JAVASCRIPT BASICS

Jakub Baierl & Šimon Lomič

JAVASCRIPT DEVELOPERS @ ACKEE

ackee

NEW TECHNOLOGIES

Why I should use Web workers?

- Javascript is single-thread environment
- Multiple scripts cannot run at the same time
- UI events, query and process large amounts of API data, and manipulate the DOM in the same time?
- Script execution happens within a single thread

WEB WORKERS

Developers simulate 'concurrency'

- setTimeout(), setInterval(), XMLHttpRequest, and event handlers
- Yes, all of these features run asynchronously

BUT...

non-blocking doesn't necessarily mean concurrency.

Asynchronous events are processed after the current executing script has yielded.

What Web worker actually is?

- The Web Workers specification defines an API for spawning background scripts in your web application
- Web Workers allow you to do things like fire up long-running scripts, but without:
    - blocking the UI
    - blocking other scripts to handle user interactions

WEB WORKERS

- Web Workers run in an isolated thread

- As a result, the code that they execute needs to be contained in a separate file

```
var worker = new Worker('task.js');
```

- If the specified file exists, the browser will spawn a new worker thread which is downloaded **asynchronously**

```
worker.postMessage(); // Start the worker.
```

How to communicate?

- Communication between a work and its parent page is done using:
    - an event model
    - and the postMessage() method

```
self.addEventListener('message', function(e) {
    self.postMessage(e.data);
}, false);
```

WEB WORKERS

Workers do **NOT** have access to:

- The DOM (it's not thread-safe)
- The window object
- The document object
- The parent object

What Sub Workers are?

- Workers have the ability to spawn child workers
- Must be hosted on **same origin** as parent

What Inline Workers are?

- What if you want to create your worker script on the fly, or create a self-contained page **without** having to create separate worker files?

With **Blob()**, you can "inline" your worker in the same HTML file as your main logic

Real-time communication without?

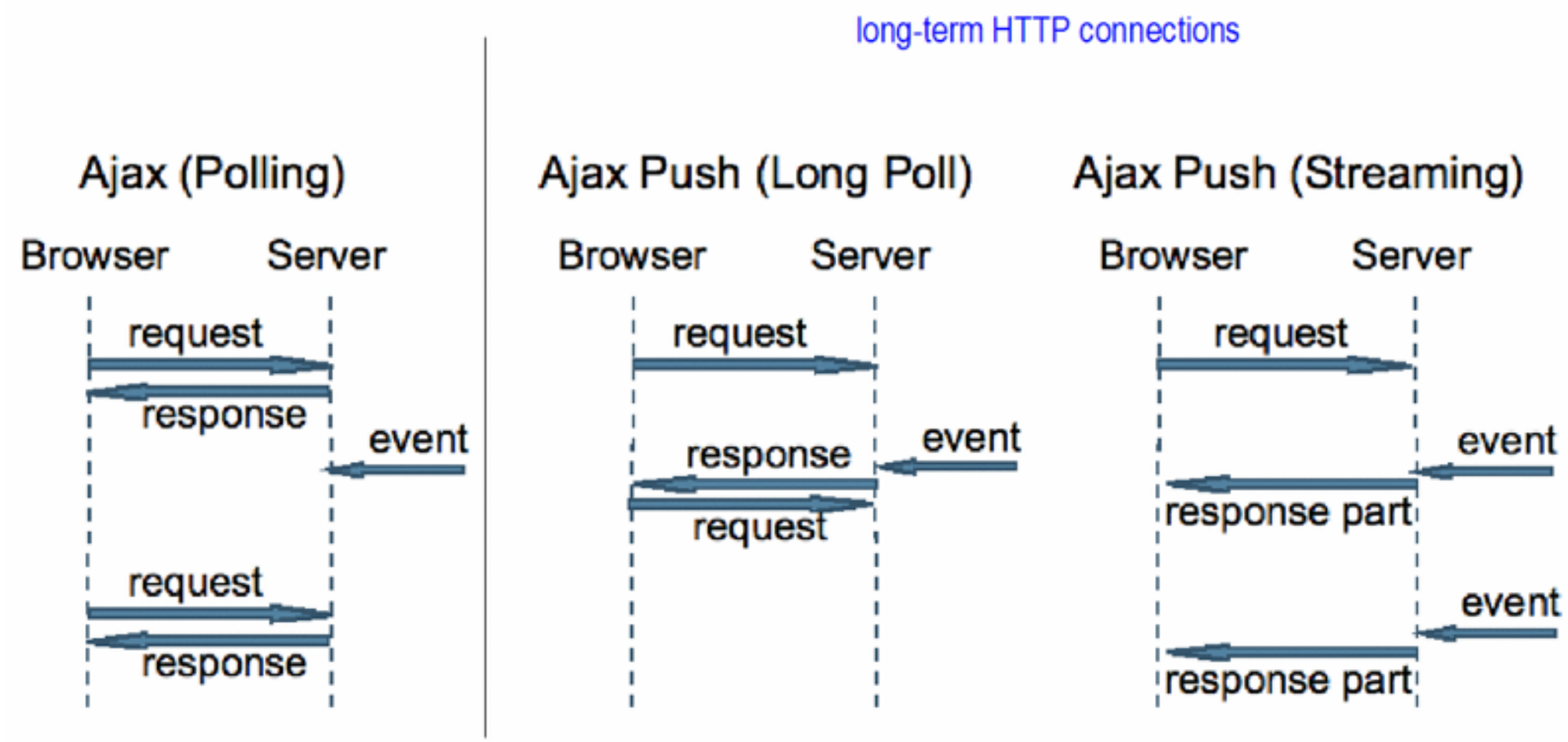
- All HTTP communication was steered by the client
    - user interaction
    - periodic **polling**

