THE NODE.JS HIGHWAY:

ATTACKS AT FULL THROTTLE

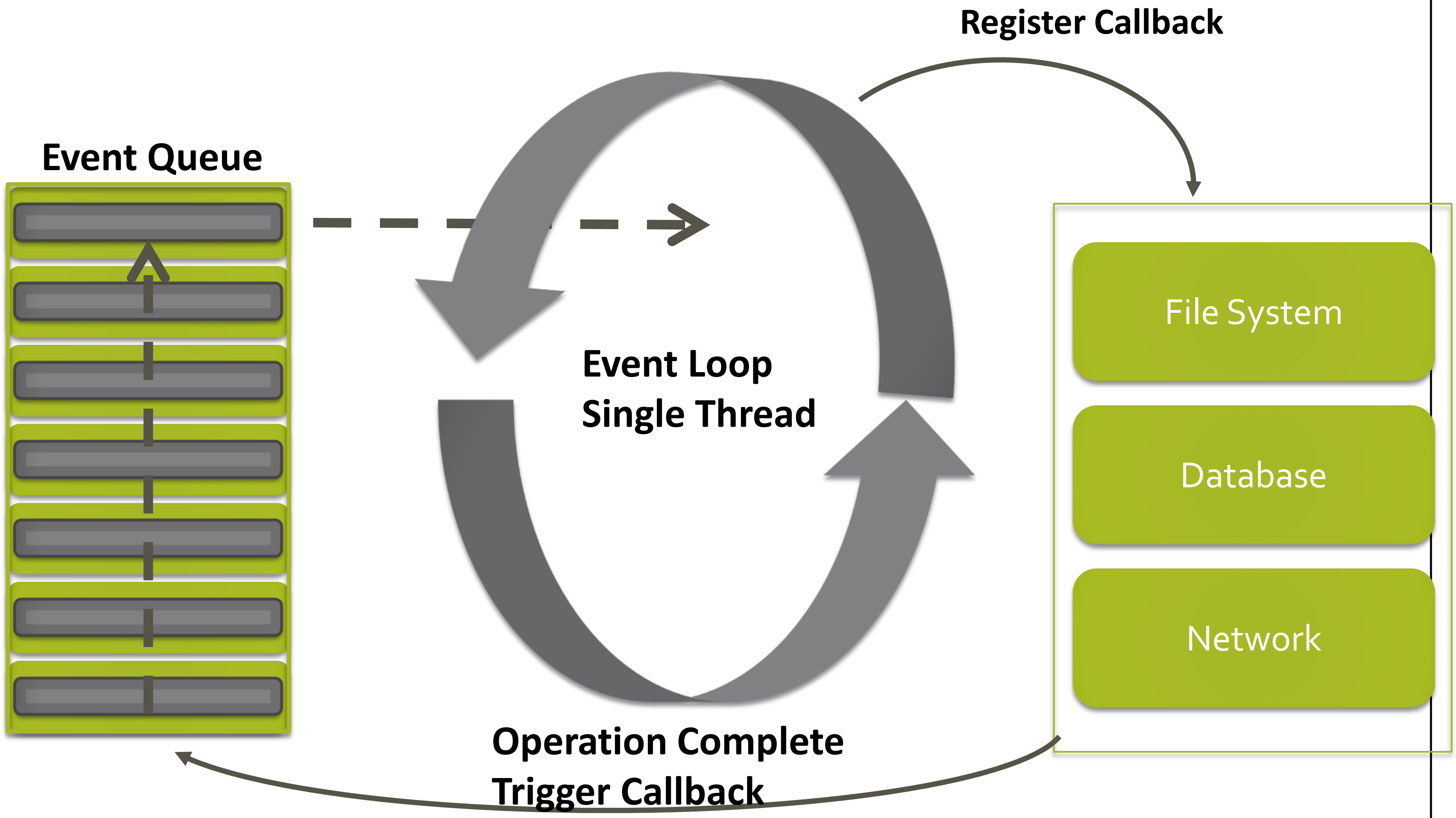
Susan St.Clair,
Solutions Architect

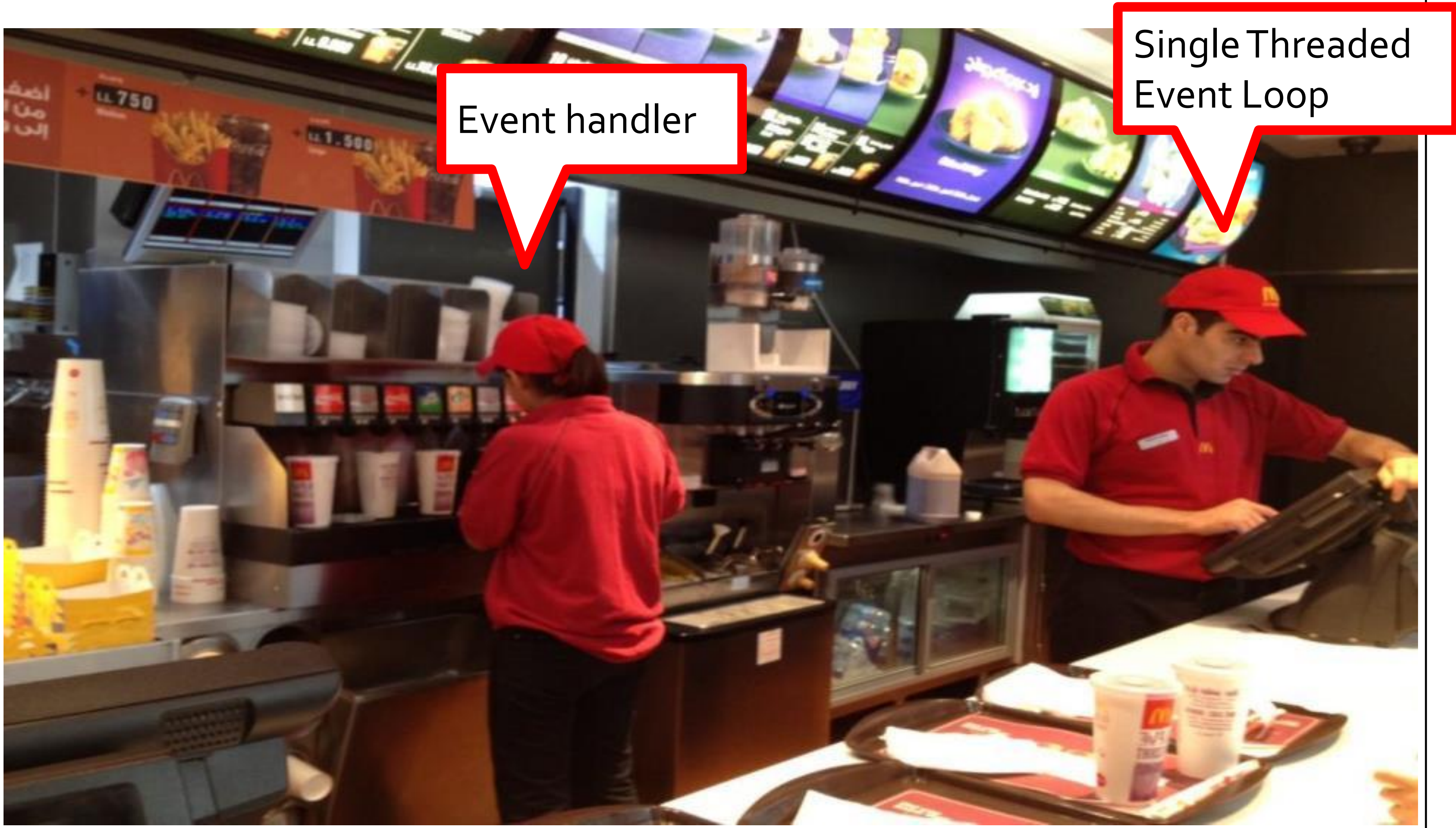
Checkmarx

Agenda

- Architecture
- DoS
- Weak Crypto
- JSON “SQLi”
- Re-DoS
