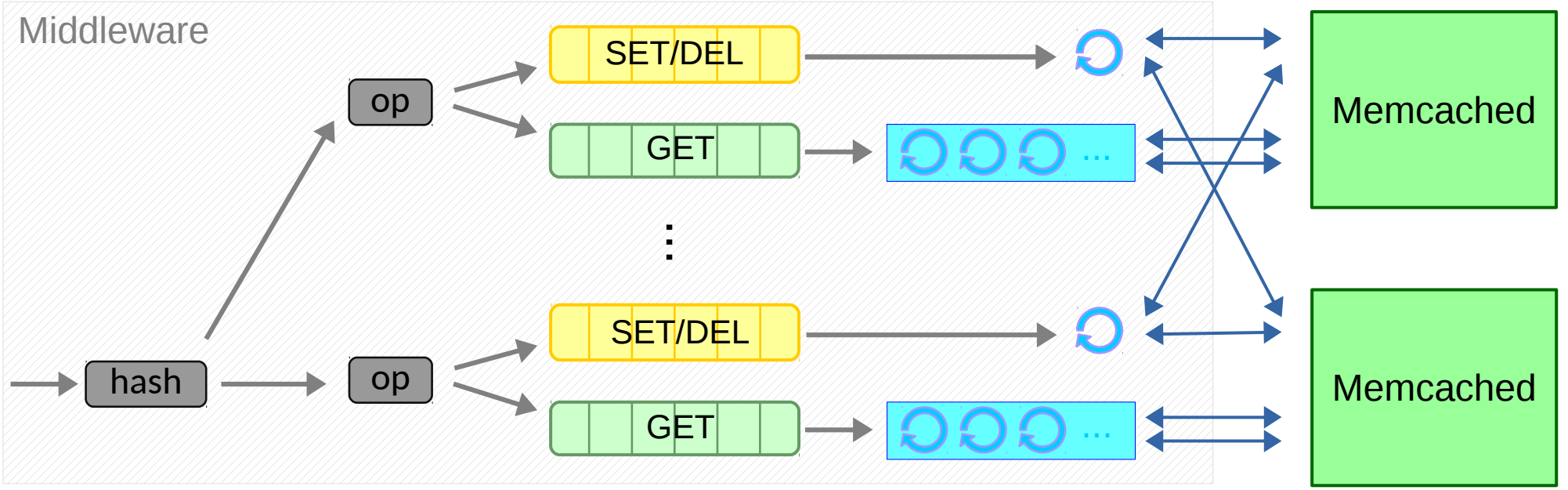
Java Development: Do’s and Don’ts

Overview

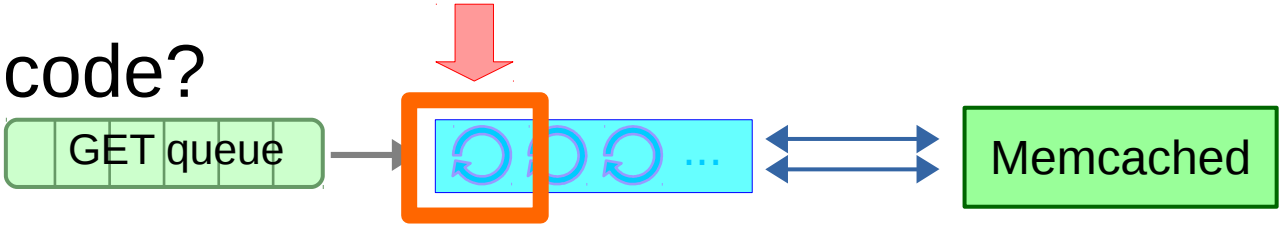
- **Asynchronous networking**
- **Data copies**
- Connection management
- Logging

ASL Project 2016: Middleware for different replication modes

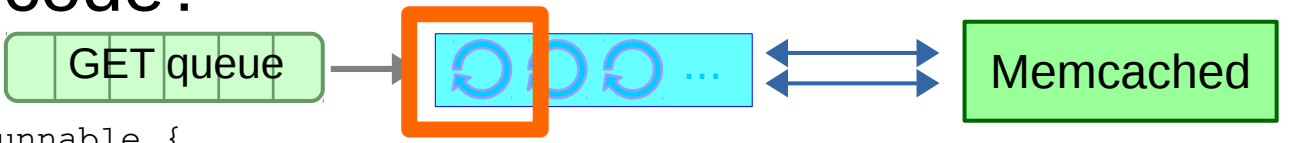


- Middleware provides multiple replication schemes
- `Sets` and `Deletes` are handled by an asynchronous thread (one for each server)
- `Gets` are handled by a synchronous reader thread pool

What is wrong with this code?

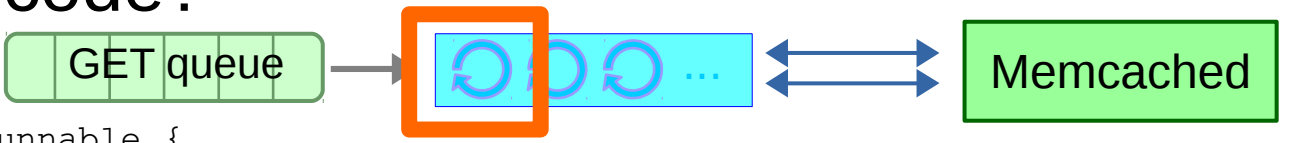


What is wrong with this code?




```
public class ReadHandler implements Runnable {
     [...]
     public ReadHandler(BlockingQueue<Request> queue, InetSocketAddress serverAddr){
          this.queue = queue;
          server = SocketChannel.open();
          server.connect(serverAddr);
          server.configureBlocking(false);
     }

     public void run(){
          while(true){
               Request request = queue.take();
               ByteBuffer command = ByteBuffer.wrap(request.getCommand());
               server.write(command);
               command.clear();

               ByteBuffer buf_header = ByteBuffer.allocate(message_size);
               int bytesRead = 0;
               do{
                    bytesRead = server.read(buf_header);
               } while(bytesRead == 0);
               [...]
     }}}
```

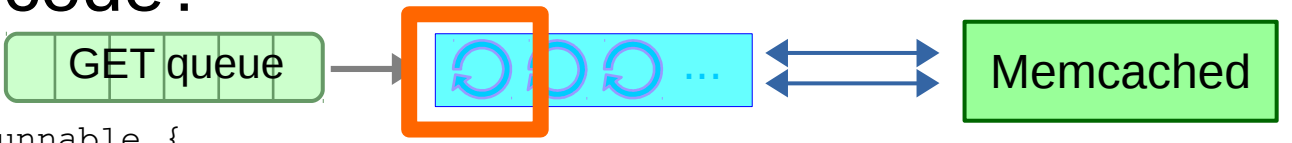
What is wrong with this code?




```
public class ReadHandler implements Runnable {
     [...]
     public ReadHandler(BlockingQueue<Request> queue, InetSocketAddress serverAddr){
          this.queue = queue;
          server = SocketChannel.open();
          server.connect(serverAddr);
          server.configureBlocking(false);
     }

     public void run(){
          while(true){
               Request request = queue.take();
               ByteBuffer command = ByteBuffer.wrap(request.getCommand());
               server.write(command);
               command.clear();

               ByteBuffer buf_header = ByteBuffer.allocate(message_size);
               int bytesRead = 0;
               do{
                    bytesRead = server.read(buf_header);
               } while(bytesRead == 0);
               [...]
     }}}
```

Socket is non-blocking.
What are the implications?

What is wrong with this code?



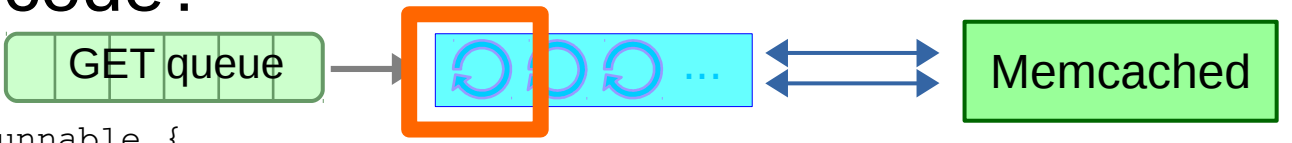

```
public class ReadHandler implements Runnable {
     [...]
     public ReadHandler(BlockingQueue<Request> queue, InetSocketAddress serverAddr){
          this.queue = queue;
          server = SocketChannel.open();
          server.connect(serverAddr);
          server.configureBlocking(false);
     }

     public void run(){
          while(true){
               Request request = queue.take();
               ByteBuffer command = ByteBuffer.wrap(request.getCommand());
               server.write(command);
               command.clear();

               ByteBuffer buf_header = ByteBuffer.allocate(message_size);
               int bytesRead = 0;
               do{
                    bytesRead = server.read(buf_header);
               } while(bytesRead == 0);
               [...]
     }}}
```

Socket is non-blocking.
What are the implications?

Hot loop!
Consumes CPU!