… to load new data from the server

Technologies that enable the server to send data to the client
"**Push**" or "**Comet**"

What about **long polling/streaming**?

- Client polls the server for new data
- Server holds the request open until new data is available



What Websocket actually is?

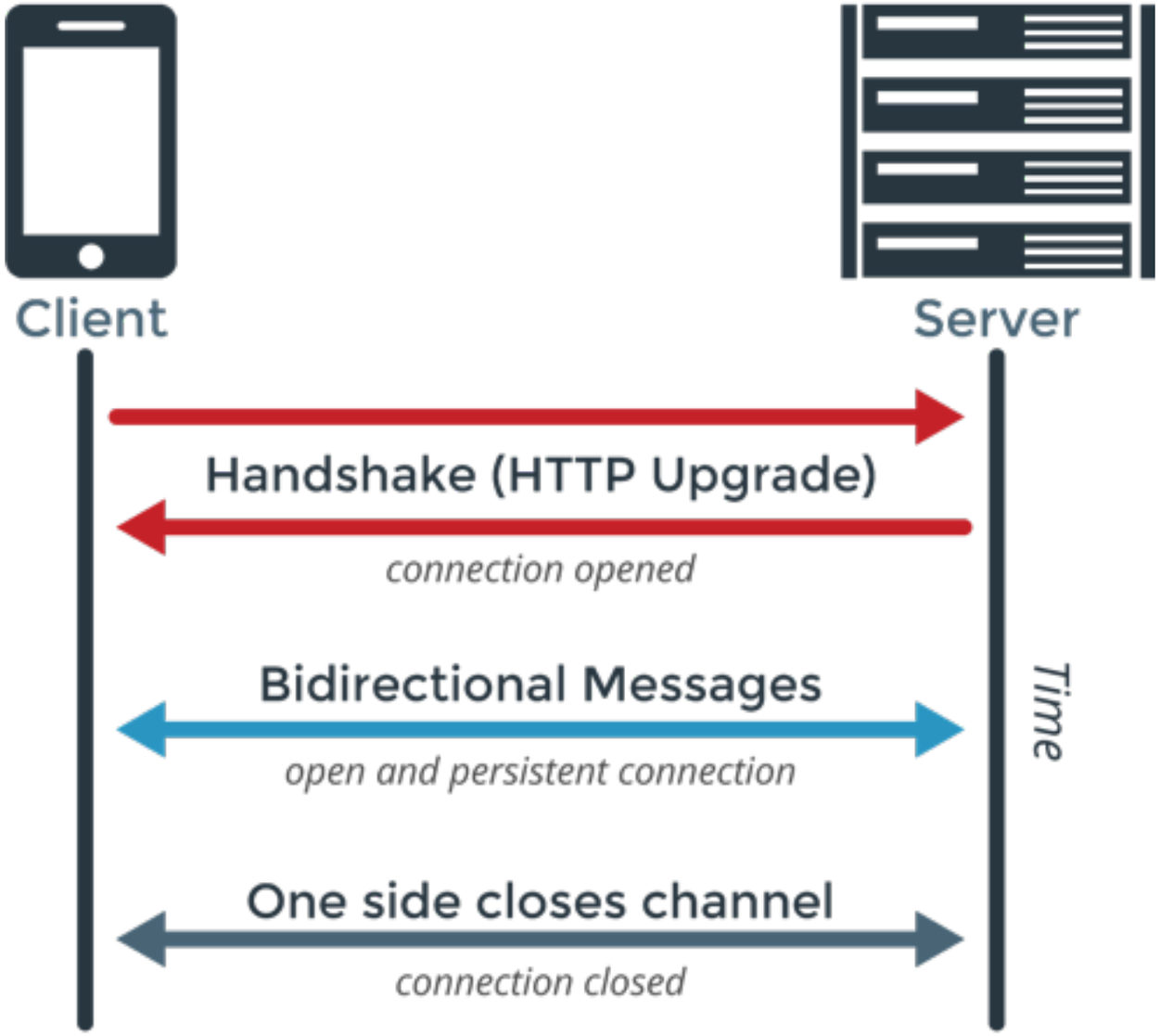
- The WebSocket specification defines an API establishing "socket" connections between a web browser and a server
- There is an persistent connection between the client and the server and both parties can start sending data at any time.

Where should I start?

```
var connection = new WebSocket('ws://localhost', ['soap', 'xmpp']);
                                                    //opens the connection
```

WEBSOCKETS

1. handshake
2. data transfer



How the communication looks like?

- Using the **send**('your message') method on the connection object

- Message from server fires **onmessage** callback function

```
connection.onmessage = function(e) {
    console.log(e.data);
};
```

Is there some issues?

- **Proxy servers** do not like “upgrade” HTTP connection
- **Cross-origin communication**
  - Make sure only to communicate with clients and servers that you trust
  - WebSocket enables communication between parties on any domain
  - It is up to the server which domains will be allowed

HTML and 2D/3D graphics?

YES! Use <**canvas**>!

- HTML element which can be used to draw graphics using scripting
- draw graphs, make photo composition or simple (and not so simple) animations

<CANVAS>

Get started!

```
<canvas id="tutorial" width="150" height="150"></canvas>
```

```
var canvas = document.getElementById('tutorial');
var ctx = canvas.getContext('2d');
```

