Advanced Networking and Distributed Systems

Microservices and Communication Frameworks

GW CSCI 3907/6907
Lucas Chaufournier and Timothy Wood

What is a Distributed System? A recap

???

What is a Distributed System? A recap

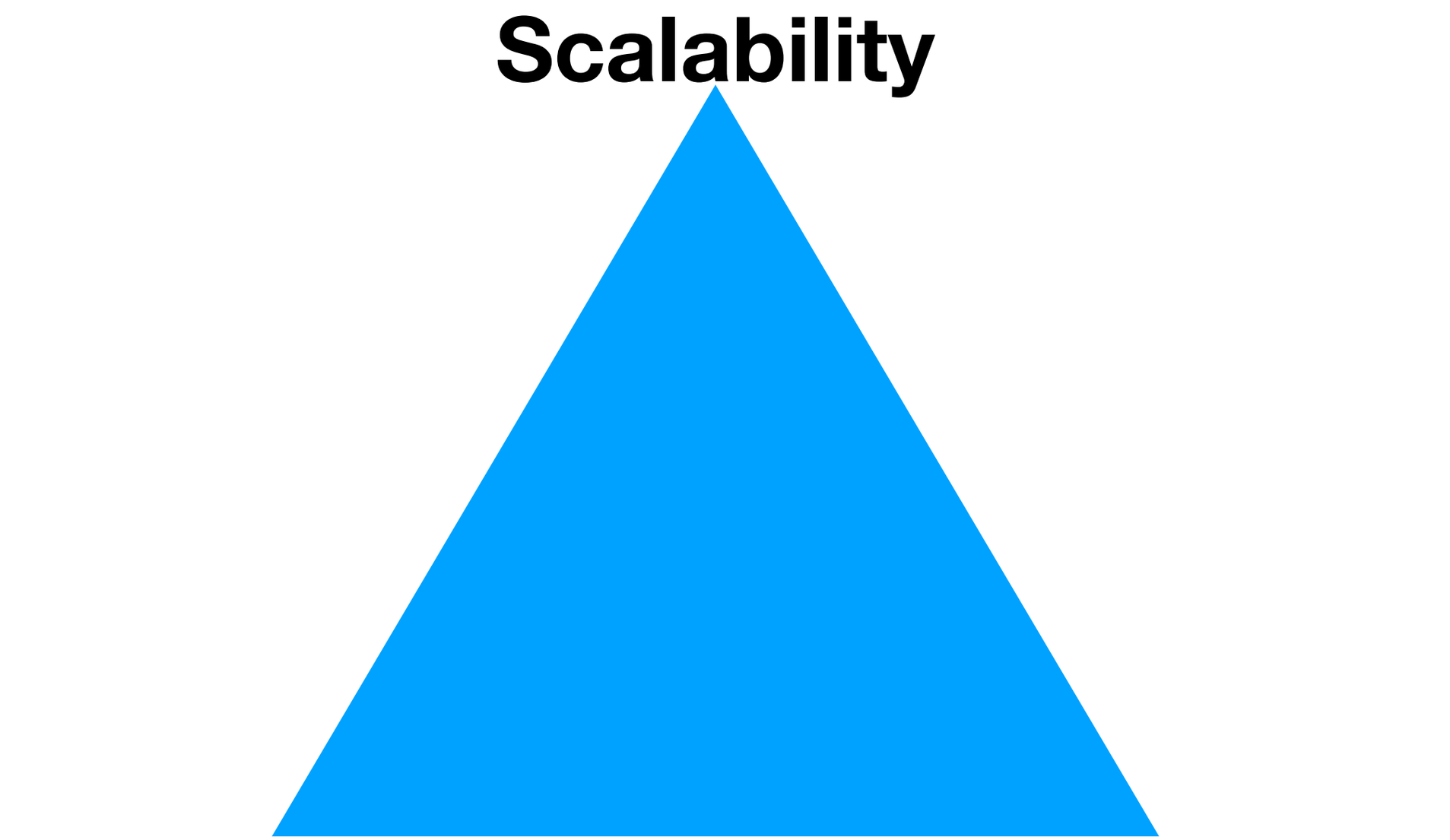
???

