CS169 Week 8 Section

Liang (Leon) Gong

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Administrivia & Agenda

Today's Agenda:

- JavaScript
- JS Coding Practice
- JS Performance
- AJAX & Security
- Node.js

JavaScript

- Atwood's Law: any application that can be written in JavaScript, will eventually be written in JavaScript.
- No. 1 in the RedMonk PL Rankings
  - Popularity based on GitHub and StackOverflow

...

But…

- JavaScript is a terribly designed languages.
- Too many confusing features and APIs
  - You should avoid them!
- Only makes sense if you look at it from a historical perspective.

Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Rule: avoid using `for..in` over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

`11 + 22 + 33 => 66`

array index
(not array value)

`0 + 1 + 2 => 3`

array index : string

`0+"0"+"1"+"2" => "0012"`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

`11 + 22 + 33 => 66`

`0 + 1 + 2 => 3`

`0+"0"+"1"+"2" => "0012"`

array index
(not array value)

array index : string

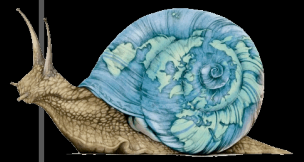
- Cross-browser issues
- Result depends on the Array prototype object

`> "0012indexOftoString..."`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```



```
for (i=0; i < array.length; i++) {
    sum += array[i];
}
```

```
function addup(element, index, array) {
  sum += element;
}
array.forEach(addup);
```

```
for (elem of Array) { ... }
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

But…

- JavaScript is a terribly designed languages.
- Too many confusing features and APIs
  - You should avoid them!
- Only makes sense if you look at it from a historical perspective.

Problematic JavaScript

- Designed and Implemented in 10 days
- Not all decisions were well-thought
- Problematic language features
  - Error prone
  - Poor performance
  - Prone to security vulnerabilities
- Problematic features are still around
  - Backward compatibility
- Political fights among the browser vendors

Problematic JavaScript

- Designed and Implemented in 10 days
- Not all decisions were well-thought
- Problematic language features
  - Error prone
  - Poor performance
  - Prone to security vulnerabilities
- Problematic features are still around
  - Backward compatibility
- Political fights among the browser vendors

Problematic JavaScript

- Designed and Implemented in 10 days
- Not all decisions were well-thought
- Problematic language features
  - Error prone
  - Poor performance
  - Prone to security vulnerabilities
- Problematic features are still around
  - Backward compatibility
- Political fights among the browser vendors

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

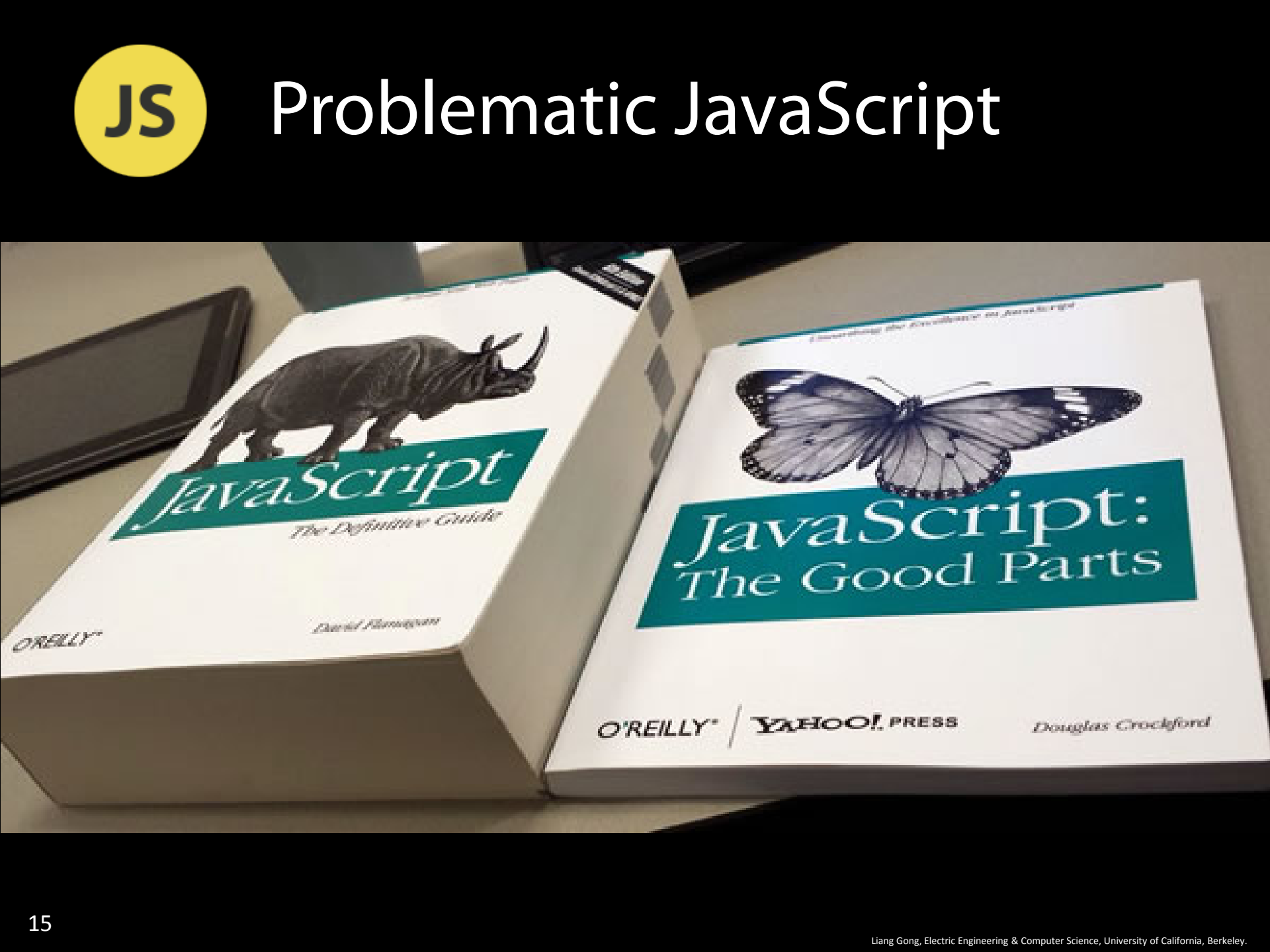
Problematic JavaScript

- Designed and Implemented in 10 days
- Not all decisions were well-thought
- Problematic language features
  - Error prone
  - Poor performance
  - Prone to security vulnerabilities
- Problematic features are still around
  - Backward compatibility
- Political fights among the browser vendors

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Problematic JavaScript

- Designed and Implemented in 10 days
- Not all decisions were well-thought
- Problematic language features
  - Error prone
  - Poor performance
  - Prone to security vulnerabilities
- Problematic features are still around
  - Backward compatibility
- Political fights among the browser vendors


JS
Problematic JavaScript
JavaScript
The Definitive Guide
O'REILLY
David Flanagan
JavaScript:
The Good Parts
O'REILLY
YAHOO! PRESS
Douglas Crockford

Problematic JavaScript

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.


Exhuming the terror of JavaScript
JavaScript:
The Evil Parts
for (p=0;p<2;p++)
document.write(
eval("publisher_"+p))
Gregor Richards
Douglas Crockford

What are coding practices?

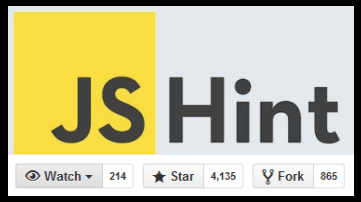
- Good coding practices
  - Informal rules
  - Improve code quality

- Better quality means:
  - Fewer correctness issues
  - Better performance
  - Better usability
  - Better maintainability
  - Fewer security loopholes
  - Fewer surprises
  - …

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Coding Practices and Lint Tools

- Existing Lint-like checkers
  - Inspect source code
  - Rule-based checking
  - Detect common mistakes
- Limitations:
  - Approximates behavior
  - Unknown aliases
  - Lint tools favor precision over soundness
- Difficulty: Precise static program analysis



Language Misuse

Avoid setting properties of primitives, which has no effect.

```
var fact = 42;
fact.isTheAnswer = true;
console.log(fact.isTheAnswer);

> undefined
```

Uncommon Values

Avoid producing ***NaN*** (Not a Number).

```
var x = 23 - "five";

> NaN
```

ALGOT
Frame/6 wire baskets/top shelf
$64
Last year's price: $NaN

ALGOT
Frame/wire baskets/rod
$204
Last year's price: $220

ALGOT
Frame/4 wire baskets/top shelf
$44
Last year's price: $48

ALGOT
Frame/4 wire baskets/top shelf
$60
Last year's price: $NaN

ALGOT
Frame/4 wire baskets
$35
Last year's price: $39

ALGOT
Frame/4 wire baskets
$51
Last year's price: $NaN

ALGOT
Frame/4 mesh baskets/top shelf
$56
Last year's price: $60

ALGOT
Frame with rod/wire baskets
$74
Last year's price: $87

ALGOT
Frame/6 wire baskets/top shelf
$64
Last year's price: $NaN

ALGOT
Frame/wire baskets/rod
$204

ALGOT
Frame/4 wire baskets/top shelf
$44

ALGOT
Frame/4 wire baskets/top shelf
$60
Last year's price: $NaN

ALGOT
Frame/4 wire baskets/top shelf
$60
Last year's price: $NaN

ALGOT
Frame/4 wire baskets
$35
Last year's price: $39

ALGOT
Frame/4 wire baskets
$51
Last year's price: $NaN

ALGOT
Frame/4 mesh baskets/top shelf
$56
Last year's price: $60

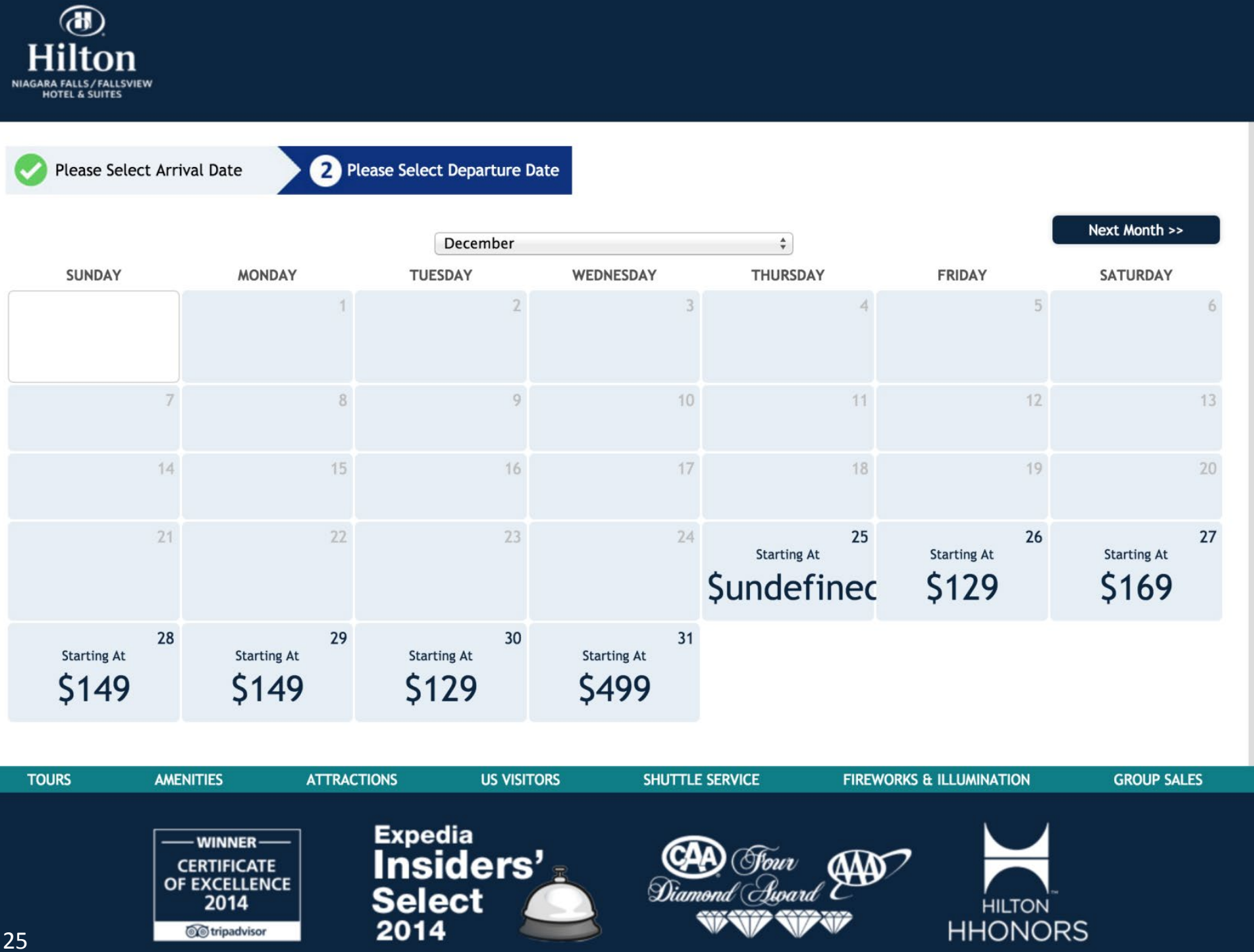
ALGOT
Frame with rod/wire baskets
$74
Last year's price: $87

Uncommon Values

Avoid concatenating undefined to string.

```
var value;
...
var str = "price: ";
...
var result = str + value;

> "price: undefined"
```


Hilton
NIAGARA FALLS / FALLSVIEW
HOTEL & SUITES
Please Select Arrival Date
2 Please Select Departure Date
Next Month >>
December
SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
Starting At
$undefinec
26
Starting At
$129
27
Starting At
$169
28
Starting At
$149
29
Starting At
$149
30
Starting At
$129
31
Starting At
$499
TOURS
AMENITIES
ATTRACTIONS
US VISITORS
SHUTTLE SERVICE
FIREWORKS & ILLUMINATION
GROUP SALES
WINNER
CERTIFICATE OF EXCELLENCE 2014
tripadvisor
Expedia Insiders' Select 2014
CAA Four Diamond Award AAA
HILTON HHONORS

Hilton
NIAGARA FALLS / FALLSVIEW
HOTEL & SUITES

Please Select Arrival Date | 2 Please Select Departure Date

Next Month >>

December

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
			3	4	5	6
7			10	11	12	13
14			17	18	19	20
21			24	25 Starting At $undefinec	26 Starting At $129	27 Starting At $169
28 Starting At $149	29 Starting At $149	30 Starting At $129	31 Starting At $499			

25 Starting At $undefinec

TOURS | AMENITIES | ATTRACTIONS | US VISITORS | SHUTTLE SERVICE | FIREWORKS & ILLUMINATION | GROUP SALES

WINNER CERTIFICATE OF EXCELLENCE 2014 tripadvisor

Expedia Insiders' Select 2014

CAA Four Diamond Award AAA

HILTON HHONORS

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

API Misuse

Beware that all wrapped primitives coerce to true.

```
var b = false;
if (new Boolean(b)) {
  console.log("true");
}

> true
```

Type Related

Avoid accessing the undefined property.

```
var x; // undefined
var y = {};
y[x] = 23; // { undefined: 23 }
```

Rule: avoid setting field on primitive values

From Google Octane's Game Boy Emulator benchmark:

```
var decode64 = "";
if (dataLength > 3 && dataLength % 4 == 0) {
  while (index < dataLength) {
    decode64 += String.fromCharCode(...);
  }
  if (sixbits[3] >= 0x40) {
    decode64.length -= 1;
  }
}
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Rule: avoid setting field on primitive values

From Google Octane's Game Boy Emulator benchmark:

```
var decode64 = "";
if (dataLength > 3 && dataLength % 4 == 0) {
  while (index < dataLength) {
    decode64 += String.fromCharCode(...);
  }
  if (sixbits[3] >= 0x40) {
⚠   decode64.length -= 1;
  }
}
```

No effect because ***decode64*** is a primitive string.

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Rule: avoid no effect operations

⚠ `window.onbeforeunload= "Twitch.player.getPlayer().pauseVideo();"`

⚠ `window.onunload= "Twitch.player.getPlayer().pauseVideo();"`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Rule: avoid no effect operations

⚠ `window.onbeforeunload= "Twitch.player.getPlayer().pauseVideo();"`

⚠ `window.onunload= "Twitch.player.getPlayer().pauseVideo();"`

```
window.onbeforeunload = function () {
    Twitch.player.getPlayer().pauseVideo();
}
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Tips: Use Contiguous Keys for Array

Use contiguous keys starting at 0 for arrays

```
var array = new Array(5000);
```

6.793s

```
var array = [];
for (var i=0;i<5000; i++) {
    array[i] = 0;
}
```

6.8s

```
for (var i=0;i<1000000000;i++) {
    array [i % 5000] = i;
}
```

```
var array = [];
array[5000-1] = 0;
```

9.163s

Tips: Use Contiguous Keys for Array

```
var array = [];
for (var i=10000;i>=0;i--){
    array[i] = i;
}
```

Tips: Use Contiguous Keys for Array

```
var array = [];
for (var i=10000;i>=0;i--){
    array[i] = i;
}
```

```
array[10000] = 10000;
array[9999] = 9999;
   ...
```

- non-contiguous array
- To save memory, JIT-engine decides to represent the array with slow data structures like hash table.

Tips: Use Contiguous Keys for Array

```
var array = [];
for (var i=10000;i>=0;i--){
    array[i] = i;
}
```

```
for (var i=0;i<=10000;i++){
    array[i] = i;
}
```

10X+ speedup!

eval is evil, do not use *eval*.