Read will return instantly

What is wrong with this code?




```
public class ReadHandler implements Runnable {
     [...]
     public ReadHandler(BlockingQueue<Request> queue, InetSocketAddress serverAddr){
          this.queue = queue;
          server = SocketChannel.open();
          server.connect(serverAddr);
          server.configureBlocking(false);
     }

     public void run(){
          while(true){
               Request request = queue.take();
               ByteBuffer command = ByteBuffer.wrap(request.getComm
               erver.write(command);
               command.clear();

               ByteBuffer buf_header = ByteBuffer.allocate(message_si
               int bytesRead = 0;
               do{
                    bytesRead = server.read(buf_header);
               } while(bytesRead == 0);
               [...]
     }}}
```

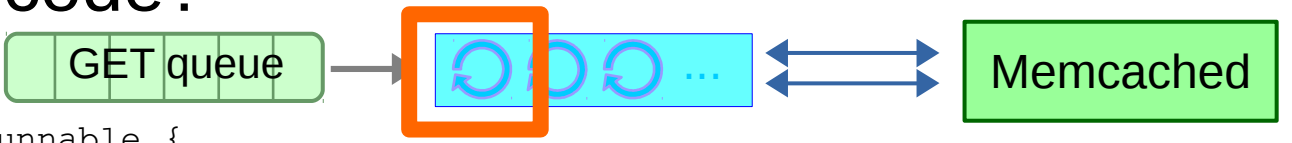
Socket is non-blocking.
What are the implications?

How about a read with timeout?

Hot loop!
Consumes CPU!

Read will return instantly

What is wrong with this code?




```
public class ReadHandler implements Runnable {
     [...]
     public ReadHandler(BlockingQueue<Request> queue, InetSocketAddress serverAddr){
          this.queue = queue;
          server = SocketChannel.open();
          server.connect(serverAddr);
          server.configureBlocking(false);
     }

     public void run(){
          while(true){
               Request request = queue.take();
               ByteBuffer command = ByteBuffer.wrap(request.getComm
               erver.write(command);
               command.clear();

               ByteBuffer buf_header = ByteBuffer.allocate(message_si
               int bytesRead = 0;
               do{
                    bytesRead = server.read(buf_header);
               } while(bytesRead == 0);
               [...]
     }}}
```

Socket is non-blocking.
What are the implications?

How about a read with timeout?
Can lead to increasing and unpredictable response time.

Hot loop!
Consumes CPU!

Read will return instantly

How much difference can this change make?

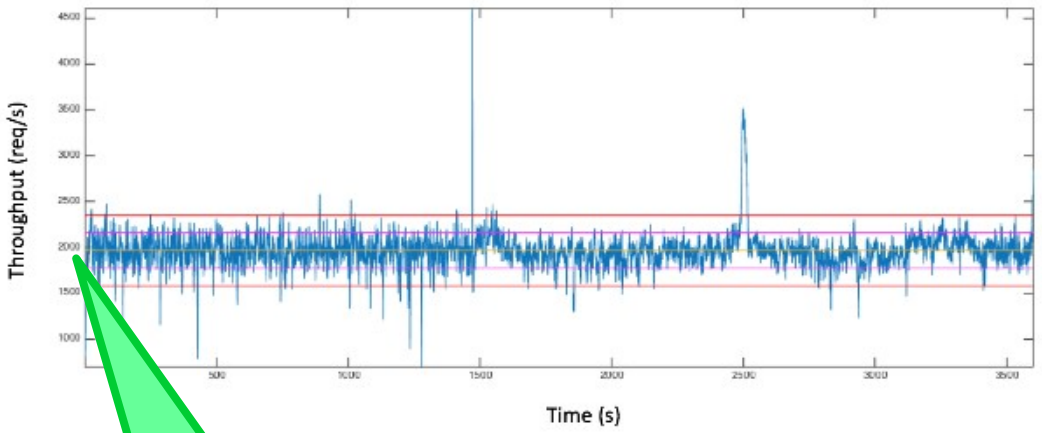


Figure 4: The ut during the one-hour experiment. The mean is 1966.6 and the standard deviation is 19

2.000 req/sec

How much difference can this change make?

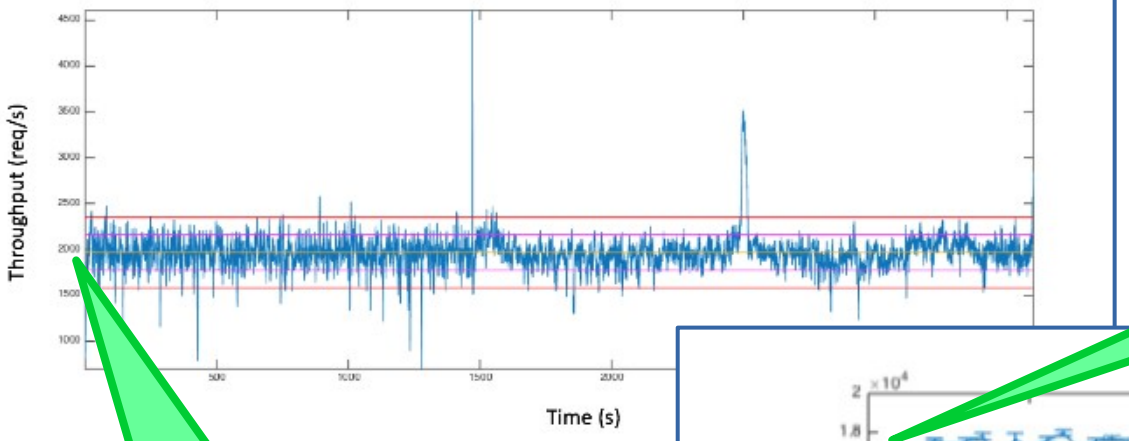


Figure 4: The ut during the one-hour experiment. T
deviation is 19

2.000 req/sec

17.000 req/sec

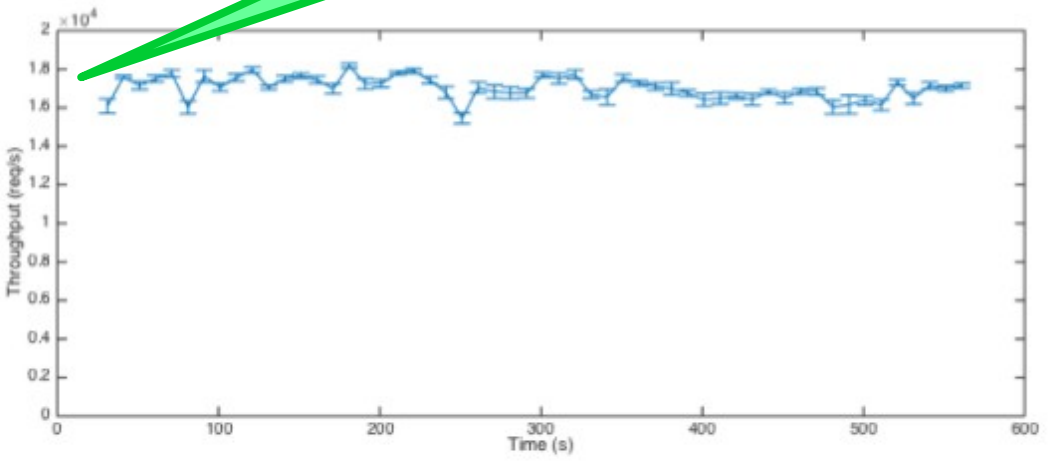
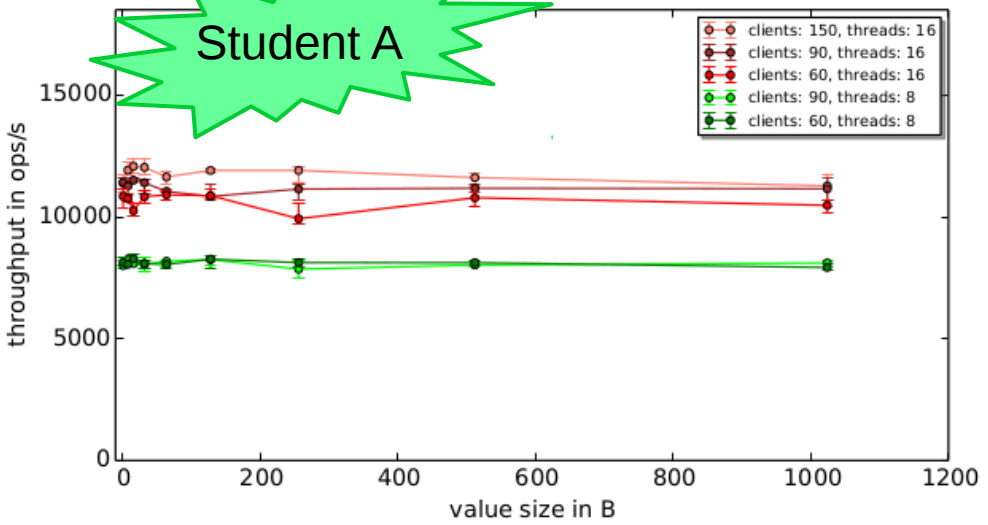


