Your backend is written in what?

JavaScript?!

@merelyChristina & @merelyAnna

Who are we?

CHRISTINA

merelyChristina

ANNA

merelyAnna

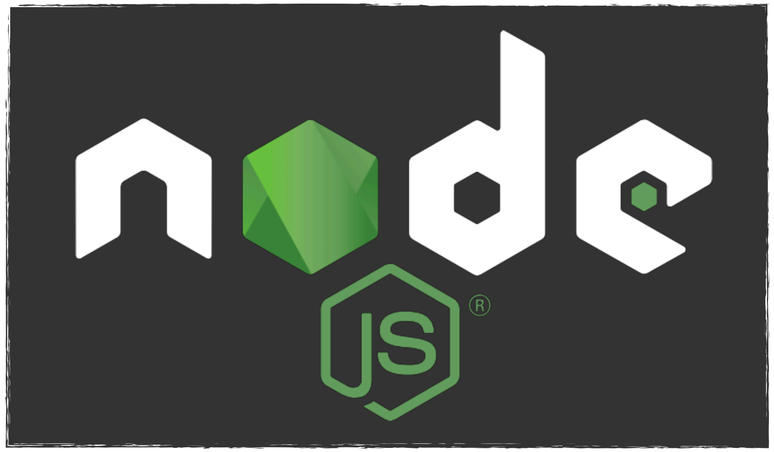
codecentric

JavaScript backends?

JavaScript backends!

Let’s talk about JavaScript runtimes

What is node.js ?



