

# The joy of building large scale systems

Suhail Patel | @suhailpatel | <https://suhailpatel.com>

YOW! 2023





ChatGPT Prompt: Generate an image of a group of diverse software engineers being afraid of the on-call pager going off



# Wait, but why?

Many of the systems (apps, services, databases, caches, queues etc.) that we build/rely on are grounded on quite poor assumptions for the hardware of today



# Choose Boring Technology

This is the spoken word version of my essay, [Choose Boring Technology](#). I have largely come to terms with it and the reality that I will never escape its popularity.

I gave this most recently at the Wikimedia Foundation's developer conference, where [Scott Ananian](#) called it "how to be old, for young people."

Here are [my other talks](#), [my website](#), and [some Medium posts](#).

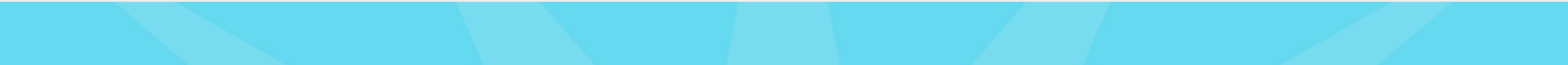
正體中文翻譯在 [這裡](#)

 Tweet

 Follow @mcfunley



#



Hey #





**Suhail Patel**

Senior Staff Engineer at Monzo

@suhailpatel



# Banking made easy

Spend, save and manage your money, all in one place.  
Open a full UK bank account from your phone, for free.

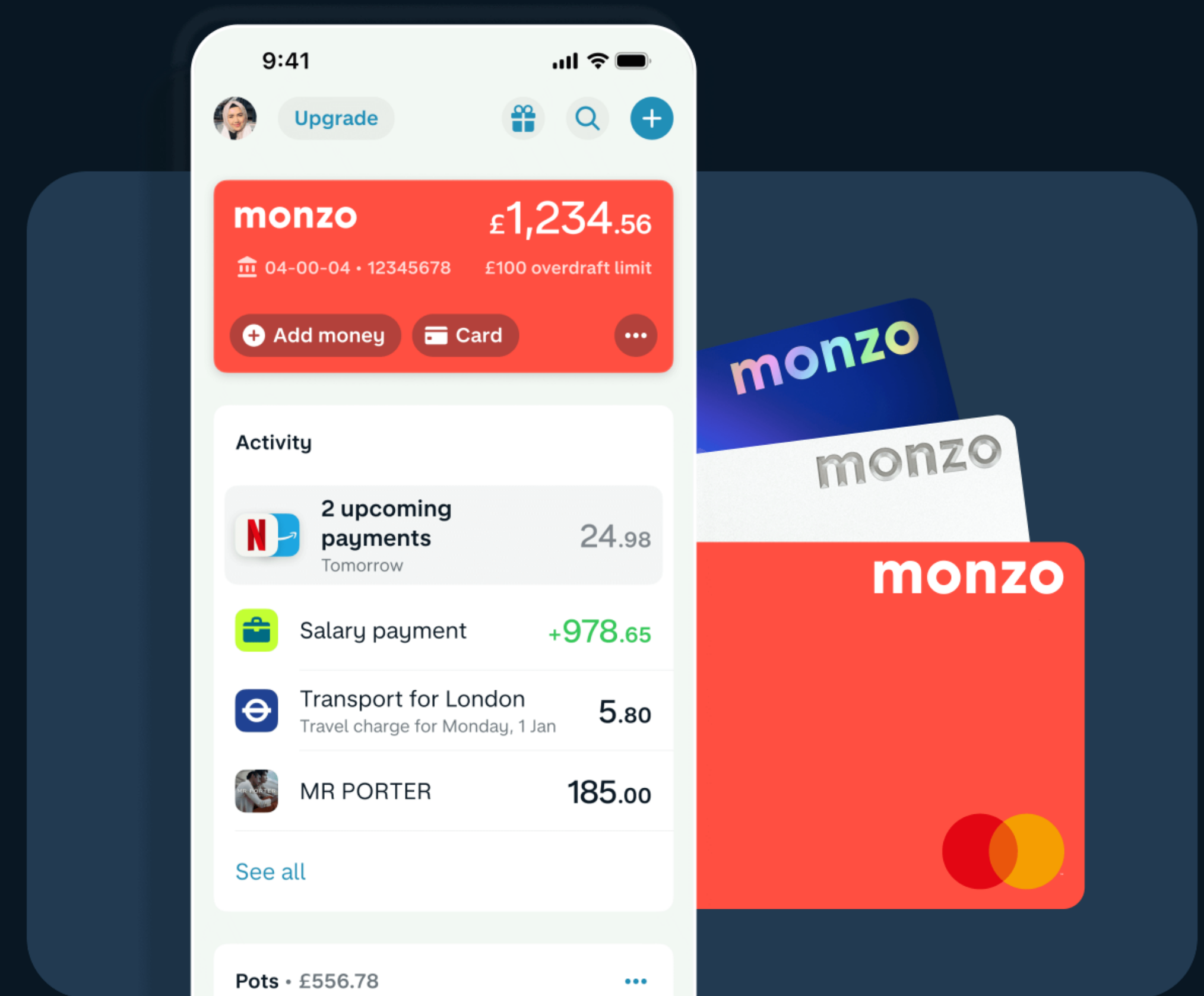
Open a Monzo account

UK residents only. Ts&Cs apply.

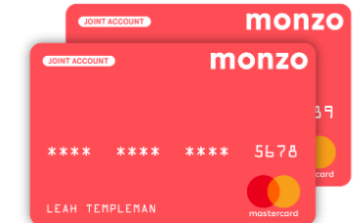
British  
bank  
awards

Best  
banking app  
Winner 2023

Smart  
money  
people







## Account Service



# Latency numbers

<b>L1 Cache Reference</b>	0.5ns
<b>L2 Cache Reference</b>	7ns
<b>Main Memory Reference</b>	100ns
<b>Send 1K bytes over 1 Gbps network</b>	10,000ns
<b>Read 4K bytes from SSD</b>	150,000ns
<b>Read 1MB sequentially from memory</b>	250,000ns
<b>Read 1MB from SSD</b>	1,000,000ns
<b>Disk Seek</b>	10,000,000ns

Source: <https://gist.github.com/jboner/2841832>



# Latency numbers

L1 Cache Reference	0.5ns	0.5 days
L2 Cache Reference	7ns	7.5 days
Main Memory Reference	100ns	100 days
Send 1K bytes over 1 Gbps network	10,000ns	27 years
Read 4K bytes from SSD	150,000ns	411 years
Read 1MB sequentially from memory	250,000ns	685 years
Read 1MB from SSD	1,000,000ns	2.7 millenniums
Disk Seek	10,000,000ns	27 millenniums

