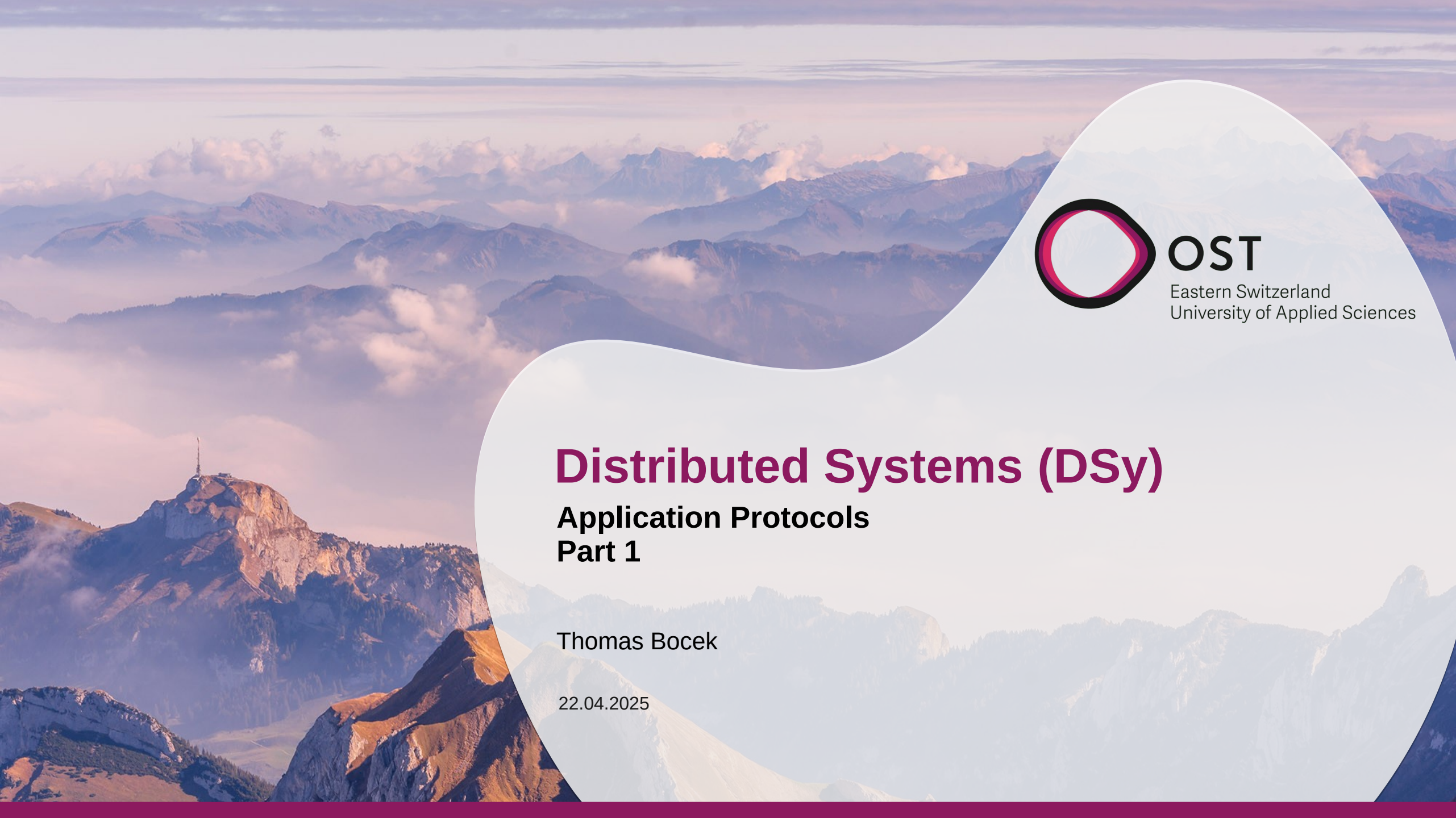


OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Application Protocols
Part 1

Thomas Bocek

22.04.2025

Learning Goals

- Lecture 7 (Application Protocols)
  - Overview over important protocols on layer 7
  - Part 1: custom protocols, ASN.1, RPC, HTTP, JSON, WebSockets, Server Sent Events

