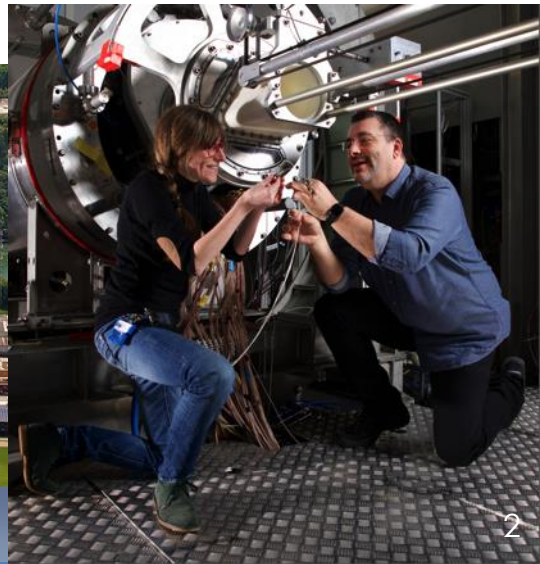
Stefan Ritt, Paul Scherrer Institute, Switzerland

REAL-TIME DATA VISUALIZATION AND CONTROL USING MODERN WEB TECHNOLOGIES

3RD REAL TIME SCHOOL, CAPE TOWN, SOUTH AFRICA, 2018

Something about myself

- Studied at University of Karlsruhe, Germany
- Head of muon physics group at the Paul Scherrer Institute, Switzerland
- Developer of
  ELOG electronic logbook (elog.psi.ch/elog)
  MIDAS DAQ system (midas.triumf.ca)
  DRS chip (www.psi.ch/drs)



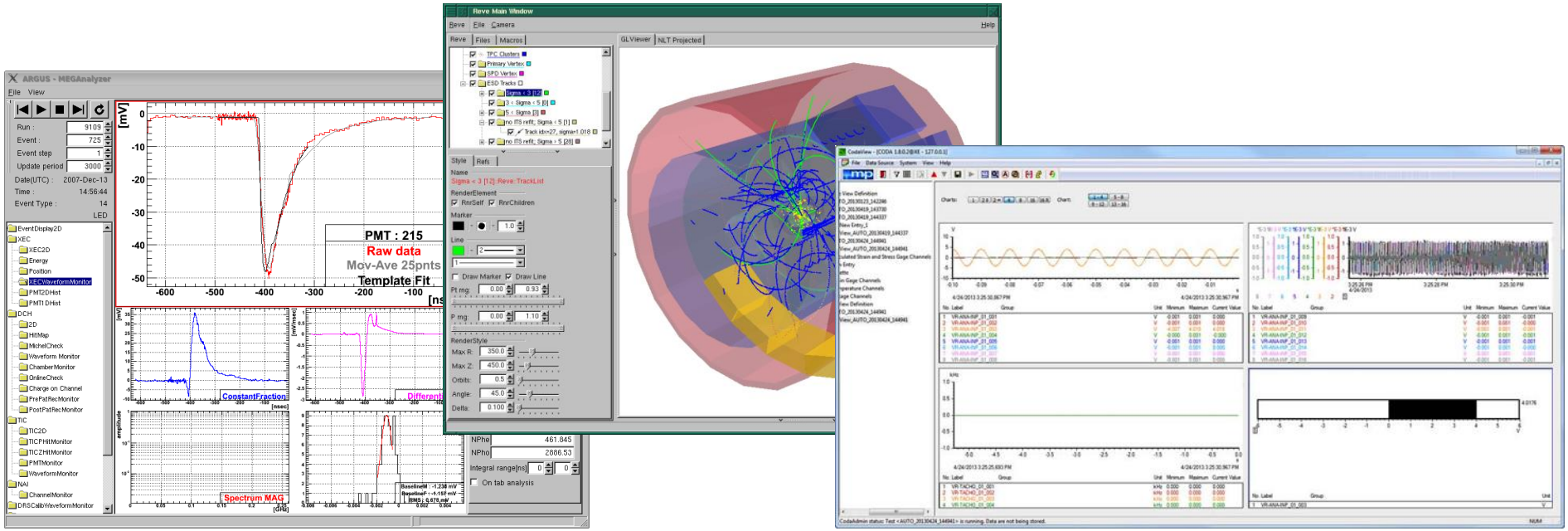
Real Tim

About this Lecture

- Web technologies from a user's point of view (will cover ~5% of HTML) – mainly examples driven
- Slow start with basics, advanced topics at the end
- Please ask questions any time

Experimental Data Visualization and Control

- Some Visualization & Control require Real-Time
- The traditional way
  - Dedicated programs (ROOT, Qt, TCL/TK, …)
  - Must be compiled for different OS
  - Require certain libraries to be installed
  - Limited smartphone support



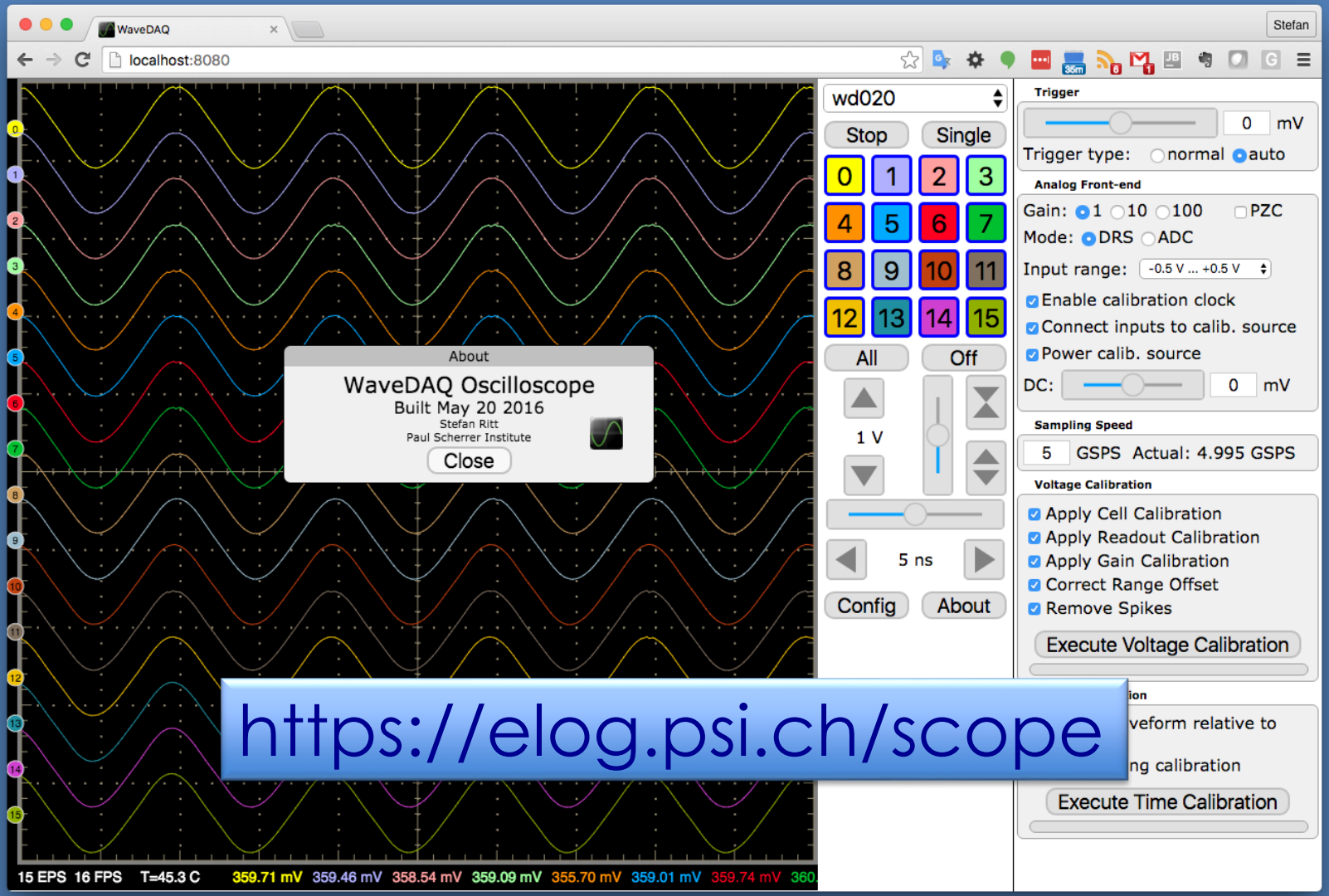
A new opportunity

HTML5 – CSS3 – JavaScript – JSON

Advantages of Web Control

- You only need a browser to access your experiment (from home, public terminal, smartphone)
- Software updates get deployed automatically
- Easy creation of web pages showing several experiments
- Modern browsers run JavaScript at the speed of native programs some years ago
- Debugging of code inside the browser

What you will learn – a little teaser



Agenda

- Basic Web technologies
  - HTML
  - CSS
  - Forms
  - JavaScript
  - DOM
- Advanced Web technologies
  - AJAX
  - JSON
  - Canvas

Agenda

- Server side implementation
  - Mongoose server
  - JSON-RPC
  - Implementation on Raspberry Pi
- Advanced web concepts
  - Push technology with Websockets
  - Floating dialog boxes
  - Dialog widgets

HTML (content) + CSS (style)

Hypertext Markup Language

```
<div>
Hello Real Time Conference
</div>
```

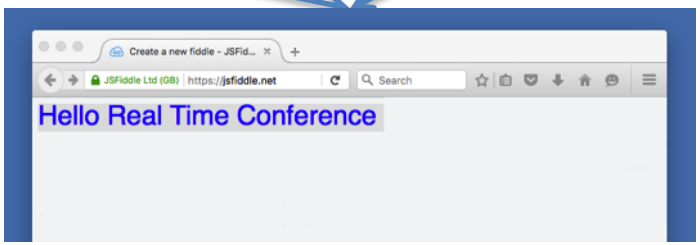
Cascading Style Sheets

Selector
Declaration
Declaration
Declaration
Declaration

```
div {
  font-family:sans-serif;
  font-size:200%;
  color:blue;
  background-color:#E0E0E0;
}
```

Property
Value

Browser




```
<style>
  div {
    color:blue;
    font-size:200%;
  }
</style>
```

```
<head>
  <link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

First HTML Page

HTML:

```
<p>Welcome to the <span>Real Time</span> School.</p>

<p>Do you enjoy the
<span class="subtle">Real Time</span> School?</p>

<p>Will you come to the next
<span class="subtle" id="next">Real Time</span> School?</p>
```

CSS:

```
body {
    font-family: sans-serif;
    font-size:120%;
    color: blue;
}

span { color: red; }

.subtle {
  font-style: italic; }

#next { color: black; }
```

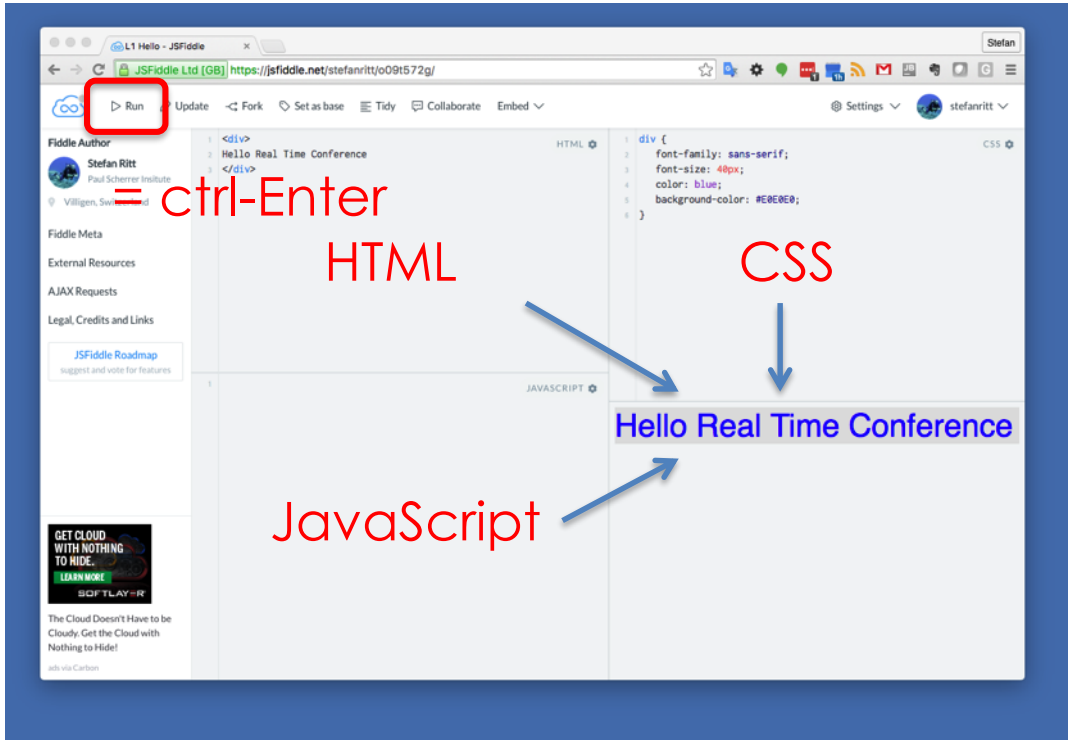
Welcome to the Real Time School.

Do you enjoy the *Real Time* School?

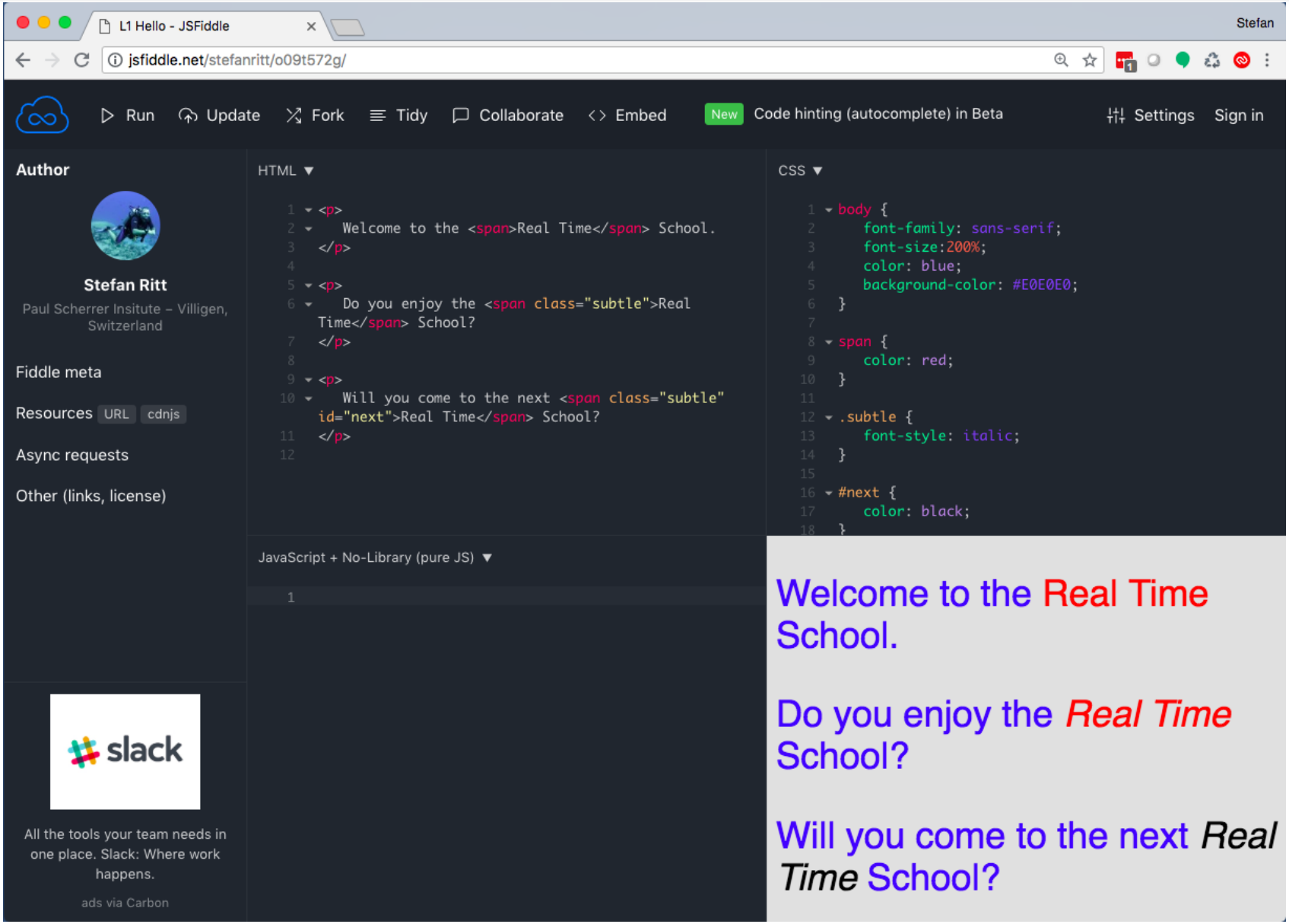
Will you come to the next *Real Time* School?

jsfiddle.net

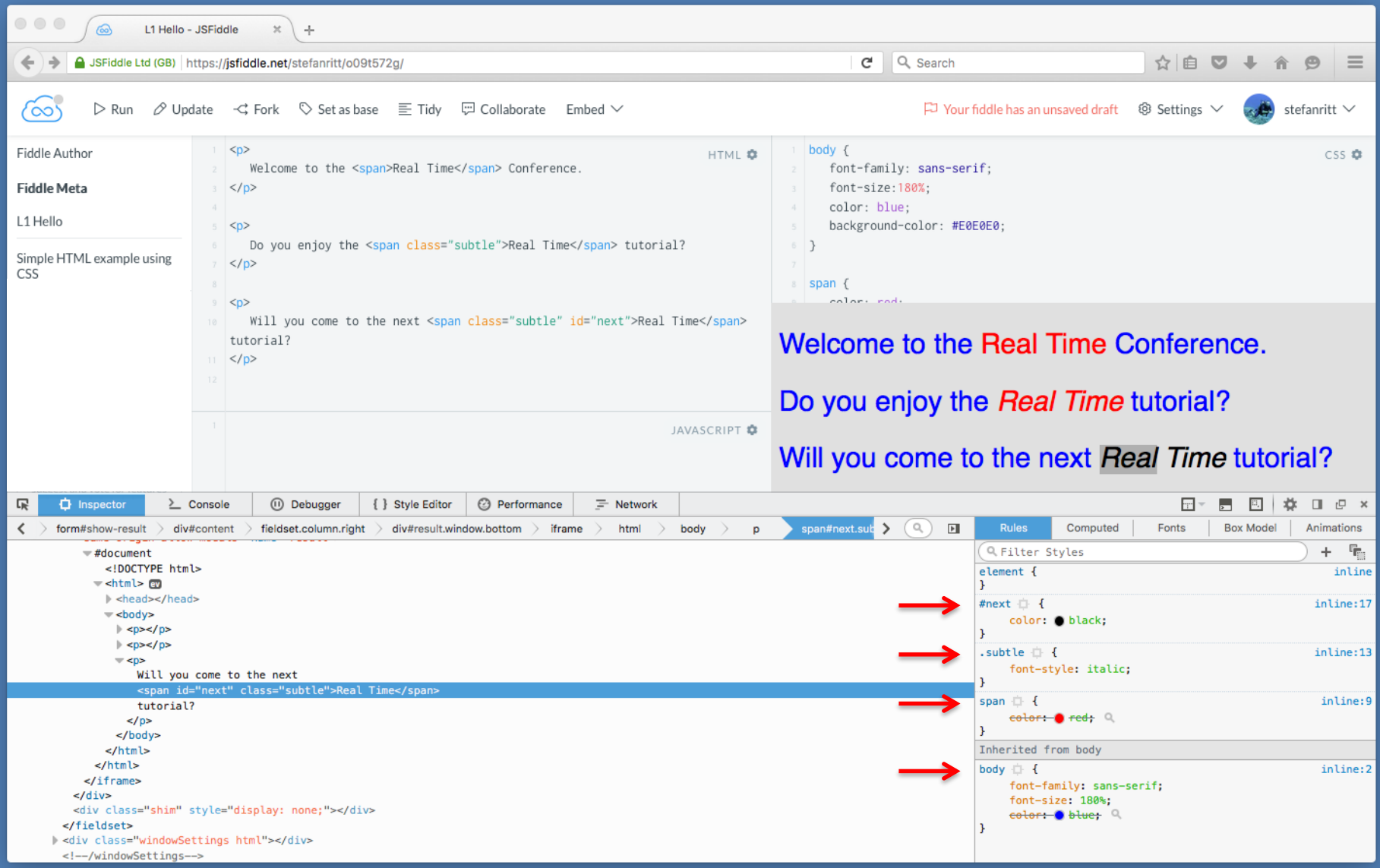
- Online tool to test HTML, CSS & JavaScript
- Documentation: doc.jsfiddle.net
- Alternatives:
  cssdeck.com
  jsbin.com
  dabblet.com



First Example



Debugging CSS



CSS Selectors

Selector	Example	Description
.	.subtle	Selects all elements with class="subtle"
#	#next	Selects all elements with id="next"
element	p	Selects all elements
element > element	div > p	Selects all <p> elements where the parent is a element
element[attribute]	input[type="text"]	Select all <input type="text"> elements
element:nth-child(i)	p:nth-child(2)	Select all <p> elements which are the second child of their parent

```
p:nth-child(odd) {
  background: #F0F0F0;
}
p:nth-child(even) {
  background: #C0C0C0;
}
```

http://www.w3schools.com/cssref/css_selectors.asp

HTML - CSS

- Do separate content and style!
  `Hello <font color="red">Conference</font>`
  is bad practice!

- Overview of CSS options:
  http://www.w3schools.com/css/

- Debug CSS e.g. with Chrome Develop Tools or Firefox Inspector

- Google Search is your friend:
  "CSS text size" → { font-size: 200%; }

- www.w3schools.com

Full minimal HTML5

```
<!doctype html>
<html lang="en">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <title>Minimal Page</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <script type="text/javascript" src="script.js"></script>
  </head>
  <body>
    Hello World
  </body>
</html>
```

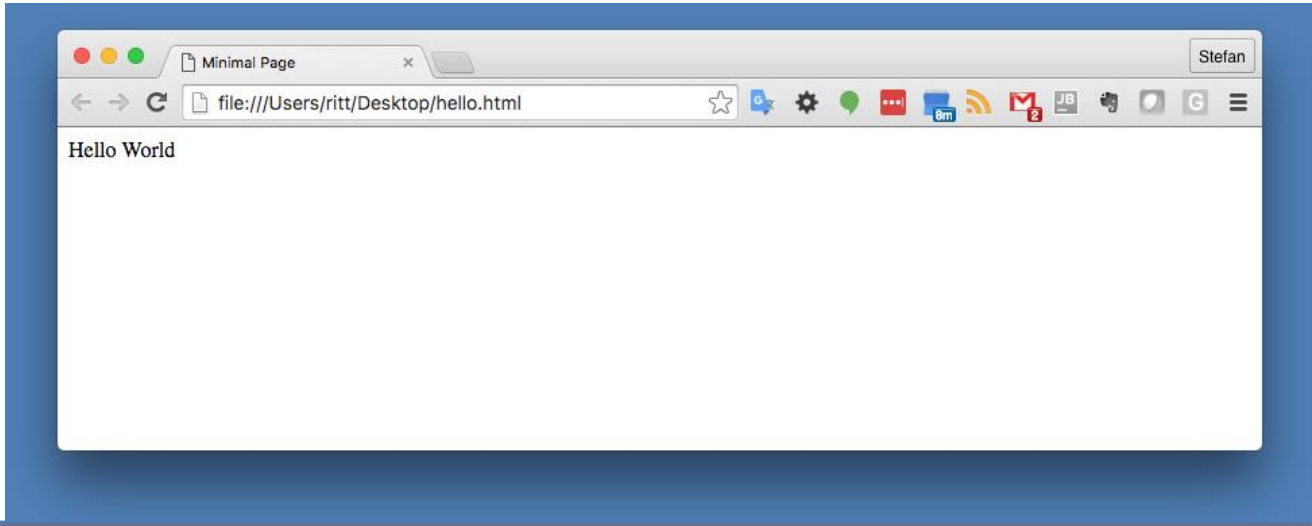
Tells the browser that this is an HTML5 file

Primary language

Encoding

Optional

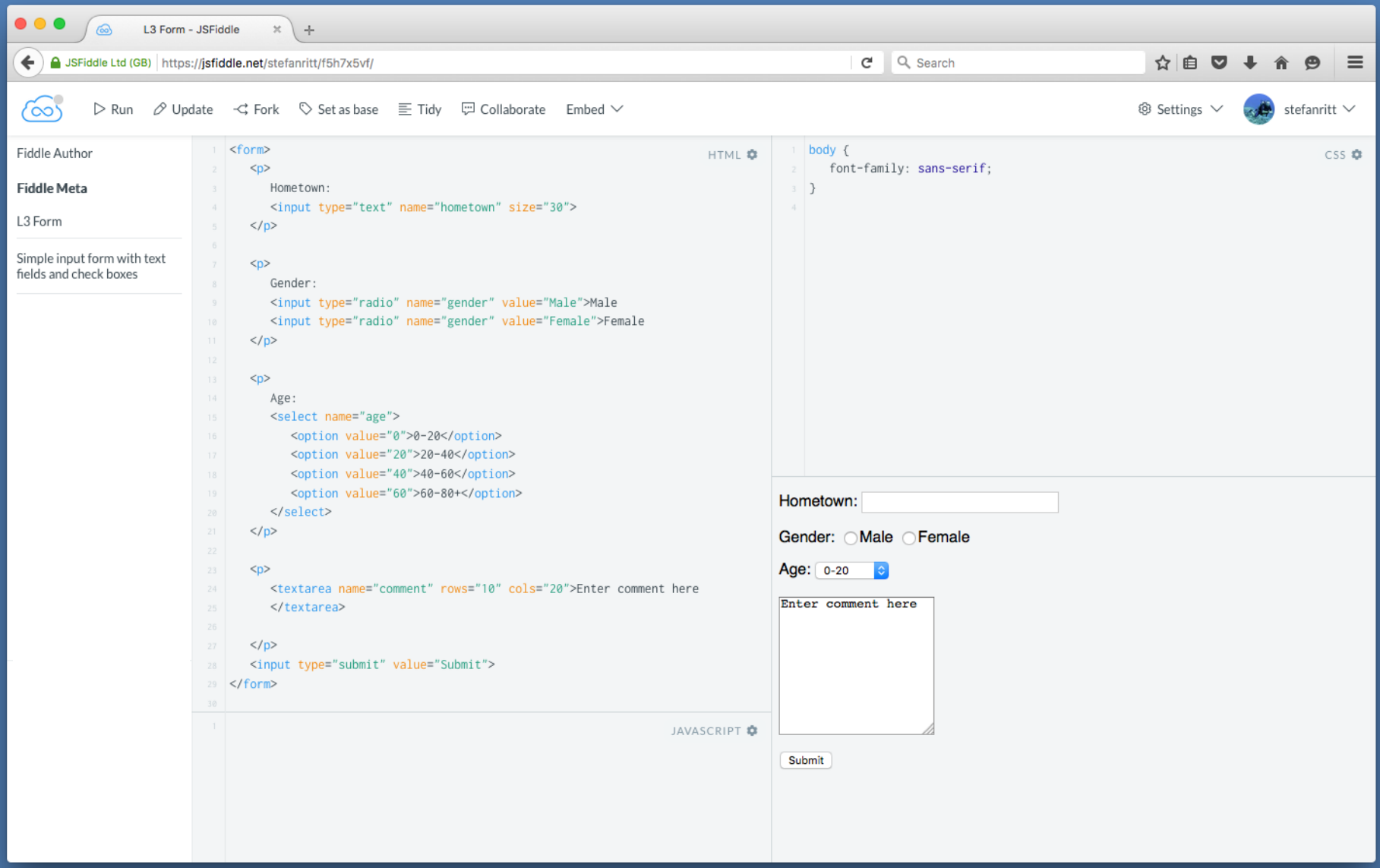
Page contents



Forms

- Forms must be embedded in <form>… </form> tag
- Text input, check box, radio box, drop-down list, submit button
- Form contents gets submitted in URL (for "get" method):
  `http://your.host?name=Stefan&gender=Male`
- Form submission is obsolete!

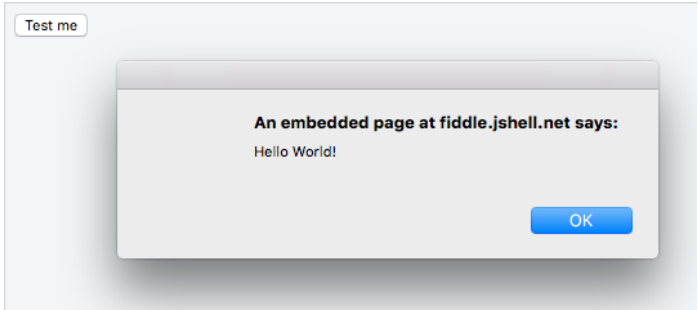
Form Example



JavaScript

- Standardized *interpreted* language running in your browser
- Very slim:

```
<input type="button" value="Test me" onclick="alert('Hello World')">
```



- Similar to C/C++
- Browser-"war" for the fastest interpreter: JavaScript runs as fast as compiled code about ten years ago
  → JS can be used for complicated things

JavaScript Example

HTML ▼

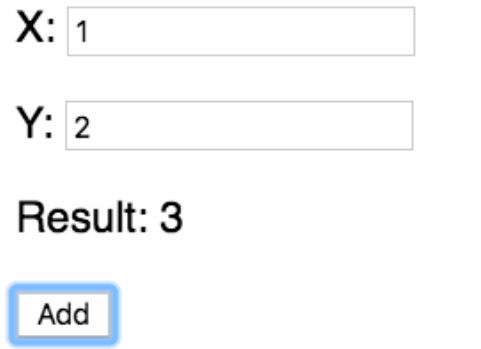
```
<p>
    X:
    <input type="text" id="x">
</p>