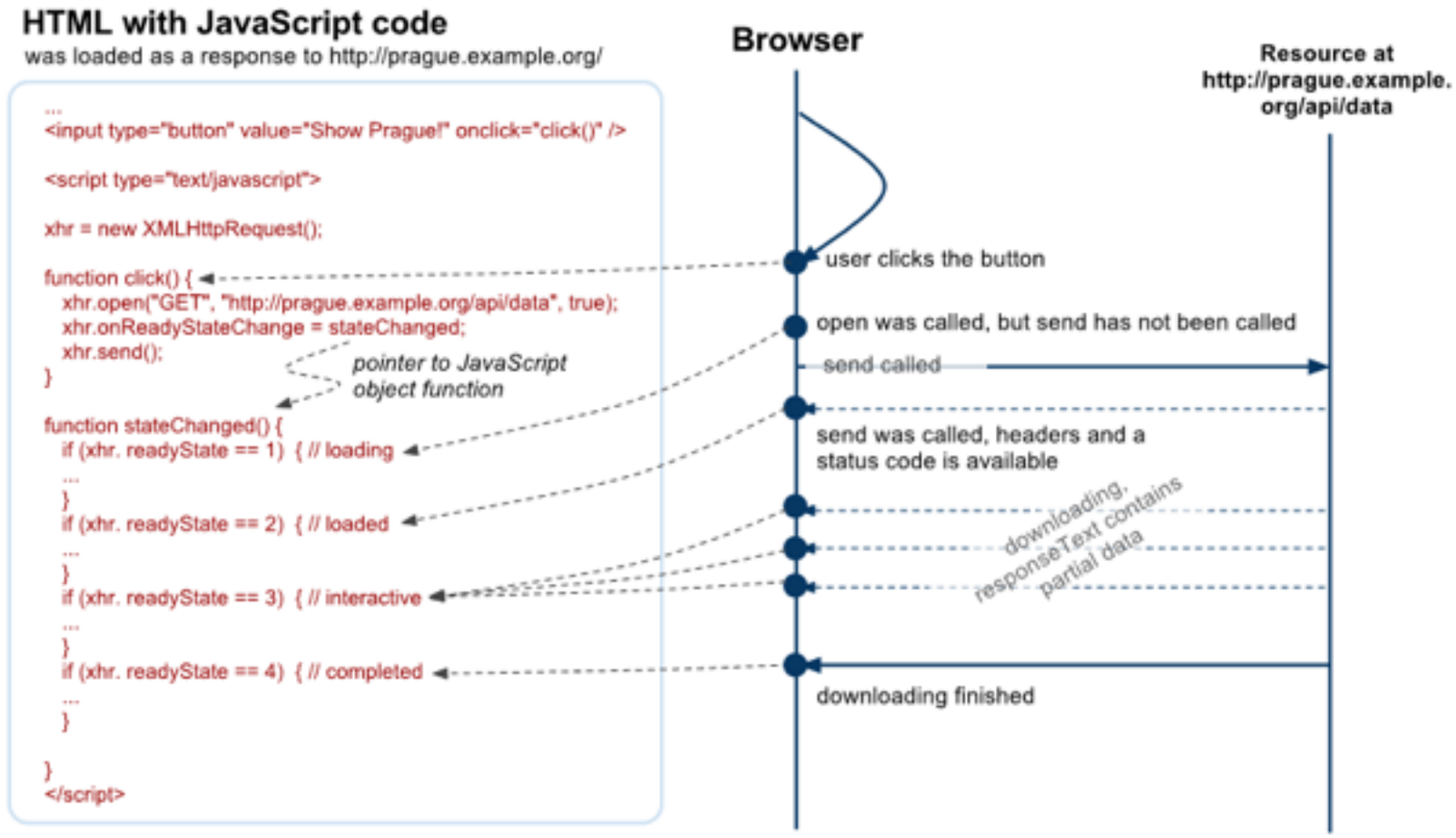
What is that **context**?

- rendering context is used to create content
- 2D to images graphs
- 3D for WebGL (based on openGL)

