COMET, HTML5 WEBSOCKETS

OVERVIEW OF WEB BASED SERVER PUSH TECHNOLOGIES

Peter R. Egli
peteregli.net

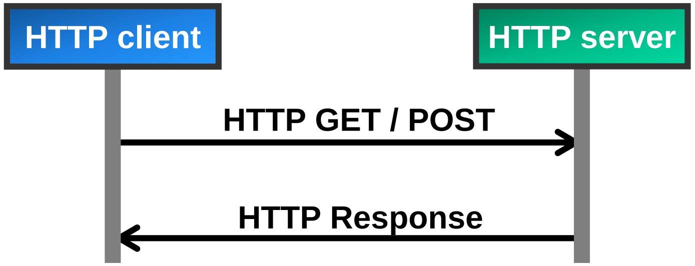
Contents

1. Server push technologies
2. HTML5 server events
3. WebSockets
4. Reverse HTTP
5. HTML5 overview

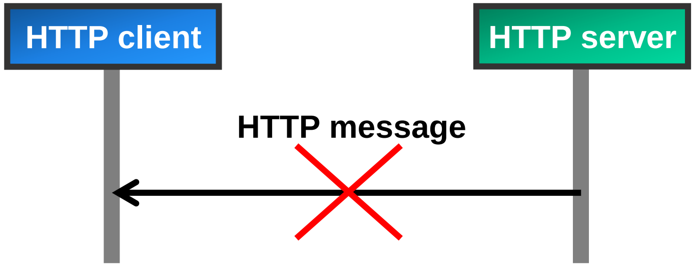
1. Server push technologies (1/7)

Problem:

HTTP does not provide full bidirectional connections between client and server.
HTTP is a request-response protocol. Asynchronous messages from the HTTP server to the client (server push) are not directly possible.



Every message from the server to the client requires a request beforehand.



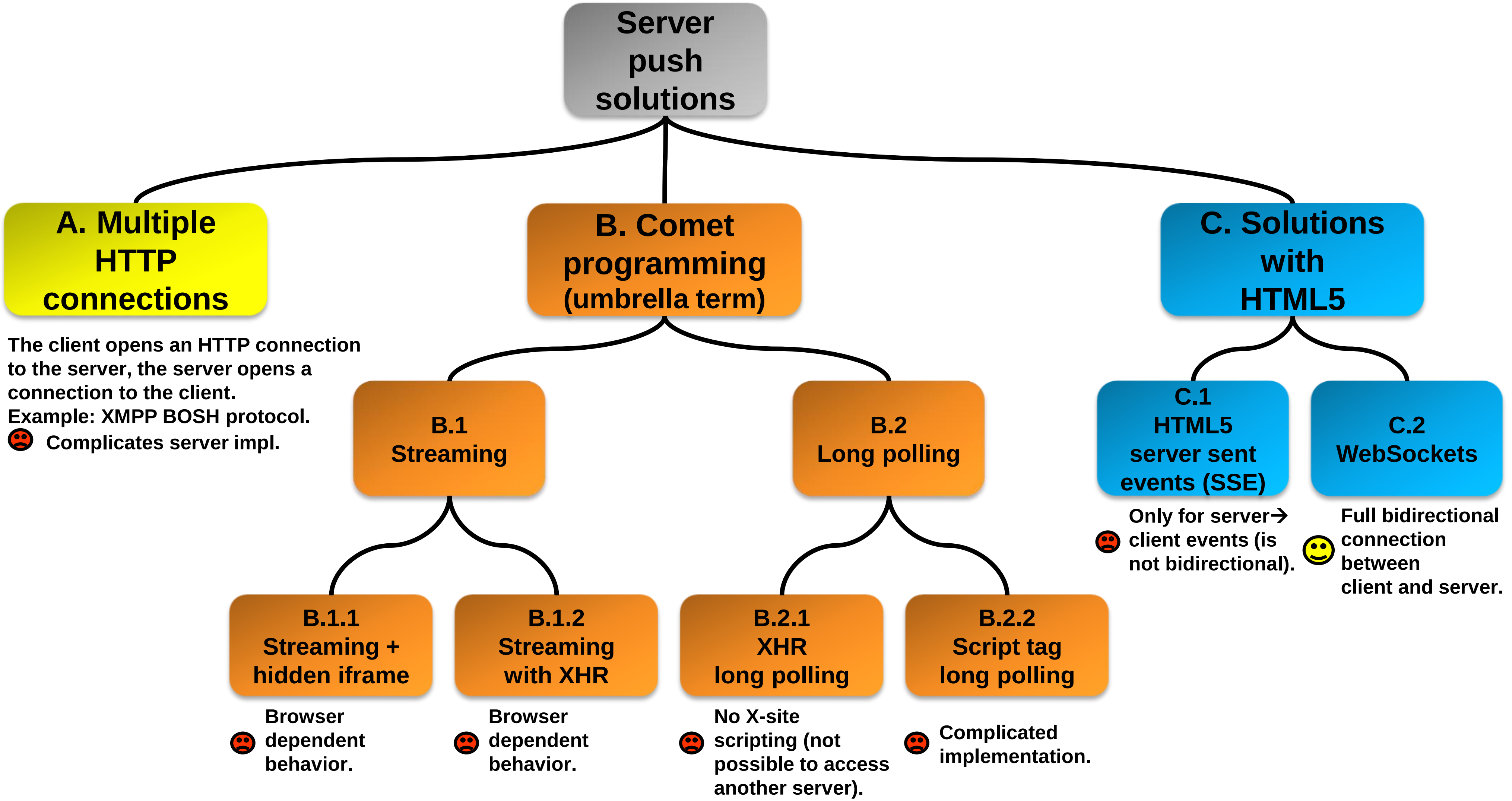
The HTTP protocol does not allow sending unsolicited messages from the server to the client.

HTTP was intentionally designed as a simple request-response protocol.
This request-response scheme makes the server implementation simple, i.e. the server is stateless and thus does not have to maintain session state for each client.

