Reasons *not* to Decouple

John Grubb
Director of Customer Care, Platform.sh

Why do developers and teams decide to build decoupled or microservice architectures?

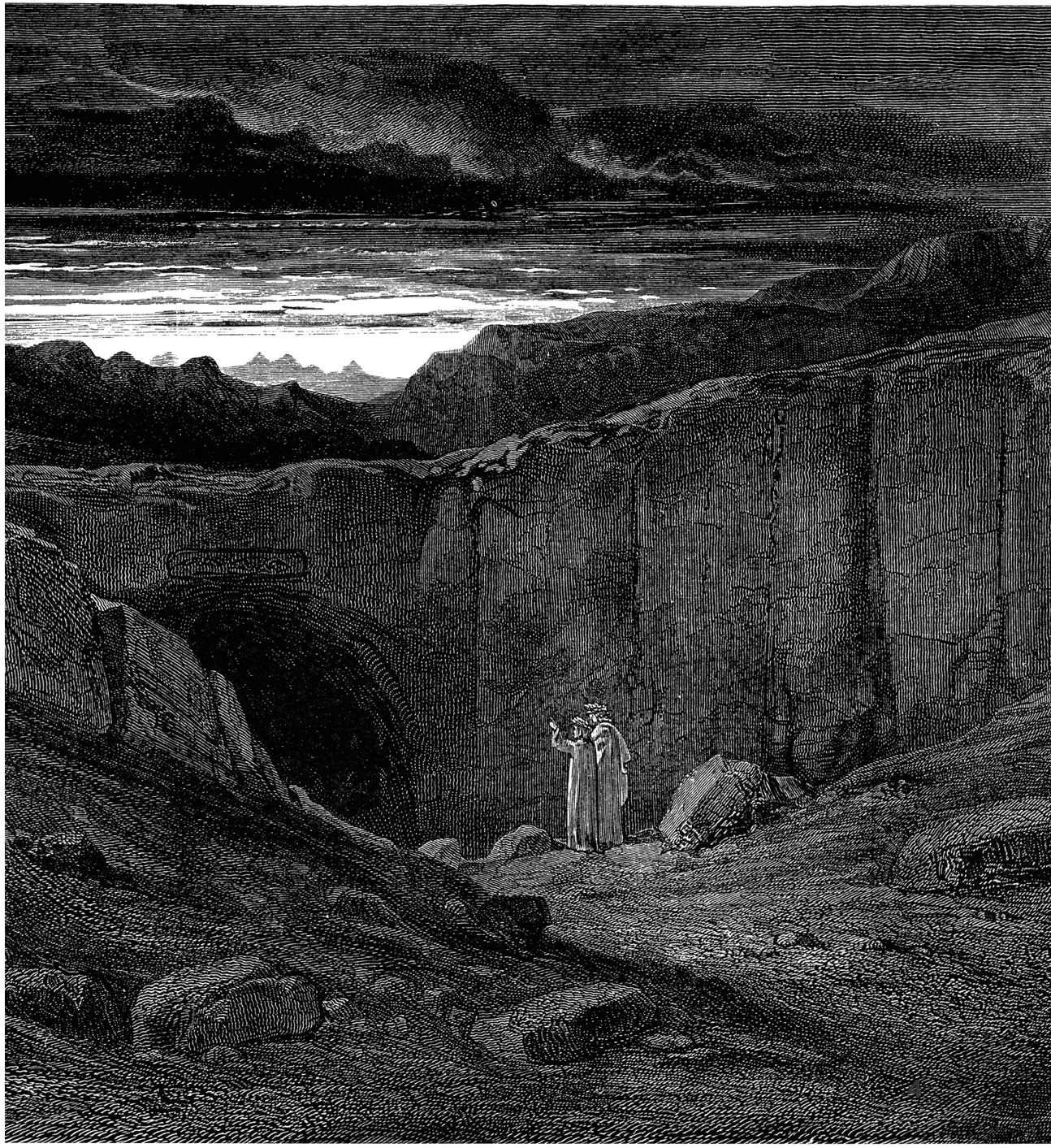
- Serving multiple devices
- Performance
- Developer happiness (because your CMSs theming layer)
- or, more realistically, a combo of the above

Why do developers and teams *really* decide to build decoupled or microservice architectures

- New tech is fun!
- New tech is good for the resume!
- [Pie in the sky ideas about being able to move more quickly here]
- You want to learn new things (arguably the best reason)

Question - how often do you run `composer update` on your Drupal projects?

