The Case of the Poisoned Event Handler

Weaknesses in the Node.js Event-Driven Architecture

James Davis, Gregor Kildow, Dongyoon Lee

Department of Computer Science

VirginiaTech
Invent the Future

This talk will answer three questions

1. What is an EHP vulnerability?

2. Do they exist in practice?

3. How can we defend against EHP attacks?

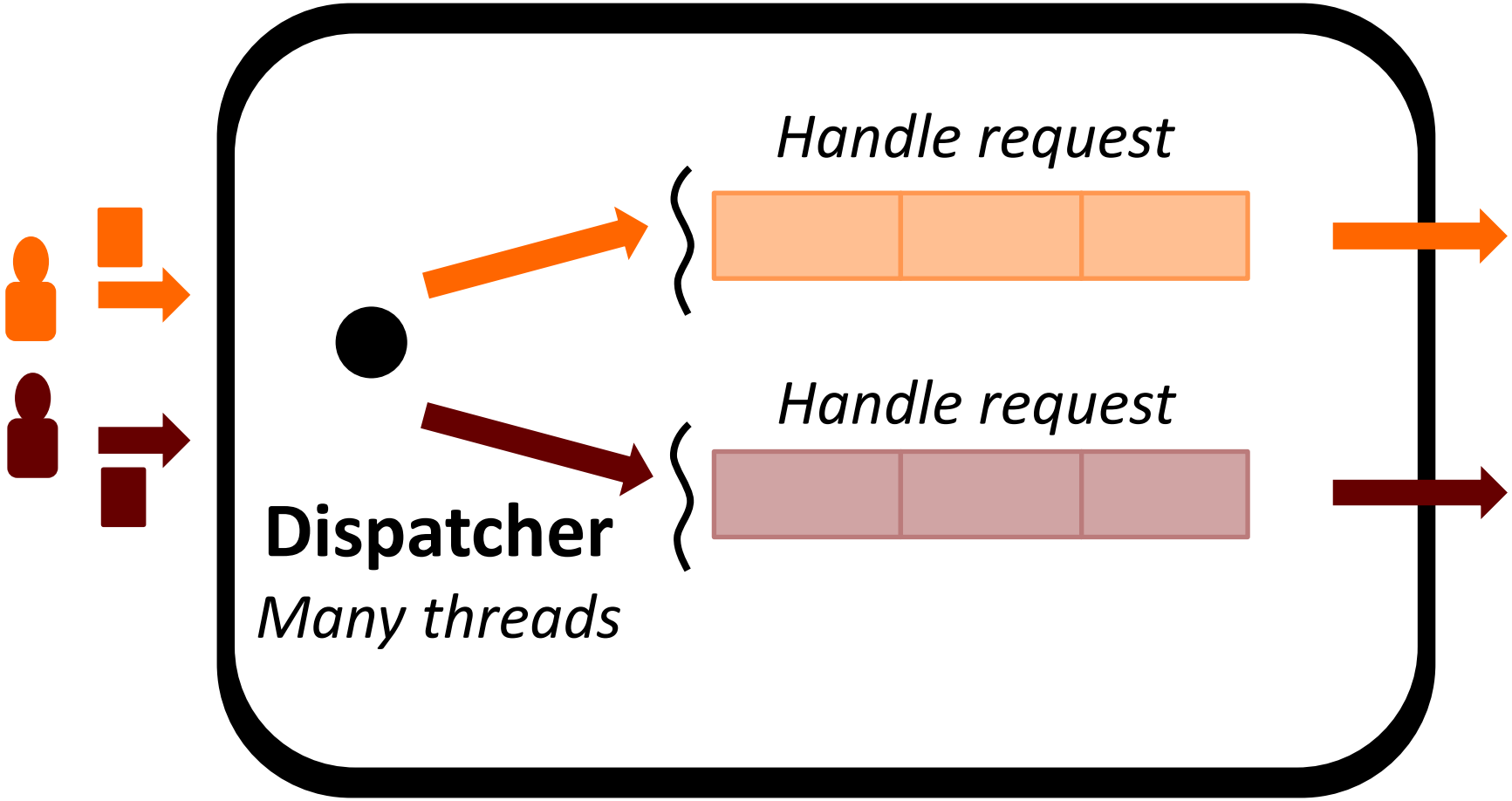
Most significant contribution

Defining EHP vulnerabilities

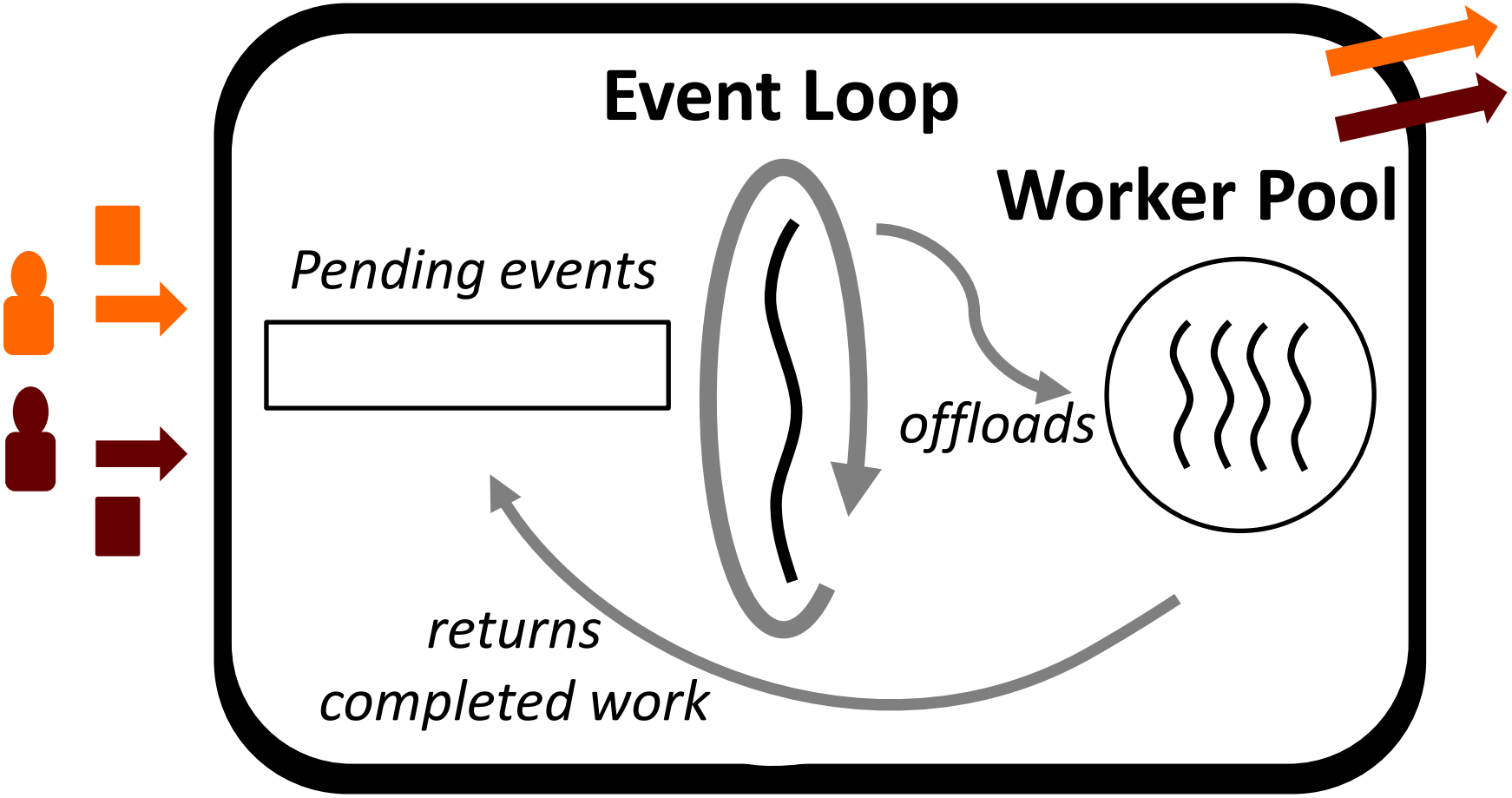
- CPU-bound
- I/O-bound

What is the Event-Driven Architecture?

