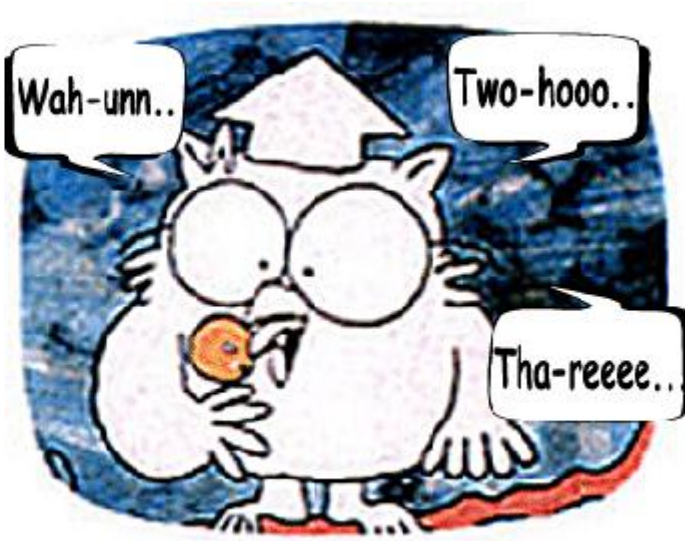
making await wait fewer ticks

Maya Lekova



Basically…

The problem:

Await, as currently specified, enqueues more jobs than necessary.

The solution:

Use "ResolvePromise" instead of always creating wrapper promises

Current Semantics: how many ticks?

```
async function f() {
 await 1;
}
```

- Create a new promise, resolve it with "1" and attach async function "resumers".
- (Tick 1) Execute resumers.

Current Semantics: how many ticks?

```
async function f() {
 let p = Promise.resolve(1); await p;
}
```

- Create a new promise **n**, resolve it with **p**, and attach resumers.
- (Tick 1) Call "then" on **p** with **n**'s resolve and reject functions.
- (Tick 2) Execute **n**'s resolve function.
- (Tick 3) Execute **n**'s handlers (the async function resumers)

Proposed Semantics

```
async function f() {
 let p = Promise.resolve(1); await p;
}
```

- Call PromiseResolve with **p** and attach async function resumers.
- (Tick 1) Execute resumers.

Basically, don't create a new promise if it's a built-in Promise **and** it's not a Promise subclass.

Semantic Implications

For "thenables", promise subclasses, and promises from other realms: no change.

For native promises: fewer interleaving points.

For native promises with a monkey-patched "then": the "then" method will no longer be called.

Performance Implications: V8

Maya Lekova:

Reducing the number of job queue turns also improves performance on multiple highly optimized async/await implementations. In a draft implementation of this proposal in V8 behind a flag:

- *The doxbee async/await performance benchmark improved with 48%*
- *The fibonacci async/await performance benchmark improved with 23%*
- *The Hapi throughput benchmark improved with 50% (when async hooks are enabled) and with 20% (when async hooks are disabled)*

Performance Implications: ChakraCore

ChakraCore currently implements the proposed semantics (almost).

Implementing the existing specification semantics will result in a performance regression. (Exact numbers not available at this time…)

Web Compatibility Implications

Edge already does this, so…?

What programs might be relying on 3 tick behavior (async unit tests)?

What programs might be relying on strange `Promise.prototype.then` monkey-patching?