Mobile Web Appplications Development with HTML5

Lecture 6: Web Sockets & Web Workers

Claudio Riva

Aalto University - Spring 2012

Lesson 6: Web Sockets and Web Workers

Push Technology

Web Sockets

Server-Sent Events

Web Workers

Push Technology

HTTP request/response protocol is designed for pulling information from a server and doesn't fit well for real-time communication

Typical applications that need server push: Instant messaging, E-mail, Auctions, Betting, Gaming, Sport results, Sensor monitoring

Workarounds

- Frequent requests (=> excessive load on the server)
- Long polling (=> keep many connections open in waiting state)
- Flash XMLSocket relays

WebSockets W3C

> Full-duplex, bi-directional communication over the Web. Both the server and client can send data at any time, or even at the same time. Only the data itself is sent, without the overhead of HTTP headers, dramatically reducing bandwidth.

```
//Socket object
var socket = new WebSocket("ws://echo.websocket.org?encoding=text");

// Send
socket.send('Hello, WebSocket');

// Callbacks
socket.onopen = function(event) { socket.send('Hello, WebSocket'); };
socket.onmessage = function(event) { alert(event.data); }
socket.onclose = function(event) { alert('closed'); }
```

WebSockets

Websockets interface:

- **`binaryType`** : `blob | arraybuffer`
- **`send`** `(DOMString | ArrayBuffer | Blob)`: send the message over the socket
- **`onopen`** : callback when the socket is ready (important!)
- **`onmessage`** : callback when a message is received on the socket
- **`onclose`** : callback when the socket is closed
- **`onerror`** : callback when there is a communication error

WebSockets - Benefits

- connection handshake uses HTTP infrastructure
- no extra ports (works across firewalls)
- no overhead
- very low latency
- clean browser interface

```
Request URL:ws://localhost:8080/
Request Method:GET
Status Code:101 Switching Protocols
Request Headers
Connection:Upgrade
Cookie:
Host:localhost:8080
Origin:null
Sec-WebSocket-Key:0/D8yRVfxGk/G4i4ye2MDA==
Sec-WebSocket-Version:13
Upgrade:websocket
(Key3):00:00:00:00:00:00:00:00
Response Headers
Connection:Upgrade
Sec-WebSocket-Accept:yBMC36LCd79X4ByuhFJfYJ3VS+Y=
Upgrade:websocket
(Challenge
Response):00:00:00:00:00:00:00:00:00:00:00:00:00:0
```

Echo Example websocketsEcho

HTML

```
%div(data-role="page")
%input#wsMessage(type='text')
%a#wsSend(data-role = 'button')
  Send
#console
```

Javascript

```
$(function() {
  var WebSocket = window.WebSocket || window.MozWebSocket;

  var socket = new WebSocket('ws://echo.websocket.org/echo?encoding=text');

  $('#wsSend').click(function() { socket.send( $('#wsMessage').val() ); })

  socket.onopen = function(event) { $('#console').append('<p>CONNECTED</p>'); };

  socket.onmessage = function(event) { $('#console').append('<p>' + event.data + '</p>'); }

  socket.onerror = function(event) { $('#console').append('<p>ERROR: ' + event.data + '</p>'); }
});
```

Sending/Receiving a JSON Object

- Sending a JSON object

```
var msg = { nickname: "John", message: "Hello World!" };
socket.send( JSON.stringify(msg) );
```

- Receiving a JSON object

```
socket.onmessage = function(event) {
  var msg = $.parseJSON( event.data );
  console.log( msg.nickname );
  console.log( msg.message );
}
```

Sending blob or arraybuffer

- To send a file(blob) from the browser to the server:

```
%input(id="fileSelect" type="file" multiple = "true")
```

```
$("#fileSelect").live('change', function(e) {
  $.each(this.files, function (i,file){
  socket.send(file);
  }
}
```

- To send array buffer from browser to server

```
var s = "Hello";
var ba = new Uint8Array(size);
for (var i = 0; i < size; i++) {
  ba[i] = s.charCodeAt(i);
}
socket.send(ba.buffer);
```

Receiving blob or arraybuffer

- To receive a blob from the server:

```
socket.binaryType = 'blob';
socket.onmessage = function (e) {
  // e.data is Blob object.
};
```

