WebSockets

[intro]

WebSockets?

- full-duplex
- TCP-based
- persistent connection
- message-oriented
- cross-origin
- standardized protocol
- JavaScript API

WebSockets in action

- http://wordsquared.com/
- http://paintwith.me/
- http://www.youtube.com/watch?v=64TcBiqmVko&feature=player_embedded
- …and more!

Why WebSockets?

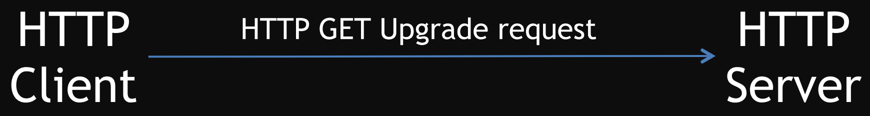
- HTTP is half-duplex
- HTTP has too much overhead
- Ajax doesn’t help
- Comet doesn’t help

WebSocket uses

- Real-time updates (sports, finance)
- Games
- Collaboration & Education
- Feeds & rich communication
- Location-based services
- Services based on real-time APIs
- User Monitoring & Logging

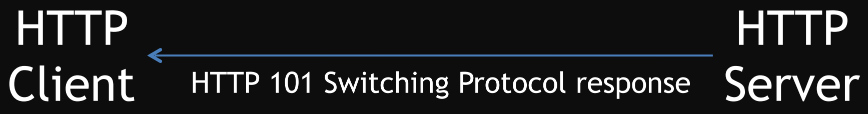
WebSocket protocol

```
GET /chat HTTP/1.1
    Host: server.example.com
    Upgrade: websocket
    Connection: Upgrade
    Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
    Origin: http://example.com
    Sec-WebSocket-Protocol: chat, superchat
    Sec-WebSocket-Version: 13
```



WebSocket protocol

```
HTTP/1.1 101 Switching Protocols
        Upgrade: websocket
        Connection: Upgrade
        Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK
+xOo=
        Sec-WebSocket-Protocol: chat
```



Using WebSockets

Server-side:

- Node.js
  - WebSocket-Node
  - Socket.IO
  - Engine.IO
- C#/.NET (IIS 8 ASP.NET 4.5)
  - XSockets.NET
  - Fleck
- Java
  - Atmosphere
- Ruby
  - EM-WebSocket

WebSocket API browser support

Google Chrome 16	Safari 6
Opera 12.10	Internet Explorer 10
Firefox 11	

WebSocket API

```
WebSocket WebSocket(
  in DOMString url,
  in optional DOMString protocols
);

WebSocket WebSocket(
  in DOMString url,
  in optional DOMString[] protocols
);
```

WebSocket API

1. Parse URL or throw SYNTAX_ERR
2. If port is blocked, throw SECURITY_ERR
3. Check sub-protocol
4. Get origin from <script>
5. Return WebSocket object
6. Establish a socket connection

WebSocket API

Status

- readyState
  - CONNECTING (= 0)
  - OPEN (= 1)
  - CLOSE (= 2)
  - CLOSING
- onopen
- onmessage
- onclose
- onerror

Methods

≤123, LOL

- close(code, reason)
- send(data)
  - String
  - ArrayBuffer
  - Blob

https://developer.mozilla.org/en-US/docs/WebSockets/WebSockets_reference/CloseEvent#Status_codes

