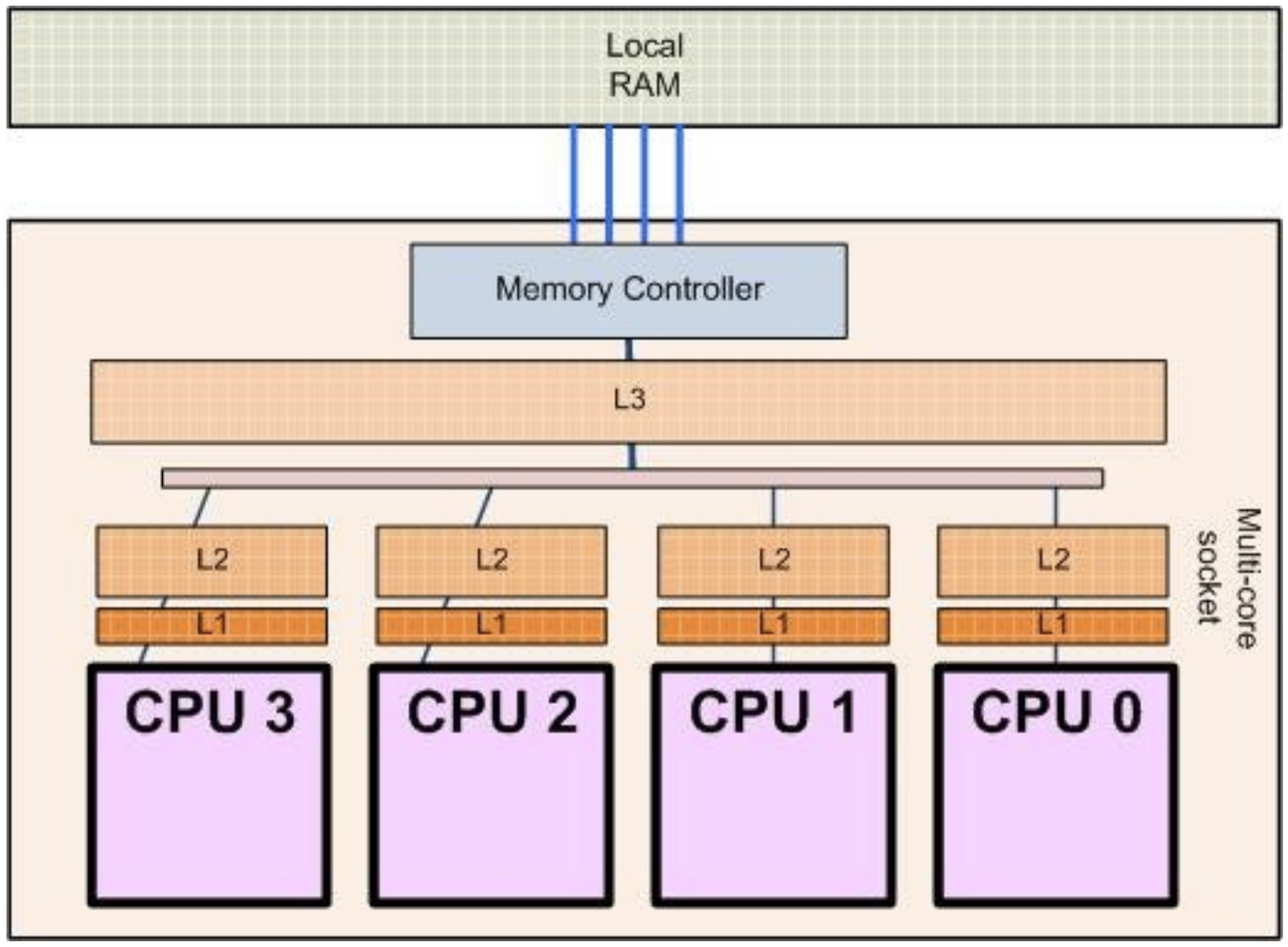
"Across the industry, today's chips are largely able to execute code faster than we can feed them with instructions and data. There are no longer performance bottlenecks in the floating-point multiplier or in having only a single integer unit. The real design action is in memory subsystems— caches, buses,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

