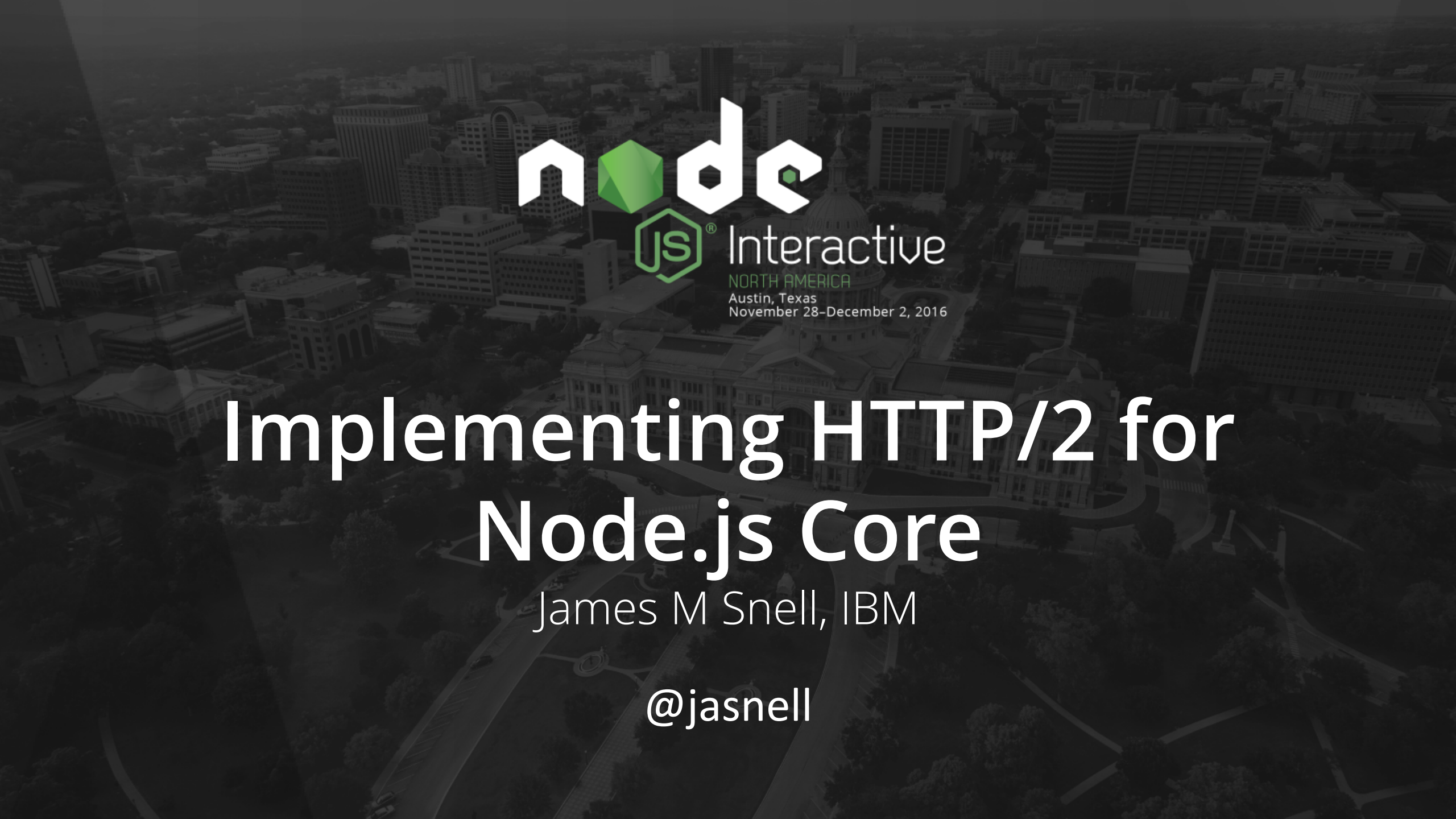

node
JS®
Interactive
NORTH AMERICA
Austin, Texas
November 28–December 2, 2016



Implementing HTTP/2 for Node.js Core

James M Snell, IBM

@jasnell

Why

Node.js is, and has been, primarily a platform for Web application development

While, Node.js presumes a “small core” philosophy for most things, it includes support for the most basic Internet protocol standards.