A distributed system is a collection of independent processes that appears to its users as a single coherent system.

-> Communication is key to collaboration!

How we establish and define this communication is critical.

A guide for judging systems



Performance

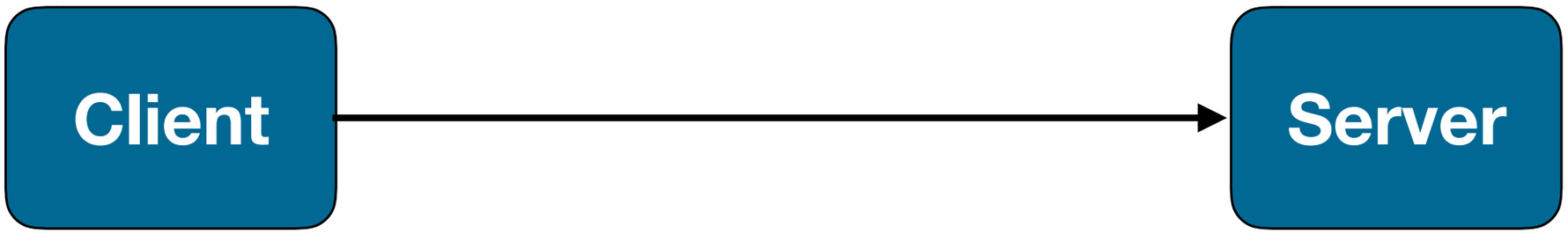
The amount of useful work accomplished by a computer system compared to time and resources used.

Examples of performance measures:

- Short response time aka Low Latency
- High Throughput
- Low Utilization of Resources

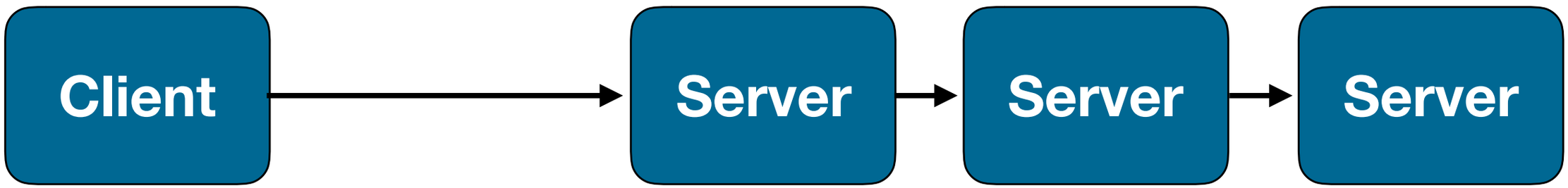
How is this different from the basic performance metrics we discussed in networking?