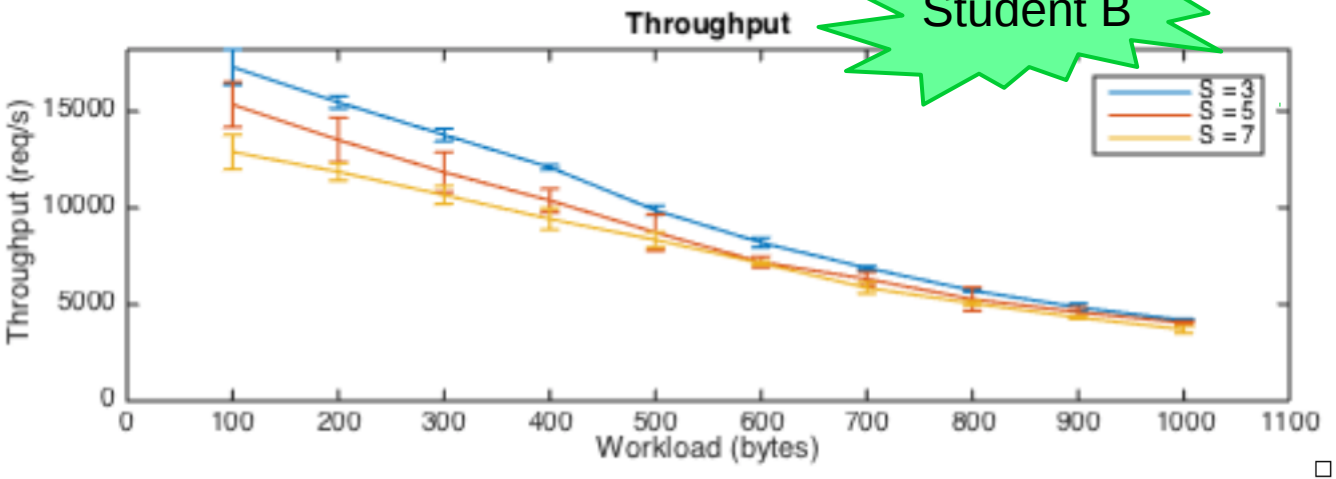
Figure 3: The throughput during the 10-minute trace. The error bars represent the 95% confidence intervals.

How about different message sizes?

Student A

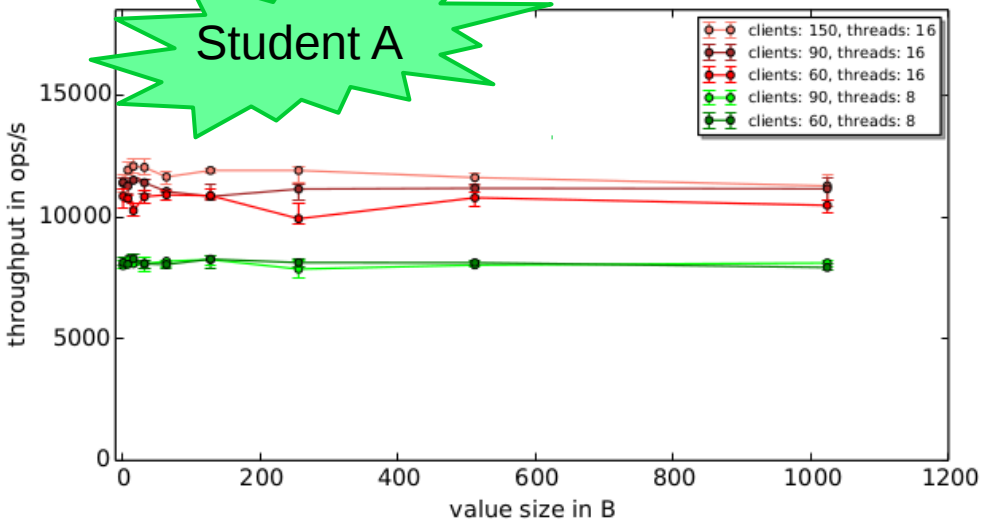


Student B



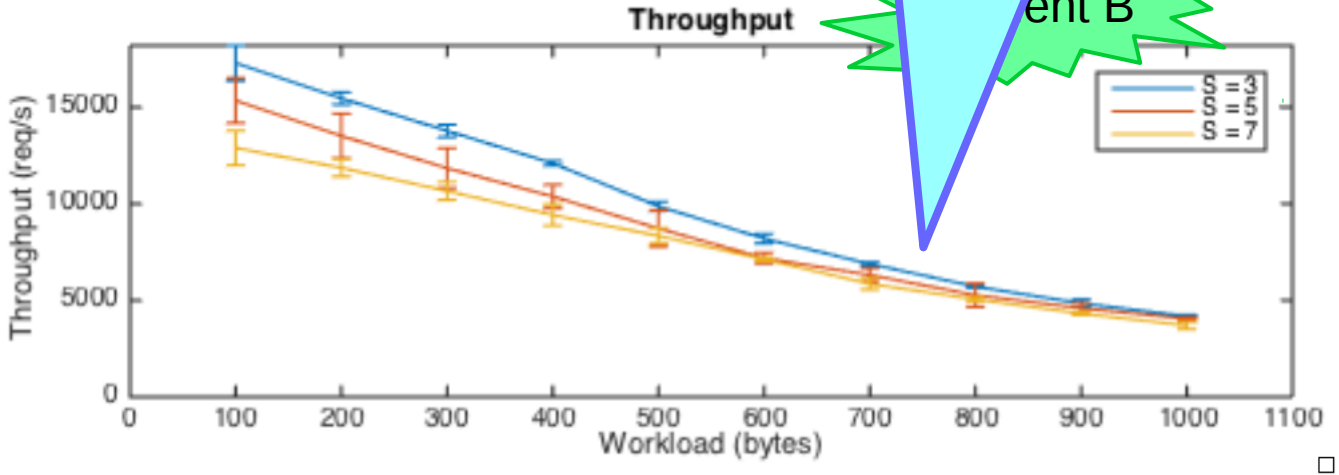
How about different message sizes?

Student A



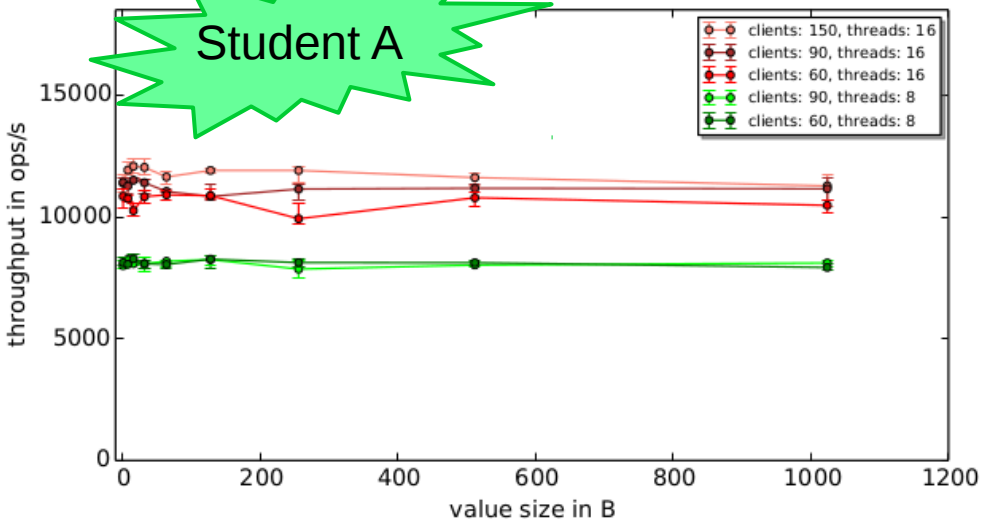
Is this network bound?

ent B



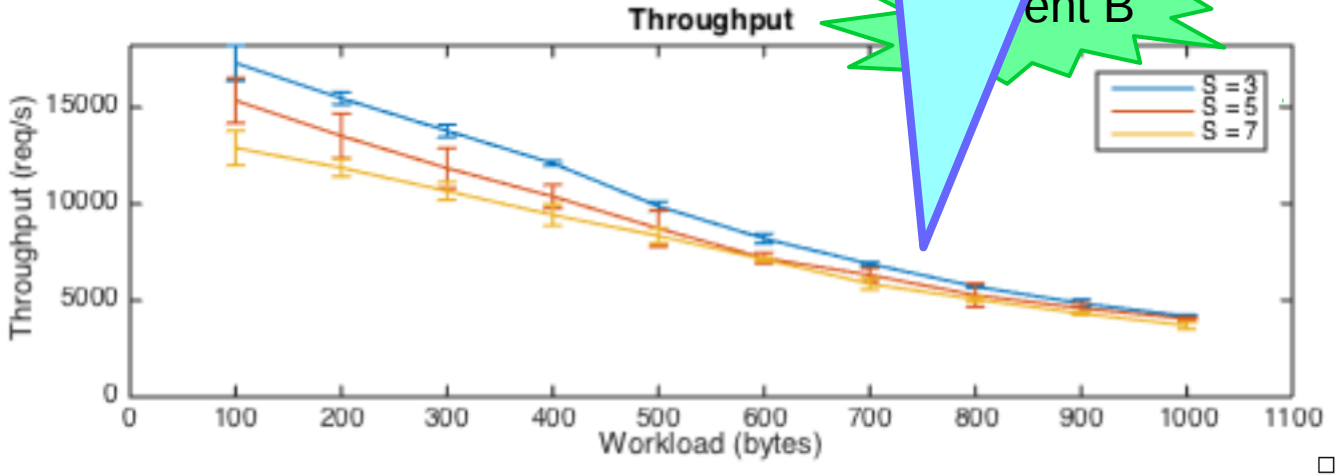
How about different message sizes?