Why

HTTP/2 support is an evolution of the HTTP support that is already included in core.

Note that It could (and still might) be implemented as a standalone module

HTTP v1.1

The current HTTP implementation in core is optimized very specifically to HTTP v1.1 in every layer.

The http_parser native library is a line-based text parser,

The JavaScript classes write HTTP text directly to the socket,

The server and client agent assume short-lived socket lifetimes.

HTTP/2 Overview

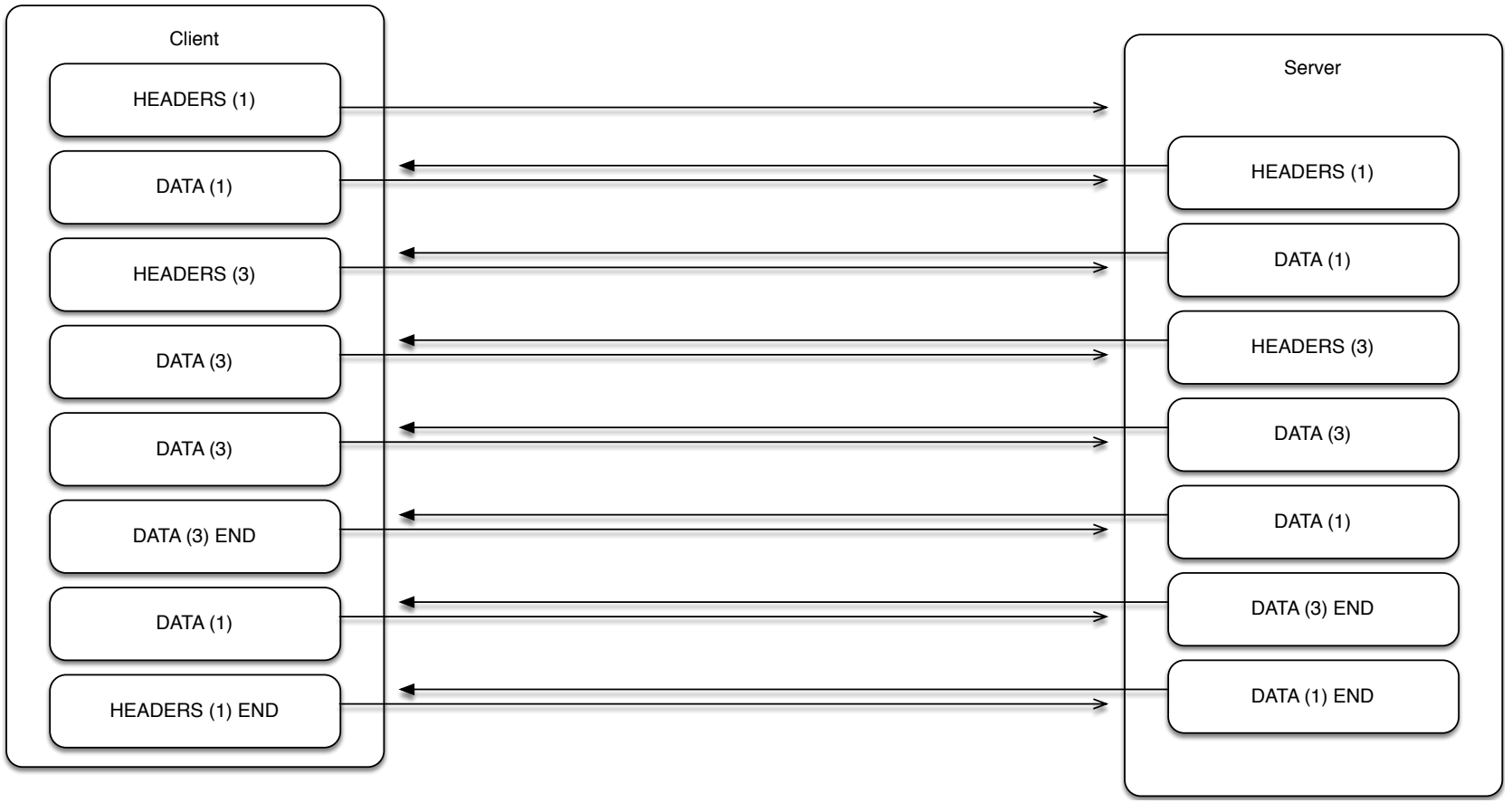
HTTP/2 is a very different protocol than HTTP v1.1

