

COMP 322: Fundamentals of Parallel Programming

Lecture 10: Event-Based Programming

Mack Joyner
mjoyner@rice.edu

<http://comp322.rice.edu>



What is an Event-Based Programming?

- Event-based programming is a paradigm where actions are performed (event handlers) in response to events.
- Events are often triggered by a user (GUI, web programming)
- Events include:
 - Mouse events (clicks, mouse over)
 - Timeouts, Intervals
 - Keyboard events (key press down/up)

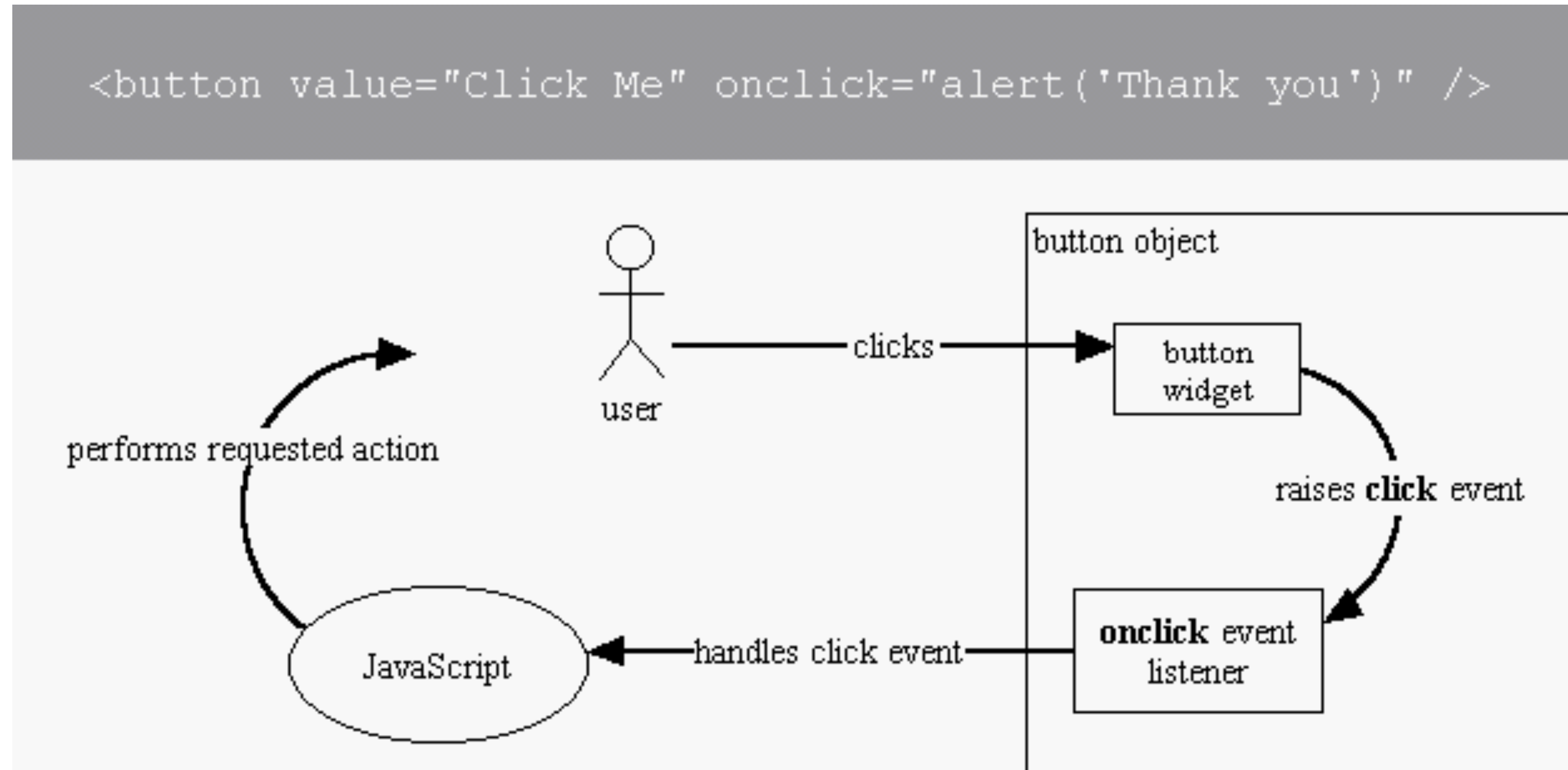
See: https://en.wikipedia.org/wiki/Event-driven_programming