Source: <https://gist.github.com/jboner/2841832>

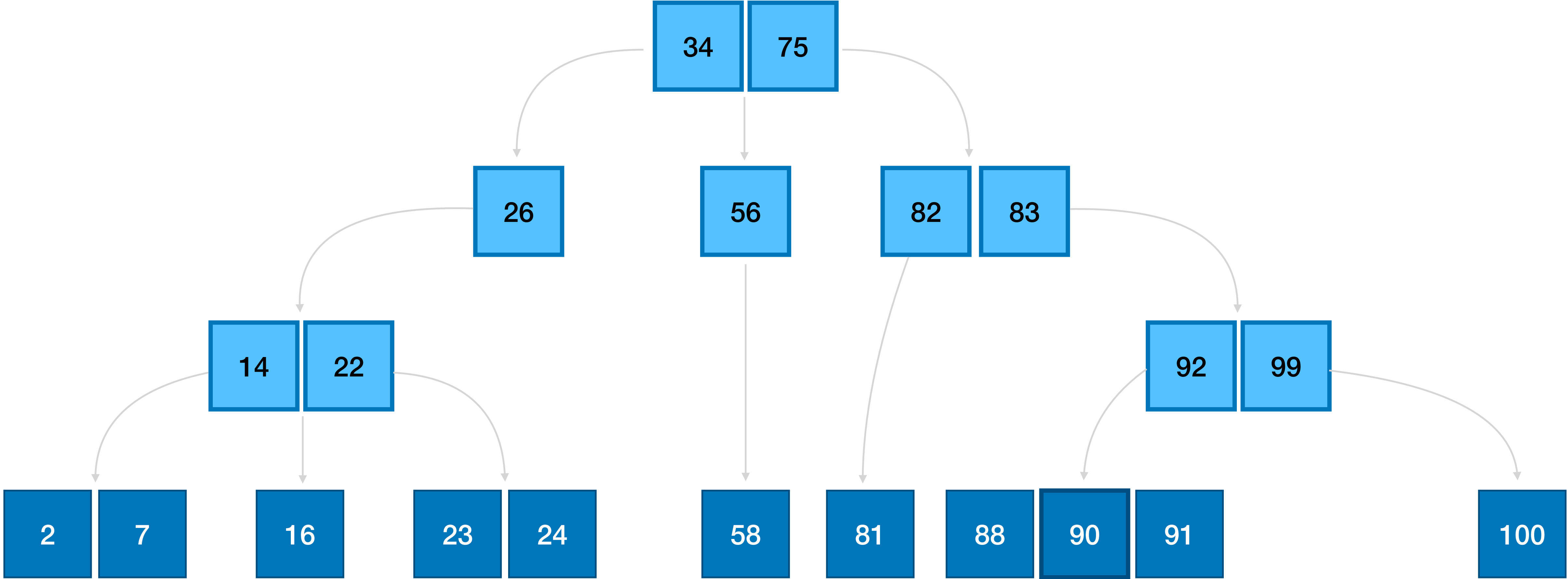


# B-Trees are used in many index implementations



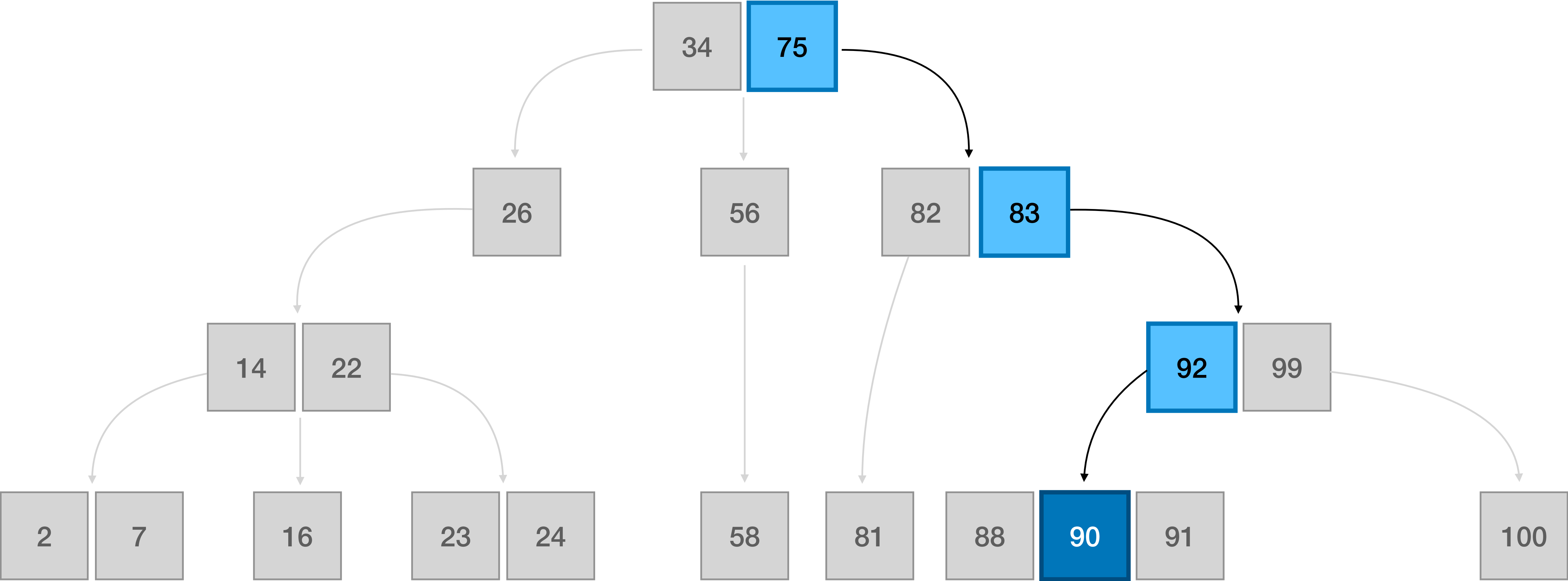


# B-Tree





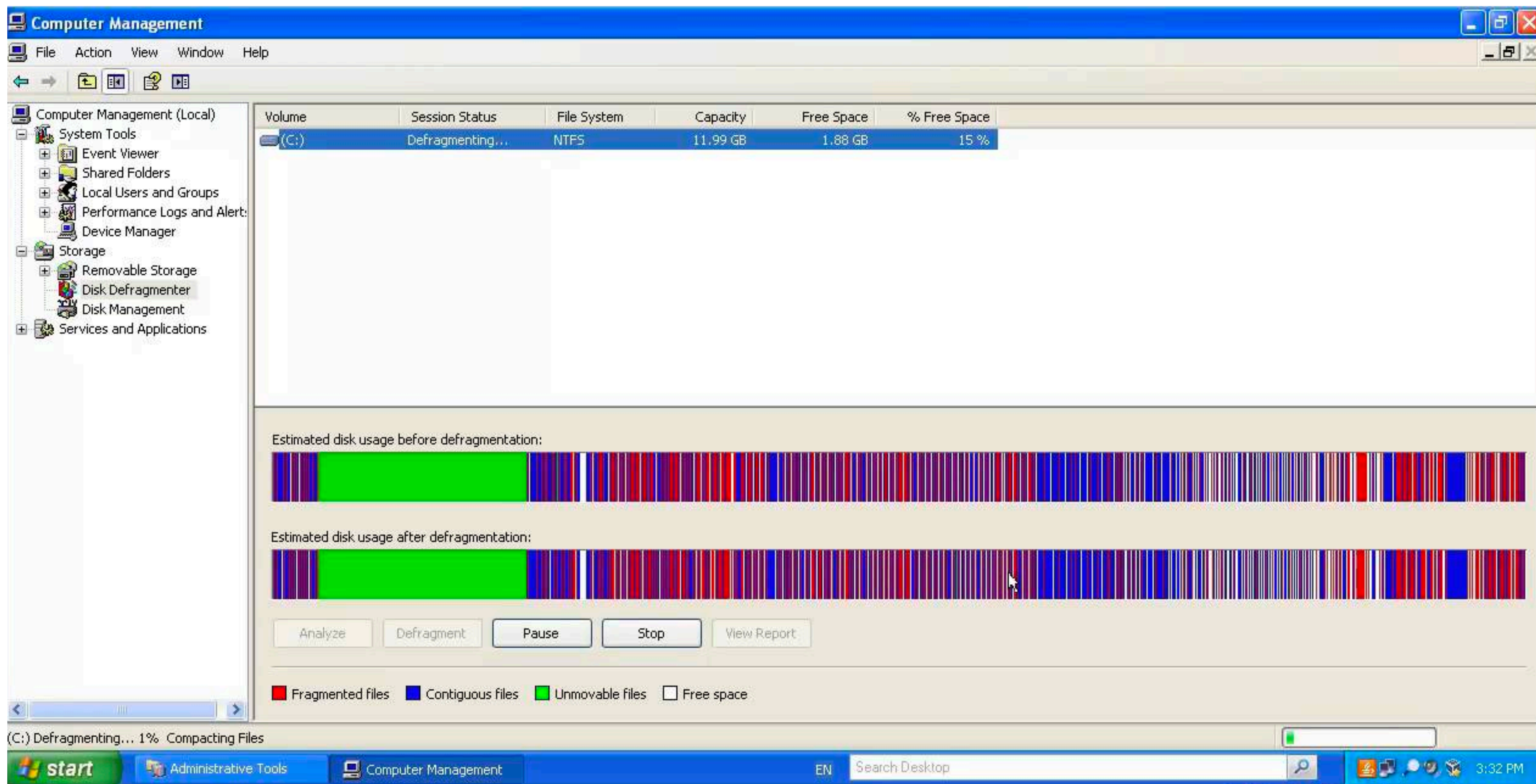
# Searching within a B-Tree





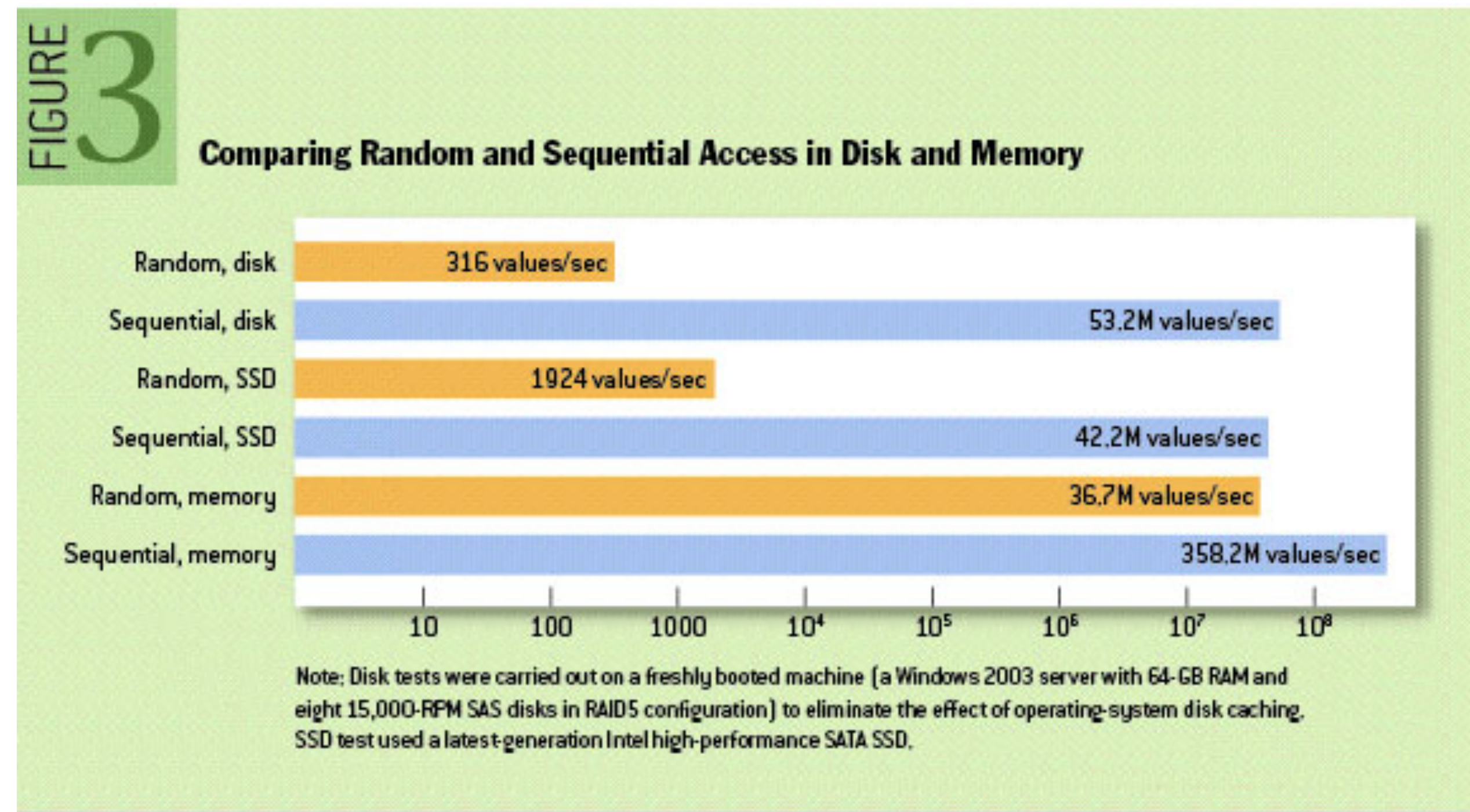








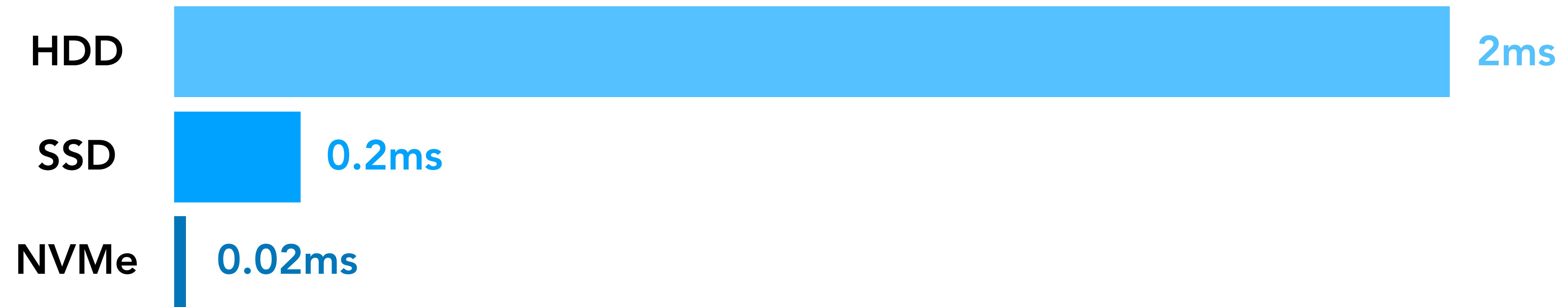
# Random & Sequential I/O



Source: The Pathologies of Big Data by Adam Jacobs (2009)  
<https://queue.acm.org/detail.cfm?id=1563874>



# Disks are really fast nowadays!



**Disk Seek Latency**

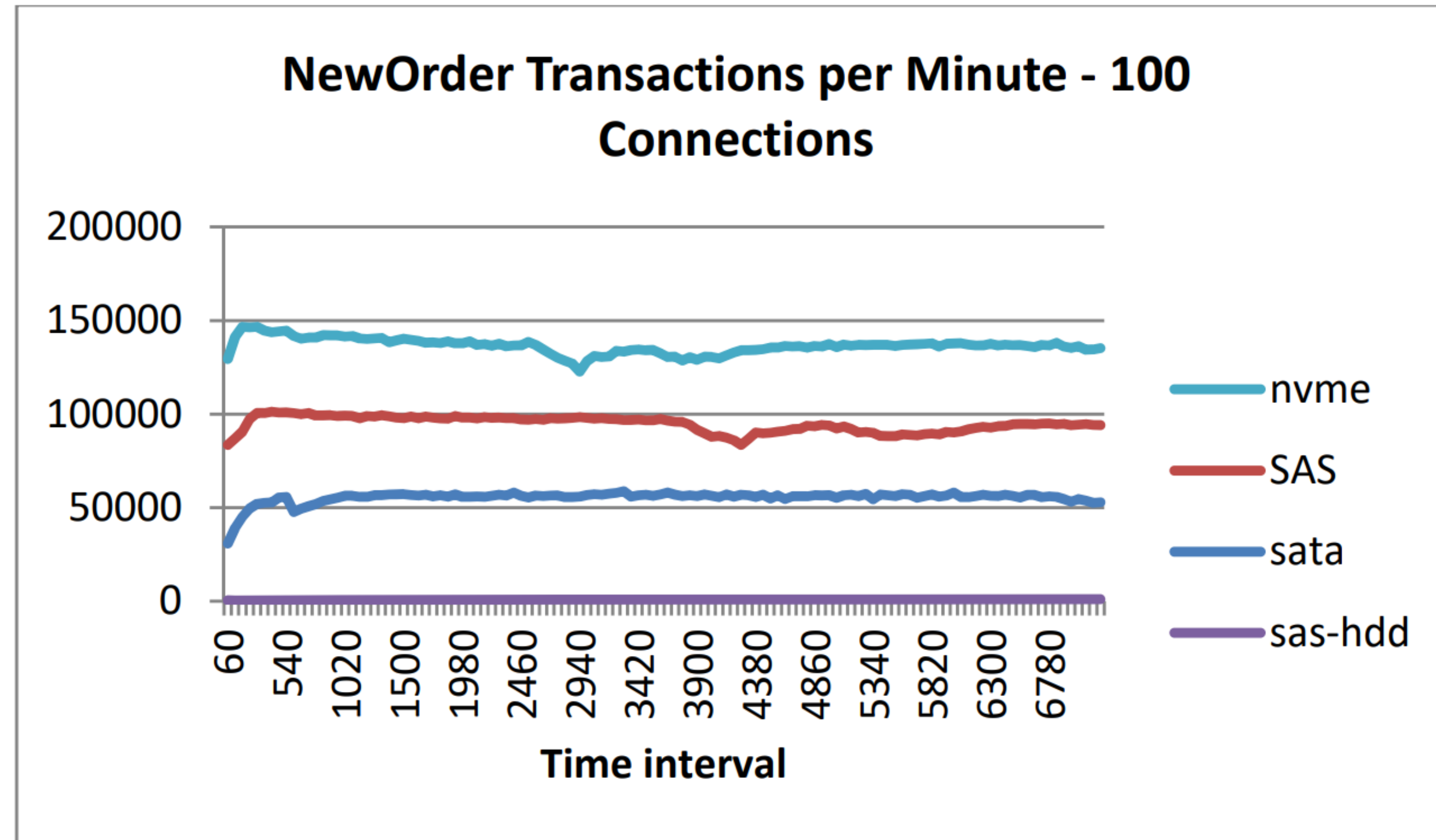
# Disks are really fast nowadays!



