
Thomas Watson
@wa7son
github.com/watson

wa7son

The Trouble with Tracers

Who am I?

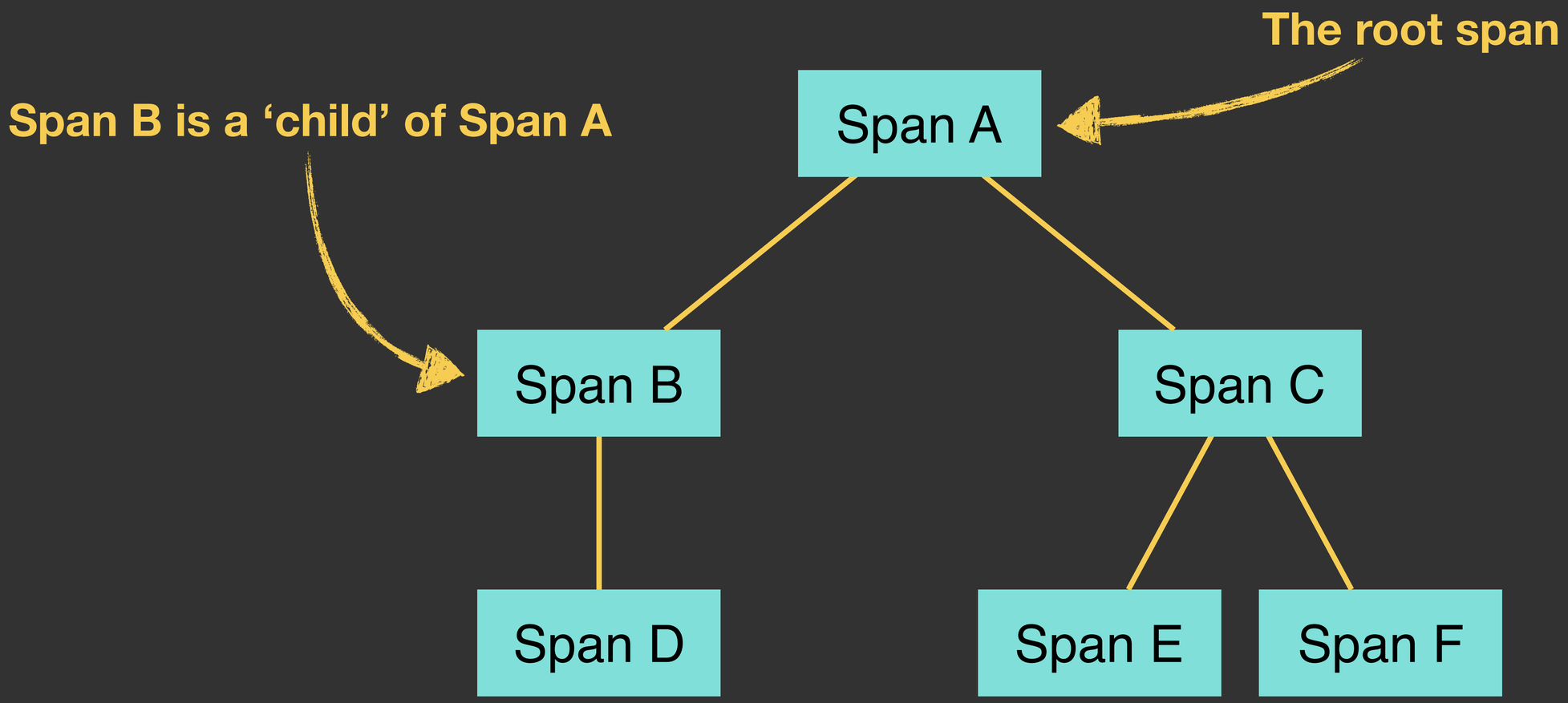
- Thomas Watson
- Open Source developer at github.com/watson
- Principal Software Engineer at Elastic
- Node.js Core Member
- Tweets as @wa7son
- Slides: github.com/watson/talks

elastic

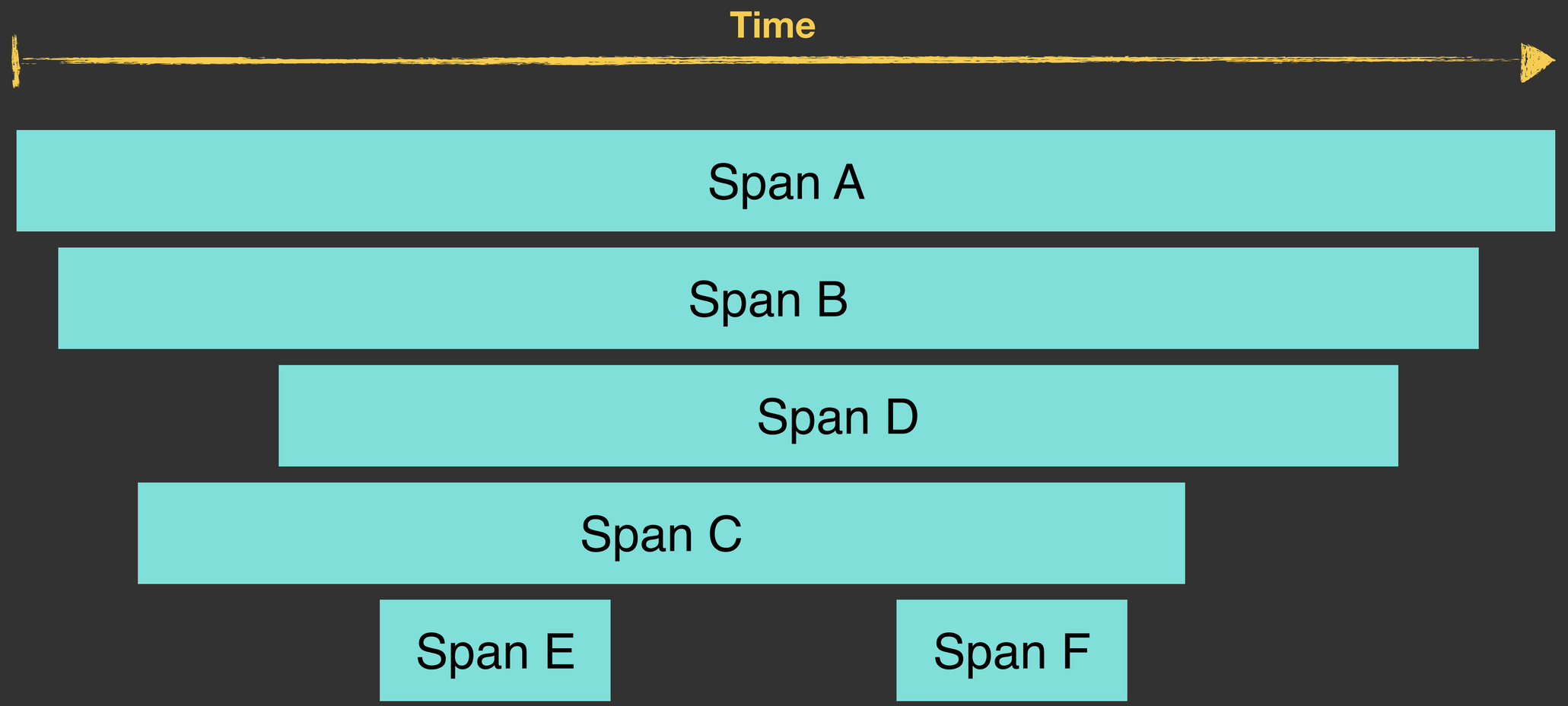
node

Tracing 101

Trace Graph

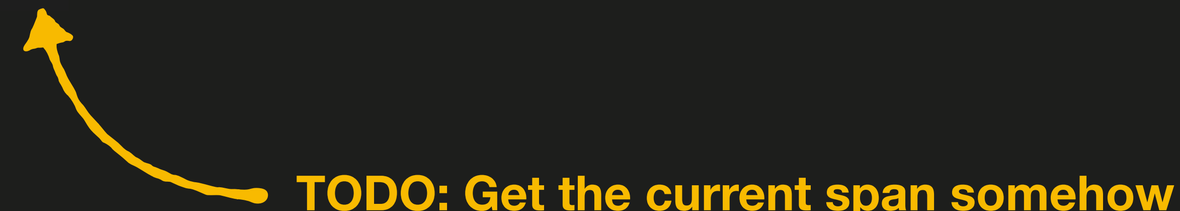

The root span
Span B is a ‘child’ of Span A
Span A
Span B
Span C
Span D
Span E
Span F

Trace Timeline



```
const span = tracer.startSpan('query', { childOf: otherSpan })

mysql.query(sql, function (err, result) {
  span.end()
  // ...
})
```

```
tracer.startSpan('query')

mysql.query(sql, processResult)

// ...

function processResult (err, result) {
  const span = ???
  span.end()
  // ...
}
```



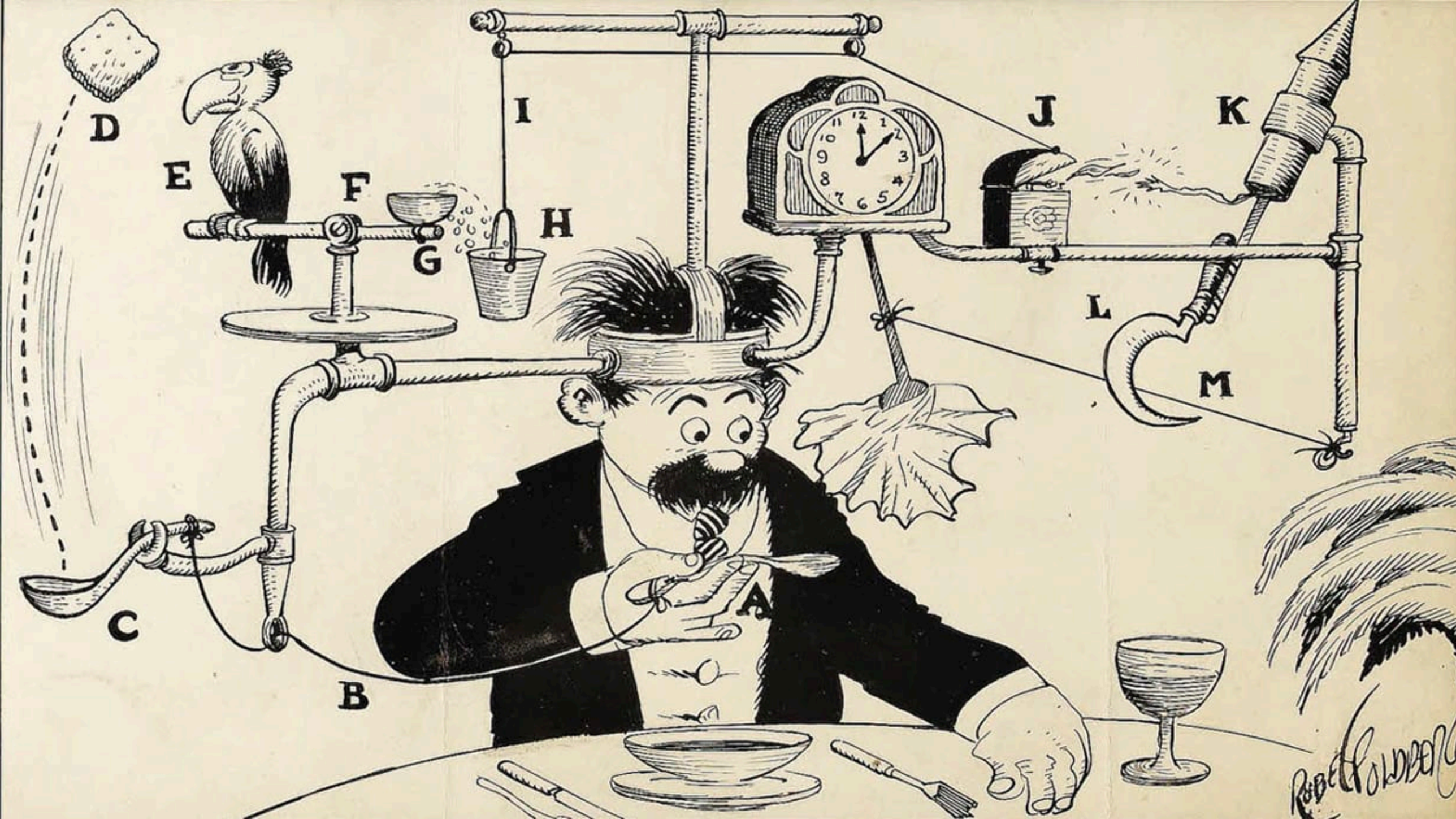
```
const span = tracer.startSpan('query')

mysql.query(sql, processResult.bind(null, { span }))

// ...

function processResult (ctx, err, result) {
  const span = ctx.span
  span.end()
  // ...
}
```

```
tracer.startSpan('query')

mysql.query(sql, processResult)

// ...

function processResult (err, result) {
  const span = tracer.getCurrentSpan()
  span.end()
  // ...
}
```

```
mysql.query(sql, function (err, result) {
  // ...
})
```

wa7son

```
mysql.qu                                    sult) {
  // ...
})
```


D
E
F
G
H
I
J
K
L
M
A
B
C

Dirty Secrets

Patching require
No built-in context propagation
Getting stack traces
Patching every async call -> async_hooks
- async await… thenables broken
- Promise destroy hook triggerd GC
- Uses numeric ids witch doesn't play nice with weak maps => hacks in place to clean it up, but those comes with risks
- What you want to patch isn't directly exported

TAV
Resource management (memory, CPU etc)
Injecting headers into HTTP requests
Babel/TypeScript defineProperty hell
Object reuse pool
Metrics: Not a lot of API's to get data from => read things from the filesystem = brittle

How do we know if
mysql is used in the current project?