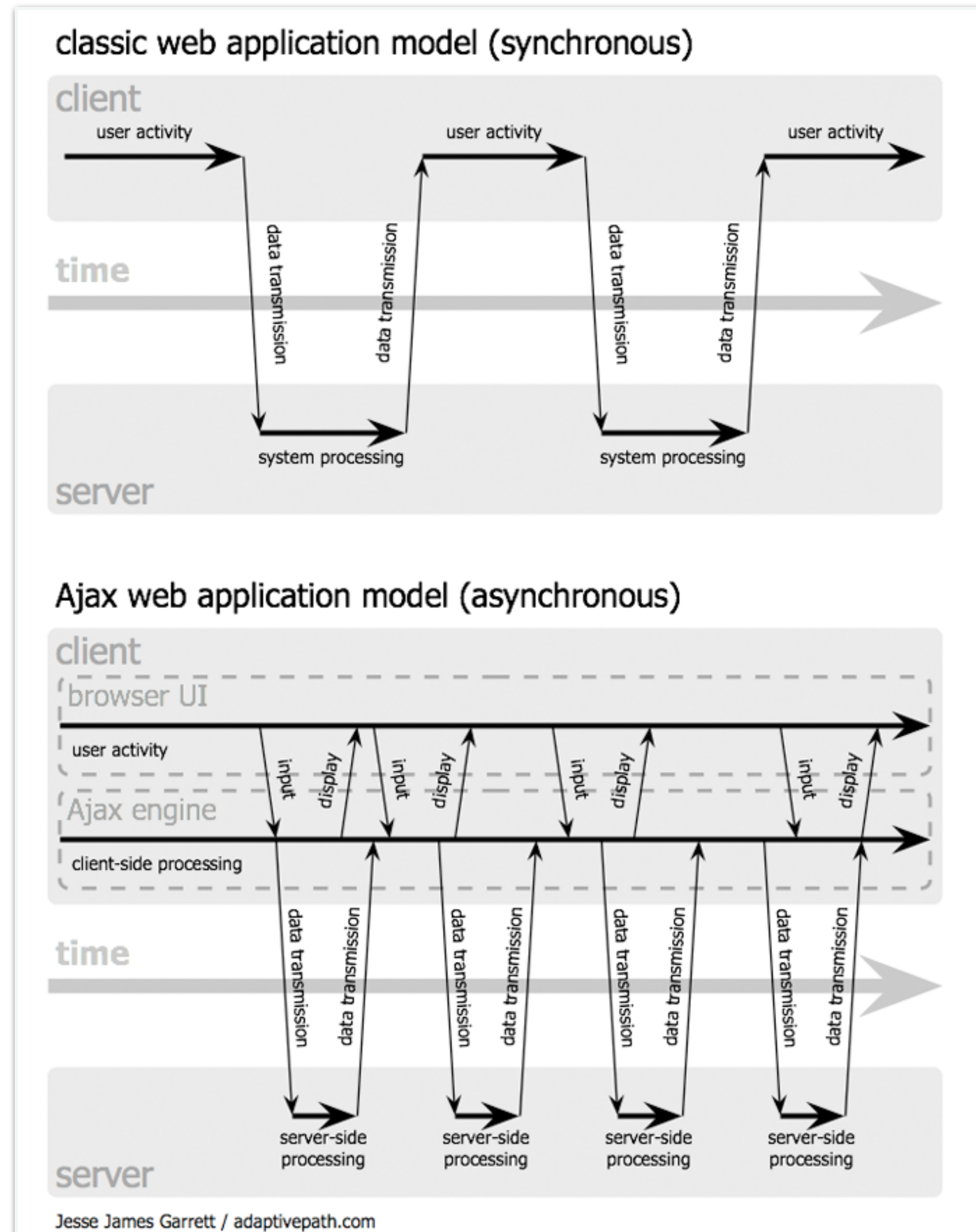
Event Handling

```
<button value="Click Me" onclick="alert('Thank you')" />
```