Disk Throughput



# Faster hardware = More throughput



Source: <https://download.semiconductor.samsung.com/resources/white-paper/best-practices-for-mysql-with-ssds.pdf>

# Improving I/O Performance via Address Remapping in NVMe Interface

DONG KYU SUNG<sup>ID1</sup>, YONGSEOK SON<sup>ID2</sup>, HYEONSANG EOM<sup>1</sup>, AND SUNGGON KIM<sup>ID3</sup>

<sup>1</sup>Department of Computer Science and Engineering, Seoul National University, Seoul 08826, South Korea

<sup>2</sup>Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea

<sup>3</sup>Department of Computer Science and Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea

Corresponding author: Sunggon Kim (sunggonkim@seoultech.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) through the Korean Government under Grant RS-2022-00166541, Grant NRF-2021R1C1C1010861, Grant NRF-2022R1A4A5034130, and Grant 2021R1F1A1106343812; and in part by the BK21 FOUR Intelligence Computing (Department of Computer Science and Engineering, SNU) funded by NRF under Grant 4199990214639.

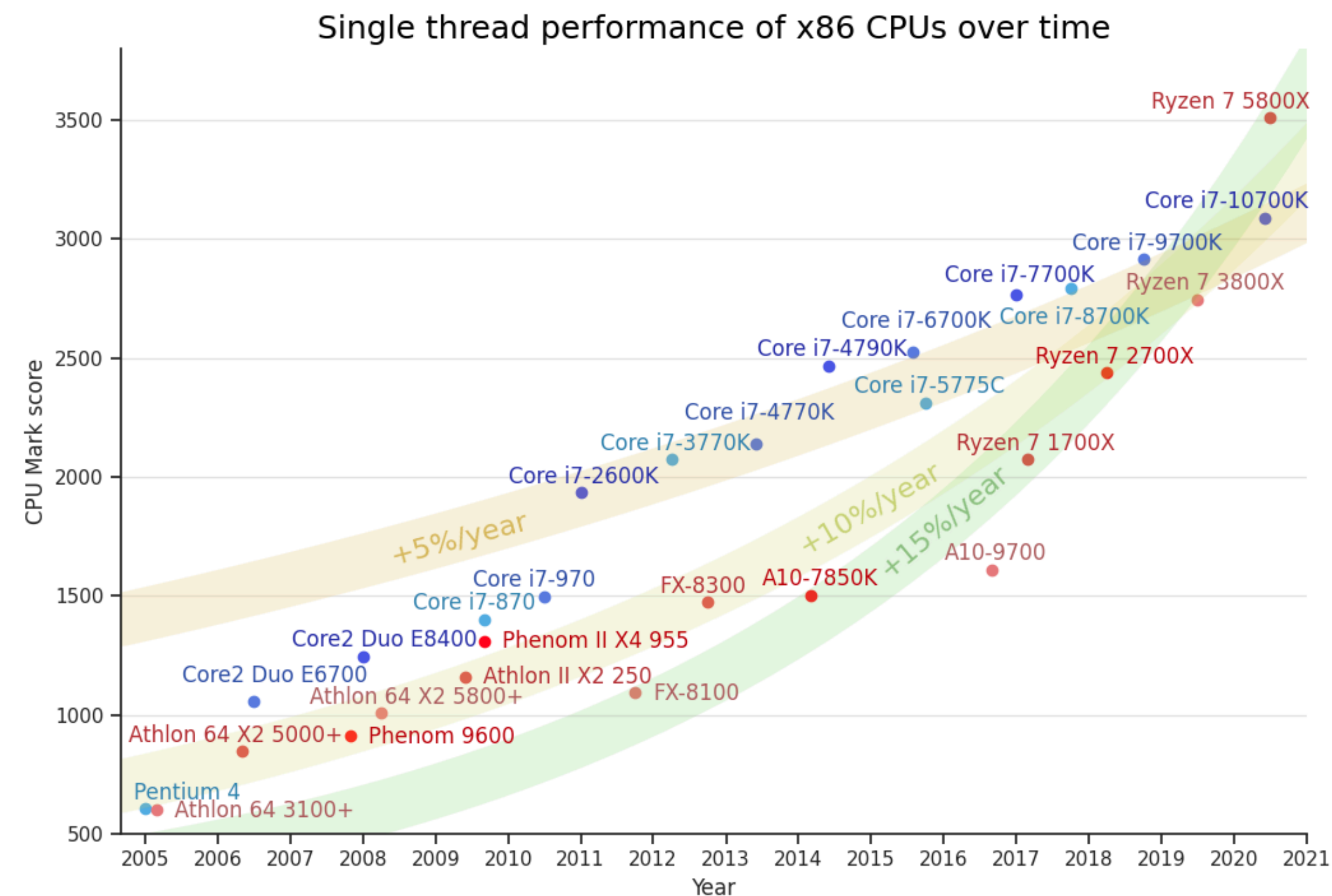
• **ABSTRACT** Recently, flash-based solid-state drives (SSDs) are widely used in industry and academia due to their higher bandwidth and lower latency compared with traditional hard disk drives (HDDs). Furthermore, SSDs with the Non-Volatile Memory Express (NVMe) interface can provide higher performance and ultra-low latency compared with the Serial AT Attachment (SATA) SSDs. Due to their high performance, NVMe SSDs are adopted in many systems as fast storage devices. However, the performance of NVMe SSDs can be negatively affected by I/O access patterns. For example, random write access patterns can have negative impacts on performance due to the unique characteristics of SSDs such as out-of-place update and garbage collection. In this paper, we propose an address remapping scheme to improve the I/O performance of NVMe SSDs. Our proposed scheme transforms random access patterns into sequential access patterns in the NVMe device driver. This allows our scheme to improve the I/O performance of NVMe SSDs while supporting widely used file systems such as EXT4, XFS, BTRFS, and F2FS without any modification to the device. Experimental results show that our proposed scheme can improve the performance of NVMe SSD by up to 64.1% compared with the existing scheme.

• **INDEX TERMS** Flash-based SSDs, NVMe interface, device driver, I/O performance, garbage collection.

Source: Improving I/O Performance via Address Remapping in NVMe Interface  
<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9947049>

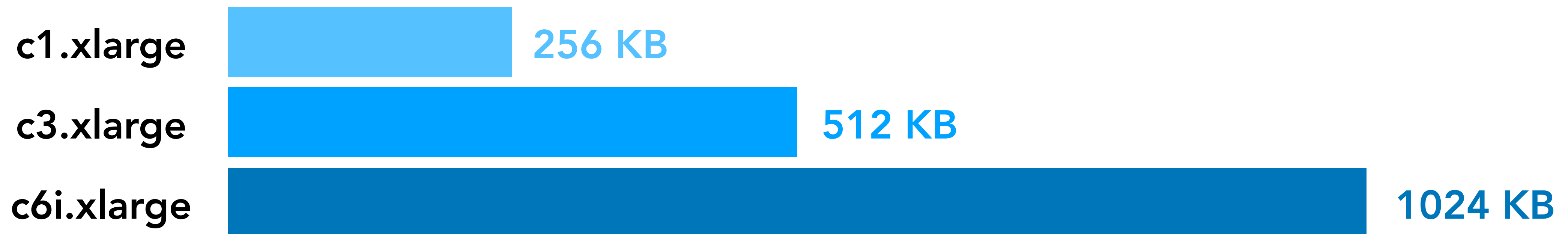


# CPUs are getting faster



Source: Evolution of Single-threaded x86 CPU Performance  
<https://mlech26l.github.io/pages/2020/12/17/cpus.html>

# CPUs are getting faster



L1 Cache



L2 Cache



# CPUs are evolving





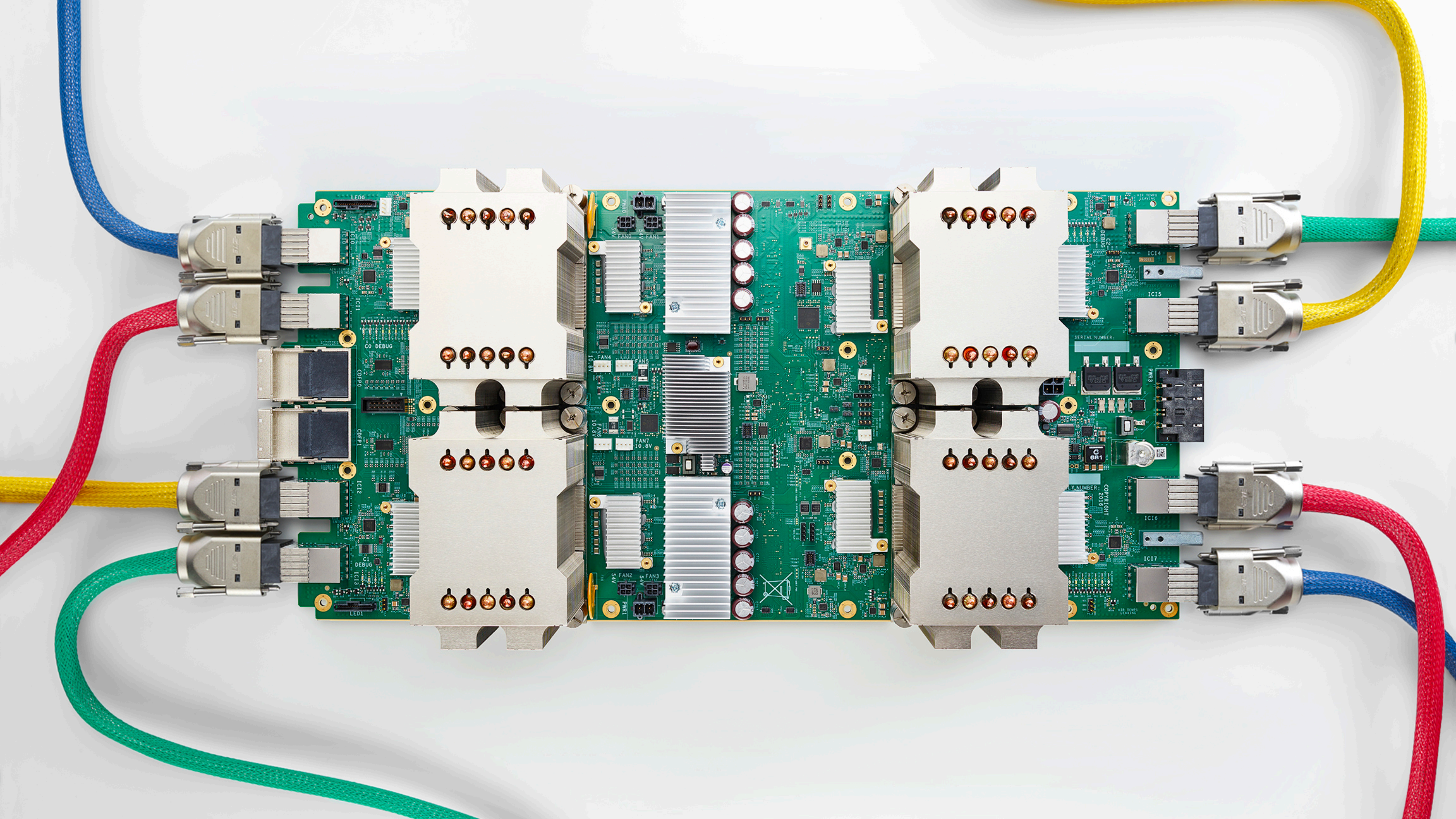




# Networks are getting faster

Machine types	vCPUs*	Memory (GB)	Max number of persistent disks (PDs) <sup>†</sup>	Max total PD size (TB)	Local SSD	Default egress bandwidth (Gbps) <sup>‡</sup>	Tier_1 egress bandwidth (Gbps) <sup>#</sup>
c2-standard-4	4	16	128	257	Yes	10	N/A
c2-standard-8	8	32	128	257	Yes	16	N/A
c2-standard-16	16	64	128	257	Yes	32	N/A
c2-standard-30	30	120	128	257	Yes	32	50
c2-standard-60	60	240	128	257	Yes	32	100







# Latency numbers

L1 Cache Reference	0.5ns	}	Much larger caches means we would hit these caches more often
L2 Cache Reference	7ns		
Main Memory Reference	100ns		3-4x higher throughput
Send 1K bytes over 1 Gbps network	10,000ns		Much larger NICs
Read 4K bytes from SSD	150,000ns		Much faster NVMe drives
Read 1MB sequentially from memory	250,000ns		20,000ns with DDR5
Read 1MB from SSD	1,000,000ns		100,000ns with NVMe
Disk Seek	10,000,000ns		20,000ns with NVMe

Source: <https://gist.github.com/jboner/2841832>



# The Free Lunch Is Over

## A Fundamental Turn Toward Concurrency in Software

By Herb Sutter

The biggest sea change in software development since the OO revolution is knocking at the door, and its name is Concurrency.