Asynchronous

Event driven

Non-blocking

JavaScript runtime

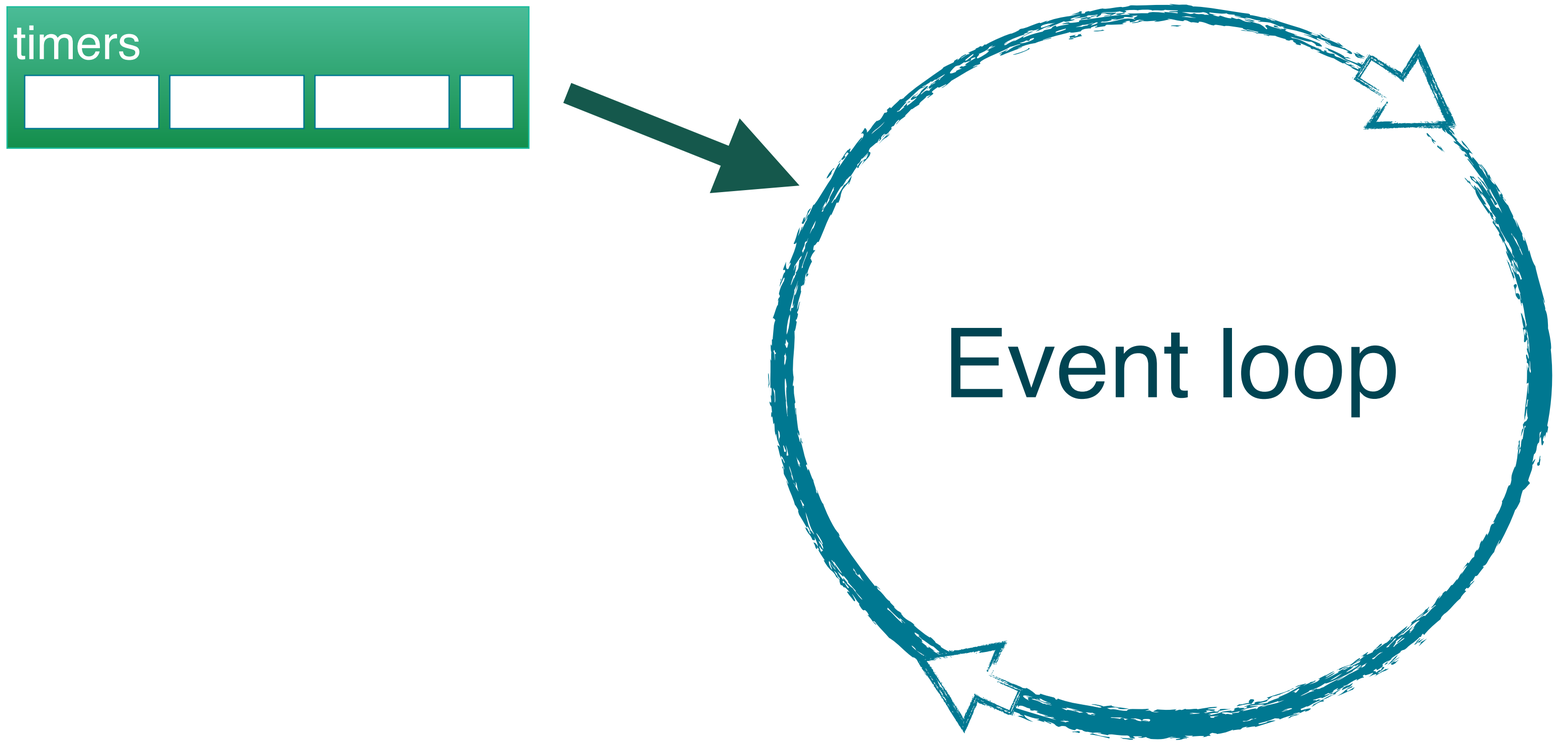
Google V8 JS Engine

Non-blocking? Event driven? What? How?