Over time different work-around techniques /solutions emerged to overcome this 'limitation' of HTTP.

1. Server push technologies (2/7)

Overview of solutions for server-push (1/3):



Server push solutions

A. Multiple HTTP connections

The client opens an HTTP connection to the server, the server opens a connection to the client.
Example: XMPP BOSH protocol.
☹ Complicates server impl.

B. Comet programming (umbrella term)

B.1 Streaming

B.1.1 Streaming + hidden iframe

☹ Browser dependent behavior.

B.1.2 Streaming with XHR

☹ Browser dependent behavior.

B.2 Long polling

B.2.1 XHR long polling

☹ No X-site scripting (not possible to access another server).

B.2.2 Script tag long polling

☹ Complicated implementation.

C. Solutions with HTML5

C.1 HTML5 server sent events (SSE)

☹ Only for server→ client events (is not bidirectional).

C.2 WebSockets

☺ Full bidirectional connection between client and server.

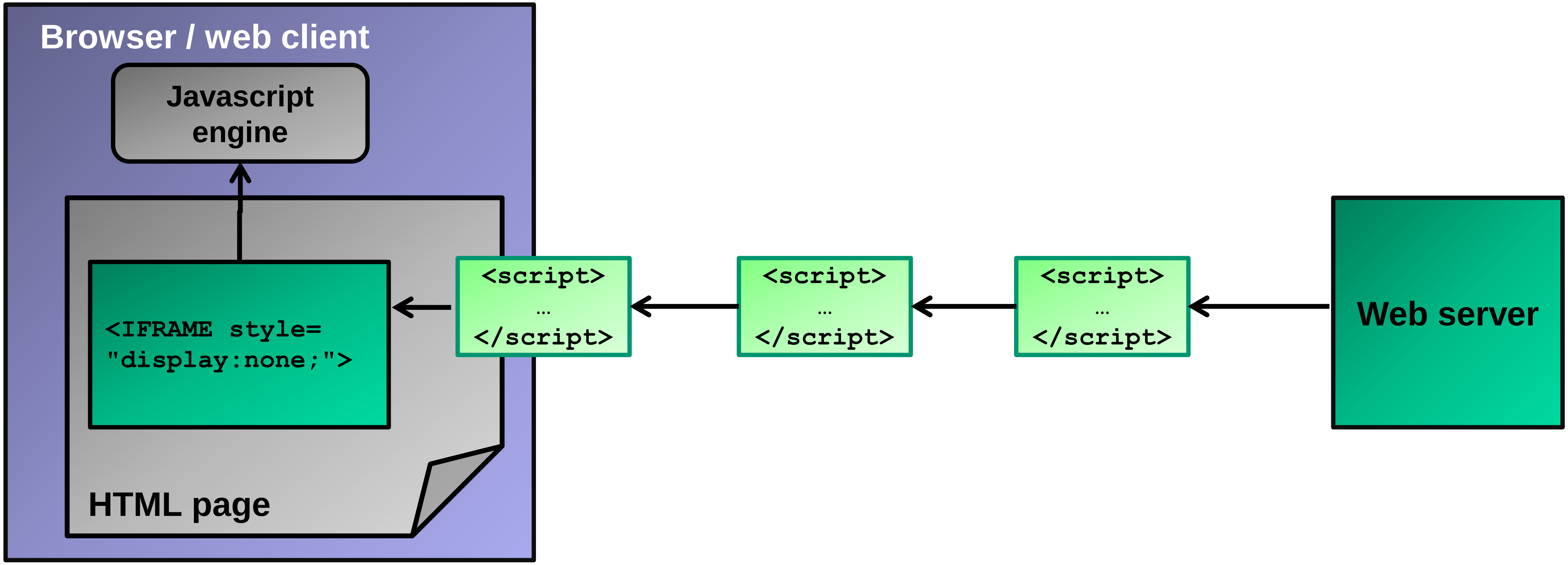
1. Server push technologies (3/7)

B.1.1 Streaming with hidden iframe (aka «forever frame»)

The HTML page contains a hidden IFRAME element.

This IFRAME element is filled with a chunked HTML page (the page is thus indefinitely long, the server can asynchronously send updates to the client).

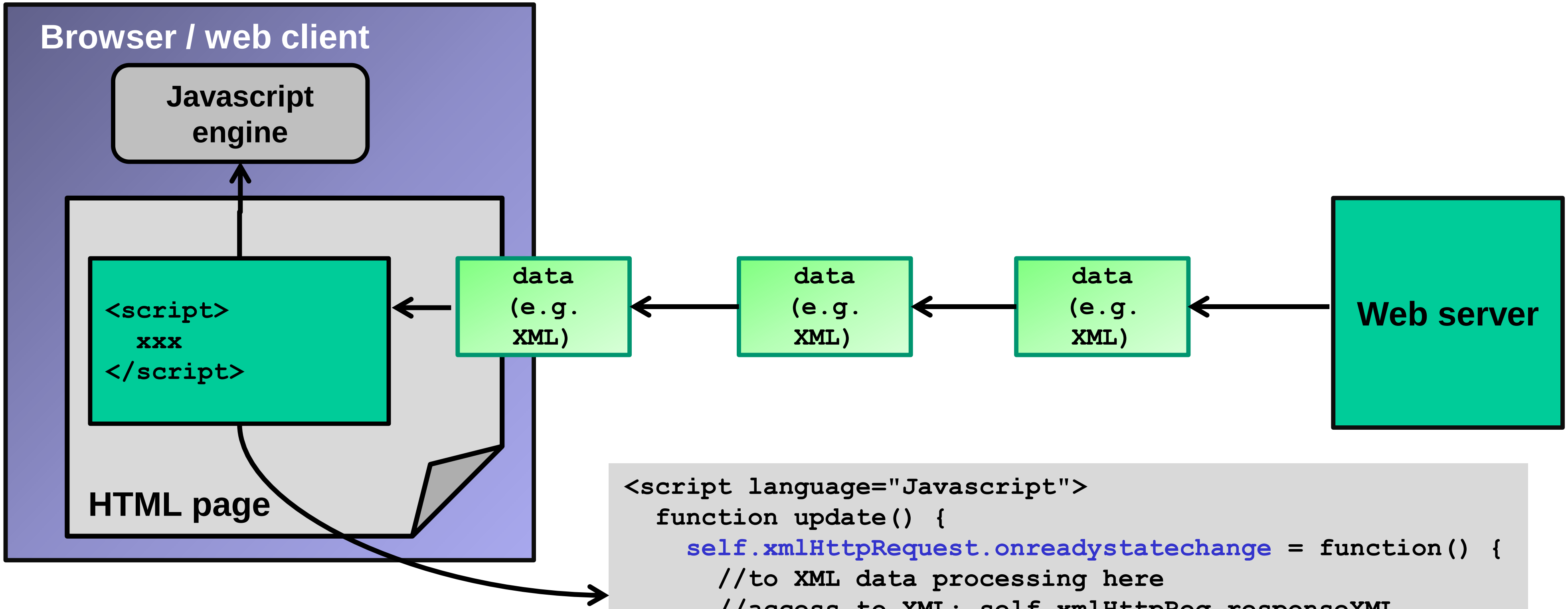
The server updates contain HTML script tags with Javascript code that are asynchronously executed on the client.