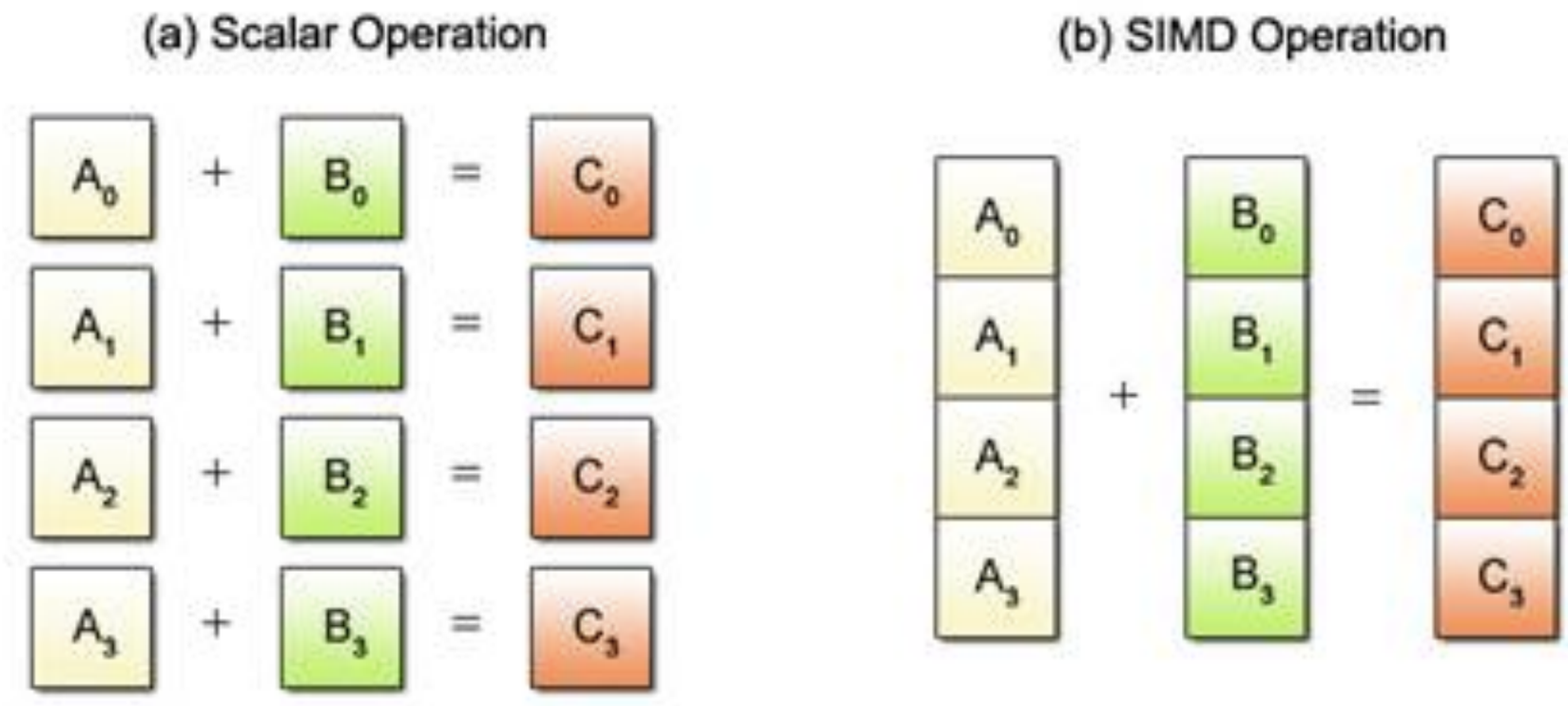
Trends in Computer Architecture

We Are In A Multicore Age



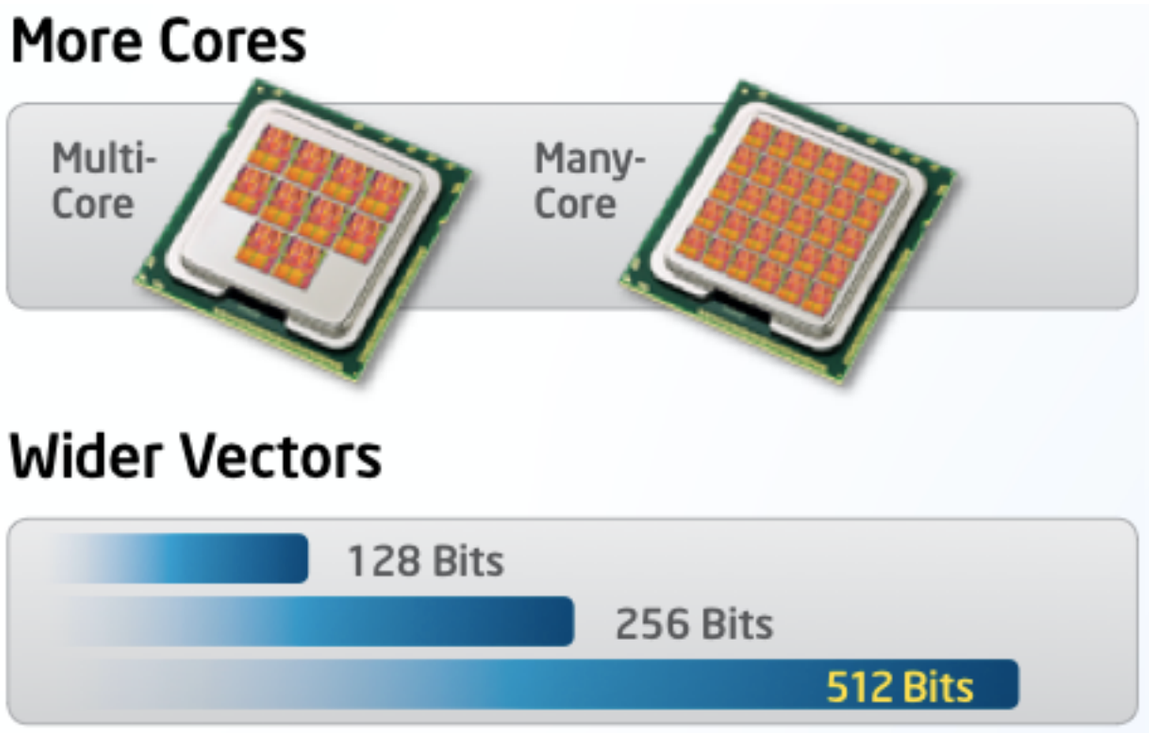
- This requires special programming measures to leverage all its potential: threads, multiprocessing

