Team

- Mahnoor Shoukat - Organizer
- Saad Mohsin - Co-Organizer
- Zohaib Adnan - Presenter

ANTEMATTER

Real-Time Client and Server Communication

Workshop 0 | April 2024

What is the purpose of these workshops?

- Improve interaction between engineers (especially those who are fully remote)
- Deliver valuable information (duh)
- Can be both technical and non-technical
- Can be a combination of practical examples and theory

…also, I really need a raise.

This is going to be a theoretical one.

Please bear with me; we'll get through this.

Most engineers know how to work with, say, a WebSocket, but they often lack a theoretical understanding of how it works under the hood.

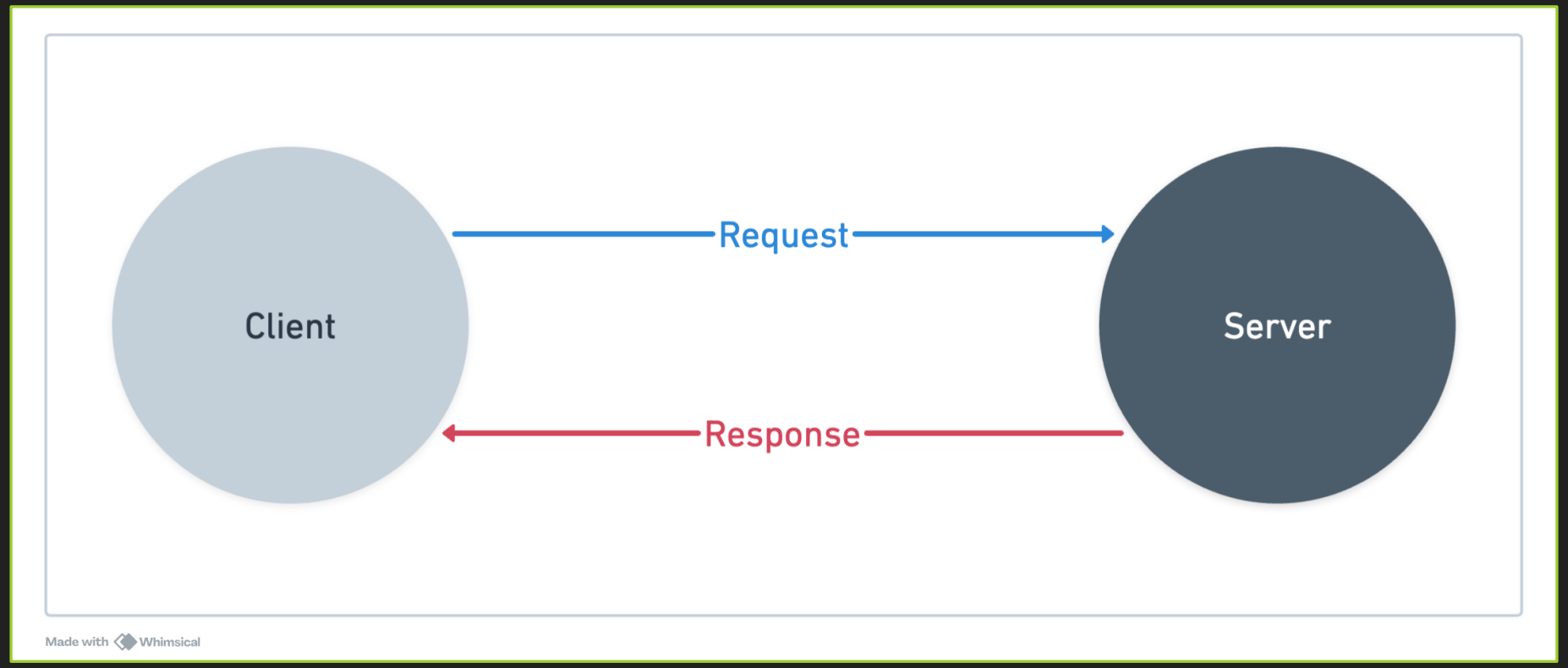
This kind of theoretical understanding may be required engineers need to optimize features on a lower level.

