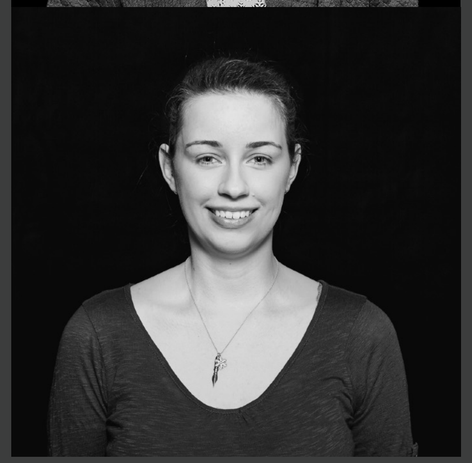
What's your backend written with?

Javascript ?!



CHRISTINA

@merelyChristina

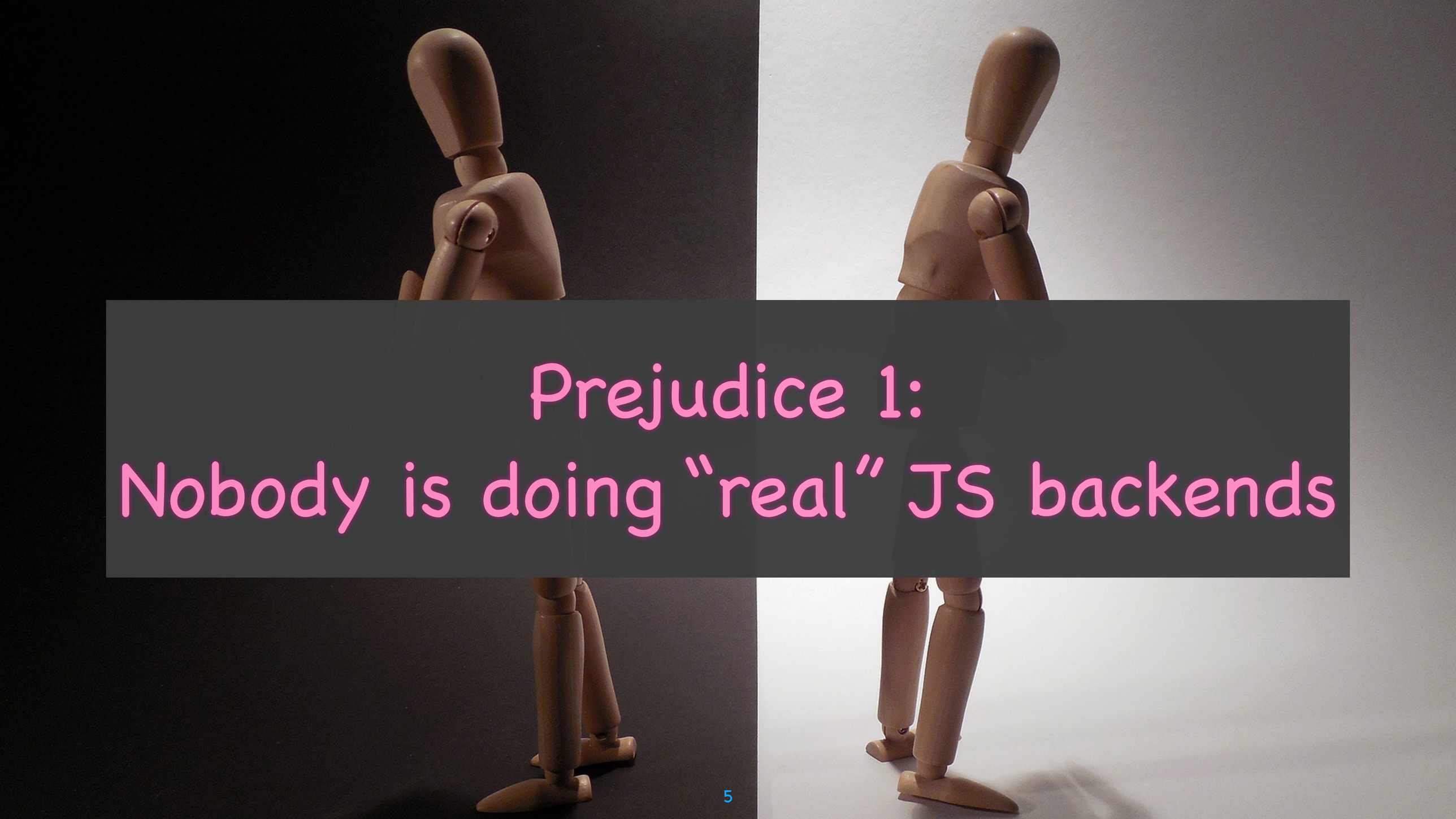
ANNA

@merelyAnna

codecentric

JavaScript backends?

JavaScript backends!

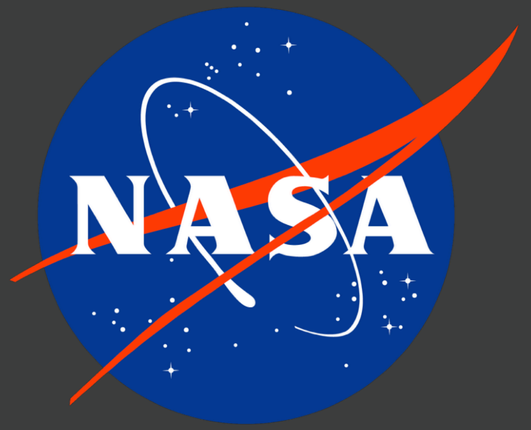


Prejudice 1:
Nobody is doing "real" JS backends

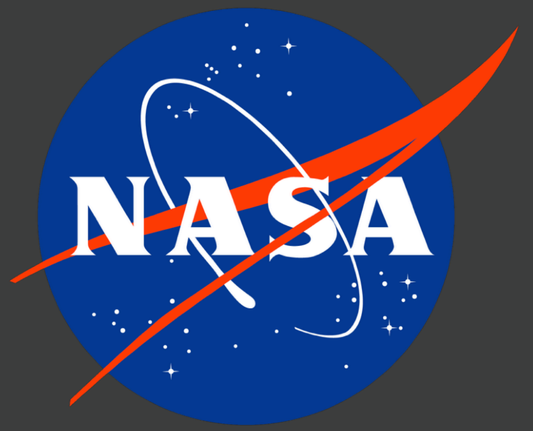
Nobody?

What about you?

Nobody?