<p>
    Y:
    <input type="text" id="y">
</p>
```

CSS ▼

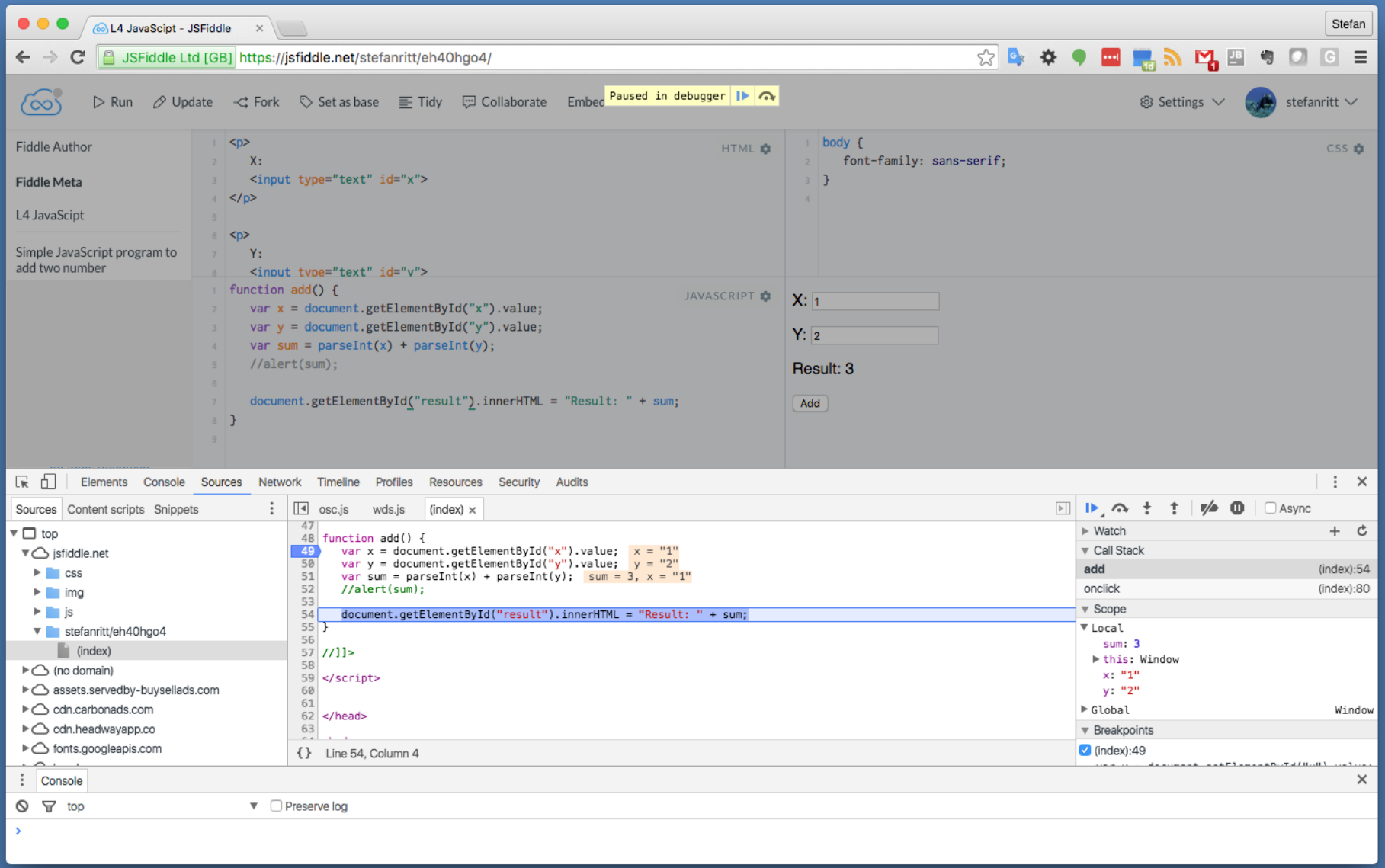
```
body {
    font-family: sans-serif;
}
```

JavaScript + No-Library (pure JS) ▼

```
function add() {
    var x = document.getElementById("x").value;
    var y = document.getElementById("y").value;
    var sum = parseInt(x) + parseInt(y);
    alert(sum);

    document.getElementById("result").innerHTML = "Result: " + sum;
}
```



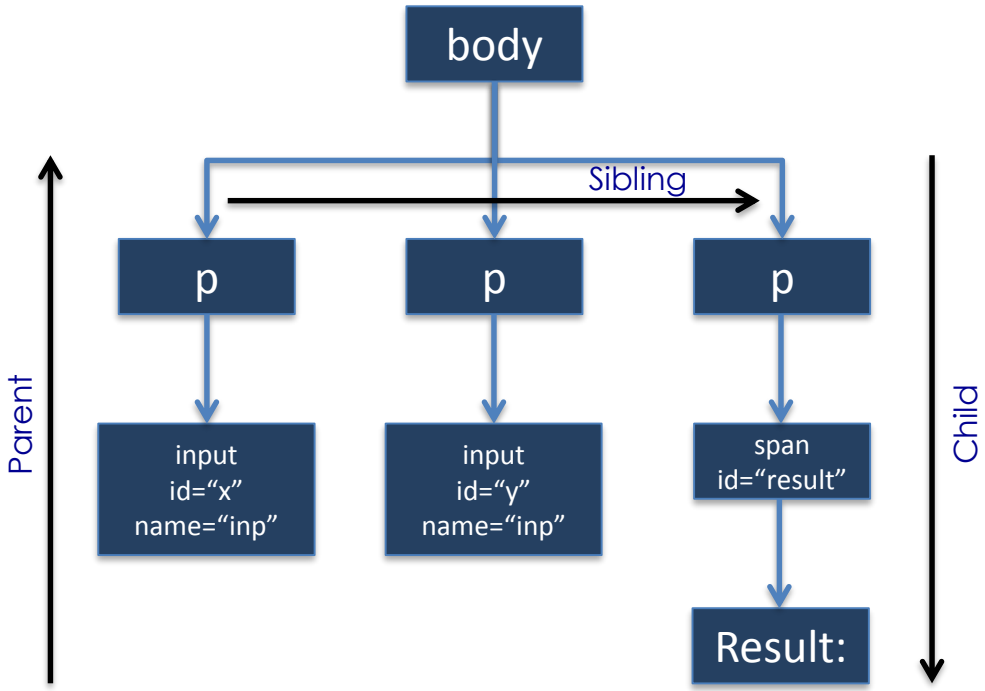
Debugging JavaScript



DOM

Document Object Model behind any web page *plus* interface to that model

```
<body>
  <p>
    <input id="x" name="inp">
  </p>
  <p>
    <input id="y" name="inp">
  </p>
  <p>
    <span id="result">
      Result:
    </span>
  </p>
</body>
```



JavaScript DOM modifications

- Access by ID:
```
var e = document.getElementById("result");
```
- Access by Name:
```
var e = document.getElementsByName("input");
for (i=0 ; i<e.length ; i++)
  e[i].value = "0";
```
- Access by Class Name:
```
document.getElementsByClassName("subtle");
```
- By relation:
```
e.firstChild e.nextSibling e.parentNode …
```
- Predefined elements:
```
document.title document.images …
```
- CSS Style:
```
e.style.color e.style.fontFamily …
```

jQuery

- jQuery is a JavaScript library which simplifies JavaScript programming
- Include it with
```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js">
```
- Use "$" to select elements and special jQuery functions

JavaScript
```
var e = document.getElementsByClass("subtle");
for (var i=0 ; i<e.length ; i++)
   e.style.color = "red";
```

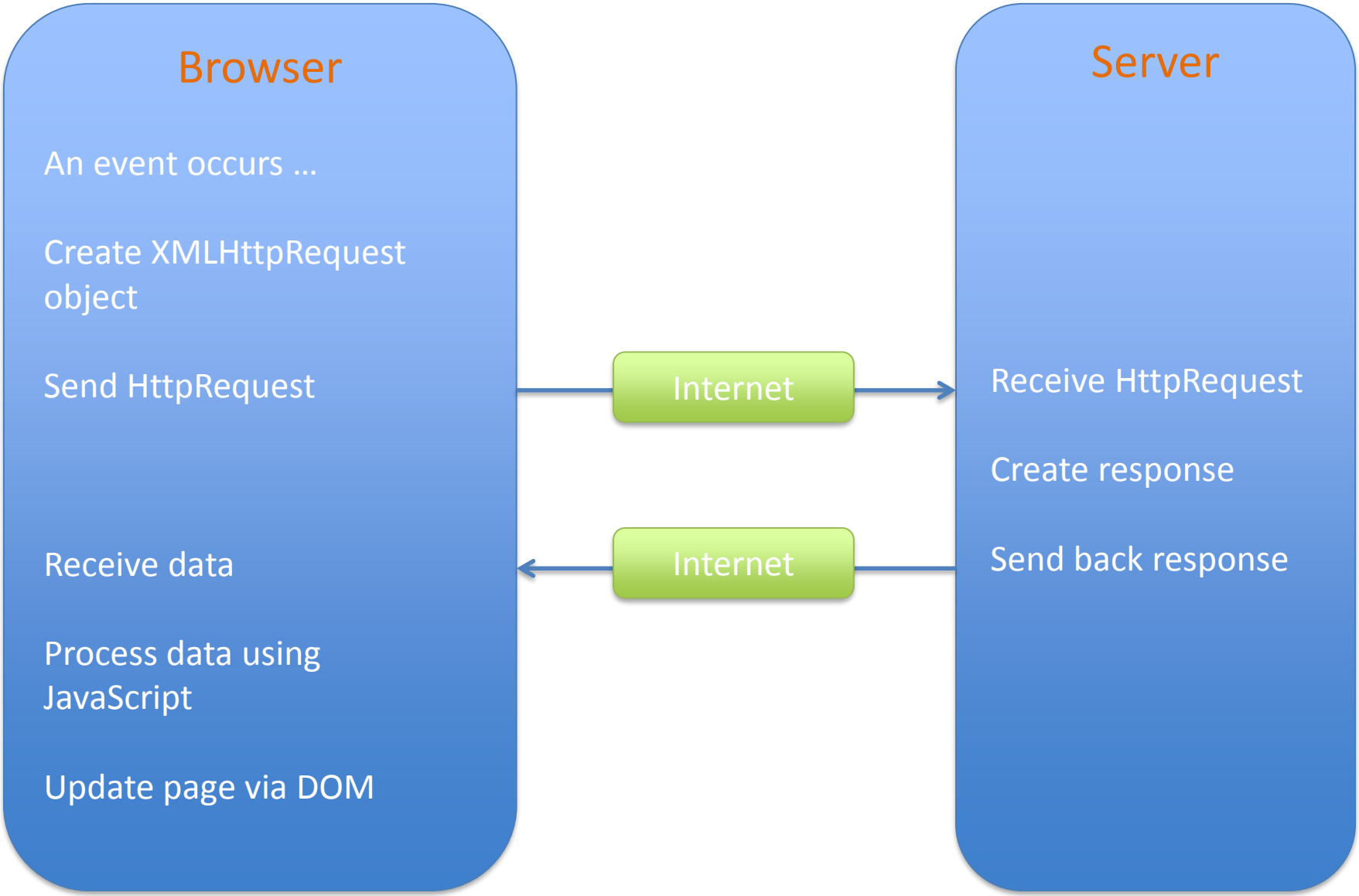
jQuery
```
$(".subtle").css("color", "red");
```

CSS selector

AJAX

- Old days: Form – Submit – New Page
- Slow, network intensive, page flicker
- Asynchronous Javascript and (XML)
  - Update web page without reloading the whole page
  - Send data to the server AFTER the page has loaded
  - Receive data from server AFTER the page has loaded
  - Technique for fast dynamic web pages
- Getting popular in 2005 by Google Suggest

How AJAX Works



AJAX example

HTML ▼

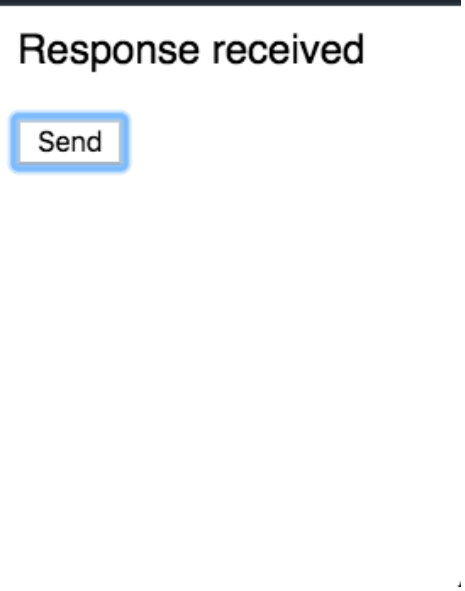
```
<div id="result">
    Idle
</div>

<p>
    <input type="button" value="Send" onclick="load()">
</p>
```

CSS ▼

```
body {
    font-family: sans-serif;
}
```

JavaScript + No-Library (pure JS) ▼

```
function load() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (xhttp.readyState == 4 && xhttp.status == 200) {
            document.getElementById("result").innerHTML =
xhttp.responseText;
        }
    };
    xhttp.open("POST", "/echo/html/", true);
    xhttp.send("html=Response received&delay=3");
}
```



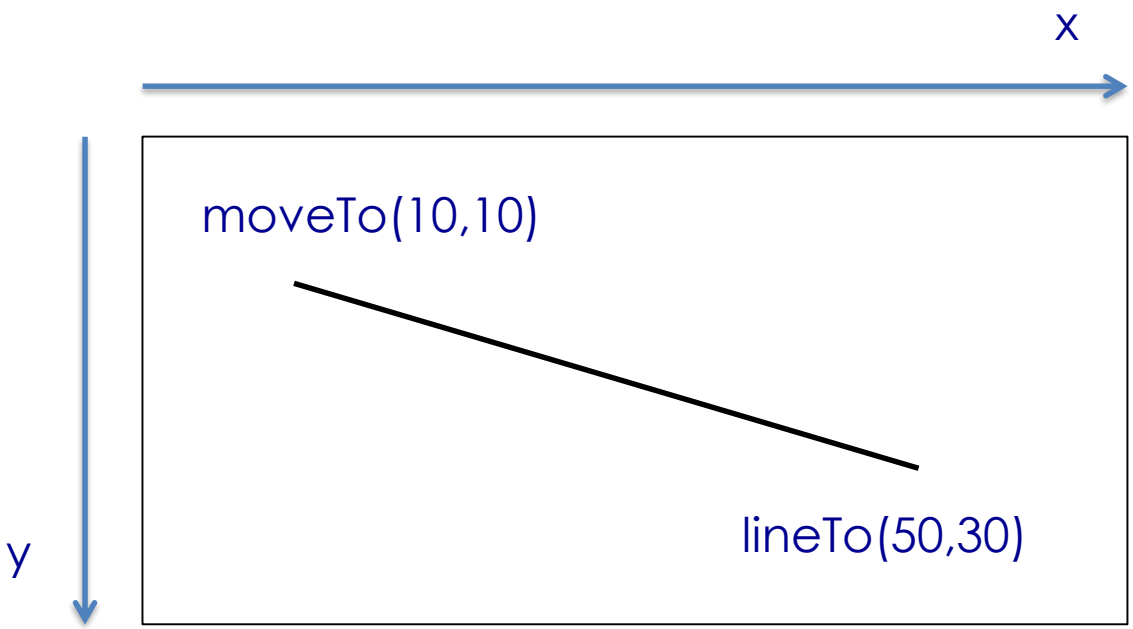
JSON

- JavaScript Object Notation
- ASCII format for storing and transportation of objects
- Pure text format, language independent, self-describing

JSON	Notation	Example
Data	“name”:”value”	“firstName”:”Stefan”
Object	{ Data, Data }	{“firstName”:”Stefan”, ”lastName”:”Ritt” }
Array	“name”:[Object,Object]	“attendees”:[{“firstName”:”Stefan”, ”lastName”:”Ritt” }, {“firstName”:”Linus”, ”lastName”:”Torvalds” }]

Canvas

- Basic HTML is mainly for text display
- Vector graphics with SVG, icons and **canvas**
- A canvas is a HTML5 element on which one can draw dynamic graphics via JavaScript



Canvas Example

HTML ▼

