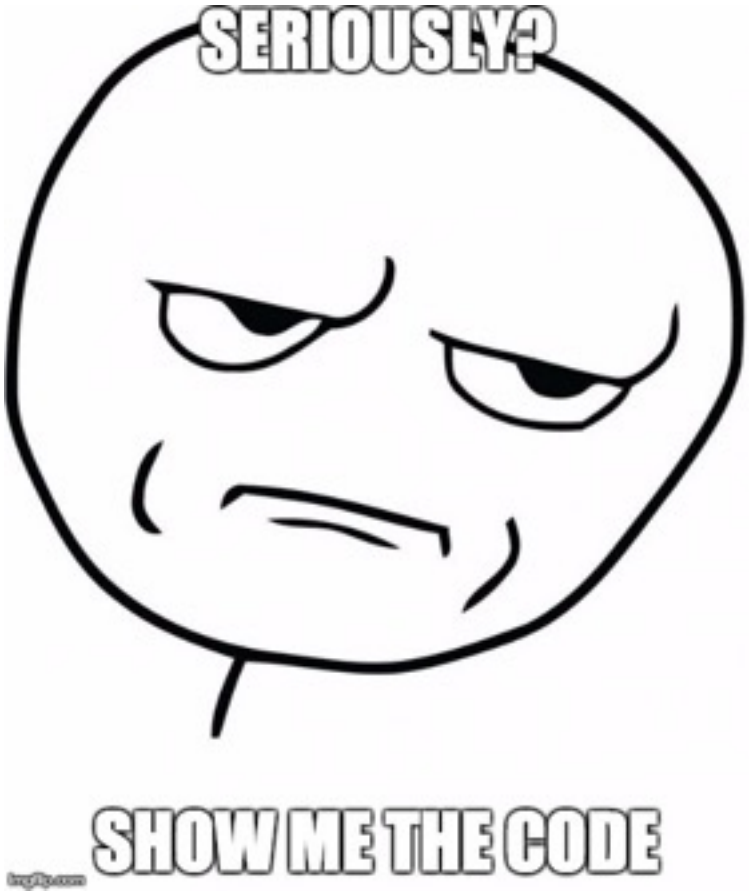
JRuby File IO using mmap

Colin Surprenant
Software Engineer
logstash/elastic

July 2015

or how to achieve **fast IO write** performance for persistence implementation

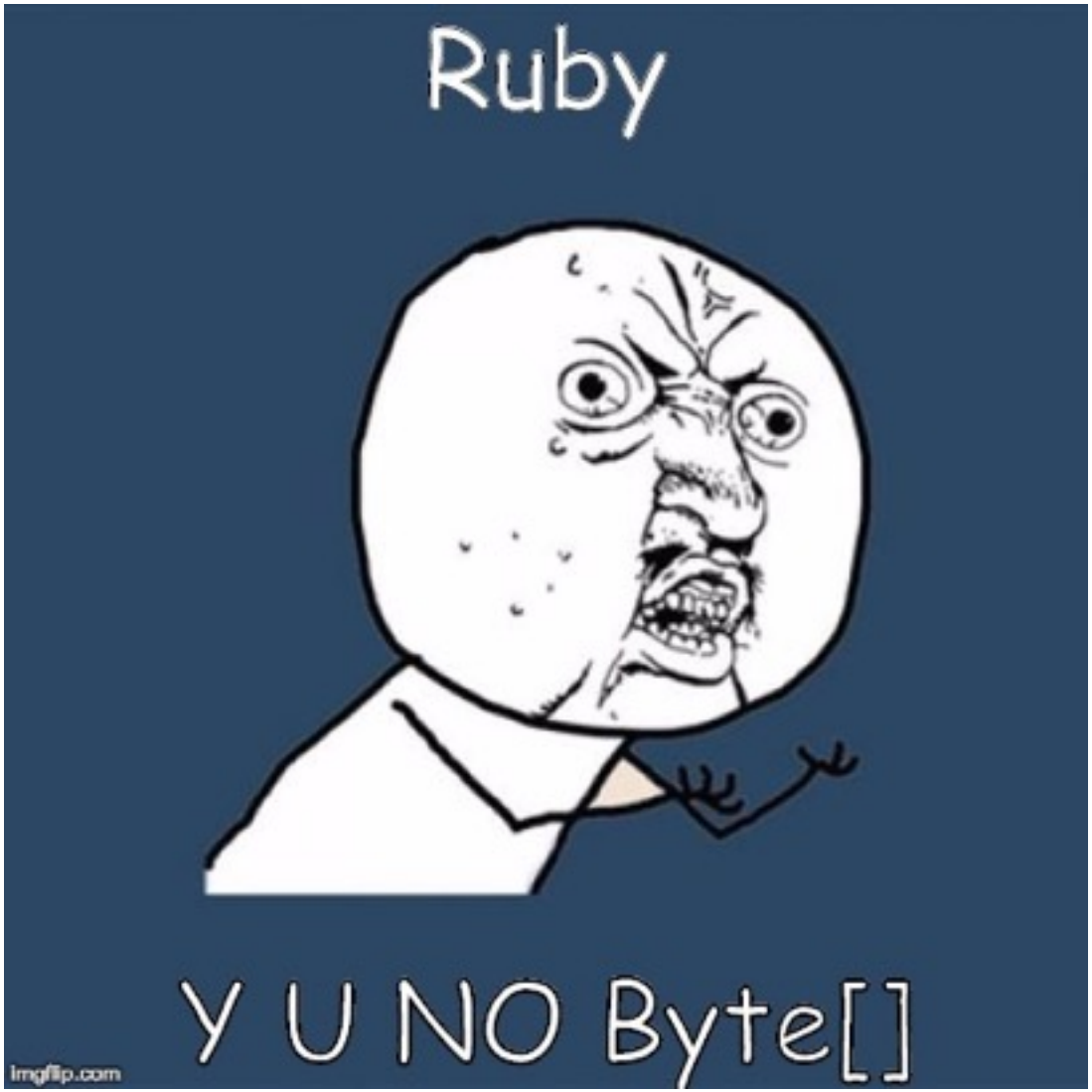
Your talk suck, just show me the code



github.com/colinsurprenant/**jrubyconf-2015**
github.com/colinsurprenant/**jruby-mmap**
github.com/colinsurprenant/**jruby-mmap-queues**

Data in Ruby

Byte[]



Data in Ruby

- No notion of **`byte[]`** in Ruby
- Use of **`String`** to hold data

- 2 important considerations
  - String charset/encoding
  - **JRuby** ↔ **Java** type conversion

https://github.com/jruby/jruby/wiki/CallingJavaFromJRuby

Ruby String Encoding

- `# encoding: utf-8`

- `Encoding::UTF_8`
- `Encoding::ASCII-8BIT`

- `String#force_encoding`
- `String#encode`
- `String#encoding`
- `String#valid_encoding?`
- `String#bytesize`

Java String Encoding

- `java.nio.charset.Charset.defaultCharset()`

- `java.nio.charset.StandardCharsets::ISO_8859_1`
- `java.nio.charset.StandardCharsets::UTF_8`

- `java -Dfile.encoding=UTF-8`
- `System.getProperty("file.encoding")`
- `ENV_JAVA[“file.encoding"]`

Ruby String Encoding

```
irb(main):001:0> s = "é"
=> "é"
irb(main):002:0> s.encoding
=> #<Encoding:UTF-8>
irb(main):003:0> s.size
=> 1
irb(main):004:0> s.bytesize
=> 2
irb(main):005:0> s.force_encoding("ASCII-8BIT")
=> "\xC3\xA9"
irb(main):006:0> s.size
=> 2
irb(main):007:0> s.bytesize
=> 2
```

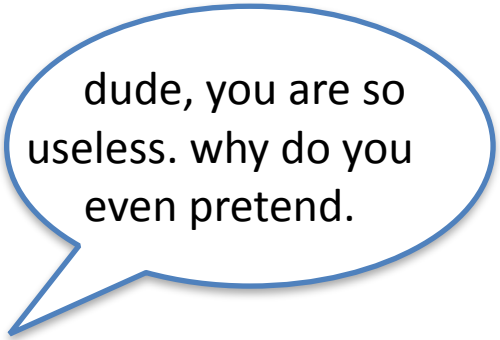
Object Persistence

dude, why are you obsessing over strings?

- All Ruby IO uses strings
- JRuby objects are not **Java** Serializable
  - you cannot benefit Java native serialization by just implementing the **`Serializable`** interface
- To persist an object, you need to serialize. Options?
  - sure, **`Marshall#dump`** ?
    - bravo genius, but note it produces a **`String`** too.
  - JSON? … ok I get it.
  - In the end you need to encode your object to feed IO so it will have to become a String at some point.

Strings

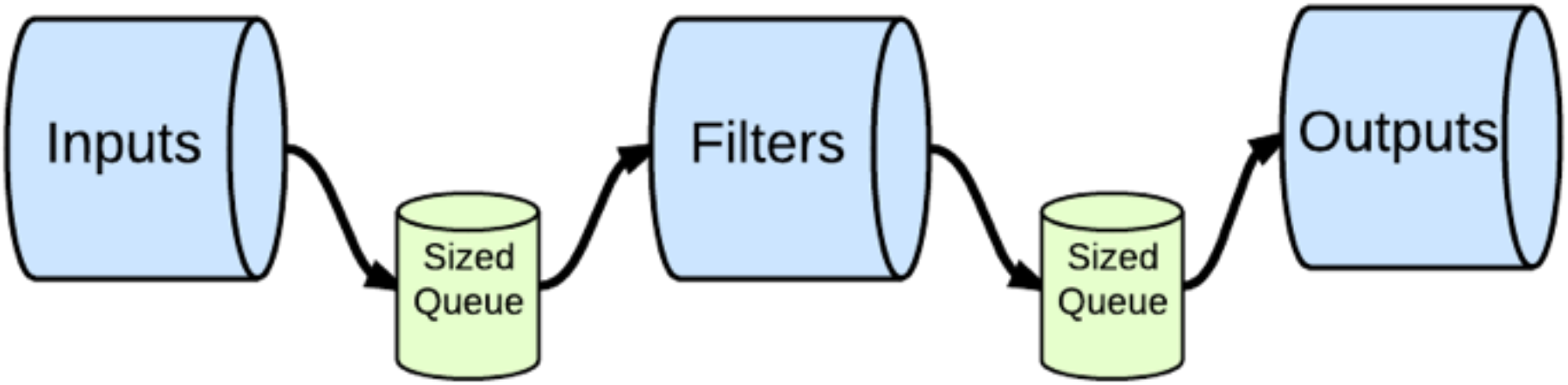
- All code and examples here just directly use strings objects.



- Object serialization strategies are not really discussed here, **that's your problem**.

- We are going to measure performance of writing String data.

Motivation - logstash pipeline



Drop-in replacement of the internal in-memory SizedQueue with a persisting implementation. Preserves back pressure mechanism.