The Problem

- Live updates (live weather or real-time analytics)
- Chatting and messaging
- Real-time collaboration (collaborative whiteboarding)

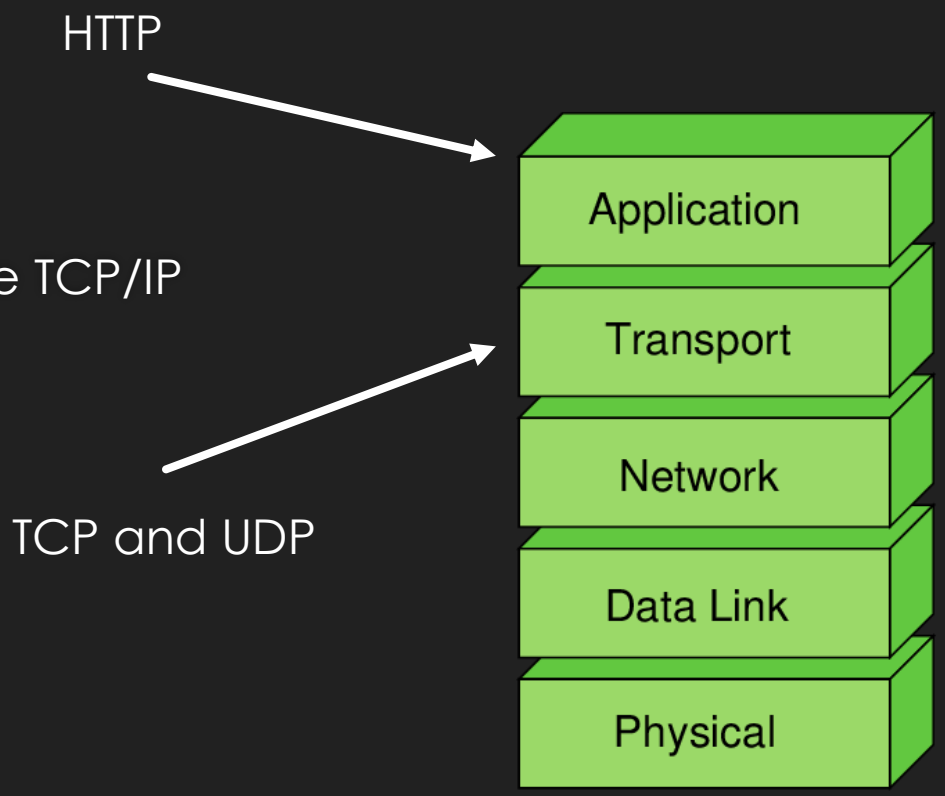
- We need to establish real-time communication between a client and a server.
- We should, ideally, be able to cater for both unidirectional and bidirectional communication.

What does client and server communication traditionally look like?