The One Thread Per Client Architecture (OTPC)



The Event-Driven Architecture (EDA)



The key difference is *multiplexing*

OTPC: Clients get dedicated threads

Preemptive multi-tasking

EDA: Clients share threads (multiplexing)

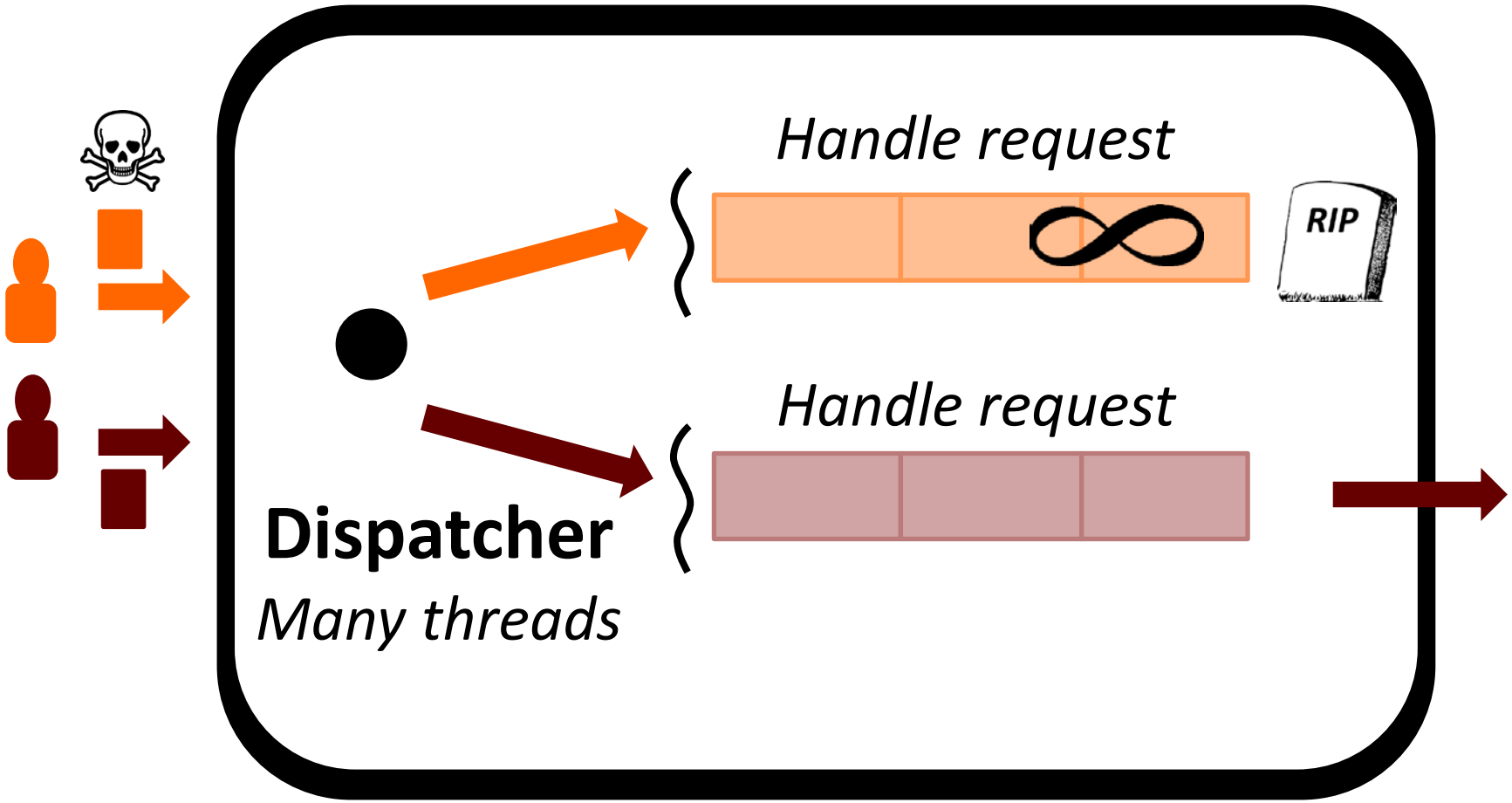
Cooperative multi-tasking

Tradeoff: efficiency vs. reliability