~~Nobody?~~

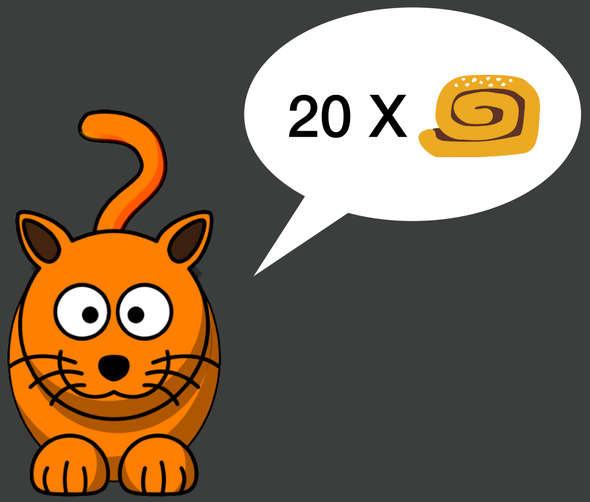


Prejudice 2:
It's only a single thread

Multi. Threading
vs.
Event Loop

Multi Thread vs Event Loop

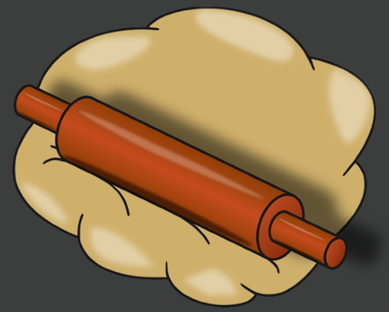
Multithreading



Multithreading

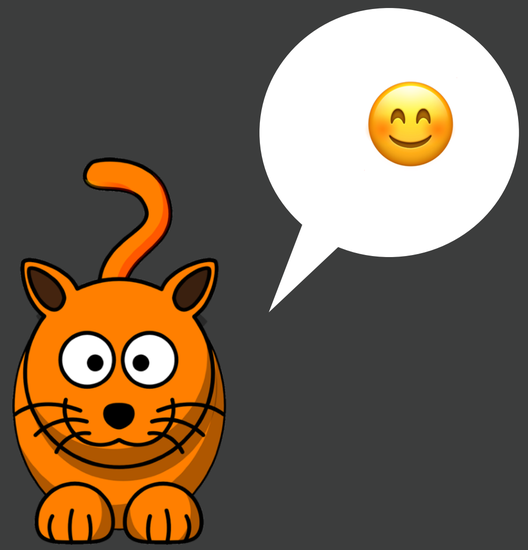
Multithreading

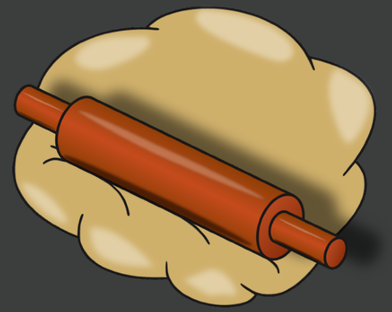
Multithreading

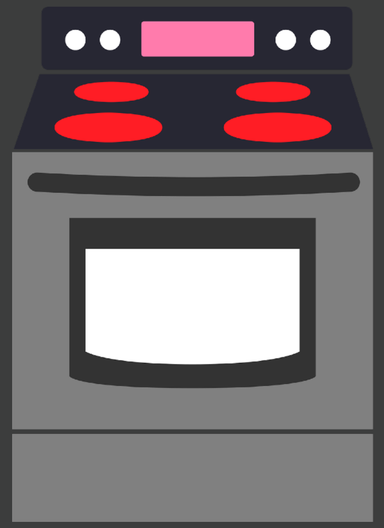


Multithreading

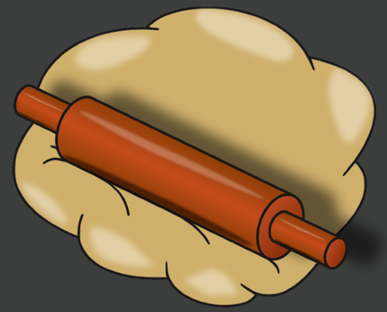
Multithreading

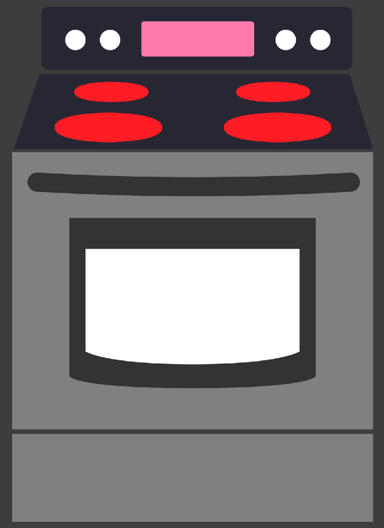






Multithreading





Event Loop

timer

callbacks

poll

check

close callback

Event Loop

timer

callbacks

poll

check

close callback

Event Loop

timer

callbacks

poll 

check

close callback

Event Loop

timer

callbacks

poll 

check

close callback

Event Loop

timer

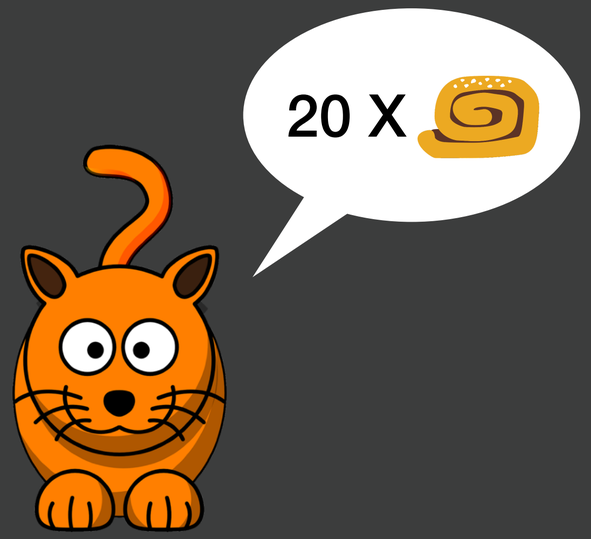
callbacks

poll 

check

close callback

Event Loop



- timer
- callbacks
- poll
- check
- close callback

Event Loop

timer

callbacks

poll

check

close callback

Event Loop

timer

callbacks

poll

check

close callback

Event Loop

timer

callbacks

poll

check

close callback

Event Loop

sleep 10min

timer

callbacks

poll

check

close callback

Event Loop

timer 

callbacks

poll

check

close callback

Event Loop

timer 

callbacks

poll

check

close callback

Event Loop

timer 

callbacks

poll

check

close callback

Event Loop

timer 

callbacks

poll

check

close callback

Event Loop

timer 

callbacks

poll

check

close callback

Event Loop

timer

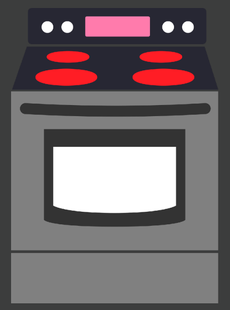
callbacks

poll

check

close callback

Event Loop



timer

callbacks

poll

check

close callback

Event Loop

timer

callbacks 

poll

check

close callback

Event Loop

timer

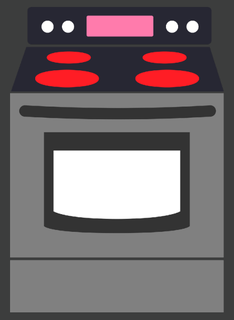
callbacks 

poll

check

close callback

Event Loop



timer

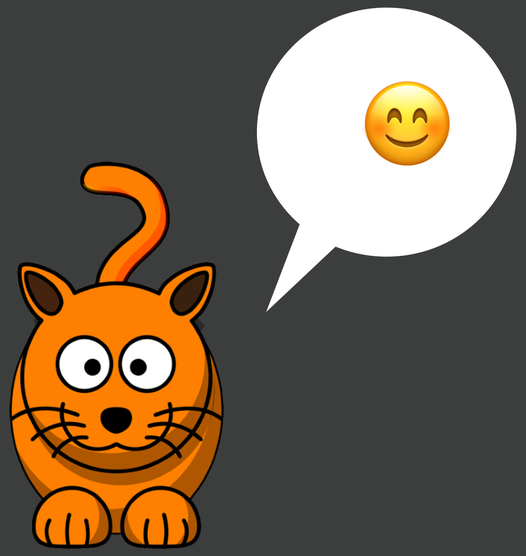
callbacks

poll

check

close callback

Event Loop



timer

callbacks

poll

check

close callback

Event Loop

timer

callbacks

poll

check

close callback 

Event Loop

timer

callbacks

poll

check

close callback 

Event Loop

timer

callbacks

poll

check

close callback 

Event Loop

timer

callbacks

poll

check

close callback 

Event Loop

timer

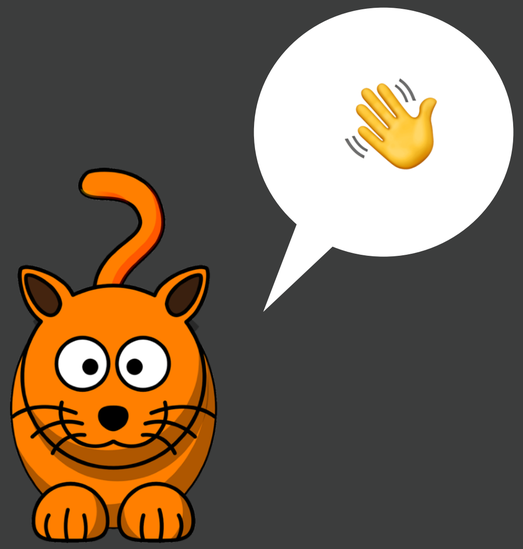
callbacks

poll

check

close callback

Event Loop



timer

callbacks

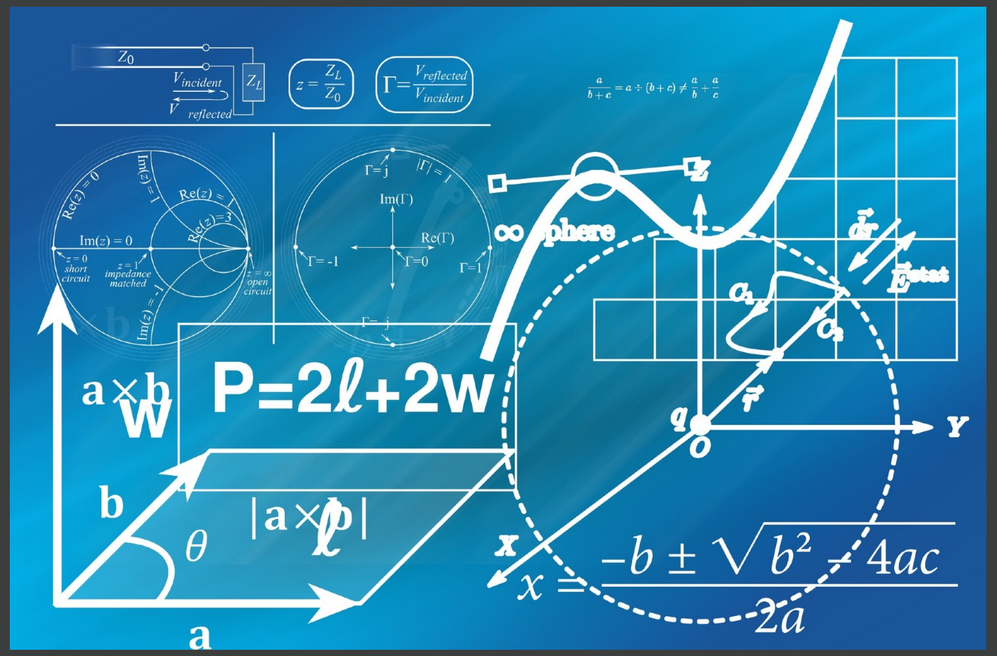
poll

check

close callback

And now …. ?

