Yandex

The Optimization of a Boost.Asio-Based Networking Server

Nikita Chumakov, Sergey Handrikov

Hello, we are Yandex

- Yandex is the leading search engine in Russia.
  - 62% of Russian search traffic
  - 25 mln unique users per day
  - 6,000+ employees

We do mail, as well

- Yandex.Mail is a free mail service, quite popular in Russia and Russian-speaking countries.
- Built in 2002
- 9 mln unique users per day
- 110 mln messages sent and received daily

There are 20 services inside Mail which parse incoming messages.

It’d be nice to optimize, we thought.

Research subject

Input/Output

- Synchronous
- Asynchronous
- Coroutine

Concurrency

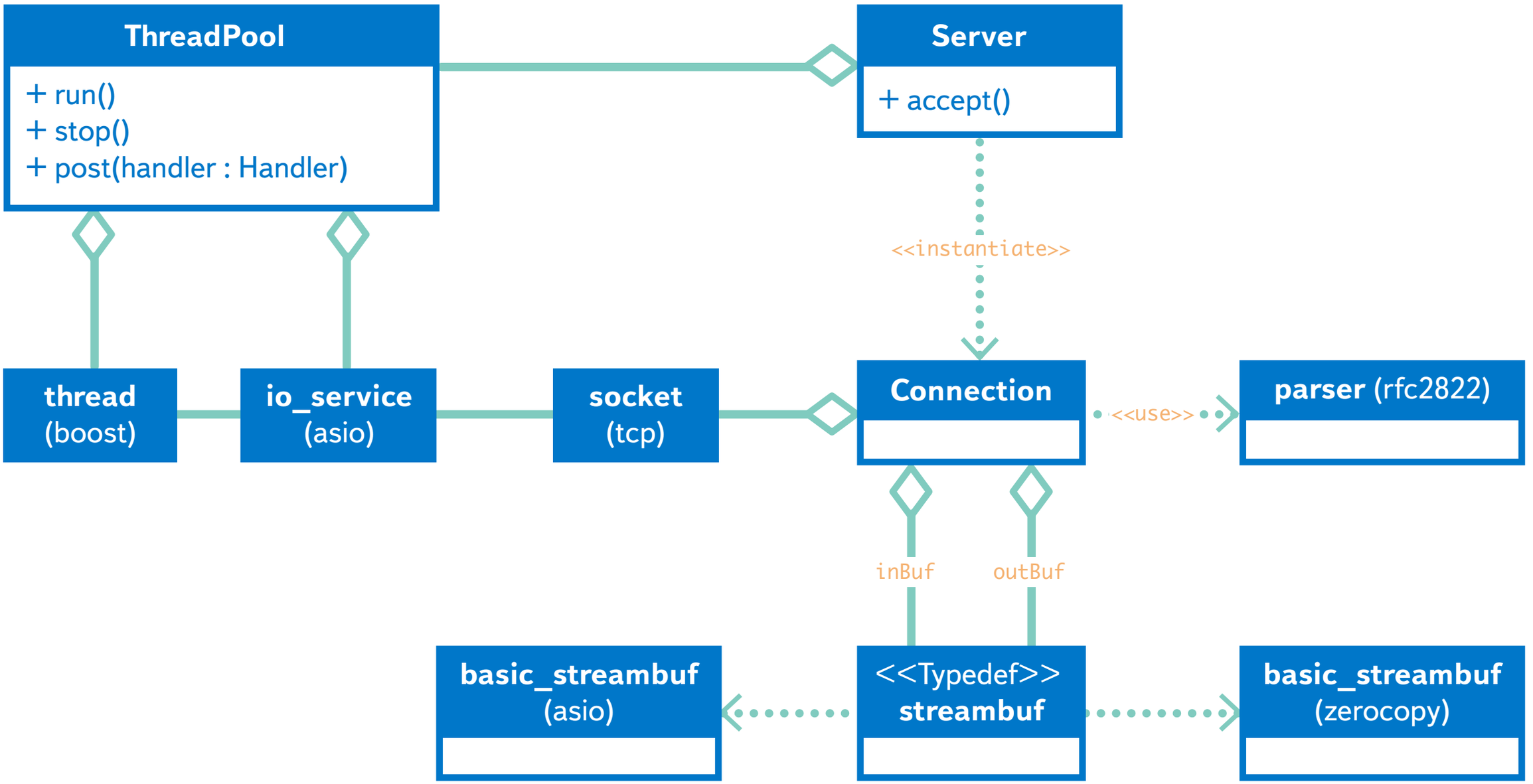
- Thread per io_service
- Multithread per io_service

Buffer for I/O

- Continuous with reallocation
- Pseudo-continuous:

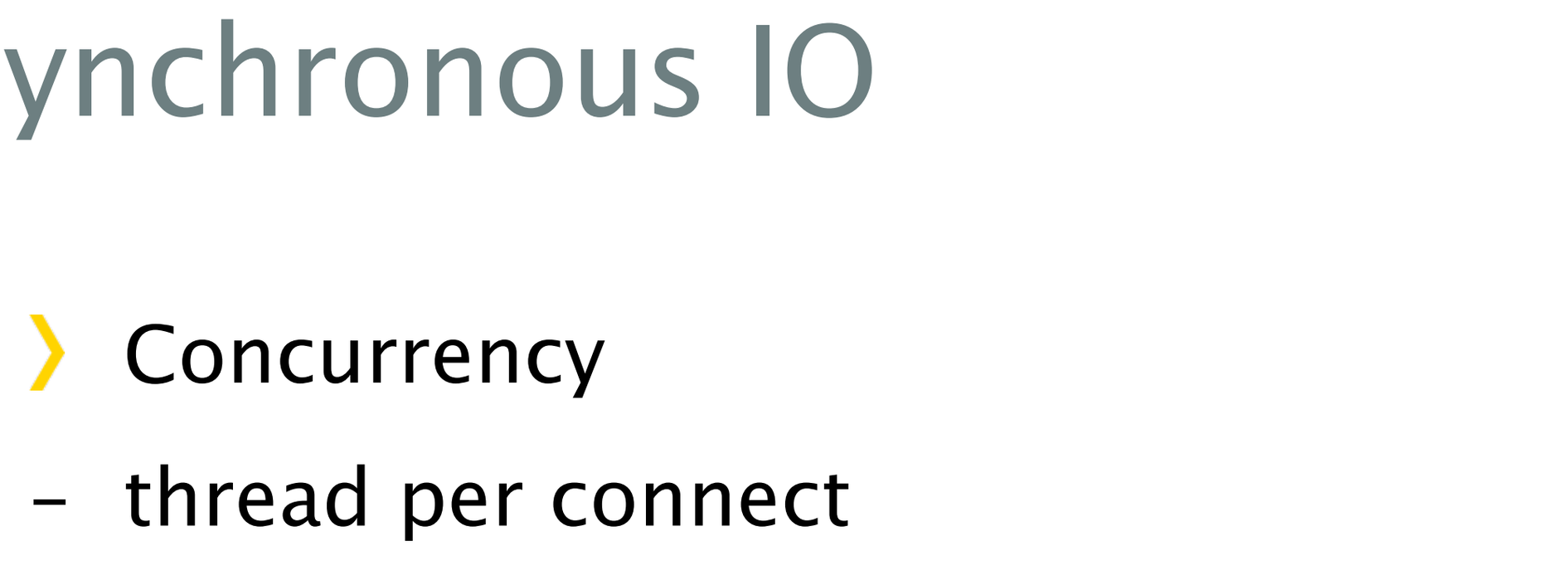
The Models

Synchronous and asynchronous server architecture



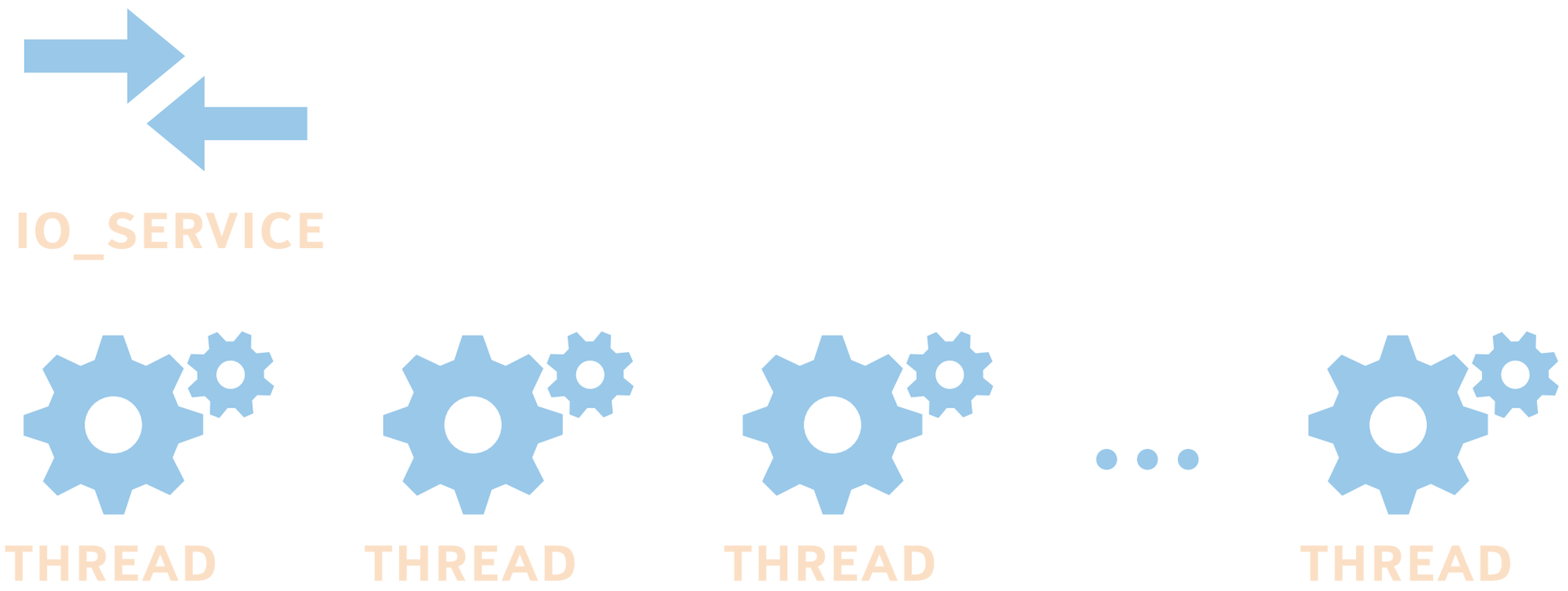
Synchronous IO

- Concurrency
- thread per connect
- process per connect



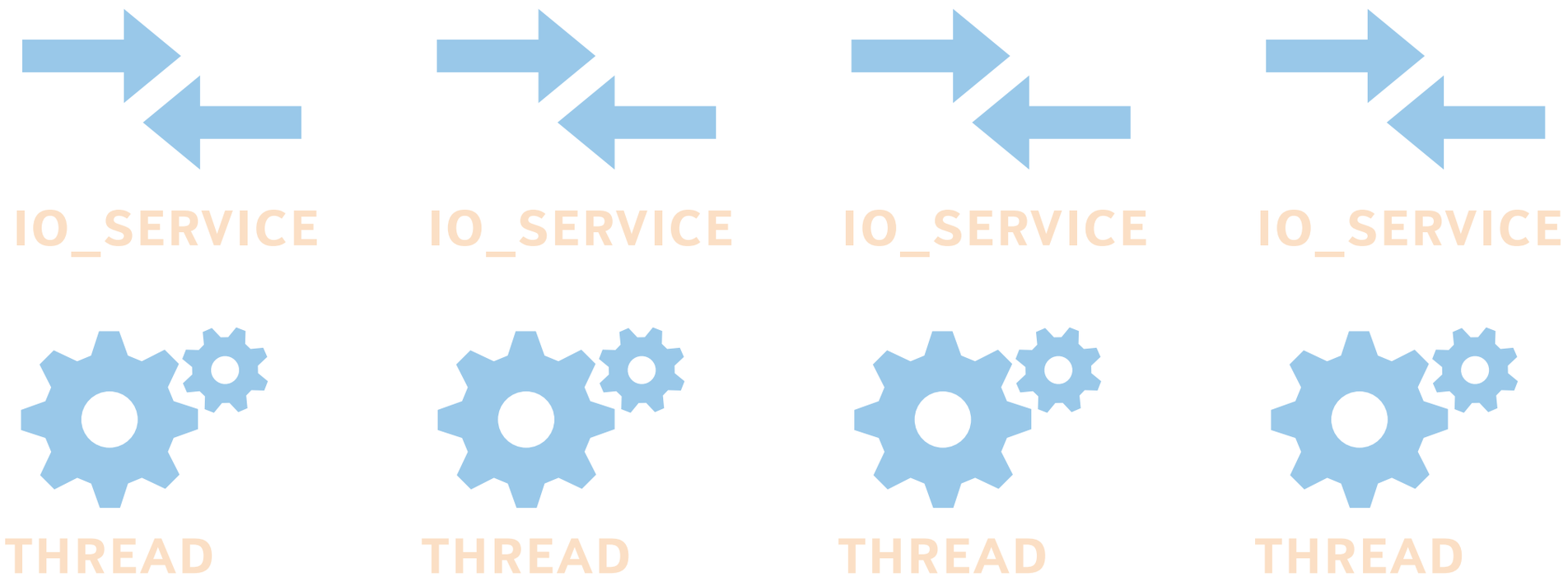
Asynchronous IO model 1-N

- Configuration
- single io_service object
- one thread per CPU core



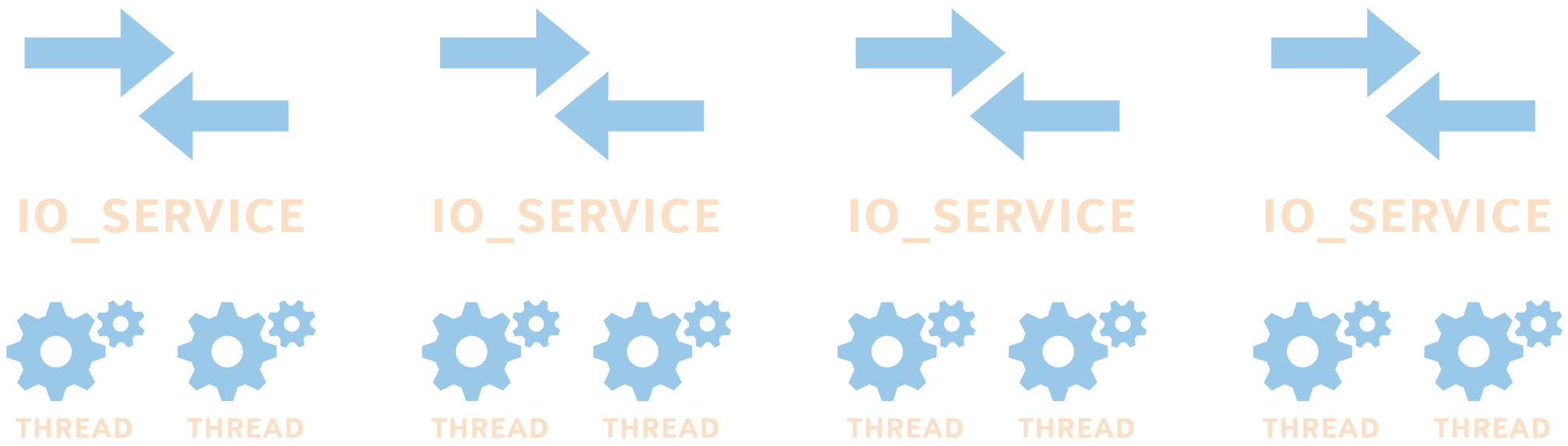
Asynchronous IO model N-N

- Configuration
- io_service object per CPU core
- each thread is bound to the dedicated CPU core



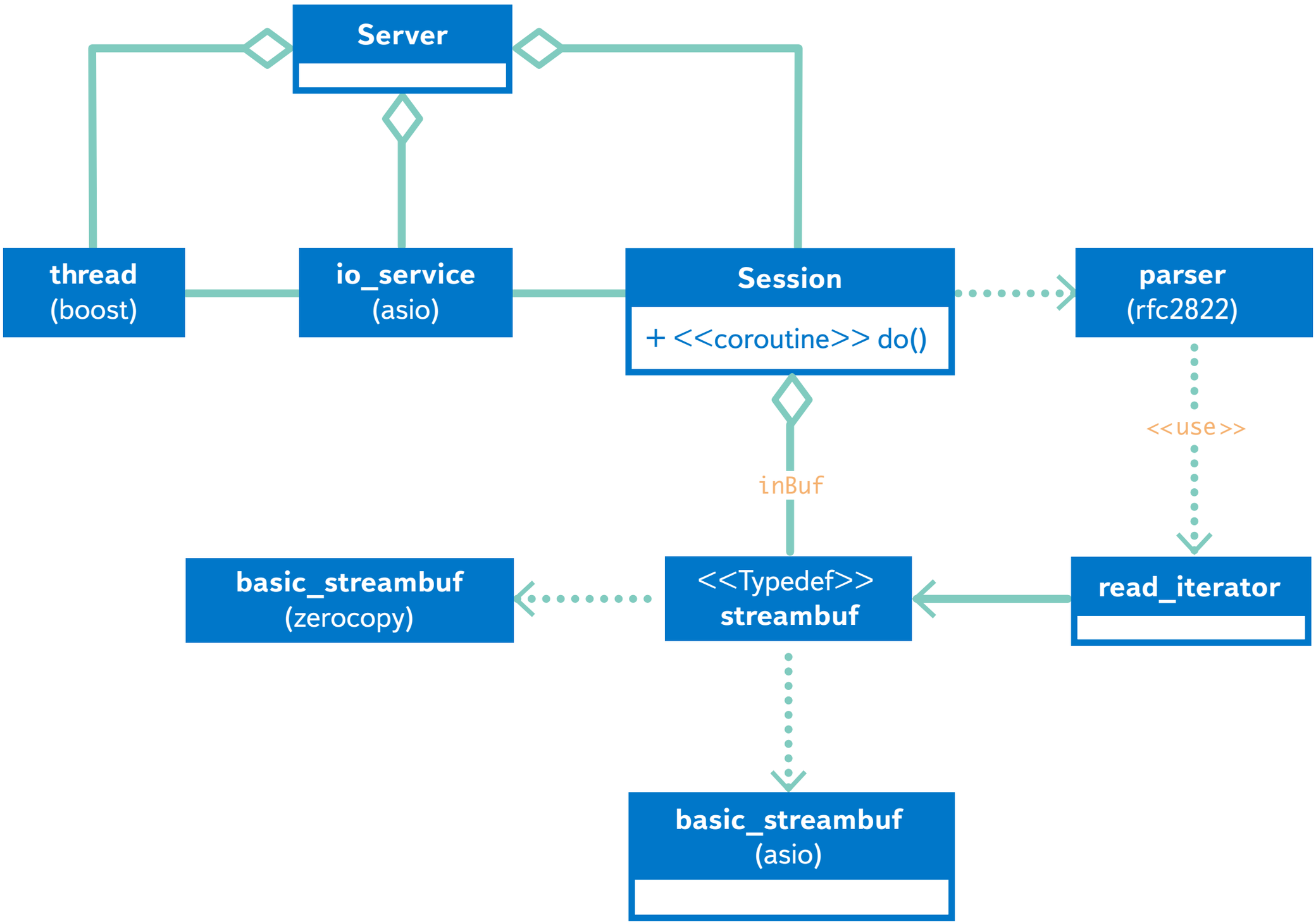
Asynchronous IO model N-2N

- Configuration
- io_service object per CPU core
- two threads per io_service (for HT CPUs)



