CityJS™
ATHENS
2024

Anna Henningsen
Staff Engineer @ MongoDB

What's in a Node.js Bug – A Case Study

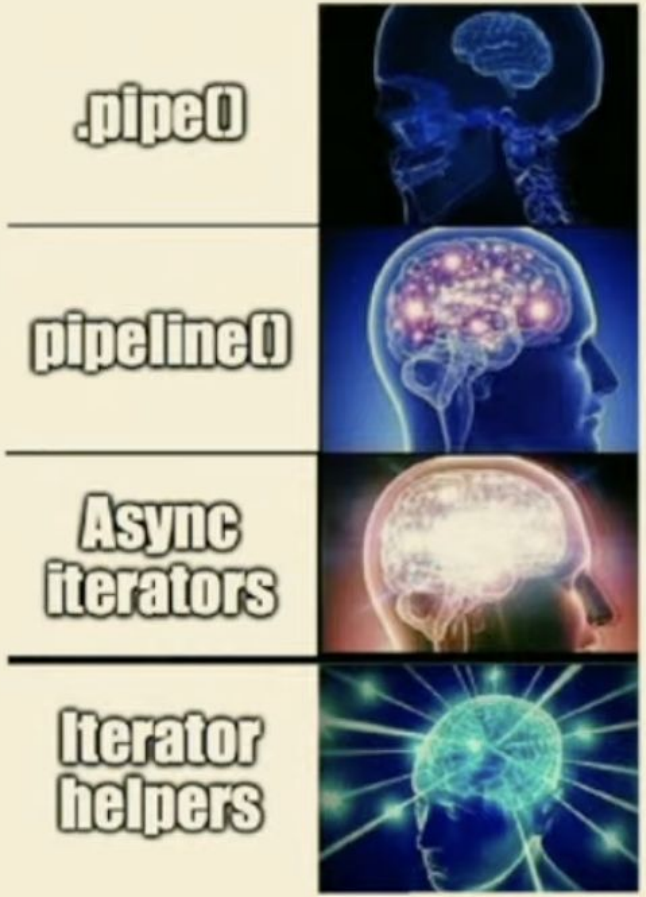
Hi all, I’m Anna!

- she/her
- Currently Staff Engineer @ MongoDB working on Developer Tools
- Former Node.js core contributor & TSC member
- Unreasonably passionate about Character Encodings