Raw IO Performance

… or storing as many objects in less time

Strategies

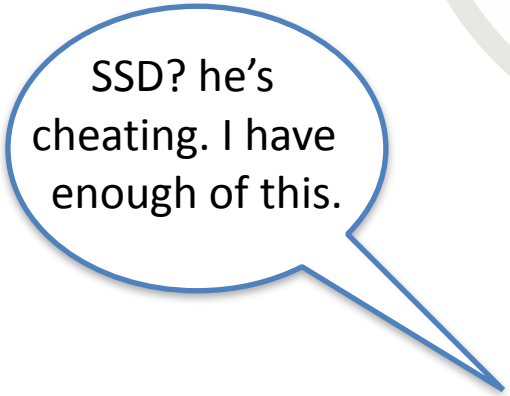
- Ruby File IO
- mmap
  - Java class
    - implicit casting, default charsets
    - implicit casting, ISO_8859_1 charset
    - explicit Ruby-side casting
    - explicit Java-side casting
  - Java JRuby extension
    - unsafe bytes (zero copy)
    - safe bytes
  - JRuby

Benchmarks

- Testing write speed
- 1/4/16k buffer sizes
- Writing N times 2GB file

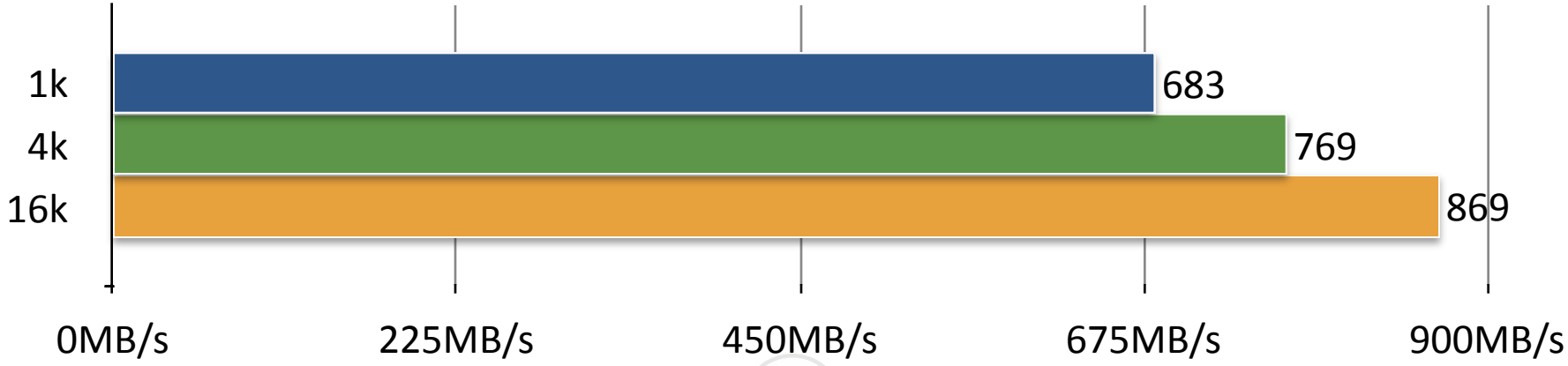
- Environment
  - MBP 13r 16GB 2.8GHz i7
  - OSX 10.10.4
  - 500GB **SSD**

  - Java 1.8.0_45
  - JRuby 1.7.21



Standard Ruby File IO

```
out = File.new(path, "w+")
…
bench("File") do |write_count, buffer|
  out.seek(0)

  write_count.times.each do
    out.write(buffer)
  end
end
```



What is mmap?

- Conventional file I/O, using read and write system calls involves copy operations between FS pages in kernel space and memory area in user space.

- mmap IO establish a virtual memory mapping from user space directly to the FS pages. With a memory-mapped file, the entire file is accessed using a ByteBuffer.

- manipulation by putting and getting bytes from an offset in the buffer. No notion of lines, EOF, …

mmap IO advantages

- User process sees file data as memory, there is **no need to issue read or write system calls**.
- If the user process accesses the mapped memory space, **page faults will bring in the file data from disk**.
- If the user modifies the mapped memory space, the affected page is marked as dirty and will be flushed to disk.
- The OS VM performs pages caching, managing memory according to system load.
- The data is always page-aligned, **no buffer copying is needed**.
- Very large files can be mapped without consuming large amounts of memory.

mmap

- user process crash: mmap file intact.
- pull the plug: all bets are off.
- `MappedByteBuffer.force()` → **`flush+fsync`**
  - **not calling** force/flush/fsync here.
- mmap performance relative to **FS type**, **free memory** for FS **cache** and read/write block size.

- mmap should be **much faster** than stream IO.

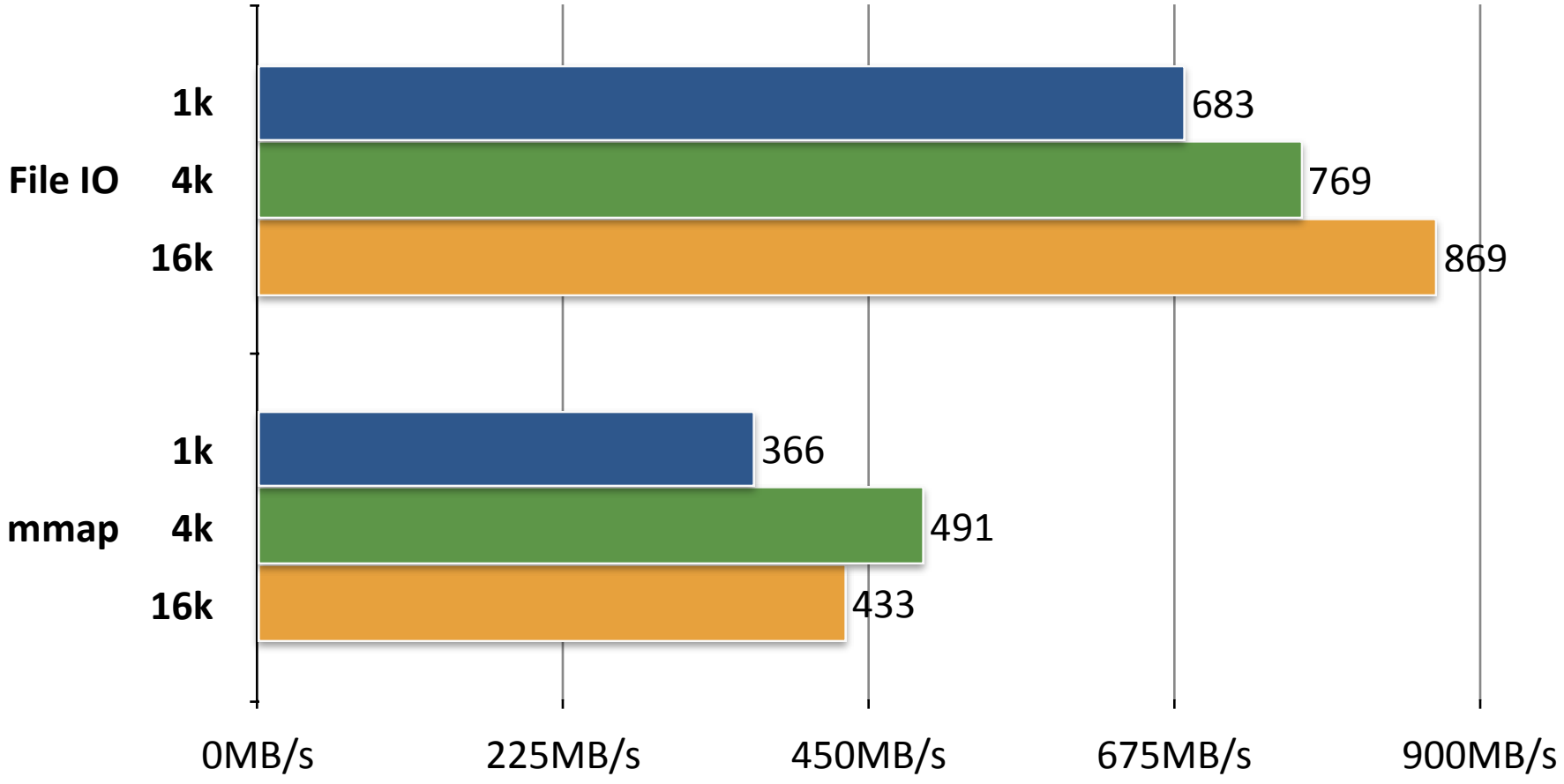
Simple Java mmap class

mmap - Java class, implicit casting, default charsets

```
bench("mmap") do |write_count, buffer|
  out.position(0)

  write_count.times.each do
    out.put_bytes(buffer)
  end
end
```

```
public void put_bytes(String data) {
  byte[] bytes = data.getBytes();
  this.buffer.put(bytes, 0, bytes.length);
}
```

mmap - Java class, implicit casting, default charsets



mmap - Java class, implicit casting, default charsets

WTF?

Are we transcoding? 😱

```
("abcdefg\n" * size).force_encoding(Encoding::ASCII_8BIT)
```

```
irb(main):002:0> java.nio.charset.Charset.defaultCharset
=> #<Java::SunNioCs::UTF_8:0x56aac163>
```

