A Sense of Time for JavaScript and Node.js

First-Class Timeouts as a Cure for Event Handler Poisoning

James C. Davis
Eric R. Williamson
Dongyoon Lee

COMPUTER SCIENCE
VIRGINIA TECH

Contributions

Attack: Event Handler Poisoning

- Definition
- Analysis

Detect + recover: First-Class Timeouts

- Concept
- Prototype

Engagement with the Node.js community

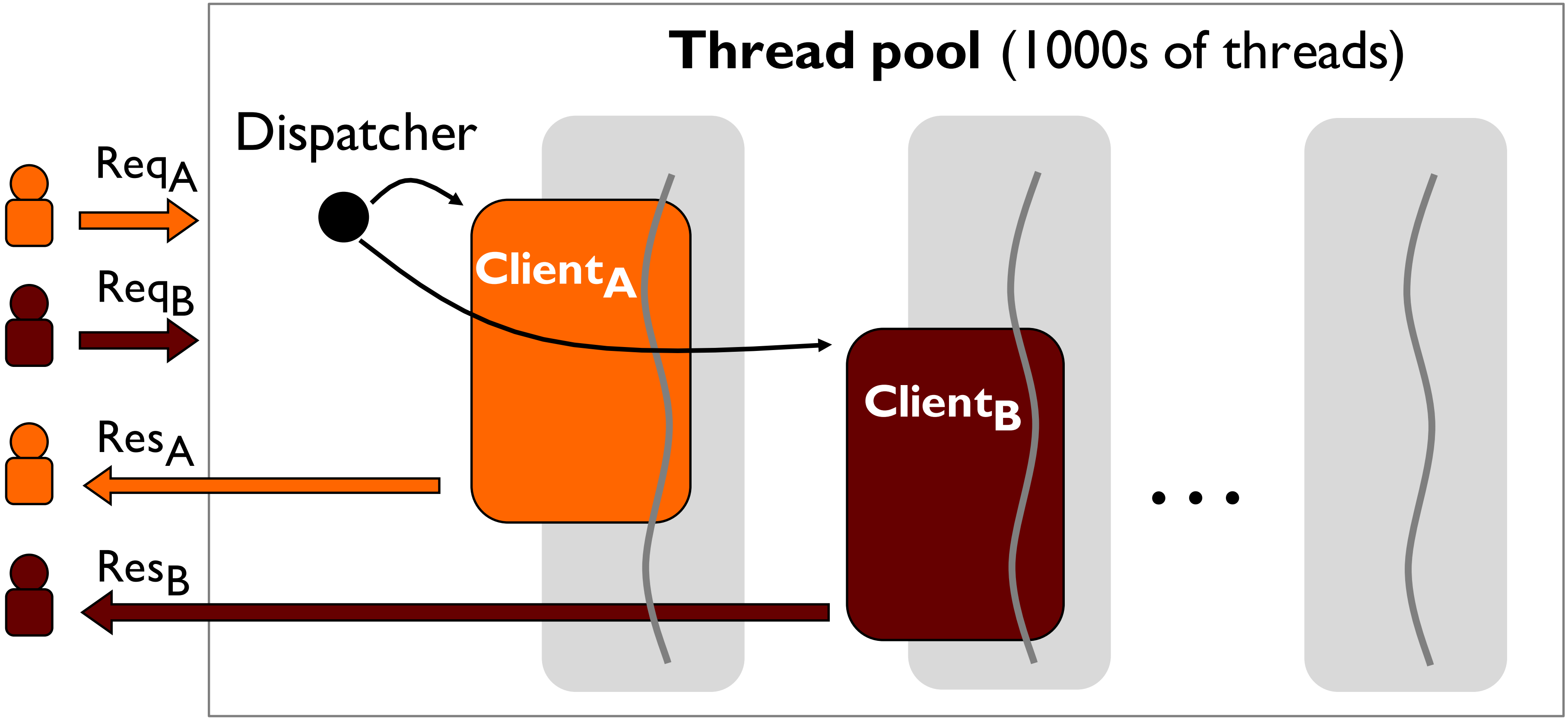
- Guide
- Core APIs: Documentation and repairs

Node.js: A JS framework for web services

7M+ developers (2017)	**2x** YoY
760K+ modules (Aug. 2018)	**2x** YoY
24B+ module downloads/month (July 2018)	**12x** YoY