- To receive an array buffer from the server

```
socket.binaryType = 'arraybuffer';
socket.onmessage = function (e) {
  // e.data is ArrayBuffer object.
};

//To access arraybuffer as uint8 array,
var ba = new Uint8Array(e.data);
for (var i = 0; i < ba.length; i++) {
  ba[i];
}
```

The Server Side

LAMP stack and HTTP request/response not suitable

Coping with large number of open socket connections

Requires high concurrency at low performance cost (non-bocking IO)

Common frameworks based on event loop:

- Node.js
- Socket.IO
- Jetty (Java)
- Ruby / Event Machine
- Python / Twisted

Echo Server with Ruby Event Machine websocketsEchoEM

- Event-driven I/O (Reactor pattern)
- Extremly high scalability, performance and stability
- Hide complexity of high-performance threaded network programming

```ruby
require 'em-websocket'

EventMachine.run {
    EventMachine::WebSocket.start(:host => "0.0.0.0", :port => 8080) do |ws|
        ws.onopen {
          puts "WebSocket connection open"

          # publish message to the client
          ws.send "Hello Client"
        }
        ws.onclose { puts "Connection closed" }
        ws.onmessage { |msg|
          puts "Recieved message: #{msg}"
          ws.send "Pong: #{msg}"
        }
    end
}
```

Channels with Ruby Event Machine

Interface to push items to a number of subscribers

```
#Create a new channel
channel = EM::Channel.new

#Subscribe to the channel
sid = channel.subscribe{ |msg| p [:got, msg] }

#Push a message on the channel to all subscribers
channel.push('hello world')
channel.unsubscribe(sid)
```

The subscribe's callback function is used when pushing the message to the subscriber

Chat Client websocketsChatEM

HTML

```
%input#nickname(type="text" value="" placeholder="Nickname")
%input#message(type='text' value="" placeholder="Start chat here")
%a#send(data-role = 'button')
  Send
#console
```

JS

```
var socket = new WebSocket("ws://" + location.hostname + ":8080");

$('#send').click(function() {
  var msg = {nickname: $('#nickname').val(), message: $('#message').val()};
  socket.send( JSON.stringify(msg) );
  $('#message').val("");
})
socket.onopen = function(event) { $('#console').prepend('<p>CONNECTED</p>'); };
socket.onmessage = function(event) {
  var msg = $.parseJSON(event.data);
  $('#console').prepend('<p>' + msg.timestamp + ' <strong>' + msg.nickname + '</strong>: ' +
msg.message + '</p>');
}
socket.onerror = function(event) { $('#console').prepend('<p>ERROR: ' + event.data + '</p>'); }
```

Chat Server websocketsChatEM

```
EventMachine.run {
  @channel = EM::Channel.new
  @users = {}
  @messages = []

  EventMachine::WebSocket.start(:host => "0.0.0.0", :port => 8080) do |ws|
    ws.onopen { ... }
    ws.onmessage { |msg| ... }
    ws.onclose { ... }
  end

  #Run a Sinatra server for serving index.html
  class App < Sinatra::Base
    set :public_folder, settings.root

    get '/' do
      send_file 'index.html'
    end
  end
  App.run!
}
```

Chat Server - ws.onopen

```
ws.onopen {
  #Subscribe the new user to the channel with the callback function for the push action
  new_user = @channel.subscribe { |msg| ws.send msg }

  #Add the new user to the user list
  @users[ws.object_id] = new_user

  #Push the last messages to the user
  @messages.each do |message|
    ws.send message
  end

  #Broadcast the notification to all users
  @channel.push  ({
    "nickname" => '',
    "message" => "New user joined. #{@users.length} users in chat",
    "timestamp" => timestamp }.to_json)
}
```

Chat Server - ws.onmessage

```
ws.onmessage { |msg|

  #Add the timestamp to the message
  message = JSON.parse(msg).merge( {'timestamp' => timestamp}).to_json

  #append the message at the end of the queue
  @messages << message
  @messages.shift if @messages.length > 10

  #Broadcast the message to all users connected to the channel
  @channel.push message
}
```

Chat Server - ws.onclose

```
ws.onclose {
  @channel.unsubscribe(@users[ws.object_id])
  @users.delete(ws.object_id)

  #Broadcast the notification to all users
  @channel.push ({
    "nickname" => '',
    "message" => "One user left. #{@users.length} users in chat",
    "timestamp" => timestamp}.to_json)
}
```