- App Re-Routing

Single Thread Architecture - Event Loop




Event handler
Single Threaded Event Loop

What’s it Good for?

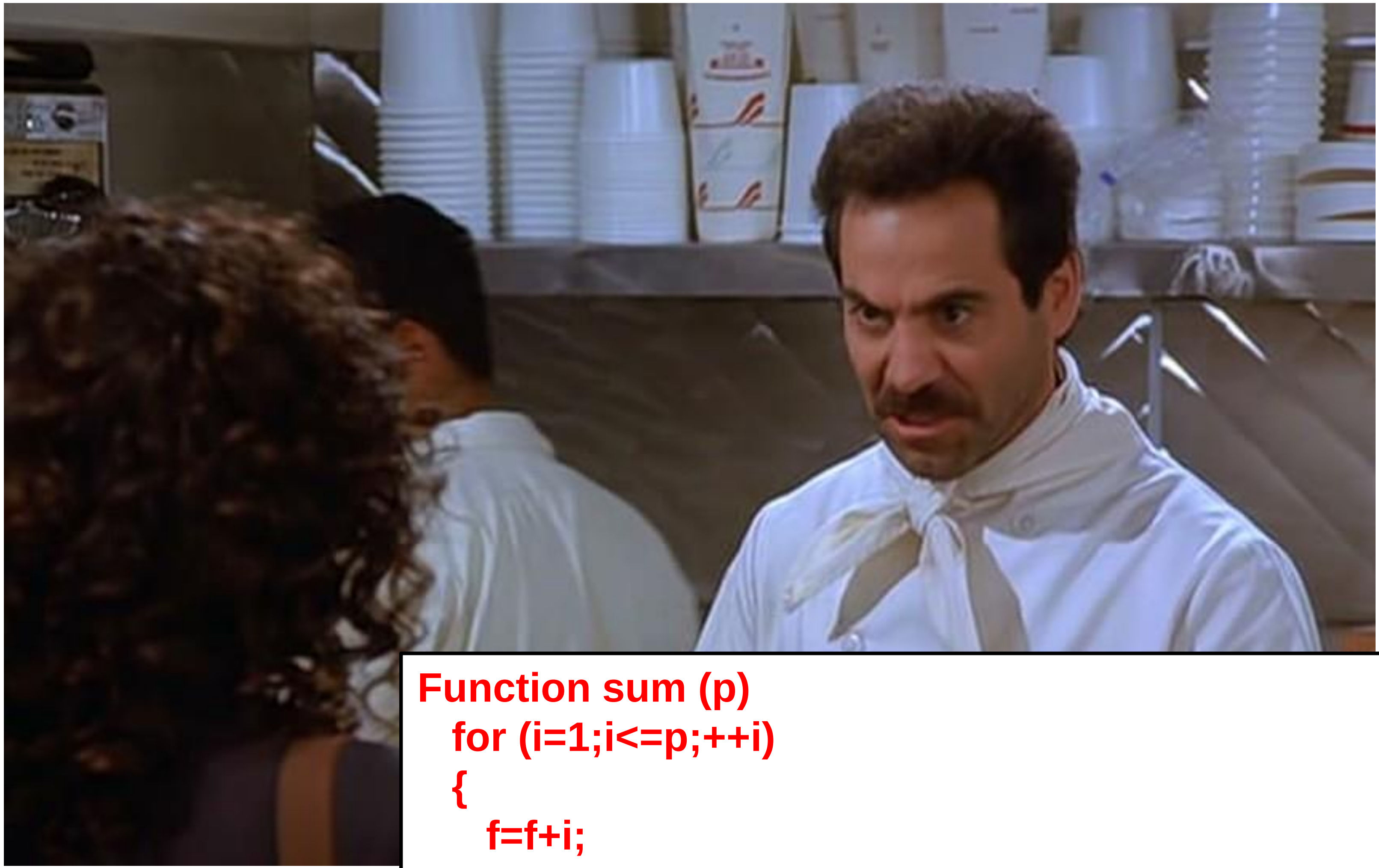
👍

- I/O intensive applications
- DB queries
- UI intensive applications (many webapps)