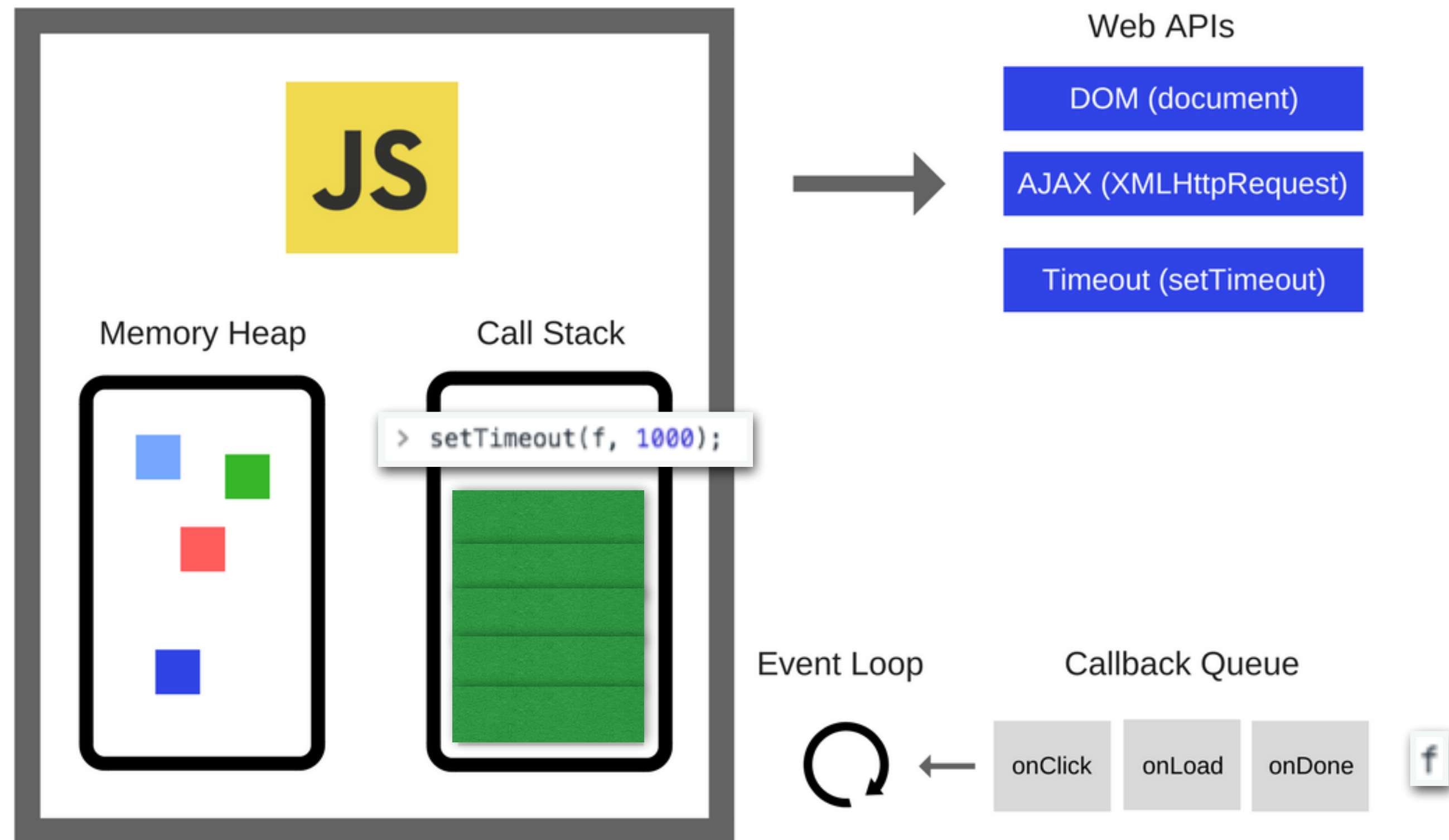
JavaScript



Asynchronous Event Handling