The Coroutine

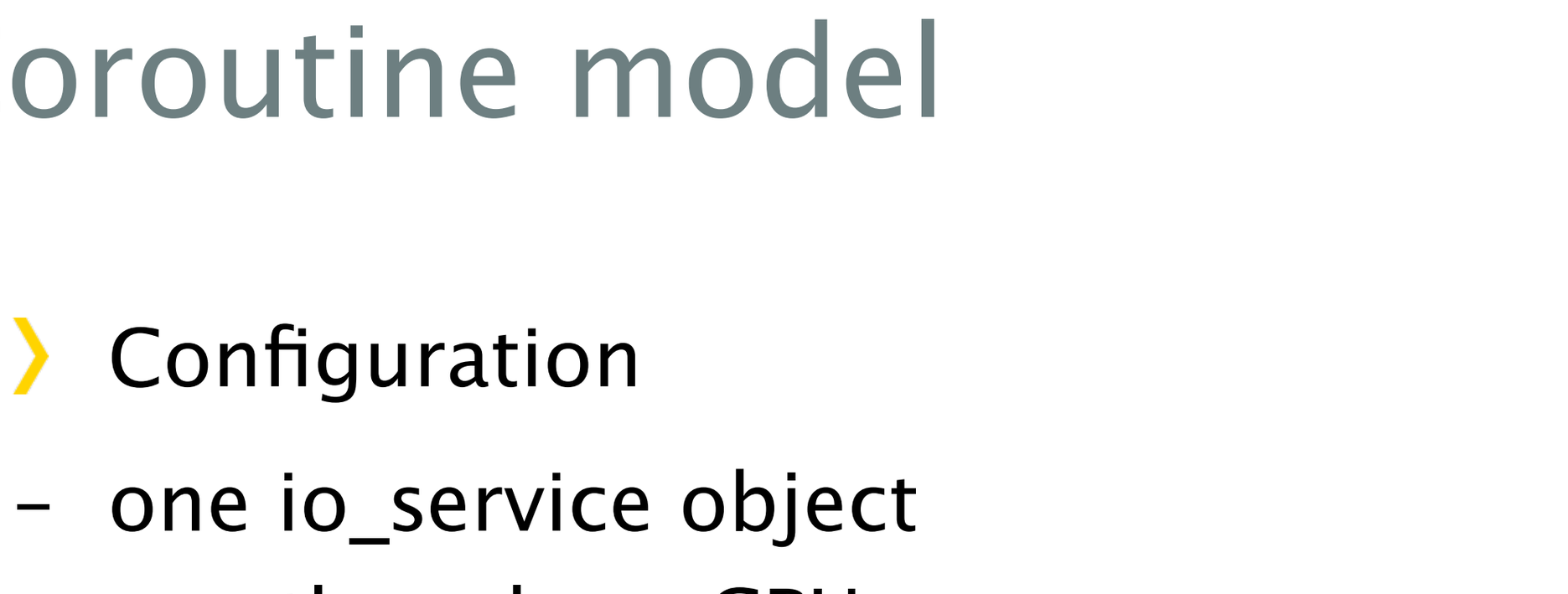
Coroutine driven sever



Workflow: read_iterator

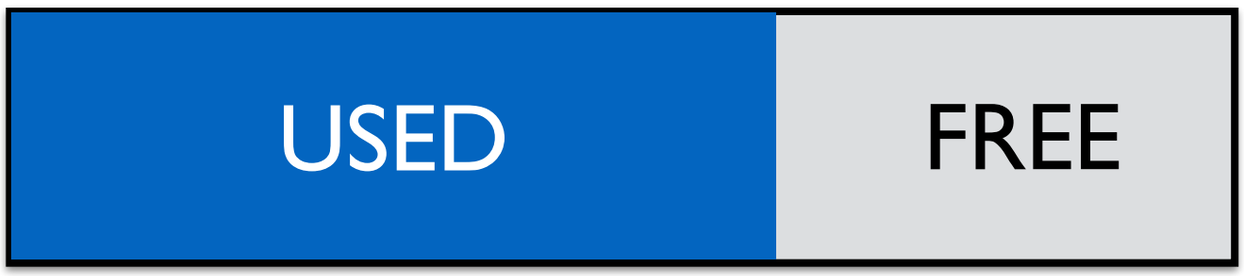
Coroutine model

- Configuration
- one io_service object
- one thread per CPU core

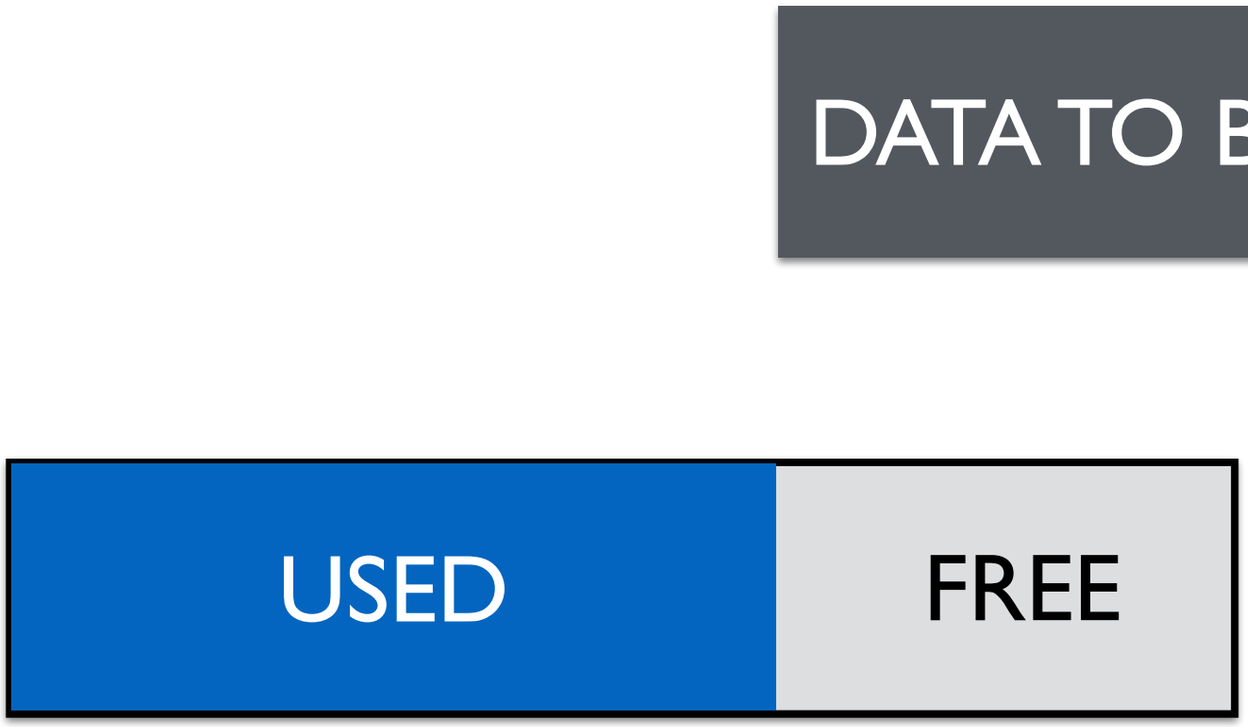


The Zerocopy Buffer

Workflow: asio

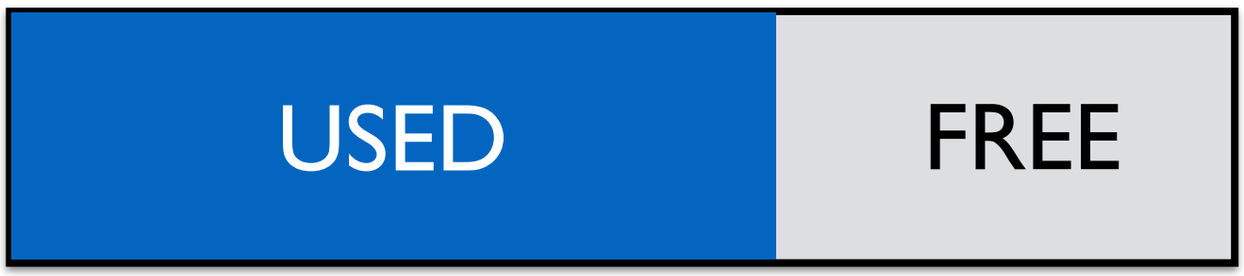


Workflow: asio



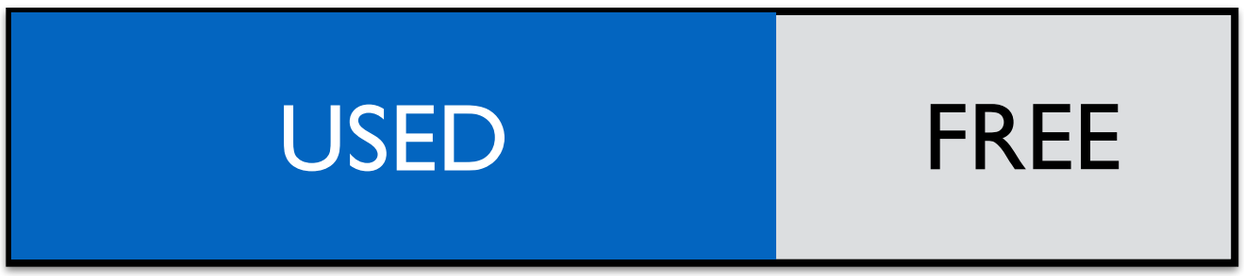
Workflow: asio

DATA TO BE WRITTEN



Workflow: asio

DATA TO BE WRITTEN



FREE

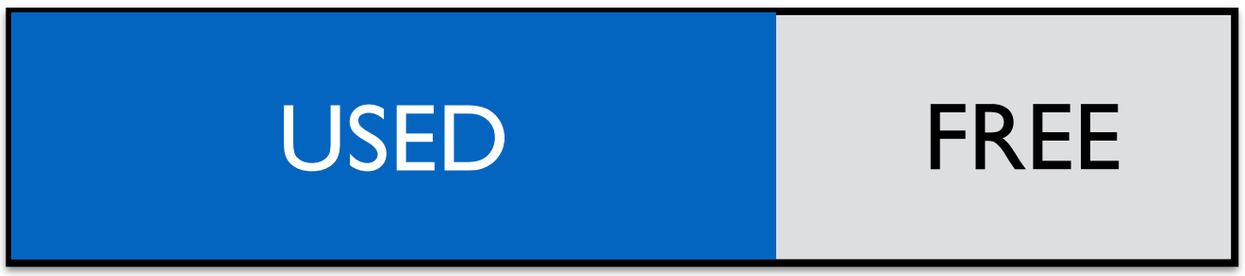
The main problems

- It allocates a bigger memory chunk while owns old one
- It copies an old data for each reallocation
- The application has to copy the data from the buffer to use it later

The solution is

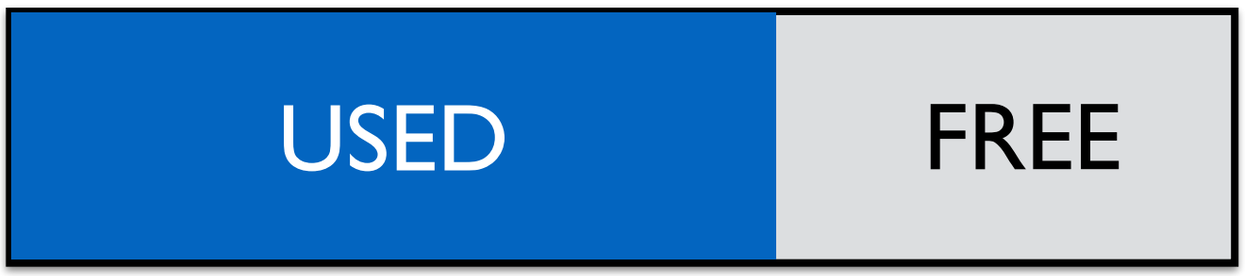
- To do not make reallocation
- To do not make any copy of the old data
- To use the data from buffer directly