AJAX

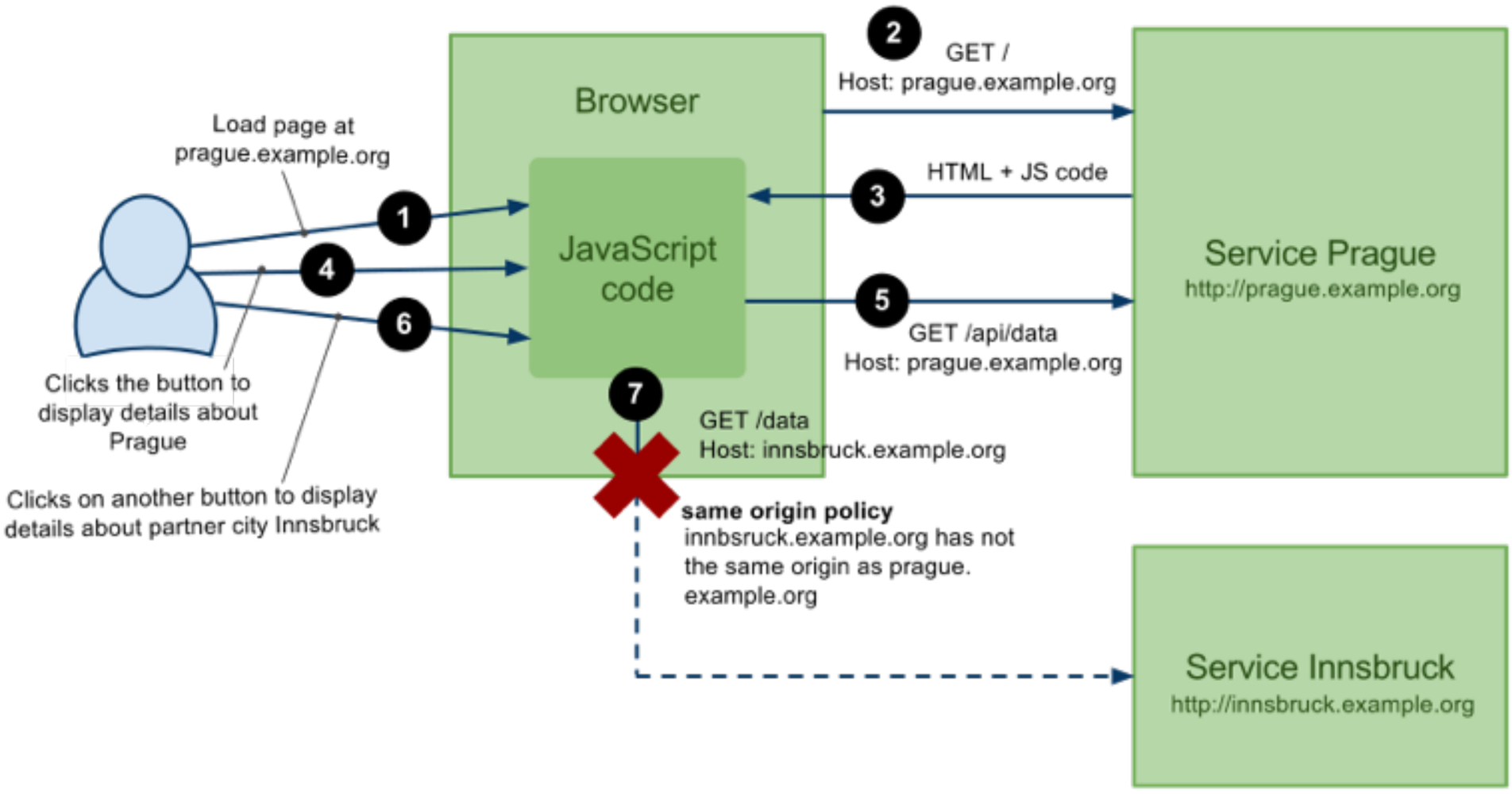
AJAX

- XMLHttpRequest
- AJAX & jQUERY
- CORS
  - client-side
  - server-side
- JSONP
- AJAX Crawling