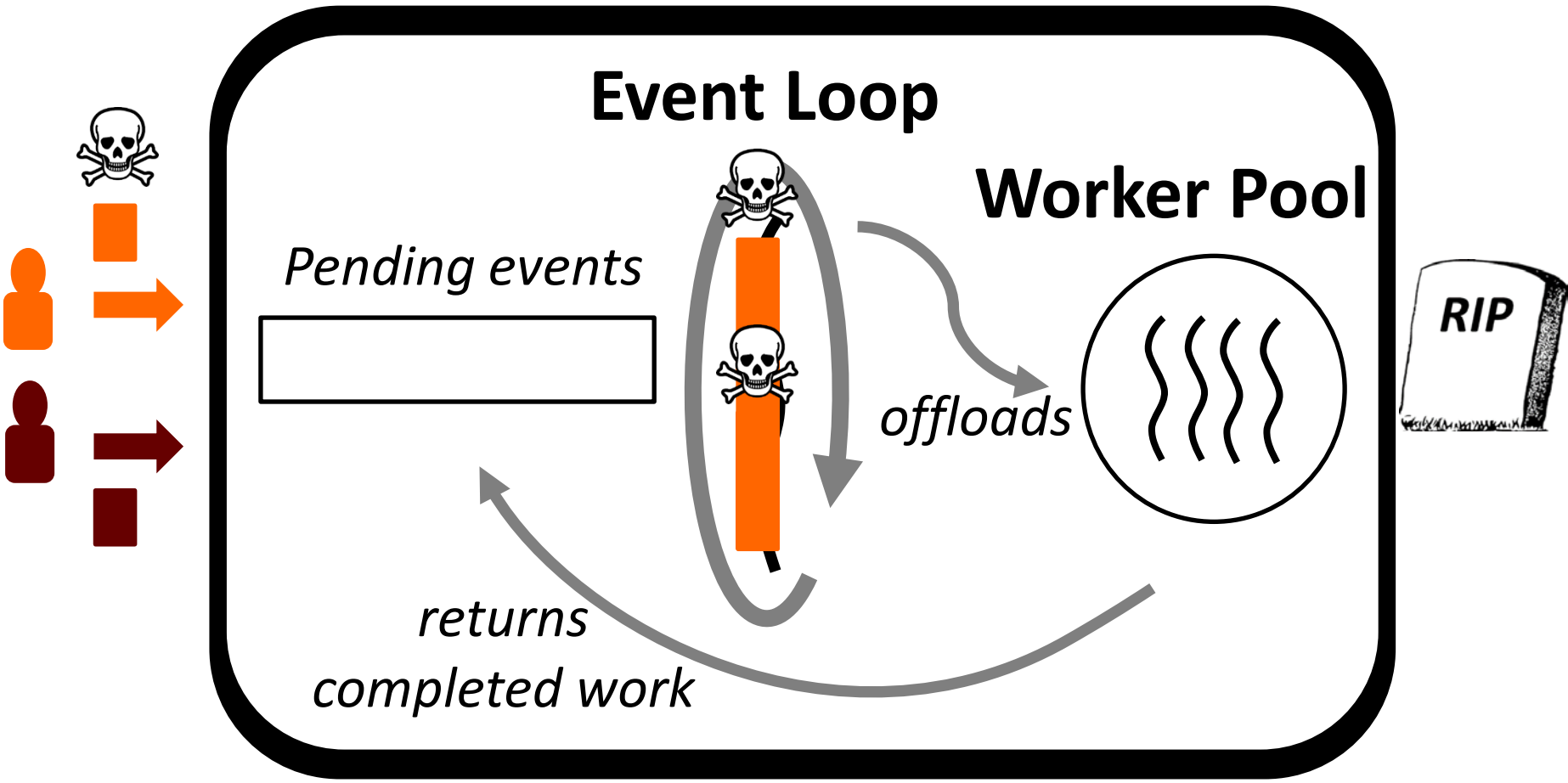
Defining

Event-Handler Poisoning Vulnerabilities

Poisoning the OTPC Architecture



Poisoning the EDA



Defining Event-Handler Poisoning Vulns.

- *Event Handler*s are shared between clients.
- If an event handler becomes delayed,

  → then every pending event becomes delayed.