```html
<h1>
Canvas Demo
</h1>

<canvas id="cvs" width="200" height="200"></canvas>
```

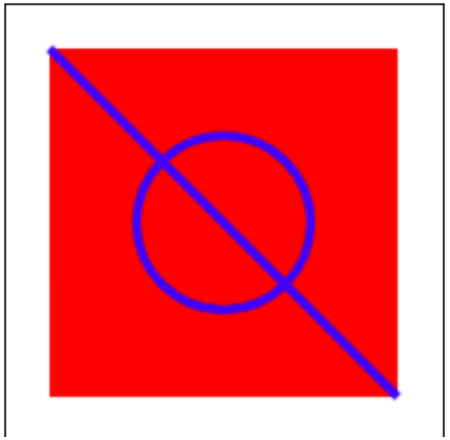
CSS ▼

```css
body {
  font-family: sans-serif;
}

canvas {
  border: 1px solid black;
}
```

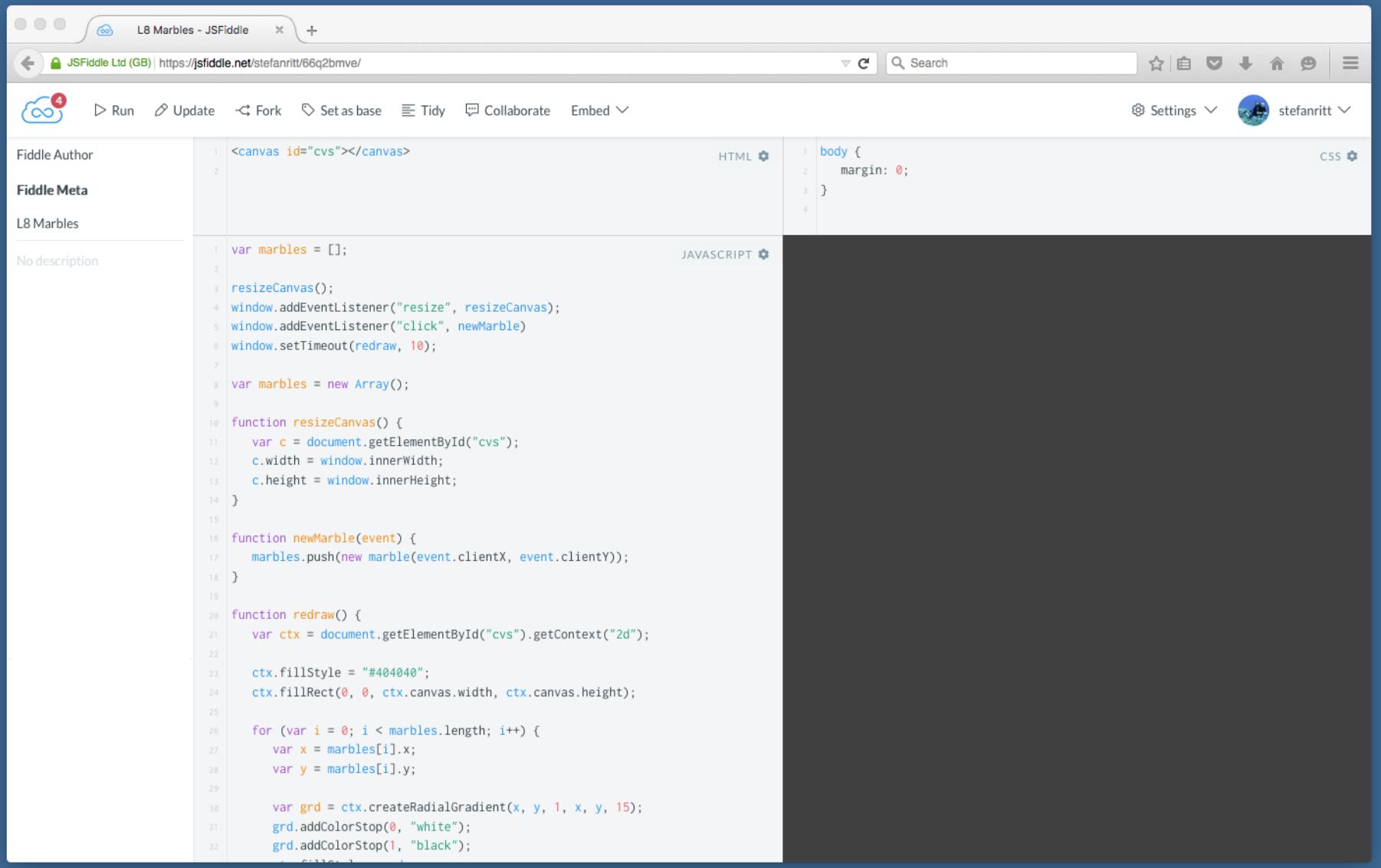
JavaScript + No-Library (pure JS) ▼

```javascript
var c = document.getElementById("cvs");
var ctx = c.getContext("2d");

ctx.fillStyle = "red";
ctx.fillRect(20, 20, 160, 160);

ctx.beginPath();
ctx.lineWidth = 4;
ctx.strokeStyle = "blue";
ctx.moveTo(20, 20);
ctx.lineTo(180, 180);
ctx.stroke();

ctx.beginPath();
ctx.arc(100, 100, 40, 0, 2 * Math.PI);
ctx.stroke();
```

Canvas Demo



More complex example



Balls 1

```
<head>
<script>
var balls = [];
var lastTime = 0;
var lastTimeFrame = 0;
var nFrames = 0;
var fps = 0;

function init() {
   resizeCanvas();

   // attach event handlers to canvas
   var c = document.getElementById("cvs");
   c.addEventListener("resize", resizeCanvas);
   c.addEventListener("click", newBall);
   c.addEventListener("touchstart", newBall, true);
   window.addEventListener("keypress", keyPress)

   window.requestAnimationFrame(frame);
}

function resizeCanvas()
{
   // stretch canvas to the whole client area
   var c = document.getElementById("cvs");
   c.width = document.documentElement.clientWidth;
   c.height = document.documentElement.clientHeight;
   if (c.height > 1000)
      c.height = 1000;
}
```

```
function newBall(e)
{
   // create new ball at cursor position
   if (e.type == "click")
      balls.push(new ball(e.clientX, e.clientY));
   else if (e.type == "touchstart") {
      e.preventDefault();
      balls.push(new ball(e.targetTouches[0].clientX,
                          e.targetTouches[0].clientY));
      }
}

function keyPress(e)
{
   var c = (typeof e.which == "number") ?
               e.which : e.keyCode;
   if (c == 'c'.charCodeAt(0)) // 'c' clears all balls
      balls = [];
   if (c == ' '.charCodeAt(0)) // space pushes up
      balls.forEach(function(m) {
         m.vy = Math.random() * 1000 - 1500;
         m.vx = Math.random() * 1000 - 500;
      });
   e.preventDefault();
}
```

Balls 2

```
function frame()
{
   var ctx =
document.getElementById("cvs").getContext("2d");

   // fill background
   ctx.fillStyle = "#606060";
   ctx.fillRect(0, 0, ctx.canvas.width,
ctx.canvas.height);

   // time since last frame
   var now = new Date().getTime(); // get time in ms
   var dt = lastTime > 0 ?
            (now - lastTime) / 1000 : 0;
   lastTime = now;

   // calculate frames per second
   nFrames++;
   if (now >= lastTimeFrame + 1000) {
      lastTimeFrame = now;
      fps = nFrames;
      nFrames = 0;
   }

   // display number of frames
   ctx.font = "20px sans-serif";
   ctx.fillStyle = "white";
   ctx.strokeStyle = "none";
   ctx.fillText(fps, 10, 20);
```

```
for (var i = 0; i < balls.length; i++) {
     var x = balls[i].x;
     var y = balls[i].y;

     var grd = ctx.createRadialGradient(x, y,
                  1, x, y, 15);
     grd.addColorStop(0, "white");
     grd.addColorStop(1, "black");
     ctx.fillStyle = grd;

     // draw ball
     ctx.beginPath();
     ctx.arc(x, y, 10, 0, 2 * Math.PI);
     ctx.fill();

     // integrate equations of motion
     balls[i].x += balls[i].vx * dt;
     balls[i].y += balls[i].vy * dt;
```

Balls 3

```
      // handle wall reflections
      if (balls[i].x + balls[i].vx*dt <= 10) {
         balls[i].x = 10;
         balls[i].vx = -balls[i].vx;
      }
      if (balls[i].x + balls[i].vx*dt >=
          ctx.canvas.width - 10) {
         balls[i].x = ctx.canvas.width - 10;
         balls[i].vx = -balls[i].vx;
      }
      if (balls[i].y + balls[i].vy*dt <= 10) {
         balls[i].vy = -balls[i].vy;
         balls[i].y = 10;
      }
      if (balls[i].y + balls[i].vy*dt >=
          ctx.canvas.height - 10) {
         balls[i].y = ctx.canvas.height - 10;
         balls[i].vy = -balls[i].vy;
      }

      // apply friction and gravity
      balls[i].vx *= 0.995; // friction
      balls[i].vy *= 0.995;
      balls[i].vy += 3;     // gravity
   }

   window.requestAnimationFrame(frame);
}
```

```
function ball(x, y)
{
   // initialize new ball with random velocity
   this.x = x;
   this.y = y;
   this.vx = Math.random() * 1000 - 500;
   this.vy = Math.random() * 1000 - 1000;
}
</script>

<style>
body { margin: 0; }
</style>
</head>

<body onload="init()">
<canvas id="cvs"></canvas>
</body>
```

Custom Slider

HTML ▼

```
<button name="slider" class="slider" type="button" data-update="update()"></button>
<div id="value"></div>
```

CSS ▼

```
body {
    font-family:sans-serif;
}
.slider {
    margin: 3px;
    width: 174px;
    height: 40px;
    border-radius: 5px;
    padding: 0px;
}
```

JavaScript + No-Library (pure JS) ▼

```
function Controls() {} // empty constructor

Controls.prototype.init = function()
{
    // scan DOM for sliders
    this.ctrlSlider = document.getElementsByName("slider");

    for (var i=0 ; i<this.ctrlSlider.length ; i++) {
        var cvs =  document.createElement("canvas");
        var sl = this.ctrlSlider[i];
        cvs.width = sl.clientWidth;
        cvs.height = sl.clientHeight;
        sl.appendChild(cvs);
        sl.canvas = cvs;

        sl.position = 0.5; // slider position 0...1
        sl.addEventListener("click",
```

0.59

Dialog Box

HTML ▼

```html
<div>
    <p>
        <input type="button" value="Dialog"
onClick="showDialog();">
    </p>
    <p id="bg">Background</p>
</div>

<div id="dialog" class="dialog">
    <div class="dialogTitle"
id="dialogTitle">Dialog Title</div>
    <div class="dialogContents">
        Dialog contents
        <br /> Dialog contents
```

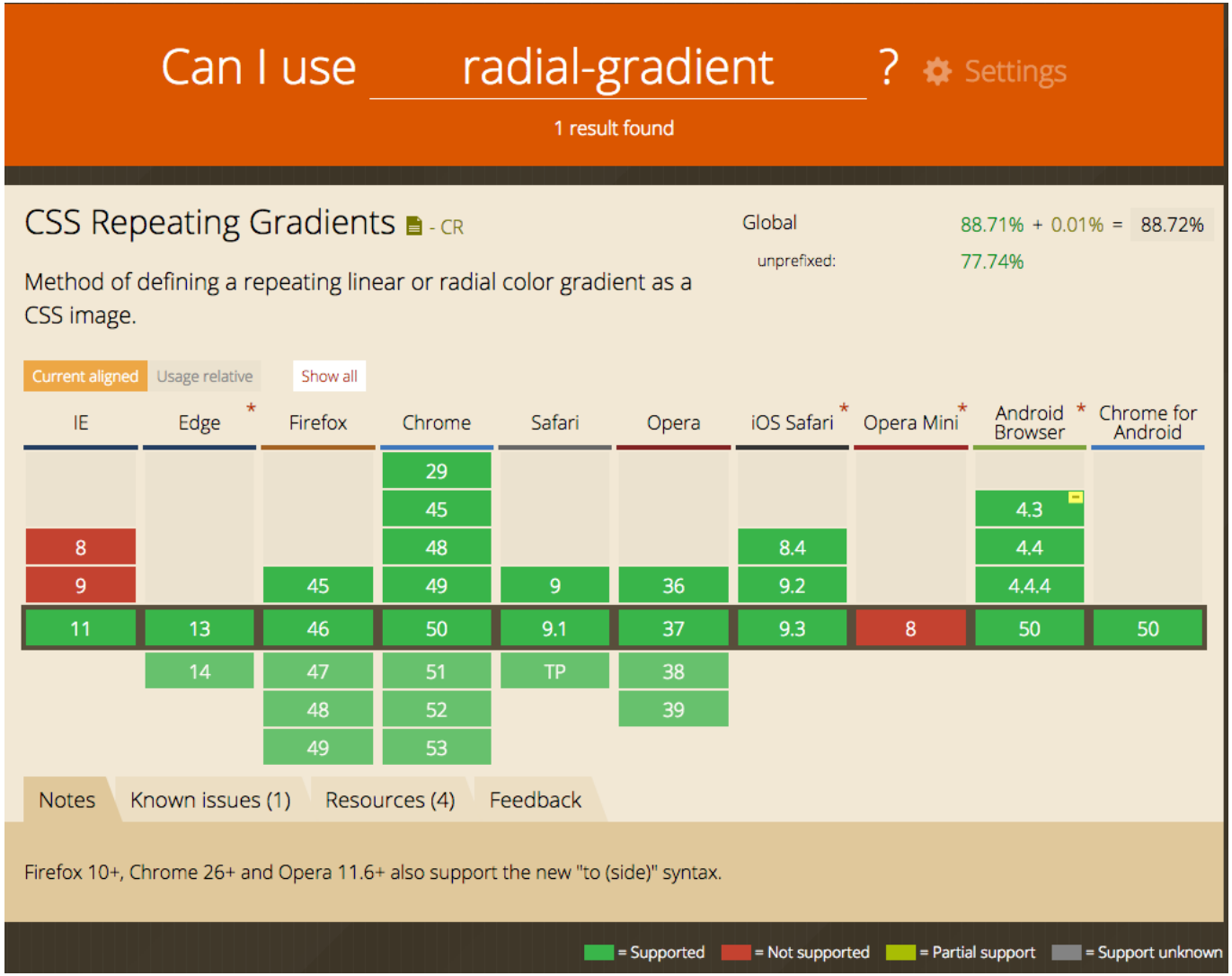
CSS ▼

