Real-Time Web Application

Thierry Sans

Overview

➡ An application is updated with the latest information without any user interaction

Different Solutions

- Long Polling
- Sever-Sent Events
- Web Sockets
- Web RTC (Real-Time Communication)

Long Polling

Short Polling vs Long Polling

Short Polling

- The frontend request an update from the backend every few seconds
- The backend replies right away regardless if there is an update or not

◉ Many request/responses are wasted

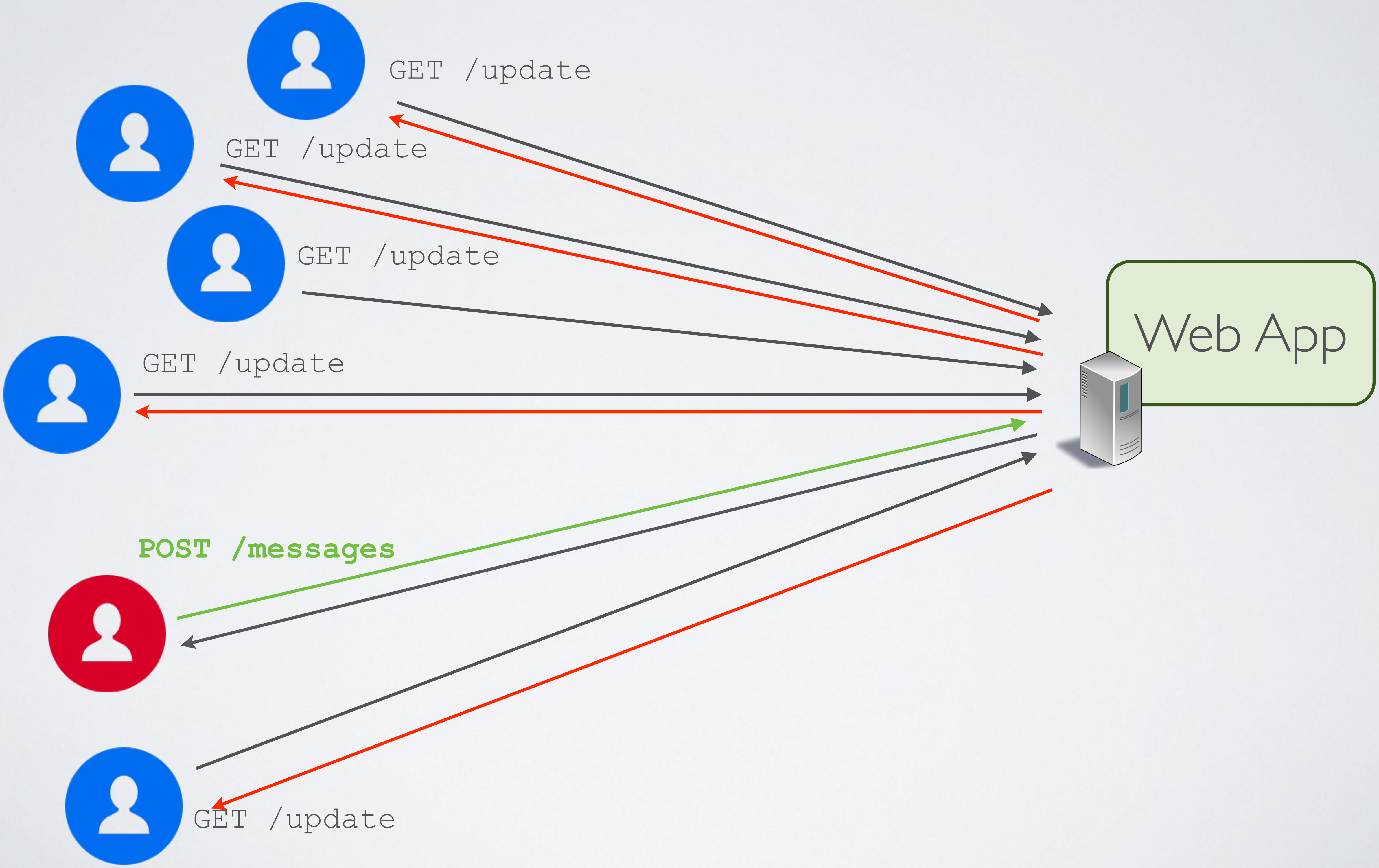
Long Polling

- The frontend request an update from the backend and wait for the response
- The backend replies to the update request only when there is an update

