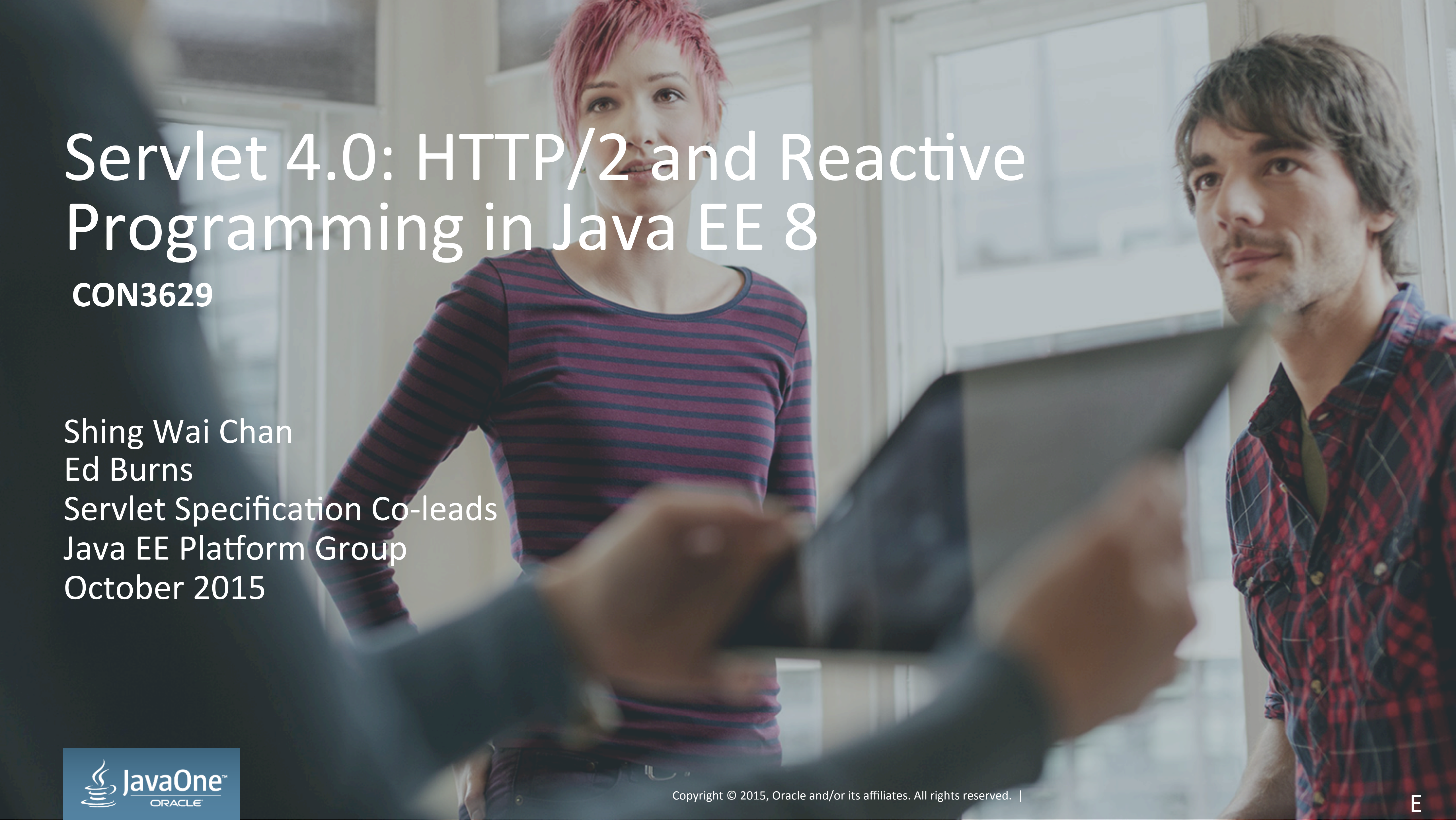
ORACLE®

JavaOne™

ORACLE®

E



# Servlet 4.0: HTTP/2 and Reactive Programming in Java EE 8

**CON3629**

Shing Wai Chan
Ed Burns
Servlet Specification Co-leads
Java EE Platform Group
October 2015

## Safe Harbor Statement

## Keep Learning with Oracle University

ORACLE
UNIVERSITY

Classroom Training | Cloud
Learning Subscription | Technology
Live Virtual Class | Applications
Training On Demand | Industries

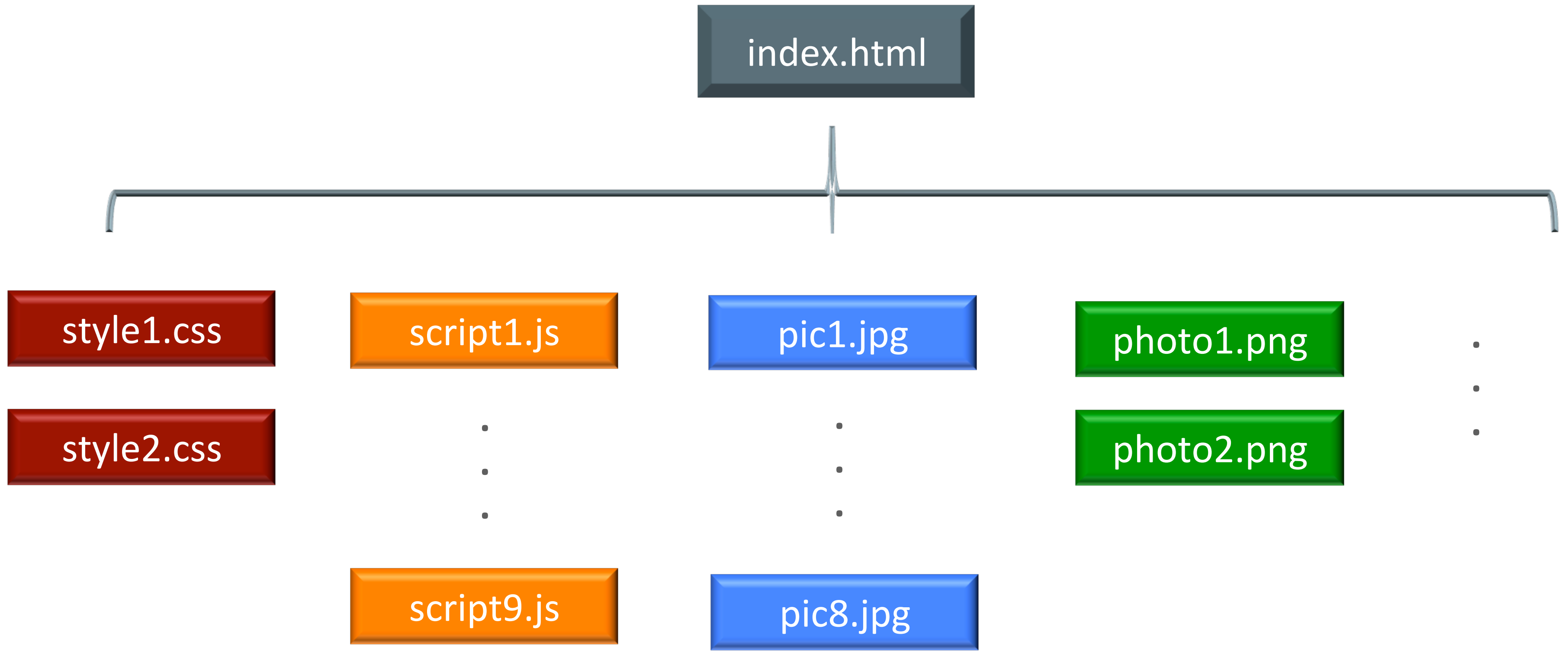
education.oracle.com

## Our Plan for Your Time Investment

## Our Plan for Your Time Investment

## Why HTTP/2?

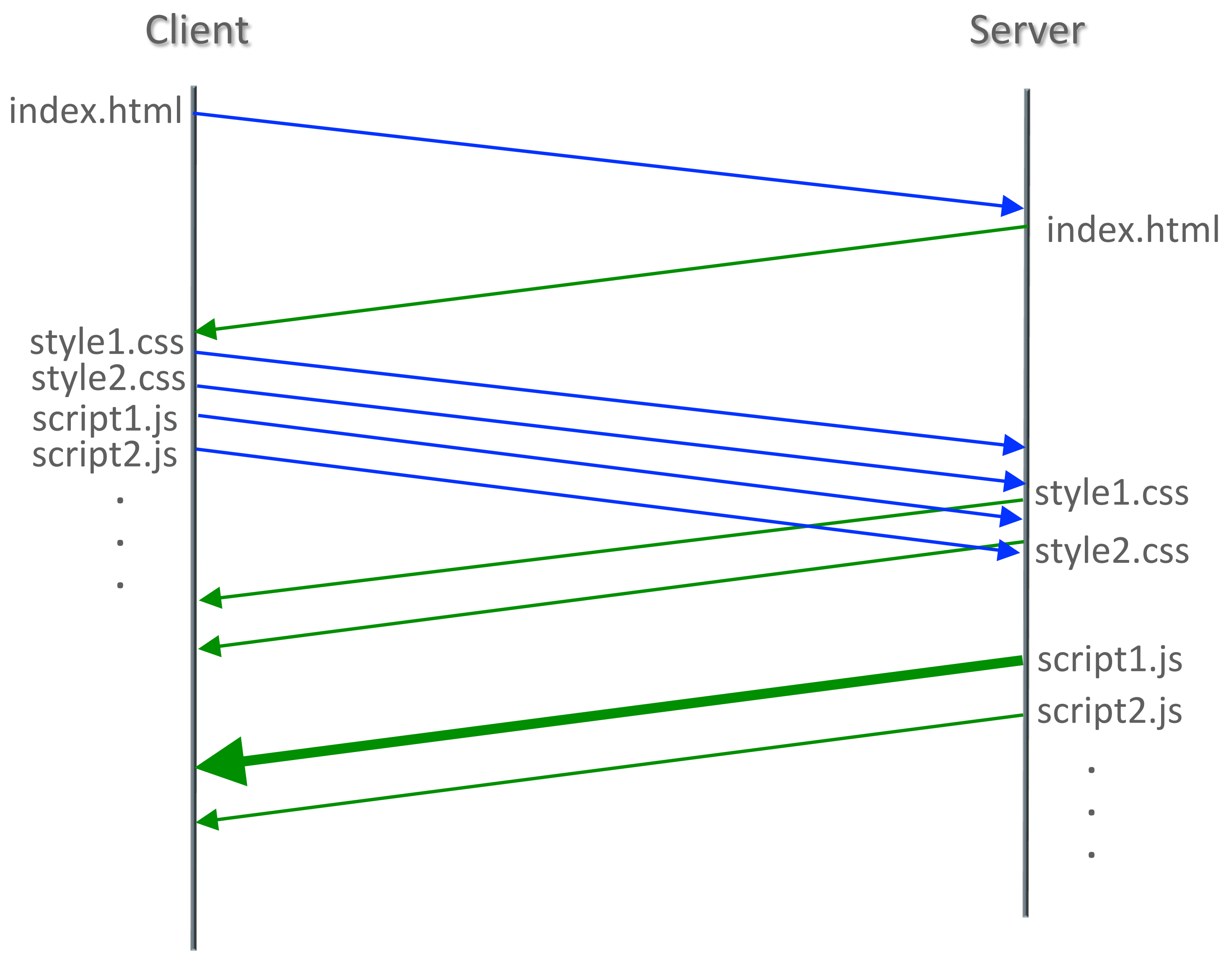
### A Real Life Example



# Why HTTP/2?

## Problems in HTTP/1.1

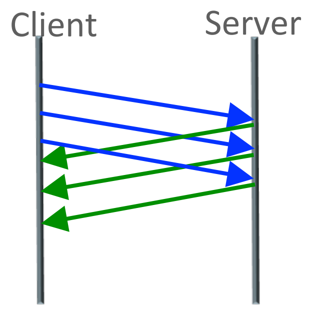
- HTTP Pipelining
- Head-of-Line blocking

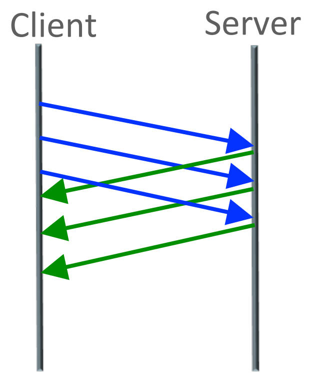


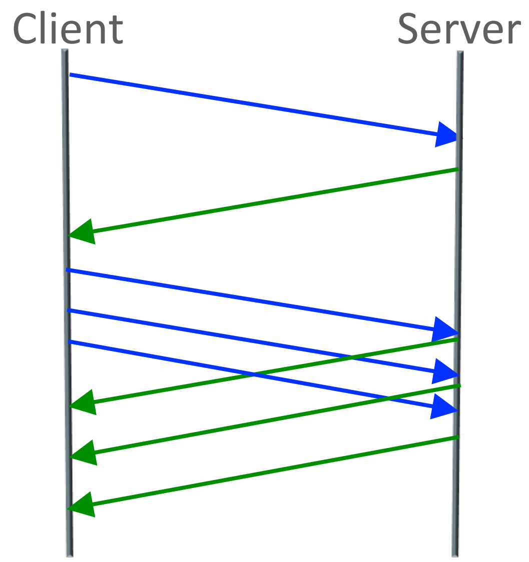
# Why HTTP/2?

