# BPF Tales

## Or: Why Did I Recompile the Kernel Just to Average Numbers?

Open Source Summit Europe 2020

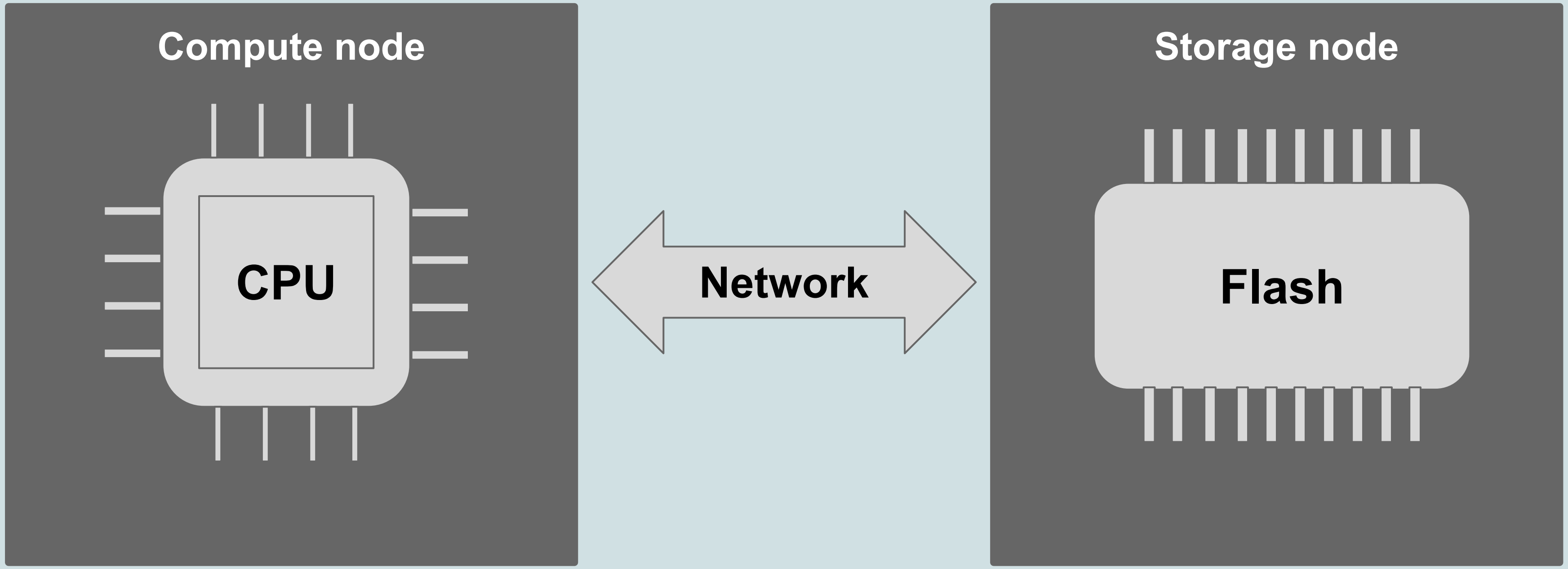
## We need some context

## The data deluge

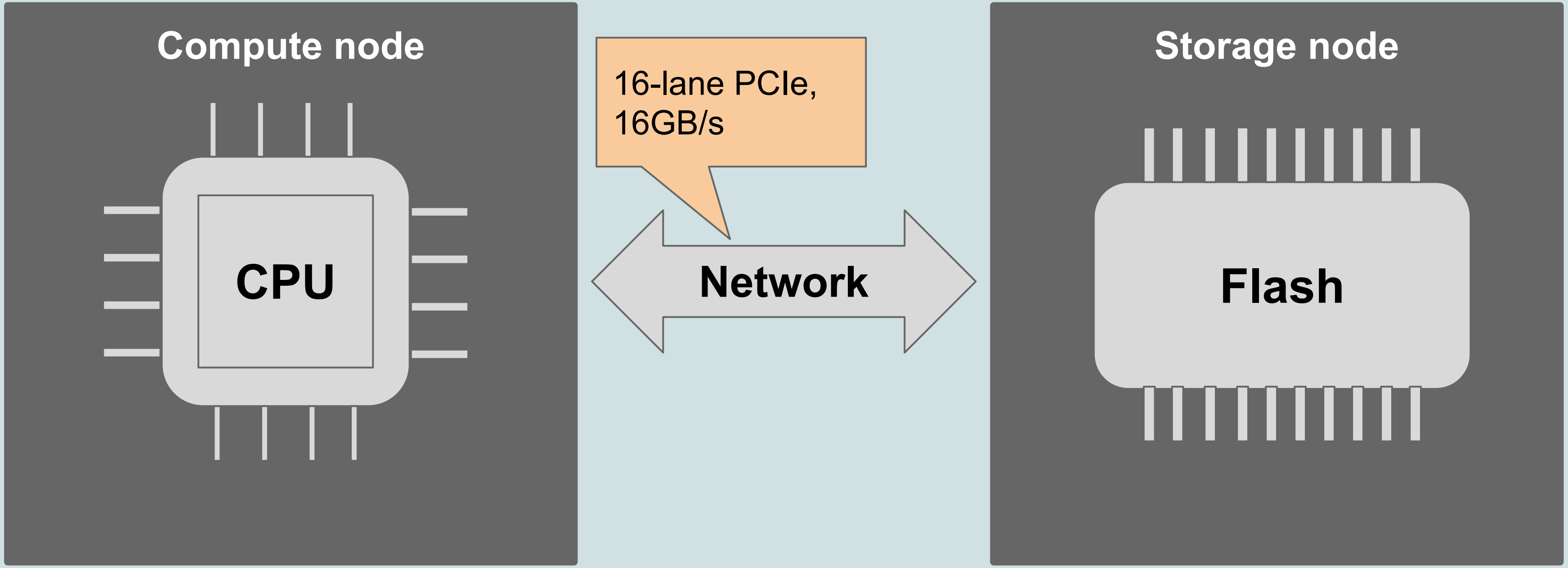
- 44ZB of stored data 2020
- 53% of companies use Big Data analytics
- $77 billion worth by 2023

