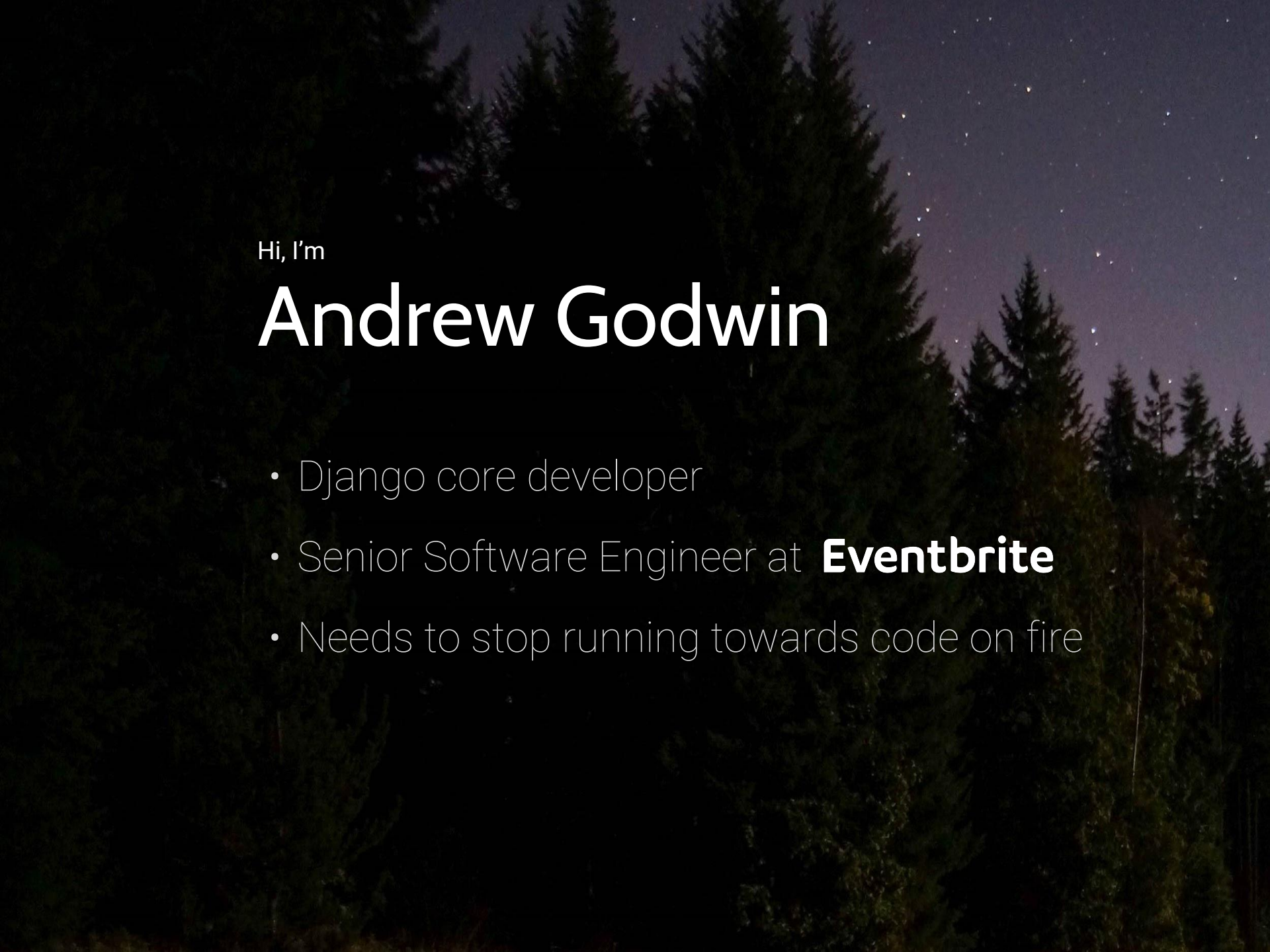


# Taking Django Distributed





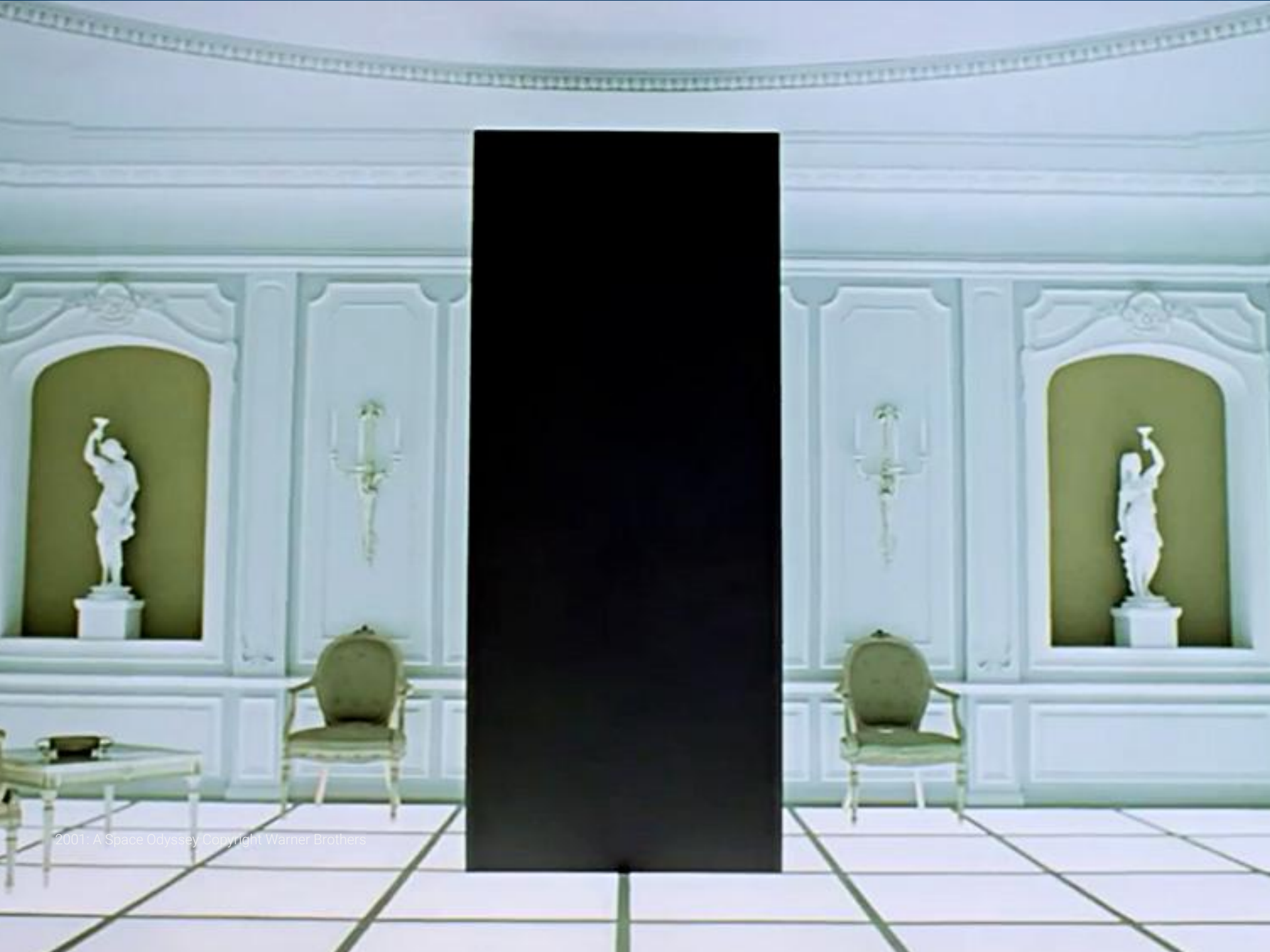
Hi, I'm

# Andrew Godwin

- Django core developer
- Senior Software Engineer at **Eventbrite**
- Needs to stop running towards code on fire

Computers hate you.

This makes distributed hard.