👎

- CPU intensive applications
- Complex business logic that requires lots of calculations

Denial of Service (DoS)



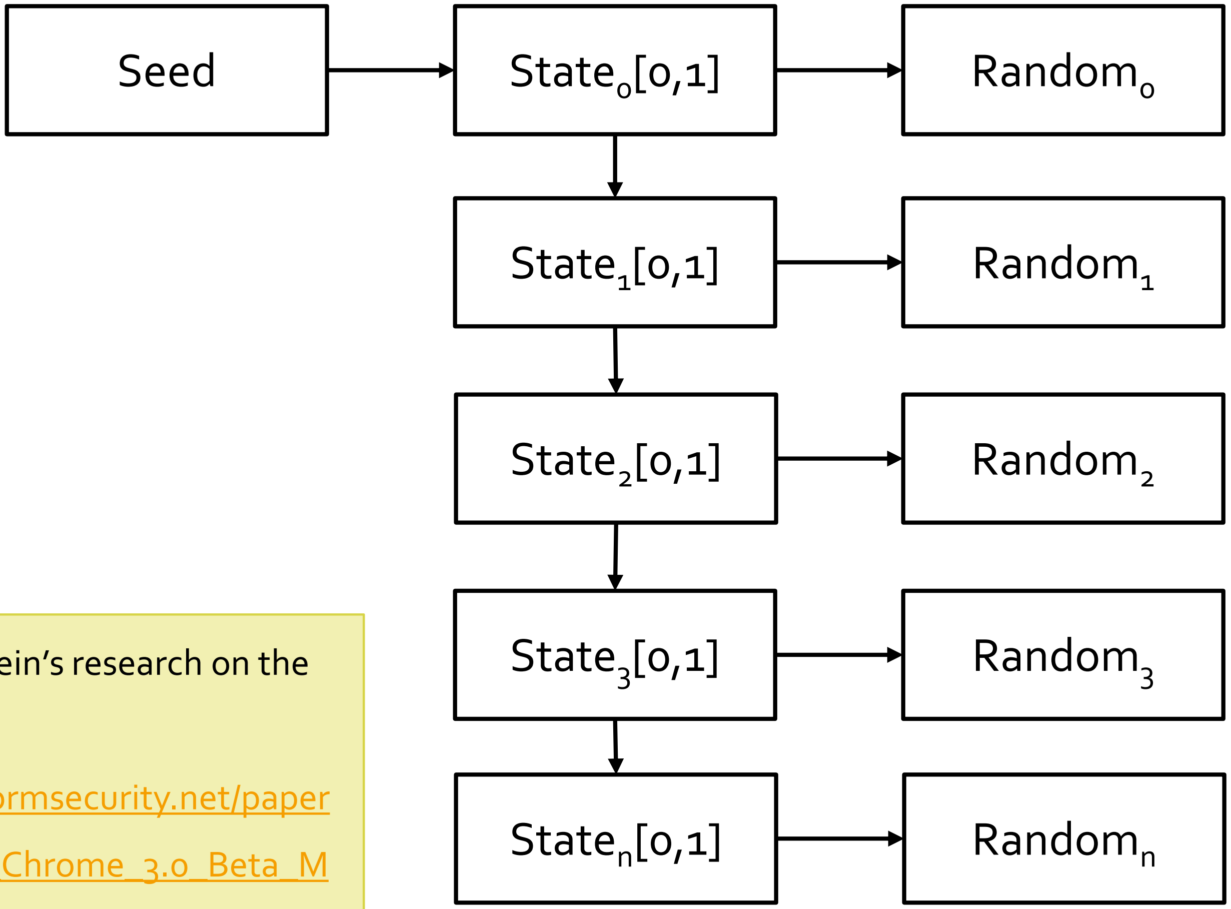
```
Function sum (p)
  for (i=1;i<=p;++i)
  {
    f=f+i;
  }
```

DoS DEMO

Weak Crypto

```
function [redacted]NewUser([redacted], userName) {
   var newUser = new User();
   [redacted]
   newUser.password = (Math.random()).toString().md5(); // generate random password
   [redacted]
   return newUser;
}
```

V8 PRNG is **known** to be weak



Check out Amit Klein's research on the subject http://dl.packetstormsecurity.net/papers/general/Google_Chrome_3.0_Beta_Math.random_vulnerability.pdf

V8's Default PRNG

```
function [redacted]NewUser([redacted], userName) {
   var newUser = new User();
   [redacted]
   newUser.password = (Math.random()).toString().md5(); // generate random password
   [redacted]
   return newUser;
}
```

Given 3 "random" new passwords – we will be able to tell all future ones

- First, we need to "reverse" the MD5 for 3 passwords to their original "random" float number
- Then, we need to compute the "state" variable to get the 4th consecutive value.

V8’s Default PRNG – So What?!

- Given 3 consecutive random numbers, the values of state[0] and state[1] can be inferred – hence all future values can be known in advance.

