Thomas Watson

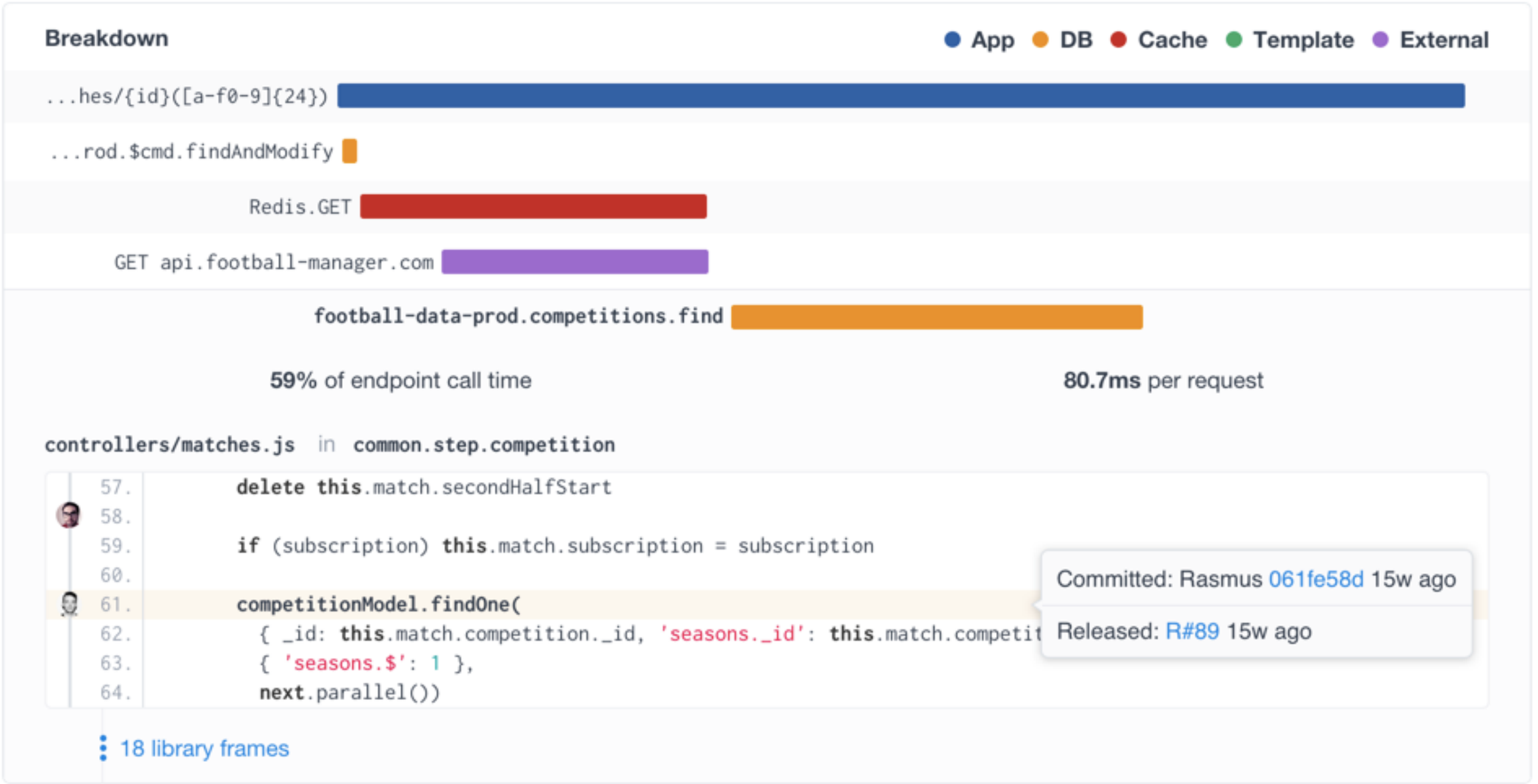
@wa7son

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.