© 2001. A Space Odyssey Copyright Warner Brothers

It's time to split things up a bit.

But how? And why?

Code

Databases

Team

There is no one solution

Read-heavy?

Write-heavy?

Spiky?

Predictable?

Chatty?

Code

# Use apps! They're a good start!

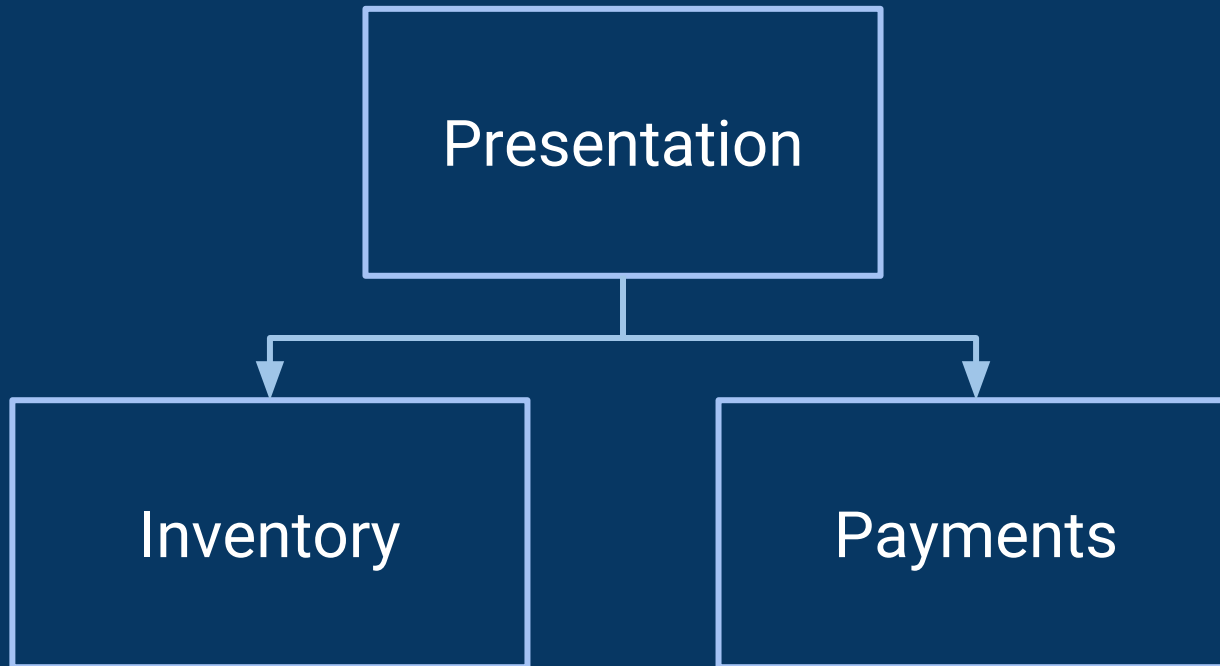
Ignore the way I wrote code for the first 5 years of Django.