✓ No request/response wasted

✓ Updates are processed as soon as they arrived

Long Polling



GET /update

GET /update

GET /update

GET /update

POST /messages

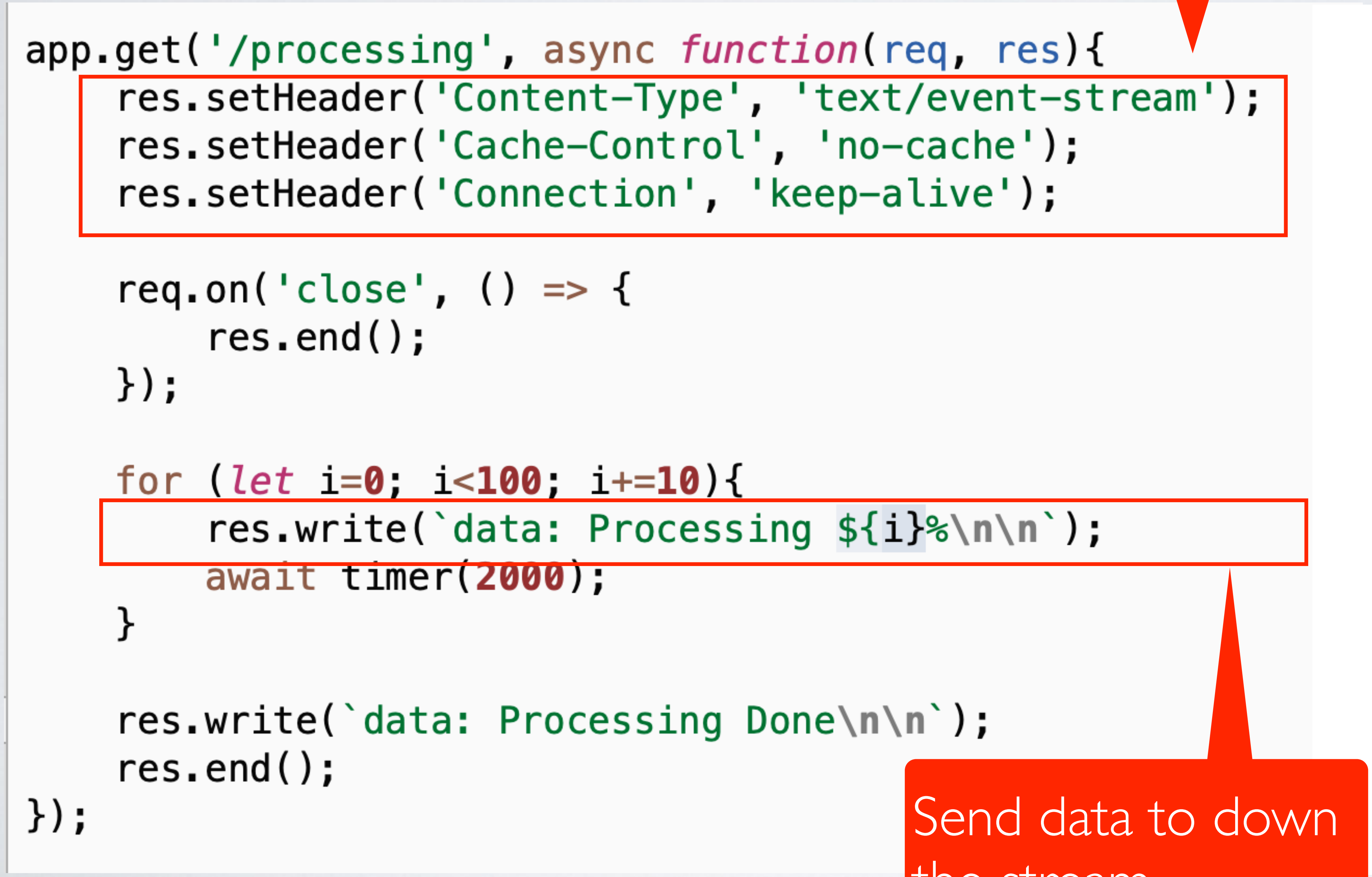
GET /update

Web App

Server-Sent Events

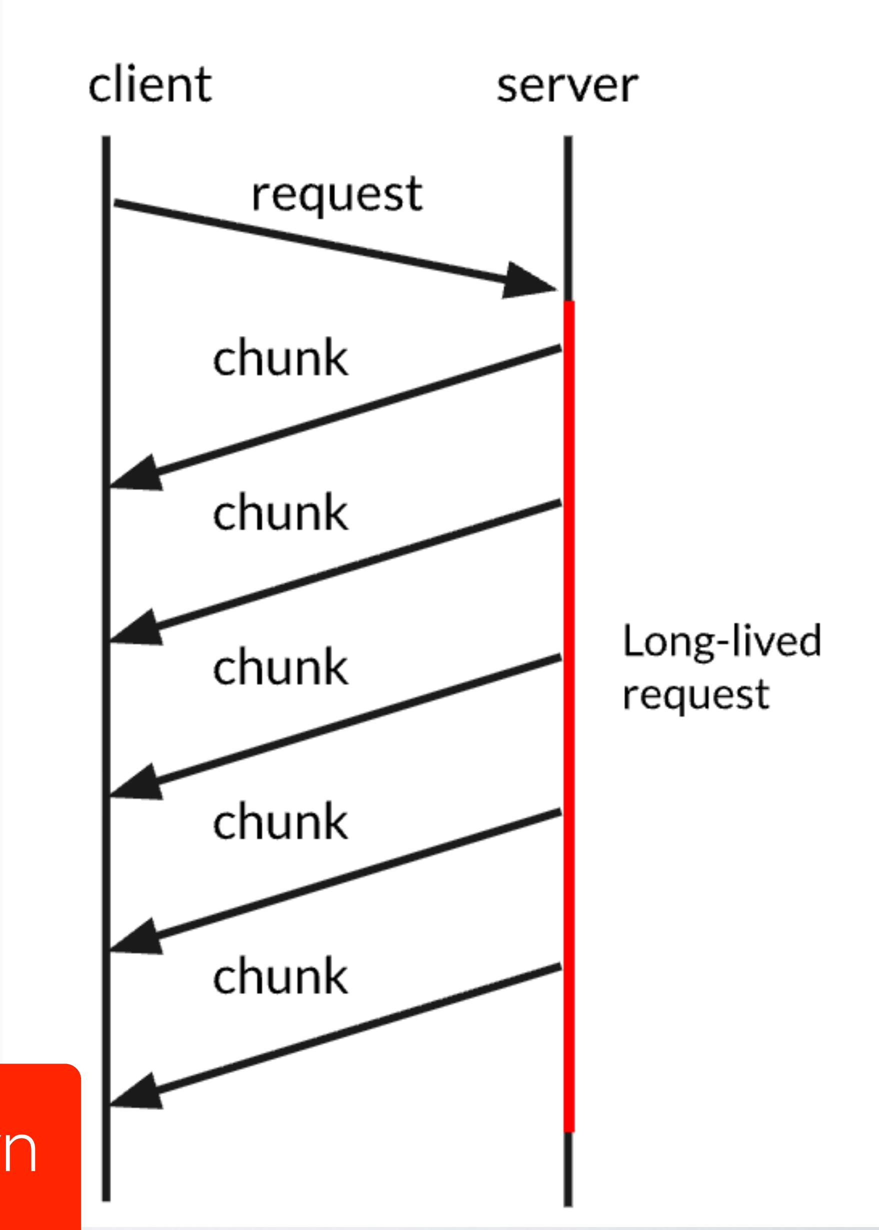
On the server side

Create stream and keep the connection opened

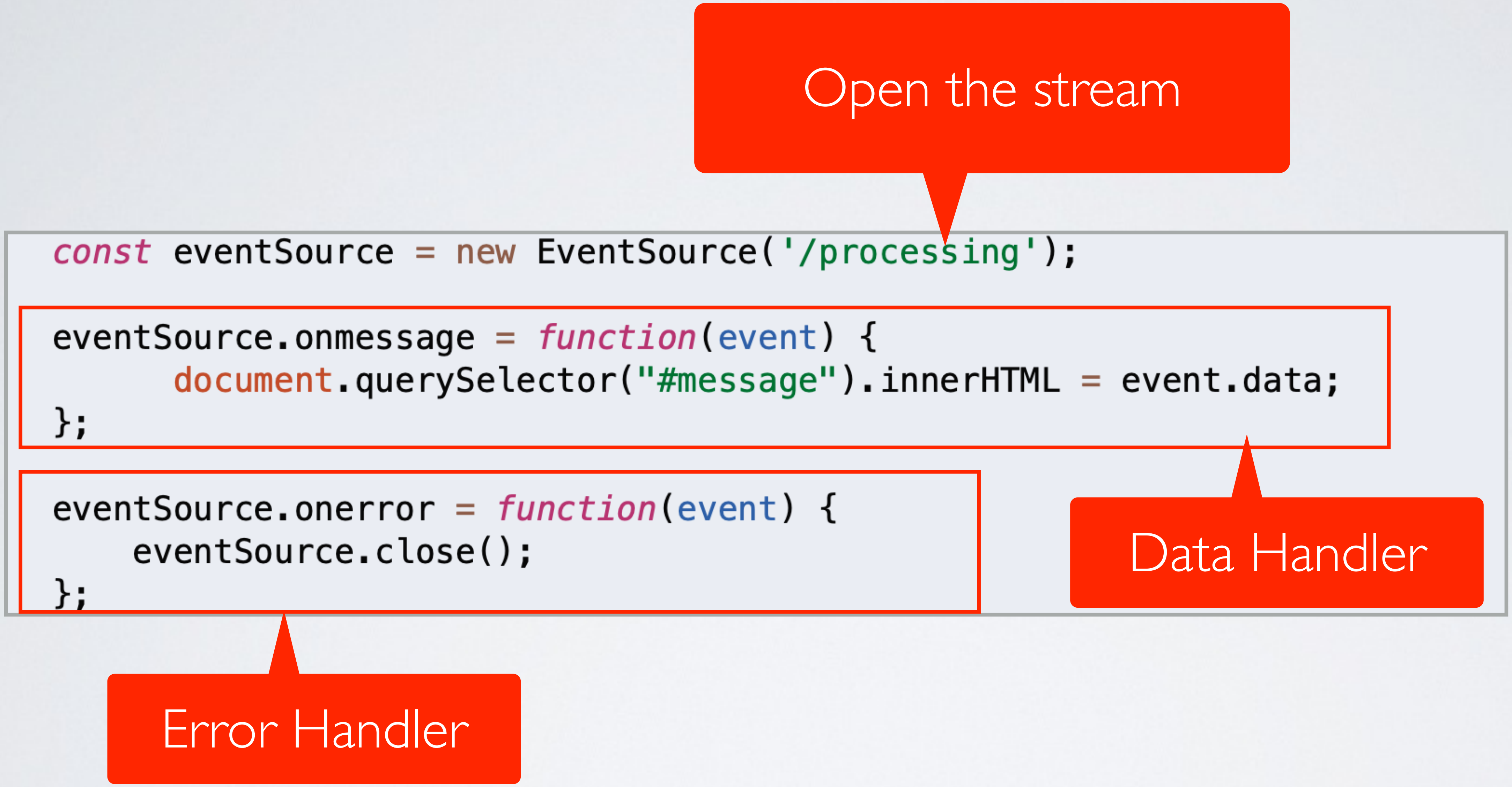


```
app.get('/processing', async function(req, res){
    res.setHeader('Content-Type', 'text/event-stream');
    res.setHeader('Cache-Control', 'no-cache');
    res.setHeader('Connection', 'keep-alive');

    req.on('close', () => {
        res.end();
    });

    for (let i=0; i<100; i+=10){
        res.write(`data: Processing ${i}%\n\n`);
        await timer(2000);
    }

    res.write(`data: Processing Done\n\n`);
    res.end();
});
```

Send data to down the stream



On the client side



```
const eventSource = new EventSource('/processing');

eventSource.onmessage = function(event) {
      document.querySelector("#message").innerHTML = event.data;
};

eventSource.onerror = function(event) {
    eventSource.close();
};
```