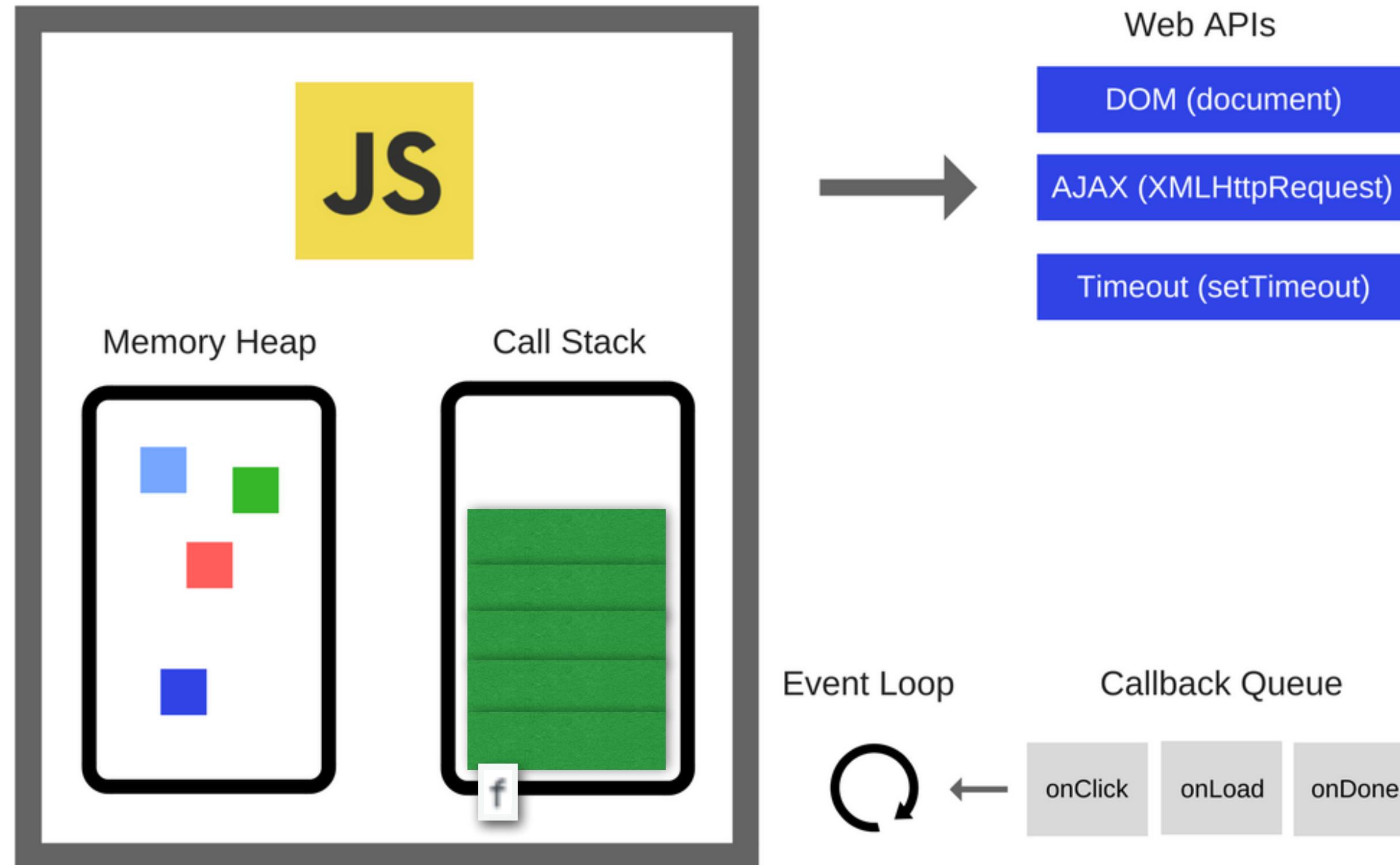
System Control over Event Response



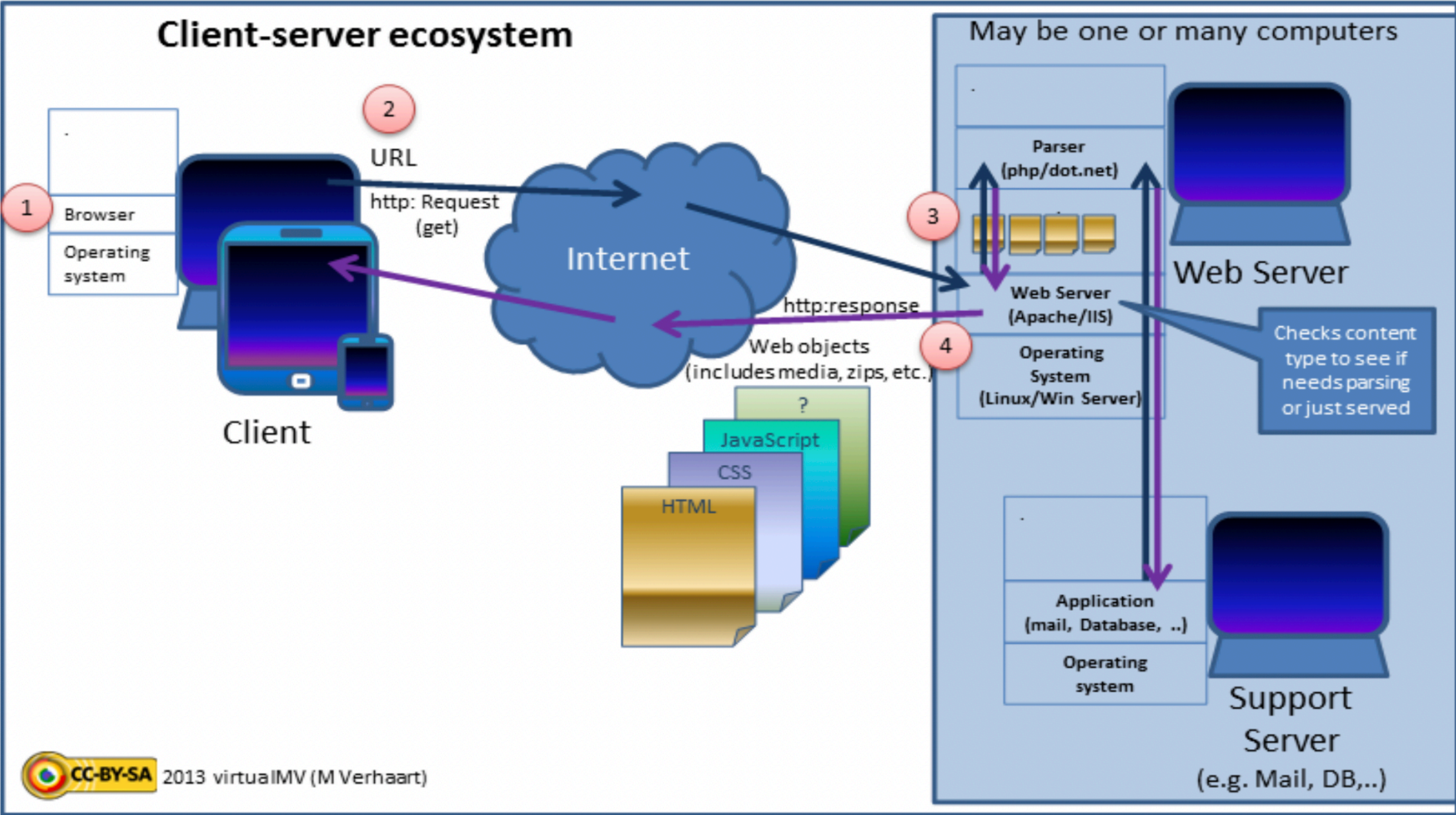
Source: <https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf>