SIMD: Single Instruction, Multiple Data

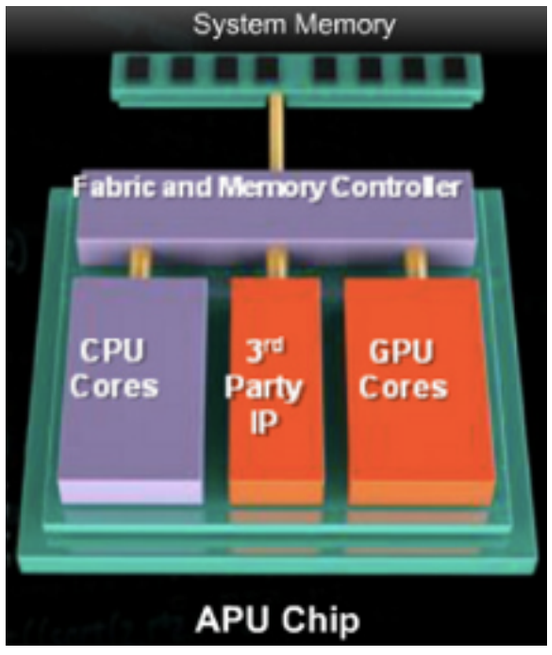


More operations in the same CPU clock

Forthcoming Trends in CPU



CPU+GPU Integration



Hierarchy of Memory By 2018 (Educated Guess)

HDD (persistent)

SSD SATA (persistent)

SSD PCIe (persistent)

XPoint (persistent)

RAM (addressable)