Disadvantages



heavy calculations

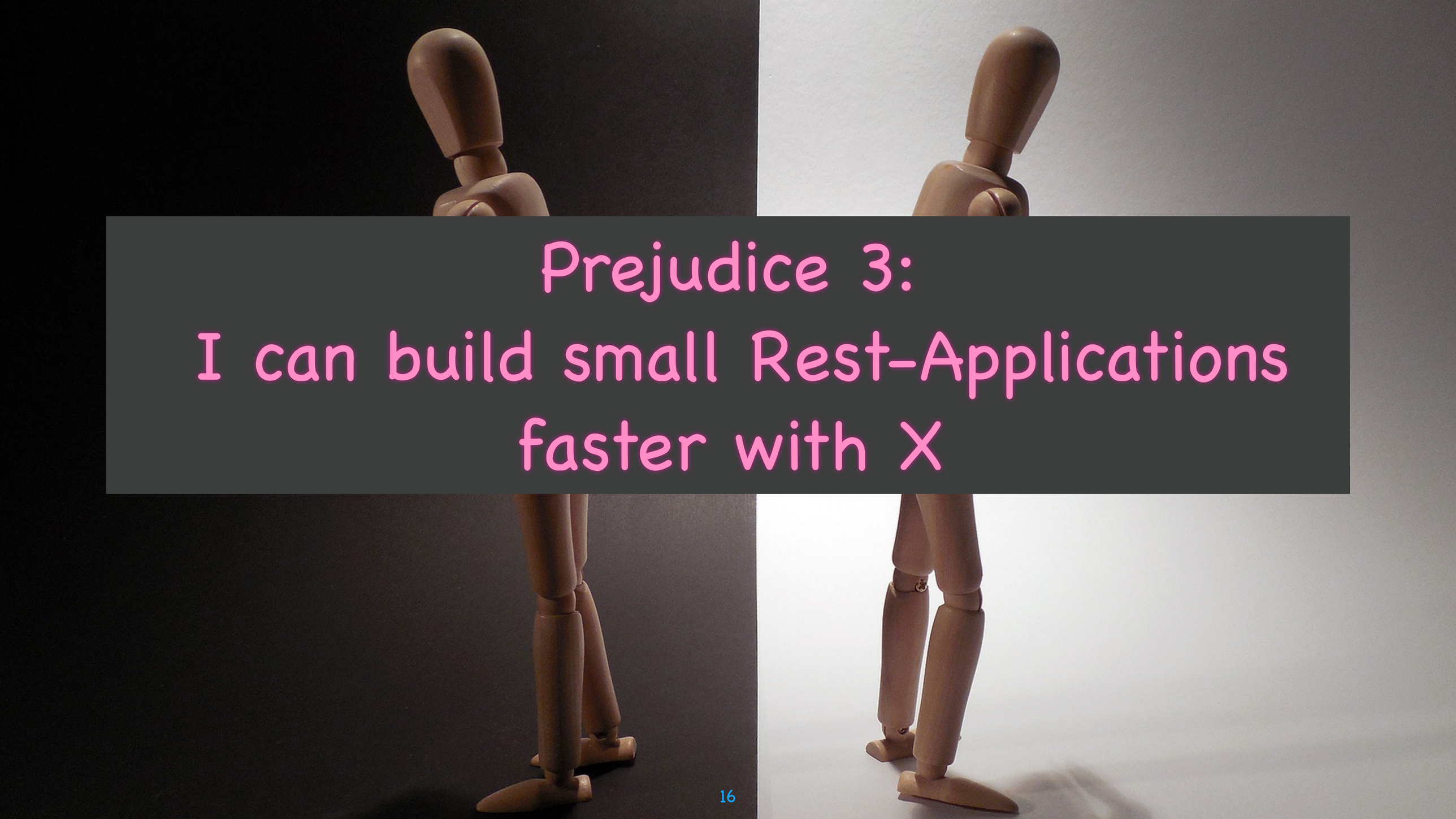
long blocking task

can not split up onto multiple cores

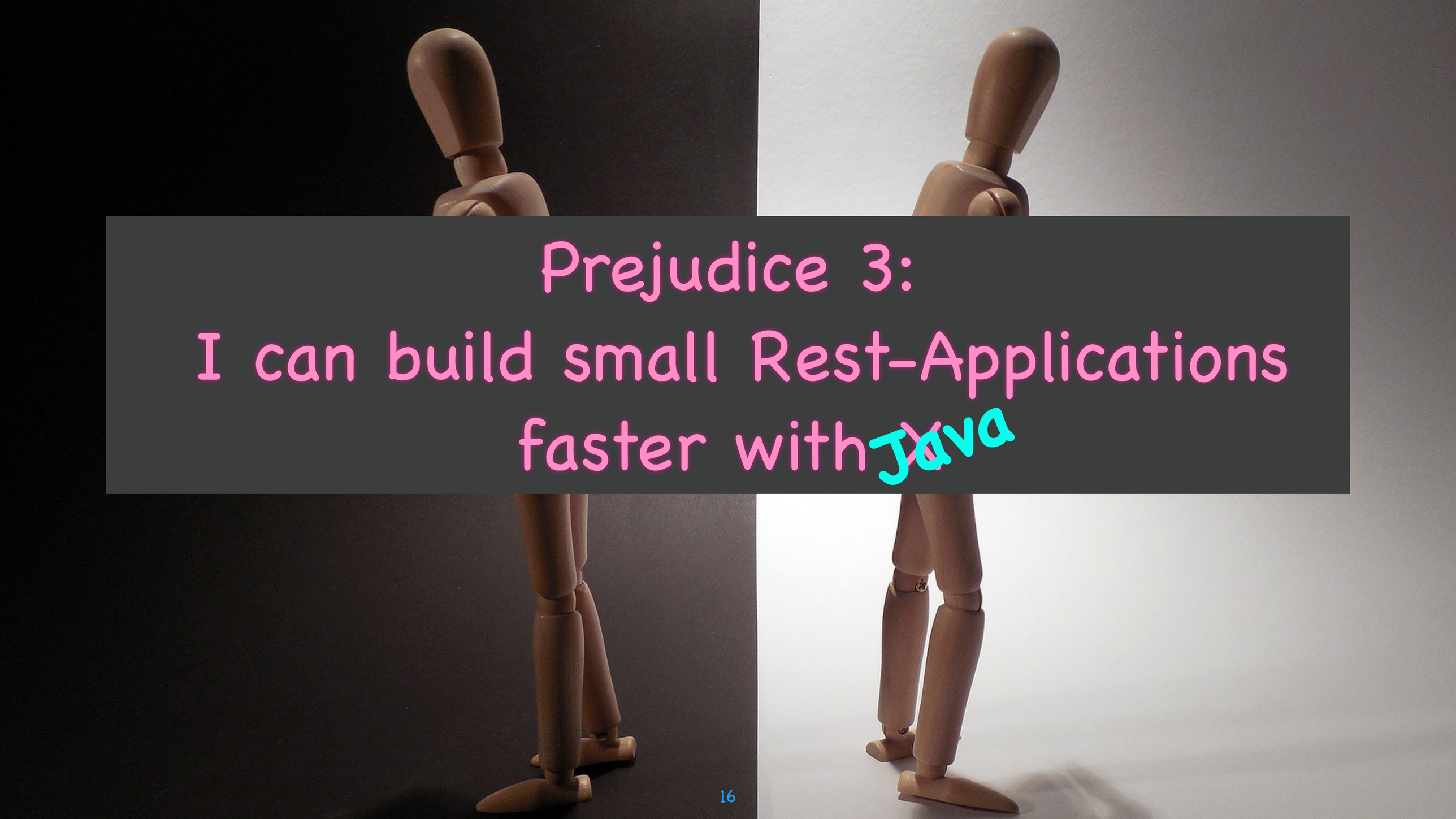
one thread per user

Advantages

- no zombies
- no deadlocks
- Fast switching between Input & Output
- Event based
- Scalable



Prejudice 3:
I can build small Rest-Applications faster with X



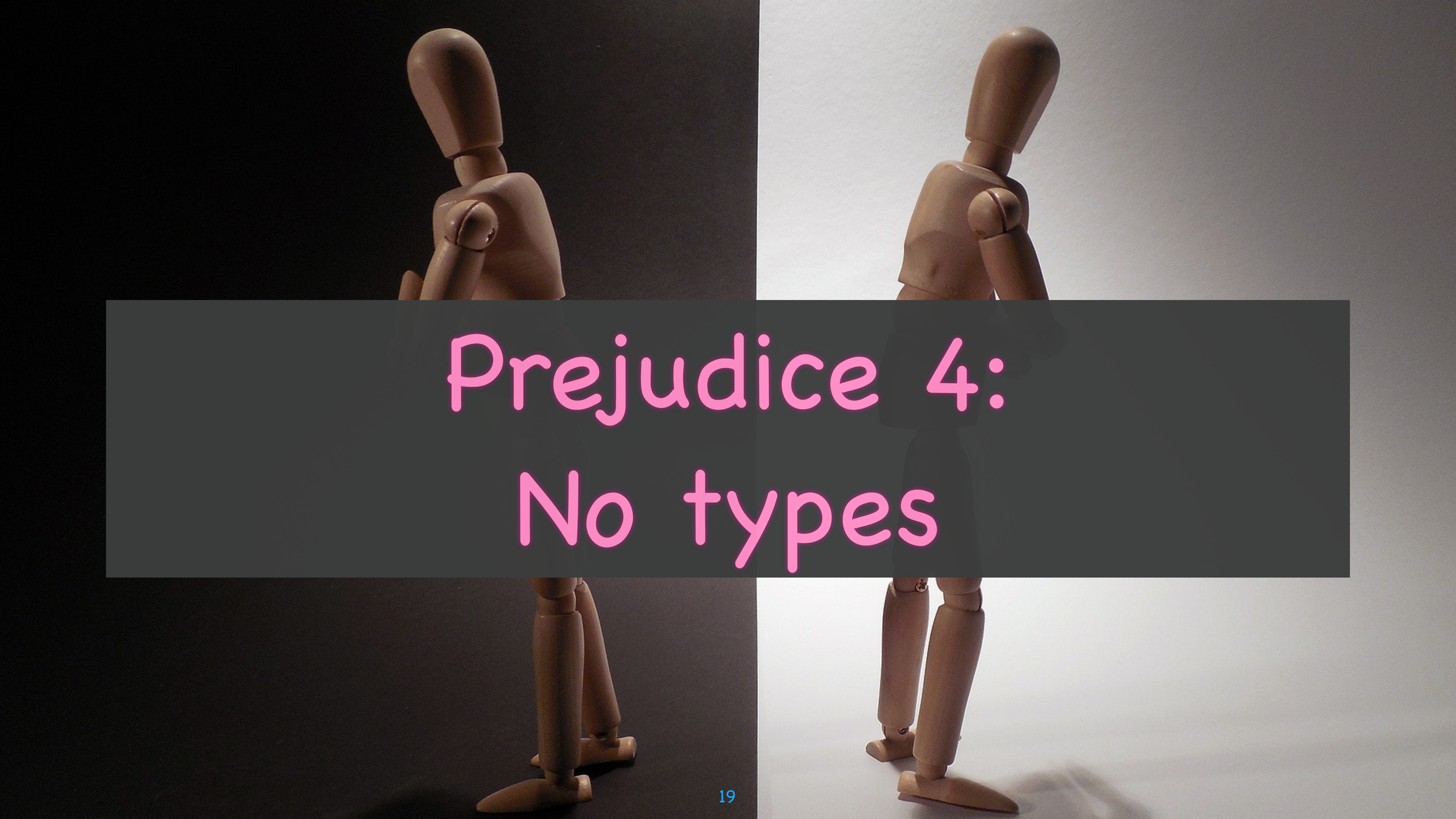
Prejudice 3:
I can build small Rest-Applications faster with Java

```
1.  const express = require("express");
2.  const app = express();
3.  const MongoClient = require("mongodb").MongoClient;
4.  const client = new MongoClient("mongodb://localhost:27017");
5.
6.  client.connect(err => {
7.    const db = client.db("test");
8.
9.    app.get("/coffee", async (req, res) => {
10.     const collection = db.collection("coffee");
11.     const coffee = await collection.find({}).toArray();
12.     res.send(coffee);
13.   });
14. });
15.
16. const port = 3000;
17. app.listen(port, () =>
18.   console.log(`Beverage server is listening on port ${port}!`)
19. );
```

Advantage

small

no overhead



Prejudice 4:
No types

No types? Really?

TS-Backend

```
1.  import * as mongoose from "mongoose";
2.
3.  export const TeaSchema = new mongoose.Schema({
4.    name: String,
5.    type: String,
6.    steepingTime: String
7.  });
8.
9.  export interface Tea extends mongoose.Document {
10.   name: String;
11.   type: String;
12.   steepingTime: String;
13. }
14.
15. export const TeaModel = mongoose.model<Tea>("tea", TeaSchema);
```

TS-Backend

```
1.  const express = require("express");
2.  const app = express();
3.
4.  mongoose.connect("mongodb://localhost:27017/test", { useNewUrlParser: true });
5.  const db = mongoose.connection;
6.
7.  db.once("open", function() {
8.    app.get("/tea", async (request: Request, response: Response) => {
9.      const tea: Tea[] = await TeaModel.find({});
10.     response.send(tea);
11.   });
12. });
13.
14. const port = 3000;
15. app.listen(port, () =>
16.   console.log(`Beverage server is listening on port ${port}!`)
17. );
```

