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?

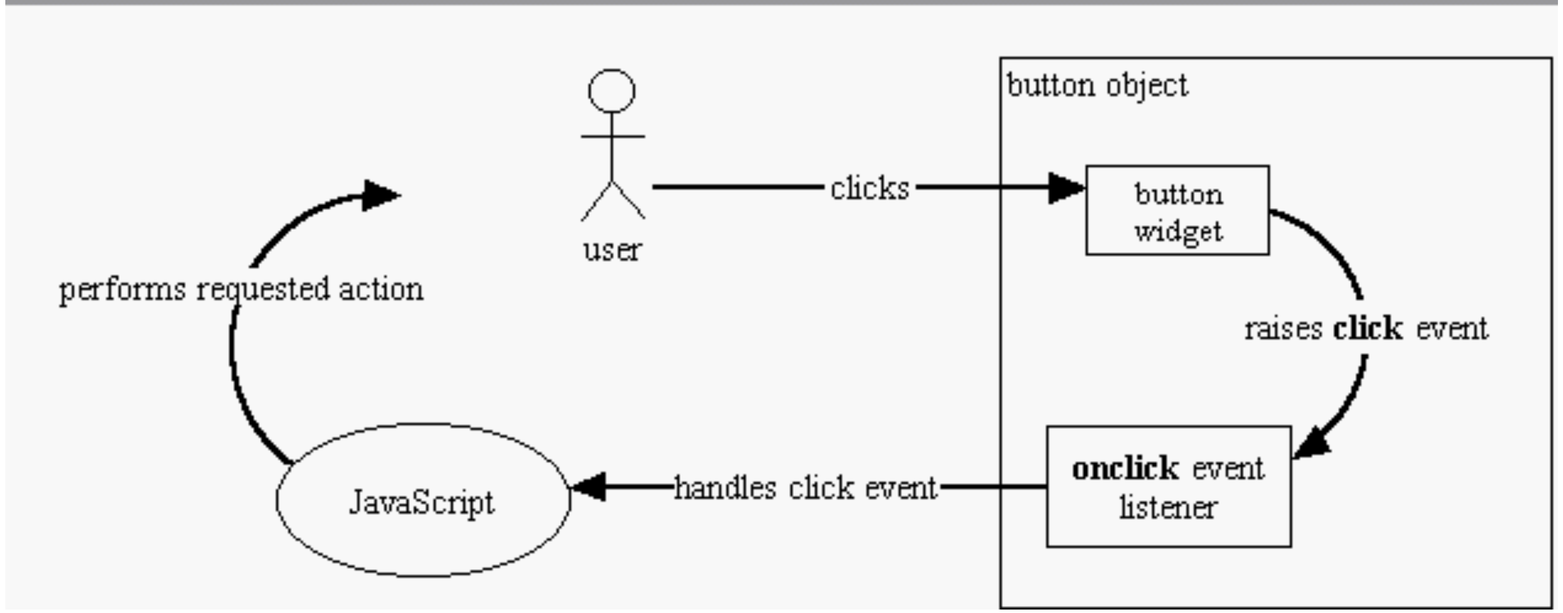
- Event-based programming is a paradigm where actions are performed (event handlers) in response to events.
- Events are often triggered by a user (GUI, web programming)
- Events include:
  - Mouse events (clicks, mouse over)
  - Timeouts, Intervals
  - Keyboard events (key press down/up)

