Making the Web Conversational

How protocols like HTTP have evolved with modern Web applications

Vipul Mathur

November 8, 2017

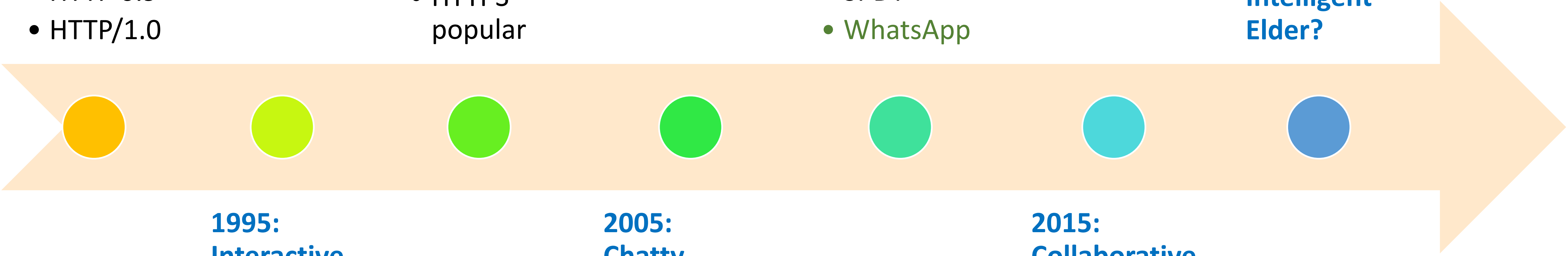
This talk is non-normative ;)

Not speaking on behalf of my employer

Overview

- Why
  - Life of the Web so far
  - Forms of communication
- What
  - Plain HTTP recap
  - Long polling and streaming
  - The WebSocket protocol
- How
  - WebSocket demo
  - Popular bi-directional Web frameworks
- Q&A

Life of The Web: Protocols, Standards, Apps



**1990:
Cute Baby**

- HTTP 0.9
- HTTP/1.0

**1995:
Interactive Child**

- HTTP/1.1
- JavaScript (Browser)