https://kommandotech.com/statistics/big-data-statistics/

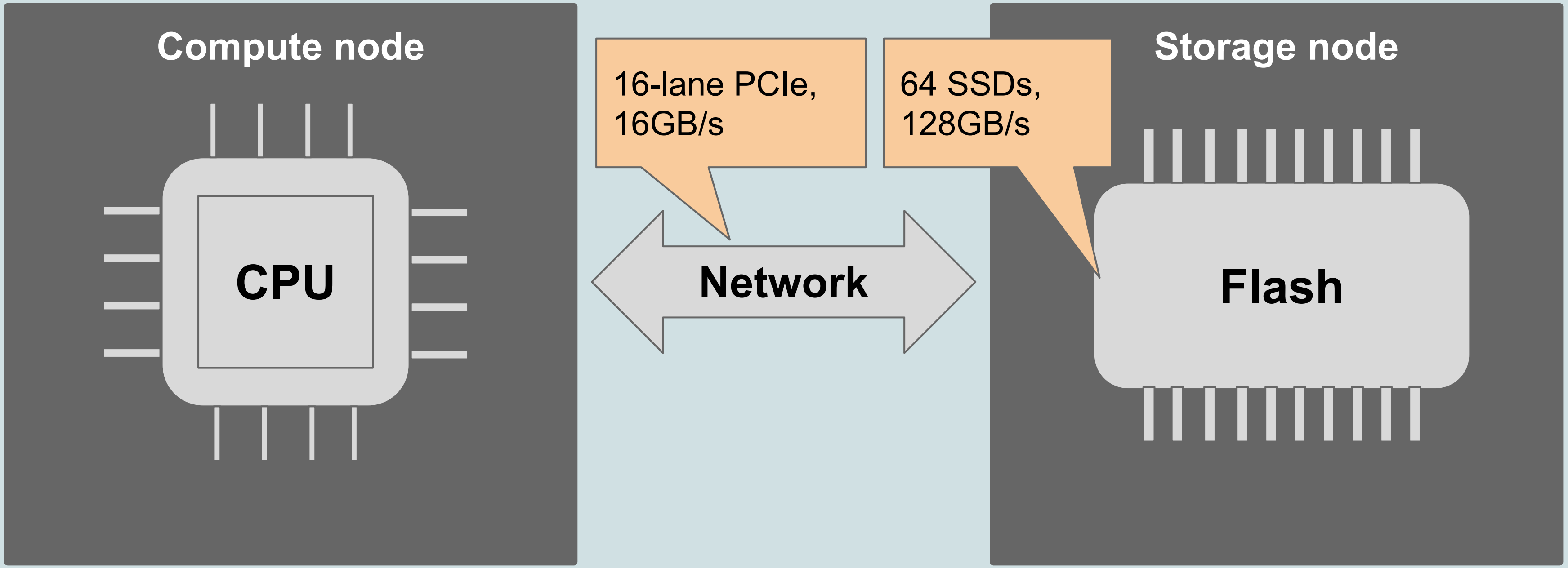
## So, this is a datacenter



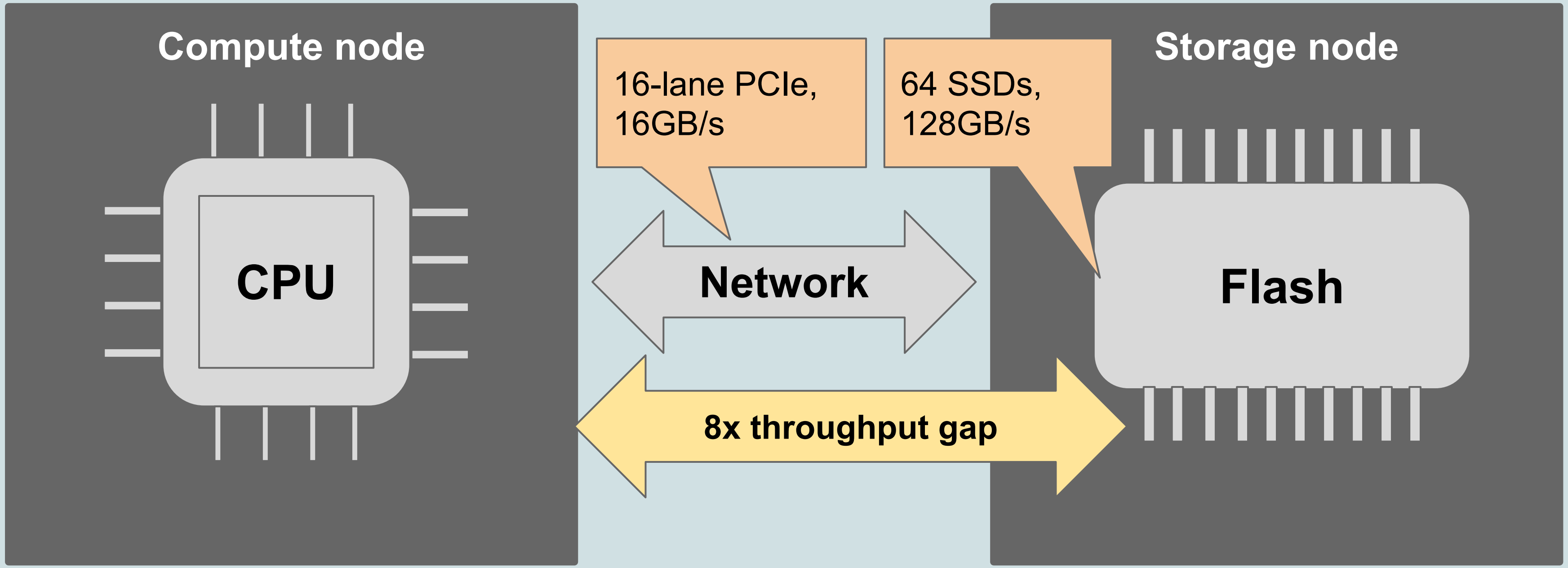
# So, this is a datacenter



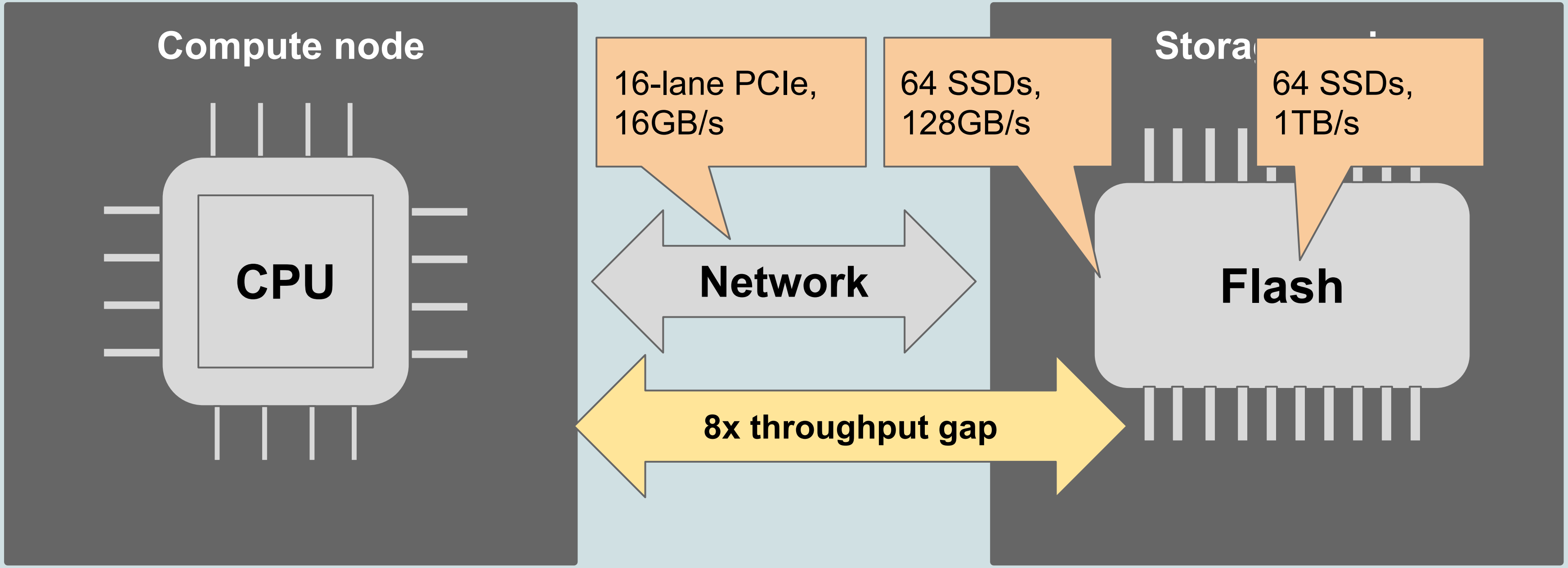
# So, this is a datacenter



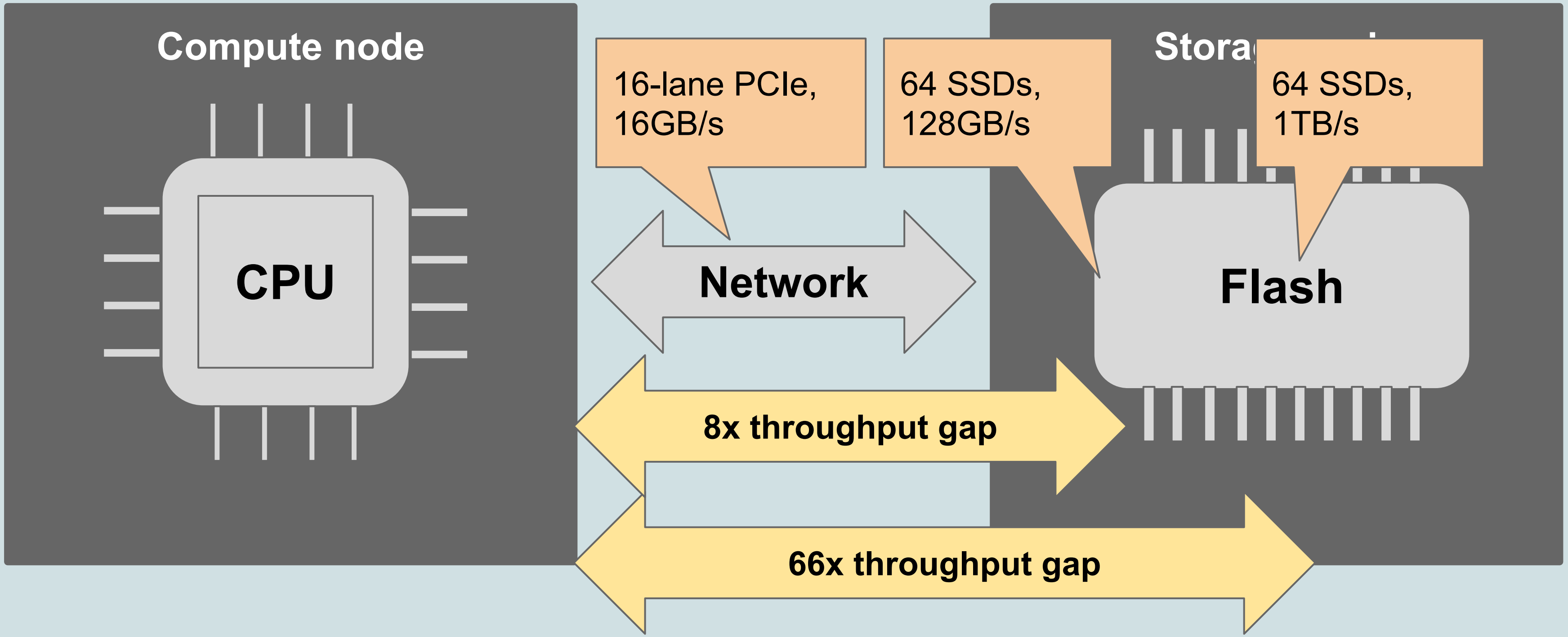
# So, this is a datacenter



# So, this is a datacenter



# So, this is a datacenter



https://cacm.acm.org/magazines/2019/6/237002-programmable-solid-state-storage-in-future-cloud-datacenters/fulltext?mobile=false

## What is happening?

- The data deluge is only getting worse
- Storage is becoming TOO fast
- Data movement is a bottleneck

## So this is what I worked on

For almost a year

## How to fix this?

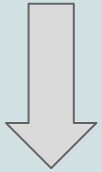
- Reduce data movement
- Compute on storage
- Optimize data transfers

$\Downarrow$

**Programmability!**

## How to fix this?

- Reduce data movement
- Compute on storage
- Optimize data transfers



Programmability!

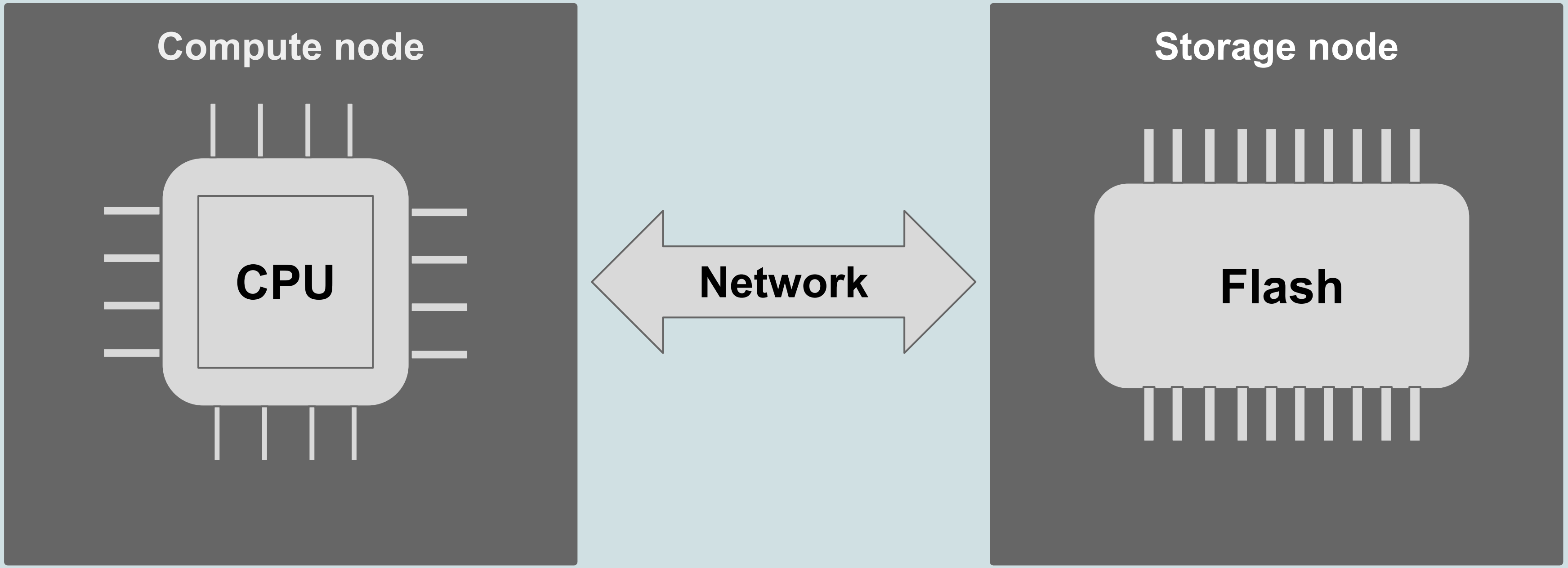
## Who am I

- Giulia Frascaria
- Vrije Universiteit, Amsterdam
- PhD candidate
- atLarge research group