How do we know when
a mysql query is performed?

```
mysql.query()
```

1. Start a span when the ‘query’ function is called
2. Associate new span with current trace
3. Make current span child of current “active” span
4. End the span when the query finishes

Patching `require()`

```
const Module = require('module')
```

```
const Module = require('module')

const module = new Module()
```

```
const Module = require('module')

const module = new Module()

module.require('http')
```

```
Module.prototype.require
```

```
const Module = require('module')

const origRequire = Module.prototype.require

Module.prototype.require = function (id) {
  const exports = origRequire.apply(this, arguments)

  if (id === 'http') {
    const request = exports.request

    exports.request = function () {
      const req = request.apply(this, arguments)

      console.log('New outgoing request to', req.getHeader('Host'))

      return req
    }
  }

  return exports
}
```

```
const Module = require('module')

const origRequire = Module.prototype.require

Module.prototype.require = function (id) {
  const exports = origRequire.apply(this, arguments)

  if (id === 'http') {
    const request = exports.request

    exports.request = function () {
      const req = request.apply(this, arguments)

      console.log('New outgoing request to', req.getHeader('Host'))

      return req
    }
  }

  return exports
}
```

```
const Module = require('module')

const origRequire = Module.prototype.require

Module.prototype.require = function (id) {
  const exports = origRequire.apply(this, arguments)

  if (id === 'http') {
    const request = exports.request

    exports.request = function () {
      const req = request.apply(this, arguments)

      console.log('New outgoing request to', req.getHeader('Host'))

      return req
    }
  }

  return exports
}
```

```
const Module = require('module')

const origRequire = Module.prototype.require

Module.prototype.require = function (id) {
  const exports = origRequire.apply(this, arguments)

  if (id === 'http') {
    const request = exports.request

    exports.request = function () {
      const req = request.apply(this, arguments)

      console.log('New outgoing request to', req.getHeader('Host'))

      return req
    }
  }

  return exports
}
```

```
const Module = require('module')

const origRequire = Module.prototype.require

Module.prototype.require = function (id) {
  const exports = origRequire.apply(this, arguments)

  if (id === 'http') {
    const request = exports.request

    exports.request = function () {
      const req = request.apply(this, arguments)

      console.log('New outgoing request to', req.getHeader('Host'))

      return req
    }
  }

  return exports
}
```

```
const Module = require('module')

const origRequire = Module.prototype.require

Module.prototype.require = function (id) {
  const exports = origRequire.apply(this, arguments)

  if (id === 'http') {
    const request = exports.request

    exports.request = function () {
      const req = request.apply(this, arguments)

      console.log('New outgoing request to', req.getHeader('Host'))

      return req
    }
  }

  return exports
}
```

```
const Module = require('module')

const origRequire = Module.prototype.require

Module.prototype.require = function (id) {
  const exports = origRequire.apply(this, arguments)

  if (id === 'http') {
    const request = exports.request

    exports.request = function () {
      const req = request.apply(this, arguments)

      console.log('New outgoing request to', req.getHeader('Host'))

      return req
    }
  }

  return exports
}
```

```
npm install require-in-the-middle
```

```
const path = require('path')
const Hook = require('require-in-the-middle')

// Hook into the express and mongodb module
Hook(['express', 'mongodb'], function (exports, name, basedir) {
  const { version } = require(path.join(basedir, 'package.json'))

  console.log('loading %s@%s', name, version)

  // expose the module version as a property on its exports object
  exports._version = version

  // whatever you return will be returned by `require`
  return exports
})
```

But wait a second…

ECMAScript Modules 🥺

wa7son

Patching Transpiled Modules or:

How I Learned to Stop Worrying and Love TypeScript & Babel

```
23  *     import { parse } from 'graphql';
24  *     import { parse } from 'graphql/language';
25  */
26
27 // The GraphQL.js version info.
28 export { version, versionInfo } from './version';
29
30 // The primary entry point into fulfilling a GraphQL request.
31 export { GraphQLArgs, graphql, graphqlSync } from './graphql';
32
33 // Create and operate on GraphQL type definitions and schema.
34 export {
35   // Definitions
36   GraphQLSchema,
37   GraphQLDirective,
38   GraphQLScalarType,
39   GraphQLObjectType,
40   GraphQLInterfaceType,
41   GraphQLUnionType,
42   GraphQLEnumType,
43   GraphQLInputObjectType,
44   GraphQLList,
45   GraphQLNonNull,
46   // Standard GraphQL Scalars
```

```
"use strict";

Object.defineProperty(exports, "__esModule", {
  value: true
});
Object.defineProperty(exports, "version", {
  enumerable: true,
  get: function get() {
    return _version.version;
  }
});
Object.defineProperty(exports, "versionInfo", {
  enumerable: true,
  get: function get() {
    return _version.versionInfo;
  }
});
Object.defineProperty(exports, "graphql", {
  enumerable: true,
  get: function get() {
    return  graphql.graphql;
```

```
"use strict";

Object.defineProperty(exports, "__esModule", {
  value: true
});
Object.defineProperty(exports, "version", {
  enumerable:
  get: functi
    return _v
  }
});
Object.define
  enumerable:
  get: functi
    return _v
  }
});
Object.defineProperty(exports, "graphql", {
  enumerable: true,
  get: function get() {
    return  graphql.graphql;
```

```
> const graphql = require('graphql')
undefined
> graphql.graphql = 'foo'
'foo'
> graphql.graphql
[Function: graphql]
> delete graphql.graphql
false
```

wa7son

```
npm install shallow-clone-shim
```

```
Object.defineProperty(exports, 'foo' {
  enumerable: true,
  get: function get () {
    return 'hello'
  }
})

const clone = require('shallow-clone-shim')

const newExports = clone({}, exports, {
  foo (descriptor) {
    // descriptor == Object.getOwnPropertyDescriptor(exports, 'foo')
    const getter = descriptor.get
    descriptor.get = function get () {
      return getter() + ' world'
    }
    return descriptor
  }
})
```

Patching Async Calls

```
const origSetTimeout = global.setTimeout

global.setTimeout = function (callback, ...args) {
  const origSpan = tracer.currentSpan

  return origSetTimeout(function () {
    const preCallbackSpan = tracer.currentSpan
    tracer.currentSpan = origSpan

    callback.apply(this, arguments)

    tracer.currentSpan = preCallbackSpan
  }, ...args)
}
```

```
const origSetTimeout = global.setTimeout

global.setTimeout = function (callback, ...args) {
  const origSpan = tracer.currentSpan

  return origSetTimeout(function () {
    const preCallbackSpan = tracer.currentSpan
    tracer.currentSpan = origSpan

    callback.apply(this, arguments)

    tracer.currentSpan = preCallbackSpan
  }, ...args)
}
```

```
const origSetTimeout = global.setTimeout

global.setTimeout = function (callback, ...args) {
  const origSpan = tracer.currentSpan

  return origSetTimeout(function () {
    const preCallbackSpan = tracer.currentSpan
    tracer.currentSpan = origSpan

    callback.apply(this, arguments)

    tracer.currentSpan = preCallbackSpan
  }, ...args)
}
```

```
const origSetTimeout = global.setTimeout

global.setTimeout = function (callback, ...args) {
  const origSpan = tracer.currentSpan

  return origSetTimeout(function () {
    const preCallbackSpan = tracer.currentSpan
    tracer.currentSpan = origSpan

    callback.apply(this, arguments)

    tracer.currentSpan = preCallbackSpan
  }, ...args)
}
```

```
const origSetTimeout = global.setTimeout

global.setTimeout = function (callback, ...args) {
  const origSpan = tracer.currentSpan

  return origSetTimeout(function () {
    const preCallbackSpan = tracer.currentSpan
    tracer.currentSpan = origSpan

    callback.apply(this, arguments)

    tracer.currentSpan = preCallbackSpan
  }, ...args)
}
```

```
const origSetTimeout = global.setTimeout

global.setTimeout = function (callback, ...args) {
  const origSpan = tracer.currentSpan

  return origSetTimeout(function () {
    const preCallbackSpan = tracer.currentSpan
    tracer.currentSpan = origSpan

    callback.apply(this, arguments)

    tracer.currentSpan = preCallbackSpan
  }, ...args)
}
```

```
const origSetTimeout = global.setTimeout

global.setTimeout = function (callback, ...args) {
  const origSpan = tracer.currentSpan

  return origSetTimeout(function () {
    const preCallbackSpan = tracer.currentSpan
    tracer.currentSpan = origSpan

    callback.apply(this, arguments)

    tracer.currentSpan = preCallbackSpan
  }, ...args)
}
```

All async API’s

- net
- http
- child_process
- timers
- dns
- fs
- zlib
- crypto
- global.Promise

```
require('async_hooks')
```

But wait a second…

Async/await + Thenables 🥺

Resource Usage 📈

```
const asyncHooks = require('async_hooks')

const currentSpans = new Map()

asyncHooks.createHook({
  init (asyncId) {
    const span = tracer.currentSpan

    if (span) {
      currentSpans.set(asyncId, span)
    }
  },

  destroy (asyncId) {
    currentSpans.delete(asyncId)
  }
})
```

```
const asyncHooks = require('async_hooks')

const currentSpans = new Map()

asyncHooks.createHook({
  init (asyncId) {
    const span = tracer.currentSpan

    if (span) {
      currentSpans.set(asyncId, span)
    }
  },

  destroy (asyncId) {
    currentSpans.delete(asyncId)
  }
})
```

```
const asyncHooks = require('async_hooks')

const currentSpans = new Map()

asyncHooks.createHook({
  init (asyncId) {
    const span = tracer.currentSpan

    if (span) {
      currentSpans.set(asyncId, span)
    }
  },

  destroy (asyncId) {
    currentSpans.delete(asyncId)
  }
})
```

```
const asyncHooks = require('async_hooks')

const currentSpans = new Map()

asyncHooks.createHook({
  init (asyncId) {
    const span = tracer.currentSpan

    if (span) {
      currentSpans.set(asyncId, span)
    }
  },

  destroy (asyncId) {
    currentSpans.delete(asyncId)
  }
})
```

```
const asyncHooks = require('async_hooks')

const currentSpans = new Map()

asyncHooks.createHook({
  init (asyncId) {
    const span = tracer.currentSpan

    if (span) {
      currentSpans.set(asyncId, span)
    }
  },

  destroy (asyncId) {
    currentSpans.delete(asyncId)
  }
})
```

```
const asyncHooks = require('async_hooks')

const currentSpans = new Map()

asyncHooks.createHook({
  init (asyncId) {
    const span = tracer.currentSpan

    if (span) {
      currentSpans.set(asyncId, span)
    }
  },

  destroy (asyncId) {
    currentSpans.delete(asyncId)
  }
})
```

```
const asyncHooks = require('async_hooks')

const currentSpans = new Map()

asyncHooks.createHook({
  init (asyncId) {
    const span = tracer.currentSpan

    if (span) {
      currentSpans.set(asyncId, span)
    }
  },

  destroy (asyncId) {
    currentSpans.delete(asyncId)
  }
})
```

Not a WeakMap 🙄

Can we trust (when) destroy is called?