**2000:
Responsive Teenager**

- CSS2
- HTTPS popular

**2005:
Chatty Adolescent**

- JavaScript (Server)
- AJAX
- Gmail
- Google Docs

**2010:
Social Adult**

- WebSocket
- SPDY
- WhatsApp

**2015:
Collaborative Worker**

- HTTP/2
- IoT

**2020:
Intelligent Elder?**

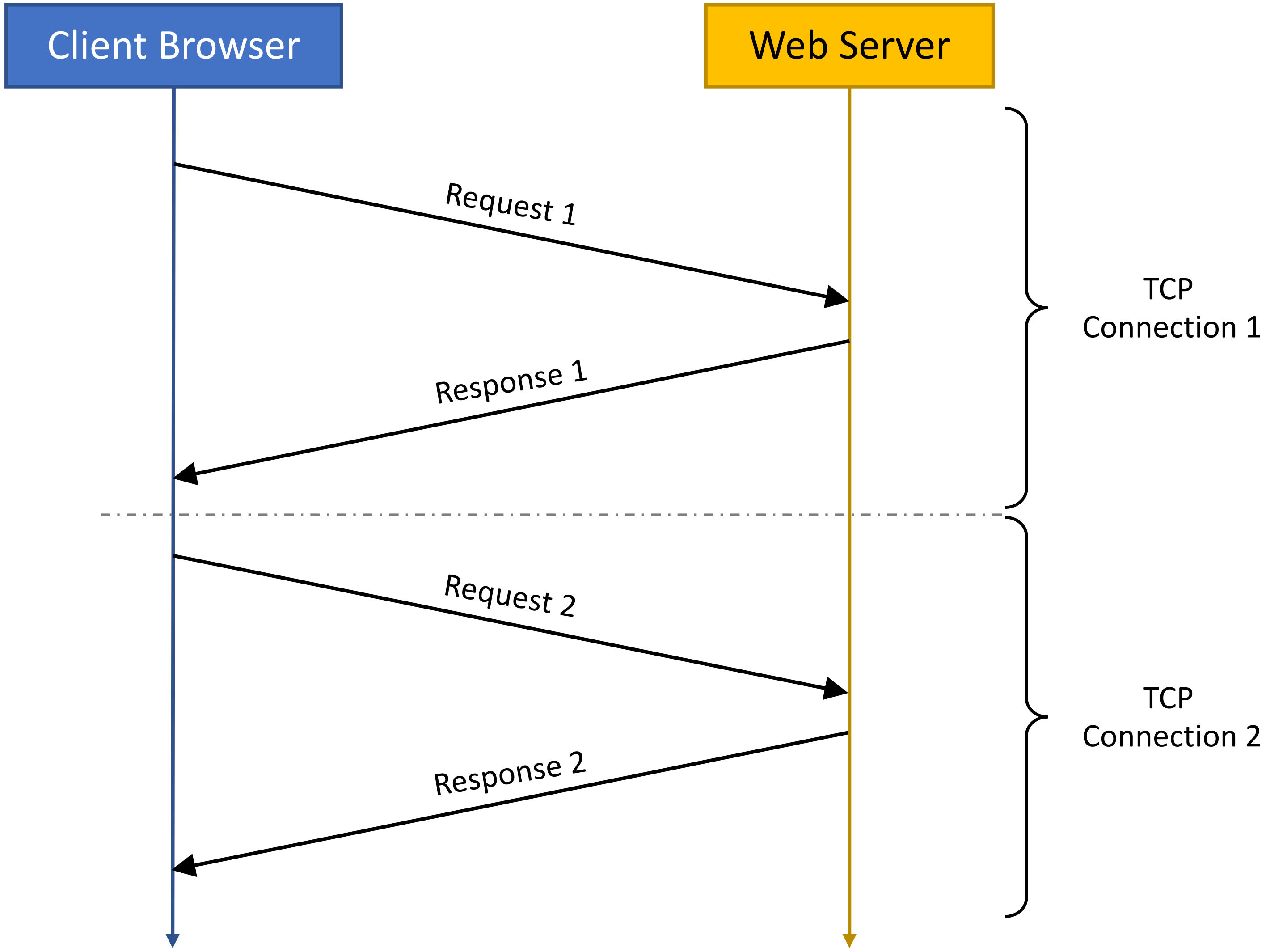
Forms of Communication

1. Unicast (1 to 1, simplex)
   - letters (snail-mail)
2. Multicast (1 to many, simplex)
   - email
3. Broadcast (1 to any, simplex)
   - TV/ radio
4. Request-response (1 to 1, half duplex)
   - Q&A like in an interview
5. Conversation (1 to 1, full duplex)
   - chat between friends
6. Omni-directional (many to many, full duplex)
   - group chat or honking in Bangalore traffic :)

- Web protocols are mostly 4, increasingly 5
- Web applications are of all types, built on top of Web protocols

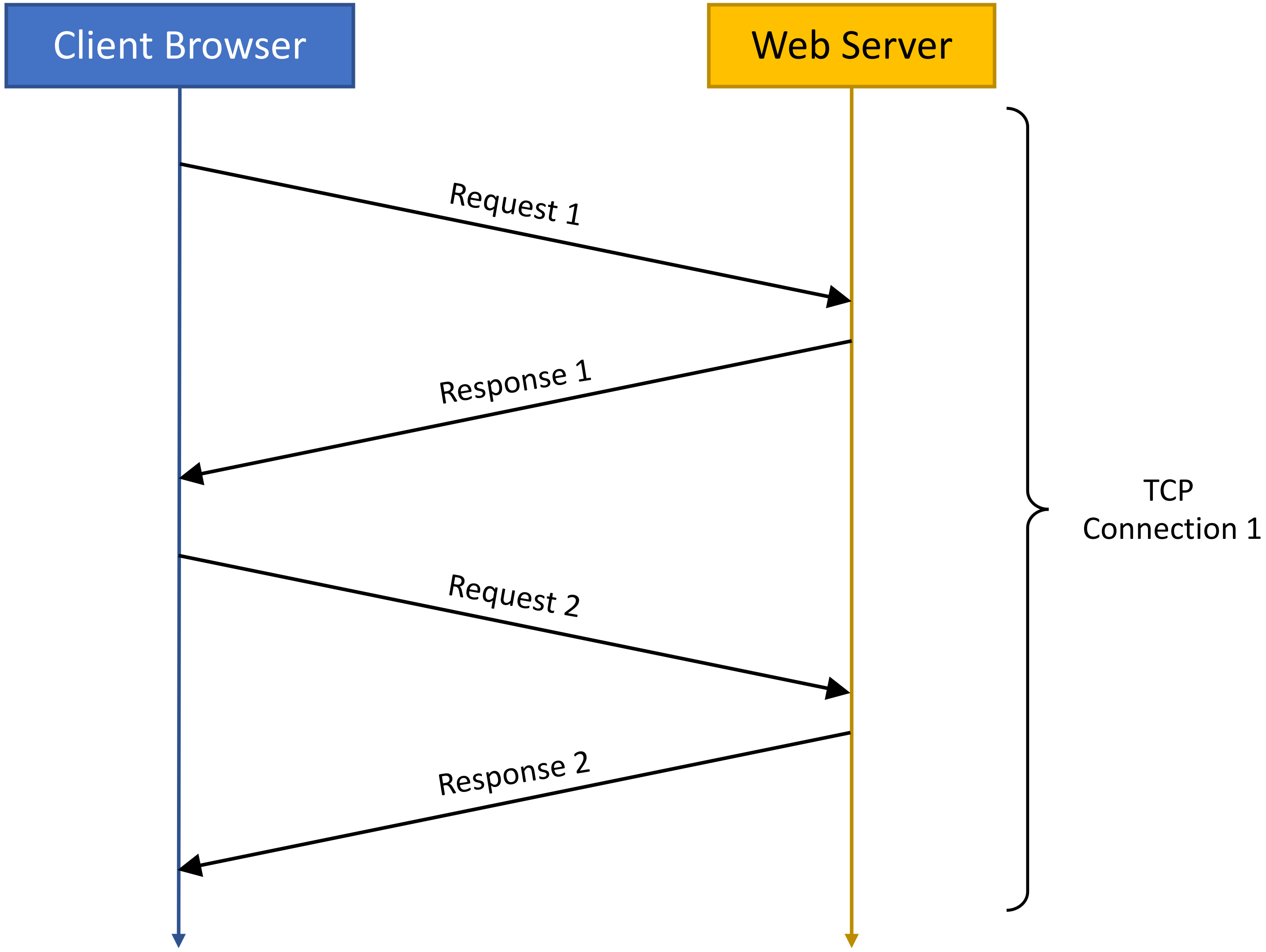
Plain HTTP Recap

- Request-response based
- Plain-text headers and body
  - human and machine readable