1. Server push technologies (4/7)

B.1.2 Streaming with XHR

The XMLHttpRequest object in Javascript provides access to the HTTP connection in the client. When a message arrives, the Javascript engine calls the `onreadystatechange` function.

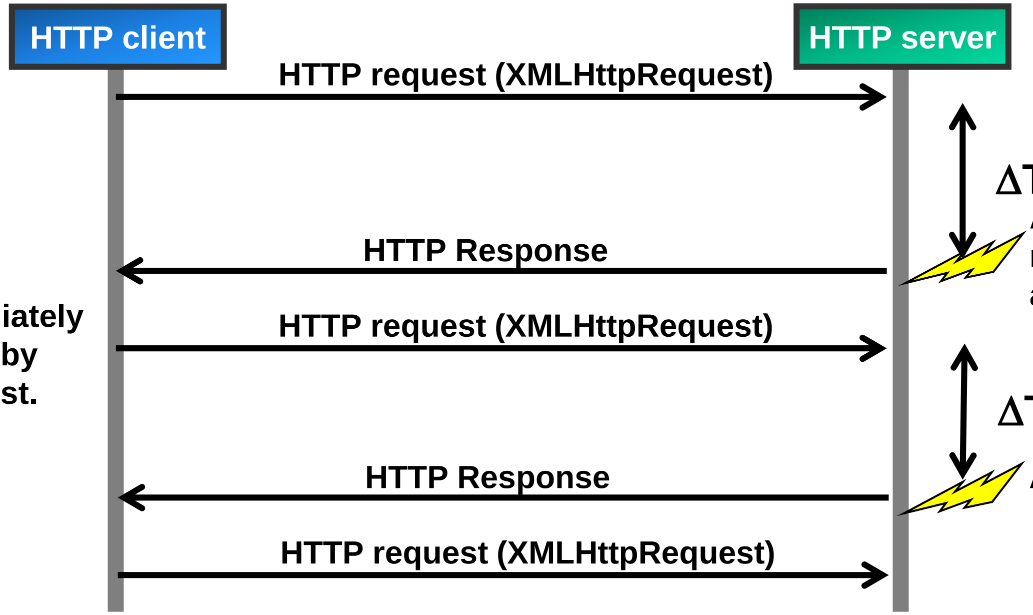


```
<script language="Javascript">
  function update() {
    self.xmlHttpRequest.onreadystatechange = function() {
      //to XML data processing here
      //access to XML: self.xmlHttpReq.responseXML
    }
}
</script>
```

1. Server push technologies (5/7)

B.2.1 XMLHttpRequest with long polling

The client sends requests through the XMLHttpRequest Javascript object.
After each response, the client "rearms" the polling by sending a new request.


HTTP client
HTTP server
HTTP request (XMLHttpRequest)
ΔT
HTTP Response
HTTP request (XMLHttpRequest)
ΔT
HTTP Response
HTTP request (XMLHttpRequest)

An event on the server (data changed) requires the server to send an asynchronous update to the client.

Client (script) immediately "rearms" the polling by sending a new request.

Another server-side event

B.2.2 Script tag long polling:

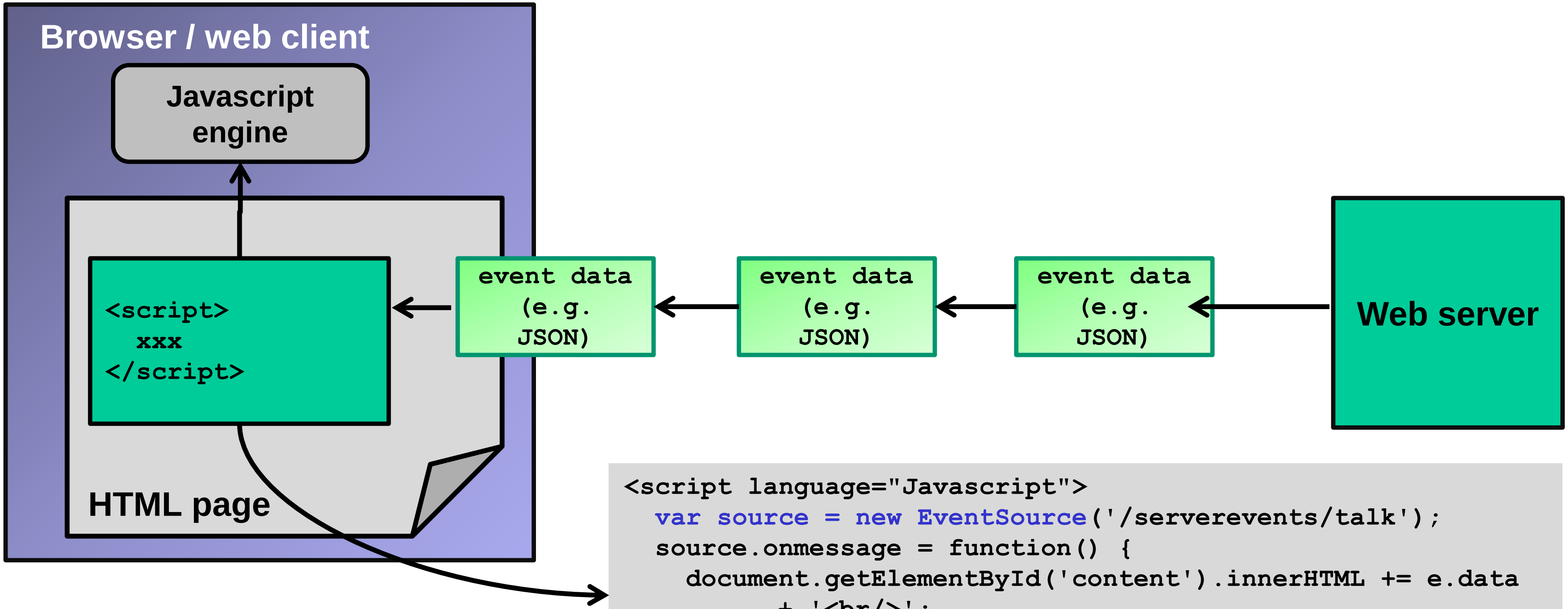
Same techniques as above, but the script code is embedded in `<script>` tags that can be filled with code from another second level domain.
Possible across different second level domains (X-site scripting), but poses a security problem.