Open the stream

Data Handler

Error Handler

Web Sockets

The idea

➡ Full-duplex client-server communication

- Similar to low-level POSIX sockets
- Allow message to be broadcasted to all connected users
- Does not rely on HTTP at all (except for initialization)

Different Technologies

- Native Web Sockets
- Socket.io (popular)

Web RTC

Real-time communication for the web

The idea

➡ Full-duplex communication between clients (browsers) and servers (possibly)

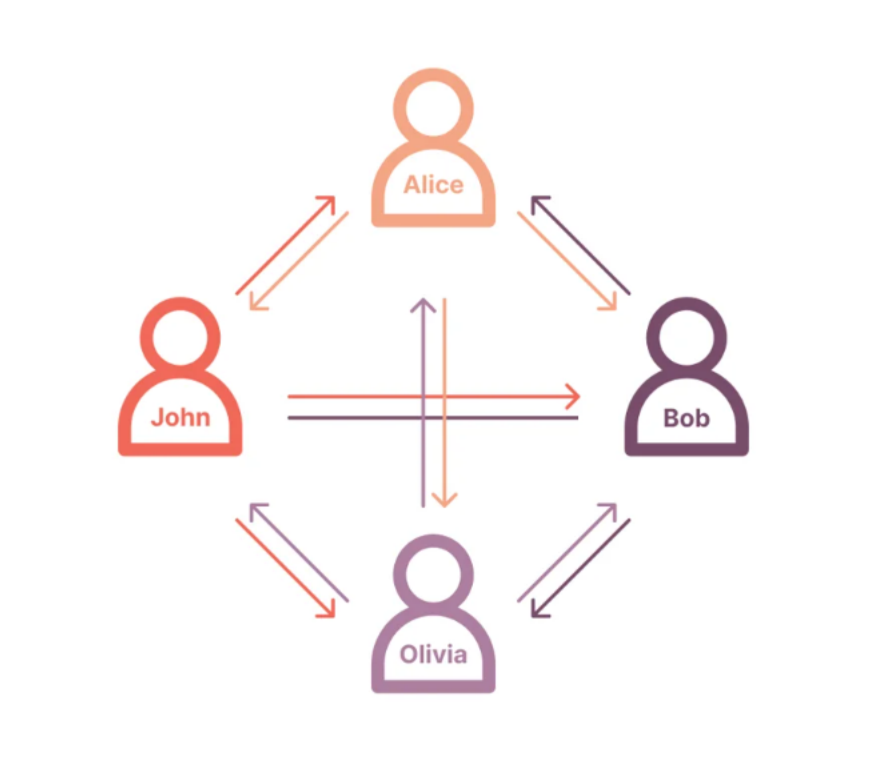
- Popular Libraries
  - Peer.js (data)
  - VideoSDK.live (audio/video/screenshare)

Different P2P Architecture

- P2P Mesh
- SFU (Selective Forwarding Unit)
- MCU (Multipoint Control Unit)

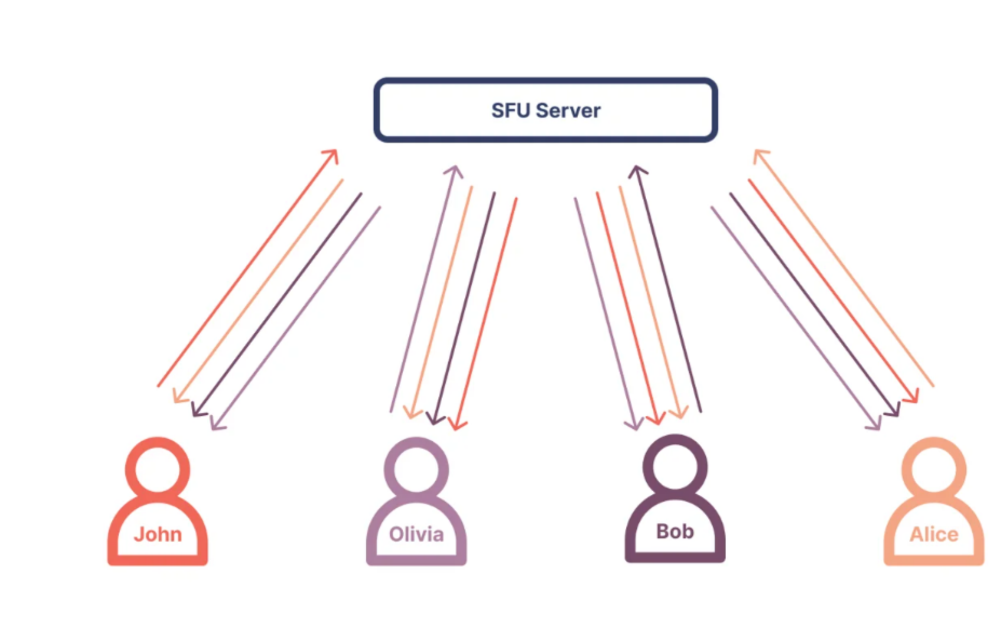
➡ https://www.digitalsamba.com/blog/p2p-sfu-and-mcu-webrtc-architectures-explained

