Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

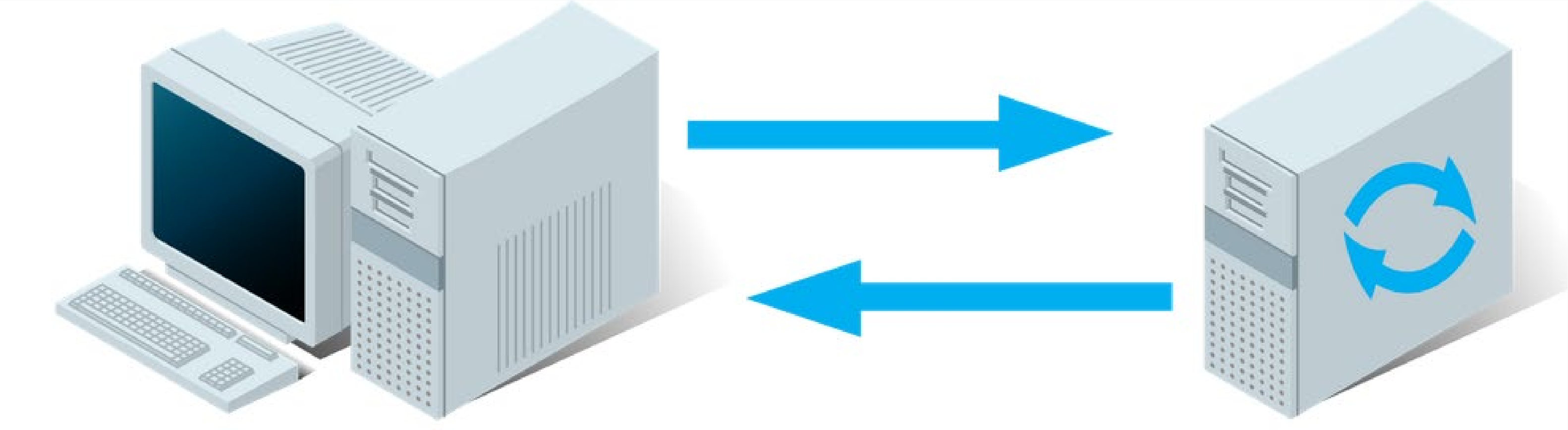
APPLICATIONS INTERNET

ASYNCHRONOUS WEB APPLICATIONS

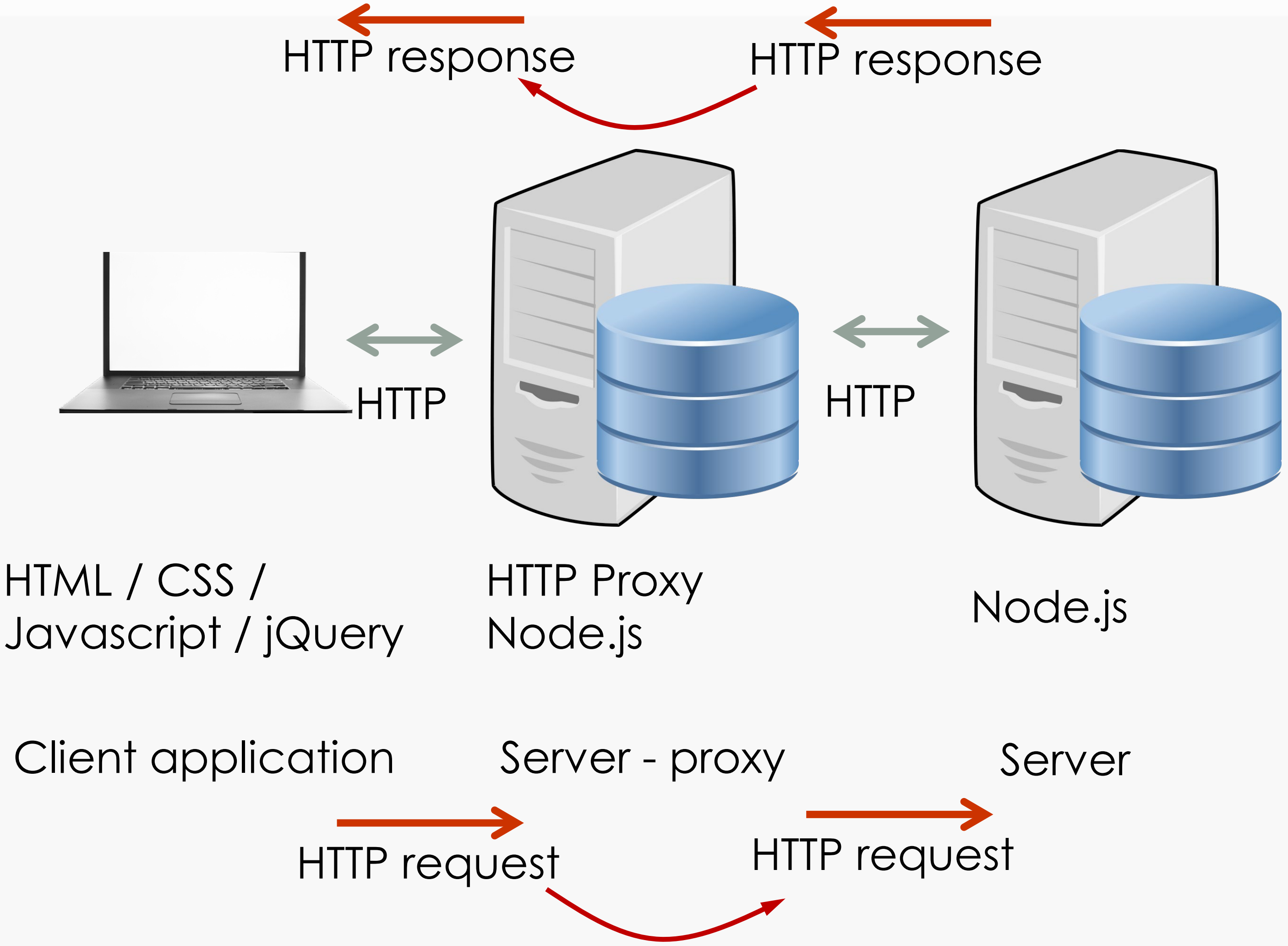
SERGE AYER - HEIA-FR – ISC
CLASSES ISC-ID-2A/D // 2023-2024