Non-blocking? Event driven? What? How?

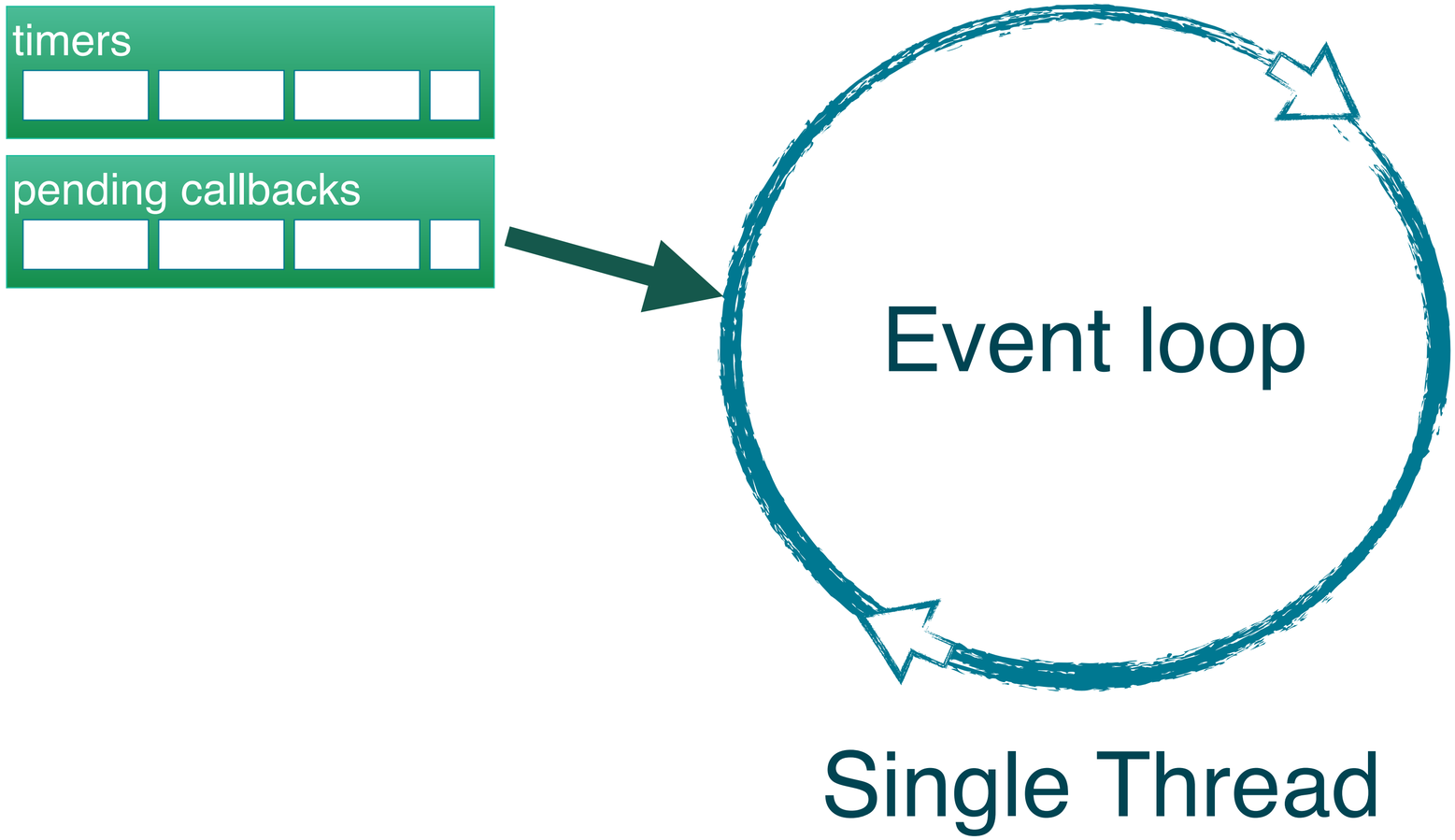
Single Thread

Non-blocking? Event driven? What? How?