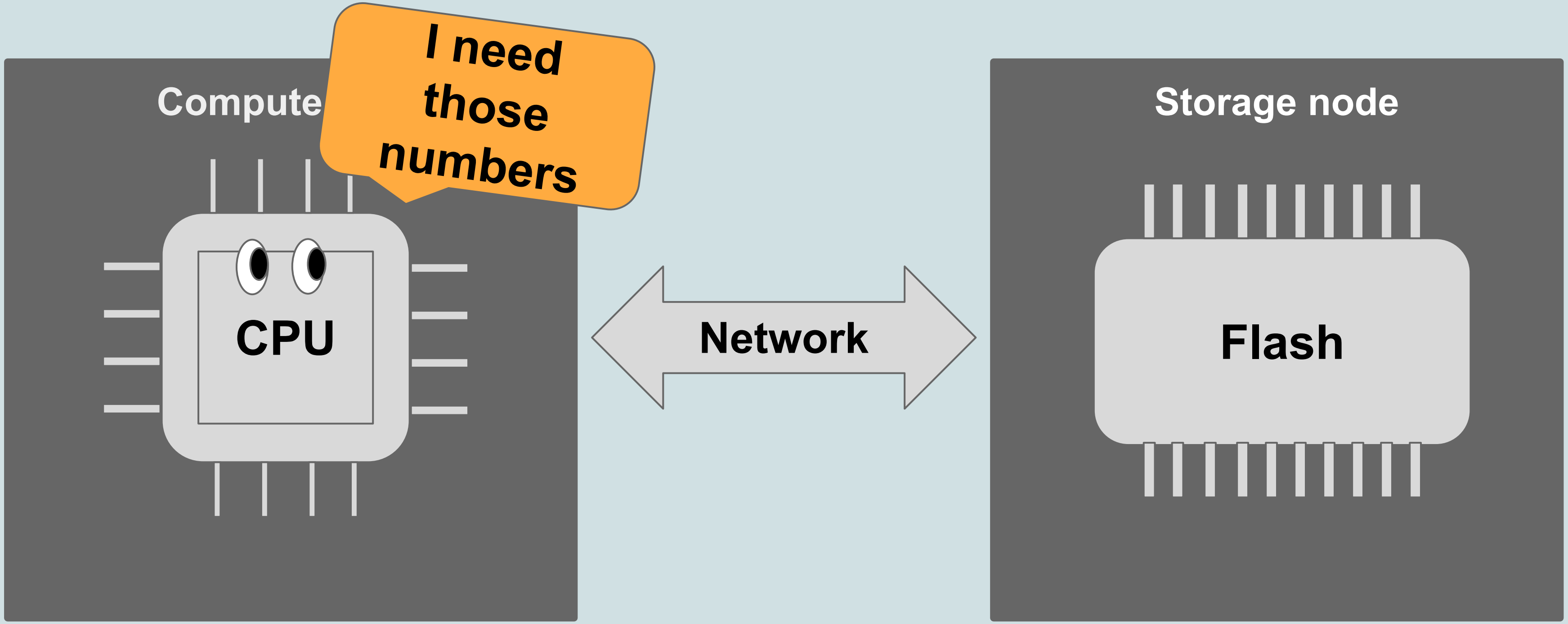
https://atlarge-research.com/

## Ok, let's start...

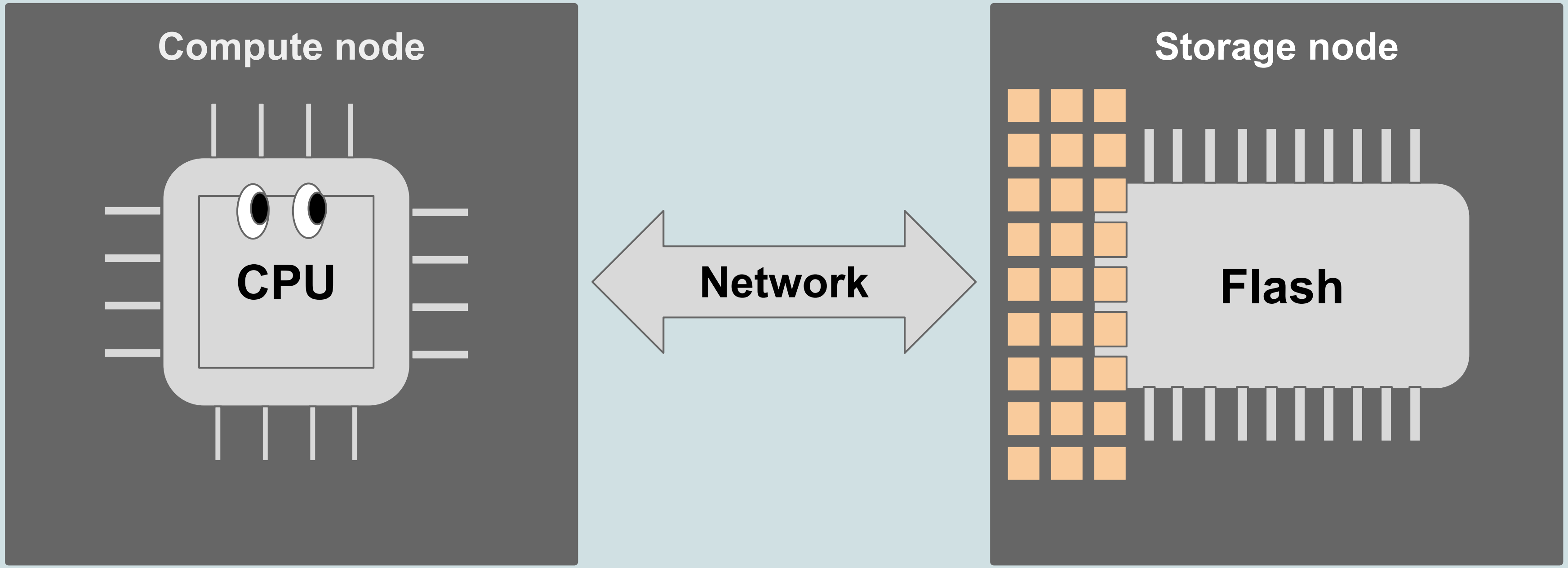
# Ok, let's average numbers



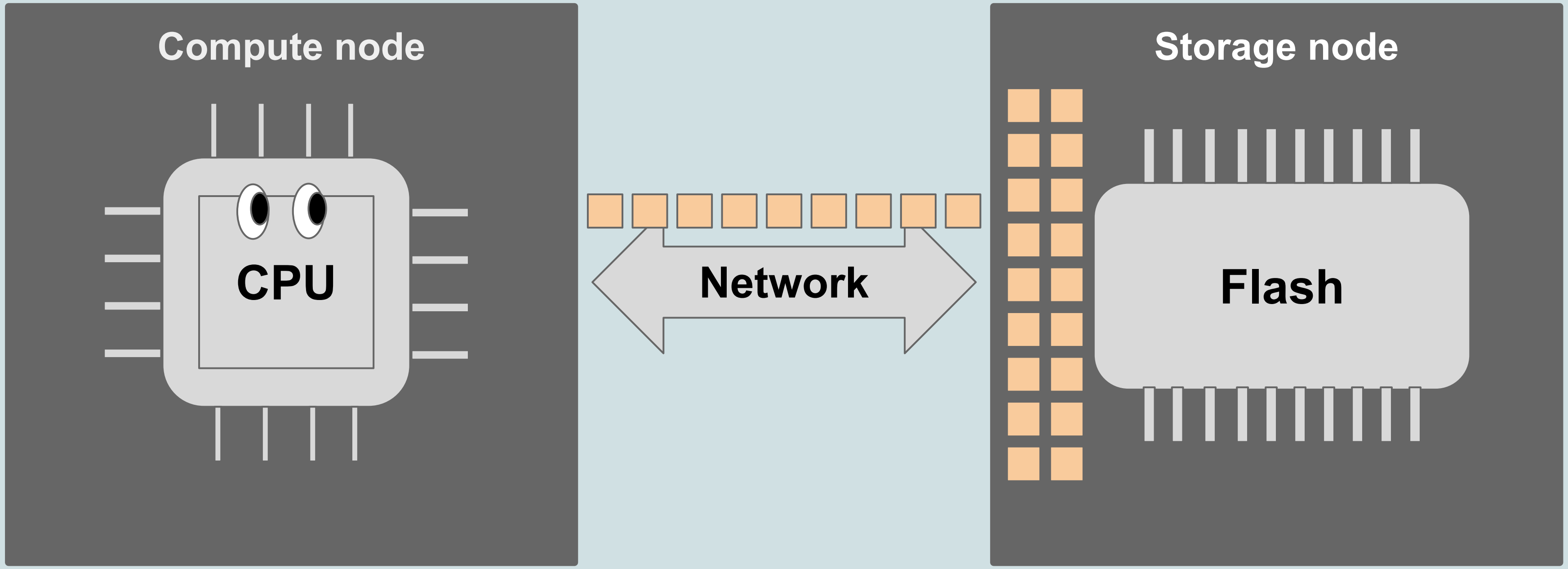
## Ok, let's average numbers



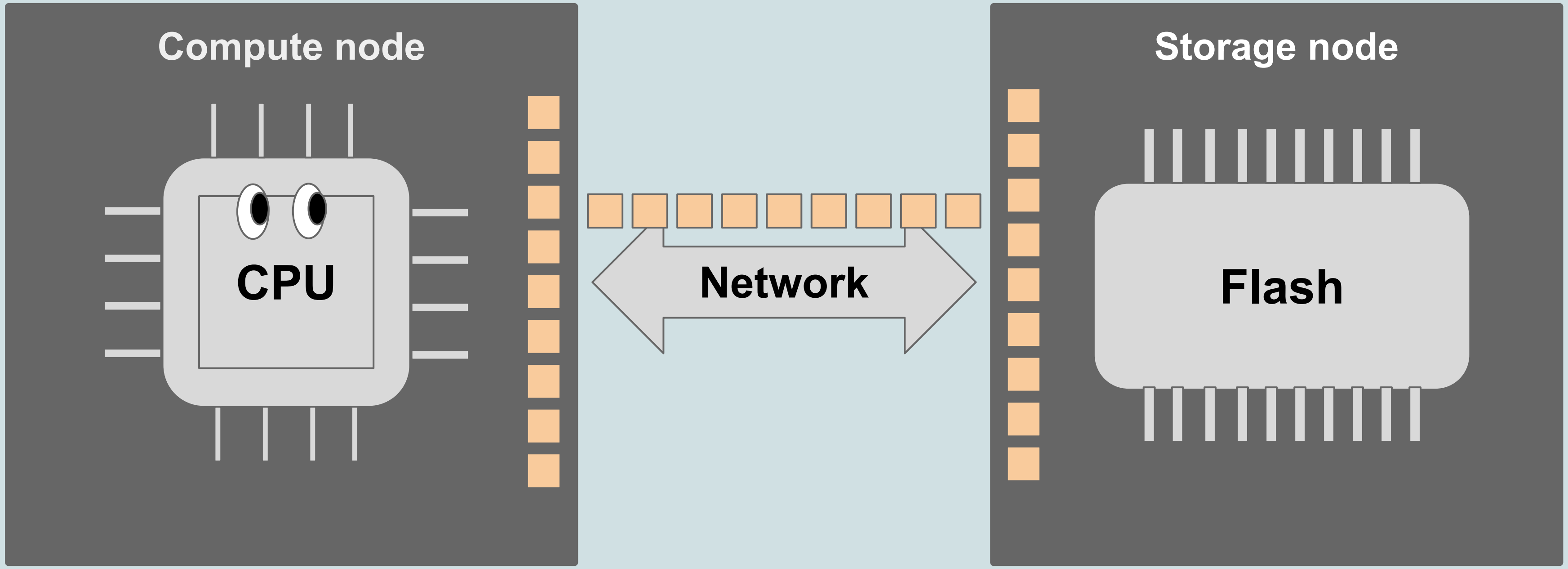
## Ok, let's compute the average



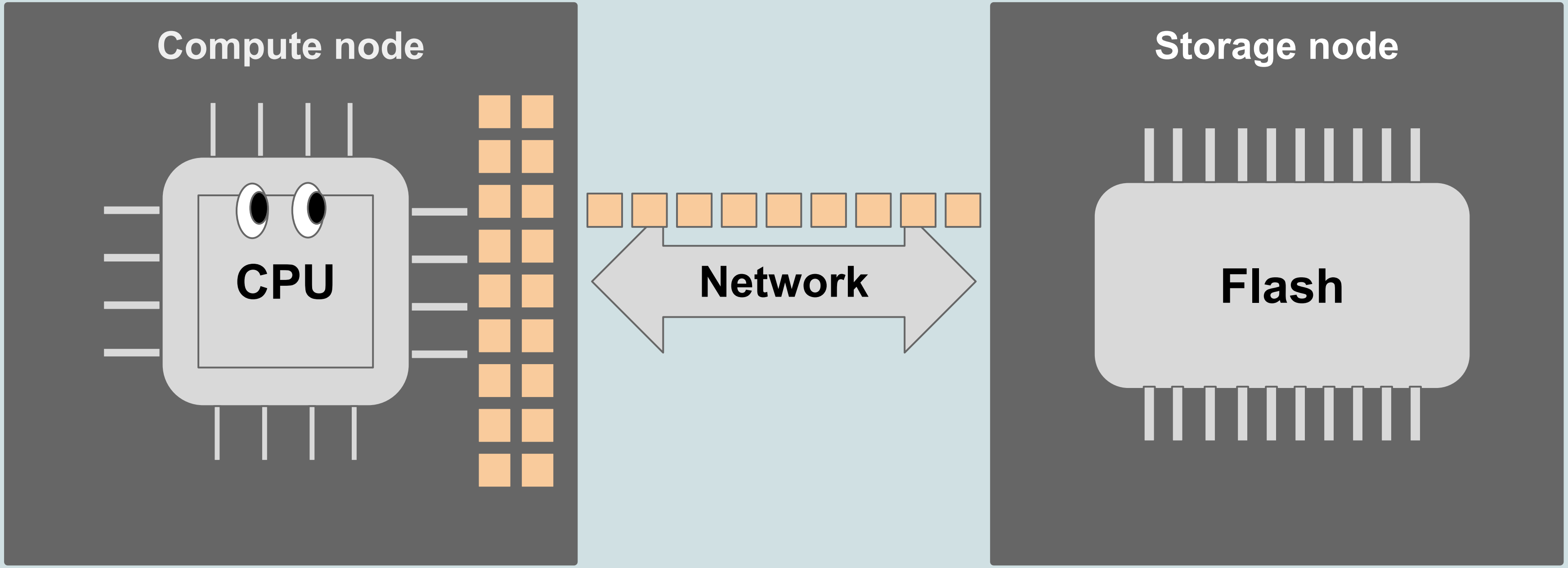
## Ok, let's compute the average



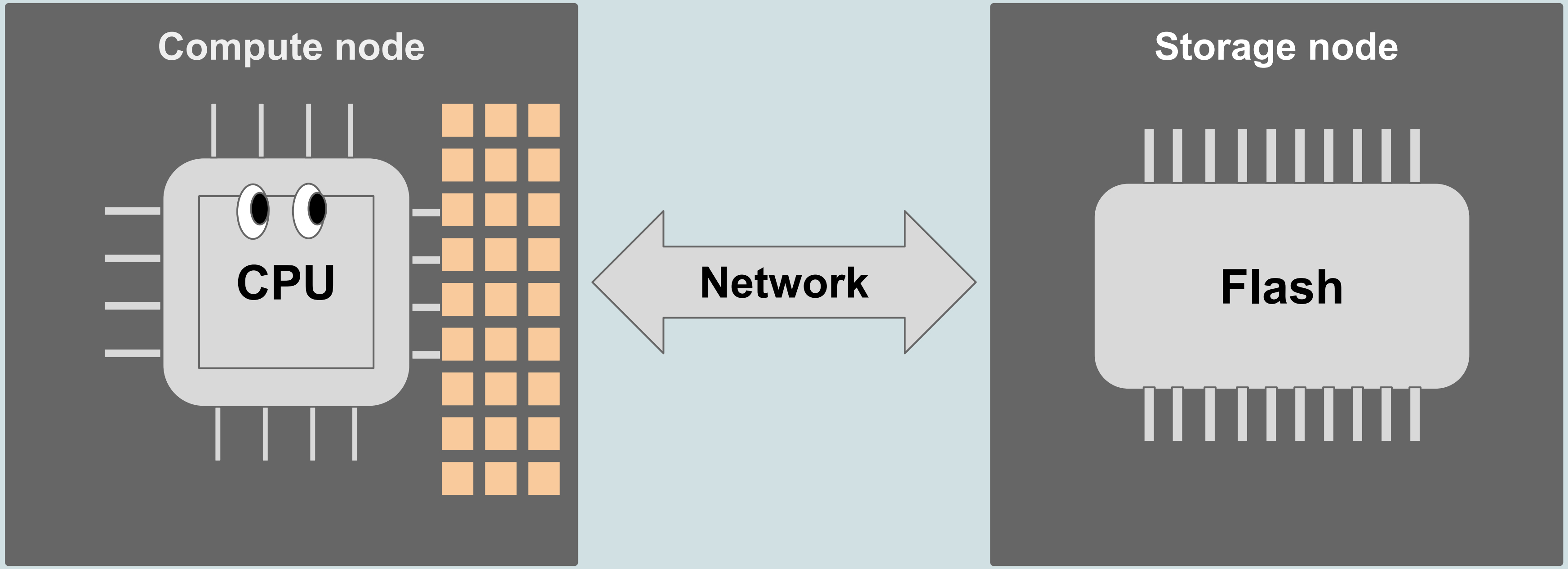