1. Server push technologies (6/7)

C.1 HTML5 server sent events (SSE):

HTML5 server sent events (SSE) are similar to web sockets. SSE allow the server to asynchronously send data to the client.

SSE differ from web sockets in that they only allow server→client communication.



```
<script language="Javascript">
  var source = new EventSource('/serverevents/talk');
  source.onmessage = function() {
    document.getElementById('content').innerHTML += e.data
          + '<br/>';
  }
}
</script>
```

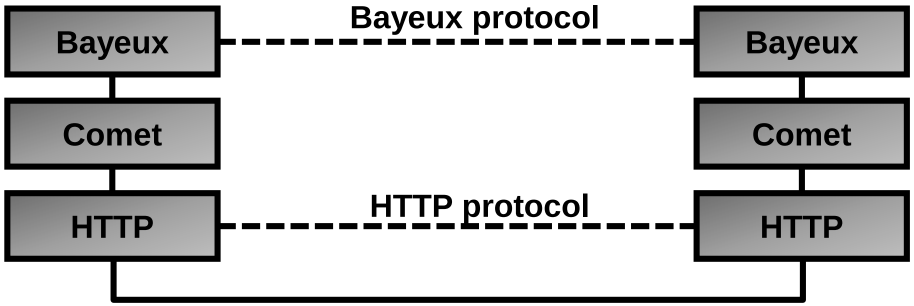
1. Server push technologies (7/7)

Higher layer protocols:

Higher layer protocols define commands and responses that can be used in a client-server application.
Examples:

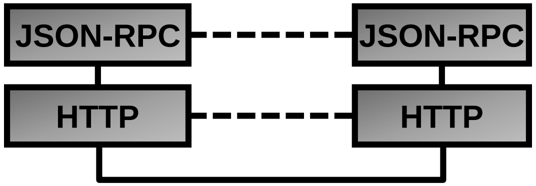
1. Bayeux protocol (by Dojo foundation):

→ Protocol on top of a comet-mechanism supporting streaming & long-polling.
→ Defines frames, commands and fields etc.

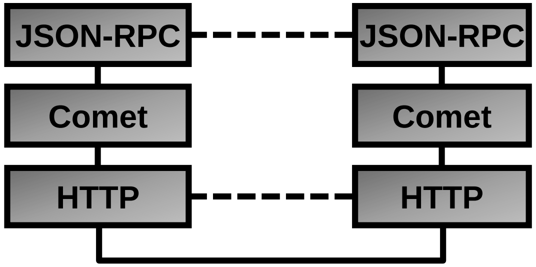


2. JSON-RPC:

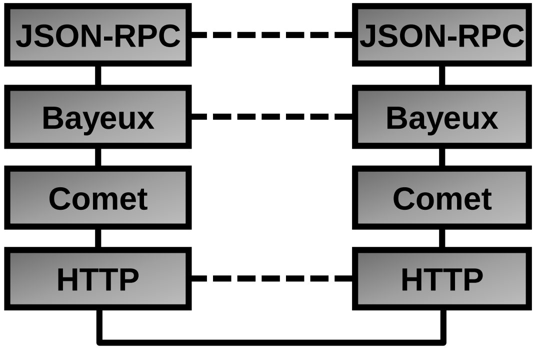
→ Protocol with 3 different message types that are mapped to HTTP: request, response, notification.
→ Peer-to-peer protocol, both client and server can send asynchronous notifications.
→ Different scenarios are possible:



JSON-RPC directly on top of HTTP. Fully bidirectional message communication of JSON-RPC is not possible.



Comet provides bidirectional communication for JSON-RPC.



JSON-RPC messages are mapped to Bayeux messaging protocol.