ARE WEB APPLICATIONS ASYNCHRONOUS?

- Web applications are synchronous in nature:
  - The user interacts with the web interface presented in the browser
  - The browser makes requests back to the server based on that user interaction, and
  - The server responds to those requests with new presentation for the user.



A WEB SYNCHRONOUS ARCHITECTURE



HTTP response

HTTP response

HTTP

HTTP

HTML / CSS /
Javascript / jQuery

HTTP Proxy
Node.js

Node.js

Client application

Server - proxy

Server

HTTP request

HTTP request

WHY DO WE NEED AN ASYNCHRONOUS WEB?

- The resource representation delivered to the user represents a snapshot in time of what is a dynamic system.
- That snapshot becomes stale in between user interactions and does not necessarily provide an accurate view onto the current state of the system.
- For achieving the asynchronous web, the server needs to be able to send responses back to the client application spontaneously!

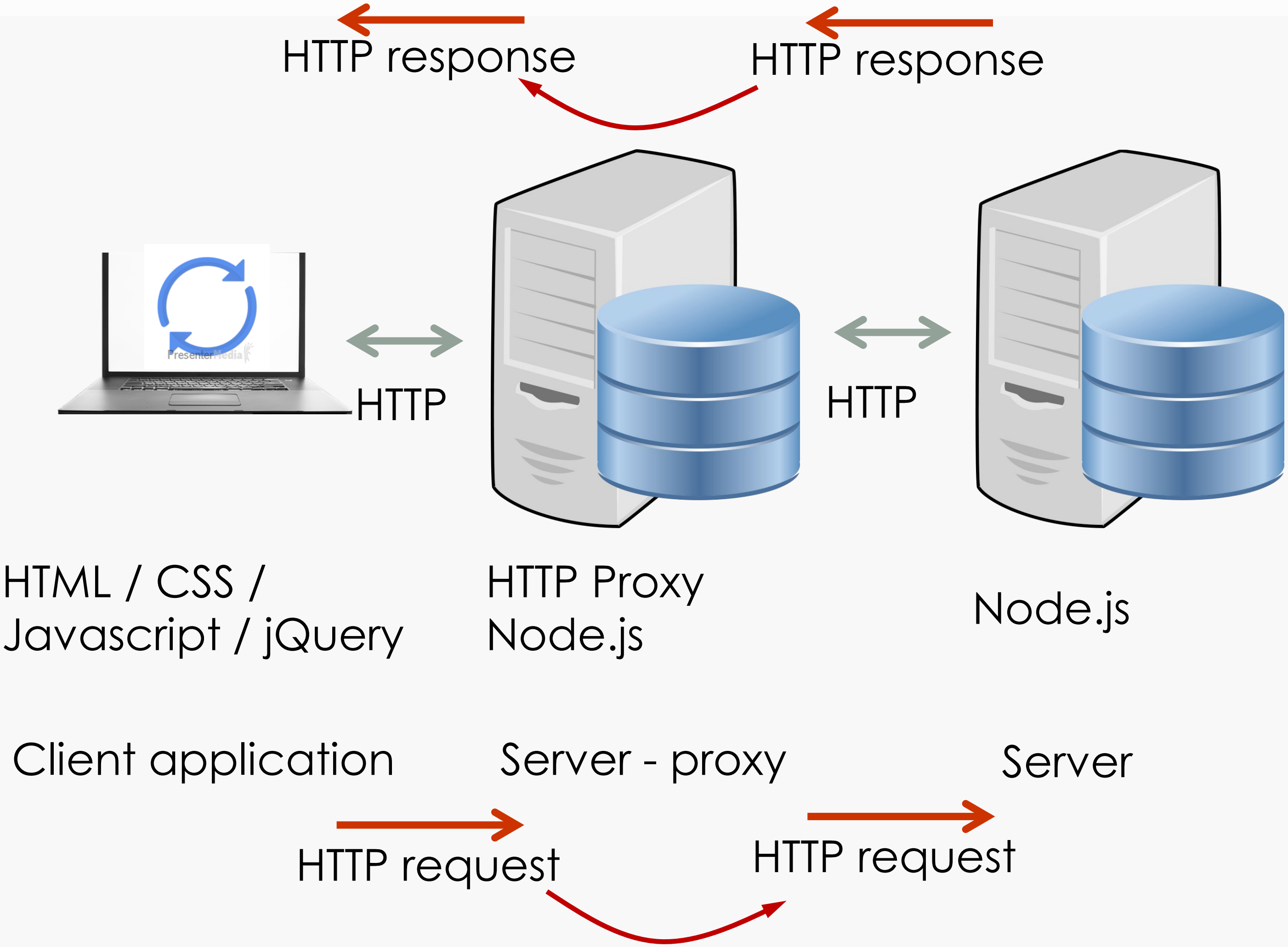
HOW DO WE IMPLEMENT AN ASYNCHRONOUS WEB?