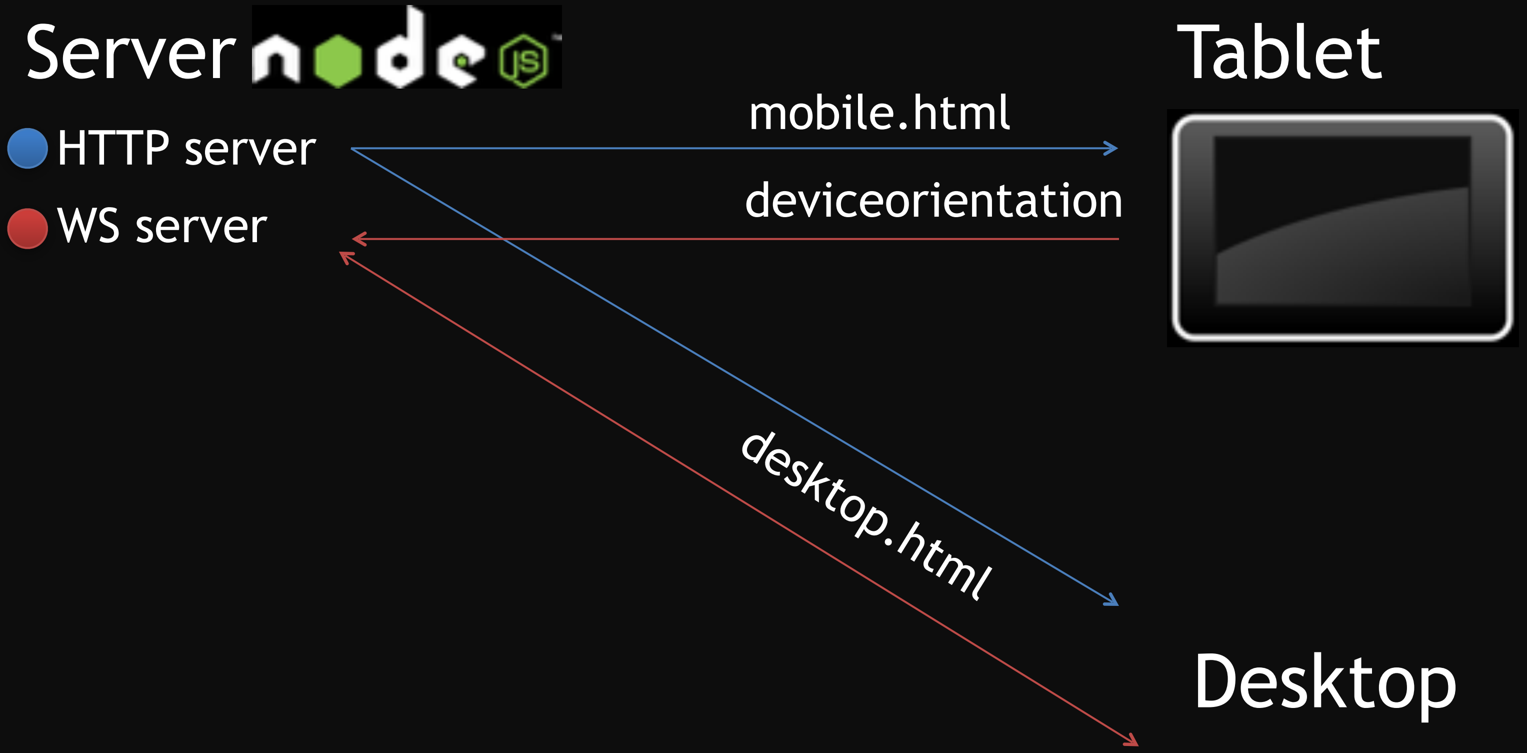
WebSocket API

```
var socket =
new WebSocket('ws://www.example.com/socketserver',
'protocolOne');

socket.onopen = function() {
    socket.send('hey');
}

socket.onmessage = function(event) {
    var msg = JSON.parse(event.data);
}
```

WebSockets Demo



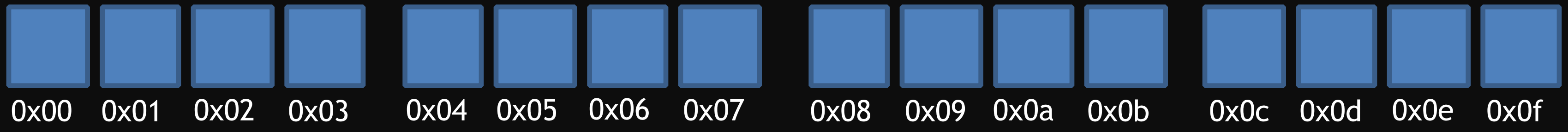
Binary data in JS [oh why]

- WebGL
- Files
- XHR2
- Canvas
- WebSockets
- DataView
- Typed arrays
- Blobs
- Uint8ClampedArray

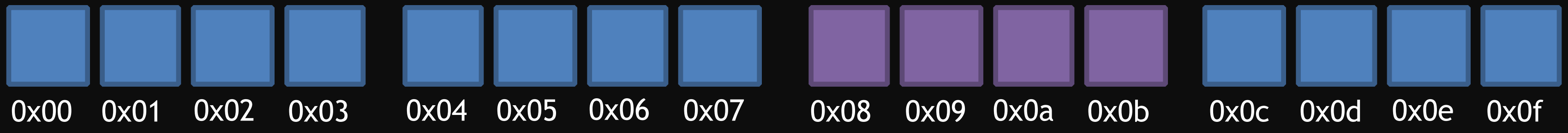
🟦 = 1 byte

Binary data in JS [is great]