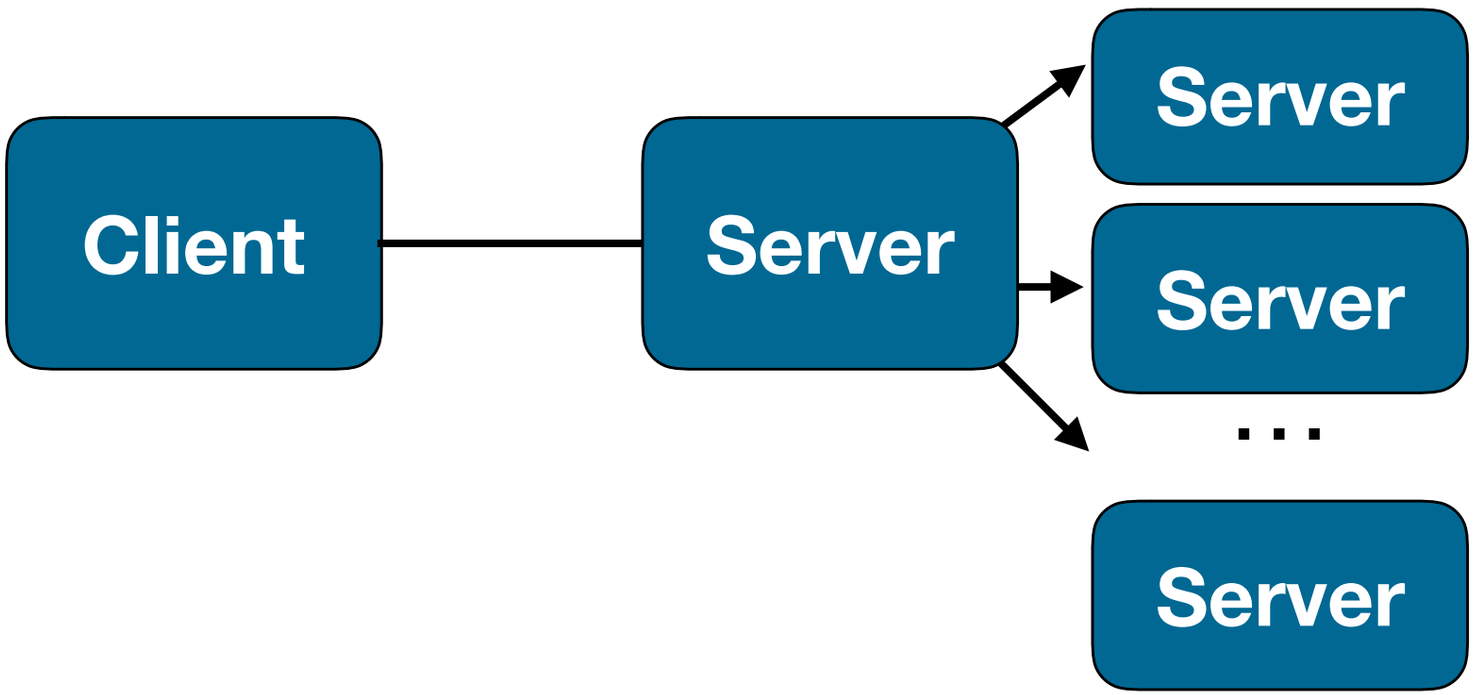
Distributed Performance



Distributed Performance