SAME ORIGIN POLICY

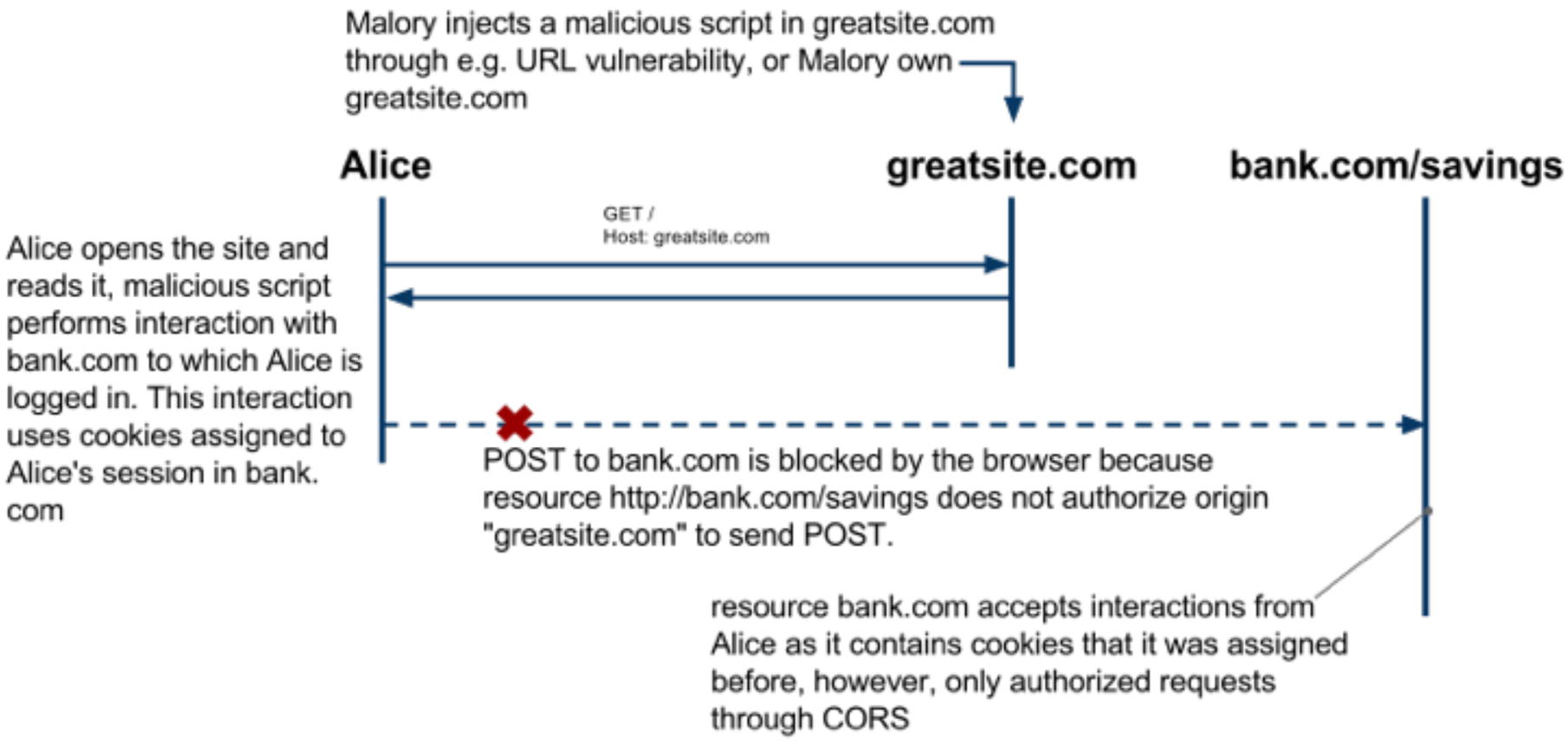
- JavaScript code can only access resources on the same domain
- Solutions: JSONP, CORS (Cross-origin Resource Sharing Protocol)