```
var fun = eval;
...
fun("var a = 1;");
```

- Correctness Issue: `eval('a++;');`
  `var evil = eval; evil('a++;');`
- Performance Issue: *eval* is super slow
- Security Issue: code injection attack.
  `eval('a=' + userInput + ';');`

`eval` is evil, do not use `eval`.

```
var fun = eval;
...
fun("var a = 1;");
```

- Correctness Issue: `eval('a++;');`
  `var evil = eval; evil('a++;');`
- Performance Issue: `eval` is super slow
- Security Issue: code injection attack.
  `eval('a=' + userInput + ';');`

eval is evil, do not use *eval*.

```
var fun = eval;
...
fun("var a = 1;");
```

- Correctness Issue: `eval('a++;');`
  `var evil = eval; evil('a++;');`
- Performance Issue: *eval* is super slow
- Security Issue: code injection attack.
  `eval('a=' + userInput + ';');`

`eval` is evil, do not use `eval`.

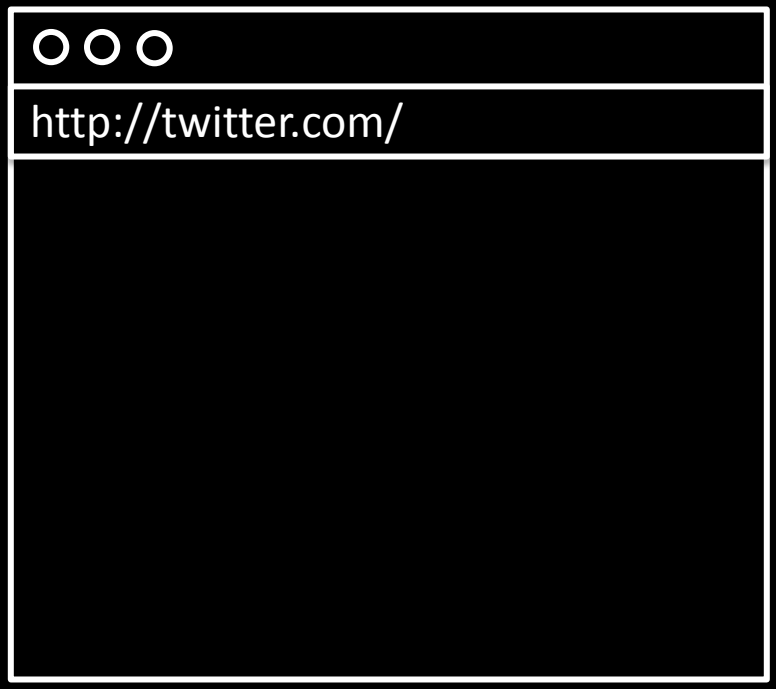
```
var fun = eval;
...
fun("var a = 1;");
```

- Correctness Issue: `eval('a++;');`
  `var evil = eval; evil('a++;');`
- Performance Issue: `eval` is super slow
- Security Issue: code injection attack.
  `eval('a=' + userInput + ';');`

AJAX

`http://twitter.com/`

AJAX



http://twitter.com/

```
<html>
    <head>...</head>
    <body>content</body>
</html>
```

AJAX

http://twitter.com/

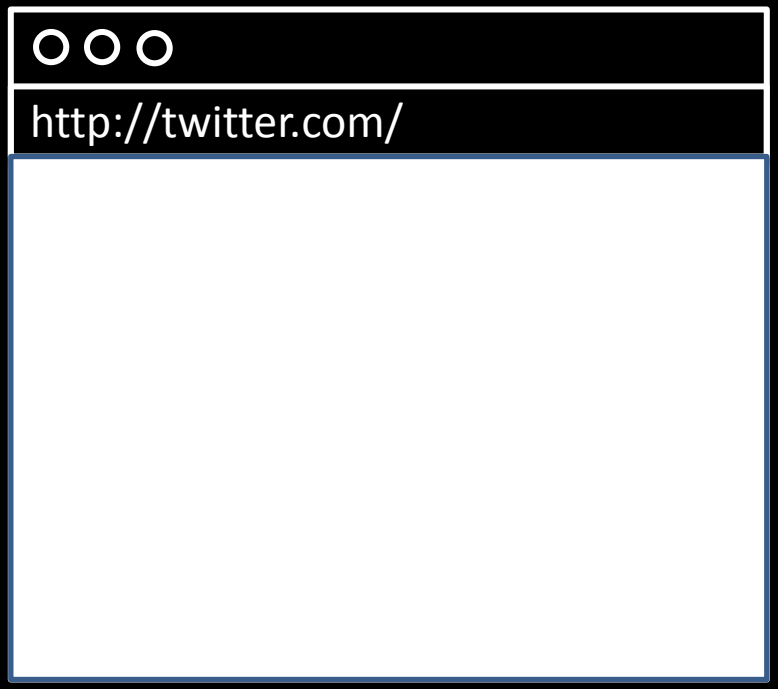
Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX

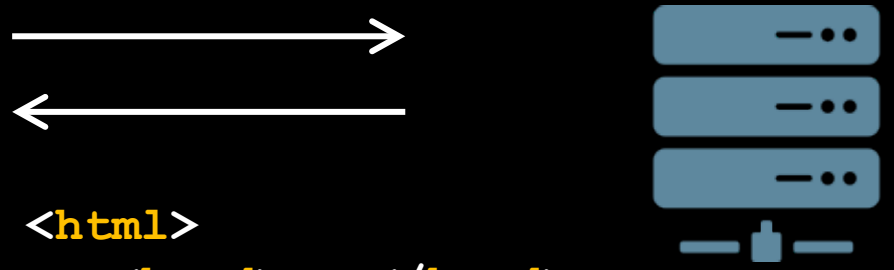
http://twitter.com/

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX



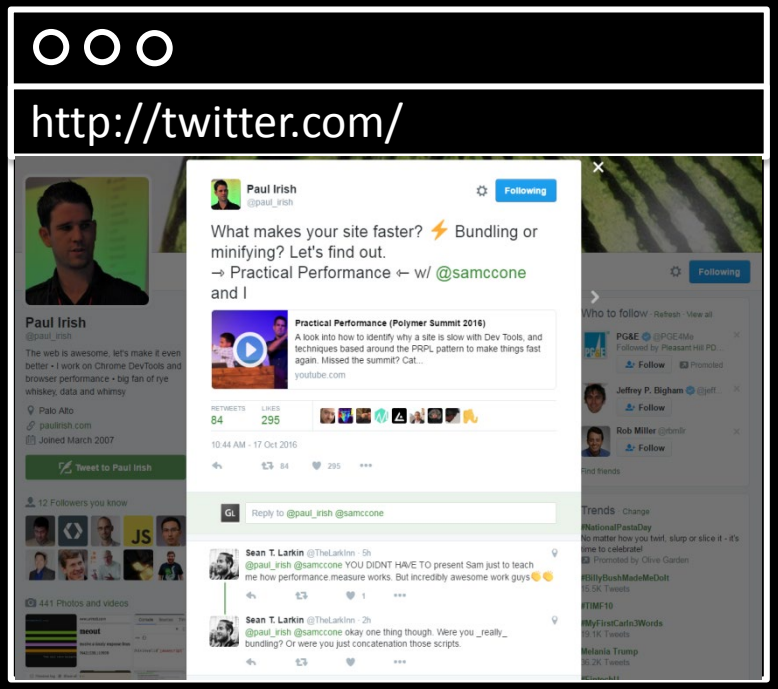
http://twitter.com/



```
<html>
    <head>...</head>
    <body>new content</body>
</html>
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX



http://twitter.com/

```
<html>
    <head>...</head>
    <body>new content</body>
</html>
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX

http://twitter.com/

```
<html>
    <head>...</head>
    <body>new content</body>
</html>
```

Redundant information has to be sent again.

Refresh the entire page → bad user experience

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

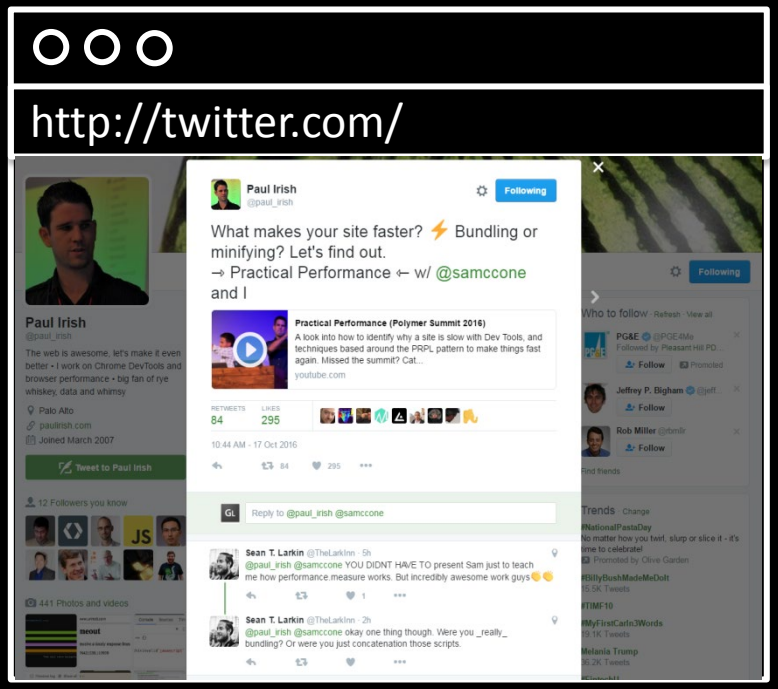
AJAX

http://twitter.com/

```
[
  ["John Doe",
    "foo@gmail.com"],
  ["John Doe",
    "bar@gmail.com"]
]
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

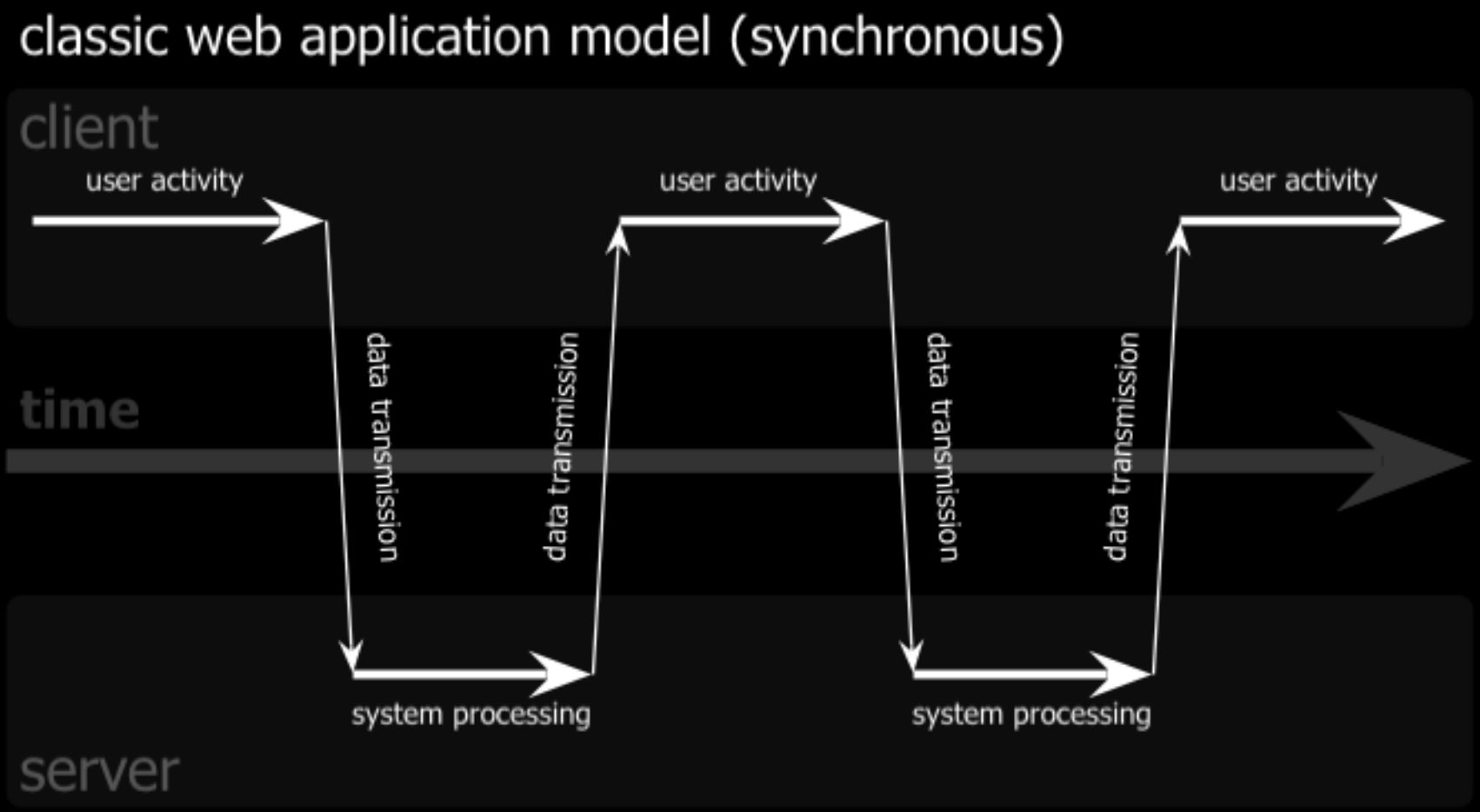
AJAX




```
[
  ["John Doe",
    "foo@gmail.com"],
  ["John Doe",
    "bar@gmail.com"]
]
```

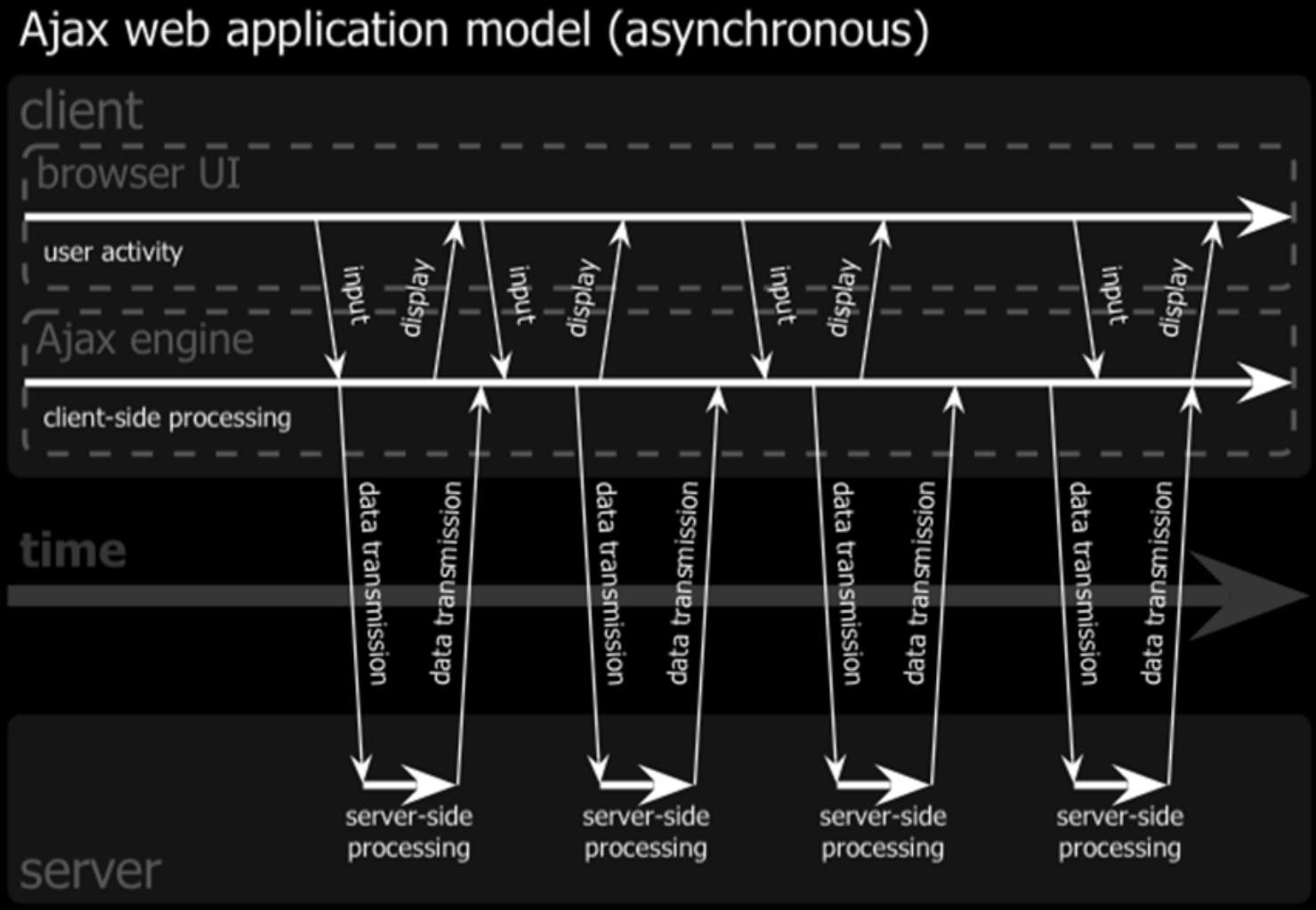
Only send the new part (in HTML, XML, or JSON).
No need to refresh the page.