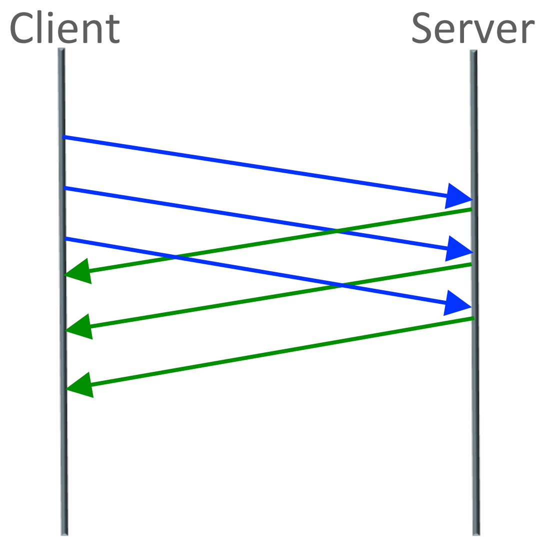
**Problems in HTTP/1.1**

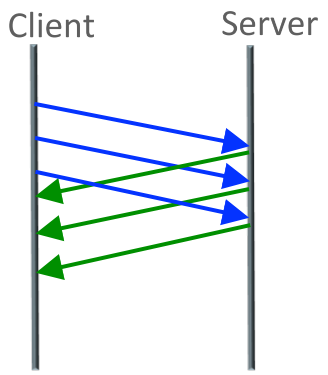
- Inefficient use of TCP sockets
- Max number of connections per host

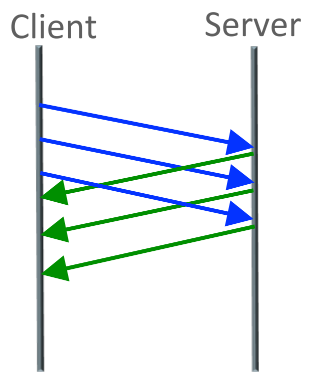












## Why HTTP/2?

**What is an optimization?**

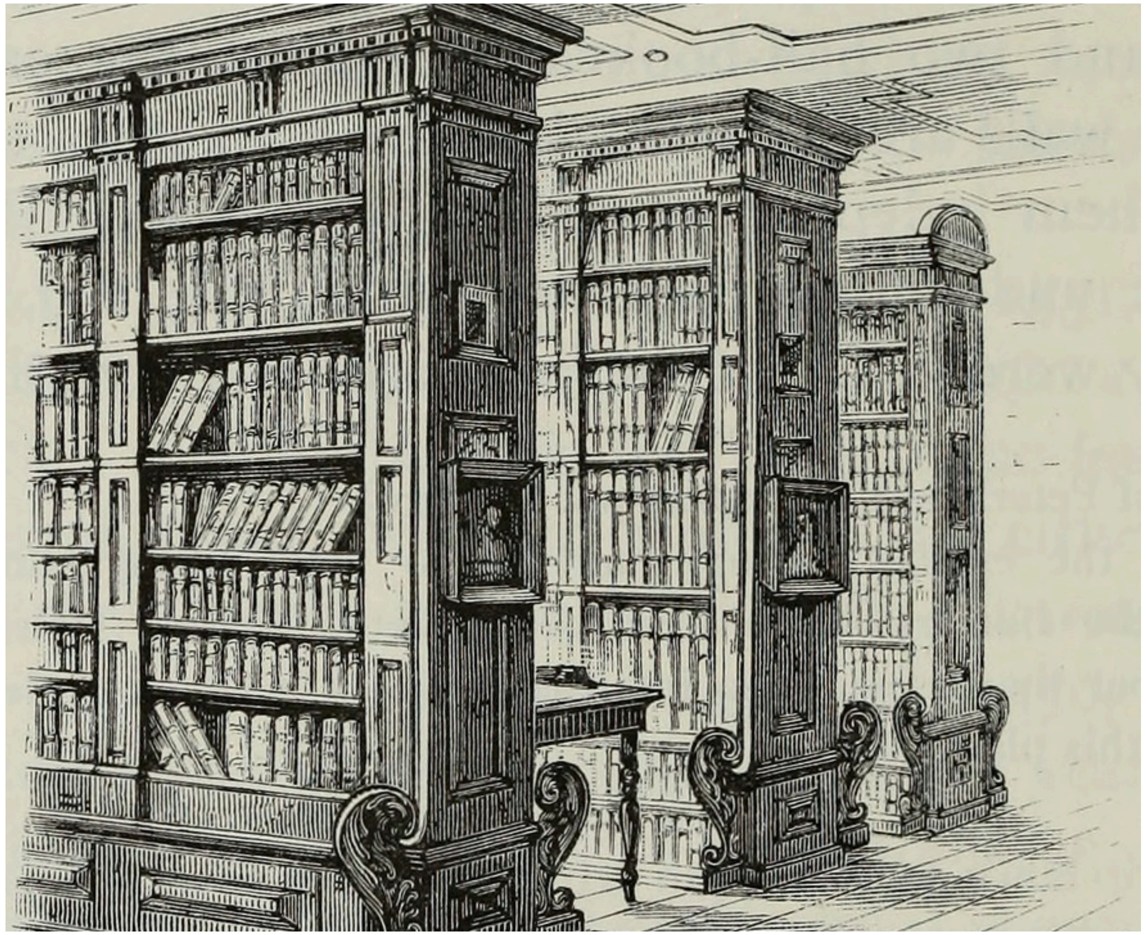
- Much of what we do in web-apps is a hack to work around shortcomings in HTTP/1.1
  - File concatenation and image sprites
  - Domain sharding
  - Inlined assets

# Our Plan for Your Time Investment

# Network Programming Review

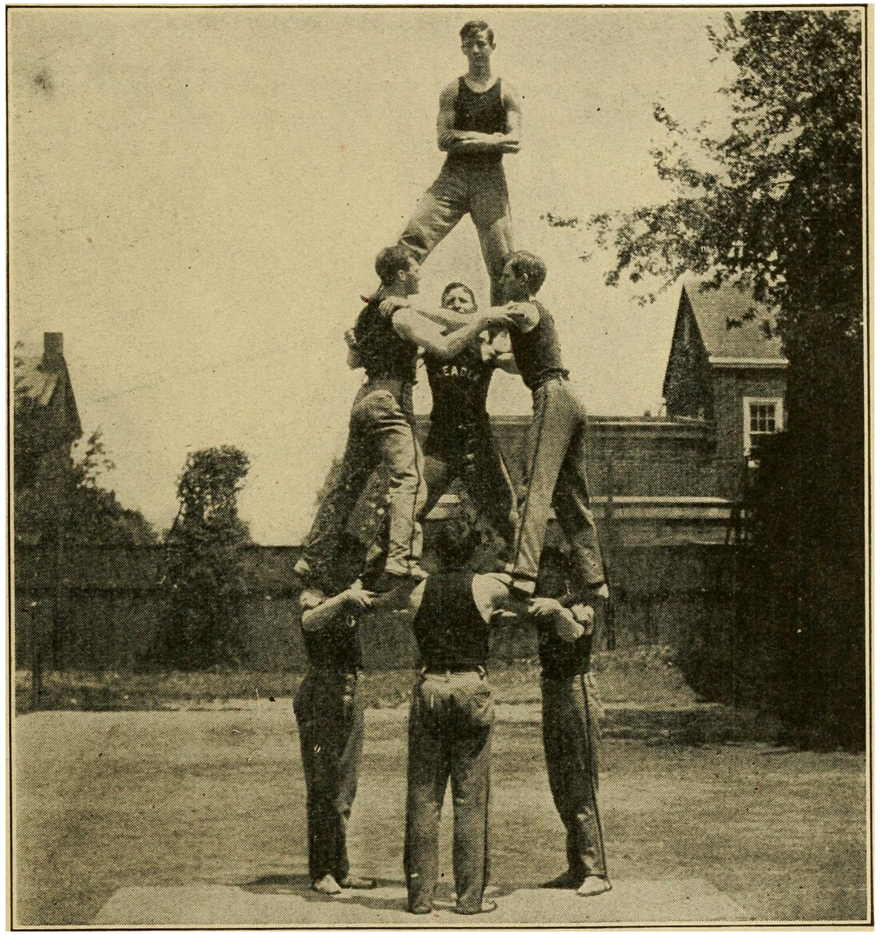
HTTP/2 is really just a new transport layer underneath HTTP/1.1

- same request/response model
- no new methods
- no new headers
- no new usage patterns from application layer
- no new usage of URL spec and other lower level specs

