View online

Download PDF

Node.js HTTP parser, what's going on?

Paolo Insogna

Node.js TSC, Principal Engineer @ **Platformatic**

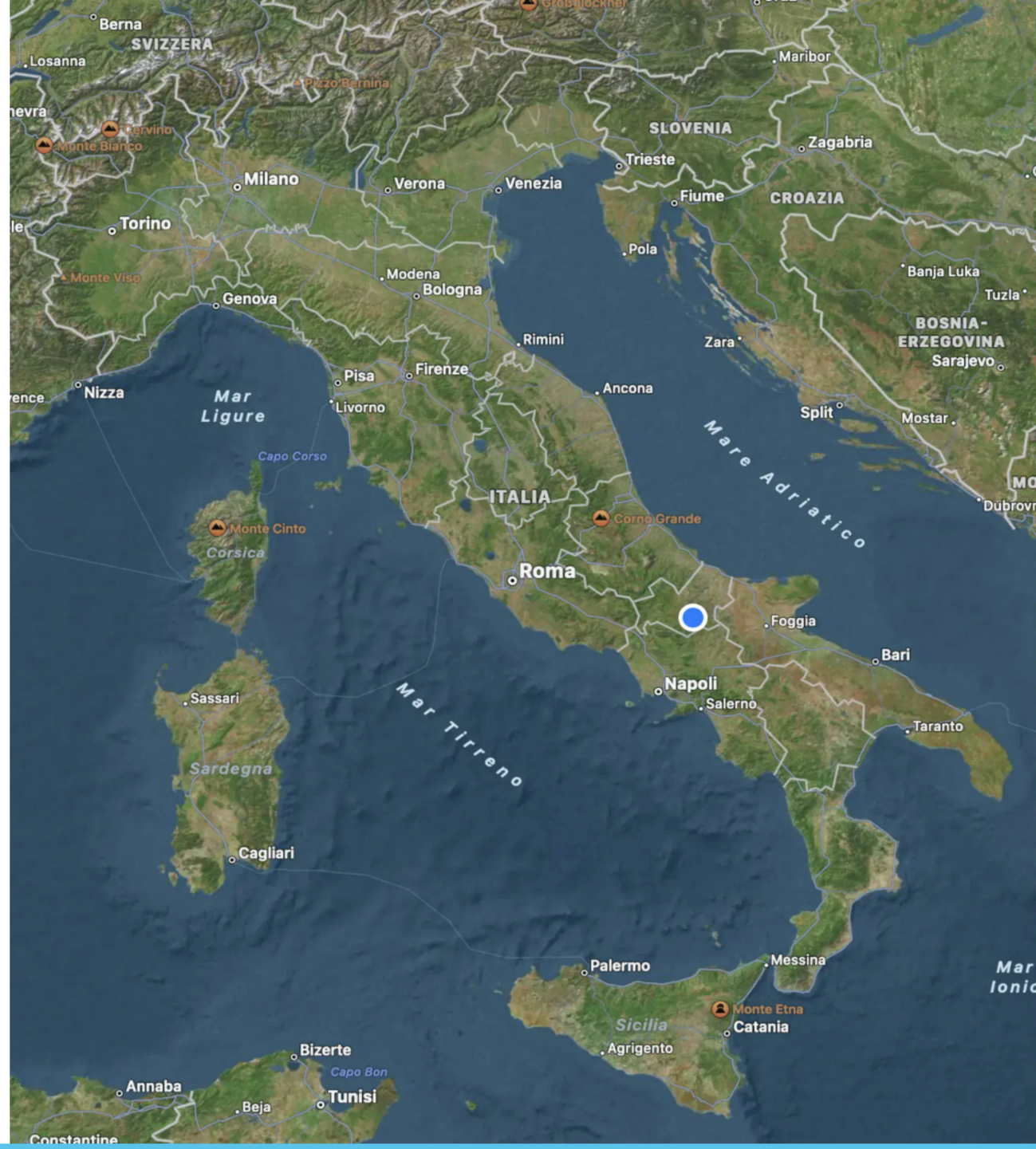
Nothing beats a classic!

Hello, I'm **Paolo**!