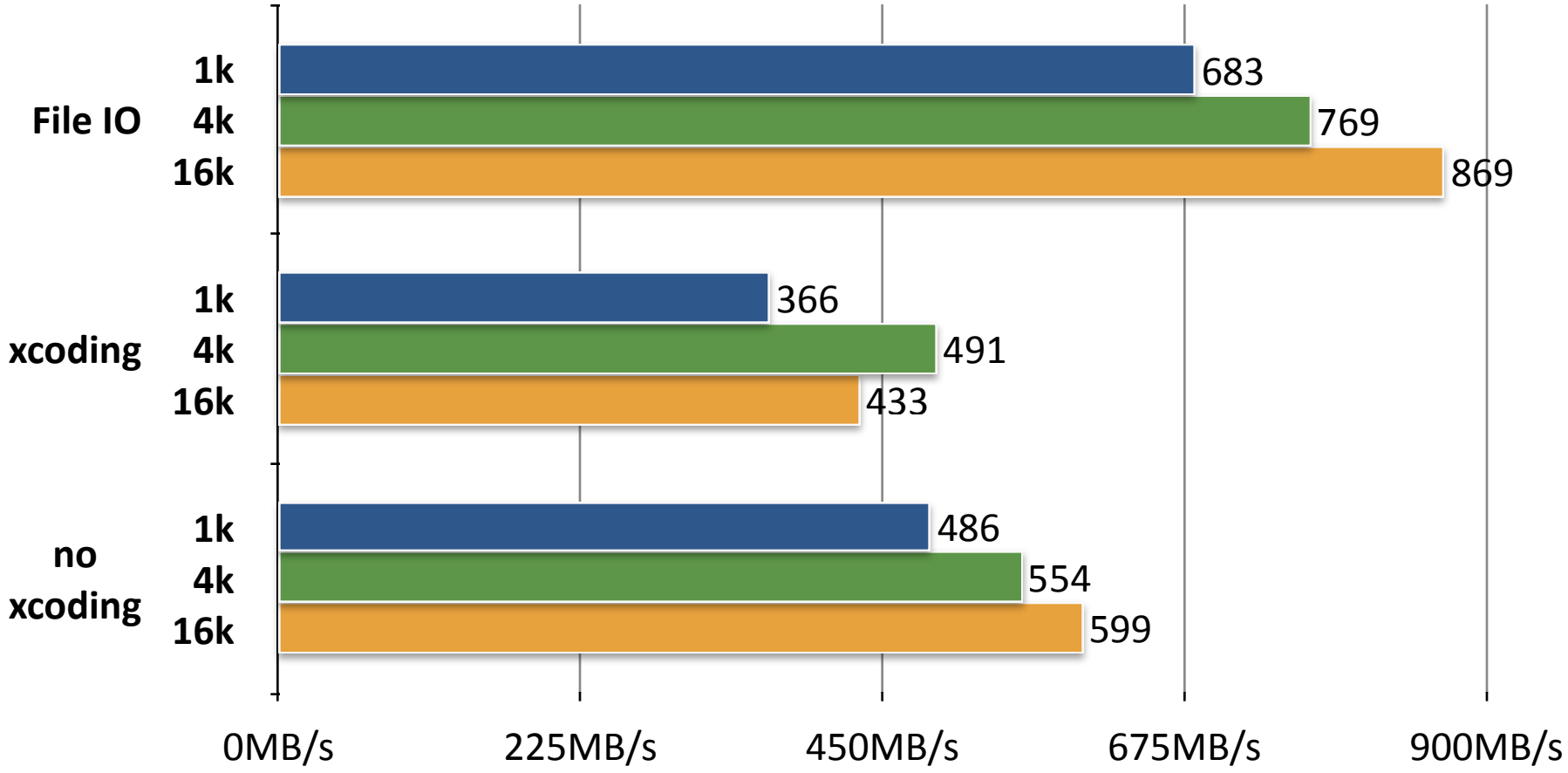
```
public final class String
  …
  public byte[] getBytes()
  public byte[] getBytes(Charset charset)
```

mmap - Java class, implicit casting, ISO_8859_1 charset

```
bench("mmap") do |write_count, buffer|
  out.position(0)

  write_count.times.each do
    out.put_bytes(buffer, StandardCharsets::ISO_8859_1)
  end
end
```

```
public void put_bytes(String data, Charset charset)
{
  byte[] bytes = data.getBytes(charset);
  this.buffer.put(bytes, 0, bytes.length);
}
```

mmap - Java class, implicit casting, ISO_8859_1 charset



mmap - Java class, implicit casting, ISO_8859_1 charset

WTF^2?

Type conversion?

```
write_count.times.each do
  out.put_bytes(buffer, StandardCharsets::ISO_8859_1)
end
```

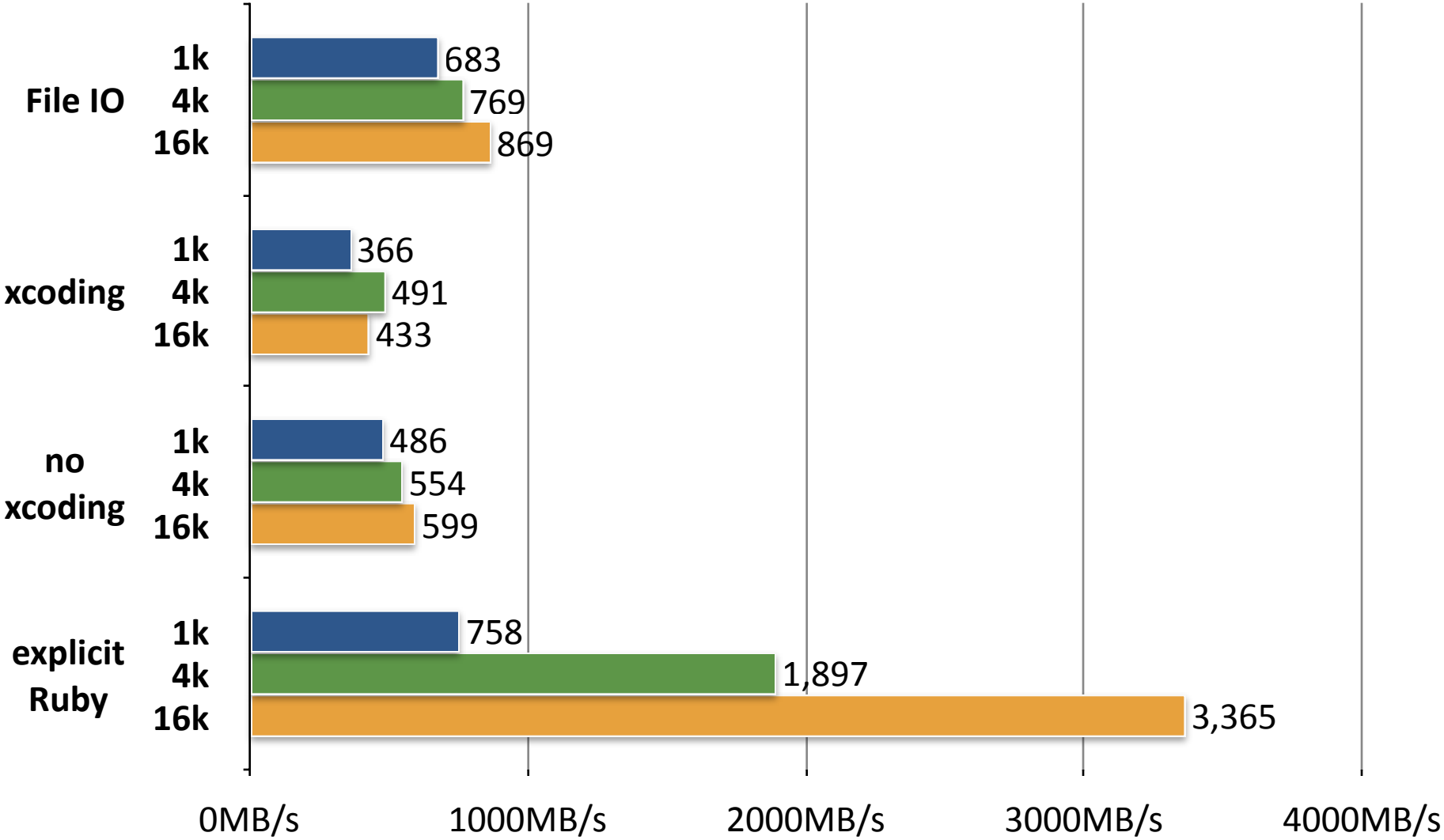
```
public void put_bytes(String data, Charset charset)
{
  …
}
```

mmap - Java class, explicit Ruby-side casting

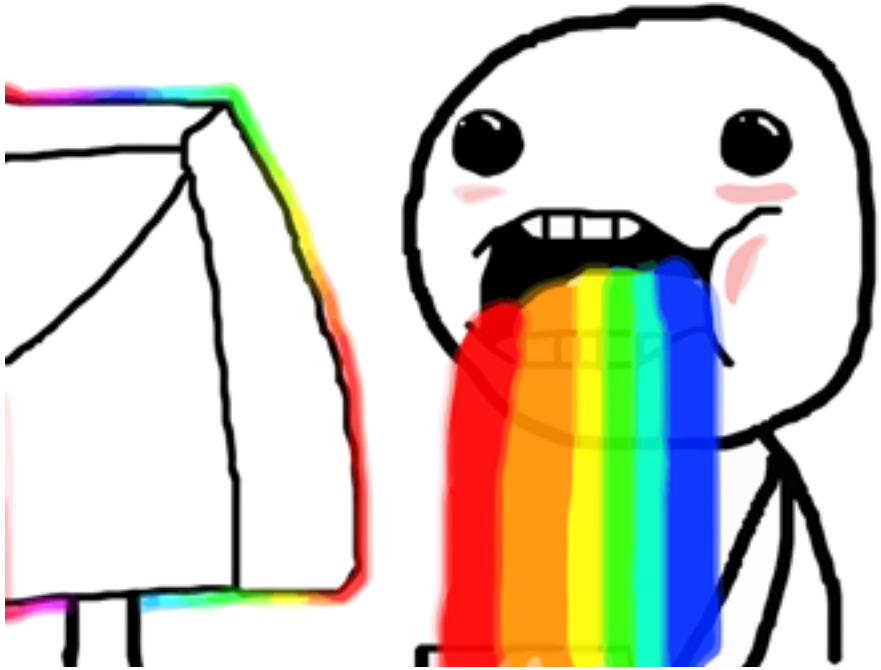
```ruby
bench("mmap") do |write_count, buffer|
  out.position(0)

  write_count.times.each do
    out.put_bytes(buffer.to_java_bytes)
  end
end
```

```java
  public void put_bytes(byte[] data) {
    this.buffer.put(data, 0, data.length);
  }
```

mmap - Java class, explicit Ruby-side casting



mmap - Java class, explicit Ruby-side casting



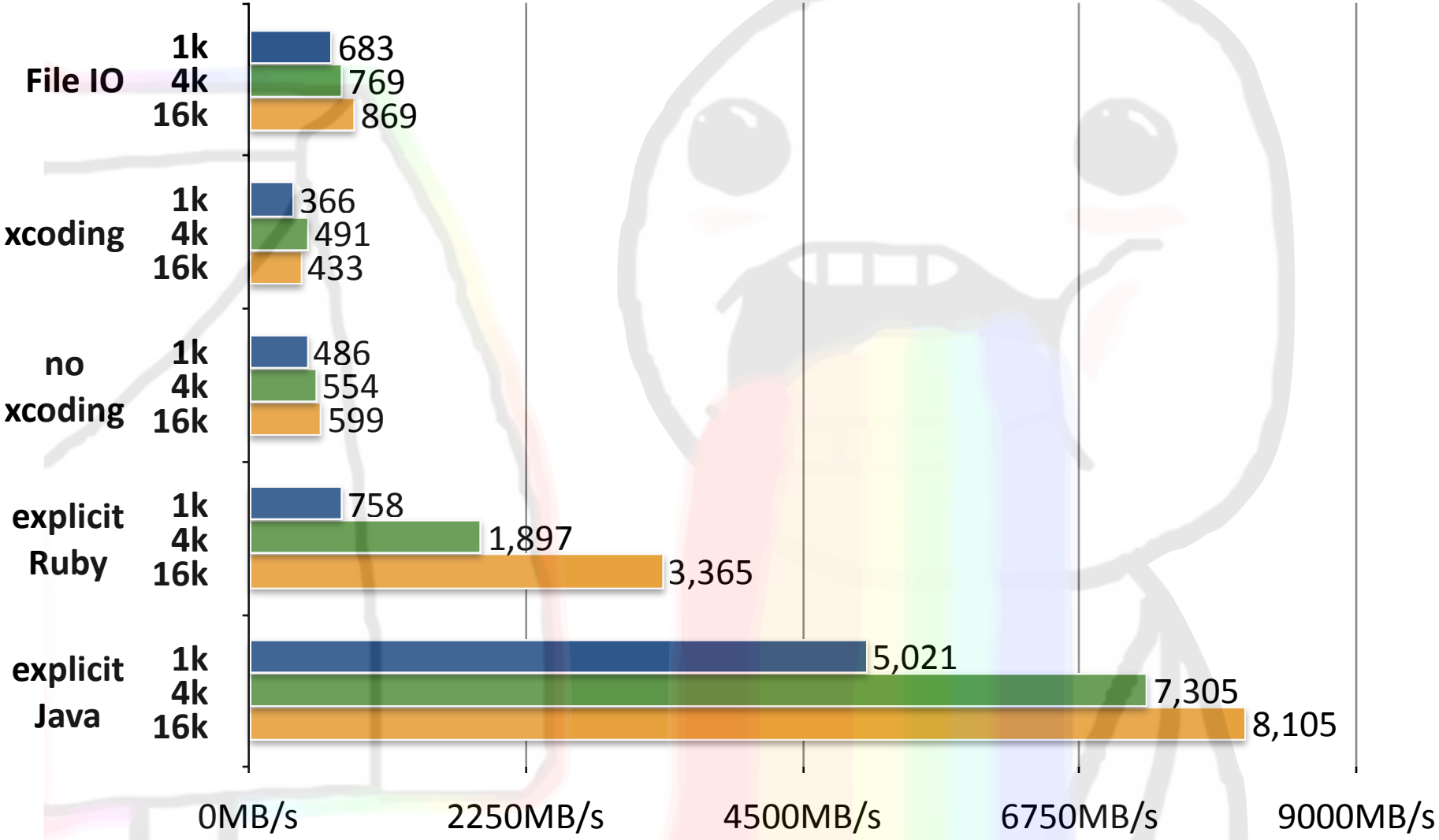
Can we do better?