`@addaleax.bsky.social`

`https://addaleax.net/cityjs-athens24.pdf`

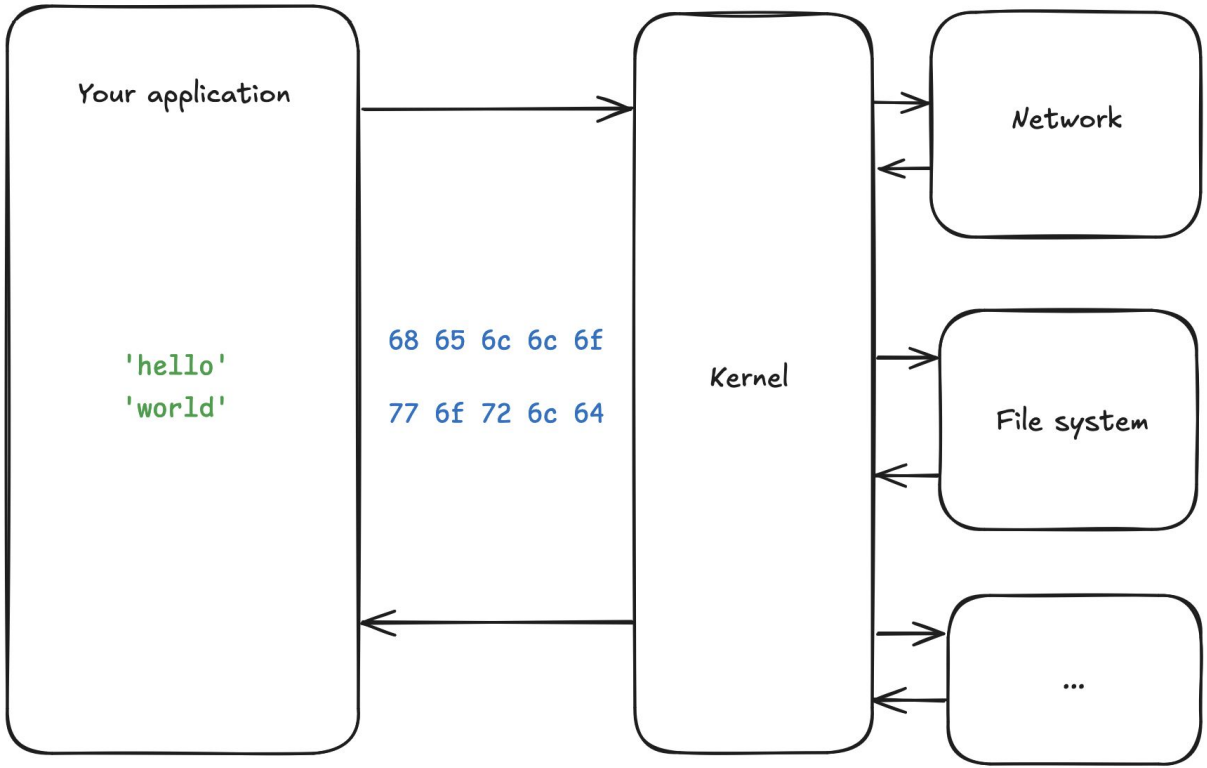
```
import { createReadStream } from "node:fs";

createReadStream(import.meta.filename)
  .map((data) => data.toString().toUpperCase())
  .compose(process.stdout);
```



N_

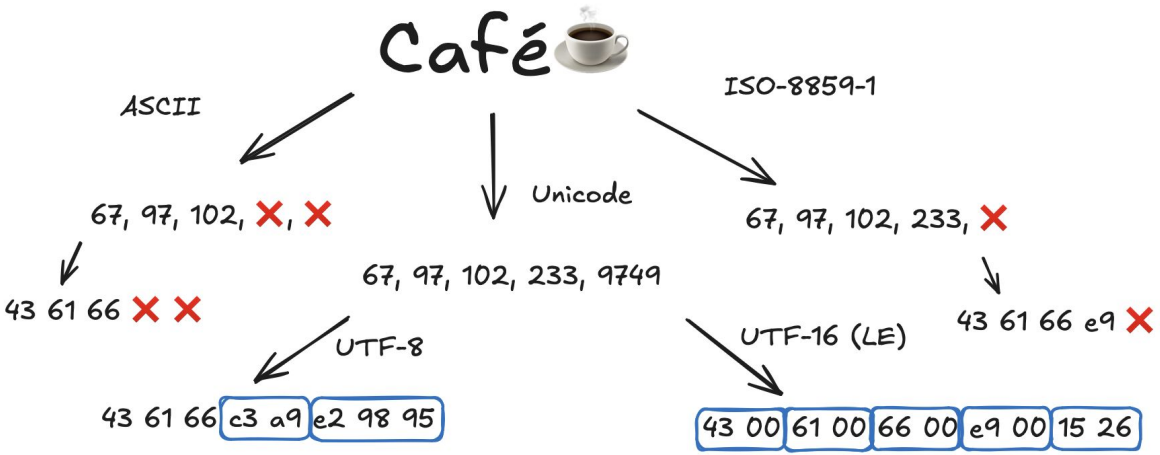
Before we dive in: A quick refresh

What's a Character Encoding?