Student A



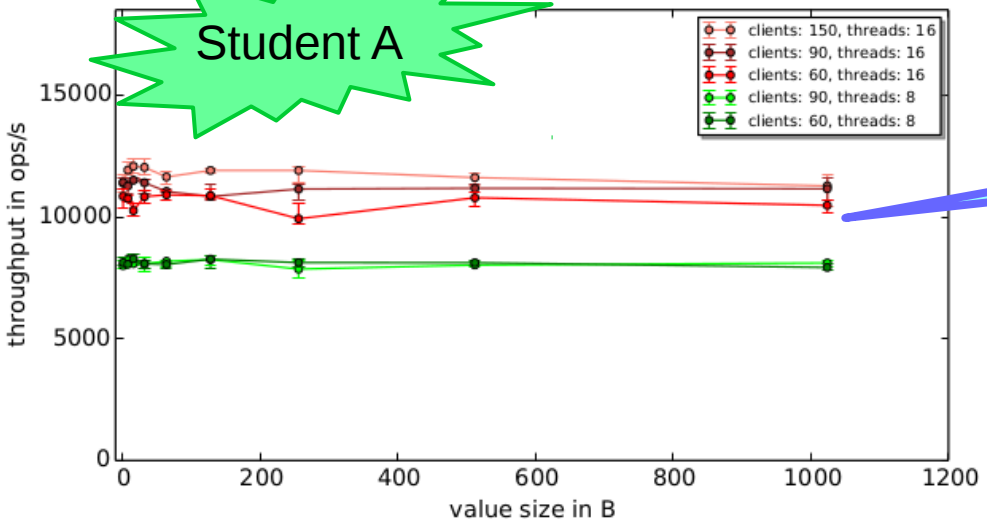
Is this network bound?
No. Sending data to 7 servers is not 2x slower than 3 servers.
Required bandwidth << network bw.

ent B



How about different message sizes?

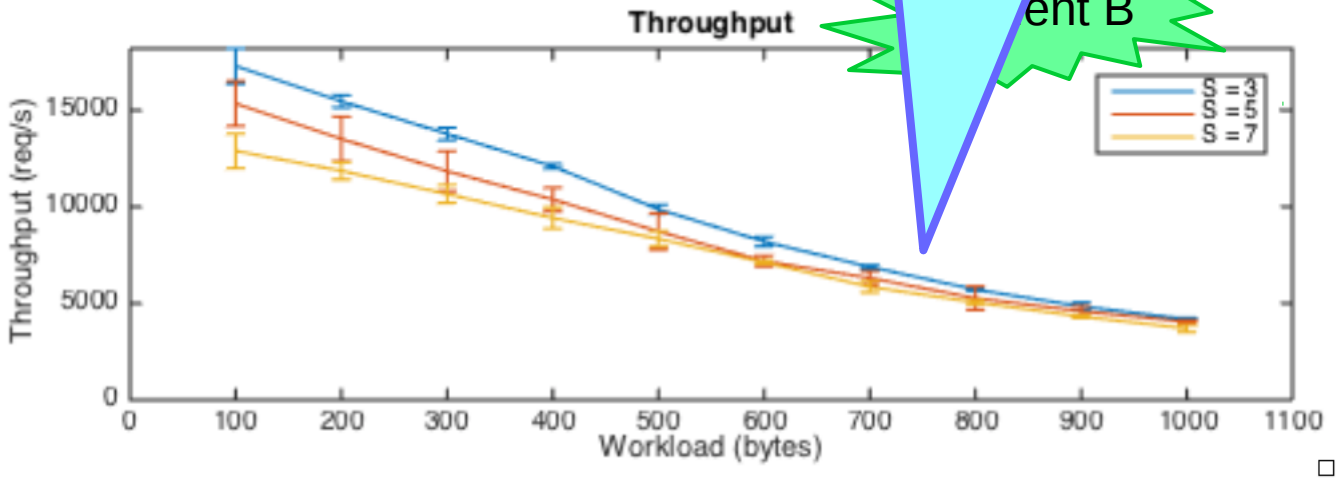
Student A



Large messages should have a higher latency.

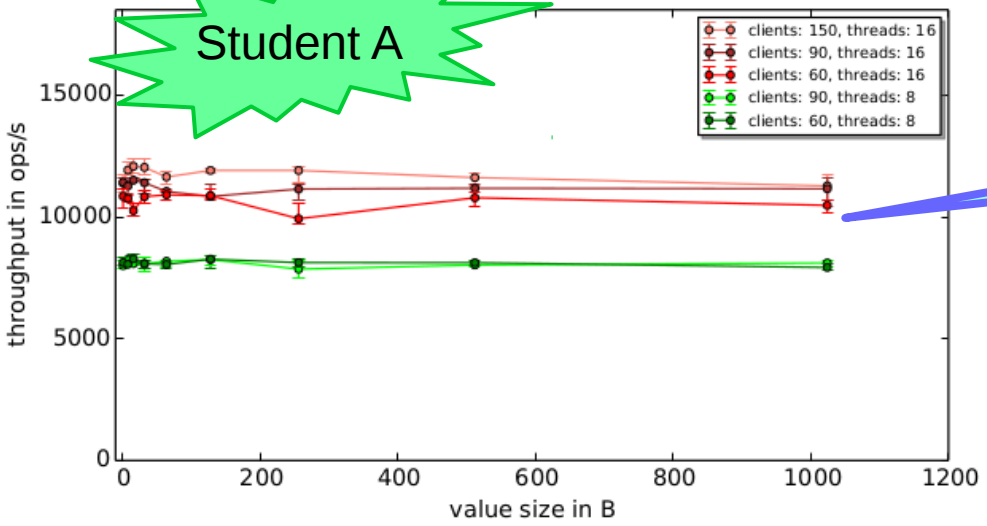
Is this network bound?
No. Sending data to 7 servers is not 2x slower than 3 servers.
Required bandwidth << network bw.

ent B



How about different message sizes?

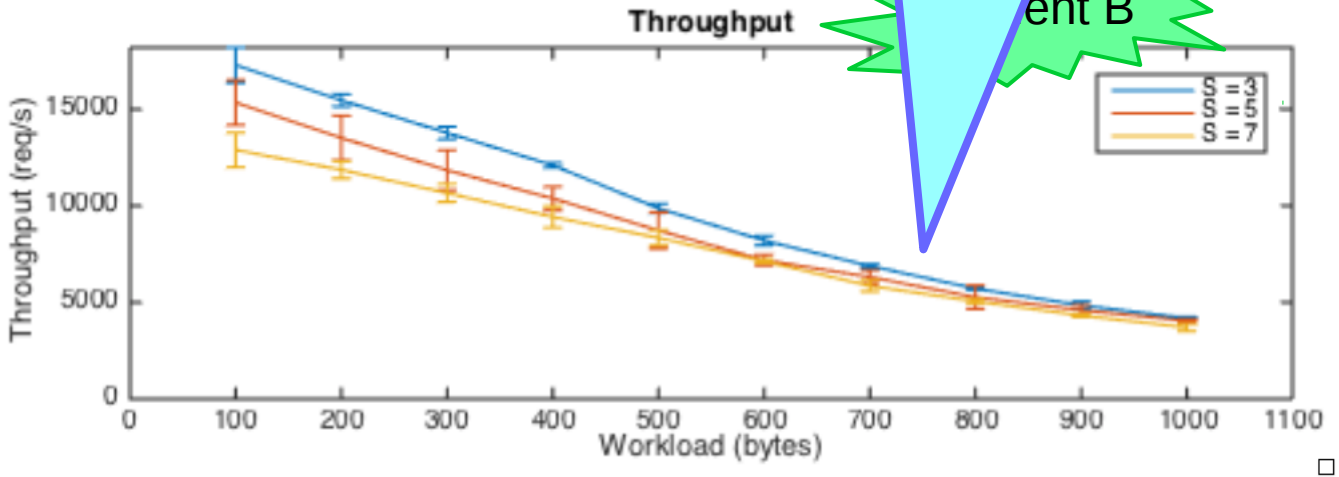
Student A



Large messages should have a higher latency.
Yes, but multiple threads will hide that latency and sustain the throughput.

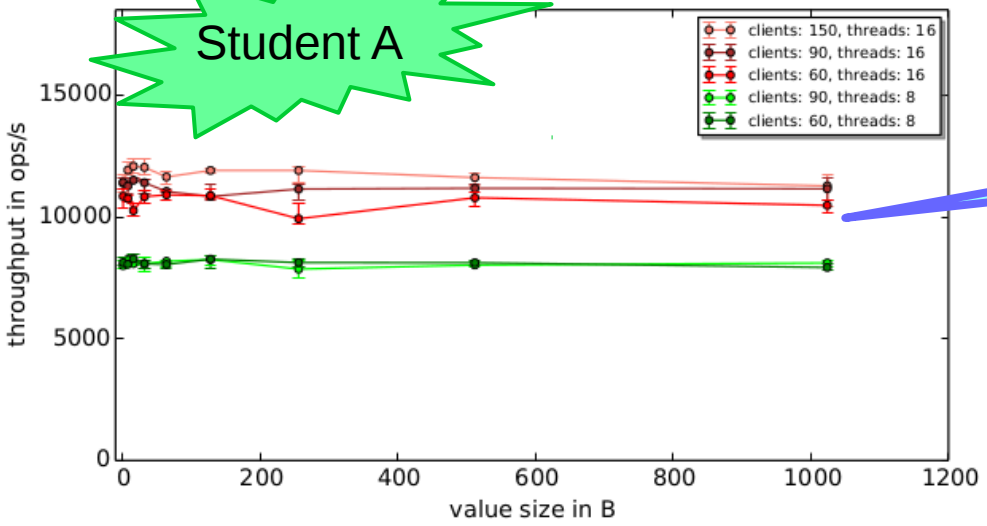
Is this network bound?
No. Sending data to 7 servers is not 2x slower than 3 servers.
Required bandwidth << network bw.

ent B



How about different message sizes?

