HTTP/2 and gRPC

References for study:

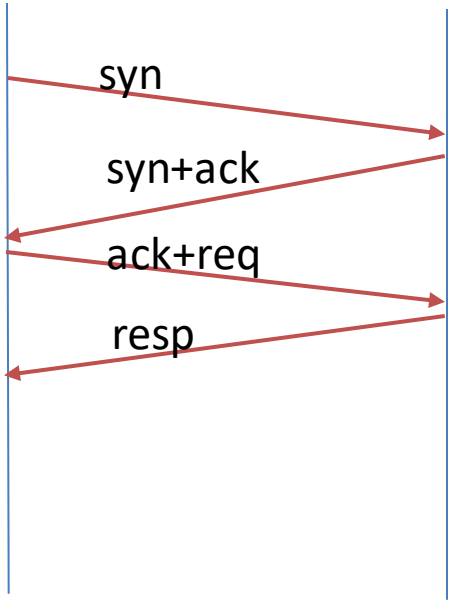
- Ilya Grigorik, "High Performance browser Networking", chapter 12 (https://hpbn.co/http2/)
- gRPC Tutorials (https://grpc.io/docs/tutorials/)
- Protocol Buffers tutorials (https://protobuf.dev/programming-guides/)

HTTP/2 and HTTP/3

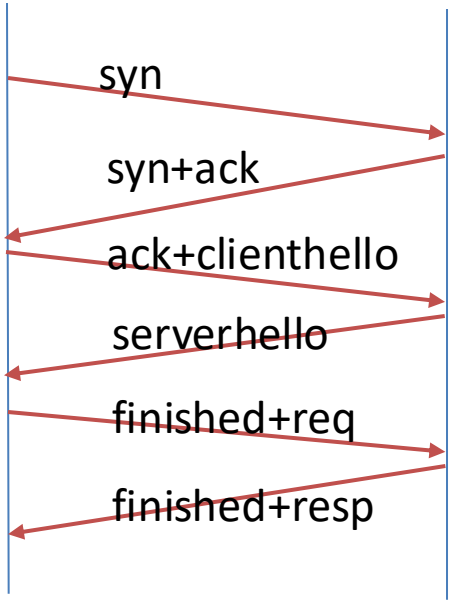
- Evolutions of HTTP for better efficiency/performance
  - HTTP/2: optimized for networks with low error rates (e.g., within data centers)
  - HTTP/3: optimized to reduce latency for long distance or high error rate connections
- Same semantics as HTTP/1.1, but different data encoding and use of transport protocols
  - No impact on existing REST APIs => they can be ported to HTTP/2 and HTTP/3 smoothly

TCP/HTTP1.1 Performance Issues

- High latency for establishing connections



HTTP: 2RTT



HTTPS:3RTT

- Multiplexing requires opening multiple TCP connections
- Character encoding is not very efficient

HTTP/2

- **HTTP/2**: Initially driven by Google (SPDY), later on taken over by IETF (RFC 7540)

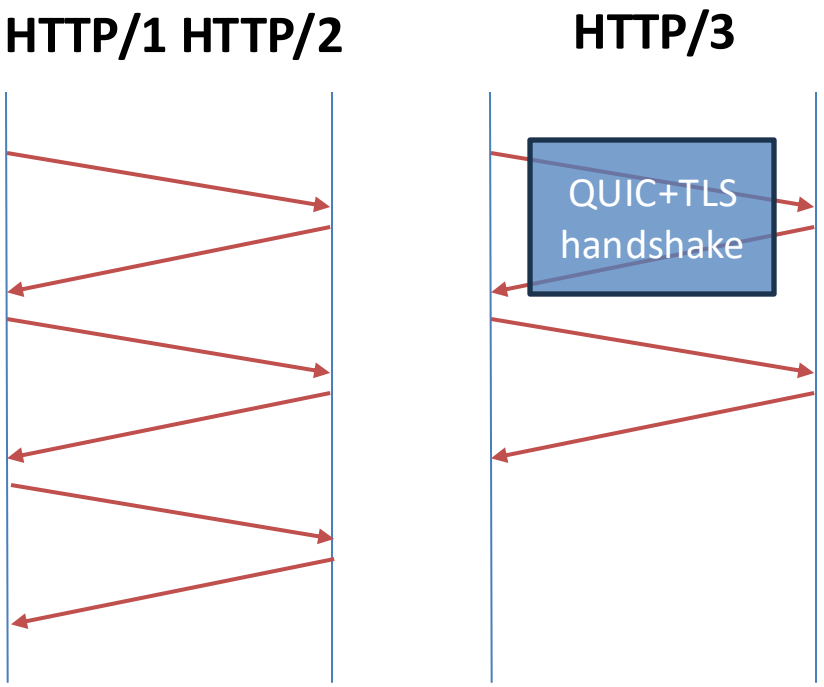
	HTTP/2	HTTP/1.1
data encoding	binary	character-oriented
multiplexing	on single TCP connection	on multiple TCP connections
compression	header and body	only body
push possibility	yes	no