The HTTP Protocol

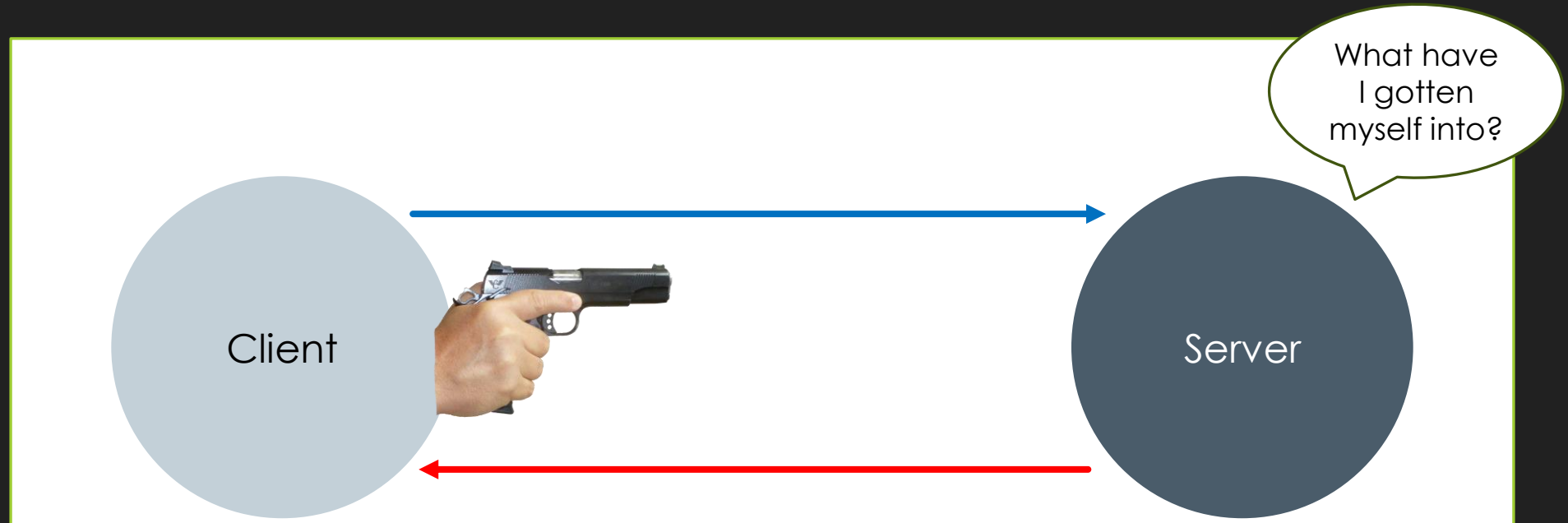
- Operates on top of the Transport layer (Layer 5 in the TCP/IP model and Layer 7 in the OSI model).
- Utilises TCP under the hood. Handshakes and acknowledgements come into play here.



Can you think of a primitive solution?

…given how the request-response cycle works.

Short Polling (a terrible approach)



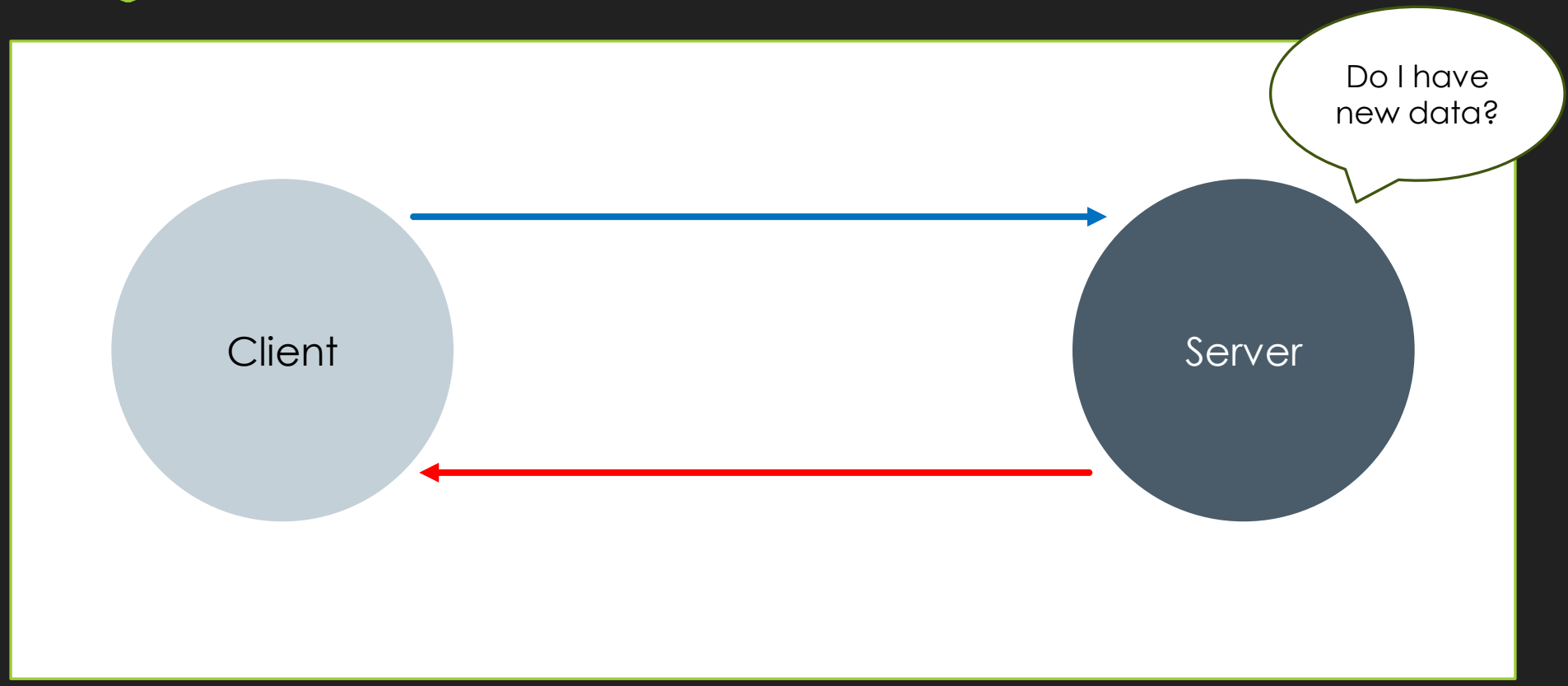
Why is short polling bad?