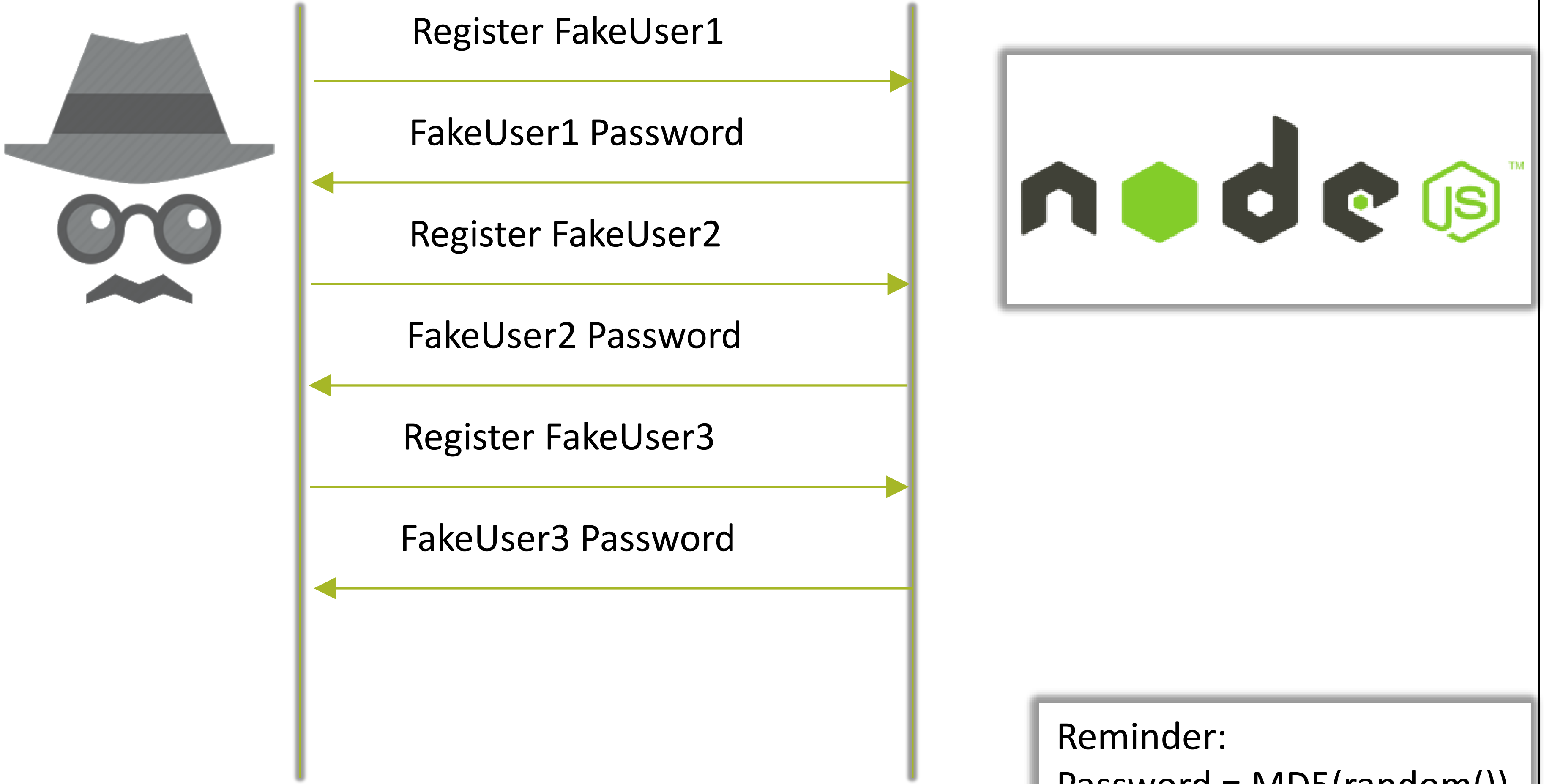
But

- In browsers, each tab has its own set of “state” variables. That’s one of the reasons this issue is treated as low-severity

But

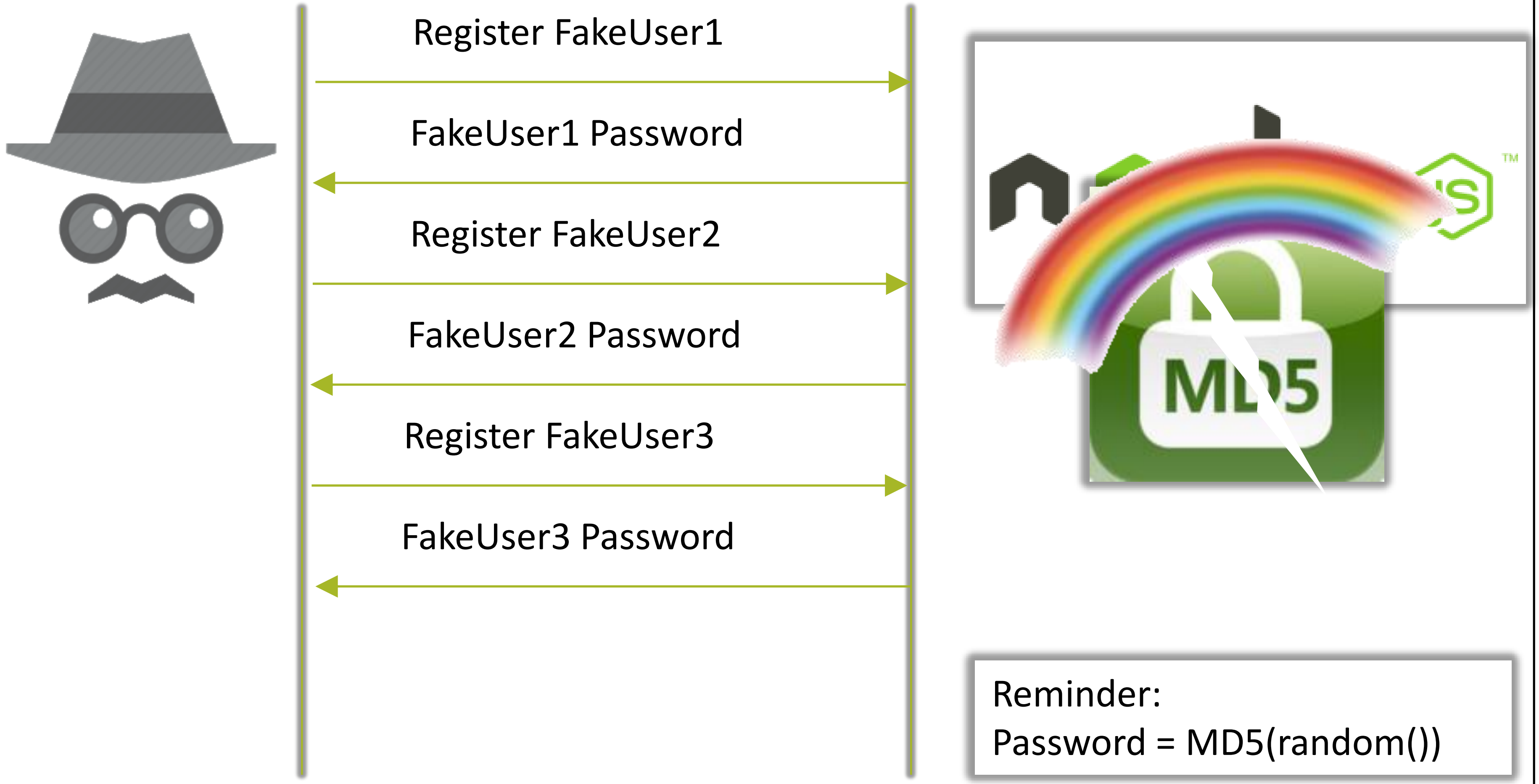
- In node.js, all users are running within the same context. Each user can tell what are the values of the global “state” variables.