Protocols

- Protocols, lecture 9: layer 4
  - TCP, UDP, (QUIC/HTTP/3)
- Designing custom protocols (e.g. Kafka)

- Needs more time to develop / test

+ Can be more efficient (space/performance)

- Protocol generators (binary): Thrift / Avro / Protocol Buffers / (ASN1)

+ IDL (interface description language) generates code

+ Standard

- Has more overhead

e.g, Avro IDL - higher-level language for authoring Avro schemata → generates Avro schema

```
//Avro IDL
@namespace("ch.ost.i.dsl")

protocol MyProtocol{
  record AMessage {
     string request;
     int code;
  }
  record BMessage {
     string reply;
  }

  BMessage GetMessage(AMessage msg);
}
```

```
{"namespace": "ch.ost.i.dsl",
 "type": "record",  "name":
"AMessage",
 "fields": [
  {"name": "request", "type":
"string"},
  {"name": "code", "type": "int"}
 ]
}
```

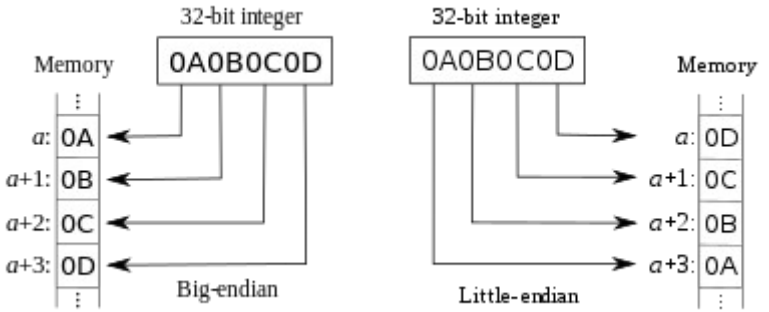

OST

Protocols

- Custom encoding/decoding
  - You control every aspect
  - You send more time on it

- Little-endian / Big-endian
  - sequential order where bytes are converted into numbers
- Networking, e.g. TCP headers: Big-endian
- Most CPUs e.g., x86: Little-endian, RISC-V: Bi-endianness

```
115     public static boolean decodeHeader(final ByteBuf buffer, final InetSocketAddress recipientSocket,
116             final InetSocketAddress senderSocket, final Message message) {
117         LOG.debug("Decode message. Recipient: {}, Sender:{}.", recipientSocket, senderSocket);
118         final int versionAndType = buffer.readInt();
119         message.version(versionAndType >>> 4);
120         message.type(Type.values()[(versionAndType & Utils.MASK_0F)]);
121         message.protocolType(ProtocolType.values()[versionAndType >>> 30]);
122         message.messageId(buffer.readInt());
123         final int command = buffer.readUnsignedByte();
124         message.command((byte) command);
125         final Number160 recipientID = Number160.decode(buffer);
126
127         //we only get the id for the recipient, the rest we already know
128         final PeerAddress recipient = PeerAddress.builder().peerId(recipientID).build();
129         message.recipient(recipient);
130
131
132         final int contentTypes = buffer.readInt();
133         message.hasContent(contentTypes != 0);
134         message.contentTypes(decodeContentTypes(contentTypes, message));
```



[source]

OST

Protocols Examples with Golang

- Example in repo DSy
  - TCP in C, golang, custom serialization
    → Anybody there? Example: 15 bytes

```
func main() {
   fmt.Println("connecting...")
   conn, _ := net.Dial(“tcp", "127.0.0.1:7000")
   defer conn.Close()
   buf := make([]byte, 15)
   buf[0]=5
   copy(buf[1:],[]byte("5Anybody there?"))
   _, _ = conn.Write(buf)
}
```

```
func main() {
   fmt.Println("listening...")
   tcpConn, _ := net.Listen("tcp", ":7000")
   conn, _ := tcpConn.Accept() //do this in a go routine
   b := make([]byte, 15)
   n, _ := conn.Read(b)
   fmt.Printf("connecting... read: %d, addr: %v, data: [%
x], decoded: %v\n",
      n, conn.RemoteAddr(), b[:n], string(b[1:]))}
```

