Design Mistakes in Node

Ryan Dahl
JS Conf Berlin
June 2018

Background

- I created and managed Node through its initial development.
- My goal was heavily focused on programming **event driven HTTP servers**.
- That focus turned out to be crucial for Server-Side JavaScript at the time.
  It wasn't obvious then but server-side JS required an event loop to succeed.

Background

When I left in 2012, I felt Node had (more or less) achieved my goals for a user friendly non-blocking framework:

- Core supported many protocols: HTTP, SSL, ...
- Worked on Windows (using IOCP) Linux (epoll) and Mac (kqueue).
- A *relatively* small core with a *somewhat* stable API.
- A growing ecosystem of external modules via NPM.

But I was quite wrong - there was so much left to do...

Critical work has kept Node growing since.

- npm (AKA "Isaac") decoupled the core Node library and allowed the ecosystem to be distributed.
- N-API is beautifully designed binding API
- Ben Noordhuis and Bert Belder built the libuv.
- Mikeal Rogers organized the governance and community.
- Fedor Indutny has had a massive influence across the code base, particularly in crypto.
- And many others: TJ Fontaine, Rod Vagg, Myles Borins, Nathan Rajlich, Dave Pacheco, Robert Mustacchi, Bryan Cantrill, Igor Zinkovsky, Aria Stewart, Paul Querna, Felix Geisendörfer, Tim Caswell, Guillermo Rauch, Charlie Robbins, Matt Ranney, Rich Trott, Michael Dawson, James Snell

I have only started using Node again in the last 6 months.

These days my goals are different.

Dynamic languages are the right tool for **scientific computing**, where often you do quick one-off calculations.

And JavaScript is a the best dynamic language.

Rather I will complain about all the warts in Node.

Bugs are never so obvious as when you're the one responsible for them.

At times Node is like nails on chalkboard to me.

It could have been so much nicer.

Regret: Not sticking with Promises

- I added promises to Node in June 2009 but foolishly removed them in February 2010.

- Promises are the necessary abstraction for async/await.

- It's possible unified usage of promises in Node would have sped the delivery of the eventual standardization and async/await.

- Today Node's many async APIs are aging badly due to this.

Regret: Security

- V8 by itself is a very good security sandbox.
- Had I put more thought into how that could be maintained for certain applications, Node could have had some nice security guarantees not available in any other language.
- Example: Your linter shouldn't get complete access to your computer and network.

Regret: The Build System (GYP)

- Build systems are very difficult and very important.
- V8 (via Chrome) started using GYP and I switched Node over in tow.
- Later Chrome dropped GYP for GN. Leaving Node the sole GYP user.
- GYP is not an ugly internal interface either - it is exposed to anyone who's trying to bind to V8.
- It's an awful experience for users. It's this non-JSON, Python adaptation of JSON.

Regret: The Build System (GYP)

- The continued usage of GYP is the probably largest failure of Node core.

- Instead of guiding users to write C++ bindings to V8,
  I should have provided a core foreign function interface (FFI).
- Many people, early on, suggested moving to an FFI (namely Cantrill)
  and regrettably I ignored them.

- (And I am extremely displeased that libuv adopted autotools.)

Regret: package.json

- Isaac, in NPM, invented package.json (for the most part).
- But I sanctioned it by allowing Node's require() to to inspect package.json files for "main"
- Ultimately I included NPM in the Node distribution, which much made it the defacto standard.
- It's unfortunate that there is a centralized (privately controlled even) repository for modules.

require("somemodule") is not specific.
There are too many places where it is defined.

your javascript program

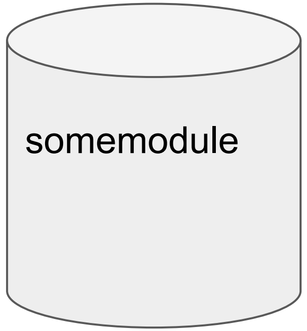
```
require("somemodule")

// Code that uses
// somemodule
```

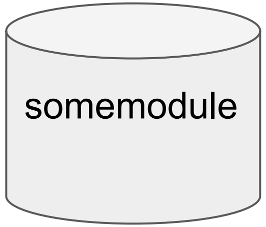
package.json

```
{
  ....
   "dependencies": {
      "somemodule": "^0.0.1"
   }
   ...
}
```