AJAX



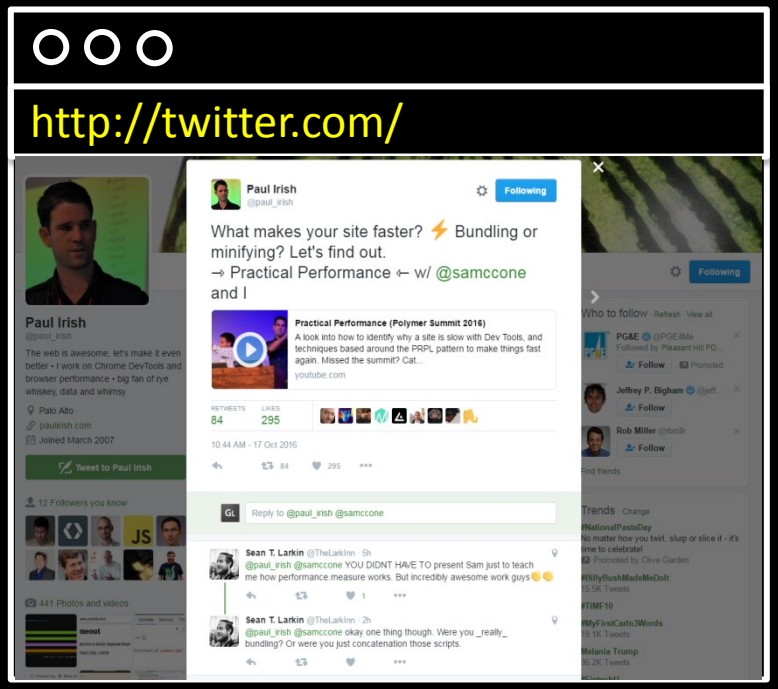
Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX & Same Origin Policy



`http://twitter.com/`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX & Same Origin Policy

Origin: http://twitter.com/

`http://twitter.com/`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX & Same Origin Policy

Origin: http://twitter.com/

AJAX request

`http://twitter.com/`

AJAX & Same Origin Policy

Origin: http://twitter.com/

AJAX request

`http://twitter.com/`

`http://google.com/`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX & Same Origin Policy

Origin: http://twitter.com/

AJAX request

`http://twitter.com/`

`http://google.com/`

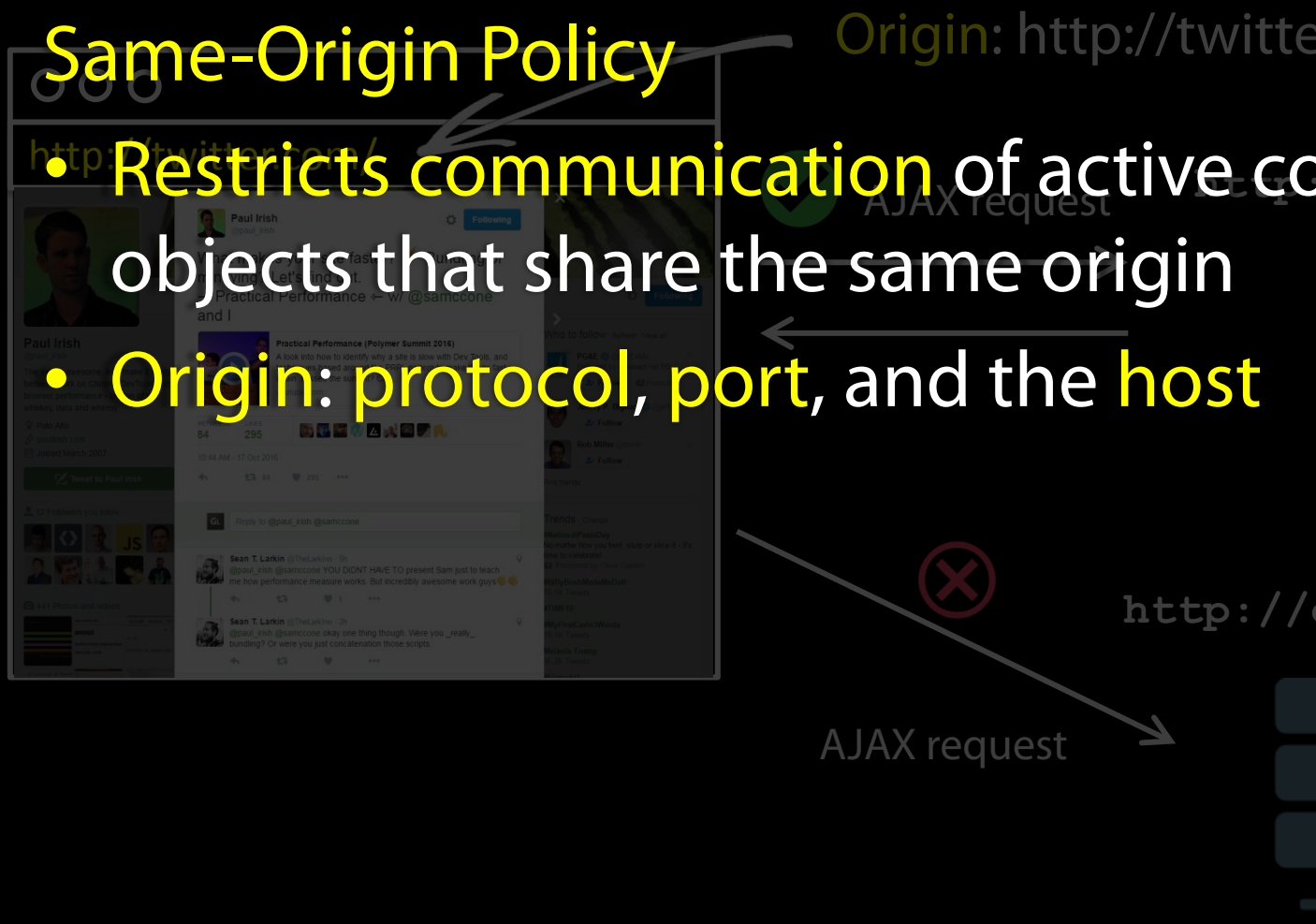
AJAX request

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

AJAX & Same Origin Policy

Same-Origin Policy

- Restricts communication of active content to objects that share the same origin
- Origin: protocol, port, and the host



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

An Attack Scenario

`http://attacker.org/`

An Attack Scenario

`http://attacker.org/`

`http://email.com/`

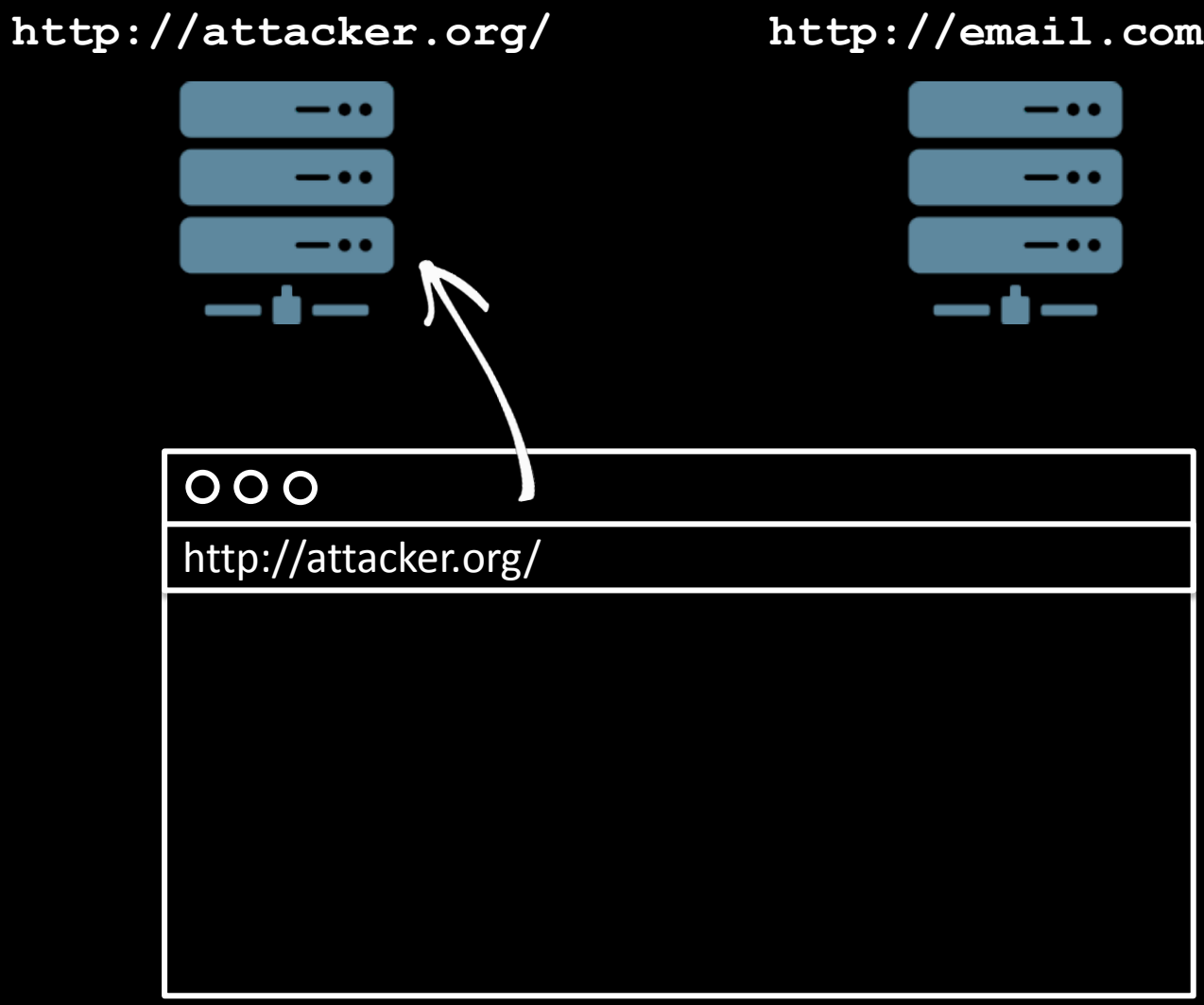
An Attack Scenario

`http://attacker.org/`

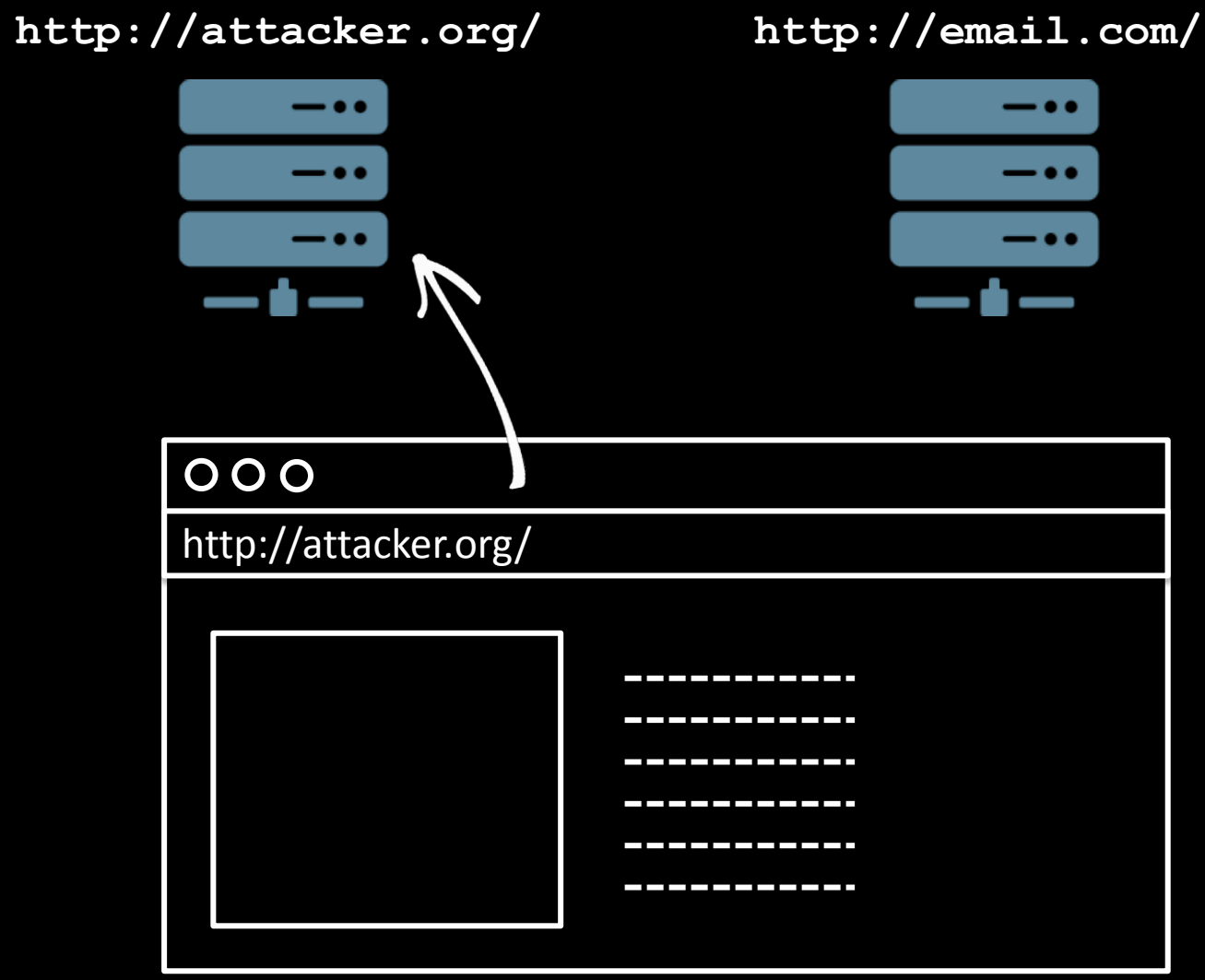
`http://email.com/`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