JS
pbeat

Instrumenting Node.js in production

HTTP Request lifetime

HTTP Request lifetime

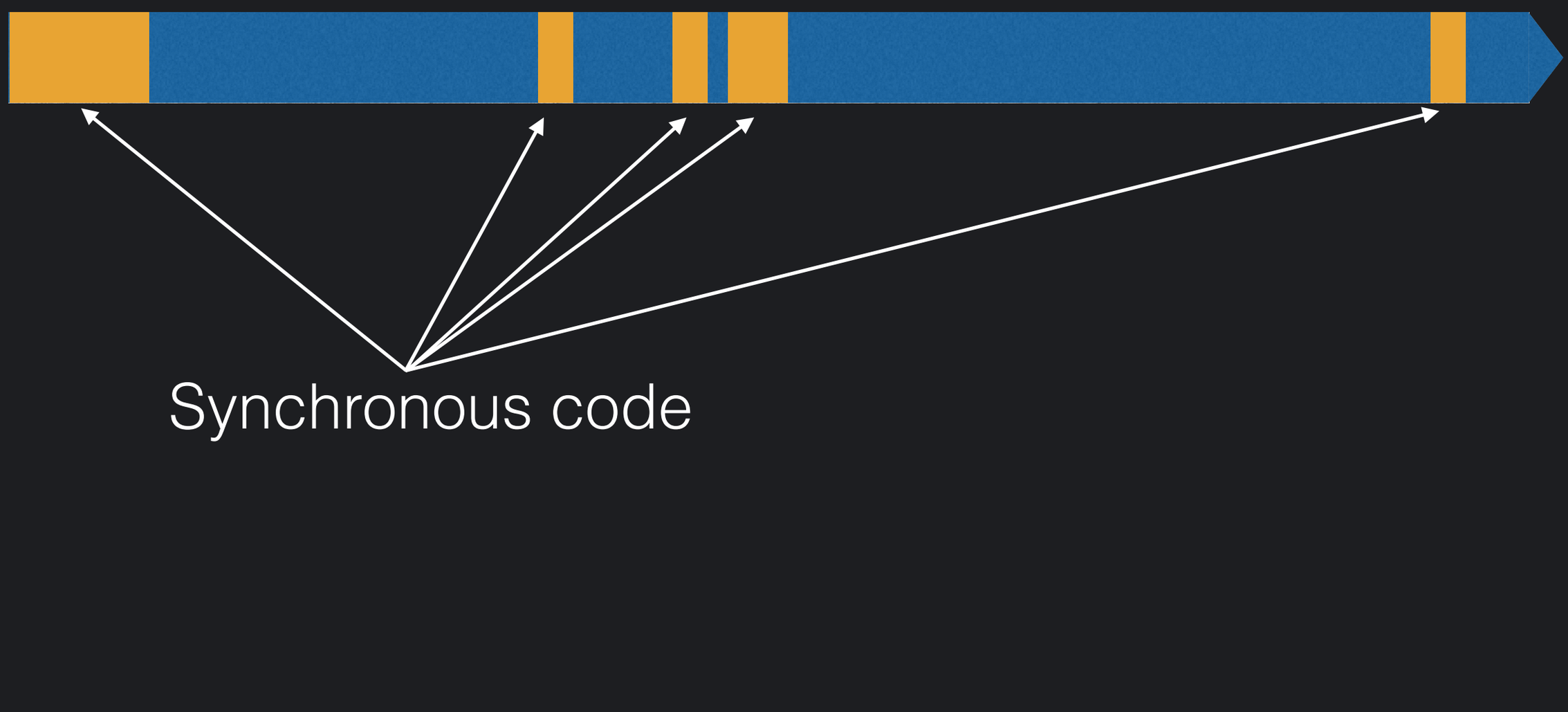
Normal solutions

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

Vocabulary

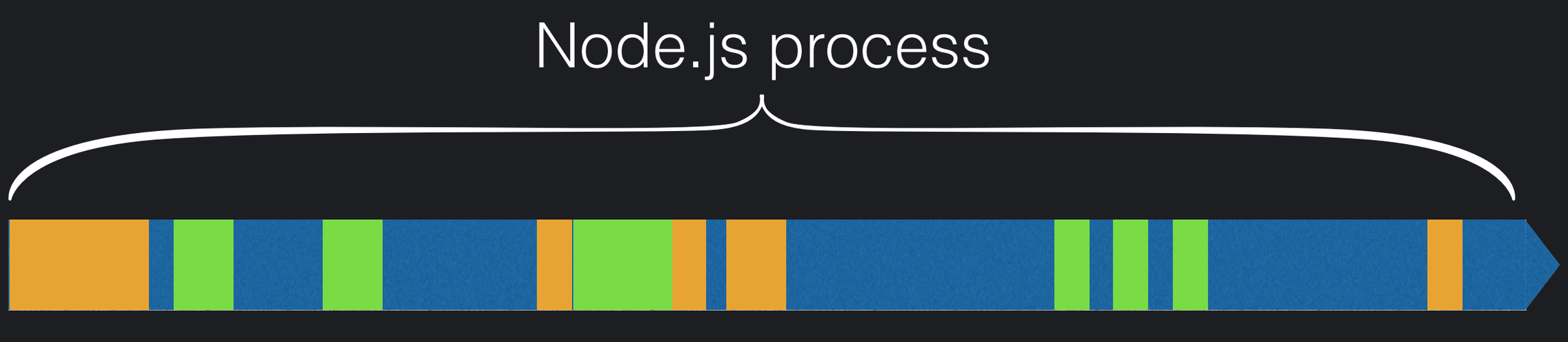
- A **trace** is a measure of how long something takes, e.g. a database query or a network request
- A **transaction** is a group of traces, e.g. all traces associated with an incoming HTTP request

