



Thomas Watson

@wa7son

[github.com/watson](https://github.com/watson)













Thomas Watson

@wa7son

[github.com/watson](https://github.com/watson)



opbeat



# Detailed activity breakdown

Simple performance breakdown that shows you where your app needs optimizing, like SQL queries, MongoDB queries, http requests to other services, etc.

## Breakdown

● App ● DB ● Cache ● Template ● External

...hes/{id}([a-f0-9]{24})

...rod.\$cmd.findAndModify

Redis.GET

GET api.football-manager.com

football-data-prod.competitions.find

59% of endpoint call time

80.7ms per request

controllers/matches.js in common.step.competition

```
57. delete this.match.secondHalfStart
58.
59. if (subscription) this.match.subscription = subscription
60.
61. competitionModel.findOne(
62.   { _id: this.match.competition._id, 'seasons._id': this.match.competition._id },
63.   { 'seasons.$': 1 },
64.   next.parallel())
```

Committed: Rasmus [061fe58d](#) 15w ago

Released: [R#89](#) 15w ago

18 library frames

# usppbeat



A black and white photograph of a person walking away from the camera on a wet, cobblestone street during a rainstorm. The person is wearing a light-colored jacket and dark pants, carrying a black bag and a white shopping bag. They are holding a dark umbrella. The background shows trees and a building in the distance. A red rectangular overlay is positioned in the center of the image, containing the text "Instrumenting Node.js in production" in white.

# Instrumenting Node.js in production



## HTTP Request lifetime





# HTTP Request lifetime





# Normal solutions

- Use a global or singleton object to store context



# Normal solutions

- ~~Use a global or singleton object to store context~~
- Pass a context object around between function calls



# Normal solutions

- ~~Use a global or singleton object to store context~~
- ~~Pass a context object around between function calls~~
- ???

# Goals

- Track HTTP request as transactions
- Instrument I/O
- Instrument potential CPU expensive operations
- Associate exceptions with transactions
- Don't manually pass context around in the app
- Plug'n'play
- Almost no overhead



# Problems

- Multiple HTTP requests active at the same time
- Not possible to associate exceptions with origin HTTP request
- No API to pass context across the async boundary

# Problems

- Multiple HTTP requests active at the same time
- Not possible to associate exceptions with origin HTTP request
- No API to pass context across the async boundary



# Solution

1. Record context when a callback is queued on the event loop
2. Restore context when the callback is dequeued from the event loop

# Solution


```
global.currentTransaction = new Transaction(req)
```

1. Record context when a callback is queued on the event loop
2. Restore context when the callback is dequeued from the event loop



# Solution

```
global.currentTransaction = new Transaction(req)
```

1. Record context when a callback is queued on the event loop
  2. Restore context when the callback is dequeued from the event loop
- 
- The diagram consists of two orange arrows pointing from the text 'global.currentTransaction' to the words 'Record' in step 1 and 'Restore' in step 2, indicating that this variable is used to store and retrieve the context during the event loop cycle.







# Patch the core

- Patch the HTTP server to access new requests
- Patch **every** async operation
  - timers
  - process.nextTick
  - Promise (native)
  - libuv
- Patch Module.\_load

# Patch the core

- Patch the HTTP server to access new requests
- Patch **every** async operation
  - timers
  - process.nextTick
  - Promise (native)
  - libuv
- Patch Module.\_load