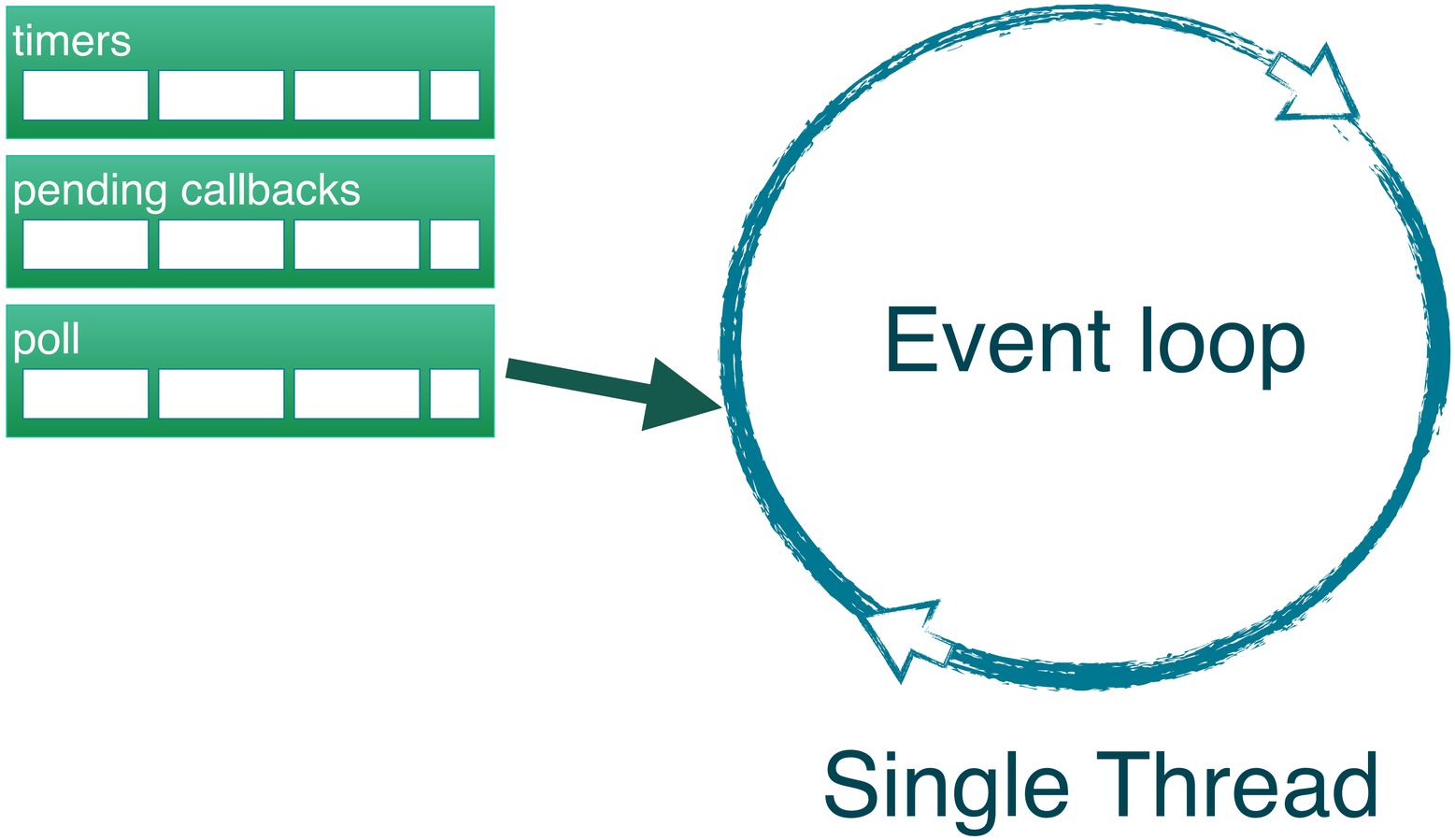


Single Thread

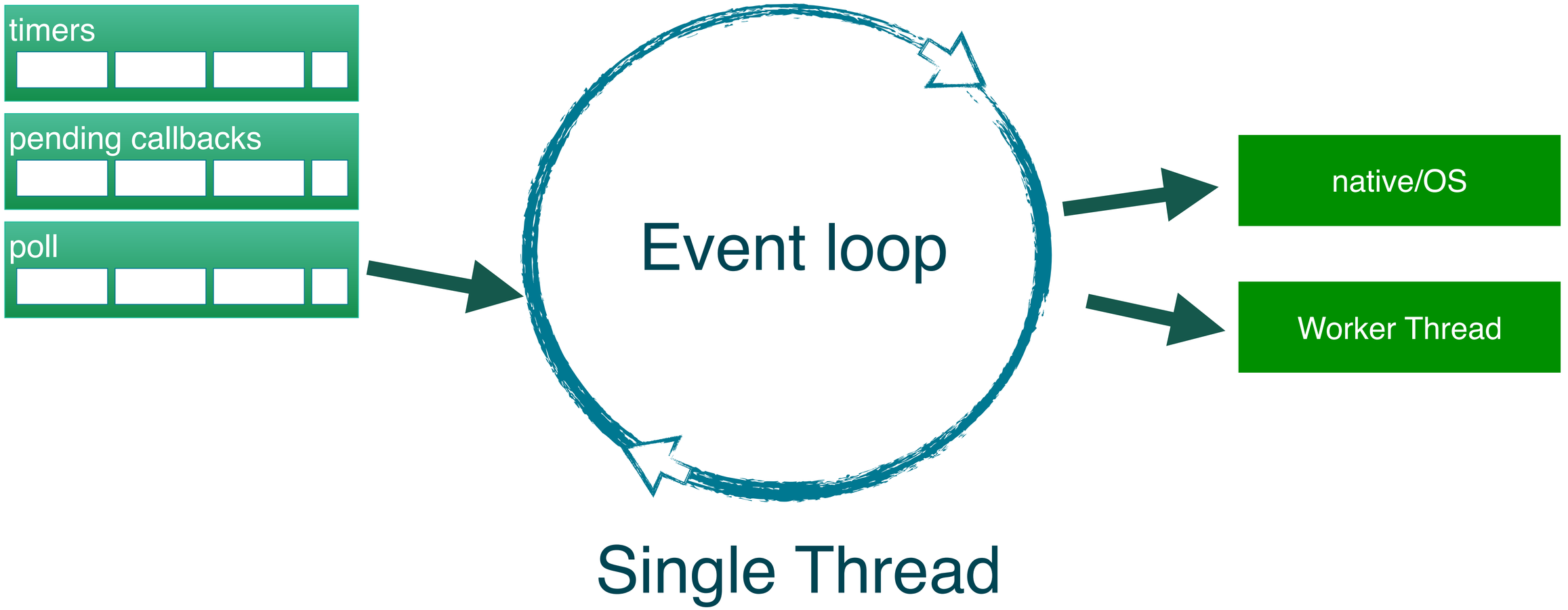
Non-blocking? Event driven? What? How?