The Event Loop



The Event Loop

The Event Loop



Microservices

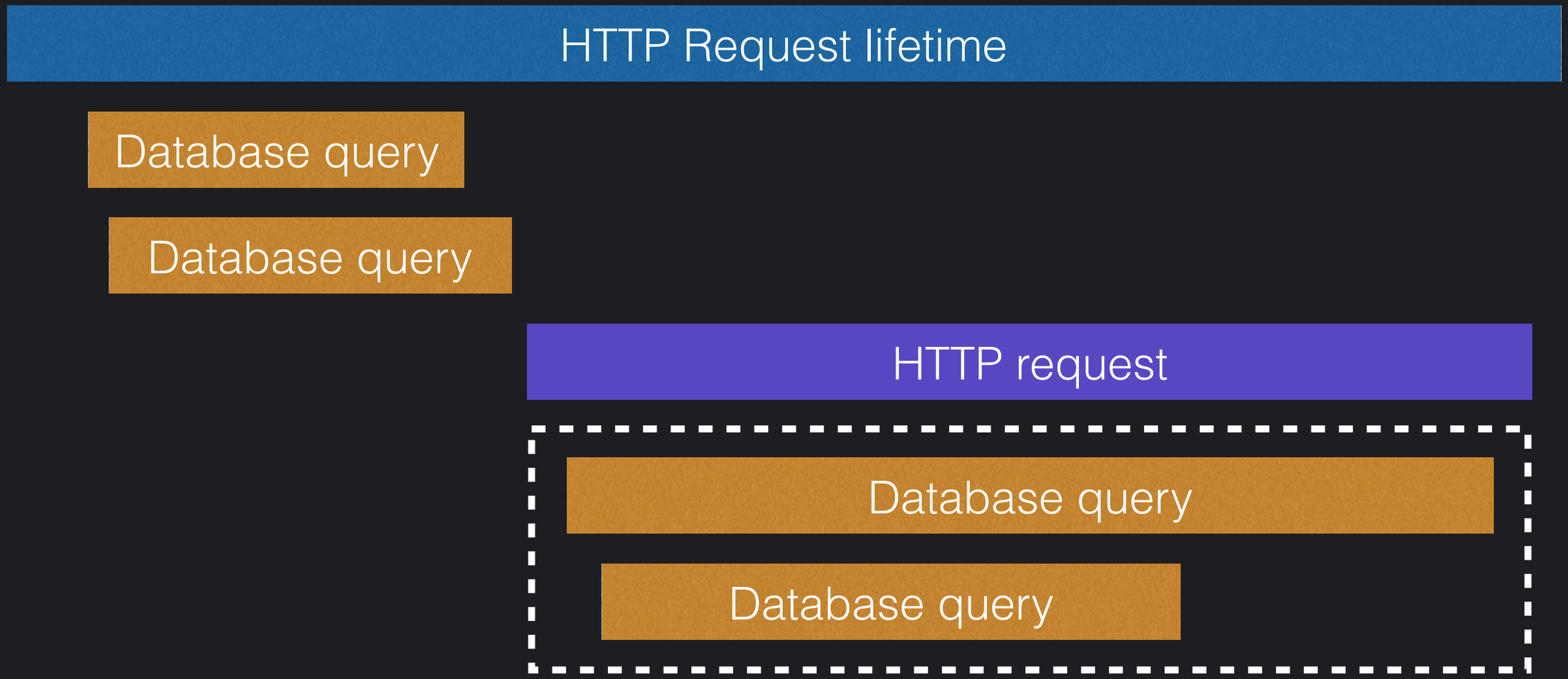
HTTP Request lifetime

Database query

Database query

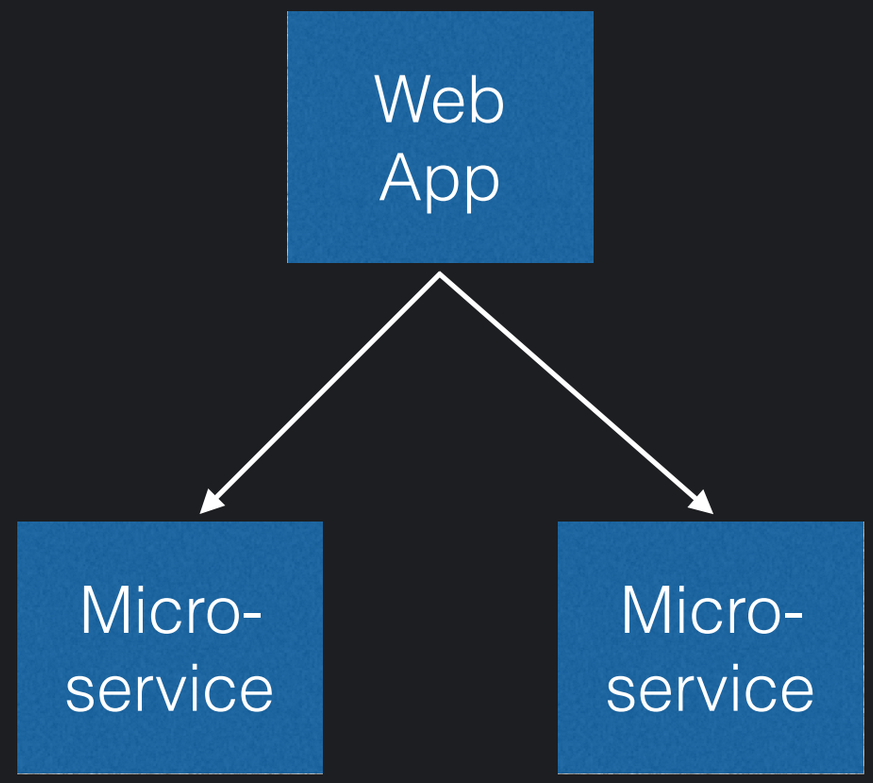