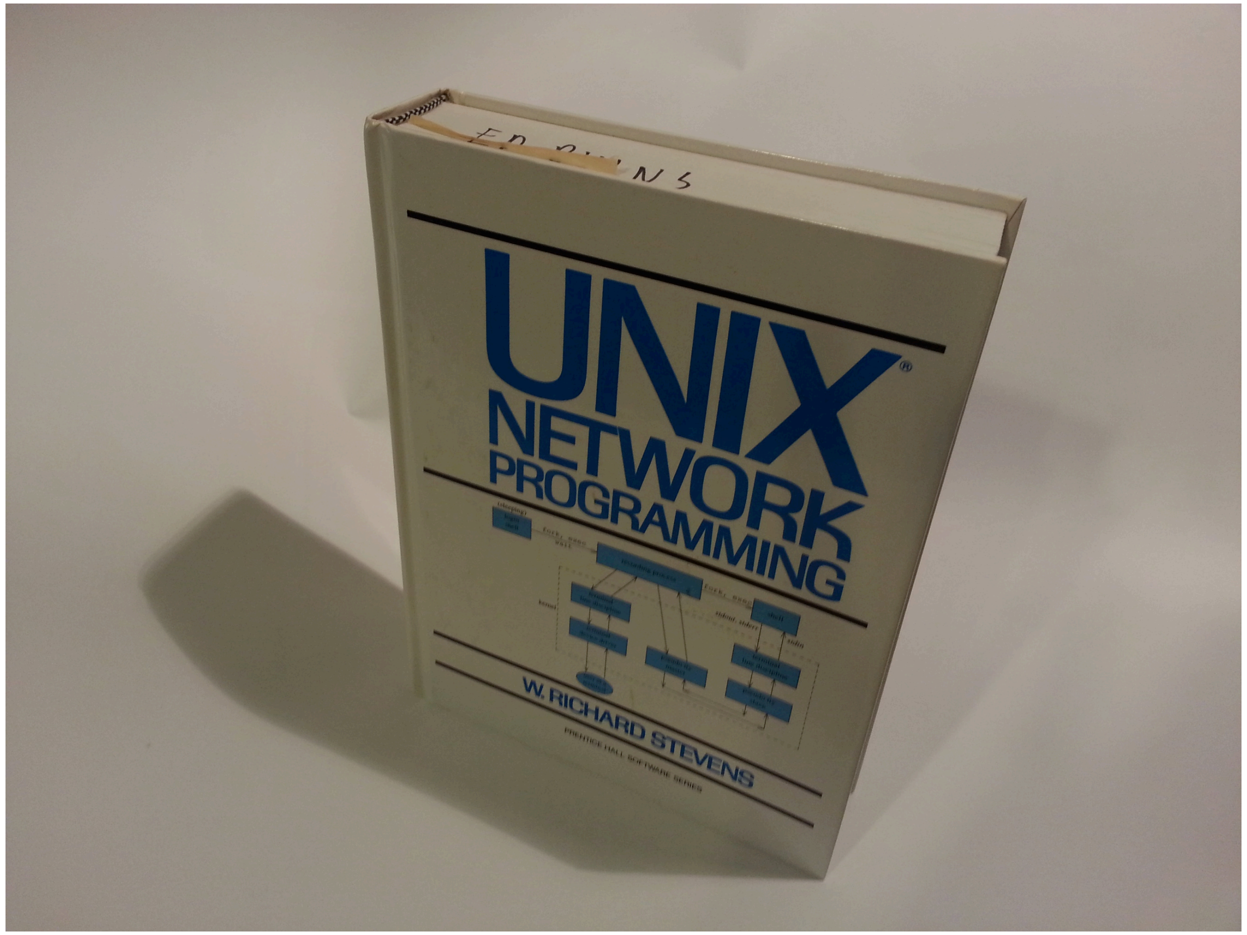


E

## Standing on the Shoulders



## Network Programming Review



## Network Programming Review

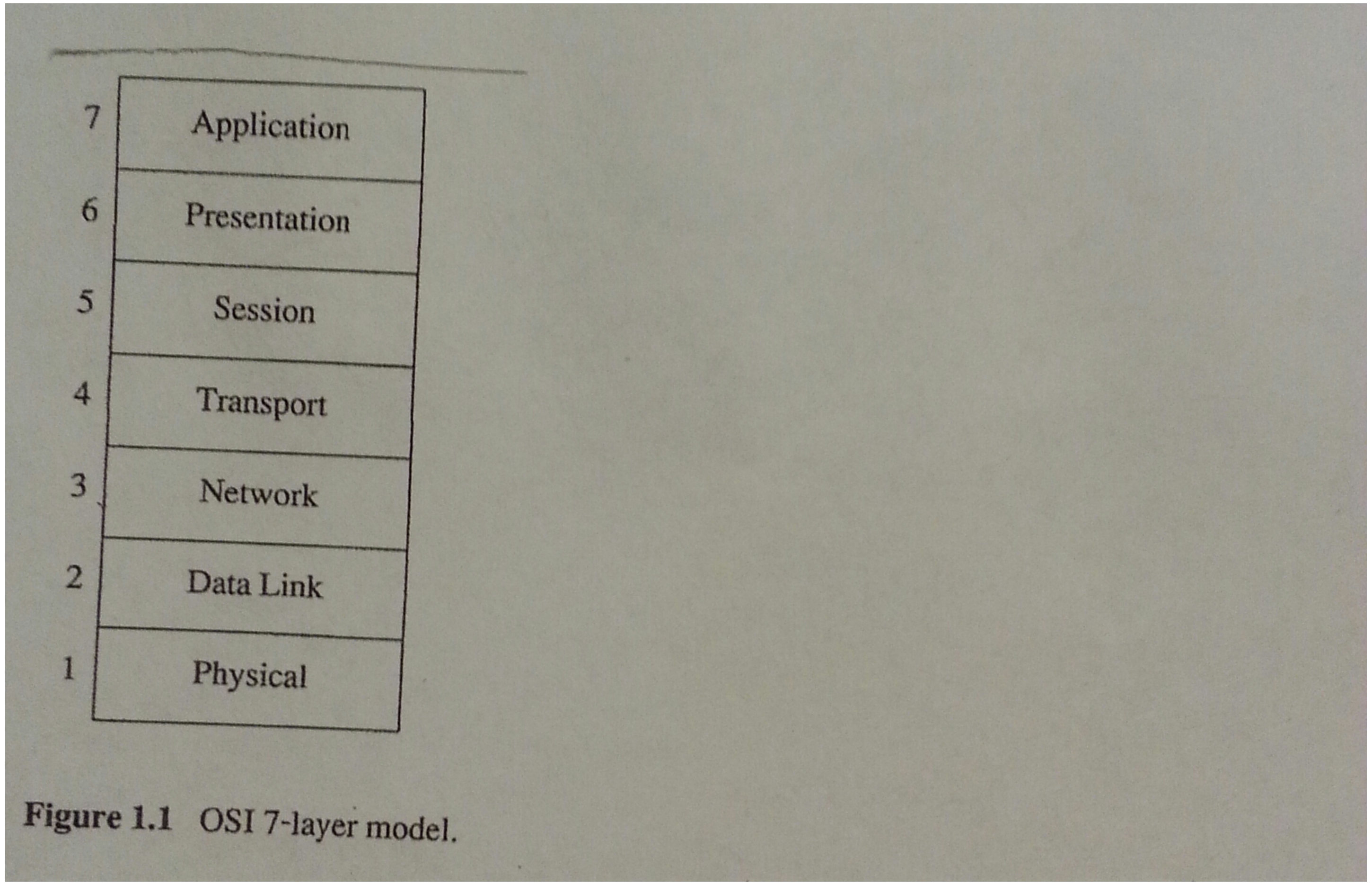


**Figure 1.1** OSI 7-layer model.

## Network Programming Review

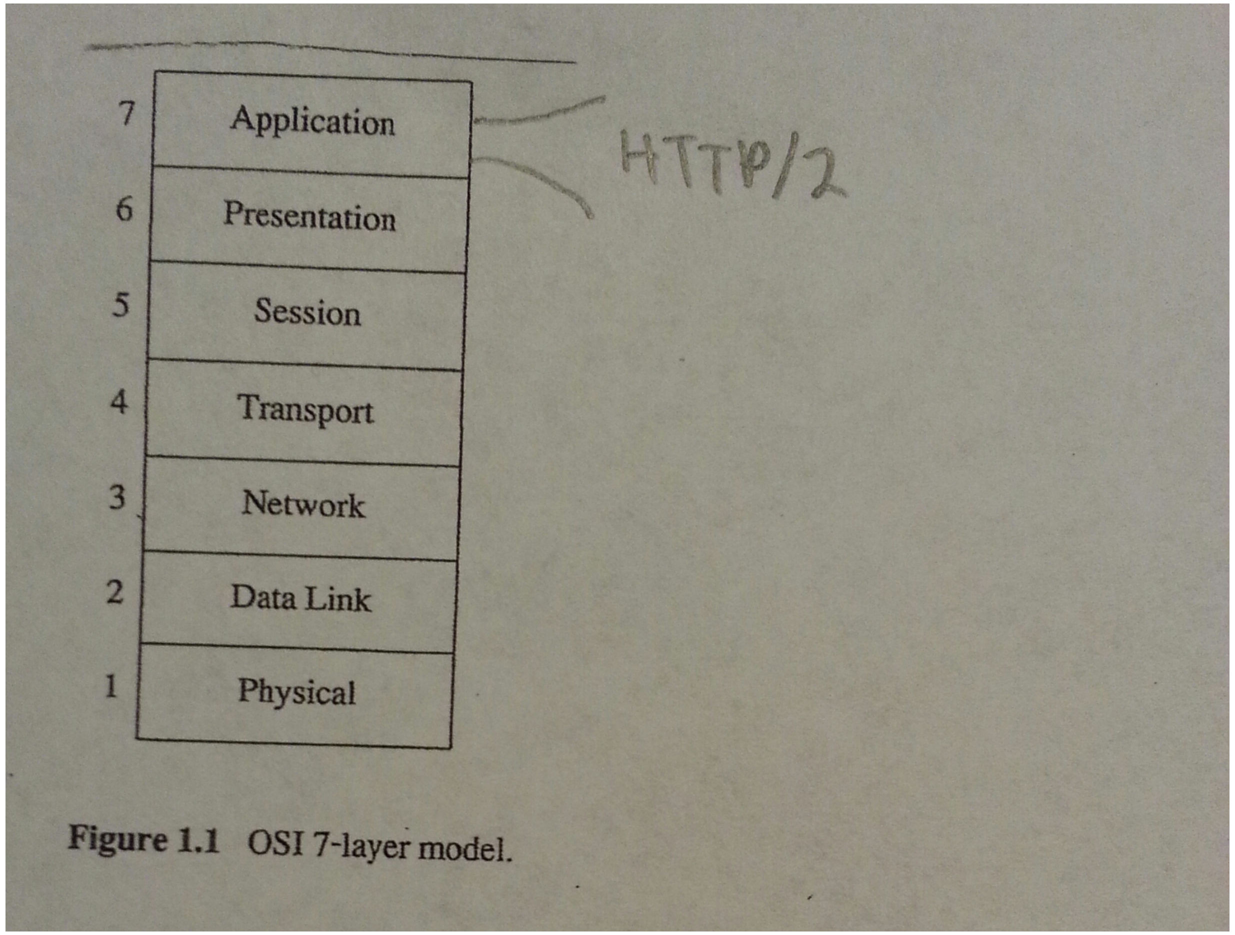


**Figure 1.1** OSI 7-layer model.

## Network Programming Review

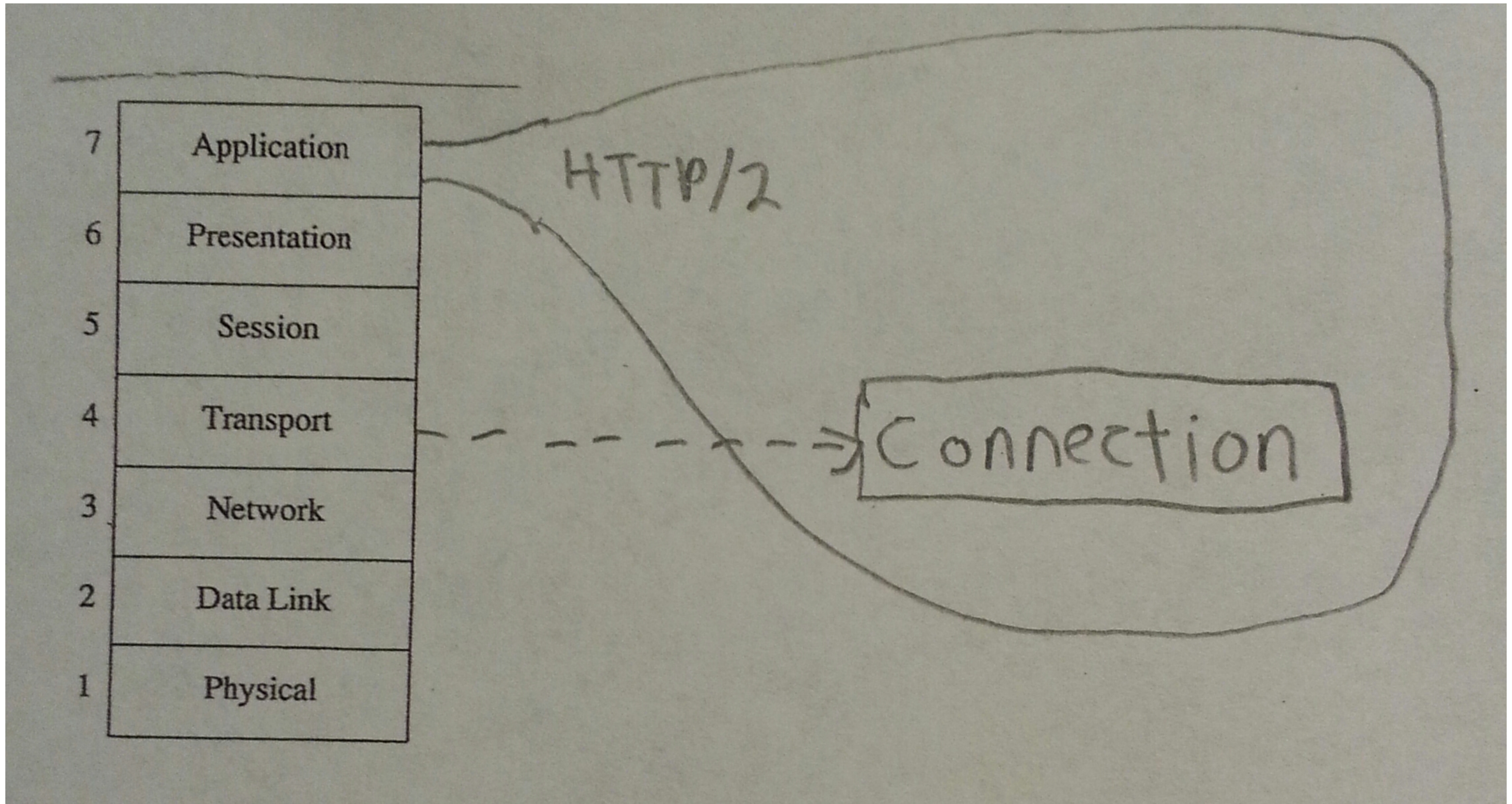


**Figure 1.1** OSI 7-layer model.

# Network Programming Review

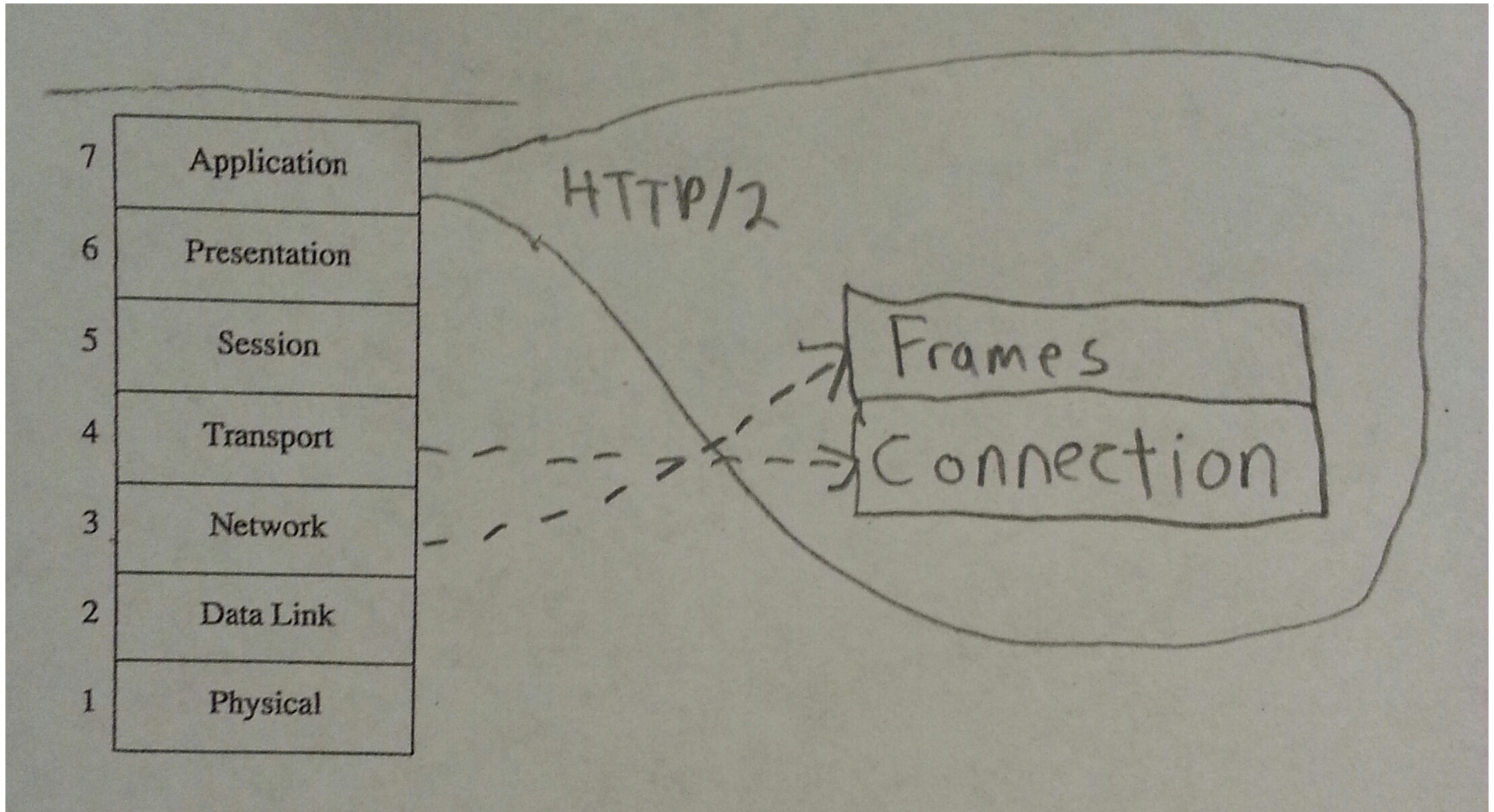


**Figure 1.1** OSI 7-layer model.

# Network Programming Review

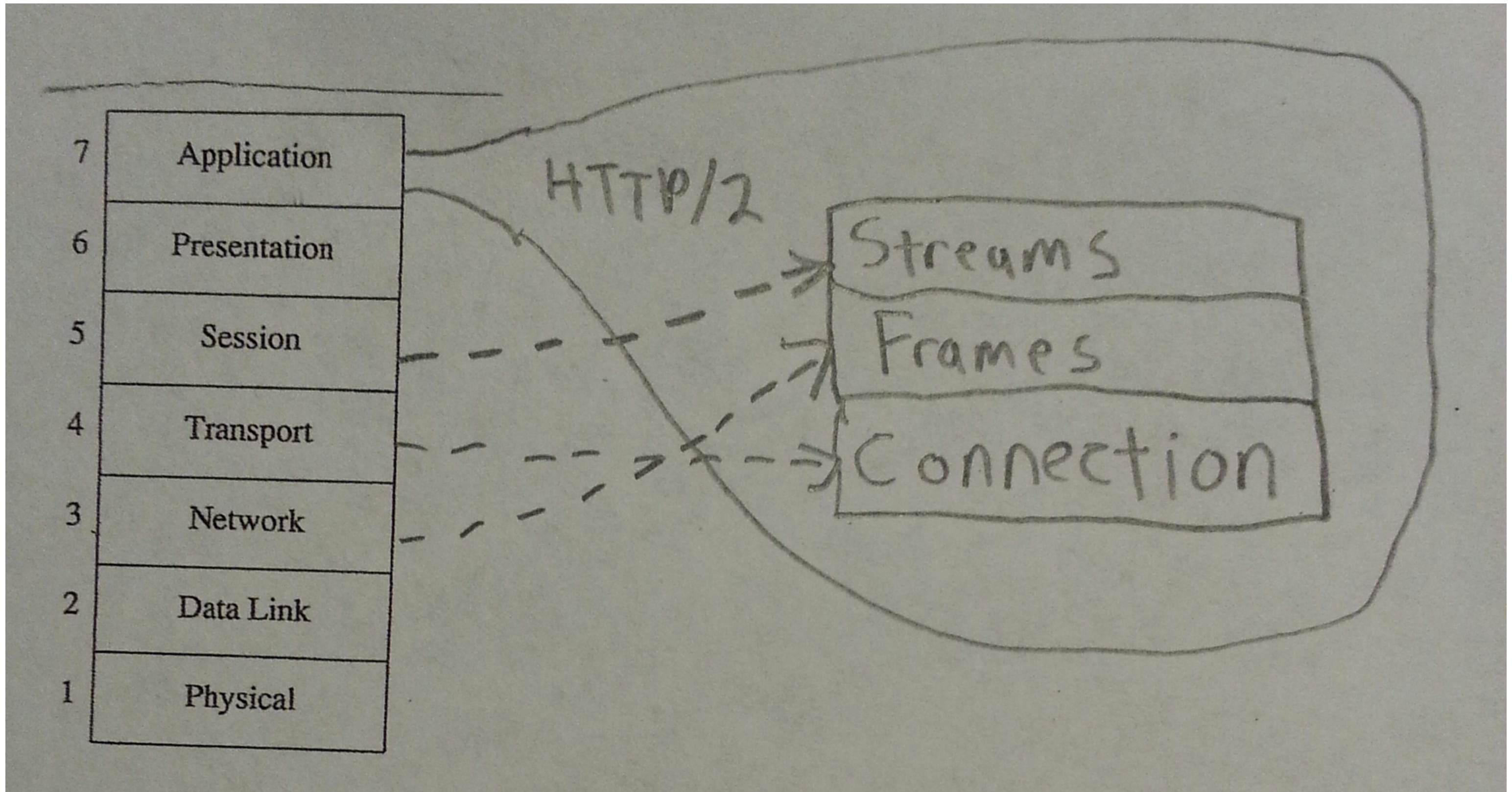


**Figure 1.1** OSI 7-layer model.

## Network Programming Review

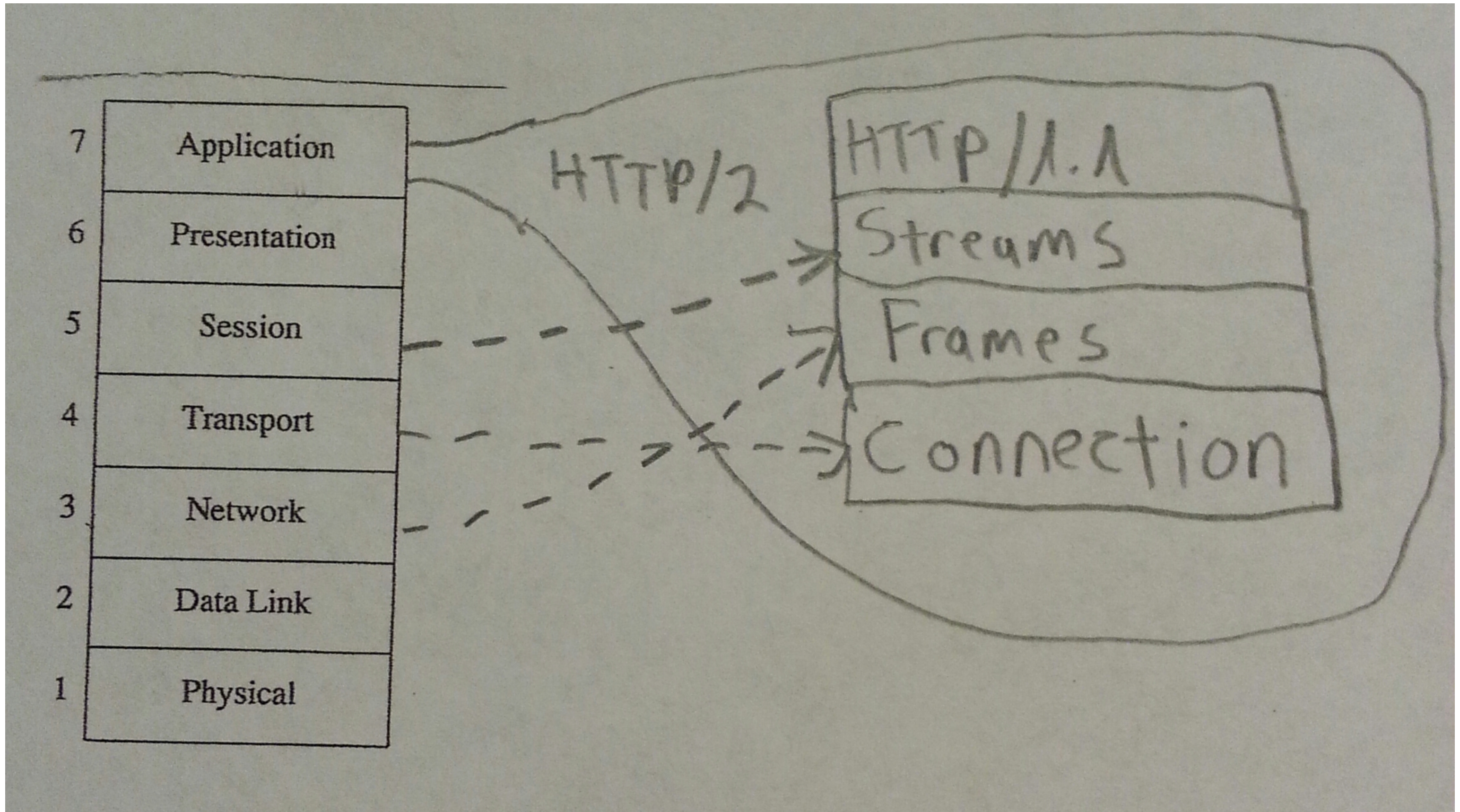


**Figure 1.1** OSI 7-layer model.

# Network Programming Review

## The Socket Angle

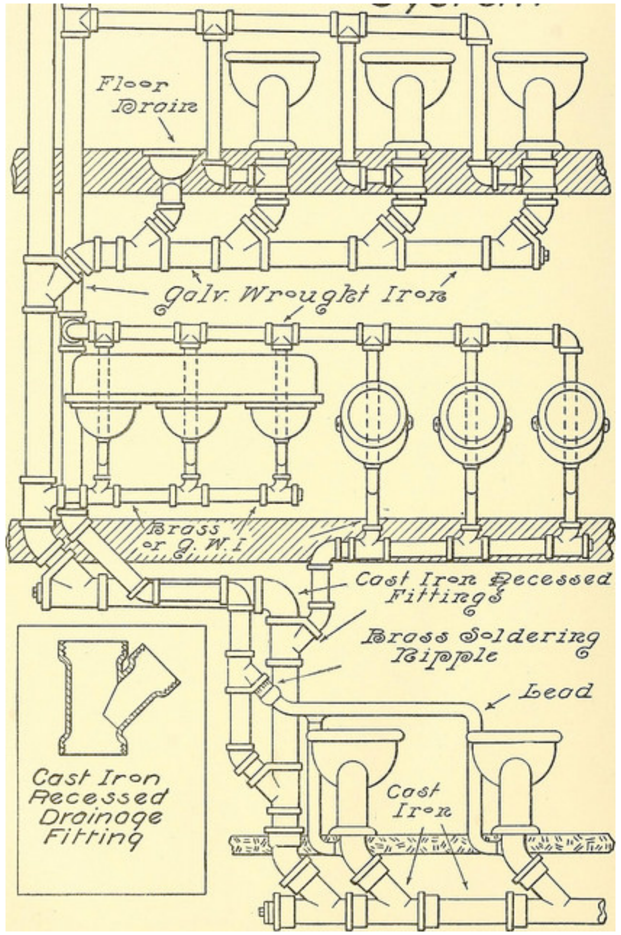
- HTTP/1.0
  - Sockets are a throwaway resource
  - Specification says very little about how sockets are to be used
  - Browsers free to open many sockets to the same server

Credit: chrisjstanley flickr

E