Generic Server websocketsDrawEM

```
ws.onopen {
   #Subscribe the new user to the channel with the callback function for the push action
   new_user = @channel.subscribe { |msg| ws.send msg }

   #Add the new user to the user list
   @users[ws.object_id] = new_user

   #Push the last messages to the user
   @messages.each do |message|
     ws.send message
   end
}
 ws.onmessage { |msg|
   #append the message at the end of the queue
   @messages << msg
   @messages.shift if @messages.length > 10

   #Broadcast the message to all users connected to the channel
   @channel.push msg
 }
 ws.onclose {
   @channel.unsubscribe(@users[ws.object_id])
   @users.delete(ws.object_id)
 }
```

Websockets + canvas + orientation ? websocketsDriveEM

Protocol Versions and Browser Support

- **Hixie-75**
  - Chrome 4.0 + 5.0
  - Safari 5.0.0
- **HyBi-00/Hixie-76**
  - Chrome 6.0 - 13.0
  - Safari 5.0.2 + 5.1
  - iOS 4.2 + iOS 5
  - Firefox 4.0 - support for WebSockets disabled.
  - Opera 11 - with support disabled.
- **HyBi-07+**
  - Chrome 14.0
  - Firefox 6.0 - prefixed: MozWebSocket
  - IE 9 + IE 10 - via downloadable Silverlight extension
- **HyBi-10**
  - Chrome 14.0 + 15.0
  - Firefox 7.0 + 8.0 + 9.0 + 10.0 - prefixed: MozWebSocket
  - IE 10 (from Windows 8 developer preview)
- **HyBi-17/RFC 6455**
  - Chrome 16
  - Firefox 11

To enable support in Opera, type his in the address bar: `opera:config#Enable%20WebSockets`

To enable support in Firefox old version, type `about:config` and enable `network.websockets`

Pusher.com

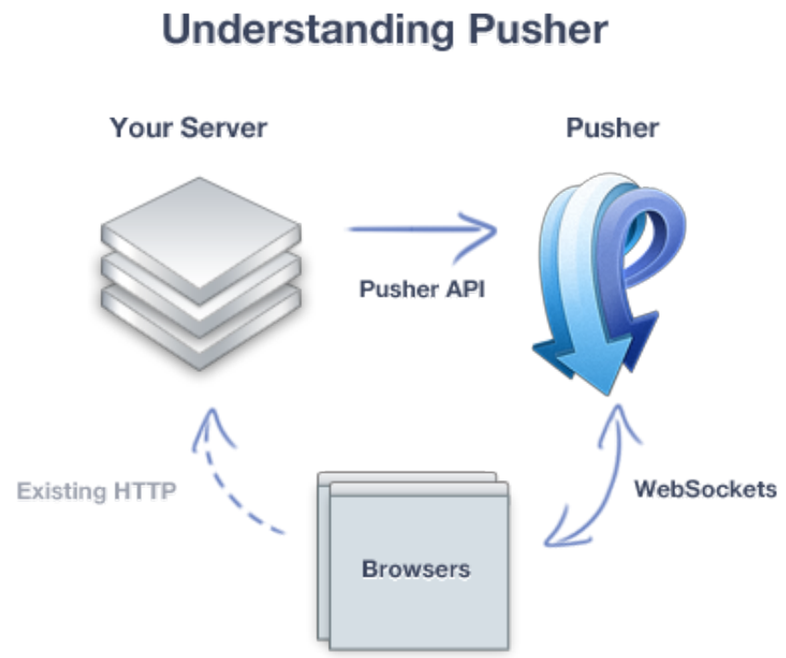
Pusher is a simple hosted API for quickly, easily and securely adding realtime bi-directional functionality via WebSockets to web and mobile apps, or any other Internet connected device.

Founded in 2009. Team is 7 members.

+40 billion messages delivered.