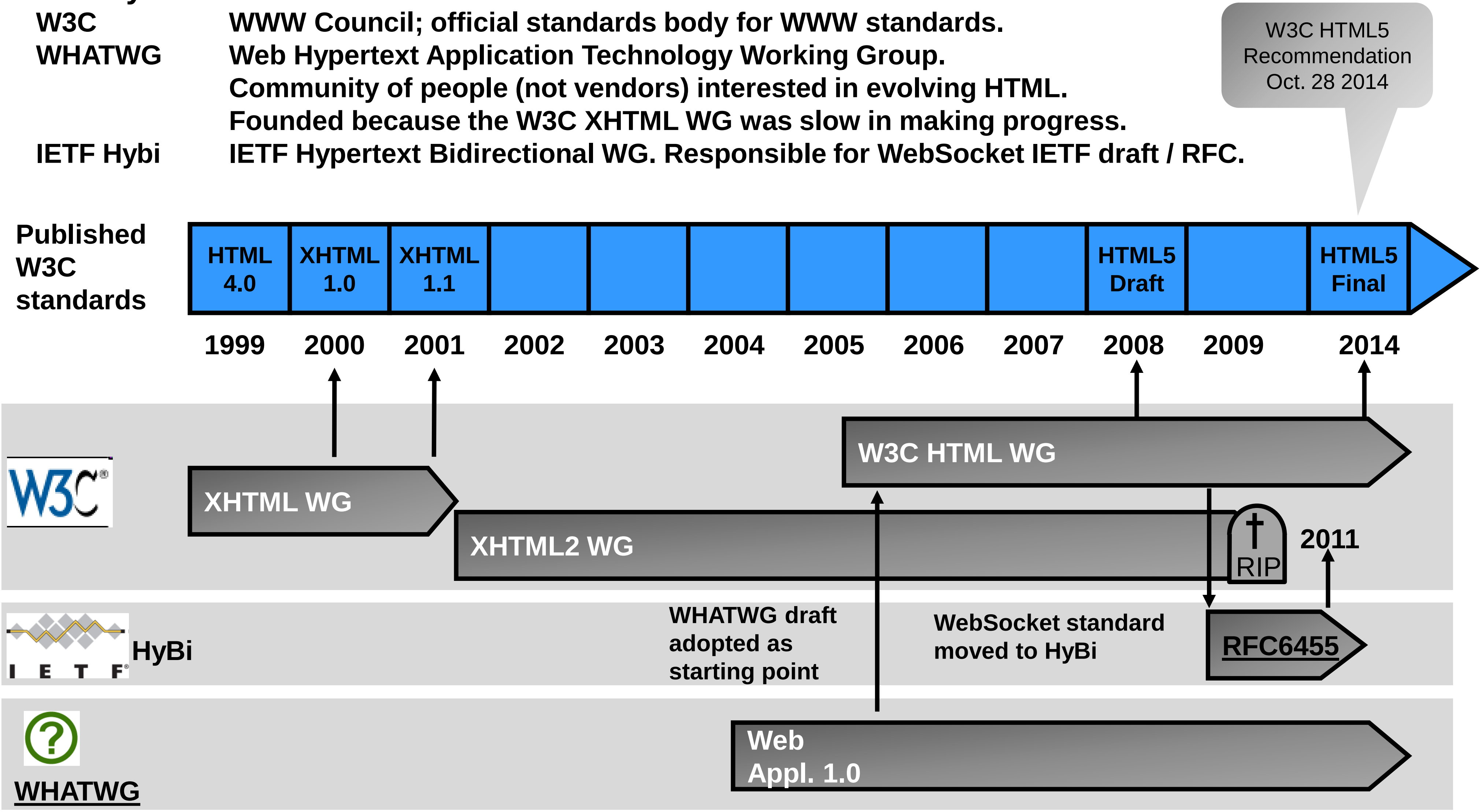
3. WebSockets (1/3)

History of HTML and WebSockets standards:

W3C WWW Council; official standards body for WWW standards.

WHATWG Web Hypertext Application Technology Working Group.
Community of people (not vendors) interested in evolving HTML.
Founded because the W3C XHTML WG was slow in making progress.

IETF Hybi IETF Hypertext Bidirectional WG. Responsible for WebSocket IETF draft / RFC.



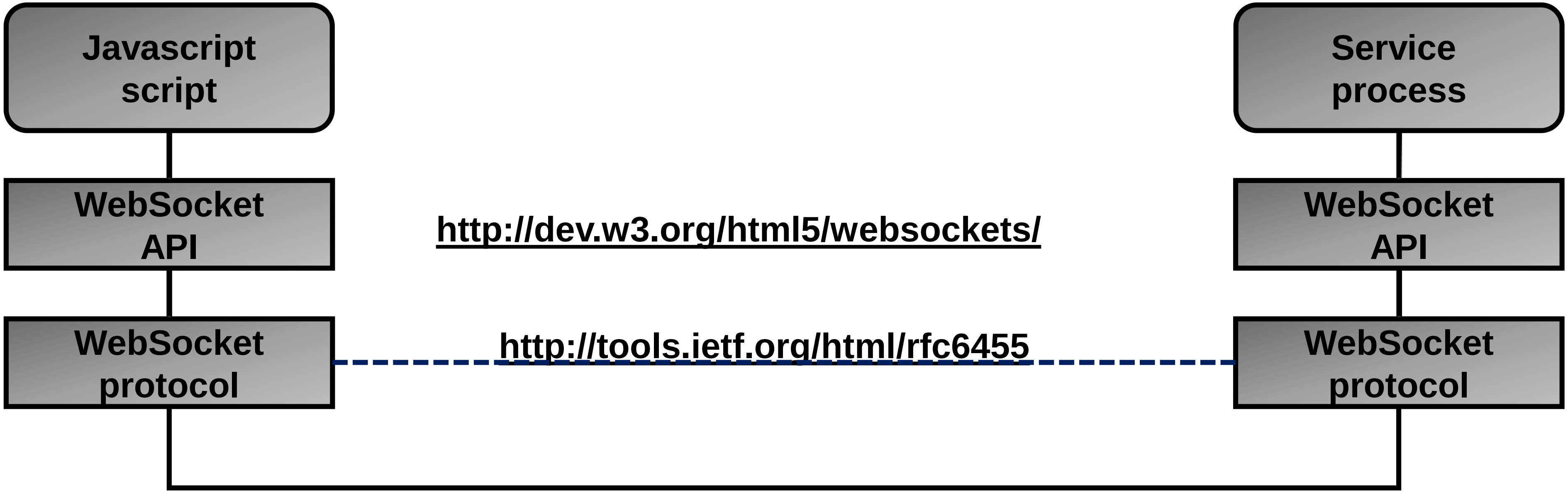
3. WebSockets (2/3):

The WebSocket protocol defines the procedures to upgrade a plain-vanilla HTTP-connection to a fully bidirectional WebSocket transport connection.