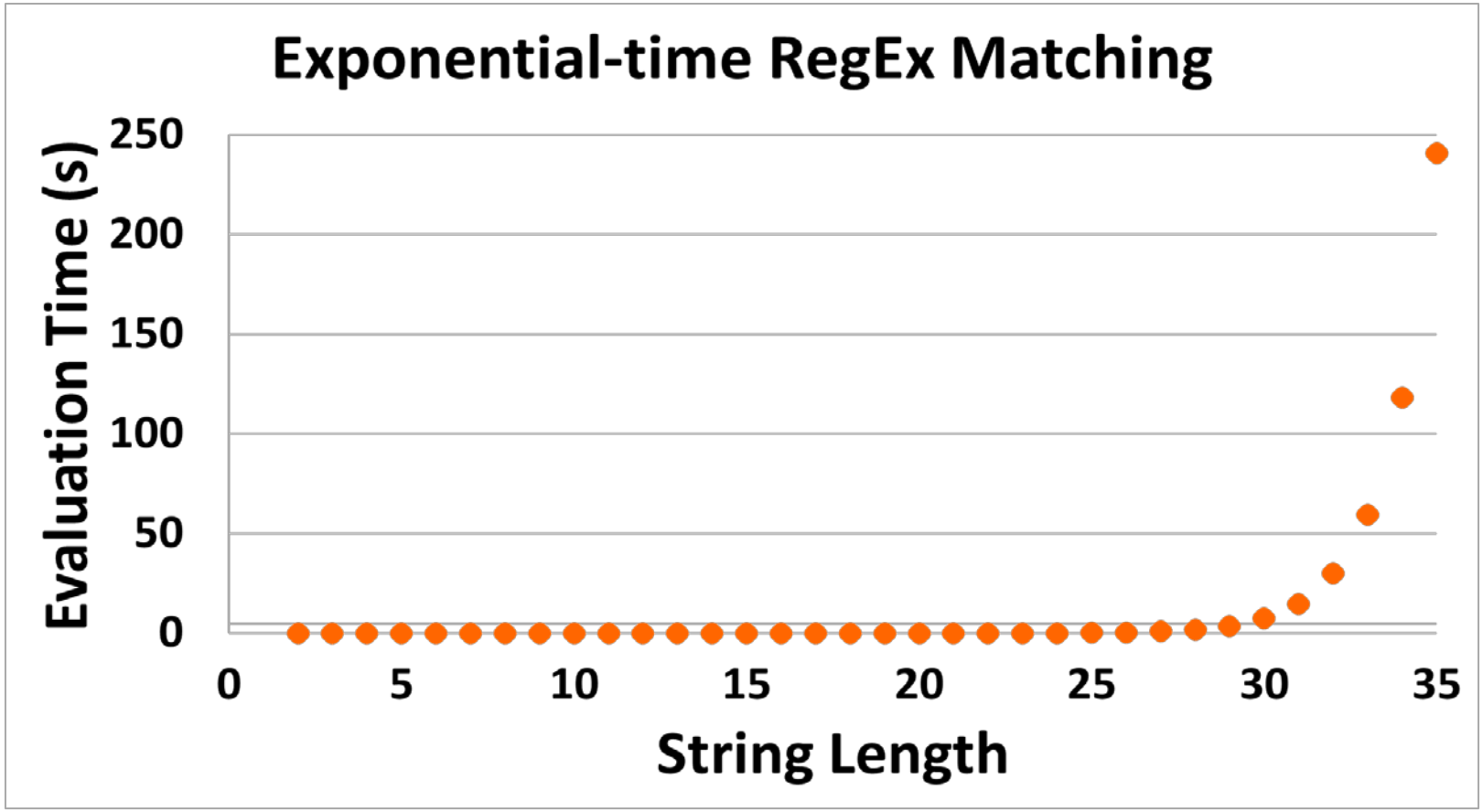
We've come full circle

DoS vulnerabilities in a single-threaded server

CPU-bound EHP vulnerabilities

Server: /(a+){40}/

Client: “a…a<”



I/O-bound EHP vulnerabilities

Server: readFileSync(*user_file*)

Client: “/dev/zero”

Do EHP vulnerabilities occur in EDA-based applications?

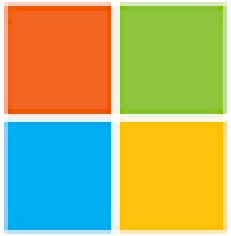
And why should I care?

Node.js: A Server-Side JavaScript EDA Framework

- “Full stack JavaScript” (Ryan Dahl, 2009)