# AsyncWrap

<https://github.com/nodejs/tracing-wg>



```
var asyncWrap = process.binding('async_wrap')
```

```
var asyncWrap = process.binding('async_wrap')
```

```
asyncWrap.setupHooks({init, pre, post, destroy})
```

```
asyncWrap.enable()
```

```
var asyncWrap = process.binding('async_wrap')

asyncWrap.setupHooks({init, pre, post, destroy})
asyncWrap.enable()

function init (uid, provider, parentUid, parentHandle) {
  log('async_wrap: init')
}
function pre (uid) {
  log('async_wrap: pre')
}
function post (uid) {
  log('async_wrap: post')
}
function destroy (uid) {
  log('async_wrap: destroy')
}
```



```
,  
function post (uid) {  
  log('async_wrap: post')  
}  
function destroy (uid) {  
  log('async_wrap: destroy')  
}  
  
var fs = require('fs')  
  
log('user: before')  
fs.open(__filename, 'r', function (err, fd) {  
  log('user: done')  
})  
log('user: after')
```

```
var asyncWrap = process.binding('async_wrap')
```

```
asyncWrap.setupHooks({init, pre, post, destroy})
```

```
asyncWrap.enable()
```

```
function init (uid, provider, parentUid, parentHandle) {
```

```
  log('async_wrap: init')
```

```
}
```

```
function pre (uid) {
```

```
  log('async_wrap: pre')
```

```
}
```

```
function post (uid) {
```

```
  log('async_wrap: post')
```

```
}
```

```
function destroy (uid) {
```

```
  log('async_wrap: destroy')
```

```
}
```

```
var fs = require('fs')
```

```
log('user: before')
```

```
fs.open(__filename, 'r', function (err, fd) {
```

```
  log('user: done')
```

```
})
```

```
log('user: after')
```

user: before

async\_wrap: init

user: after

async\_wrap: pre

user: done

async\_wrap: post

async\_wrap: destroy

```
var asyncWrap = process.binding('async_wrap')

asyncWrap.setupHooks({init, pre, post, destroy})
asyncWrap.enable()

function init (uid, provider, parentUid, parentHandle) {
  console.log('async_wrap: init')
}
function pre (uid) {
  console.log('async_wrap: pre')
}
function post (uid) {
  console.log('async_wrap: post')
}
function destroy (uid) {
  console.log('async_wrap: destroy')
}

var fs = require('fs')

console.log('user: before')
fs.open(__filename, 'r', function (err, fd) {
  console.log('user: done')
})
console.log('user: after')
```



```
var asyncWrap = process.binding('async_wrap')
```

```
asyncWrap.setupHooks({init, pre, post, destroy})
```

```
asyncWrap.enable()
```

```
function init (uid, provider, parentUid, parentHandle) {
```

```
  console.log('async_wrap: init')
```

```
}
```

```
function pre (uid) {
```

```
  console.log('async_wrap: pre')
```

```
}
```

```
fun
```

```
  c FATAL ERROR: node::AsyncWrap::AsyncWrap init hook threw
```

```
}
```

```
function destroy (uid) {
```

```
  console.log('async_wrap: destroy')
```

```
}
```

```
var fs = require('fs')
```

```
console.log('user: before')
```

```
fs.open(__filename, 'r', function (err, fd) {
```

```
  console.log('user: done')
```

```
})
```

```
console.log('user: after')
```

```
fs.writeFileSync(1, util.format('%s\n', msg))
```

```
fs.writeFileSync(1, util.format('%s\n', msg))
```

```
process._rawDebug(msg)
```

---



```
var asyncWrap = process.binding('async_wrap')
```

```
asyncWrap.setupHooks({init, pre, post, destroy})
```

```
asyncWrap.enable()
```

```
function init (uid, provider, parentUid, parentHandle) {
```

```
    process._rawDebug('async_wrap: init')
```

```
}
```

```
function pre (uid) {
```

```
    process._rawDebug('async_wrap: pre')
```

```
}
```

```
function post (uid) {
```

```
    process._rawDebug('async_wrap: post')
```

```
}
```

```
function destroy (uid) {
```

```
    process._rawDebug('async_wrap: destroy')
```

```
}
```

```
var fs = require('fs')
```

```
process._rawDebug('user: before')
```

```
fs.open(__filename, 'r', function (err, fd) {
```

```
    process._rawDebug('user: done')
```

```
})
```

```
process._rawDebug('user: after')
```