Requiring a new implementation if it's going to be added to Node.js core

HTTP/2 Overview

HTTP/2 moves to a binary framing protocol instead of the text and line-delimited approach used by HTTP v1.1.

Data is fragmented into frames and multiple “streams” can be used over a single socket connection.



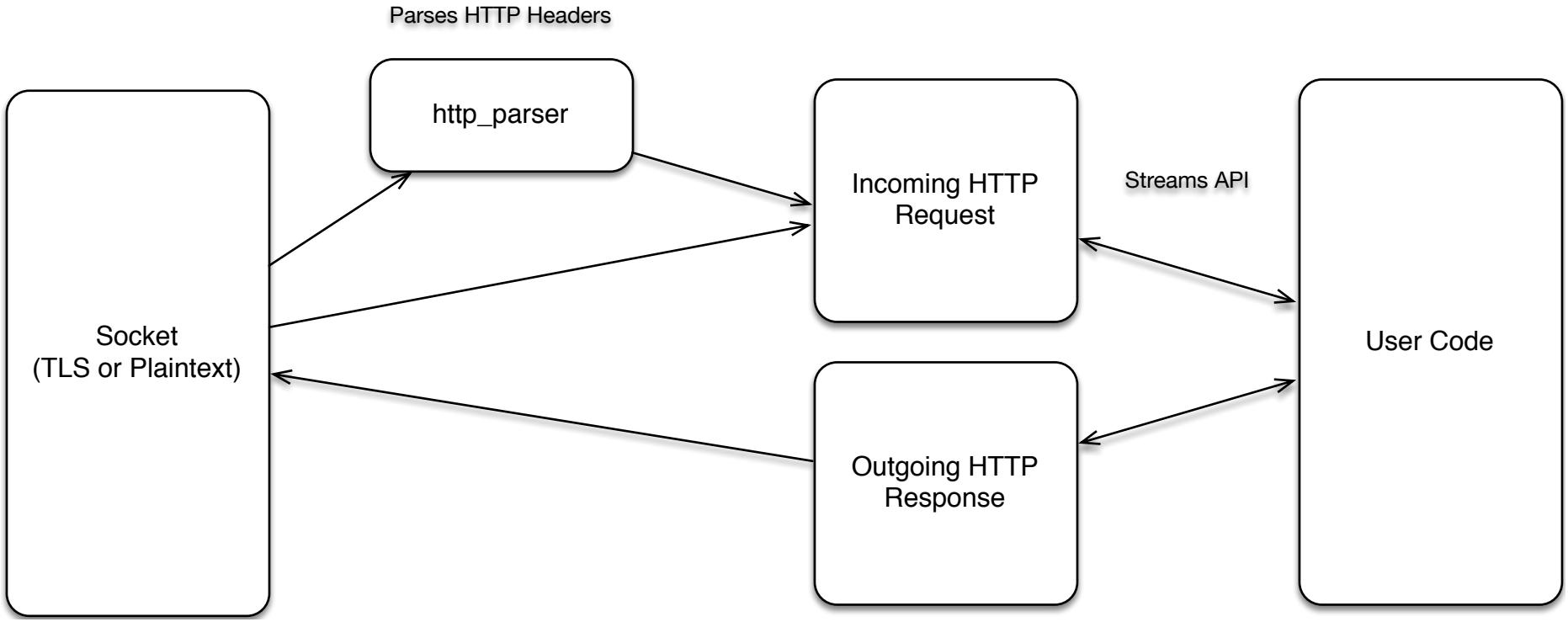
HTTP/2 Overview

… also adds stateful header compression, server push, stream prioritization, flow control, new protocol extensibility mechanisms, new requirements and new limitations.