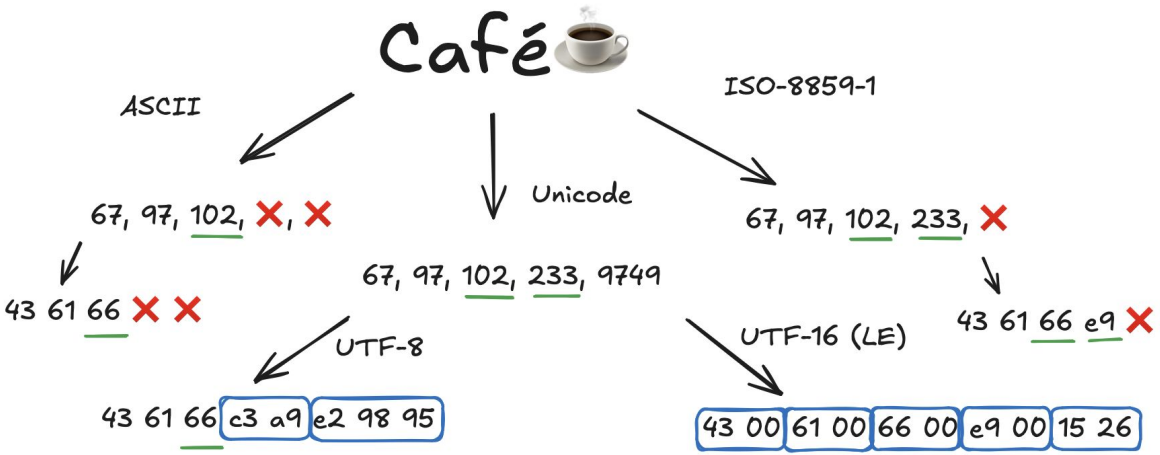


What's a Character Encoding?

What's a Character Encoding?



What's a Character Encoding?



JS uses UTF-16, right...?

JS uses UTF-16, right...?

Actually, JS engines are *smart*:

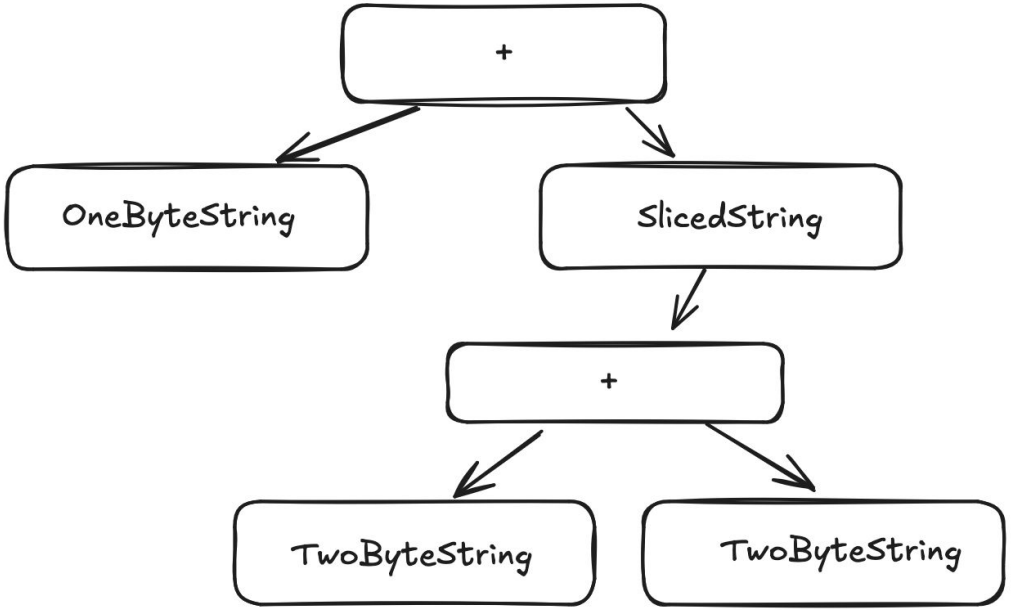
```
$ grep 'V8_OBJECT class' deps/v8/src/objects/string.h
V8_OBJECT class String : public Name {
V8_OBJECT class InternalizedString : public String{
V8_OBJECT class SeqOneByteString : public SeqString {
V8_OBJECT class SeqTwoByteString : public SeqString {
V8_OBJECT class ConsString : public String {
V8_OBJECT class ThinString : public String {
V8_OBJECT class SlicedString : public String {
V8_OBJECT class UncachedExternalString : public String {
V8_OBJECT class ExternalString : public UncachedExternalString {
V8_OBJECT class ExternalOneByteString : public ExternalString {
V8_OBJECT class ExternalTwoByteString : public ExternalString {
```