*This article appeared in [Dr. Dobb's Journal](#), 30(3), March 2005. A much briefer version under the title "[The Concurrency Revolution](#)" appeared in [C/C++ Users Journal](#), 23(2), February 2005.*

**Update note: The CPU trends graph last updated August 2009 to include current data and show the trend continues as predicted. The rest of this article including all text is still original as first posted here in December 2004.**

Your free lunch will soon be over. What can you do about it? What *are* you doing about it?

The major processor manufacturers and architectures, from Intel and AMD to Sparc and PowerPC, have run out of room with most of their traditional approaches to boosting CPU performance. Instead of driving clock speeds and straight-line instruction throughput ever higher, they are instead turning *en masse* to hyperthreading and multicore architectures. Both of these features are already available on chips today; in particular, multicore is available on current PowerPC and Sparc IV processors, and is coming in 2005 from Intel and AMD. Indeed, the big theme of the 2004 In-Stat/MDR Fall Processor Forum was multicore devices, as many companies showed new or updated multicore processors. Looking back, it's not much of a stretch to call 2004 the year of multicore.

And that puts us at a fundamental turning point in software development, at least for the next few years and for applications targeting general-purpose desktop computers and low-end servers (which happens to account for the vast bulk of the dollar value of software sold today). In this article, I'll describe the changing face of hardware, why it suddenly does matter to software, and how specifically the concurrency revolution matters to you and is going to change the way you will likely be writing software in the future.

Arguably, the free lunch has already been over for a year or two, only we're just now noticing.

## The Free Performance Lunch

There's an interesting phenomenon that's known as "Andy giveth, and Bill taketh away." No matter how fast processors get, software consistently finds new ways to eat up the extra speed. Make a CPU ten times as fast, and software will usually find ten times as much to do (or, in some cases, will feel at liberty to do it ten times less efficiently). Most classes of applications have enjoyed free and regular performance gains for several



# Free lunch?

“No matter how fast processors get, software consistently finds new ways to eat up the extra speed. Make a CPU ten times as fast, and software will usually find ten times as much to do (or, in some cases, will feel at liberty to do it ten times less efficiently)”

Herb Sutter

<http://www.gotw.ca/publications/concurrency-ddj.htm>



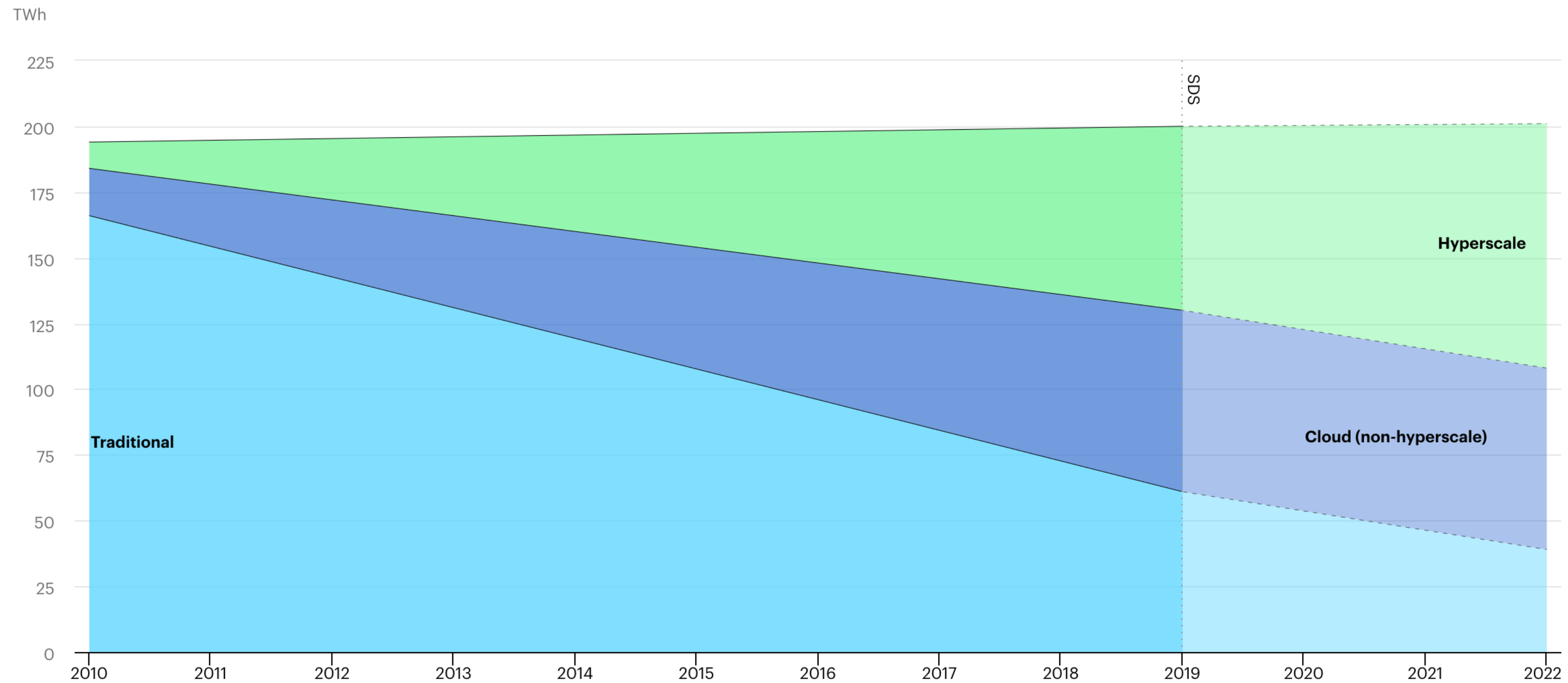
# Free lunch?

“Efficiency and performance optimization will get more, not less, important”

Herb Sutter

<http://www.gotw.ca/publications/concurrency-ddj.htm>





IEA. Licence: CC BY 4.0

● Traditional ● Cloud (non-hyperscale) ● Hyperscale

<https://www.iea.org/data-and-statistics/charts/global-data-centre-energy-demand-by-data-centre-type-2010-2022>



## Four principles for writing energy and carbon-efficient software

[Subscribe](#)[Article Options](#) **Jennifer\_Huffstetler** 

Employee

04-20-2023

 2  0  2,748

Intel is known for its industry leading hardware solutions and for decades we have been a strong environmental steward. This stewardship has included efforts to accelerate sustainability through increasingly efficient products to developing innovative energy conserving features for our processors and platforms. With this work, Intel is leading the enablement of a more sustainable data center and compute industry. With global data center energy consumption accounting for 0.9-1.3% of total global energy demand ([IEA, Sept 2022](#)), that's a great thing.

But global and substantial impact is best achieved by looking at *all* areas in compute that can provide sustainability value. According to Intel estimates, infrastructure and software inefficiency count for over 50% of greenhouse gas (GHG) emissions in the data center. This illuminates that it's not only what's in your data center that matters, but how you use it. For example, infrastructure inefficiency can be addressed through greater server utilization.

For this blog, though, I will focus on software inefficiency as there are ways to make your software more carbon and energy efficient without compromising its functionality or performance. In fact, making your software more energy and carbon efficient is likely the *fastest* way to improve the environmental impact of your IT operations. Here are four principles that can guide you in designing and developing software that minimizes its energy consumption and carbon footprint.

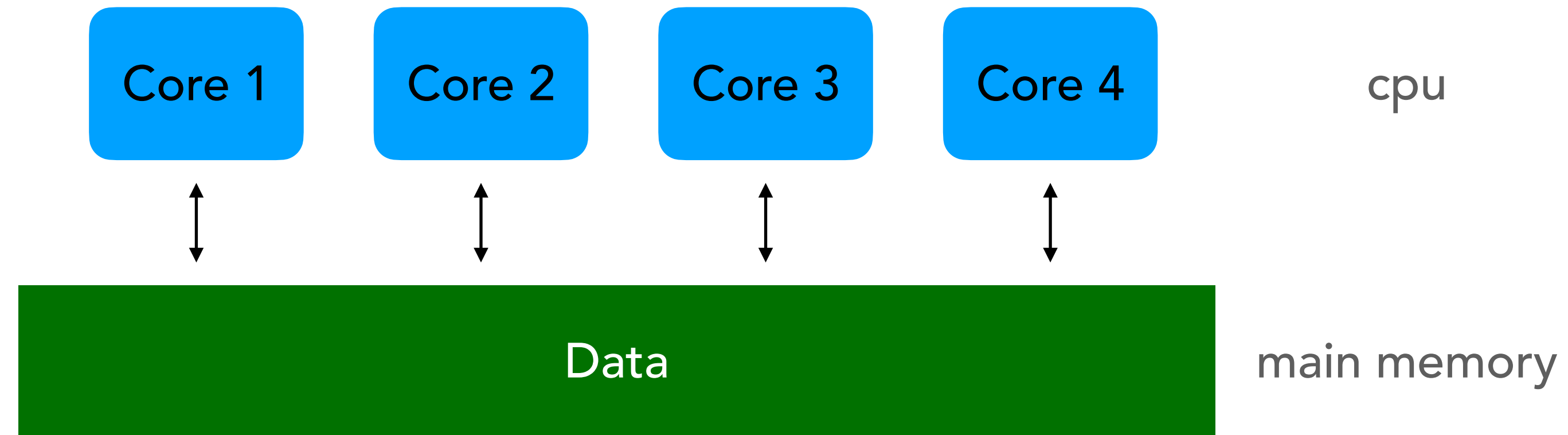
<https://community.intel.com/t5/Blogs/Thought-Leadership/Big-Ideas/Four-principles-for-writing-energy-and-carbon-efficient-software>