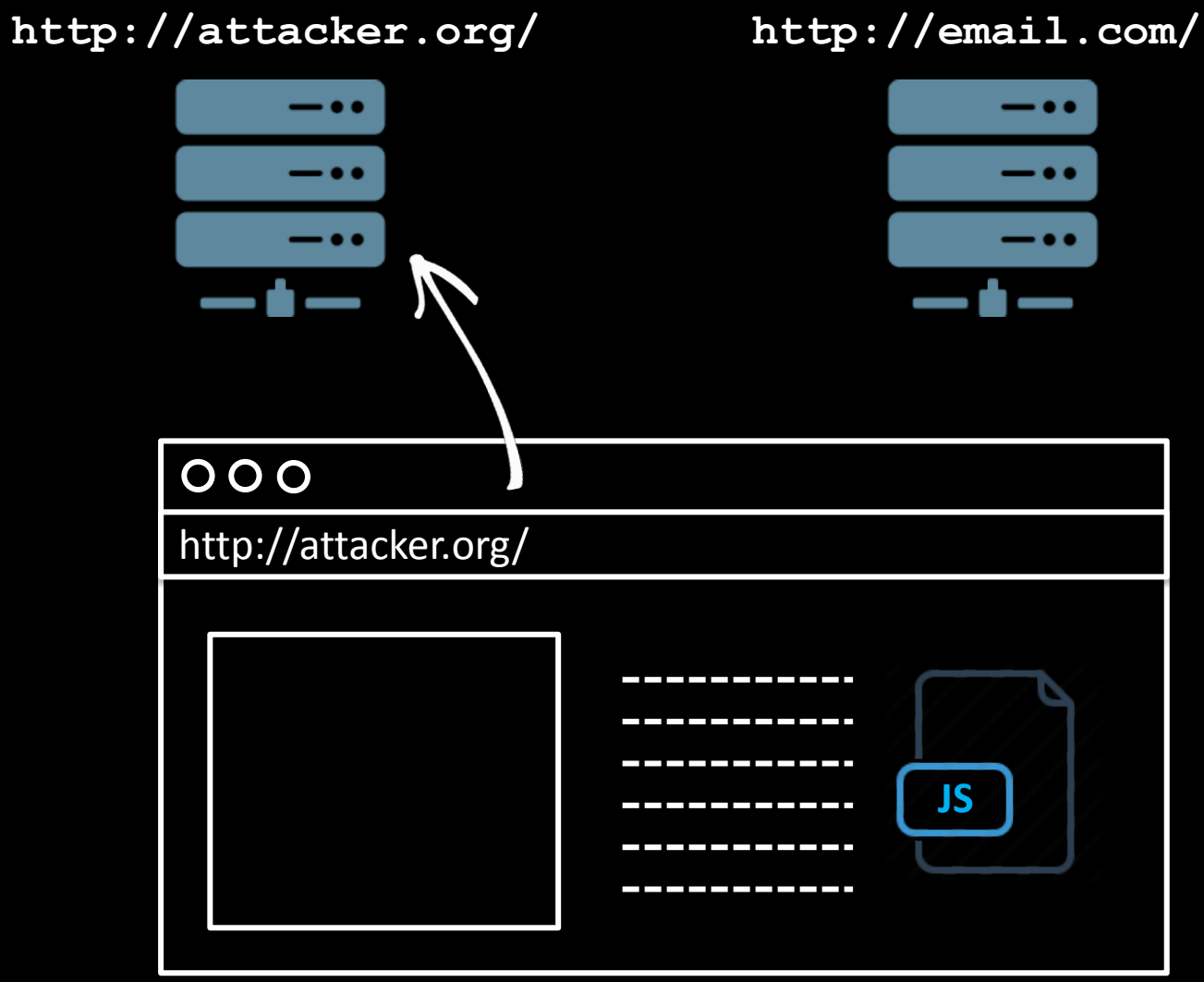
An Attack Scenario



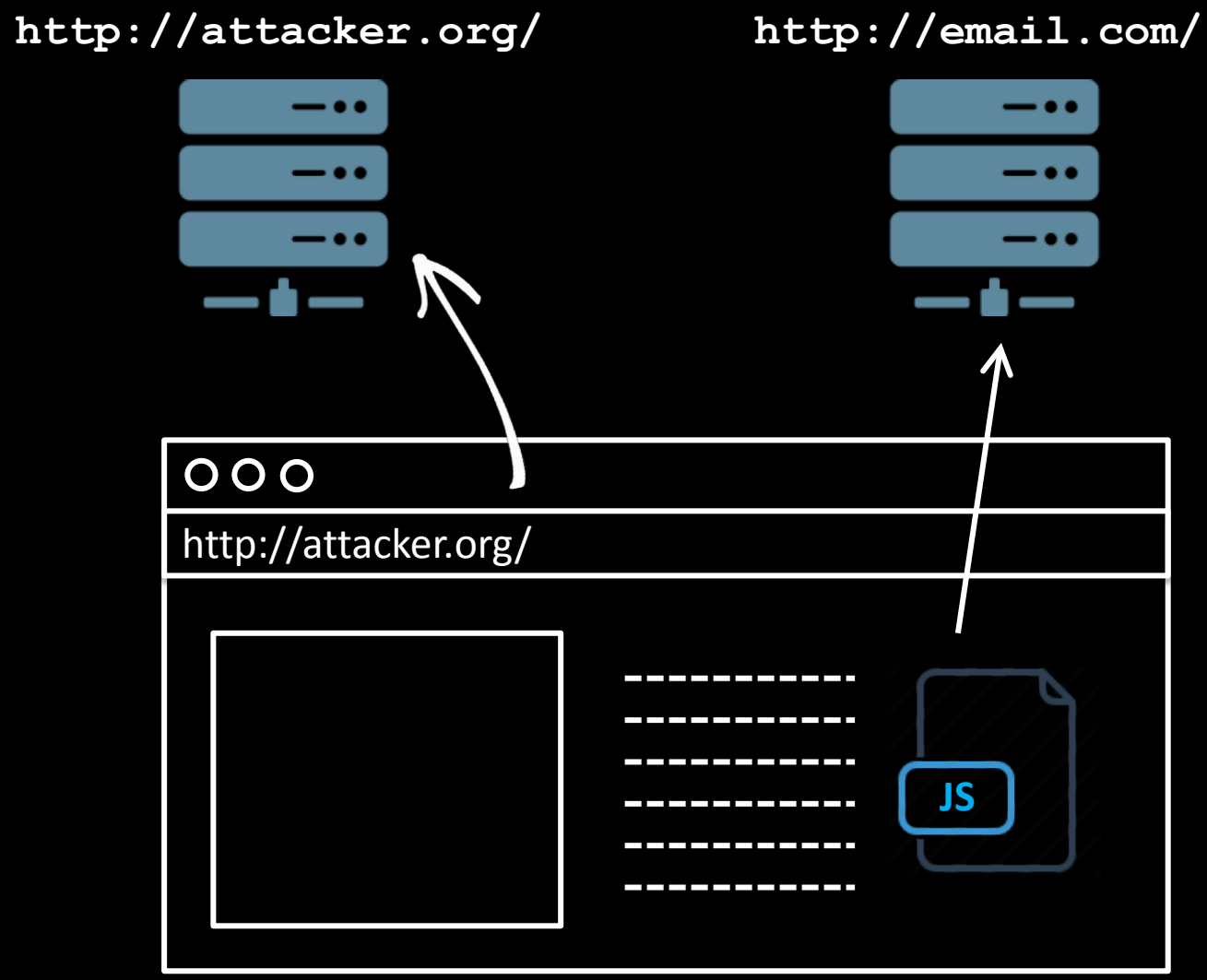
An Attack Scenario



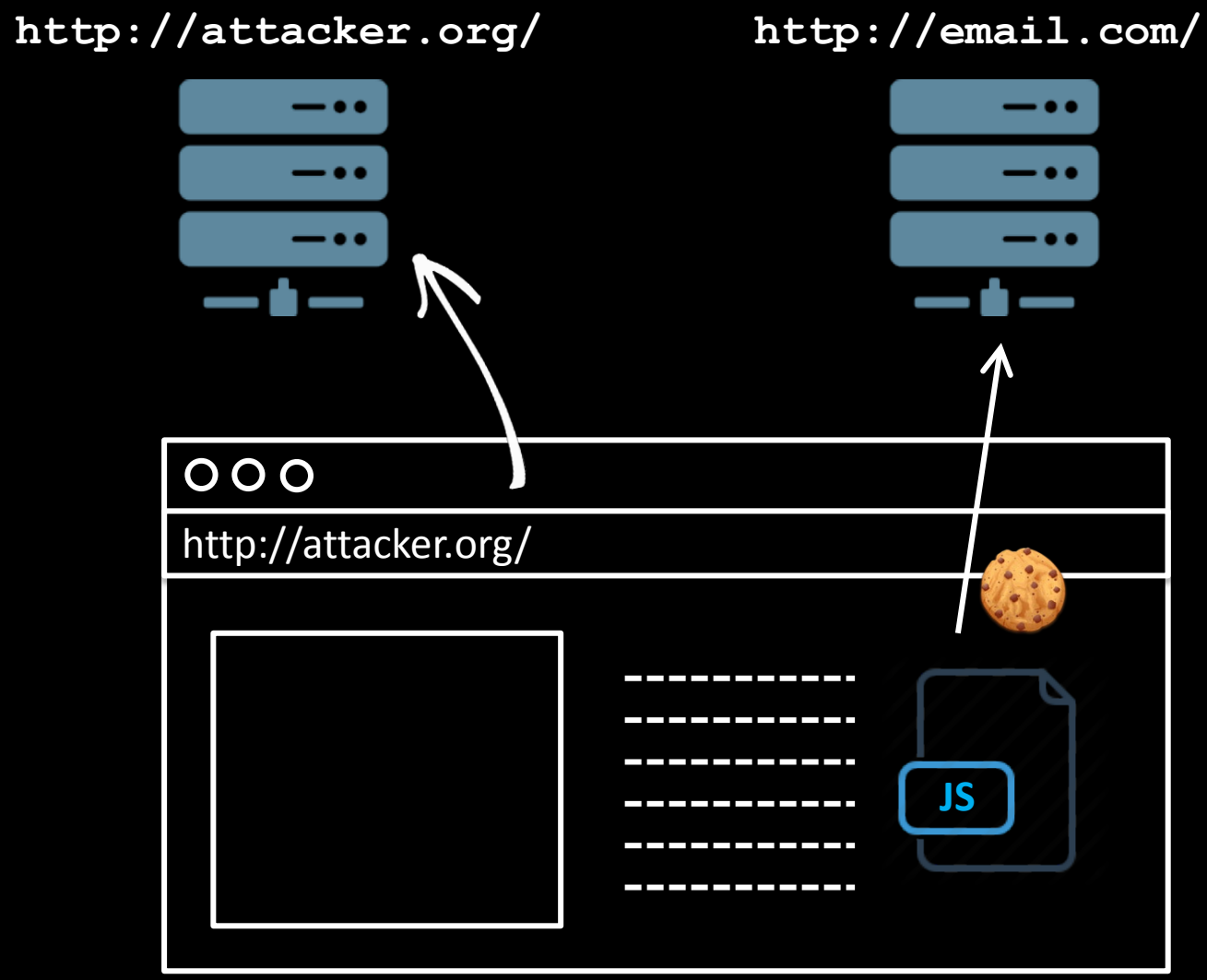
An Attack Scenario



An Attack Scenario



An Attack Scenario



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

An Attack Scenario

http://attacker.org/ http://email.com/

Same-Origin Policy

- Restricts communication of active content to objects that share the same origin
- Origin: protocol, port, and the host

JSON Hijacking (2006)

`http://attacker.org/`

`http://gmail.com/`

JSON Hijacking (2006)

`http://attacker.org/`

`http://gmail.com/`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

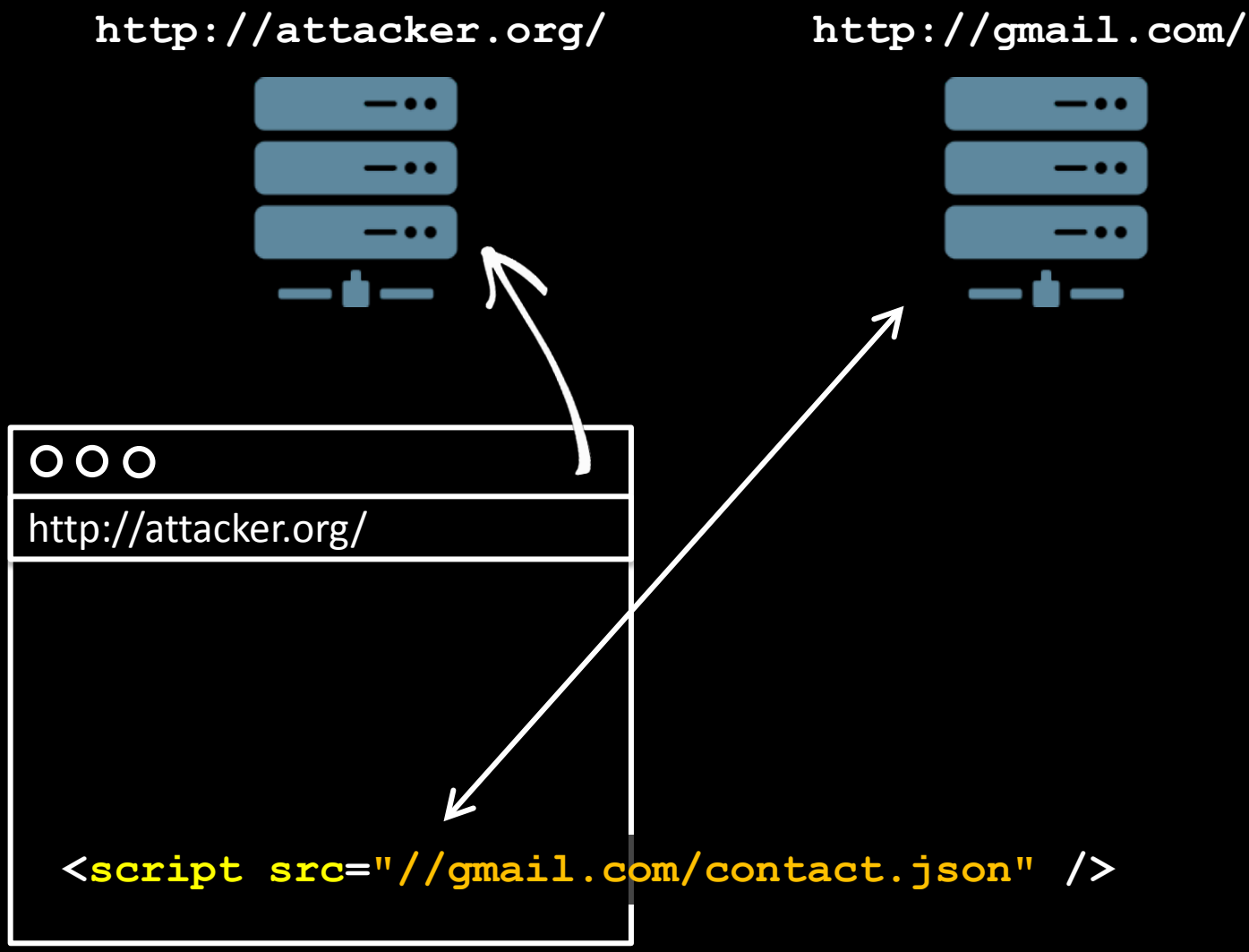
JSON Hijacking (2006)

http://attacker.org/

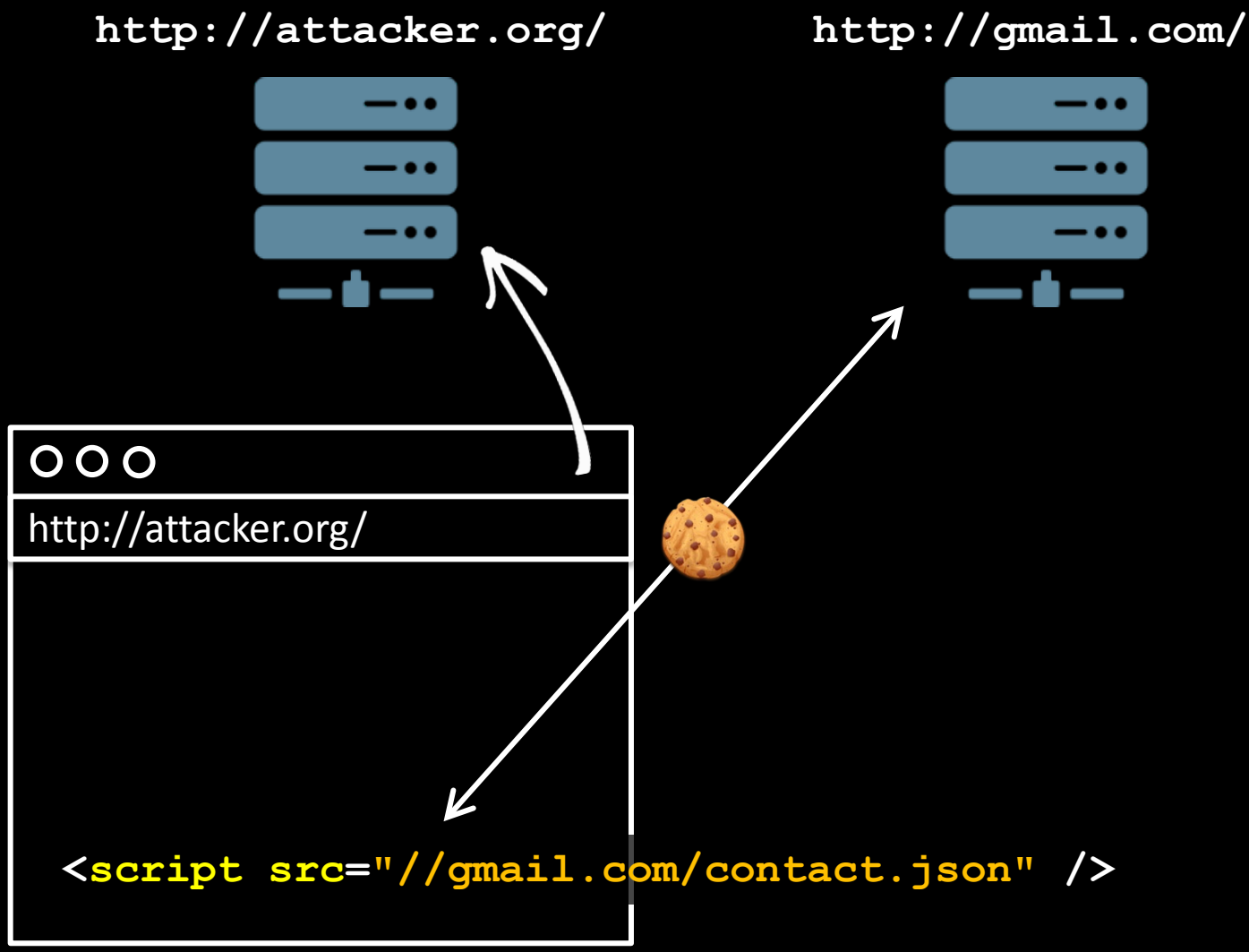
http://gmail.com/

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

JSON Hijacking (2006)

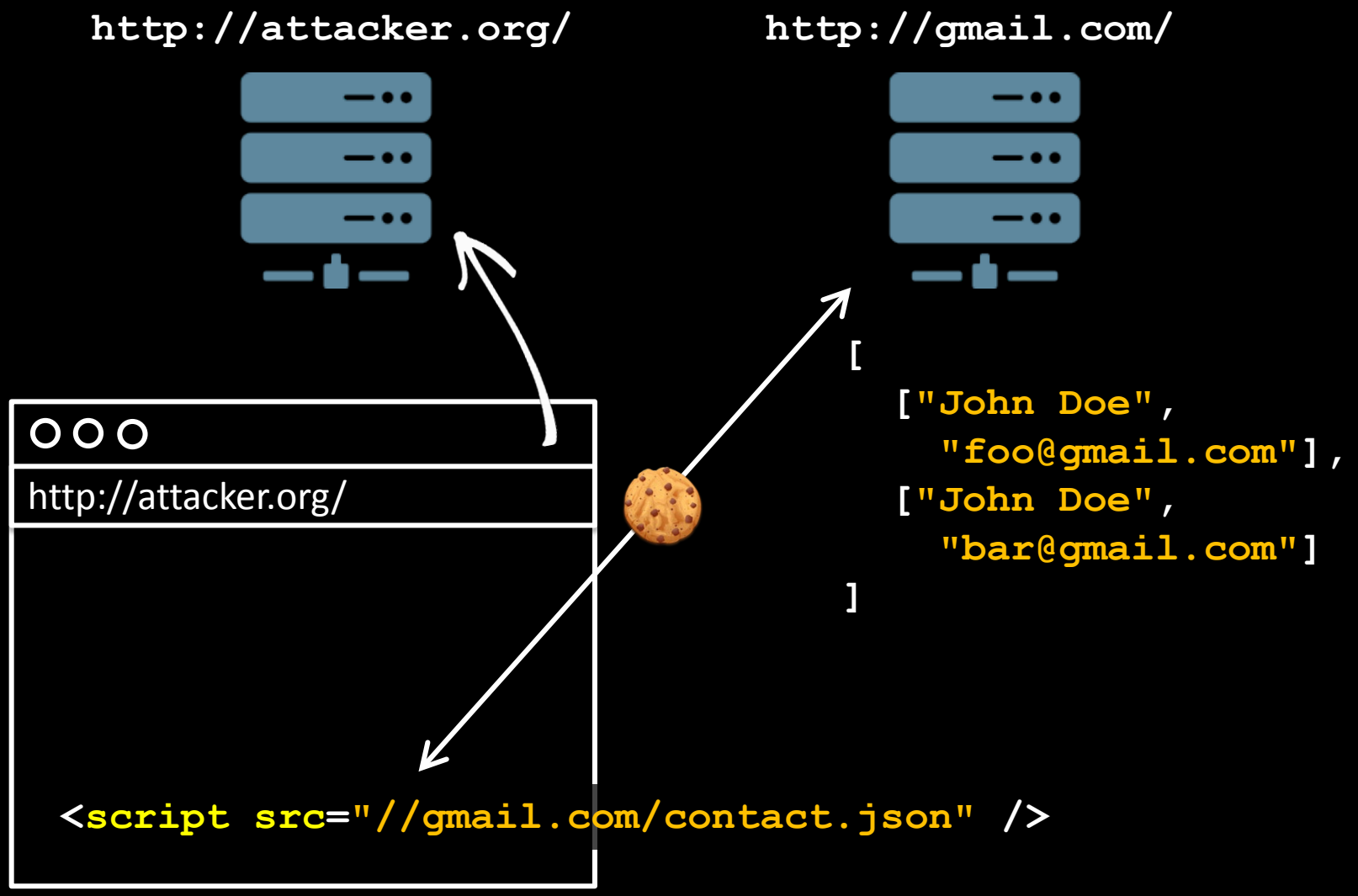


JSON Hijacking (2006)