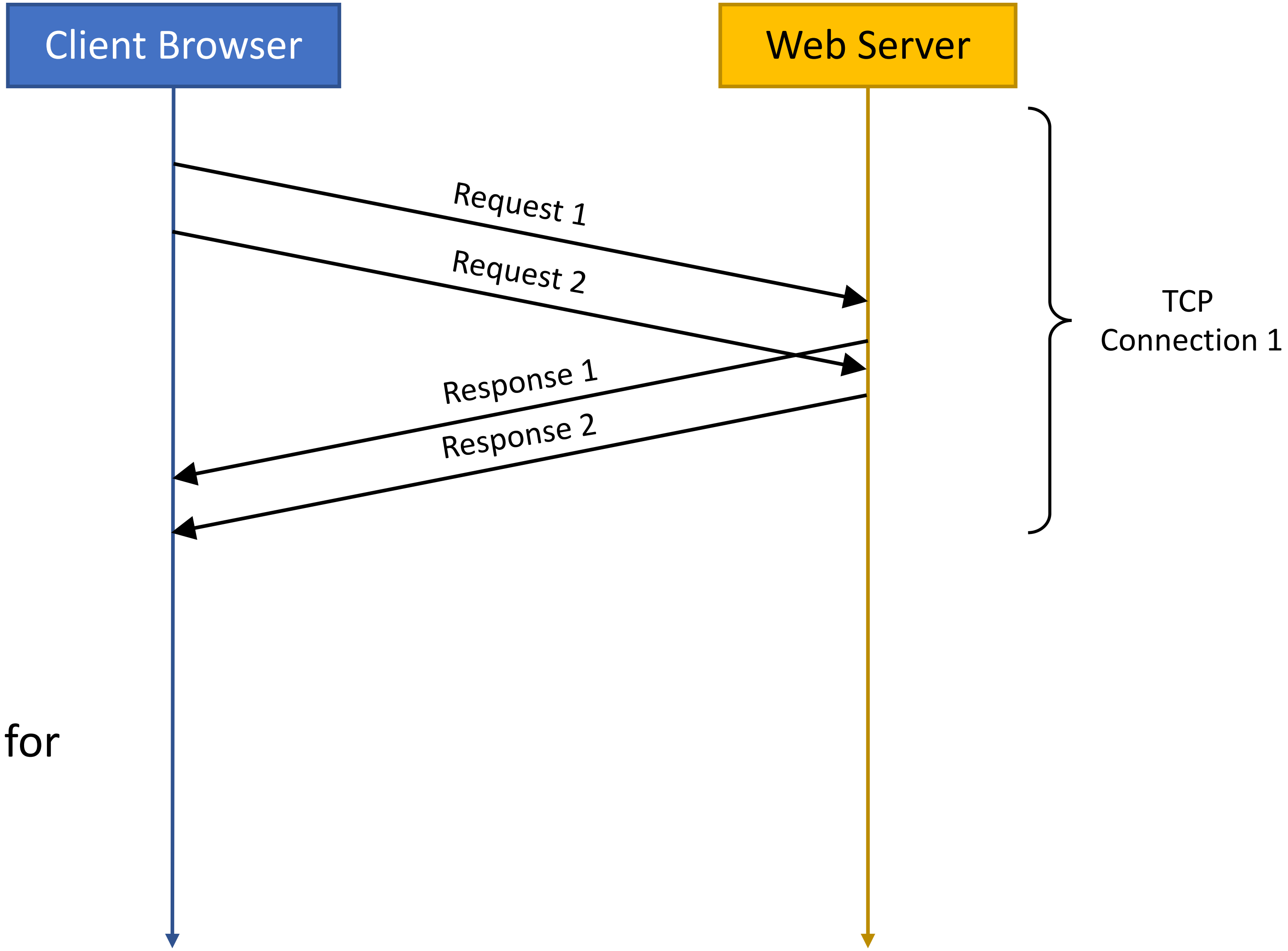
Plain HTTP Recap

- Request-response based
- Plain-text headers and body
  - human and machine readable
- Persistent connections
  - multiple request-response pairs
  - on single TCP connection