# Formalise interfaces between apps

Preferably in an RPC style

# Split along those interfaces

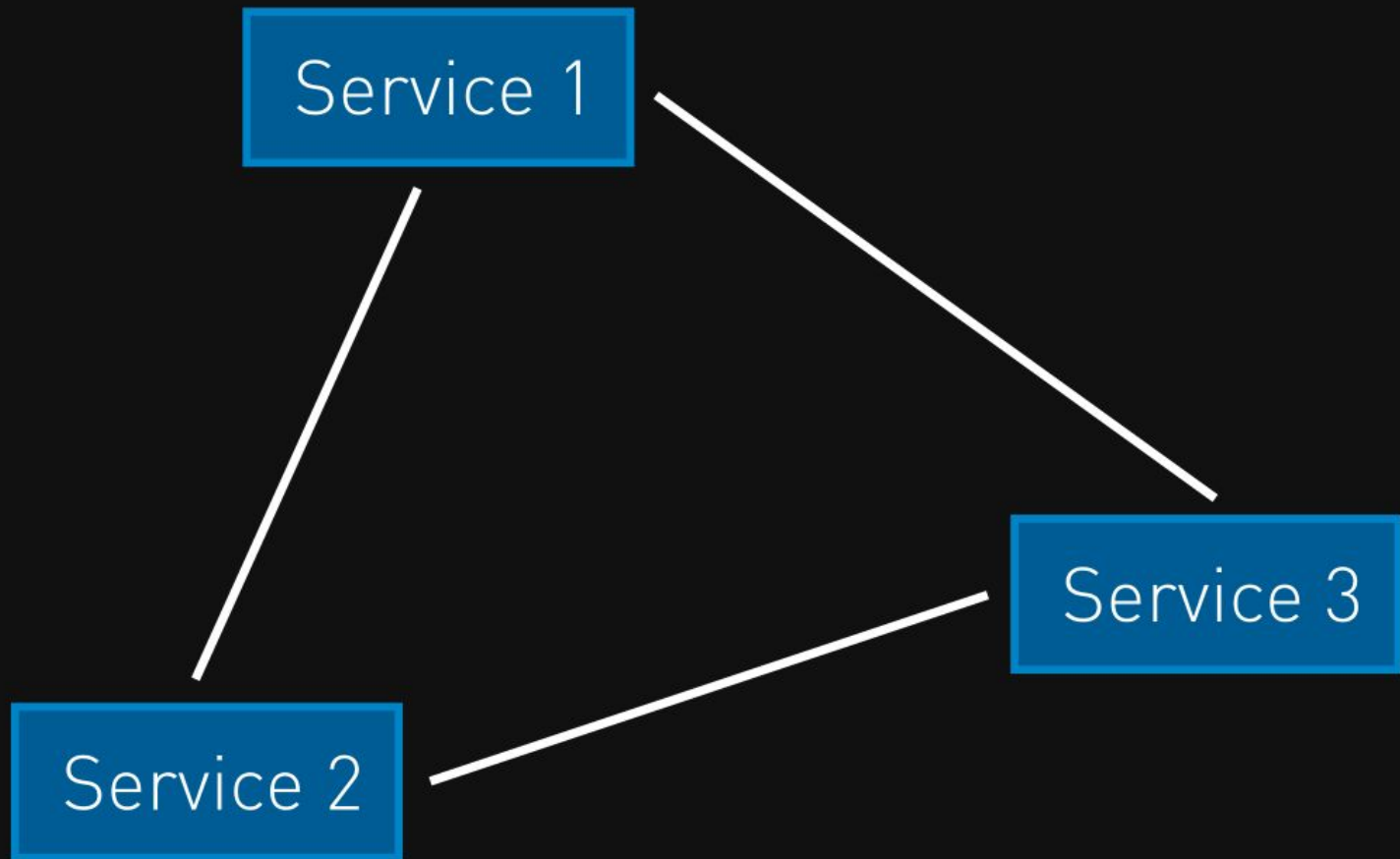
Into separate processes, or machines