JS uses UTF-16, right...?

```
'a' + 'bcd\u2615f'.repeat(3).substring(2)
```

JS uses UTF-16, right...?

```
$ node --allow-natives-syntax -e "%DebugPrint(
     'a' + 'bcd\u2615f'.repeat(3).substring(2))"
DebugPrint: 0x276f94d8359: [String]: uc"ad\u2615fbcd\u2615fbcd\u2615f"
0x3b2096501c31: [Map] in
 - type: CONS_STRING_TYPE
```



What happened in Node.js 22.7.0?

Ouch!

UTF+8 encodings are broken #54543

Closed **nikhilro** opened this issue on Aug 24 · 18 comments

Buffer.from() output breaks after

Closed **unilynx** opened this issue on Aug 23 · 20 comments

nikhilro co

Textencoder decode to UTF8 with fatal true th… #54628

Version

22.7.0

Platform

Closed **eldoy** opened this

Flaky text encoding regression in Node 22 #54718

ent

How often does it reproduce? Is there a required condition?

The required condition is that the fetched text should have special characters. It seems to happen sporadically, and is not possible to recreate deterministically.

Buffer to string conversion fails after a few KB of load. #54738

Closed **ThorstenEngel** opened this issue on Sep 3 · 1 comment

What steps will reproduce the bug?

Hey everyone, I'm not sure how to reproduce but latest node can't parse UTF+8 anymore. It works for the first minute or two (or couple hours if I remove Datadog APM instrumentation) but then returns garbage on the same request. I'm just using postgres.js to fetch and nest.js for the HTTP server. No fancy buffer manipulation.

Wait ... what?!

Flaky text encoding regression

It seems to happen sporadically, and is not possible to recreate deterministically.