Step 1



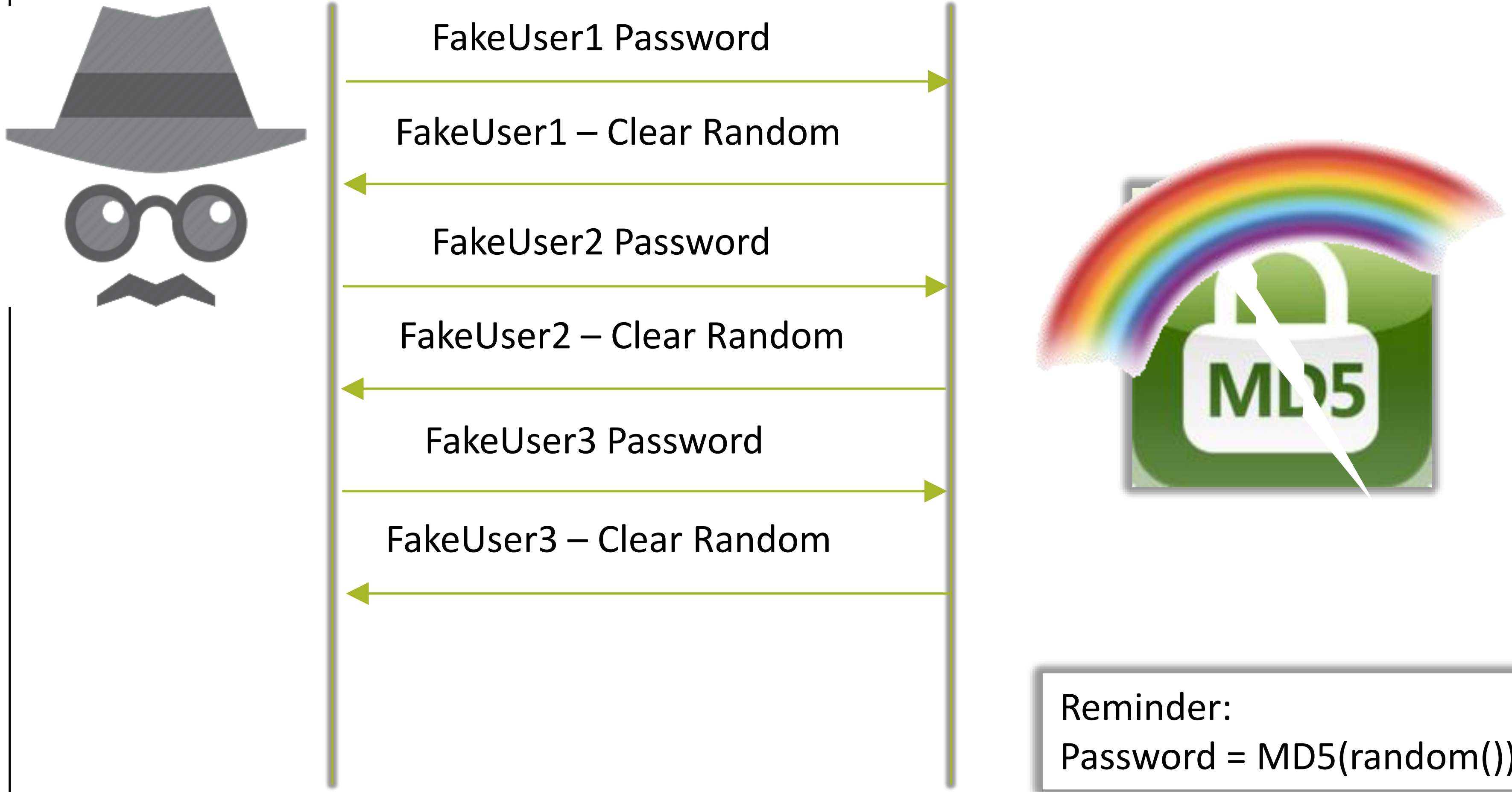
Reminder:
Password = MD5(random())