Userland Callback Queues

```
class Client {
  constructor () {
    this.pool = new Pool({ size: 5 })
    this.queue = []
  }

  request (query, callback) {
    const socket = this.pool.getAvailableSocket()

    if (socket) {
      socket.send(query, (err, res) => {
        callback(err, res)

        if (this.queue.length > 0) {
          const [query, callback] = this.queue.shift()
          this.request(query, callback)
        }
      })
    } else {
      this.queue.push([query, callback])
    }
  }
}
```

```
class Client {
  constructor () {
    this.pool = new Pool({ size: 5 })
    this.queue = []
  }

  request (query, callback) {
    const socket = this.pool.getAvailableSocket()

    if (socket) {
      socket.send(query, (err, res) => {
        callback(err, res)

        if (this.queue.length > 0) {
          const [query, callback] = this.queue.shift()
          this.request(query, callback)
        }
      })
    } else {
      this.queue.push([query, callback])
    }
  }
}
```

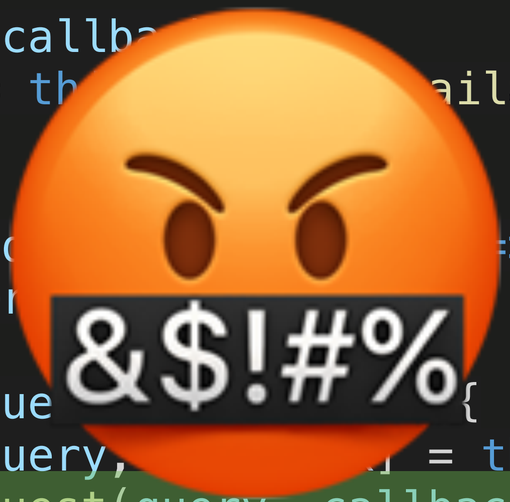
```
class Client {
  constructor () {
    this.pool = new Pool({ size: 5 })
    this.queue = []
  }

  request (query, callback) {
    const socket = this.pool.getAvailableSocket()

    if (socket) {
      socket.send(query, (err, res) => {
        callback(err, res)

        if (this.queue.length > 0) {
          const [query, callback] = this.queue.shift()
          this.request(query, callback)
        }
      })
    } else {
      this.queue.push([query, callback])
    }
  }
}
```

```
class Client {
  constructor () {
    this.pool = new Pool({ size: 5 })
    this.queue = []
  }

  request (query, callback) {
    const socket = this.pool.getAvailableSocket()

    if (socket) {
      socket.send(query, (err, res) => {
        callback(err, res)

        if (this.queue.length > 0) {
          const [query, callback] = this.queue.shift()
          this.request(query, callback)
        }
      })
    } else {
      this.queue.push([query, callback])
    }
  }
}
```

```
class Client {
  constructor () {
    this.pool = new Pool({ size: 5 })
    this.queue = []
  }

  request (query, callback) {
    const socket = this.pool.getAvailableSocket()

    if (socket) {
      socket.send(query, (err, res) => {
        callback(err, res)

        if (this.queue.length > 0) {
          const [query, callback] = this.queue.shift()
          this.request(query, callback)
        }
      })
    } else {
      this.queue.push([query, callback])
    }
  }
}
```

```
class Client {
  constructor () {
    this.pool = new Pool({ size: 5 })
    this.queue = []
  }

  request (query, callback) {
    const socket = this.pool.getAvailableSocket()

    if (socket) {
      socket.send(query, (err, res) => {
        callback(err, res)

        if (this.queue.length > 0) {
          const [query, callback] = this.queue.shift()
          this.request(query, callback)
        }
      })
    } else {
      this.queue.push([query, callback])
    }
  }
}
```

```
class Client {
  constructor () {
    this.pool = new Pool({ size: 5 })
    this.queue = []
  }

  request (query, callba
    const socket = th                ailableSocket()

    if (socket) {
      socket.send(c                  => {
        callback(er

        if (this.que                 {
          const [query,              ] = this.queue.shift()
          this.request(query, callback)
        }
      })
    } else {
      this.queue.push([query, callback])
    }
  }
}
```



Called in the context of the previous `socket.send()` call

```
class Client {
  constructor () {
    this.pool = new Pool({ size: 5 })
```

Which is called in the context of the `st()` call

This.currentSpan turns to null after a POST request #515

Closed rannygmx opened this issue on 15 Aug 2018 · 20 comments

```
    if (socket) {
      socket.send(query, (err, res) => {
        callback(err, res)

        if (this.queue.length > 0) {
          const [query, callback] = this.queue.shift()
          this.request(query, callback)
        }
      })
    } else {
      this.queue.push([query, callback])
    }
  }
}
```

Called in the context of the previous `socket.send()` call

watson commented on 15 Aug 2018 Member

Hi **@rannygmx**

Thanks for letting us know about this issue. The problem you're experiencing is most likely related to a context propagation issue. This happens when your application performs an async operation that we didn't anticipate. This is when a callback is scheduled to be called after some async operation finishes, but we fail to register it. If so, we don't know which transaction was the active transaction when the callback is finally called, and so `currentTransaction` is suddenly `null`.

90% of the time this is because of what's called a "user-land callback queue", where for instance a database-driver maintains an internal pool of connections to the databases. When you then make a database query, but all connections in the pool are in use, your query is put into a queue that waits until one of the connections in the pool becomes available. If the Node.js agent doesn't know about this pool, we'll also not know which transaction should be activated once the result of the query is returned.

Step one in figuring out what's going on in your particular case is to take a look at your dependencies. If you don't mind, could you please share with us the list of dependencies in your package.json file?

class Client {

rannygmx commented on 15 Aug 2018 Author + ...

Thanks for the explanation,
This is my dependencies list:

```
"dependencies": {
"async": "^2.4.1",
"body-parser": "^1.12.3",
"compression": "^1.4.3",
"cors": "^2.6.0",
"cron": "^1.2.1",
"dom-js": "^0.0.9",
"elastic-apm-node": "^1.7.0",
"elasticsearch": "^13.1.1",
"errorhandler": "^1.3.5",
"express": "^4.12.3",
"kafka-node": "^0.3.2",
"lodash": "^4.13.1",
"method-override": "^2.3.2",
```

Interservice Communication


<magic header>
GET /products HTTP/1.1
Host: www.example.com
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0

```
GET /products HTTP/1.1
Host: www.example.com
<magic header>
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0
```

```
GET /products HTTP/1.1
Host: www.example.com
traceparent: 00-82c5500f40667e5500e9ae8e9711553c-992631f881f78c3b-01
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0
```

Candidate Recommendation

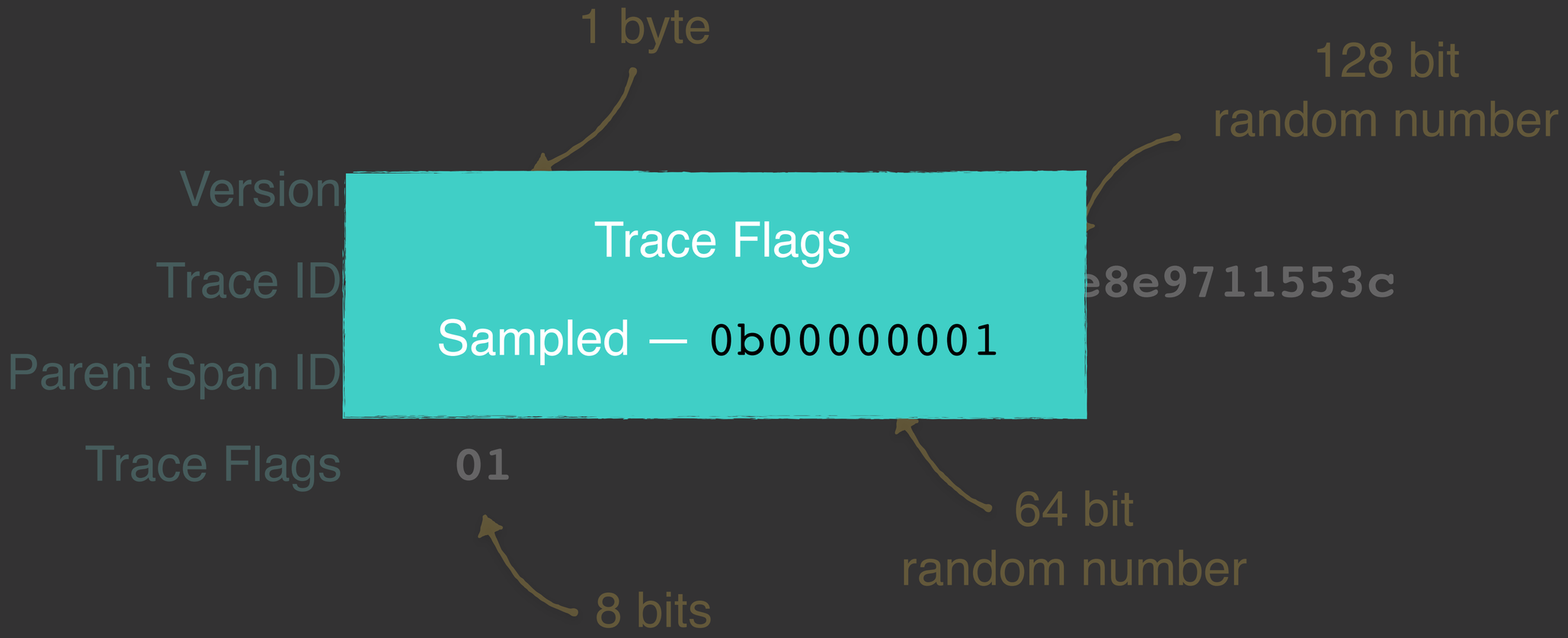
```
GET /products H    1.1
Host: www.example.com
traceparent: 00-82c55                                  -992631f881f78c3b-01
Date: Mon, 29 Oct 201
Connection: keep-alive
Content-Length: 0
```

Trace Context Working Group

```
GET /products HTTP/1.1
Host: www.example.com
traceparent: 00-82c5500f40667e5500e9ae8e9711553c-992631f881f78c3b-01
Date: Mon, 29 Oct 2018 16:11:05 GMT
Connection: keep-alive
Content-Length: 0
```