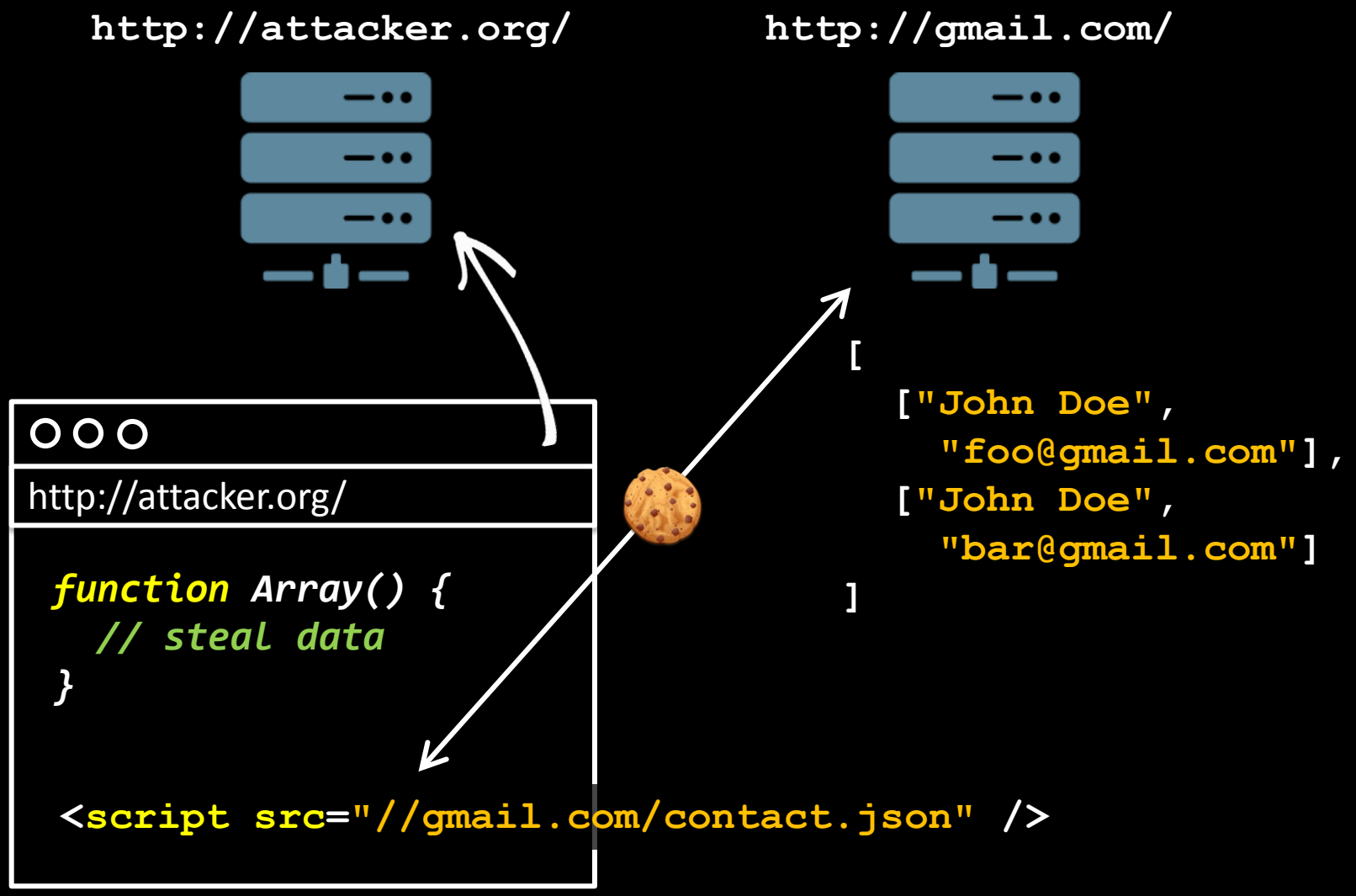
Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

JSON Hijacking (2006)



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

JSON Hijacking (2006)



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Cross-Site Script Inclusion (XSSI)

`http://attacker.org/`

`http://vuln.com/`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

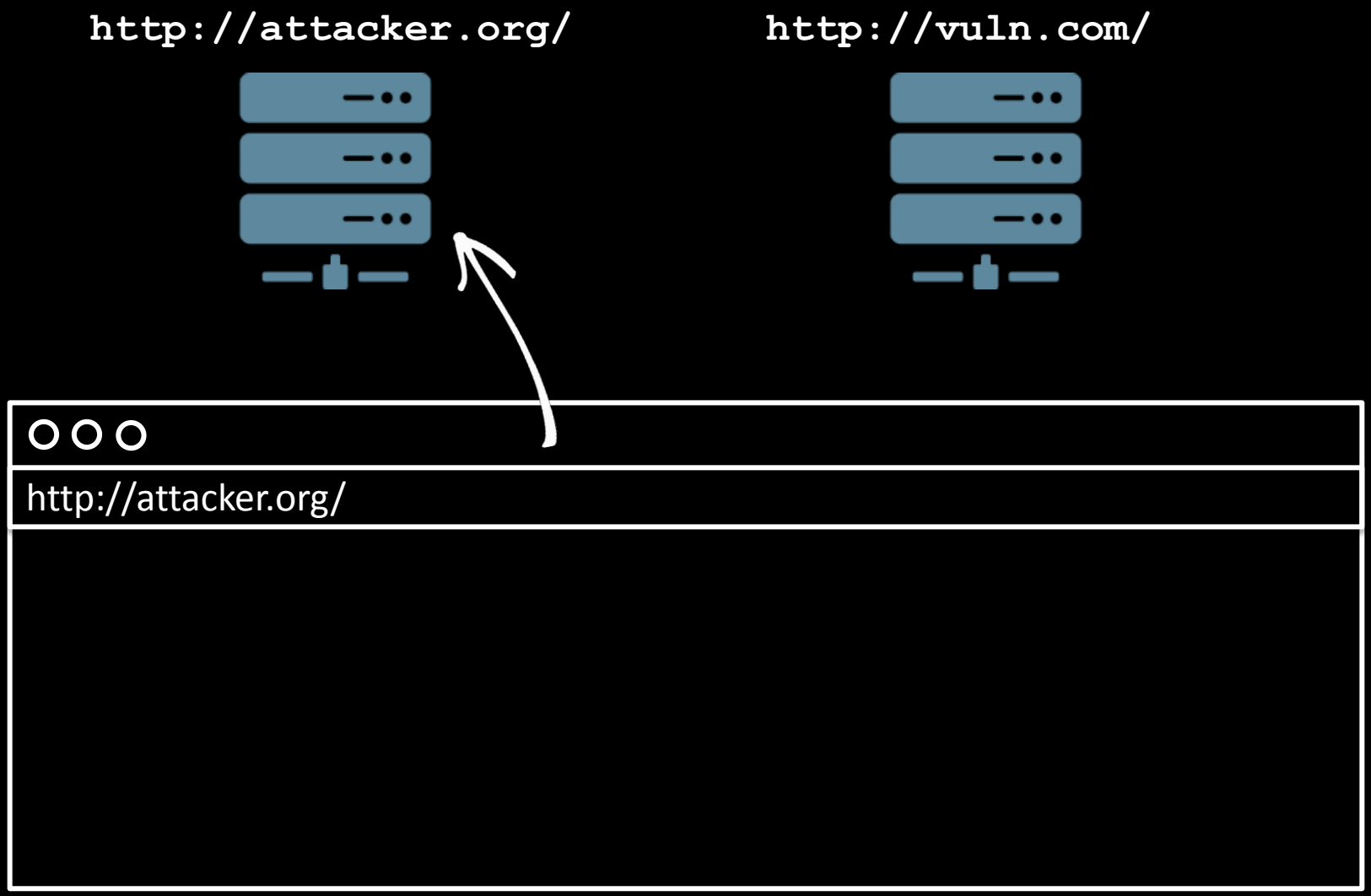
Cross-Site Script Inclusion (XSSI)

`http://attacker.org/`

`http://vuln.com/`

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Cross-Site Script Inclusion (XSSI)

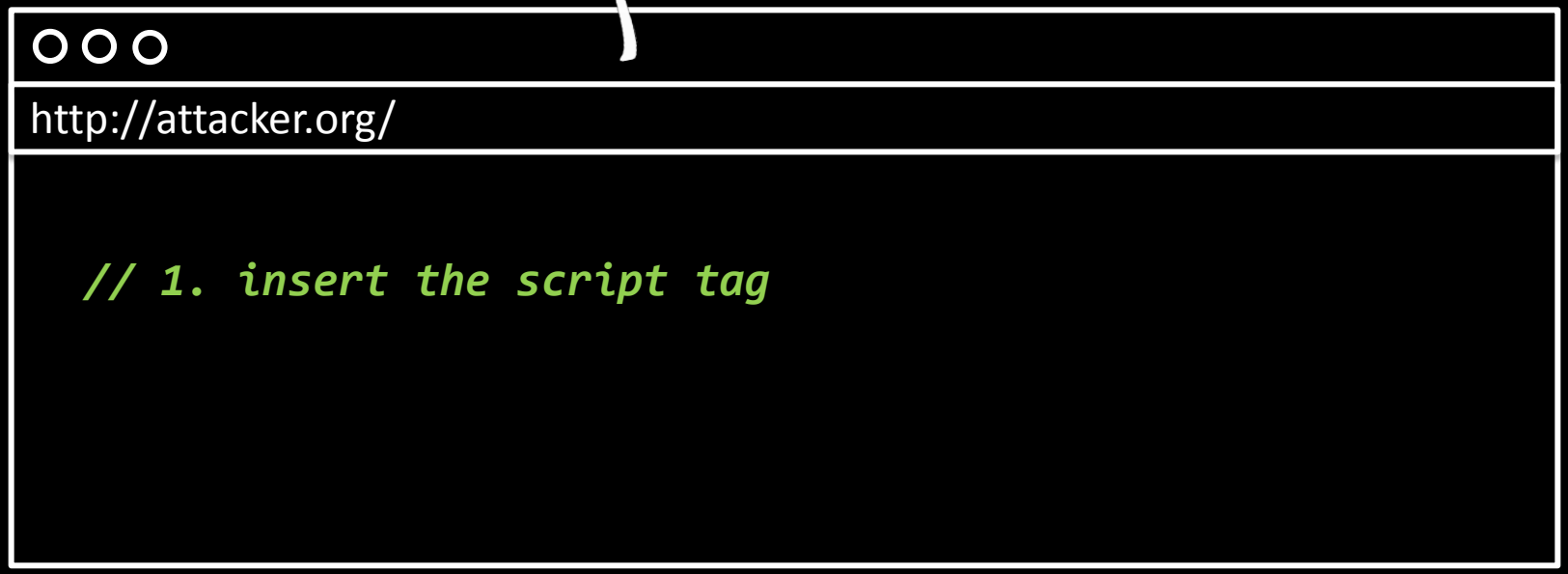


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Cross-Site Script Inclusion (XSSI)