- ASN1, defined in 1984. Standard interface description language (IDL), used for serializ / deserializ – used e.g., certs - details matter!
  - Example in golang

```
30 13 02 01 05 16 0e 41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f
```

```
30 – type tag indicating SEQUENCE
13 – length in octets of value that follows
  02 – type tag indicating INTEGER
  01 – length in octets of value that follows
    05 – value (5)
  16 – type tag indicating IA5String
     (IA5 is generally US-ASCII)
  0e – length in octets of value that follows
    41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f – value ("Anybody there?")
```

  - Example: 21 bytes, XML: 48 bytes → binary more efficent, XML is readable

```
<code>5</code>
<message>Anybody there?</message>
```


OST

Protocols Examples

- Avro: data serialization system
  - Remote procedure call and data serialization framework
  - Use: Hadoop (Big-data framework)
  - LinkedIn go-avro library
    - Define a message in JSON (benchmarks) or IDL – no code generation
  - Example: 16 bytes, assuming both have the same IDL
- Protobuf: data serialization from Google
  - IDL, goals: smaller and faster than XML

```
syntax = "proto3";
message AMessage {
  int32 code = 1;
  string message = 2;
}
```

  - Use: nearly all inter-machine communication at Google

```
m := &pb.AMessage{Id: 5, Message: "Anybody there?"}
out, err := proto.Marshal(m)
```

  - Example: 18 bytes
- Thrift - RPC Framework from Facebook
  - IDL and binary protocol

```
service TestService {
    void AMessage(1:i32 int, 2:string message)
}
```

  - Example with RPC: 49 bytes transferred
- CBOR / MessagePack
  - And even more...

OST

RPC Examples

- RPC - (Remote Procedure Call) lets programs execute code on remote machines.
  - E.g., gRPC, uses HTTP/2 for transport, uses Protocol Buffers
  - Features: authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts, many languages
  - Example: 171 / 124 (wireshark) – request/reply

```
syntax = "proto3";

service MessageService {
  rpc SendMessage (AMessage)
returns(Empty);
}

message AMessage {
  int32 code = 1;
  string message = 2;
}

message Empty {}
```

- JSON/REST/HTTP
  - Human-readable text to transmit data
  - Is it RPC? Yes/No
    - REST ideally stateless, RPC can be stateful
    - REST responses contain all metadata, RPC often needs interface definitions
  - Parsing overhead, JSON slower than binary protocol - benchmarks
  - Often used for web apps, example: 39 bytes

```
[
    {
        "id": "bitcoin",
        "name": "Bitcoin",
        "symbol": "BTC",
        "rank": "1",
        "price_usd": "9324.08",
        "price_btc": "1.0",
        "24h_volume_usd": "9039300000.0",
        "market_cap_usd": "158560288125",
        "available_supply": "17005462.0",
```


OST

Application Protocol: HTTP

- HTTP (HyperText Transfer Protocol): foundation of data communication for www
- Started in 1989 by Tim Berners-Lee
  - HTTP/1.1 published in 1997
  - HTTP/2 published in 2015
    - More efficient, header compression, multiplexing
  - HTTP/3 published in 2022
- Request / response (resource)
- HTTP resources identified by URL
  - https://dsl.i.ost.ch/design/ost_logo.svg

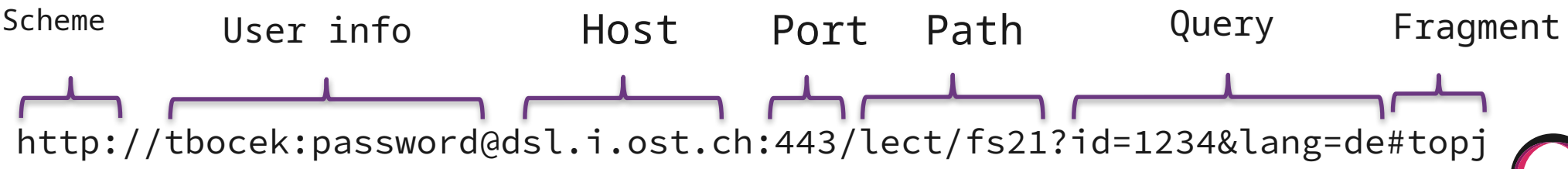
- Text-based protocol

```
openssl s_client -connect dsl.i.ost.ch:443 -showcerts
… TLS handshake …
GET /
```