Step2



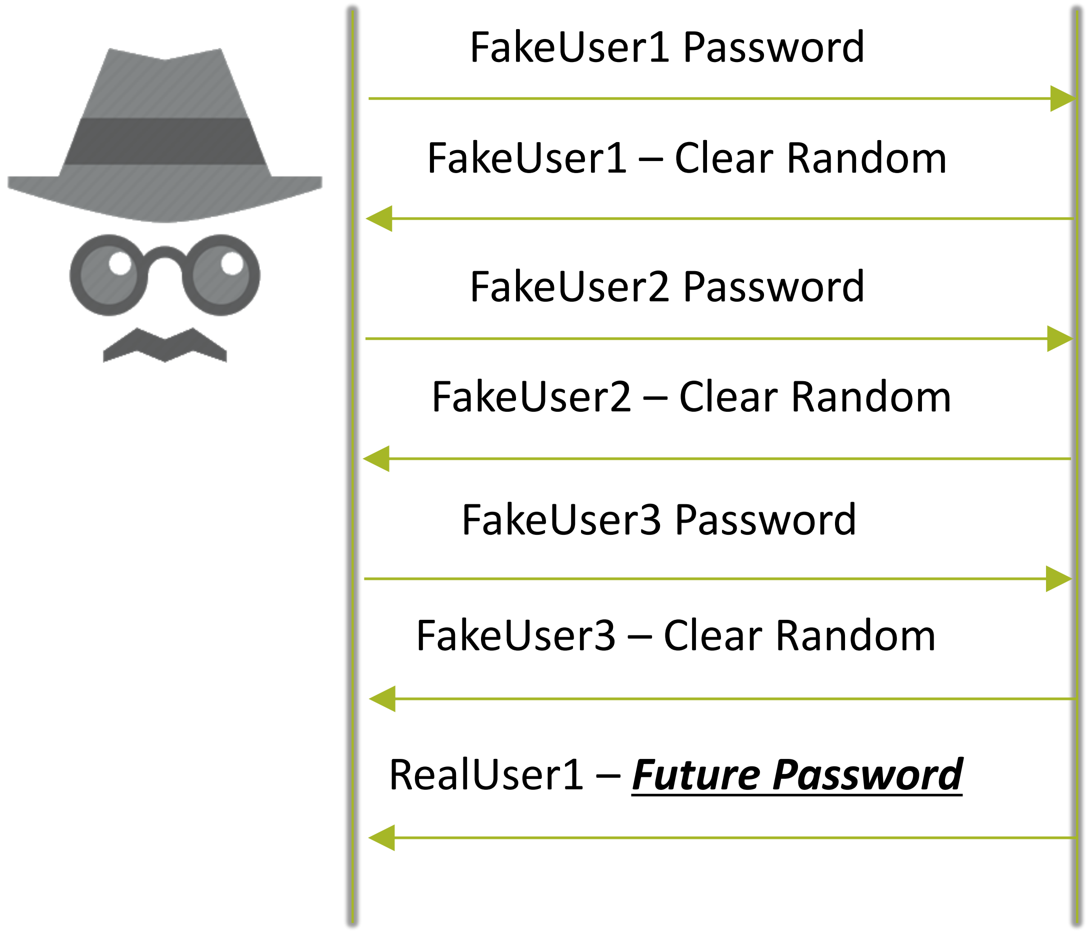
Reminder:
Password = MD5(random())

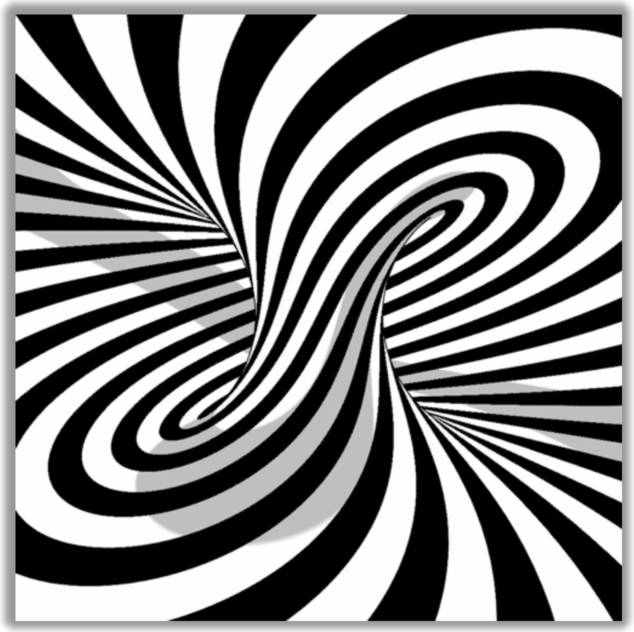
Step3



Reminder:
Password = MD5(random())

Step 4





Reminder:
Password = MD5(random())

PASSWORD GUESSING DEMO

Architecture

- MongoDB
  - Document-oriented database.
  - Classified as NoSQL
  - Doesn’t use the traditional table-based structure
  - Stores JSON documents in its dynamic schemas.

Mongo Queries

Data is inserted and stored as JSON

db.products.insert

```
db.products.insert( { item: "card", qty : 15 } )
db.products.insert( { name: “elephant", size: 1700 } )
```

db.products.find

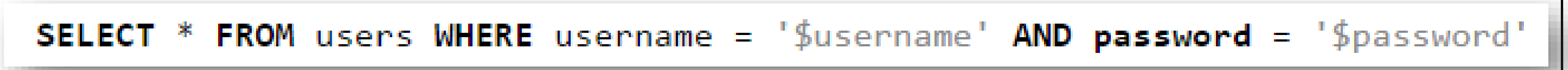
```
db.products.find()                         - Find all of them
db.products.find( { qty: 15 })             - Find based on equality
db.products.find( { qty: { $gt: 25 } } ) - Find based on criteria
```

Queries as described using JSON

```
var obj;
obj.qty=15;
db.products.find(obj)
```

Security – User Supplied Data

- Can you spot the vulnerabilities in the code?
- Traditional SQL:



```
SELECT * FROM users WHERE username = '$username' AND password = '$password'
```

- JSON:

```
name = req.query.username;
pass = req.query.password;
db.users.find({username: name, password: pass});
…
If exists ….
```

Security – User Supplied Data

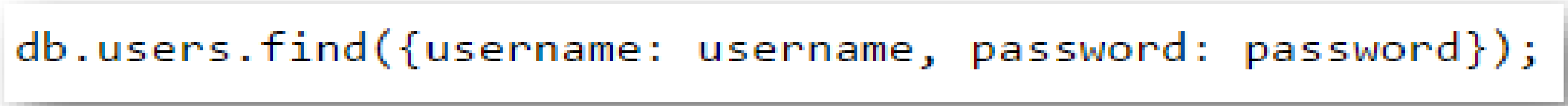
```
name = req.query.username;
pass = req.query.password;
db.users.find({username: name, password: pass});
```

What if we use the following query:

```
db.users.find({username: {$gt, “a”},
                    password : {$gt, “a”}}
```

JSON-based SQL Injection

- Node.JS, being a JSON based language, can accept JSON values for the .find method:



```
db.users.find({username: username, password: password});
```