HTTP/3

- RFC 9114 (June 2022)
- TCP replaced by QUIC+UDP

HTTP/1 HTTP/2
(TLS)
TCP
IP

HTTP/3
QUIC (+TLS)
UDP
IP

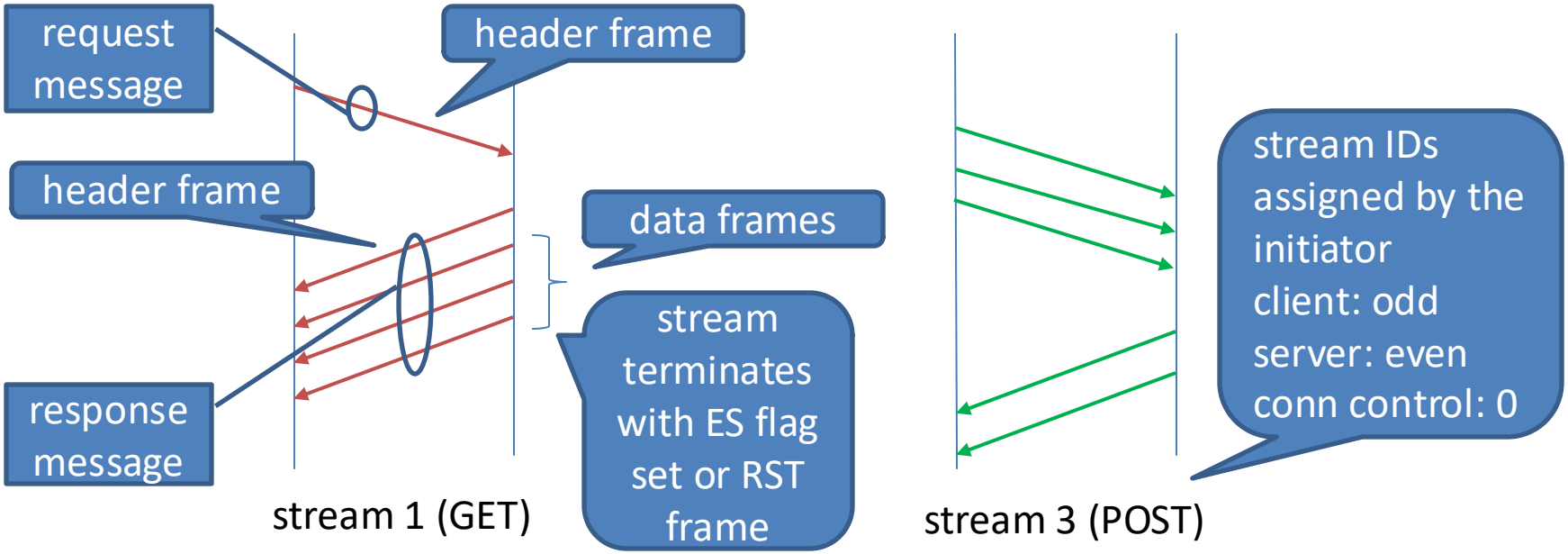


HTTP/2 HTTP/3
Adoption and Performance

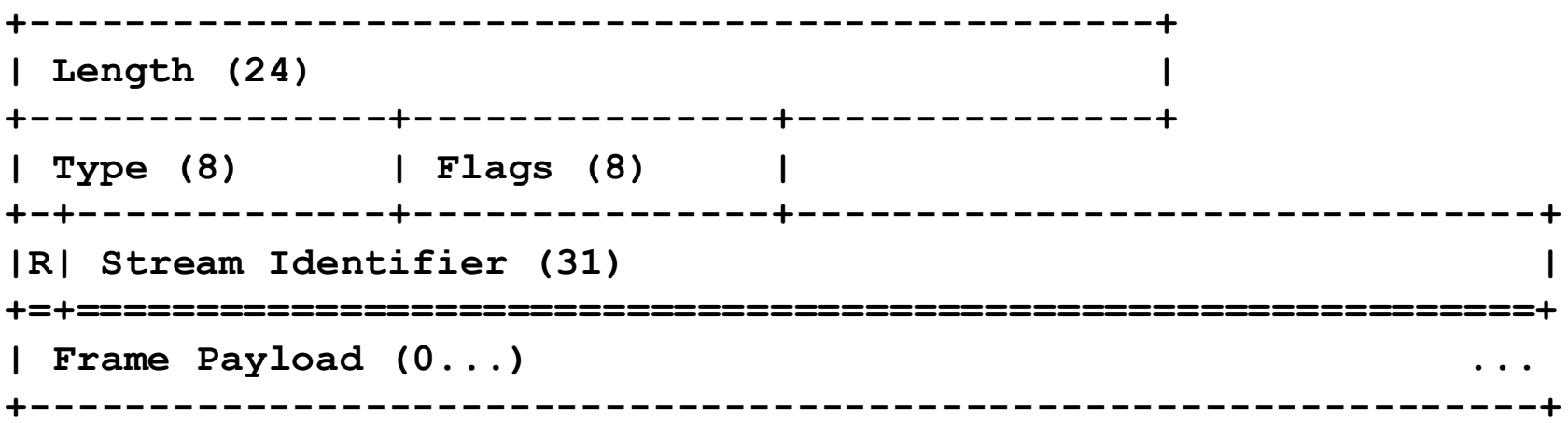
- Adoption in 2021 (Web Almanac, Percentage of Page loads):
  - HTTP/1 40%
  - HTTP/2 45% (but push adoption <1%)
  - HTTP/3 15%
- Performance:
  - HTTP/2: big boost wrt HTTP/1
  - HTTP/3: better than HTTP/2 only for long distance or high error rate/unrealiable connections

HTTP/2: Frames and Streams

- Frame: binary message exchanged by client and server
- Stream: independent, bidirectional sequence of frames
  - corresponds to an ordered sequence of logical operations (Request/Response, Push), i.e., of HTTP messages