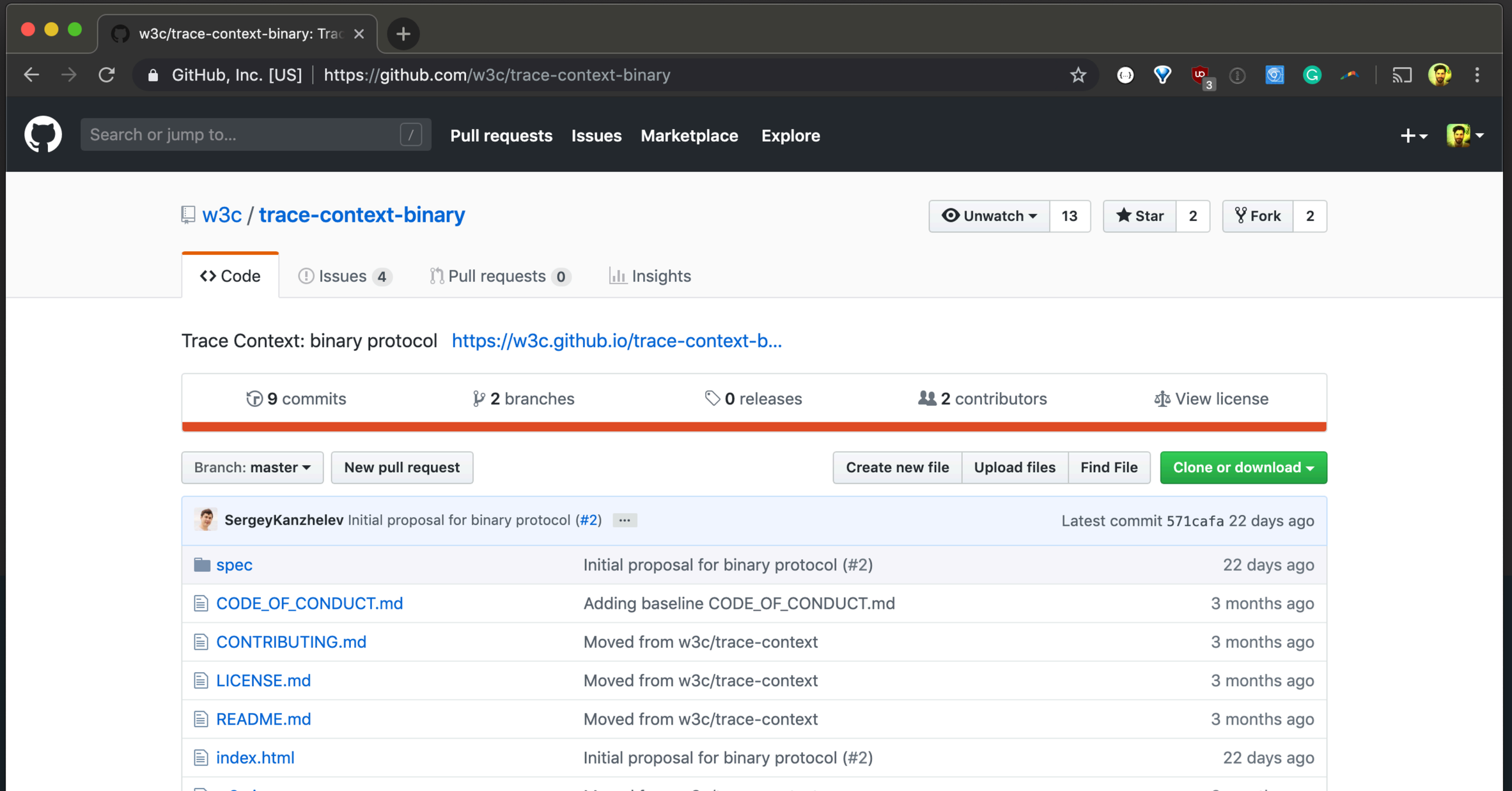
traceparent

traceparent



wa7son

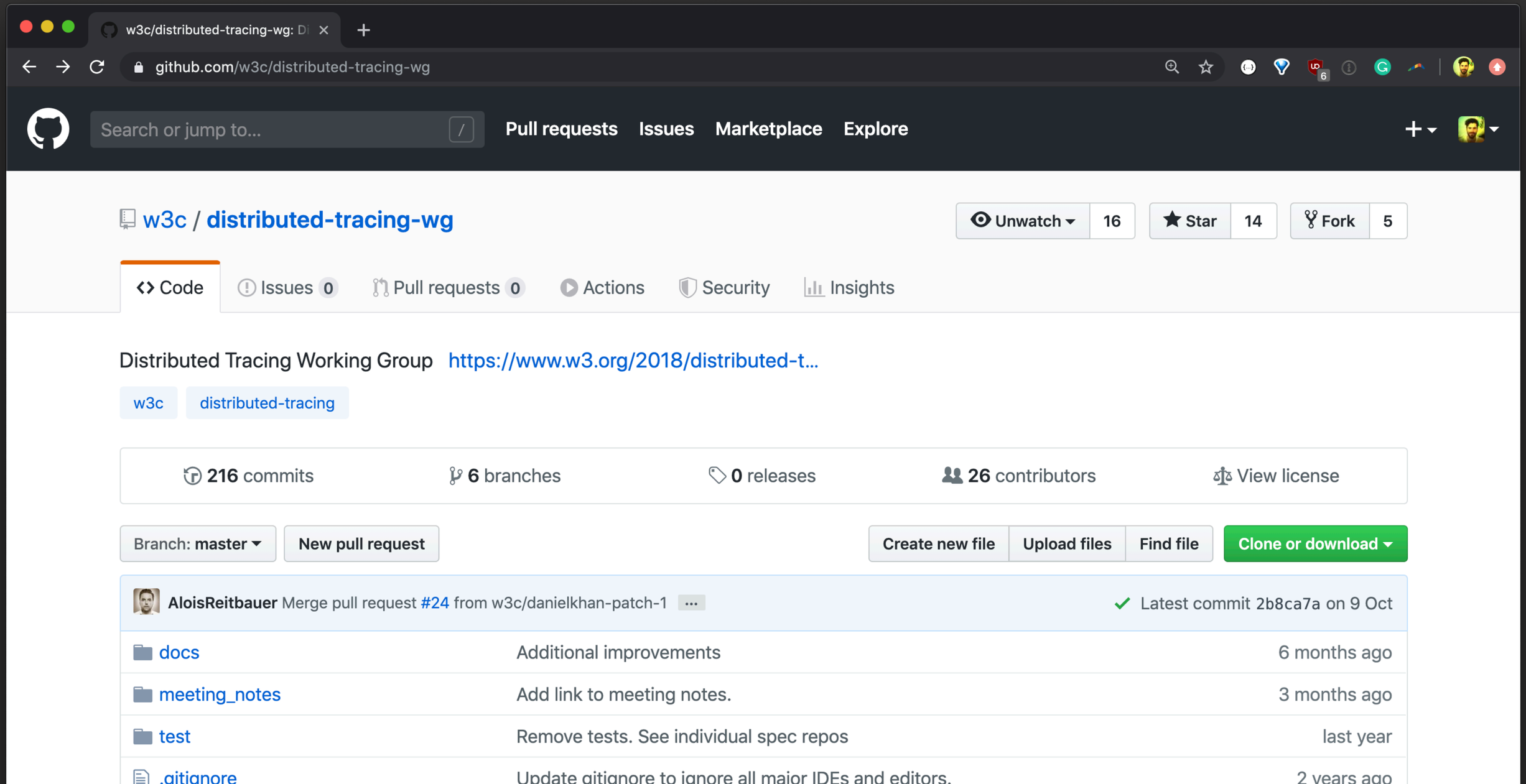
github.com / w3c / trace-context-binary



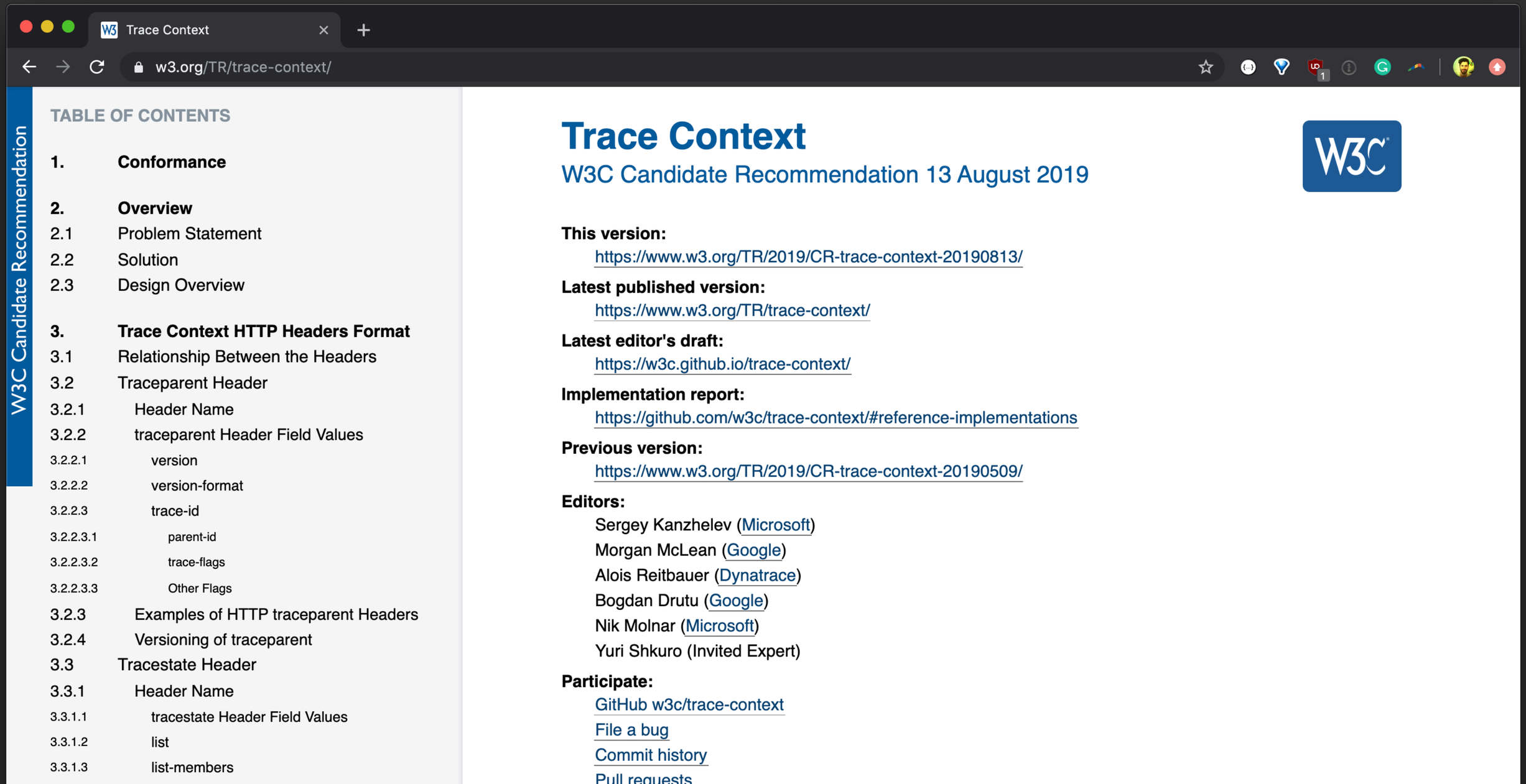
Standards

wa7son

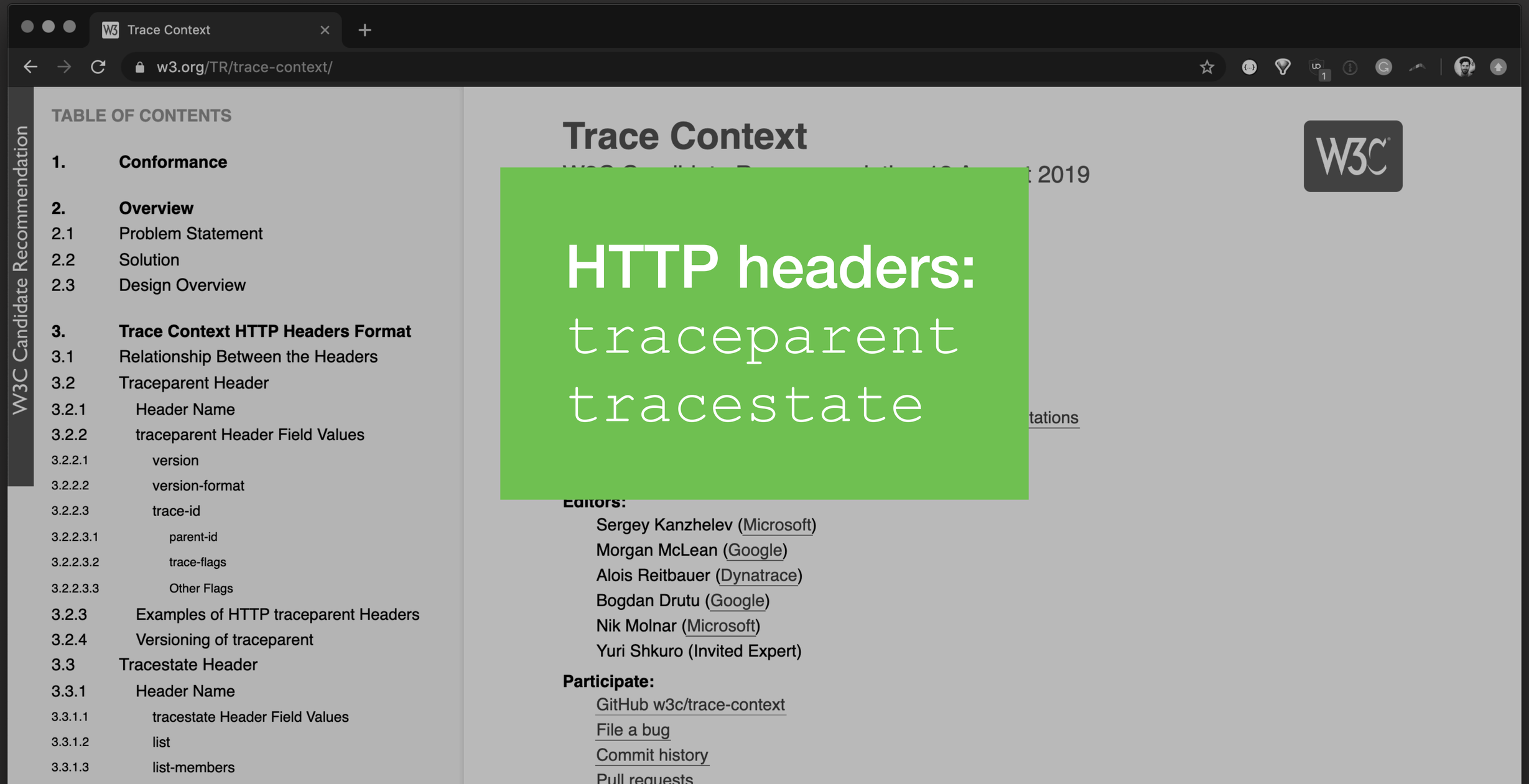
github.com / w3c / distributed-tracing-wg