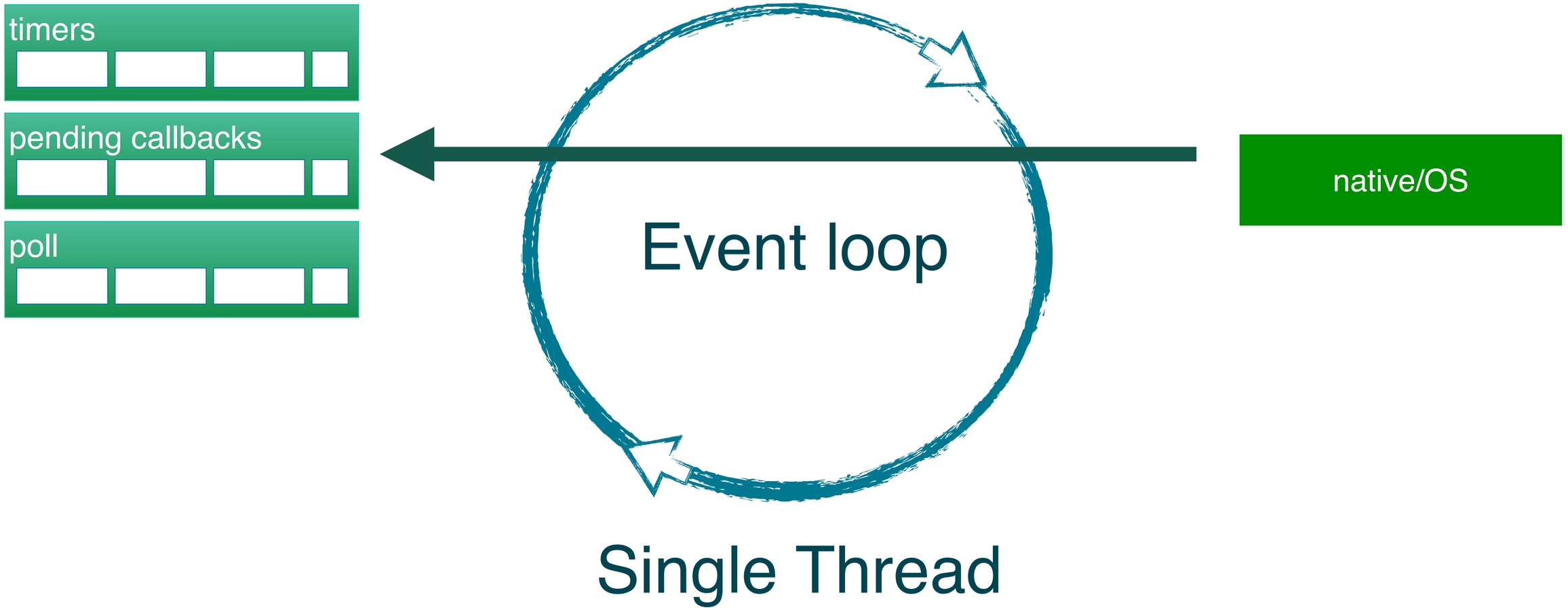
Non-blocking? Event driven? What? How?



Non-blocking? Event driven? What? How?