```css
body {
    background-color:#E0FFE0;
}
#bg {
    font-family: sans-serif;
    font-size: 96px;
    color: #A0A0FF;
}
.dialog {
    font-family: sans-serif;
    background-color: #F0F0F0;
    width: 200px;
    text-align: center;
    padding: 0;
```

JavaScript + No-Library (pure JS) ▼

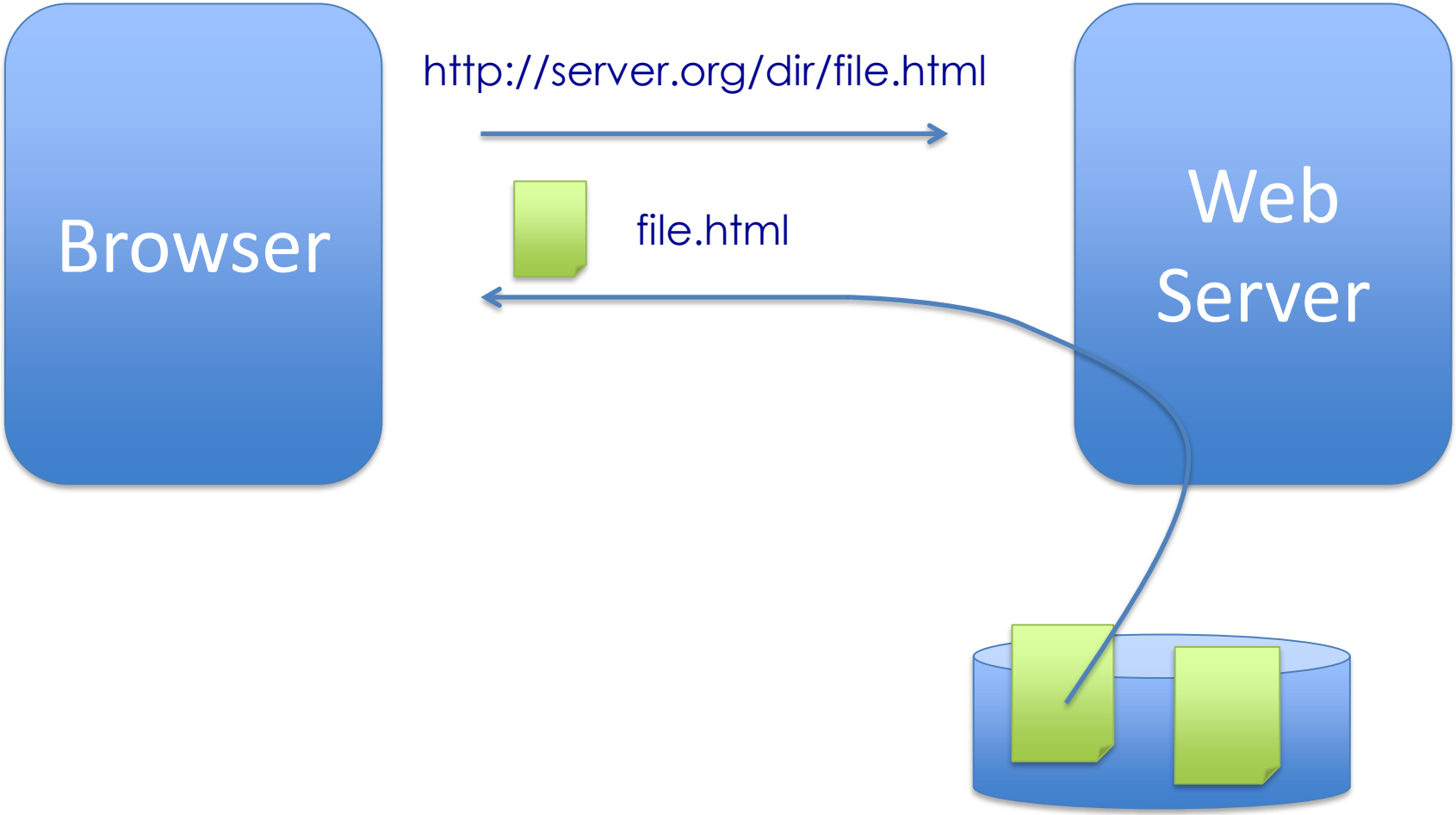
```javascript
function showDialog() {
    var e = document.getElementById("dialog");
    e.style.display = "block";
    e.style.left =
document.documentElement.clientWidth / 2 -
e.offsetWidth / 2 + "px";
    e.style.top =
document.documentElement.clientHeight / 2 -
e.offsetHeight / 2 + "px";

    window.addEventListener("mousedown",
dialogDrag, true);
    window.addEventListener("mousemove",
dialogDrag, true);
```

Browser Compatibility http://caniuse.com



Web servers

www.apache.org

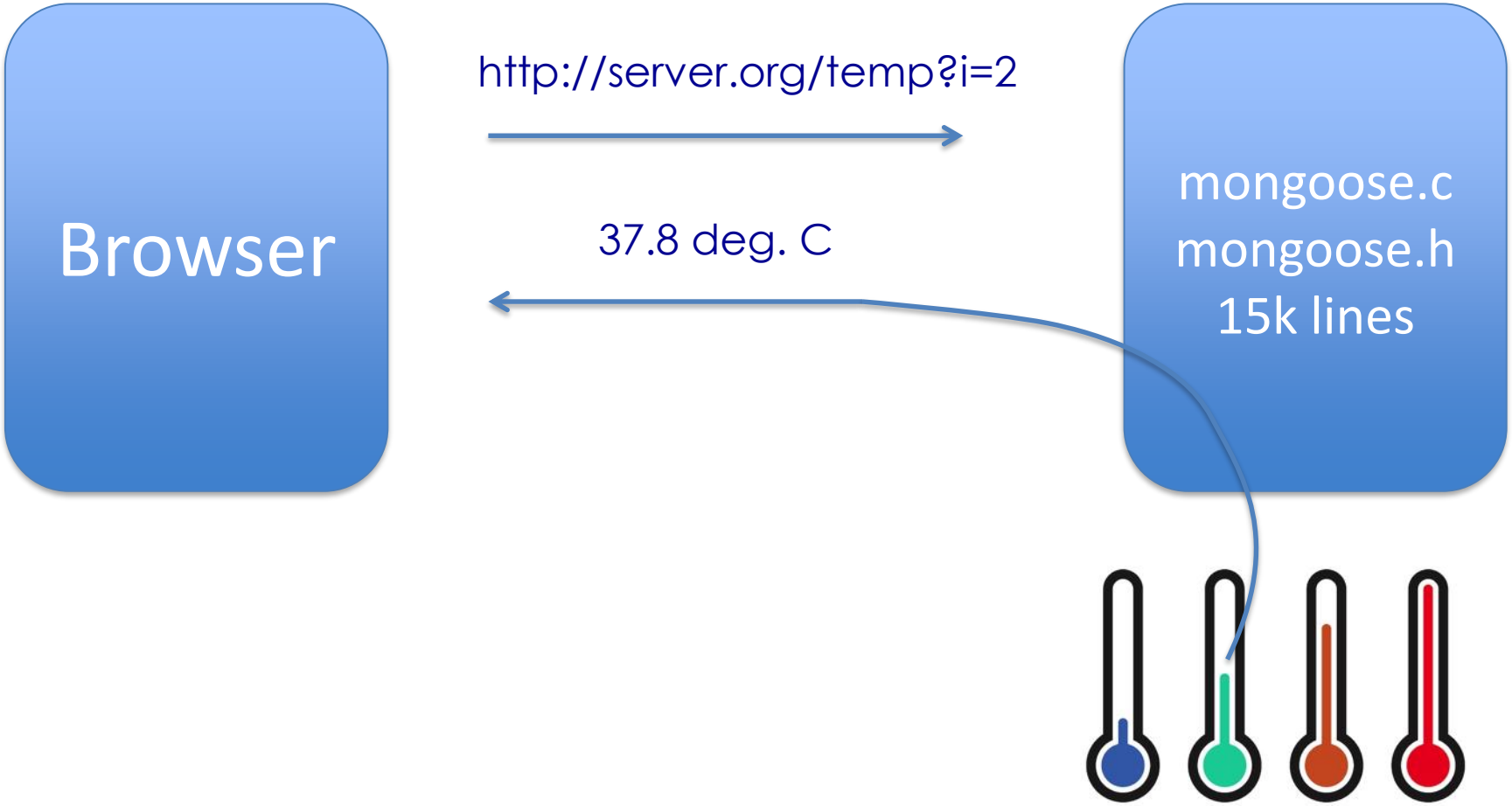


Web Servers

- Web servers are historically file based
- Apache is a large complex but powerful server
- For real-time control and monitoring, we need:
  - Slim server (e.g. embedded devices)
  - Servers with direct access to hardware
  - Support of protocols such as Websockets and SSL
  - Should compile everywhere
  - Easy to install and extend

Mongoose Server

https://github.com/cesanta/mongoose



Minimal mongoose program 1

```
#include "mongoose.h"

static struct mg_serve_http_opts s_http_server_opts;
static char *s_http_port = "8080";

int main(int argc, char *argv[])
{
  struct mg_mgr mgr;
  struct mg_connection *nc;

  mg_mgr_init(&mgr, NULL);
  nc = mg_bind(&mgr, s_http_port, ev_handler);
  if (nc == NULL) {
    fprintf(stderr, "Error starting server on port %s\n", s_http_port);
    exit(1);
  }

  s_http_server_opts.document_root = ".";
  s_http_server_opts.enable_directory_listing = "yes";

  printf("Starting server on port %s\n", s_http_port);

  while(1) {
    mg_mgr_poll(&mgr, 1000);
  }

  return 0;
}
```

Minimal mongoose program 2

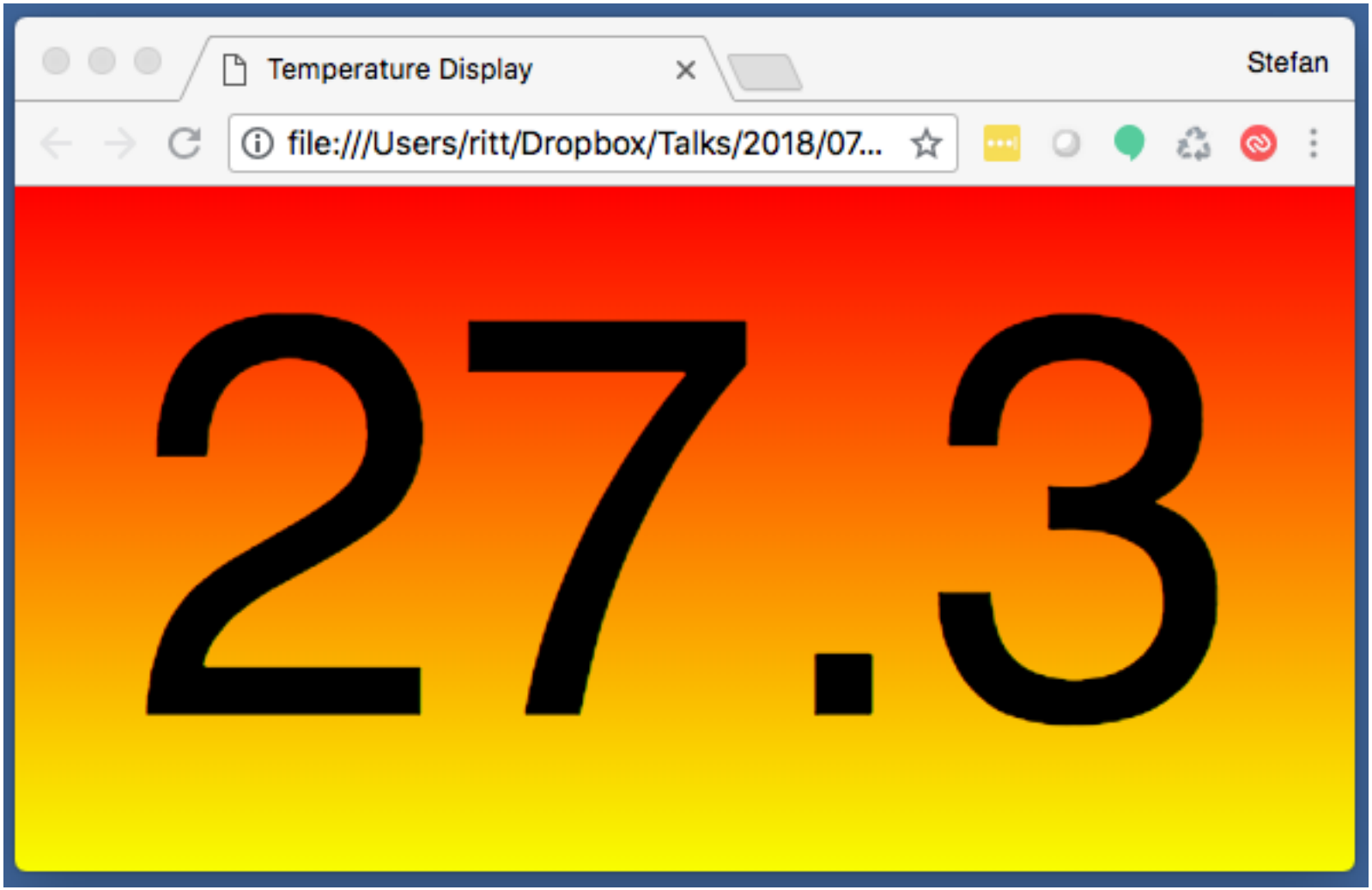
```
static void ev_handler(struct mg_connection *nc, int ev, void *ev_data)
{
  char str[100];
  int  index;
  double temp;
  struct http_message *hm = (struct http_message *) ev_data;

  if (ev == MG_EV_HTTP_REQUEST) {
    if (mg_vcmp(&hm->uri, "/temp") == 0) {

      // retrieve index from URL
      mg_get_http_var(&hm->query_string, "index", str, sizeof(str));
      index = atoi(str);

      // read temperature
      temp = 20+(int)(100.0*rand()/RAND_MAX)/10.0;

      // send reply
      mg_printf(nc, "%s", "HTTP/1.1 200 OK\r\nTransfer-Encoding: chunked\r\n\r\n");
      mg_printf_http_chunk(nc, "{\r\n");
      mg_printf_http_chunk(nc, "  \"index\": %d,\r\n", index);
      mg_printf_http_chunk(nc, "  \"temp\": %lg\r\n", temp);
      mg_printf_http_chunk(nc, "}\r\n");
      mg_send_http_chunk(nc, "", 0); // send empty chunk as end of response
    } else {
      // serve static content
      mg_serve_http(nc, hm, s_http_server_opts);
    }
  }
}
```

HTML page

```
<!DOCTYPE html>
<html>
<head>
   <title>Temperature Display</title>

<style>
   body {
      margin: 0;
   }
   #temp {
      height: 300px;
      background: yellow;
      background: linear-gradient(red,
yellow);

      font-family: sans-serif;
      font-size: 250px;
      text-align: center;
   }
</style>
```

```
<script>

window.setTimeout(load, 1000);

function load()
{
   var xhttp = new XMLHttpRequest();
   xhttp.onreadystatechange = function() {
      if (xhttp.readyState == 4 &&
          xhttp.status == 200) {
         o = JSON.parse(xhttp.responseText);
         document.getElementById("temp").innerHTML
= o.temp;
         window.setTimeout(load, 1000);
      }
   };
   xhttp.open("GET", "/temp?index=1", true);
   xhttp.send();
}