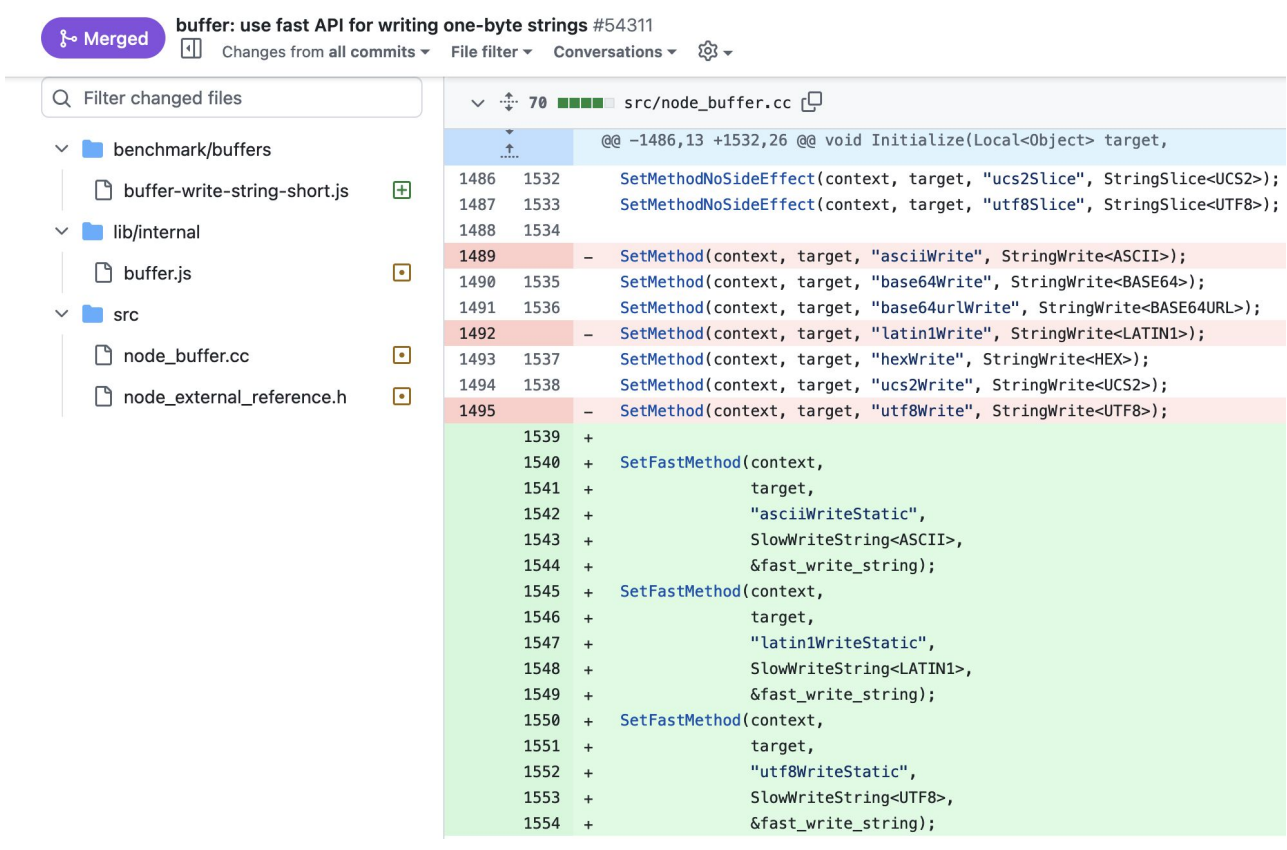
fails after a few KB of load

It works for the first minute or two

Step 1: Minimal reproduction

```
$ cat test.js
for (let i = 0; ; i++) {
  const str = Buffer.from('café', 'utf8').toString('utf8');
  if (str !== 'café') {
     console.error('Bad result after ' + i + ' iterations: ' + str);
     break;
  }
}
$ node test.js
Bad result after 10279 iterations: caf
$ node test.js
Bad result after 9583 iterations: caf
```

Step 2: git bisect to the rescue!



Step 3: 3x fast_write_string...?

```
1540  +  SetFastMethod(context,
1541  +                target,
1542  +                "asciiWriteStatic",
1543  +                SlowWriteString<ASCII>,
1544  +                &fast_write_string);
1545  +  SetFastMethod(context,
1546  +                target,
1547  +                "latin1WriteStatic",
1548  +                SlowWriteString<LATIN1>,
1549  +                &fast_write_string);
1550  +  SetFastMethod(context,
1551  +                target,
1552  +                "utf8WriteStatic",
1553  +                SlowWriteString<UTF8>,
1554  +                &fast_write_string);
```

Step 4: Let's play Find the Bug

```
uint32_t FastWriteString(Local<Value> receiver,
                         const v8::FastApiTypedArray<uint8_t>& dst,
                         const v8::FastOneByteString& src,
                         uint32_t offset,
                         uint32_t max_length) {
  uint8_t* dst_data;
  CHECK(dst.getStorageIfAligned(&dst_data));
  CHECK(offset <= dst.length());
  CHECK(dst.length() - offset <= std::numeric_limits<uint32_t>::max());

  max_length = std::min<uint32_t>(dst.length() - offset, max_length);

  memcpy(dst_data, src.data, max_length);

  return max_length;
}
```

Why did it happen?