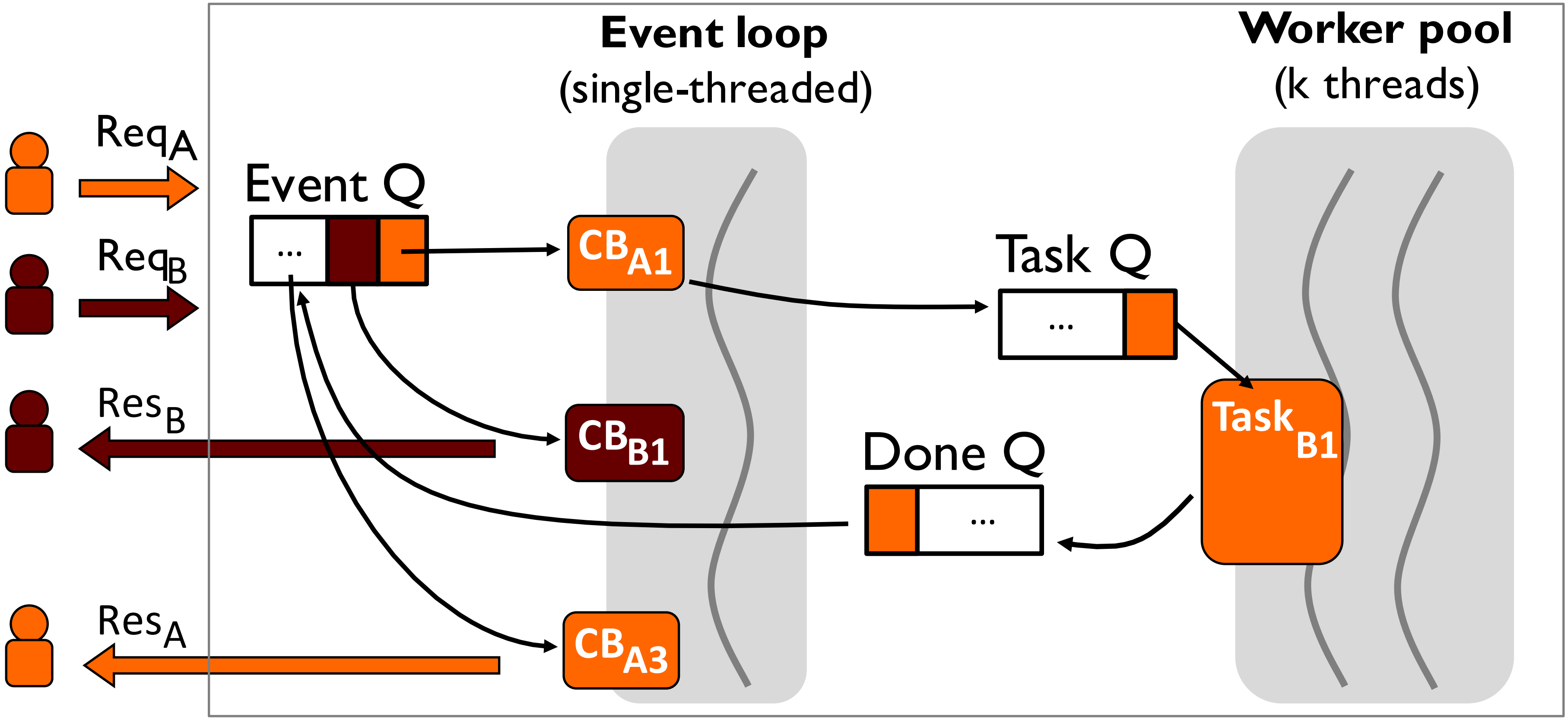
Web server architectures

One Thread per Client Architecture (OTPCA)



- Each client gets its own worker thread
- Multithreading enables scalability
- Example: Apache

Event-Driven Architecture (EDA)



- Clients multiplexed; shared threads reduce threading overhead
- Cooperative multitasking via (1) Partitioning and (2) Offloading
- Example: node JS

Server architecture dictates programming style

OTPCA

Preemptive multi-tasking

Synchronous

```
def serveFile(req):
  cont =
    readFile(req.file)
  z = zip(cont)
  e = encrypt(z)
  return e
```

EDA

Cooperative multi-tasking

Asynchronous

```
def serveFile(req):
  cont = await
    readFile(req.file)
  z = await zip(cont)
  e = await encrypt(z)
  return e
```

Event Handler Poisoning Attacks
(EHP)

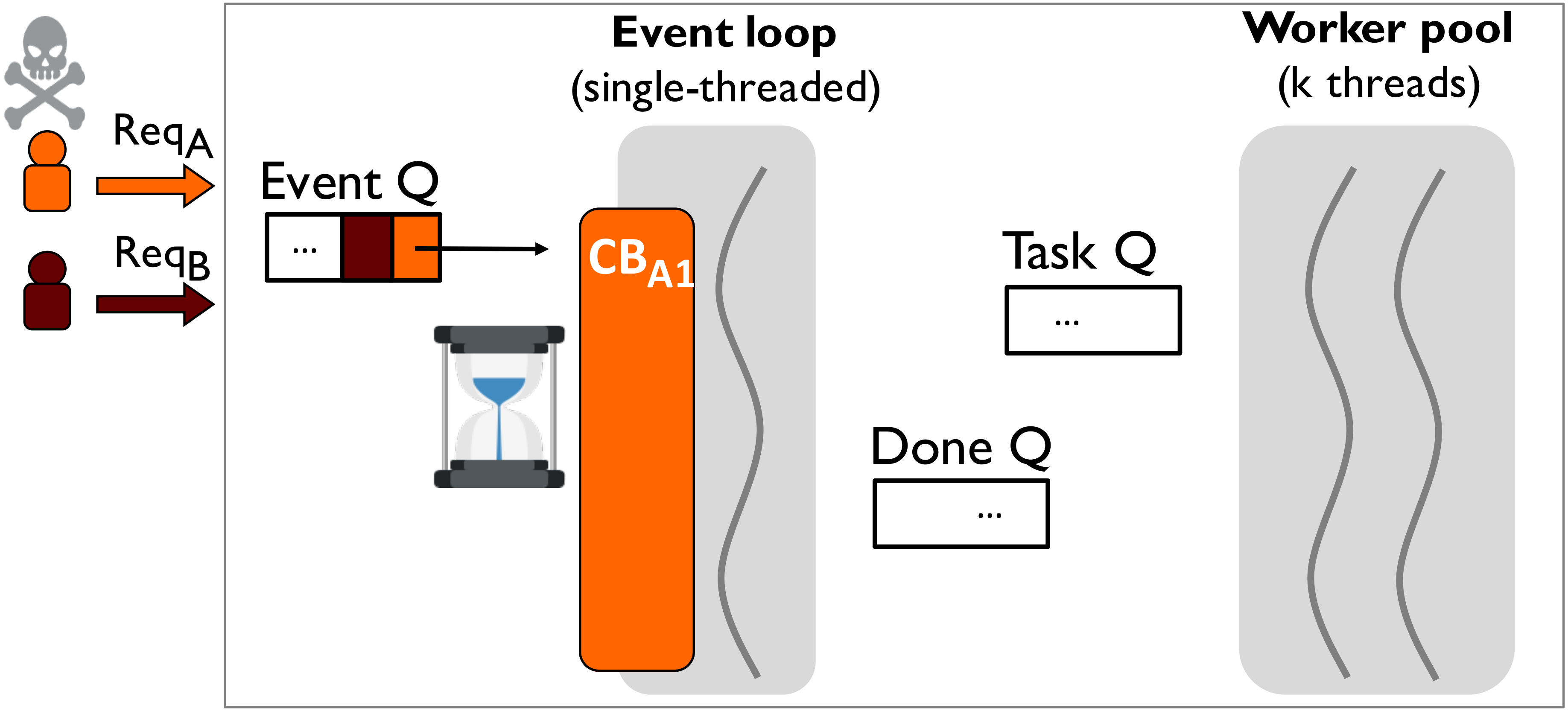
The EDA gains **efficiency**, loses **isolation**

Architecture	Threads	# Threads	Multi-tasking
OTPCA	Dedicated	Thousands	Preemptive
EDA	Shared	Tens	Cooperative