- How can this be achieved with the HTTP protocol?
  - HTPP is following the request-response model.
  - The server cannot send a response to a non-existent request.
  - The request-response mechanism needs to be manipulated.
- The most straight forward way for the web application to get a more accurate view of the system is with a basic polling mechanism.
  - Polling through XMLHttpRequest/fetch techniques.
  - Send requests on a regular basis.

IMPLEMENTING FETCH BASIC POLLING

- Requires only the use of basic Fetch asynchronous requests on the client side
  - As implemented in the codelabs and your client application.
- No changes are required on the server side
  - The standard HTTP request/response mechanism is not modified.
  - A response to each request is sent synchronously be the server.
- The only change is that requests are made on a regular basis
  - It gives to the client continuous opportunities to update the representation of the observed system.

A WEB SYNCHRONOUS ARCHITECTURE



HTTP response

HTTP response

HTTP

HTTP

HTML / CSS /
Javascript / jQuery

HTTP Proxy
Node.js

Node.js

Client application

Server - proxy

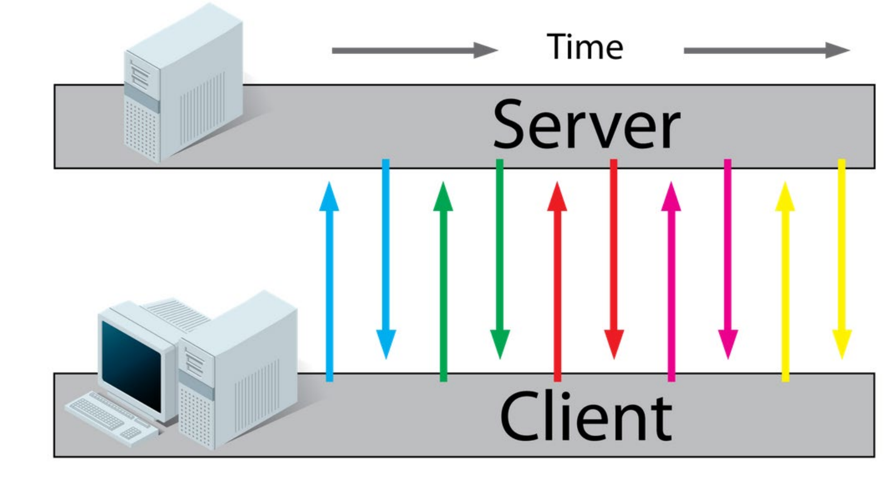
Server

HTTP request

HTTP request

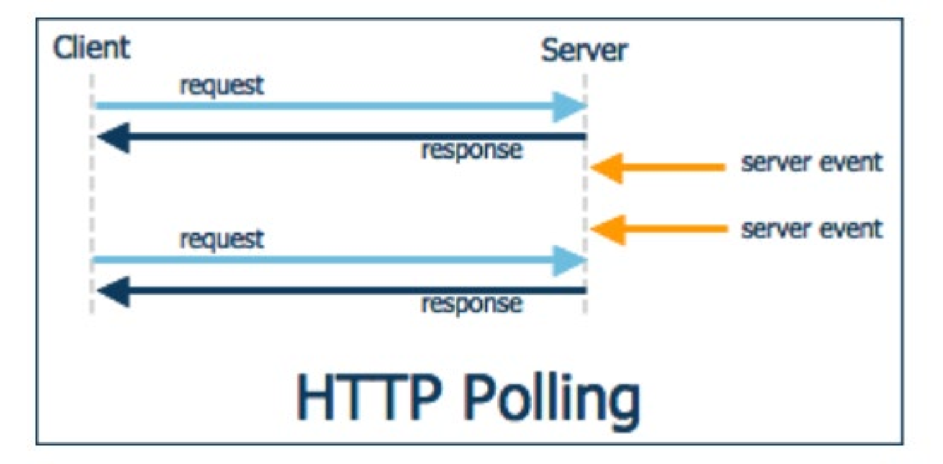
BASIC POLLING

1. The client requests a web page from a server using regular HTTP.
2. The requested webpage executes Javascript on the client for issuing requests to the server at regular intervals.
3. The server synchronously computes each response and sends it back to the client.



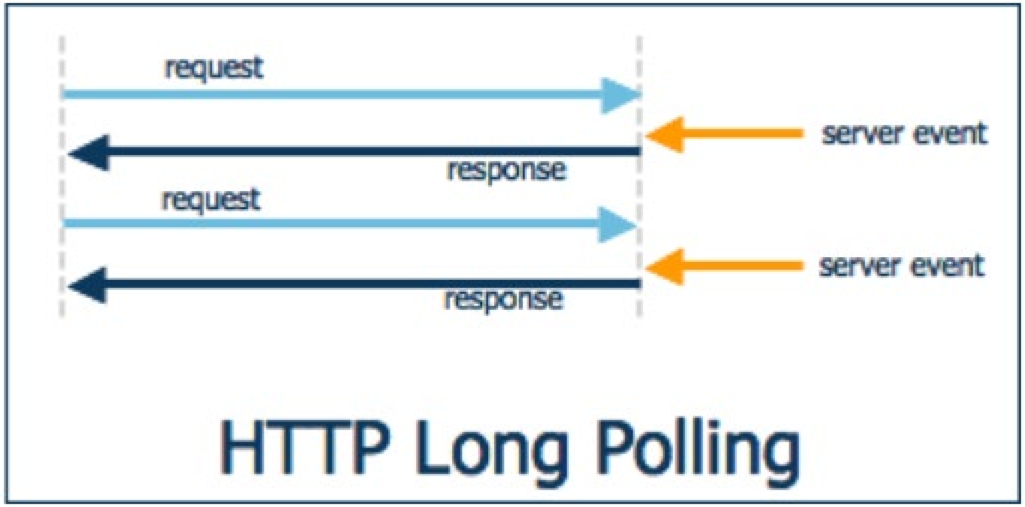
IS BASIC POLLING ENOUGH?