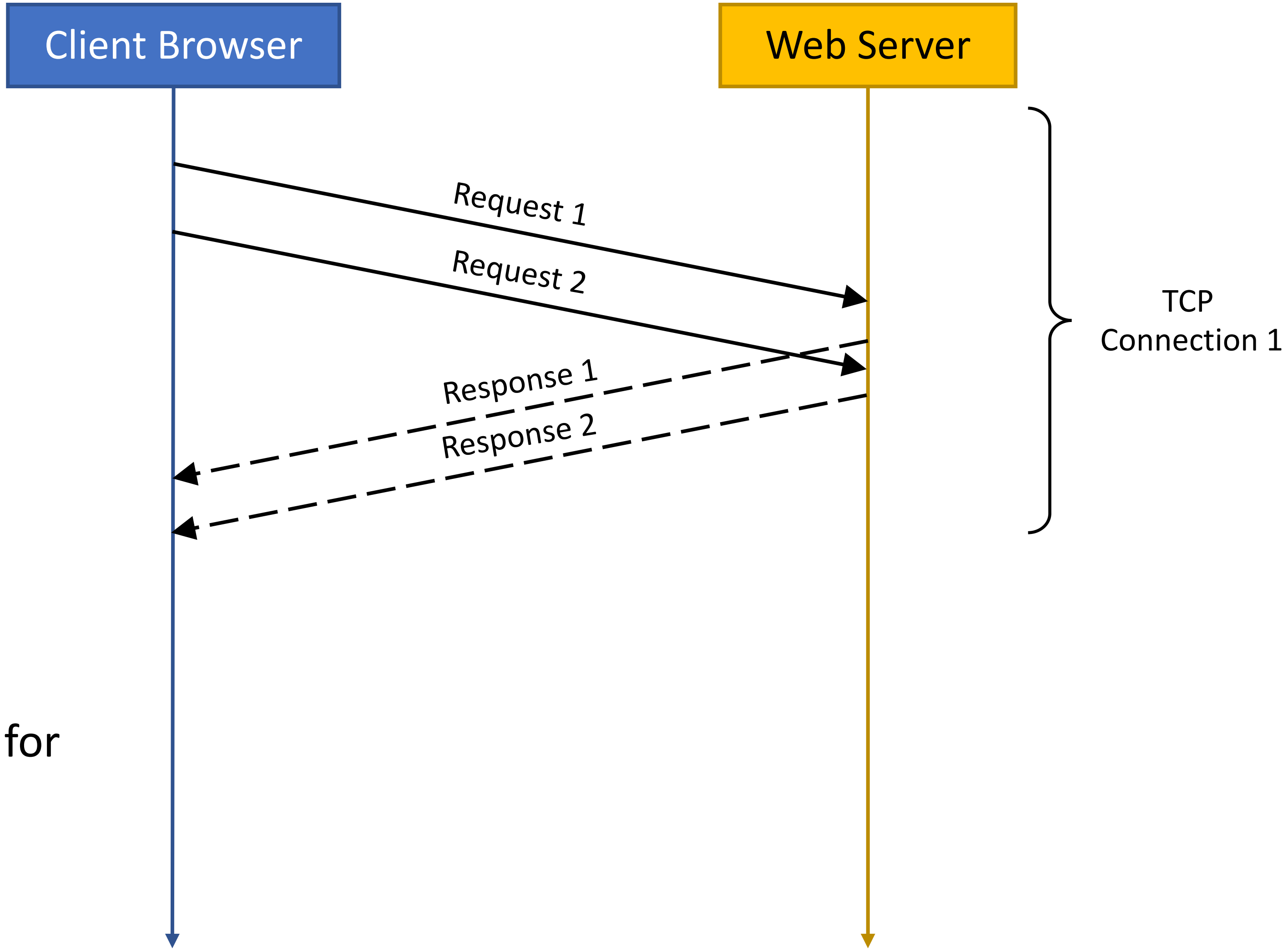
Plain HTTP Recap

- Request-response based
- Plain-text headers and body
  - human and machine readable
- Persistent connections
  - multiple request-response pairs
  - on single TCP connection
- Pipelining
  - multiple requests without waiting for response
  - on single TCP connection



Plain HTTP Recap

- Request-response based
- Plain-text headers and body
    - human and machine readable
- Persistent connections
    - multiple request-response pairs
    - on single TCP connection
- Pipelining
    - multiple requests without waiting for response
    - on single TCP connection
- Chunked transfer encoding
    - allow response to be broken into chunks
    - allow headers after body



Bi-directional Communication Over HTTP

Client to Server (TCP Conn #1)

- Regular HTTP requests

Server to Client (TCP Conn #2)

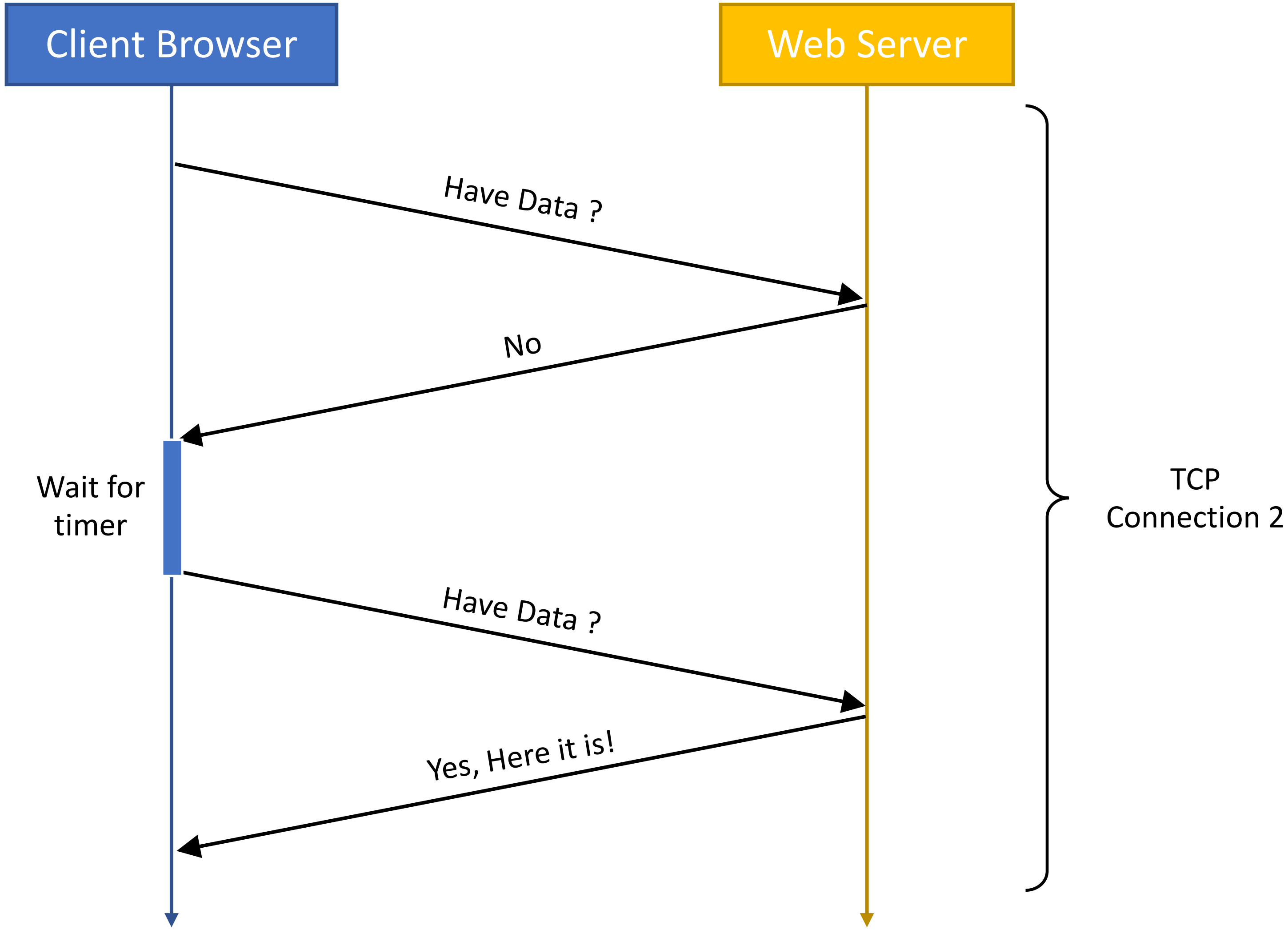
- Periodic Polling
- Long polling
- Streaming