That's all

Advantage

small

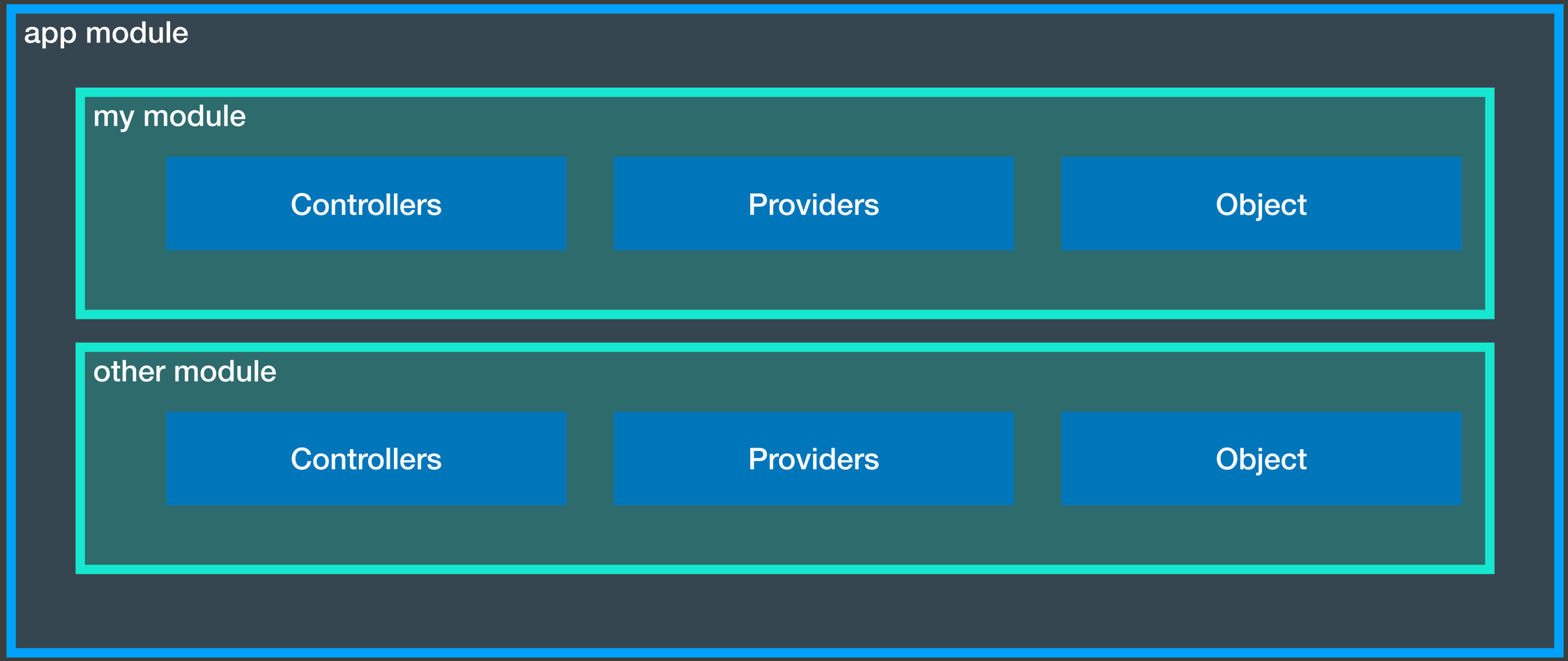
little overhead

types

Prejudice 5:
I have to do everything by myself!

There's a framework for that...

Core fundamentals



Object/Model

```
1.  import * as mongoose from 'mongoose';
2.
3.  export const TeaSchema = new mongoose.Schema({
4.    name: String,
5.    type: String,
6.    steepingTime: String,
7.  });
8.
```

Provider

```
1.  @Injectable()
2.  export class TeaService {
3.    private readonly teas: Tea[] = [];
4.
5.    async getAllTeas(): Promise<Tea[]> {
6.      this.teas.push({
7.                      name: 'earl grey',
8.                      type: 'black',
9.                      steepingTime: '3min'
10.                   });
11.     return Promise.resolve(this.teas);
12.   }
13. }
```

Controller

```
1.  @Controller()
2.  export class TeaController {
3.    constructor(private readonly teaService: TeaService) {}
4.
5.    @Get('/tea')
6.    getAllTeas(): Promise<Tea[]> {
7.      return this.teaService.getAllTeas();
8.    }
9.  }
```

Module

```
1.  import { Module } from '@nestjs/common';
2.  import { TeaController } from './tea.controller';
3.  import { TeaService } from './tea.service';
4.  import { MongooseModule } from '@nestjs/mongoose';
5.  import { TeaSchema } from './tea.schema';
6.
7.  @Module({
8.    imports: [],
9.    controllers: [TeaController],
10.   providers: [TeaService],
11. })
12. export class TeaModule {}
```

APP module

```
1. @Module({
2.   imports: [TeaModule],
3.   controllers: [],
4.   providers: [],
5. })
6. export class AppModule {}
```

Bootstrap code

```
1.  import { NestFactory } from '@nestjs/core';
2.  import { AppModule } from './app.module';
3.
4.  async function bootstrap() {
5.    const app = await NestFactory.create(AppModule);
6.    await app.listen(3000);
7.  }
8.  bootstrap();
```

One disadvantage

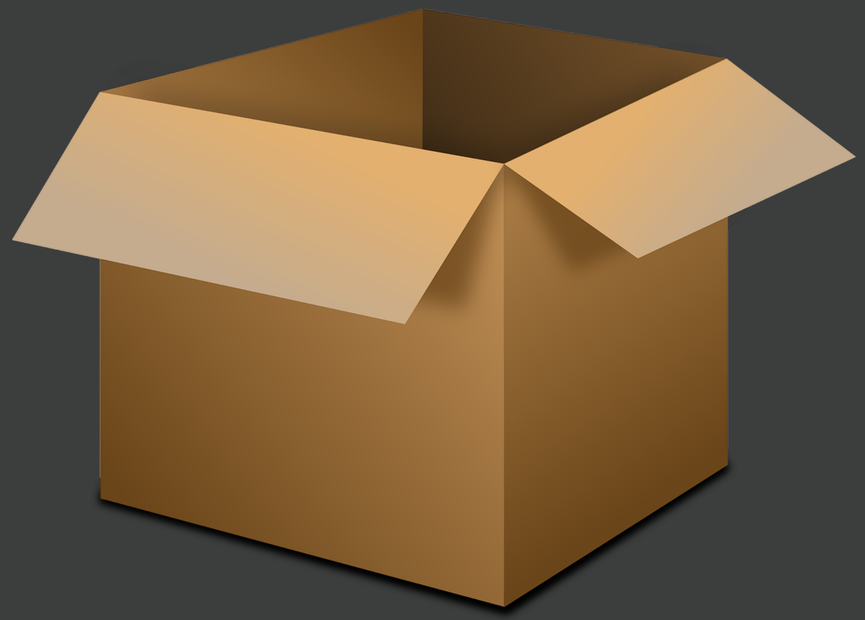
more overhead

Advantage

GraphQL

Mongo

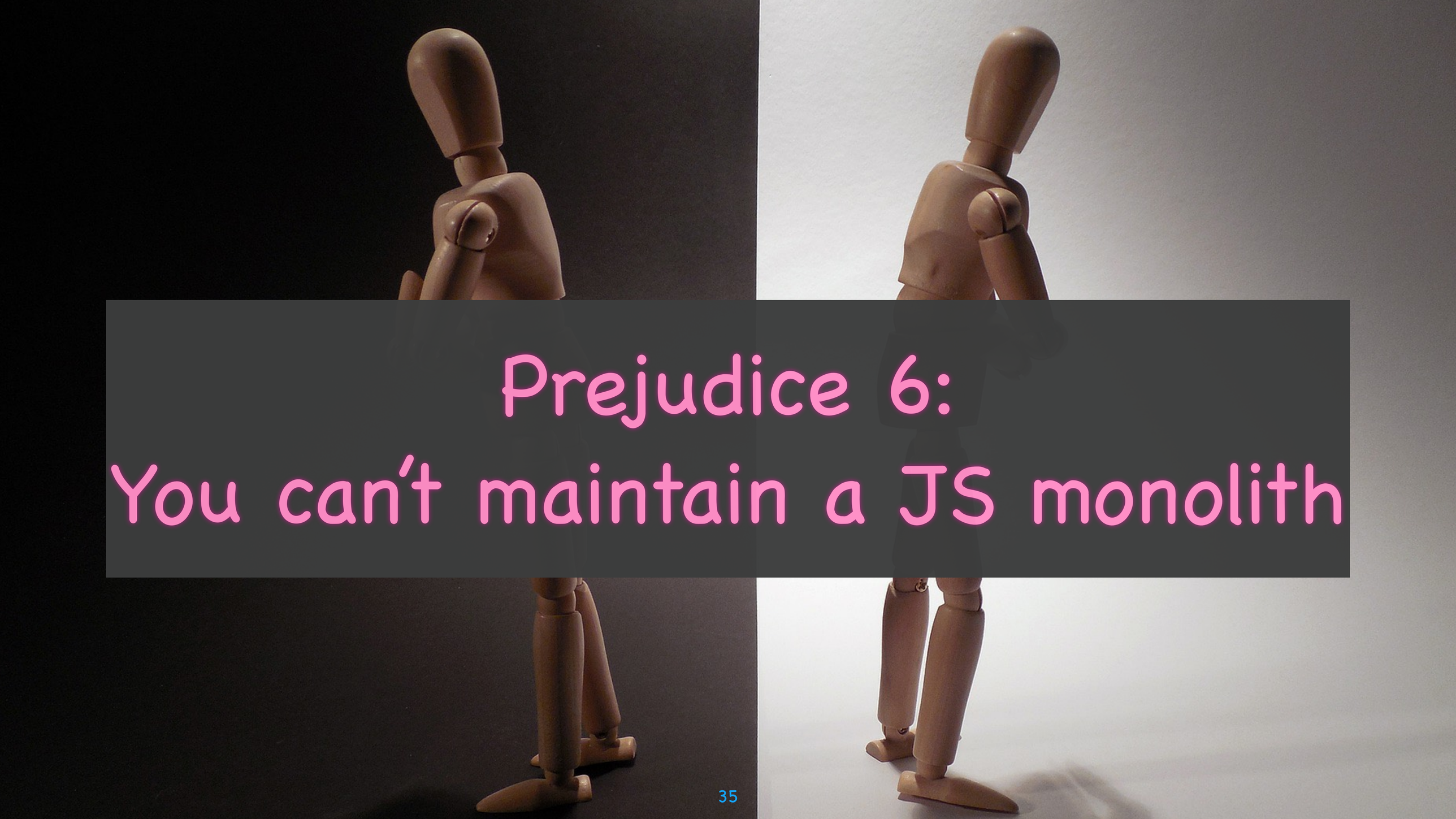
Sockets



Security
(Helmet)