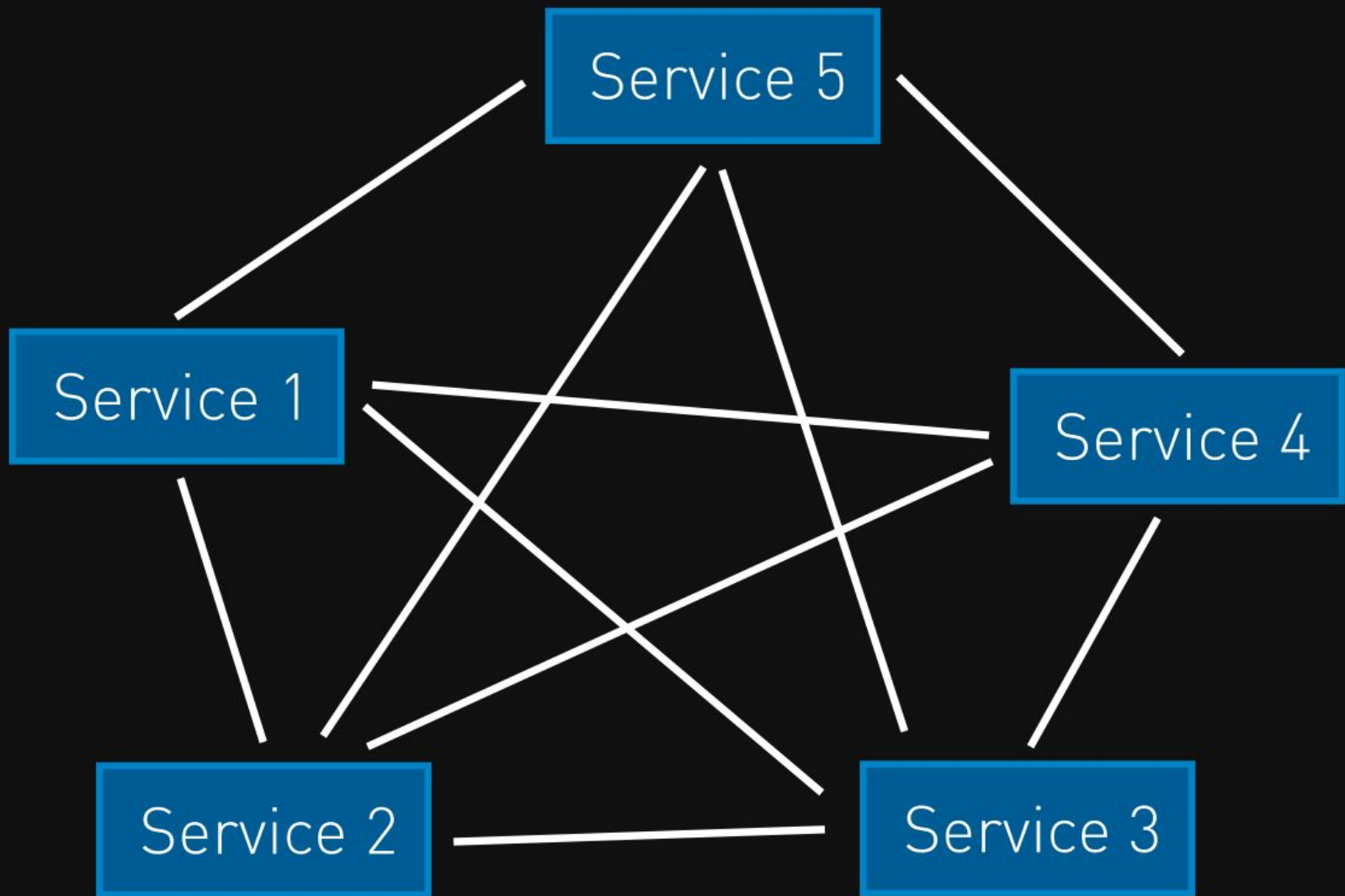


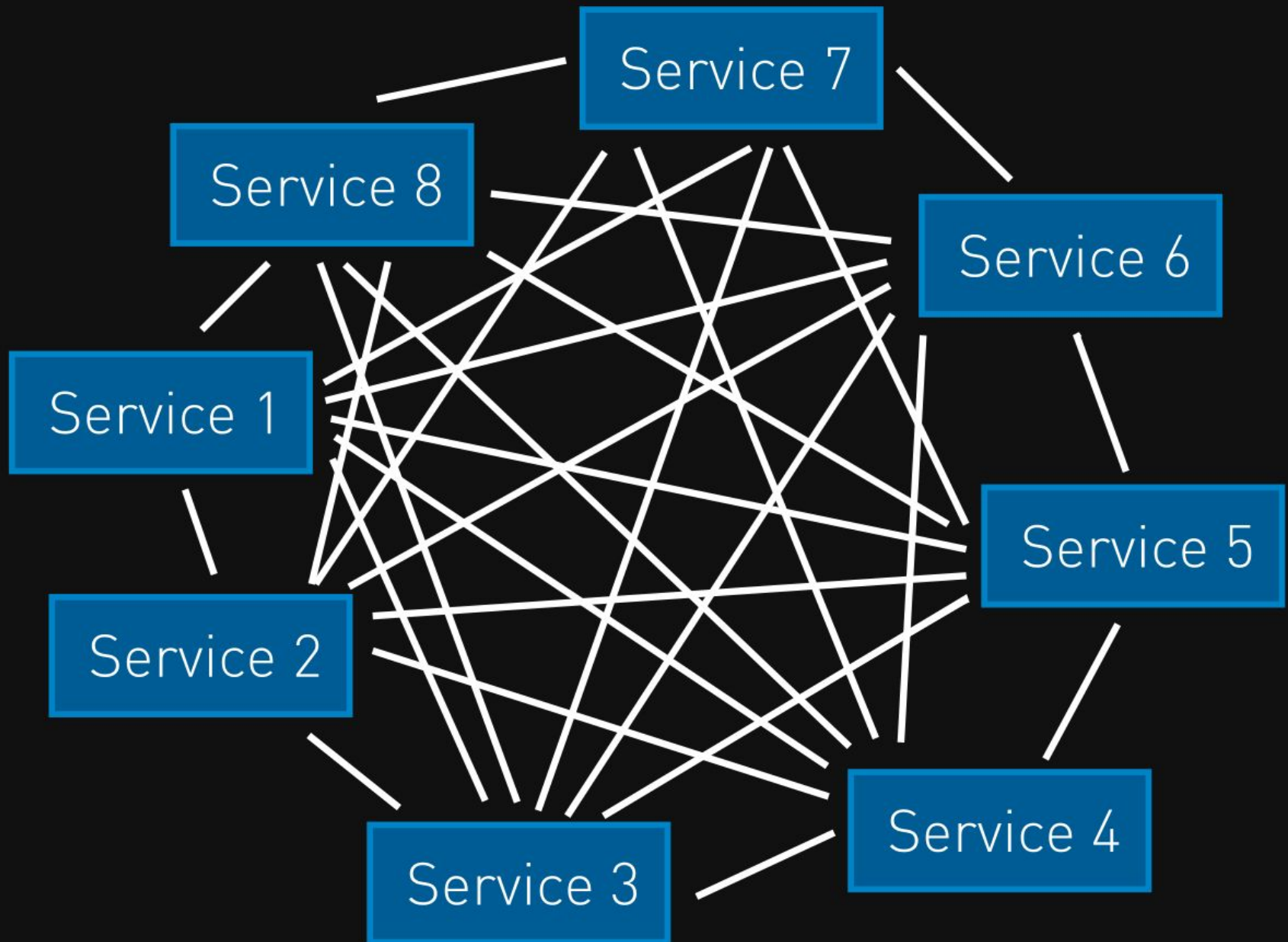
# How do you communicate?

HTTP? Channels? Smoke signals?