The WebSocket Javascript API defines the functions that give access to the WebSocket layer.

- W3C is responsible for the WebSocket Javascript API standard.
- IETF is responsible for the WebSocket protocol standard.
- The WebSocket protocol standard is published as IETF RFC6455.

See also http://www.websocket.org/.



3. WebSockets (3/3):

WebSocket handshake procedure for upgrading an HTTP connection:
The client sends a normal HTTP GET request, but requests to upgrade to a WebSocket connection. Afterwards the connection remains active until it is closed and the client and server can exchange messages based on an application protocol.

Client → server request (example from http://tools.ietf.org/html/rfc6455):

`GET /chat HTTP/1.1`
`Host: server.example.com`
	Standard HTTP method and Host field lines.

`Upgrade: websocket`
`Connection: Upgrade`
	Upgrade to a WebSocket connection.

`Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==`
`Origin: http://example.com`
	Security fields to protect the handshake against man-in-the-middle attacks.

`Sec-WebSocket-Protocol: chat, superchat`	**Application protocols that the client supports.**
`Sec-WebSocket-Version: 13`	**Field for protocol version checks.**

Server→Client response:

`HTTP/1.1 101 Switching Protocols`
`Upgrade: websocket`
`Connection: Upgrade`
`Sec-WebSocket-Accept: s3pPLMBiTxa…GzbK+xOo=`
	Server acknowledges to upgrade to WebSocket. The server confirms acceptance for upgrading with the Sec-WebSocket-Accept field.

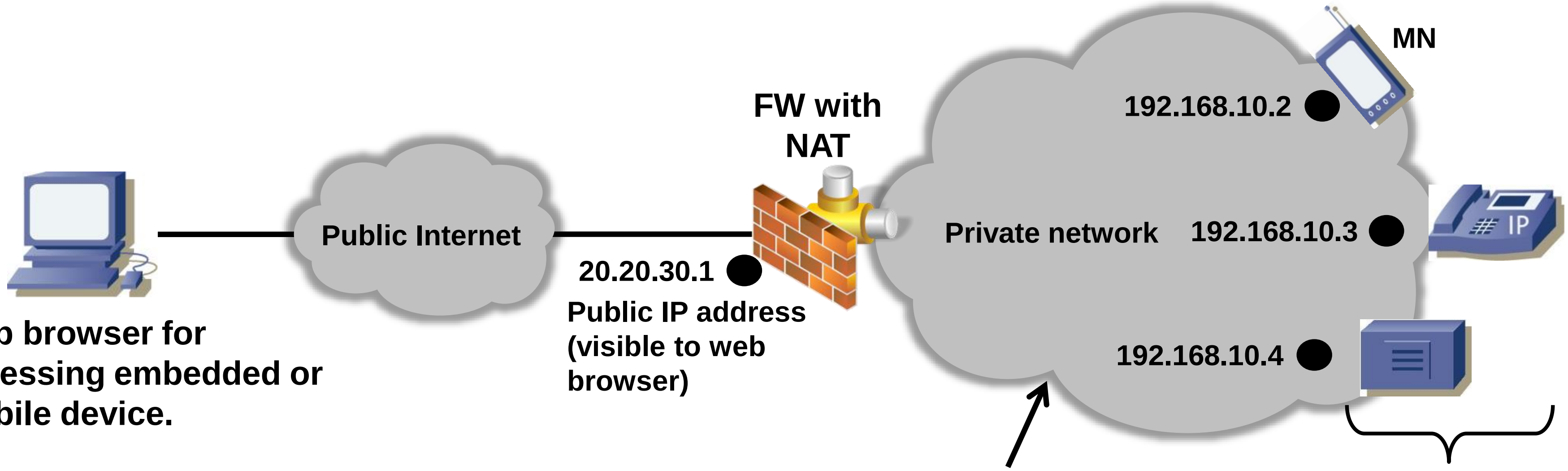
`Sec-WebSocket-Protocol: chat`	**Application protocol (WebSocket subprotocol) that the server selects.**

4. Reverse HTTP (1/5):

Problem:

In many cases (embedded) clients are not accessible (visible) from the Internet because they are in a private network or behind a firewall with NAT (Network Address Translation). Thus it is impossible to connect to the embedded web server of such a device with a web browser from outside the private network.

N.B.: The mobile or embedded devices are able to connect to the Internet (outgoing connections are possible if permitted by the firewall).



Web browser for accessing embedded or mobile device.

Public IP address (visible to web browser)

Network with private IP addresses (RFC1918) such as a mobile provider network or a LAN.

Mobile and embedded devices with private IP addresses

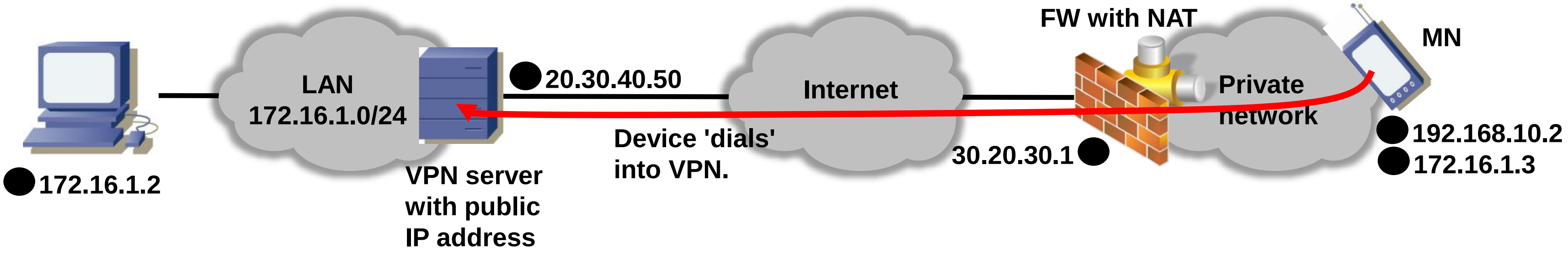
MN:	Mobile Node
NAT:	Network Address Translation
LAN:	Local Area Network
FW:	Firewall

4. Reverse HTTP (2/5):

Solution 1: VPN

Through VPNs both the devices and the web browser are hooked into the same (private) network (172.16.1.0/24 in the example below).