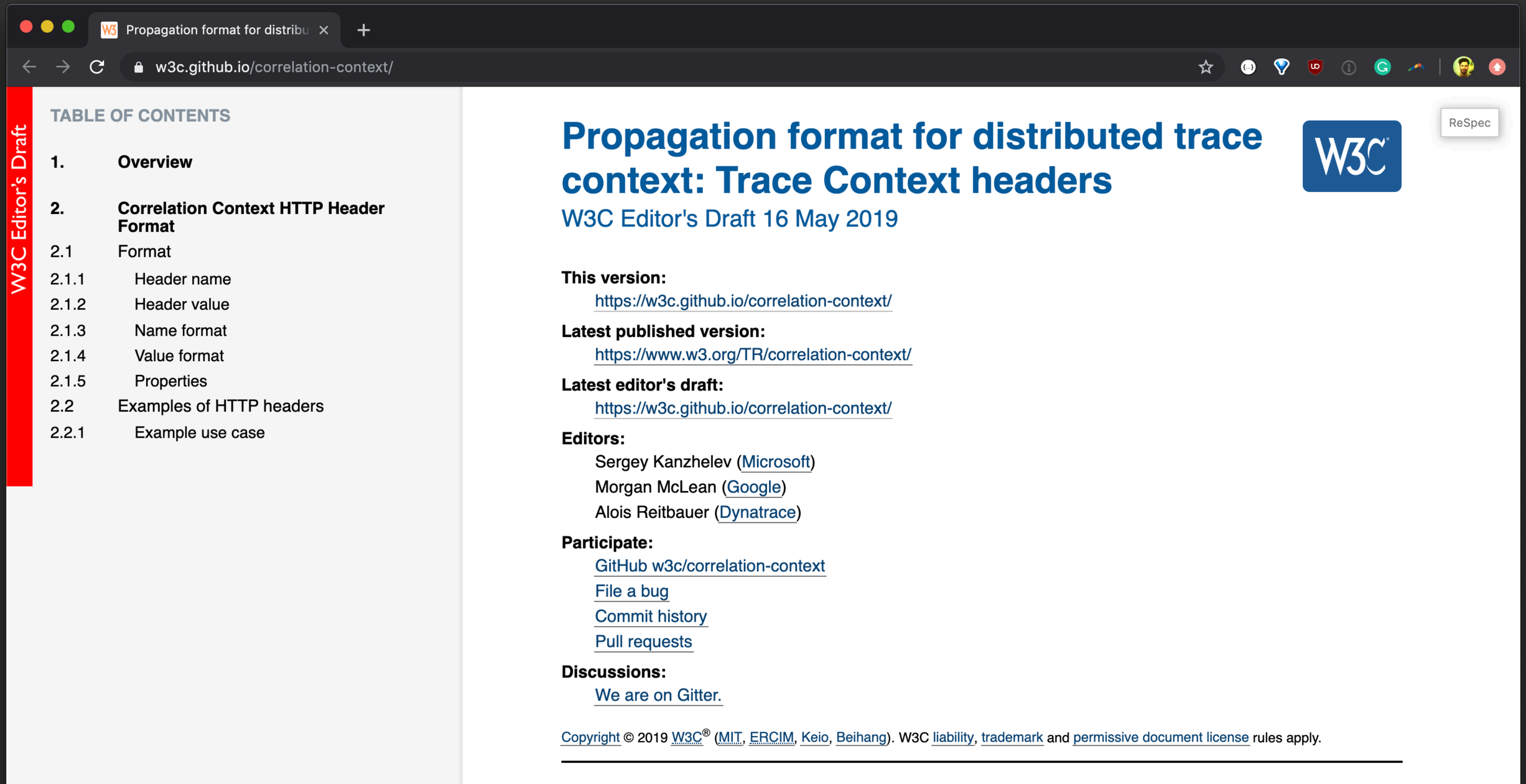
w3.org / TR / trace-context



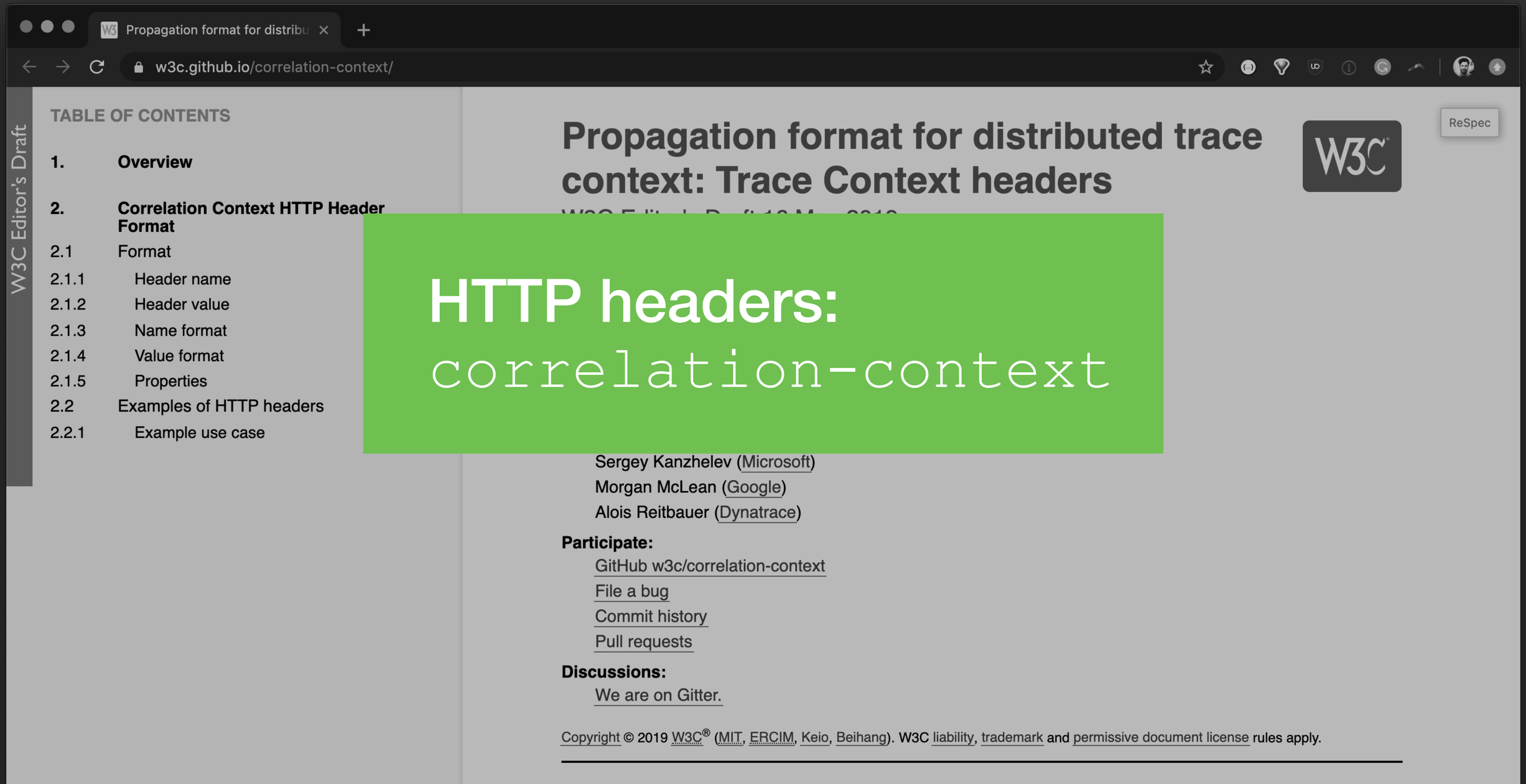
w3.org / TR / trace-context



w3c.github.io / correlation-context

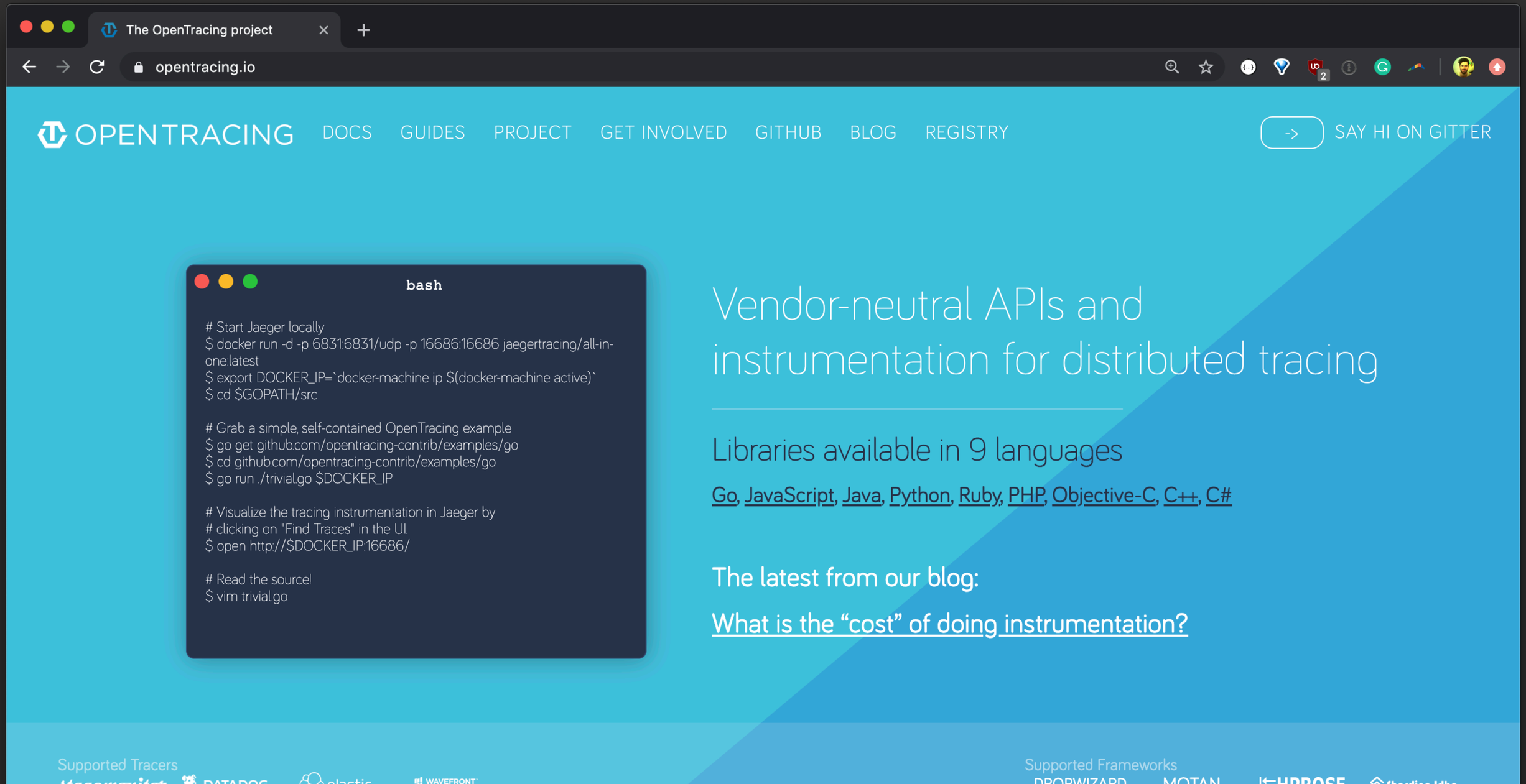


w3c.github.io / correlation-context



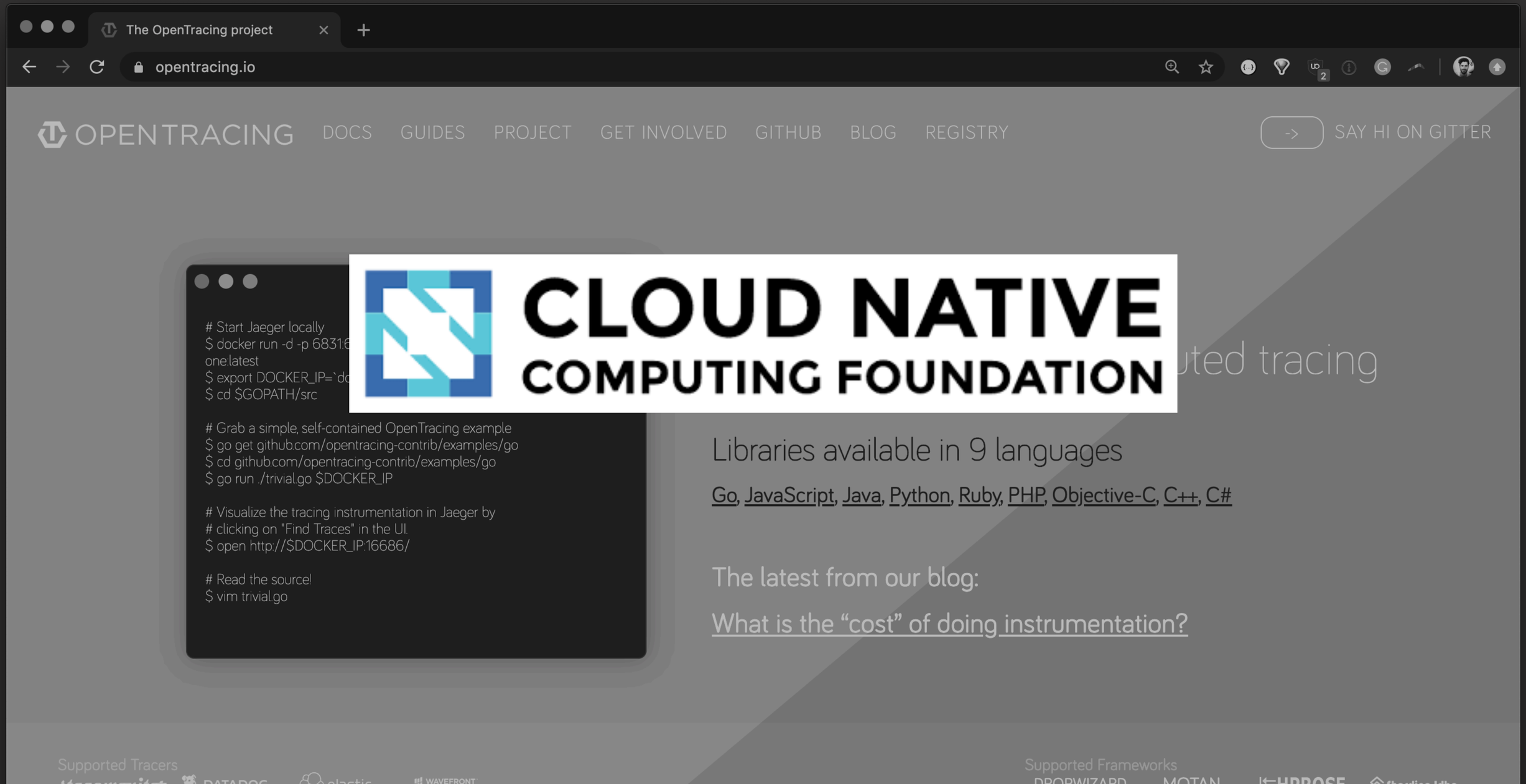
wa7son

opentracing.io