HTTP request

Microservices

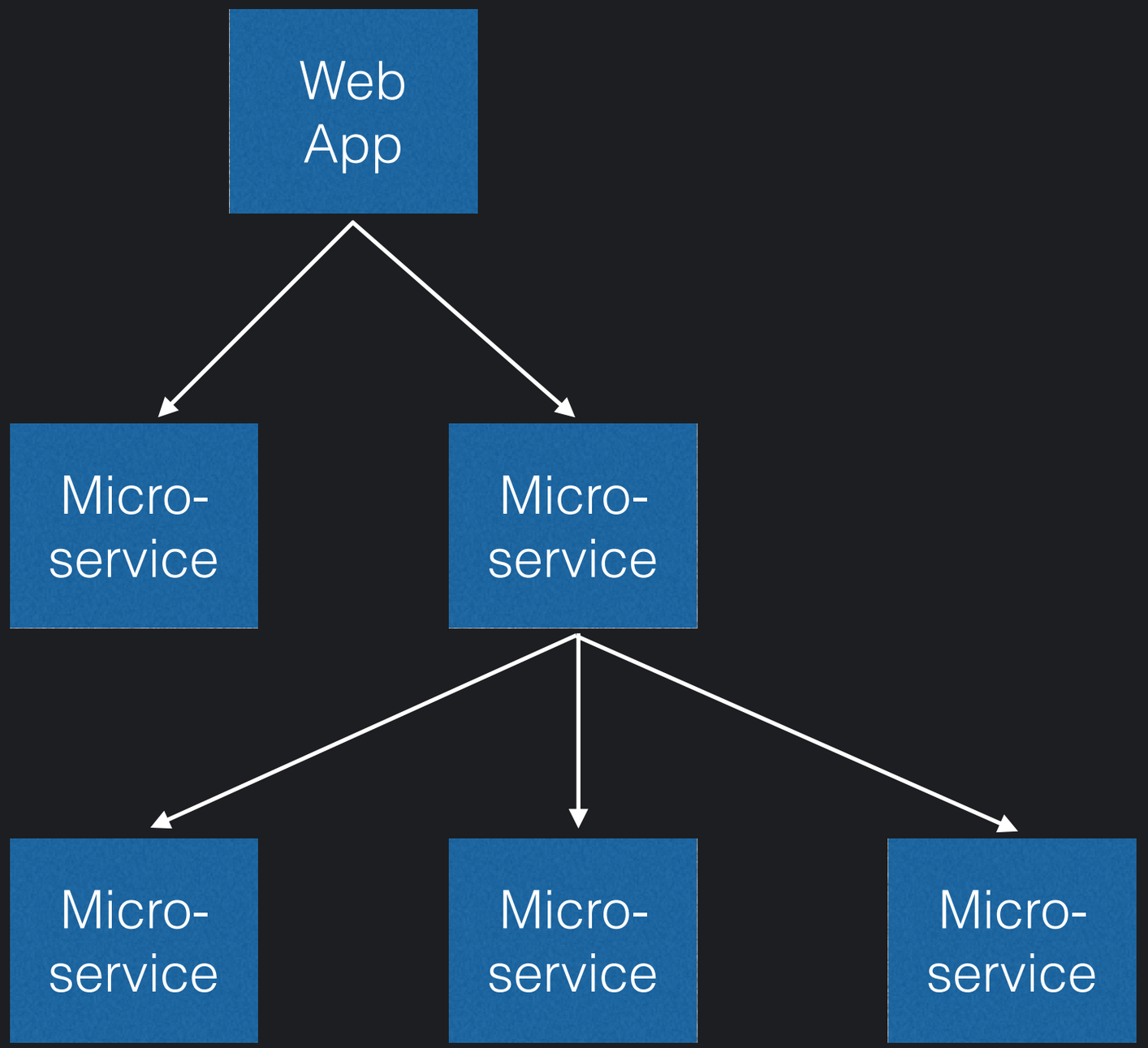


Microservices

Microservices



Microservices



Goals

- Track HTTP request using transactions
- Instrument I/O
- Instrument potential CPU expensive operations
- Keep context across requests to microservices
- Associate exceptions with HTTP requests
- Don’t manually pass context around in the app
- Plug’n’play
- Almost no overhead