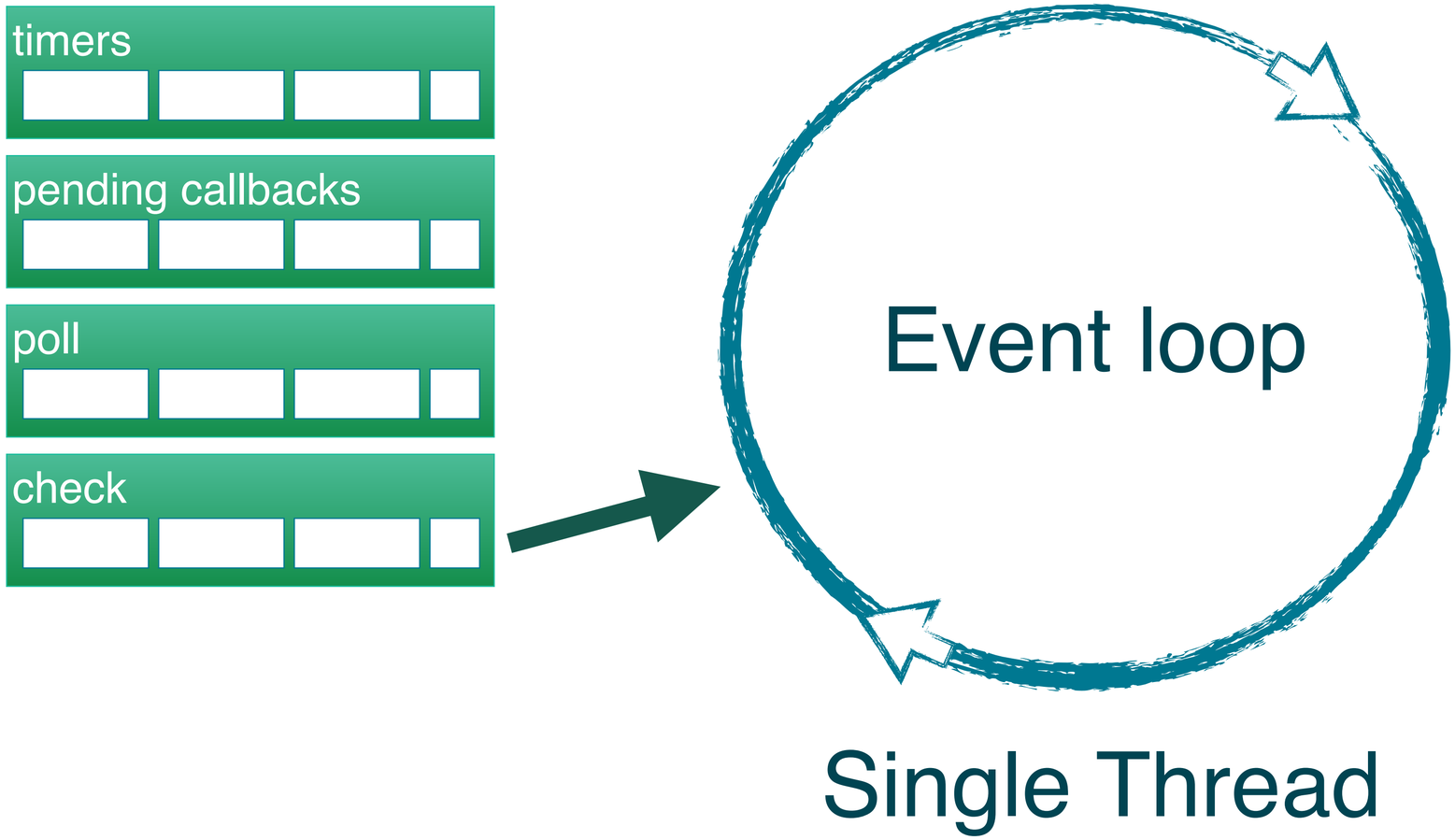


Non-blocking? Event driven? What? How?

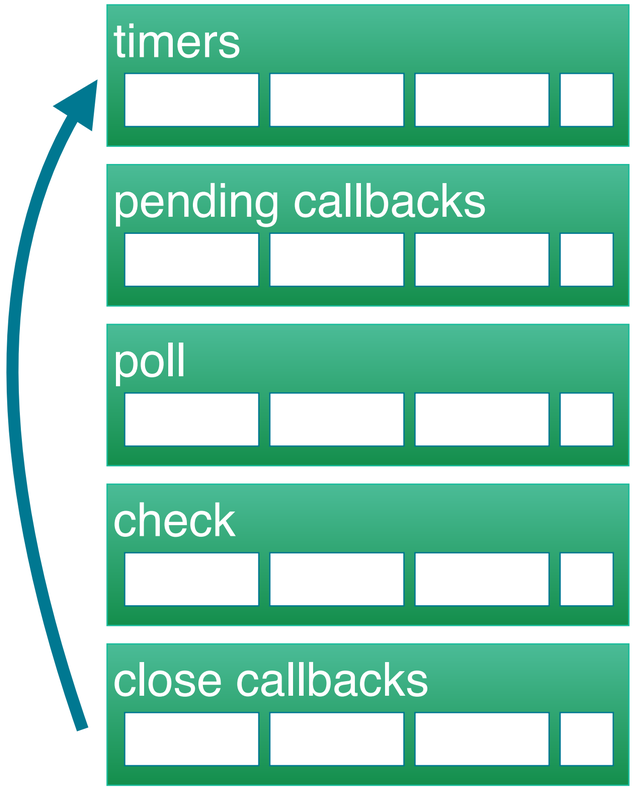


Single Thread

Non-blocking? Event driven? What? How?



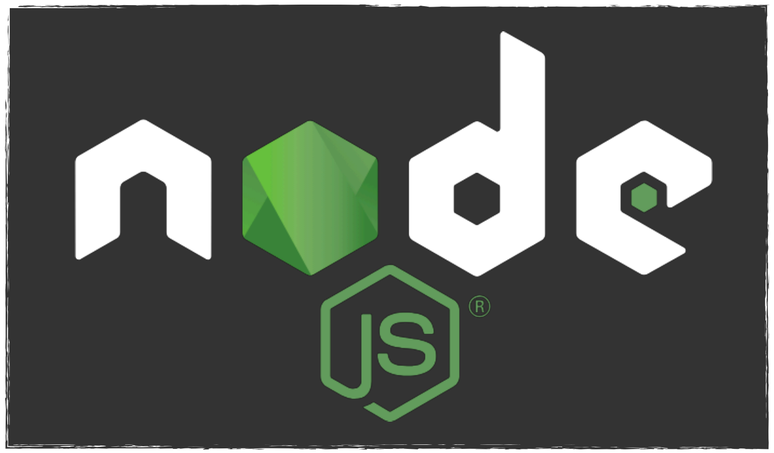
Non-blocking? Event driven? What? How?