## Ok, let's compute the average



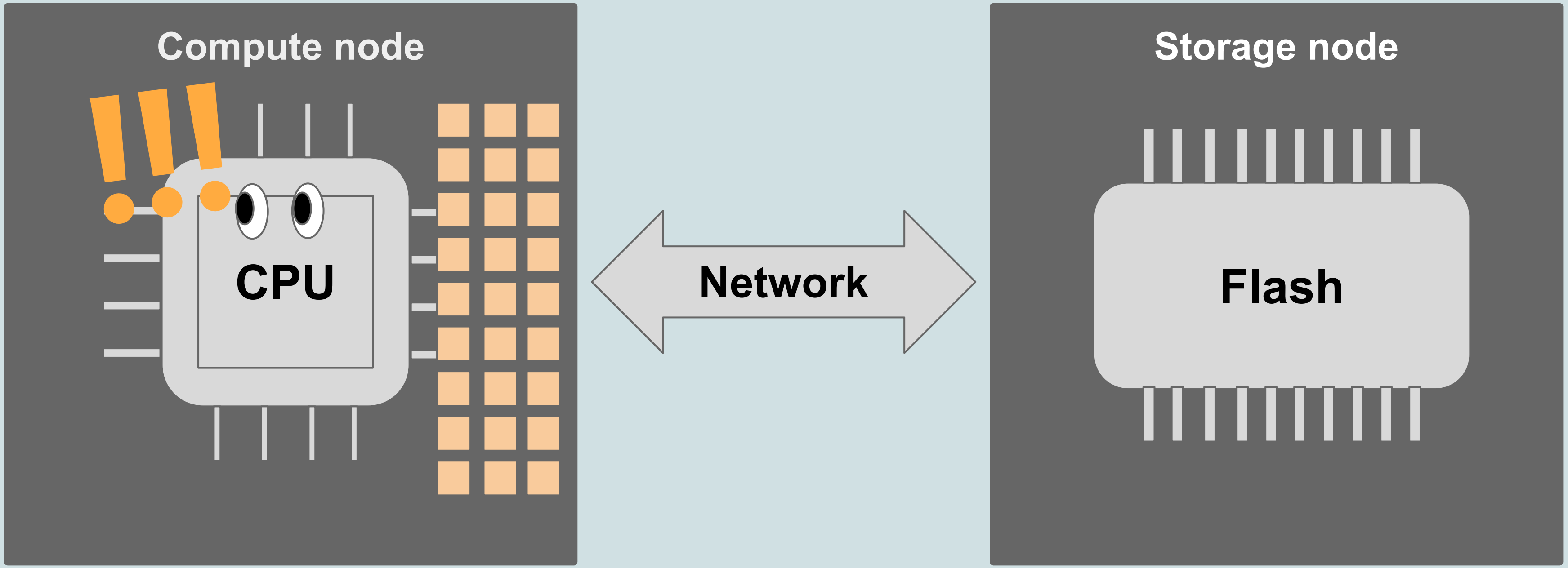
## Ok, let's compute the average



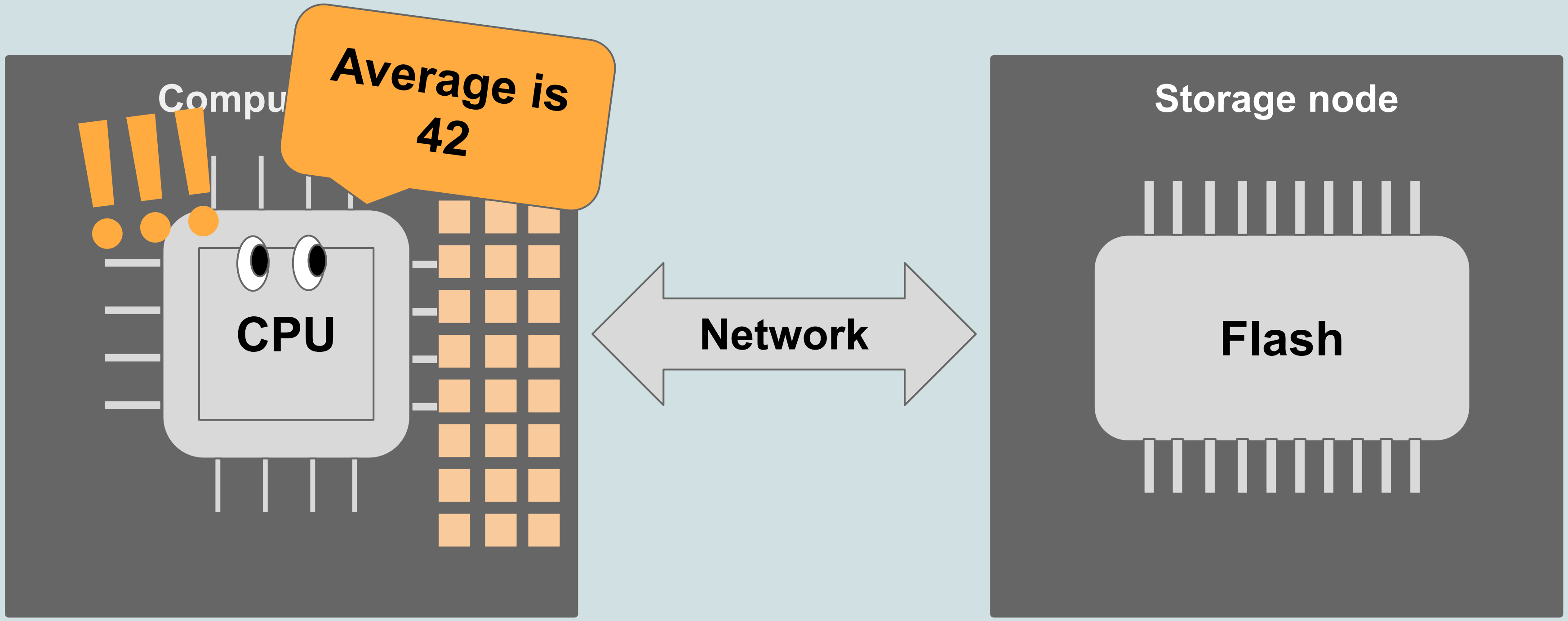
## Ok, let's compute the average



# Ok, let's compute the average



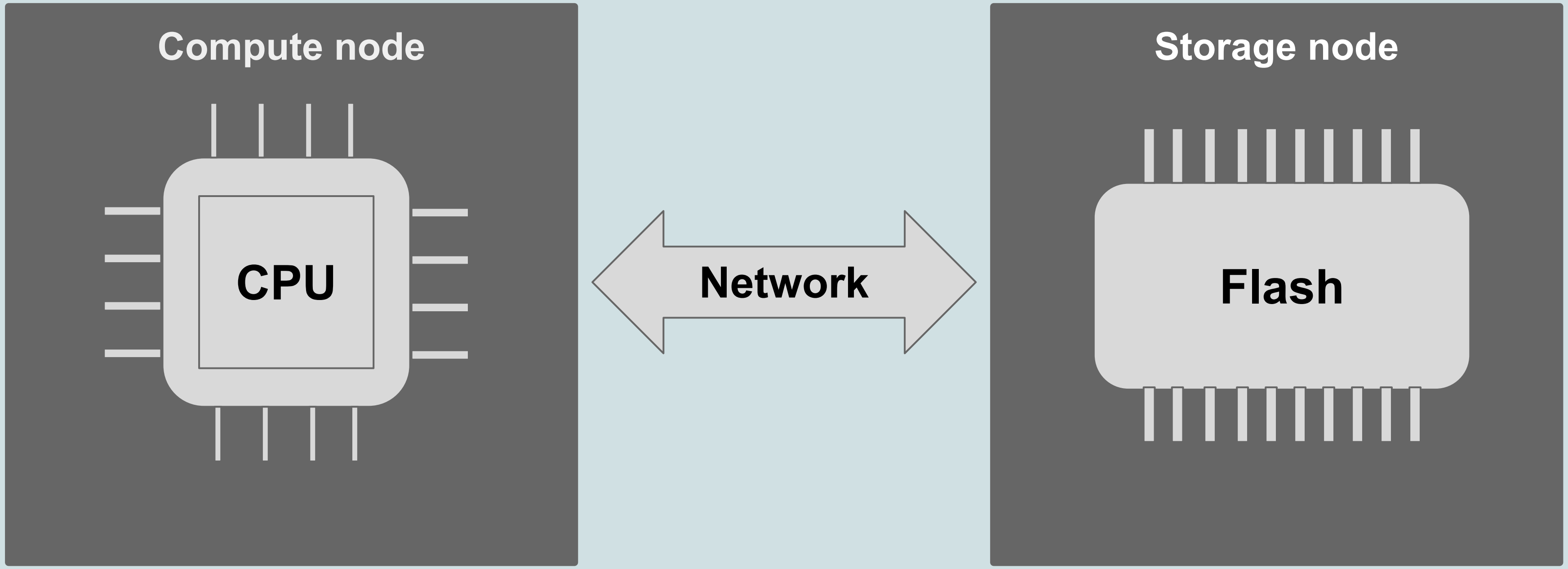
# Ok, let's compute the average



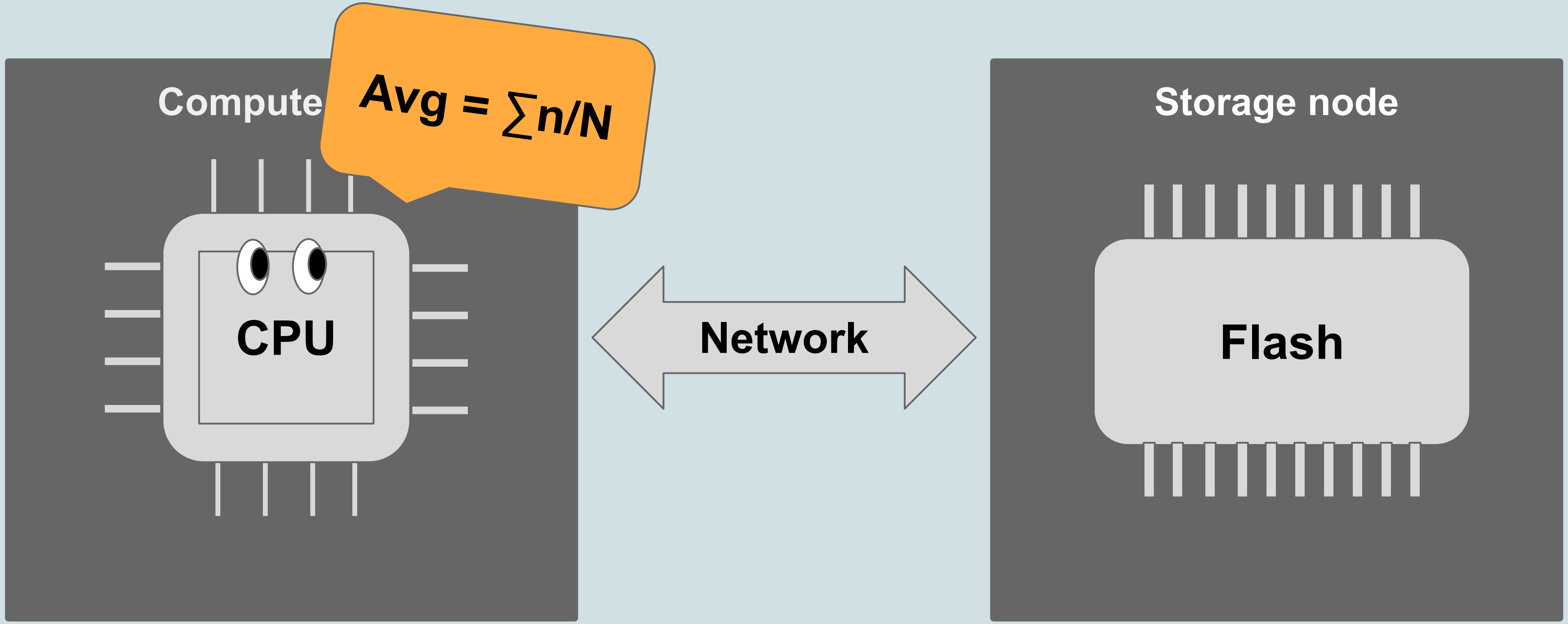
## This doesn’t make sense

- Huge data transfer
- High latency
- Unnecessary network congestion
- Bottleneck on storage