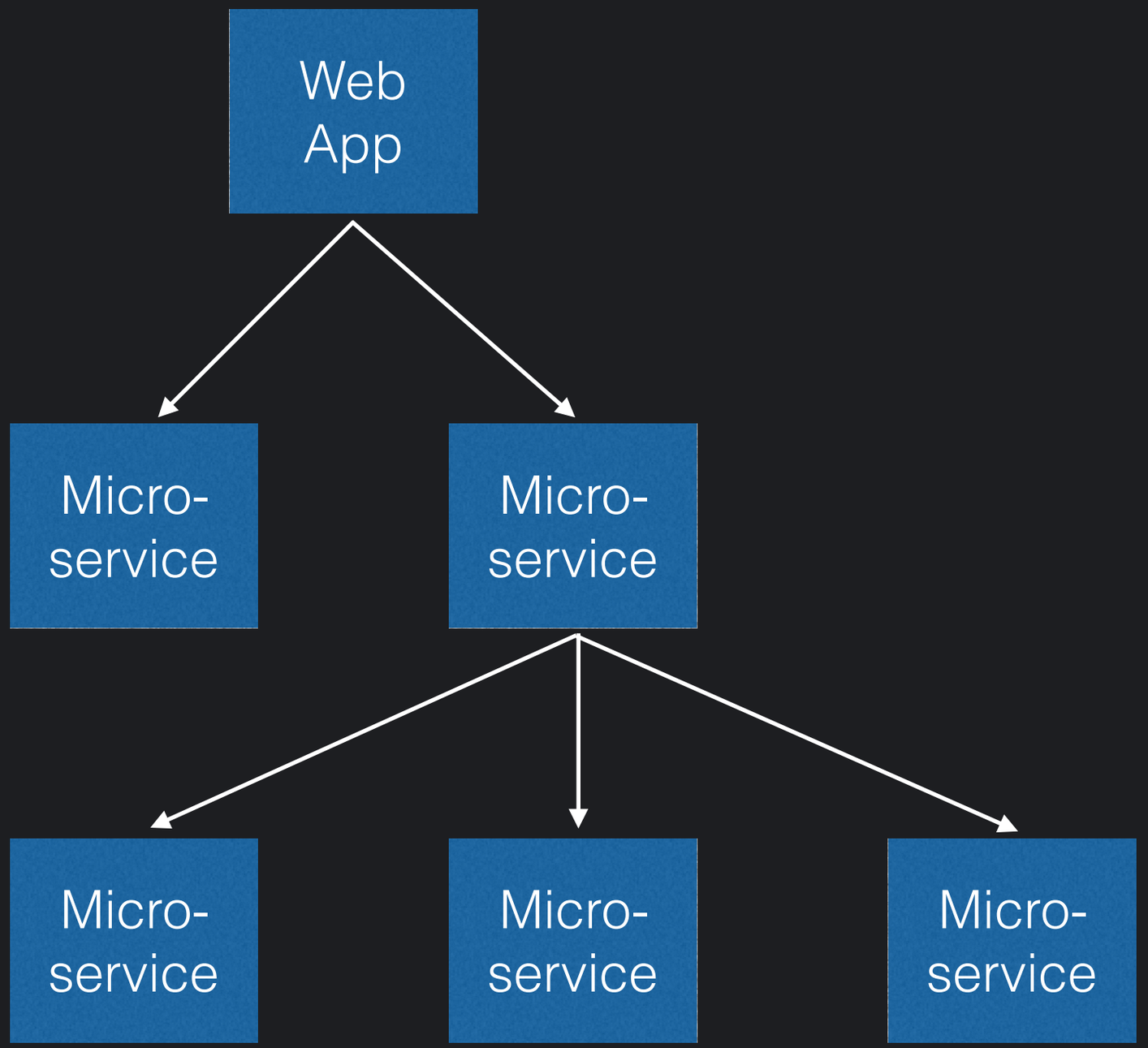
Problems

- 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
- How do we keep context across requests to microservices?