Single Thread

native/OS

What else ?

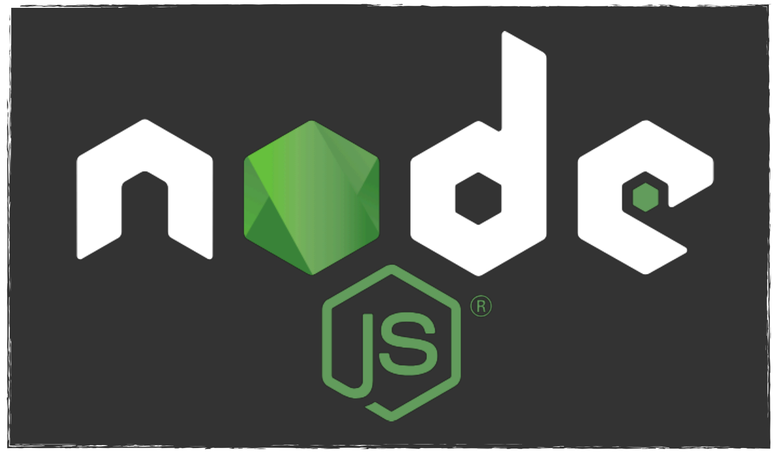


- Events
- File system
- HTTP/HTTPS
- …


Okay, we have a runtime!
But we also need a server!

A server ….