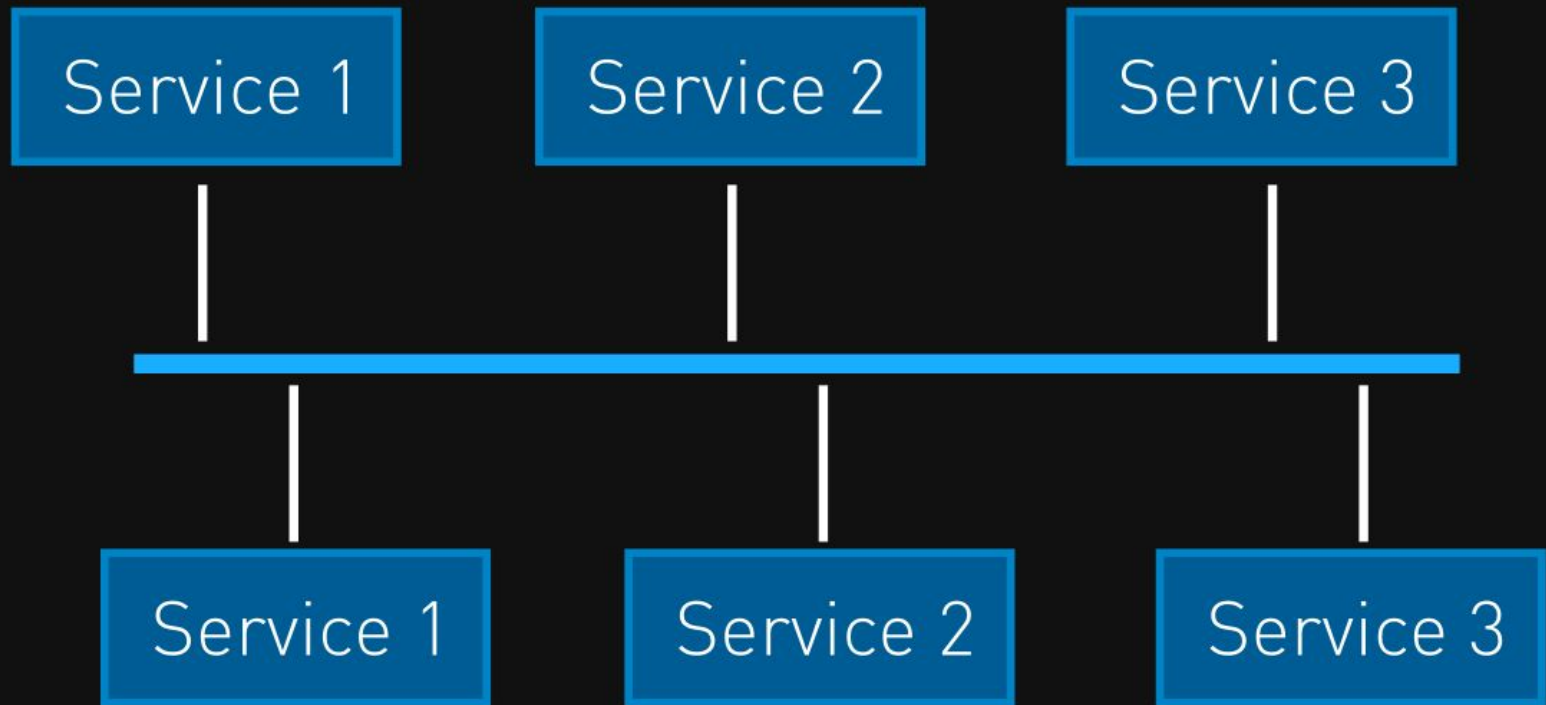
# Direct Communication







# Message Bus



# Databases

# Vertically Partitioned Database



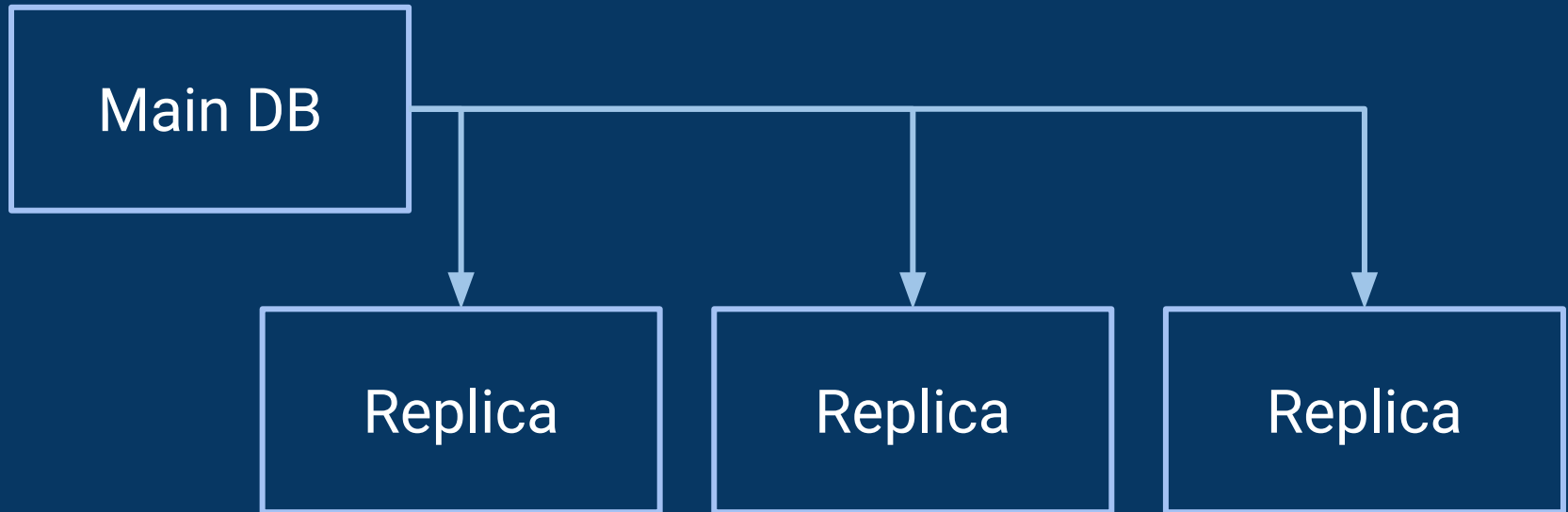
Users

The diagram illustrates a vertically partitioned database structure. It consists of three rectangular boxes stacked vertically, each representing a different table. The top box is labeled 'Users', the middle box is labeled 'Images', and the bottom box is labeled 'Comments'. Each box has a thin white border and is centered on the slide.