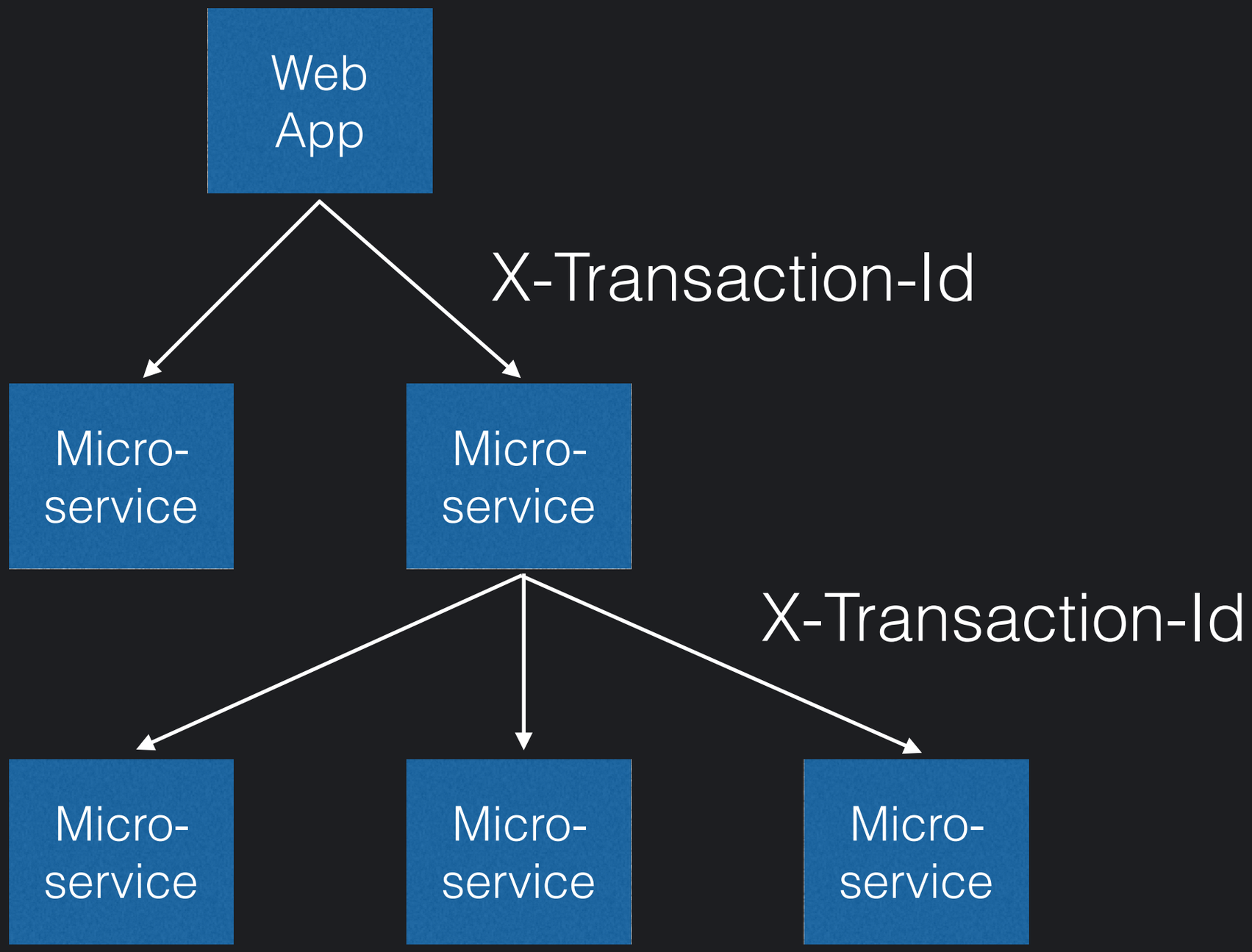
Problems

- 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
- How do we keep context across requests to microservices?

Microservices



Microservices



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
- How do we keep context across requests to microservices?

Solution

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

Solution

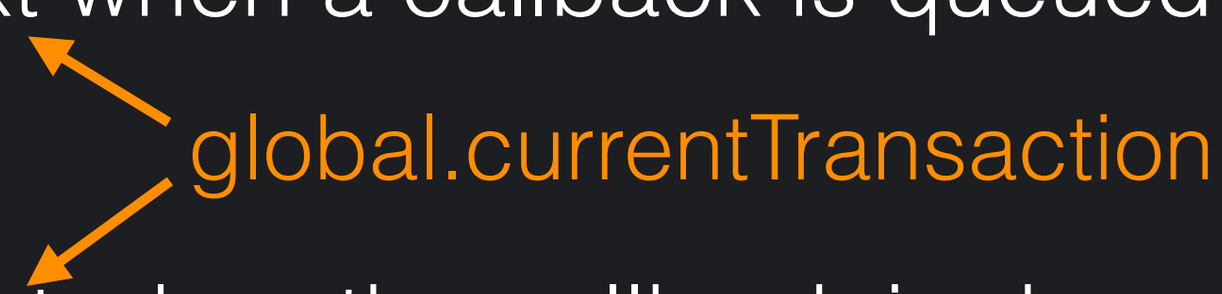
global.currentTransaction = new Transaction(req)

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



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
}
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')
```

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

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