Node.js Technical Steering Committee Member

Platformatic Principal Engineer

paoloinsogna.dev ShogunPanda p_insogna pinsogna



We all love HTTP!

Which HTTP are you?

The choice is narrow

Even if the HTTP protocol is more than 30 years old, only **three** current versions of it exist as of today. The others are considered obsolete.

1 **HTTP/1.1**
The last version of the initial protocol. By far the most famous and most used.

2 **HTTP/2**
Developed on top of SPDY to remove some problems of HTTP.

3 **HTTP/3**
Developed on top of QUIC to solve TCP problems.

HTTP/1.1: Can't beat a classic

The grandparent
Developed in 1989 and by far the mostly used (80% of websites).

Textual protocol
Parsing can be easily implemented in any language.

Keep-Alive and FIFO pipelining
A single TCP connection can be reused.

HTTP/2: Don't break too much

The parent
Development started in 2012 and end up being standardized in 2015.

Baby-steps
The semantics are thes same, only the TCP connection usage has changed.

Multiplexing and real parallelism
It uses a SPDY derived communication protocol for better multiplexing.

HTTP/3

The sibling
Development started in 2018 and end up being standardized in 2022.

Farewell, TCP!
QUIC is now used, which means HTTP has switched to UDP.

No head of line blocking
Using multiplexing over UDP minimizes problems introduced by lost packets.

What about Node.js?

1+2

HTTP/1.1 and HTTP/2

Node.js has a stable implementation.

3

HTTP/3

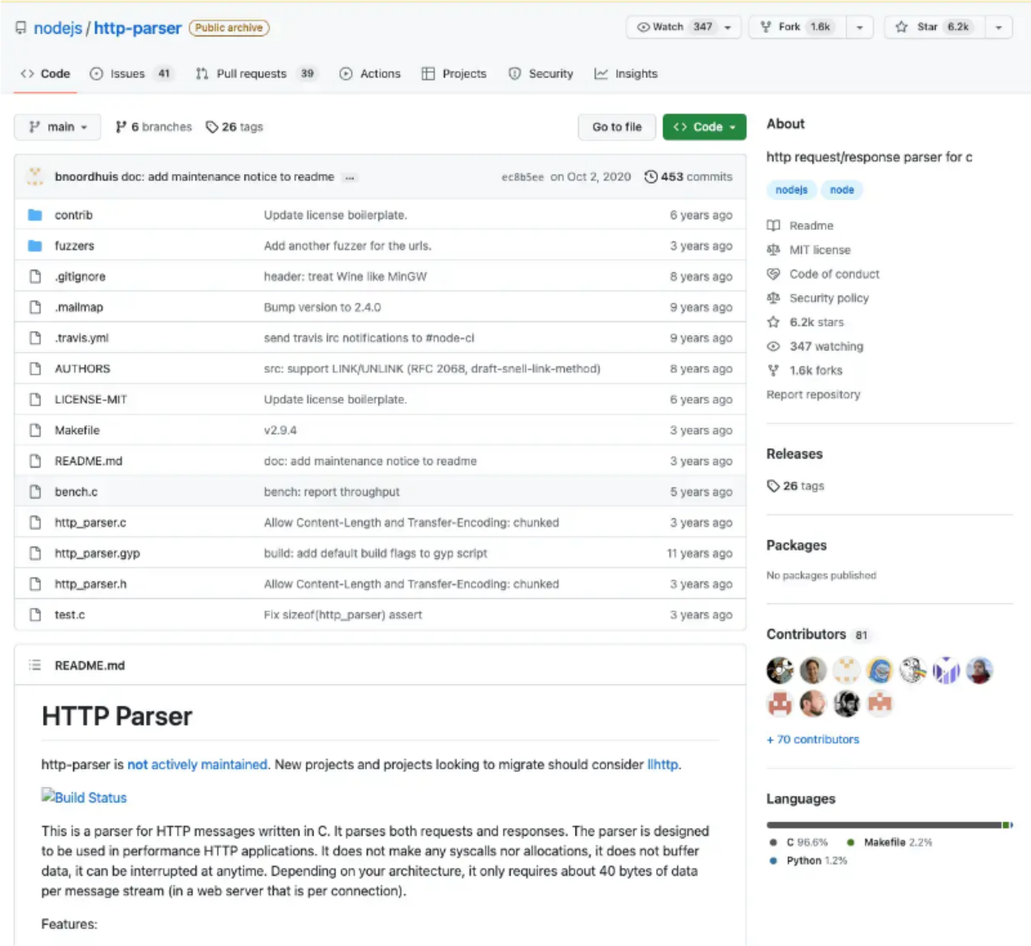
Work in progress, **stay tuned!**

Let's focus!