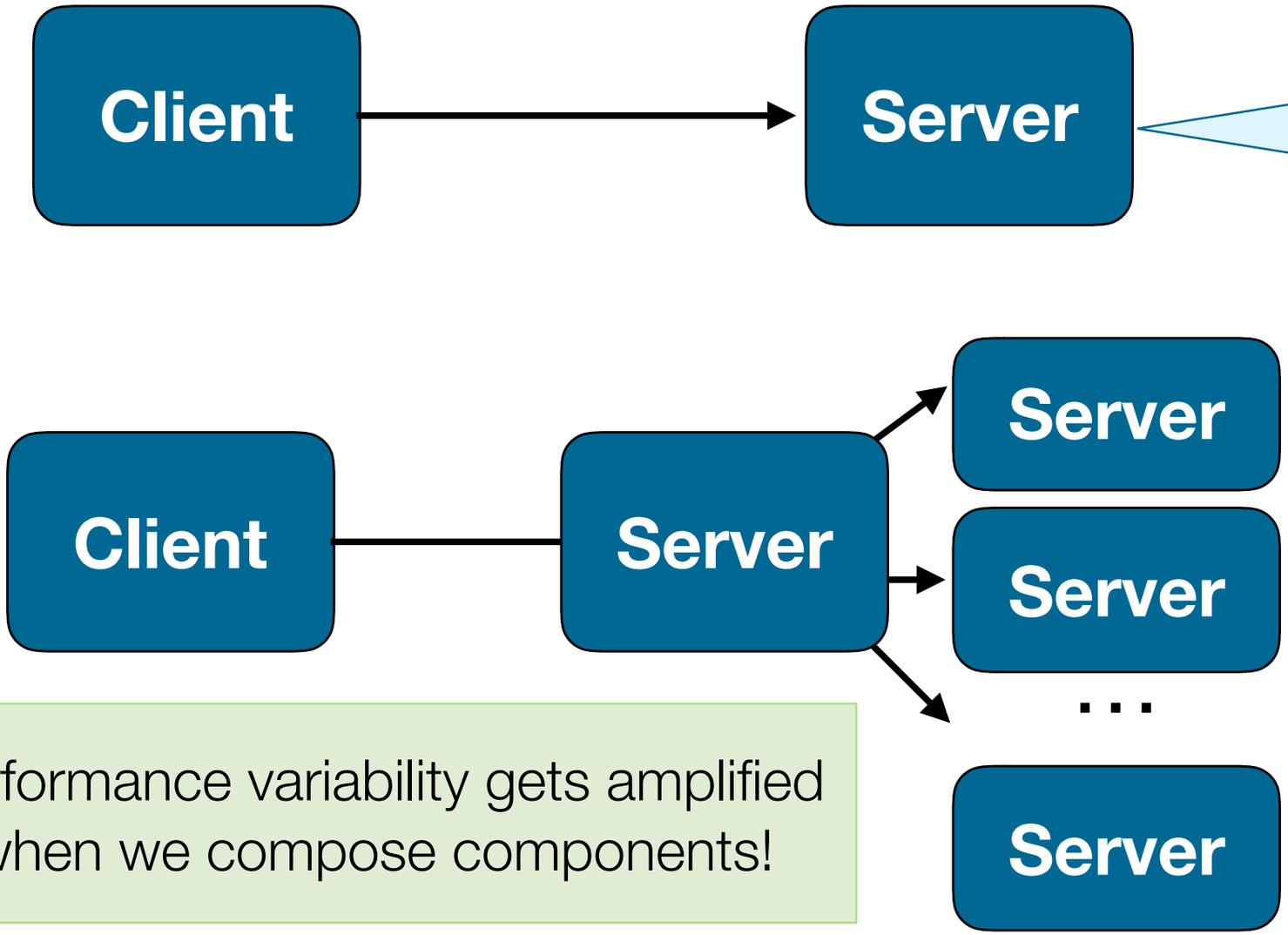
Communication and interacting components can compound performance problems