WebSocket as a service

Based on EM-WebSocket



Basic interface for pusher

- **Channels** : each application has a number of channels, and every client can choose which channels it connects to.

```
//Subscribe
var channel = pusher.subscribe(channelName);
//Unsubscribe
pusher.unsubscribe(channelName);
```

- **Events** : are 'named messages' that are sent to all clients that are registered to a channel

```
//Bind to an event on a channel
channel.bind(eventName, callback);
//Bind to an event on any channel
pusher.bind(eventName, callback);
```

- **Publisher (server)** : Publish an event to a channel on the server side

```
Pusher['test-channel'].trigger('test_event', '{"hello":"world"}')
```

Chat using Pusher - Client websocketsChatPusher

```
var pusher = new Pusher('8dc1dcd216474ec35b02');
var chatChannel = pusher.subscribe('chat');

chatChannel.bind('say', function(msg) {
  $('#console').prepend('<p>' + msg.timestamp + ' <strong>' + msg.nickname + '</strong>: ' +
msg.message + '</p>');
});

$('#send').click(function(event) {
  event.preventDefault();
  var msg = {nickname: $('#nickname').val(), message: $('#message').val()};

  $.post( 'http://' + location.host + '/say',msg,'json');
  $('#message').val("");
})
```

Chat using Pusher - Server websocketsChatPusher

```
require 'sinatra'
require 'json'
require 'pusher'

Pusher.app_id = '13324'
Pusher.key = '8dc1dcd216474ec35b02'
Pusher.secret = '6ae6292fa86e2a559643'

def timestamp
  Time.now.strftime("%H:%M:%S")
end

get '/' do
  #Serve the chat client
  File.read('index.html')
end

post '/say' do
  message = params.merge( {'timestamp' => timestamp}).to_json

  #Use the REST Pusher API so pass the message that needs to be broadcasted
  #to all clients that are connected to the chat channel
  Pusher['chat'].trigger('say', message)
end
```

Server-sent Events

- Possiblity to subscribe to a stream of updates that are generated by a server
- Only support one-way notificaions from the server
- Simpler to use than WebSockets
- Based on HTTP protocol (can be emulated in JS if not available)
- Automatic reconnect and resumable with event IDs

```
//Subscribe to the updates
var source = new EventSource('/events');

source.onopen = function(event) { console.log("CONNECTED"); }
source.onmessage = function (event) { console.log(event.data); }
source.onerror = function(event) { console.log(event.data); }
source.addEventListener('login', function(e) { console.log(e.data); }, false);
```

Message Format

```
data: this is a simple message
<blank line>
```

```
data: This is a message
data: on multiple lines
<blank line>
```

```
id: 25
data: This is a message with an id
<blank line>
```

```
id: 25
event: news
data: This message is of type news
<blank line>
```

```
data: {
data: "msg": "A JSON message",
data: "id": 12345
data: }
<blank line>
```

Sinatra Streaming API

It allows us to stream repsonses until the client closes the connection:

```
get '/' do
  stream do |out|
    out << "It's gonna be legen -\n"
    sleep 0.5
    out << " (wait for it) \n"
    sleep 1
    out << "- dary!\n"
  end
end
```

```
set :server, :thin
connections = []

get '/' do
  # keep stream open
  stream :keep_open { |out| connections << out }
end

post '/' do
  # write to all open streams
  connections.each { |out|
    out << params[:message] << "\n"
  }
  "message sent"
end
```

Chat using SSE - Server websocketsChatSSE

```ruby
users = []
messages = []

get '/' do
  send_file 'index.html'
end

get '/chat', provides: 'text/event-stream' do
  stream :keep_open do |out|
    users << out

    #callback is fired when the stream is closed.
    out.callback { users.delete(out) }
  end
end

post '/chat' do
  #Add the timestamp to the message
  message = params.merge( {'timestamp' => timestamp}).to_json

  #append the message at the end of the queue
  messages << message
  messages.shift if messages.length > 10

  users.each { |out| out << "data: #{message}\n\n"}
end
```

Chat using SSE - Client websocketsChatSSE

```
var es = new EventSource('/chat');
es.onmessage = function(e) {
  var msg = $.parseJSON(event.data);
  $('#console').prepend('<p>' + msg.timestamp + ' <strong>' + msg.nickname + '</strong>: ' +
msg.message + '</p>');
}

$('#send').click(function(event) {
  event.preventDefault();
  var msg = {nickname: $('#nickname').val(), message: $('#message').val()};

  $.post( '/chat',msg,'json');
  $('#message').val("");
})
```

WebWorkers W3C

- Javascript is single-threaded environment (one thread for UI events, data processing and DOM manipulation)
- Web Workers introduce an API for spawning background scripts for long-running scripts without blocking the UI
- Two types: Dedicated workers and shared workers
- Communication across threads happens via messages