Workflow: zerocopy

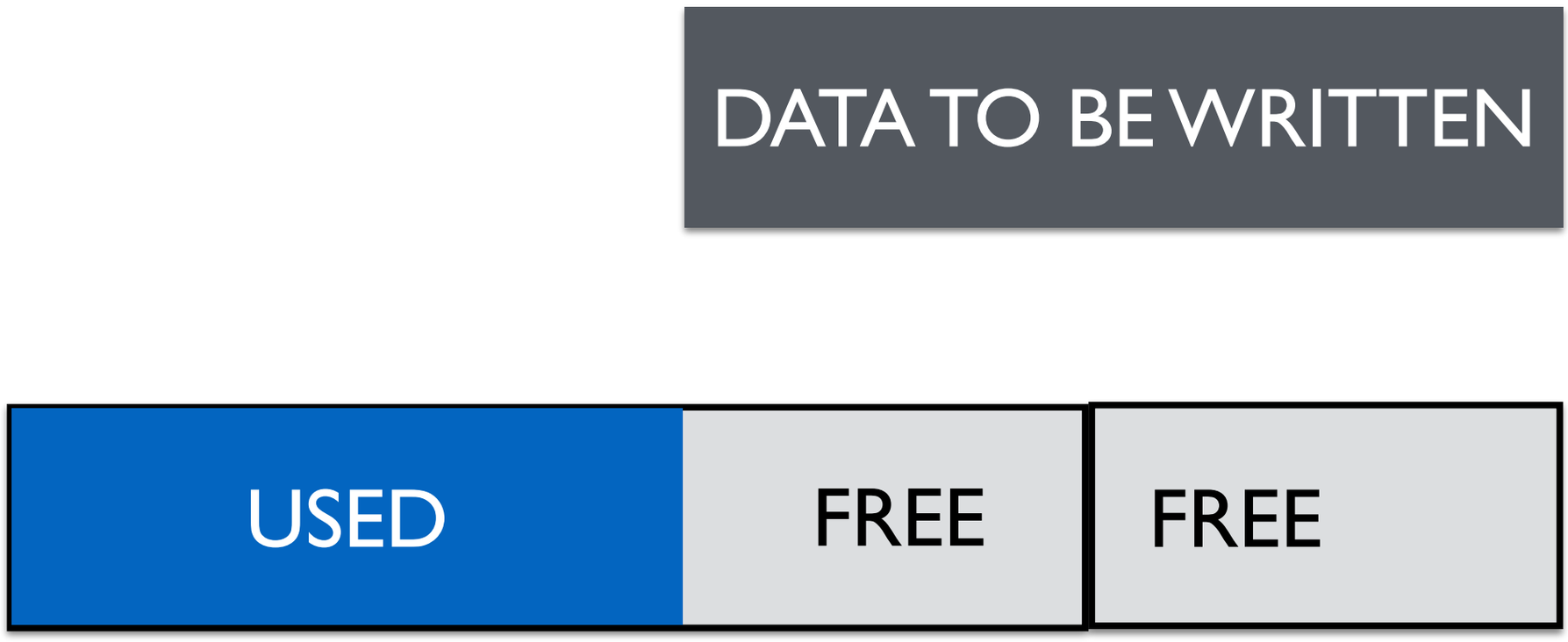


Workflow: zerocopy

DATA TO BE WRITTEN

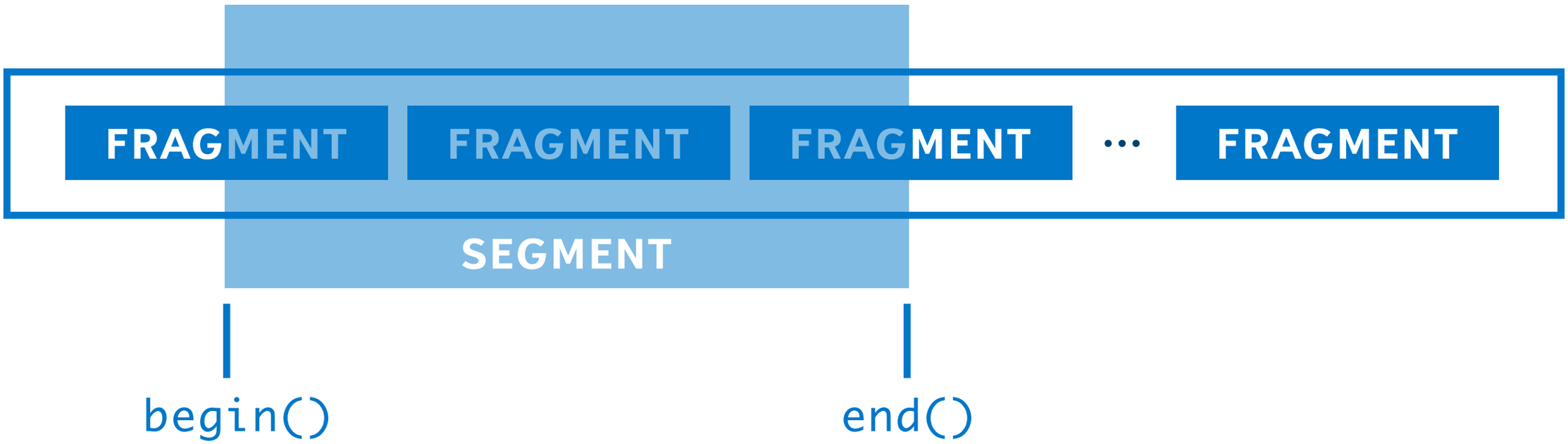


Workflow: zerocopy



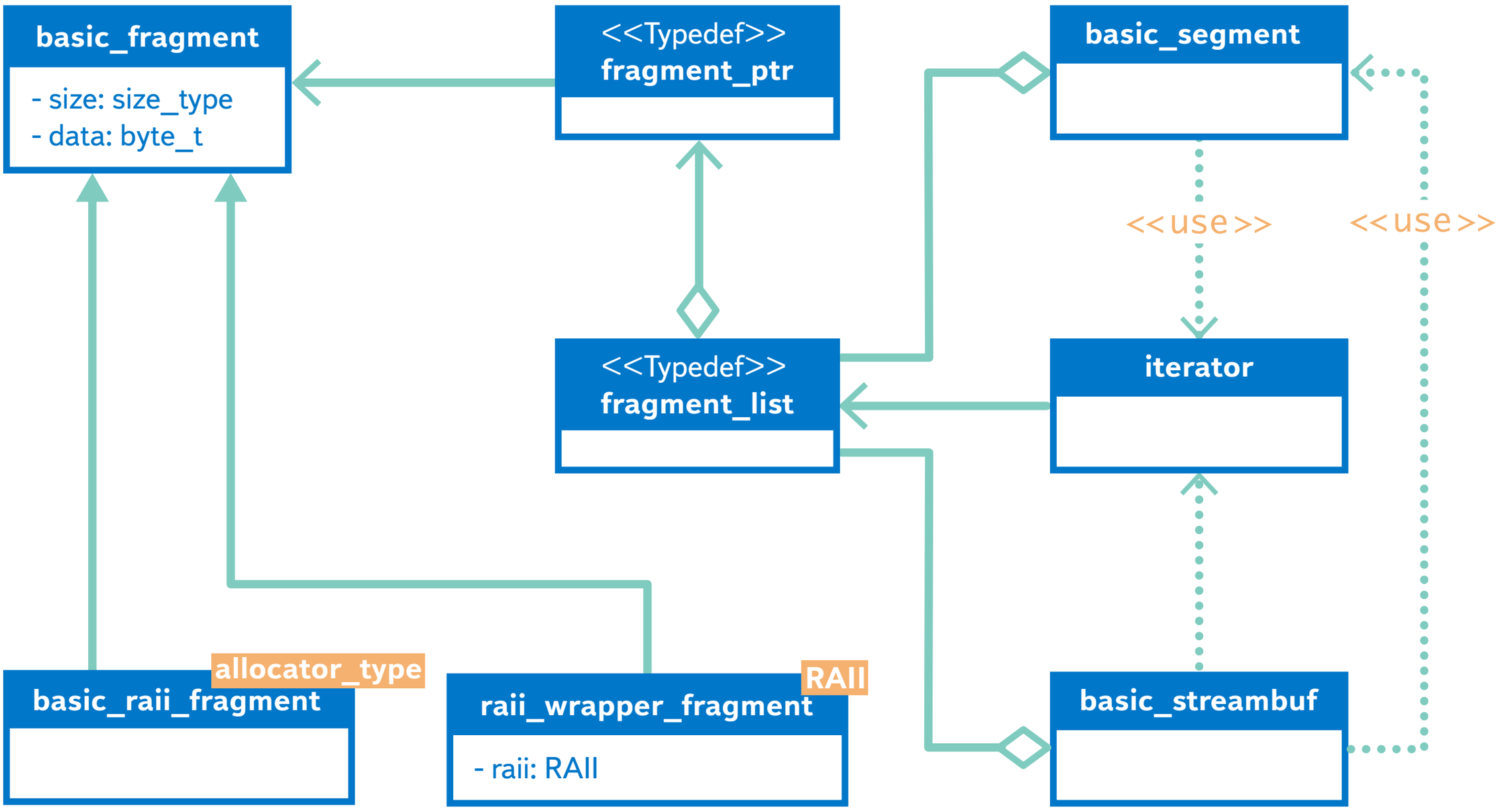
Workflow: zerocopy

The concept

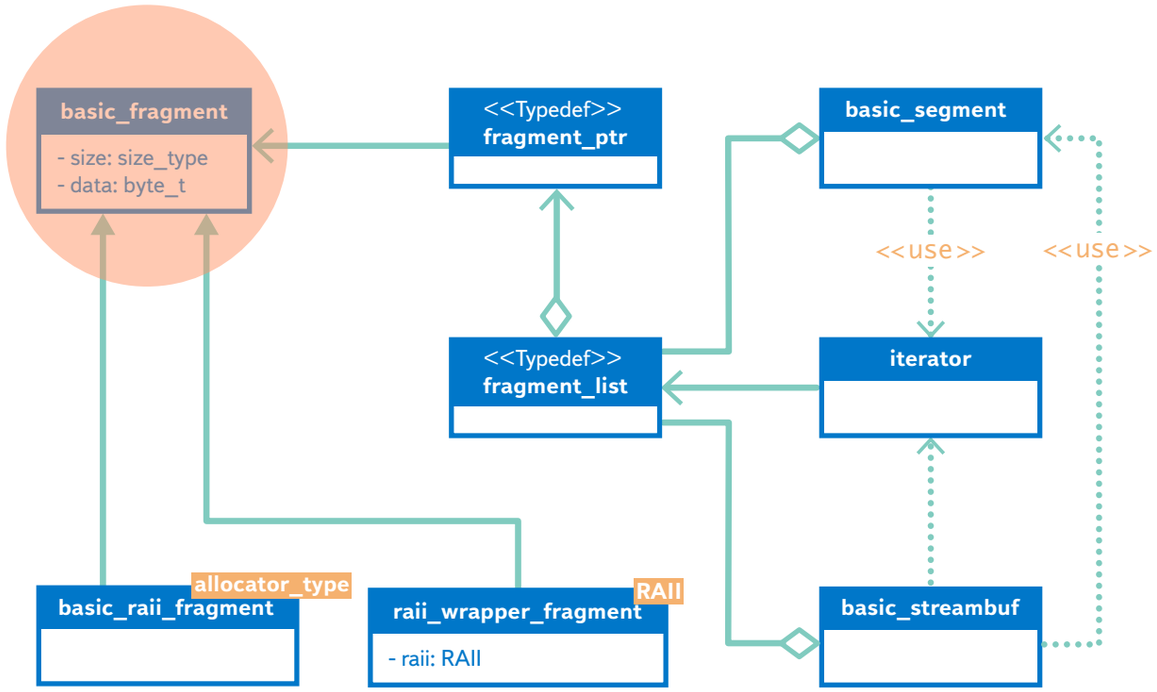


It looks like a continuous memory interval.

Zerocopy architecture



Zerocopy architecture

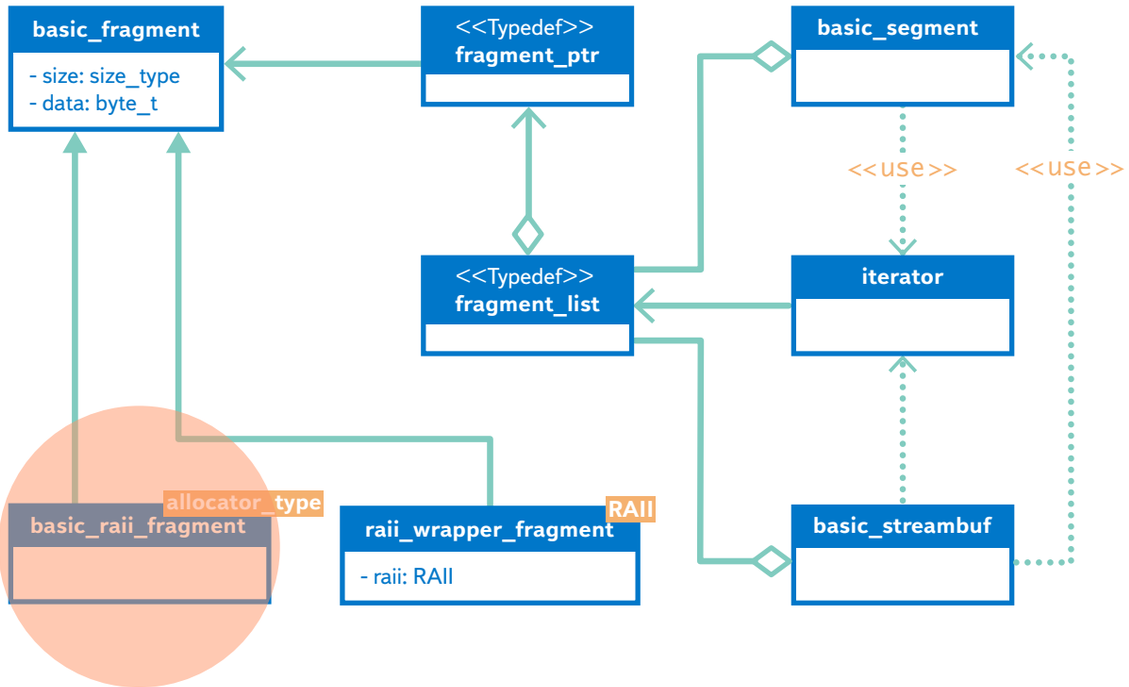


basic_fragment

- size: size_type
- data: byte_t

+ begin()
+ end()
+ cbegin()
+ cend()
+ size()
+ contains(const_iterator)
+ basic_fragment(byte_t *, size_type)
# set_data(byte_t *, size_type)

Zerocopy architecture

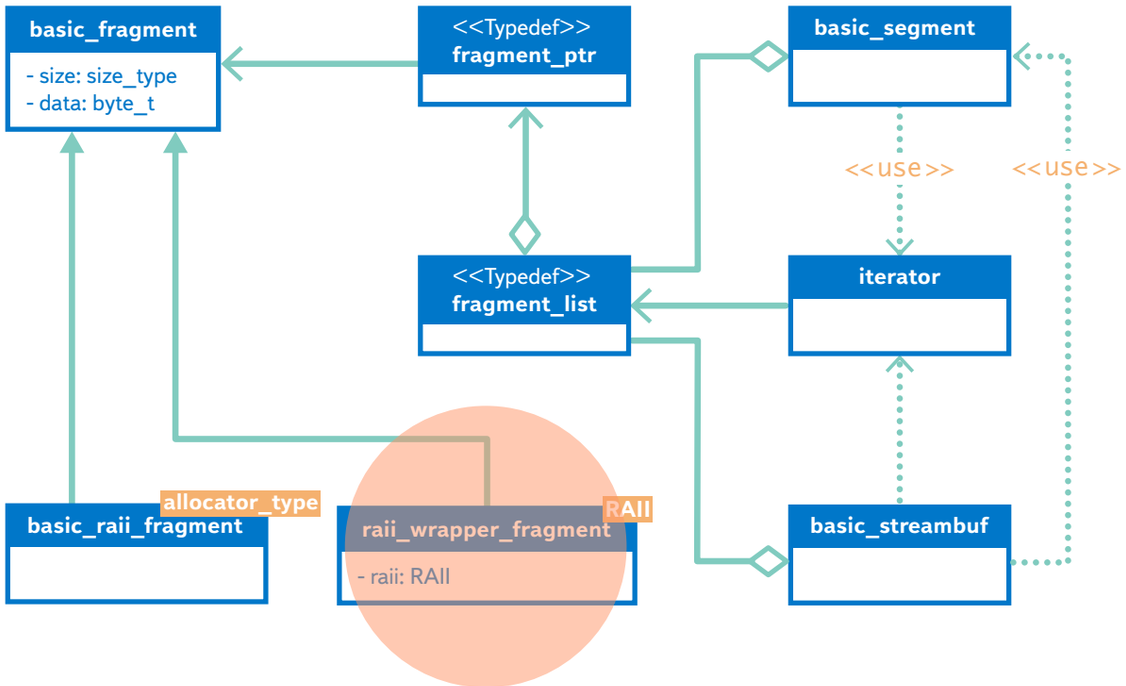


allocator_type

basic_raii_fragment

+ basic_raii_fragment(size_type, allocator_type)
- allocate(size_type)
- deallocate()