- Follow up: How often do you spend time unbreaking things after that?
- Follow up: How often do you update your `npm` deps?
- Follow up: How often do you spend time unbreaking things after *that*?



If you were aware of all the nuances you're going to meet while building that decoupled app, you would not build that decoupled app.

Donald Rumsfeld

Exotic technologies are literally Satan

Reasons NOT to decouple

John Grubb
Director of Customer Care, Platform.sh

@johnnygrubb on Twitter
@jgrubb on the Platform.sh Slack

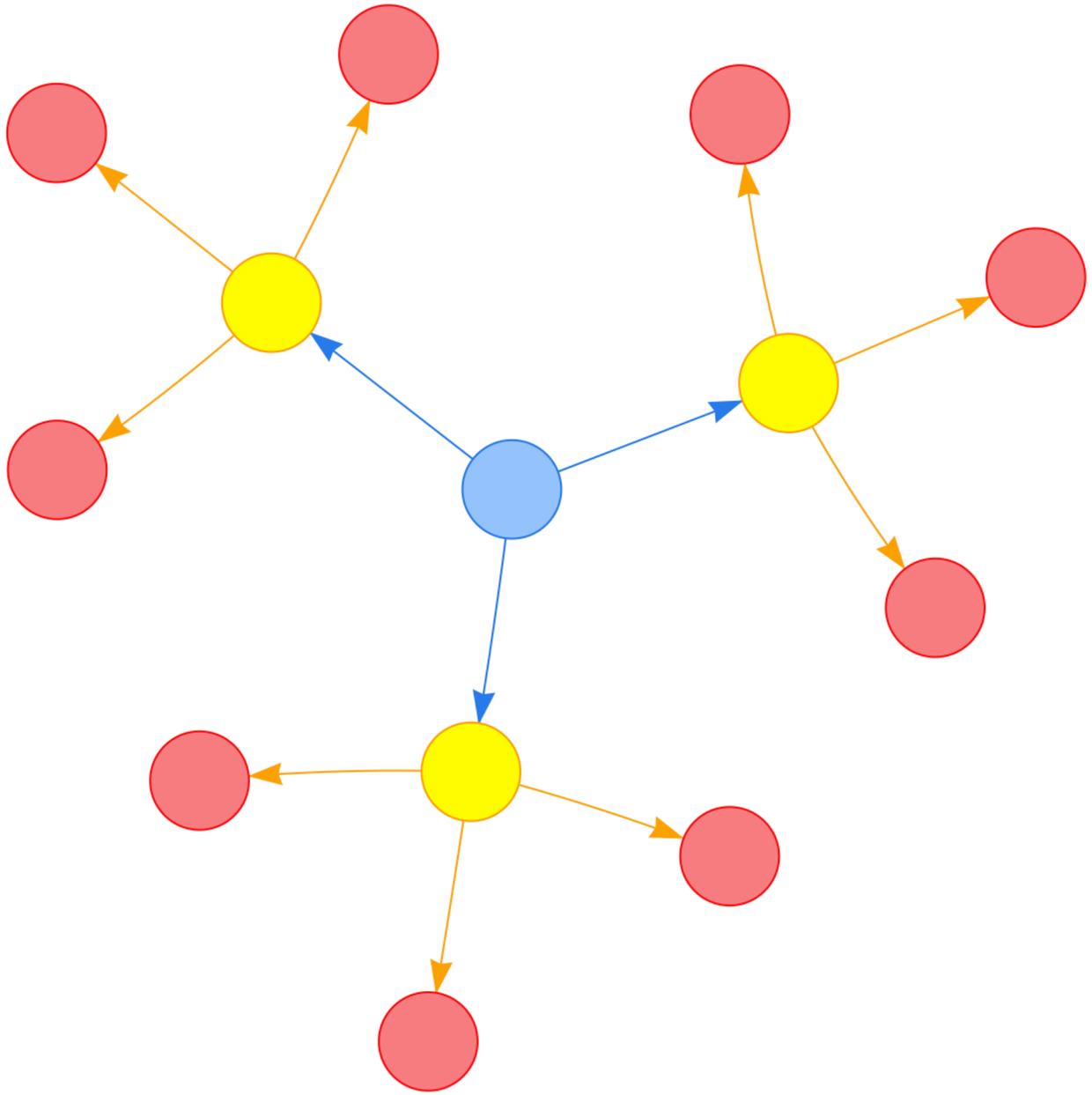
About Myself

- Began with Drupal in 2009
- Built a few decoupled sites using mostly Angular 1.x and D7
- Working from Platform.sh 4 years
- Lead the Customer Care team
  - Customer Success Managers
  - Customer Success Engineers - onboarding and professional services