HTTP/2 Overview

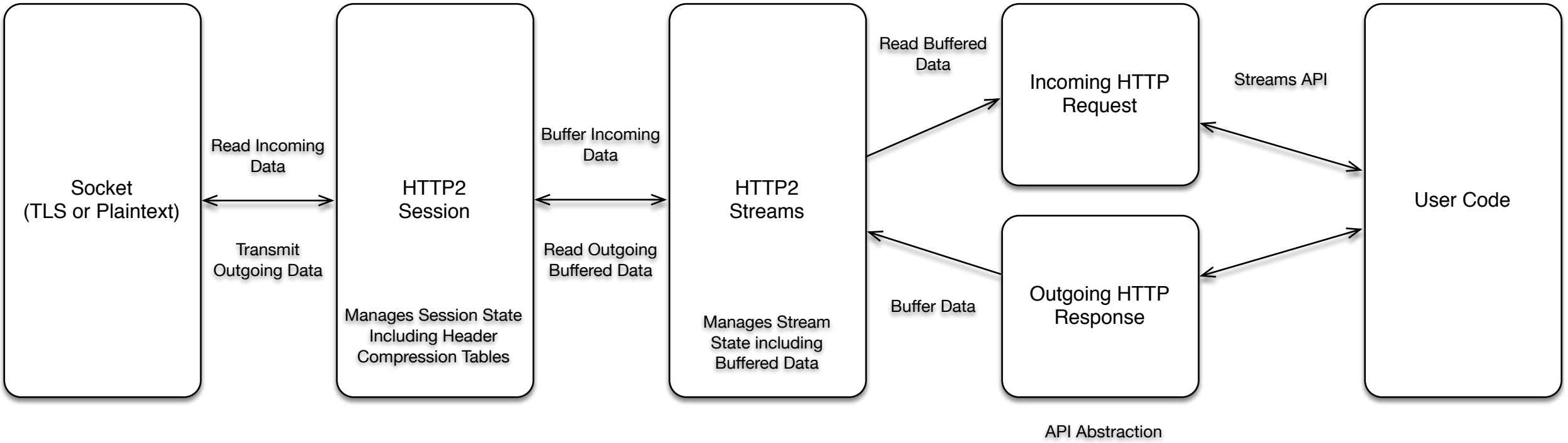
- HTTP/2 adds new constraints that Node.js currently does not have:
  - HTTP/2 socket connections are long-lived.
  - Multiple request/response transactions occur over a single socket connection.
  - Errors in an individual request/response transaction do not necessarily cause the socket connection to be destroyed.
  - Protocol controls and data are fragmented into discreet frames that are transmitted in priority sequence.
  - Headers are compressed and sequenced using a stateful algorithm.
  - Client implementations in Chrome, Firefox and other browsers require the use of TLS.
  - User code cannot be permitted to directly read from, or write to, the socket.