The original parser

http_parser was the original HTTP/1.1 parser of Node.js

It existed since the early days of Node.js and it was one of its older dependencies.



http_parser: the goods

Good performance

The original parser had good performance, around 1.5 million requests per second.

Lenient

It supported both HTTP/0.9, HTTP/1.0 and HTTP/1.1 specs and non compliant clients.

Well tested

It had a comprehensive and battle tested test suite.

http_parser: the bads

Written in C
It is not the most user friendly language.

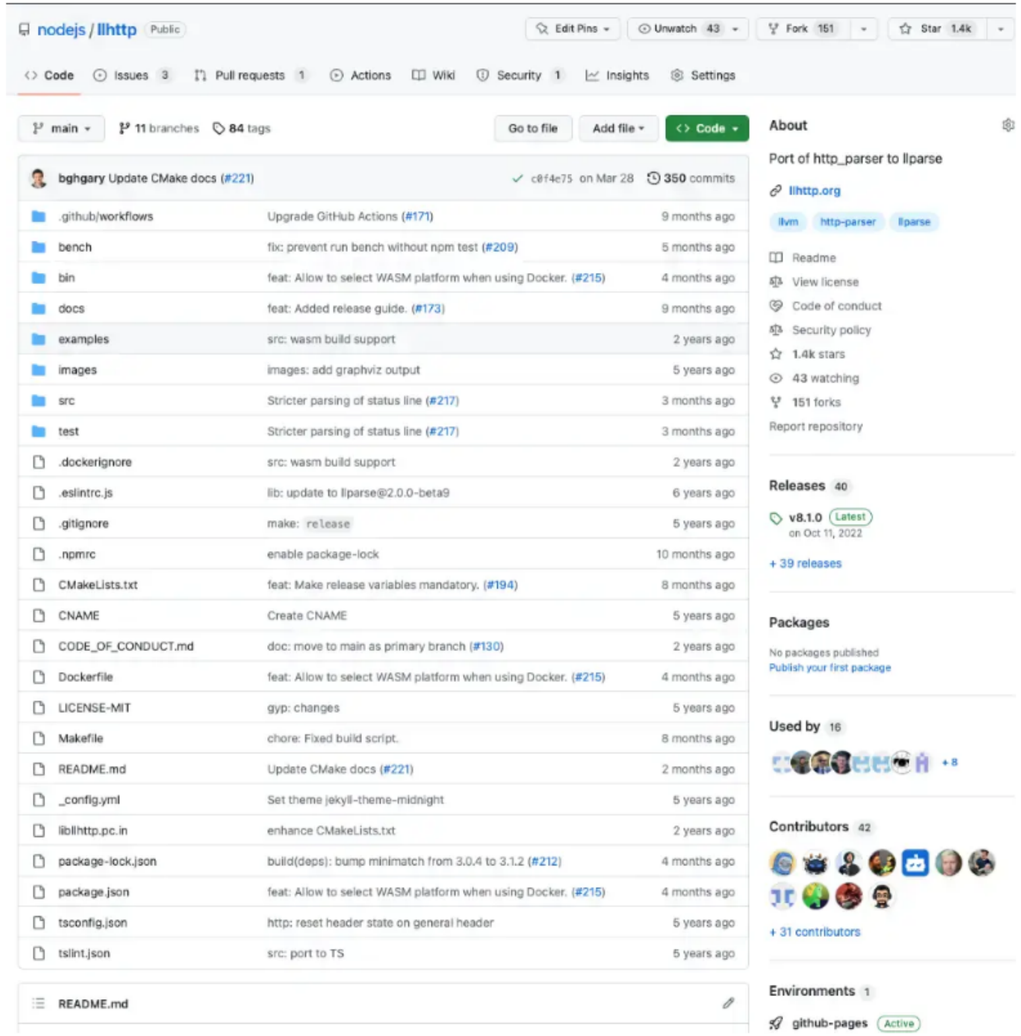
Hard to maintain
Over the course of ten years, the codebase became unmaintainable.