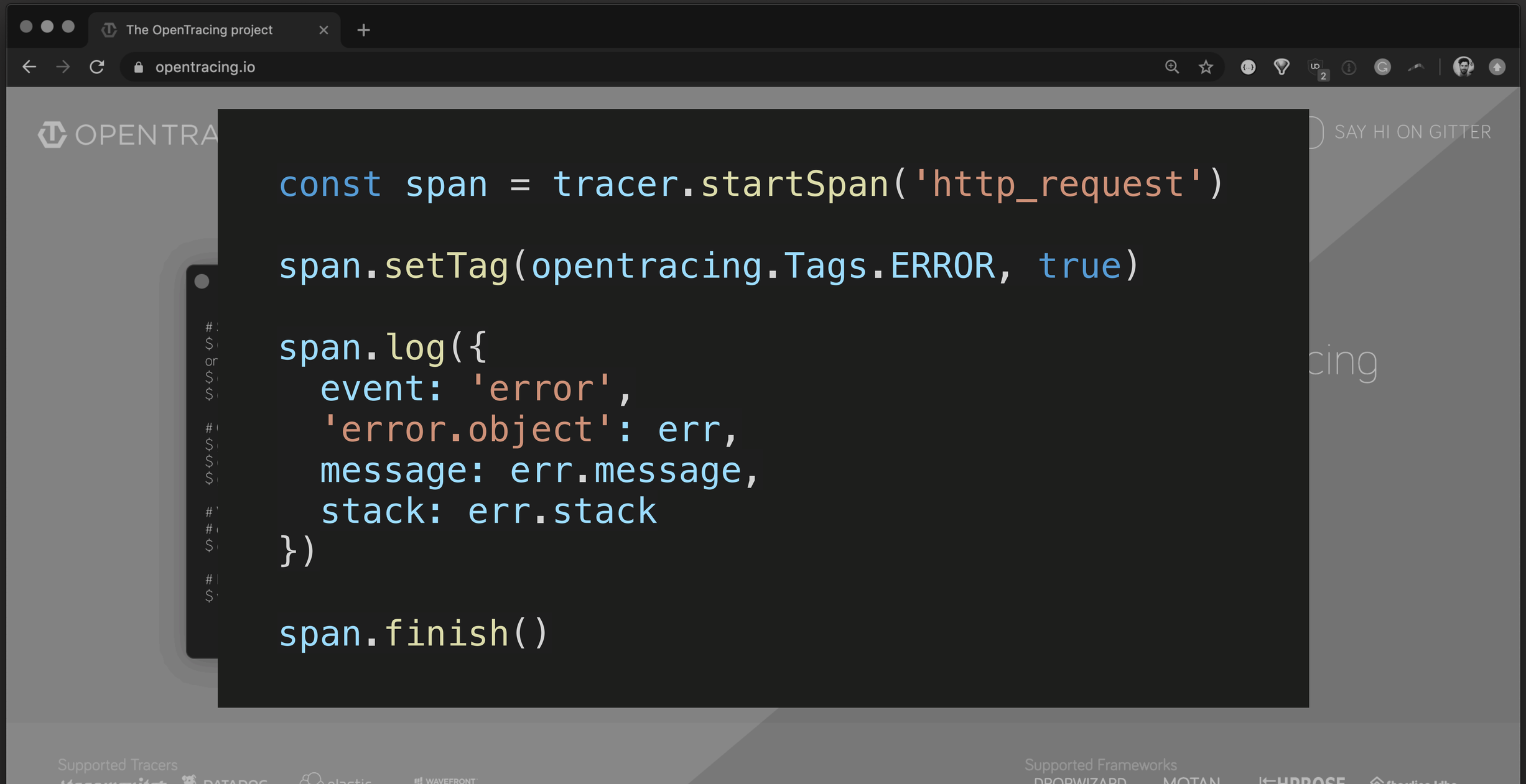


wa7son

opentracing.io



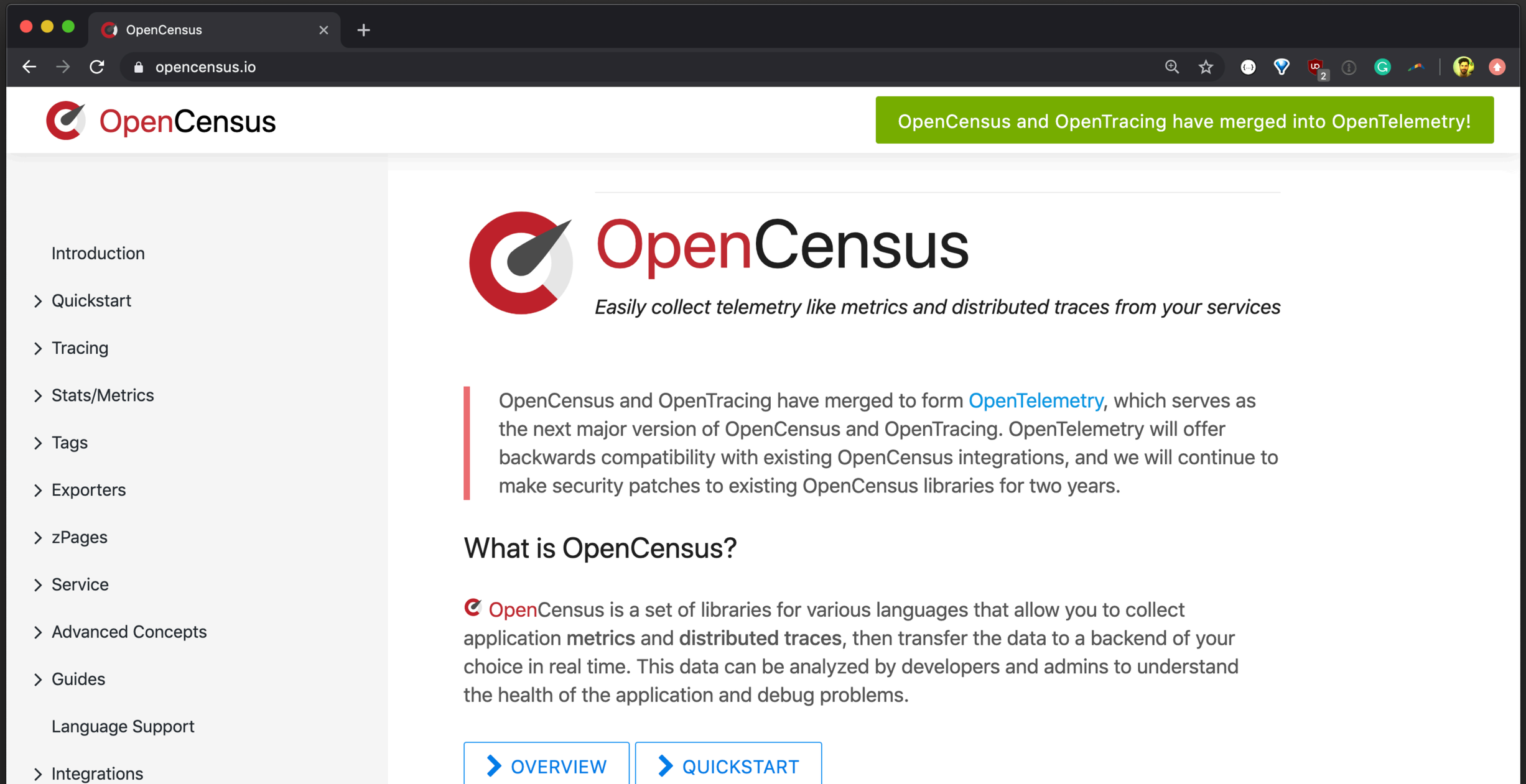
opentracing.io




```
const span = tracer.startSpan('http_request')

span.setTag(opentracing.Tags.ERROR, true)

span.log({
  event: 'error',
  'error.object': err,
  message: err.message,
  stack: err.stack
})

span.finish()
```