mmap - Java class, explicit Java-side casting

```
bench("mmap") do |write_count, buffer|
  out.position(0)

  write_count.times.each do
    out.put_ruby_string(buffer)
  end
end
```

```
public void put_ruby_string(RubyString data) {
  ByteList byteList = data.getByteList();
  this.buffer.put(byteList.unsafeBytes(), 0, byteList.length());
}
```

mmap - Java class, explicit Java-side casting



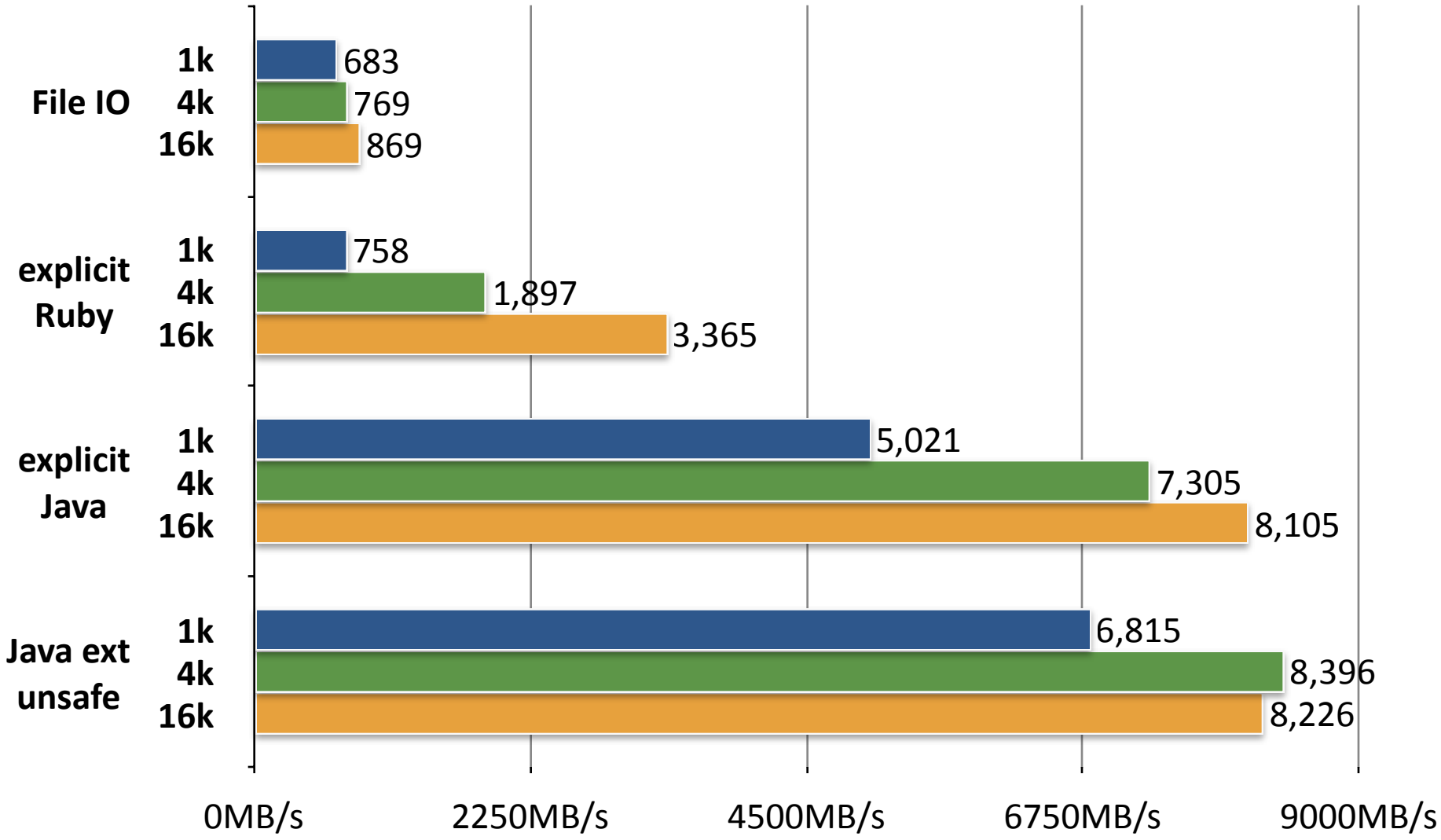
Java JRuby extension

mmap - Java JRuby extension, unsafe bytes

```
bench("mmap", 16) do |write_count, buffer|
  out.position = 0

  write_count.times.each do
    out.put_bytes(buffer)
  end
end
```

```
@JRubyMethod(name = "put_bytes", required = 1, optional = 2)
public void put_bytes(ThreadContext context, IRubyObject[] args) {
    ByteList byteList = ((RubyString)args[0]).getByteList();
    if (args.length == 3) {
        int offset = RubyNumeric.num2int(args[1]);
        int length = RubyNumeric.num2int(args[2]);
        this.buffer.put(((byteList.unsafeBytes(), offset, length);
    } else if (args.length == 1) {
       this.buffer.put(byteList.unsafeBytes(), 0, byteList.length());
    } else {
        throw context.runtime.newArgumentError("Invalid number of parameters");
    }
}
```

mmap - Java JRuby extension, unsafe bytes

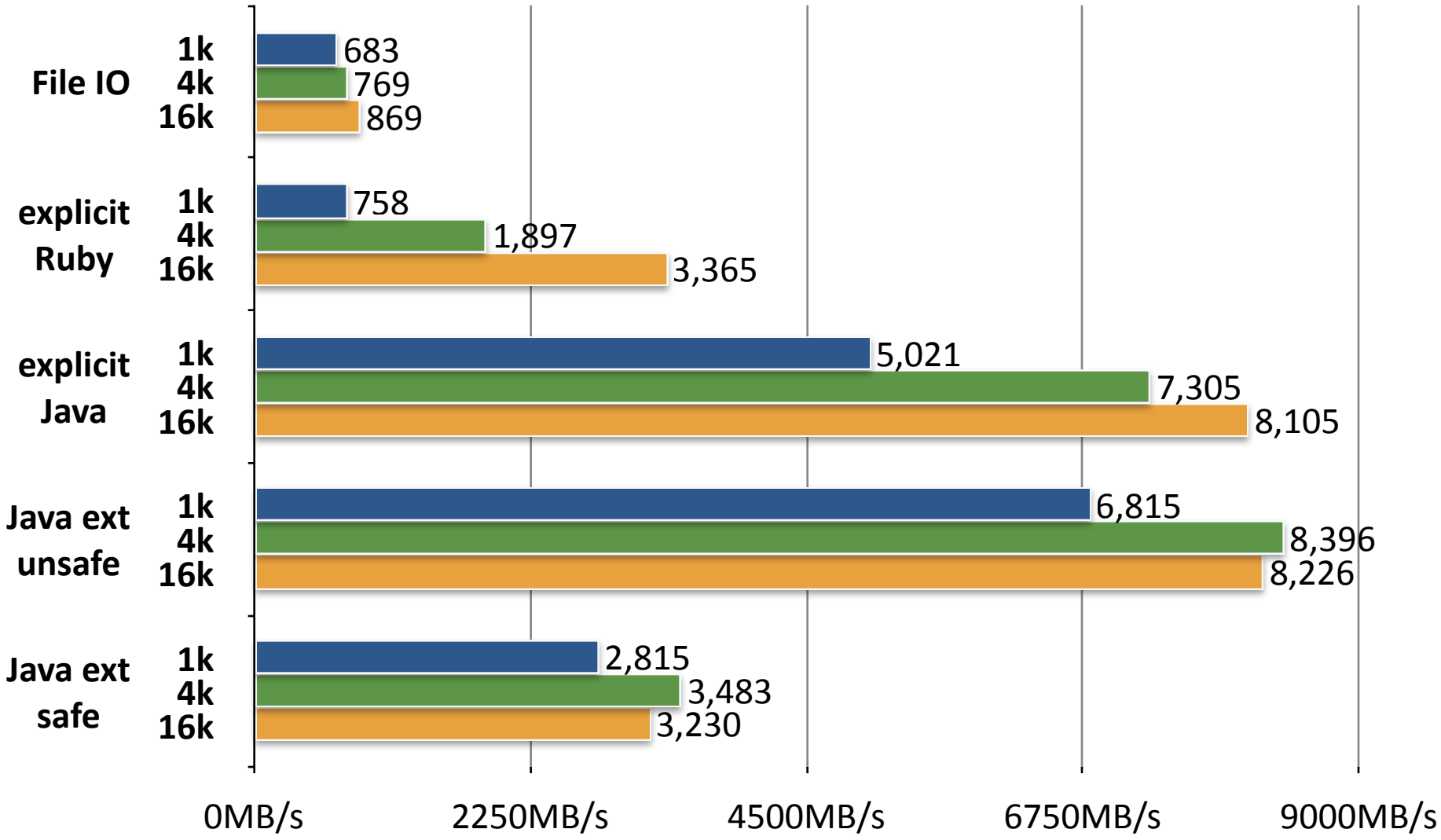


mmap - Java JRuby extension, safe bytes

```
bench("mmap", 16) do |write_count, buffer|
  out.position = 0

  write_count.times.each do
    out.put_bytes_copy(buffer)
  end
end
```

```
@JRubyMethod(name = "put_bytes_copy", required = 1, optional = 2)
public void put_bytes_copy(ThreadContext context, IRubyObject[] args) {
    byte[] bytes = (RubyString)args[0]).getBytes()
    if (args.length == 3) {
        int offset = RubyNumeric.num2int(args[1
        int length = RubyNumeric.num2int(args[2
        this.buffer.put((bytes, offset, length);
    } else if (args.length == 1) {
        this.buffer.put(bytes);
    } else {
        throw context.runtime.newArgumentError("Invalid number of parameters");
    }
}
```