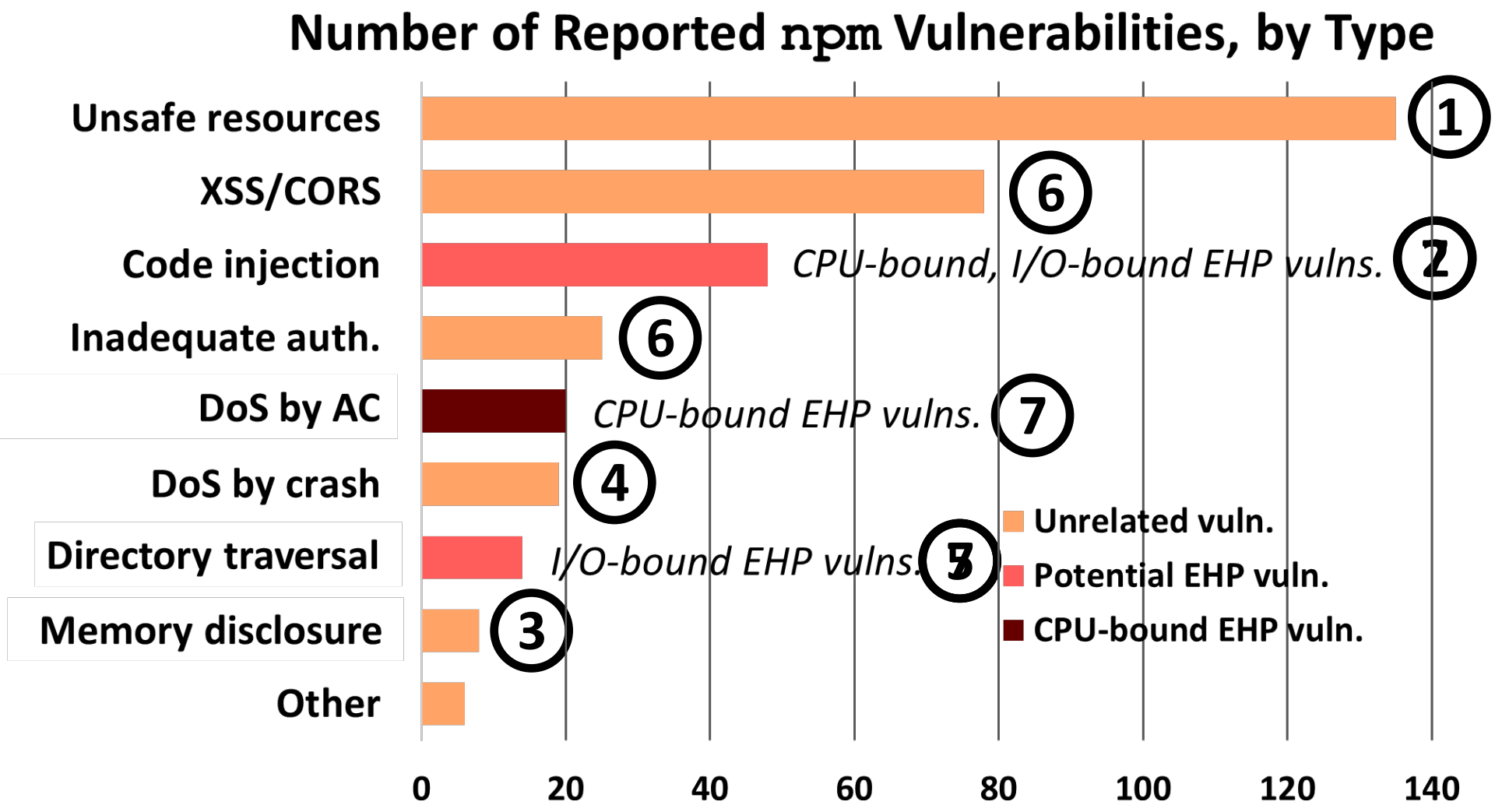
- 3.5M+ developers (April 2016)
- 450K+ modules (March 2017)
- 2B+ module downloads/week (March 2017)





EHP vulnerabilities in Node.js applications

Vulnerabilities: a mix of normal and unique



Known EHP Vulns.

CPU-bound

- *A too-expensive synchronous API*, x1
- *ReDoS*: vuln regex, x19
- *Code injection*: “while(1);”, x48

I/O-bound

- *Directory traversal*: “read from /dev/null”, x14
- *Code injection*: “cat /dev/null”, x48