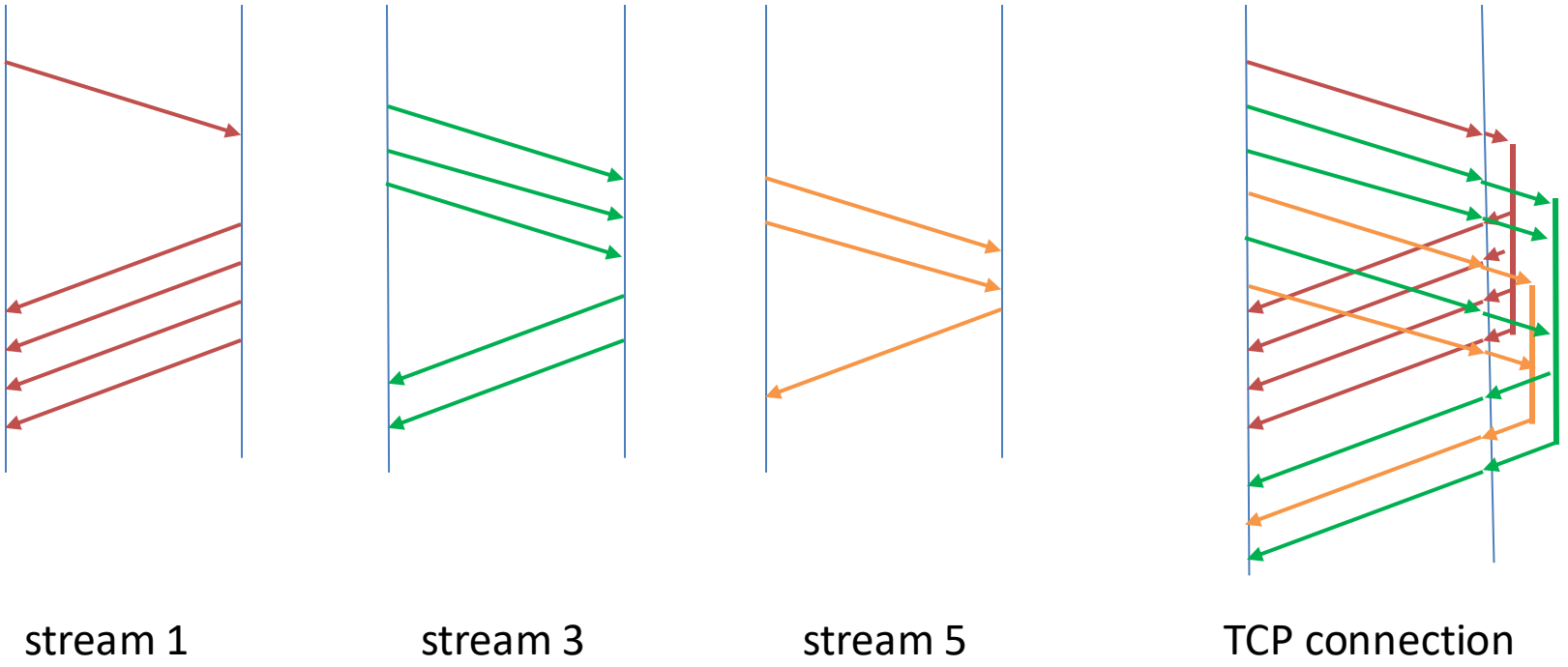


Frame Layout



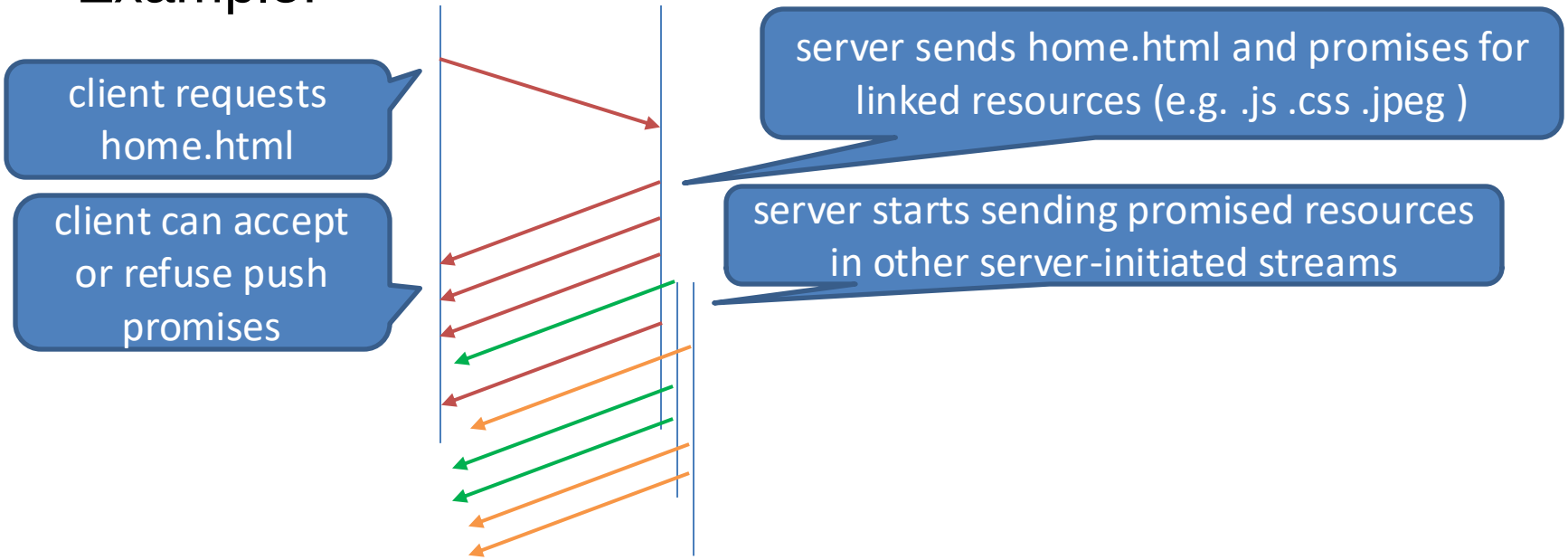
Stream Multiplexing

- Streams are multiplexed on a single TCP connection
  - ordering of the frames of each stream preserved
  - window-based flow control added to regulate stream rates