mmap - Java JRuby extension, safe bytes



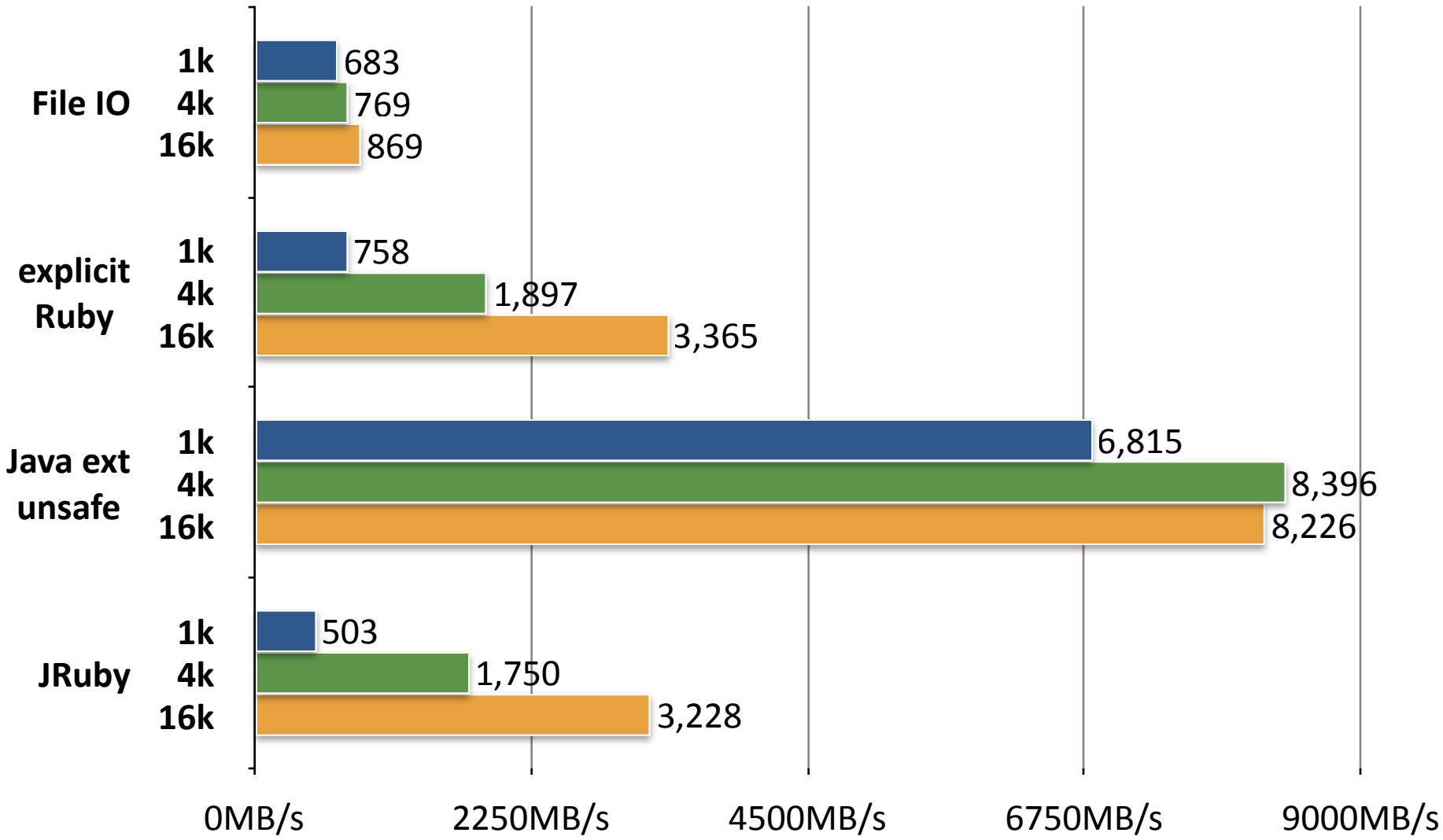
JRuby calling Java

mmap - JRuby

```
bench("mmap") do |write_count, buffer|
  out.position = 0

  write_count.times.each do
    out.put_bytes(buffer)
  end
end
```

```
module Mmap
  class ByteBuffer
    def initialize(path, size)
      @channel = RandomAccessFile.new(Java::JavaIo::File.new(path), "rw").get_channel
      @buffer = @channel.map(FileChannel::MapMode::READ_WRITE, 0, size)
    end

    ...

    def put_bytes(data)
      @buffer.put(data.to_java_bytes, 0, data.bytesize)
    end
  end
end
```

mmap - JRuby



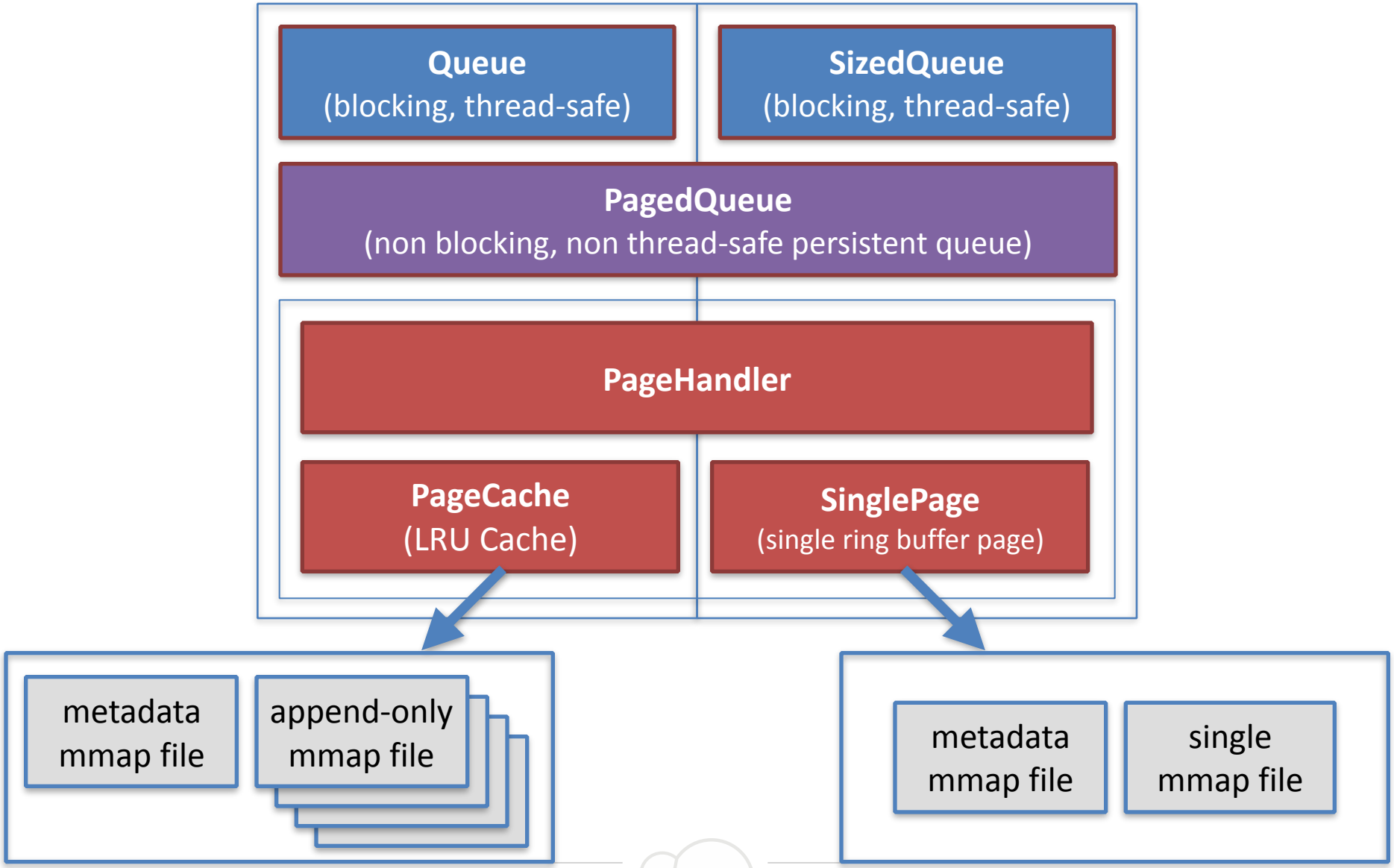
SO?

that's all you got bro?

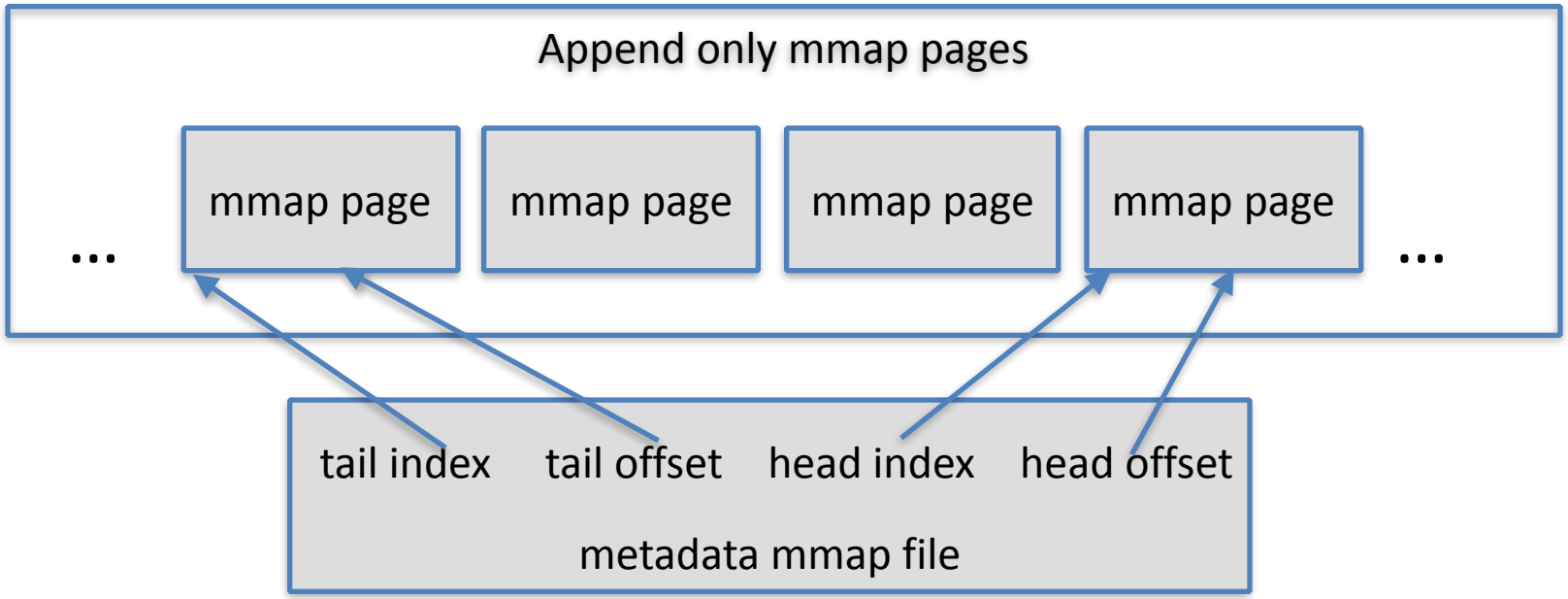
your time is up. thank you, goodbye.