- Bringing asynchronous techniques into the client does not make the web asynchronous!
  - Trade-off between timely updates and network load.
  - It is possible for multiple server events to occur between polls.
  - It is possible that no server event occurs between polls.
  - The potential for a stale view of the system persists.



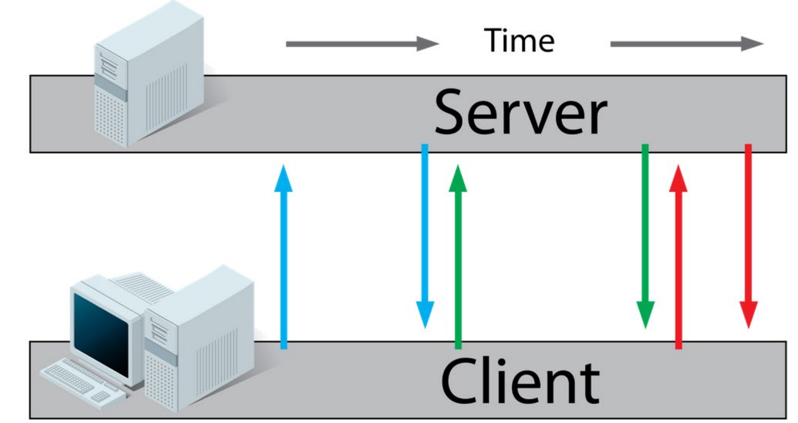
ALTERNATIVES FOR AN ASYNCHRONOUS WEB

- Consider HTTP Long Polling:
  - The request is made in anticipation of a future response.
  - The response is blocked until some event occurs on the server side.



LONG POLLING

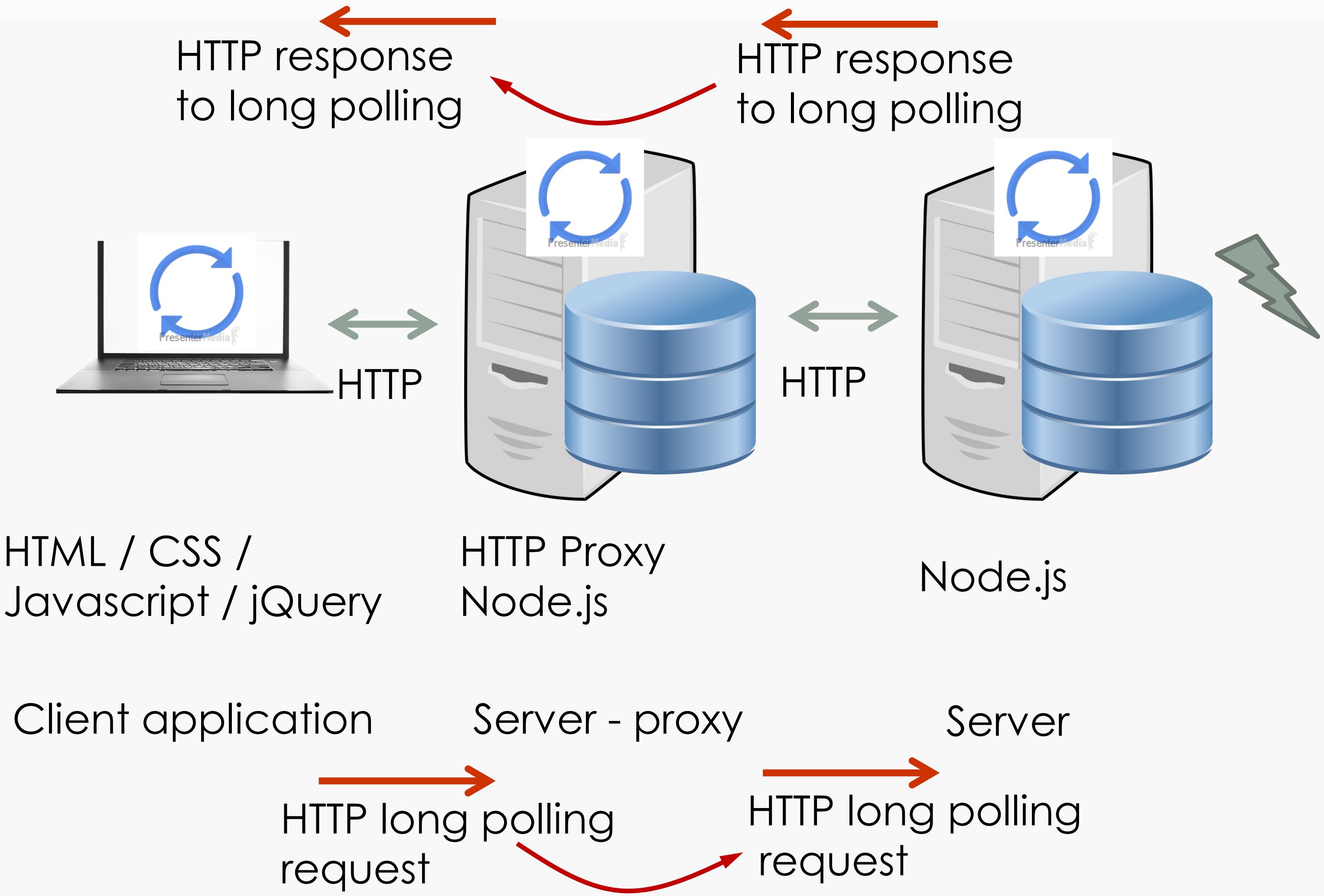
1. A client requests a webpage from a server using regular HTTP (see HTTP above).
2. The requested webpage executes JavaScript which requests a resource from the server.
3. The server does not immediately respond with the requested information but waits until there's **new** information available.
4. When there's new information available, the server responds with the new information.
5. The client receives the new information and immediately sends another request to the server, re-starting the process.