Vulnerability-prone
The maintanability problem made it also hard to promptly address bugs and vulnerabilities.

The current parser

llhttp is the current HTTP parser.

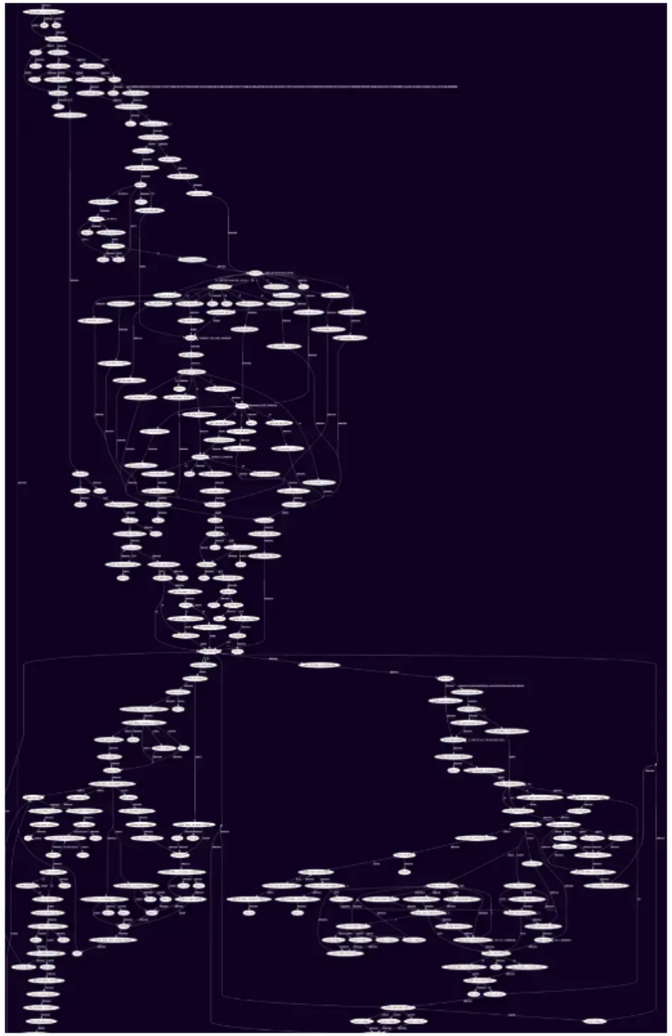
Written by Fedor Indutny in 2019, is the default since Node.js 12.