Images

Comments

# Single main database with replication



Partition Tolerant

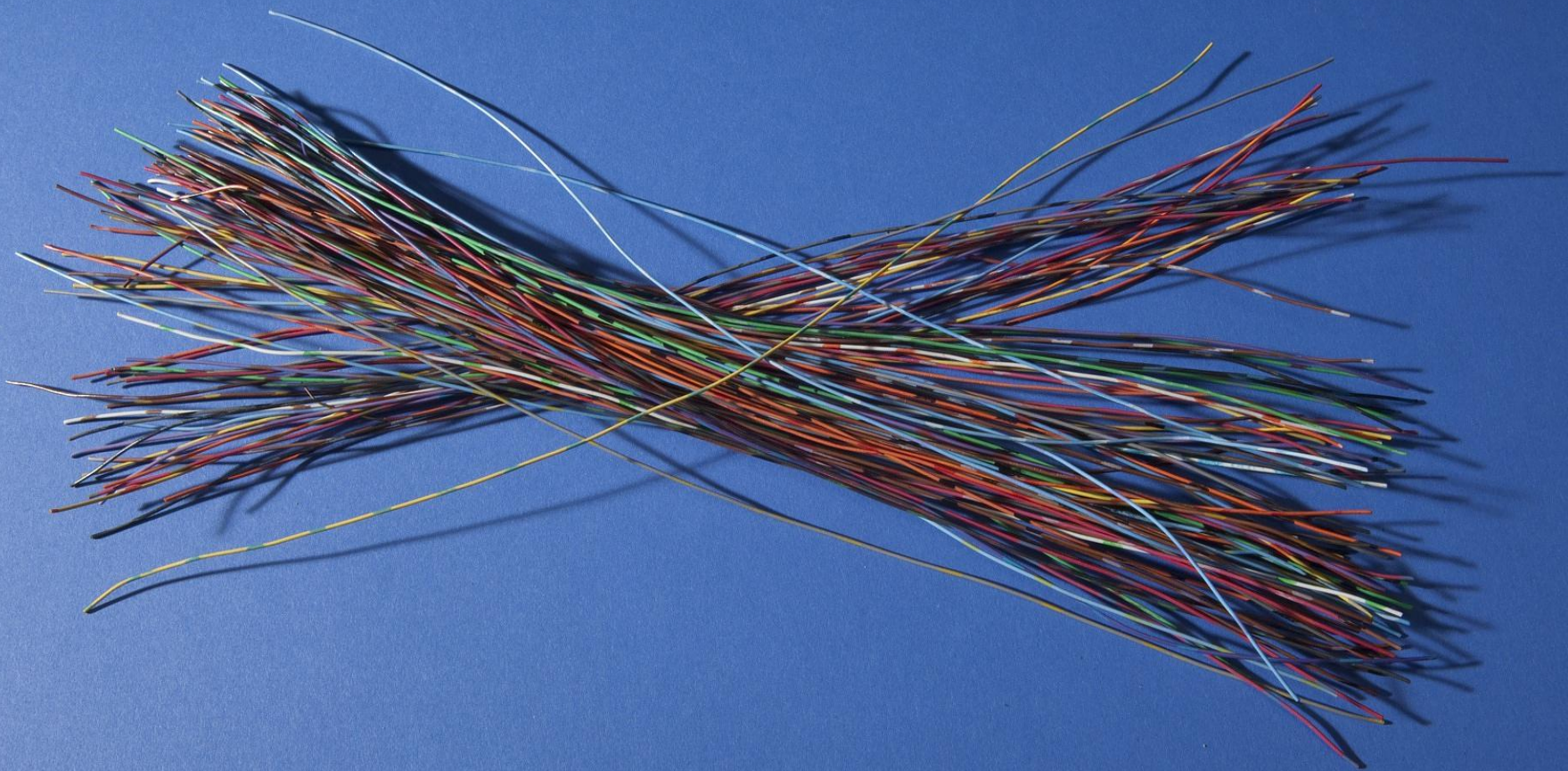


Available

Consistent

# Non-consistency is everywhere

It's sneaky like that



# Load Balancing

Equally balanced servers

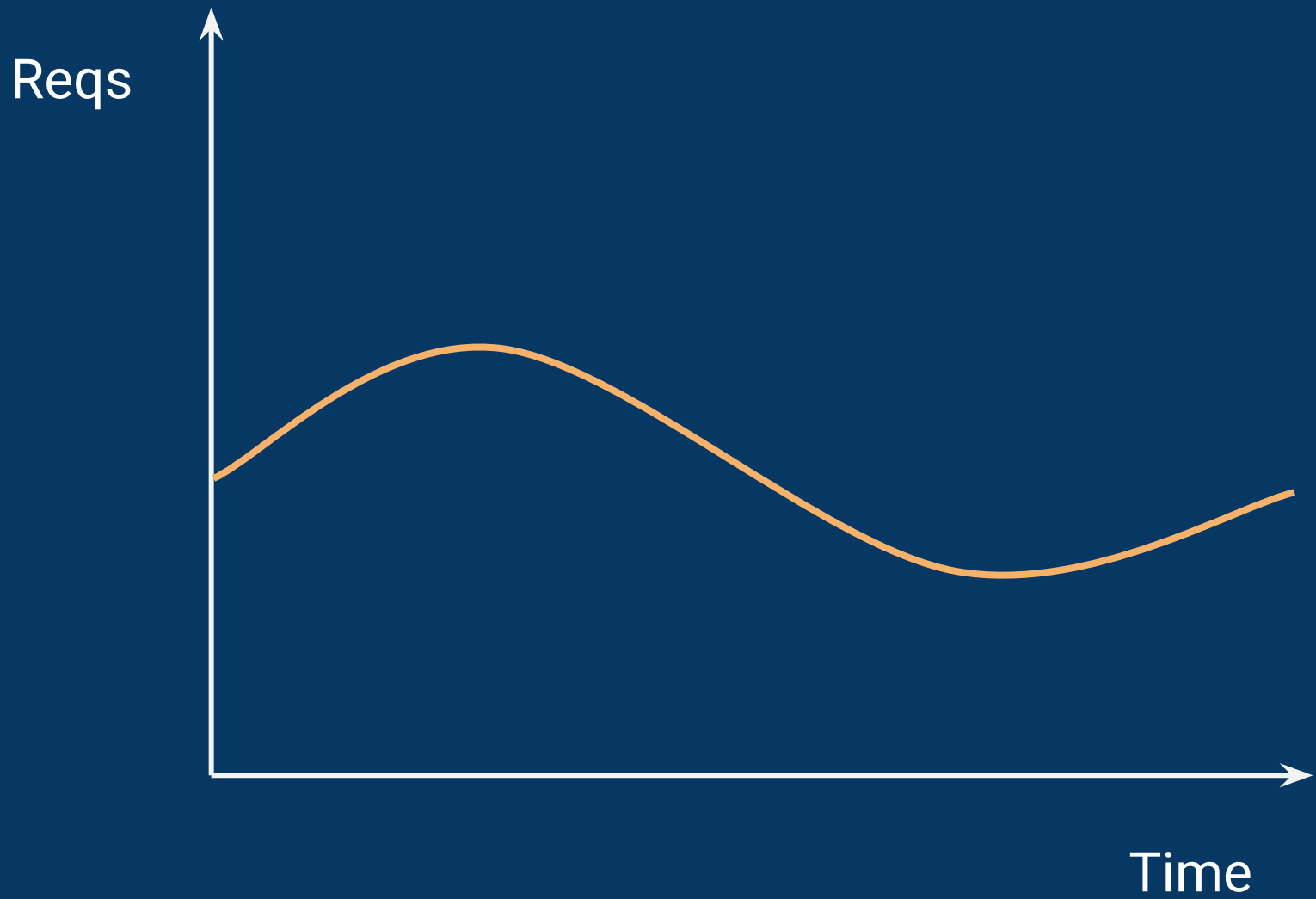