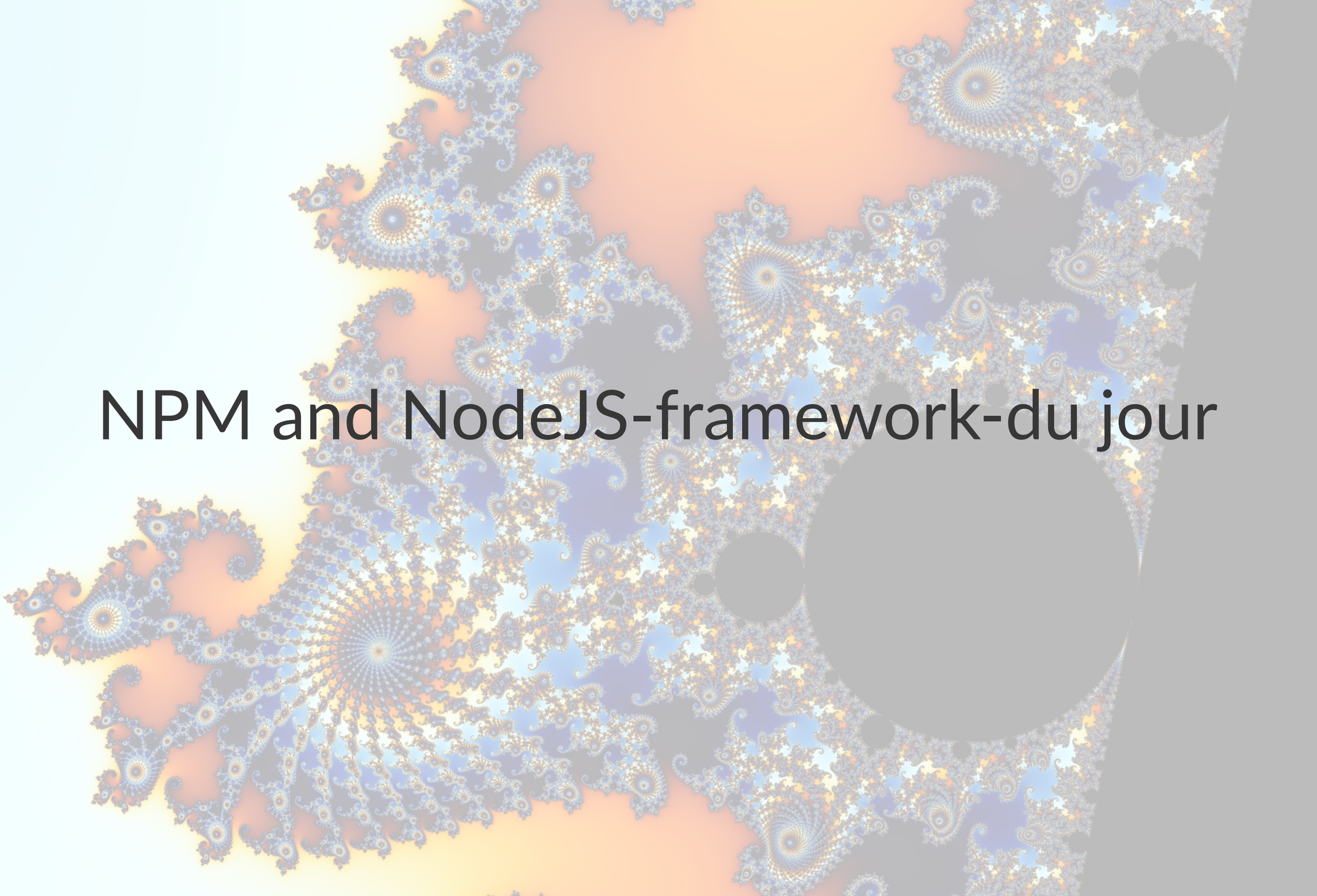
A chat about dependencies

Managing dependencies is hard, often thankless work.

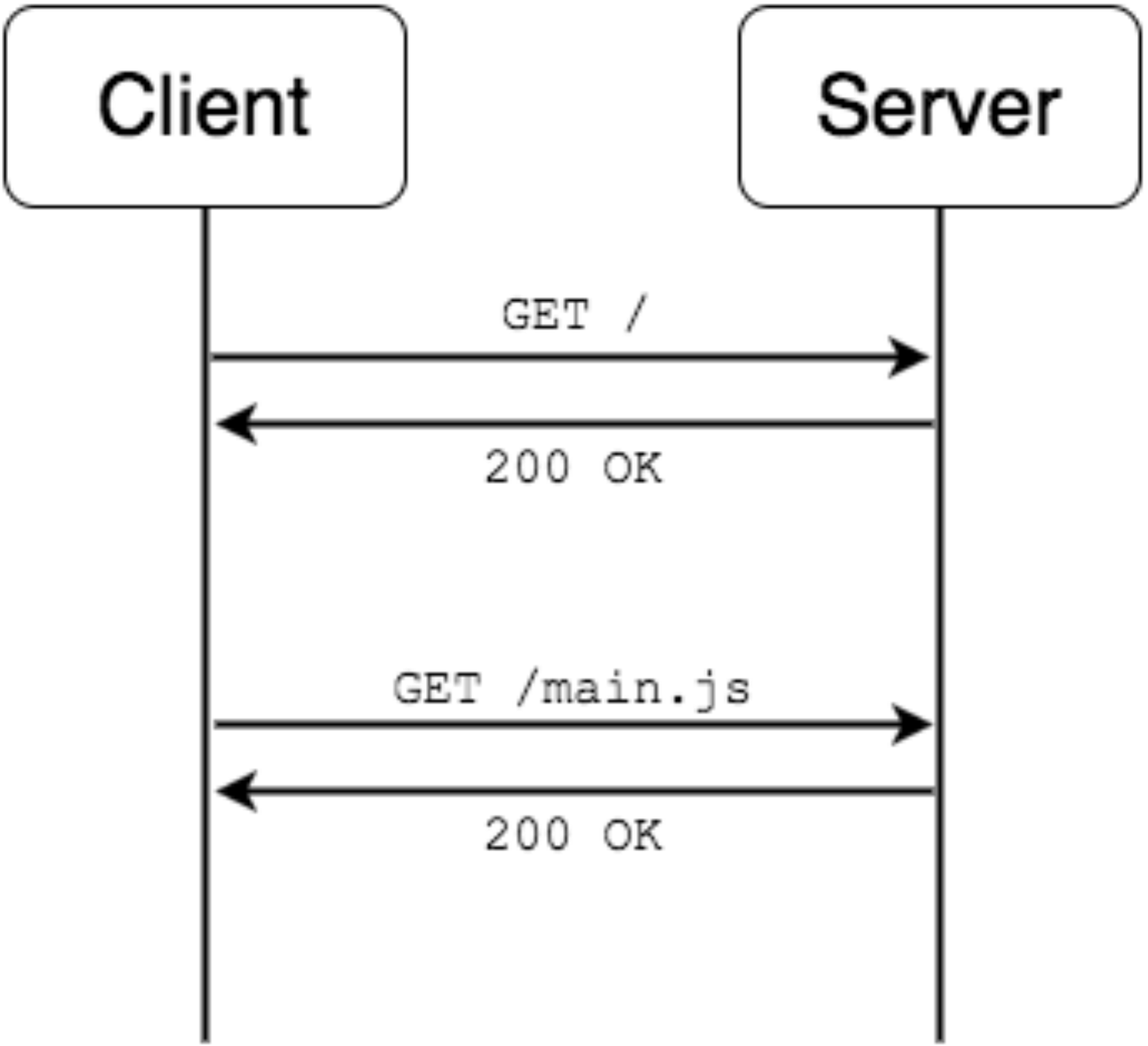
- they go out of date, often without telling you
- updating them is rarely easy in practice, sometimes extremely fraught if a site has been left untended.