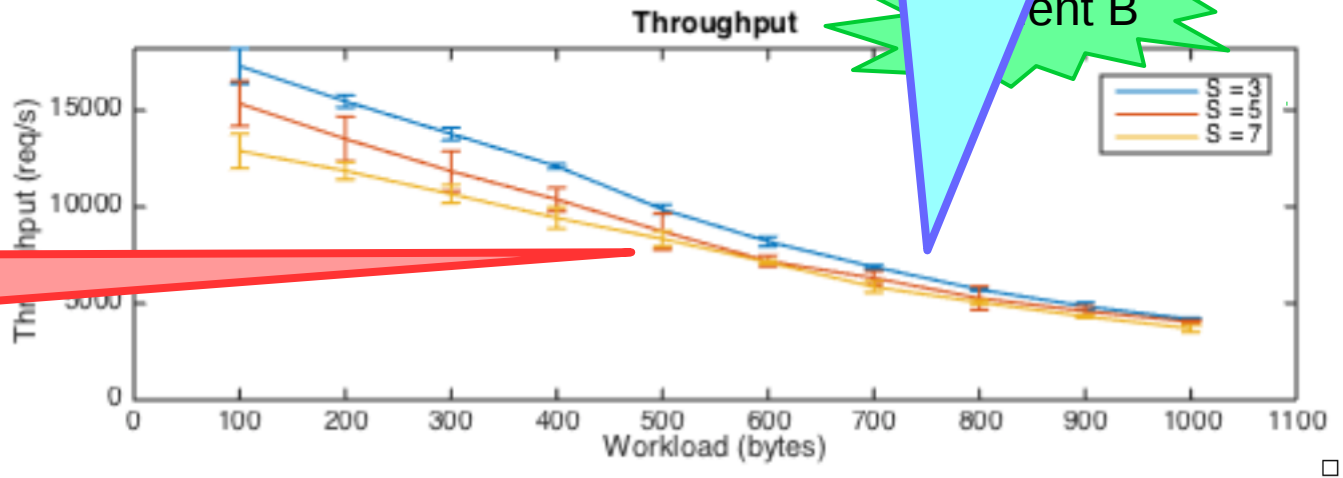
Student A



Large messages should have a higher latency.
Yes, but multiple threads will hide that latency and sustain the throughput.

Is this network bound?
No. Sending data to 7 servers is not 2x slower than 3 servers.
Required bandwidth << network bw.

ent B



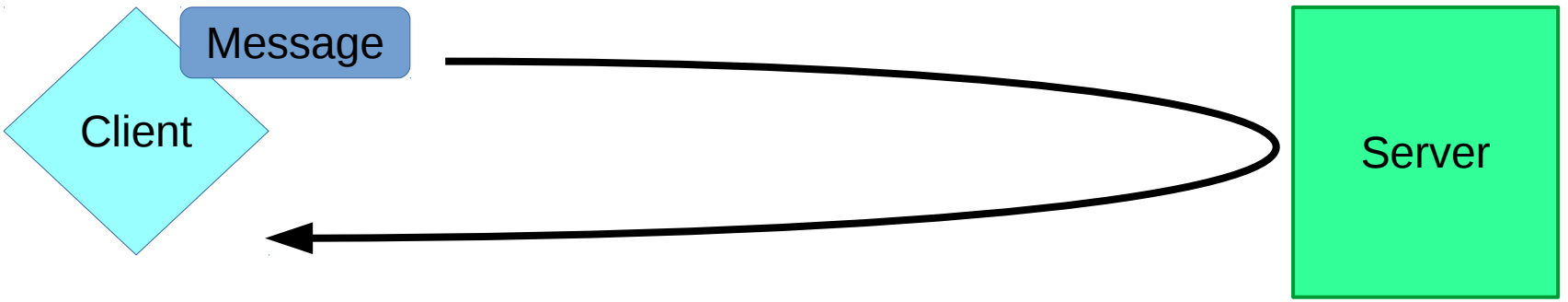
Data is copied across buffers!

```
String s1 = "request";
String s2 = new String(s1);
```

Overview

- Asynchronous networking
- Data copies
- **Connection management**
- Logging

Network Microbenchmark: Echo server



- Client connects to server and sends message
- Server reads message and sends it back
- Client reads message

What is wrong with this code?

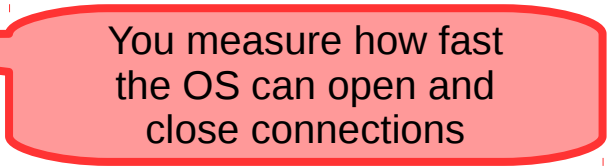
Client Code:

```
long startTime = System.nanoTime();

for (int i = 0; i < 100000; ++i) {
      Socket socket = new Socket(hostName, portNumber);
      PrintWriter out = new PrintWriter(socket.getOutputStream());
      BufferedReader in = new BufferedReader(new
                  InputStreamReader(socket.getInputStream()));

      out.println(m);
      out.flush();
      String t = in.readLine();

      out.close();
      in.close();
      socket.close();
}

long stopTime = System.nanoTime();
System.out.println("It took " + (stopTime-startTime) + "ns");
```

What is wrong with this code?

Client Code:

```
long startTime = System.nanoTime();

for (int i = 0; i < 100000; ++i) {
      Socket socket = new Socket(hostName, portNumber);
      PrintWriter out = new PrintWriter(socket.getOutputStream());
      BufferedReader in = new BufferedReader(new
                  InputStreamReader(socket.getInputStream()));

      out.println(m);
      out.flush();
      String t = in.readLine();

      out.close();
      in.close();
      socket.close();
}

long stopTime = System.nanoTime();
System.out.println("It took " + (stopTime-startTime) + "ns");
```



Keep connections open

Client Code:

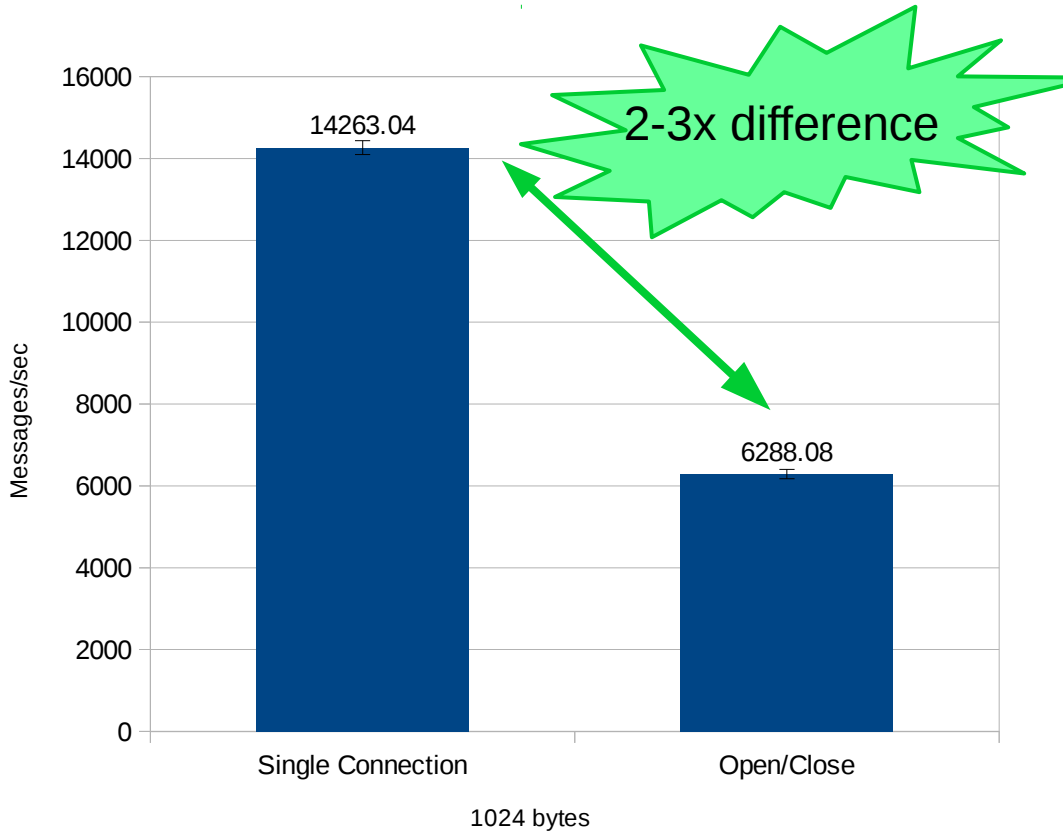
```
long startTime = System.nanoTi

for (int i = 0; i < 100000; ++i) {
      Socket socket = new Socket(hostName, portNumber);
      PrintWriter out = new PrintWriter(socket.getOutputS      ());
      BufferedReader in = new BufferedReader(new
                  InputStreamReader(socket.getInputStream()));

      out.println(m);
      out.flush();
      String t = in.readLine();

      out.close();
      in.close();
      socket.close();
}

long stopTime = System.nanoTime();
System.out.println("It took " + (stopTime-startTime) + "ns");
```