Several Practical Combinations Implemented

- BOSH (Bidirectional-streams Over Synchronous HTTP)
- Comet (e.g. Pushlets)
- Bayou
- Server-Sent Events

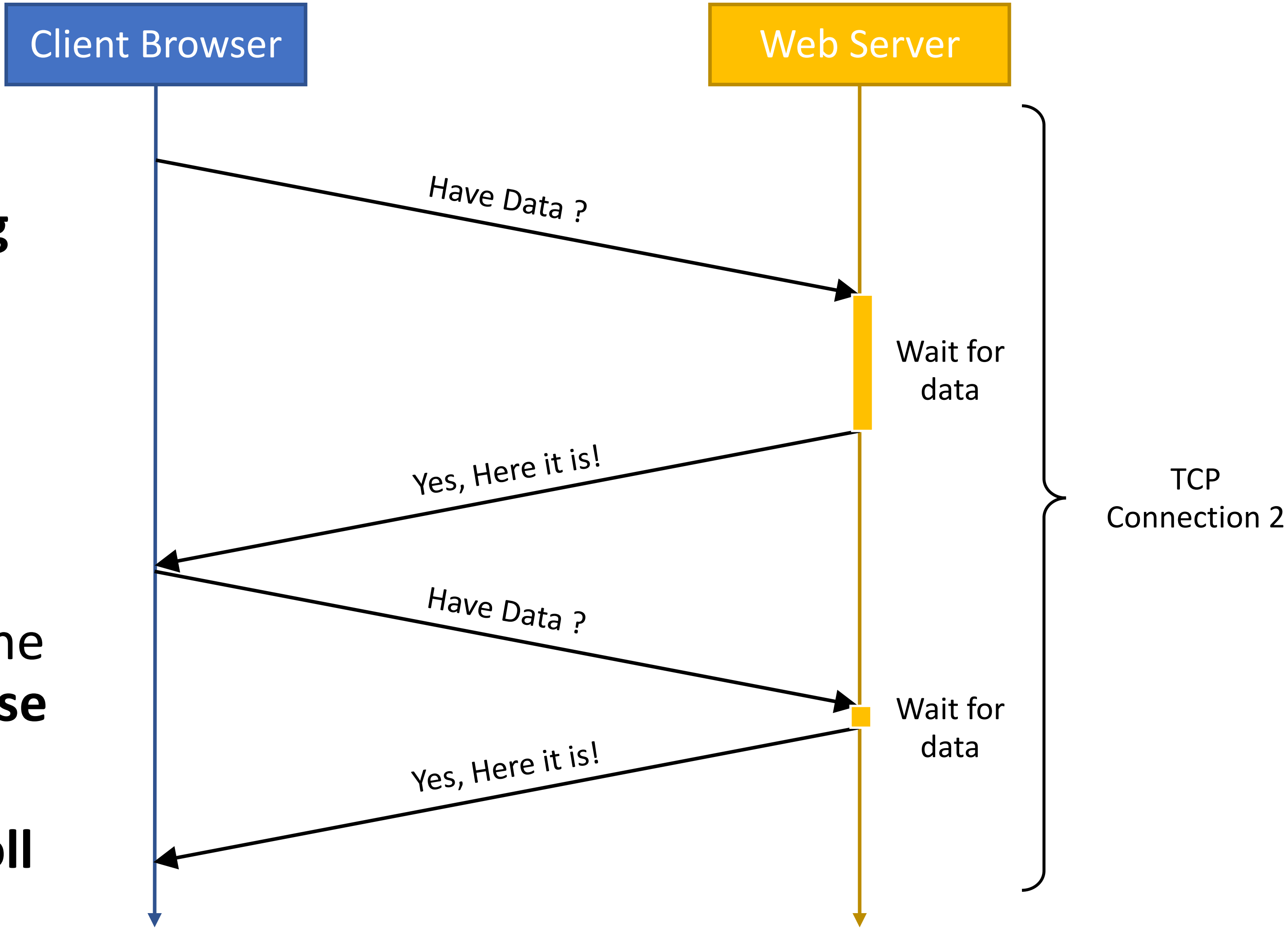
Periodic Polling

1. The client makes a **polling request** to check if server has data.
2. The **server sends a response** to the client, even if there is no data.
3. The client waits for some time and **repeats polling** from step 1.



HTTP Long Polling

1. The client makes an **initial long poll request** and waits for a response.
2. The **server defers its response** until an update is available, or timeout has occurred.
3. When an update is available, the **server sends complete response** to the client.
4. The client sends a **new long poll request**, either immediately or after a pause.