L4

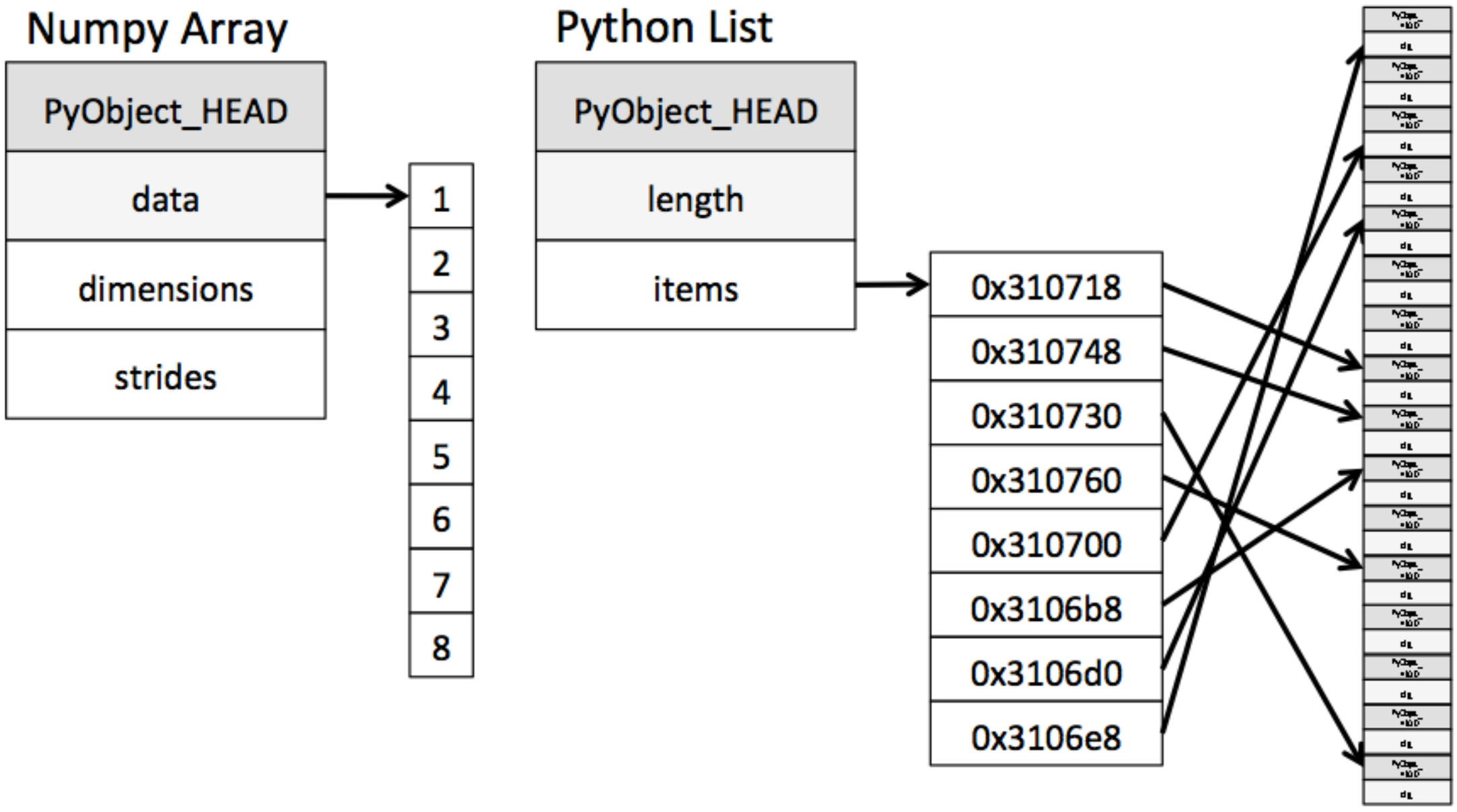
L3

L2

L1

9 levels will be common!

Which Structure Is More Efficient for Computing?



Attr: Jake Vanderplas

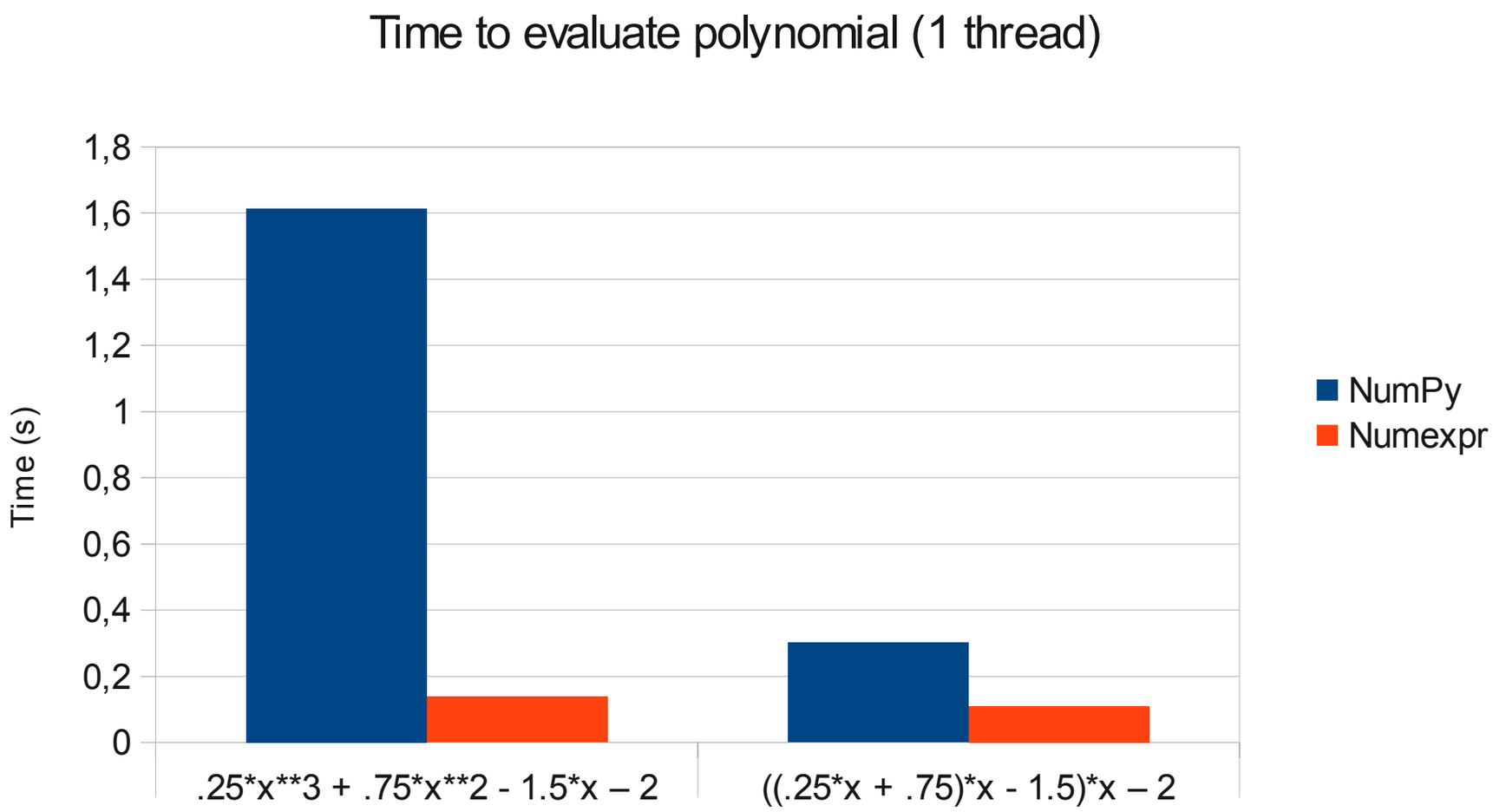
numexpr

- 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 bcolz (among others) can all leverage numexpr