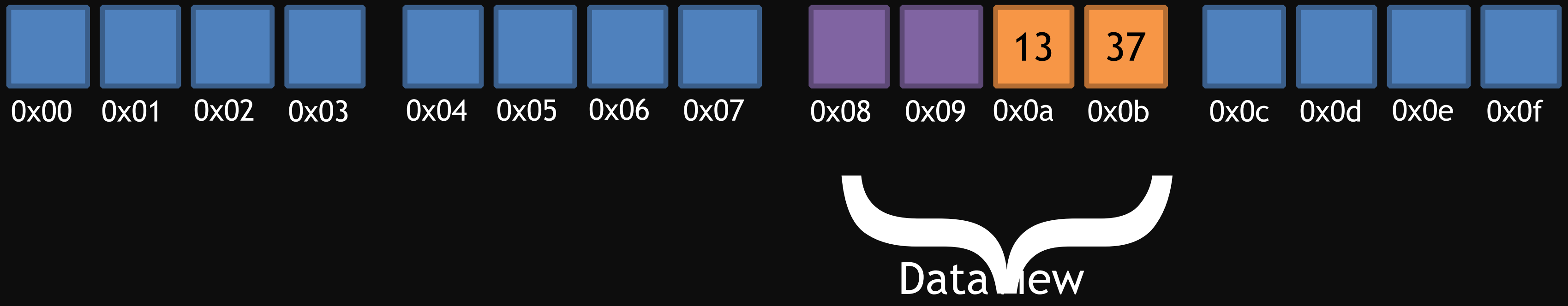
```
var buffer = new ArrayBuffer(16)
```



```
var dv = new DataView(buffer, 8, 4)
```



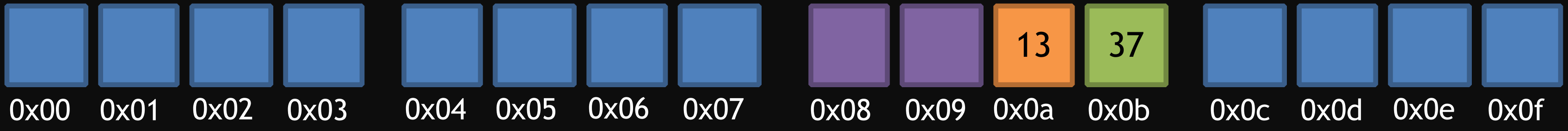
```
dv.setUint16(2, 0x1337)
```



🟦 = 1 byte

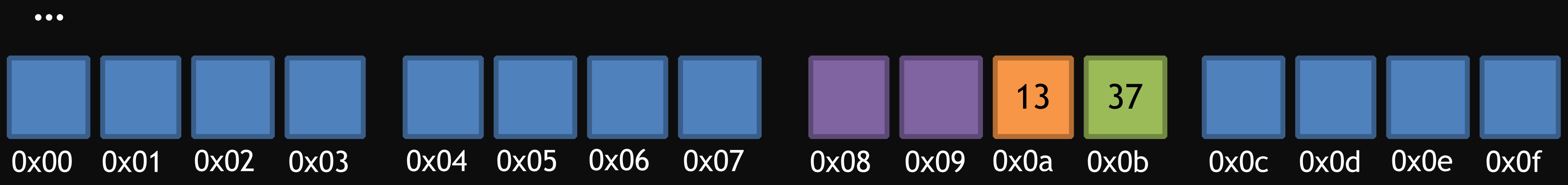
Binary data in JS [is great]