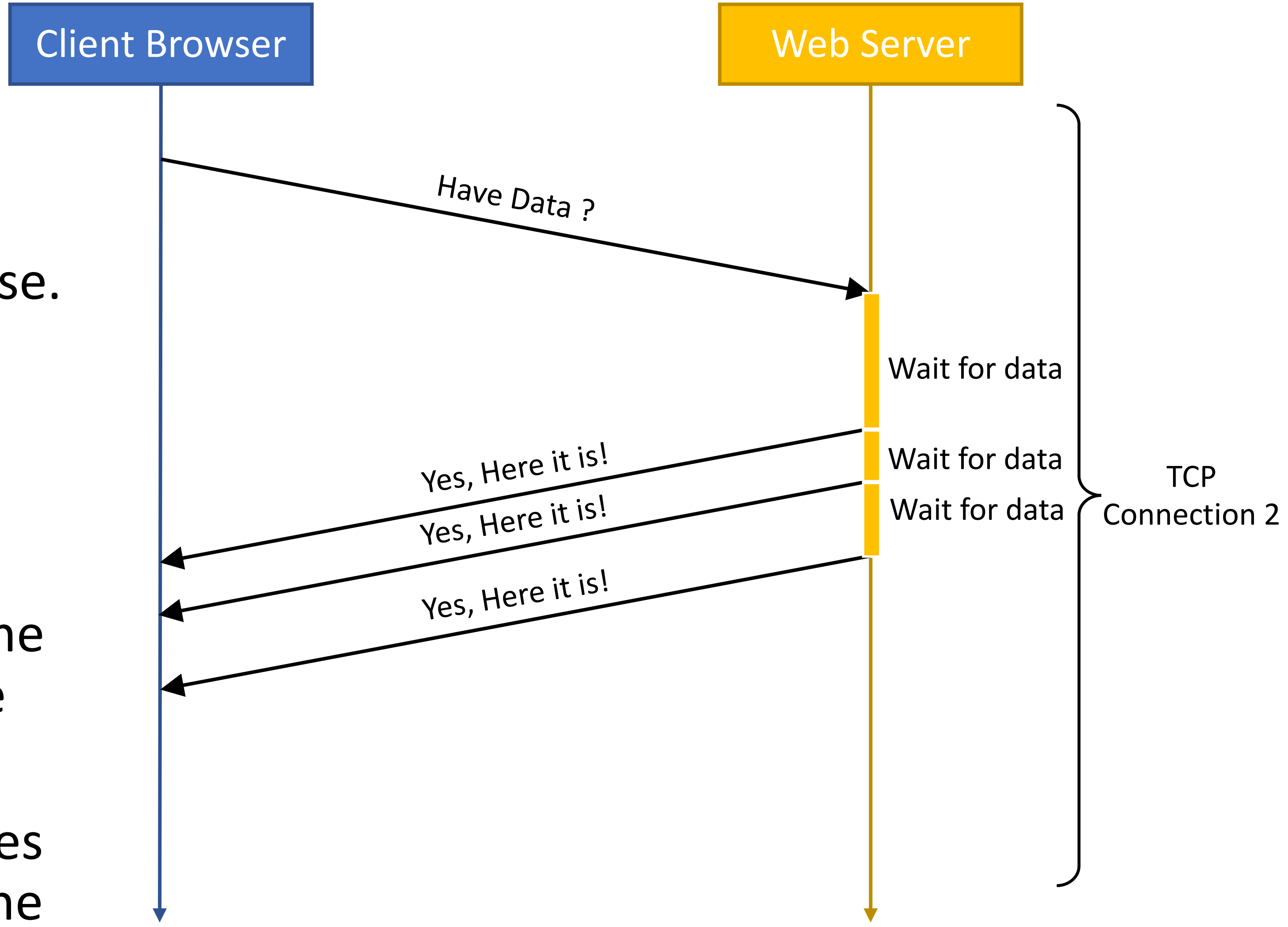
Long Polling: Issues

Issues

1. Header overhead
2. Unnecessary high maximal latency (> 1.5 RTT)
3. Frequent TCP connections (persistence helps)
4. Resource overhead at client and server
5. Unneeded buffering during higher loads
6. Timeouts
7. Caching

HTTP Streaming

1. The client makes an **initial request** and waits for a response.
2. The **server defers its response** until an update is available, or until a particular status or timeout has occurred.
3. When an update is available, the **server sends a response** to the client.
4. The data sent by the server does not terminate the request or the connection. The **server returns to step 3**.



Streaming: Issues

1. Network intermediaries (proxies, gateways) may buffer response
2. Unnecessary maximal latency (> 1.5 RTT) due to re-establishing streaming to avoid client memory limits
3. Client buffering (library to app)
4. Framing requirements not met by chunking (due to re-chunking)

Pushlets

- Old ~2002
- Implementation of Comet
- Publish/ subscribe mechanism
- Java servlets on server
- Push JavaScript snippets from server to client using HTTP streaming
- Whitepaper

Server-Sent Events (SSE)

- Proposed ~2009
- W3C Recommendation ~2015
- JavaScript EventSource API part of HTML5
- Built over HTTP streaming
- Good support by most modern browsers
  - Not supported by IE
  - 'Under Consideration' for Edge 16
- Uses Content-Type: text/event-stream

Upgrade to WebSocket!

RFC 6455

“Historically, creating web applications that need bidirectional communication between a client and a server (e.g., instant messaging and gaming applications) has required an **abuse of HTTP** to poll the server for updates while sending upstream notifications as distinct HTTP calls”

Introduction of RFC6455 (WebSocket Protocol) referring to RFC6202 (Long Polling and Streaming Issues and Best Practices)

(**emphasis** mine)

WebSocket vs. Long Polling/ Streaming

Problems Addressed

- Server resources: multiple TCP connections per client (up/ down)
- Protocol overhead: long header per HTTP message
- Client complexity: track pair of connections and state to one server

Solution Approach

- Use a single TCP connection for traffic in both directions.