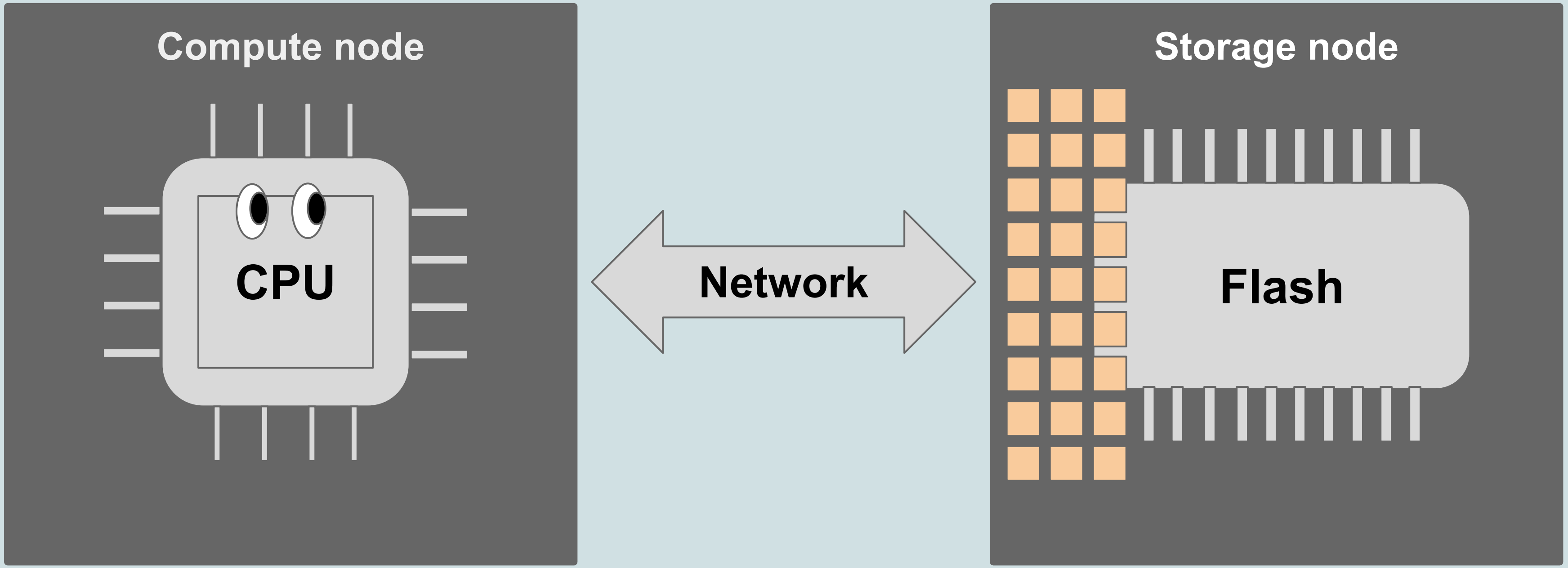
## Let's move it to the storage!



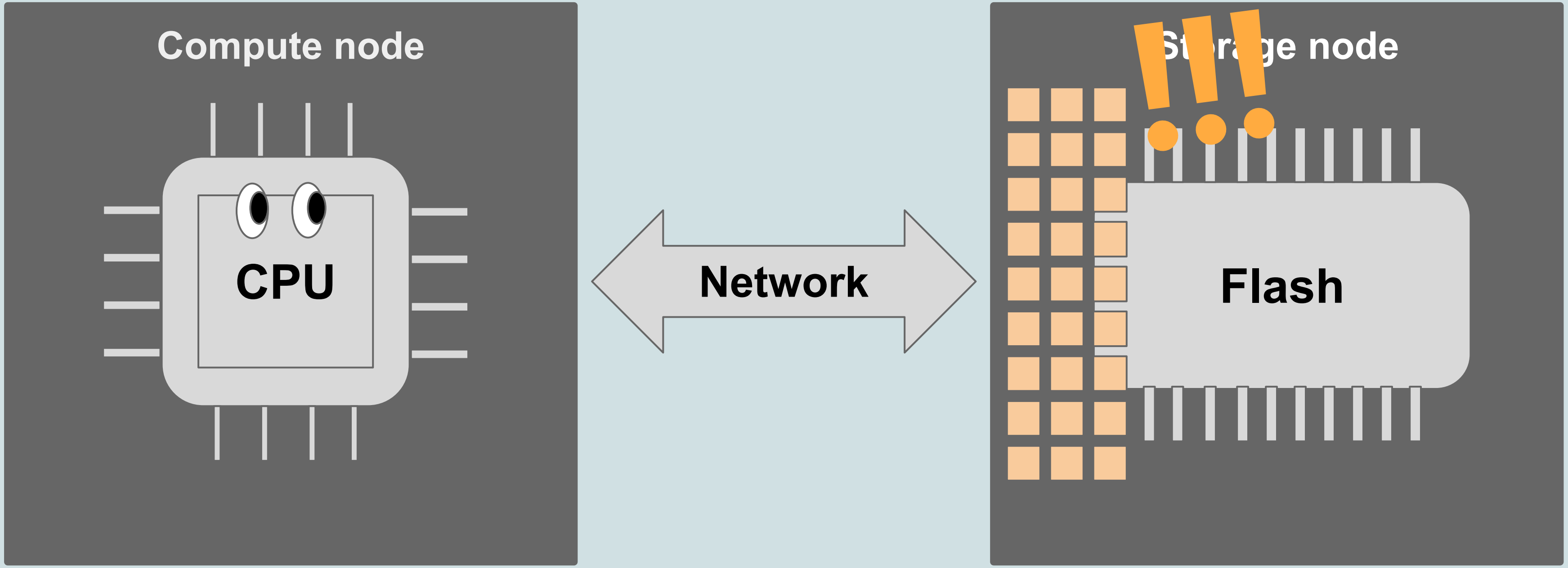
# Let's move it to the storage!



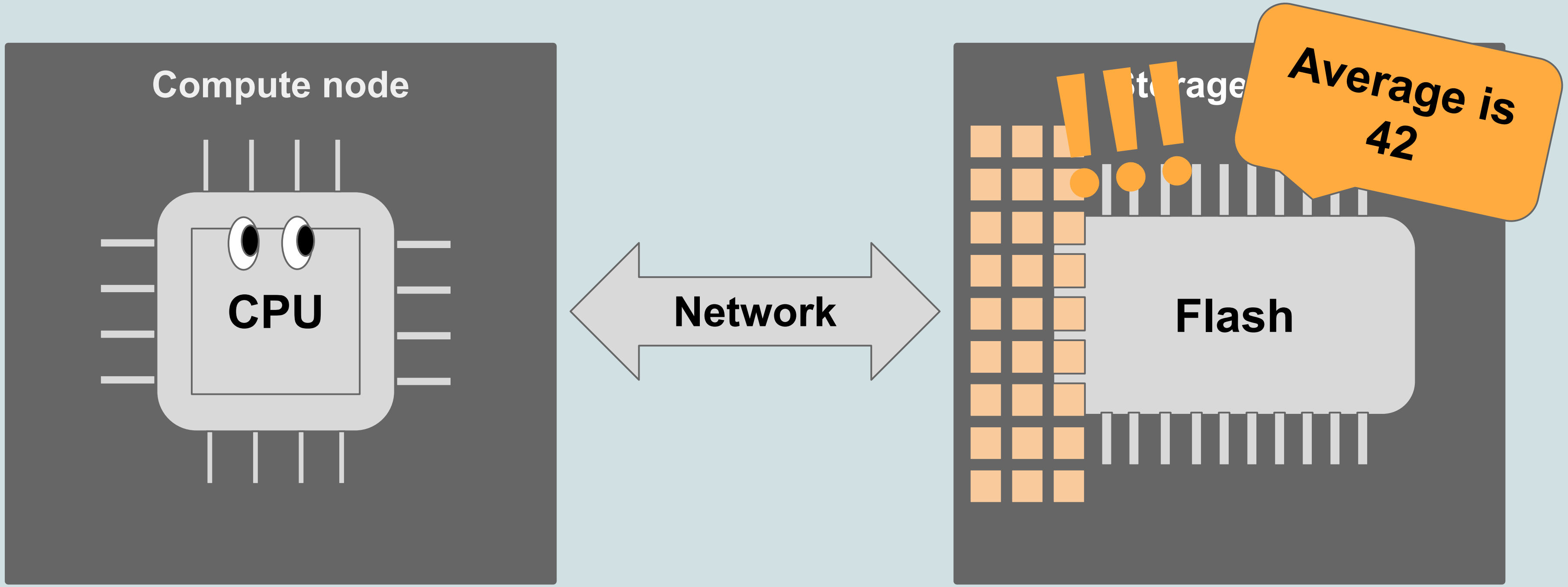
## Let's move it to the storage!



# Let's move it to the storage!



## Let's move it to the storage!



## This makes sense!

- Only transfer result
- Reduced network congestion
- Use storage throughput

## How to fix this?

## Datacenter requirements

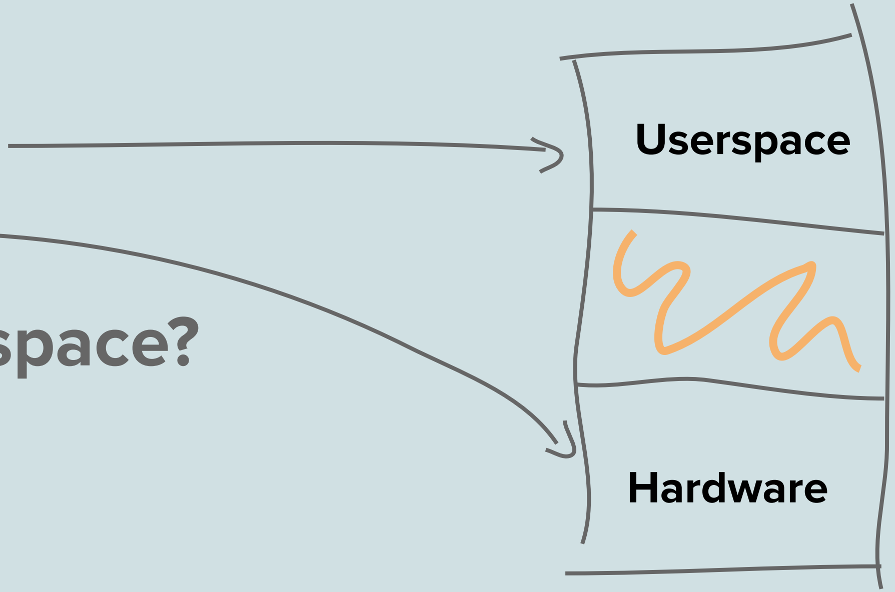
- Multitenancy
- Isolation and security
- High performance
- Low deployment cost

## And my requirement

- Multitenancy
- Isolation and security
- High performance
- Low deployment cost
- Reduce data movement

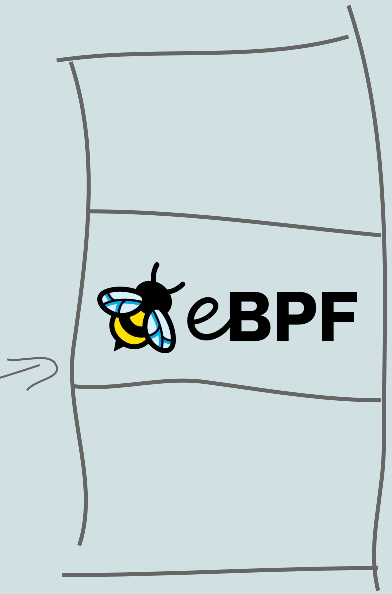
# What are the options?

- Python, Rust, Go...
- ASIC, FPGAs
- What about kernel space?