## Network Programming Review

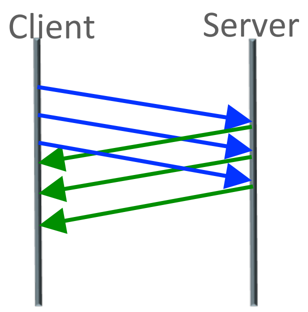
### The Socket Angle

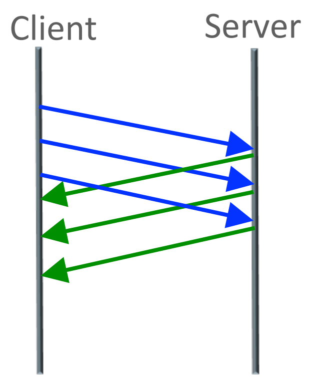


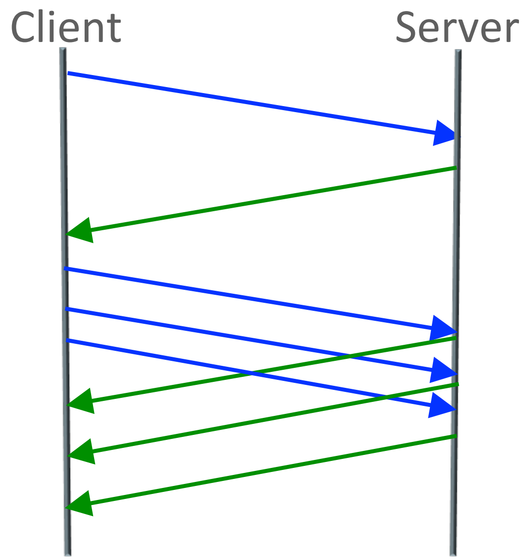
- HTTP/2
  - Sockets seen as a scarce resource
  - Specification says much about how they are to be used
  - Only one open per server

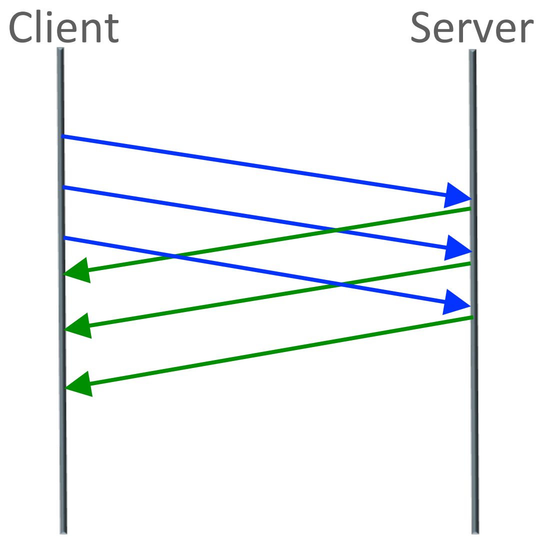
# Network Programming Review

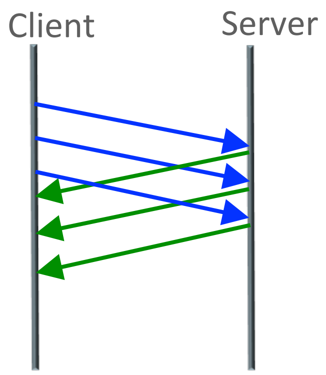
## The Socket Angle

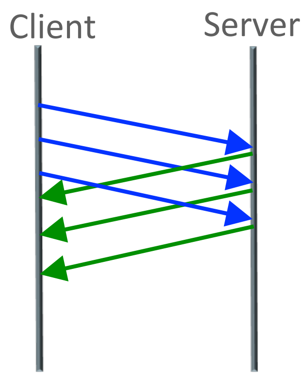






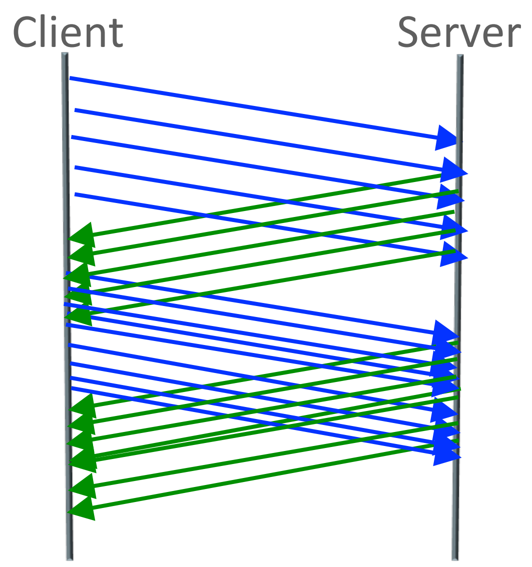






# Network Programming Review

## Solution in HTTP/2



## Network Programming Review

**The Adoption Angle**

- HTTP/1.0 was designed to be easy to implement with contemporary development practices of 1991
  - text based protocol
  - leaves flow control to the TCP layer
  - easy to write a parser
  - simple socket lifecycle

E

## Network Programming Review

**The Adoption Angle**

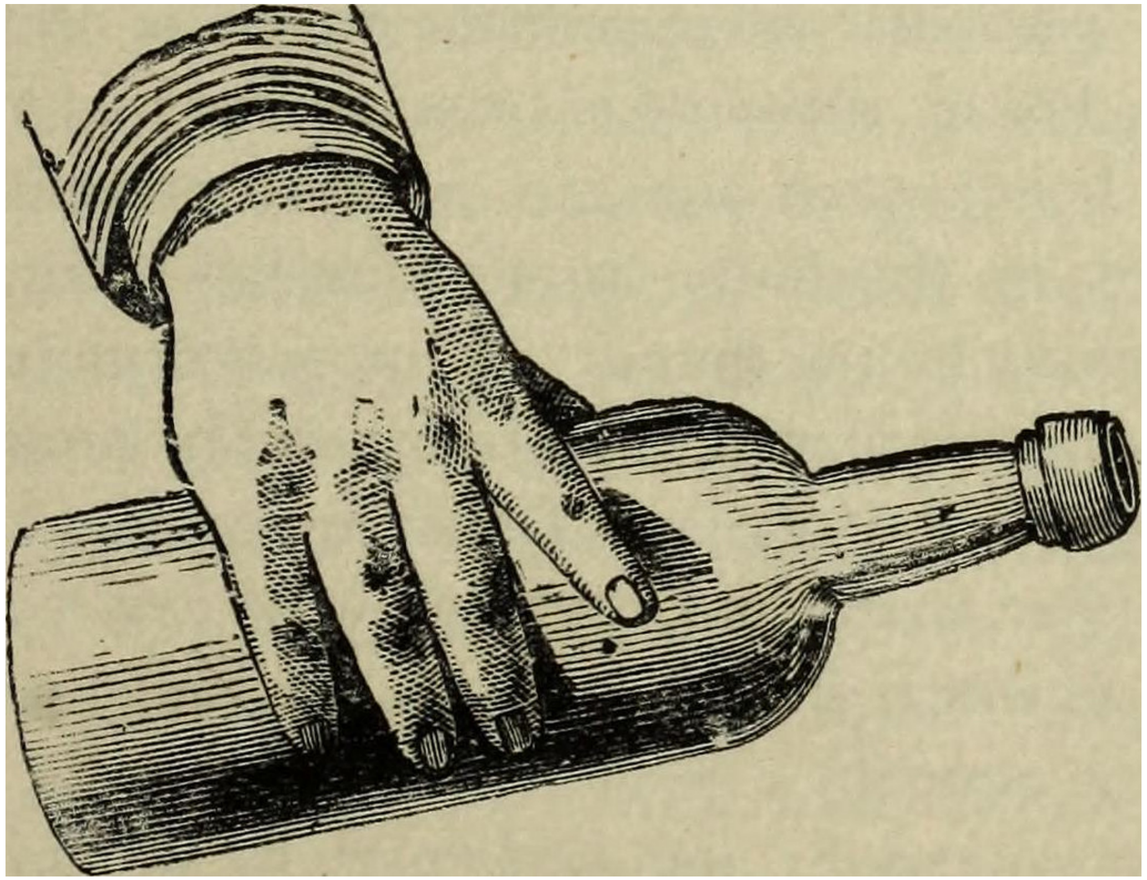
- HTTP/2 is much more complicated to implement
  - state machine
  - header compression
  - binary framing (arguably easier than text based for parsing)