</script>
</head>

<body>
   <div id="temp">0.0</div>
</body>

</html>
```

Temperature Web Page

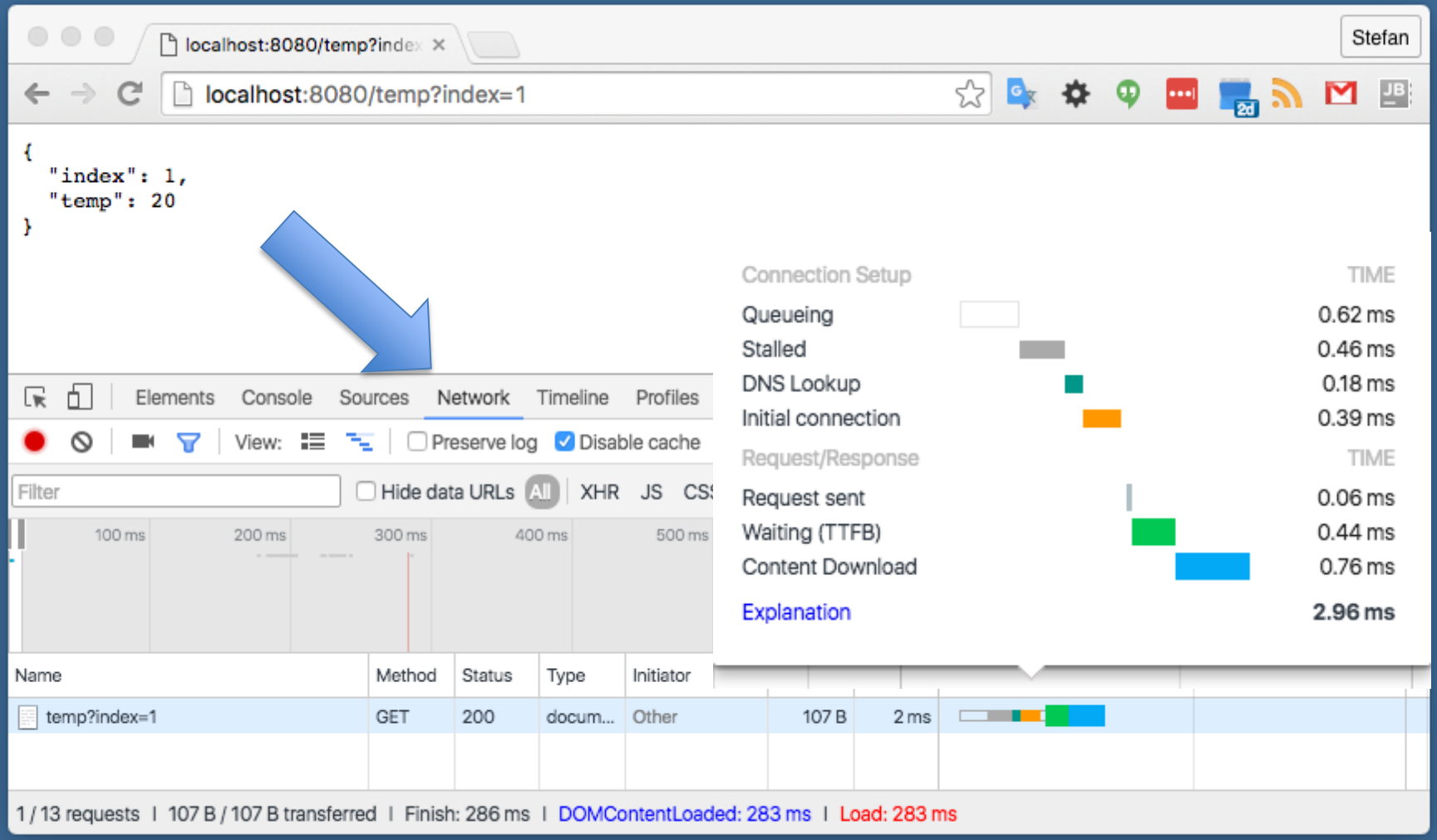


HTTP Requests

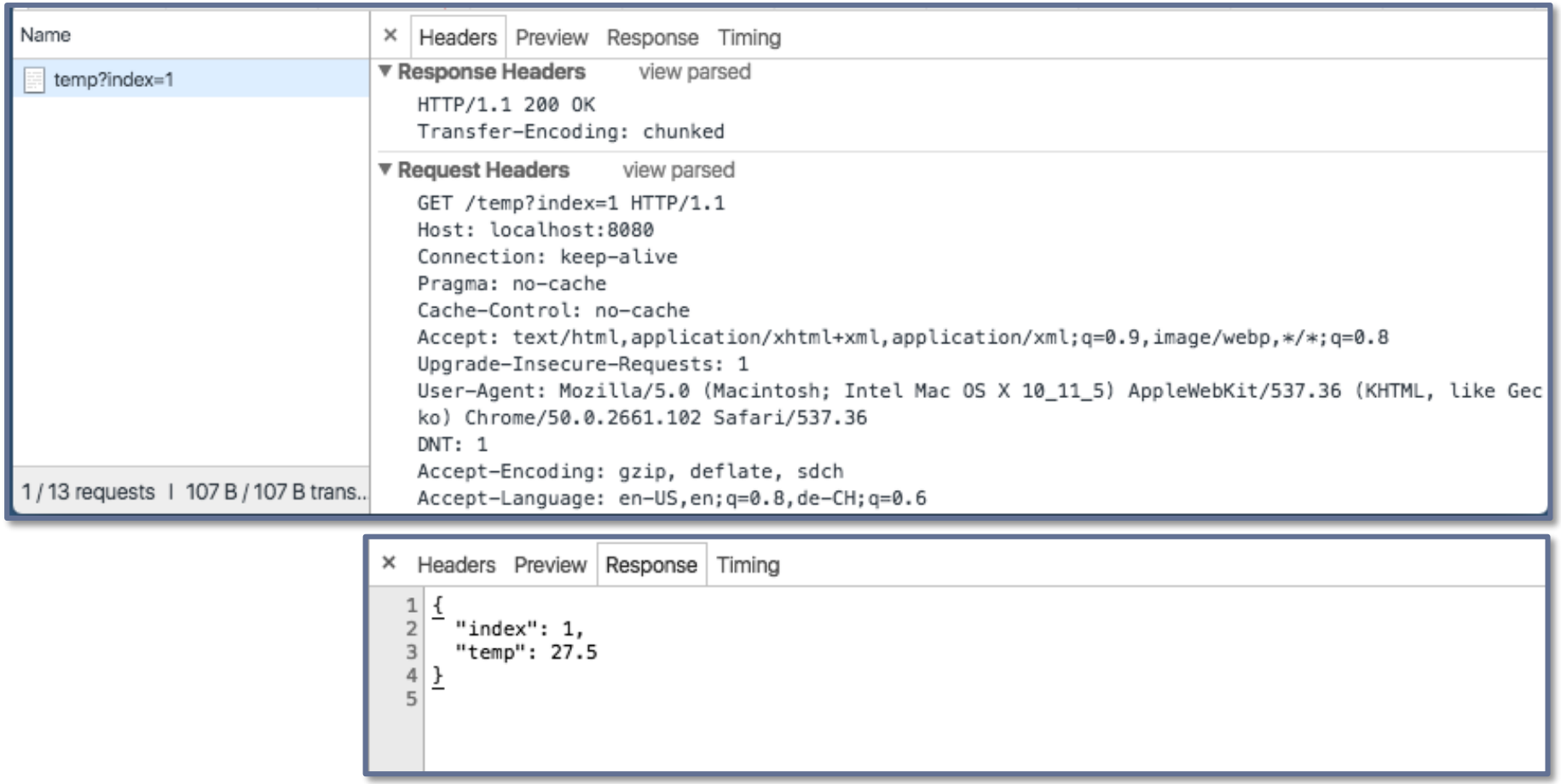
- Hypertext Transfer Protocol defined in https://tools.ietf.org/pdf/rfc2616.pdf
- For http:// requests, browser opens a TCP connection to port 80 of the server, sends a request header, and receives a response in plain text
- For https:// requests, encryption (SSL) is used to port 443
- Exercises with Raspberry Pi will use port 8080
- Typical request:

```
GET /temp?index=1 HTTP/1.1\r\n
```

Network monitor (Chrome Developer Tools)

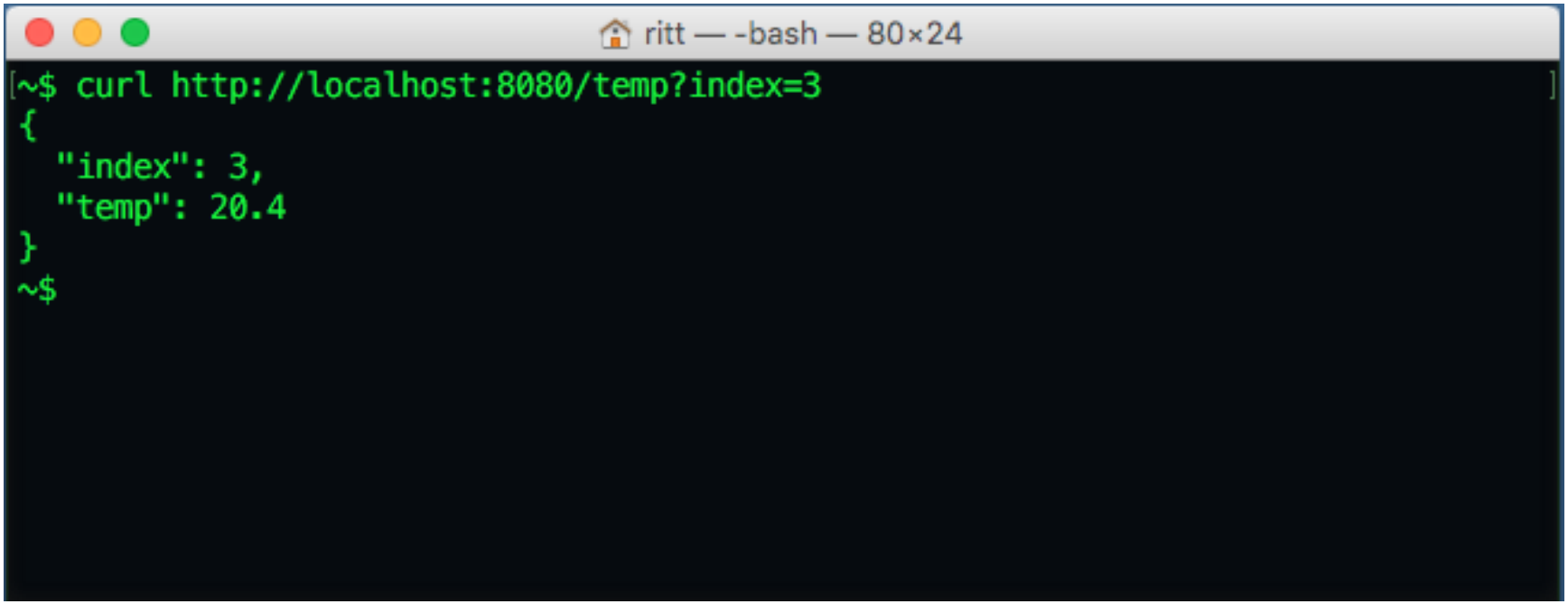


HTTP Request and Response



CURL

- Command line URL program to call servers from the terminal



JSON-RPC

- JSON-RPC implements Remote Procedure Calls:

→ {
"method":"getLight",
"params":
{"index":1},
"id":1234
}

← {
"result":"on",
"error":null,
"id":1234
}

- ID: value of any type to match the response with the request

JSON-RPC Client

```
function rpc_call()
{
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
      o = JSON.parse(xhttp.responseText);
      document.getElementById("result").innerHTML = "Result: "+o.result;
    }
  };
  var request = {};
  request.jsonrpc = "2.0";
  request.method = "sum";
  request.params = [];
  request.params[0] = parseFloat(document.getElementById("n1").value);
  request.params[1] = parseFloat(document.getElementById("n2").value);
  request.id = 1;

  xhttp.open("POST", "/json-rpc", true);
  xhttp.setRequestHeader("Content-type", "application/json");
  xhttp.send(JSON.stringify(request));
}
...
<body>
  <input type="text" id="n1">
  <input type="text" id="n2">
  <div id="result">Result:</div>
  <input type="button" value="Add" onClick="rpc_call()">
</body>
```

```
{
  “jsonrpc”:”2.0”,
  “method”:”sum”,
  “params”:[1,2],
  “id”:1
}
```

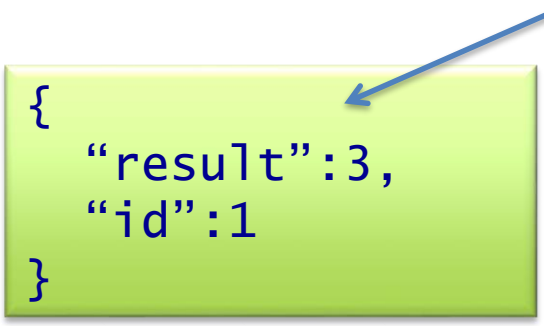
JSON-RPC Sever 1

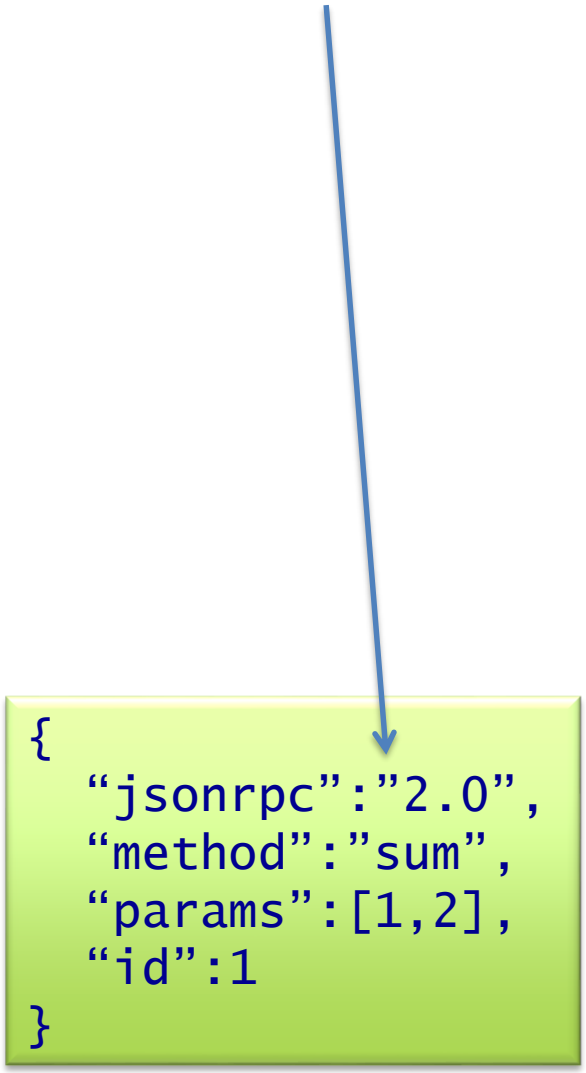
```
static void ev_handler(struct mg_connection *nc, int ev, void *ev_data)
{
  struct http_message *hm = (struct http_message *) ev_data;
  static const char *methods[] = { "sum", NULL };
  static mg_rpc_handler_t handlers[] = { rpc_sum, NULL };
  char buf[100];

  if (ev == MG_EV_HTTP_REQUEST) {
    if (mg_vcmp(&hm->uri, "/json-rpc") == 0) {
      mg_rpc_dispatch(hm->body.p, hm->body.len, buf, sizeof(buf),
                      methods, handlers);
      mg_printf(nc, "HTTP/1.0 200 OK\r\nContent-Length: %d\r\n"
                "Content-Type: application/json\r\n\r\n%s",
                (int) strlen(buf), buf);
      nc->flags |= MG_F_SEND_AND_CLOSE;
    } else {
      // serve static content
      mg_serve_http(nc, hm, s_http_server_opts);
    }
  }
}
```