Zerocopy architecture



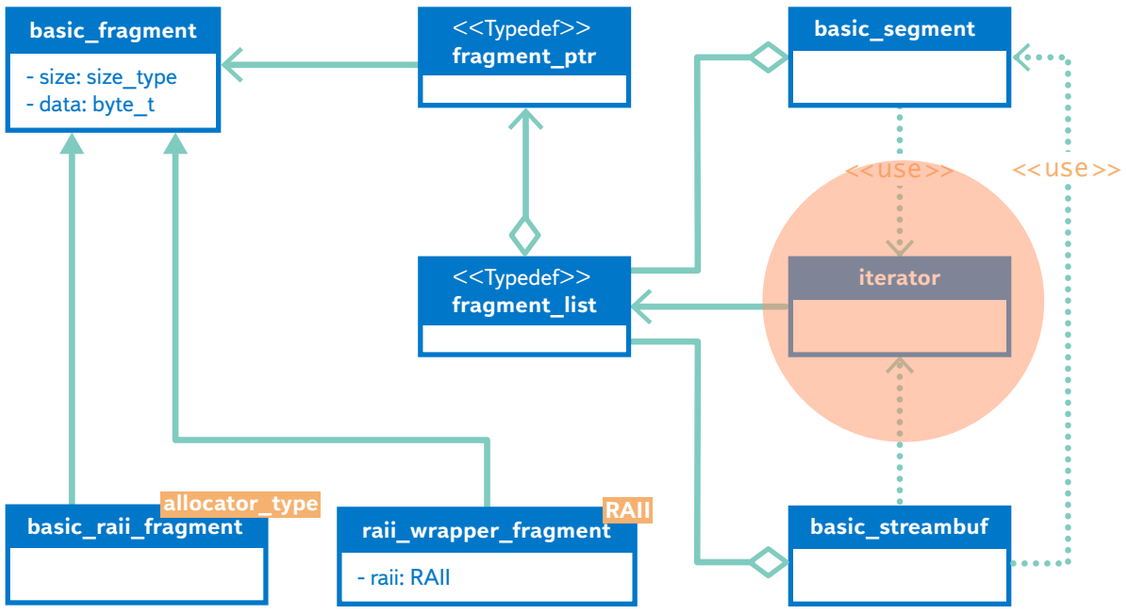
RAII

raii_wrapper_fragment

- raii: RAII

+ raii_wrapper_fragment
  (byte_t *, size_type, RAII)

Zerocopy architecture

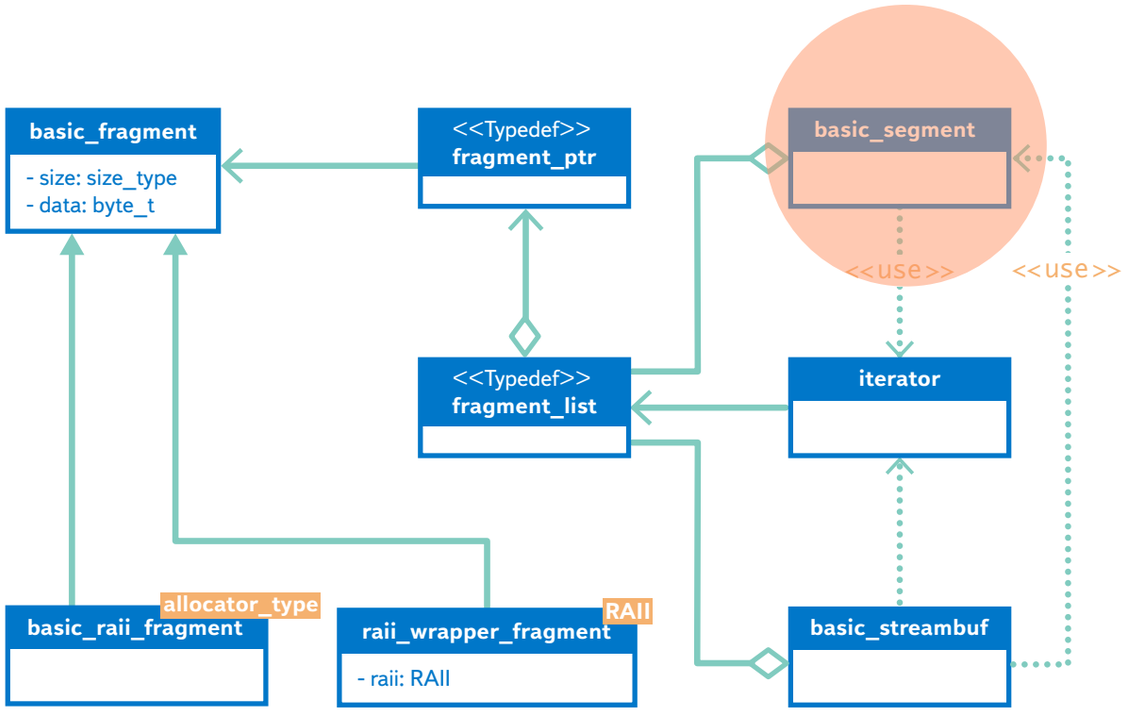


iterator

-list: fragment_list&
-fragment: f_iterator
-pos: p_iterator

+operator ->()
+operator *()
+operator ++()
+operator --()

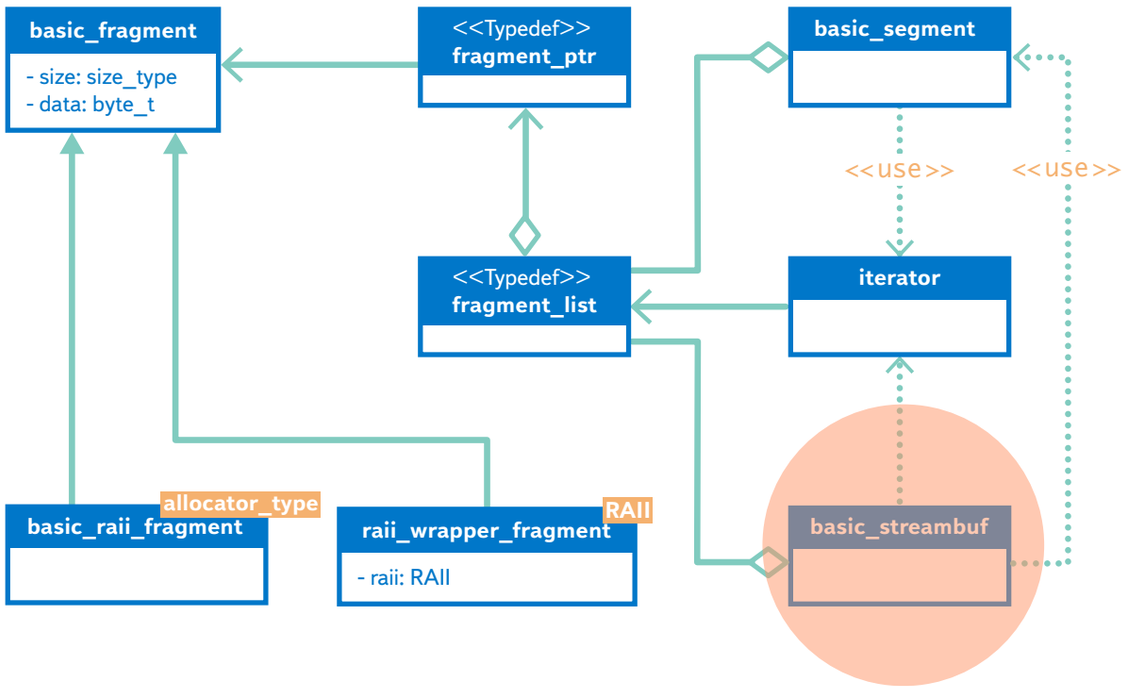
Zerocopy architecture



basic_segment

+ begin()
+ end()
+ cbegin()
+ cend()
+ append(basic_segment)
+ size()
+ empty()
+ swap(basic_segment &)

Zerocopy architecture



basic_streambuf

+ size()
+ data()
+ consume(size_type)
+ commit(size_type)
+ prepare(size_type)
+ xsputn(char_type *, streamsize)
+ xsgetn(char_type *, streamsize)
+ begin()
+ end()
+ segment_type detach(iterator)

How to get data from the

Should we use the std::istream?

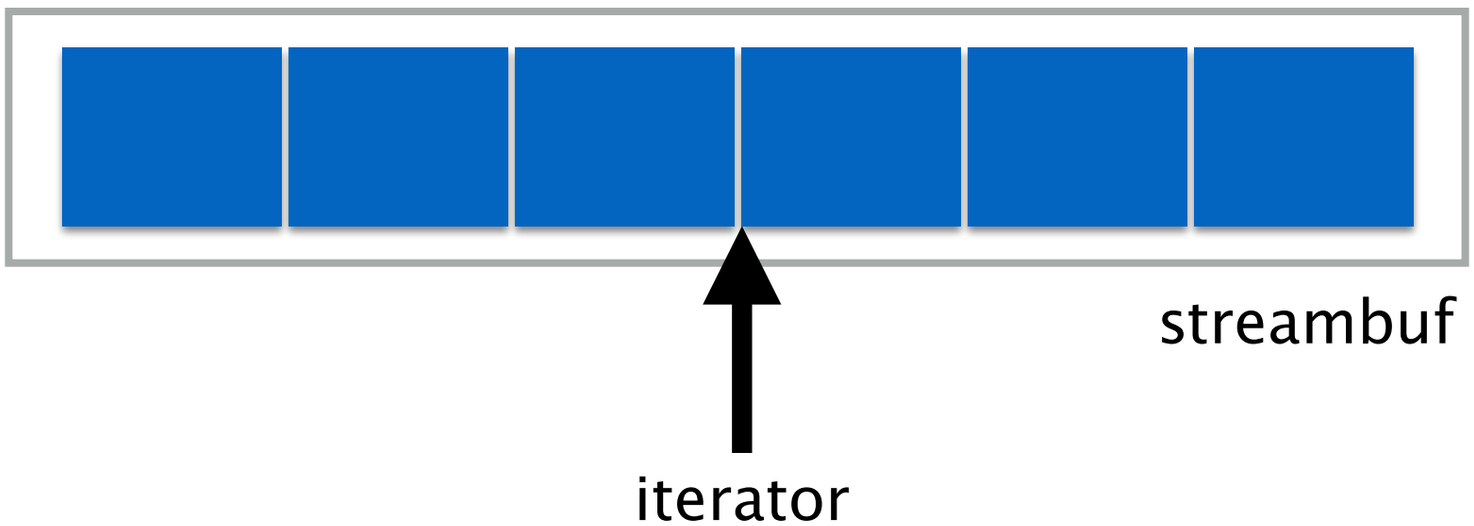
```
zerocopy::streambuf inBuf;

// Get some data into the inBuf

std::istream s(&inBuf);
s.unsetf(std::ios::skipws);

spirit::istream_iterator first(s);
spirit::istream_iterator last;

rfc822::parse(first, last);
```

How to get data from the

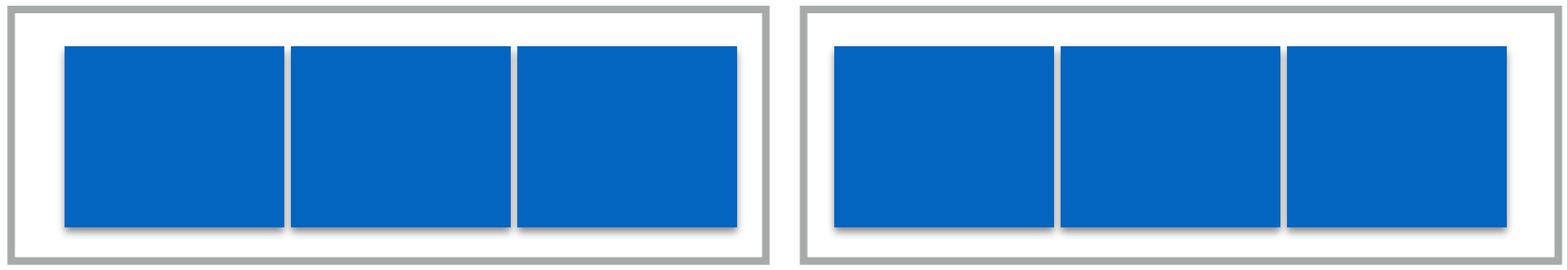
We should use the streambuf::detach(iterator)

```
zerocopy::streambuf inBuf;

// Get some data into the inBuf

auto segment = inBuf.detach(inBuf.end());

rfc822::parse(segment.begin(), segment.end());
```