SAME ORIGIN POLICY

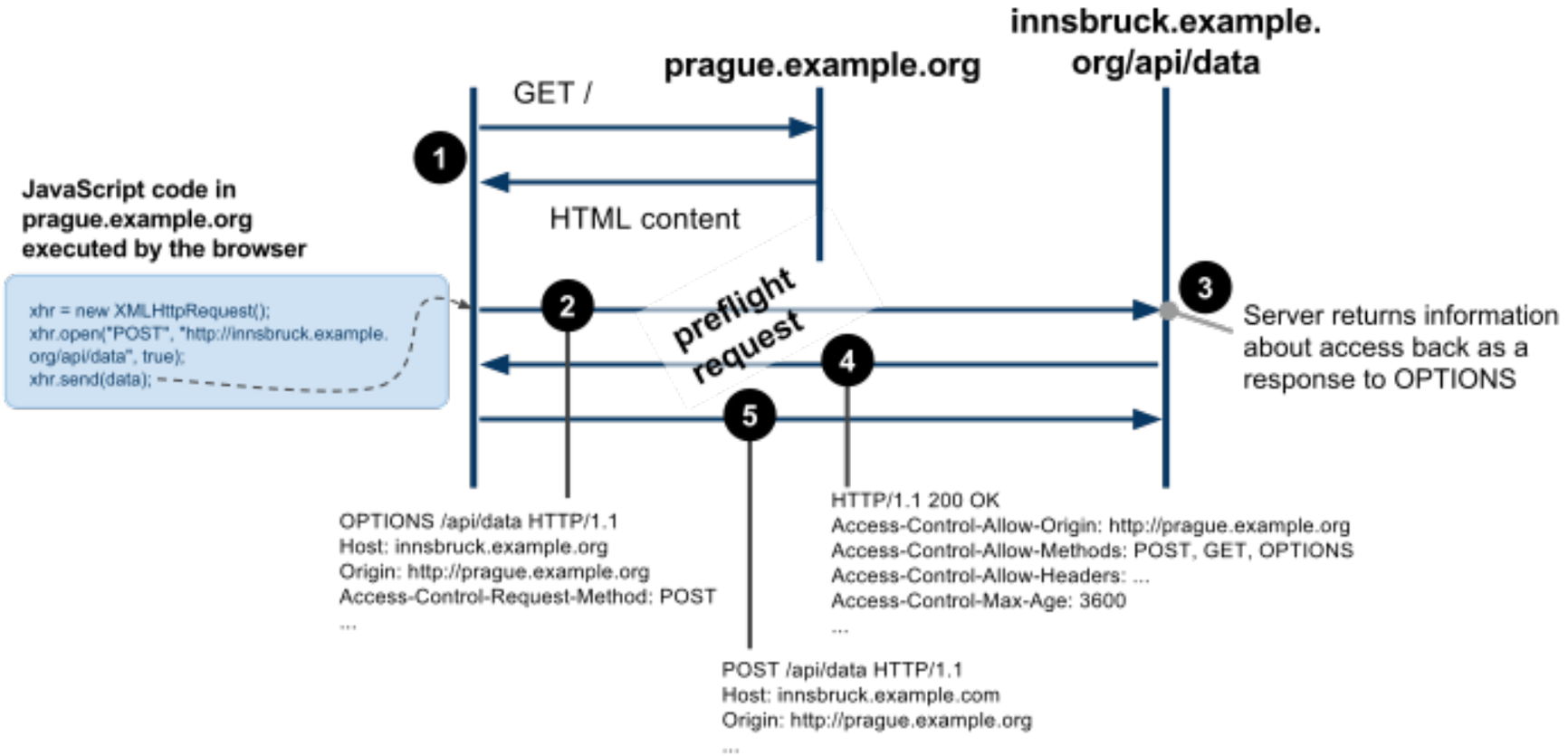
- without same origin policy is possible to do this POST

Danger !!! **Danger !!!**



CORS

- Headers:
  - Origin – identifies the origin of the request
  - Access-Control-Allow-Origin – defines who can access the resource



JAKUB BAIERL

@borecekbaji
jakub.baierl@ackee.cz