EXPRESS

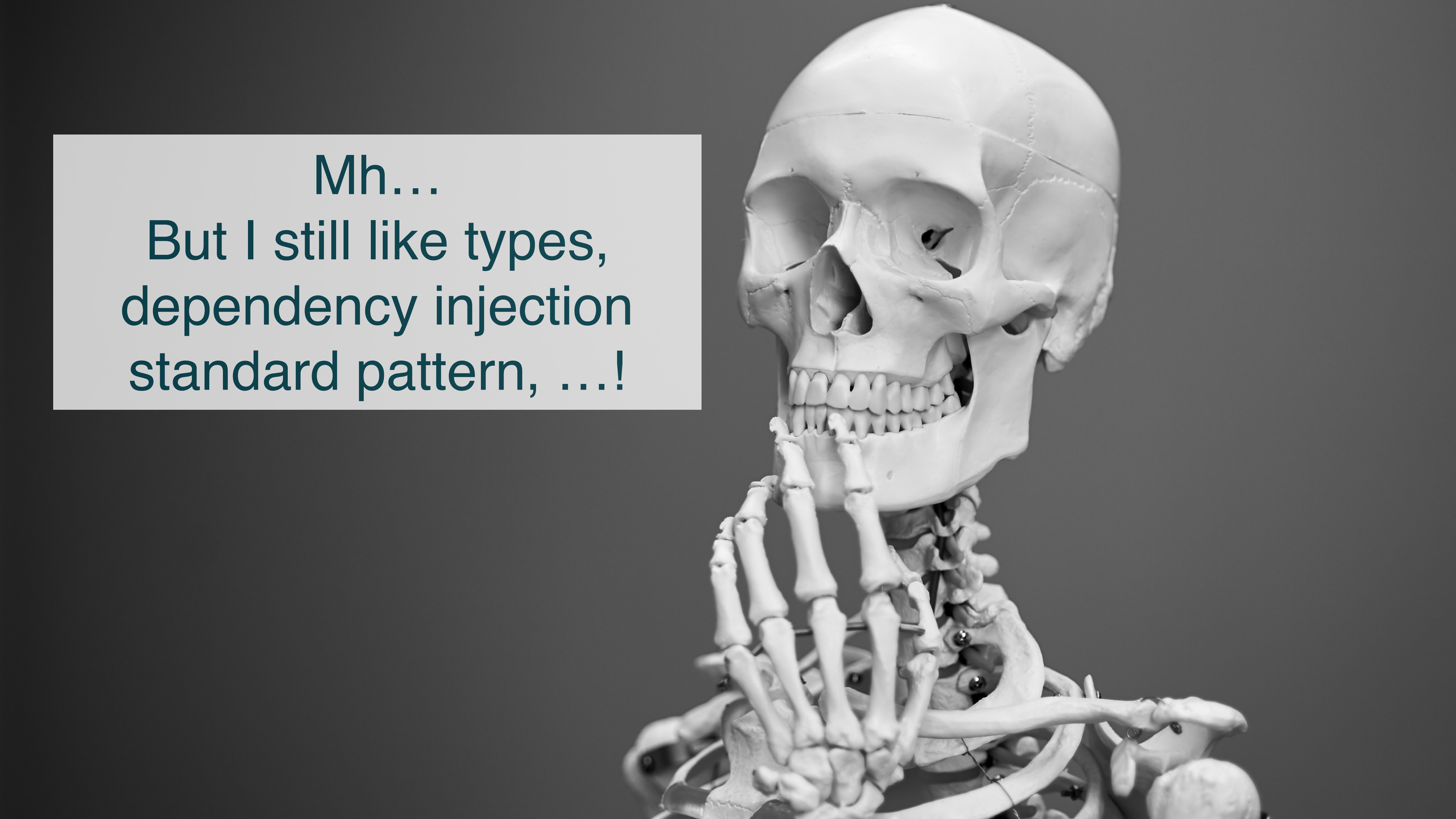


Web application framework

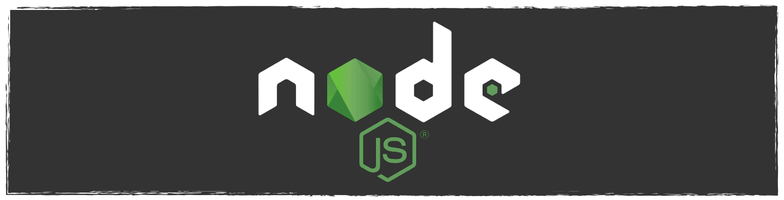
Express

```
const express = require("express");
const app = express();

const MongoClient = require("mongodb").MongoClient;
const client = new MongoClient("mongodb://localhost:27017");

client.connect(err => {
  const db = client.db("test");

  app.get("/coffee", async (req, res) => {
    const collection = db.collection("coffee");
    const coffee = await collection.find({}).toArray();
    res.send(coffee);
  });
});

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

Mh…
But I still like types, dependency injection standard pattern, …!



nest.js



scalable Node.js server-side applications

write in type script

nest.js

```
export interface Tea extends Document {
  name: string;
  type: string;
  steepingTime: string;
}
```

nest.js

```
@Injectable()
export class TeaService {
  private readonly teas: Tea[] = [];

  async getAllTeas(): Promise<Tea[]> {
    this.teas.push({ name: 'earl grey', type: 'black', steepingTime: '3min' });
    return Promise.resolve(this.teas);
  }
}
```

nest.js

```
@Controller()
export class TeaController {
  constructor(private readonly teaService: TeaService) {}

  @Get('/tea')
  getAllTeas(): Promise<Tea[]> {
    return this.teaService.getAllTeas();
  }
}
```

nest.js

```
@Module({
  imports: [],
  controllers: [TeaController],
  providers: [TeaService],
})
export class TeaModule {}
```

nest.js

```
async function bootstrap() {
  const app = await NestFactory.create(TeaModule);
  await app.listen(3000);
}
bootstrap();
```

Real time communication?
Sounds like socket

socket.io

```
const app = express();
const server = http.createServer(app);
const io = socketServer(server);

io.on("connection", function(socket) {
  console.log("Connected to Socket!!" + socket.id);

  socket.on("GET_WATER", payload => {
    const water = { id: "1", type: "sparkling", taste: "lemon" };
    io.emit("SERVE_WATER", water);
  });
});

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

socket.io

```
WebSocketGateway();
export class SearchGateway {
  @WebSocketServer() server;

  @SubscribeMessage('GET_WATER')
  async onEvent(client, data: any): Promise<any> {
    return {
      event: 'SERVE_WATER',
      data: { id: '1', type: 'sparkling', taste: 'lemon' },
    };
  }
}
```

Okay.

...

...

...

Now what?

When to consider a JavaScript-backend?

Fast switching between input and output	Fewer context switches
Scaleability	Socket-communication
Typeless	…

So….

… should I now write all my backends with JavaScript?


NO!
Please don’t.

When not to consider a JavaScript-backend?

File based operations	Calculations
Multiple CPUs needed	Blocking tasks
Single Thread per user	…


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

QUESTIONS?

COME TALK TO US! :)

CHRISTINA

merelyChristina

ANNA

merelyAnna