E

## Network Programming Review

**The Adoption Angle**

- HTTP/2 is much more complicated to implement
  - No more
    `telnet host 80`
    GET /somepage.html \r\n\r\n



## Network Programming Review

### The Adoption Angle

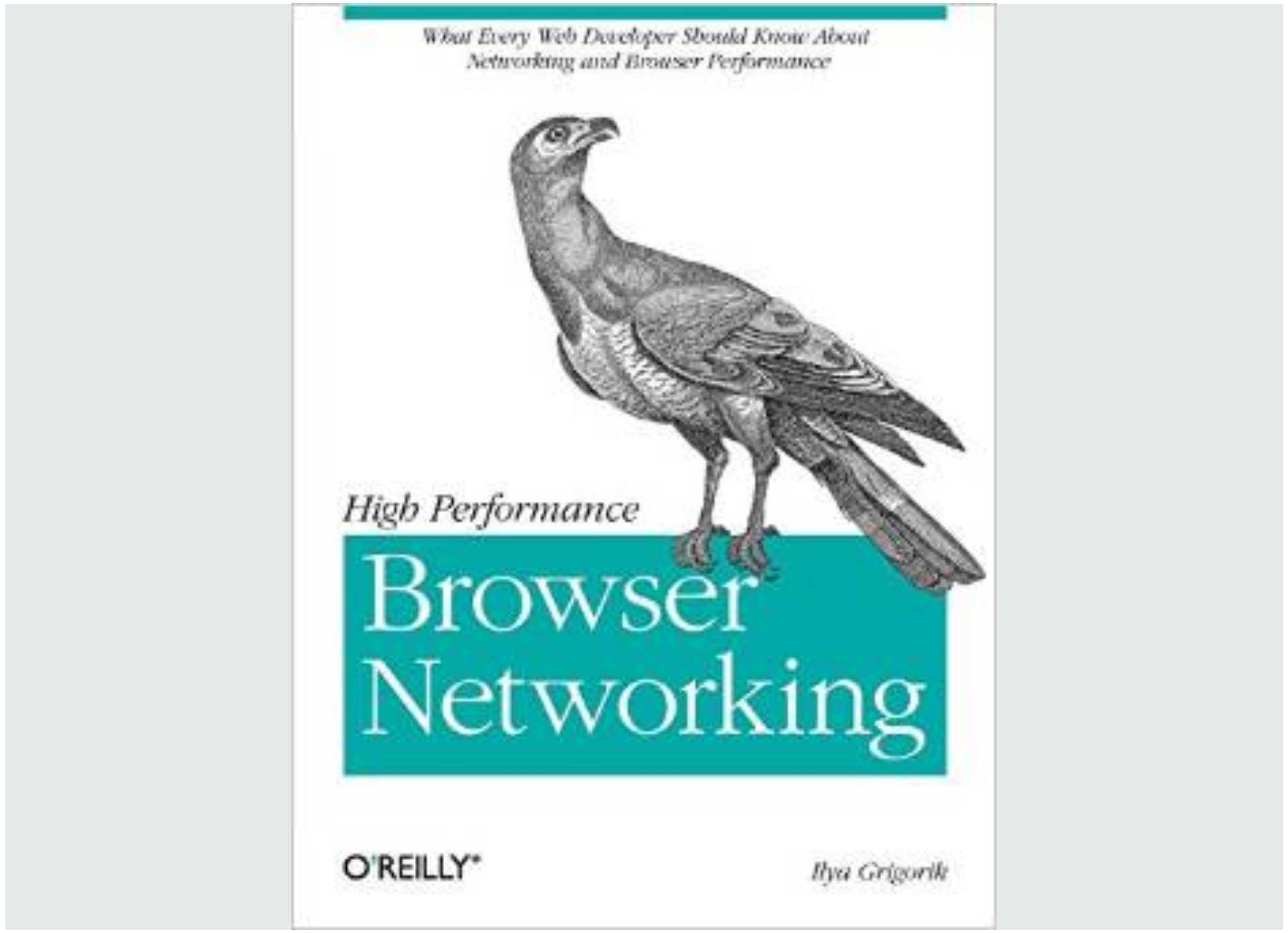
- HTTP/2 is much more complicated to implement
  - No more
    `telnet host 80`
    GET /somepage.html \r\n\r\n

## HTTP/2 Big Ticket Feature Review

- Request/Response multiplexing
- Binary Framing
- Stream Prioritization
- Server Push
- Header Compression
- Upgrade from HTTP/1.1
  - ALPN
  - 101 Switching Protocols

E

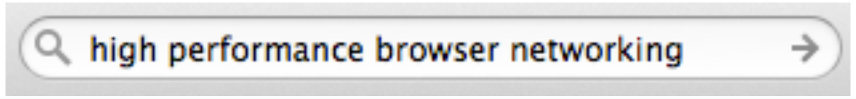
# HTTP/2 Big Ticket Feature Review


What Every Web Developer Should Know About Networking and Browser Performance

High Performance Browser Networking

O'REILLY

Ilya Grigorik


high performance browser networking

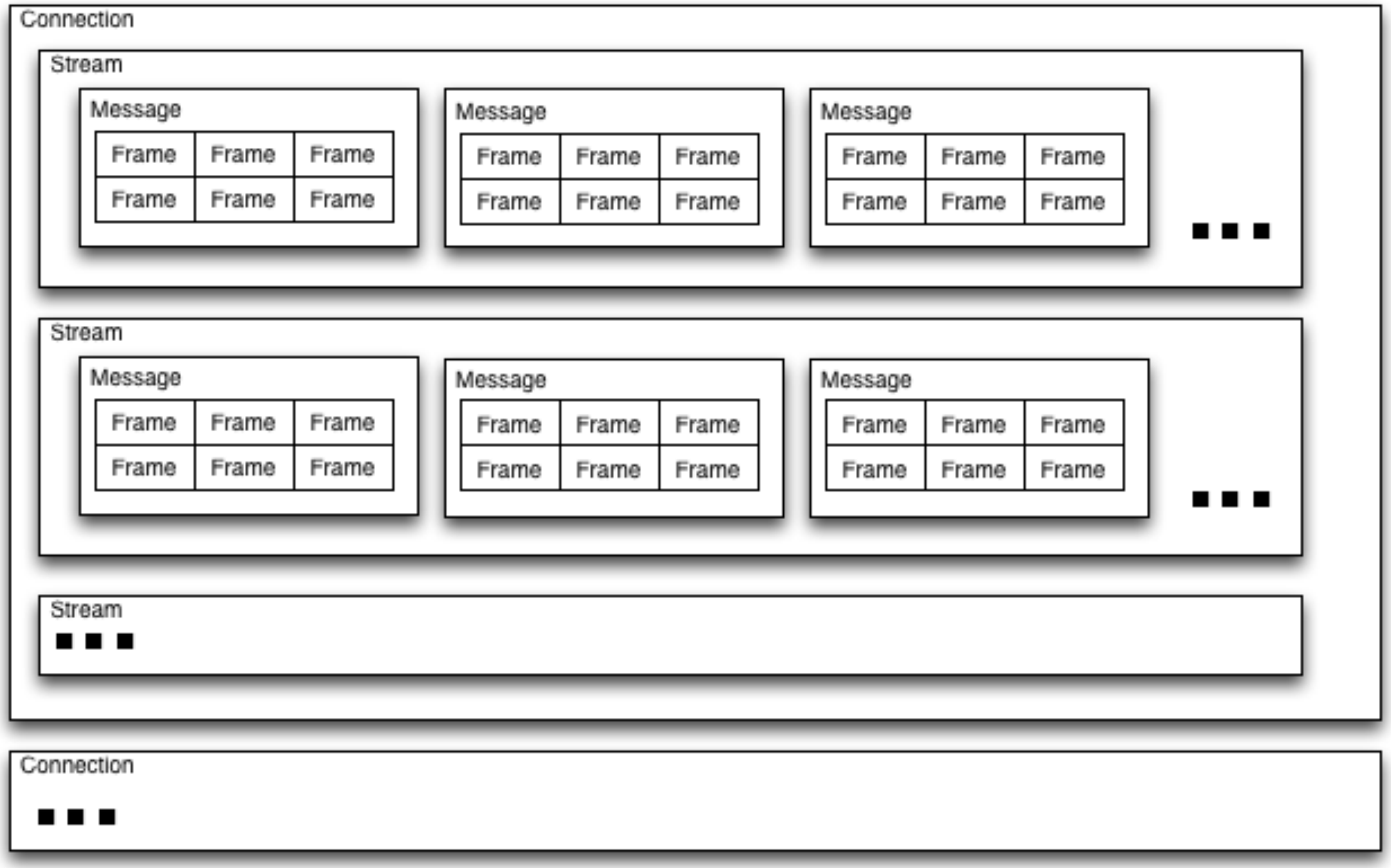
## HTTP/2 Request Response Multiplexing

**Lets you do more things with a single TCP connection**

- Fully bi-directional
- Enabled by defining some terms
  - *Connection*
    A TCP socket
  - *Stream*
    A “channel” within a connection
  - *Message*
    A logical message, such as a request or a response
  - *Frame*
    The smallest unit of communication in HTTP/2.

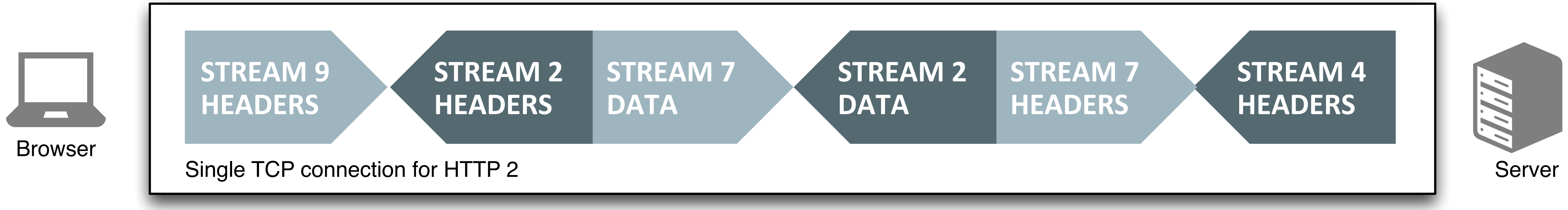
## HTTP/2 Request Response Multiplexing

### Connections, Streams, Messages, Frames



## HTTP/2 Request Response Multiplexing

**Connections, Streams, Messages, Frames**

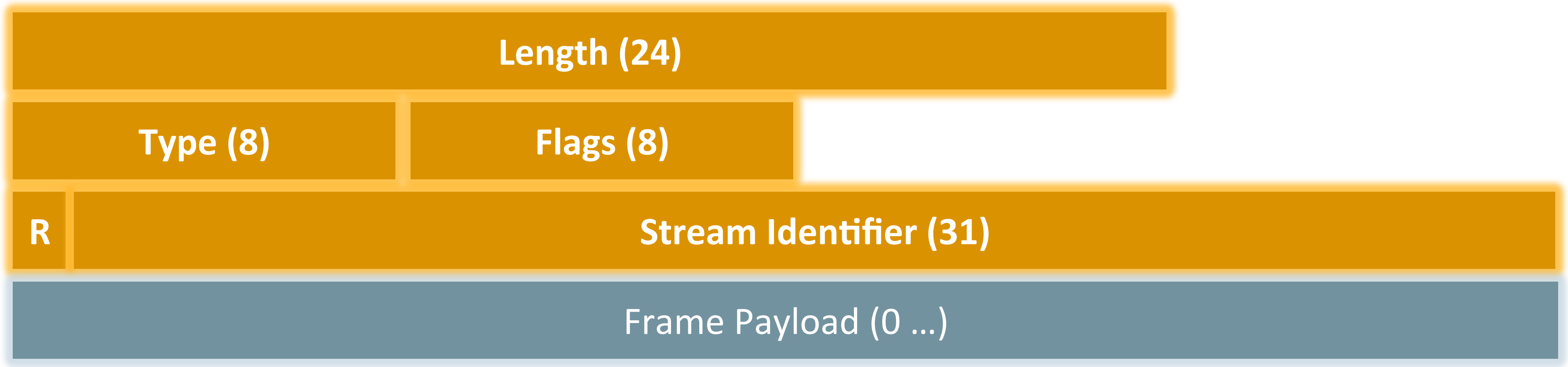


- Once you break the communication down into frames, you can interweave the logical streams over a single TCP connection.
- Yet another idea from the 1960s is new again.