Computing with numexpr



Power Expansion

Numexpr expands expression:

```
0.25*x**3 + 0.75*x**2 + 1.5*x - 2
```

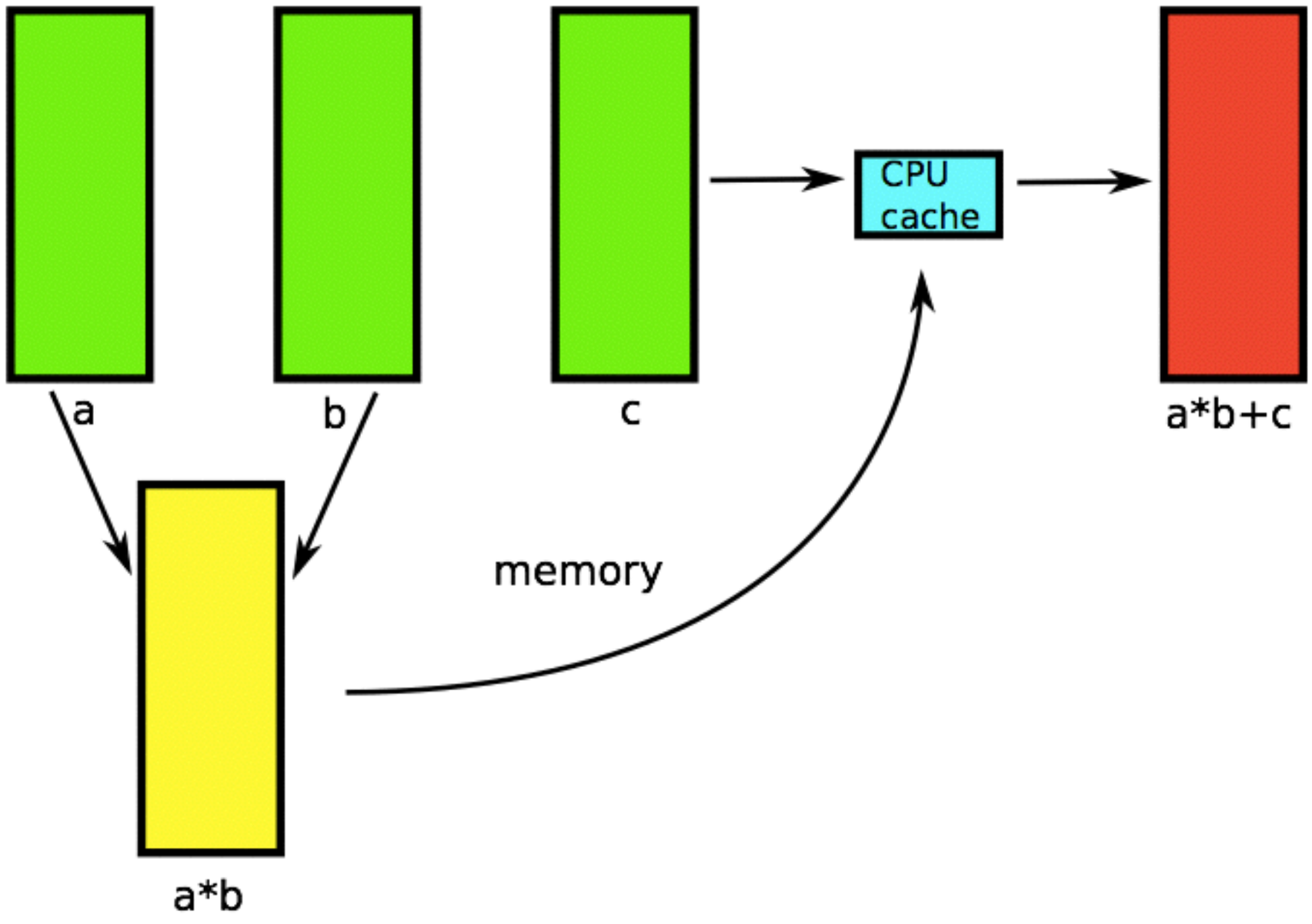
to:

```
0.25*x*x*x + 0.75*x*x + 1.5*x - 2
```