## What are the options?

- Python, Rust, Go...
- ASIC, FPGAs
- What about kernel space?
- eBPF is included



## eBPF 101

- Javascript for the kernel
- Programmable
- In-kernel VM
- Efficient
- Safe and formally verified

## eBPF for networking

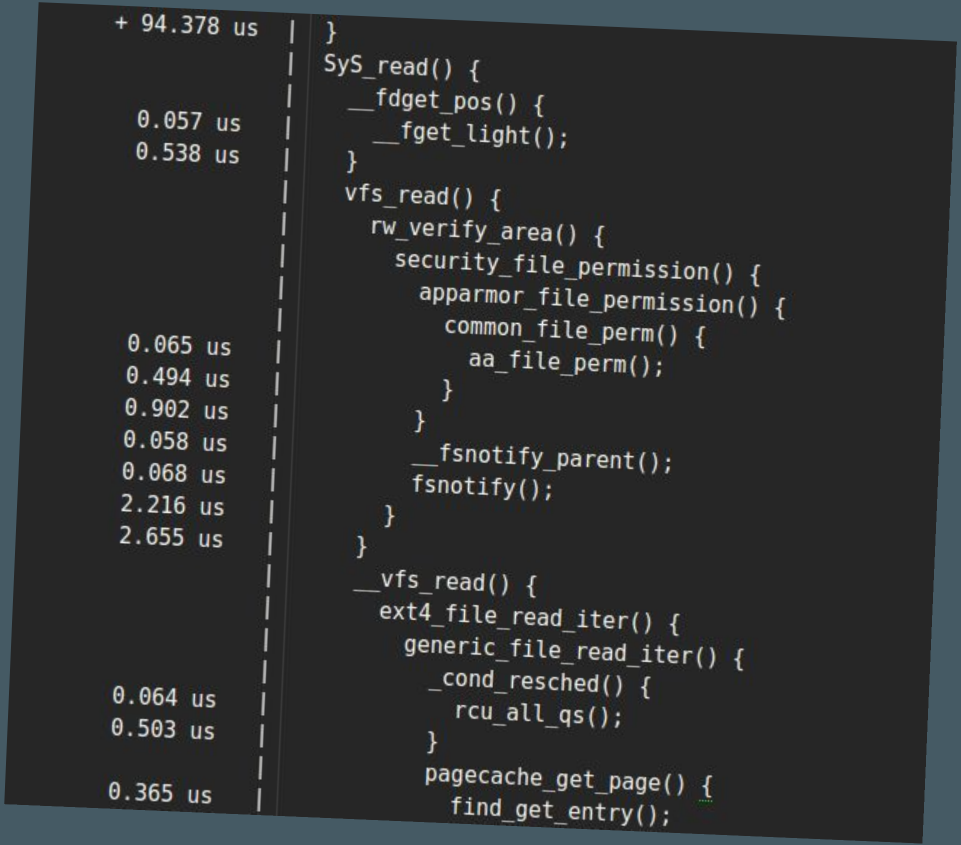
- Packet inspection
- Modify packets
- Drop packets

# eBPF for networking

- ~~Packet inspection~~ READ
- ~~Modify packets~~ WRITE
- ~~Drop packets~~ DROP

## Warning:
## kernel code ahead

# The read() path

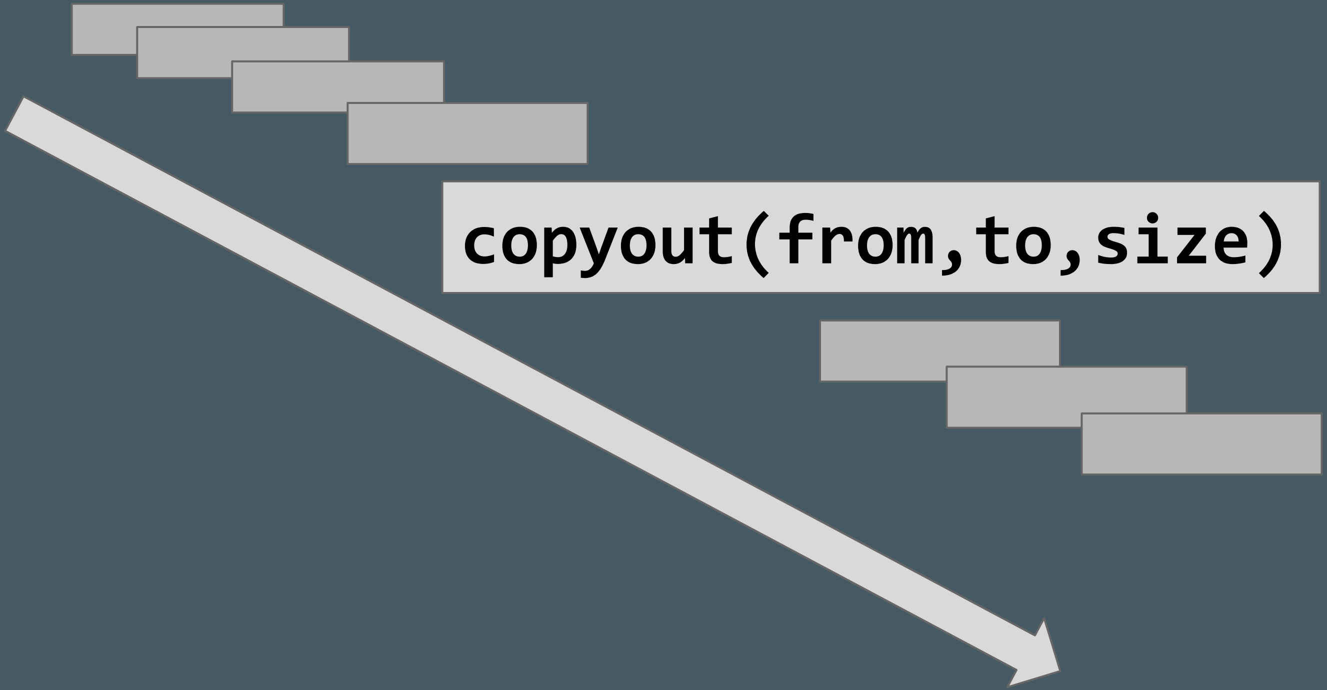


```
 + 94.378 us   |  }
               |  SyS_read() {
               |    __fdget_pos() {
   0.057 us    |      __fget_light();
   0.538 us    |    }
               |    vfs_read() {
               |      rw_verify_area() {
               |        security_file_permission() {
               |          apparmor_file_permission() {
               |            common_file_perm() {
   0.065 us    |              aa_file_perm();
   0.494 us    |            }
   0.902 us    |          }
   0.058 us    |          __fsnotify_parent();
   0.068 us    |          fsnotify();
   2.216 us    |        }
   2.655 us    |      }
               |      __vfs_read() {
               |        ext4_file_read_iter() {
               |          generic_file_read_iter() {
               |            _cond_resched() {
   0.064 us    |              rcu_all_qs();
   0.503 us    |            }
               |            pagecache_get_page() {
   0.365 us    |              find_get_entry();
```

## What am I looking for?

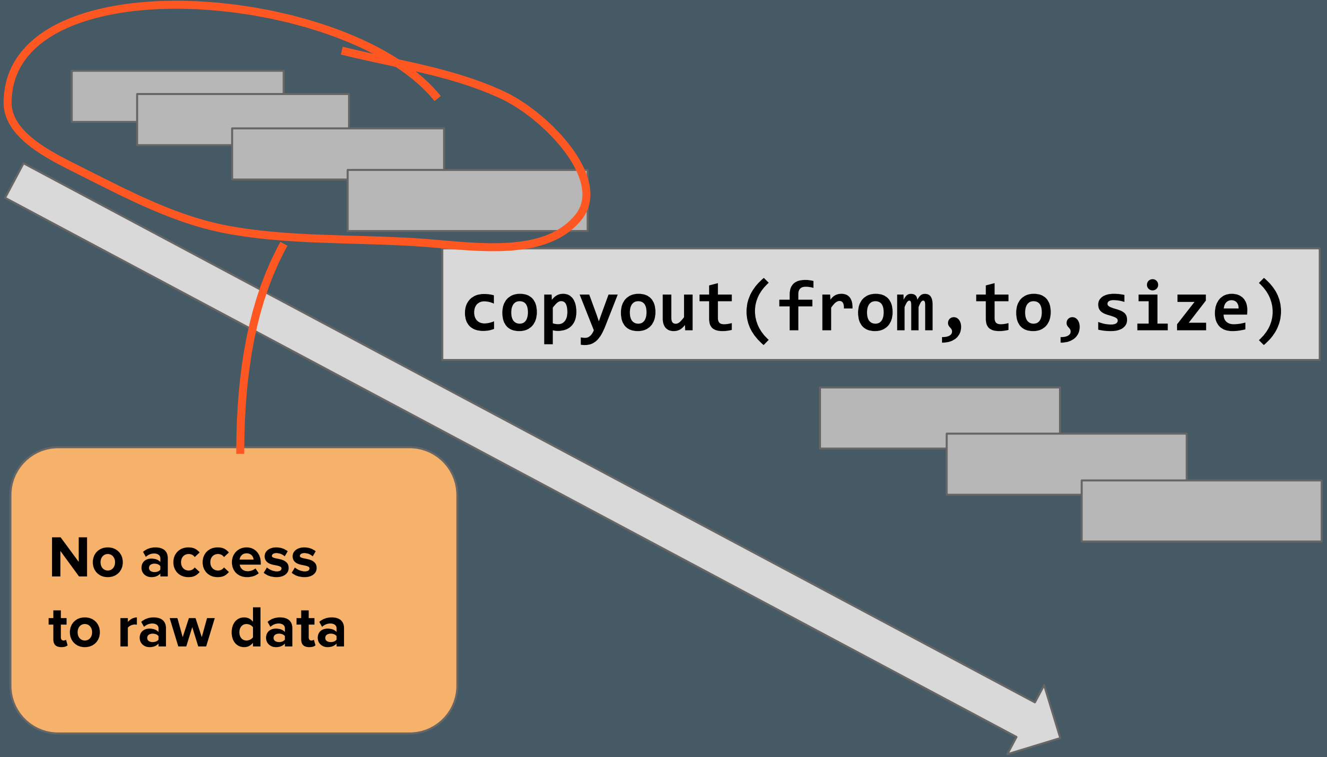
- Identify user and file
- Access to the data
- Execute custom code