Event Handlers = limited resource

Exhaust resource → DoS

Behavior during EHP attack on the Event Loop



- Event loop is poisoned
- Throughput drops to zero

On the worker pool:
k malicious requests

Vulnerable server

ReDoS

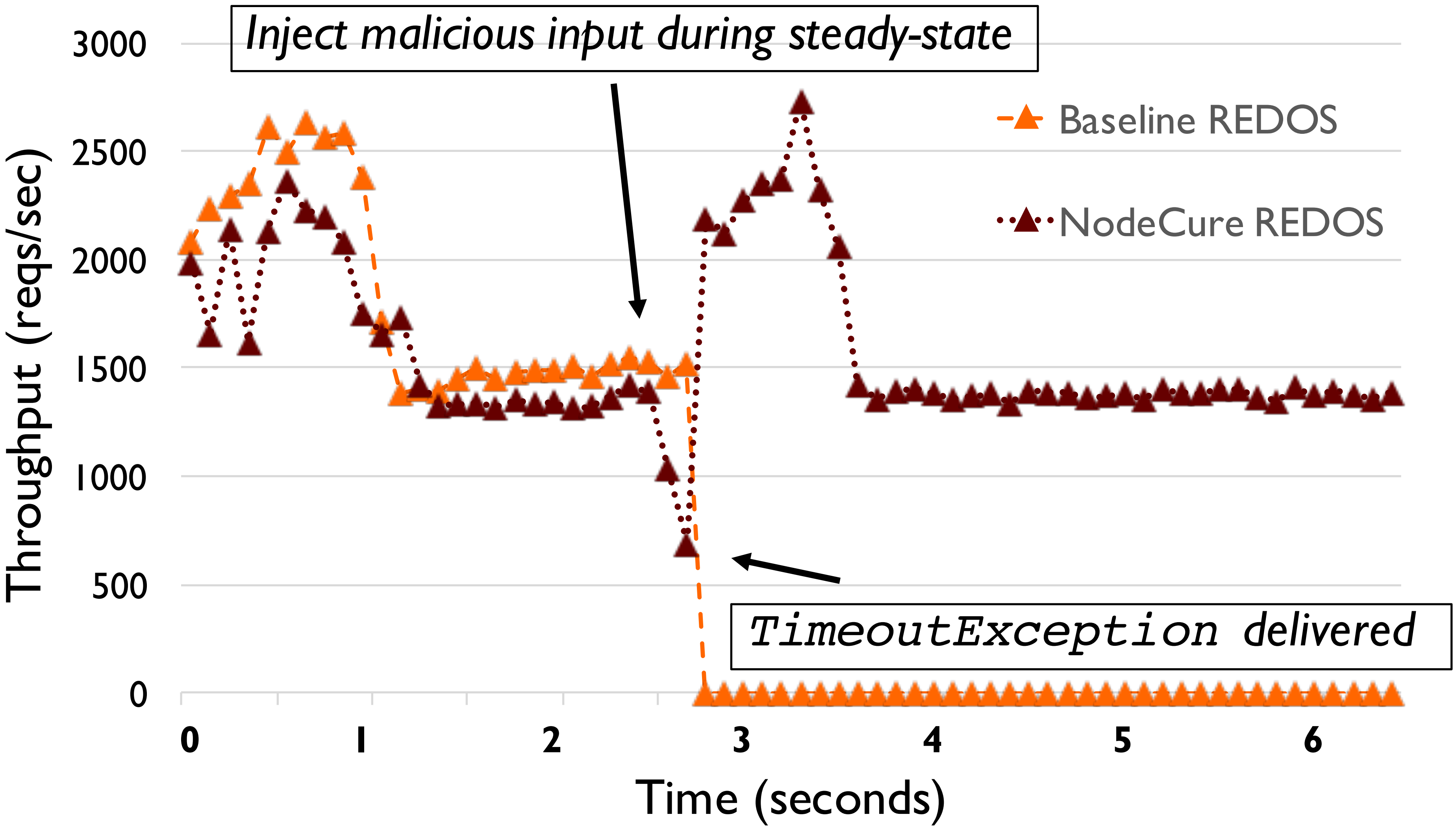
Super-linear regex

```
def serveFile(name):
  if name.match(/(\/.+)+$/):
    readFile(name)
    .then(
        ...
    )
```

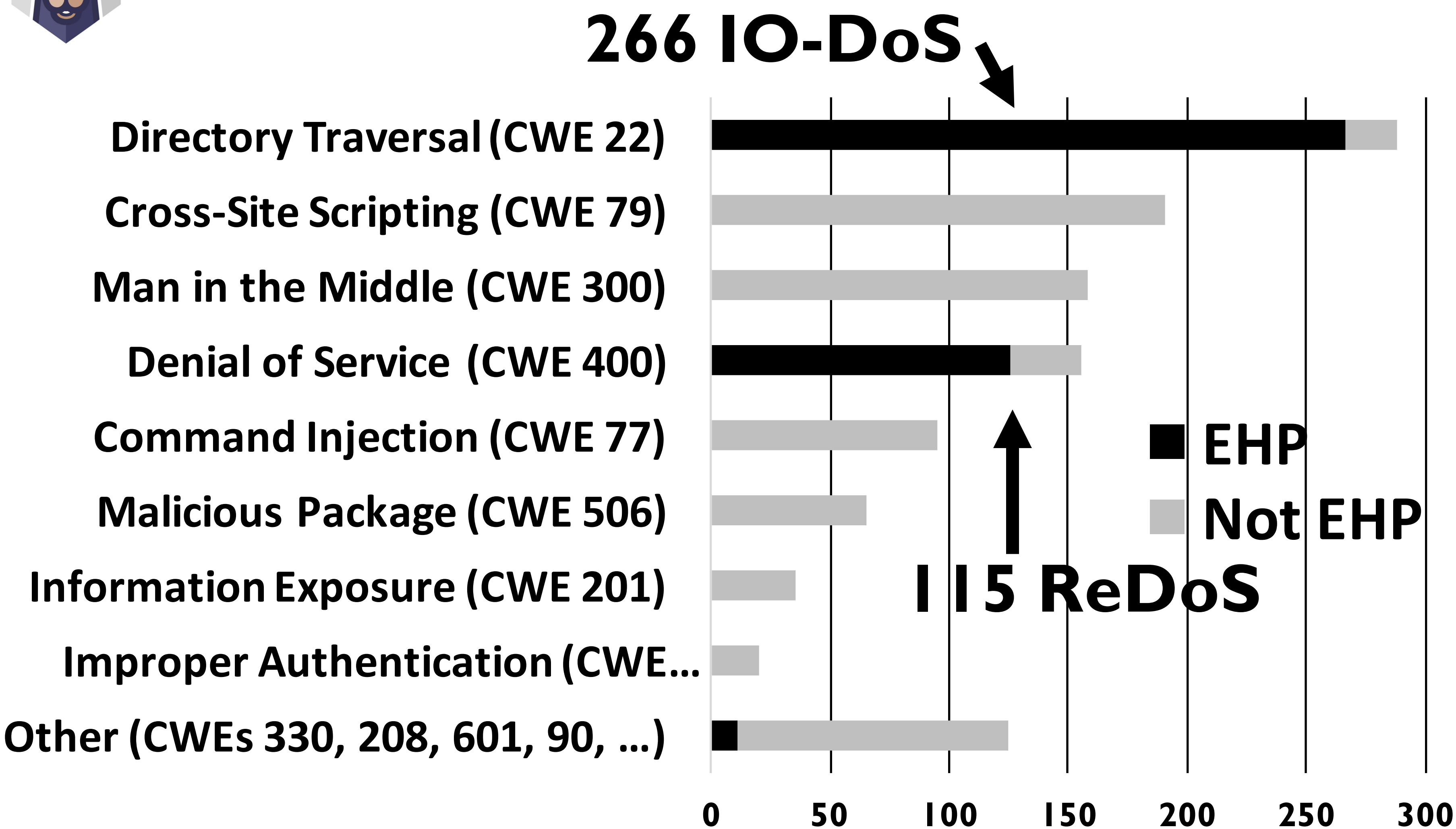
IO-DoS

Arbitrary file read

ReDoS-based EHP attack



35% of NPM vulnerabilities enable EHP



What should we do about EHP?