user: before

async\_wrap: init

user: after

async\_wrap: pre

user: done

async\_wrap: post

async\_wrap: destroy

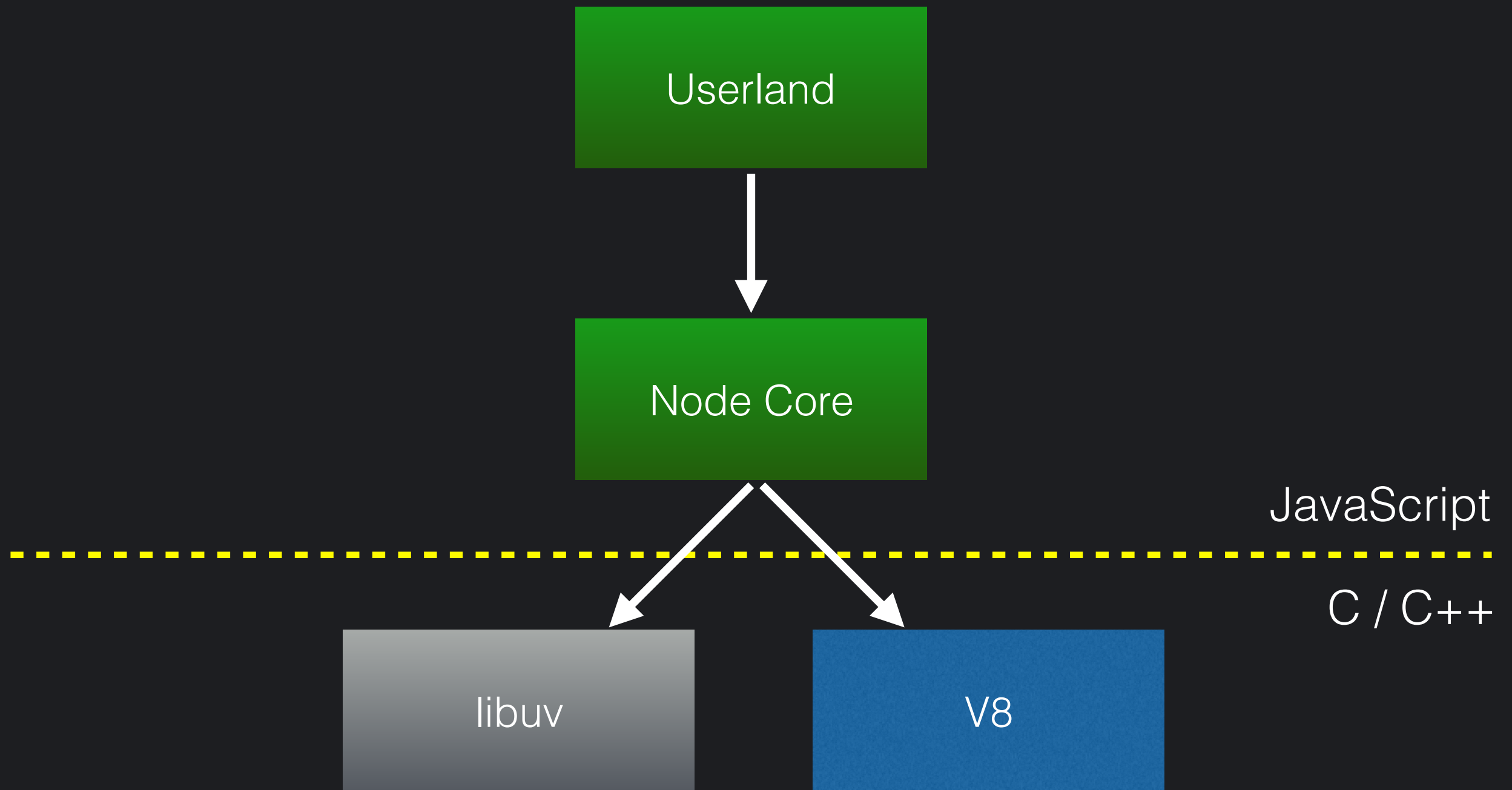
```
function init (uid, provider, parentUid, parentHandle) {  
  // this      => current handle  
  // uid       => 1, 2, 3...  
  // provider  => 0 - 23 (asyncWrap.Providers)  
  // parentUid => 1, 2, 3...  
  // parentHandle => parent `this`  
}
```

```
> var asyncWrap = process.binding('async_wrap')
undefined
> asyncWrap.Providers
{ NONE: 0,
  CRYPTO: 1,
  FSEVENTWRAP: 2,
  FSREQWRAP: 3,
  GETADDRINFOREQWRAP: 4,
  GETNAMEINFOREQWRAP: 5,
  HTTPPARSER: 6,
  JSSTREAM: 7,
  PIPEWRAP: 8,
  PIPECONNECTWRAP: 9,
  PROCESSWRAP: 10,
  QUERYWRAP: 11,
  SHUTDOWNWRAP: 12,
  SIGNALWRAP: 13,
  STATWATCHER: 14,
  TCPWRAP: 15,
  TCPCONNECTWRAP: 16,
  TIMERWRAP: 17,
  TLSWRAP: 18,
  TTYWRAP: 19,
  UDPWRAP: 20,
  UDPSENDWRAP: 21,
  WRITWRAP: 22,
  ZLIB: 23 }
```



```
function init (uid, provider, parentUid, parentHandle) {  
  // this      => current handle  
  // uid        => 1, 2, 3...  
  // provider    => 0 - 23 (asyncWrap.Providers)  
  // parentUid   => 1, 2, 3...  
  // parentHandle => parent `this`  
}
```