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

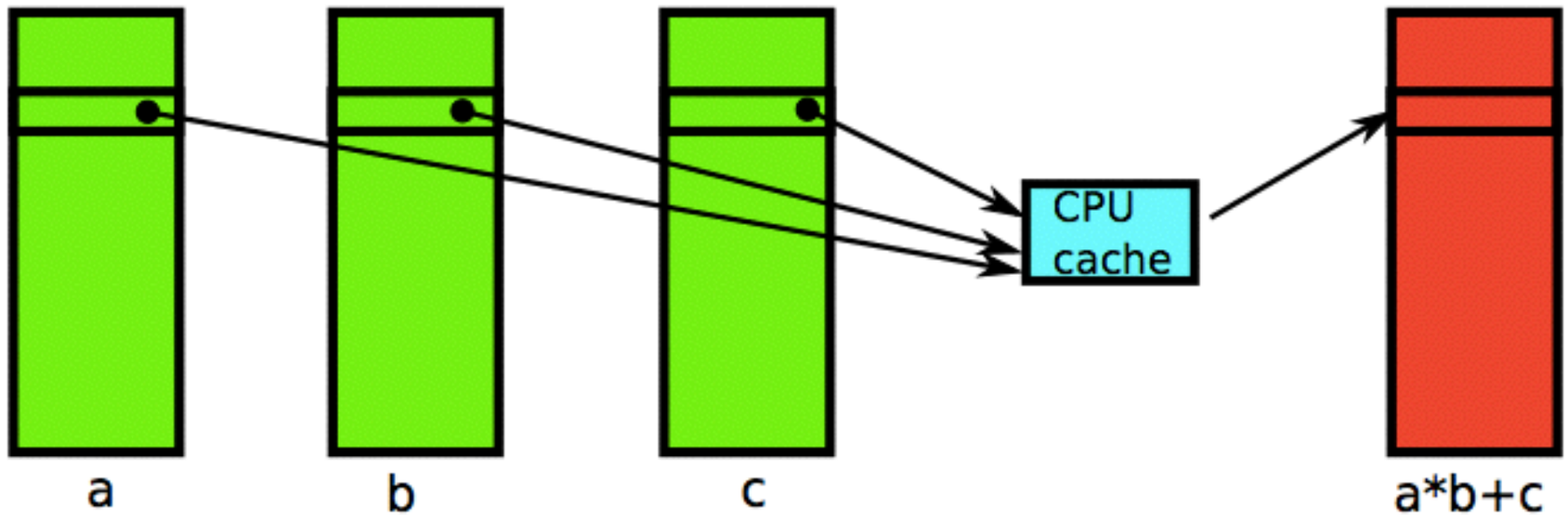
Computing with NumPy

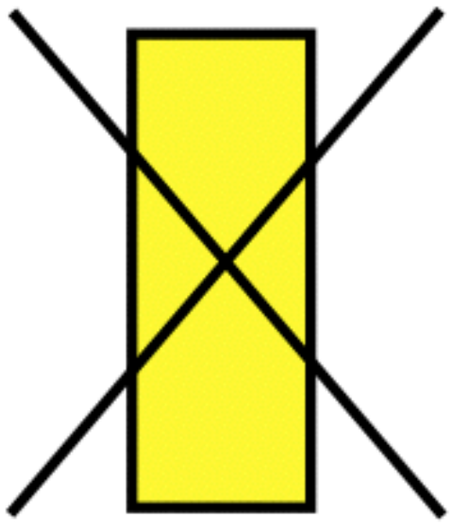
Temporaries go to memory



Computing with numexpr

Temporaries stay in cache