System Control over Event Response



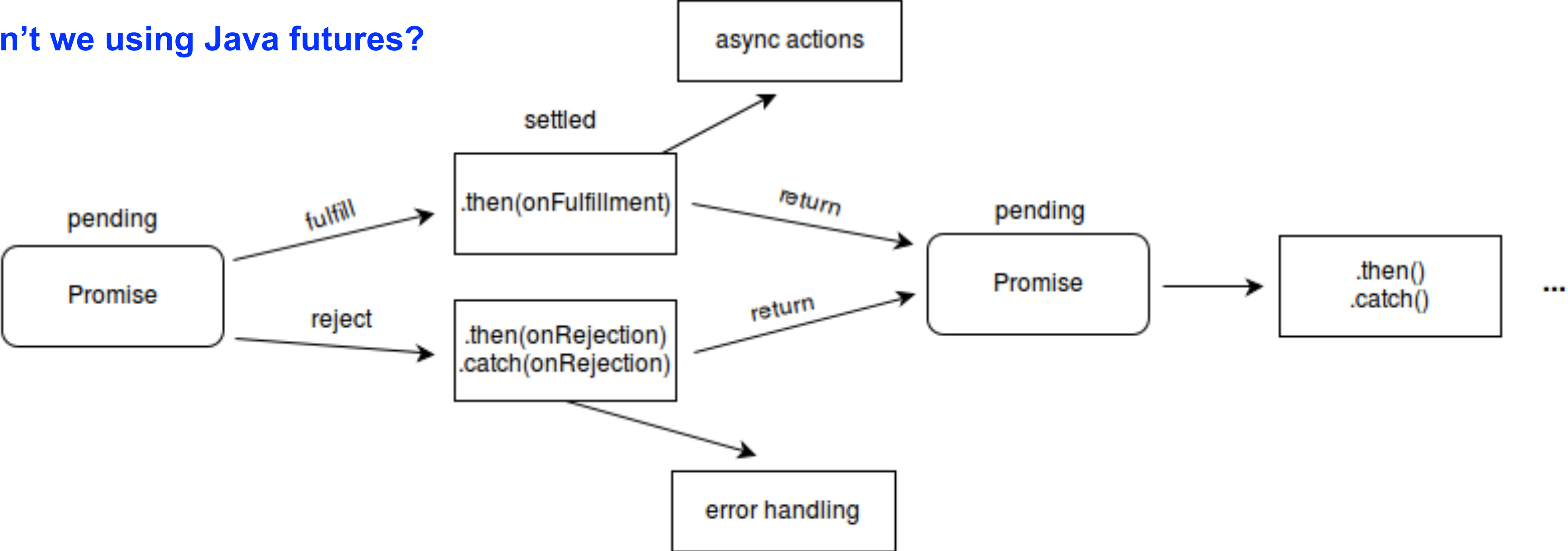
World Wide Web



JavaScript Promises

Java has both Futures (since JDK 1.5) and Promises (CompletableFuture since JDK 8)

Wait...then why aren't we using Java futures?



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
See also <http://www.html5rocks.com/en/tutorials/es6/promises/>



Promise: Fetching Web Data

Inline function callbacks
with arrow notation

```
> fetch
< f fetch() { [native code] }
> fetch('https://jsonplaceholder.typicode.com/posts').then(r => r.json())
< ▼ Promise {<pending>} ⓘ
  ▶ __proto__: Promise
    [[PromiseStatus]]: "resolved"
    ▼ [[PromiseValue]]: Array(100)
      ▶ 0: {userId: 1, id: 1, title: "sunt aut facere repellat provident occaecati
      ▶ 1: {userId: 1, id: 2, title: "qui est esse", body: "est rerum tempore vitae
      ▶ 2: {userId: 1, id: 3, title: "ea molestias quasi exercitationem repellat qu
      ▶ 3: {userId: 1, id: 4, title: "eum et est occaecati", body: "ullam et saepe
      ▶ 4: {userId: 1, id: 5, title: "nesciunt quas odio", body: "repudiandae venia
      ▶ 5: {userId: 1, id: 6, title: "dolorem eum magni eos aperiam quia", body: "u
      ▶ 6: {userId: 1, id: 7, title: "magnam facilis autem", body: "dolore placeat
      ▶ 7: {userId: 1, id: 8, title: "dolorem dolore est ipsam", body: "dignissimos
      ▶ 8: {userId: 1, id: 9, title: "nesciunt iure omnis dolorem tempora et accusa
      ▶ 9: {userId: 1, id: 10, title: "optio molestias id quia eum", body: "quo et
      ▶ 10: {userId: 2, id: 11, title: "et ea vero quia laudantium autem", body: "d
      ▶ 11: {userId: 2, id: 12, title: "in quibusdam tempore odit est dolorem", bod
      ▶ 12: {userId: 2, id: 13, title: "dolorum ut in voluptas mollitia et saepe qu
      ▶ 13: {userId: 2, id: 14, title: "voluptatem eligendi optio", body: "fuga et
      ▶ 14: {userId: 2, id: 15, title: "eveniet quod temporibus", body: "reprehende
      ▶ 15: {userId: 2, id: 16, title: "sint suscipit perspiciatis velit dolorum re
      ▶ 16: {userId: 2, id: 17, title: "fugit voluptas sed molestias voluptatem pro
      ▶ 17: {userId: 2, id: 18, title: "voluptate et itaque vero tempora molestiae"
      ▶ 18: {userId: 2, id: 19, title: "adipisci placeat illum aut reiciendis qui",
      ▶ 19: {userId: 2, id: 20, title: "doloribus ad provident suscipit at", body:
      ▶ 20: {userId: 3, id: 21, title: "asperiores ea ipsam voluptatibus modi minim
```