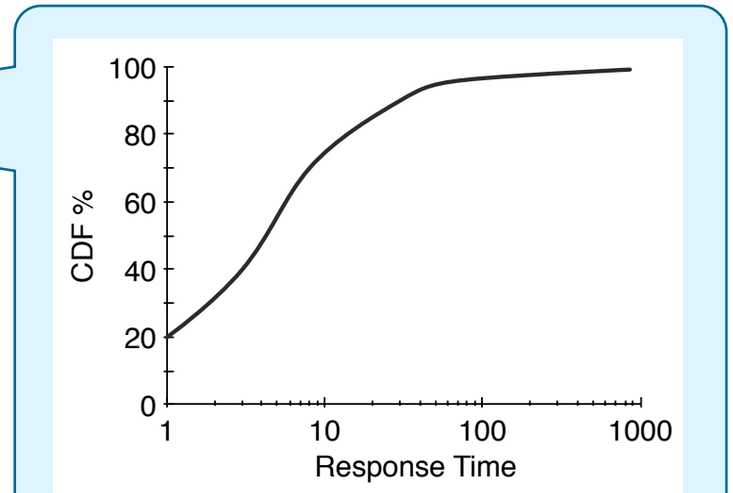




Distributed Performance

Communication and interacting components can compound performance problems





99th %ile -> 1 second

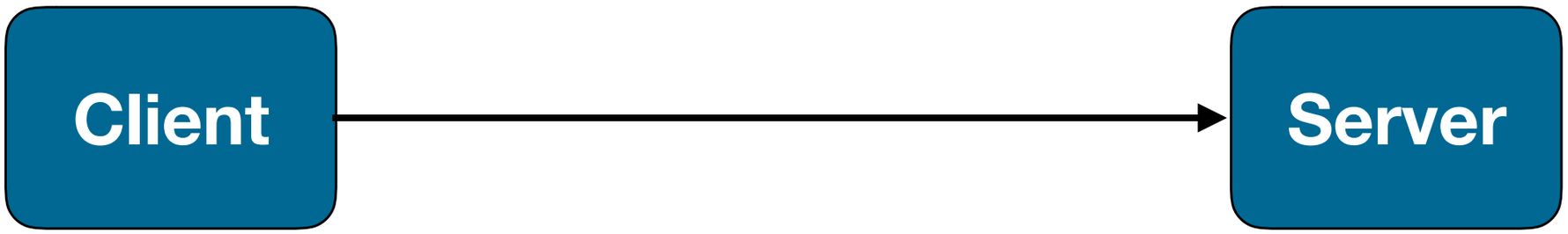
Performance variability gets amplified when we compose components!

Availability

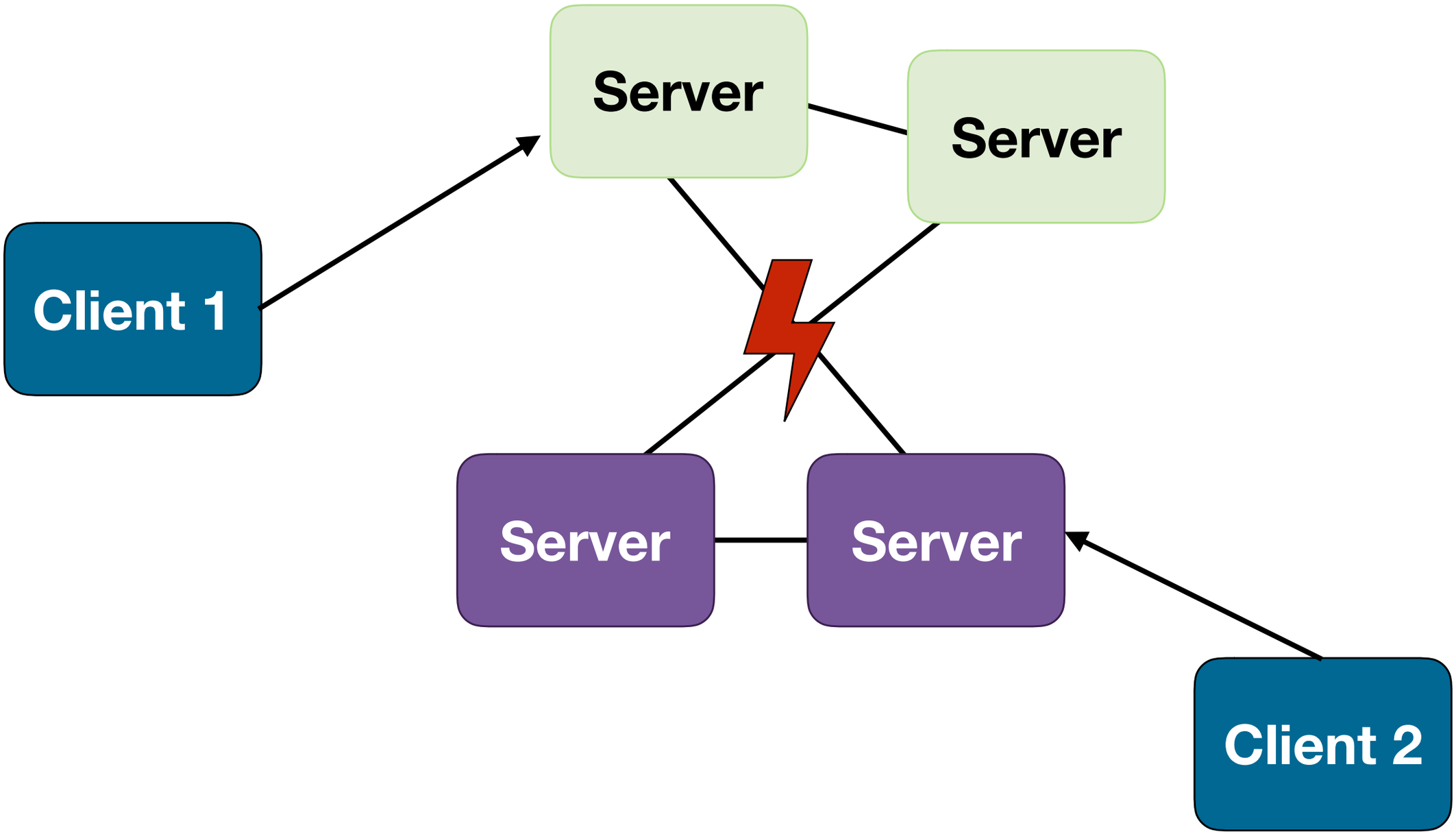
Availability: The proportion of time a system is running and able to do its job.