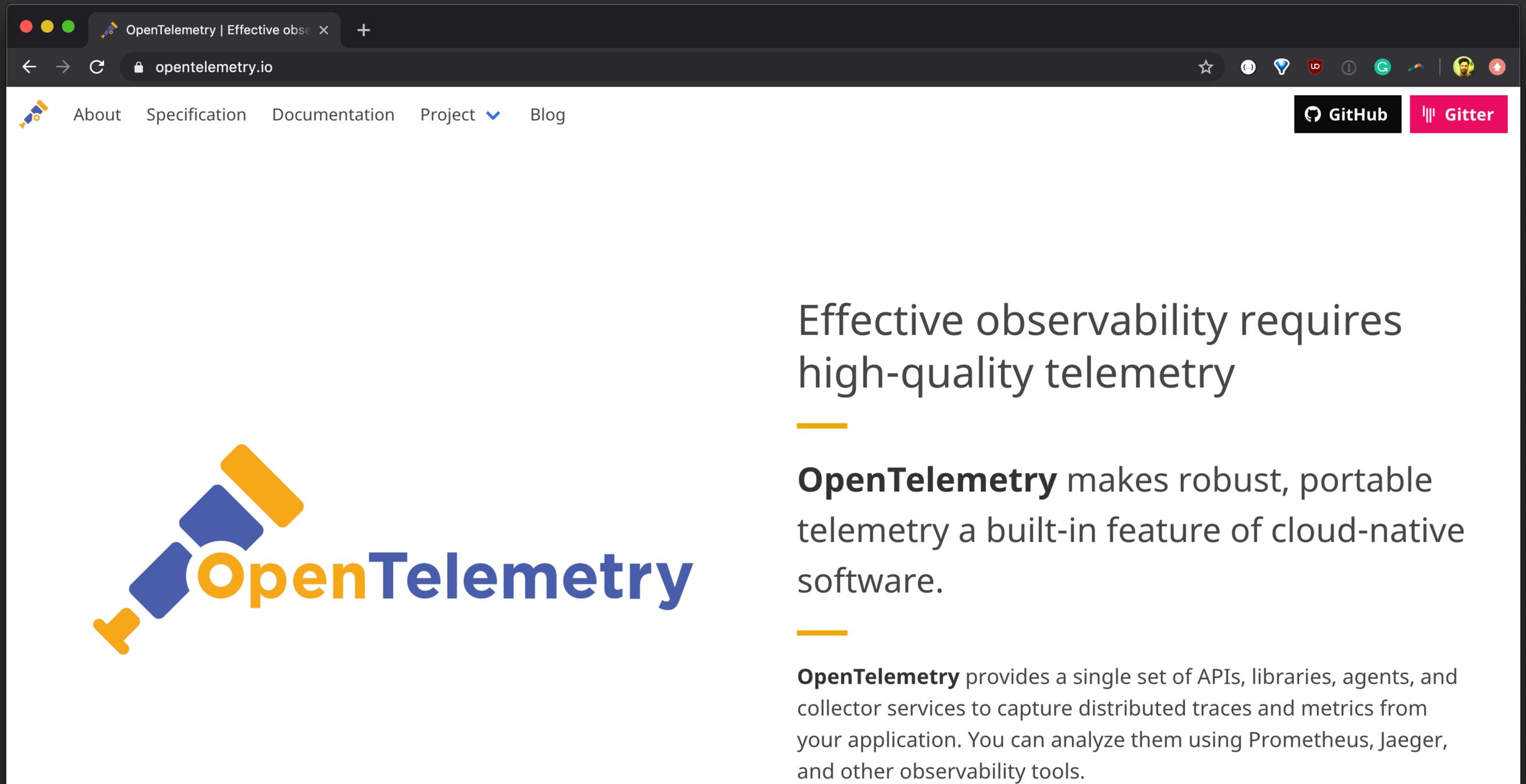
wa7son

opencensus.io



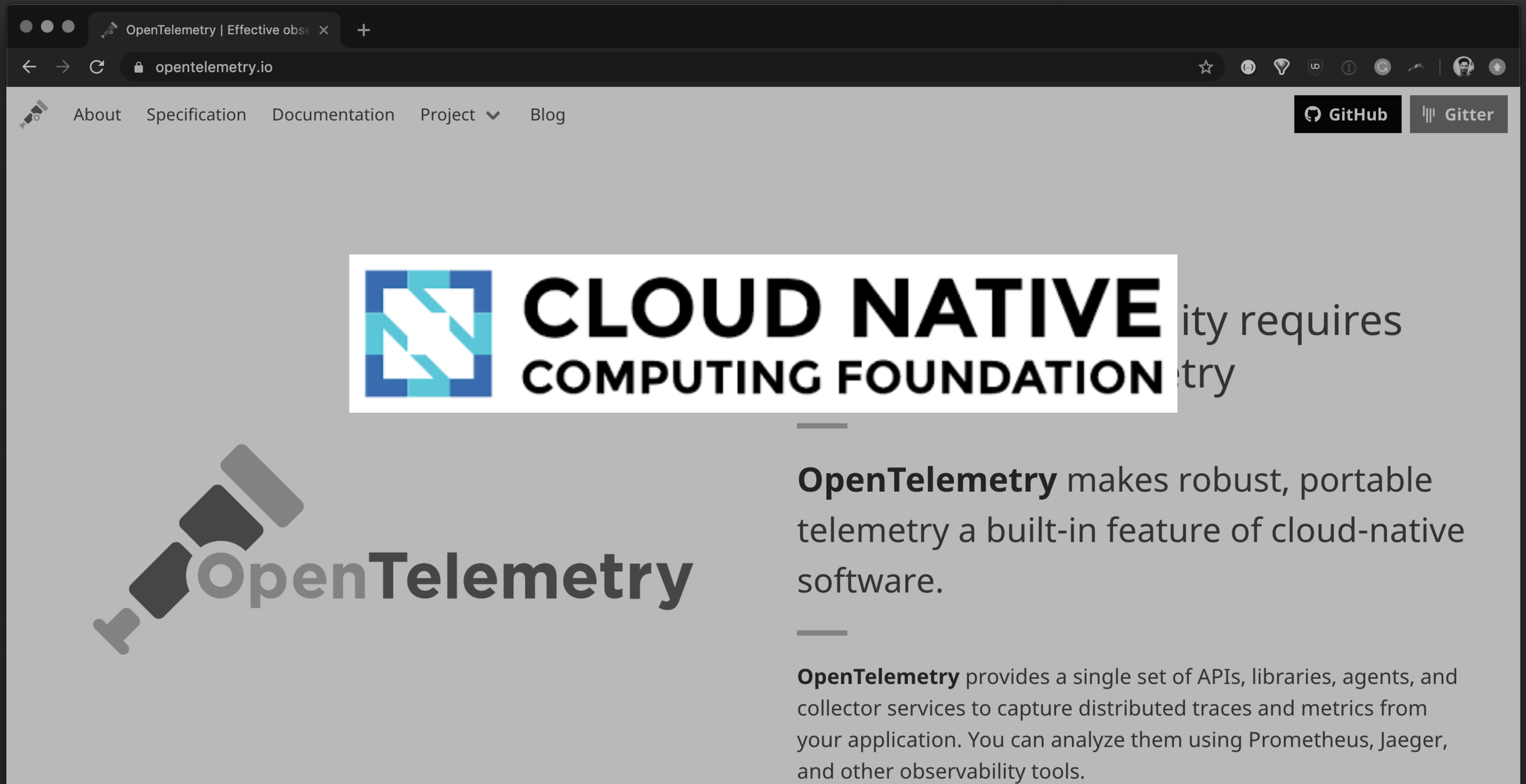
wa7son

opentelemetry.io



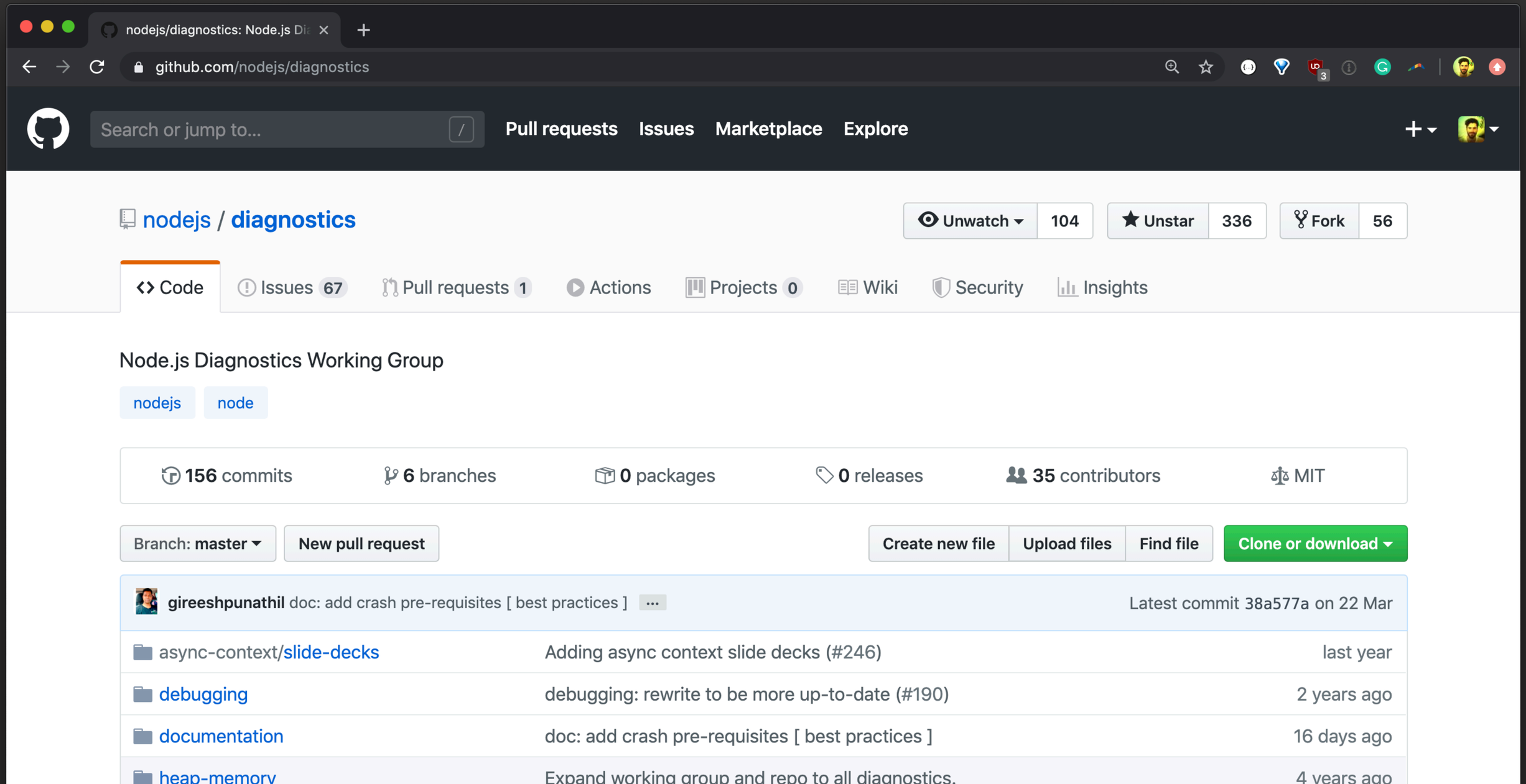
wa7son

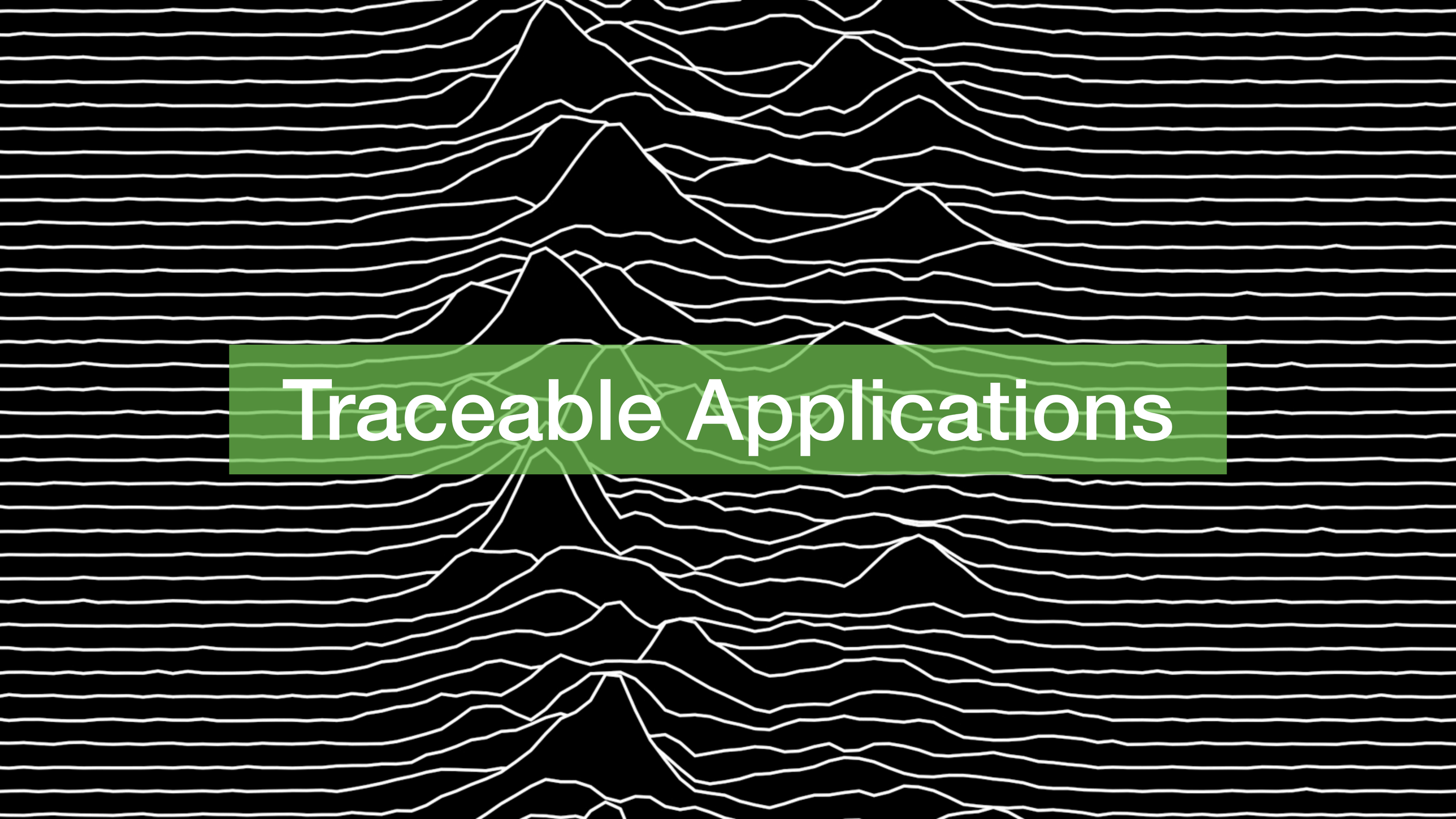
opentelemetry.io



wa7son

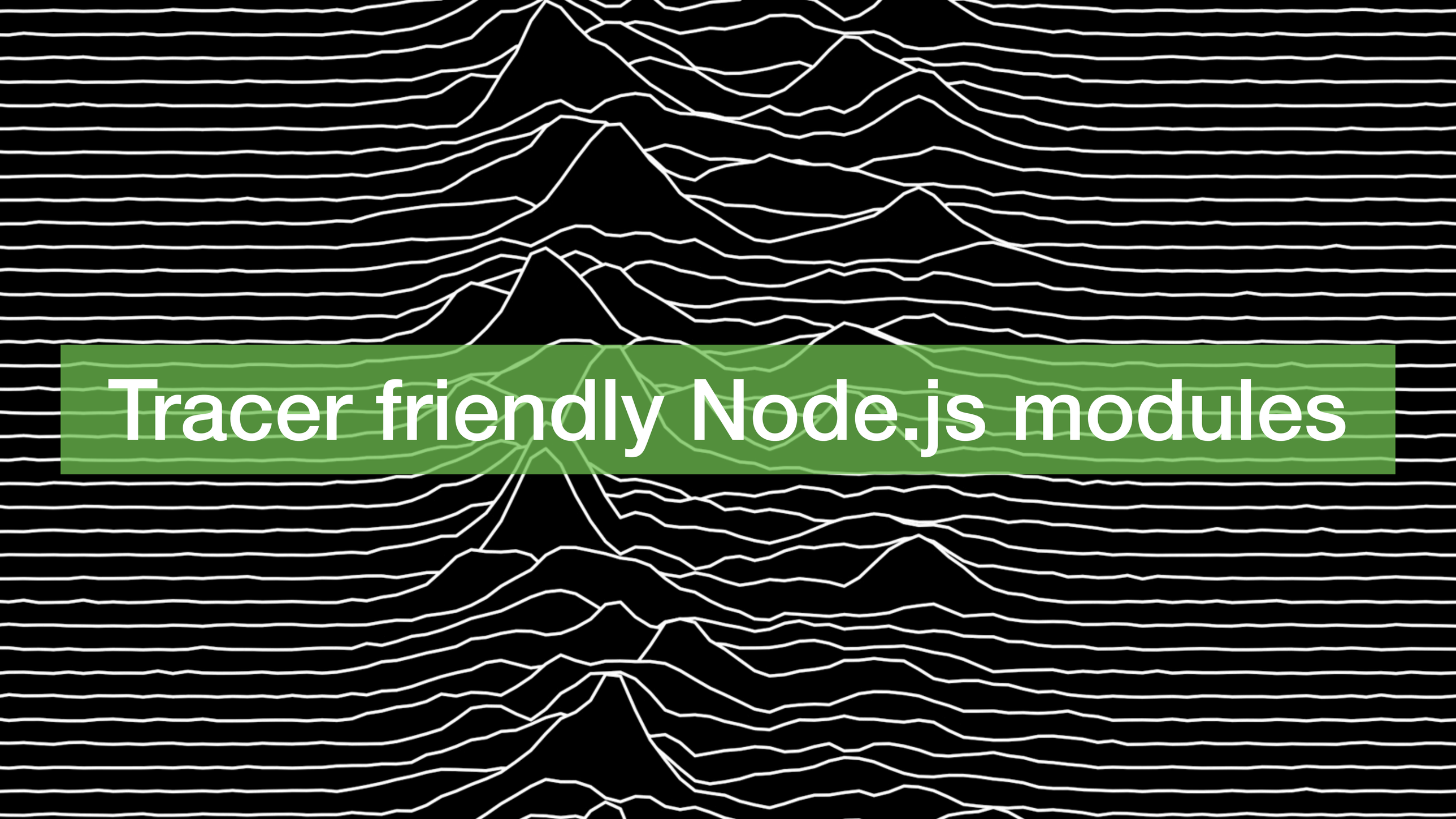
github.com / nodejs / diagnostics




Traceable Applications

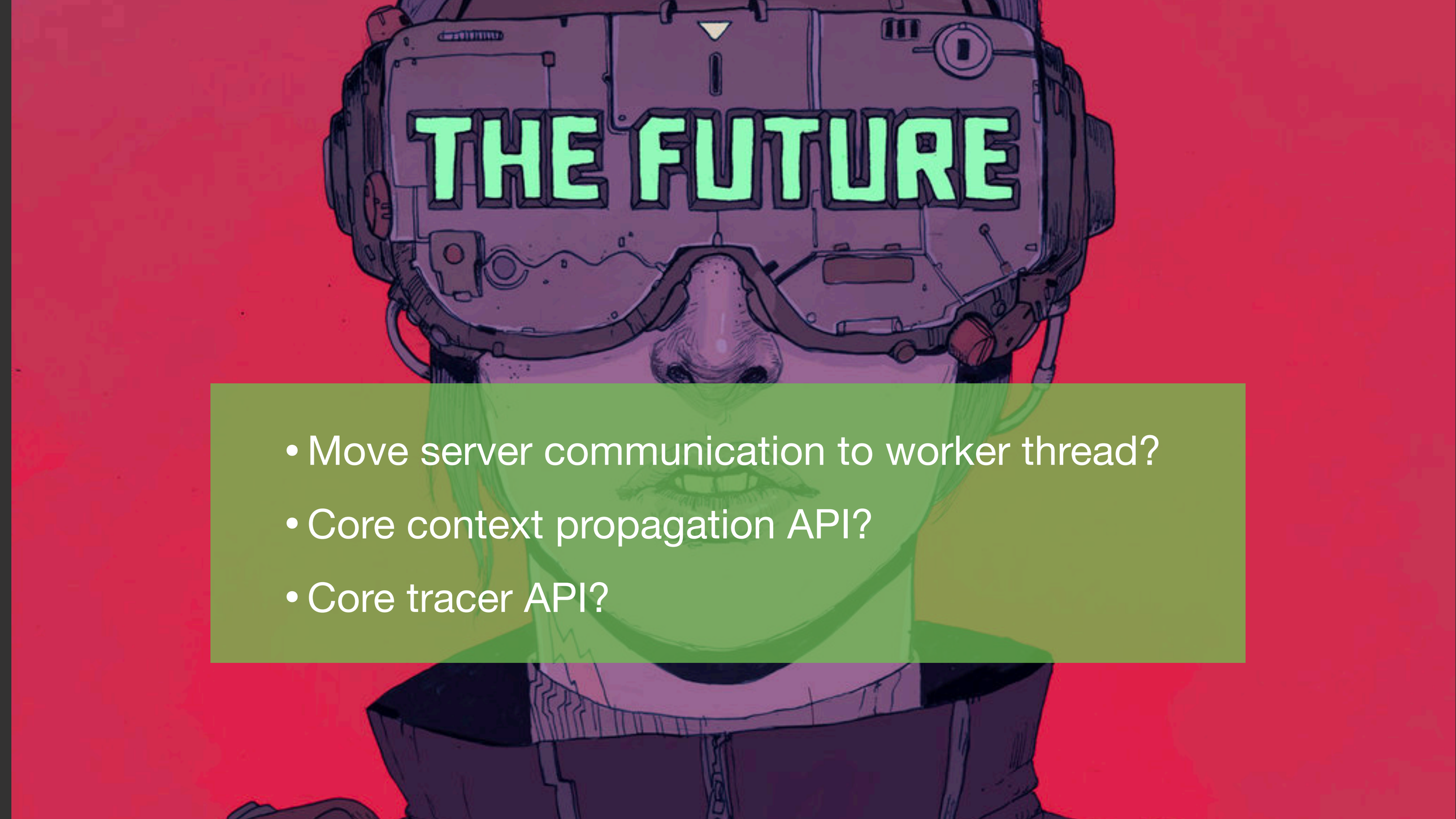
Traceable Applications

- Bad:
  - Async/await + thenables