Detaching Segment

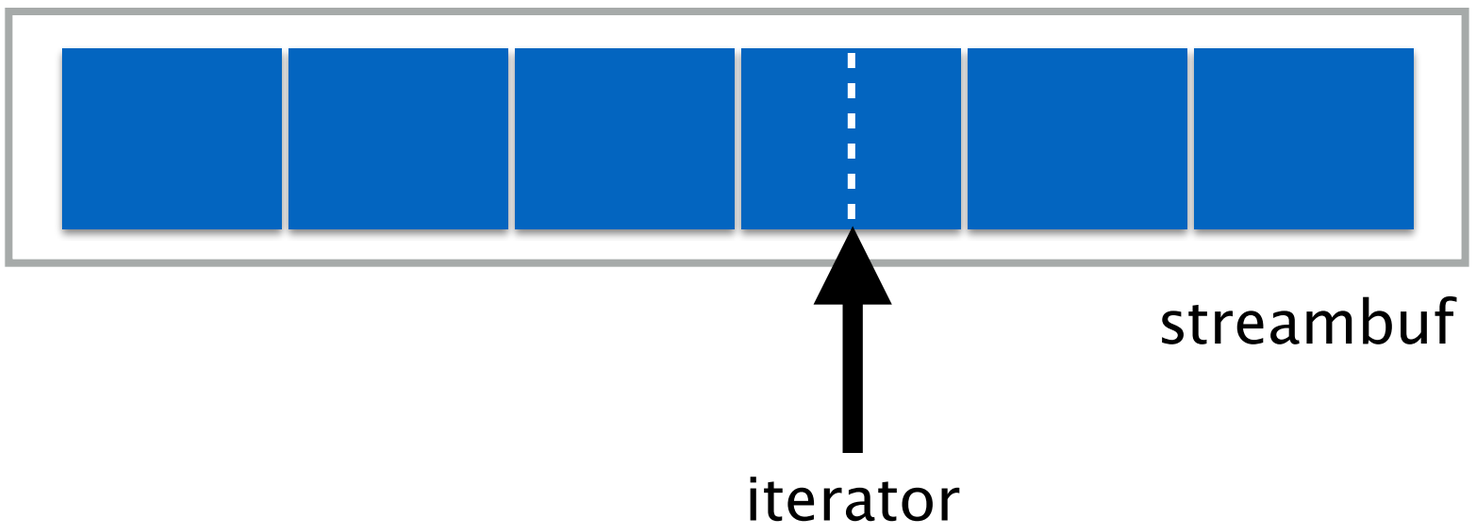


Detaching Segment

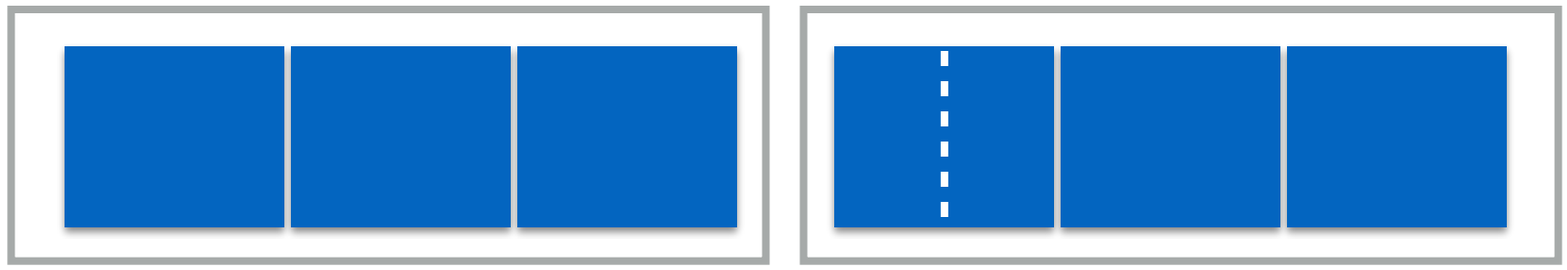


segment streambuf

Detaching Segment

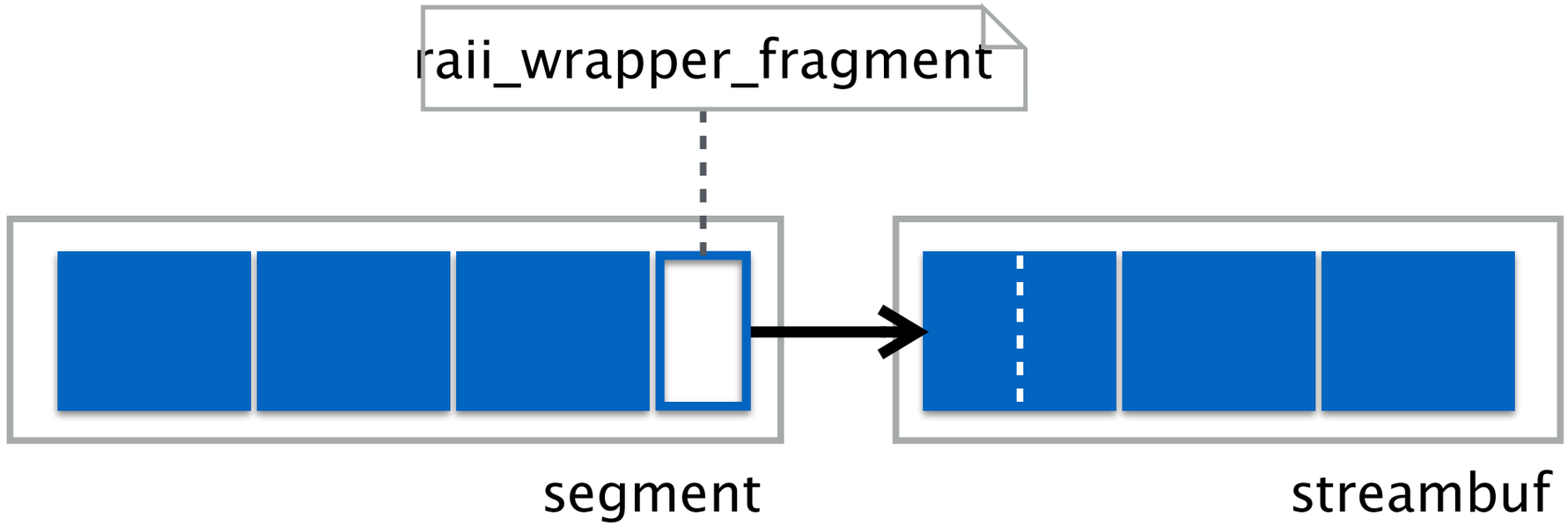


Detaching Segment

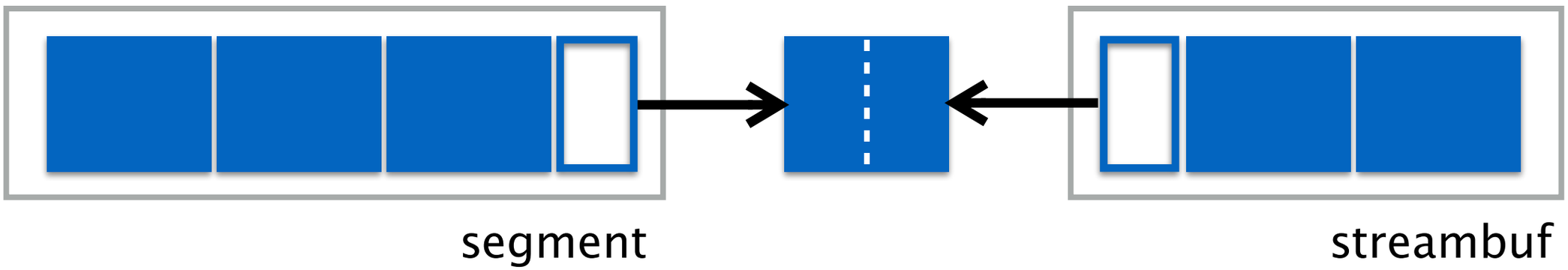


segment streambuf

Detaching Segment

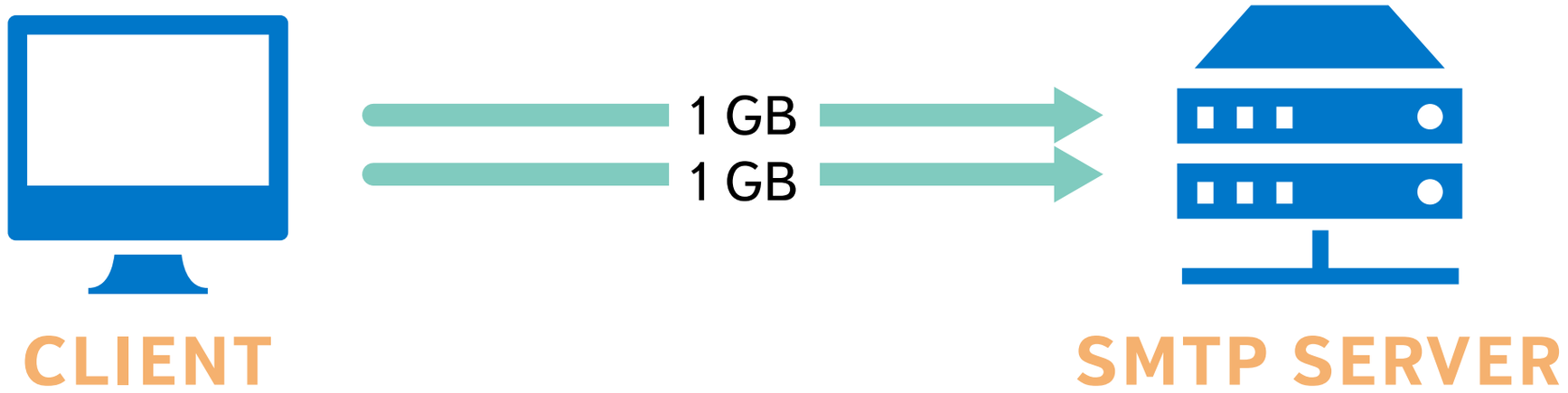


Detaching Segment



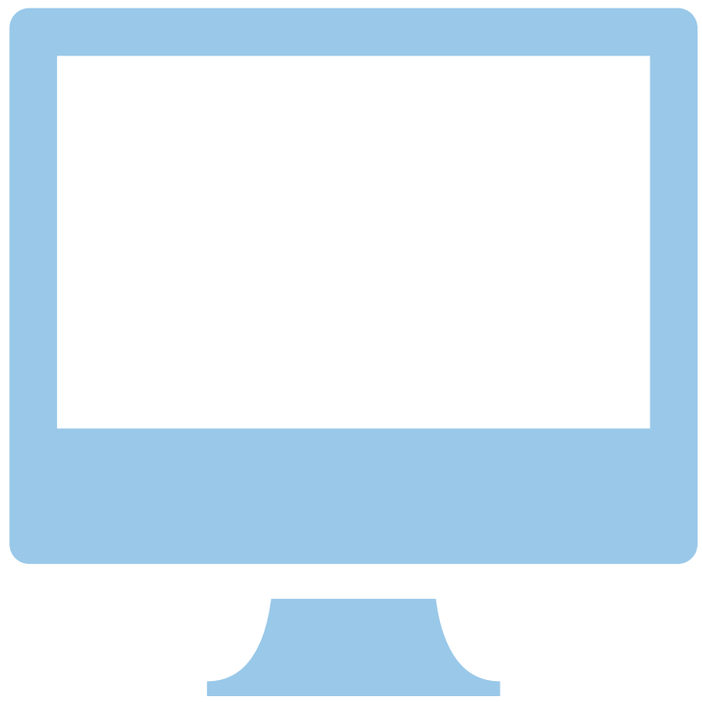
The experiment

Methodology



The Load

- Client architecture
  - Boost.Asio based asynchronous multithreaded client
- Load setup
  - 6000 connections
  - 9KB mean message size



Hardware

- CPU
  - 2 x Intel(R) Xeon(R) CPU E5645 @ 2.40GHz
  - 24 threads, 12 cores, 12 MB cache
- Memory
  - 48 Gb (6 x 8Gb)
- Network
  - 2 x Intel Corporation 82574L Gigabit Network Connection

The setup of the server and the client is identical.