Open the connection once and reuse it

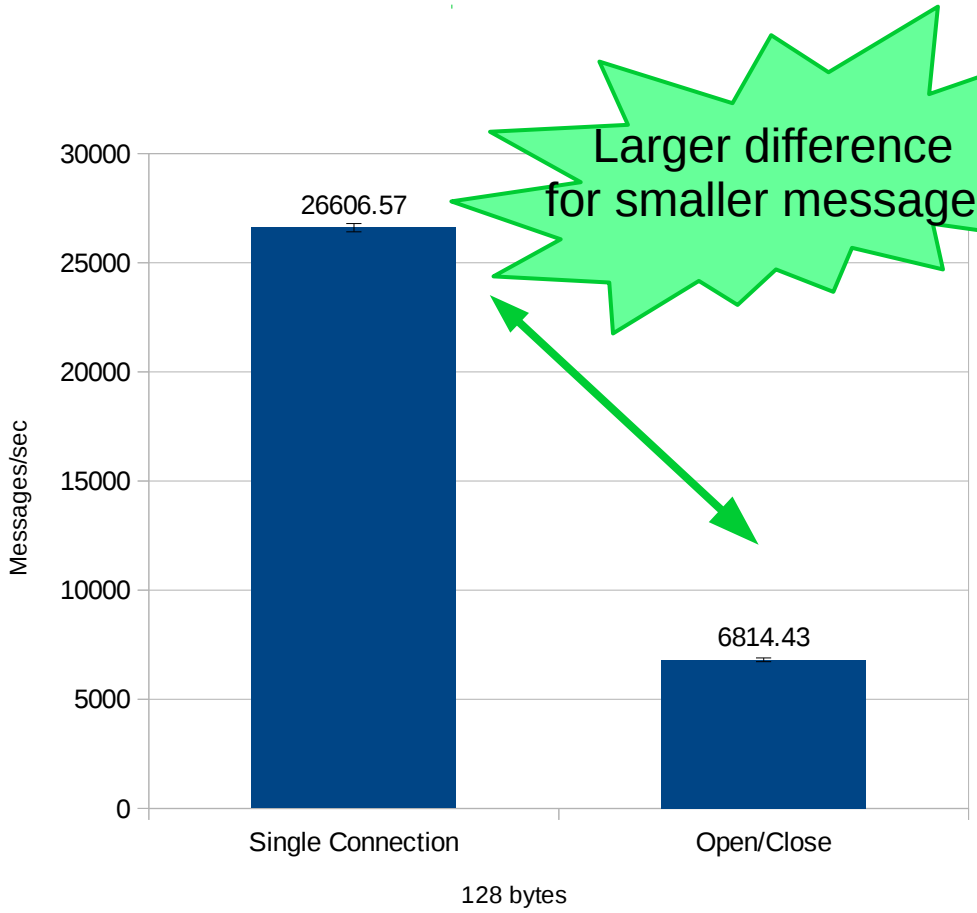
Client Code:

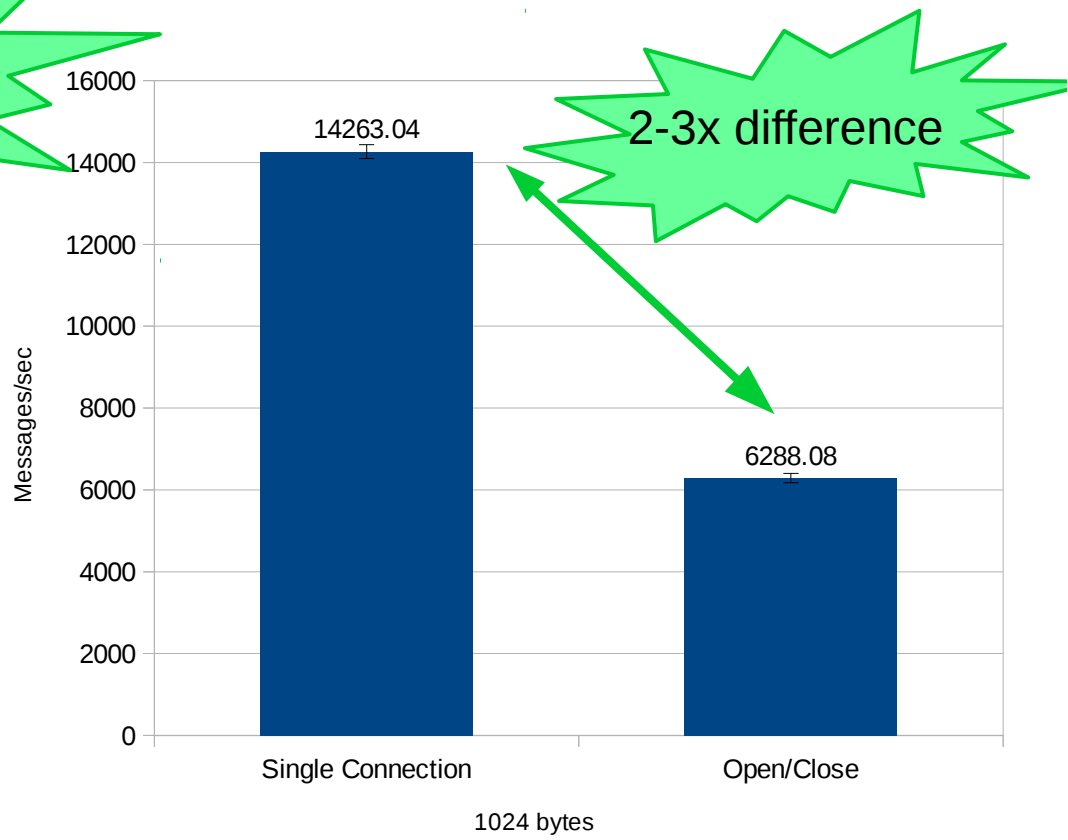
```
long startTime = System.nanoTime();

Socket socket = new Socket(hostName, portNumber);
PrintWriter out = new PrintWriter(socket.getOutputStream())
BufferedReader in = new BufferedReader(new
            InputStreamReader(socket.getInputStream()));

for (int i = 0; i < 100000; ++i) {
      out.println(m);
      out.flush();
      String t = in.readLine();
}

out.close();
in.close();
socket.close();

long stopTime = System.nanoTime();
System.out.println("It took " + (stopTime-startTime) +
"ns");
```

What is the impact?

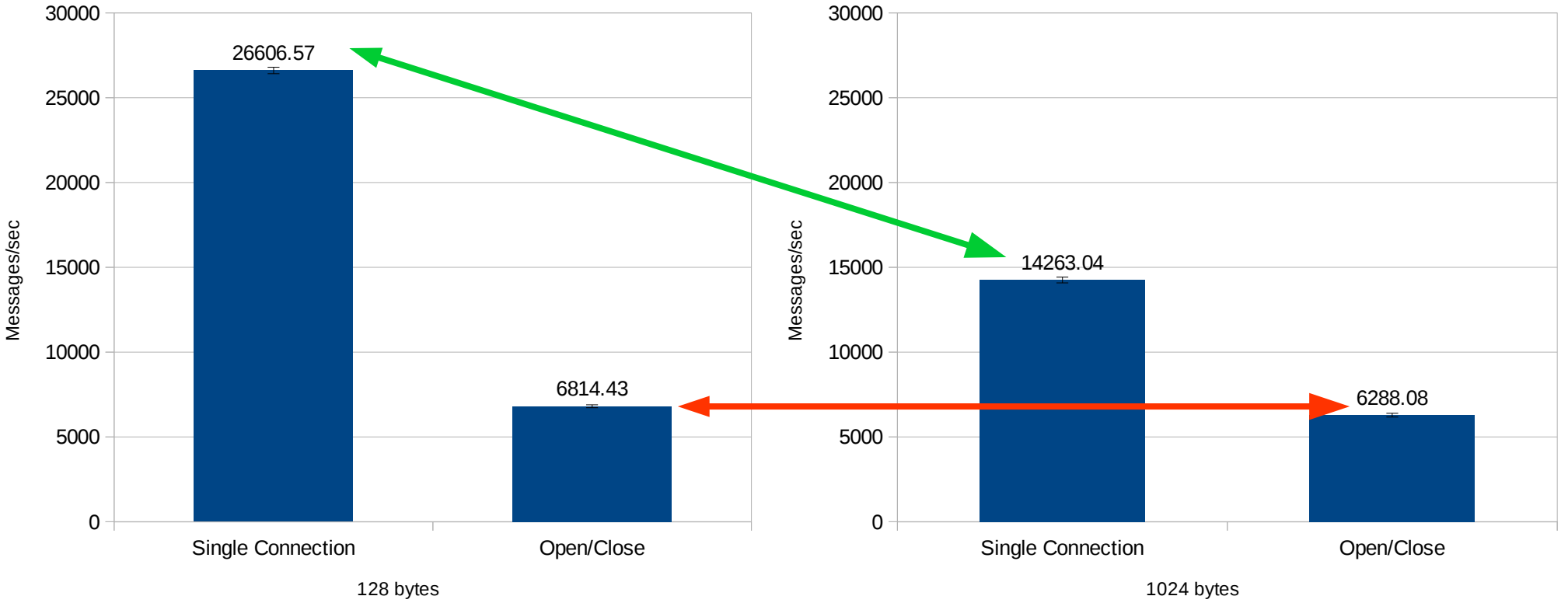


What is the impact?