- Challenges:
  - Userland callback queue
  - Custom protocols, message queues etc
  - Storing all data


Tracer friendly Node.js modules

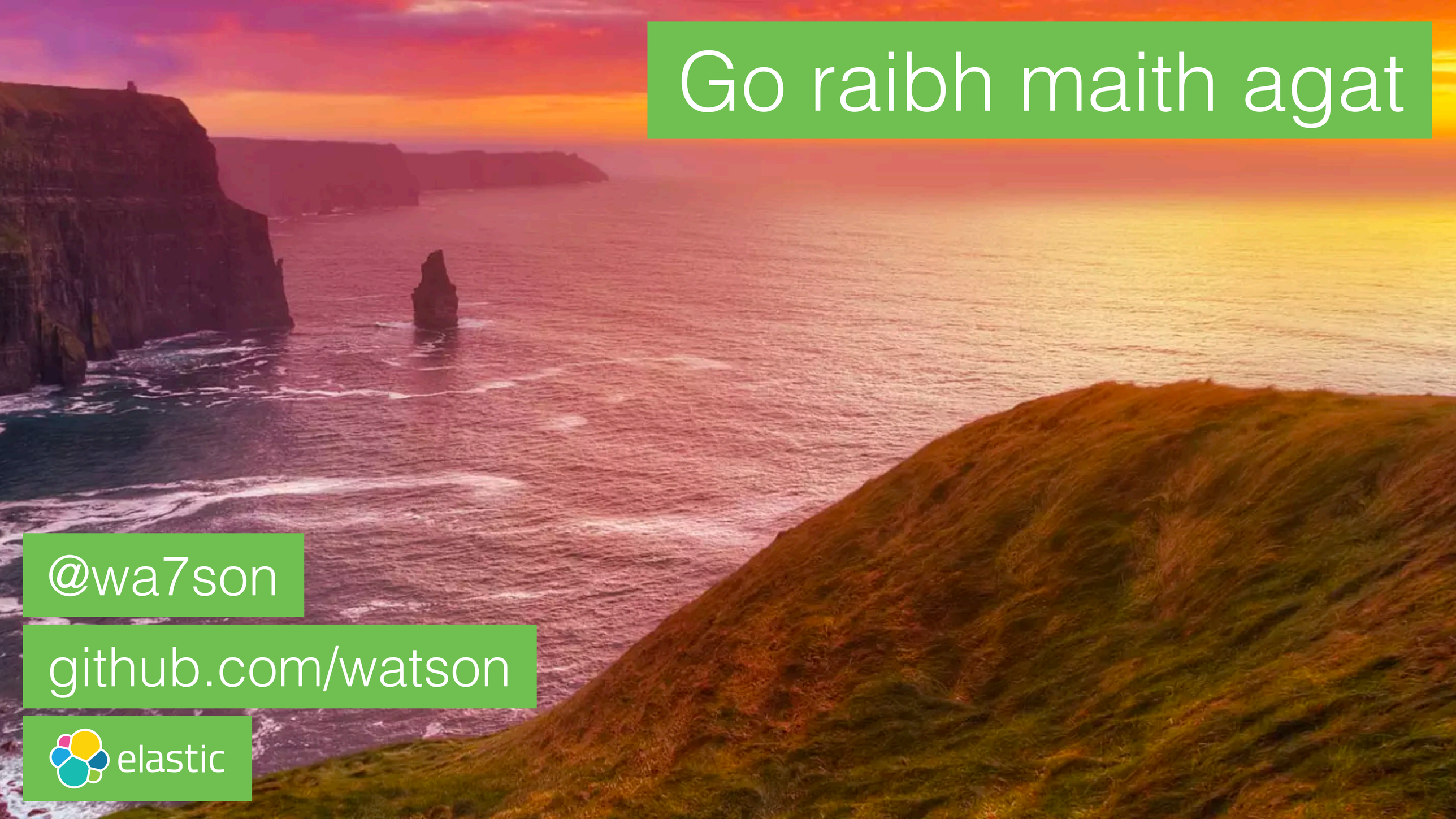
Tracer friendly Node.js modules

- Bad:
  - Userland callback queue
  - Async/await + thenables
- Want:
  - Hooks



- Move server communication to worker thread?
- Core context propagation API?
- Core tracer API?


Mmm.


Go raibh maith agat
@wa7son
github.com/watson
elastic