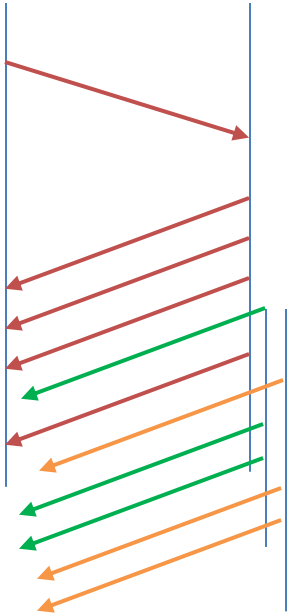


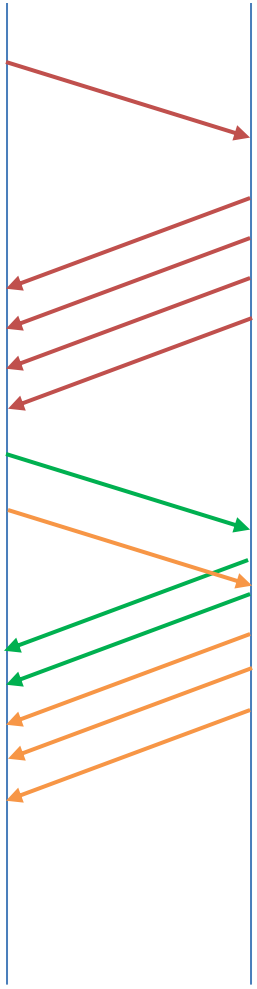
Push Mechanism

- A server can push additional representations, anticipating client's requests
  - a way to trade bandwidth for latency
- Example:



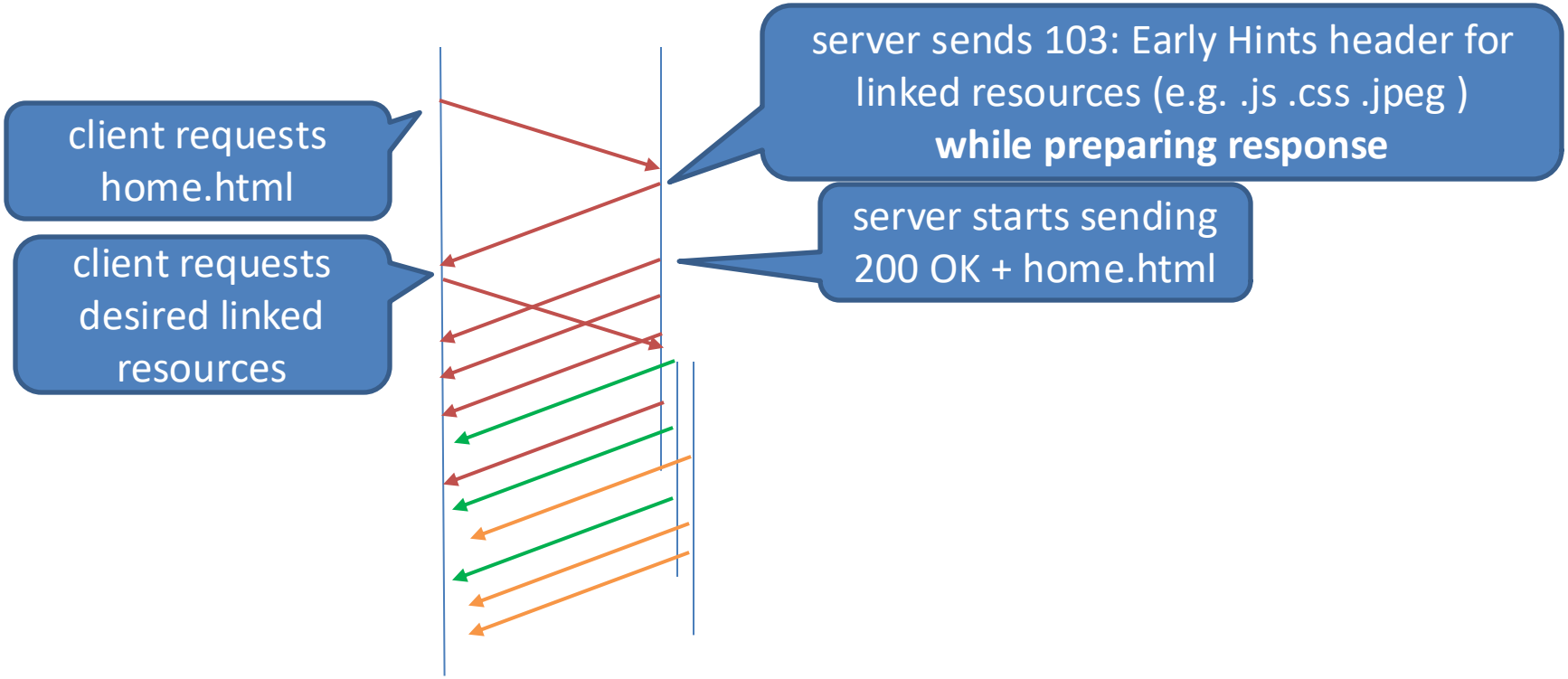
Push vs No-Push





Early Hints

- Alternative to server push which saves bandwidth
- Example:



Main Frame Types

- HEADERS
- DATA
- SETTINGS
- RST_STREAM — signal termination of a stream
- PUSH_PROMISE
- GOAWAY — initiate (graceful) connection termination
- WINDOW_UPDATE — flow control message

Initiating an HTTP/2 Connection

- If the client knows the server supports HTTP/2
  - After initiating the TCP connection, the client sends the 24-bytes preface string ("PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n") followed by a SETTINGS frame
  - The server responds with its preface+SETTINGS
  - Streams are opened
- If the client has no prior knowledge, HTTP/2 has to be negotiated

HTTP/2 Negotiation

- TLS-based Negotiation (HTTPS)
  - HTTP/2 can be negotiated via the TLS **Application Layer Protocol Negotiation (ALPN)**
    - client requests one of a list of protocols (including HTTP/2)
    - server selects one and informs the client in the response
    - if HTTP/2 has been selected, HTTP/2 handshake occurs as in the previous case (preface + SETTINGS exchange)
- HTTP-based Negotiation (HTTP)
  - HTTP/2 can be negotiated via the **HTTP1.1 Upgrade** Mechanism

HTTP/2 Upgrade

– The client starts an HTTP/1.1 connection with upgrade header:

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64 encoded HTTP/2 SETTINGS>
```

– The server responds with:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c
```

h2c: HTTP/2, no TLS
Similar procedure for h2: HTTP/2 over TLS

– The conversation continues as in the previous cases (preface + SETTINGS exchange)

Error Management

- In addition to the HTTP/1 error reporting mechanisms (Response-level Error Reporting), HTTP/2 offers other error reporting mechanisms
  - Stream-level Errors occurring in the management of single streams
  - Connection-level Errors occurring in the management of the TCP connection
- The result is a more precise report, especially about non-completed requests