Fault Tolerance: Ability of a system to behave in a well defined manner once faults occur.

Distributed Availability



Distributed Availability



Scalability

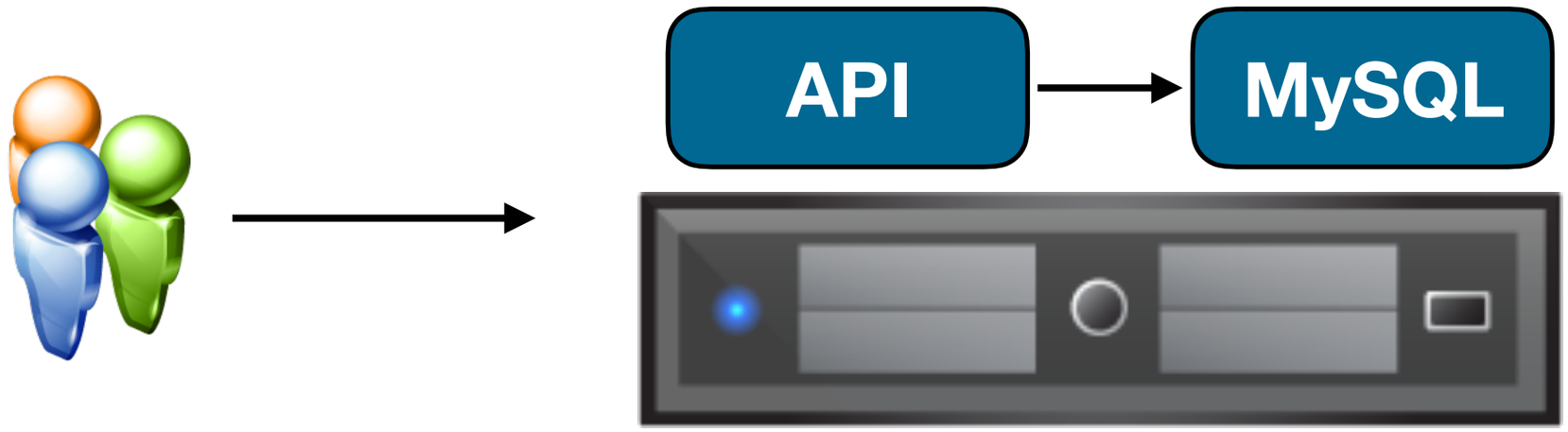
Capability of a system, network or process to handle a growing amount of work in a capable manner.

Considerations in scaling:

- **Size**: Adding more resources should make a system faster.
- **Geographic**: Adding more resources in more regions should decrease response time for users.
- **Administration**: Adding more resources should not make it harder to administer the system.