NPM's database



Local node_modules folder



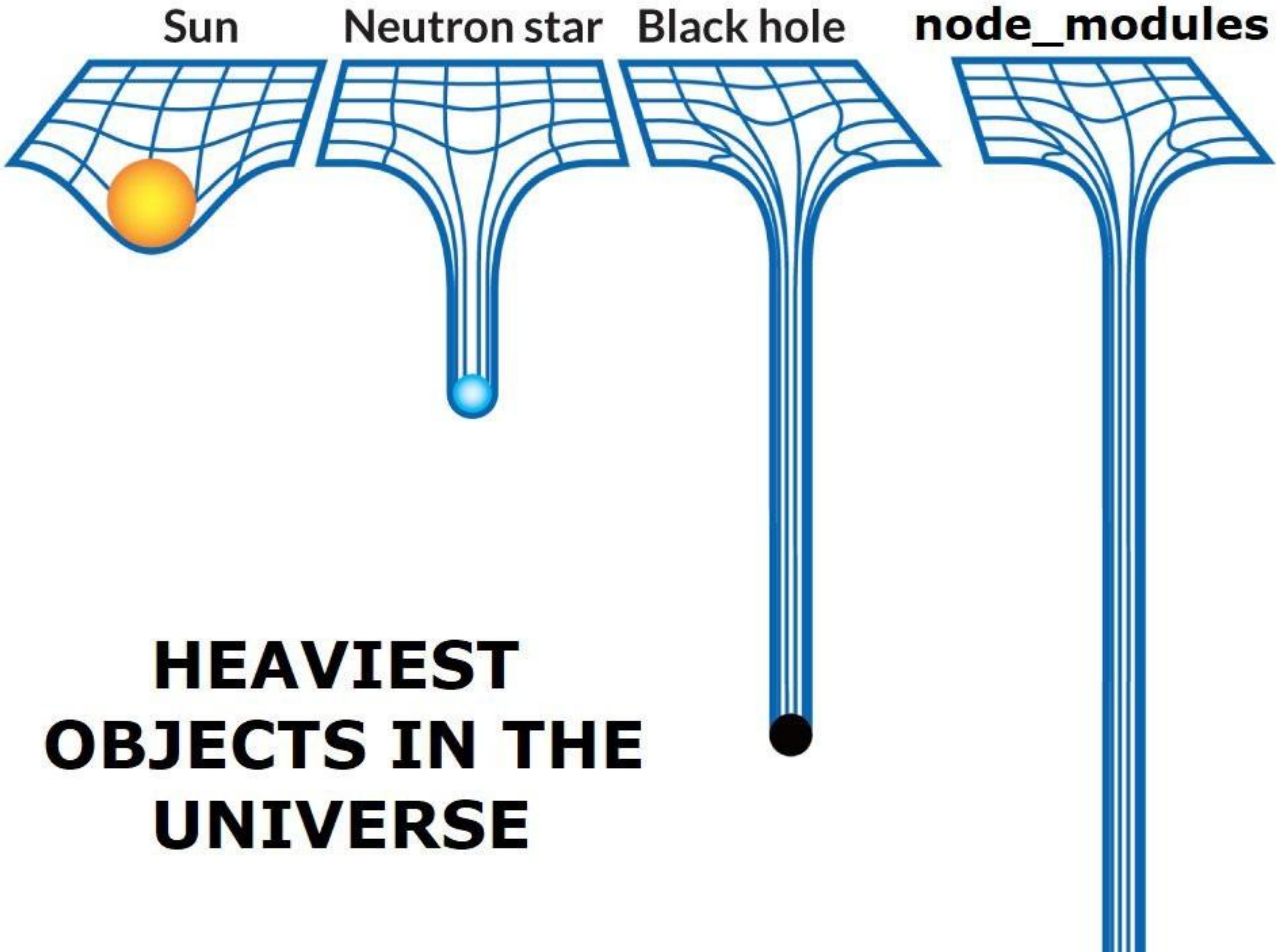
Regret: package.json

Allowing package.json gave rise to the concept of a "module" as a directory of files.