```
{
  “jsonrpc”:”2.0”,
  “method”:”sum”,
  “params”:[1,2],
  “id”:1
}
```

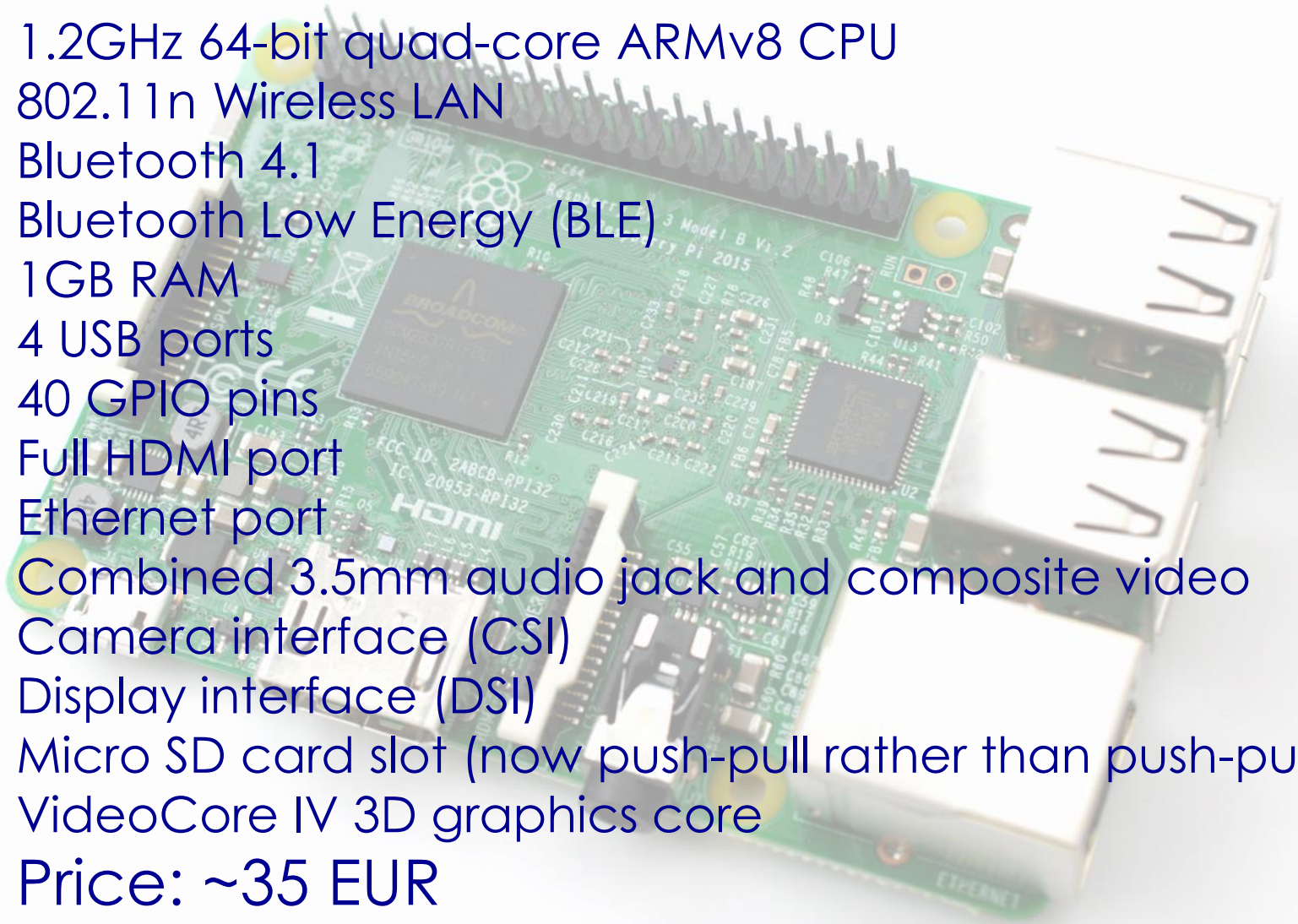
JSON-RPC Server1

```
static int rpc_sum(char *buf, int len, struct mg_rpc_request *req) {
  double sum = 0;
  int i;

  if (req->params[0].type != JSON_TYPE_ARRAY)
    return mg_rpc_create_std_error(buf, len, req,
                                   JSON_RPC_INVALID_PARAMS_ERROR);

  for (i = 0; i < req->params[0].num_desc; i++) {
    if (req->params[i + 1].type != JSON_TYPE_NUMBER)
      return mg_rpc_create_std_error(buf, len, req,
                                     JSON_RPC_INVALID_PARAMS_ERROR);

    sum += strtod(req->params[i + 1].ptr, NULL);
  }
  return mg_rpc_create_reply(buf, len, req, "f", sum);
}
```





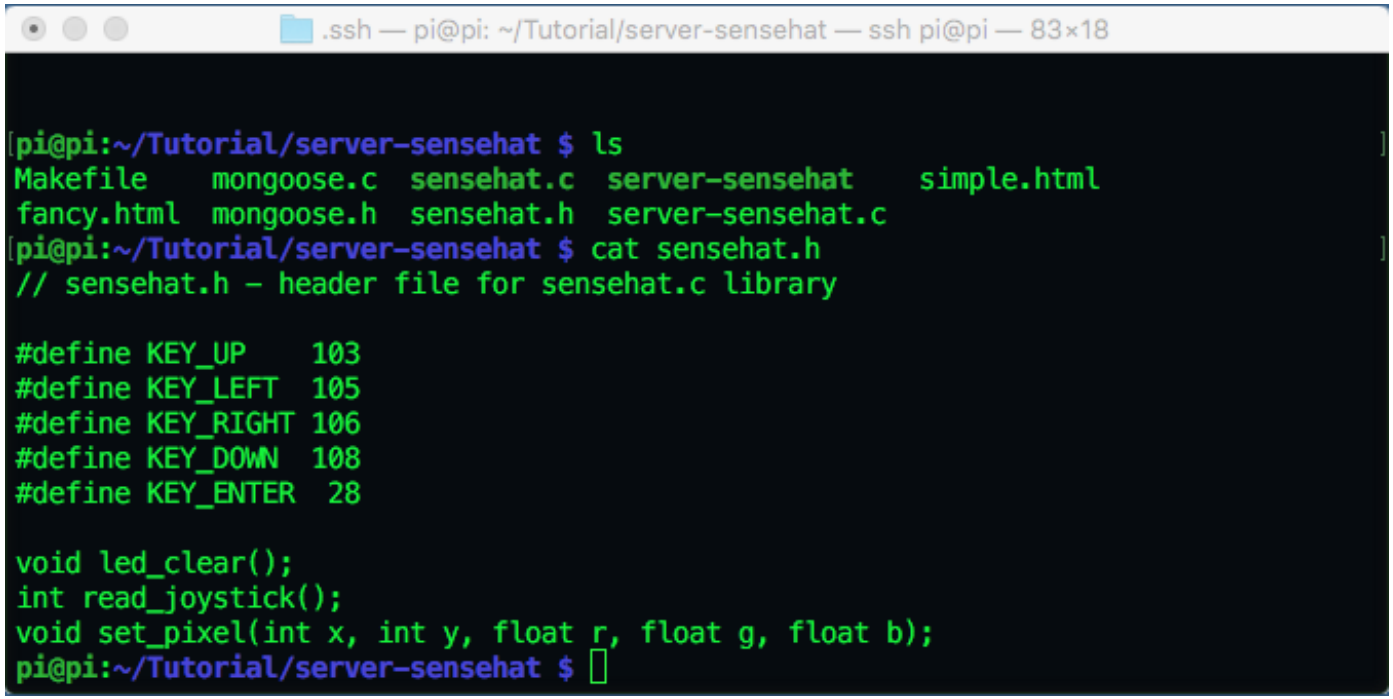
Raspberry Pi



- 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 1GB RAM
- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot (now push-pull rather than push-push)
- VideoCore IV 3D graphics core
- Price: ~35 EUR

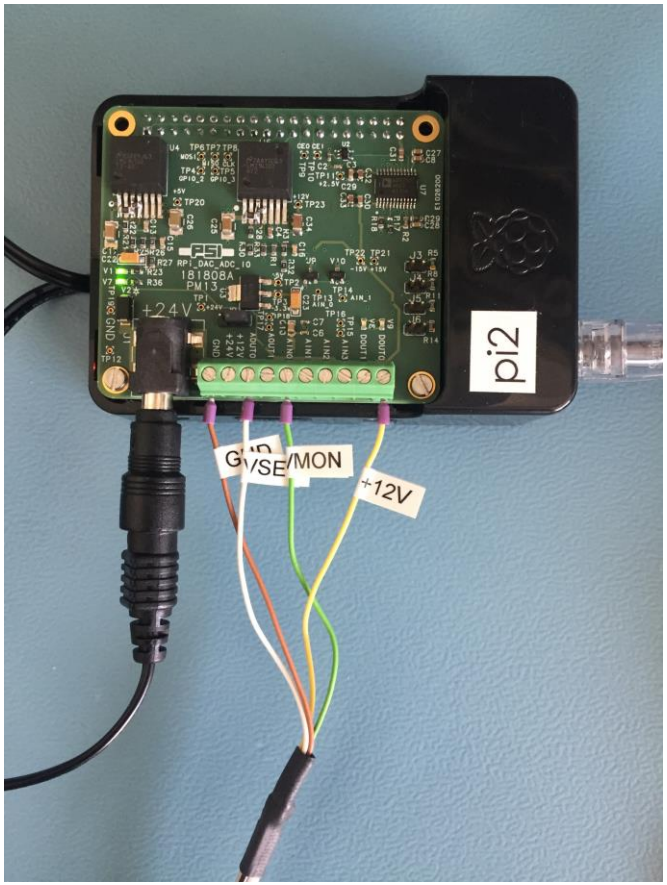
Mongoose on Raspberry Pi

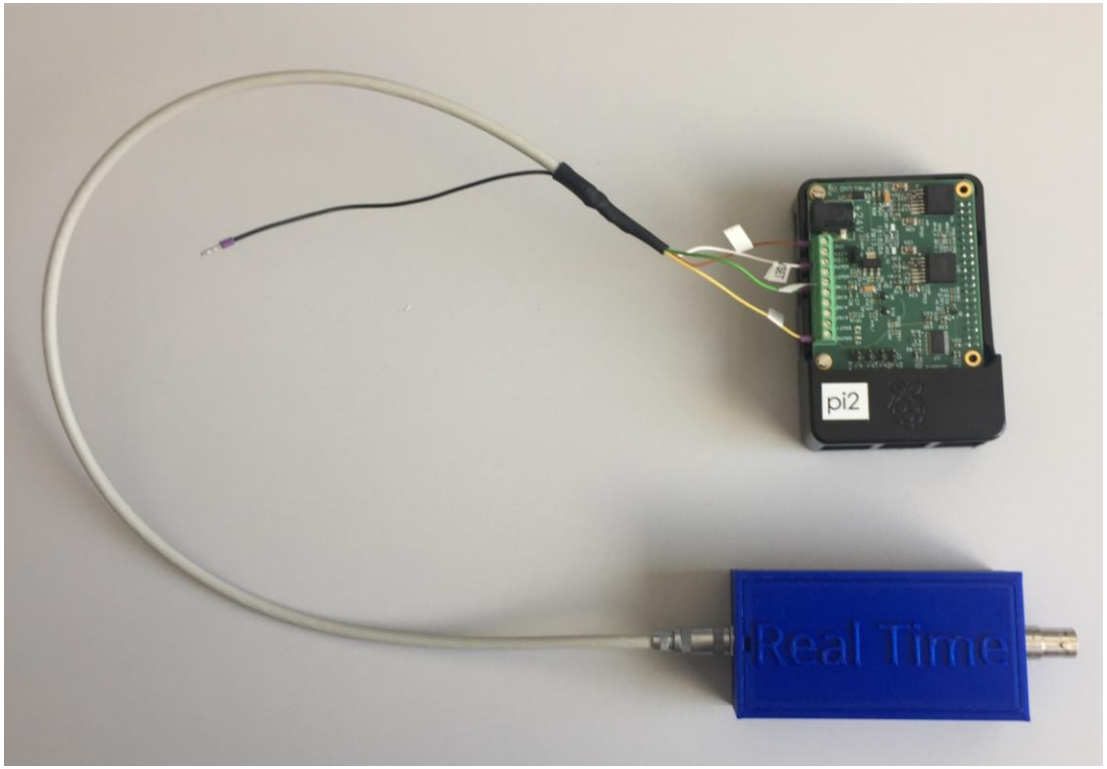
- mongoose compiles on Raspberry Pi out of the box
- For hardware access, one needs to link specific libraries to the server



ADC / DAC Board

- Raspberry Pi “hat” with 4 channel 16 bit ADC and 2 channel 16 bit DAC





Programming DAC through SPI

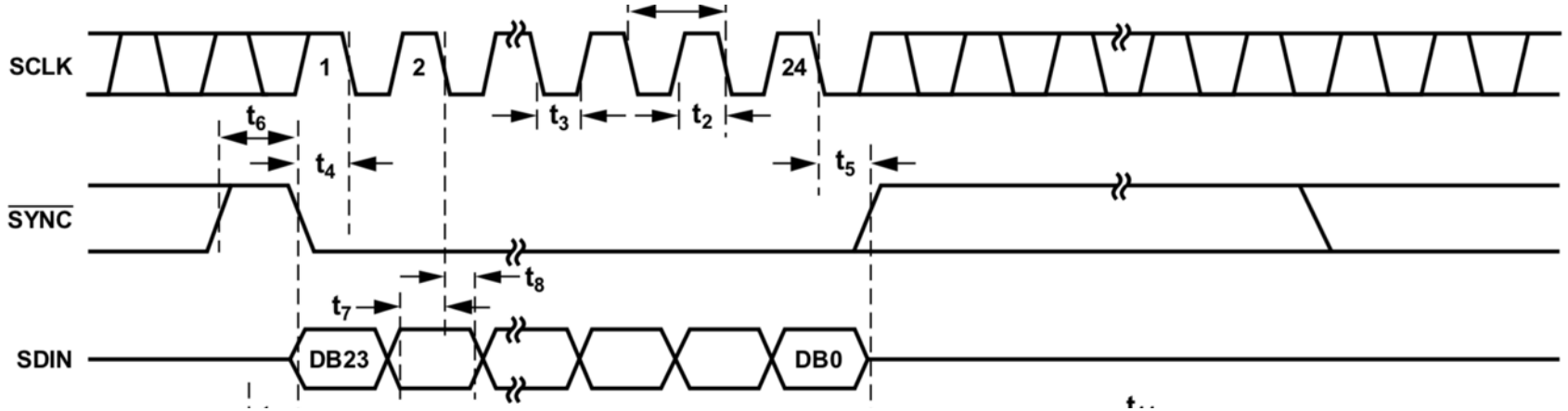


Table 18. Programming the AD5752 DAC Register

MSB LSB

R/$\overline{W}$	Zero	REG2	REG1	REG0	A2	A1	A0	DB15 to DB0
0	0	0	0	0	DAC address			16-bit DAC data

0x00 0x1234

Server 1