- ☺ Provides security if combined with authentication and encryption.
- ☹ More complex installation. Requires a VPN client on the devices.
- ☹ Increased performance requirements on clients and servers (encryption, VPN protocol).

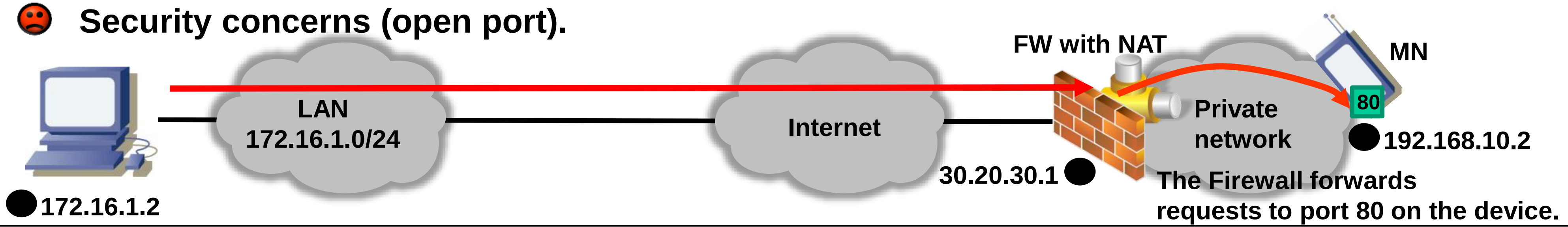


Device 'dials' into VPN.

Solution 2: Port Forwarding

NAT firewall forwards port 80 to 192.168.10.2.

- ☺ No additional gear required.
- ☹ NAT firewall must be under control for adding the port forwarding rule.
- ☹ Security concerns (open port).



The Firewall forwards requests to port 80 on the device.

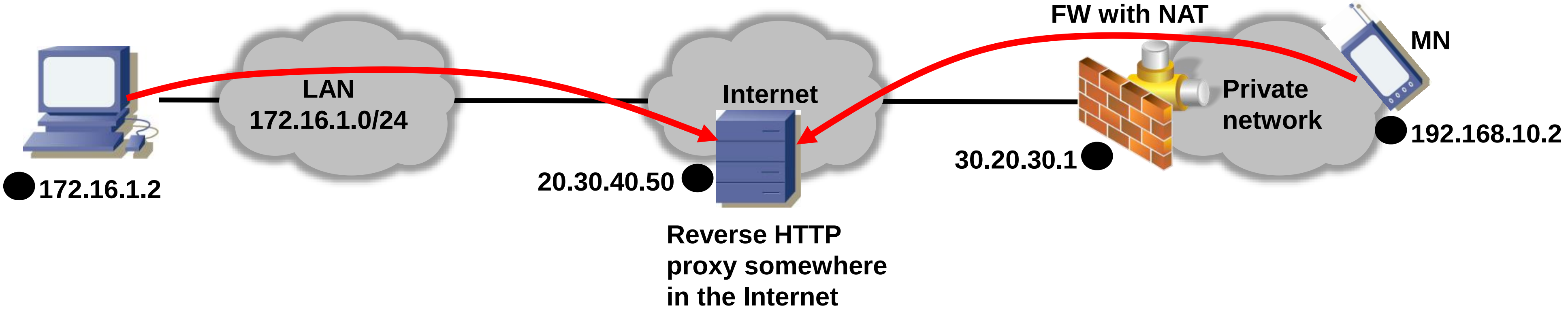
4. Reverse HTTP (3/5):

Solution 3: Reverse HTTP (1/3)

Reverse HTTP is an experimental protocol (see IETF draft http://tools.ietf.org/html/draft-lentczner-rhttp-00) that allows HTTP client and server to switch roles (client becomes server and vice versa). The firewall lets the packets pass since the mobile node opens the connection.

During connection setup, HTTP client and server switch roles (client becomes server and vice versa).