# HTTP/2 Binary Framing

**Enabled by dumping newline delimited ASCII**

- Solves Head-Of-Line (HOL) blocking problem



- Type field can be DATA, HEADERS, PRIORITY, RST_STREAM, SETTINGS, PUSH_PROMISE, PING, GOAWAY, WINDOW_UPDATE, CONTINUATION

## HTTP/2 Binary Framing

**Example 1**

```
GET /index.html HTTP/1.1
Host: example.com
Accept: text/html
```

```
HEADERS
  + END_STREAM
  + END_HEADERS
    :method: GET
    :scheme: http
    :path: /index.html
    :authority: example.org
    accept: text/html
```

## HTTP/2 Binary Framing

**Example 2**

```
HTTP/1.1 200 OK
Content-Length: 11
Content-Type: text/html

Hello World
```

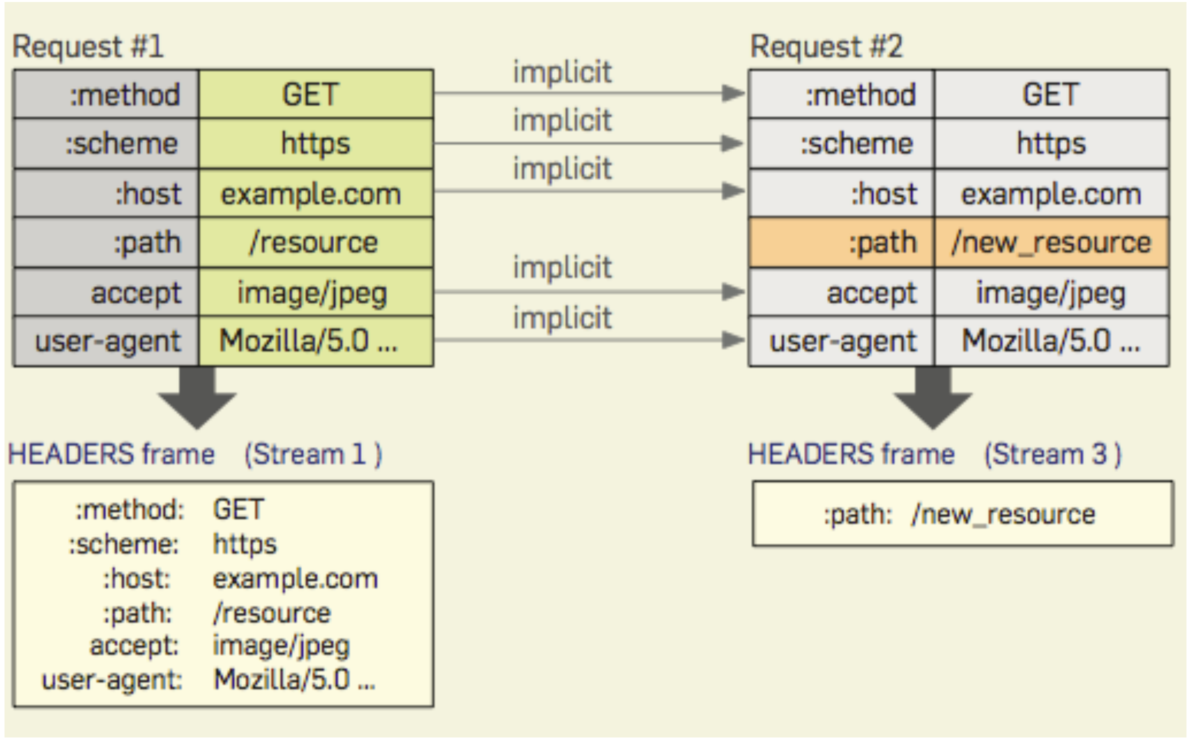
```
HEADERS
   - END_STREAM
   + END_HEADERS
      :status: 200
      content-length: 11
      content-type: text/html

DATA
   + END_STREAM
Hello World
```

## HTTP/2 Header Compression

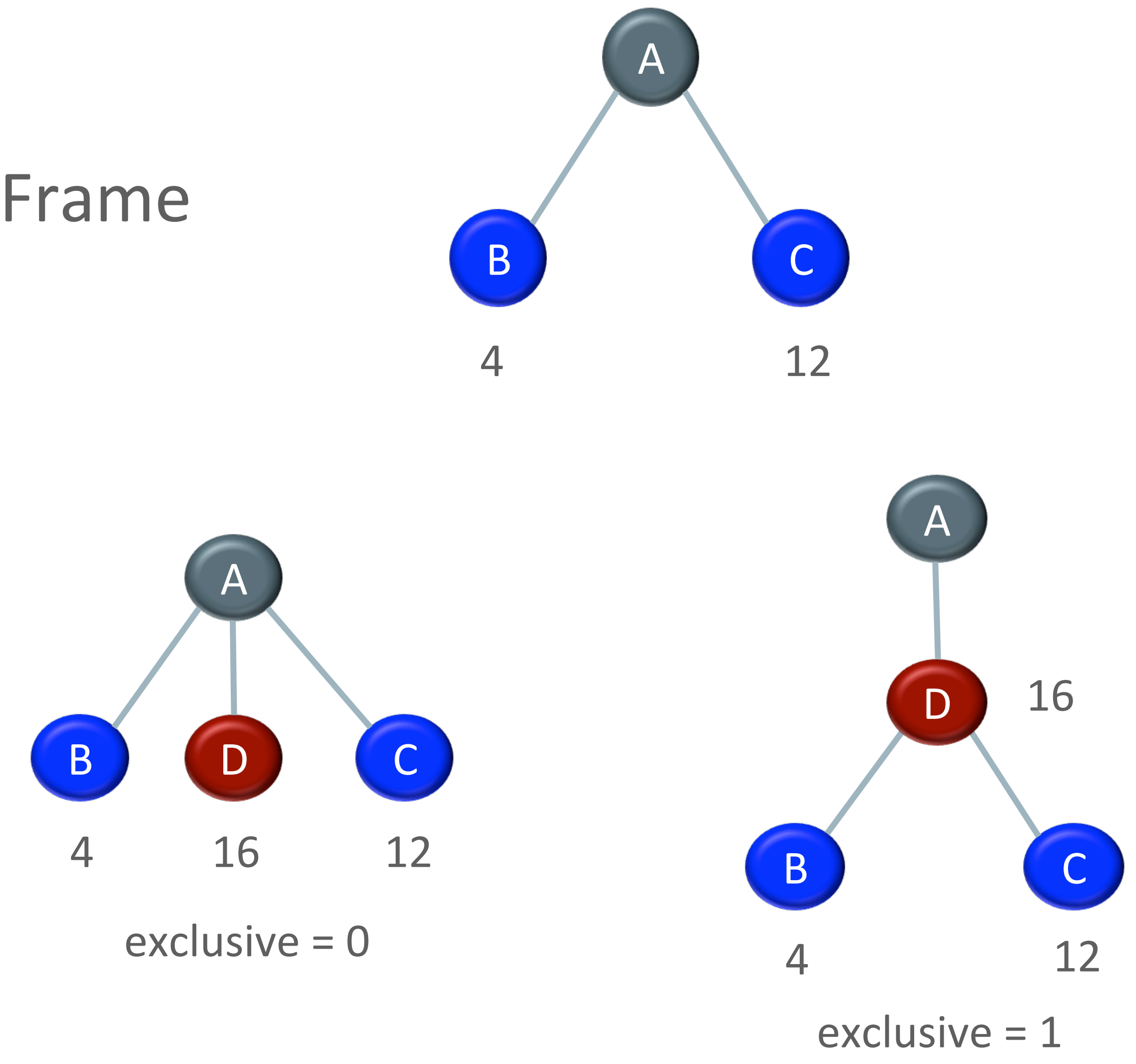
**Known as HPACK**

- Observation: most of the headers are the same in a given connection
  - Host: Accept: user-agent: etc.
- Why send them every time?
- Have the server and the client keep tables of headers, then just send references and updates to the tables.



## HTTP/2 Stream Prioritization

- Stream Dependency in HEADERS Frame
- PRIORITY frame type
- An additional 40 bytes
  - Stream id (31)
  - Weight (8): [1, 256]
  - Exclusive bit (1)
- Only a suggestion



## HTTP/2 Server Push

- Eliminates the need for resource inlining.
- Lets the server populate the browser’s cache in advance of the browser asking for the resource to put in the cache.
- No corresponding JavaScript API, but can be combined with SSE
  - Server pushes stuff into the browser’s cache.
  - Server uses SSE to tell the browser to go fetch it (but we know it’s already in the browser’s cache).

S

## HTTP/2 Upgrade from HTTP/1.1

**Secure or not-secure?**

- Not secure (h2c)
  - We have to use port 80
  - Use existing 101 Switching Protocols  from HTTP/1.1
- Secure (h2)
  - Application Layer Protocol Negotiation (ALPN)

S

## Criticism of HTTP/2

**Everybody's a Critic**

- Poul Henning-Kamp's rant just before WGLC
  - http://queue.acm.org/detail.cfm?id=2716278

## Criticism of HTTP/2

**Everybody's a Critic**

- Poul Henning-Kamp's rant just before WGLC
  - http://queue.acm.org/detail.cfm?id=2716278

## Criticism of HTTP/2

**Everybody's a Critic**

- Poul Henning-Kamp's rant just before WGLC
  - http://queue.acm.org/detail.cfm?id=2716278

Credit: Michael Fritz

E

## Criticism of HTTP/2

**Everybody's a Critic**

- HOL blocking is still a problem, just shuffled around
  - HOL blocking can still happen in HEADERS frames
- No h2c in Firefox or Chrome
  - Mention the IETF RFC-7258
- Carbon footprint for all that HPACK  encoding/decoding
- Numerous new DoS attack vectors
- HTTP/2 is orthogonal to WebSocket

E

## Don't take my word for it

- HTTP/2 isn't one spec, it's two specs
  - HTTP/2 protocol
  - HPACK
- Built on top of many other specs

E

## Don't take my word for it

RFC 7540/7541

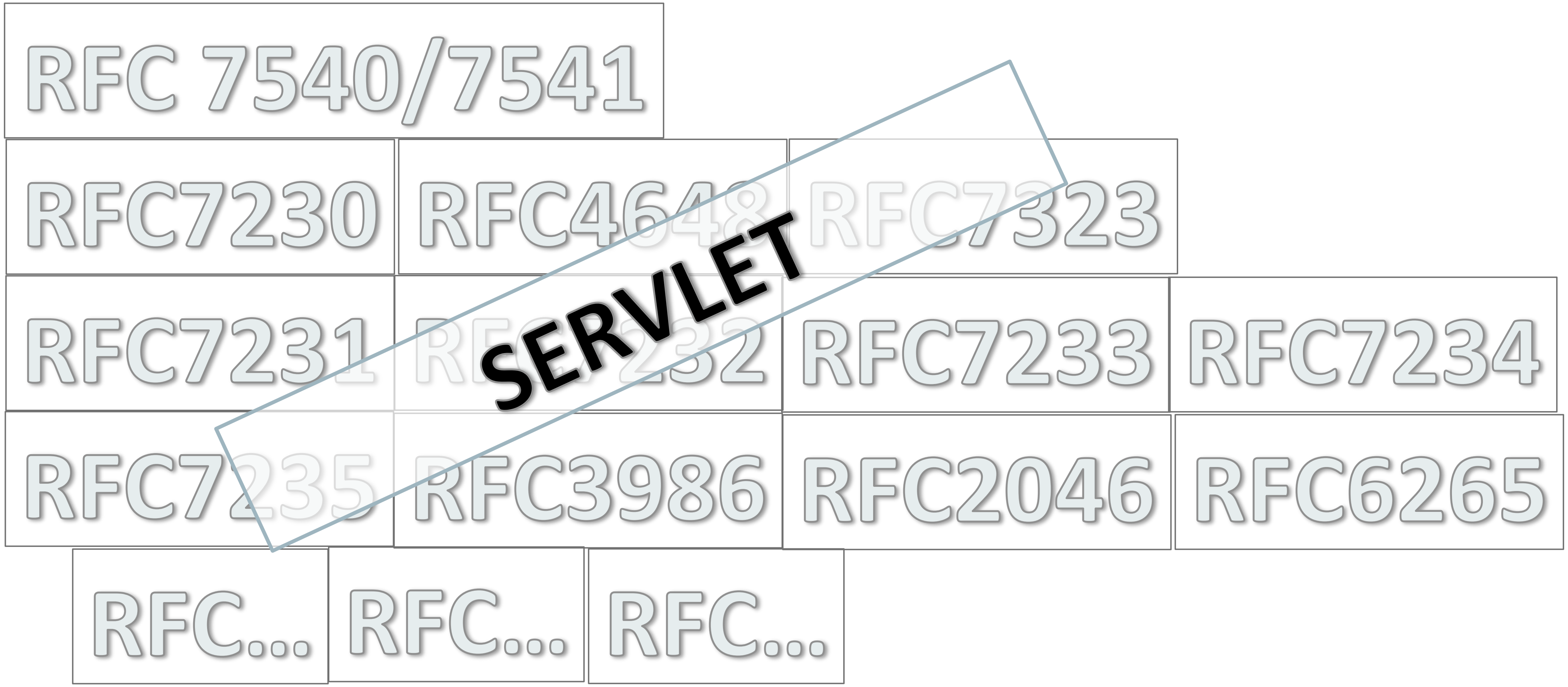
RFC7230 RFC4648 RFC7323

RFC7231 RFC7232 RFC7233 RFC7234

RFC7235 RFC3986 RFC2046 RFC6265

RFC... RFC... RFC...