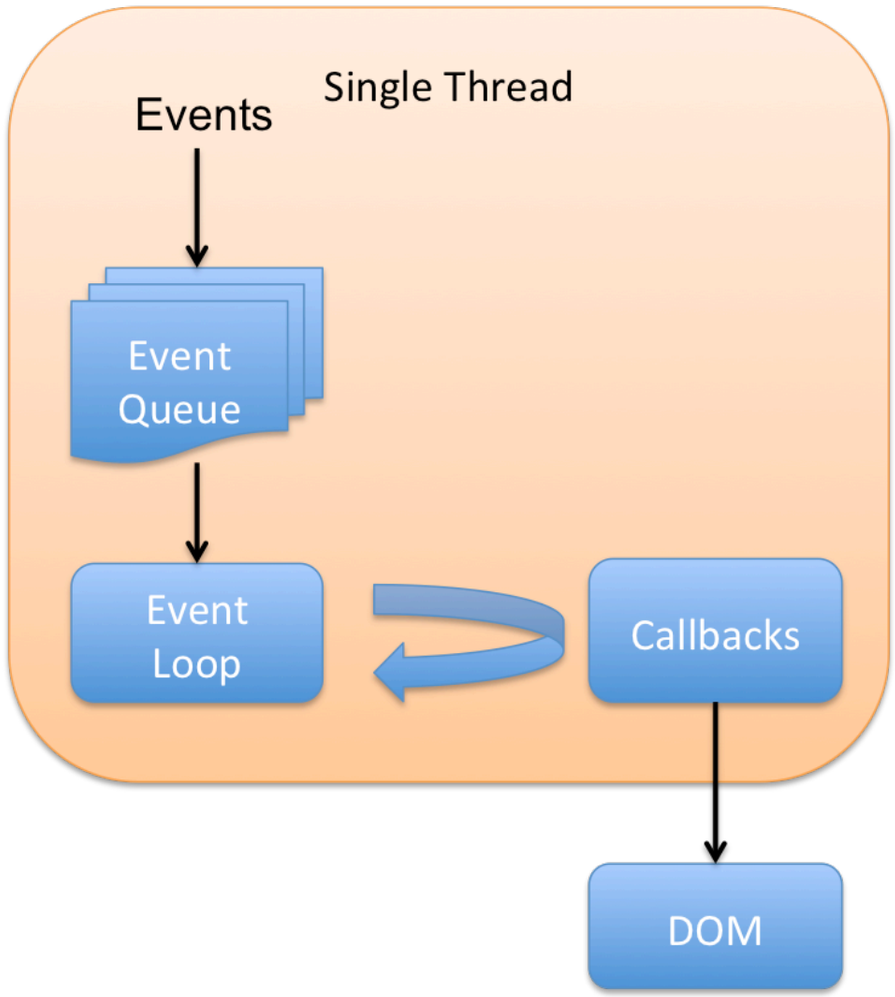
main.js

```
var worker = new Worker('worker.js');
worker.postMessage('Hello Worker');
worker.onmessage = function (e) {
  console.log(e.data);
}
```