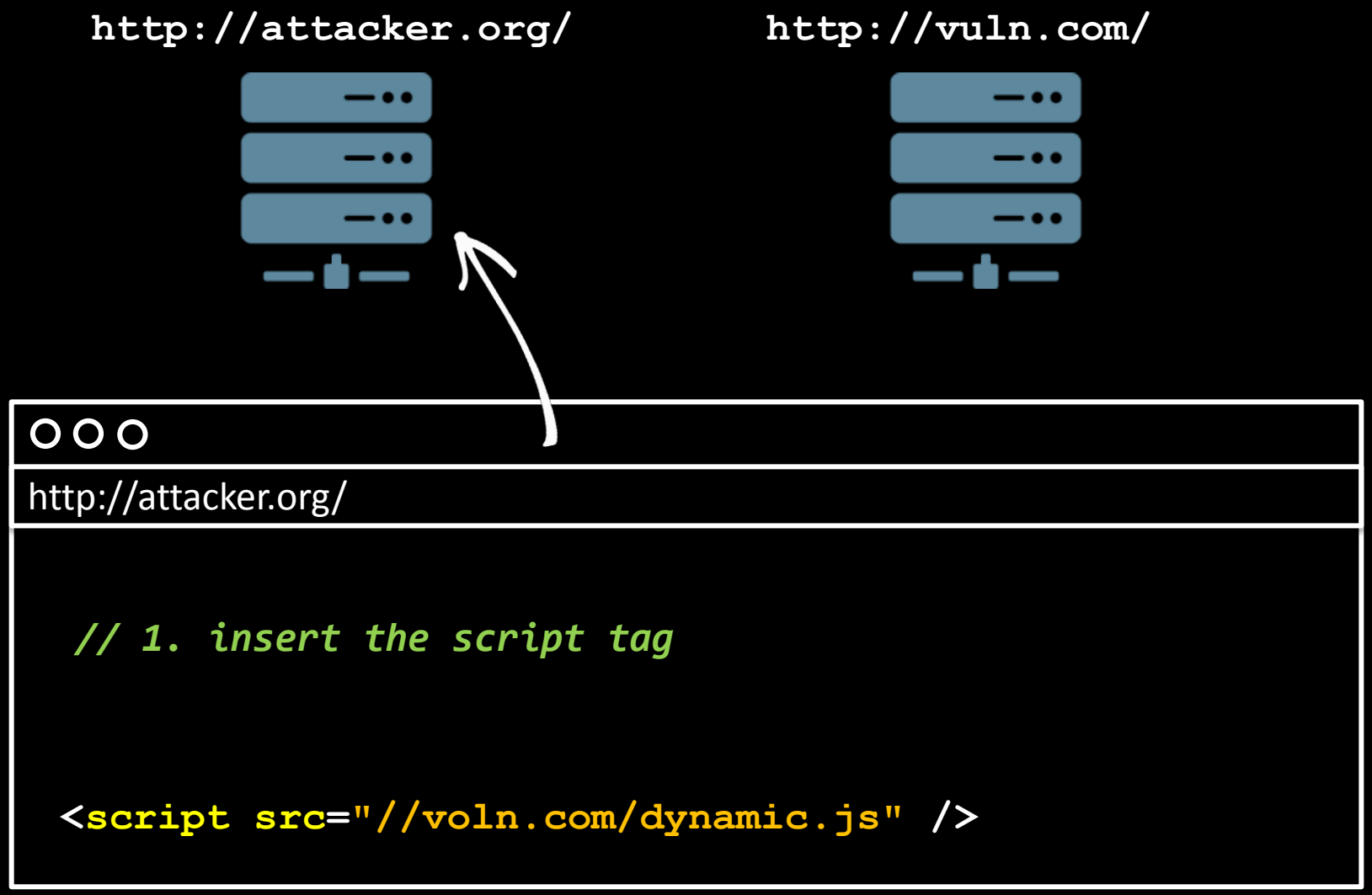
`http://attacker.org/`

`http://vuln.com/`



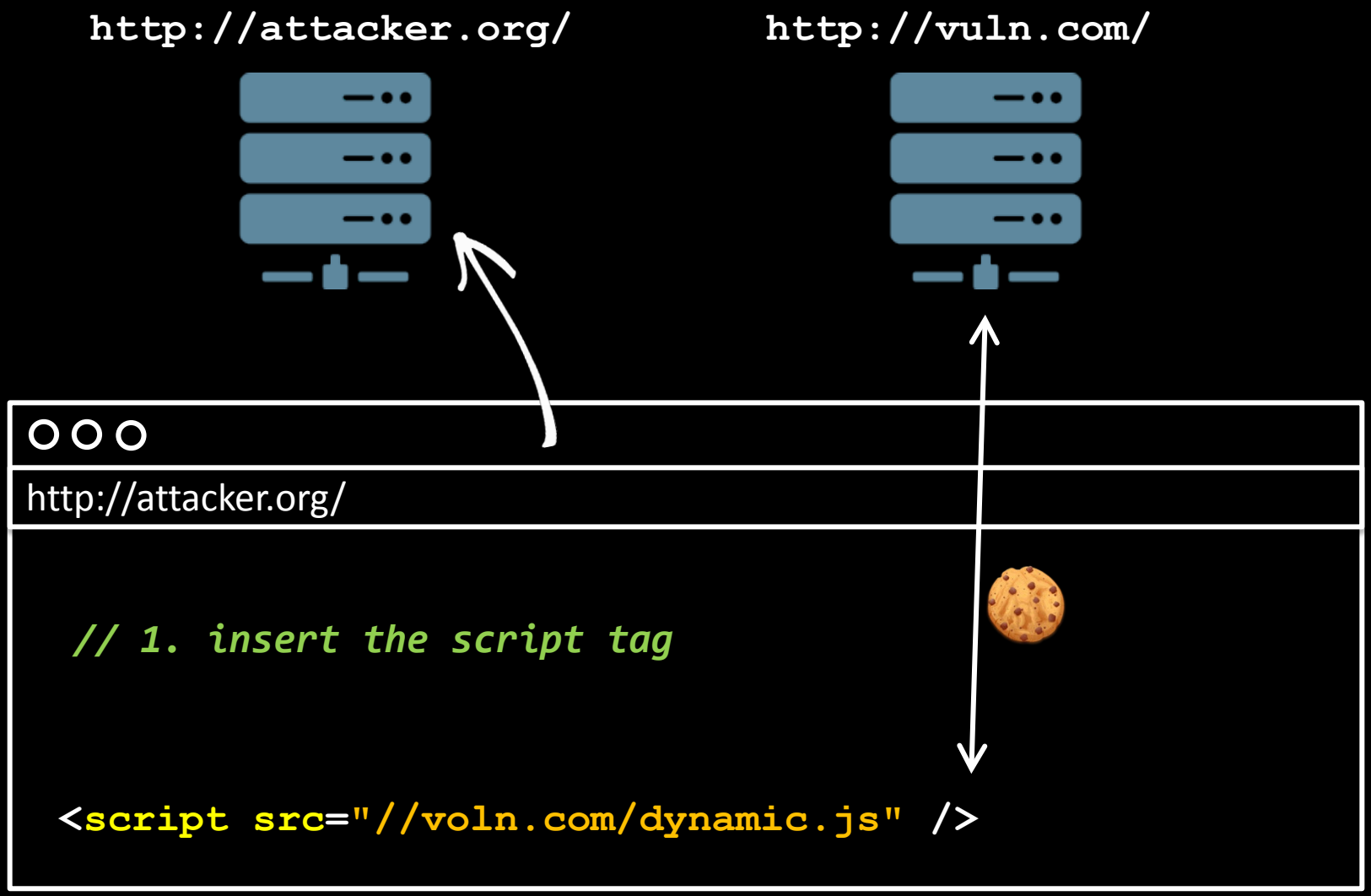
Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Cross-Site Script Inclusion (XSSI)



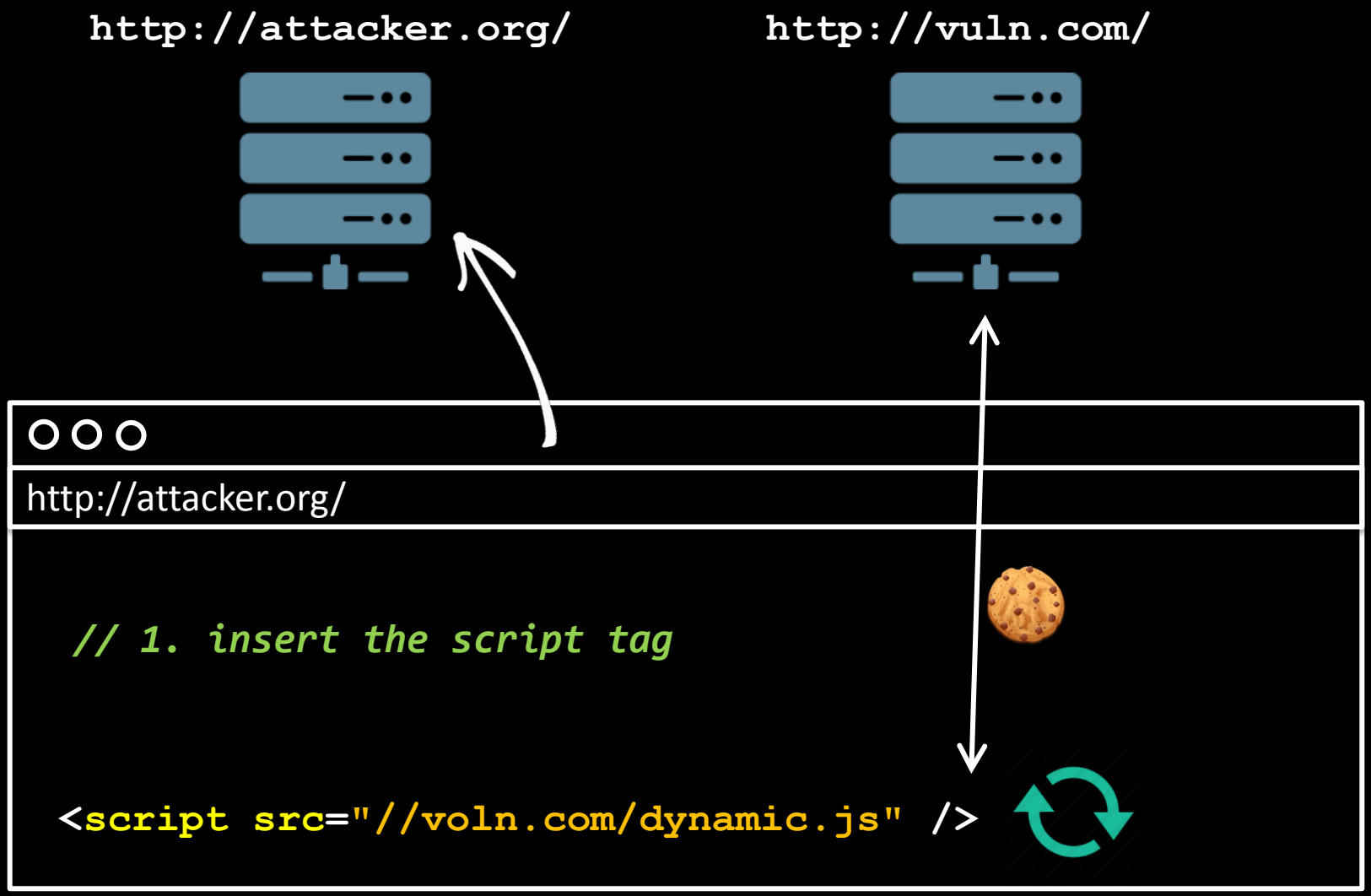
Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Cross-Site Script Inclusion (XSSI)



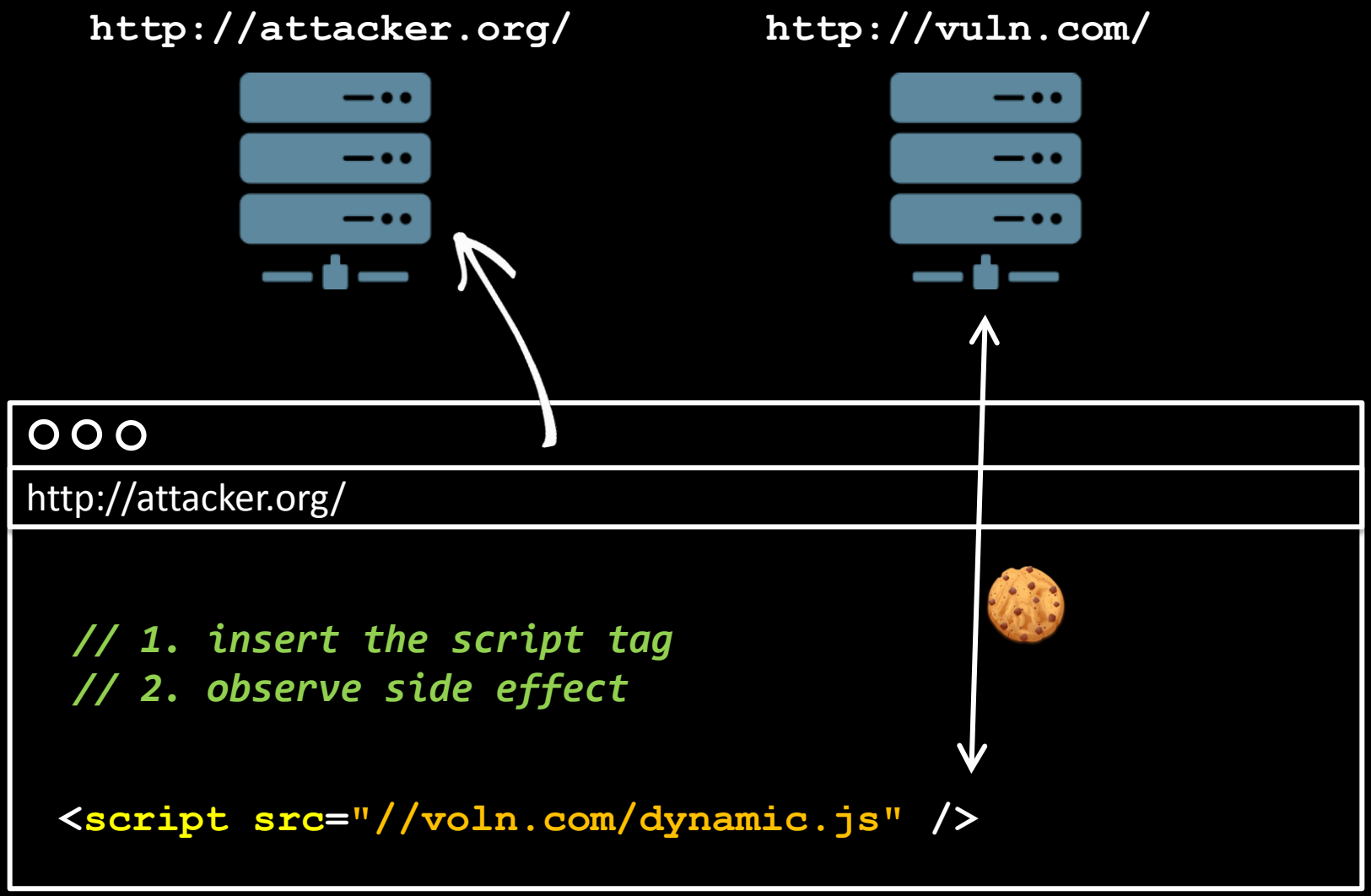
Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Cross-Site Script Inclusion (XSSI)



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Cross-Site Script Inclusion (XSSI)



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

RQ: Can such data be leaked in a similar way?

Leaking data stored in global variables:

dynamic.js

```
// local variable at top level
var first_name = "John";

// variable missing the "var" keyword
last_name = "Doe";

// global variable
window.user_email = "john@doe.com";
```

evil.js

```
console.log(first_name);
console.log(last_name);
Console.log(user_email);
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

RQ: Can such data be leaked in a similar way?

Leaking data via global functions:

dynamic.js

```
function example() {
  var email = "john@doe.com";
  window.MyLibrary.doSomething(email);
}
```

evil.js

```
window.MyLibrary = {};
window.MyLibrary.doSomething = function(email) {
  console.log(email);
}
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

RQ: Can such data be leaked in a similar way?

Leaking data via built-in APIs:

dynamic.js

```
function example() {
  var email = "john@doe.com";
  JSON.stringify(email);
}
```

evil.js

```
JSON.stringify = function (data) {
  sendToAttackerBackend(data);
}
```


Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

RQ: Can such data be leaked in a similar way?

Other APIs:

```
ArrayBuffer, Map, Set, WeakMap, WeakSet
decudeURI, decudeURIComponent,
encodeURI, encodeURIComponent,
escape, unescape
...
```

```
JSON.stringify = function (data) {                    evil.js
  sendToAttackerBackend(data);
}
```

Prototypical Inheritance

// object1 → Object.prototype → null

```
var object1 = {a: 1};
```

// object2 → object1 → Object.prototype → null

```
var object2 = Object.create(object1);

console.log(object2.a)
> 1
```

RQ: Can such data be leaked in a similar way?

Leaking data via the ***this*** reference:

dynamic.js

```
(function () {
  var secret_values = ["john@doe.com", "top secret"];
  secret_values.forEach(function (secret) {
    // do something secret here
  })
})();
```

evil.js

```
Array.prototype.forEach = function(callback) {
  console.log(this); // this points to the array
}
```

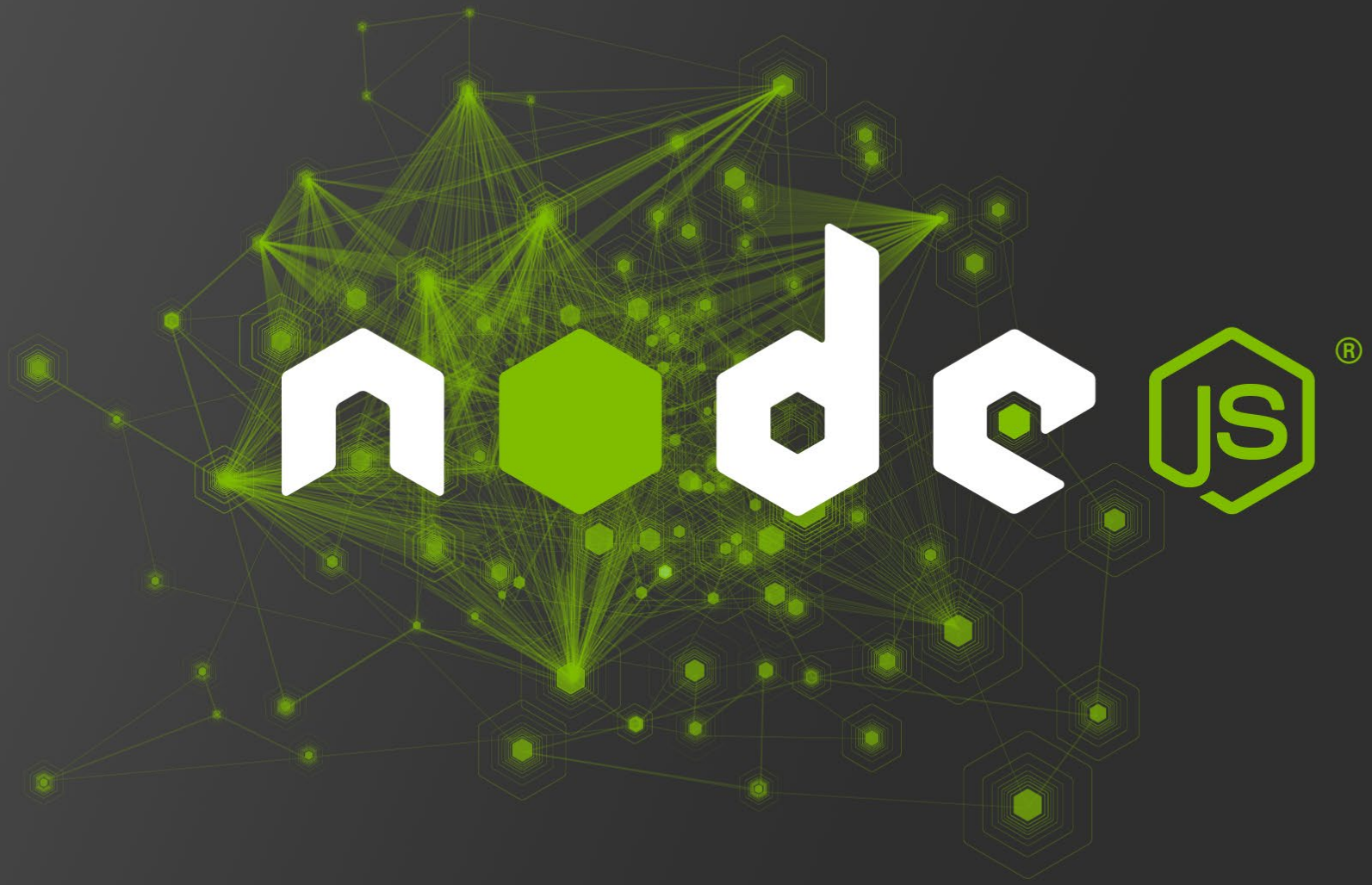

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

RQ: Can such data be leaked in a similar way?