- The server is under tremendous load.
- The network is going to become congested.
- Each request will incur HTTP overhead.
- We'd eventually have to deal with high latency and dropped connections.

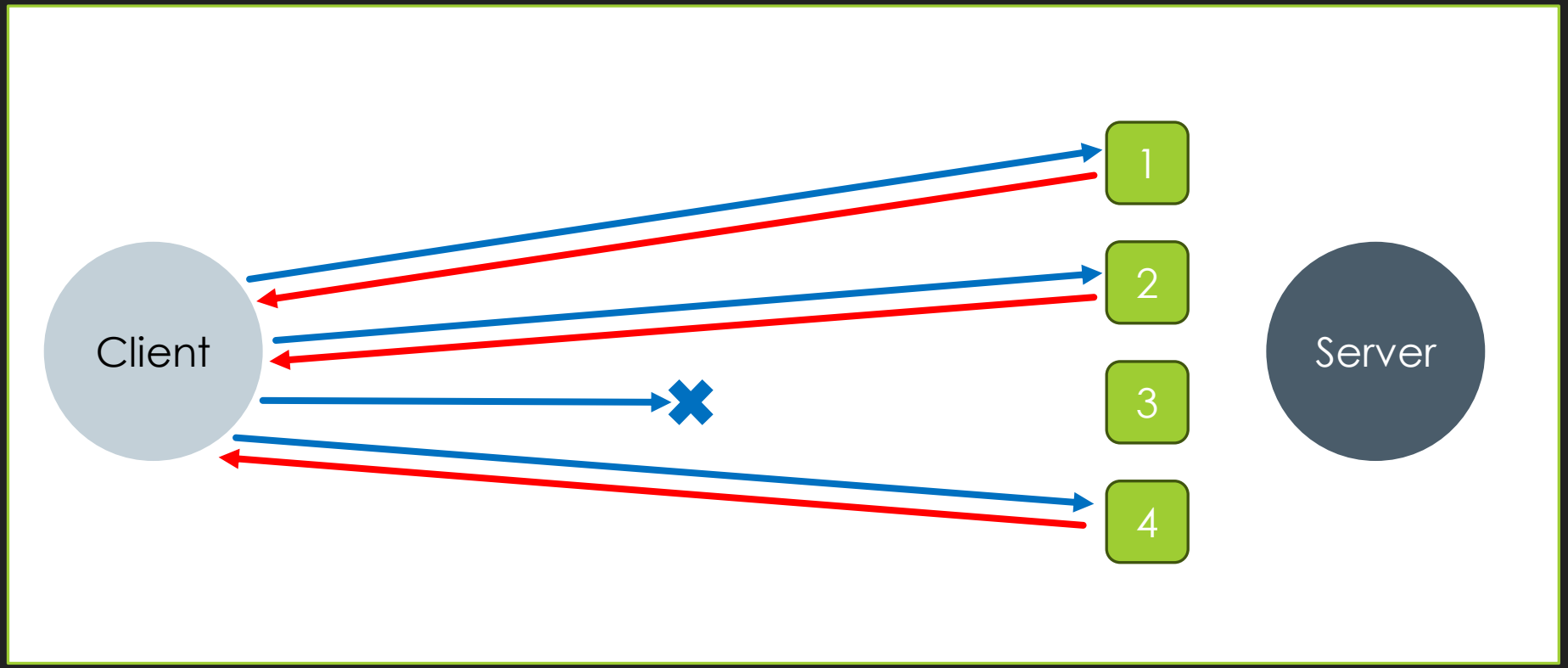
HTTP overhead includes DNS resolution, a TCP handshake, and a full set of HTTP headers.