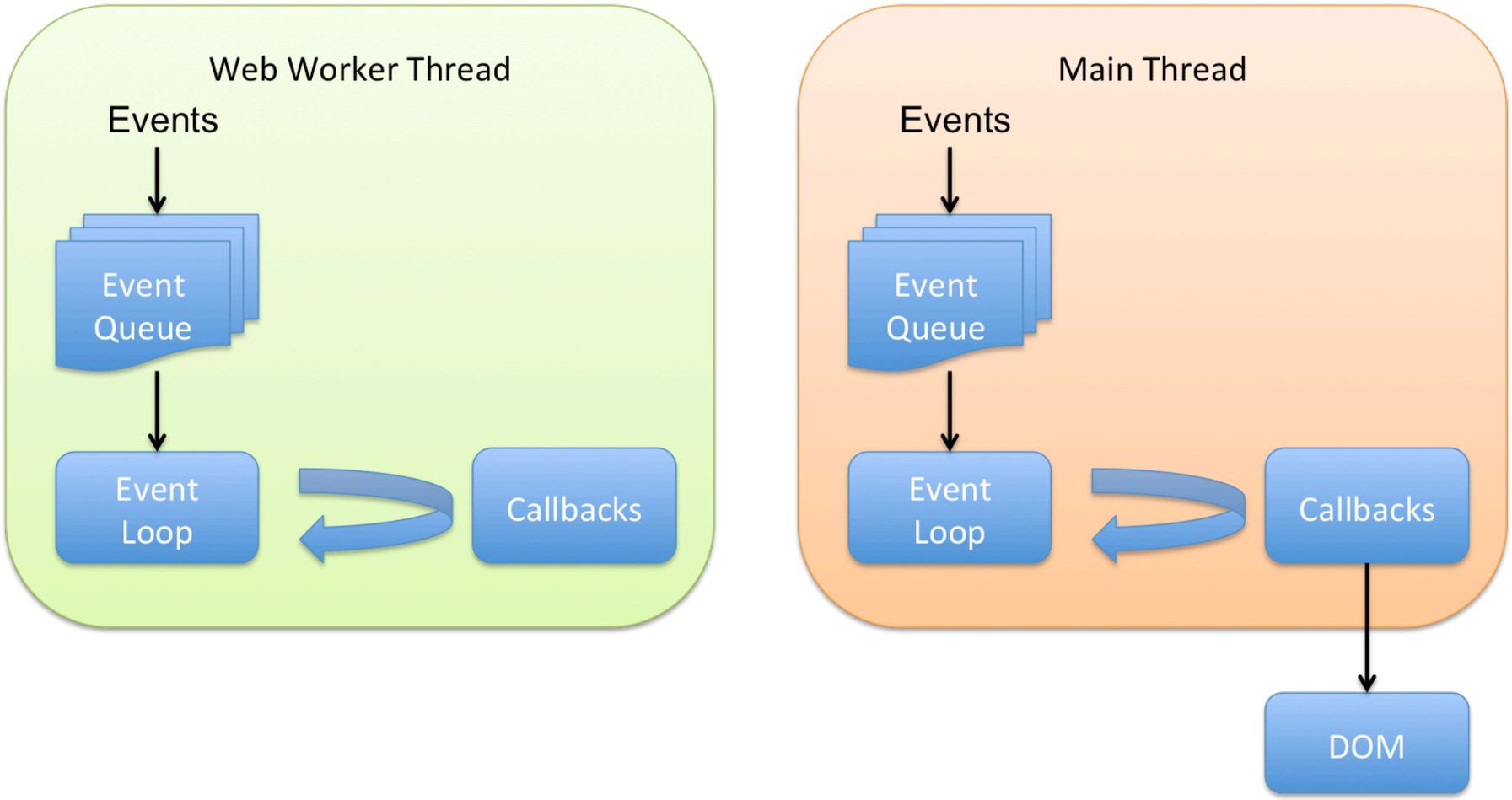
worker.js

```
this.onmesage = function(e) {
  postMessage('You said' + e.data);
}
```