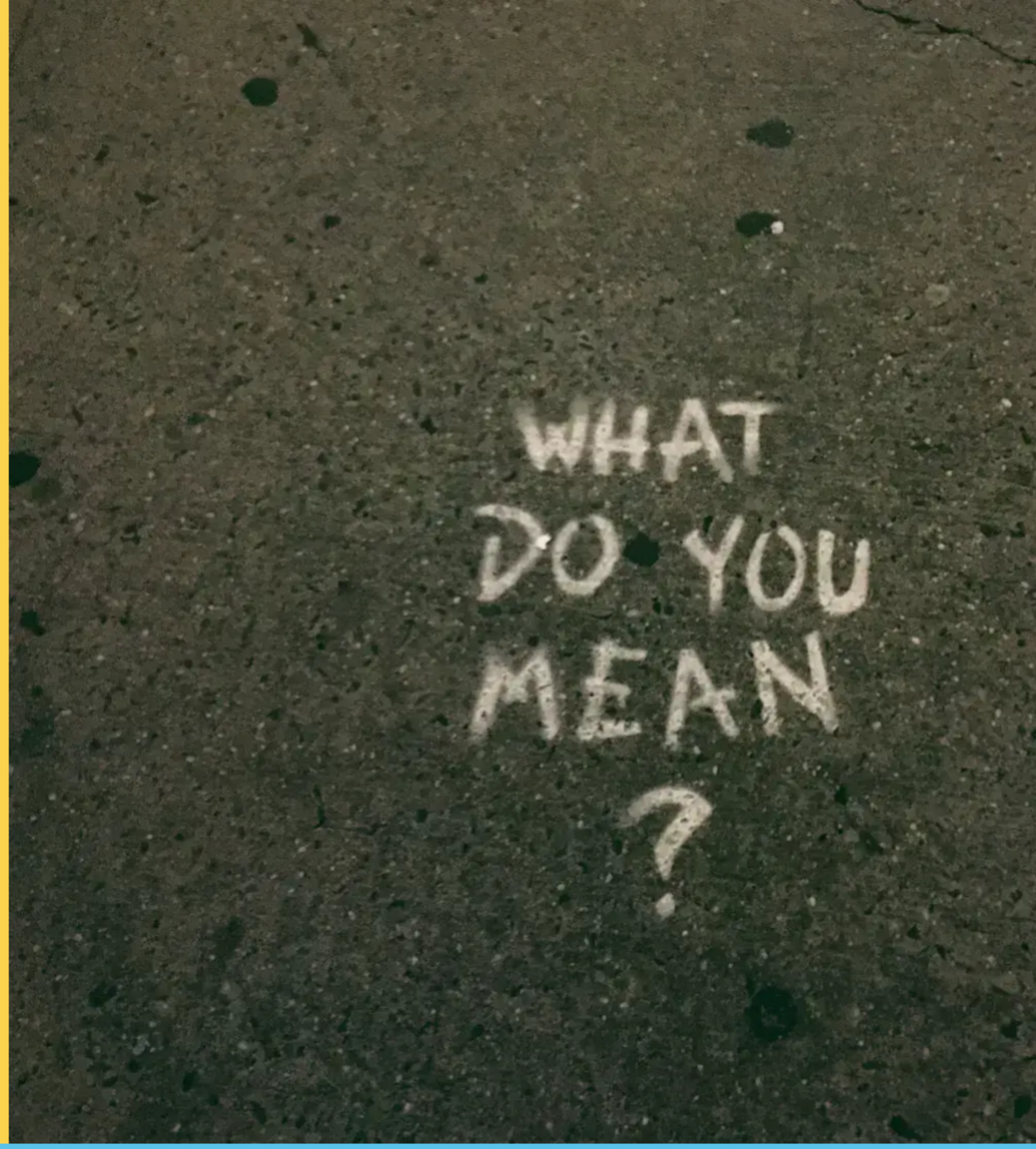
How does it work?

llhttp is a state based HTTP parser based on llparse.

llparse is capable to generate a very performant C code out of a TypeScript description of the possible states.



Wait, what?!



Yes, you got it right!

There's more!

The http_parser test suite has been ported and extended in llhttp.

The test suite is now described in **Markdown**, transpiled at runtime to a **C source code** and then compiled and executed.