This is usually around 5-7% (given an MTU of 1500 bytes).

Long Polling (a *slightly* better approach)



A Potential Problem



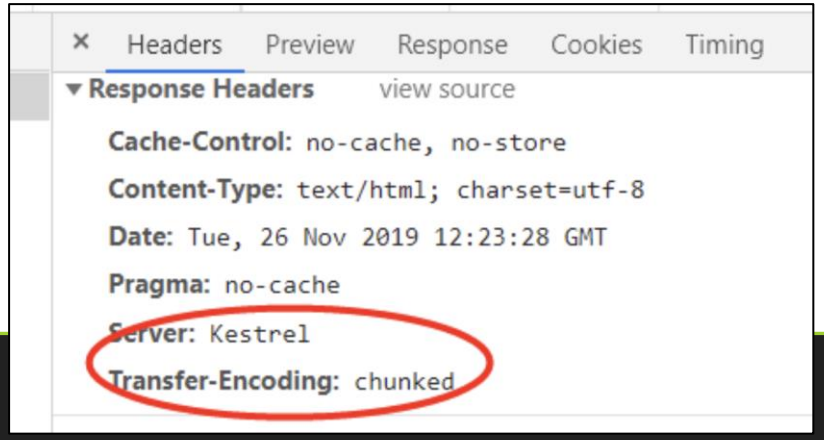
HTTP (Server) Streaming

- We need to devise an approach in which connections from the client are never closed – even after a response has been delivered.
- Again, this is NOT something that the HTTP protocol was designed for.