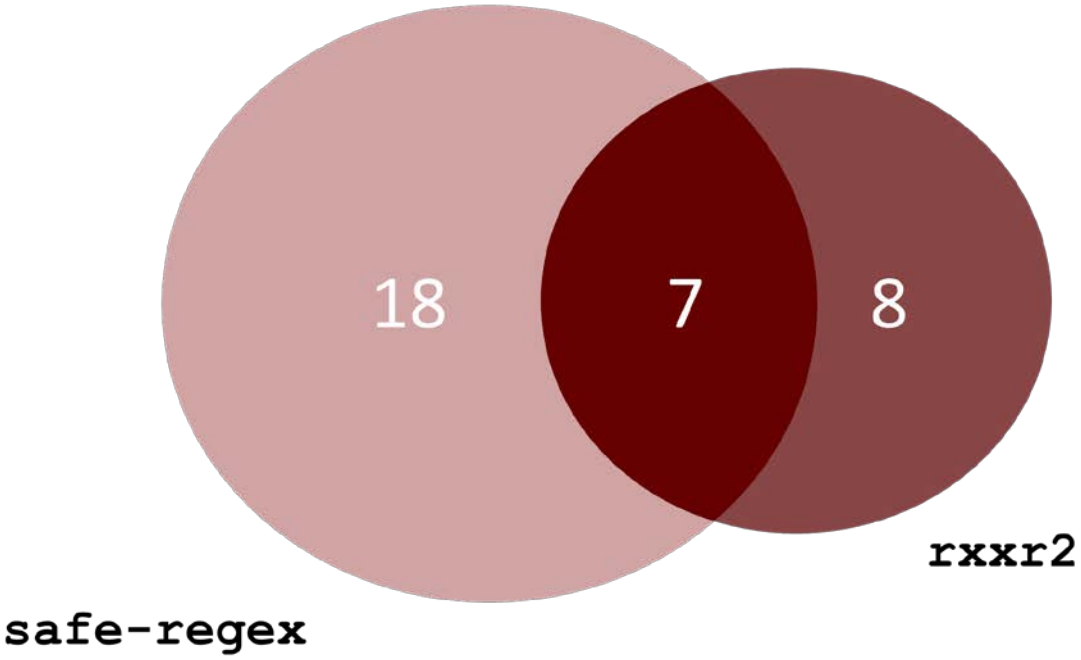
Identifying

Event-Handler Poisoning Vulnerabilities

Case study: ReDoS Oracles

How do you identify a vulnerable RegEx?

1. Extract RegExes from vulnerability reports → 44 possibilities
2. Semantics-preserving transform
3. Test with `safe-regex` and `rxxr2`



ReDoS in the wild

Are there vulnerable RegExes in the wild?

1. Instrument Node.js to emit RegExes as they are defined
2. Run test suites of 71 “most starred” npm modules
3. 450K declarations → 16K unique RegExes

Oracle	# vuln.
`safe-regex`	700+
`rxxr2`	20

Preventing

Event-Handler Poisoning Vulnerabilities

C-WCET Partitioning

- EHP: “because complexity is not O(1)”
- Constant-time Worst Case Execution Time
  - **Cooperative**
  - **Predictable**

EDA systems → ***responsiveness*** ← Real-time systems

Achieving C-WCET Partitioning

Truncated*

- Restrict input, e.g. maximum length

General

- Identify algorithm's kernel(s)
- Perform constant amount of work in each pass
- Repeat until complete
- Store ongoing results

C-WCET Addition: Truncated

```
var MAX_NUMS = 10;
function add (nums, callback):
  var sum = 0
  for i in 0 .. MAX_NUMS:
    sum += nums[i]
  callback(sum)
```

C-WCET Addition: General

```
function add (nums, callback):
  var sum = 0
  function add_one (ix):
    sum += nums[ix]
    if (done):
      callback(sum)
    else:
      do_later(add_one, ix+1)
  add_one(0)
```

Cool, what next?

Potential research directions

1. TimeoutExceptions
2. Hybrid RegEx engine
3. C-WCET Node.js

Stuff I didn't talk about

- Comparing popularity of different EDA frameworks
- Discussion of I/O-bound EHP vulnerabilities
- Does anyone actually do C-WCET partitioning?