Node.js contributors care about performance

- Deno, Bun & friends are claiming significant performance benefits over Node.js
- Optimizing Node.js corresponds to real $$$ savings for users
- Optimizing Node.js is a fairly viable path to finding ways to contribute to Node.js

The ‘Fast API’ in V8

- Historically, C++ ↔ JS boundary crossings have been some of the most expensive parts of JS apps
- V8’s Fast API provides a way to *directly* call C++ code from JS
- The Fast API requires some pre-conditions to be met, e.g. “nice data layout”, “never triggers GC”, “no calls back into JS”, etc.

The ‘Fast API’ in V8

- Historically, C++ ↔ JS boundary crossings have been some of the most expensive parts of JS apps
- V8’s Fast API provides a way to *directly* call C++ code from JS
- The Fast API requires some pre-conditions to be met, e.g. “nice data layout”, “never triggers GC”, “no calls back into JS”, etc.

- Write a C++ function twice, once “slow”, once “fast”
- Fast calls only kick in after (potentially asynchronous) optimization!

It all comes together

```
$ cat test.js
for (let i = 0; ; i++) {
  const str = Buffer.from('café', 'utf8').toString('utf8');
  if (str !== 'café') {
     console.error('Bad result after ' + i + ' iterations: ' + str);
     break;
  }
}
$ node test.js
Bad result after 10279 iterations: caf
$ node test.js
Bad result after 9583 iterations: caf
```

JS uses UTF-16, right...?

Actually, JS engines are *smart*:

```
$ grep 'V8_OBJECT class' deps/v8/src/objects/string.h
V8_OBJECT class String : public Name {
V8_OBJECT class InternalizedString : public String{
V8_OBJECT class SeqOneByteString : public SeqString { // ISO-8859-1
V8_OBJECT class SeqTwoByteString : public SeqString { // UTF-16
V8_OBJECT class ConsString : public String {
V8_OBJECT class ThinString : public String {
V8_OBJECT class SlicedString : public String {
V8_OBJECT class UncachedExternalString : public String {
V8_OBJECT class ExternalString : public UncachedExternalString {
V8_OBJECT class ExternalOneByteString : public ExternalString {
V8_OBJECT class ExternalTwoByteString : public ExternalString {
```

Found the bug!

```
uint32_t FastWriteString(Local<Value> receiver,
                         const v8::FastApiTypedArray<uint8_t>& dst,
                         const v8::FastOneByteString& src,
                         uint32_t offset,
                         uint32_t max_length) {
  uint8_t* dst_data;
  CHECK(dst.getStorageIfAligned(&dst_data));
  CHECK(offset <= dst.length());
  CHECK(dst.length() - offset <= std::numeric_limits<uint32_t>::max());

  max_length = std::min<uint32_t>(dst.length() - offset, max_length);

  memcpy(dst_data, src.data, max_length);

  return max_length;
}
```

Found the bug!

```
uint32_t FastWriteString(Local<Value> receiver,
                         const v8::FastApiTypedArray<uint8_t>& dst,
                         const v8::FastOneByteString& src,
                         uint32_t offset,
                         uint32_t max_length) {
  uint8_t* dst_data;
  CHECK(dst.getStorageIfAligned(&dst_data));
  CHECK(offset <= dst.length());
  CHECK(dst.length() - offset <= std::numeric_limits<uint32_t>::max());

  max_length = std::min<uint32_t>(dst.length() - offset, max_length);

  memcpy(dst_data, src.data, max_length);

  return max_length;
}
```

Comes in as ISO-8859-1

Should write UTF-8 to `dst_data`

Don’t we have tests?

Don’t we have tests?

- Coverage indicates that the new code paths *were* taken – just not in the relevant tests
- The PR also added new benchmarks which exercised these code paths but did not check results

Don’t we have tests?

- Coverage indicates that the new code paths *were* taken – just not in the relevant tests
- The PR also added new benchmarks which exercised these code paths but did not check results
- Node.js releases go through Canary In The Goldmine (“CITGM”) ecosystem testing – all failures were triaged and none were related

What can we learn from it?