P2P Mesh



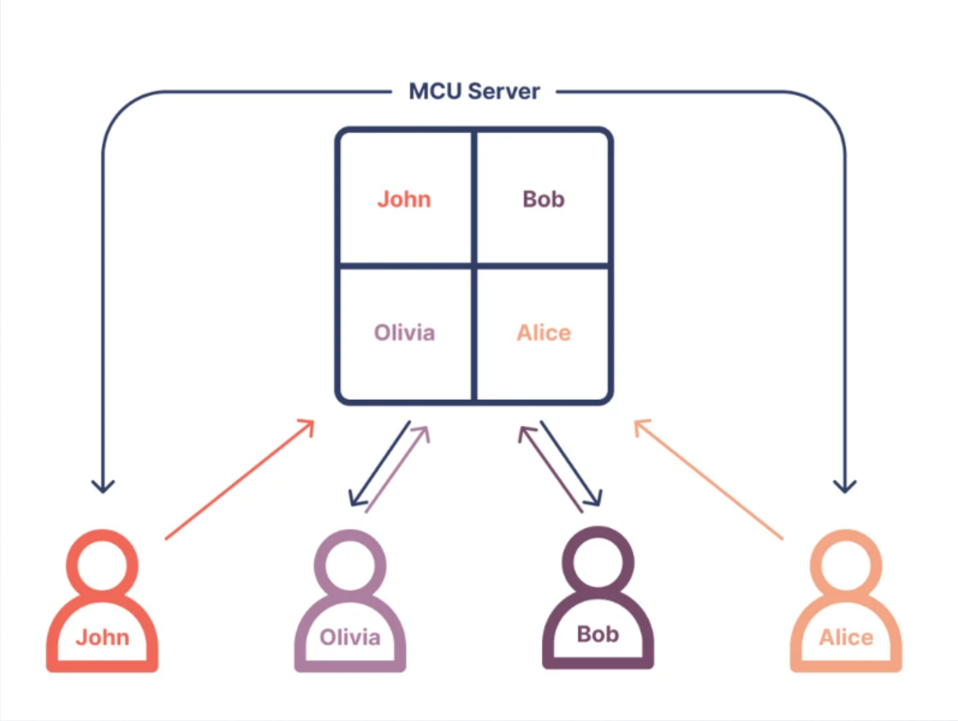
- ➡ Each peer broadcast messages to all other peers
- ✓ No server required and better privacy
- ◉ Worst scalability : requires additional client's bandwidth as the number of peers grows

SFU (Selective Forwarding Unit)



- ➡ Central server in charge of broadcasting messages to all peers
- ✓ Better scalability on client's side: 1 upload but n downloads (but the server can choose what to broadcast)
- ◉ Complexity on the server side:
  - Server's bandwidth increases with participants
  - Might need to ensure privacy (End-to-End Encryption required)
- ★ Popular Architecture for video conferencing applications

MCU (Multipoint Control Unit)



➡ Central server aggregates all streams into one (a.k.a mixer)

✓ Best scalability on client's side : 1 upload and 1 download

◉ Greater complexity on server's side:
mixing streams is a computing intensive task