elastic

Persistent Queues



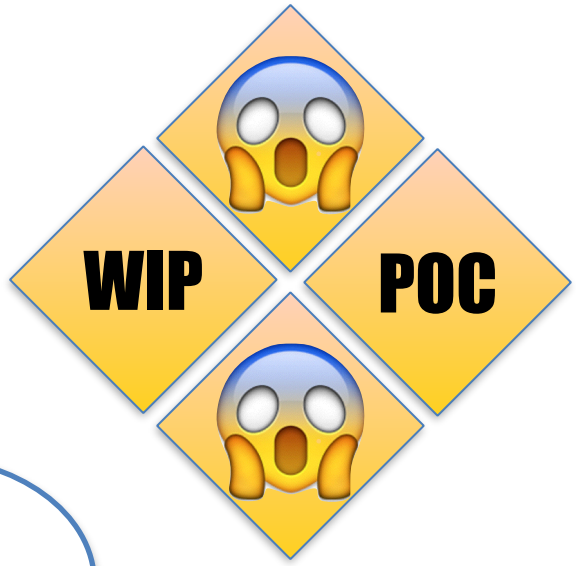
Persistent Queues

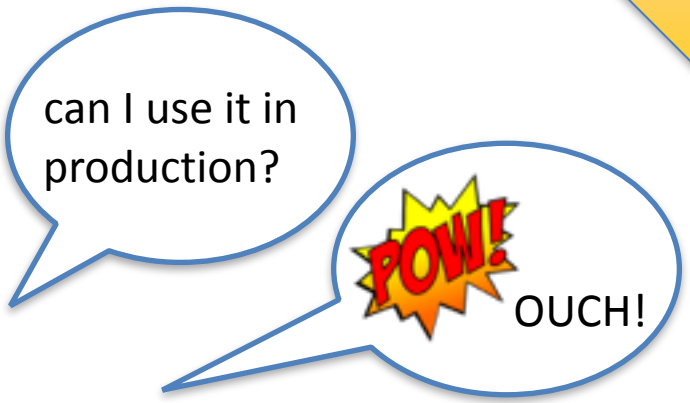


Persistent Queues

TEH CODE IZ IN DA GITHUBS

github.com/colinsurprenant/jruby-mmap-queues





Persistent Queues Benchmarks

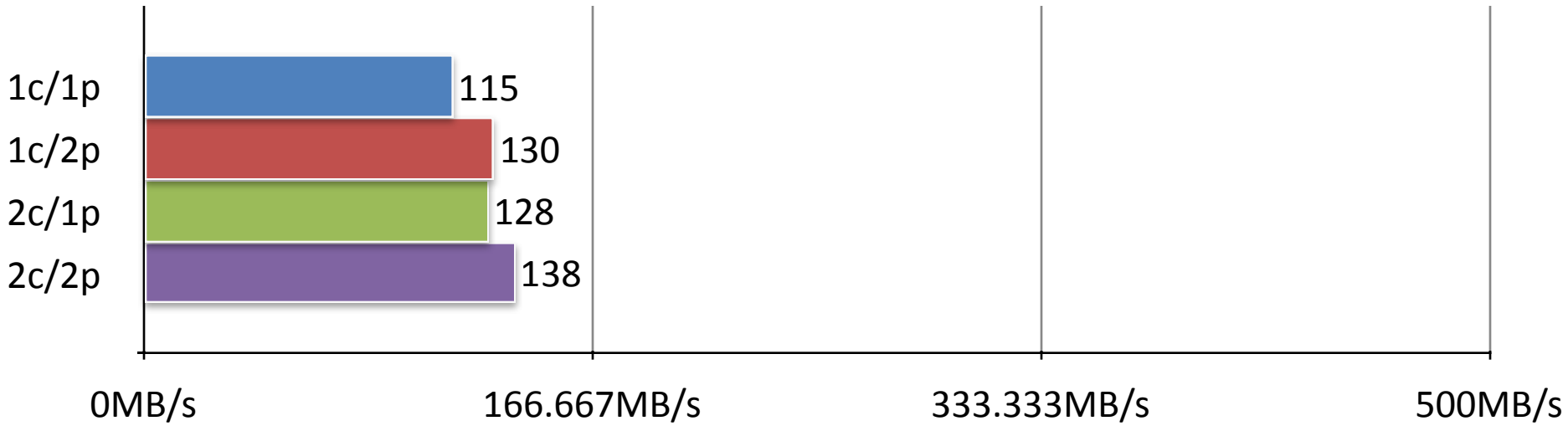
- No serialization
- 1k String objects
- 2GB mmap page size
- 2 items PageCache
- 2M items pushed per producer

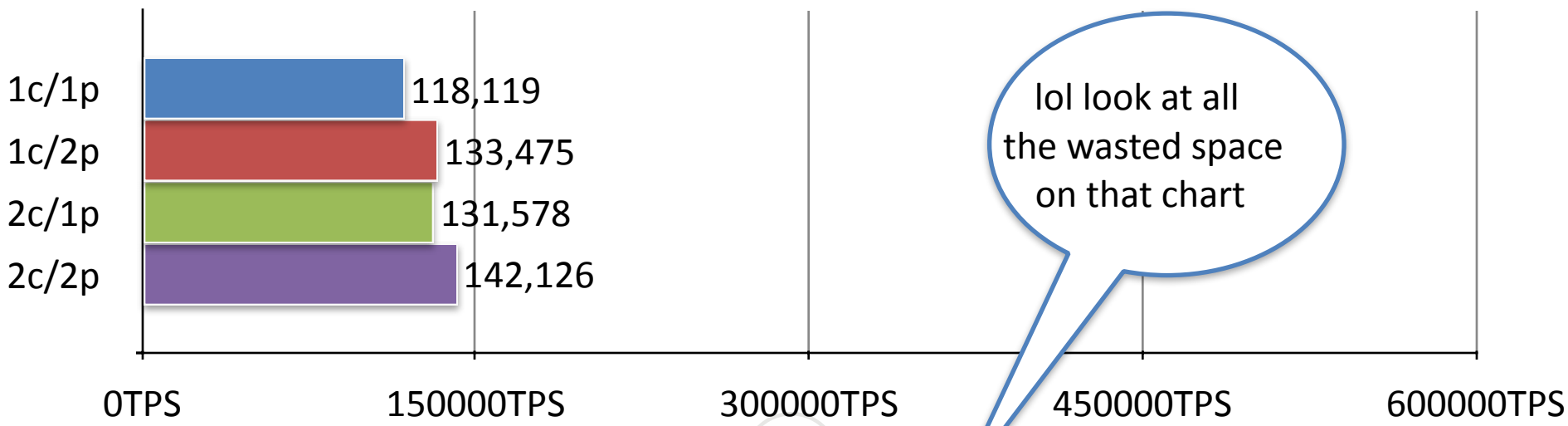
Persistent SizedQueue

- Limited queue size
- Dual queues implementation
  - push to both **persistent** queue & **in-memory** queue
    - push serialized object to persistent queue
    - push original object to in-memory queue
  - pop from memory queue only
    - **avoids deserialization** cost on pop
    - just **updates metadata** of persistent queue

sized queues sucks.