Stream-level and Connection-level Error Management

- Two types of errors:
  - connection-level error (the TCP connection is compromized)
  - stream-level error (only one stream is compromized)
- Error codes:
  - NO_ERROR (0x0)
  - PROTOCOL_ERROR (0x1) generic protocol error
  - INTERNAL_ERROR (0x2)
  - FLOW_CONTROL_ERROR (0x3) flow control protocol violation
  - SETTINGS_TIMEOUT (0x4)
  - STREAM_CLOSED (0x5)
  - FRAME_SIZE_ERROR (0x6)
  - ...

Other ways to learn about status of Non-completed Requests

- The GOAWAY frame indicates the highest stream number that might have been processed.
  - Requests on streams with higher numbers are therefore guaranteed to be safe to retry.
- The REFUSED_STREAM error code can be included in a RST_STREAM frame to indicate that the stream is being closed prior to any processing having occurred.
  - Any request that was sent on the reset stream can be safely retried.

gRPC

- RPC mechanism that works on top of HTTP/2
  - exploits performance-oriented HTTP/2 features
    - binary messages, header compression, etc.
  - uses HTTP/2 as transport (like SOAP)
  - typical data representation: protocol buffers
- multi-language
- at most once semantics
- security management (e.g. via TLS)
- multiple ways of operation
  - synchronous, asynchronous, stream-oriented

Protocol Buffers

- System-independent data representation and IDL
  - abstract language
  - binary representation
- Bindings for all major programming languages

Protocol Buffers IDL v3

- C-like language (termination, blocks, comments)
- Example

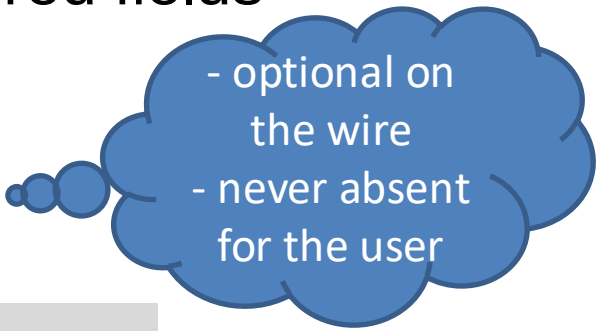
```
syntax = "proto3";

service HelloService {
    rpc SayHello (HelloRequest) returns (HelloResponse);
}

message HelloRequest {
    string greeting = 1;
}

message HelloResponse {
    string reply = 1;
}
```

Scalar Data Types

types	description
float, double	single/double precision floating point
int32, int64	variable-length encoding integer, inefficient for negative values
uint32, uint64	variable-length encoding unsigned integer
sint32, sint64	variable-length encoding signed integer, more efficient for negative values
fixed32, fixed64	fixed-length integer
sfixed32,sfixed64	signed fixed-length integer
bool	boolean
bytes	byte sequence (no longer than 2^{32})
string	UTF-8 encoded or 7-bit ASCII string (no longer than 2^{32})

Message Types

- User-defined named data structures (a message to be used for RPC or part of it)
- Collection of typed, named, and numbered fields
  - each field can be
    - optional singular (default): 0 or 1 repetitions
    - optional repeated: 0 or more