Caching

Authentication



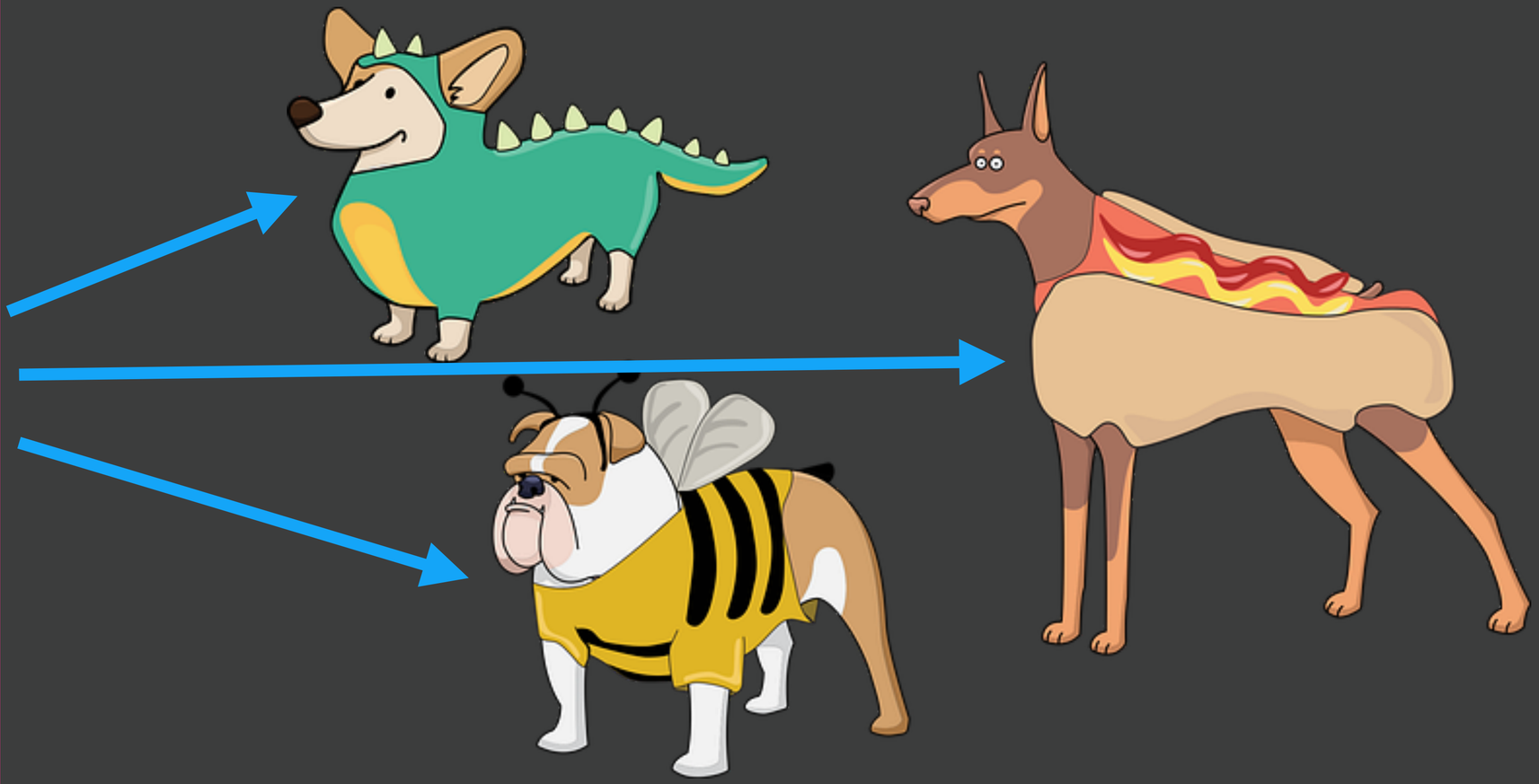
Prejudice 6:
You can't maintain a JS monolith

So what?!

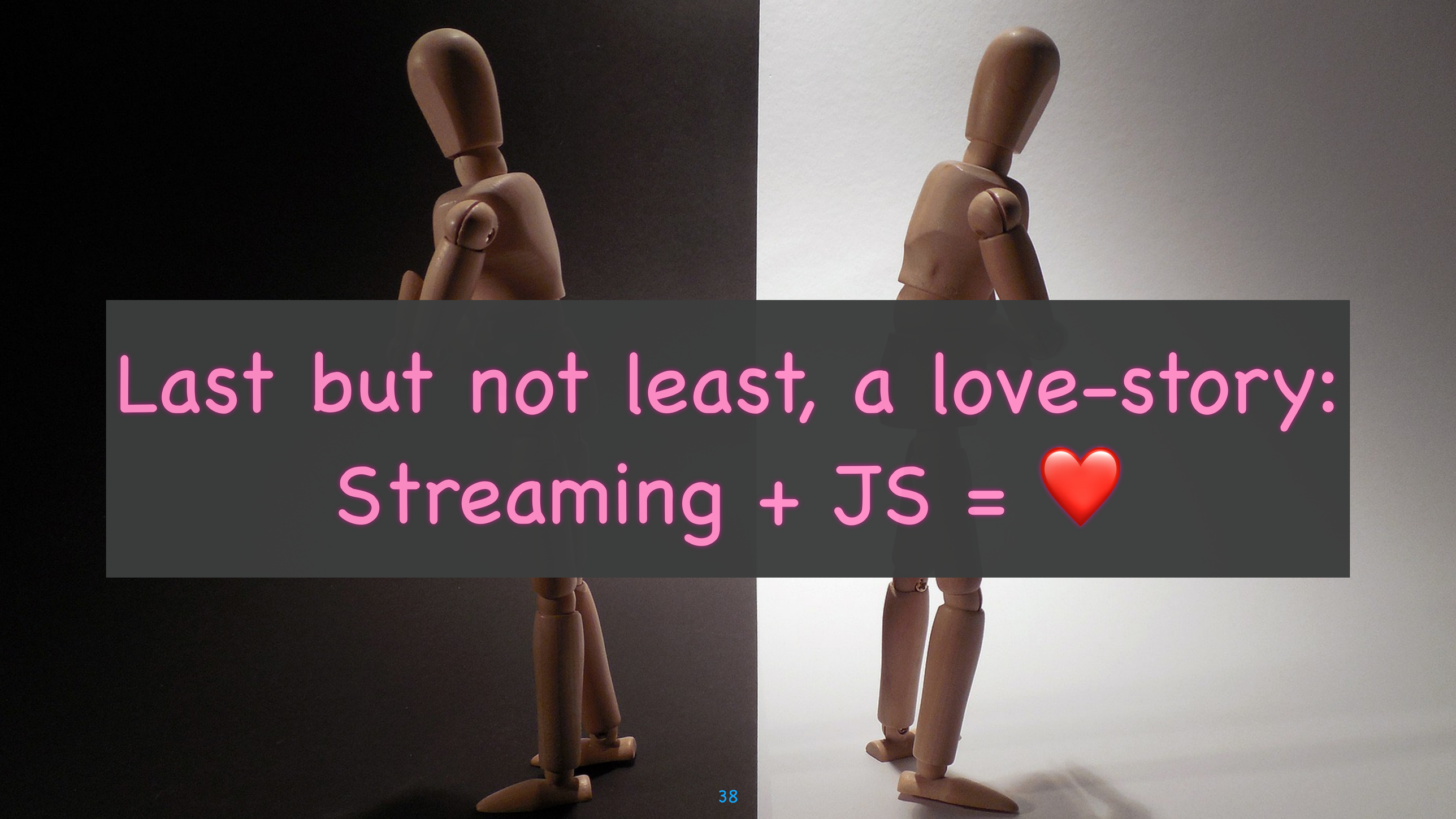
This is a language independent problem.

Bye bye, monolith!

Big, mean monolith



Friendly, little services


Last but not least, a love-story:
Streaming + JS = ❤️

```
1.  const express = require("express");
2.  const http = require("http");
3.  const socketServer = require("socket.io");
4.
5.  const app = express();
6.  const server = http.createServer(app);
7.  const io = socketServer(server);
8.
9.  io.on("connection", function(socket) {
10.   console.log("Connected to Socket!!" + socket.id);
11.
12.   socket.on("GET_WATER", payload => {
13.     const water = { id: "1", type: "sparkling", taste: "lemon" };
14.     io.emit("SERVE_WATER", water);
15.   });
16. });
17.
18. server.listen(3000, () => {
19.   console.log(`Beverage server is listening on port ${port}!`);
20. });
21.
```

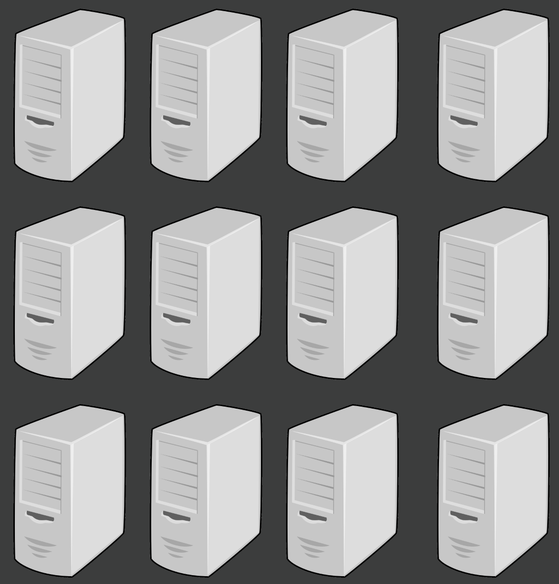
socket.io

```
1.    WebSocketGateway();
2.    export class SearchGateway {
3.      @SubscribeMessage('GET_WATER')
4.      async onEvent(client, data: any): Promise<any> {
5.        return {
6.          event: 'SERVE_WATER',
7.          data: { id: '1', type: 'sparkling', taste: 'lemon' },
8.        };
9.      }
10.   }
11.
```