HTTP v1.1 Server Flow



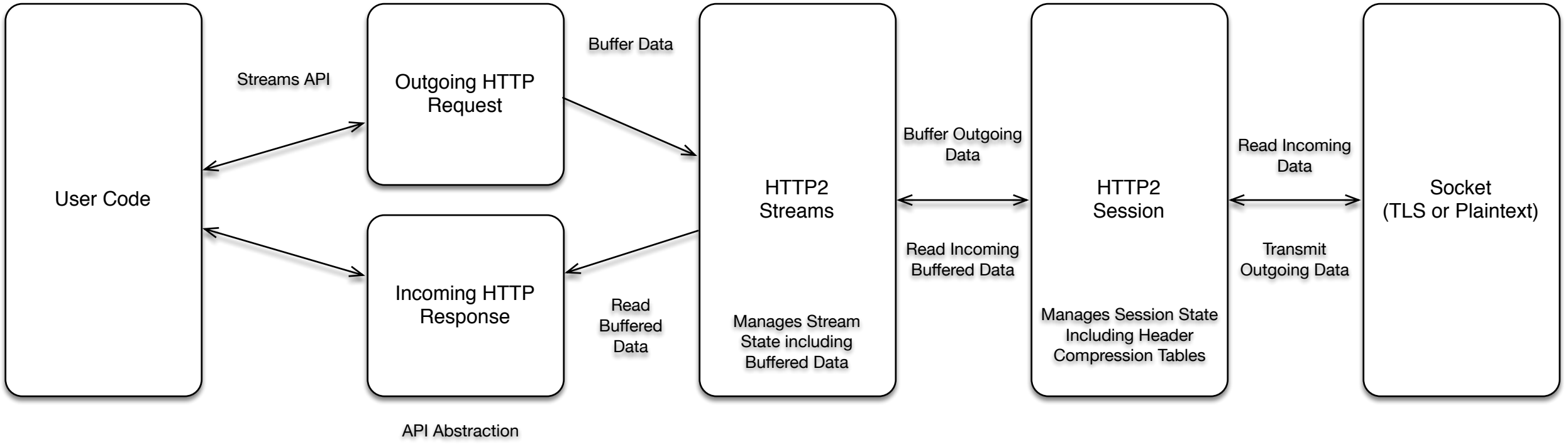
Performance is optimized by streamlining the flow of data between the user code and socket

HTTP/2 Server Flow



Because Data is chunked, sequenced, and multiplexed over a single connection, data writes and reads must be buffered. Compression, stream priority and flow control state must also be maintained.

HTTP/2 Client Flow



HTTP/2 API Plan

- The goal is to expose two distinct, supported API levels:
  - Low Level Http2Session/Http2Stream classes
  - High Level HTTP-specific API modeled after current HTTP v1.1 API

- In order of priority, API design decisions are being driven by:
  - Performance and Security considerations
  - Expected API compatibility
  - Support for new HTTP/2 protocol features

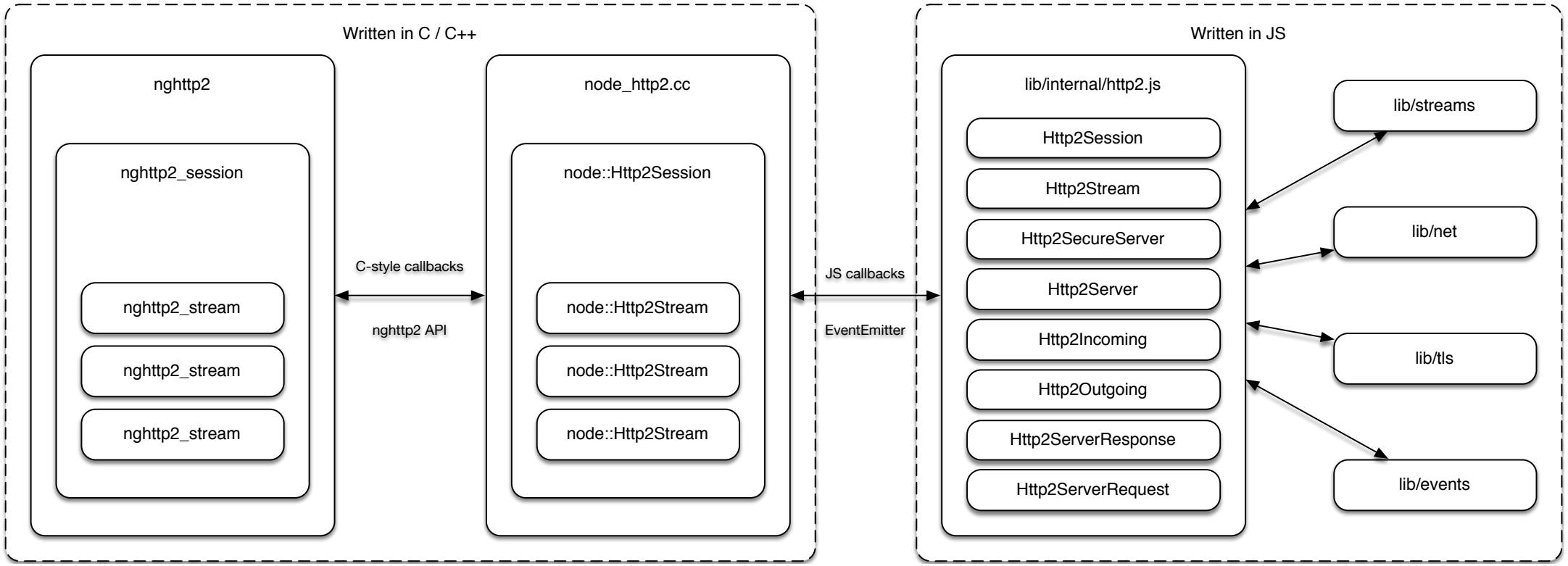
- Design of the Low Level API will be finalized after implementation of the High Level API is complete