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

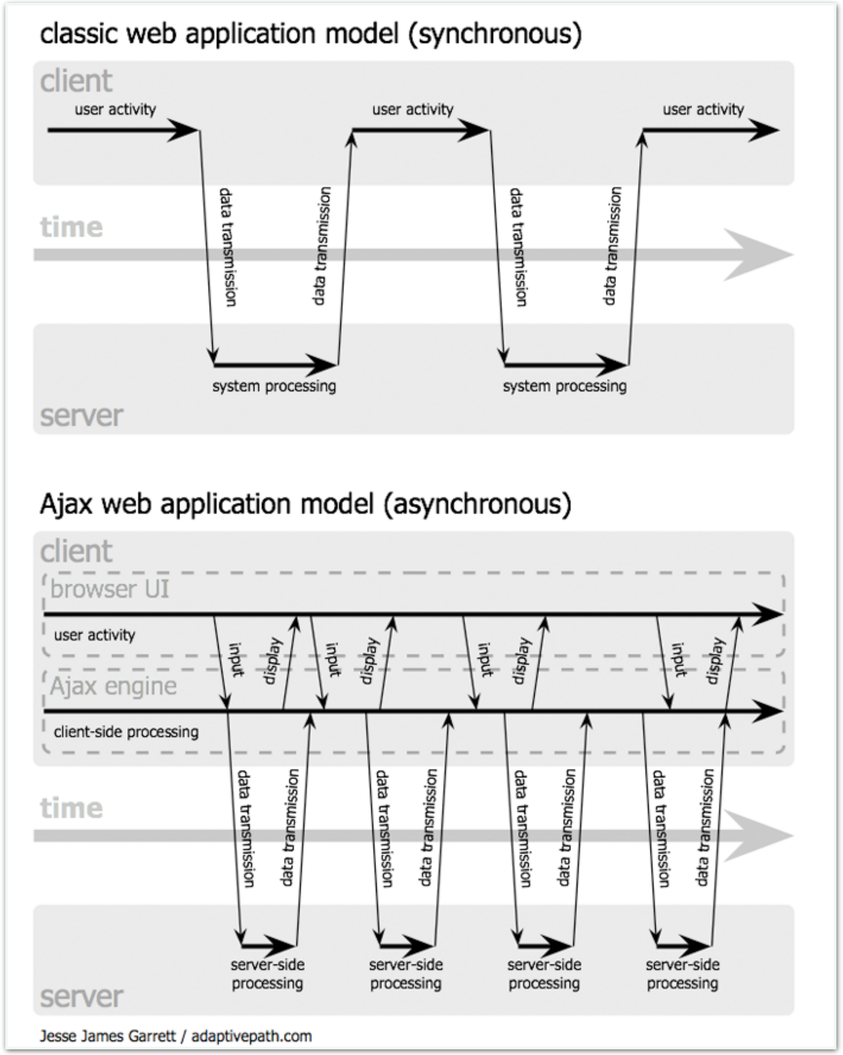
Event Handling

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

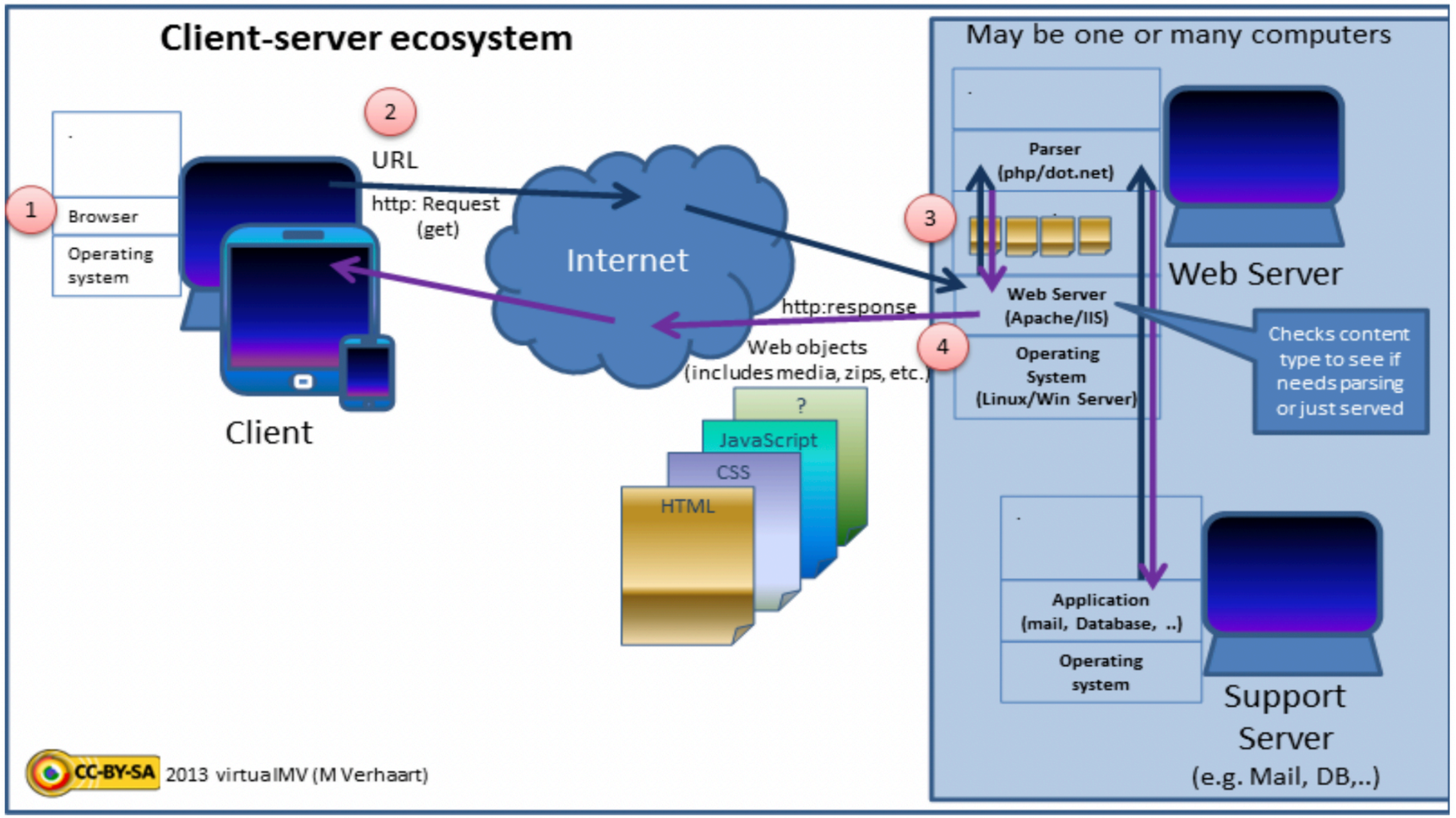
JavaScript



Asynchronous Event Handling



World Wide Web



Promise: Fetching Web Data

Inline function callbacks with arrow notation

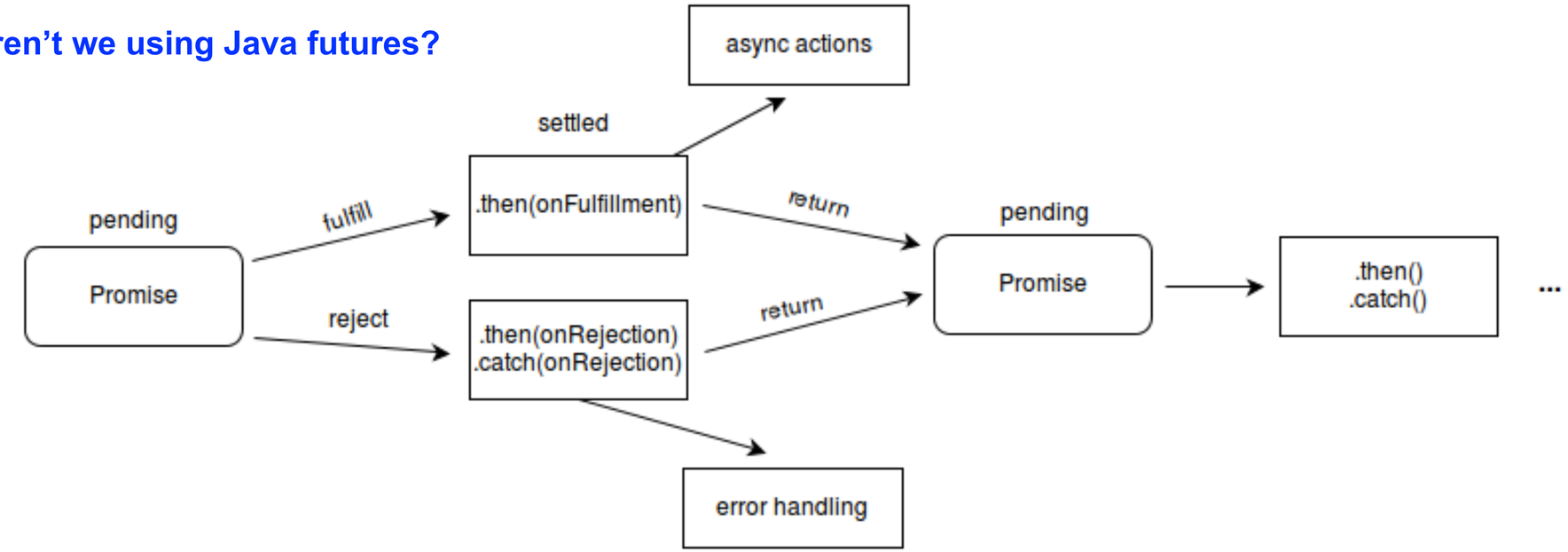
Same as:
```
.then(function(r) {
    return r.json()
})
```
`r.json()` returns a *Promise*, the next `then()` is called when `json()` resolves.

```
> fetch
ƒ 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
```


JavaScript Promises

Java has both Futures (since JDK 1.5) and Promises (CompleteableFuture 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/

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

User Registration with Futures

User Registration

- Register New User: registerNewUser(username, password)
- Login User: loginUser(username)
- Logout User: logoutUser(username)

Login/Logout registered users with Futures

```
var username = …
var password = …
…