This is not a strictly necessary abstraction - and one that doesn't exist on the web.

package.json now includes all sorts of unnecessary information.
License? Repository? Description?
It's **boilerplate noise**.

If only relative files and URLs were used when importing, the path defines the version. There is no need to list dependencies.



HEAVIEST OBJECTS IN THE UNIVERSE

Regret: node_modules

It massively complicates the module resolution algorithm.

vendored-by-default has good intentions, but in practice just using $NODE_PATH wouldn't have precluded that.

Deviates greatly from browser semantics.

It's my fault and I'm very sorry.
Unfortunately it's impossible to undo now.

Regret: require("module") without the extension ".js"

* Needlessly less explicit.

* Not how browser javascript works. You cannot omit the ".js" in a script tag src attribute.

* The module loader has to query the file system at multiple locations trying to guess what the user intended.

Regret: index.js

I thought it was cute, because there was index.html ...

It needlessly complicated the module loading system.

It became especially unnecessary after require supported package.json.

My problems with Node are almost entirely around how it manages user code.

In contrast to the focus on evented I/O early on, the module system was essentially an afterthought.

With this in mind, I have long thought about how it could be done better....

Disclaimer: I'm presenting a **very nascent** prototype.

Unless you're eager to roll up your sleeves and jump into lldb, **don't bother trying to build it**.

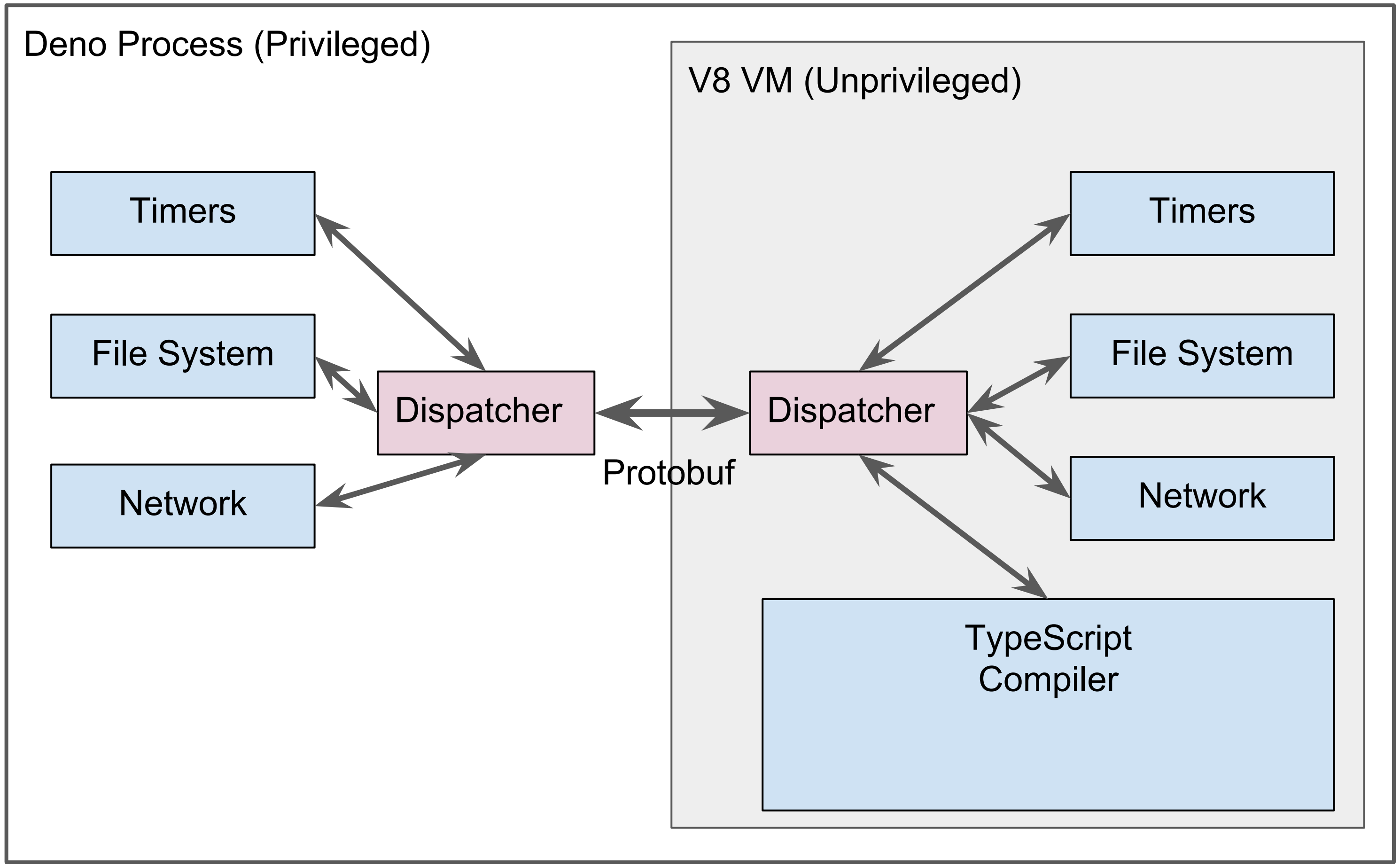
That said...