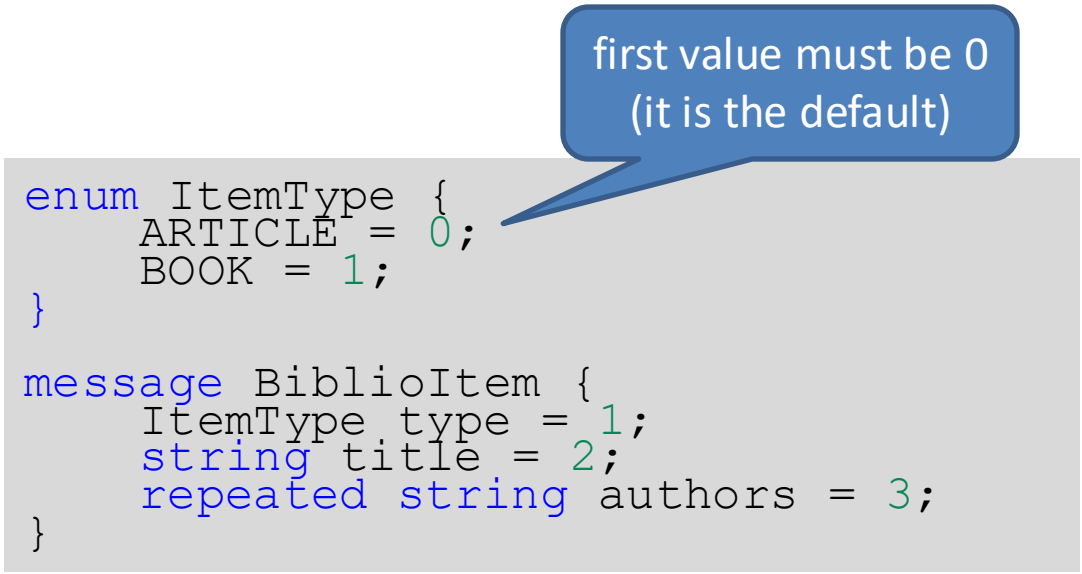



```
message Book {
     string isbn = 1;
     string title = 2;
     repeated string authors = 3;
}
```

  - if a field is absent, its default value will be returned when the message will be read

Enum Types

- Integers with variable-length encoding that can take a limited, well-defined set of named values



Default Values

Type	Default Value
numeric types	0
bool	false
bytes	the empty sequence of bytes
string	the empty string
enum types	the first value (0)
message types	unset (language dependent)

Map Types

- Shortcut syntax for associative maps

```
message Biblio {
    map<string, BiblioItem> items = 1;
    string description = 2;
}
```

  - map fields cannot be repeated
  - entry ordering is not relevant (it is not preserved on the wire)
  - field map<key_type,value_type> map_field = N; is equivalent to

```
message MapFieldEntry {
  key_type key = 1;
  value_type value = 2;
}

repeated MapFieldEntry map_field = N;
```

Oneof

- A set of fields such that at most one of them can be set to a nondefault value
  - the fields in the set share space on the wire

```
message BiblioItem {
    string title = 1
    repeated string authors = 2;
    oneof extension {
        Article article = 3;
        Book book = 4;
    }
}
```

  - setting one field in the set will automatically unset all the others
  - oneof fields cannot be **repeated** (and cannot be **maps**)

Managing Large Specifications

- Import feature

```
import "biblio.proto" ;
```

- Packages
  - A proto file can have a package declaration

```
package it.polito.dsp.biblio;
message Biblio {... }
```

  - Name clashes can be avoided by using package declarations and package specifiers

```
import "biblio.proto";
message Topic {
    it.polito.dsp.biblio.Biblio biblio = 1;
    ...
```

Nesting

- Message and Enum type declarations can be nested inside other Message types
- Nested declarations can be referred by the dot notation

```
message BiblioItem {
    message Article {...}
    message Book {...}

    string title = 1
    repeated string authors = 2;
    oneof extension {
        Article artcle = 3;
        Book book = 4;
    }
}
message Articles {
    BiblioItem.Article article;
    ...
}
```

RPC Types Supported by gRPC

- Classical Synchronous RPC
- Two-Way Asynchronous RPC
- Streaming mode (independently for request and response)

Defining RPC Interfaces in Proto Files

- An RPC interface can be defined by a **service** block
  - each RPC procedure has its **rpc** declaration
    - Request message type
    - Response message type
    - **stream** qualifier can precede request or response type

```
service UpdateService {
 rpc update(UpdRequest) returns (UpdResponse);
 rpc updateCS(stream UpdRequest) returns (UpdResponse);
 rpc updateSS(UpdRequest) returns (stream UpdResponse);
 rpc updateDS(stream UpdRequest) returns (stream UpdResponse);
}
```

Example

- Interface Definition for the Bank Account Example:
  - addDeposit and addWithdrawal operations