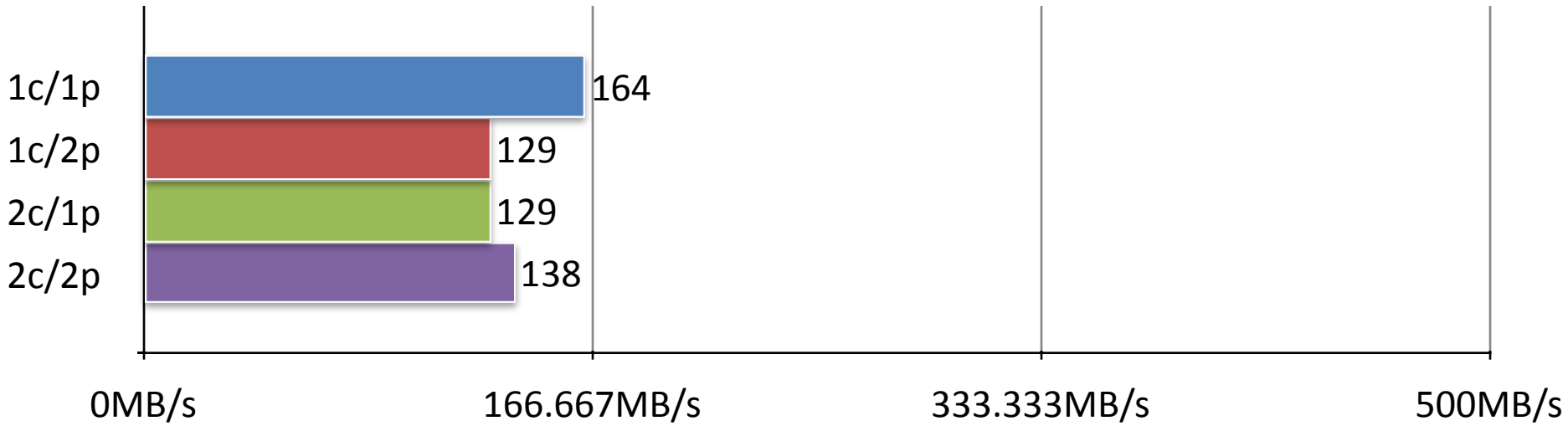
Persistent SizedQueue / PageCache

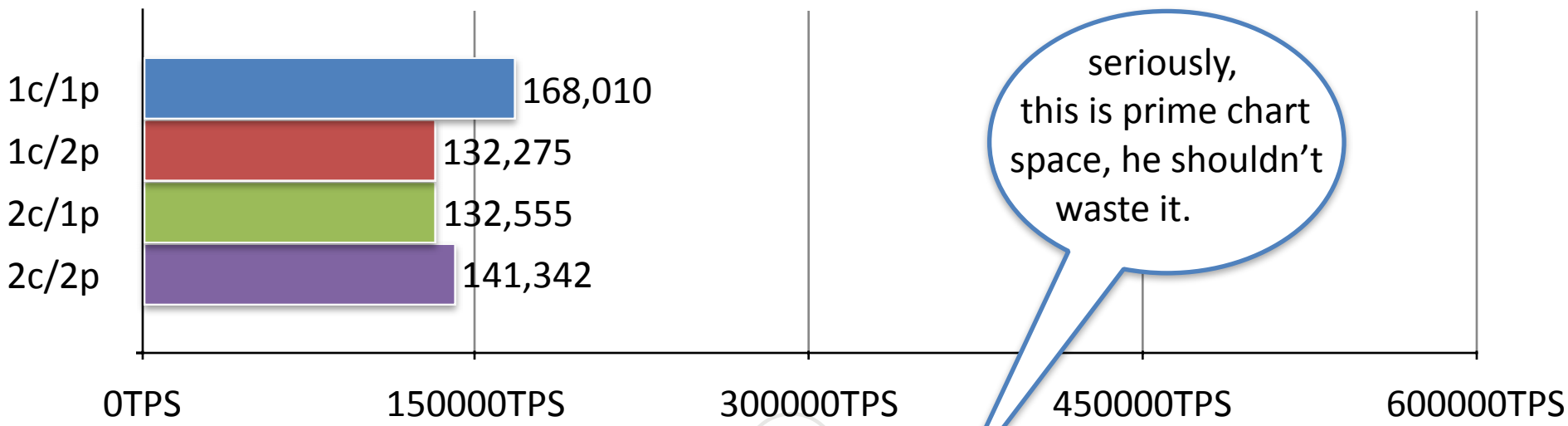




lol look at all the wasted space on that chart

Persistent SizedQueue / SinglePage



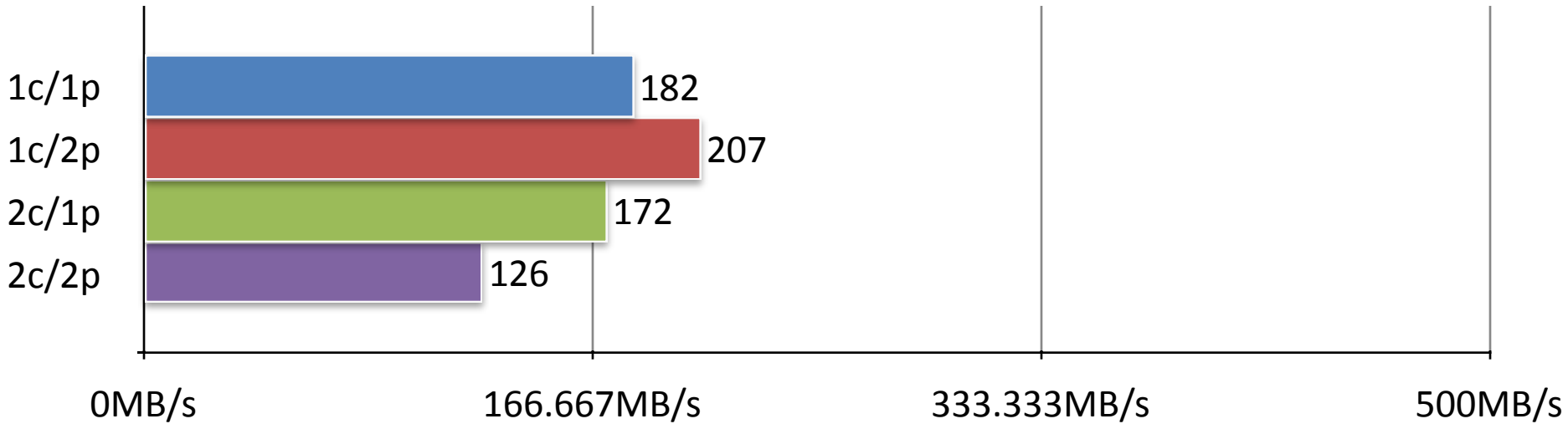


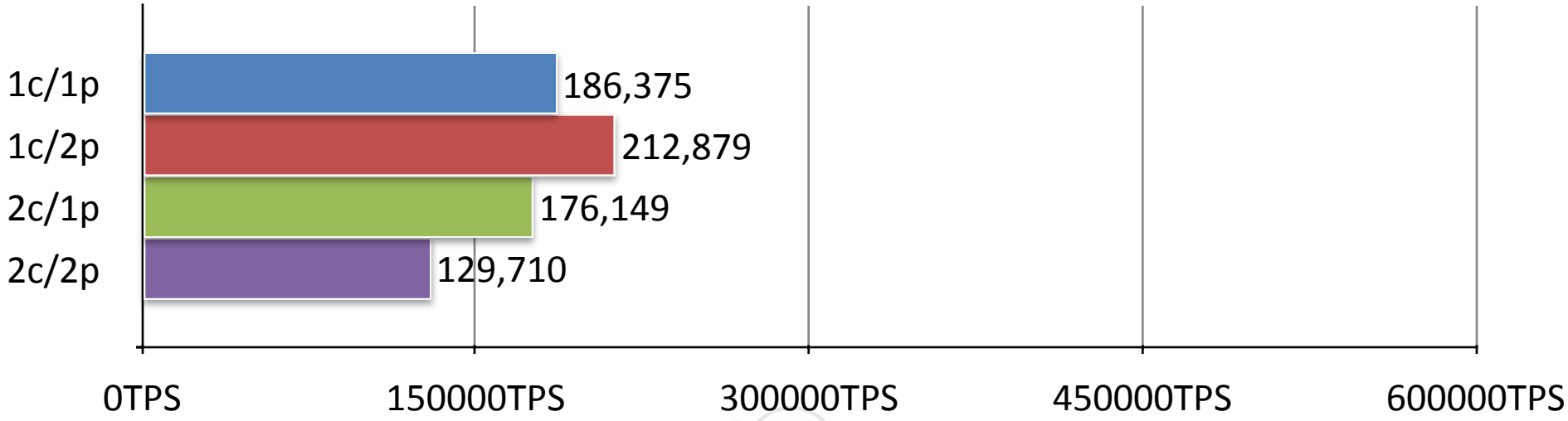
seriously,
this is prime chart
space, he shouldn't
waste it.

Persistent Queue

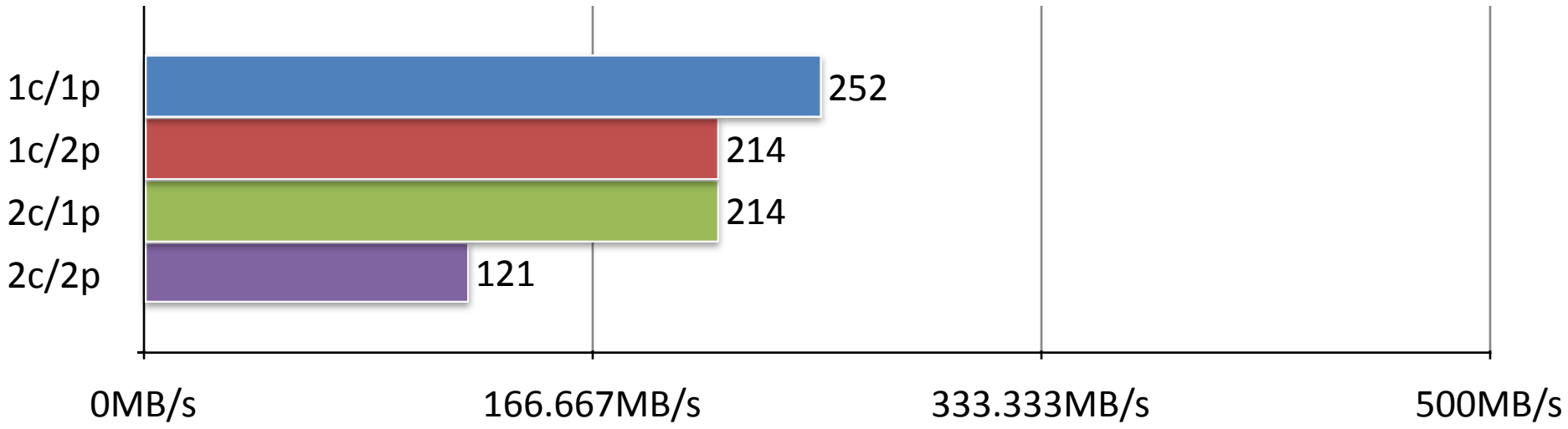
- Unlimited queue size
- Push & pop on persistent queue
  - serialize on push
  - deserialize on pop
- Essentially a thread-safety wrapper around **PagedQueue**