Other APIs:

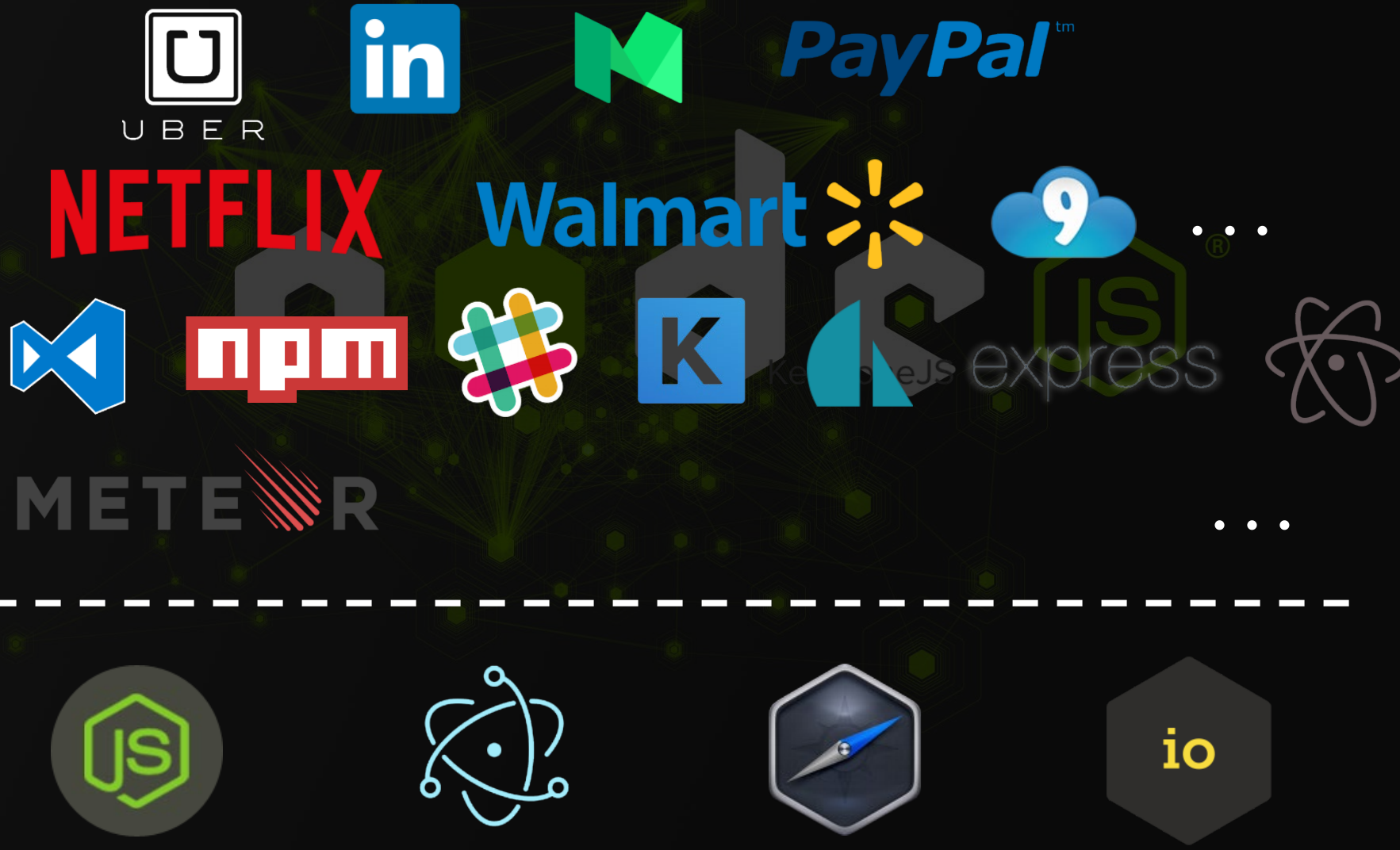
String.prototype.concat
String.prototype.substring
String.prototype.toString
String.prototype.replace
Array.prototype.join
Array.prototype.map
Array.prototype.forEach
...

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.



Liang Gong, Cal Ph.D. Student, Research Intern in Microsoft Research, Redmond.

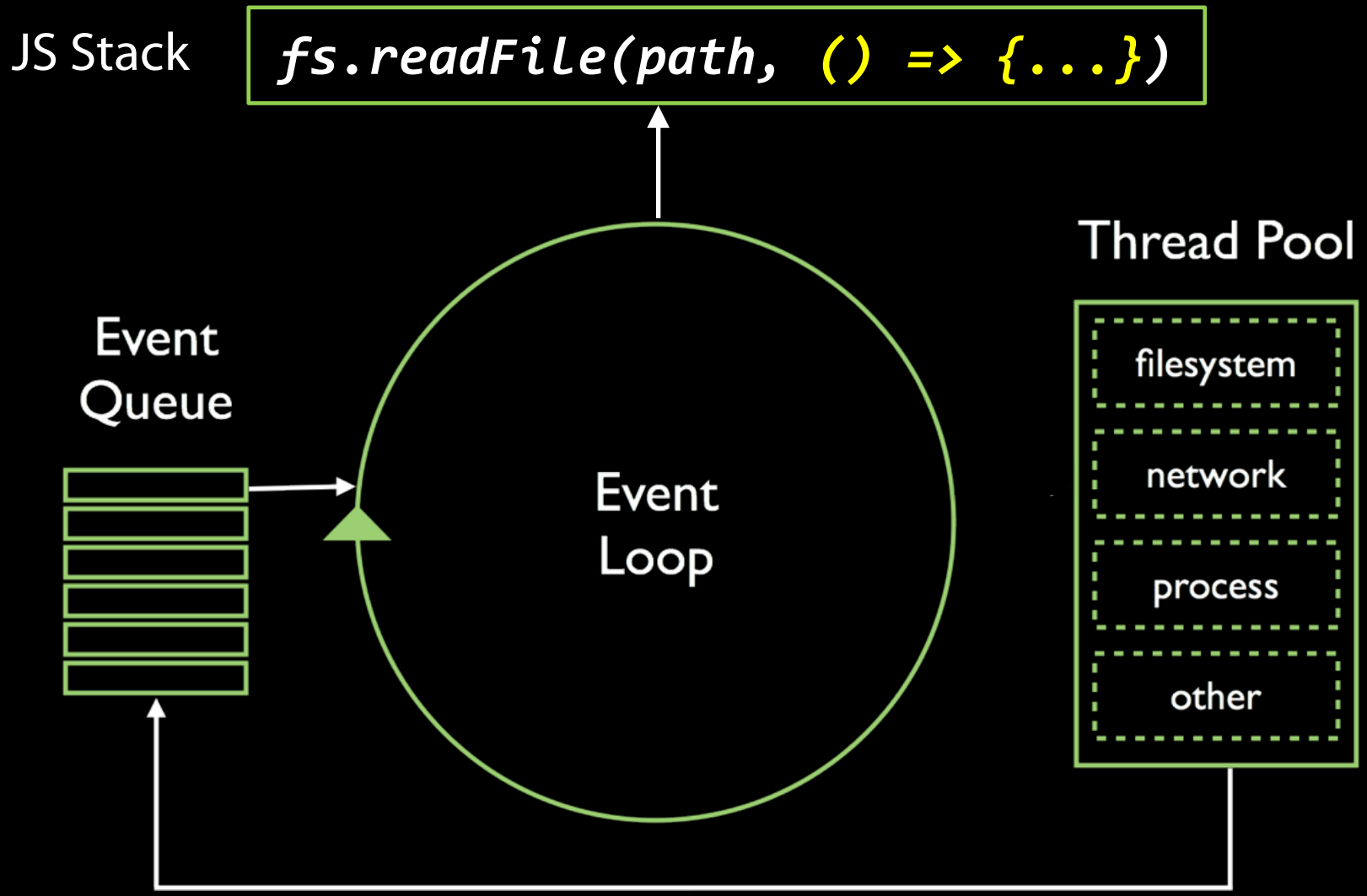
Motivation: Node.js is Rising



Async I/Os: Blessing and Curse

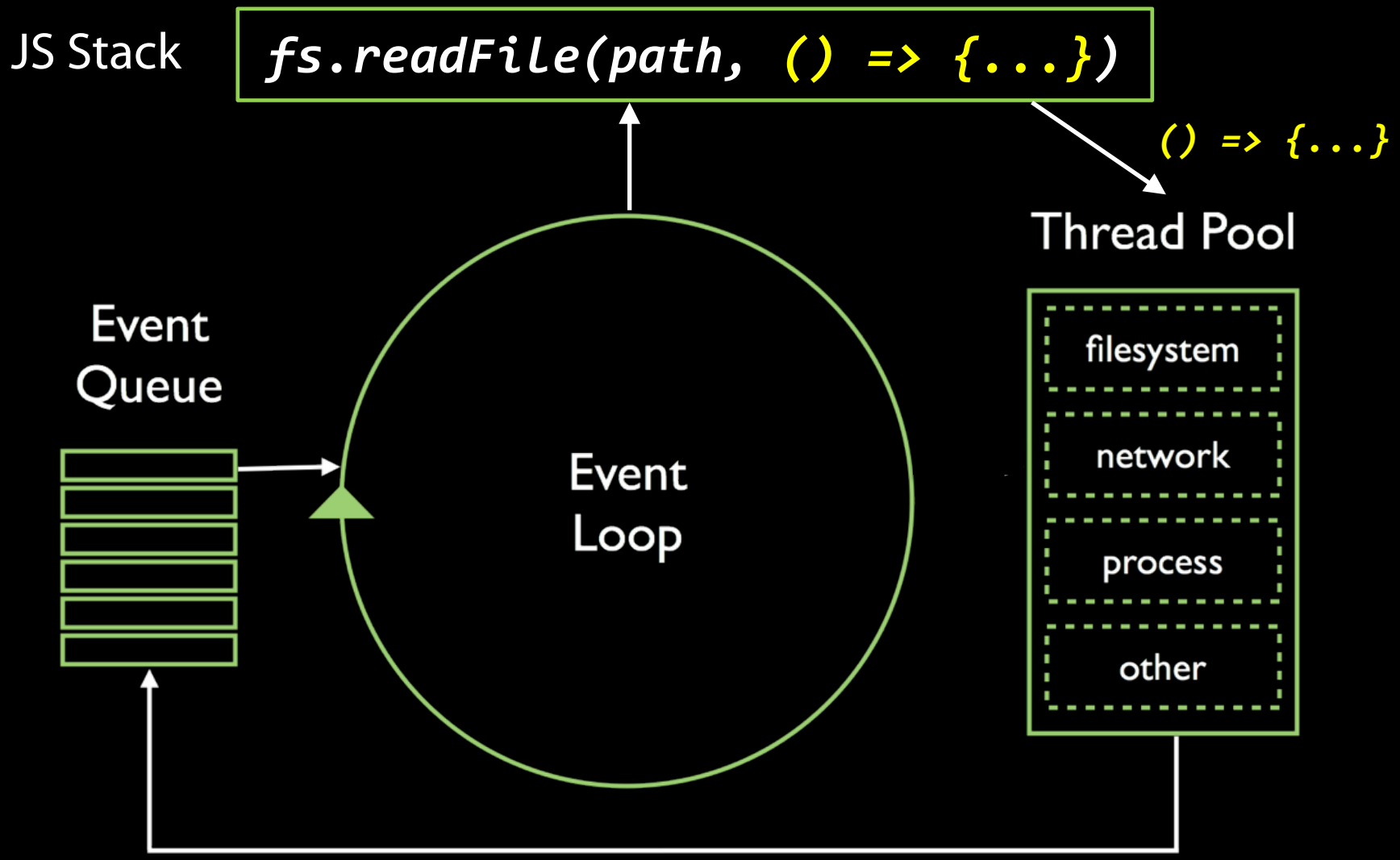
- Node.js: server-side JS runtime, single threaded, async programming model
- Asynchronous API: permits other processing to continue before the transmission has finished.
  - ✓ No time wasted on waiting blocking I/Os
  - ✓ Fast for I/O heavy tasks
  - ✗ Callback hell
  - ✗ Problematic for debugging

Event Loop and Callback



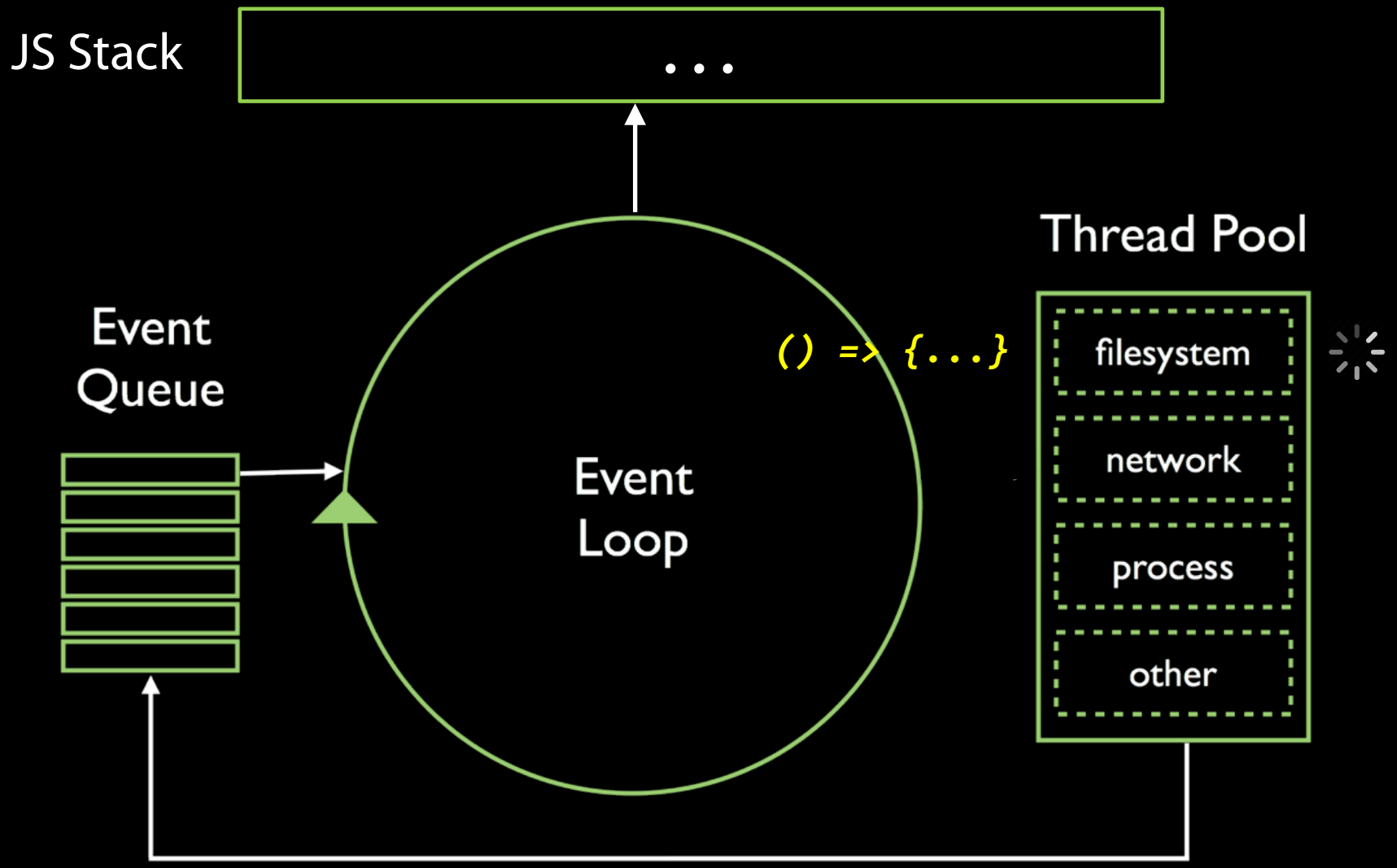
Liang Gong, Berkeley Ph.D. Student, Research Intern in Microsoft Research, Redmond.

Event Loop and Callback



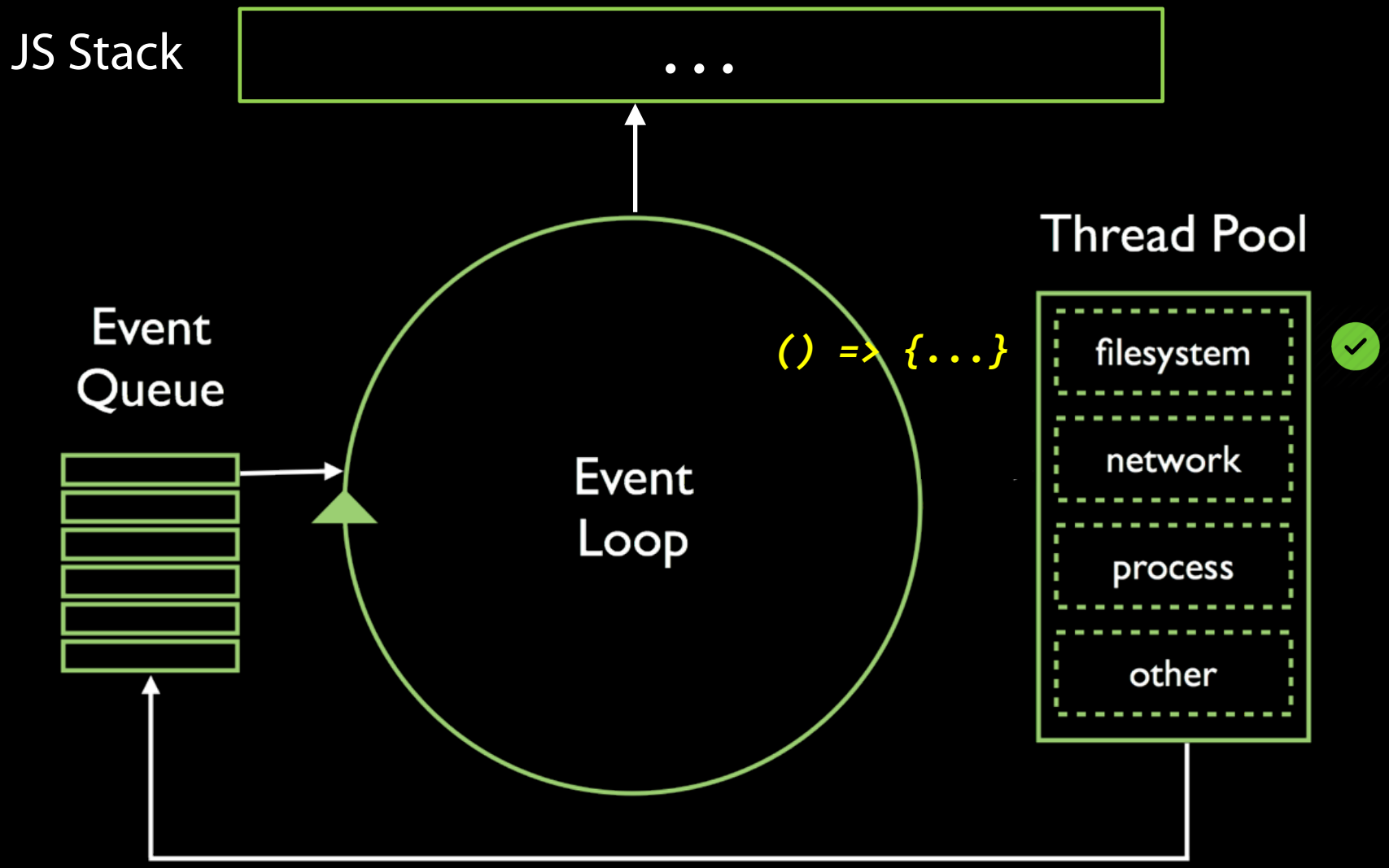
Liang Gong, Berkeley Ph.D. Student, Research Intern in Microsoft Research, Redmond.