- HTTP is a stateless protocol
  - Server maintains no state
- Browser sends a bit more…

Request Headers (359 B)

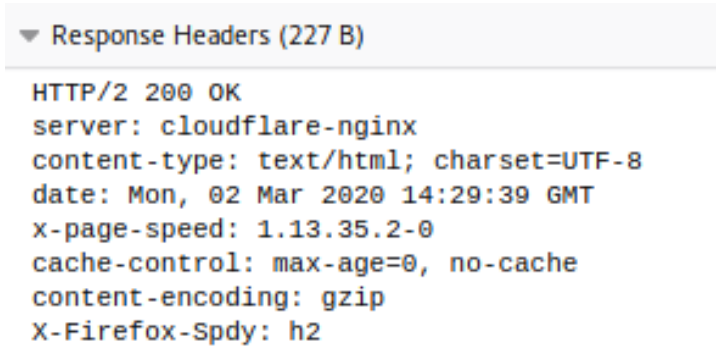
```
Host: dsl.hsr.ch
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
TE: Trailers
```



OST

Application Protocol: HTTP

- Response
  - Header



  - Status Code: 200
    - List: from 1xx (information response), 2xx (success) – 200 OK, 3xx (redirection), 4xx (client error), 404 Not Found, 403 Forbidden (access slides outside HSR), 5xx (server errors)
  - Content

```
<!DOCTYPE html>
<html>
<head>
    <title>Distributed Systems and Ledgers Lab</title>
    <link rel="stylesheet" type="text/css"
href="design/layout.css"/>
…
```

- Request methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH
- Web server one-liner - with netcat:
  - while true; do { echo -e "HTTP/1.1 200 OK\n\n<h1> Hallo" } | nc -vl -p 8080 -c; done
- Every Webbrowser has dev tools to show request / responses
  - Firefox, Chrome: ctrl+shift+i / F12
  - Used regularly

OST

WebSockets

- Full-duplex communication over TCP [overview]
  - REST / JSON is in one direction
- How can the server notify the browser (client?)
  - Polling
    - Short: request e.g. every 0.5s
    - Long: request until timeout or reply
  - Server Sent Events (alternative) SSE
    - One way communication from server to browser (client)
    - Server receives a regular HTTP request, keeps connection open (`Accept or Content-Type: text/event-stream`), server can now push data to the client
  - WebSockets
    - Two way communication, RFC 6455

- HTTP handshake, then upgrade to communication channel

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Server response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

  - Data can be text or binary
- With SSL/TLS → wss://
  - Some configuration required on LBs / RRs

OST

WebSockets / SSE

- GoLang / HTML/JS Example
  - https://github.com/tbocek/DSy
  - Many languages supports WebSockets

```
<script type="text/javascript">
    const connection = new WebSocket('ws://localhost:8080/ws');
    connection.onopen = function () {
        connection.send('start');
    };
    let counter = 0;
    connection.onmessage = function (e) {
        console.log('update websocket: ' + counter++);
        const date = JSON.parse(e.data);
        const element = document.getElementById('text');
        element.innerHTML = 'Time: ' + date.now;
    };
</script>
```

- Server Sent Events Example
  - Uses standard HTTP connections
  - Native browser support via EventSource API
    - Double newline (\n\n) marks end of each message
  - Automatic reconnection if connection is lost
  - Max number of concurrent connections per domain

```
<script type="text/javascript">
    const evtSource = new EventSource("/sse");
    let counter = 0;

    evtSource.onmessage = function(event) {
        console.log('update SSE: ' + counter++);
        const date = JSON.parse(event.data);
        const element = document.getElementById("text");
        element.innerHTML = 'Time: ' + date.now;
    };

    evtSource.onerror = function() {
        console.log('SSE connection error');
        evtSource.close();
    };
</script>
```


OST