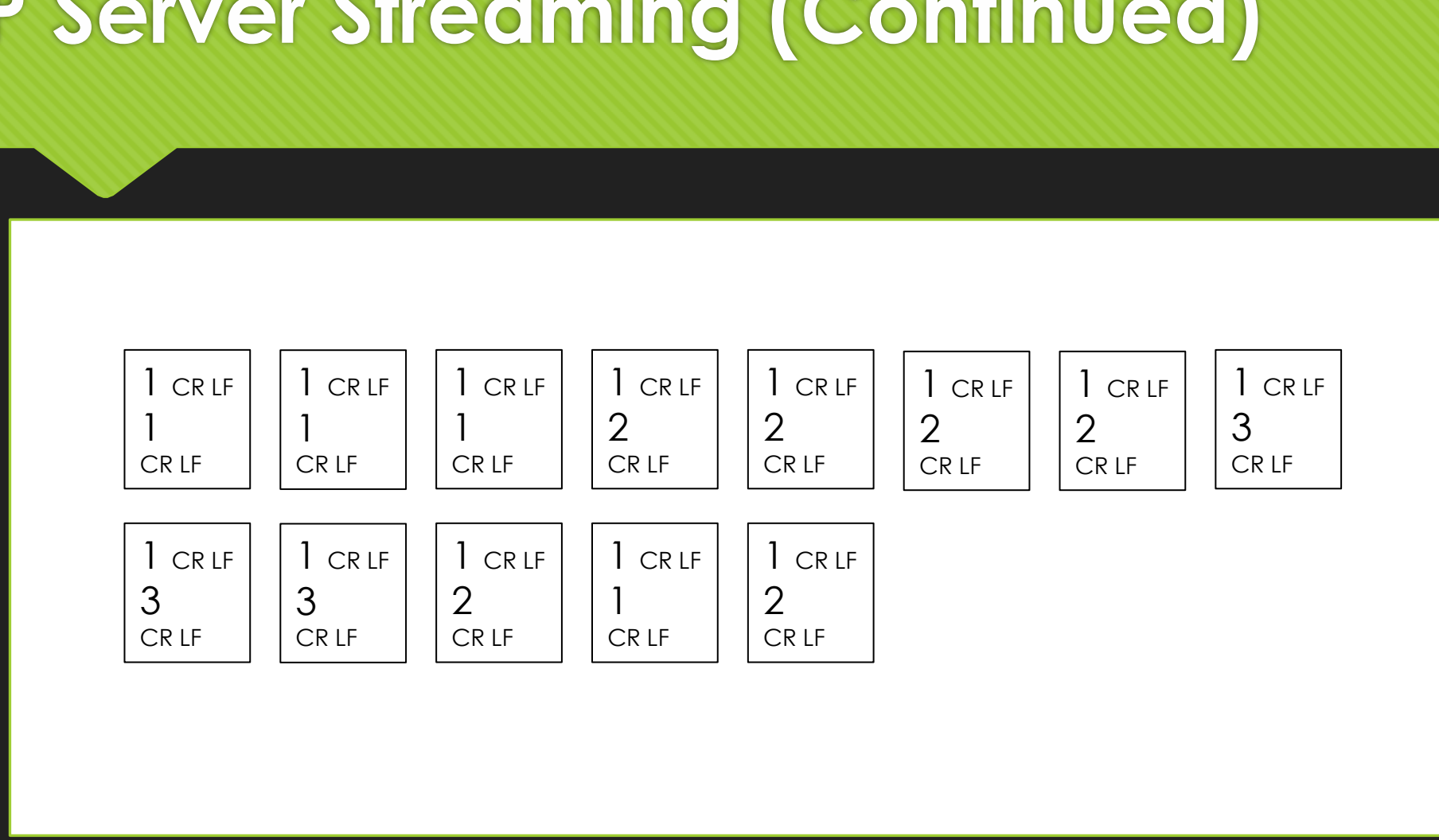
Chunked Transfer Encoding (RFC 9112)

"Antematter in chunks."

4 CR LF Ante CR LF	B CR LF matter in c CR LF	6 CR LF hunks. CR LF	0 CR LF CR LF



HTTP Server Streaming (Continued)



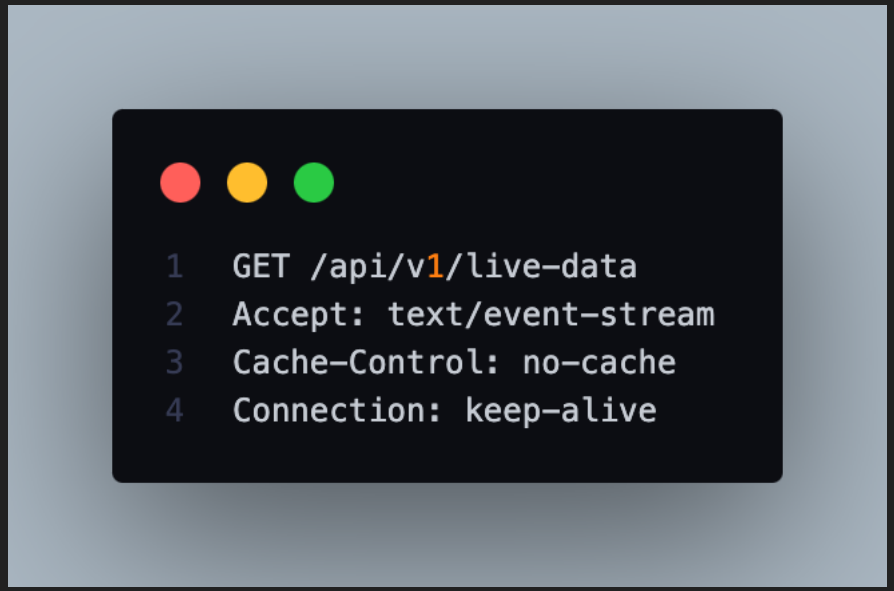
HTTP Streaming (Continued)

- More efficient than long polling, since we aren't repeatedly opening and closing connections. However, this is still not an optimal solution.
- We'd need to handle reconnection ourselves. We'd need to handle back pressure ourselves.
- We're going to hit a wall as soon as we need bidirectional communication.