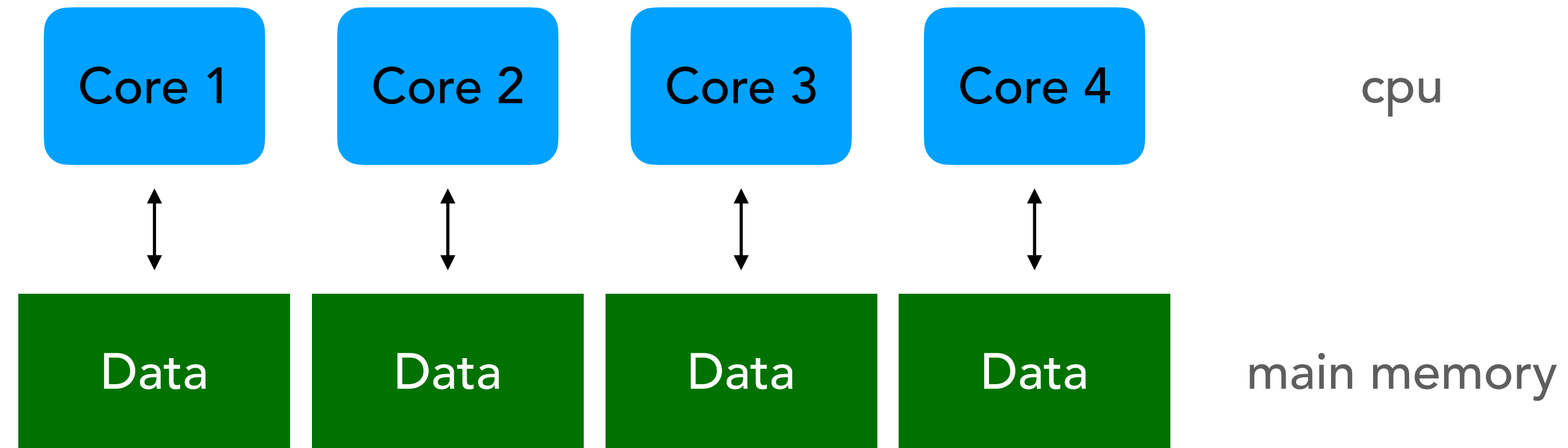
# Thread per core



Shared Everything Architecture



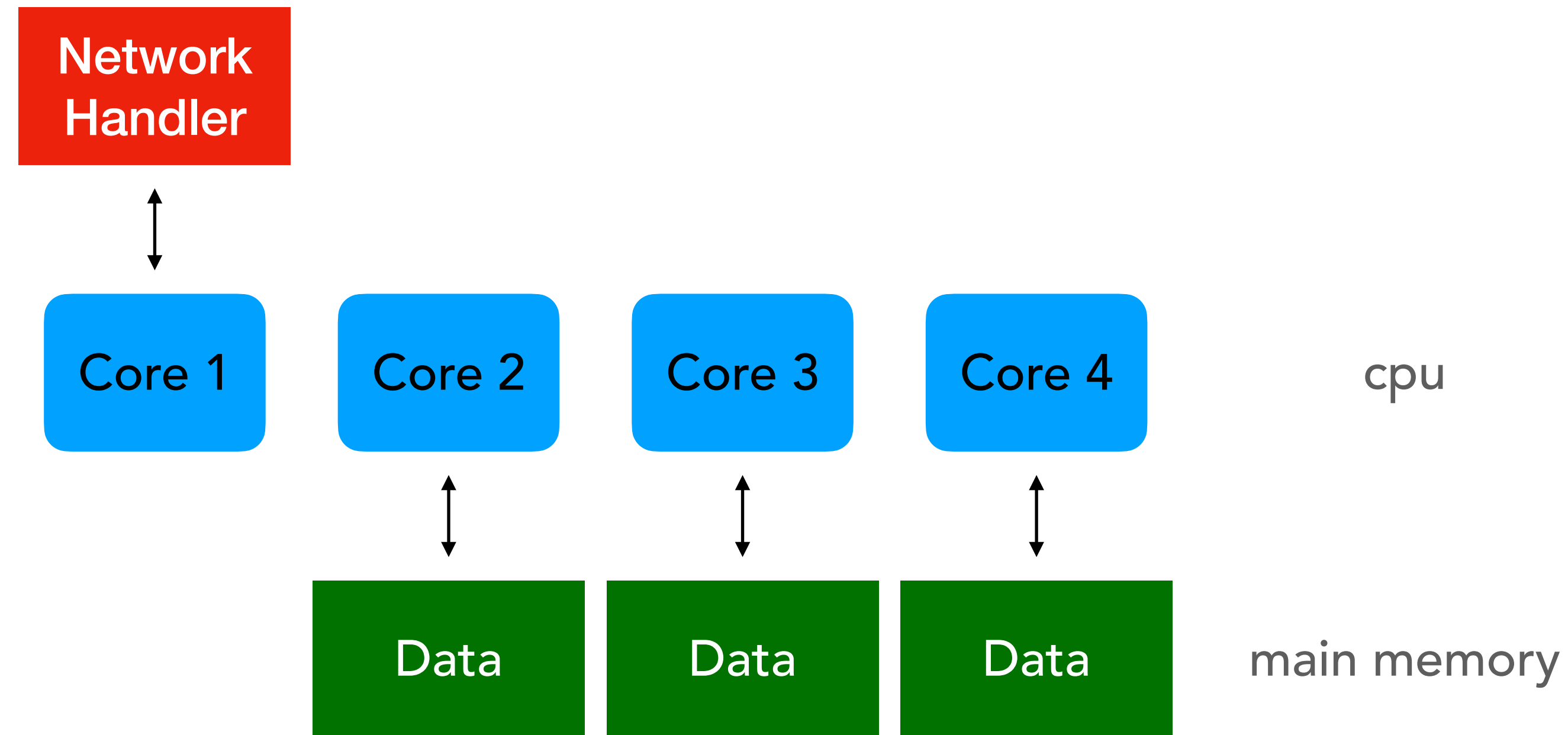
# Thread per core



Shared Nothing Architecture



# Thread per core



Shared Nothing Architecture

Source: The Impact of Thread-Per-Core Architecture on Application Tail Latency  
[https://helda.helsinki.fi/bitstream/handle/10138/313642/tpc\\_ancs19.pdf](https://helda.helsinki.fi/bitstream/handle/10138/313642/tpc_ancs19.pdf)



# SEASTAR.

[Home](#)[Architecture »](#)[Blog](#)[Seastar Applications](#)[Documentation](#)[Frequently Asked Questions](#)[ScyllaDB](#)

Seastar is an advanced, open-source C++ framework for high-performance server applications on modern hardware. Seastar is used in [Scylla](#), a high-performance NoSQL database compatible with Apache Cassandra. Applications using Seastar can run on [Linux](#) or [OSv](#).

[Get Started](#)

Seastar is the first framework to bring together a set of extreme architectural innovations, including:

[Shared-nothing design](#): Seastar uses a shared-nothing model that shards all requests onto individual cores.

[High-performance networking](#): Seastar offers a choice of network stack, including conventional Linux networking for ease of development, DPDK for fast user-space networking on Linux, and native networking on OSv.

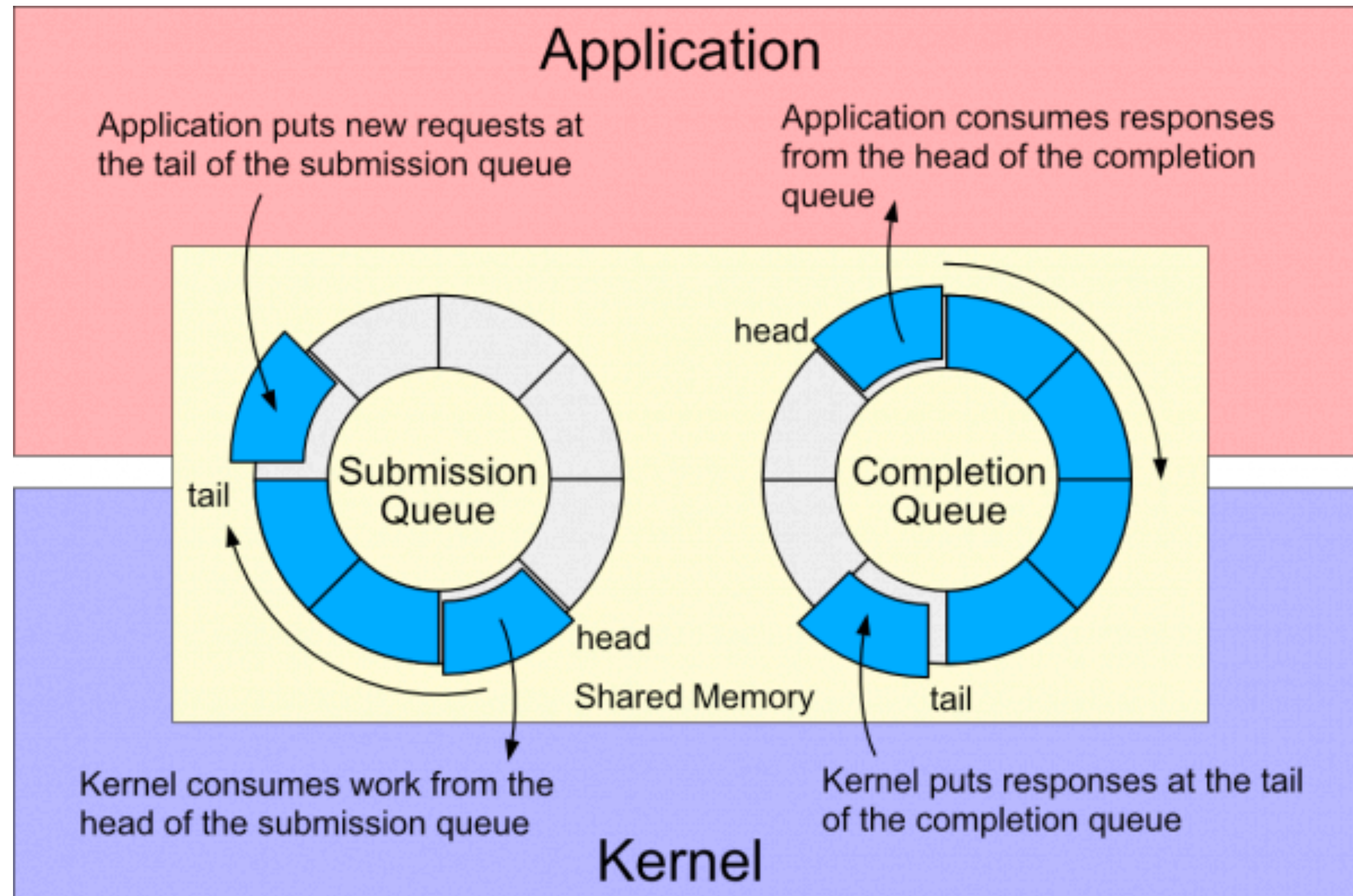
[Futures and promises](#): An advanced new model for concurrent applications that offers C++ programmers both high performance and the ability to create comprehensible, testable high-quality code.

[Message passing](#): A design for sharing information between CPU cores without time-consuming locking

<https://seastar.io>



# io\_uring

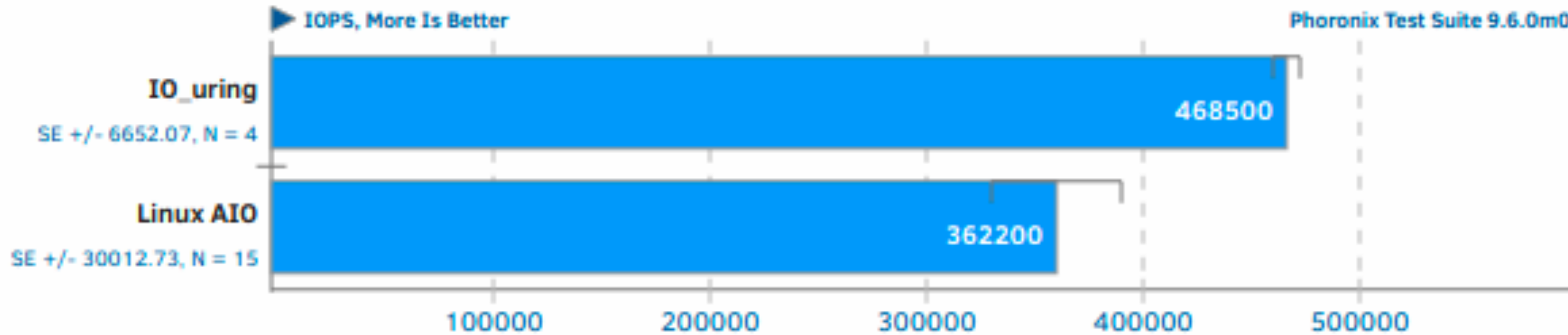


Source: <https://developers.redhat.com/articles/2023/04/12/why-you-should-use-iouring-network-io>



# Flexible IO Tester v3.18

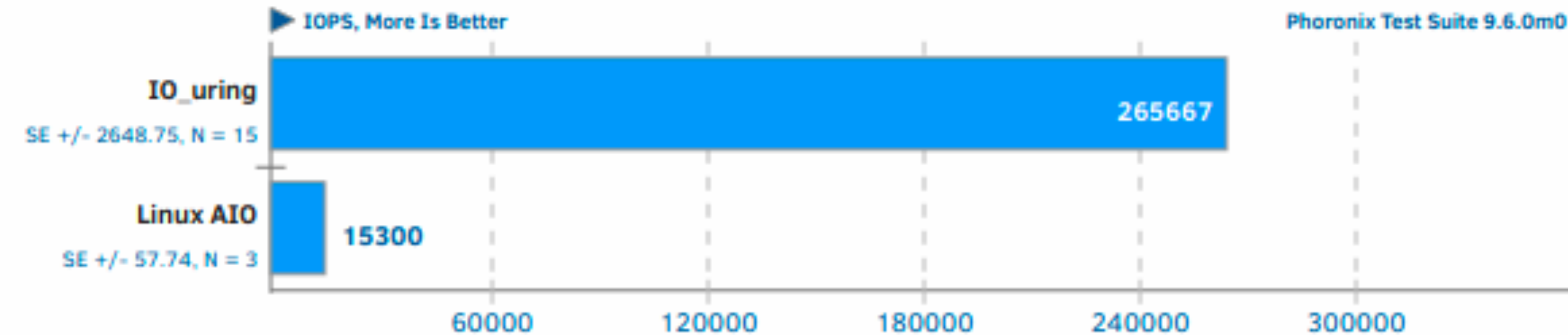
Engine Comparison (Type: Random Write - - Buffered: No - Direct: No - Block Size: 4KB - Disk Target: Default Test Directory)