Event Loop and Callback



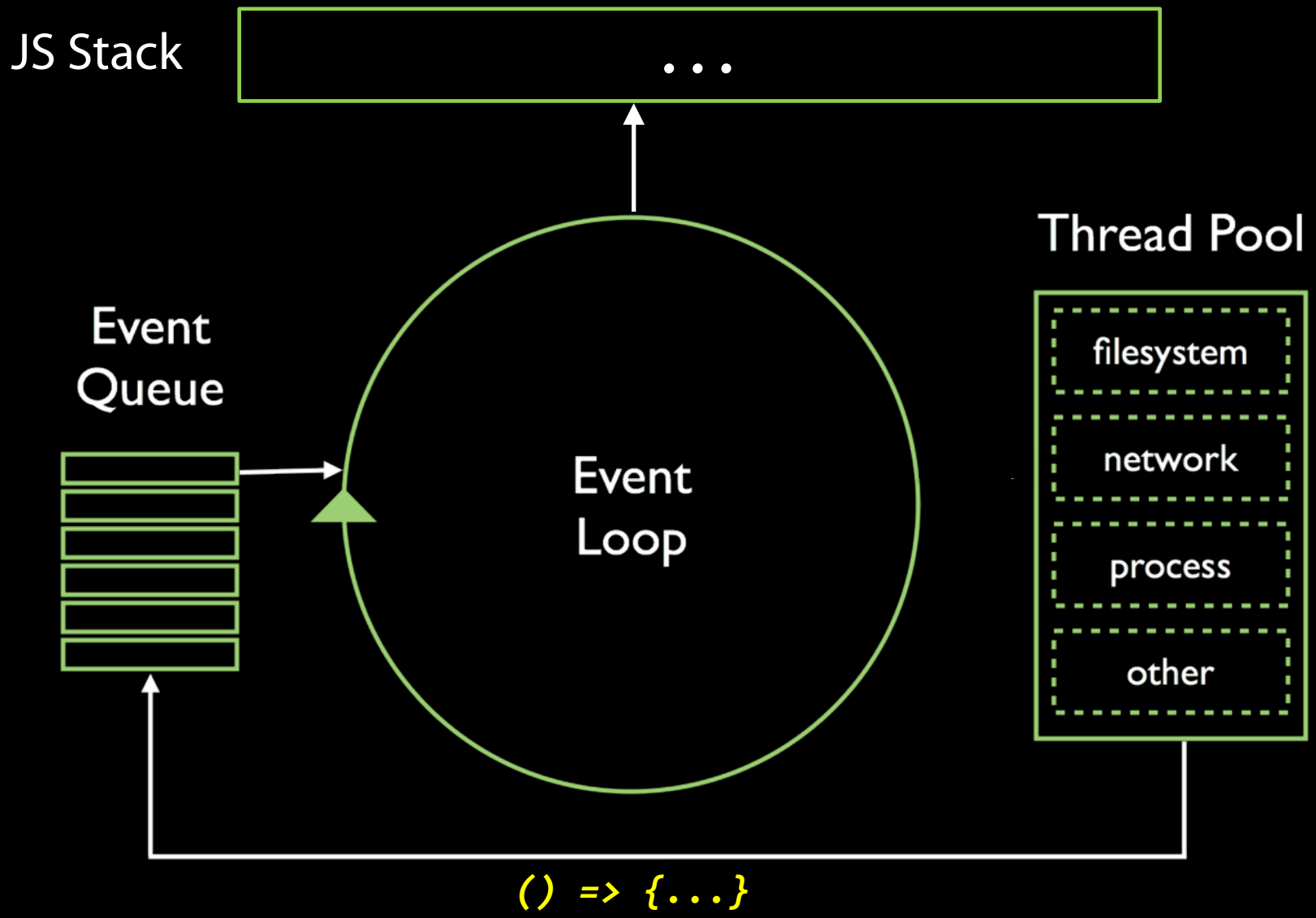
Liang Gong, Berkeley Ph.D. Student, Research Intern in Microsoft Research, Redmond.

Event Loop and Callback



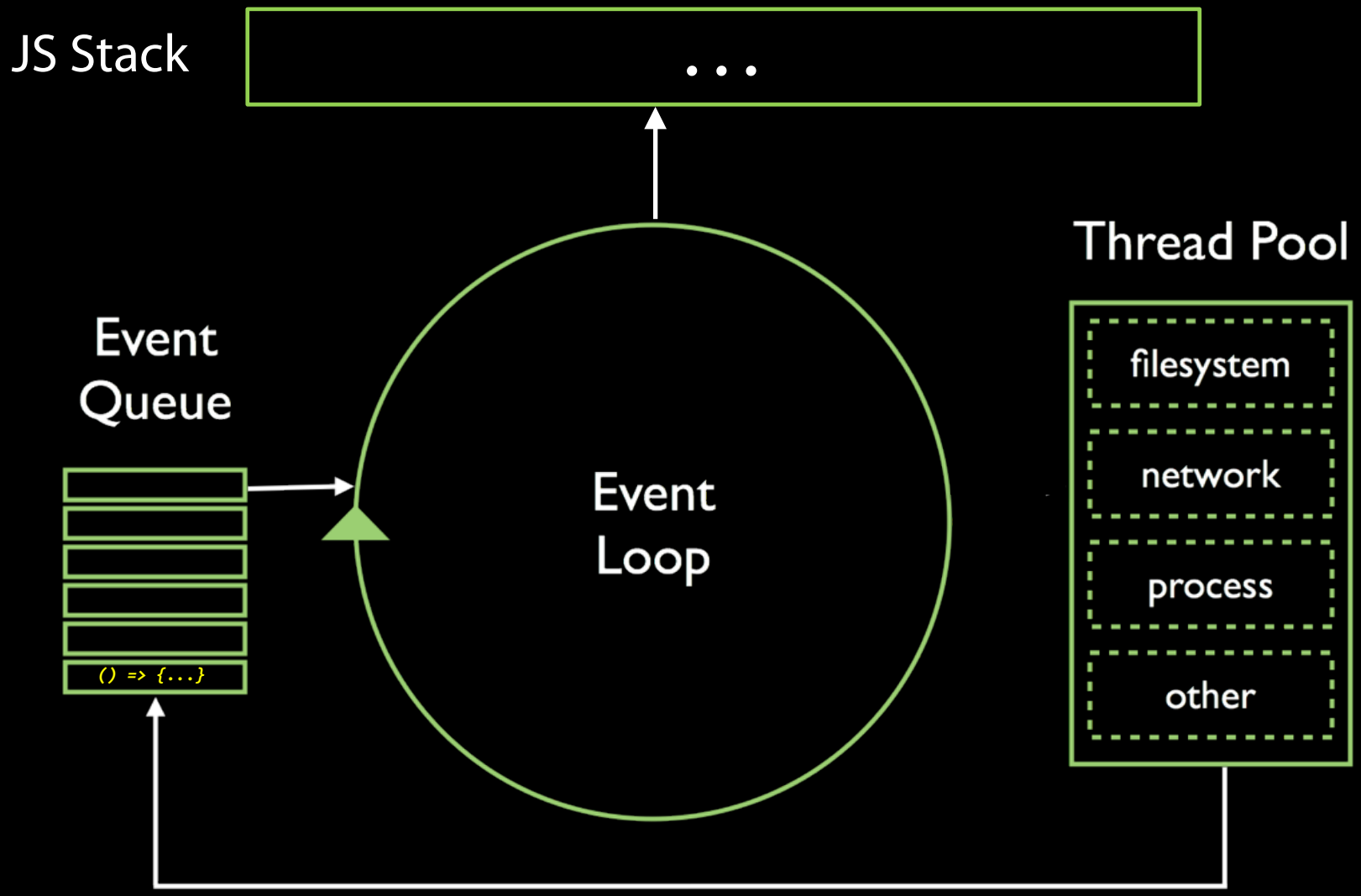
Liang Gong, Berkeley Ph.D. Student, Research Intern in Microsoft Research, Redmond.

Event Loop and Callback



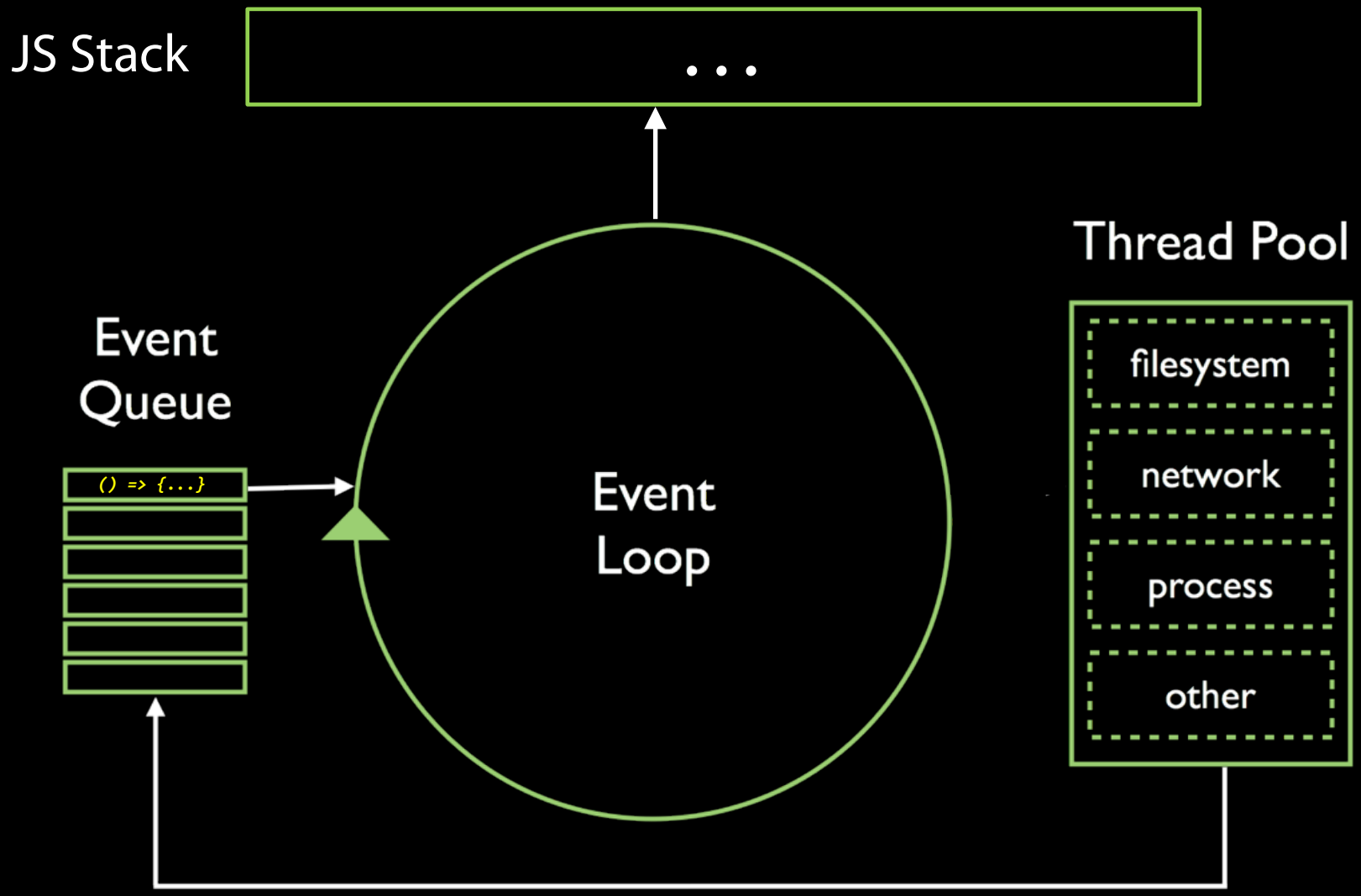
Liang Gong, Berkeley Ph.D. Student, Research Intern in Microsoft Research, Redmond.

Event Loop and Callback



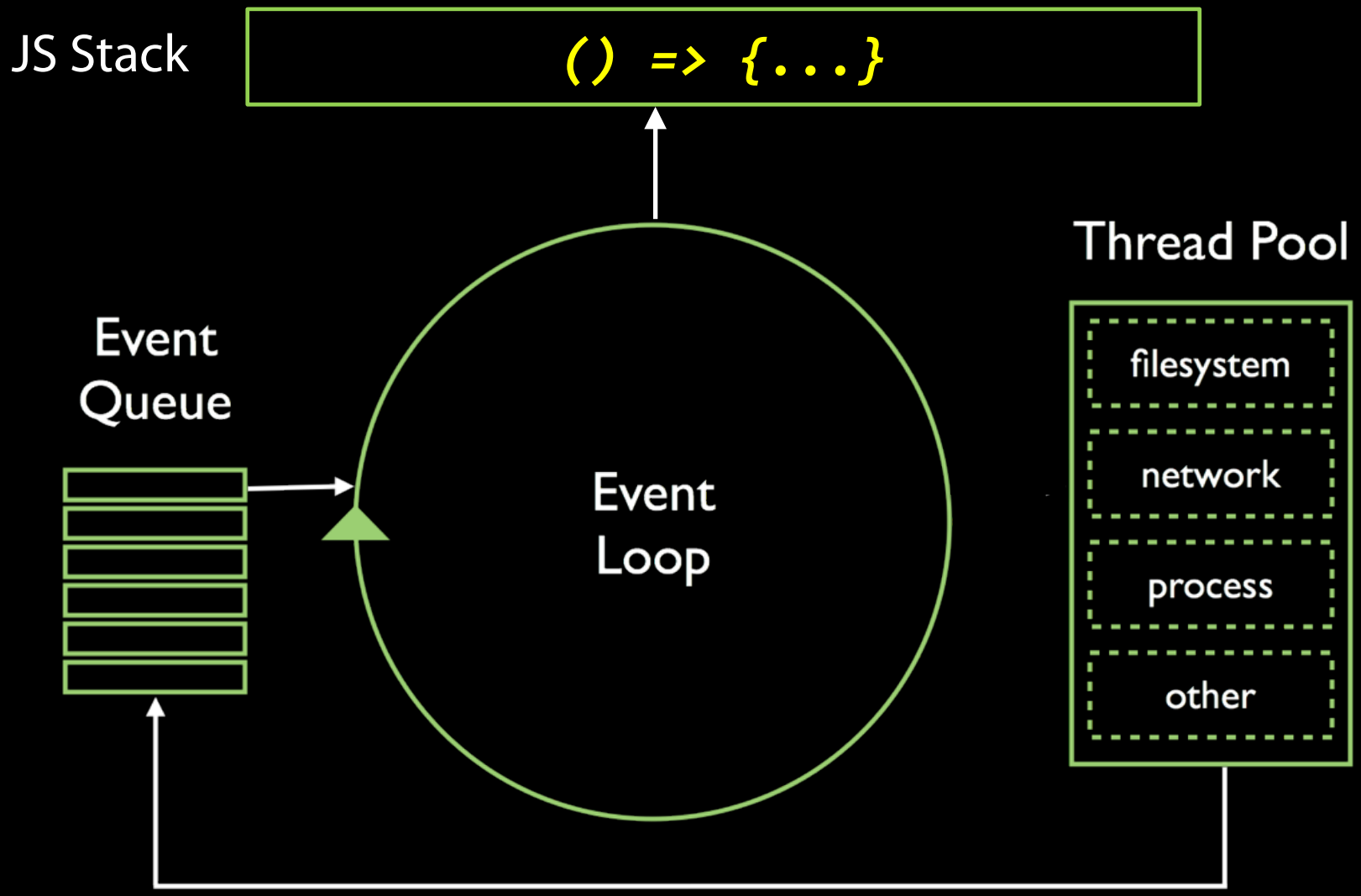
Liang Gong, Berkeley Ph.D. Student, Research Intern in Microsoft Research, Redmond.

Event Loop and Callback



Liang Gong, Berkeley Ph.D. Student, Research Intern in Microsoft Research, Redmond.

Event Loop and Callback



Liang Gong, Berkeley Ph.D. Student, Research Intern in Microsoft Research, Redmond.

Trouble for Debugging

```
1 var fs = require('fs');
2 function callback(err, data) {
3       if (err) throw new Error();
4 }
5 function main() {
6       fs.readFile('file1.txt', callback);
7       fs.readFile('file2.txt', callback);
8 }
9 main();
```

```
> if (err) throw new Error();
>                ^
> Error
>     at <dir>\example.js:3:17
>     at fs.js:207:20
>     at Object.oncomplete (fs.js:107:15)
```