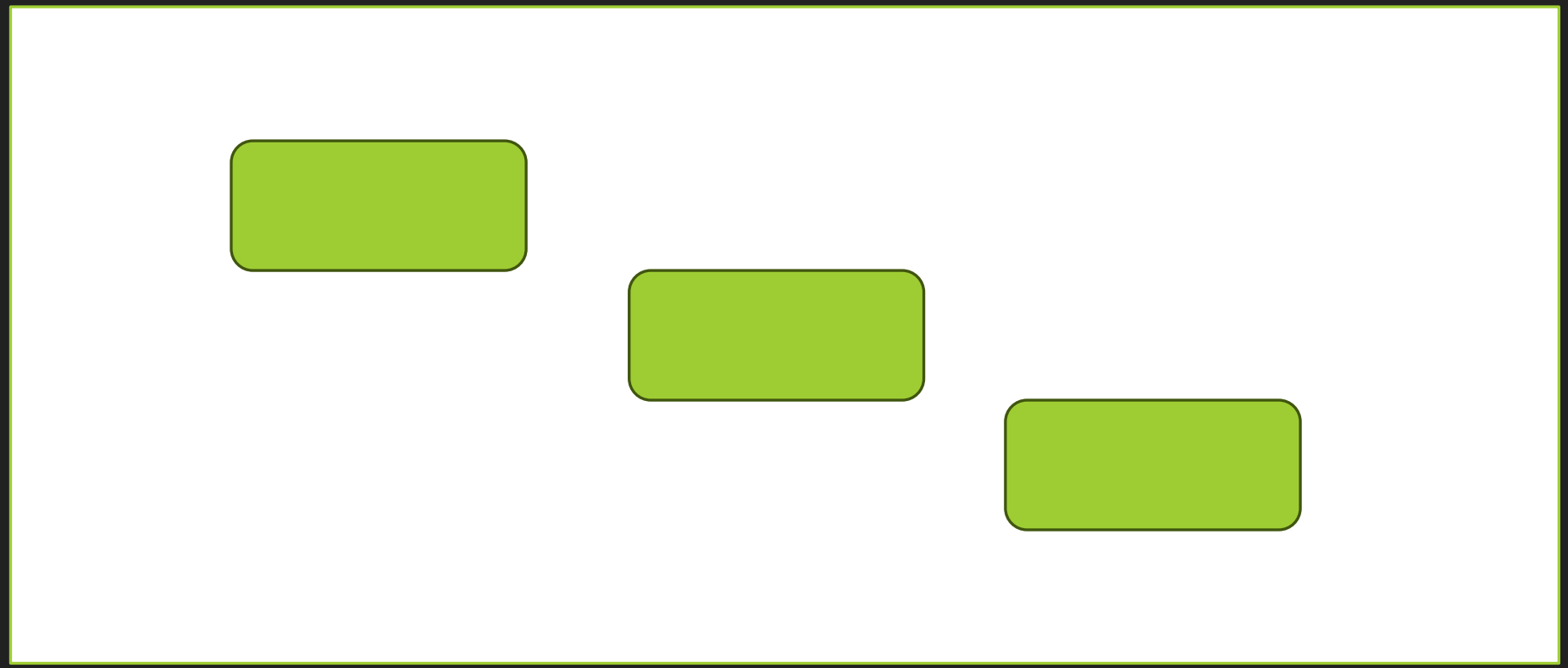
- The HTTP/2 specification disallows use of the 'Transfer-Encoding' header altogether.

Server-Sent Events (SSE)

- A standardised API in the HTML5 spec. Built on top of HTTP server streaming.
- Typically utlilises UTF-8 encoding to send text-based ‘events’.



SSE (Continued)



Let’s try ditching HTTP altogether.

The (Glorious) WebSocket