- A user can bypass it by sending

```
http:///server/page?user[$gt]=a&pass[$gt]=a
```

http://blog.websecurify.com/2014/08/hacking-nodejs-and-mongodb.html

PASSWORD BYPASS DEMO

JSON-base SQL Injection Defense

You can use the following:

```
db.users.find({username: username});
```

Then

```
bcrypt.compare(candidatePassword, password, cb);
```

JSON-based SQL Injection

```
db.users.find({username: username});
```

This can lead to Regular Expression Denial of Service through the {"username": {"$regex": "........}}

JSON-based NoSQL Injection

- So always validate the input length, structure and permitted characters

- Remembering that Node.js is highly sensitive to CPU-intensive tasks, and there’s a single thread for user-code – ReDoS is really bad

NodeJS as a Webserver

- Recap
  - With Node.js there is no web server
  - Traditional web-servers (IIS, Tomcat) have strict separation between the application, the server, and the OS
- Run-time Server Poisoning
  - Node.js server runs in a single thread; if corrupted, server behavior can be altered
  - Alterations will last for all subsequent requests.

‘Evil EVAL’

- EVALuates a string.
  - At the context of the current applicative user within the context of the application.
  - In .net/java, eval can’t control the web server or other users’ threads

- Node.js is server-less so corrupting “current” thread, harms all users

Express

Express.js (Wikipedia) :

"a Node.js web application framework, designed for building single-page, multi-page, and hybrid web applications."

```
app.get('/add', function(req,res) {
        var data=req.query;
        return res.render('index',
                {message: eval(req.query.a + '+' + req.query.b)});
}
```

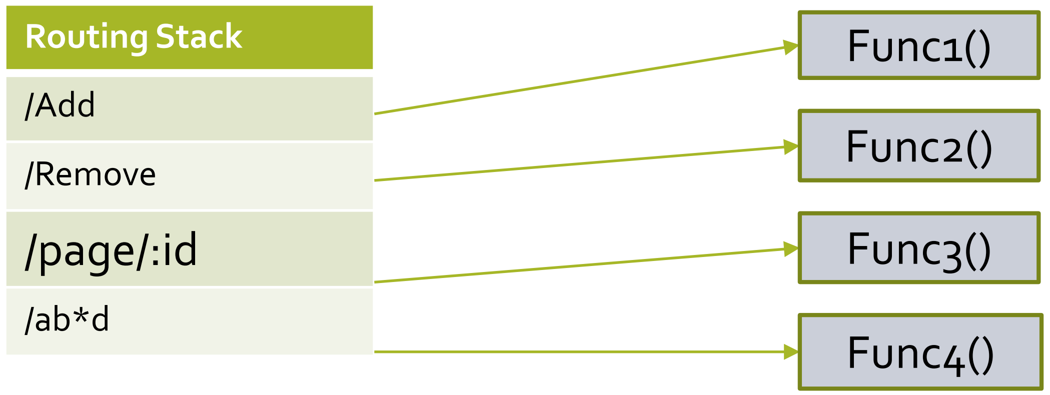
Routing

```
http://server/add?a=3&b=8
```

11 (!)

Server Routing

Maintained in an ordered list (although called “stack” by express).