To wrap it up...

- Naming matters – people who don’t know your code will work with it
- Don’t just make sure code is covered, test each *change* you’re making

To wrap it up...

- Naming matters – people who don't know your code will work with it
- Don't just make sure code is covered, test each *change* you're making
- Character encodings can be a leaky abstraction
- This wouldn't have happened if clean breaks in character encodings weren't so unpopular

To wrap it up...

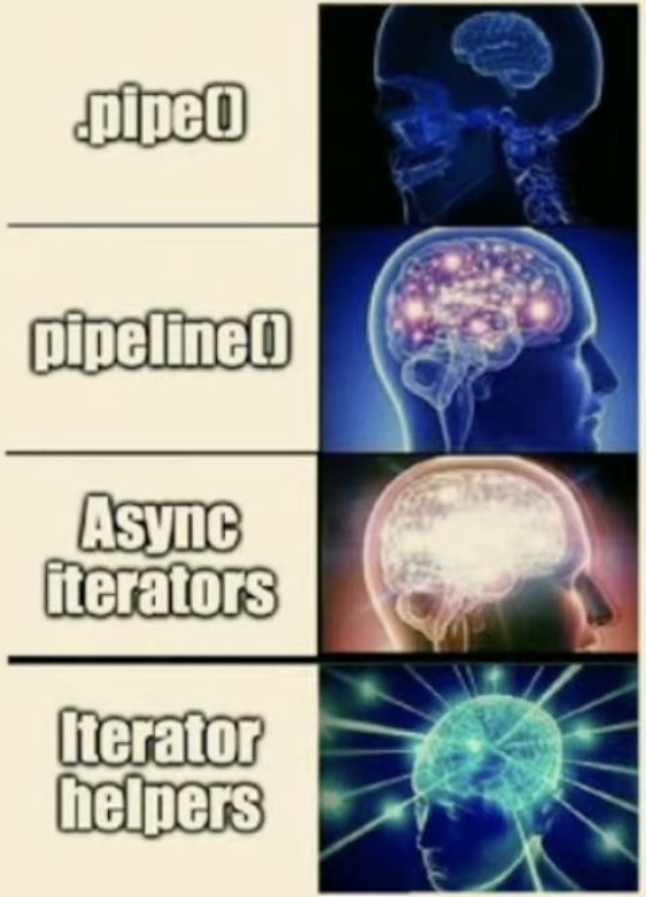
- Naming matters – people who don’t know your code will work with it
- Don’t just make sure code is covered, test each *change* you’re making
- Character encodings can be a leaky abstraction
- This wouldn’t have happened if clean breaks in character encodings weren’t so unpopular
- V8 is smart! It will store strings in many different ways and even optimize your code *while it runs*
- V8 is dumb! There is a clear gap for testability of the fast API

To wrap it up...

- Naming matters – people who don't know your code will work with it
- Don't just make sure code is covered, test each *change* you're making
- Character encodings can be a leaky abstraction
- This wouldn't have happened if clean breaks in character encodings weren't so unpopular
- V8 is smart! It will store strings in many different ways and even optimize your code *while it runs*
- V8 is dumb! There is a clear gap for testability of the fast API
- Node.js & V8 engineers constantly work on performance
- Be careful with using latest Node.js versions in production