Naïve 1: Restart the server

Idea

- Heartbeat on each Event Handler
- If any heartbeats fail, restart the server

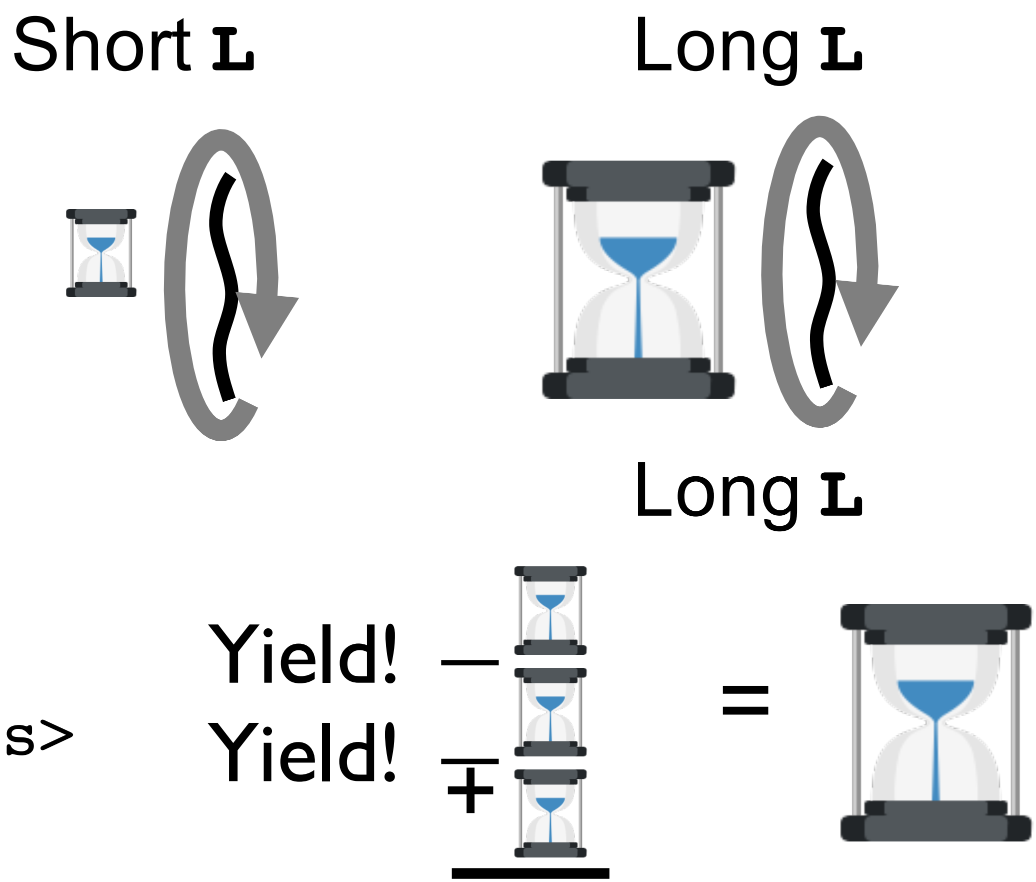
Problems

- Every connected client gets DoS'd
- Repeat attacks

Naïve 2: Prevent through partitioning

```
def sum(L):
   s = 0
   for n in L:
      s += n
   return s
```

```
async def sum(L):
   s = 0
   until done:
      s += <10 numbers>
      yield
   return s
```



Partitioning is partial and ad hoc

Only protects code under the application dev.'s control

- Not **modules**
- Not **framework**
- Not **language**

Good for algorithms – but how to meaningfully partition I/O?

Ongoing maintenance burden

Our proposed solution:

First-Class Timeouts

First-Class Timeouts

Analogy

Buffer overflow	→	Out of bounds exception
EDA "time overflow"	→	Timeout exception

Idea

Time-aware cooperative multi-tasking

- Bound the **synchronous time** of every Callback and Task
- Deliver a `TimeoutException` if this bound is exceeded

Analysis

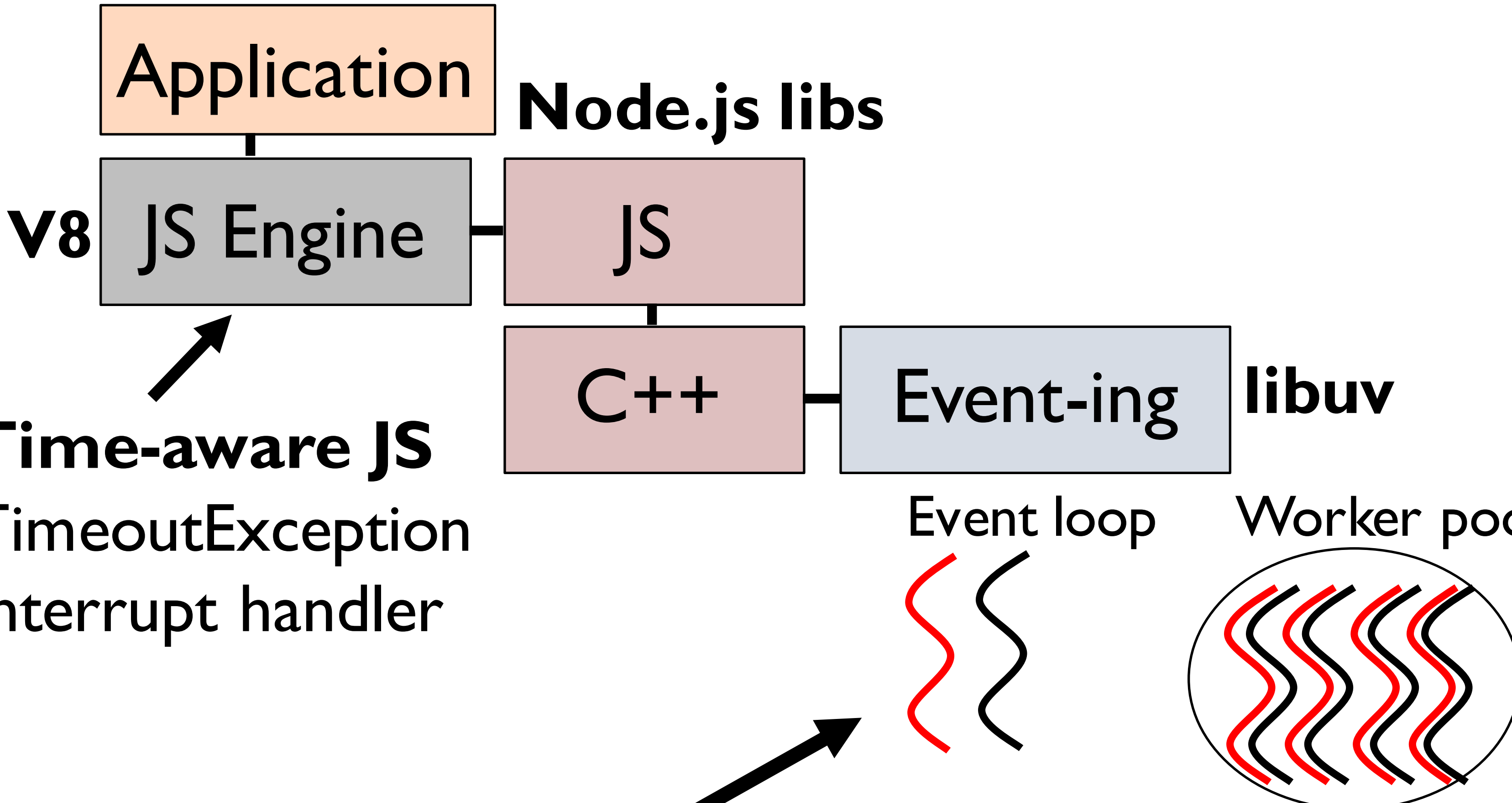
- **Soundly** defeats EHP attacks
- **Straightforward refactoring**: `try-catch` in Promise chains
- **Non-destructive**: Existing clients unharmed