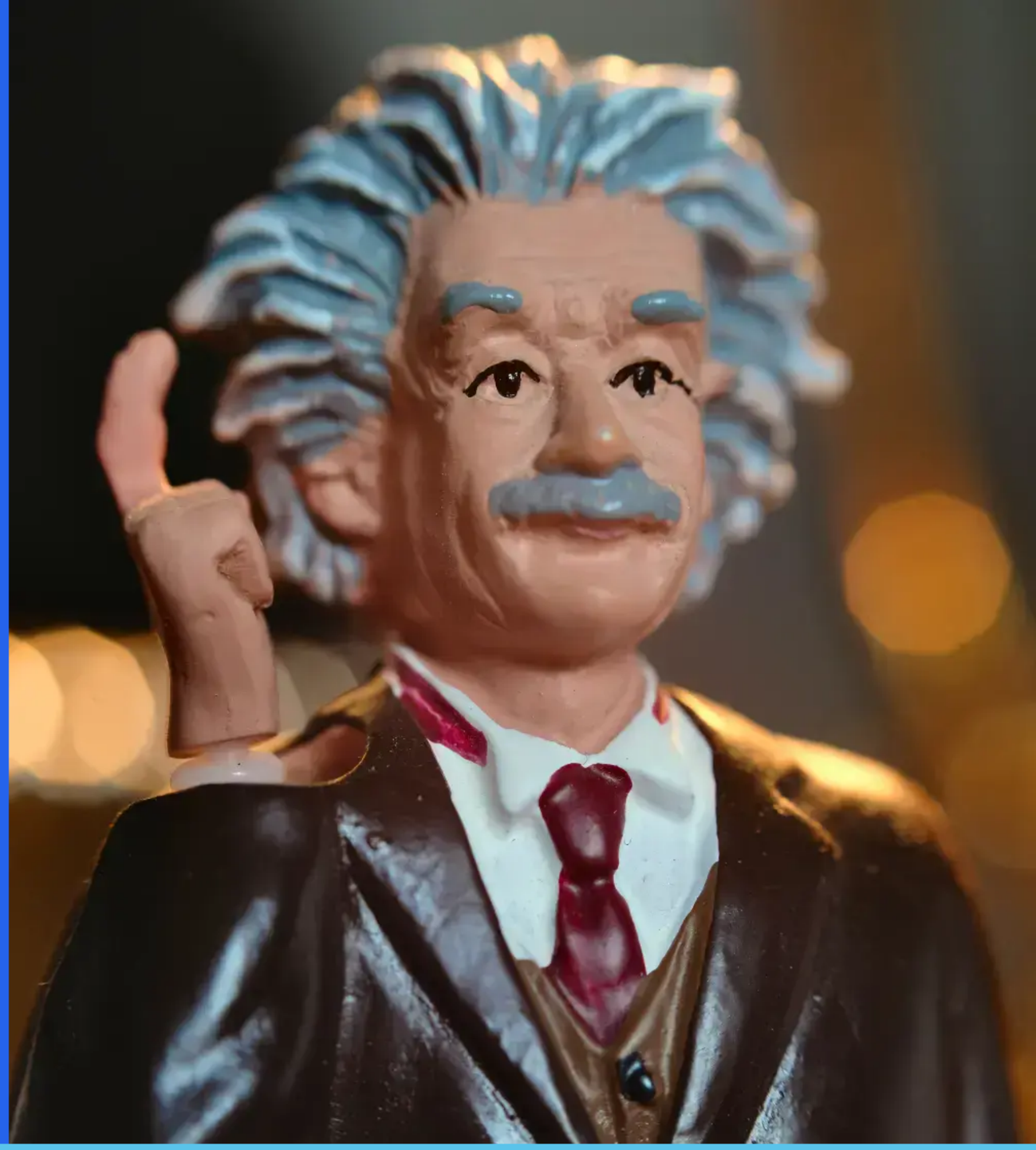
```
### Invalid HTTP version with lenient


```http
GET / HTTP/5.6


```


```log
off=0 message begin
off=0 len=3 span[method]="GET"
off=3 method complete
off=4 len=1 span[url]="/"
off=6 url complete
off=11 len=3 span[version]="5.6"
off=14 version complete
off=18 headers complete method=1 v=5/6 flags=0 content_length=0
off=18 message complete
```
```

That's a genius in action!



llhttp: what's wrong with it?

Hard to debug and release
The transpilation makes hard to debug issues.

Backward compatibility
Supporting obsolete versions of HTTP introduces unneeded complexity.

You give them a finger, they take the arm
Leniency-prone approach opens the door for a lot of edge cases.

Where are the docs?

Do we have the solution?

Yes, start fresh!

Keep the goods

Piece of art design

llhttp has a wonderful
and performant architecture.

Testability

The test suite is invaluable
to ensure correctness.

What shall we change?

Better transpilation

Try to transpile to a human readable form and then generate static libs and include headers.

Use a higher level language

Rust is a good candidate that can be used for both developer input and transpilation output.

Modern and strict

Only support latest HTTP/1.1 standard (RFC 9110) and initially do have any leniency logic.

Work in progress, stay tuned!

One last thing™

“If there is no struggle, there is no progress.”

Frederick Douglass

Thank you!

Paolo Insogna

Node.js TSC, Principal Engineer

@p_insogna

paolo.insogna@platformatic.dev

Platformatic