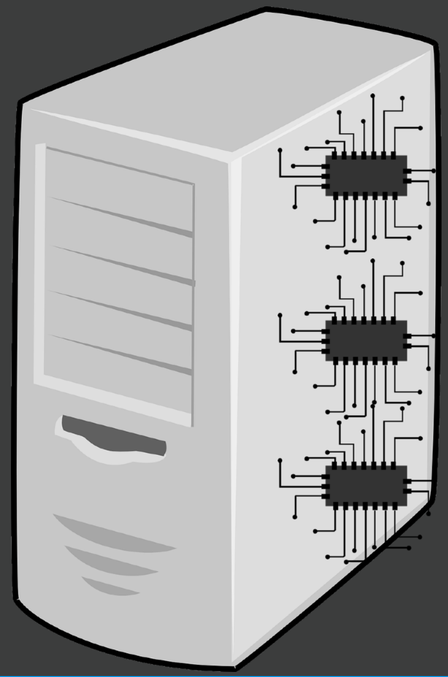

Things to keep in mind

Scaling

horizontal



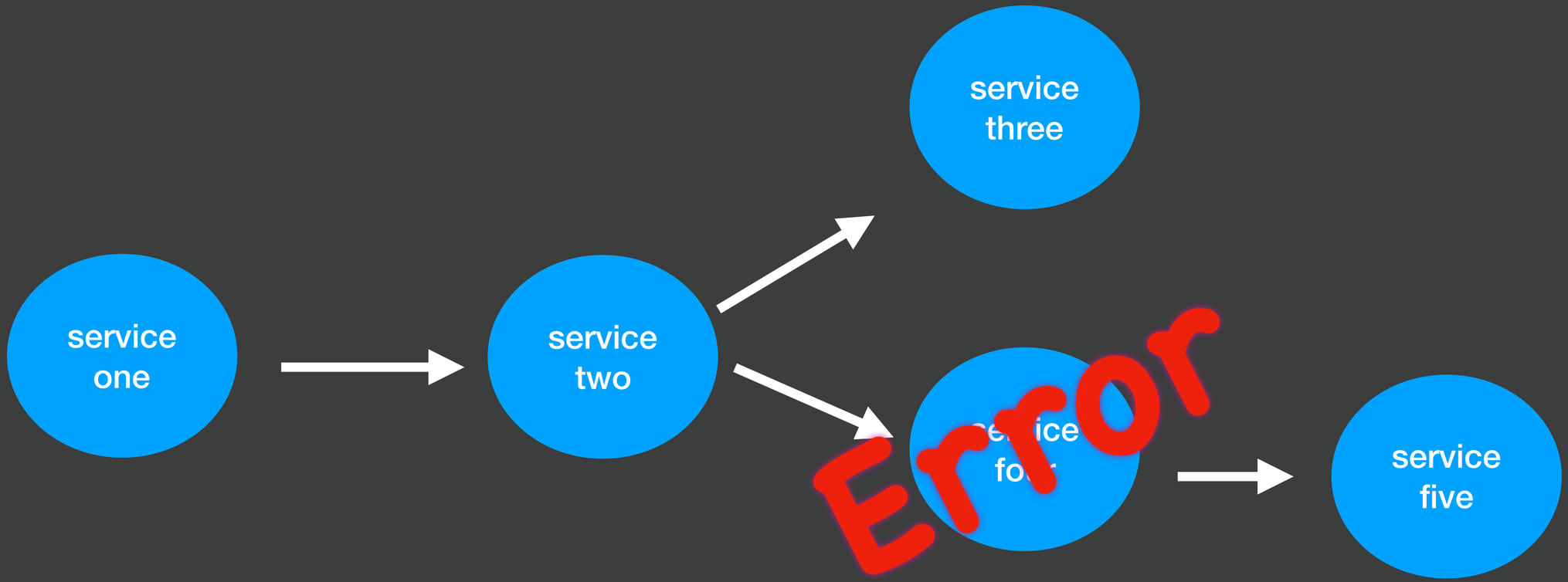
vertical



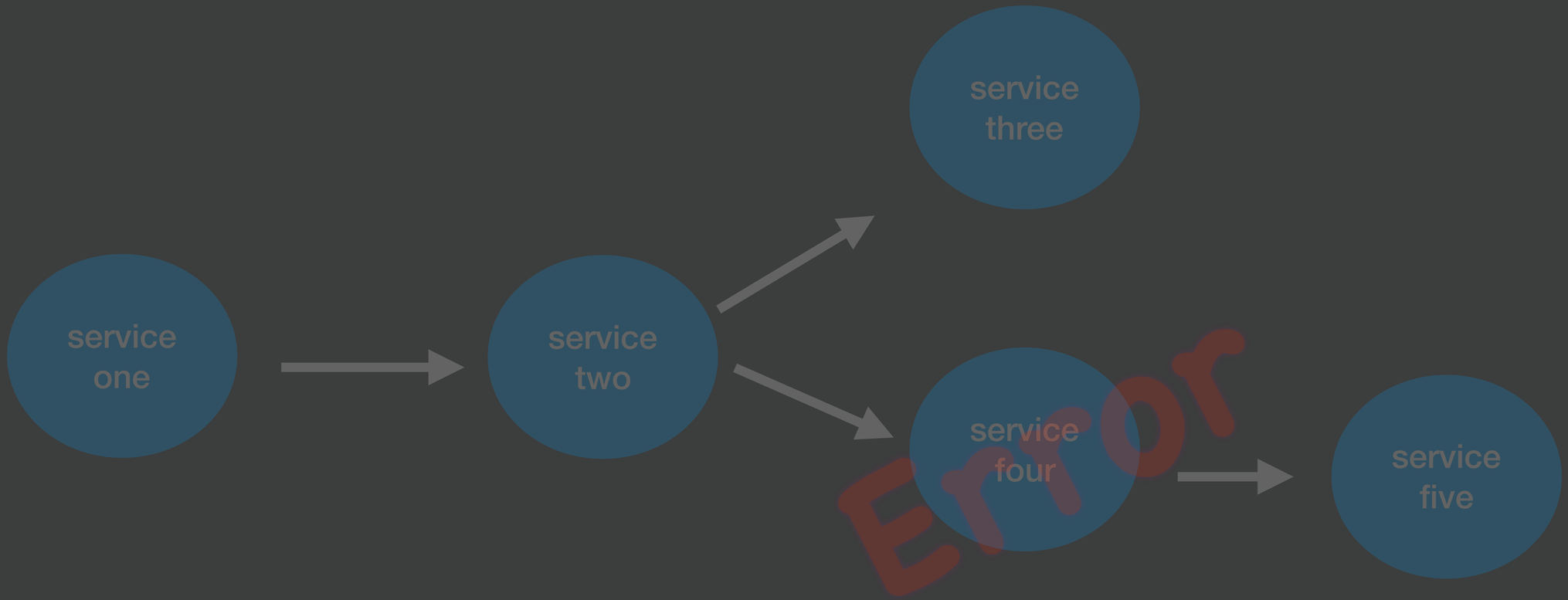
👍 one node instance per core

👎 adding more cpus for one node instance

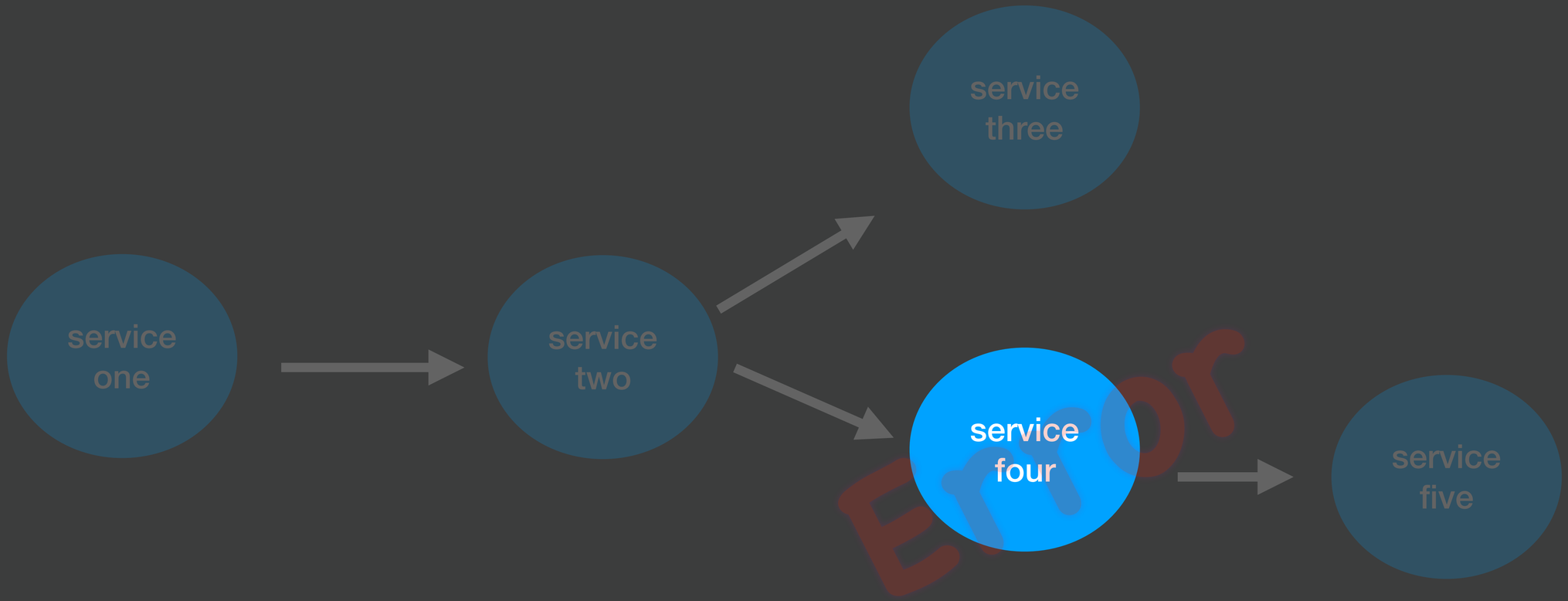
Logging & Tracing



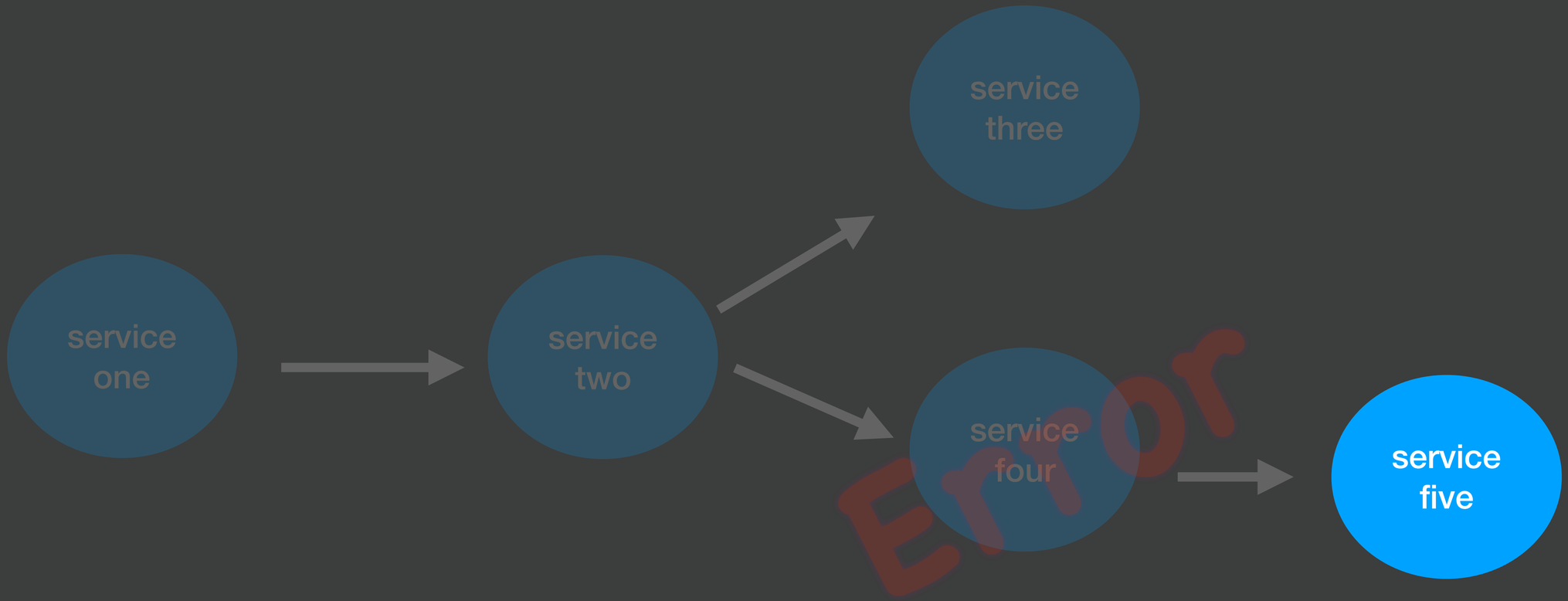
Where's the error?