Node.cure
Design and Evaluation

Desired behavior

Event Handler	Old behavior	New behavior
Event Loop	Unbounded execution	Throw `TimeoutException`
Worker Pool	"	Return `TimeoutException`

Adding first-class timeouts to Node.js



Time-aware JS

TimeoutException

Interrupt handler

Time-aware Event Loop

Timeout Watchdog

Monitor CB entry/exit

Set T.E. interrupt

Time-aware Worker Pool

Managers watch for timeouts

Disposable Workers

Node.cure prototype

- **Built** on Node.js v8.8.1 (LTS)
  - 4 KLoC across 50 files
- **Compatible**
  - Passes Node.js core test suite*
- **Available** on

Security guarantees

- Every vulnerable **Language** and **Framework** API is safe
  - Applications built with these APIs are safe, too!
- Passes our EHP test suite
  - All vulnerable Node.js APIs
  - Including all used in the npm vulnerabilities

However

- **Detect**: Must choose timeout thresholds (*Goldilocks problem*)
- **Respond**: Tight threshold or blacklisting

Performance penalty

Micro-benchmarks

Component	Overhead
New interrupt	0%
Instr. CBs	1.01-2.4 x
I/O buffers	1.3 x

Macro-benchmarks (summary)

App. type	Overhead
Server	0-2 %
Utility	0-8 %
Middleware	6-24 %

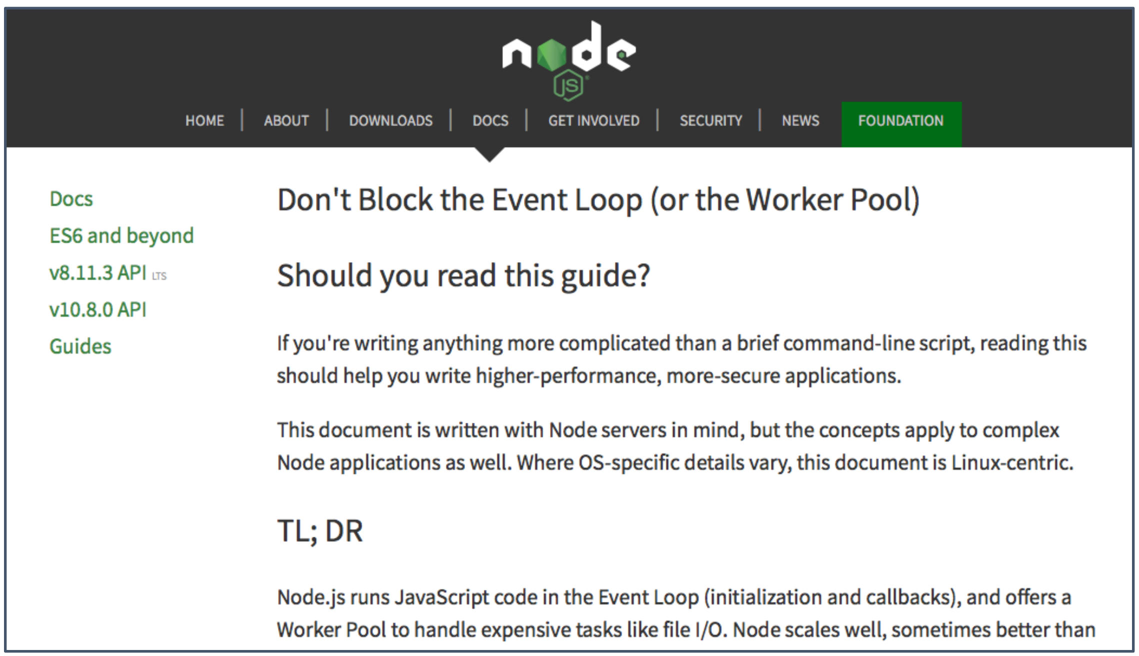
Community Engagement

Guide on nodejs.org

Reviews Node.js architecture

EHP attacks + examples

Advice about npm module safety



Changes to Node.js core

Documentation

readFile

randomBytes
randomFill

spawn

Code

Closing Remarks

The EDA has an EHP problem.

First-class timeouts can cure it.

We:

- **Defined** an attack
- **Demonstrated** its presence in the wild
- **Designed** and **prototyped** a defense
- **Disseminated** to the practitioner community

Thank you for your attention!