1. (CC) gcc options: -rdynamic -std=gnu99 -ffast-math -include -O3 -U\_FORTIFY\_SOURCE -march=native -lrt -laio -lz -lpthread -lm -ldl

# Flexible IO Tester v3.18

Engine Comparison (Type: Random Read - - Buffered: Yes - Direct: No - Block Size: 4KB - Disk Target: Default Test Directory)



1. (CC) gcc options: -rdynamic -std=gnu99 -ffast-math -include -O3 -U\_FORTIFY\_SOURCE -march=native -lrt -laio -lz -lpthread -lm -ldl

<https://www.phoronix.com/news/Linux-5.6-IO-uring-Tests>



Product

Solutions

Open Source

Pricing

Search

/

Sign in

Sign up

libuv / libuv

Public

Notifications

Fork3.4k

Star21.5k

<> Code

Issues106

Pull requests47

Discussions

Actions

Security

Insights

linux: introduce io\_uring support #3952

New issue

Merged

bnoordhuis merged 1 commit into libuv:v1.x from bnoordhuis:iou on Apr 18

Conversation52

Commits1

Checks36

Files changed7

+662-35

bnoordhuis commented on Apr 12 • edited

Member

Add io\_uring support for several asynchronous file operations:

• read, write

• fsync, fdatsync

• stat, fstat, lstat

io\_uring is used when the kernel is new enough, otherwise libuv simply falls back to the thread pool.

Performance looks great; an 8x increase in throughput has been observed.

This work was sponsored by ISC, the Internet Systems Consortium.

55

75

54

bnoordhuis force-pushed the iou branch from 8dd8b3e to efc9bbf last month

Compare

bnoordhuis mentioned this pull request on Apr 12

Use io\_uring for read/write/fsync on Linux #2322

7 tasks

Reviewers

trevnorris

✓

santigimeno

✓

clason

espoal

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

https://github.com/libuv/libuv/pull/3952

39

@suhailpatel | YOW! 2023



# Systems programming languages





# New tricks

src/lib.rs

```
use pyo3::prelude::*;

/// Formats the sum of two numbers as string.
#[pyfunction]
fn sum_as_string(a: usize, b: usize) -> PyResult<String> {
    Ok((a + b).to_string())
}

/// A Python module implemented in Rust. The name of this function must match
/// the `lib.name` setting in the `Cargo.toml`, else Python will not be able to
/// import the module.
#[pymodule]
fn string_sum(_py: Python<'>, m: &PyModule) -> PyResult<()> {
    m.add_function(wrap_pyfunction!(sum_as_string, m)?)?;
    Ok(())
}
```

Finally, run `maturin develop`. This will build the package and install it into the Python virtualenv previously created and activated. The package is then ready to be used from `python`:

```
$ maturin develop
# lots of progress output as maturin runs the compilation...
$ python
>>> import string_sum
>>> string_sum.sum_as_string(5, 20)
'25'
```

<https://pyo3.rs/v0.18.3/>



**Table 4.** Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Source: Energy Efficiency across Programming Languages  
<https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf>



# Date Parsing in Python

```
$ python3
Python 3.10.9 (main, Dec 15 2022, 17:11:09) [Clang 14.0.0 (clang-1400.0.29.202)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import datetime
>>> datetime.datetime.fromisoformat('2023-06-15T09:00:00-05:00')
datetime.datetime(2023, 6, 15, 9, 0, tzinfo=datetime.timezone(datetime.timedelta(seconds=-18000)))
```



# Date Parsing in Python with Rust!

```
$ pip install maturin
```

```
$ maturin new riso8601
```

✓ 🙋 Which kind of bindings to use?

📖 Documentation: <https://maturin.rs/bindings.html> · pyo3

✨ Done! New project created riso8601

```
$ cd riso8601
```

```
$ ls
```

Cargo.lock

Cargo.toml

pyproject.toml src

target

# Date Parsing in Python with Rust!

src/lib.rs

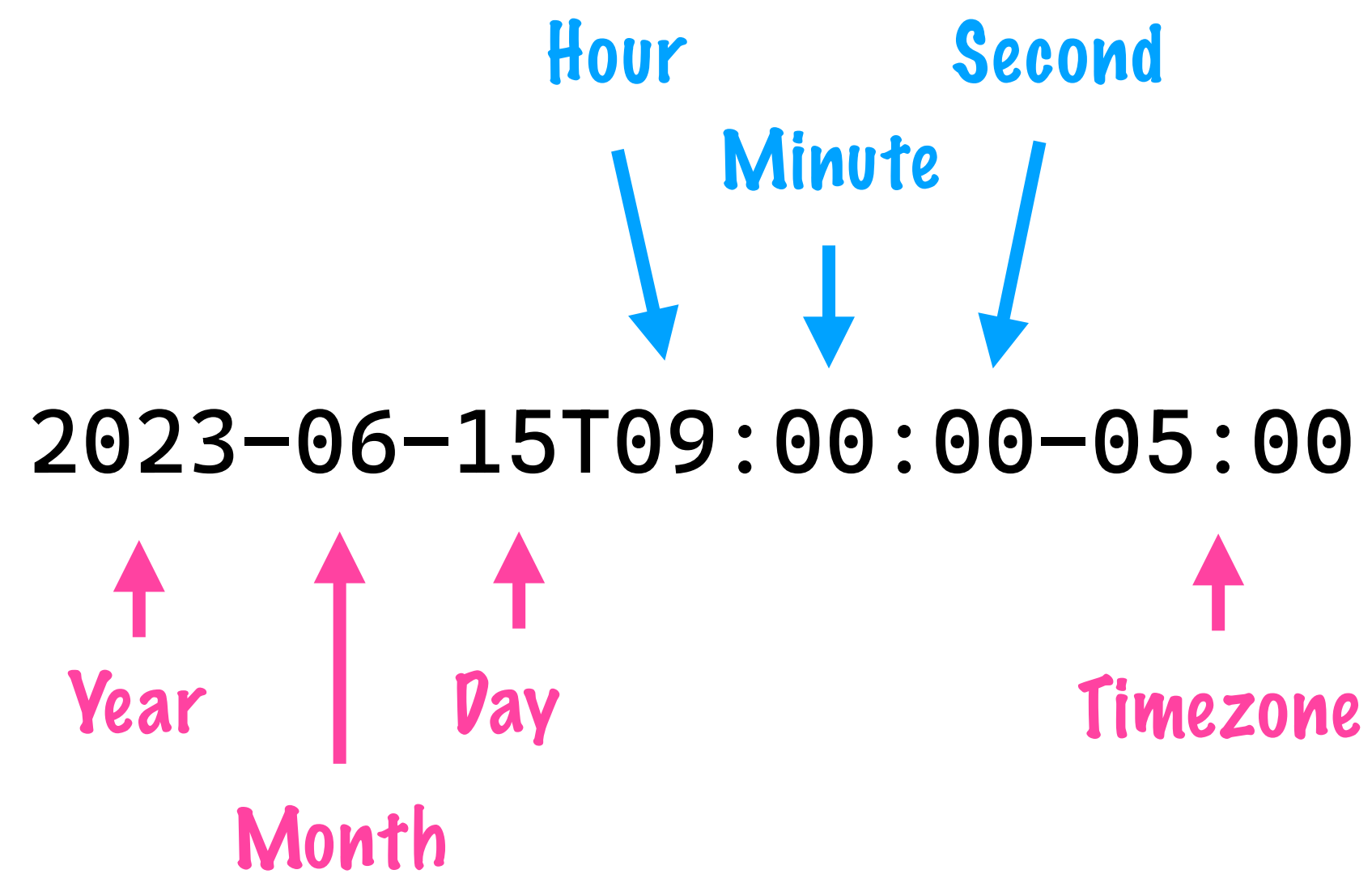
```
#[pyfunction]
fn parse_datetime<'p>(py: Python<'p>, input: &str) → PyResult<&'p PyDateTime> {
    // Our implementation goes here ...
}

#[pymodule]
fn riso8601(_py: Python, m: &PyModule) → PyResult<()> {
    m.add_wrapped(wrap_pyfunction!(parse_datetime))?;
    Ok(())
}
```



# Date Parsing in Python with Rust!

Hour Minute Second  
2023-06-15T09:00:00-05:00  
Year Month Day Timezone



```
datetime.datetime(2023, 6, 15, 9, 0, tzinfo=  
    datetime.timezone(datetime.timedelta(days=-1, seconds=68400)))
```

src/lib.rs

```
#[pyfunction]
fn parse_datetime<'p>(py: Python<'p>, input: &str) → PyResult<&'p PyDateTime> {
    let year: i32 = match input[point..point + 4].parse() {
        Ok(val) if val > 0 ⇒ val,
        Ok(_) ⇒ return Err(ParseError::new_err("year needs to be above 0")),
        _ ⇒ return Err(ParseError::new_err("invalid time string (year)")),
    };

    // Do a dash check, advance our point position accordingly
    match input[point + 4..point + 5].as_ref() {
        "-" ⇒ point = point + 5,
        _ ⇒ point = point + 4,
    }

    let month: u8 = match input[point..point + 2].parse() {
        Ok(val) if (1..=12).contains(&val) ⇒ val,
        Ok(_) ⇒ return Err(ParseError::new_err("month needs to be between 1-12")),
        _ ⇒ return Err(ParseError::new_err("invalid time string (month)")),
    };

    // More code to split out the primary components from the date string ...

    return PyDateTime::new(py, year, month, day, hour, minute, second, ms, Some(tz));
}
```