Javascript Event Loop



Webworkers and Main Thread



What a worker can do and not do

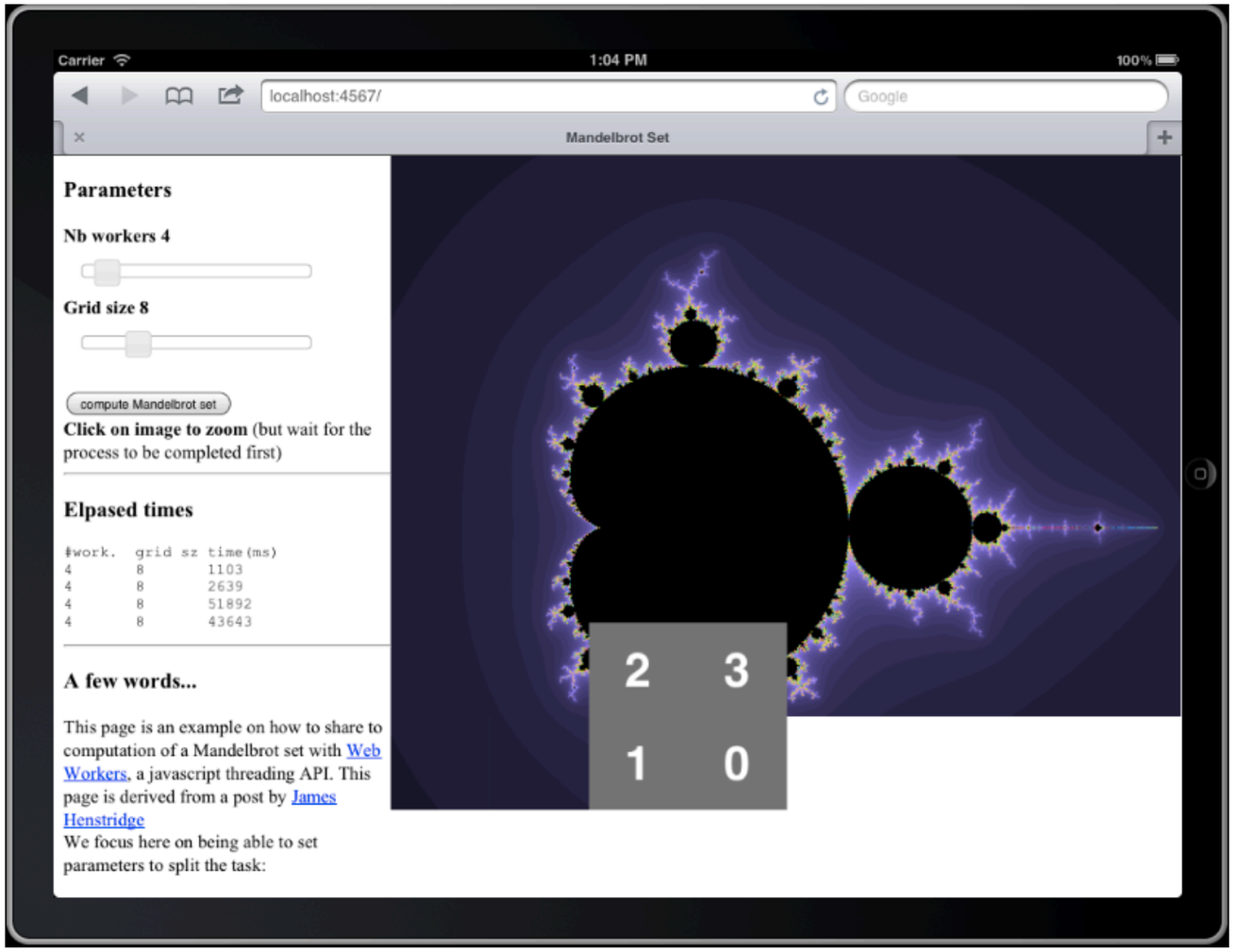
What a worker can do:

- Post and listen for messages
- Close the worker
- Access navigator and location objects
- XMLHttpRequest
- Use timers
- Application cache
- importScripts()
- Websockets and server-sent events
- IndexedDB
- Spawn other workers

What a worker cannot do:

- Access the DOM
- Access the window object
- Access the document object
- Access the parent object

Mandelbrot Renderer



The worker

```
self.onmessage = function (event) {
  var data = event.data;
  var max_iter = data.max_iter;
  var escape = data.escape * data.escape;
  data.values = [];
  for (var j = data.j_start; j < data.j_end; j++) {
    var c_j = data.r_min + (data.r_max - data.r_min) * (j-data.j_start) / data.height;
    for (var i = data.i_start; i < data.i_end; i++) {
      var c_i = data.c_min + (data.c_max - data.c_min) * (i-data.i_start) / data.width;
      var z_j = 0, z_i = 0;
      for (iter = 0; z_j*z_j + z_i*z_i < escape && iter < max_iter; iter++) {
  // z -> z^2 + c
  var tmp = z_j*z_j - z_i*z_i + c_i;
  z_i = 2 * z_j * z_i + c_j;
  z_j = tmp;
      }
      if (iter == max_iter) {
  iter = -1;
      }
      data.values.push(iter);
    }
  }
  self.postMessage(data);
}
```

Main Thread

```
init : function(options){
  var me=this;
  this.workers = [];

  this.nb_workers=options.nbworkers;

  for (var i = 0; i < this.nb_workers; i++) {
    var worker = new Worker("worker.js");
    worker.label=""+i;
    worker.onmessage = function(event) {
      me.received_block(event.target, event.data)
    }
    worker.idle = true;
    this.workers.push(worker);
  }
  ...
}
process_block: function(worker) {
  ...
  worker.postMessage(data);
  ...
}
```

Message passing

Messages passed between threads are copied not shared

Object is serialized and de-serialized each time

Supported objects by postMessage:

- RegExp
- Blob, File and FileList
- ImageData

Transfarable objects for high-performance operations (only in Chrome 17)