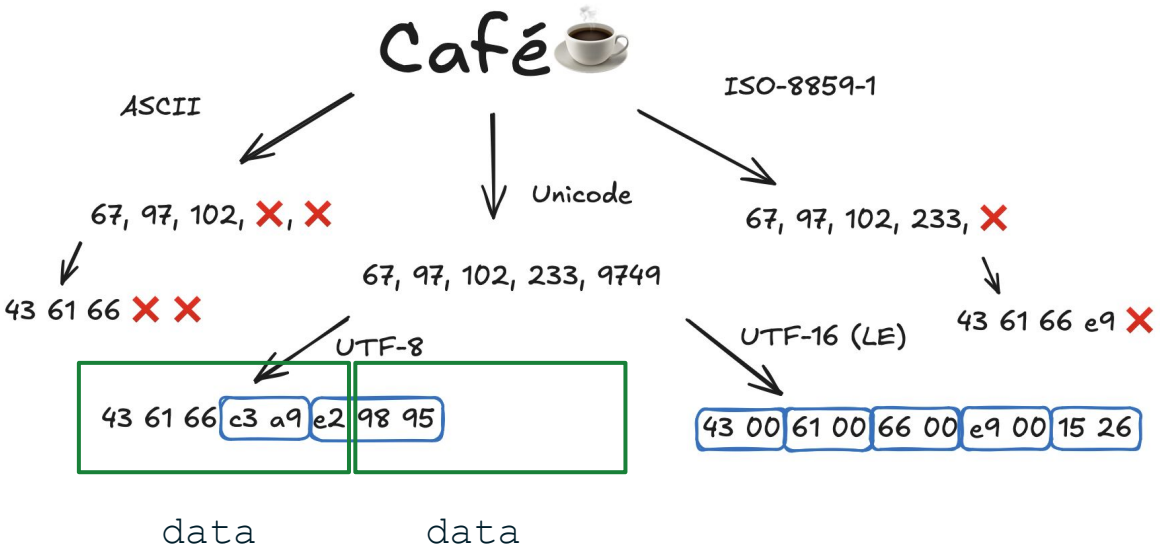
Thank you for your time.

```
import { createReadStream } from "node:fs";

createReadStream(import.meta.filename)
  .map((data) => data.toString().toUpperCase())
  .compose(process.stdout);
```



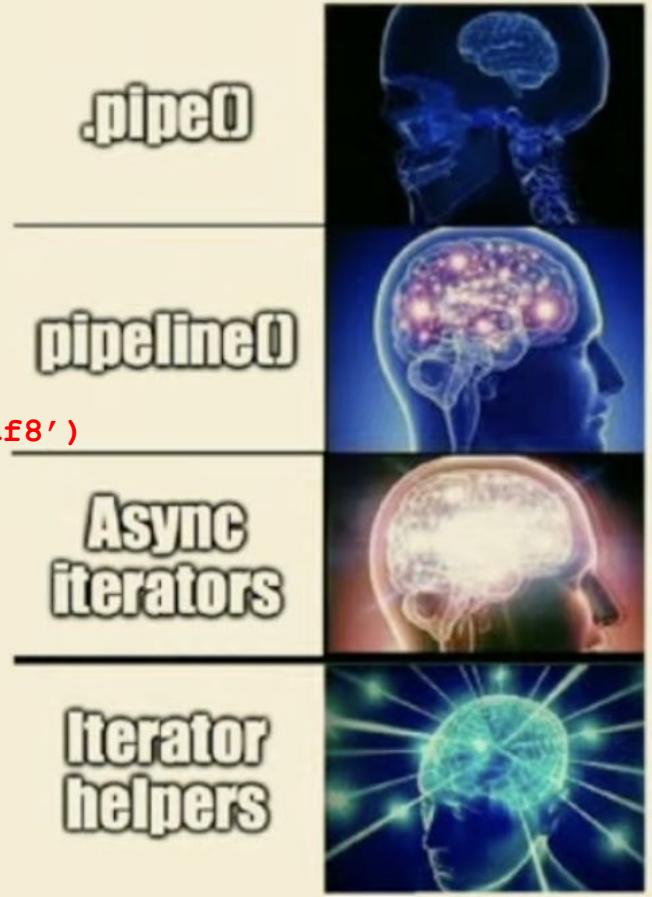
N_

What's a Character Encoding?



```
import { createReadStream } from "node:fs";

createReadStream(import.meta.filename)
  .map((data) => data.toString().toUpperCase())
  .compose(process.stdout);
```

.setEncoding('utf8')



N_