```
static int rpc_sethv(char *buf, int len, struct mg_rpc_request *req)
{
  double hv;
  unsigned char spi_buf[2];
  unsigned int d;

  hv = atof(req->params[1].ptr);

    d = (unsigned int) (hv / 2000 * 65535);
    spi_buf[0] = 0x00;
    spi_buf[1] = d >> 8;   // MSB
    spi_buf[2] = d & 0xFF; // LSB
    wiringPiSPIDataRW(spi_fd0, spi_buf, 3);

  return mg_rpc_create_reply(buf, len, req, "i", 1);
}
```

Server 2

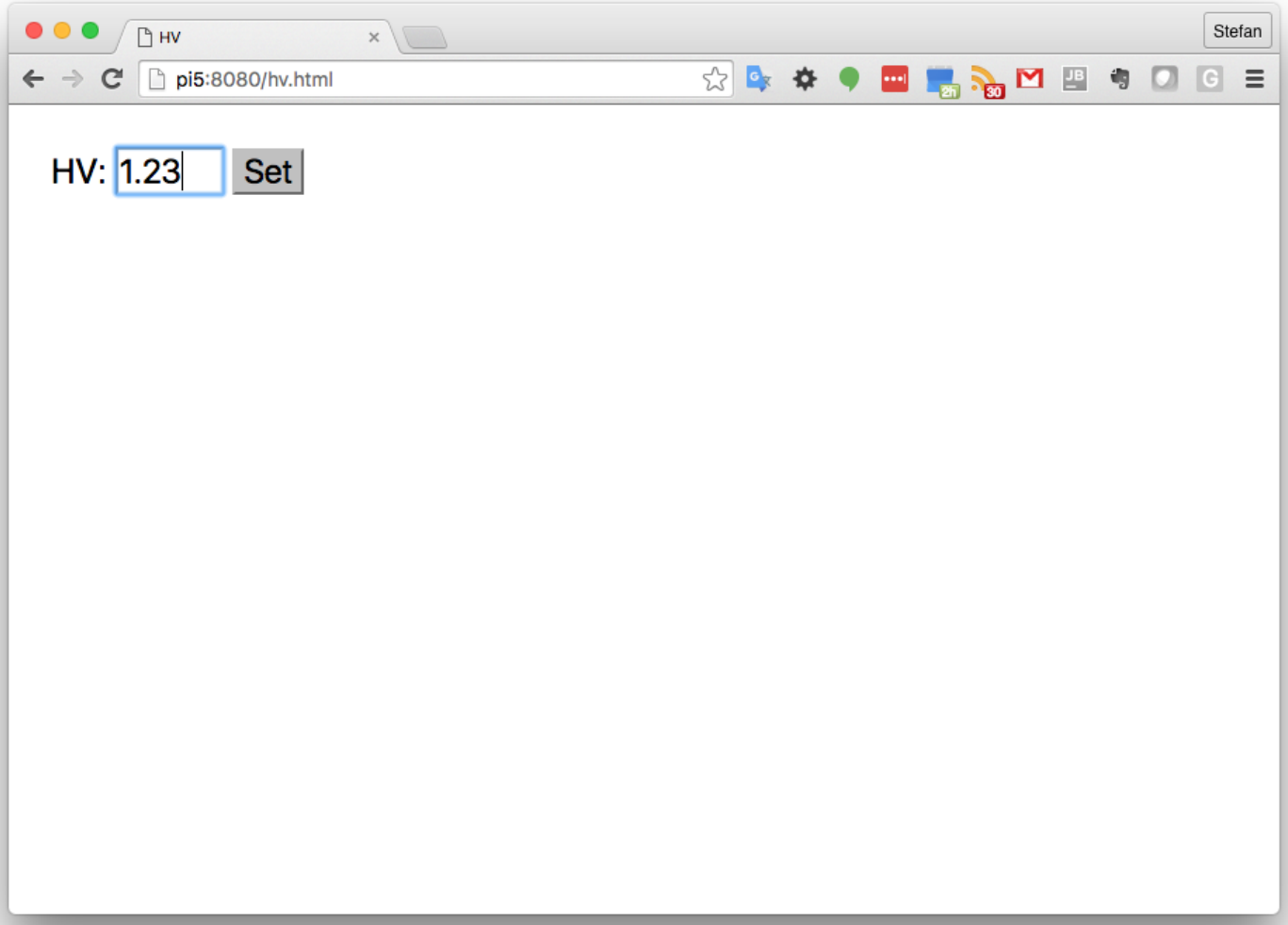
```
static void ev_handler(struct mg_connection *nc, int ev, void *ev_data)
{
  struct http_message *hm = (struct http_message *) ev_data;
  static const char *methods[] = { "sethv", NULL };
  static mg_rpc_handler_t handlers[] = { rpc_sethv, NULL };
  char buf[1000];

  if (ev == MG_EV_HTTP_REQUEST) {
    if (mg_vcmp(&hm->uri, "/json-rpc") == 0) {
      mg_rpc_dispatch(hm->body.p, hm->body.len, buf, sizeof(buf),
                      methods, handlers);
      mg_printf(nc, "HTTP/1.0 200 OK\r\nContent-Length: %d\r\n"
                "Content-Type: application/json\r\n\r\n%s",
                (int) strlen(buf), buf);
      nc->flags |= MG_F_SEND_AND_CLOSE;
    } else {
      // serve static content
      mg_serve_http(nc, hm, s_http_server_opts);
    }
  }
}
```

Client HV

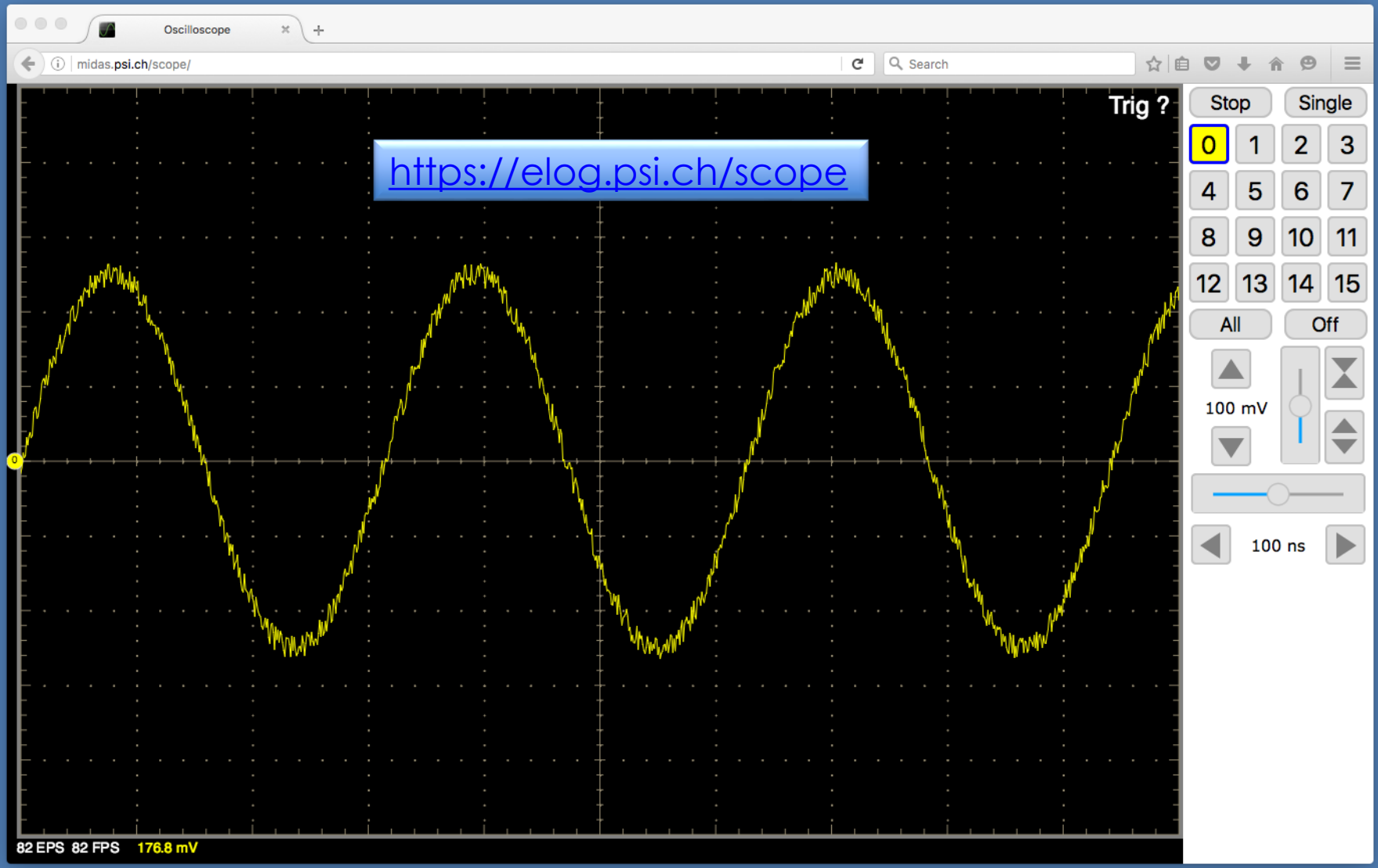
```
<!DOCTYPE html>
<html>
<head>
  <title>HV</title>

<style>
  body {
    font-family: sans-serif;
    font-size: 24px;
    margin: 30px;
  }
  input {
    line-height: 24px;
    font-size: 24px;
    -webkit-appearance:none;
  }
</style>
<script>
```

```
function rpc_sethv()
{
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
      o = JSON.parse(xhttp.responseText);
    }
  };

  var request = {};
  request.jsonrpc = "2.0";
  request.method = "sethv";
  request.params = [];
  request.params[0] = document.getElementById("hv").value;
  request.id = 1;

  xhttp.open("POST", "/json-rpc", true);
  xhttp.setRequestHeader("Content-type", "application/json");
  xhttp.send(JSON.stringify(request));
}

</script>
</head>

<body>
  HV: <input type="text" id="hv" value="0" size="5">
  <input type="button" value="Set" onclick="rpc_sethv()">
</body>

</html>
```

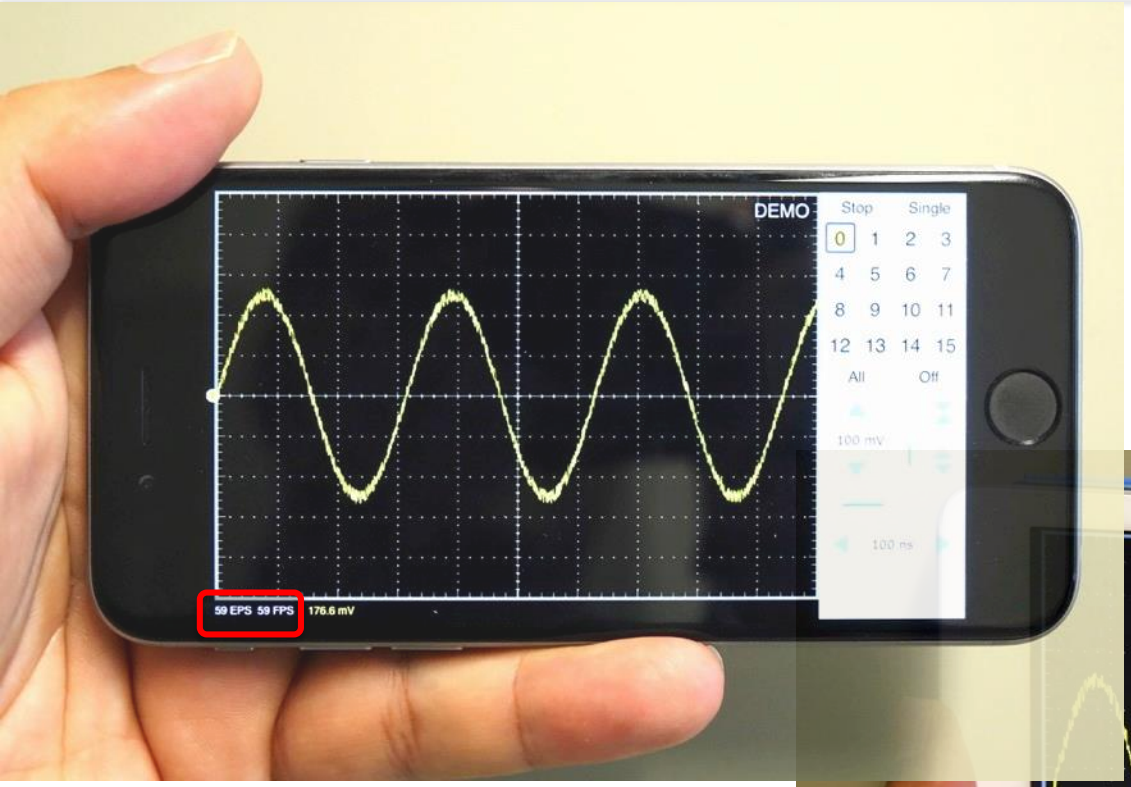
Client HV

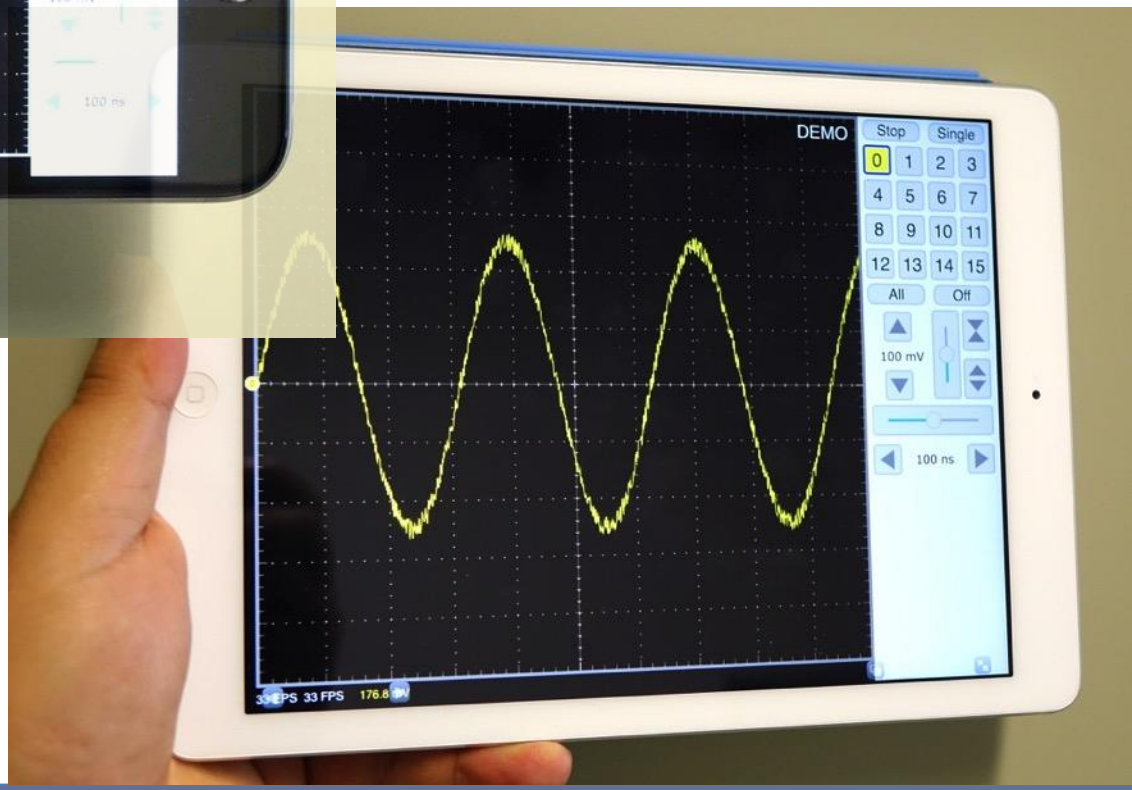


Oscilloscope Application



Smartphone and Tablet





Conclusion: Future of visualization

- Experiment data can be displayed on almost any browser
  - HTML5 is standardized through W3C
  - No software to install, no libraries required
  - Central updates of software
  - Only raw (binary) data needs to be transferred (as opposed to remote desktop applications)
  - Perfect for remote monitoring and control
  - Easy to learn (Scope took me one week from scratch)
- Web server can be incorporated into any web device ("Internet of Things (IoT)")
- Slides on Indico