Software

- OS
  - Ubuntu 14.04
- Boost v1.55
  - Asio
  - Spirit 2
- Compilers
  - GCC 4.9.0 beta

Measurements

Server-side

- RAM consumption
- CPU consumption
- Network load

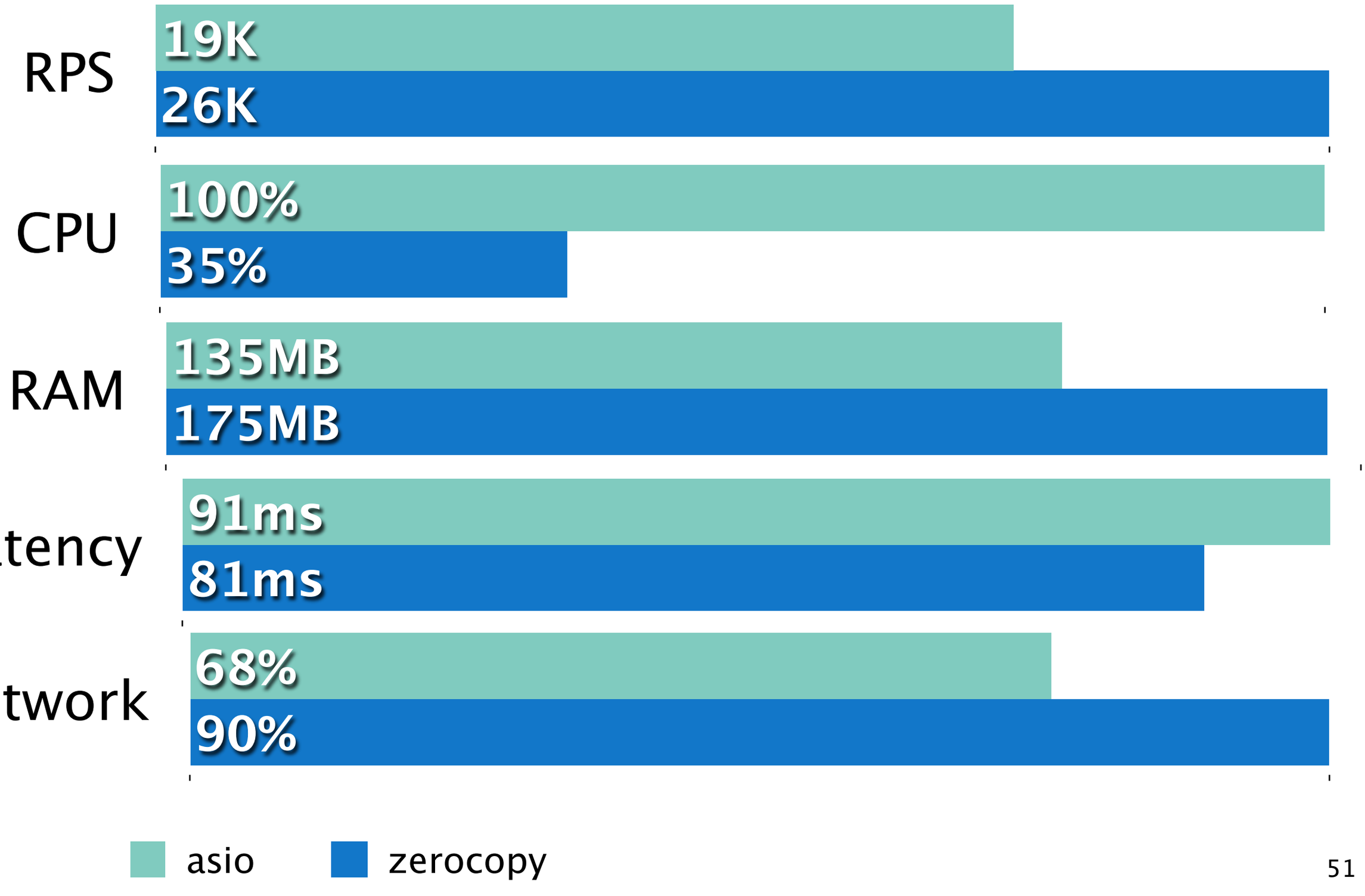
Client-side

- Latency
- RPS

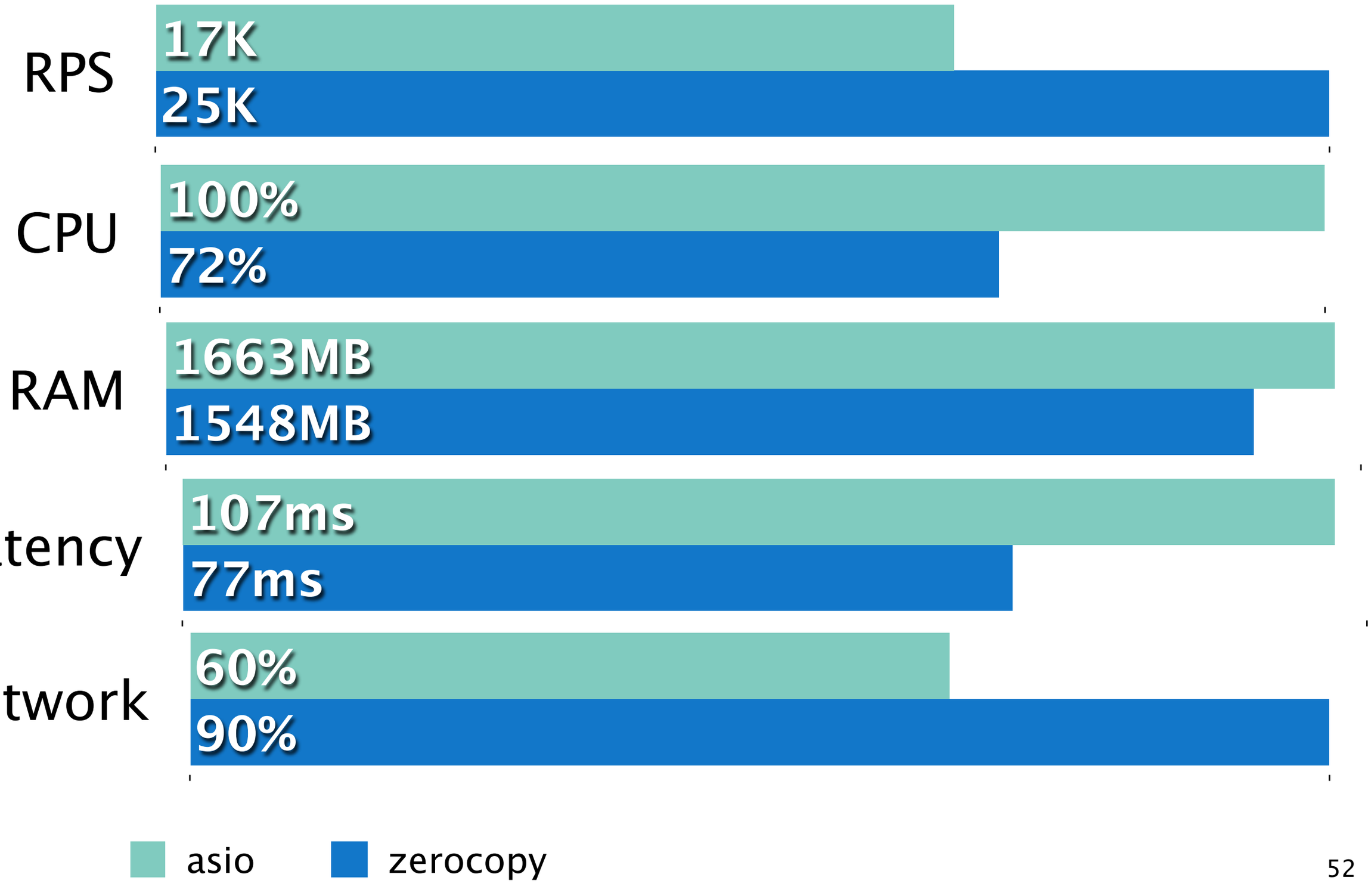
Results

Guess who's the winner

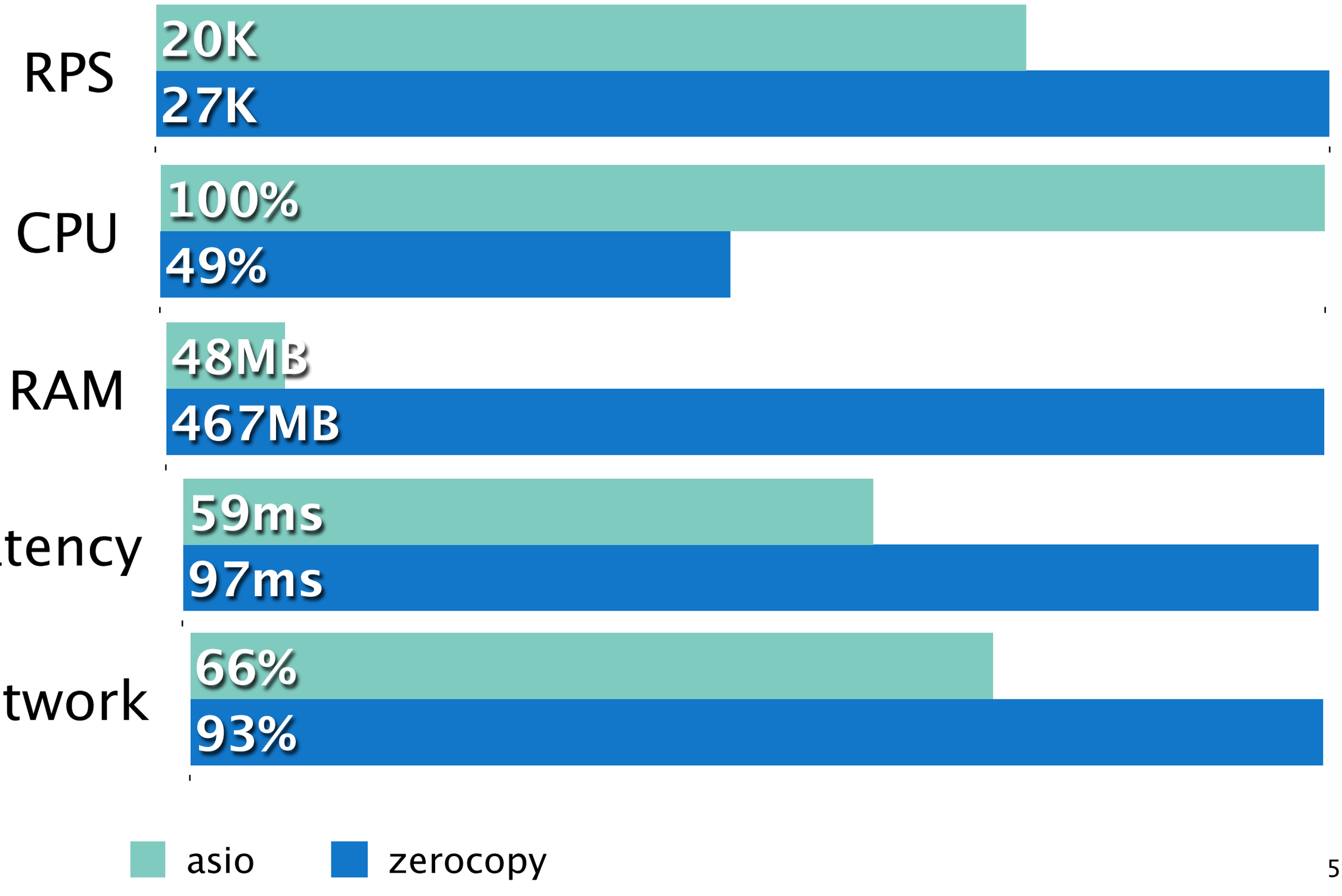
Multithread Synchronous IO



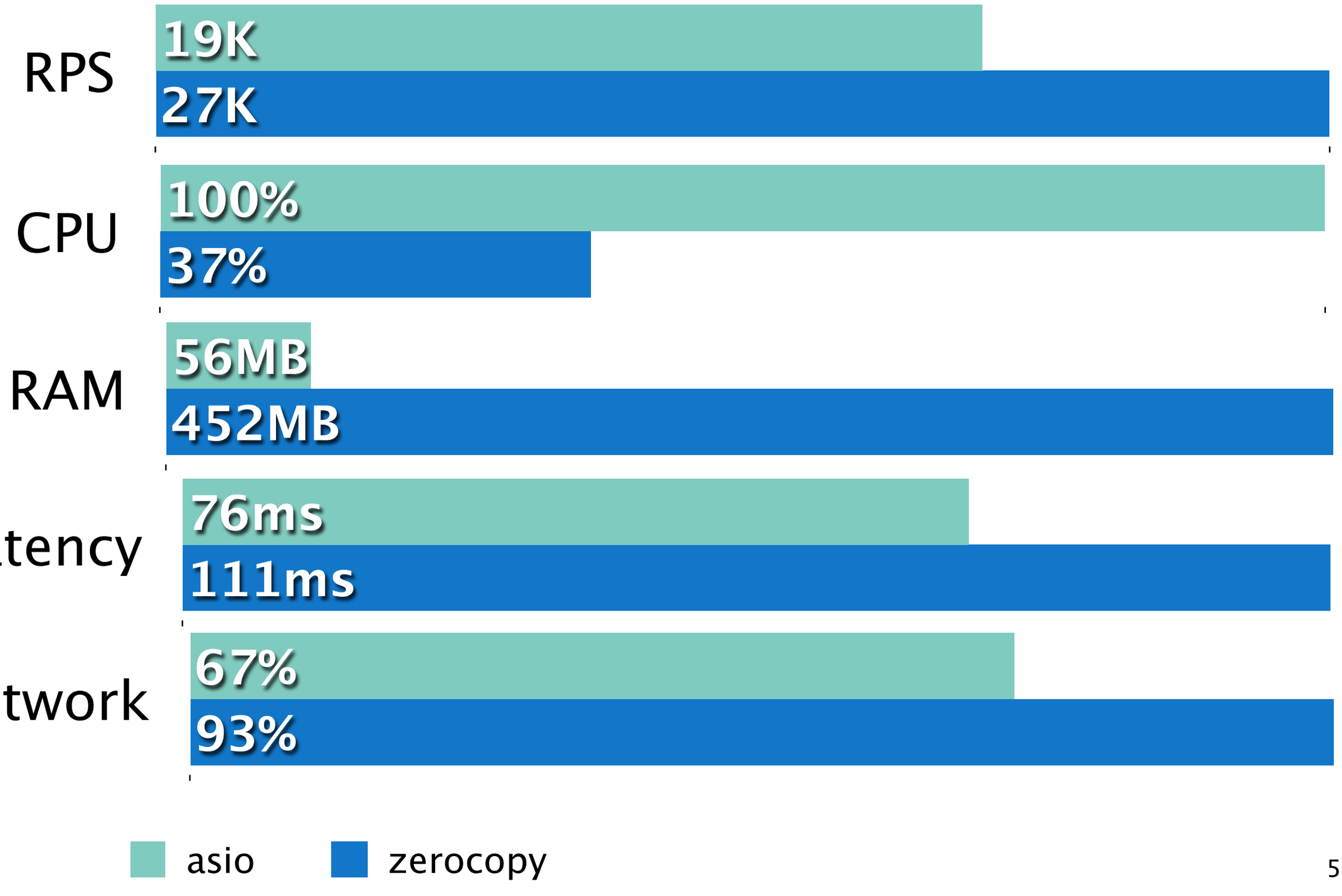
Multiprocess Synchronous IO



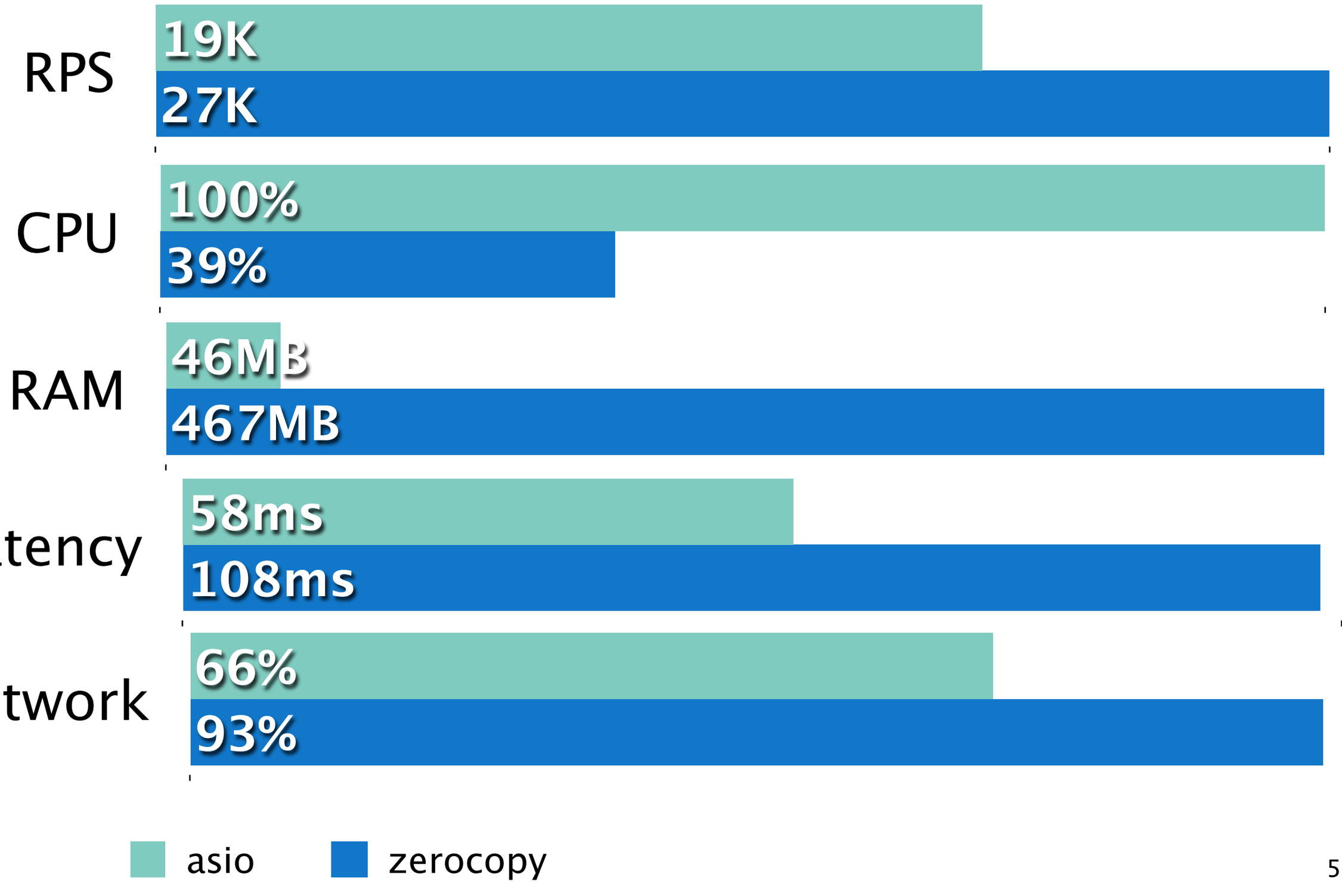
Asynchronous IO “1—N”