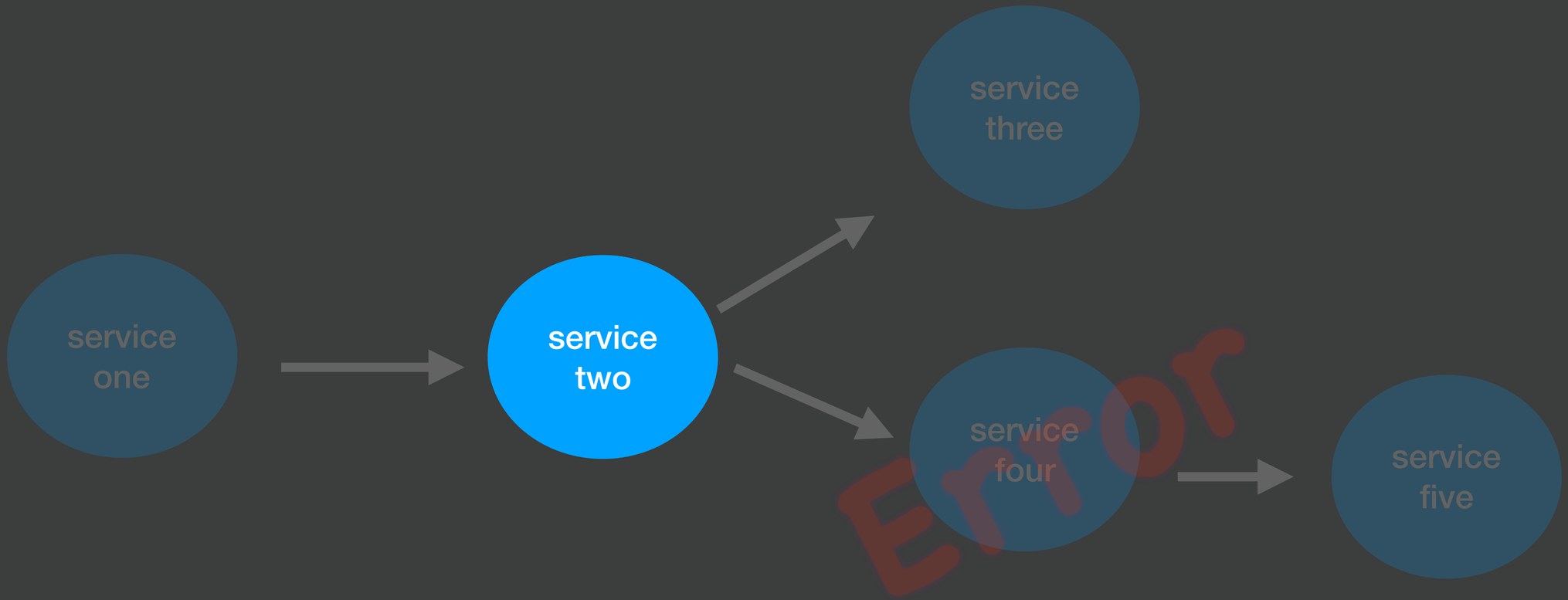
Where's the error?



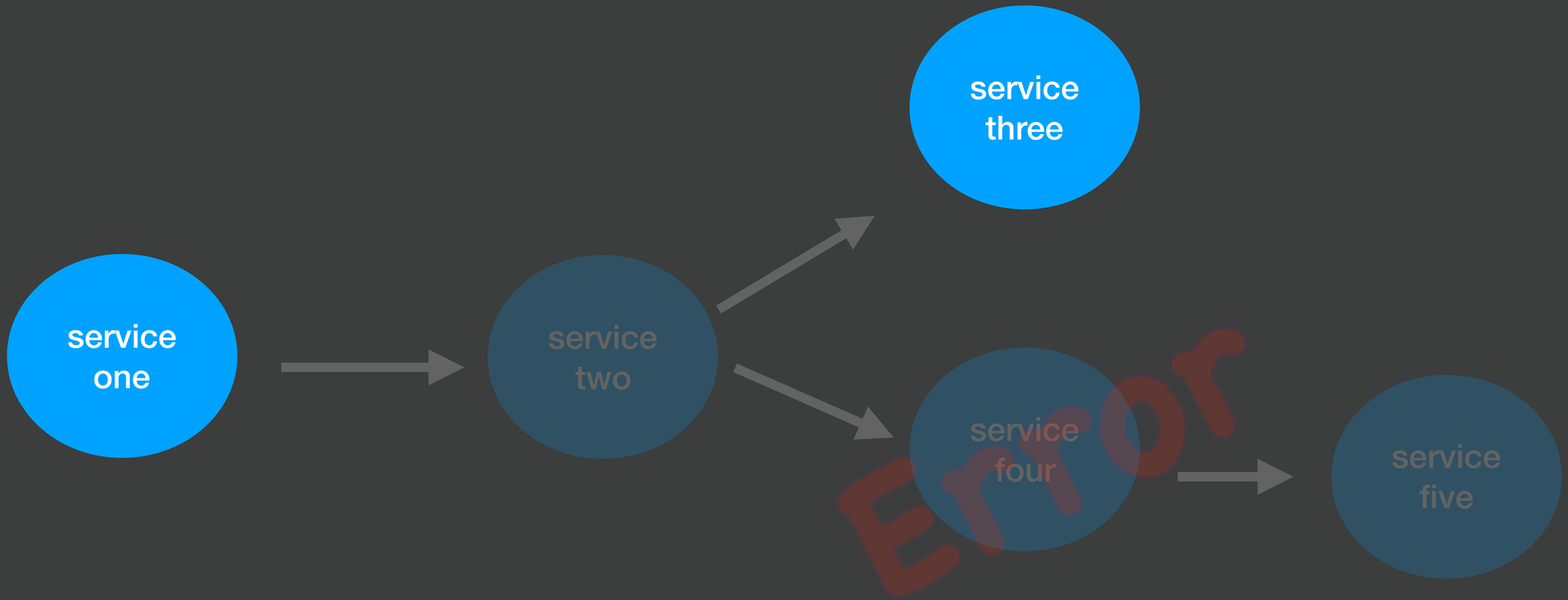
Where's the error?



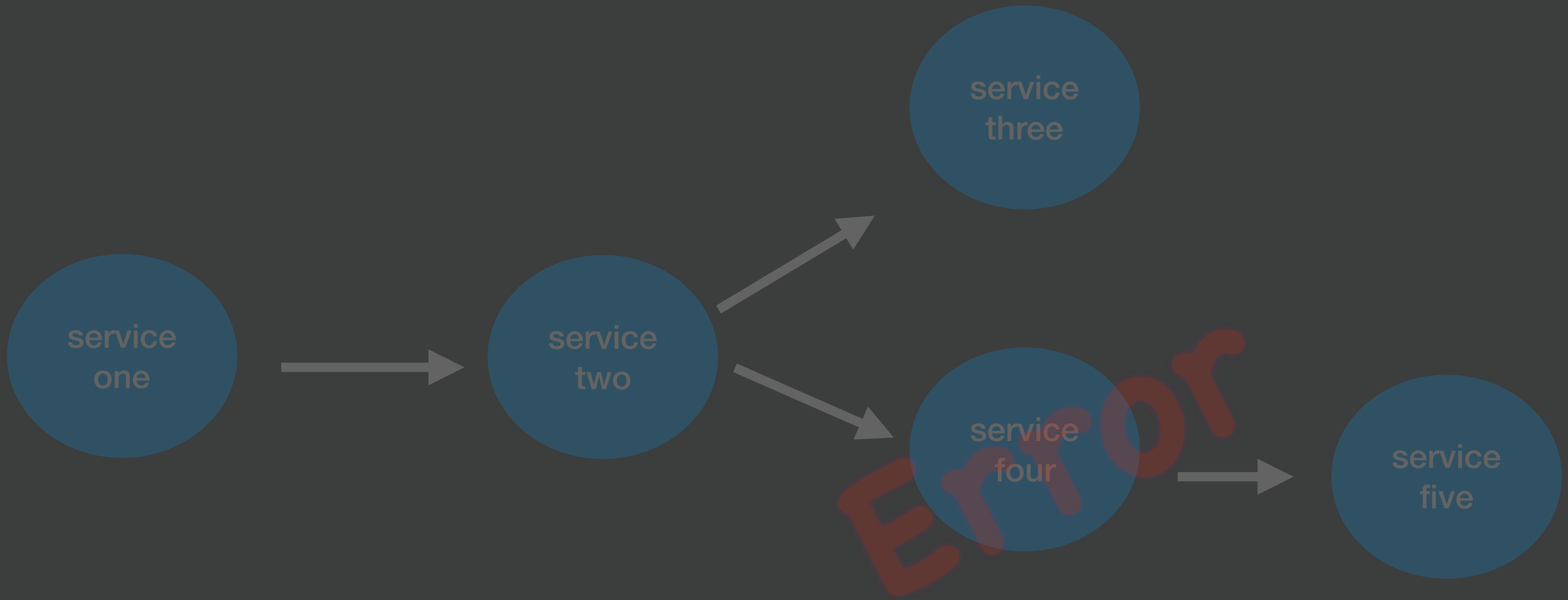
Where's the error?



Where's the error?



Where's the error?



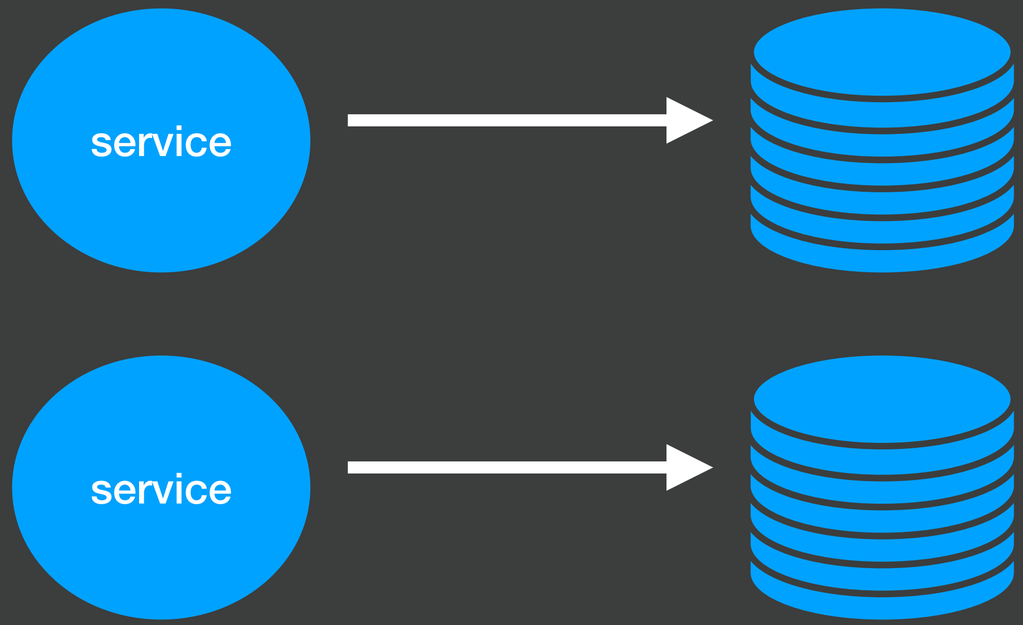
Stateless

Stateless

how to replicate it?