Consistent load times

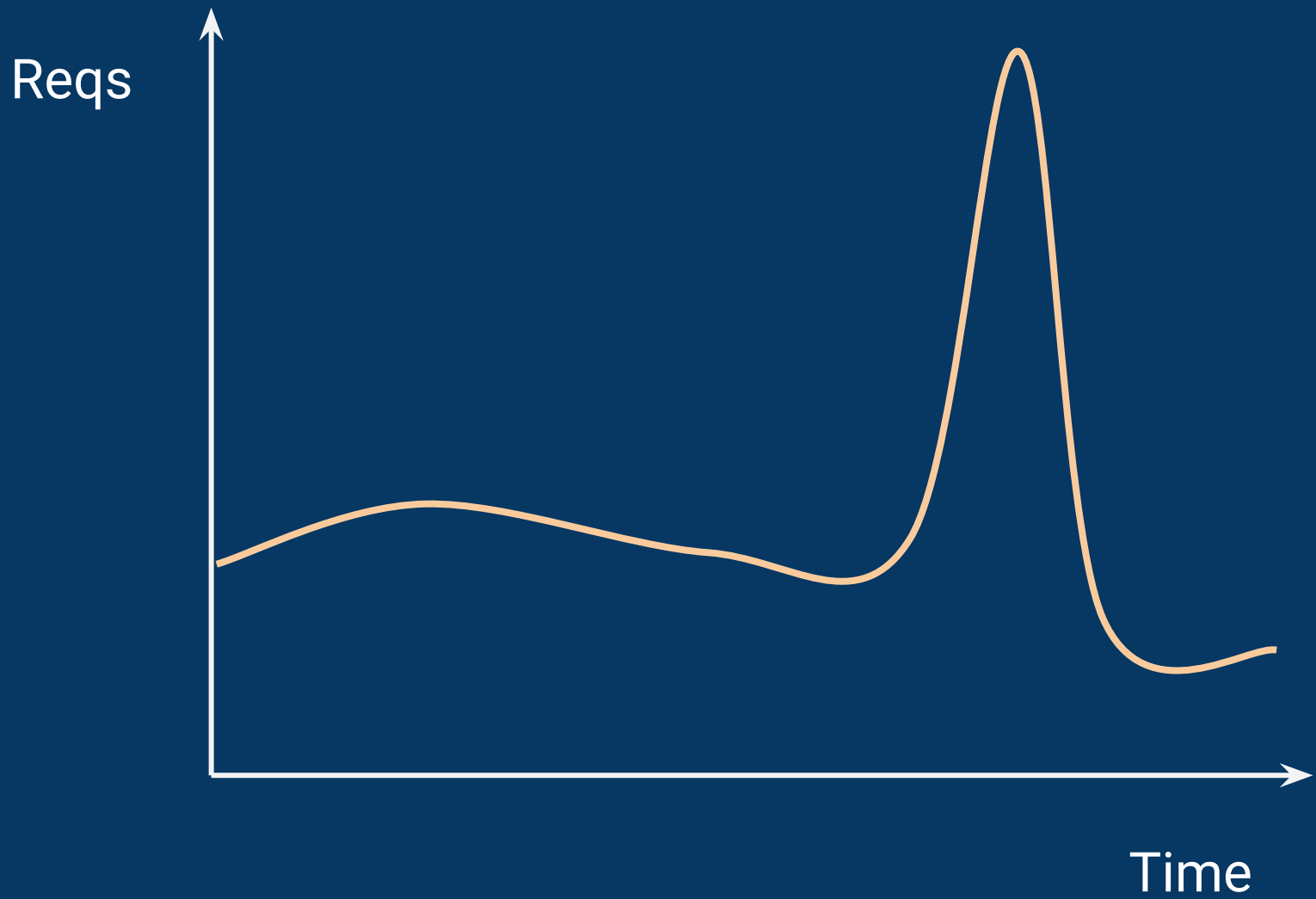
Similar users

Split logic

Different processor loads

Wildly varying users





WEBSOCKETS

# WEBSOCKETS

- They can last for hours
- There's not many tools that handle them
- They have 4 different kinds of failure