IBM

HTTP/2 Implementation

- Majority of the heavy lifting falls on the nghttp2 library – a native HTTP/2 implementation written in C
- nghttp2 is without question the best HTTP/2 library implementation available.
- A small amount of additional C/C++ code is introduced into the Node.js binary to bridge the nghttp2 API into Node.js
- The implementation in Node.js is split between C/C++ and JS

HTTP/2 In Core



HTTP/2 Server

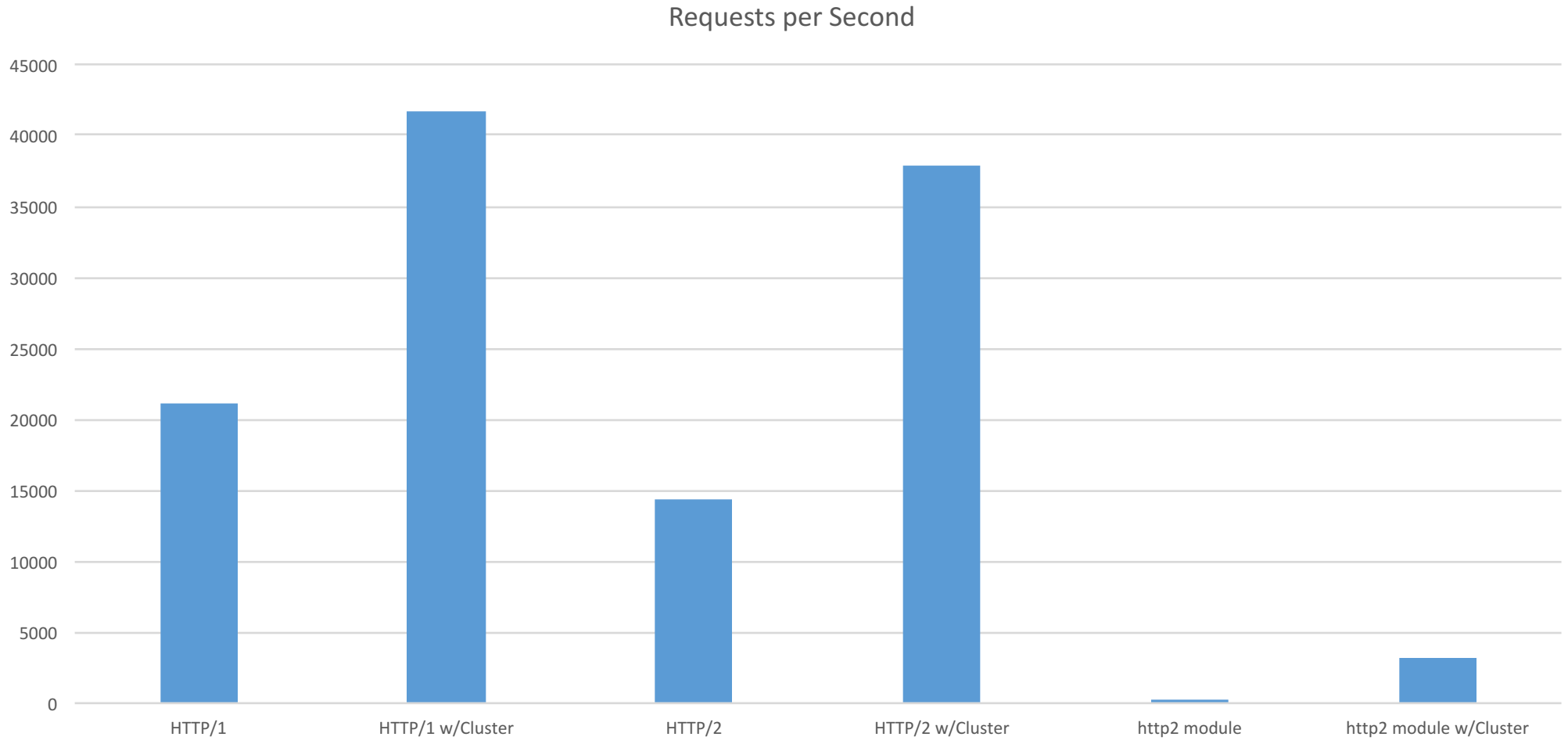

```
const http2 = require('http').HTTP2;
const options = { // Get TLS Options };
const server = http2.createSecureServer(options, (req, res) =>
{
  res.writeHead(200, {'content-type': 'text/plain'});
  res.end('Hello World');
});

server.listen(80, 'localhost', () => {
  console.log('Ready!');
});
```