Deno https://github.com/ry/deno

A secure TypeScript runtime on V8

Deno Goals: Security

- Utilize the fact that JavaScript is a secure sandbox.
  - By default a script should run without any network or file system write access.
  - Users can opt in to access via flags: **--allow-net --allow-write**
  - This allows users to run untrusted utilities (like a linter)
- Do not allow arbitrary native functions to be bound into V8
  - All system calls are done by message passing (protobuf serialization)
  - There are exactly two native functions: send and recv.
  - This both simplifies the design and makes the system easier to audit.


Deno Process (Privileged)
V8 VM (Unprivileged)
Timers
File System
Network
Dispatcher
Protobuf
Dispatcher
Timers
File System
Network
TypeScript Compiler

Deno Goals: Simplify the module system

- No attempt at compatibility with existing Node modules.
- Imports are relative or absolute URLs ONLY. (See Semantic Versioning)

```
import { test } from "https://unpkg.com/deno_testing@0.0.5/testing.ts"
import { log } from "./util.ts"
```

- Imports must provide an extension.
- Remote URLs are fetched and cached indefinitely on the first load.
  Only if the **--reload** flag is provided will the resource be fetched again.
- Vendoring can be done by specifying a non-default cache dir.

Goal: TypeScript compiler built into the executable.

- TS is **absolutely beautiful**.
  - It has finally delivered a practical optionally typed language.
  - Allows code to grow seamlessly from quick hacks to large, well structured machinery.
- Deno hooks into the TS compiler to do module resolution and incremental caching of build artifacts.
- Unmodified TS files should not recompile.
- Normal JS should work too (but that's trivial since TS is a superset of JS)
- It should use V8 Snapshots to for fast startup (not yet in the prototype)

Deno Goal: Ship only a single executable with minimal linkage

```
> ls -lh deno
 -rwxrwxr-x 1 ryan ryan 55M May 28 23:46 deno
 > ldd deno
        linux-vdso.so.1 =>  (0x00007ffc6797a000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f104fa47000)
        libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f104f6c5000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f104f3bc000)
        libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f104f1a6000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f104eddc000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f104fc64000)
```

Deno Goals: Take advantage of 2018

- Bootstrap the runtime by compiling Node modules with Parcel into a bundle. (This is a massive simplification over what Node had to do.)

- Great Infrastructure exists now in native code:
  - EG Don't need to worry about HTTP. Someone else has already made it work. (That was not the case in Node.. web server was 100% hand rolled)
  - Currently the non-JS part of Deno is using **Go** but I'm not totally sold and researching alternatives now that the prototype is done.
    - Rust might be a good choice.
    - C++ might still be a good choice if it allowed others to build their own Denos targeting Go or Rust?

Deno Goals: Misc

- Always die immediately on unhandled promises. (Insanely this is not the case in Node.)
- Support top-level await (Not yet in prototype)
- Browser compatible where functionality overlaps.

Deno https://github.com/ry/deno

It's only a month old. **It is very much not usable**.

But I'm happy with the design so far.

Comments? Questions? Concerns?
ry@tinyclouds.org

These slides: http://tinyclouds.org/jsconf2018.pdf