IMPLEMENTING LONG POLLING

- Changes are required on the server side. Why ?
- Remember that HTTP 1.1 is based on permanent connections
  - In HTTP 1.1, a connection is kept alive and reused for multiple requests.
  - This is required for reduced communication lag.
- Processing on the server side requires a request to be maintained per connection
  - Each HTTP connection between a client and a server is associated with one request on the server side.
  - Once a connection is closed, the dedicated request is destroyed on the server side.

A WEB SYNCHRONOUS ARCHITECTURE



HTTP response
to long polling

HTTP response
to long polling

HTTP

HTTP

HTML / CSS /
Javascript / jQuery

HTTP Proxy
Node.js

Node.js

Client application

Server - proxy

Server

HTTP long polling
request

HTTP long polling
request

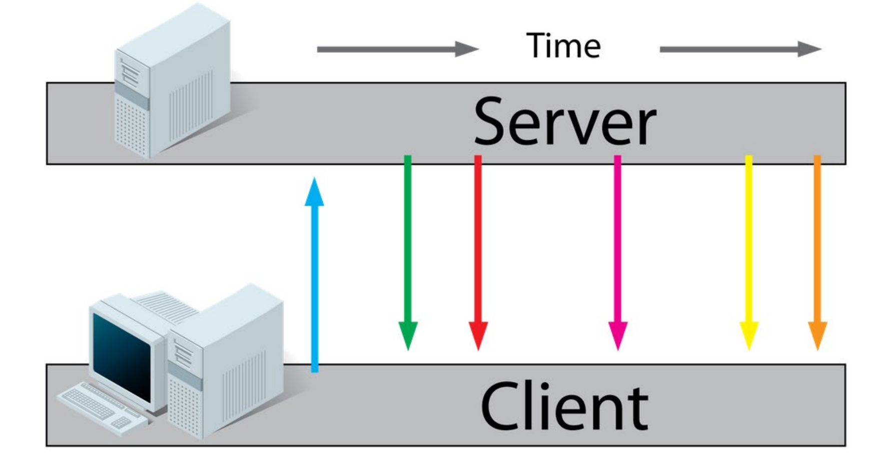
IS LONG POLLING THE ULTIMATE SOLUTION FOR WOT APPLICATIONS?

- Client have to reconnect periodically after connection is closed due to timeouts or after data is received.
- This adds a lot of HTTP overhead since it is constantly establishing and tearing down HTTP connections.
- Is there a way to reduce this overhead, while allowing the server to send data to the client continuously ?
  - The client can open a single long-lived HTTP connection.
  - The server then unidirectionally sends data when requested.
  - There is no need for the client to request it or do anything but wait for messages.
  - This solution is called Server-Sent Events (SSE) and is implemented in HTML5 using EventSource

SERVER-SENT EVENTS

1. A client requests a webpage from a server using regular HTTP.
2. The requested webpage executes Javascript which opens a connection to the server - using an EventSource object.
3. The server sends an event to the client when there's new information available.

SERVER-SENT EVENTS



- This allows real-time traffic from server to client.
- The server definitely requires asynchronous processing.
- Invoking an EventSource from another domain requires special care.

EVENTS STREAM FORMAT

- The event stream message has a “text/event-stream” Content-Type.
- An event stream is a simple stream of text data which must be encoded using UTF-8.
- Messages in the event stream are separated by a pair of newline characters.
- A colon as the first character of a line is in essence a comment, and is ignored.
- Each message consists of one or more lines of text listing the fields for that message.
- Each field is represented by the field name, followed by a colon, followed by the text data for that field's value.

EVENTS STREAM FORMAT

- Fields:
  - event: the event’s type.
  - data: the data field for the message (the payload)
    - Can be JSON
  - id: the event’s id.
  - retry: the reconnection time.
- Example:

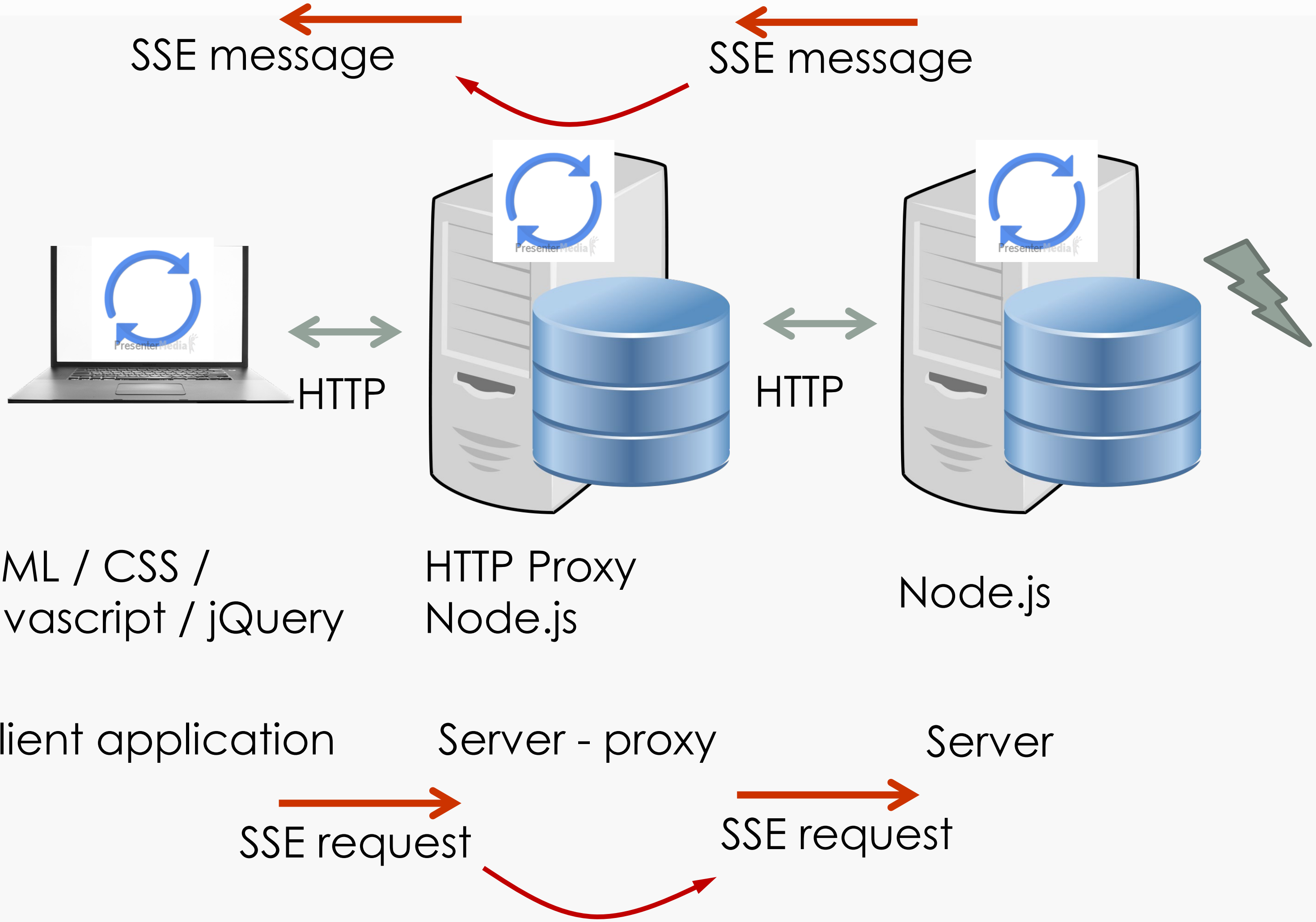
```
event: text
data: some text

event: userconnect
data: {"username": "bobby", "time": "02:33:48"}
```

EVENT SOURCE IMPLEMENTATION

- SSEs are sent over traditional HTTP.
  - That means they do not require a special protocol or server implementation.
- Client side:
  - If the connection drops, the EventSource fires an error event and automatically tries to reconnect.
  - The server can also control the timeout before the client tries to reconnect.
- Server side:
  - A server can only accept EventSource requests if the HTTP request says it can accept the event-stream MIME type.
  - A server needs to maintain a list of all the connected users in order to emit new events.