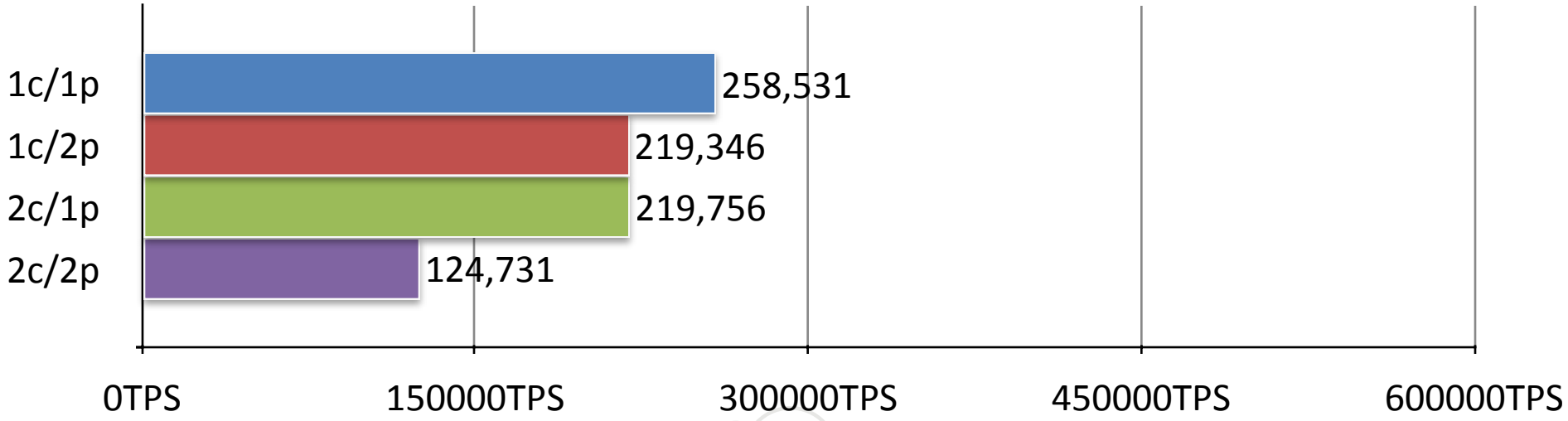
Persistent Queue **Read/Write**



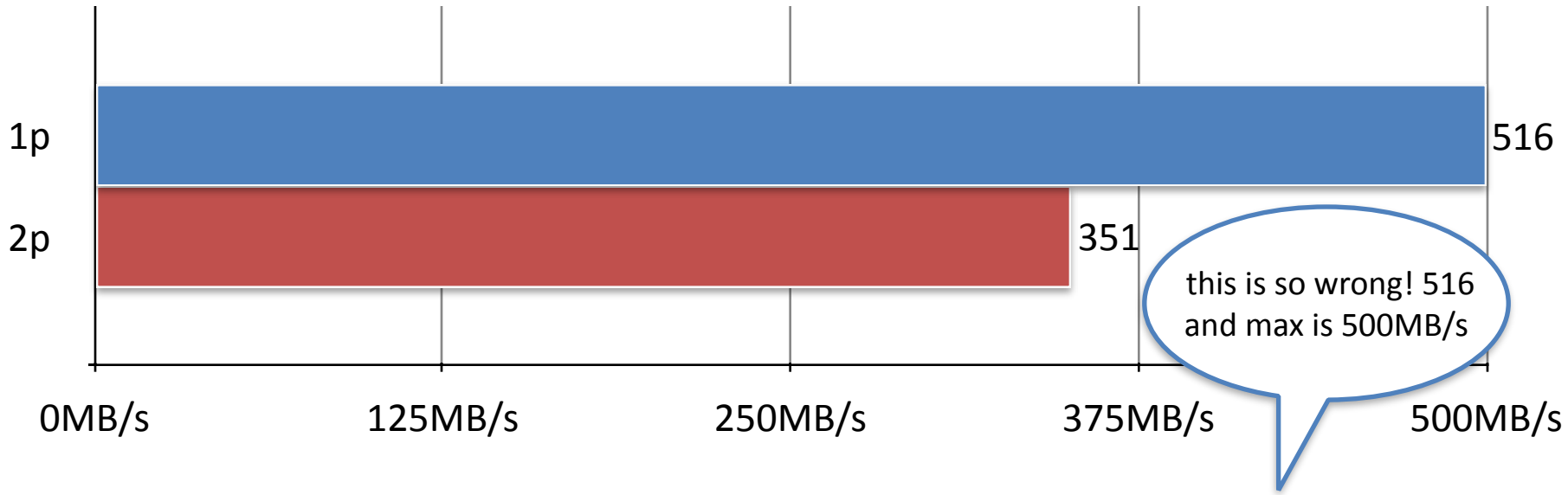


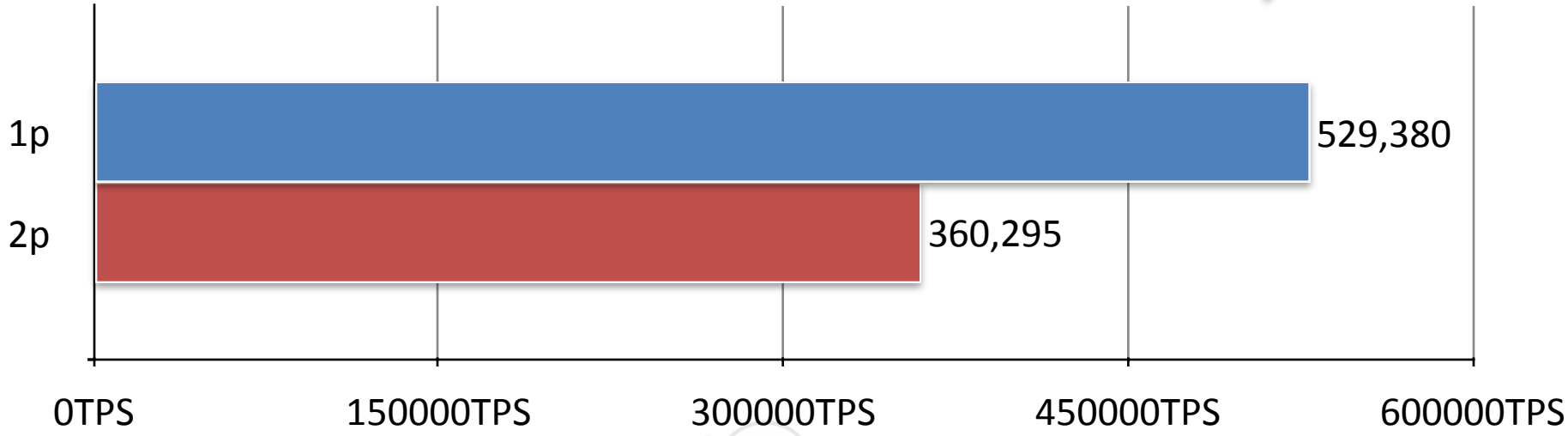
Persistent Queue **Write then Read**





Persistent Queue **Write Only**

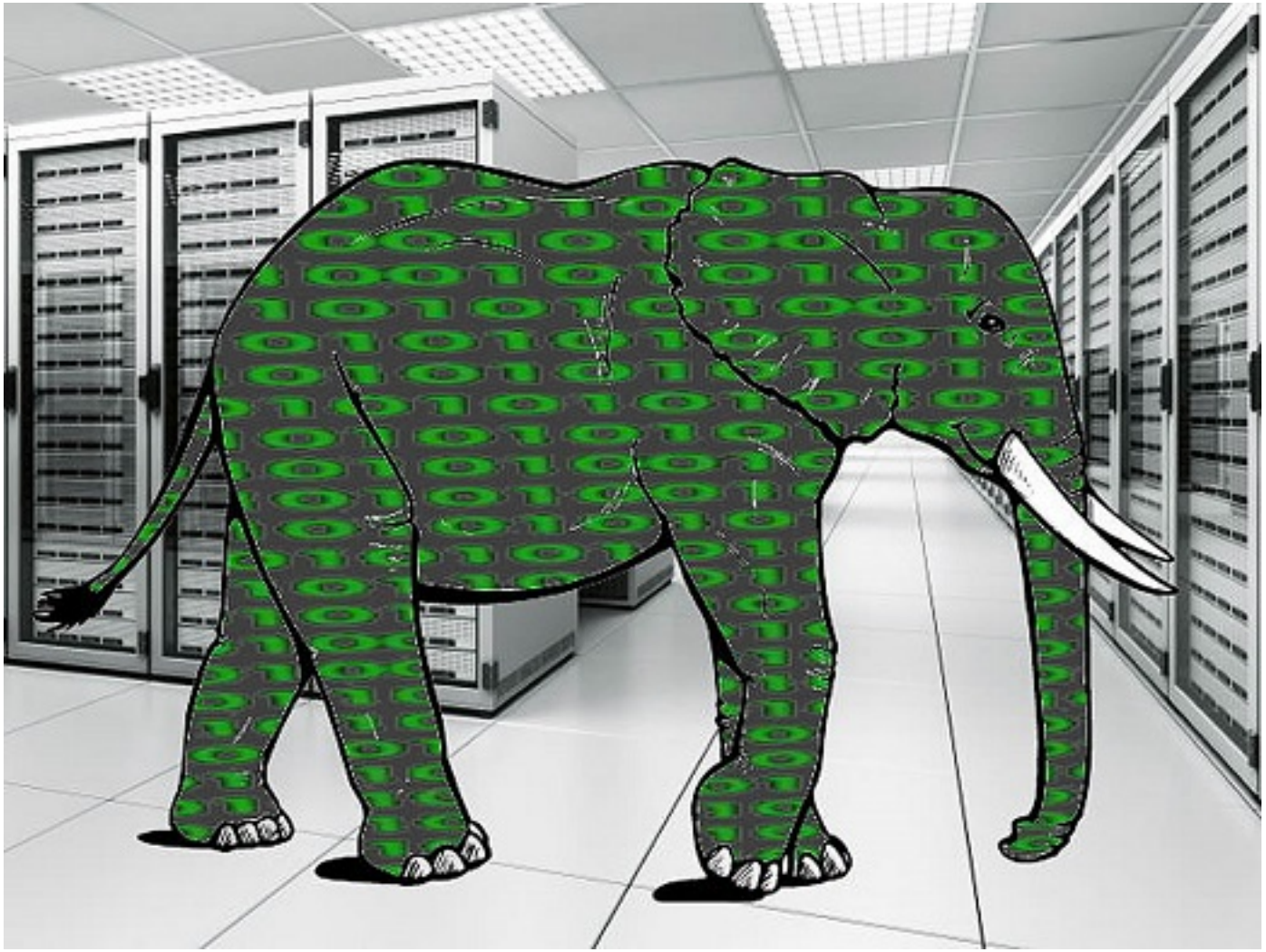




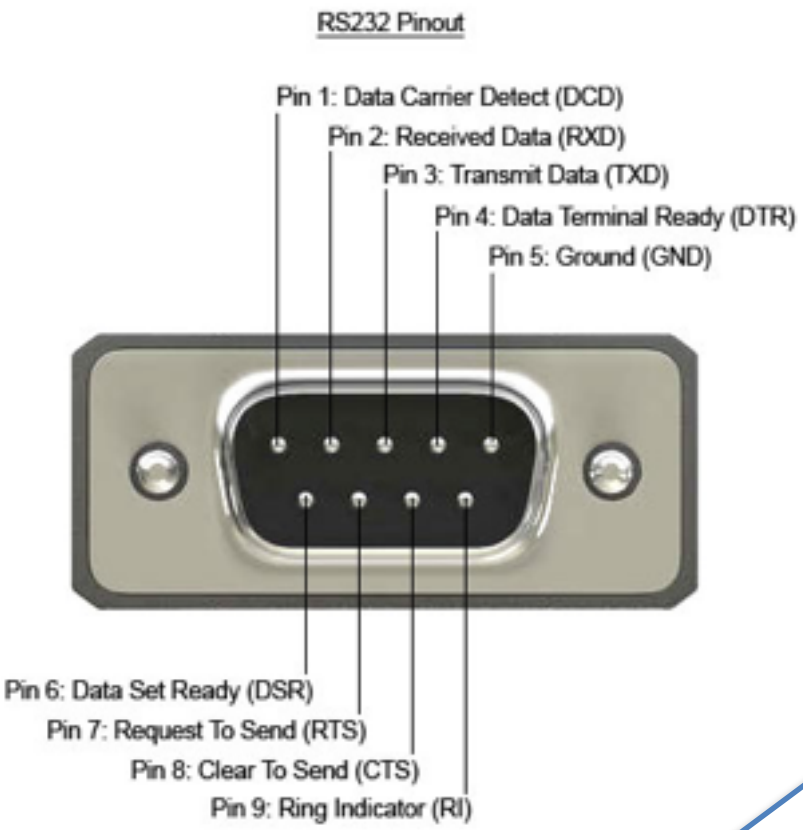
Questions/Notes

- Is the SizedQueue dual-queues idea really faster?
- Is the PageCache LRU optimal?
  - why not just use 1 head + 1 tail page cache?
- Can we find better page size / cache size?
- How does that performs on spinning disks?
- Faster alternatives to pages + metadata algo?
- Review page & metadata write sequences for resiliency
  - add strategic force() call?
- implement kafka-like multi-consumers api?

Elephant in the Room



Elephant in the Room



Serialization!

Hey look, there's **Guy Boertje**, author or the **JrJackson** JSON gem right there!

Yeah, he's the one who should fix all the things!

Thank you!

Colin Surprenant
github.com/colinsurprenant
@colinsurprenant

www.elastic.co

elastic