dv.getInt8(2) -> 0x13

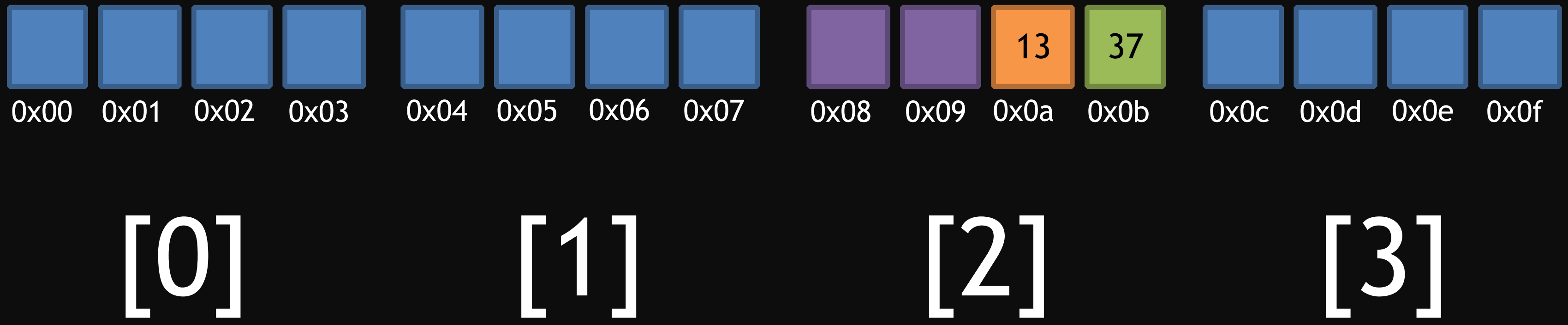


Int8, Uint8:	🟦
Int16, Uint16:	🟦 x2
Int32, Uint32, Float32:	🟦 x4
Float64:	🟦 x8

Binary data in JS [is great]



```
var arr = new Int32Array(buffer)
```



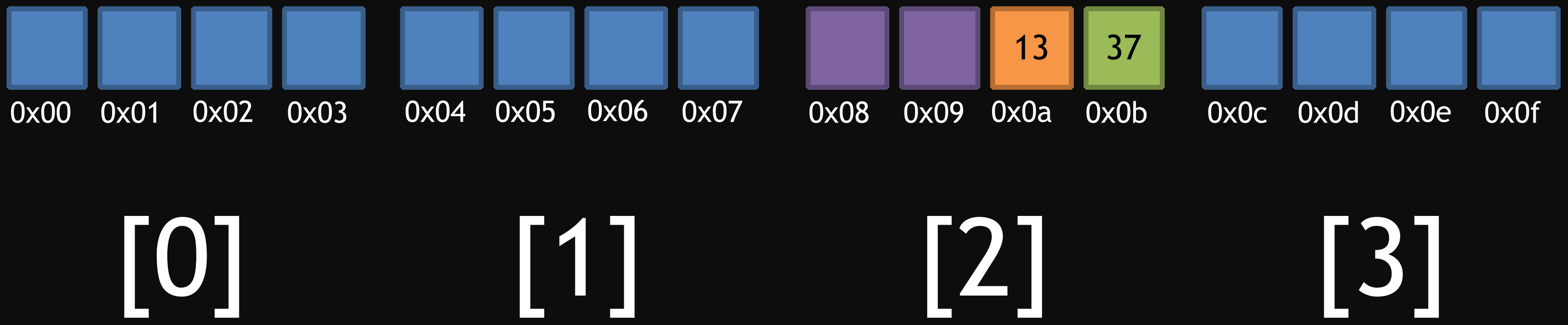
arr[2] ?

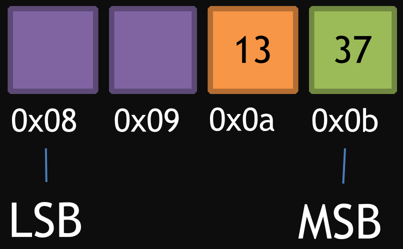
WAT

🟦 = 1 byte

Binary data in JS [is great?..]

var arr = new Int32Array(buffer)





```
dv.setUint16(2, 0x1337, true)
```

0x37130000

Blobs

~~BlobBuilder~~

```
Blob Blob(
  [optional] Array parts,
  [optional] BlobPropertyBag properties
);
```

```
Blob slice(
  optional long long start,
  optional long long end,
  optional DOMString contentType
};
```

Binary data over WS

…just send it already!

opcode	meaning
0	*Continuation Frame*
1	*Text Frame*
2	*Binary Frame*
8	*Connection Close Frame*
9	*Ping Frame*
10	*Pong Frame*

socket.binaryType = “arraybuffer” | “blob”

The WebSocket Challenge

ws://wsc.jit.su

1. ArrayBuffer
2. JSON.stringify
3. Use Chrome Network Tab —> Frames
4. —> { msg:"challenge_accepted", name: "Socketeers" }
5. <— { msg: "auth", auth_token: "6f7sd8s78"}
6. Task1 request: —> { msg: "task_one", auth_token: "6f7sd8s78"}
7. Task1 server response:
   <— { msg: "compute", operator:"+/-/*", operands:[4,5]}
8. Task1 send result:
   —> { msg: "task_one_result", result: 9, auth_token: "6f7sd8s78" }
9. Task2 request: —> { msg: ???, auth_token: "6f7sd8s78" }
10. Task2 server response: <—
    { msg: "binary_sum", bits: 8/16 } and ArrayBuffer (16 bytes)
    Convert to an *unsigned* typed array according to the `bits` field
11. Task2 send result:
    —> { msg: "task_two_result", result: 0, auth_token: "6f7sd8s78" }

Check message type with typeof evt.data === 'string'

The End

[you are awesome]