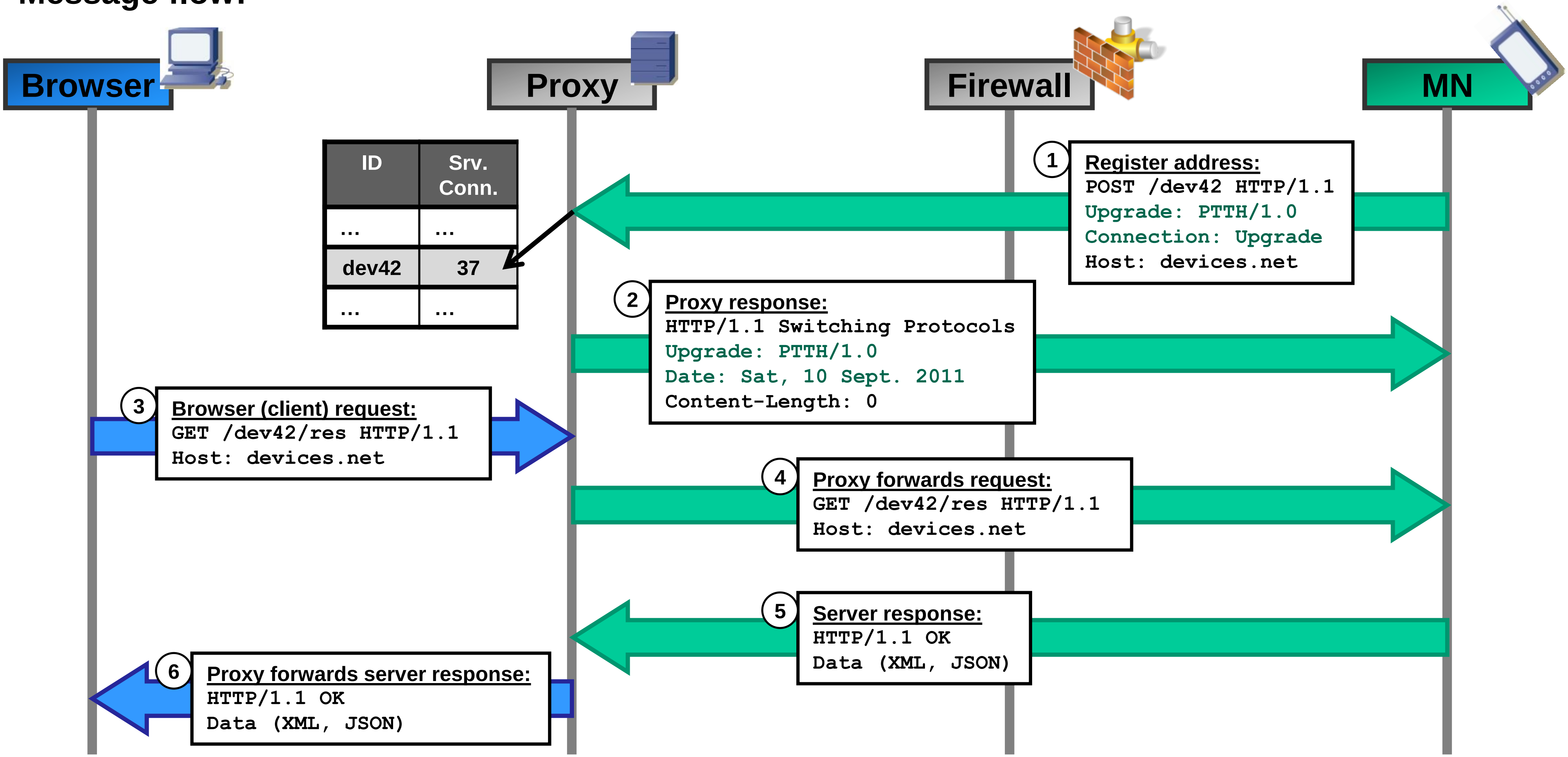
- ☺ **No VPN required (security could be satisfied with HTTPs).**
- ☺ **Can be combined with WebSockets or COMET protocols (see above) for server push.**
- ☹ **Reverse HTTP proxy required (there are open source solutions such as yaler.org).**



4. Reverse HTTP (4/5):

Solution 3: Reverse HTTP (2/3)

Message flow:



ID	Srv. Conn.
…	…
dev42	37
…	…

1 **Register address:**

```
POST /dev42 HTTP/1.1
Upgrade: PTTH/1.0
Connection: Upgrade
Host: devices.net
```

2 **Proxy response:**

```
HTTP/1.1 Switching Protocols
Upgrade: PTTH/1.0
Date: Sat, 10 Sept. 2011
Content-Length: 0
```

3 **Browser (client) request:**

```
GET /dev42/res HTTP/1.1
Host: devices.net
```

4 **Proxy forwards request:**

```
GET /dev42/res HTTP/1.1
Host: devices.net
```

5 **Server response:**

```
HTTP/1.1 OK
Data (XML, JSON)
```

6 **Proxy forwards server response:**

```
HTTP/1.1 OK
Data (XML, JSON)
```

4. Reverse HTTP (5/5)

Solution 3: Reverse HTTP (3/3)

1. The mobile node registers with the proxy:
The mobile node opens an outgoing HTTP connection to the proxy (the proxy must be accessible both by the mobile node and the web browser, so usually the proxy is somewhere in the Internet).
The URL in the POST request contains an ID of the client (dev42 in the example).
The mobile node requests to upgrade the connection to reverse HTTP with the HTTP header fields `Upgrade: PTTH/1.0` and `Connection: Upgrade`.

2. Proxy accepts the connection switch:
The proxy accepts the connection upgrade by responding with `HTTP/1.1 Switching Protocols, Upgrade: PTTH/1.0.`
The proxy adds the mobile client's ID (dev42) to its connection table (dev42 is accessible through HTTP connection 37).
From now on the proxy acts like an HTTP client while the mobile node is an HTTP server.

3. Browser request:
The user enters the URL http://www.devices.net/dev42/res in the browser in order to access the resource 'res' on the mobile node. The host in the URL is the proxy's domain name address, so the browser sends the request to the proxy.

4. Proxy forwards request:
The proxy consults its connection table and finds that dev42 is accessible through HTTP connection 37. The proxy forwards the browser request through HTTP connection 37.

5. Mobile node's HTTP server response:
The web server application on the mobile node retrieves the requested resource 'res' and sends the response back to the proxy.

6. Proxy forwards response:
The proxy forwards the response back to the browser.

5. HTML5

HTML5 ~= HTML + CSS + Javascript

New features in HTML5 (incomplete list):

New feature	Description
Microdata	Annotate HTML pages with semantic information (→ semantic web). Example microdata standards: schema.org
Web storage	Key value pair storage to store arbitrary data (no more size restrictions as with cookies).
Web SQL database	Javascript API to access a browser-integrated SQL database. This specification is not maintained anymore (see http://dev.w3.org/html5/webdatabase/).
Web workers	Background tasks that execute Javascript code. Allows to better separate the GUI from application code processing (no more lengthy processing required in GUI event handlers).
Semantic tags	New tags with semantics such as <SECTION>, <ARTICLE>, <HEADER>.
Audio and video tags	Direct support for audio and video in HTML, no more plugins required (Flash, Silverlight).
Canvas2D and Canvas3D	Canvas object for drawing shapes.
SVG support	SVG (Scalable Vector Graphics) is now part of the HTML standard.
New CSS selectors	nth-children(even/odd), first-child etc.
Web fonts	Support for font standards like OTF (Open Type Font).

Check your browser‘s HTML5 support:
→ http://modernizr.github.io/Modernizr/test/index.html