Composer and Drupal



NPM and NodeJS-framework-du jour

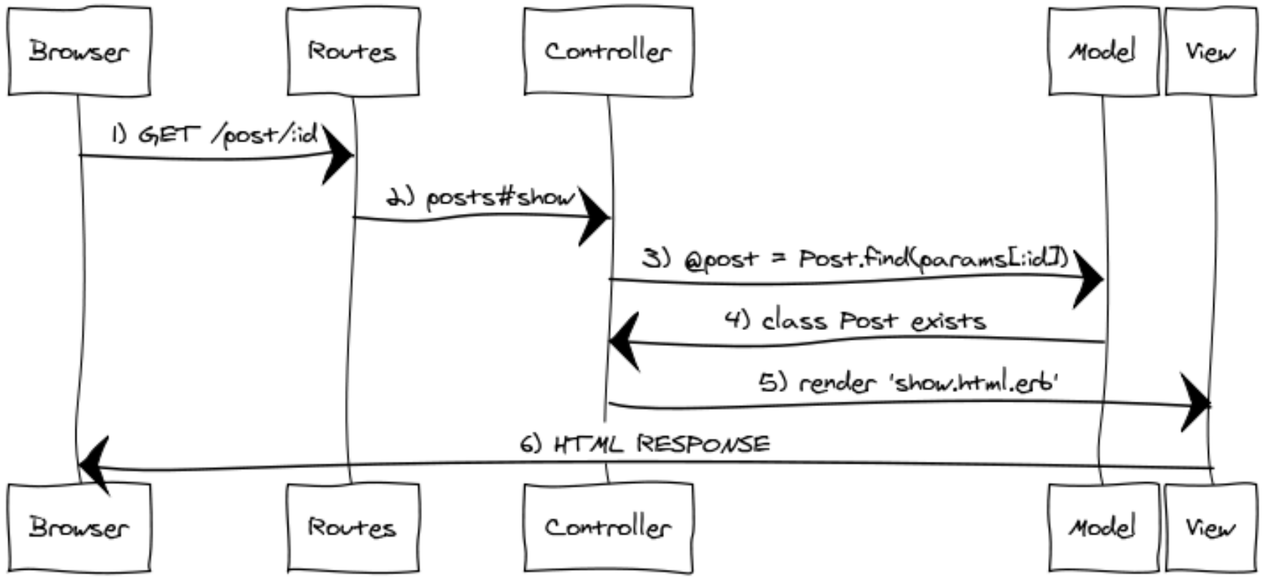


The dependency between your frontend and your backend.

Your dependency on your developers

Drupal devs are expensive. Javascript devs are also expensive. You need both now, unless you're going to have one person do both.

The Distributed Monolith



- Distributed apps have a dependency on the *network*. The network is orders of magnitude slower than your local dev environment.
- Where and how do you cache? Where and how do you clear things out?

Special mention: GraphQL

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

Usecase

```
Frontend app -> GraphQL server -> Drupal
                                  \--> some web service.
```

re: the development process

- How do you keep frontend and backend in sync?
- How do you keep them in sync across development branches?
- What if the frontend depends on a backend feature/change?

re: the deployment process

- If the frontend depends on a backend feature, how do you roll those out without losing your mind?

Are you really sure about this?

Parting advice from your friendly cloud hosting vendor

1. Just think about these points in the planning phase. Think about network latency. Think about the deployment strategy.
2. Deploy to dev environments out in the wild. Deploy early and often. This will give you some idea of how the Thing performs out in the wild, and across the big, bad network.
3. Budget time to figure these things out in your estimates.