<https://github.com/suhailpatel/riso8601/blob/master/src/lib.rs>

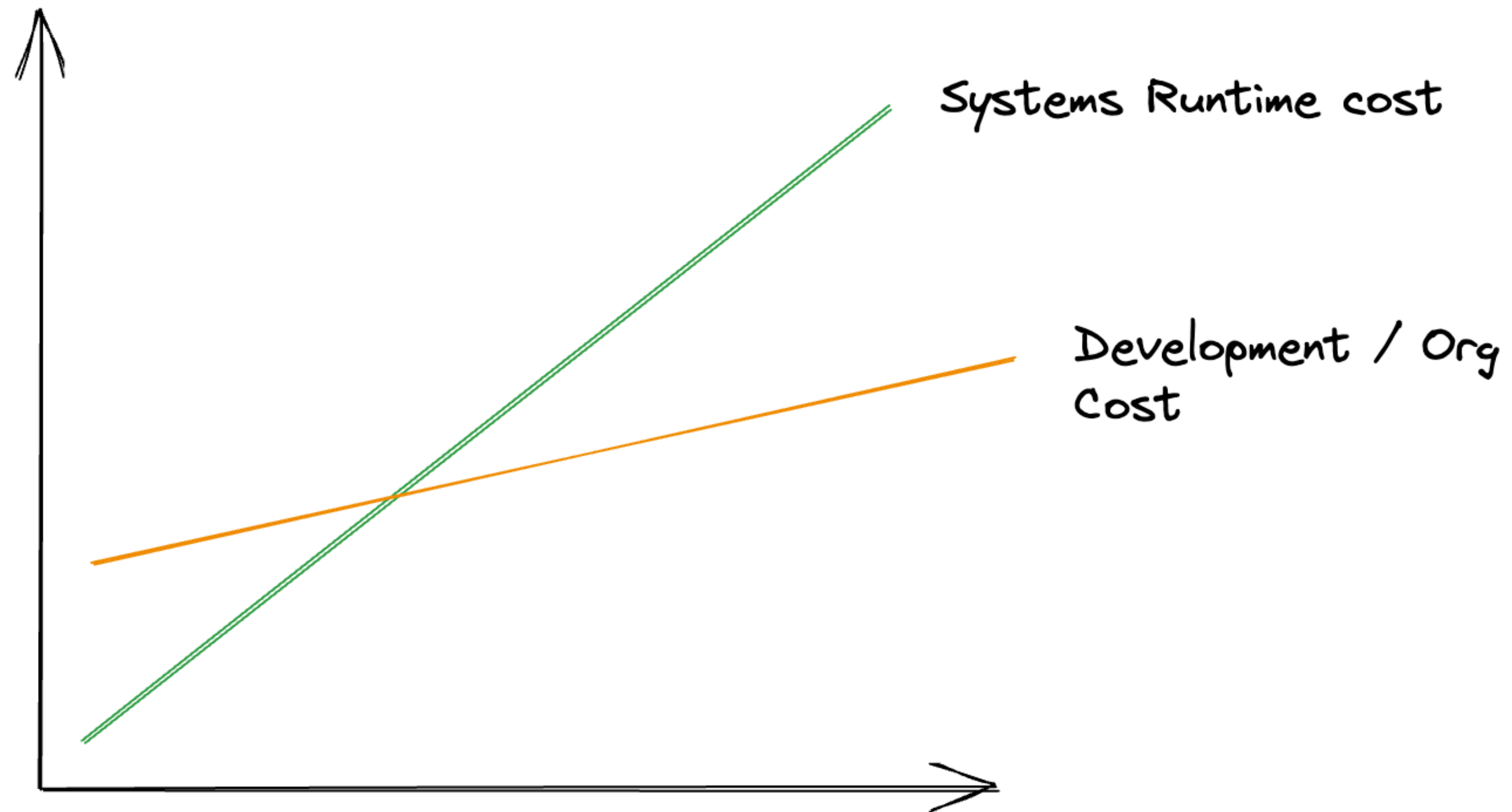


# Date Parsing in Python with Rust!

```
$ python3
Python 3.10.9 (main, Dec 15 2022, 17:11:09) [Clang 14.0.0 (clang-1400.0.29.202)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import riso8601
>>> riso8601.parse_datetime("2023-06-15T09:00:00-05:00")
datetime.datetime(2023, 6, 15, 9, 0, tzinfo=datetime.timezone(datetime.timedelta(seconds=-18000)))

>>> riso8601.parse_datetime("bad timestamp")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
riso8601.ParseError: invalid time string (year)
```

# Inflection points

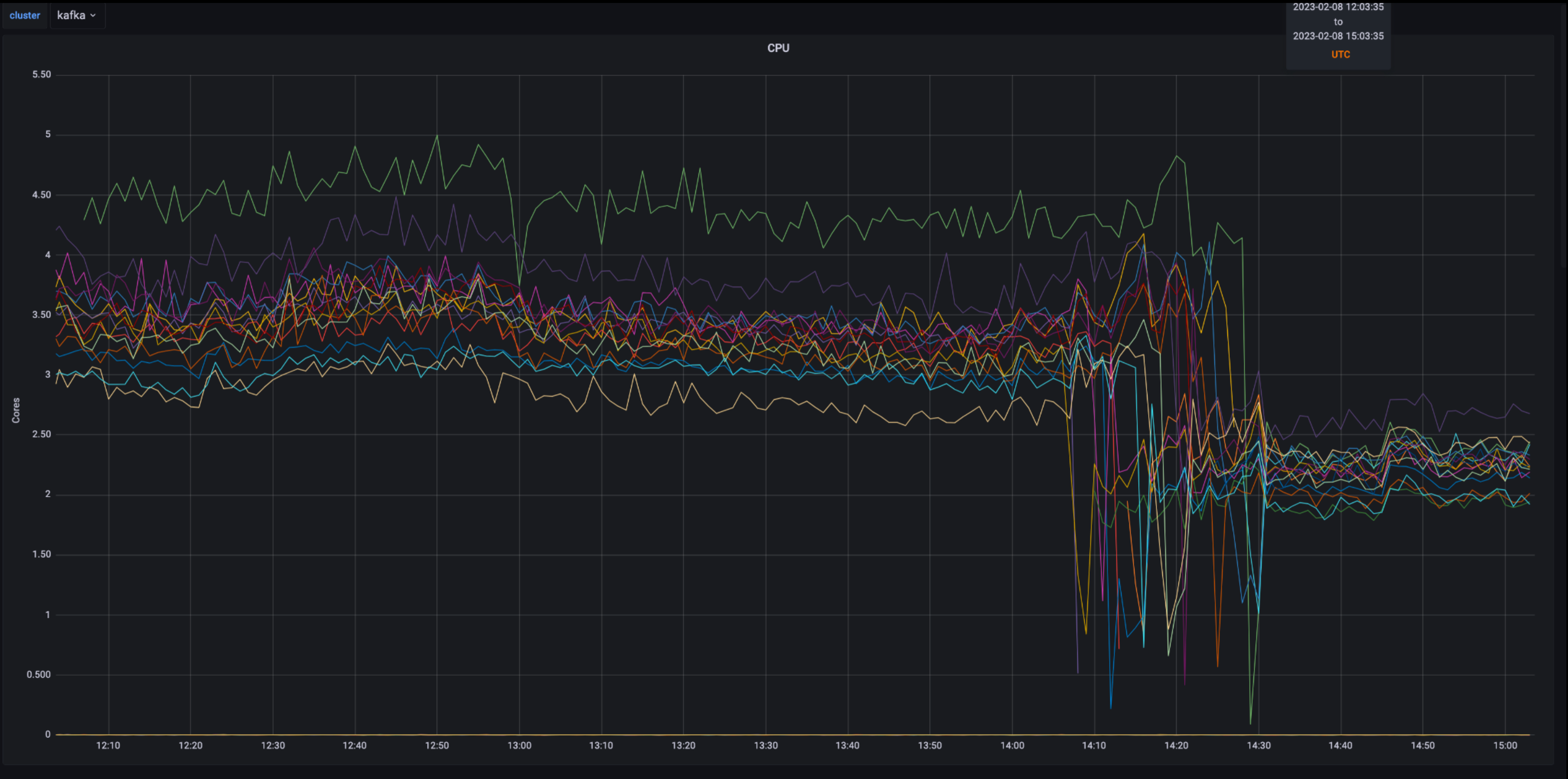




# New tricks

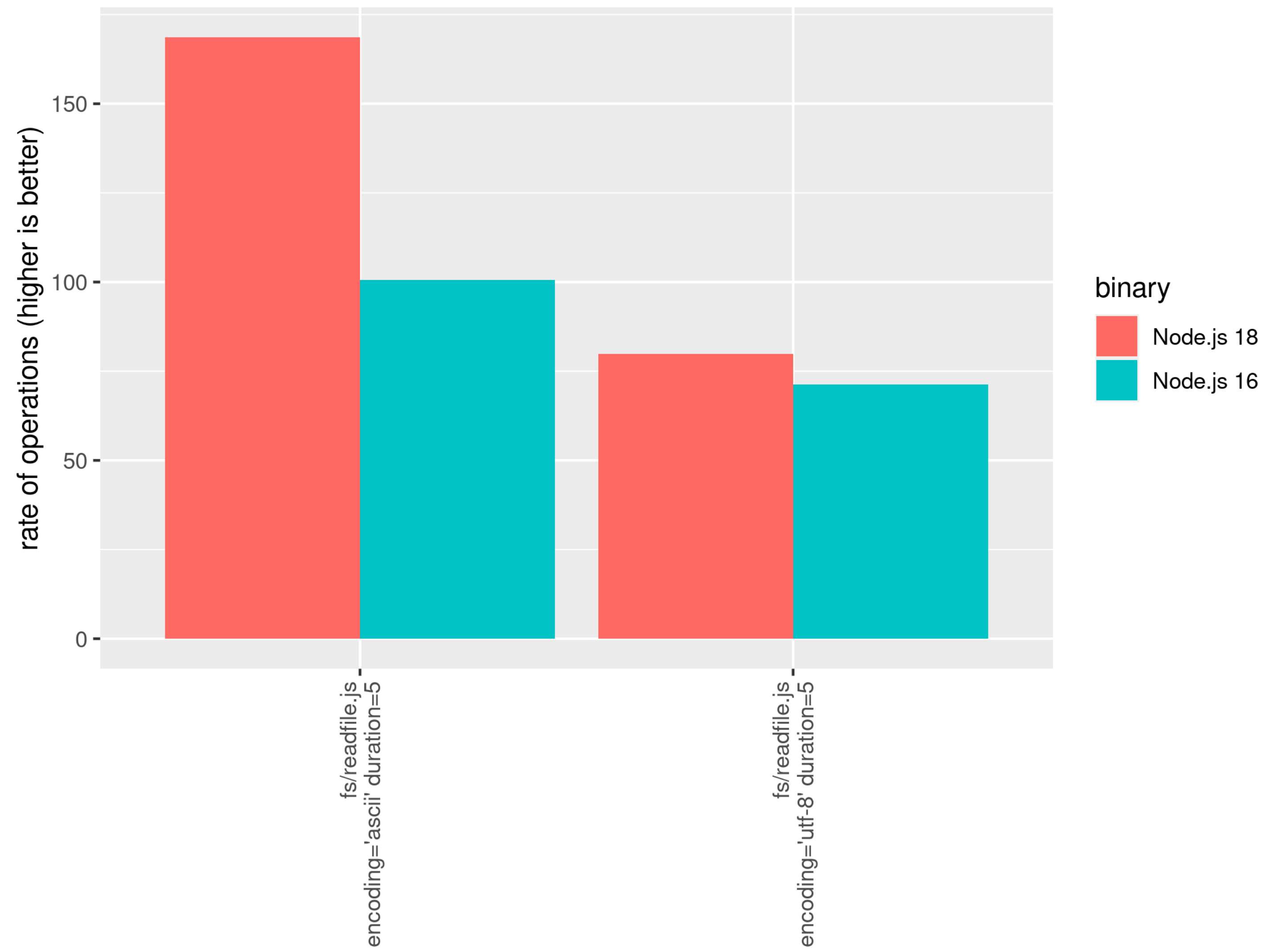


**-XX:+UseZGC -Xmx=<max heap size>**





# New tricks



# New tricks

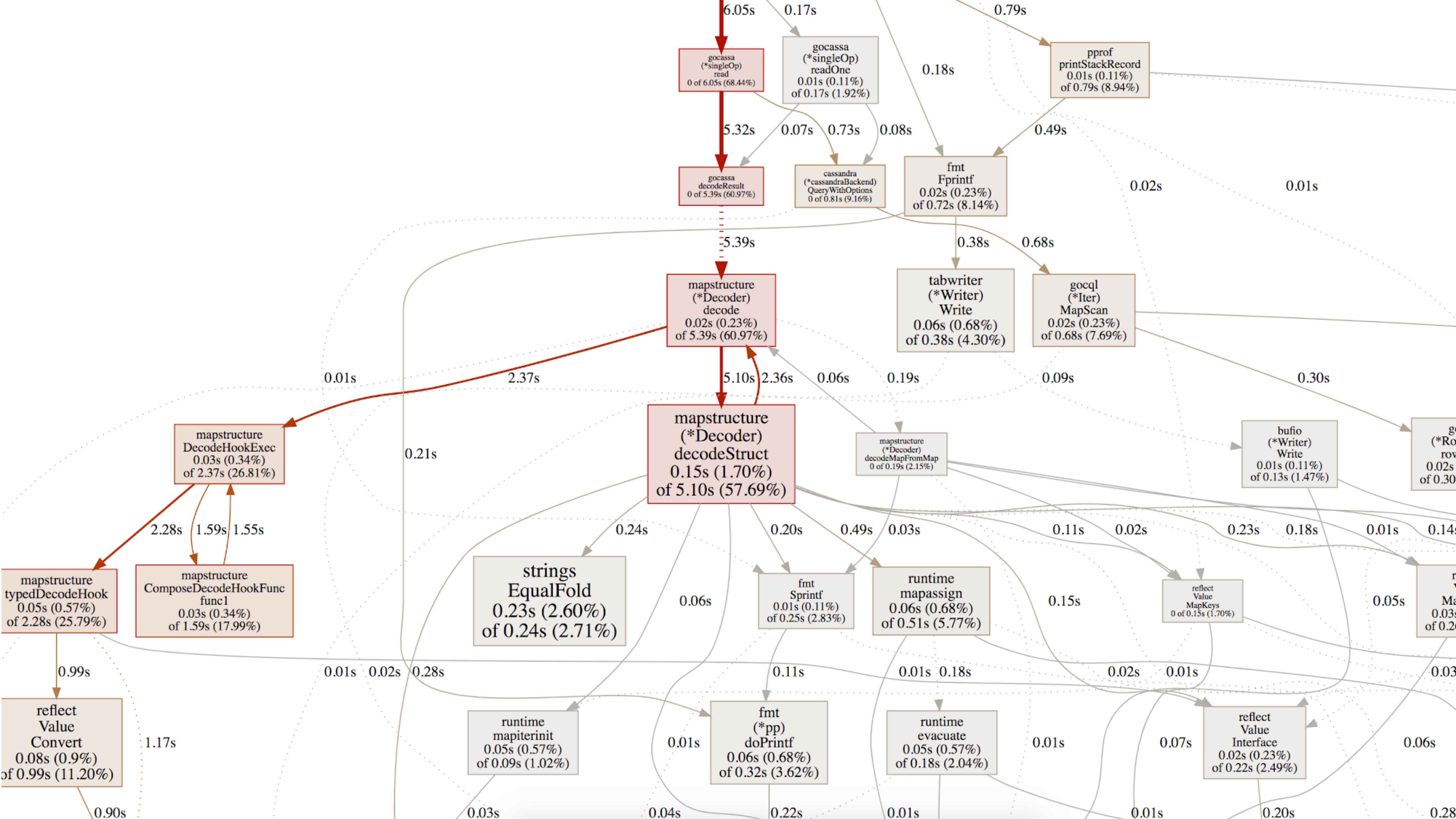
## Faster CPython

CPython 3.11 is an average of **25% faster** than CPython 3.10 as measured with the **pyperformance** benchmark suite, when compiled with GCC on Ubuntu Linux. Depending on your workload, the overall speedup could be 10–60%.