Same as:
.then(function(r) {
 return r.json()
})
r.json() returns a Promise, the next
then() is called when json() resolves.



JavaScript Async/Await (like HJ Futures/Data Driven Tasks)

- **Async** functions always return Promise
- **Await** can only be inside async
- Expression after **await** is like using Promise **then**

```
(async () => {  
  const connector = mongoose.connect(connectionString)  
  const username = process.argv[2].split('=')[1]  
  
  let user = await connector.then(async () => {  
    return findUser(username)  
  })  
  
  if (!user) {  
    user = await createUser(username)  
  }  
})
```

```
console.log(user)  
process.exit(0)  
})()
```

Need response before sending result



Login/Logout registered users with Futures

```
var username = ...
var password = ...
...
var regUser = future(() -> registerNewUser(username, password)); // { username: user, result: "success" or "failure"}
...
var logUser = future(() -> loginUser(username, password)); // {userId: id, result: "success" or "failure" }
...
var loggedIn = future(() -> isLoggedIn(id)); // { userId: id, result: "success" or "failure" }
...
var logOut = future(() -> logoutUser(id)); // { result: "success" or "failure" }
...
```

What future dependencies are missing?



Login/Logout registered users with Futures

```
var username = ...
var password = ...
...
var regUser = future(() -> registerNewUser(username, password)); // { username: user, result: "success" or "failure" }
...
var logUser = future(() -> { if (regUser.get().result.equals("success"))
    return loginUser(username, password); // {userId: id, result: "success" or "failure" }
    return {result: "failure" };
});
...
var loggedIn = future(() -> { if (logUser.get().result.equals("success"))
    return isLoggedIn(logUser.get().userId); // {userId: id, result: "success" or "failure" }
    return {result: "failure" };
});
...
var logOut = future(() -> { if (loggedIn.get().result.equals("success"))
    return logoutUser(loggedIn.get().userId); // { result: "success" or "failure" }
    return {result: "failure" };
});
...
```



Login/Logout registered users with DDTs

```
var username = ...
var password = ...
...
var regUser = newDataDrivenFuture();
var logUser = newDataDrivenFuture();
var loggedIn = newDataDrivenFuture();
var logOut = newDataDrivenFuture();
...
async(() -> regUser.put(registerNewUser(username, password))); // { username: user, result: "success" or "failure"}
...
asyncAwait(regUser, () -> { if (regUser.safeGet().result.equals("success"))
    logUser.put(loginUser(username, password)); // {userId: id, result: "success" or "failure"}
    else
    logUser.put({result: "failure" }); });
...
asyncAwait(logUser, () -> { if (logUser.safeGet().result.equals("success"))
    loggedIn.put(isLoggedIn(logUser.safeGet().userId)); // {userId: id, result: "success" or "failure" }
    else
    loggedIn.put({result: "failure" }); });
...
asyncAwait(loggedIn, () -> { if (loggedIn.safeGet().result.equals("success"))
    logOut.put(logoutUser(loggedIn.safeGet().userId)); // { result: "success" or "failure" }
    else
    logOut.put({result: "failure" }); });
...
...

```