## Somewhere in the read path…

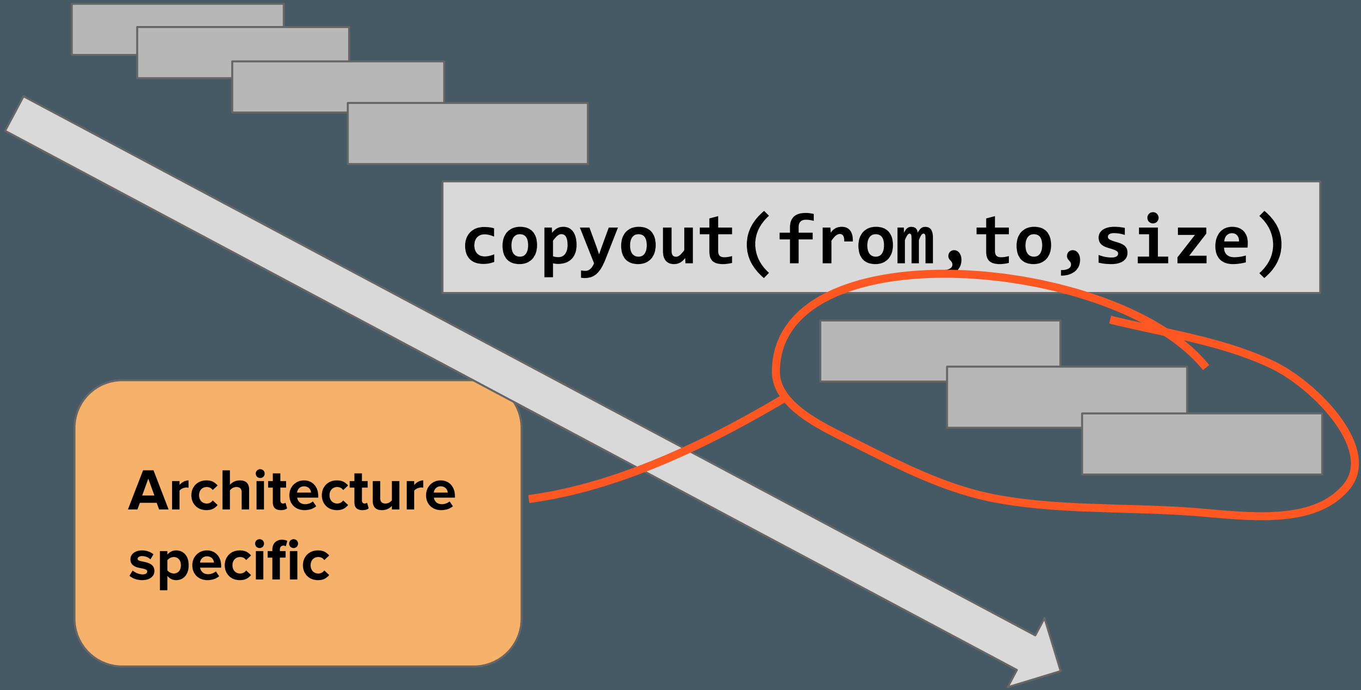


```
copyout(from,to,size)
```

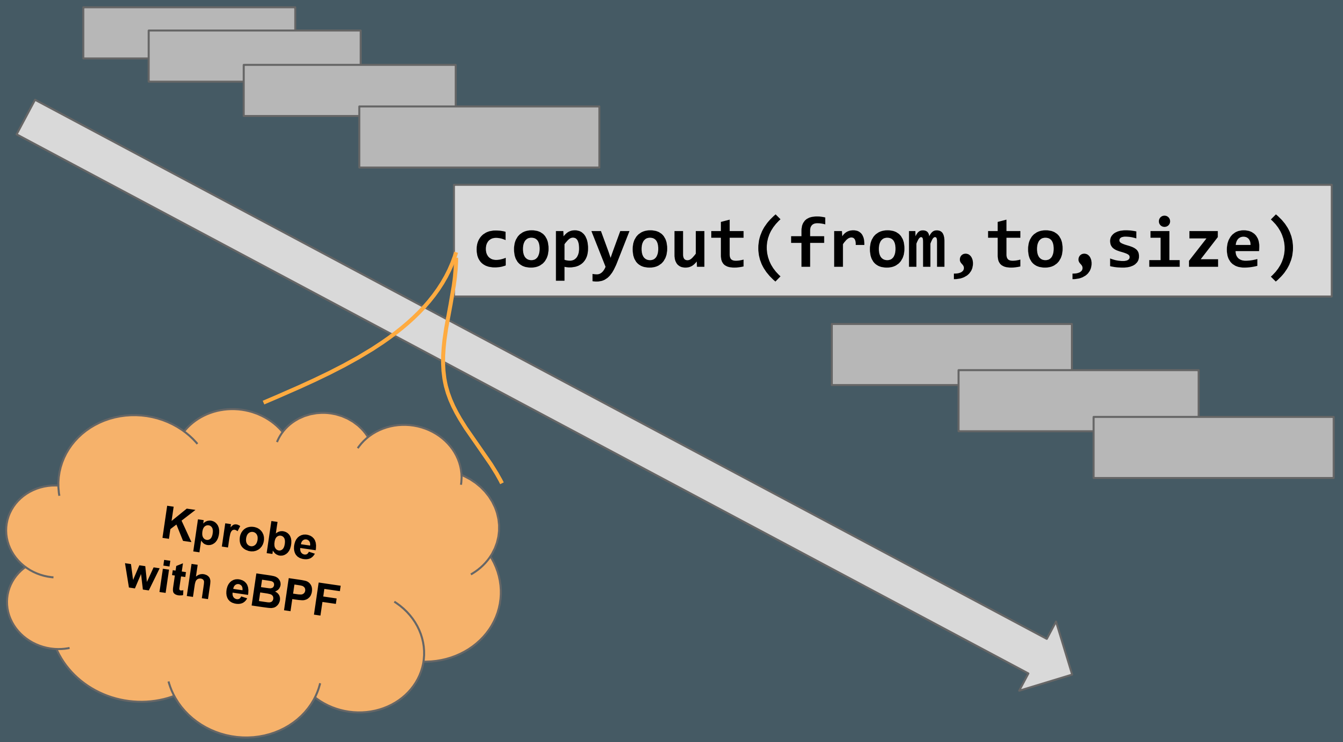
# Somewhere in the read path...



## Somewhere in the read path...

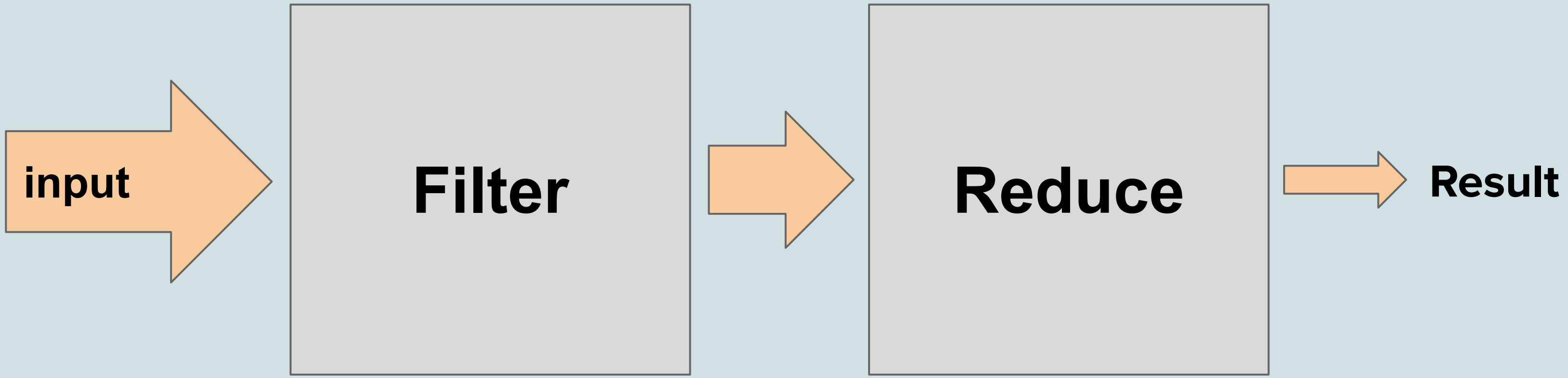


## Somewhere in the read path…



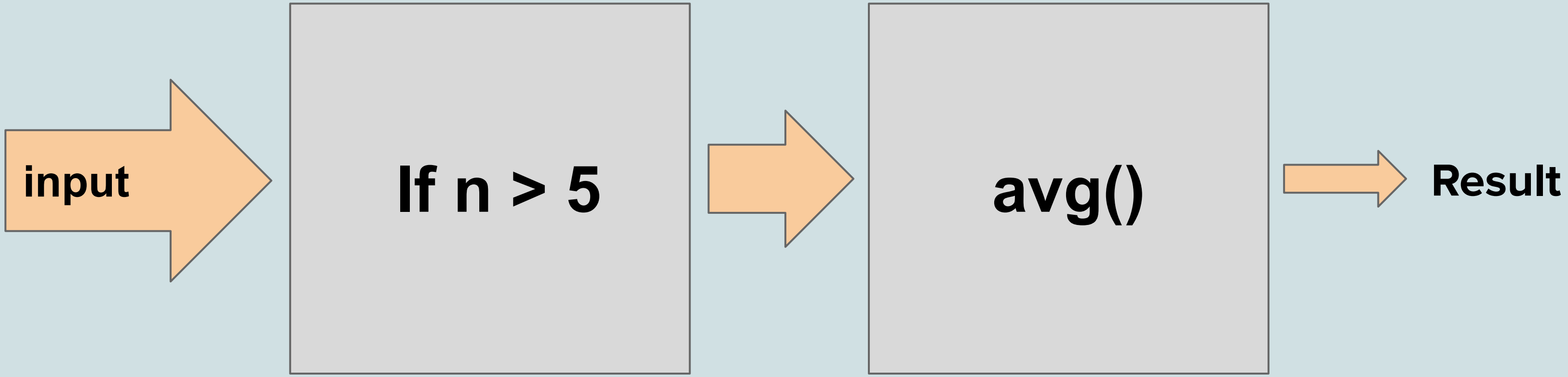
## So, all according to plan?

## This is the prototype



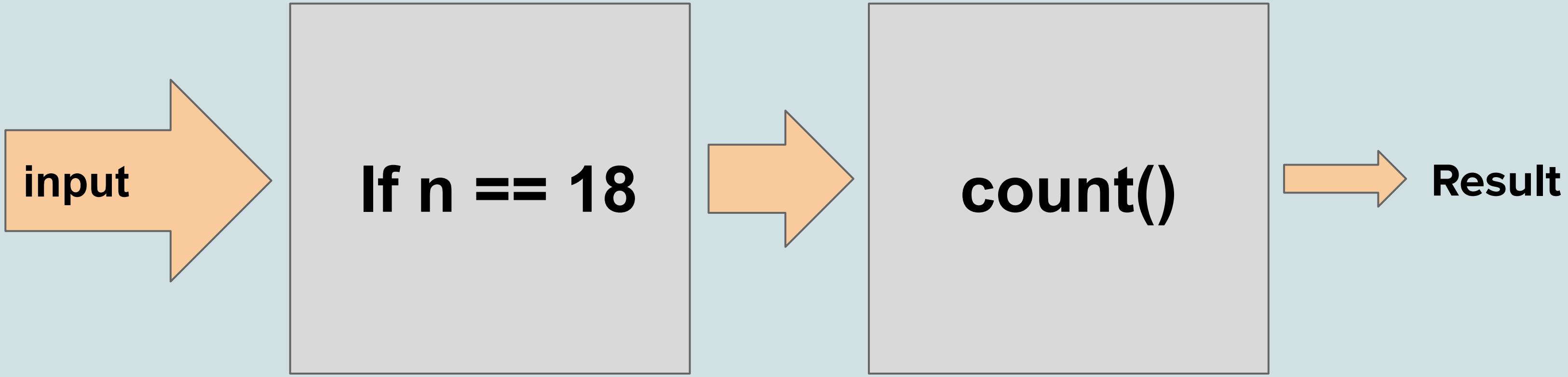
[https://github.com/giuliafrascaria/ebpf-data-filter](https://github.com/giuliafrascaria/ebpf-data-filter)