## Don't take my word for it



E

## Our Plan for Your Time Investment

E

## Reactive Programming

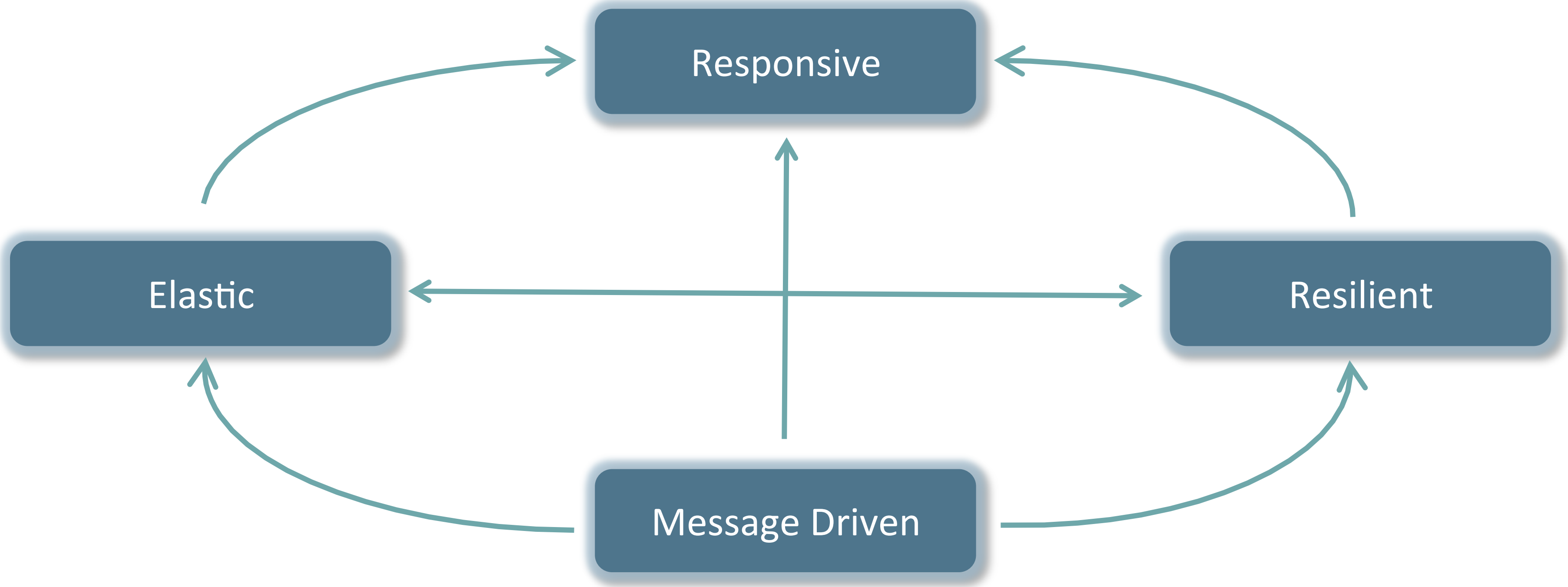
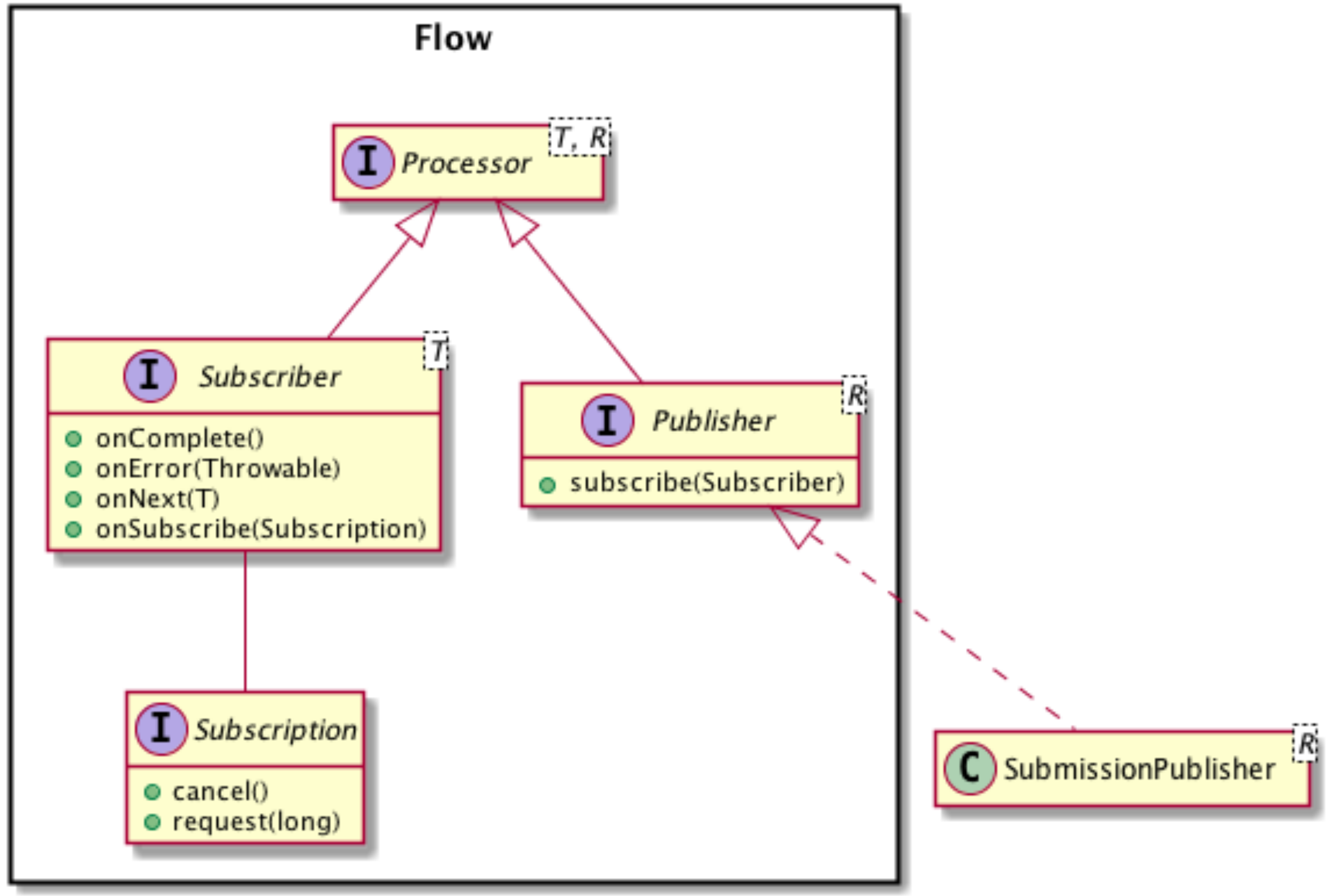
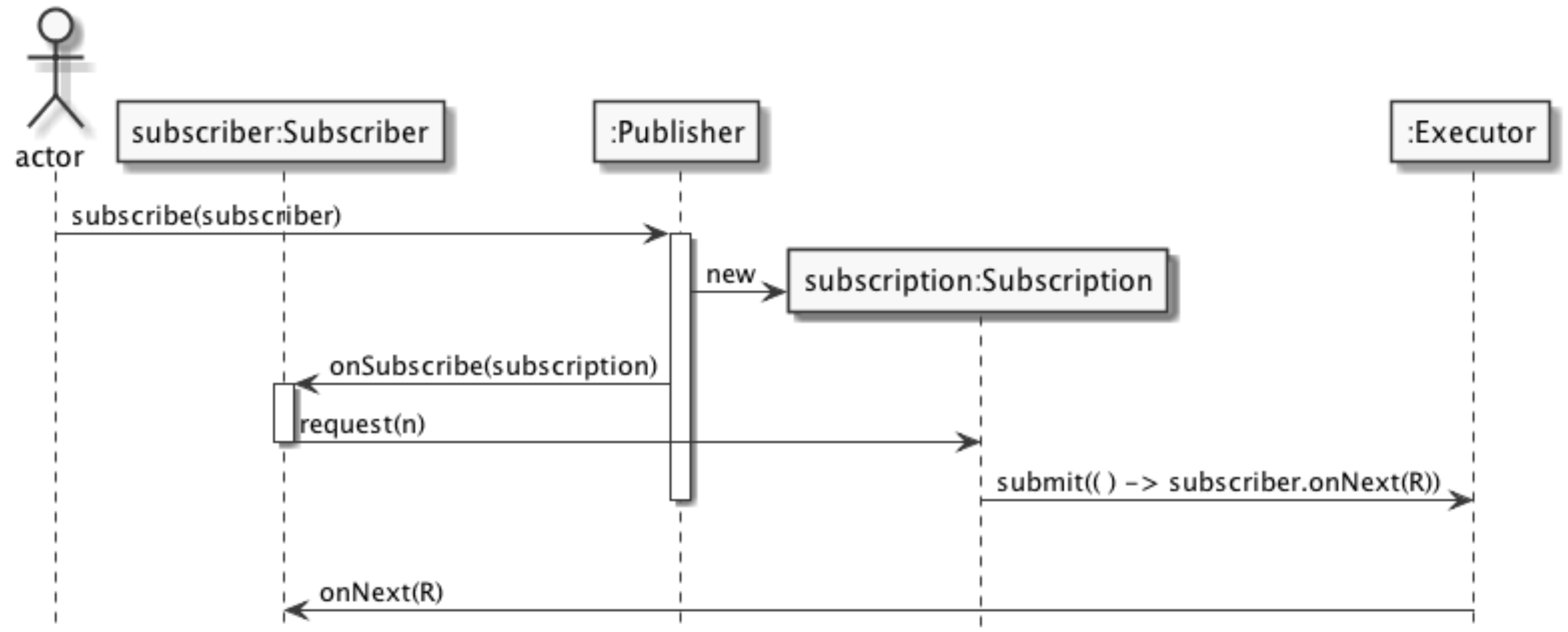


image credit: reactivemanifesto.org

# Java SE 9 Support for Reactive Programming (JEP 266)

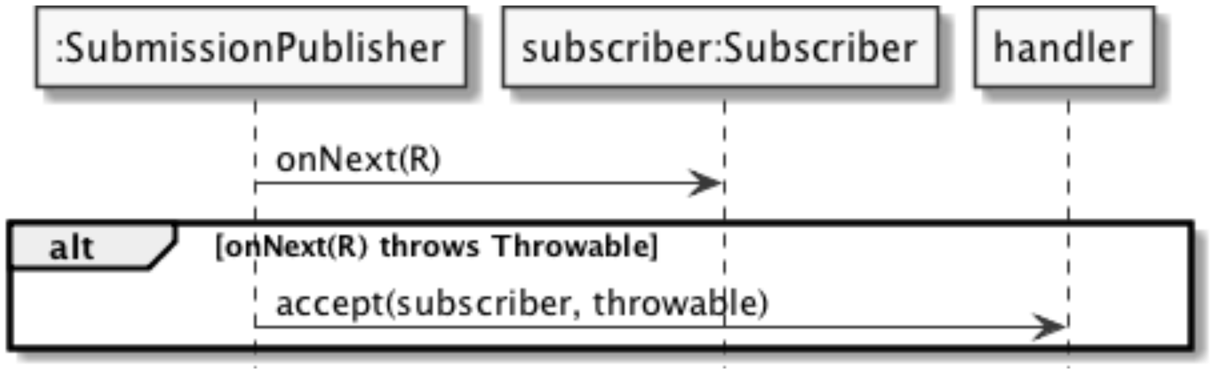


## Example


actor
subscriber:Subscriber
:Publisher
:Executor
subscribe(subscriber)
new
subscription:Subscription
onSubscribe(subscription)
request(n)
submit(( ) -> subscriber.onNext(R))
onNext(R)

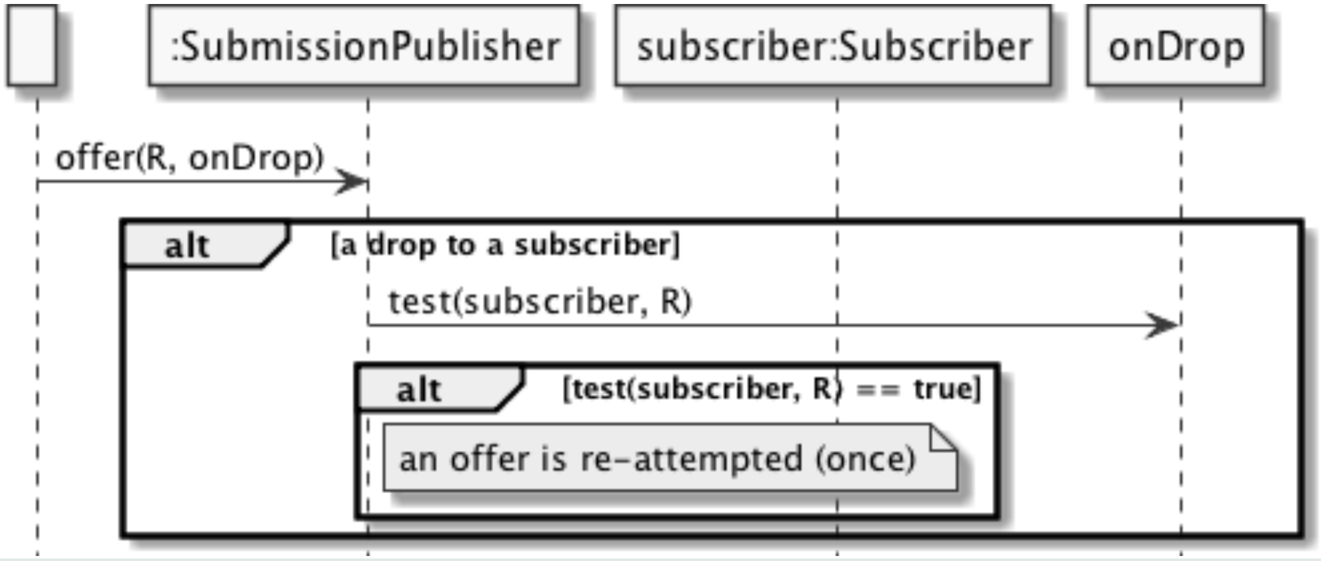
## SubmissionPublisher<R>

- Reactive-stream publishers
  - On drop handling and/or blocking for flow control
- Constructor parameters
  - Executor
  - int maxBufferCapacity for each subscriber’s buffer
  - BiConsumer<? super Flow.Subscriber<? Super R>,? super Throwable> handler



## SubmissionPublisher<R>

- Methods
  - int submit(R item)
    - result: the estimated maximum lag (number of items submitted but not yet consumed) among all current subscribers
  - int offer(R item, BiPredicate<Flow.Subscriber<? Super R>,? super R> onDrop)
    - result: if negative, the (negative) number of drops; otherwise an estimate of maximum lag
  - …



## Asynchronous in Servlet 3.0

**Async in Java EE 6!**

- `ServletRequest#startAsync`
- `AsyncContext`
  - `#addListener, #dispatch, #complete`
- `AsyncListener`
  - `#onComplete, #onError, #onStartAsync, #onTimeout`
- Event-driven
  - Server-Sent Events

## Non-blocking IO in Servlet 3.1

- `ServletInputStream`
  - `#setReadListener, #isReady`
- `ServletOutputStream`
  - `#setWriteListener, #isReady`
- `ReadListener`
  - `#onDataAvailable, #onAllDataRead, #onError`
- `WriteListener`
  - `#onWritePossible, #onError`

## Servlet 4.0 Reactive Programming on Flow Control?

- Pull delivery (epoll style)
- Push delivery
  - Explicit flow-control signals
  - Poll based flow-control

## Our Plan for Your Time Investment

## Our Plan for Your Time Investment

S

# Abstractions Endure

## Servlet API is Well Positioned to Enable HTTP/2 Optimizations

- Allow frameworks to effectively leverage server push
  - flexible strategies for leveraging push
- Leverage ALPN

## Servlet 4.0 Big Ticket New Features

**Challenges in Exposing HTTP/2 Features in Servlet API**

- Existing API is designed for One Request == One Response.
- HTTP/2 destroys this assumption.
- It will be challenging to do justice to the new reality of One Request == One or More Responses.
- We must not simply bolt the “One or More Responses” concept onto some convenient part of the existing API.