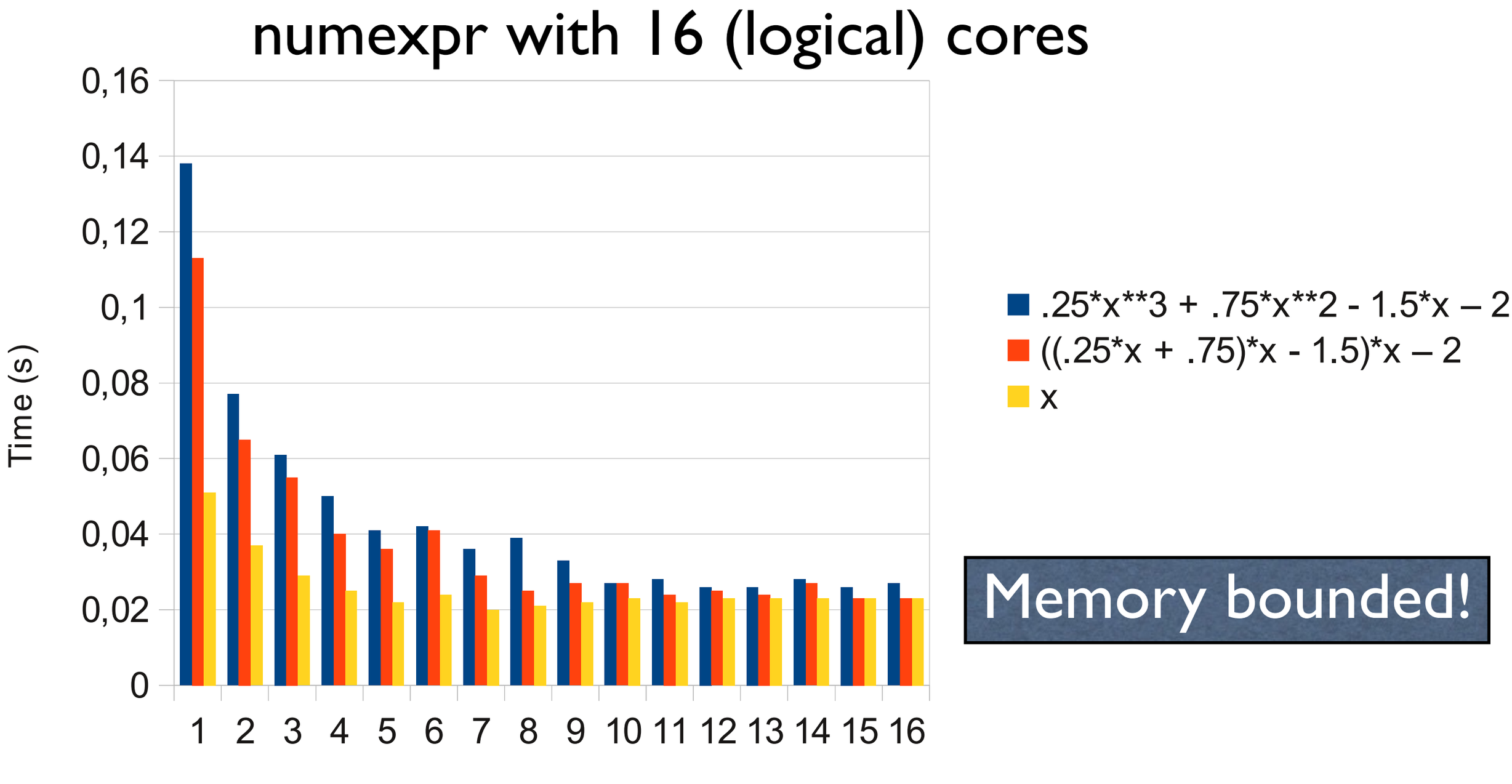




memory

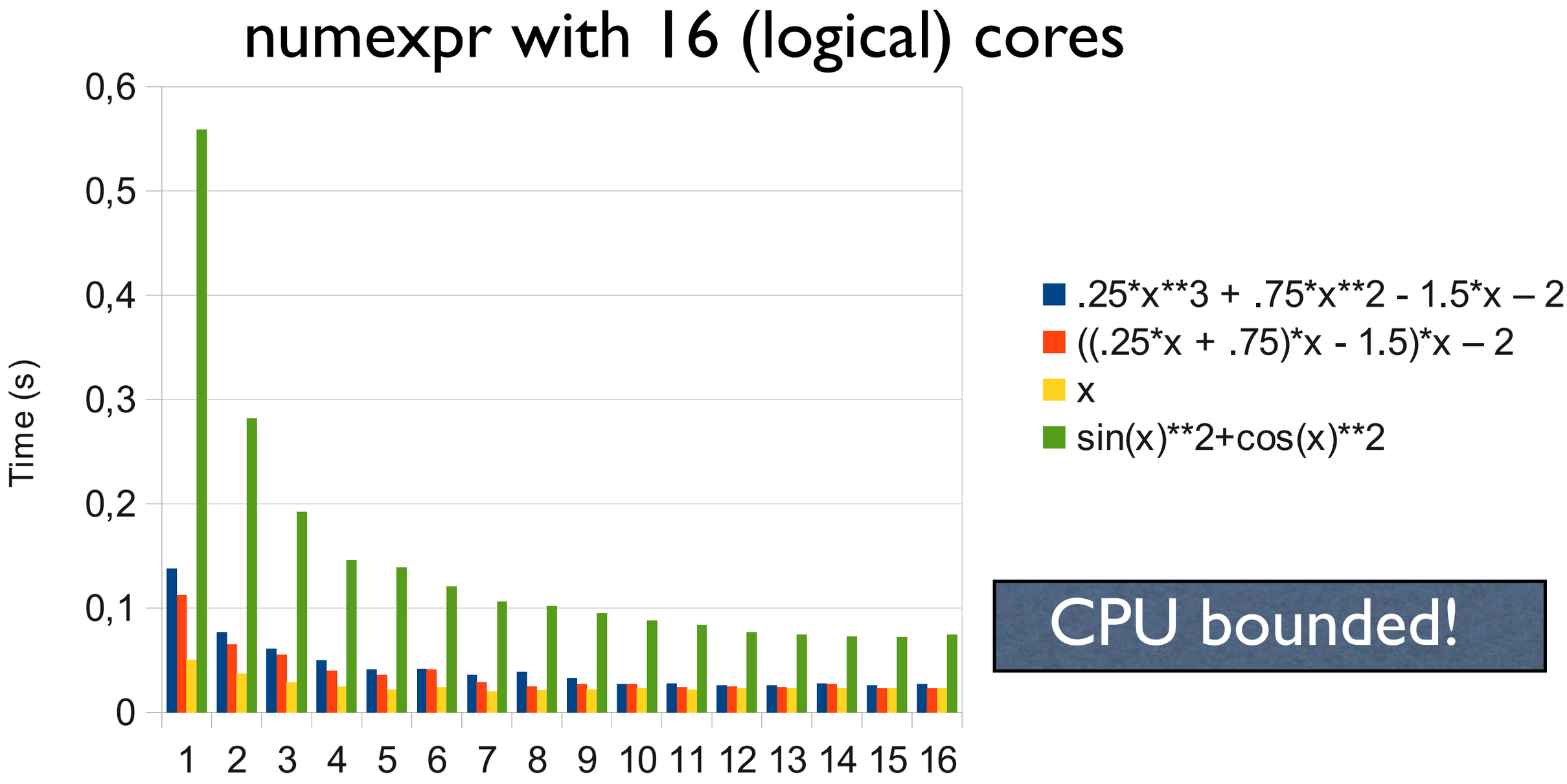
Multithreaded numexpr and Beyond: Numba

numexpr Allows Multithreading for Free



Memory bounded!

Transcendental Functions



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]