A WEB SYNCHRONOUS ARCHITECTURE



SSE message

SSE message

HTTP

HTTP

HTML / CSS / Javascript / jQuery

HTTP Proxy
Node.js

Node.js

Client application

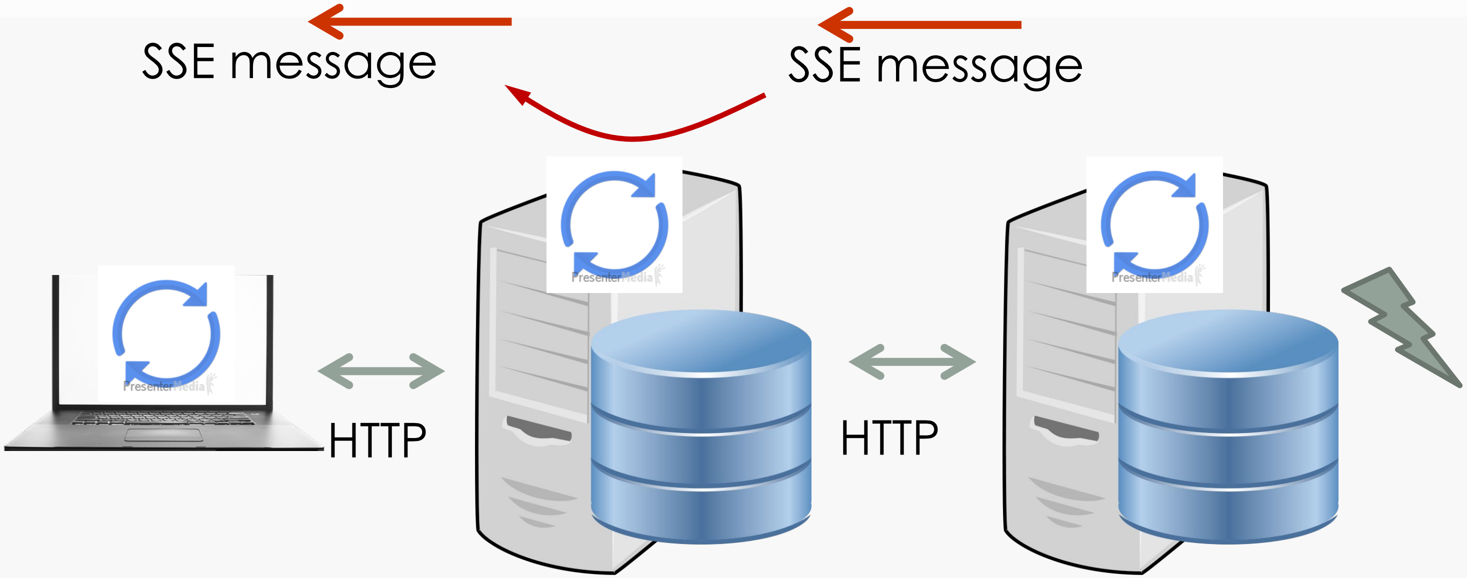
Server - proxy

Server

SSE request

SSE request

A WEB SYNCHRONOUS ARCHITECTURE



SSE message

SSE message

HTTP

HTTP

HTML / CSS / Javascript / jQuery

HTTP Proxy
Node.js

Node.js

Client application

Server - proxy

Server

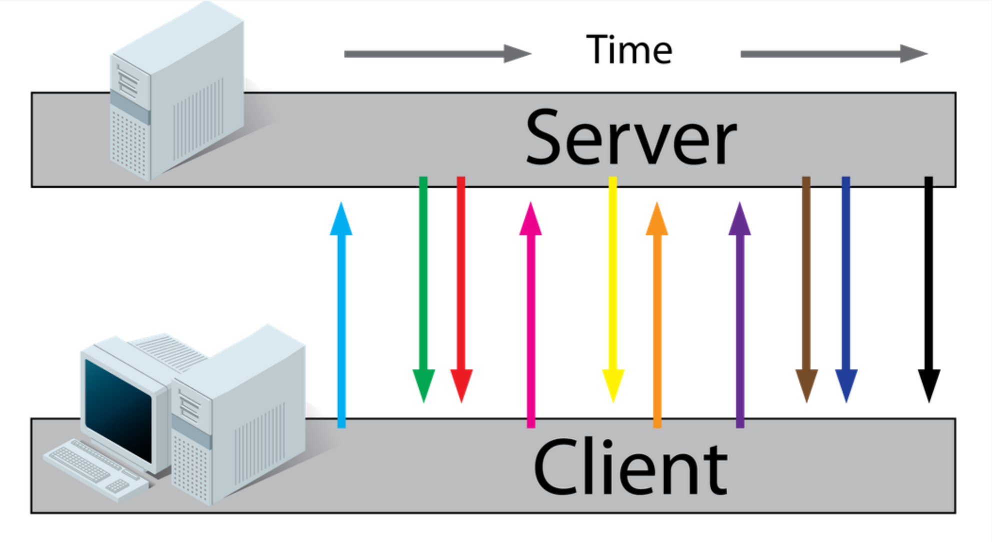
WEBSOCKETS FOR THE ASYNCHRONOUS WEB

- Allowing bi-directional communication between the client and the server can be achieved with WebSockets.
- WebSockets work without the overhead of an HTTP protocol.
  - It uses its own protocol, which is defined by the IETF (RFC 6455).
- The WebSockets API can be used by web applications to open and close connections and to send and receive messages.
  - It is defined in a W3C Specification.

WEB SOCKETS

1. A client requests a webpage from a server using regular HTTP.
2. The requested webpage executes JavaScript which opens a connection with the server.
3. The server and the client can now send each other messages when new data (on either side) is available.

WEB SOCKETS



- Real-time traffic from the server to the client and from the client to the server.
- The server definitely requires asynchronous processing.
- It is possible to connect with a server from another domain.

WEB SOCKETS

- Benefits
  - Reduce unnecessary network traffic and latency – using full duplex communication through a single TCP connection
  - Streaming through proxies and firewalls, with the support of upstream and downstream communication simultaneously
- API
  - The client must initialize the connection to the server by creating a WebSocket JavaScript object

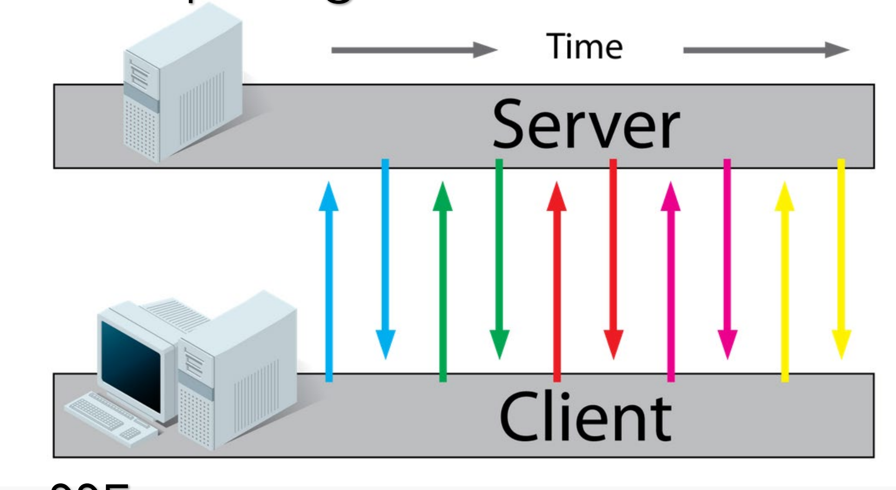
    var socket = new WebSocket(«ws://echo.websocket.org»)
  - Event based API with the following events
    - Open - onopen
    - Message - onmessage
    - Close – onclose
    - Error - onerror
  - Event handlers are implemented by setting callback functions or with the help of the addEventListener method