COMPUTER SCIENCE
VIRGINIA TECH

Bonus Material

Choosing a timeout

- The tighter the timeout, the less effective the EHP attack
- Loose timeouts → blacklist attackers
  - No DDoS (threat model)
  - Blacklisting is relatively easy with First-Class Timeouts because the `TimeoutException` is delivered in the context of the malicious request

Programming with First-Class Timeouts

- Choose timeout – minimize CB variance during tuning
  - Goldilocks problem
- Add error handling – a global exception handler and per-request handlers
- New first-class asynchronous primitives like async/await and Promises make this possible
- We only support global timeouts but could refine thresholds on a per-CB and per-Task basis

Various ideas towards EHP-safety

- Heartbeat
- Partitioning
- First-class timeouts

Within the EDA paradigm

- Larger worker pool
- Preemptible callbacks and tasks
- Speculative concurrent execution
- Serverless

Dedicating resources to each client: OTPCA

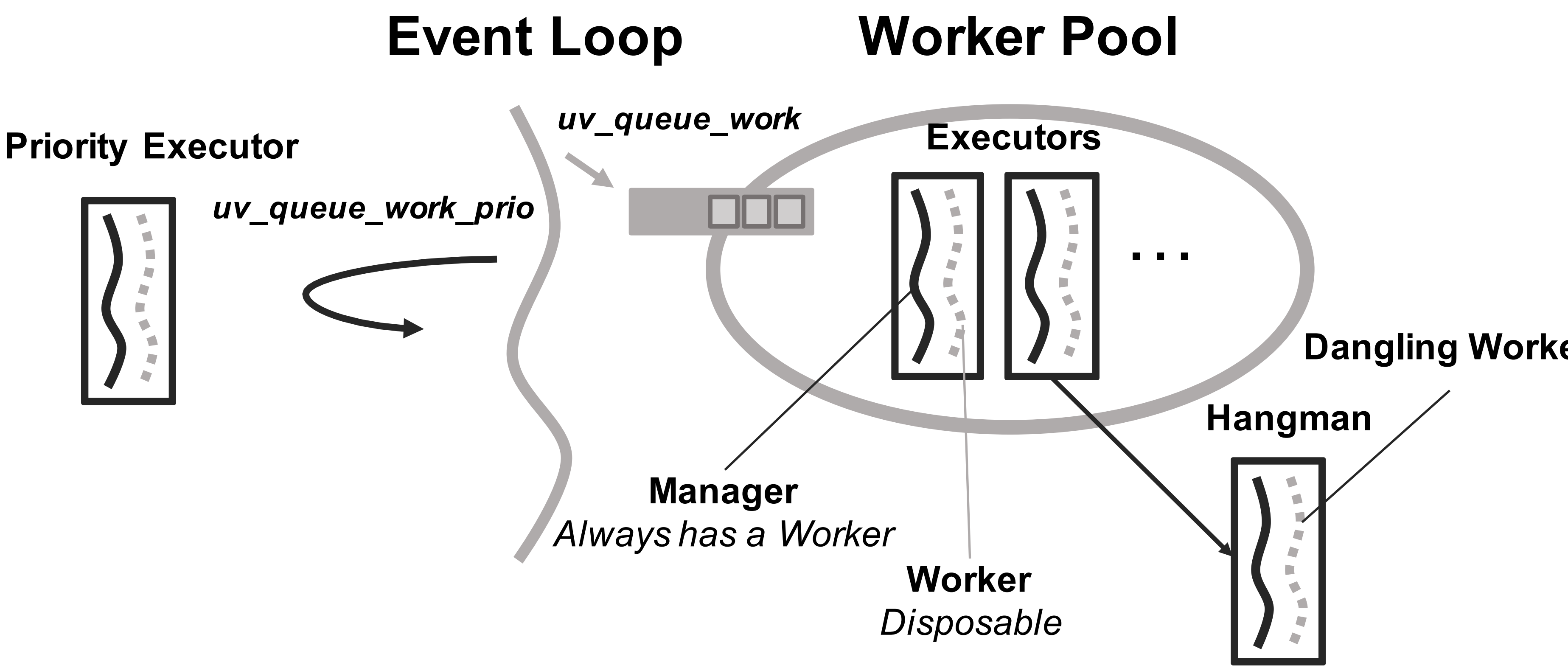
Threat model

- Attacker can trigger worst-case behavior
- No DDoS

Thus:

- Include EHP, a problem unique to EDA
- Exclude DDoS, a general problem for problem web servers

More details on time-aware Event Handlers



Implementation details

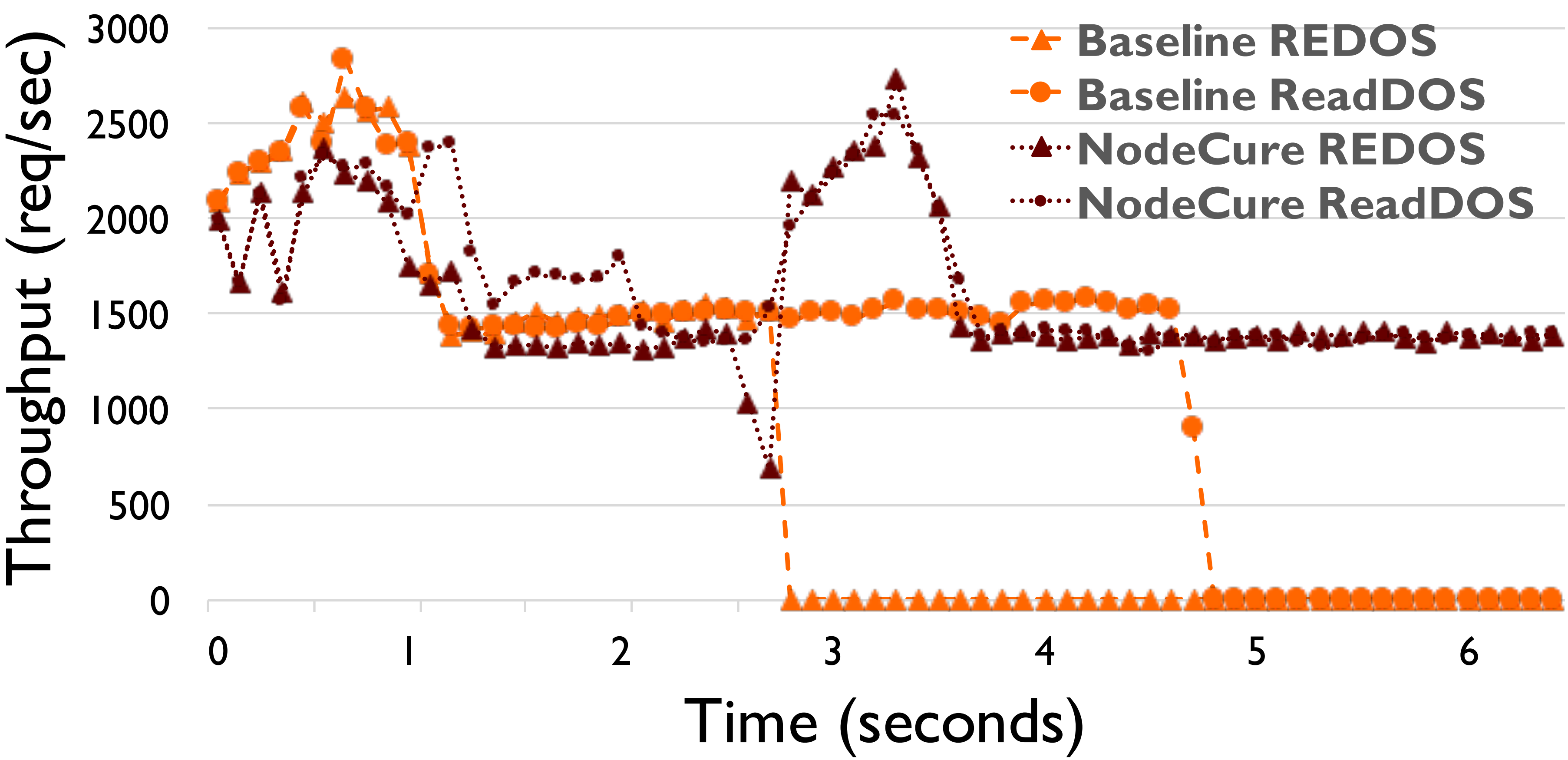
Layer	Changes
Language	• Add `TimeoutException` • Add interrupt
Framework	• Timeout Watchdog • Handle T.E. from async APIs • Offload sync. APIs • Time-aware C++ add-ons
Application	• Handle T.E.