This project focuses on two major areas in Python: **Faster Startup** and **Faster Runtime**. Optimizations not covered by this project are listed separately under **Optimizations**.

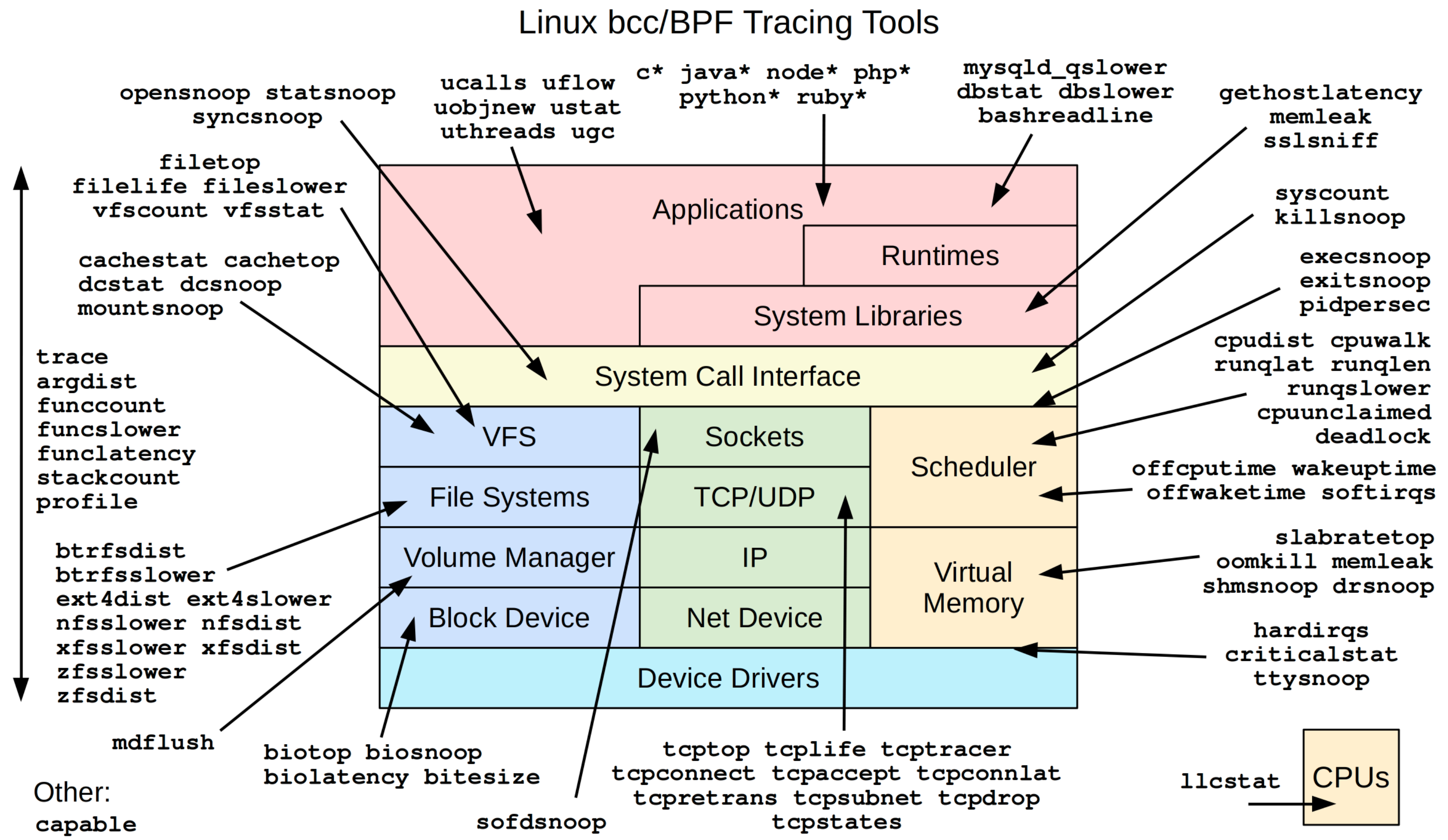
<https://docs.python.org/3/whatsnew/3.11.html>







# eBPF



<https://github.com/iovisor/bcc#tools> 2019

<https://github.com/iovisor/bcc>







# llama.cpp optimisations

ggerganov / llama.cppPublic

NotificationsFork4.1kStar28.5k

<> Code

Issues243

**Pull requests54**

Discussions

Actions

Projects4

Wiki

Security

Insights

Make loading weights 10-100x faster #613

Mergedjart merged 9 commits into ggerganov:master from jart:loader on Mar 30

Conversation37Commits9Checks22Files changed11

jart commented on Mar 30

Contributor

This is a breaking change that's going to give us three benefits:

1. Your inference commands should load 100x faster
2. You may be able to safely load models 2x larger
3. You can run many concurrent inference processes

This was accomplished by changing the file format so we can mmap() weights directly into memory without having to read() or copy them thereby ensuring the kernel can make its file cache pages directly accessible to our inference processes; and secondly, that the file cache pages are much less likely to get evicted (which would force loads to hit disk) because they're no longer competing with memory pages that were needlessly created by gigabytes of standard i/o.

The new file format supports single-file models like LLaMA 7b, and it also supports multi-file models like LLaMA 13B. Our Python tool now merges the foo.1, foo.2, etc. files back into a single file so that the C++ code which maps it doesn't need to reshape data every time. That's made llama.cpp so much simpler. Much of its load code has now been deleted.

Furthermore, this change ensures that tensors are aligned properly on a 32-byte boundary. That opens the door to seeing if we can get

Reviewers

sw

pgoodman

mgy

bakkot

Green-Sky

ggerganov

Assignees

No one assigned

Labels

breaking changeperformance

Projects

None yet

Milestone

<https://github.com/ggerganov/llama.cpp/pull/613>

57

@suhailpatel | YOW! 2023

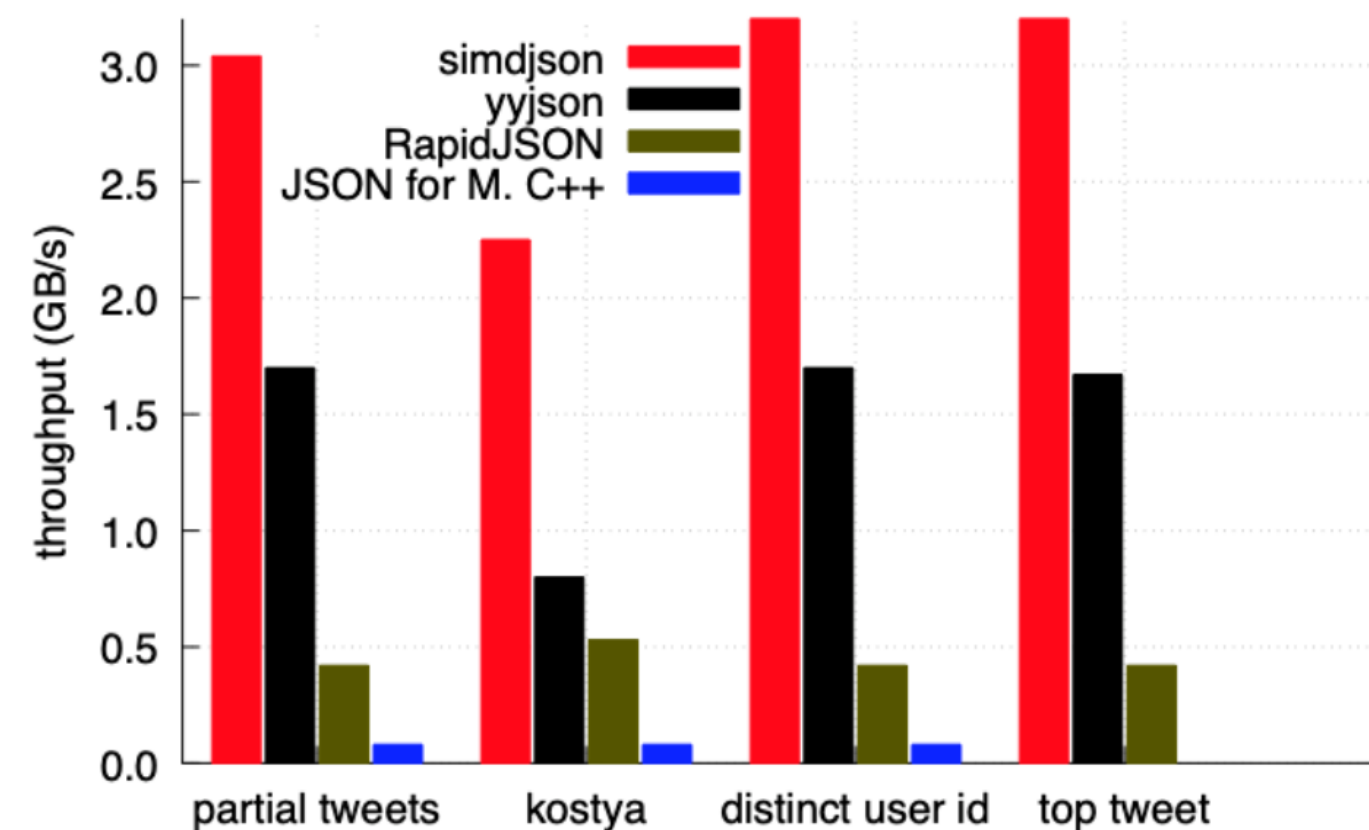


# Faster JSON parsing with simdjson

## Performance results

The simdjson library uses three-quarters less instructions than state-of-the-art parser [RapidJSON](#). To our knowledge, simdjson is the first fully-validating JSON parser to run at [gigabytes per second](#) (GB/s) on commodity processors. It can parse millions of JSON documents per second on a single core.

The following figure represents parsing speed in GB/s for parsing various files on an Intel Skylake processor (3.4 GHz) using the GNU GCC 10 compiler (with the -O3 flag). We compare against the best and fastest C++ libraries on benchmarks that load and process the data. The simdjson library offers full unicode ([UTF-8](#)) validation and exact number parsing.



The simdjson library offers high speed whether it processes tiny files (e.g., 300 bytes) or larger files (e.g., 3MB). The following plot presents parsing speed for [synthetic files over various sizes generated with a script](#) on a 3.4 GHz Skylake processor (GNU GCC 9, -O3).

<https://github.com/simdjson/simdjson>

# Wait, but why?

Many of the systems (apps, services, databases, caches, queues)  
that we build/rely on are grounded on quite poor assumptions  
for the hardware of today



# Wait, but why?

Many of the systems (apps, services, databases, caches, queues) that we build/rely on are grounded on quite poor assumptions for the hardware of today

Software can keep pace, but there's some work needed to yield huge results, power new kinds of systems and reduce compute costs



The Joy of Building Large Scale Systems

# Thank you!

Suhail Patel | @suhailpatel | <https://suhailpatel.com>

YOW! 2023