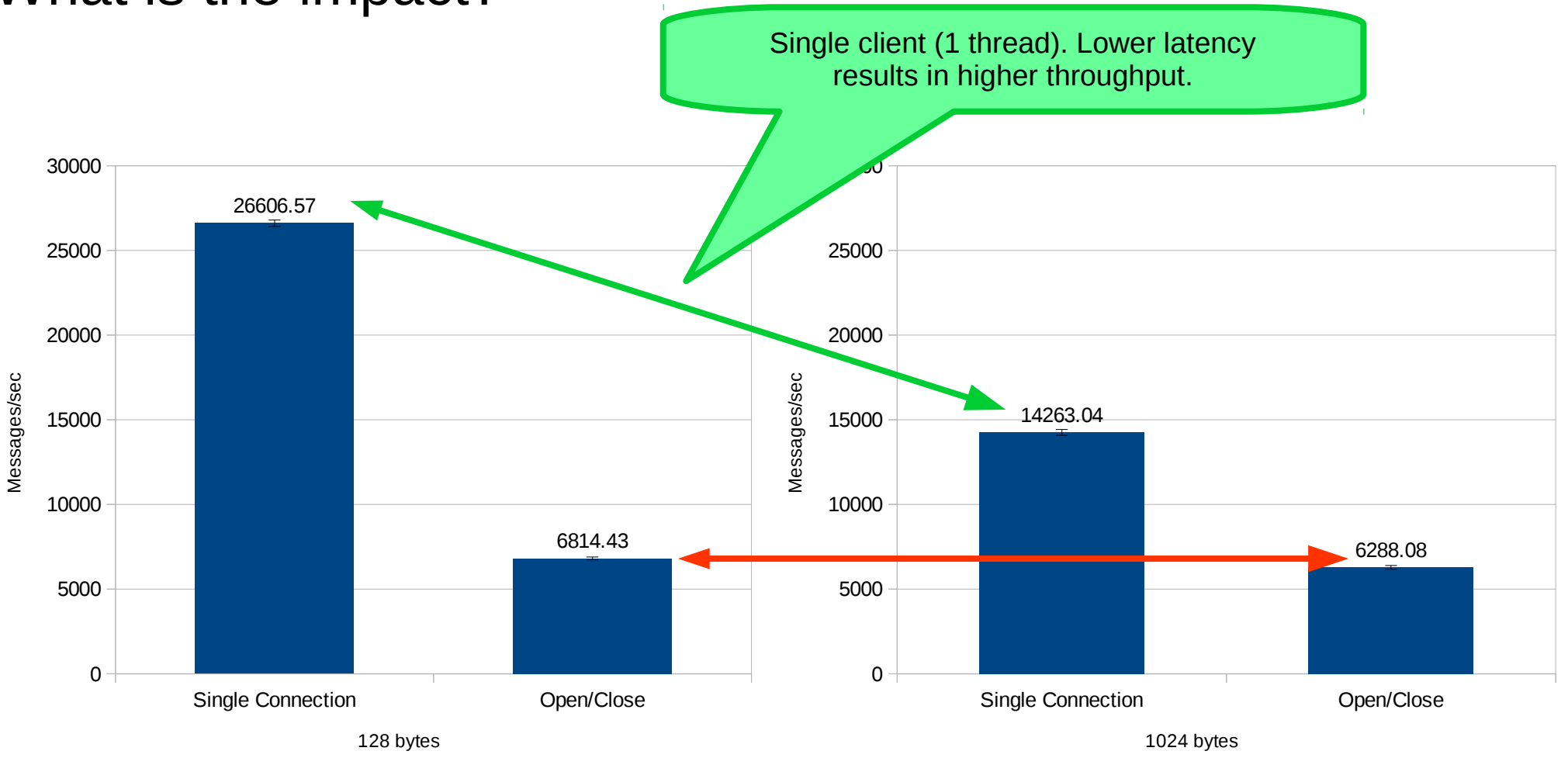




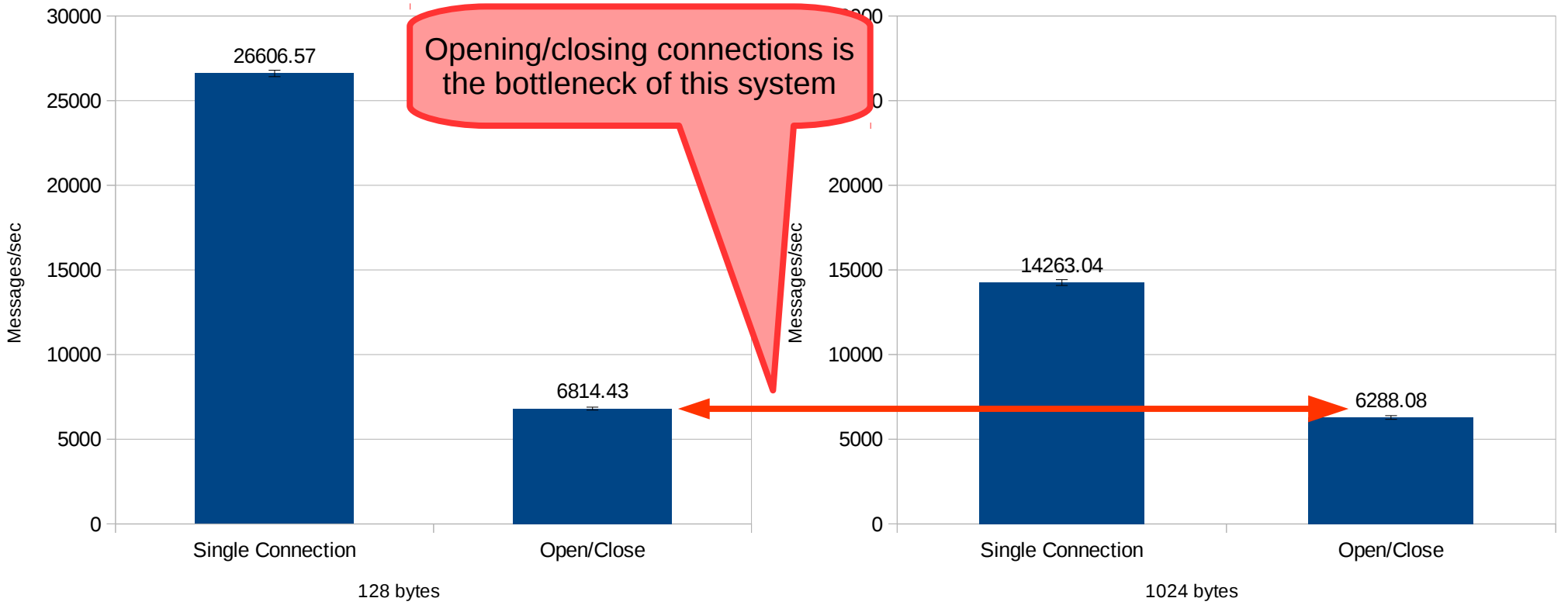
What is the impact?



What is the impact?



What is the impact?



Overview

- Asynchronous networking
- Data copies
- Connection management
- **Logging**

Logging to files vs main memory

```
...
for (int i = 0; i < 1000000; ++i) {
   int keyHash = hashFunction(keys[i]);
   logging.println("Key " + keys[i] + " hashes to "
                        + keyHash);
   logging.flush();
}
...
```

Logging to files vs main memory

```
...
for (int i = 0; i < 1000000; ++i) {
   int keyHash = hashFunction(keys[i]);
   logging.println("Key " + keys[i] + " hashes to "
                           + keyHash);
   logging.flush();
}
...
```