var regUser = future(() -> registerNewUser(username, password)); // returns { user: username, result: “success” or “failure”}
…
var logUser = future(() -> loginUser(username, password)); // returns {user: username, result: “success” or “failure” }
…
var logOut = future(() -> logoutUser(username)); // returns { user: username, 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)); // returns { user: username, result: “success” or “failure”}
…

var logUser = future(() -> { if (regUser.get().result.equals(“success”))
                                    return loginUser(username, password); // returns {user: username, result: “success” or “failure”}
                                 return {user: username, result: “failure” };
                               });
…

var logOut = future(() -> { if (logUser.get().result.equals(“success”))
                                    return logoutUser(logUser.get().user)); // returns { user: username, result: “success” or “failure” }
                                 return {user: logUser.get().user, result: “failure” };
…
```

Are there any issues here? Should we use DDF/DDTs?

Login/Logout registered users with DDTs

```
var username = …
var password = …
…
var regUser = newDataDrivenFuture();
var logUser = newDataDrivenFuture();
var logOut = newDataDrivenFuture();
…
async(() -> regUser.put(registerNewUser(username, password))); // returns { user: username, result: “success” or “failure”}
…

asyncAwait(regUser, () -> { if (regUser.safeGet().result.equals(“success”))
                                logUser.put(loginUser(username, password)); // returns {user: username, result: “success” or “failure”}
                             else
                                logUser.put({user: username, result: “failure” });      });
                           …

asyncAwait(logUser, () -> { if (logUser.safeGet().result.equals(“success”))
                               logOut.put(logoutUser(logUser.safeGet().user)); // returns { user: username, result: “success” or “failure” }
                             else
                               logOut.put({user: logUser.safeGet().user, result: “failure” });     });
…
```