## This is the prototype



https://github.com/giuliafrascaria/ebpf-experiments

## This is the prototype

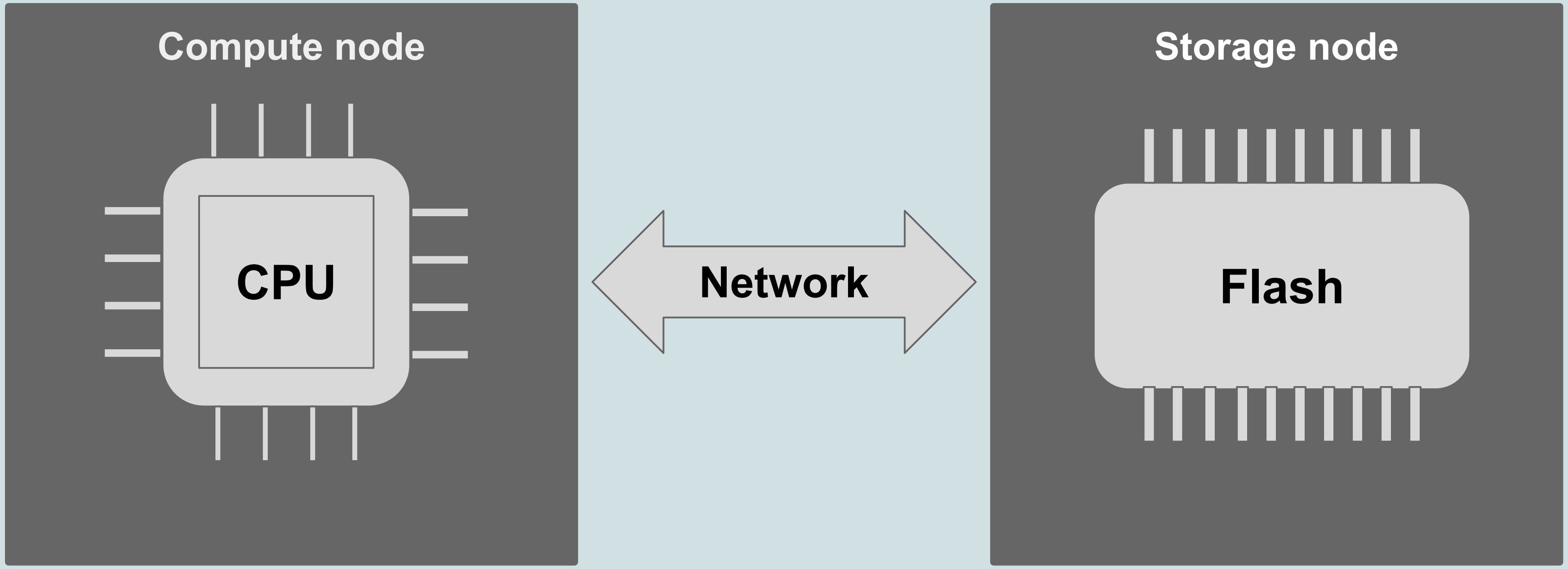


https://github.com/giuliafrascaria/ebpf-experiments

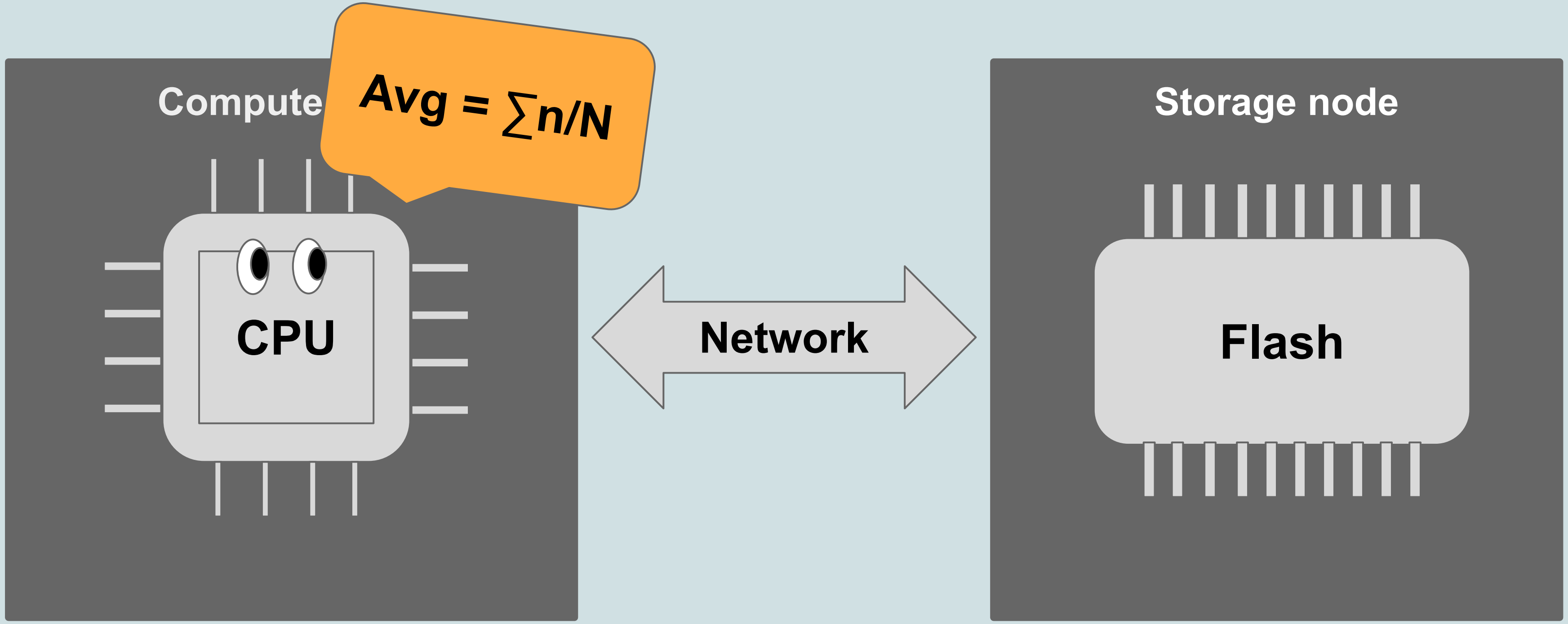
## Why Filter-Reduce?

- Keep a standard interface
- Offer users some flexibility
- MapReduce
- Relational data processing
- Maximum data reduction

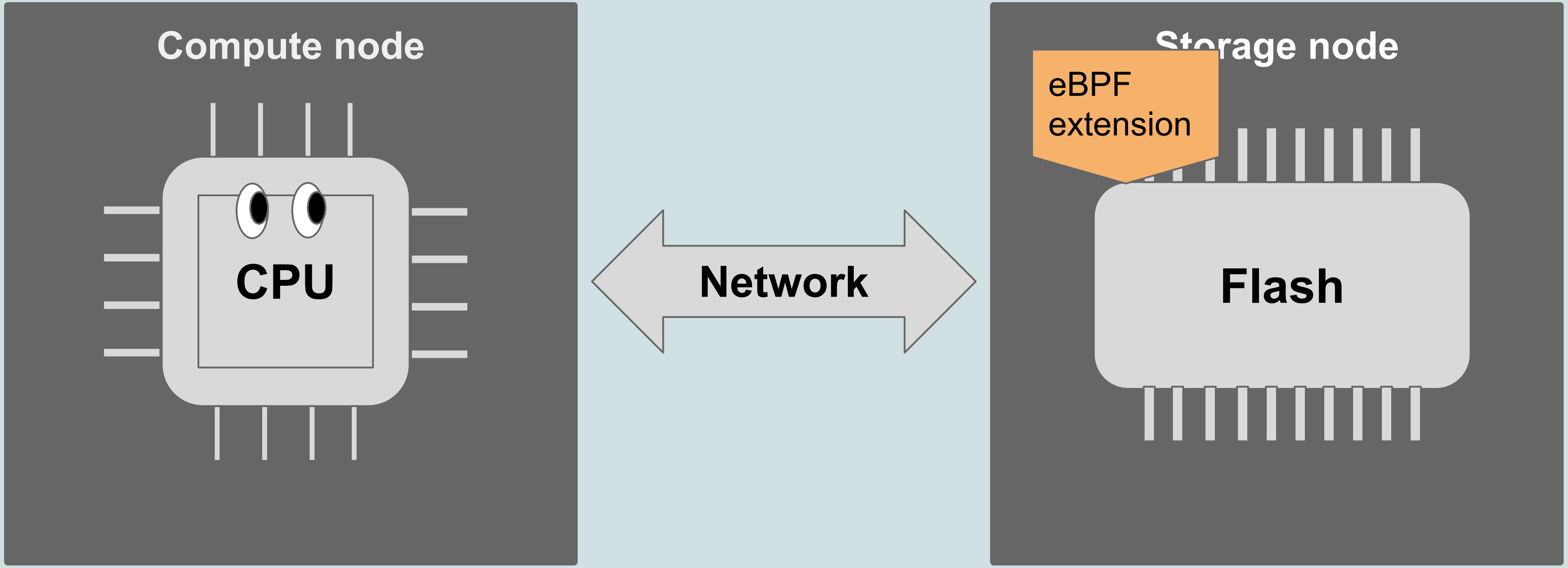
## So actually this happens



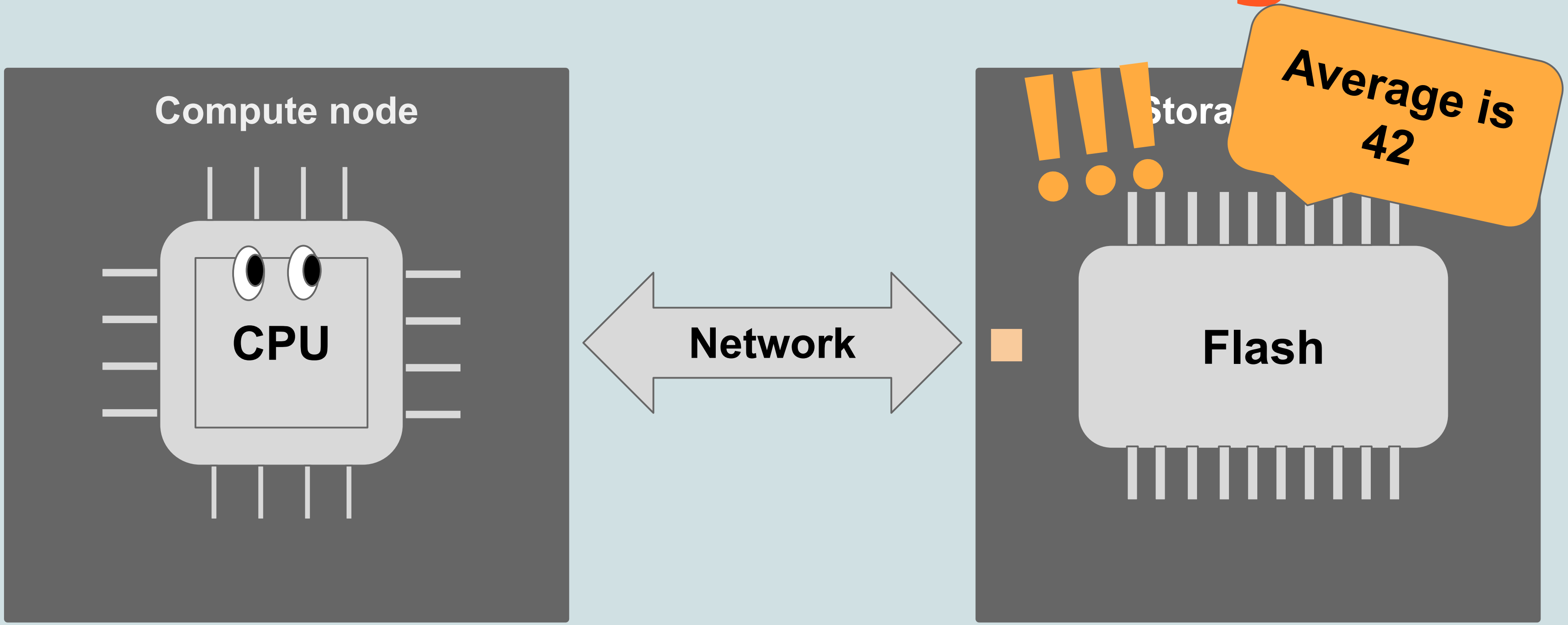
## Let's move it to the storage!



## Let's move it to the storage!



## Let's move it to the storage!



## Thanks to eBPF:

- Reduce network congestion
- Avoid kernel to user copy
- Users do not read data
- Only result is shared

eBPF runs in a kernel sandbox

## So, all according to plan?

## Well you read the title

## First of all, where to hook the bpf extensions

- Copyout is not an exported symbol
- Necessary to use modified Linux kernel

## Second, what about data access?

- Even read-only, you need a helper function
- Adds overhead
- Direct pointer access would be more efficient

## Third, how much can I process?

- No dynamic allocation
- Stack limited to 512 bytes
- Need to iterate in batches
- eBPF instruction number limit

## Well at least it computes the average, right?

- Allow to use helpers to convert char to int
- Floating point division not supported in kernel

## So, to summarize

- Conceptually same as networking
- Achieves data movement reduction
- Safe, isolated execution

But we need more support for I/O!

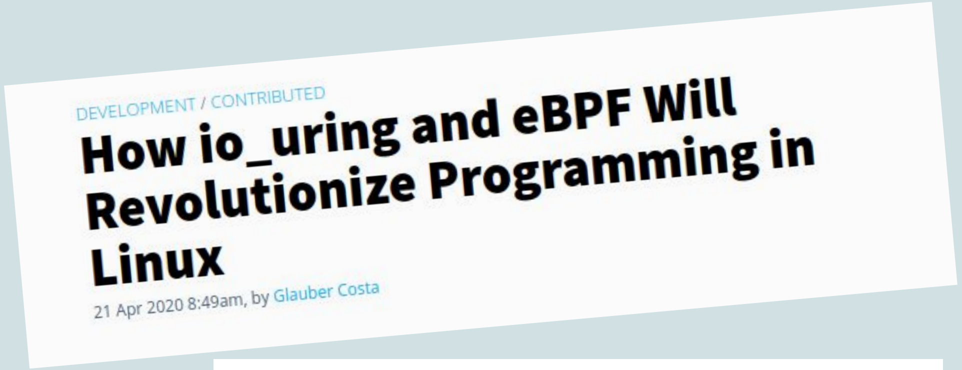
## Where are we headed

## eBPF is not slowing down

- Tracing: BCC, bpftrace…
- Networking: Cilium…
- Security: KRSI, Falco…

Is programmability the next step?

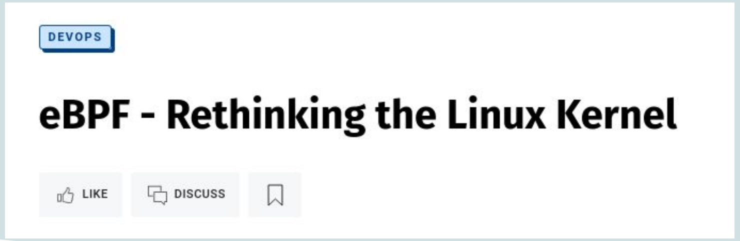
## We're not alone, will you join?



DEVELOPMENT / CONTRIBUTED

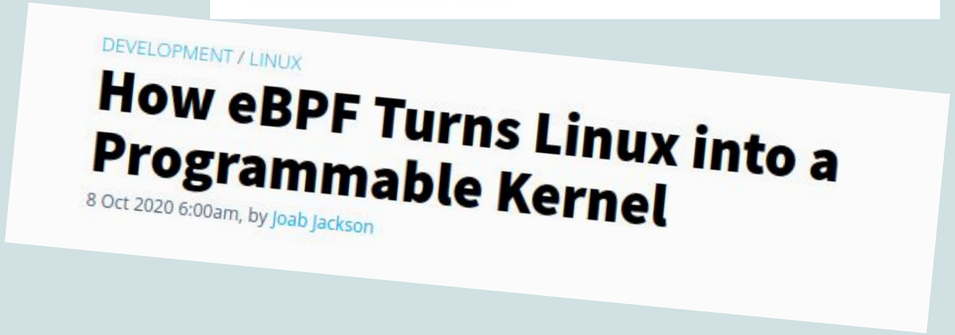
How io_uring and eBPF Will Revolutionize Programming in Linux

21 Apr 2020 8:49am, by Glauber Costa



DEVOPS

eBPF - Rethinking the Linux Kernel



DEVELOPMENT / LINUX

How eBPF Turns Linux into a Programmable Kernel

8 Oct 2020 6:00am, by Joab Jackson

https://thenewstack.io/how-io_uring-and-ebpf-will-revolutionize-programming-in-linux/
https://thenewstack.io/how-ebpf-turns-linux-into-a-programmable-kernel/
https://www.infoq.com/presentations/facebook-google-bpf-linux-kernel/