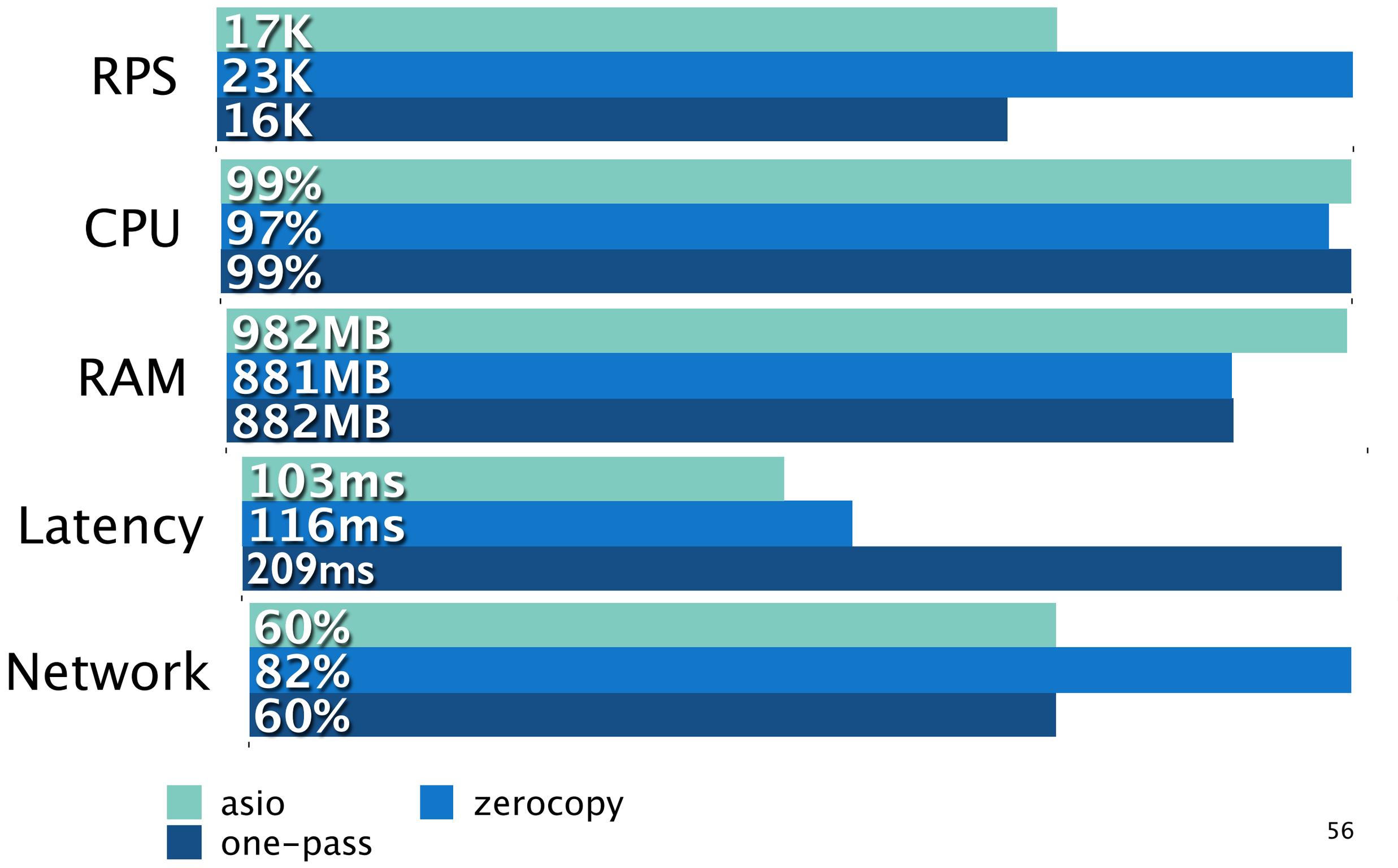
Asynchronous IO “N—N”



Asynchronous IO “N—2N”



Coroutine


RPS
17K
23K
16K
CPU
99%
97%
99%
RAM
982MB
881MB
882MB
Latency
103ms
116ms
209ms
Network
60%
82%
60%
asio
zerocopy
one-pass

Zerocopy advantages

- RPS advantage more than 35%
- Latency advantage more than 10%
- CPU utilization – up to 75%

What next?

- We would like to see the zerocopy stream buffer implementation in Boost.Asio

The Project52

- Please, explore the code at https://github.com/YandexMail/project52

Thank you.