WEB SOCKETS

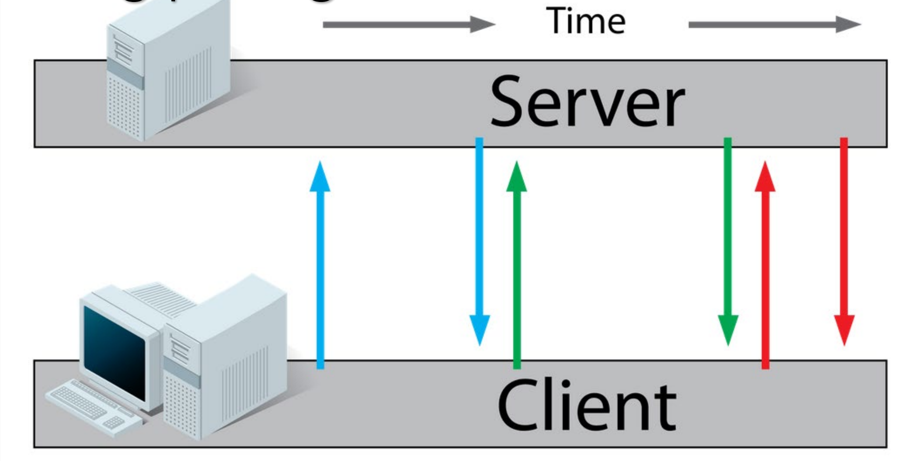
- API
  - Actions
    - send() for sending a message
    - close() for closing the connection
  - Message content can be
    - Text as a string
    - Binary data as a ArrayBuffer
- Advantages summarized
  - Bi-directional
  - Full Duplex
  - Single TCP connection (HTTP connection upgraded)

WRAP UP

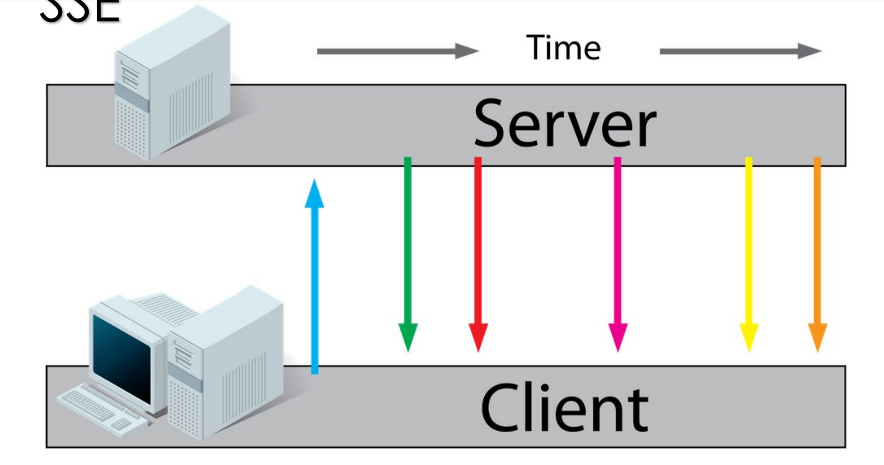
Basic polling



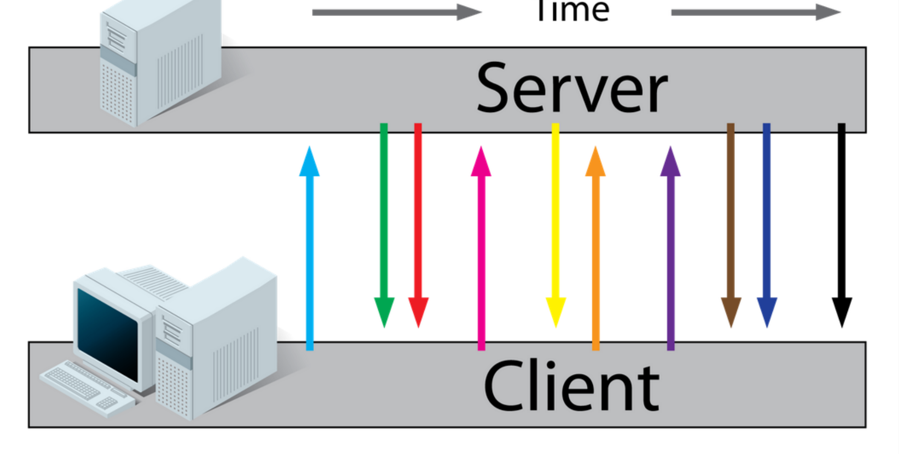
Long polling



SSE



WebSockets



SPECIFICATIONS OF EVENT-DRIVEN APIS

- As for RESTful APIs, one standard is emerging for standardizing Event-Driven APIs

  AsyncAPI

- Open source initiative
- Make working with Event-Driven APIs as easy as with RESTful APIs:
  - Documentation
  - Code generation
  - Discovery
  - Event management