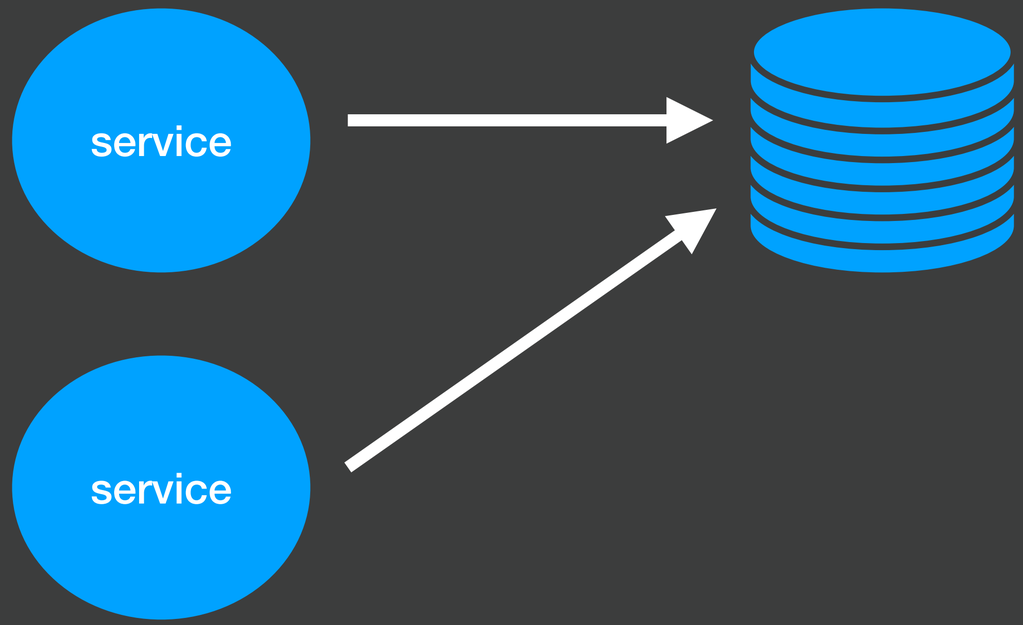
Stateless

same data ?



Stateless

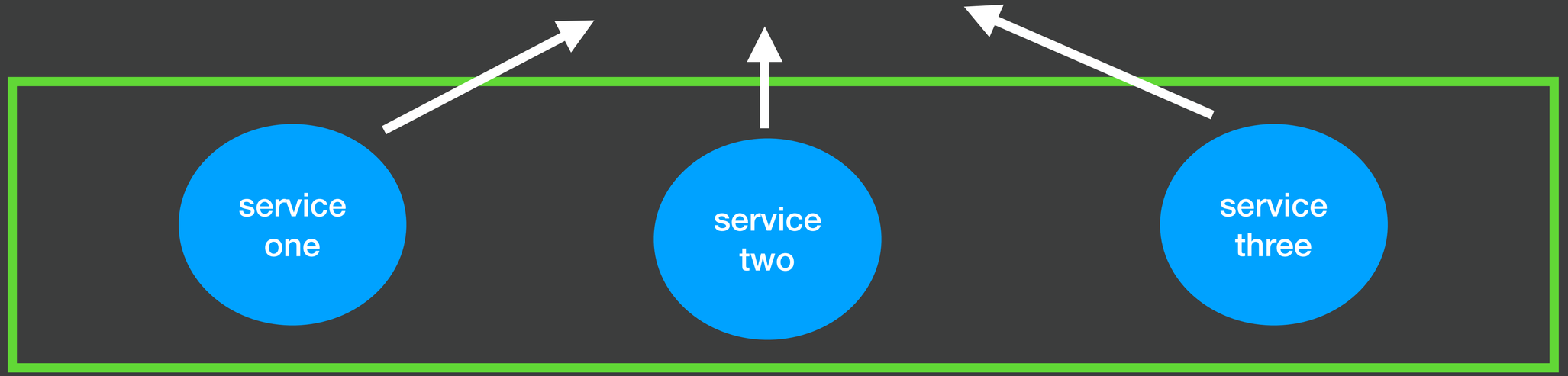
transition management



Independent

let's update the library

library X



Security

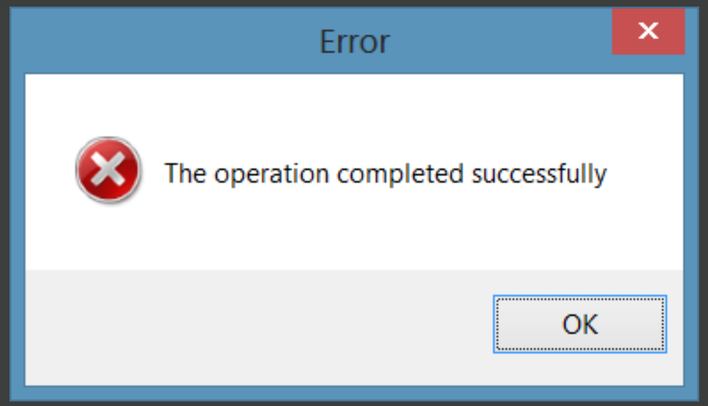
Security

Encrypt data	Control access
Keep dependencies up to date	…

Security

Security

Reliability



Error Handling

Testing

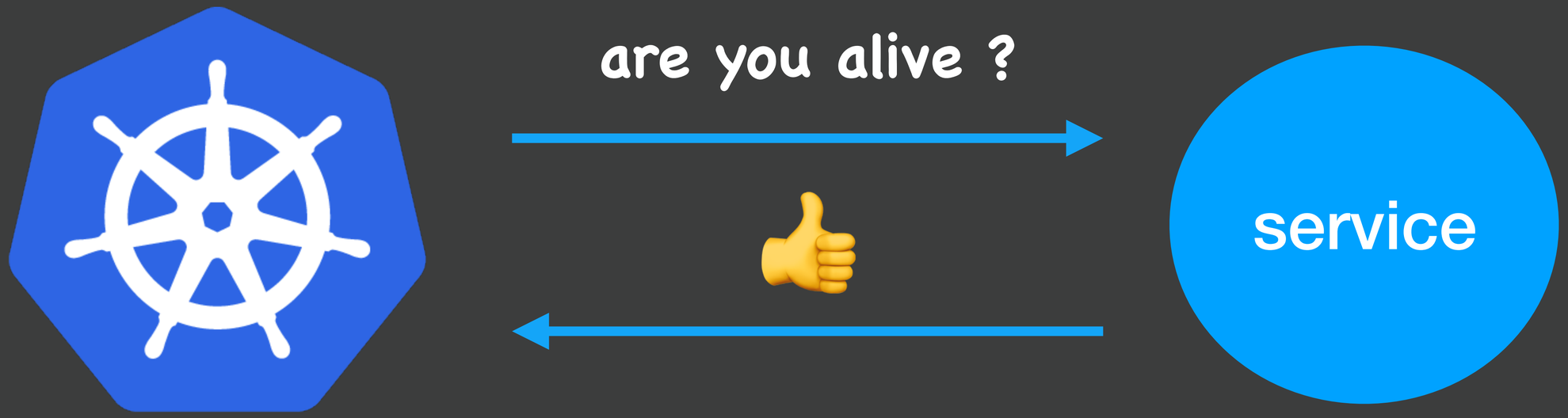
Chaos Engineering

Health endpoints

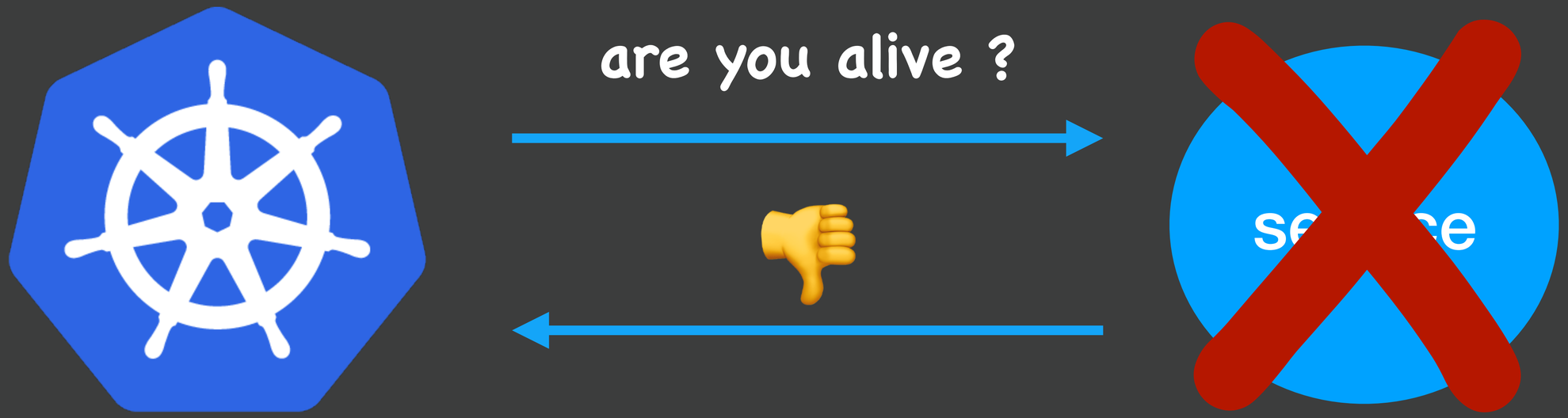
are you alive ?

service

Health endpoints



Health endpoints



There are tools for that

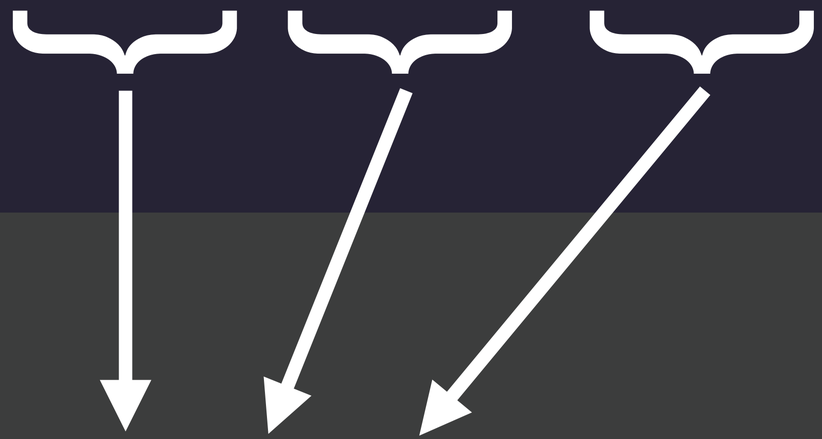
```
const server = http.createServer(app);
const { createTerminus } = require("@godaddy/terminus");

createTerminus(server, {
  healthChecks: {
    "/healthcheck": () => console.log("health check")
  }
});

const port = 3000;
server.listen(port, () => {
  console.log(`Beverage server is listening on port ${port}!`);
});
```

Environment Properties

```
mongoose.connect("mongodb://username:password@host:port/dbname", {
  useNewUrlParser: true
});
```



shouldn't be hardcoded here

Environment Properties

```
require("dotenv").config();

const url = `mongodb://${process.env.DB_USER}:${process.env.DB_PASSWORD}
@${process.env.DB_HOST}:${process.env.DB_PORT}/dbname`;

mongoose.connect(url, {
  useNewUrlParser: true
});
```

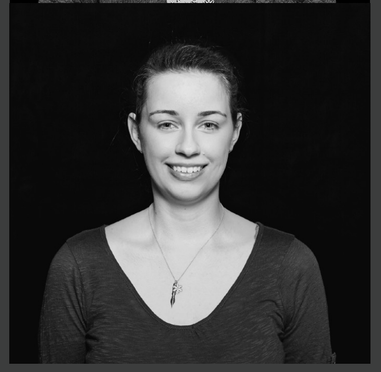
Think outside the box


Evaluate the situation.
Consider JavaScript.
Choose the best tool. :)

QUESTIONS?

Come talk to us!

CHRISTINA
@merelyChristina



ANNA
@merelyAnna

We're around today
and at the party later