## Servlet 4.0 Big Ticket New Features

**HTTP/2 Features**

- Request/Response multiplexing
- Binary Framing
- Stream Prioritization
- Server Push
- Header Compression
- Upgrade from HTTP/1.1
  - ALPN
  - 101 Switching Protocols

## Servlet 4.0 Big Ticket New Features

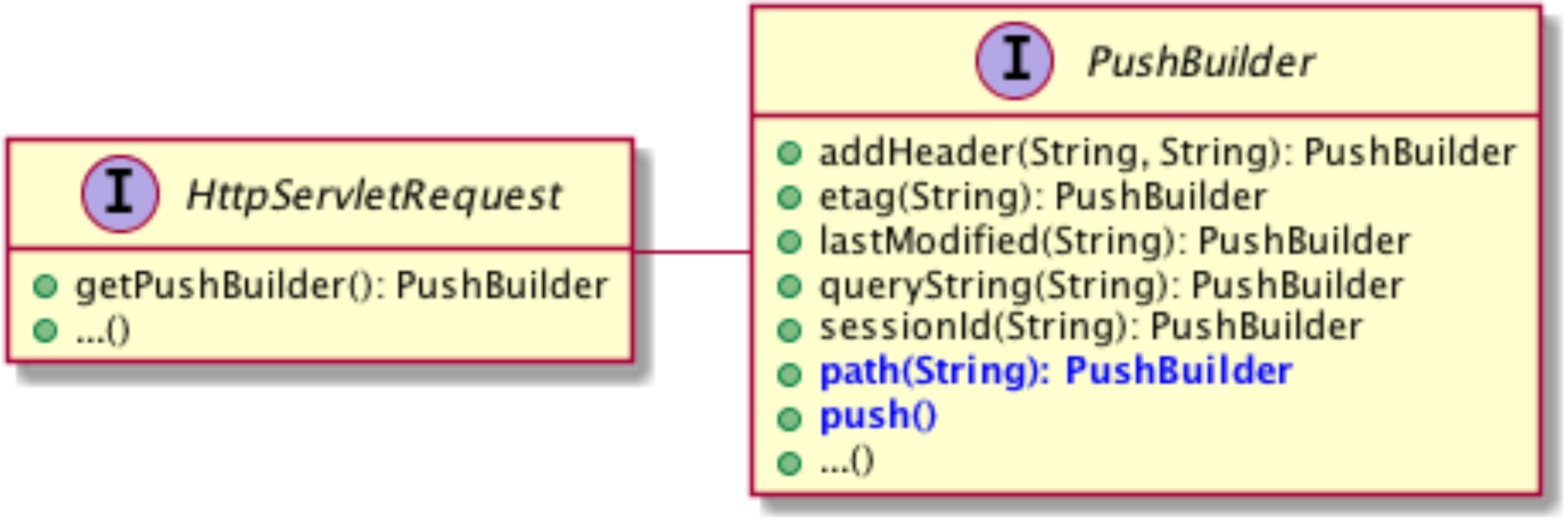
### HTTP/2 Features Potentially Exposed in Servlet API

- Request/Response multiplexing
- Binary Framing
- Stream Prioritization
- Server Push
- Header Compression
- Upgrade from HTTP/1.1
  - ALPN
  - 101 Switching Protocols

## Servlet 4.0 Big Ticket New Features

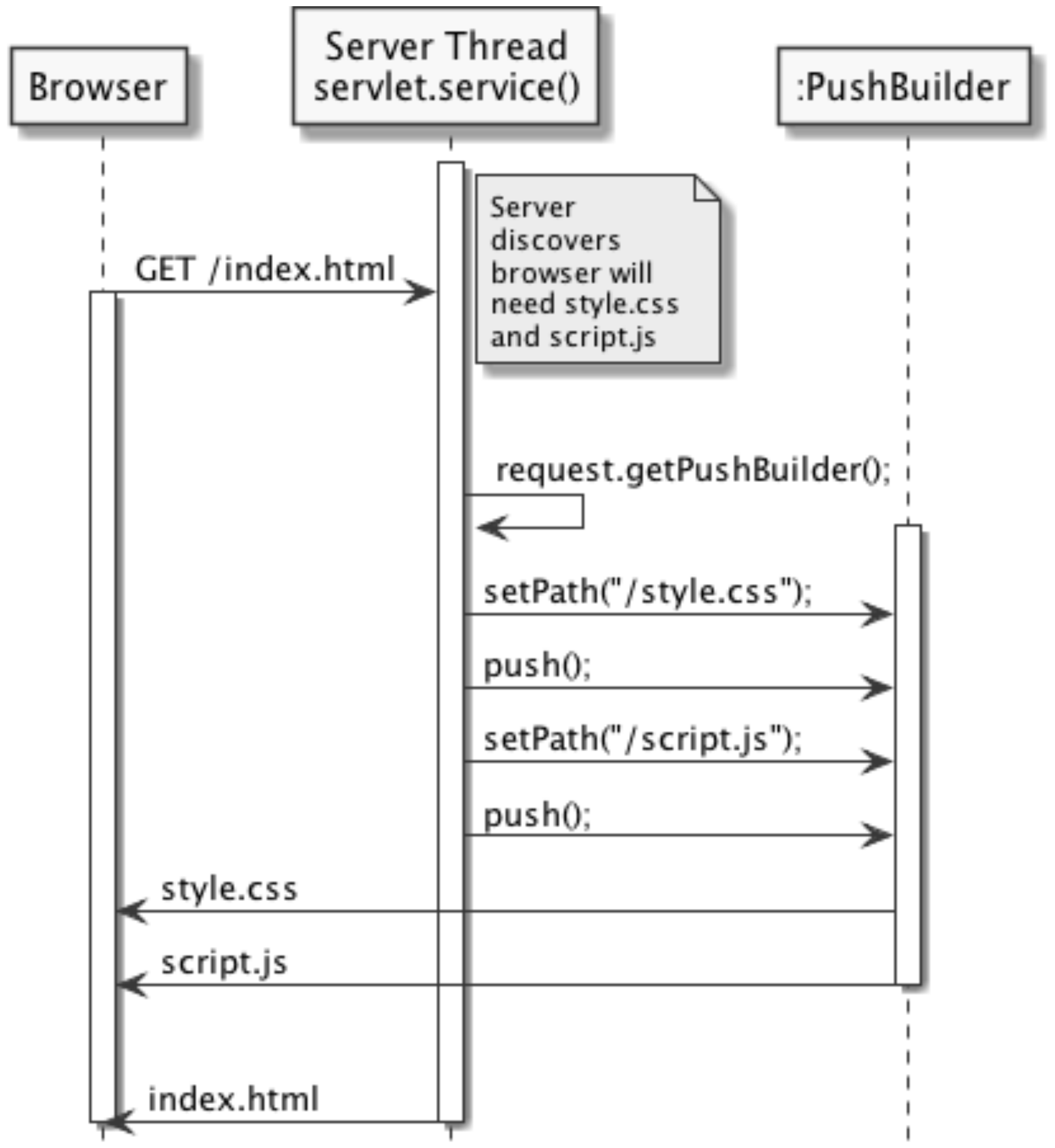
- HTTP/2 Required, including ALPN and HPACK
- HTTP/2 Server Push
  - Push resource to client for a given url and headers
  - Not at all a replacement for WebSocket
  - Really useful for frameworks that build on Servlet, such as JSF
  - Builder API
- Ease of Use

# javax.servlet.http.PushBuilder


I HttpServletRequest
getPushBuilder(): PushBuilder
...()
I PushBuilder
addHeader(String, String): PushBuilder
etag(String): PushBuilder
lastModified(String): PushBuilder
queryString(String): PushBuilder
sessionId(String): PushBuilder
path(String): PushBuilder
push()
...()

## Servlet 4.0 Big Ticket New Features

### Server Push via Builder API



## Server Push

**Example of Potential Use from JSF**

Stock FacesServlet

```
public class FacesServlet implements Servlet {
public void service(ServletRequest req,
                    ServletResponse resp) throws IOException, ServletException {
    //..
    HttpServletRequest request = (HttpServletRequest) req;
    try {
        ResourceHandler handler =
            context.getApplication().getResourceHandler();
        if (handler.isResourceRequest(context)) {
            handler.handleResourceRequest(context);
        } else {
            lifecycle.attachWindow(context);
            lifecycle.execute(context);
            lifecycle.render(context);
        }
    }
}
```

## Server Push

### Example of Potential Use from JSF

```
public class ExternalContextImpl extends ExternalContext {
    //…
    public String encodeResourceURL(String url) {
        if (null == url) {
            String message = MessageUtils.getExceptionMessageString
                (MessageUtils.NULL_PARAMETERS_ERROR_MESSAGE_ID, "url");
            throw new NullPointerException(message);
        }

        ((HttpServletRequest) request).getPushBuilder().path(url).push();
        return ((HttpServletResponse) response).encodeURL(url);
    }
    //…

}
```

## Servlet 4.0 Ease of Use

Community Feature

- Add Java SE 8 default methods
  - `ServletContextAttributeListener, ServletContextListener, ServletRequestAttributeListener, ServletRequestListener, HttpSessionActivationListener, HttpSessionAttributeListener, HttpSessionBindingListener.java, HttpSessionListener`
- Add `default` to `Filter#init`, `#destroy`
- Add `GenericFilter` and `HttpFilter`
- `<default-context-path> element in web.xml`

## Our Plan for Your Time Investment

E

## Java SE 9 Support for HTTP/2

- JEP 110 http://openjdk.java.net/jeps/110
- Easy to use API
- Covers only the most common use cases
- Supports both HTTP/1.1 and 2
- Builds on Java API classes going back to Java 1.2!

E

## Java SE 9 Support for HTTP/2

GET

HttpClient.Builder

POST

HttpRequest.Builder

HttpRequest

E

## Java SE 9 Support for HTTP/2

**Small footprint**

- A handful of classes
  - HttpClient, built by HttpClient.Builder
    - Holds information for creating one or more HttpRequests
  - HttpRequest, built by HttpRequest.Builder
    - one request/response interaction
  - HttpResponse
  - Body Processors
    - HttpRequestBodyProcessor
    - HttpResponseBodyProcessor

## Java SE 9 Support for HTTP/2

**Small footprint**

- Blocking mode: one thread per request/response
  - send request
  - get response
- Non-blocking mode
  - Using ExecutorService and CompletableFuture
- Full support for HTTP/2 Server Push

## Java SE 9 Support for HTTP/2

```
HttpResponse response = HttpRequest
           .create(new URI("http://www.foo.com"))
               .headers("Foo", "foovalue", "Bar", "barvalue”)
               .GET()
               .response();

      int statusCode = response.statusCode();
      String responseBody = response.body(asString());
```

## Java SE 9 Support for HTTP/2: Simple Case: Blocking

```
//Simple blocking -- all execution from calling thread
HttpResponse r1 = HttpRequest.create(new URI("http://www.foo.com/"))
  .GET()
  .response();
int responseCode = r1.statusCode());
String body = r1.body(asString());

HttpResponse r2 = HttpRequest.create(new URI("http://www.foo.com/"))
  .GET()
  .response();
System.out.println("Response was " + r1.statusCode());

Path body = r2.body(asFile(Paths.get("/tmp/response.txt")));
// Content stored in /tmp/response.txt
HttpResponse r3 = HttpRequest.create(new URI("http://www.foo.com/"))
  .body(fromString("param1=1, param2=2"))
  .POST()
  .response();
Void body = r3.body(ignoreBody()); // body is Void in this case
```

E

## Java SE 9 Support for HTTP/2: Async Case

```
List<URI> targets = ... // fetch list of URIs async and store in Files
List<CompletableFuture<File>> futures = targets.stream()
  .map(target -> {
    return HttpRequest
      .create(target).GET().responseAsync().thenCompose(response -> {
        Path dest = Paths.get("base", target.getPath());
        if (response.statusCode() == 200) {
          return response.bodyAsync(asFile(dest));
        } else {
          return CompletableFuture.completedFuture(dest);
        }
      })
      // convert Path -> File
      .thenApply((Path dest) -> {
        return dest.toFile();
      });
    }).collect(Collectors.toList());
// all async operations waited for here
CompletableFuture.allOf(futures.toArray(new CompletableFuture<?>[0])).join();
```

## Java SE 9 Support for HTTP/2

**HTTP/2 features**

- Negotiation of HTTP/2 from 1.1
  - ALPN or plaintext
- Server Push
  - Support for PUSH_PROMISE frames
- HPACK parameters

E

## Our Plan for Your Time Investment

## Summary and Current Status

- Servlet 4.0 brings HTTP/2 to Java EE
  - 100% compliant implementation of HTTP/2
  - Expose key feature to the API
    - Server Push

## Summary and Current Status: HTTP/2

- December 2015 Submit HTTP/2 to IESG for consideration as a Proposed Standard **DONE**
- January 2015 Submit HTTP/2 to RFC Editor **DONE**
- May 2015 Publish HTTP/2 as an RFC 7540/7541 **DONE**

E

## Summary and Current Status

- JSR-369 formed on 22 September 2014
- Early Draft Review posted on JCP.org last week!

## How to Get Involved

- Adopt a JSR
  - http://glassfish.org/adoptajsr/
- The Aquarium
  - http://blogs.oracle.com/theaquarium/
- Java EE 8 Reference Implementation
  - http://glasfish.org

## Safe Harbor Statement

## Session Surveys

### Help us help you!!

- Oracle would like to invite you to take a moment to give us your session feedback. Your feedback will help us to improve your conference.
- Please be sure to add your feedback for your attended sessions by using the Mobile Survey or in Schedule Builder.

## Other Sessions of Interest

- The Rise of 1.0: How Reactive Streams and Akka Streams Change the JVM Ecosystem [CON3633]
  Thursday, Oct 29, 4:00 p.m. | Hilton—Continental Ballroom 4

## Integrated Cloud

Applications & Platform Services

JavaOne™

ORACLE®