```

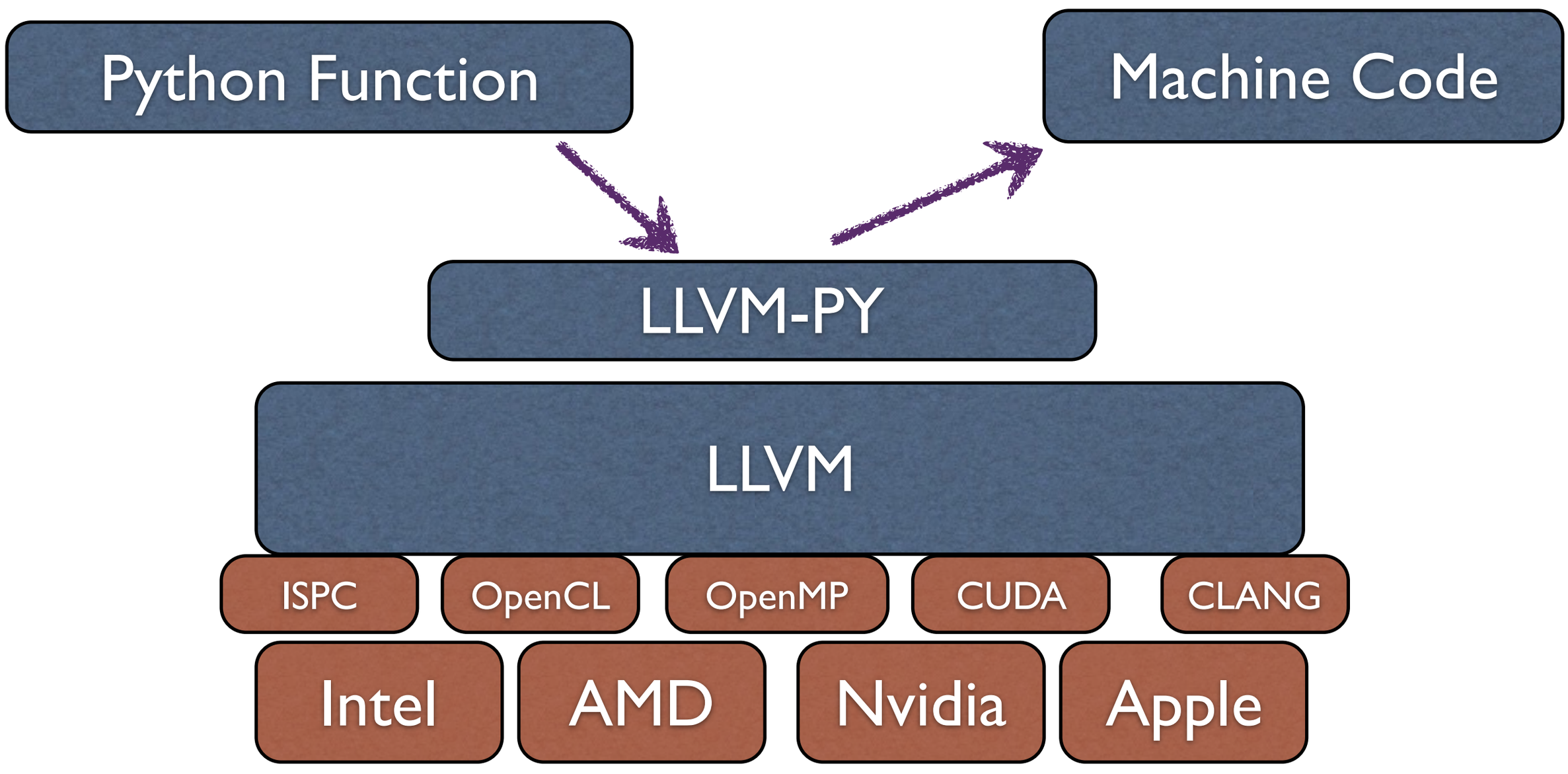
- In particular, it cannot deal with things like:

```
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



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)
- Pelita!