Closing thoughts

1. Event-handler poisoning vulnerabilities are real.

2. Open questions in detection and defense.

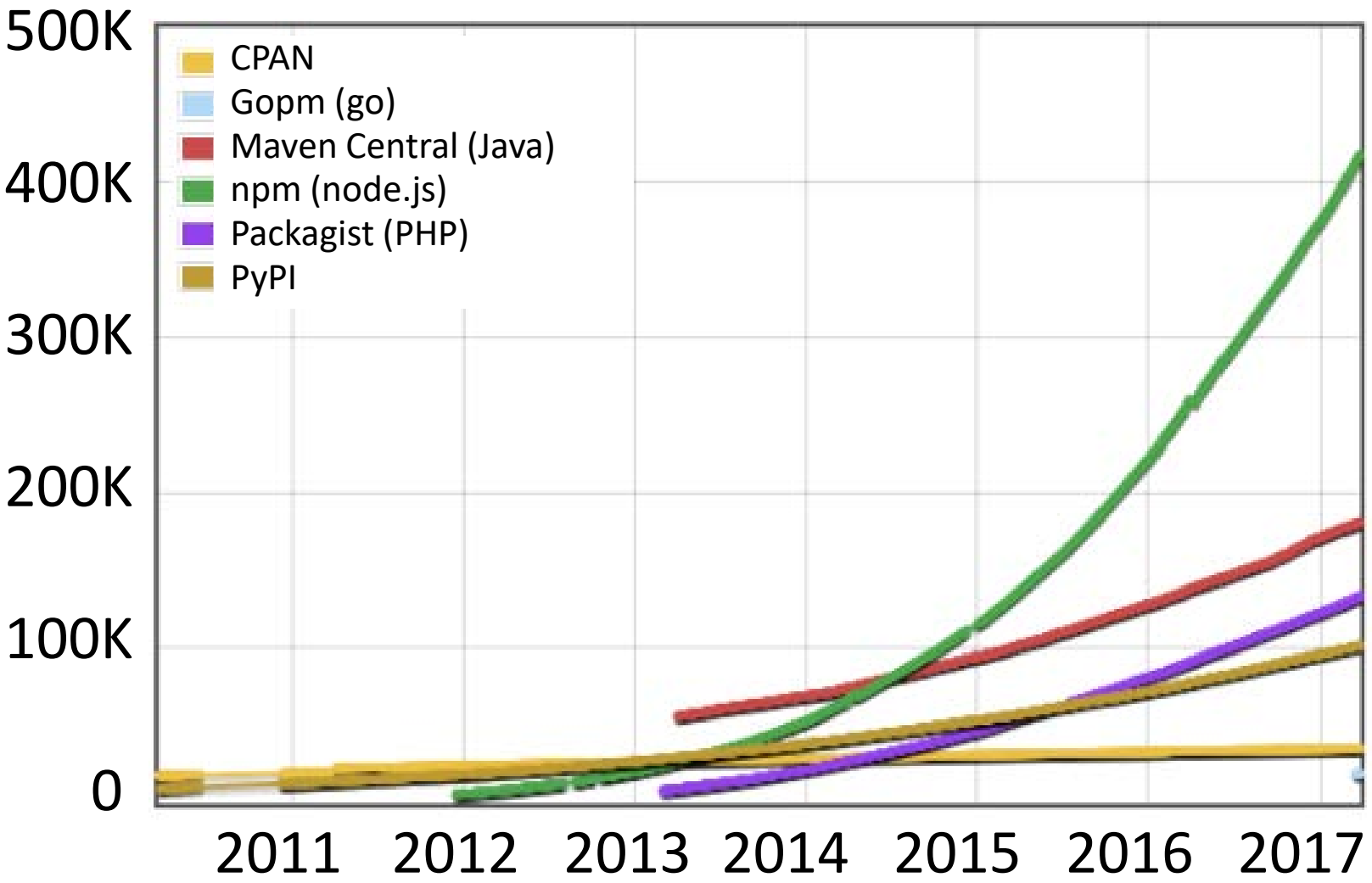
Additional material

Node.js has the most framework developers

Framework	Contributors	Commits	Releases
Node.js	1176	15666	399
Libuv	247	3643	183
EventMachine	113	1141	27
Reactor	56	4249	42
Twisted	35	20751	38

Node.js has the most modules

Module counts for different languages



www.modulecounts.com, 27 March 2017

“Eternal I/O Attacks” → C-WCET partitioning

Server code	Evil input	Effect
`readFileSync(f)`	**/dev/zero**	Poisons event loop (until ENOMEM)
`readFile(f, cb)`	**/dev/zero**	Poisons worker pool (until ENOMEM)
`readFileStream(f, cb)`	**/dev/random**	Poisons worker pool
`read or write`	**FIFO**	Poisons handler

C-WCET in the wild

Do npm module developers use C-WCET partitioning?

1. First 20 modules from each of:

   "string", "string manipulation", "math" "algorithm"
2. Read documentation
3. Read implementation

Community recommendations

1. Improve vulnerability reporting
2. Improve module documentation
3. Use multiple RegEx oracles

Fuller list of potential research directions

1. TimeoutExceptions
2. Hybrid RegEx engine
3. An accurate RegEx oracle
4. C-WCET-partitioning
5. C-WCET Node.js