C++ add-ons

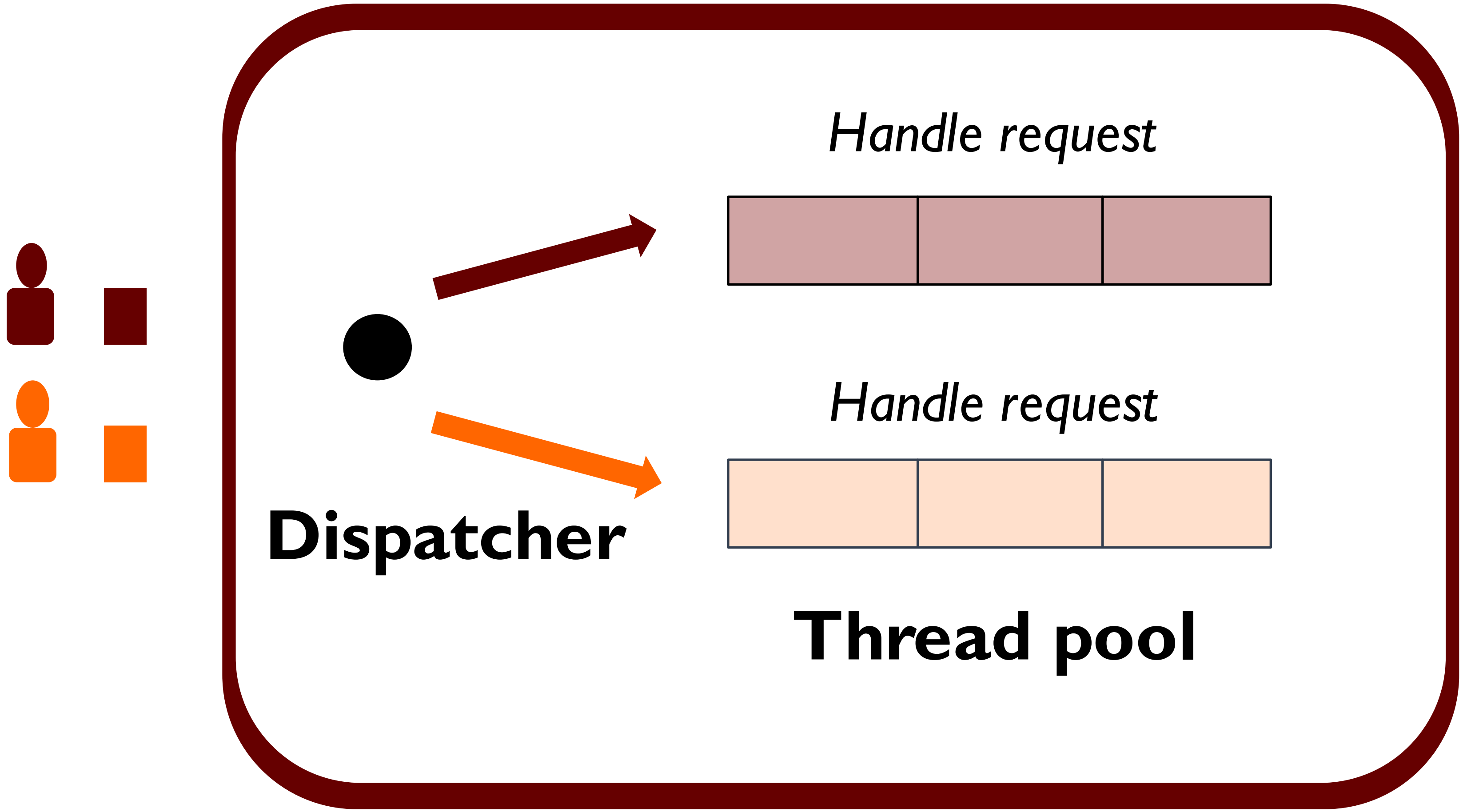
- Node.js applications can contain:
  - Pure JavaScript
  - C++ add-ons
    - e.g. for performance or using systems libraries
- Application-defined C++ add-ons are unprotected by F.C.T
  - Must be made time-aware, similar to how we made Node.js's own C++ bindings time-aware
  - Only 0.7% of npm modules have C++ add-ons