HTTP/2 Server w/Push

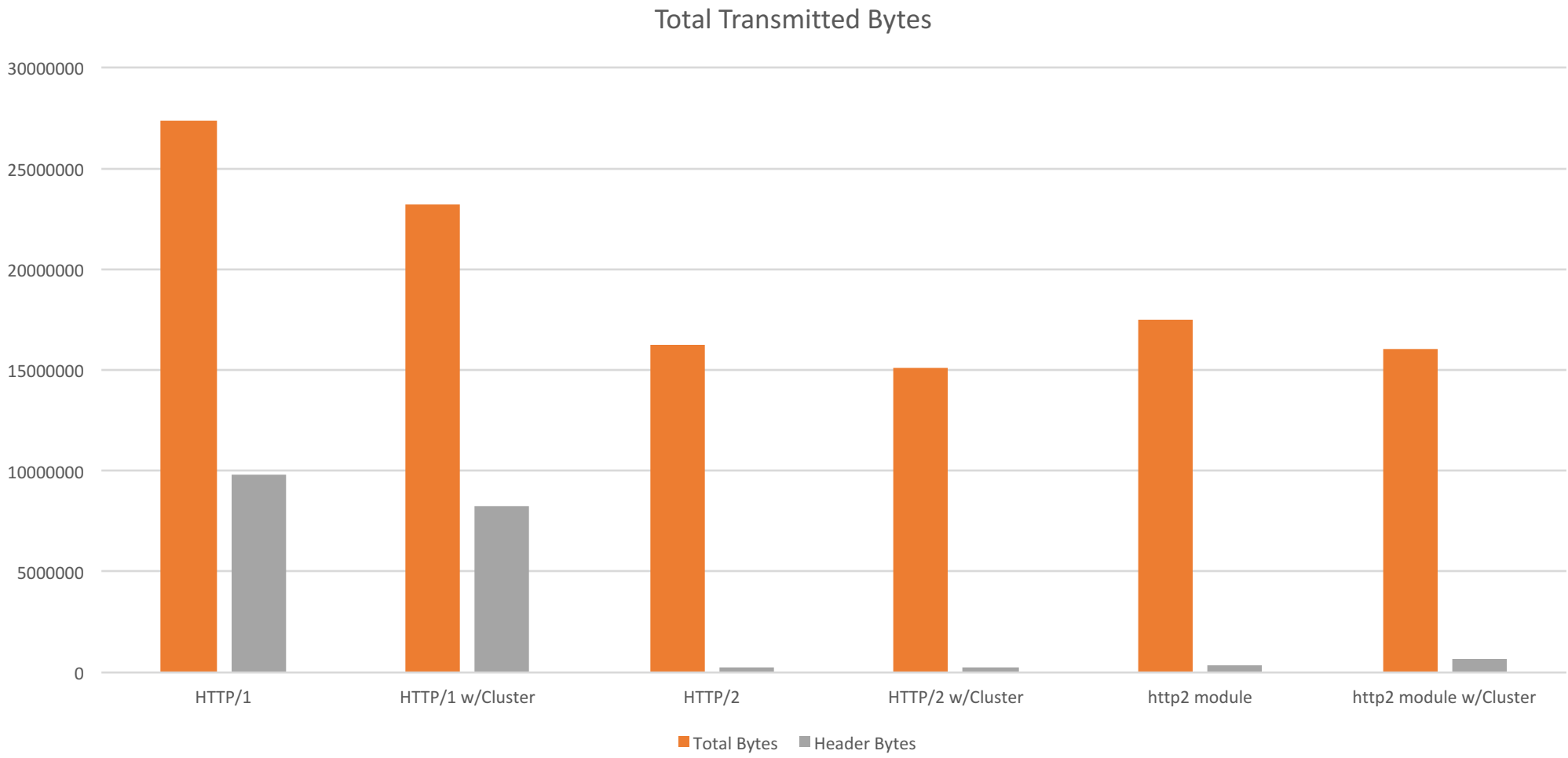

```
const http2 = require('http').HTTP2;
const options = {  // Get TLS Options};
const server = http2.createSecureServer(options, (req, res) => {
  res.writeHead(200, {'content-type': 'text/plain'});

  // The push API is temporary
  const image = fs.createReadStream('local/path/to/image');
  const pushResponse = res.createPushResponse();
  pushResponse.path = '/myimage.jpg';
  pushResponse.push((req, res) => image.pipe(res));
  res.end('Hello World');
});

server.listen(80, 'localhost', () => {
  console.log('Ready!');
});
```

IBM

Current Performance Profile



Current Performance Profile



Current Performance Profile

- The benchmark tested sending 100,000 requests, using 50 clients split across 8 threads.
- HTTP v1.1 test used keep-alive to pipeline requests.
- The HTTP v1.1 implementation was faster but *failed* on 15,652 requests due to concurrency limitations. Only 84,348 requests were successfully served.
- The HTTP/2 implementation successfully served 100% of requests.
- The HTTP/2 implementation sent 96.92% fewer header bytes and 40.86% fewer total bytes.

Current Performance Profile

Note that the HTTP/2 numbers show the performance of the current development image. The numbers may change significantly once additional performance and reliability enhancements are made.

HTTP/2 TODO

- My current goal is to have an experimental implementation ready to go for Node.js v8.x (may not make it into v8.0.0, but should be able to land as a semver-minor PR in a v8.x minor)

- There is still much to do:
  - HTTP v1.1 > HTTP/2 Upgrade
  - Flow Control and Prioritization APIs
  - Client (in progress!)
  - Performance Optimization
  - Conformance Testing
  - API Refinement
  - Unit and Integration Testing
  - Documentation

https://github.com/nodejs/http2

IBM

Thank you.

Twitter: @jasnell

Github: @jasnell

Gmail: jasnell@gmail.com

IBM: jasnell@us.ibm.com