# Handle Objects



# Handle Objects

```
const TCPConnectWrap = process.binding('tcp_wrap').TCPConnectWrap;
const TCP = process.binding('tcp_wrap').TCP;

const req = new TCPConnectWrap();
req.oncomplete = oncomplete;
req.address = address;
req.port = port;

const socket = new TCP();
socket.onread = onread;
socket.connect(req, address, port);

// later
socket.destroy();
```

# Handle Objects

```
const TCPConnectWrap = process.binding('tcp_wrap').TCPConnectWrap;  
const TCP = process.binding('tcp_wrap').TCP;
```

```
const req = new TCPConnectWrap();  
req.oncomplete = oncomplete;  
req.address = address;  
req.port = port;
```

**req === this**

```
const socket = new TCP();  
socket.onread = onread;  
socket.connect(req, address, port);
```

**socket === this**

```
// later  
socket.destroy();
```



# Timers

```
log('user: before #1')
setTimeout(function () {
  log('user: done #1')
}, 2000)
log('user: after #1')
```

```
log('user: before #2')
setTimeout(function () {
  log('user: done #2')
}, 2000)
log('user: after #2')
```

# Timers

```
log('user: before #1')
setTimeout(function () {
  log('user: done #1')
}, 2000)
log('user: after #1')
```

```
log('user: before #2')
setTimeout(function () {
  log('user: done #2')
}, 2000)
log('user: after #2')
```

```
user: before #1
async_wrap: init
user: after #1
user: before #2
user: after #2
```

# Timers

```
log('user: before #1')
setTimeout(function () {
  log('user: done #1')
}, 2000)
log('user: after #1')

log('user: before #2')
setTimeout(function () {
  log('user: done #2')
}, 2000)
log('user: after #2')
```

```
user: before #1
async_wrap: init
user: after #1
user: before #2
user: after #2
```

```
async_wrap: pre
user: done #1
user: done #2
async_wrap: post
async_wrap: destroy
```

Show some real code



# AsyncWrap Gotchas

- Handle creation time
- `console.log`
- `process.nextTick`
- Timers
- Promises
- Multiple AsyncWrap's

# ES6 Modules

A scenic view of a harbor at dusk. The water is calm, reflecting the warm lights from the buildings and the cool blue light from the sky. Several boats are docked along the quay, their masts and rigging visible against the twilight sky. The buildings are multi-story, with many windows glowing with light. The overall atmosphere is peaceful and picturesque.

# Slides & Code

[github.com/watson/talks](https://github.com/watson/talks)



An aerial photograph of a city street, likely in Oslo, Norway. The street is wide and has tram tracks. Buildings line both sides of the street. In the background, mountains are visible under a clear sky. A red box with the word 'Takk!' is in the top right corner.

# Takk!

@wa7son

[github.com/watson](https://github.com/watson)

 opbeat