WebSocket Protocol and API

- WebSocket protocol
  - Defined in RFC6455
  - Status: Proposed Standard ~2011
- JavaScript WebSocket API
  - Part of HTML5
  - W3C Candidate Recommendation ~2012

- Pointer: Watch the excellent talk Inside WebSockets by Leah Hanson

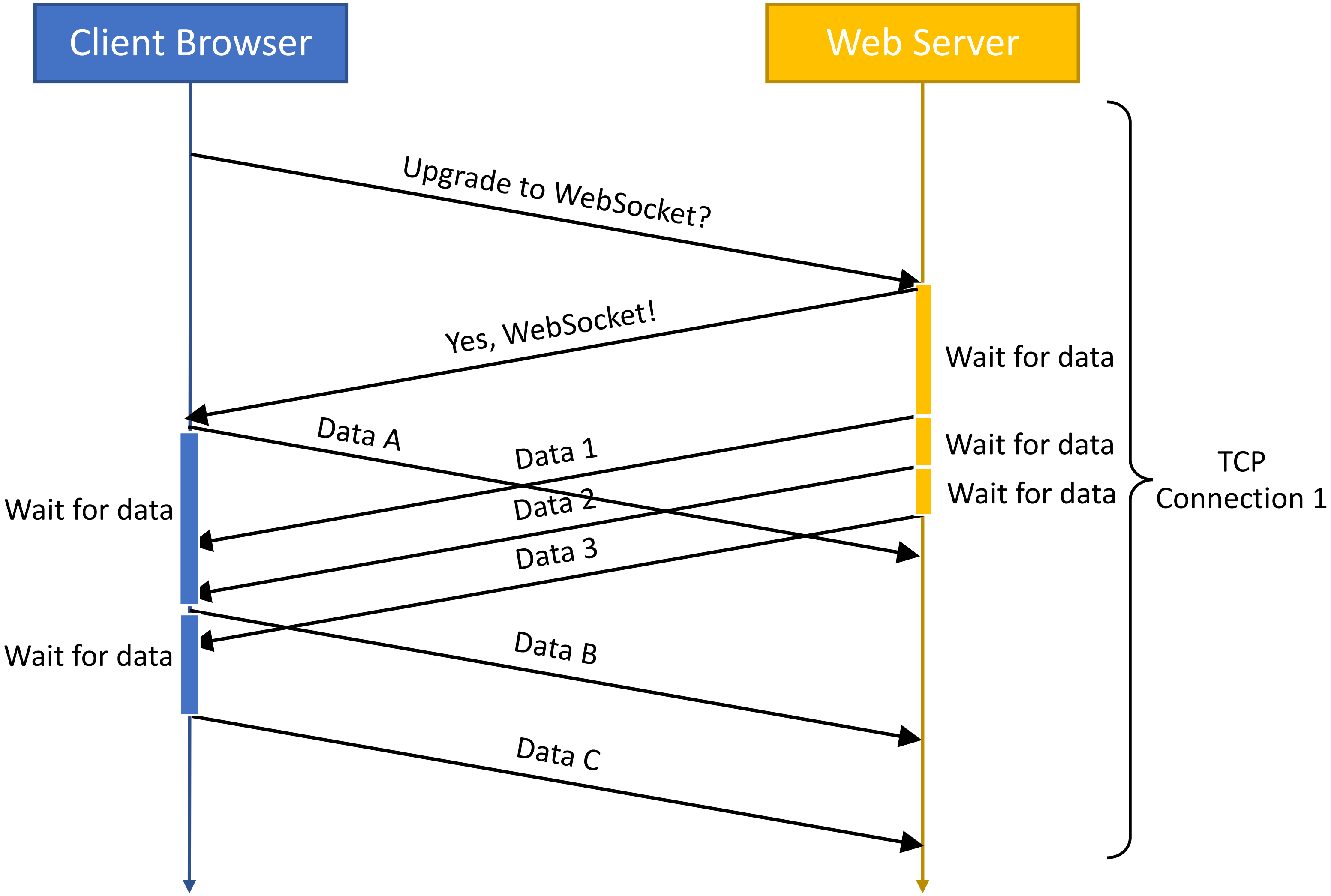
WebSocket

Advantage!

Only one TCP connection for both directions

Basic Steps

1. Opening handshake
2. Data exchange (two-way)
3. Closing handshake



WebSocket: Highlights

- Bi-directional communication over single TCP connection
- Either client or server can send a message anytime
- Low resource overhead at client and server
- No maximal RTT (except for opening handshake)
- Higher efficiency due to binary framing
- Designed to work well with existing Web infrastructure
  - Start with plain HTTP and 'upgrade' to WebSocket
  - Uses ports 80 and 443 for WS and WSS
  - Can tunnel through HTTP proxies via HTTP CONNECT

WebSocket Demo

- Simple echo client
  - https://www.websocket.org/echo.html
  - http://janodvarko.cz/test/websockets/
- Inspect WS with Firefox/ Chrome developer tools
  - Ctrl/Cmd + Shift + I
  - Network Tab → WS
- Firefox WebSocket Monitor extension
  - Extend developer tools with better WS support
  - Visualize WS sessions
  - https://github.com/firebug/websocket-monitor