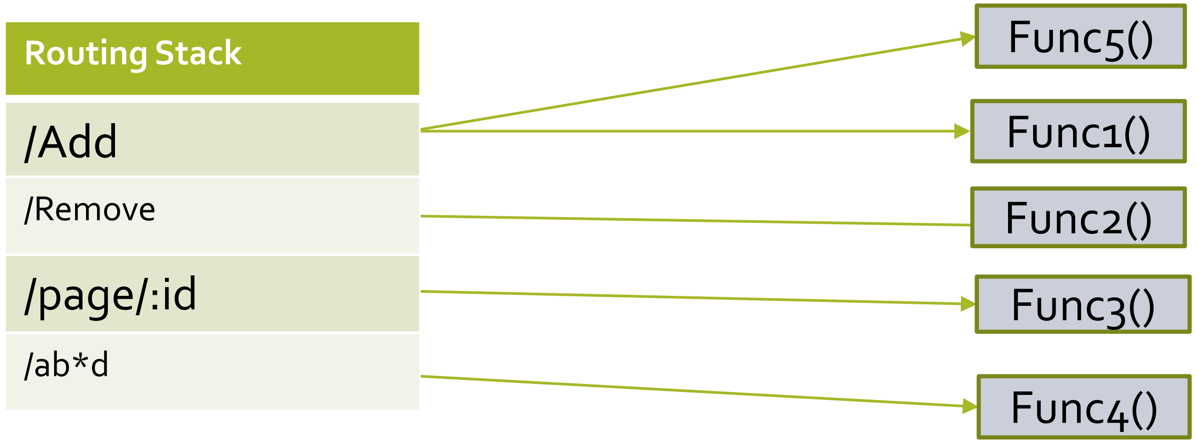


- The stack is accessible in runtime: **app._router.stack**

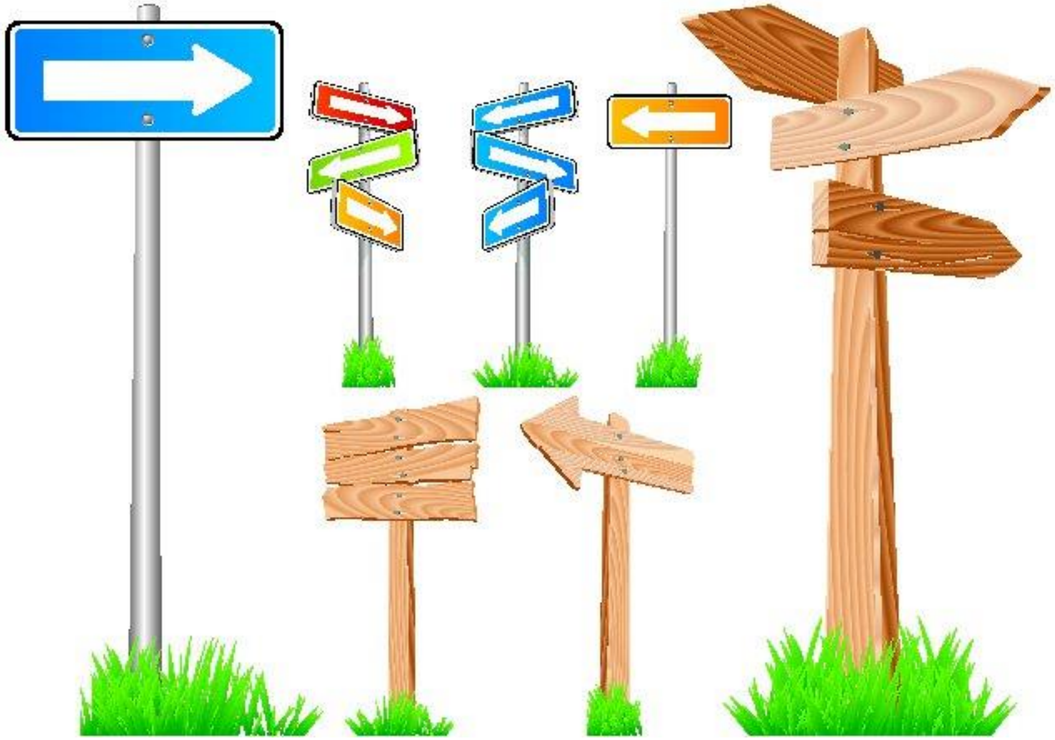
Run-time Server Poisoning

- ‘Stack’ is accessible at run-time (read & write!)
- Replace existing routing with new one
  - Affects <u>all</u> users connecting to system with NO apparent impact to the source code

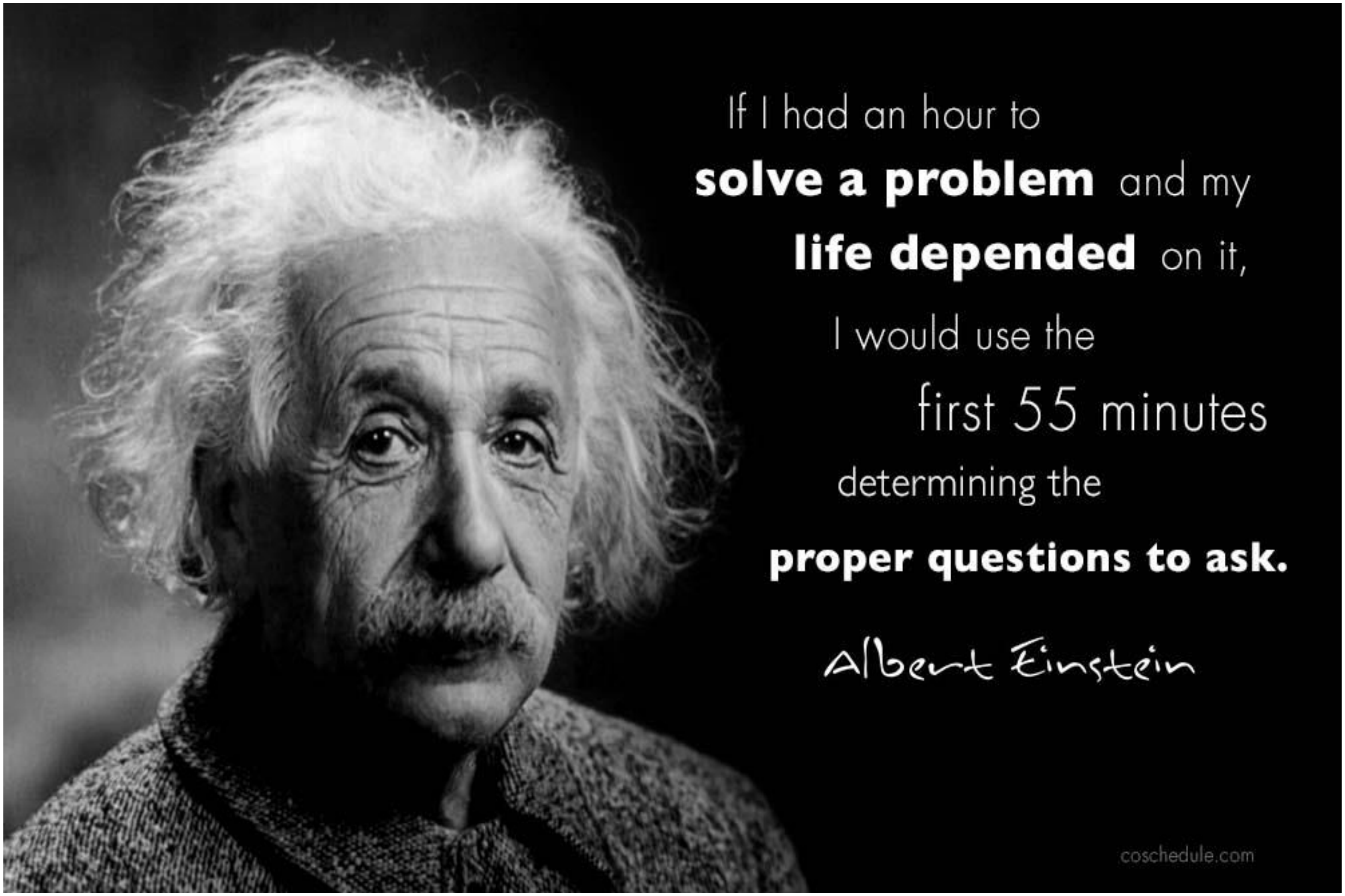
Server Routing Change



```
app._router.stack.splice(3,1); // remove routing entry
app.get('/add',function(req, res) // add new routing
            {
                        return res.render('index',
                                    {message: req.query.a * req.query.b}
                                                );
            });
```



Routing Stack


If I had an hour to
solve a problem and my
life depended on it,
I would use the
first 55 minutes
determining the
proper questions to ask.
Albert Einstein
coschedule.com