```
fs.writeSync(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')
```

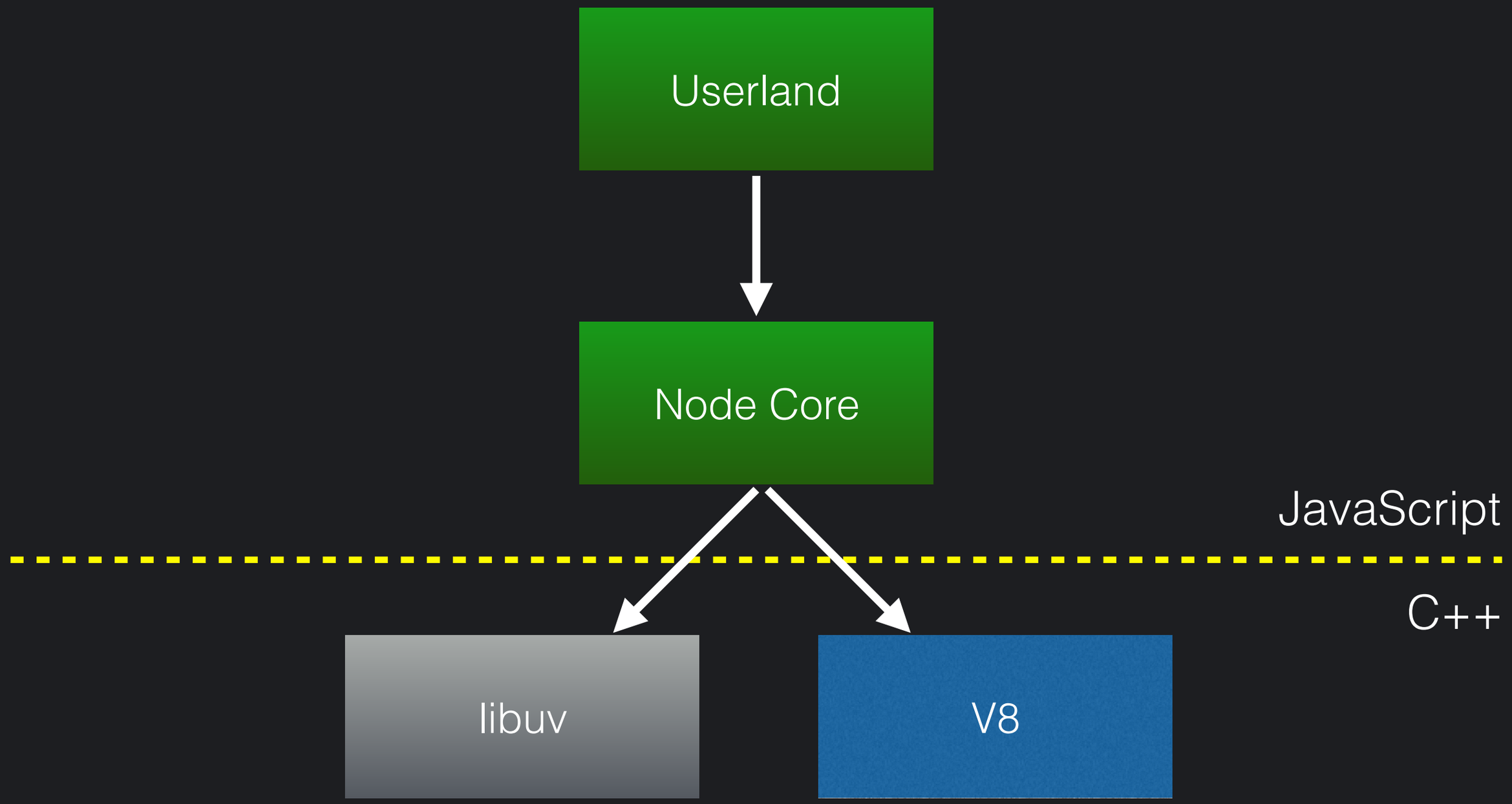
```
user: before
async_wrap: init
user: after
async_wrap: pre
user: done
async_wrap: post
async_wrap: destroy
```

```
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,
  WRITEWRAP: 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();
```

net.connect

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();
```

req === this

socket === this

net.connect

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
```

AsyncWrap Gotchas

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

Slides & Code

github.com/watson/talks

Thank you!

Any questions?

@wa7son

github.com/watson

opbeat