WS: Other Notable Points

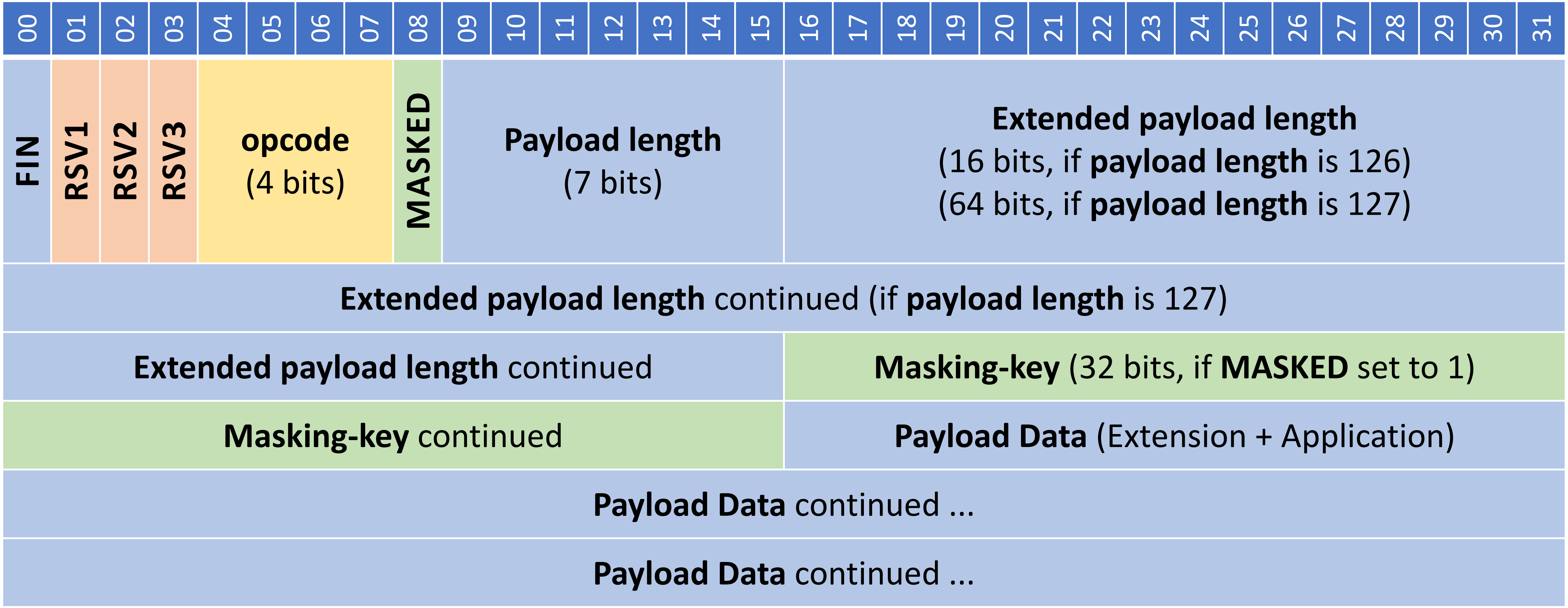
- Text (UTF-8) and binary data
- Messages and Framing
- Ping, Pong for keep-alive
- Frame masking from client to server

- Good implementations in browsers, and support in servers
- WS plain and WSS using TLS
- Sub-protocols (e.g. chat)
- Extensions (e.g. compression)

WebSocket Data Framing

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

WebSocket Data Framing



<table>
<tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td><td>05</td><td>06</td><td>07</td><td>08</td><td>09</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td><b>FIN</b></td><td><b>RSV1</b></td><td><b>RSV2</b></td><td><b>RSV3</b></td><td colspan="4"><b>opcode</b>
(4 bits)</td><td><b>MASKED</b></td><td colspan="7"><b>Payload length</b>
(7 bits)</td><td colspan="16"><b>Extended payload length</b>
(16 bits, if <b>payload length</b> is 126)
(64 bits, if <b>payload length</b> is 127)</td></tr>
<tr><td colspan="32"><b>Extended payload length</b> continued (if <b>payload length</b> is 127)</td></tr>
<tr><td colspan="16"><b>Extended payload length</b> continued</td><td colspan="16"><b>Masking-key</b> (32 bits, if <b>MASKED</b> set to 1)</td></tr>
<tr><td colspan="16"><b>Masking-key</b> continued</td><td colspan="16"><b>Payload Data</b> (Extension + Application)</td></tr>
<tr><td colspan="32"><b>Payload Data</b> continued ...</td></tr>
<tr><td colspan="32"><b>Payload Data</b> continued ...</td></tr>
</table>

WebSocket Extensions

Multiplexing (HyBi WG Draft)

- Provides logical channels in one TCP connection
- Each channel equivalent to a WebSocket connection
- Avoid head-of-line blocking by using different channels
- Current state: Expired (~2014)
- HTTP/2 maybe?

Compression (RFC 7692)

- Compress payload data of a message
- Negotiate parameters during opening handshake
- DEFLATE algorithm default
- Current state: Proposed standard (~2015)
- Browser support: minimal

Interesting Bi-Directional Frameworks

SockJS: WebSocket emulation

- WebSocket-like API even without WebSocket transport
- Focus on cross-browser compatibility
- Try native WebSocket first
- Automatically fall back transports (WS > streaming > polling)
- Multiple server implementations (JavaScript, Python, Java, Scala, Ruby, Go, Erlang, …)

Socket.IO: Real-time framework

- HTTP long-poll first
- Later upgrade to WebSocket if possible
- Handles disconnects
- Built-in keep-alive
- Supports namespaces
- Goes well beyond just WebSocket semantics
- JavaScript only

Q&A

My Coordinates

Twitter: @VipulMathur

LinkedIn: https://linkedin.com/in/VipulMathur