# Design for failure, and then use it!

Kill off sockets early and often.

Team

# Developers are people too!

They need time and interesting things

# Technical debt can be poisonous

But you need a little bit to compete

# Single repo? Multiple repos?

Each has distinct advantages.

# Teams per service? Split responsibility?

Do you split ops/QA across teams too?

# Ownership gaps

They're very hard to see.

# Strategies

# Don't go too micro on those services

It's easier in the short term, but will confuse you in the long term.

# Communicate over a service bus

Preferably Channels, but you get to choose.

# Work out where to allow old data

Build in deliberate caching or read only modes

# Design for future sharding

Route everything through one model or set of functions

# Expect long-polls/sockets to die

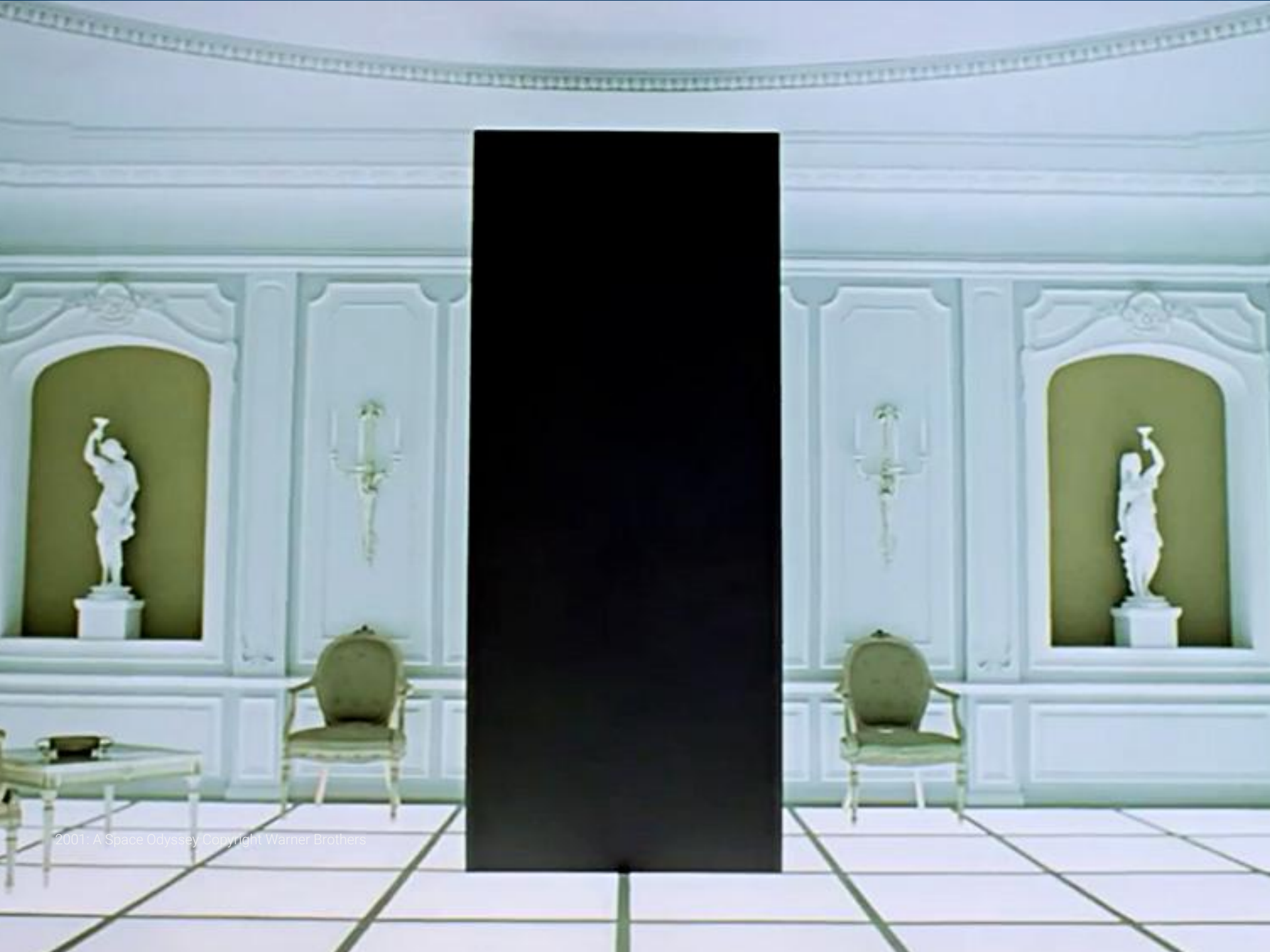
Design for load every time, and treat as a happy optimisation

# Independent, full-stack teams

From ops to frontend, per major service

# Architect as a part-time position

You need some, but not in an ivory tower



© 2001. A Space Odyssey Copyright Warner Brothers

# Maybe, just maybe, keep that monolith

A well maintained and separated one beats bad distributed

# Thanks.

Andrew Godwin  
@andrewgodwin    aeracode.org