Avoid excessive logging/flushing in critical sections

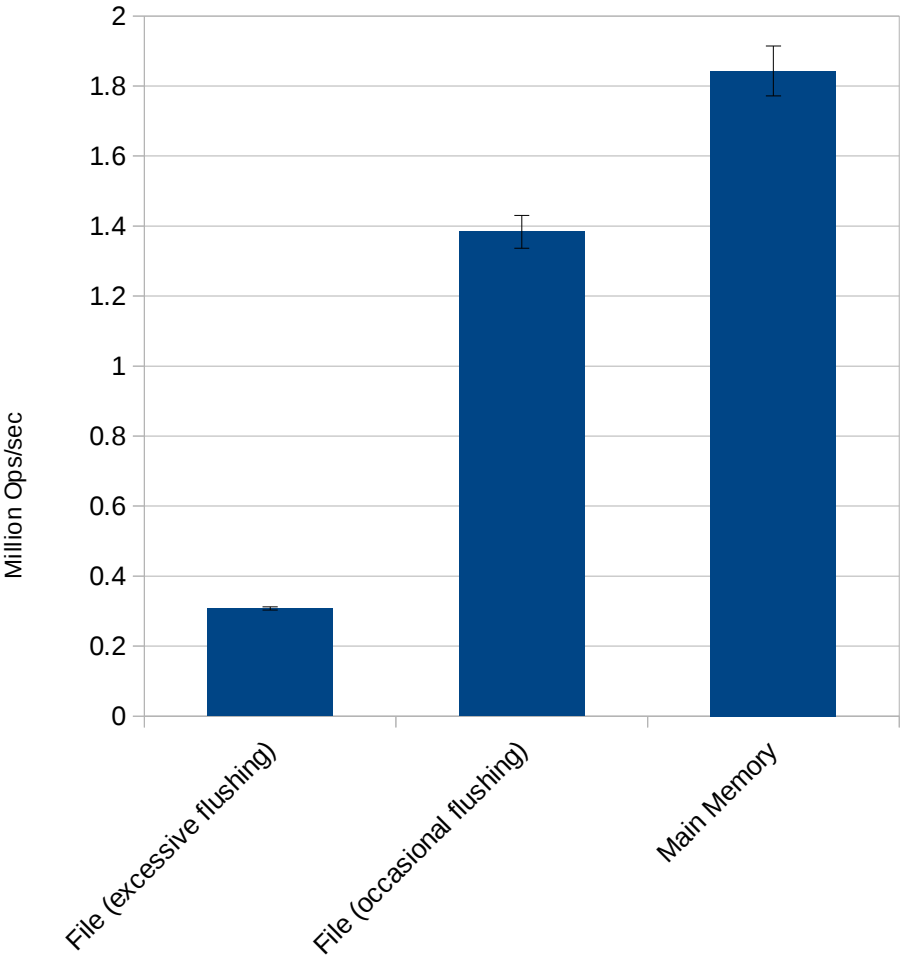
```
...
for (int i = 0; i < 1000000; ++i) {
   values[i] = hashFunction(keys[i]);
}
...
for (int i = 0; i < 1000000; ++i) {
   logging.println("Key " + keys[i] + " hashes to “
                           + values[i]);
}
logging.flush();
logging.close();
...
```

Logging to files vs main memory

```
...
for (int i = 0; i < 1000000; ++i) {
   int keyHash = hashFunction(keys[i]);
   logging.println("Key " + keys[i] + " hashes to "
                           + keyHash);
   logging.flush();
}
...
```

Avoid excessive logging/flushing in critical sections

```
...
for (int i = 0; i < 1000000; ++i) {
   values[i] = hashFunction(keys[i]);
}
...
for (int i = 0; i < 1000000; ++i) {
   logging.println("Key " + keys[i] + " hashes to “
                           + values[i]);
}
logging.flush();
logging.close();
...
```

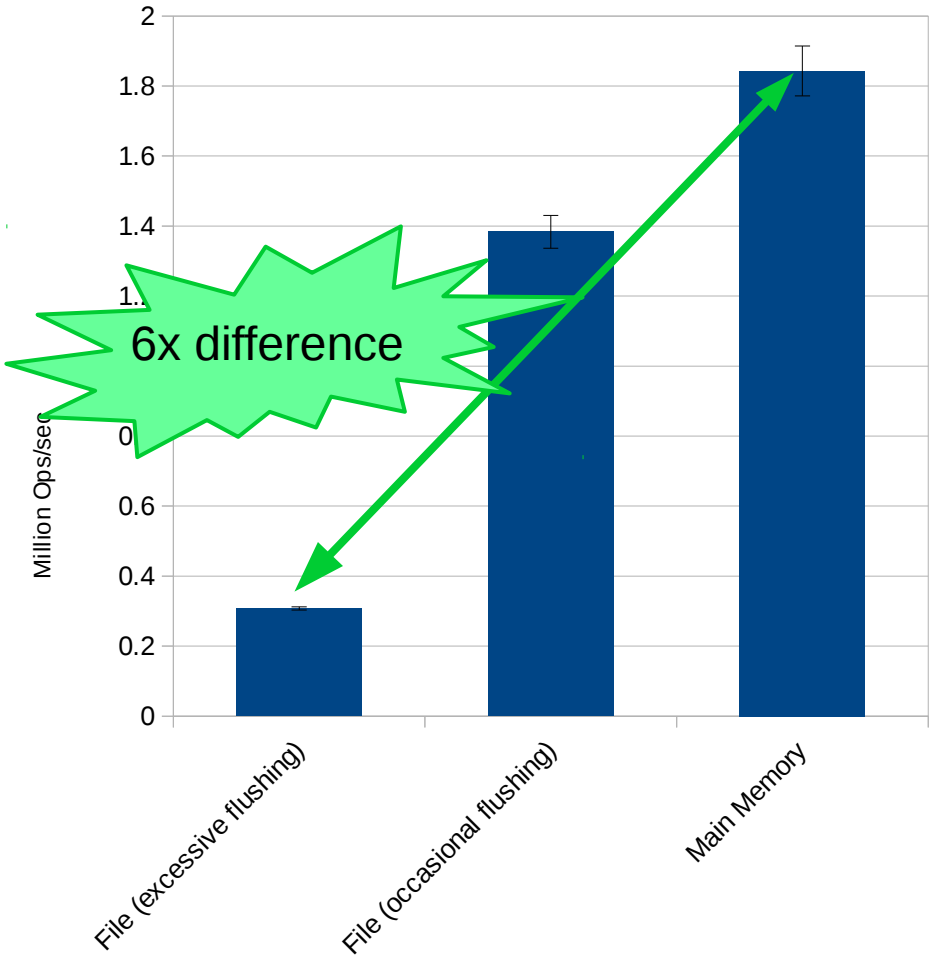


Logging to files vs main memory

```
...
for (int i = 0; i < 1000000; ++i) {
   int keyHash = hashFunction(keys[i]);
   logging.println("Key " + keys[i] + " hashes to "
                           + keyHash);
   logging.flush();
}
...
```

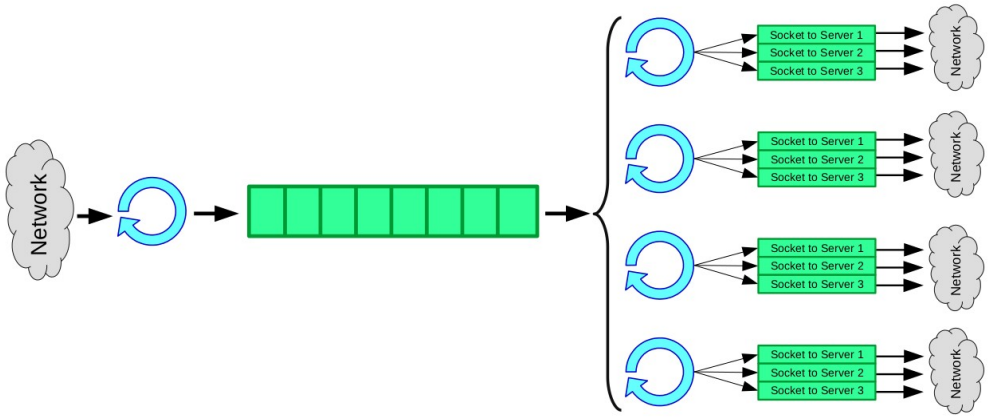
Avoid excessive logging/flushing in critical sections

```
...
for (int i = 0; i < 1000000; ++i) {
   values[i] = hashFunction(keys[i]);
}
...
for (int i = 0; i < 1000000; ++i) {
   logging.println("Key " + keys[i] + " hashes to "
                           + values[i]);
}
logging.flush();
logging.close();
...
```



Recurring questions about **threads/queues**

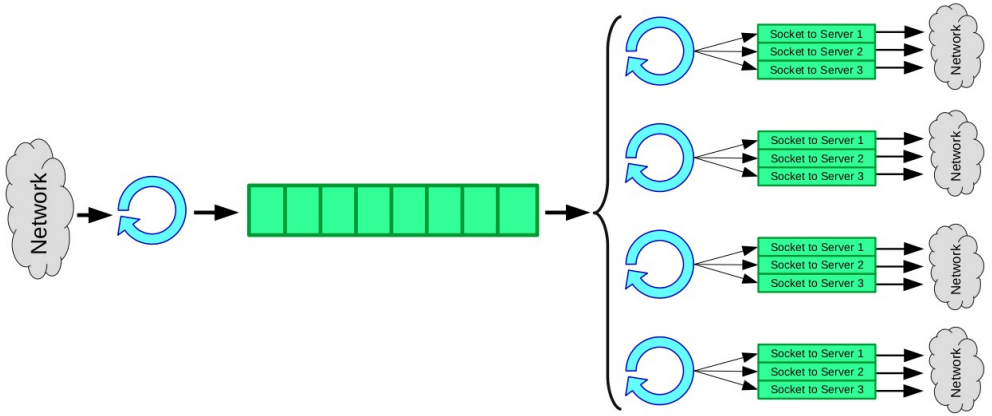
Recurring questions about **threads**



How many net threads are allowed? / Can I use helper threads? / Can I deviate from the thread design in the figure?

It is OK to use helper thread(s), but the fundamental operations (data receiving and sending, request decoding) has to happen as shown in the figure. Points of measurement should be equivalent to the given design also!

Recurring questions about **queues**



What type of objects are enqueued in the queue? / Where should I parse the requests?

Logically, each element in the queue must correspond to a single request. The exact implementation is up to you, but in the report you must explain why the implementation fulfills this condition. Work division between the different types of threads is up to you, but the measurement points must also correspond to the description.

Take-away message: Be careful with ...

- Non-blocking and asynchronous I/O
- Copying of large data buffers
- Opening and closing connections
- Flushing your data to disk in hot loops