Experimental slides

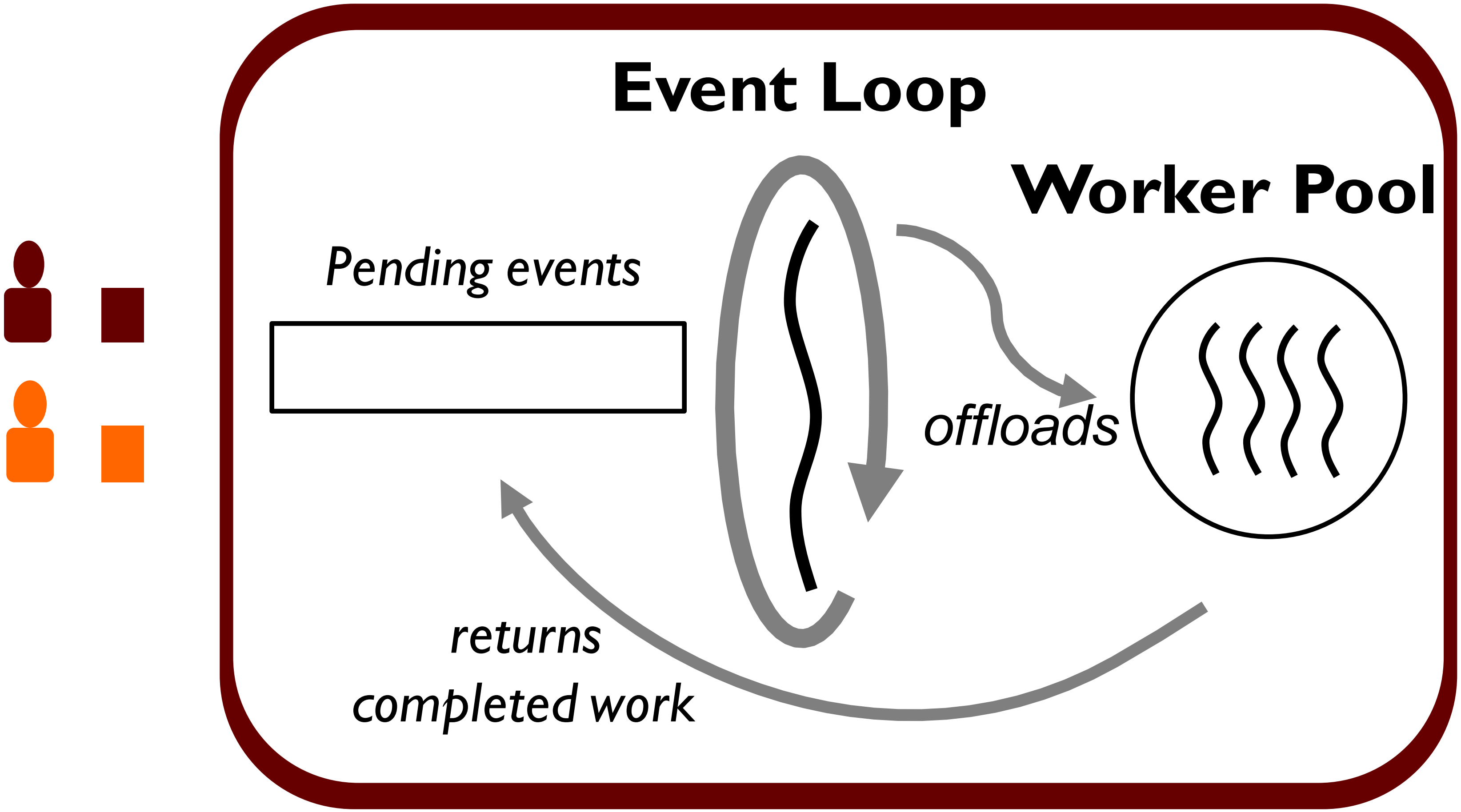
Node.js attack – with ReDoS and IO-DoS



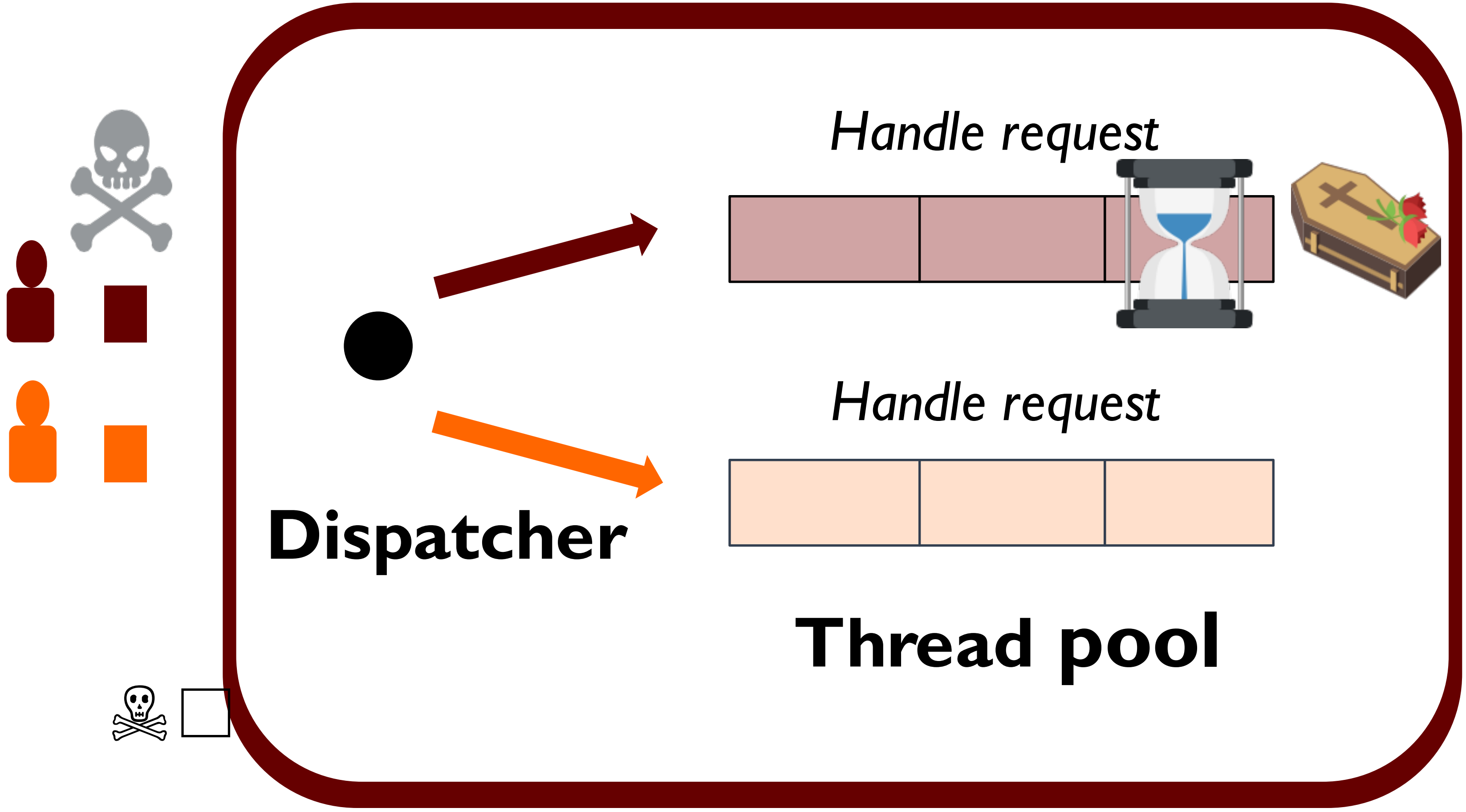
One-thread-per-client architecture (OTPCA)



Event-driven architecture (EDA)



Long-running request in OTPCA



Long-running request in EDA