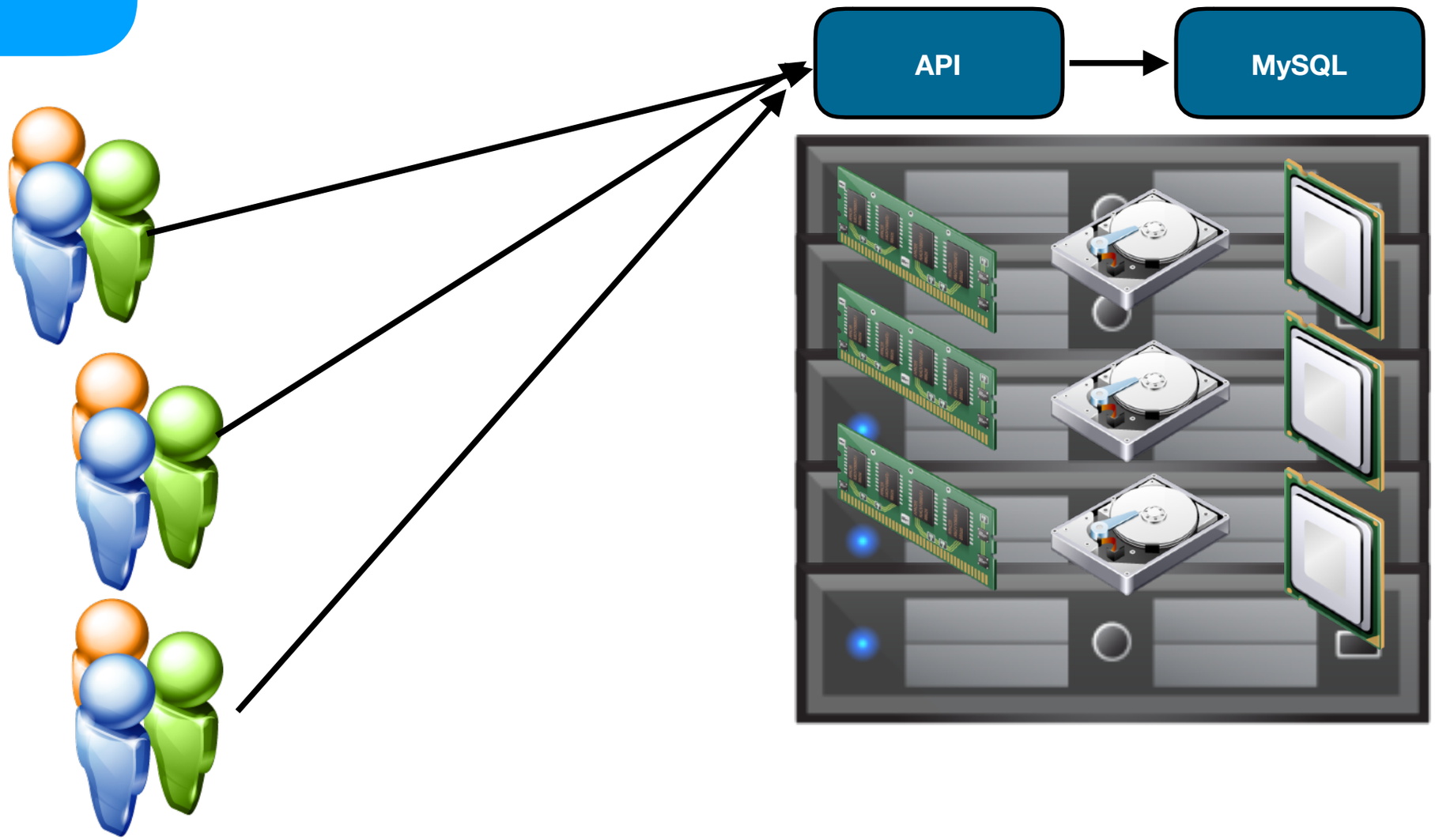
The Monolith

How can we scale as a monolith?



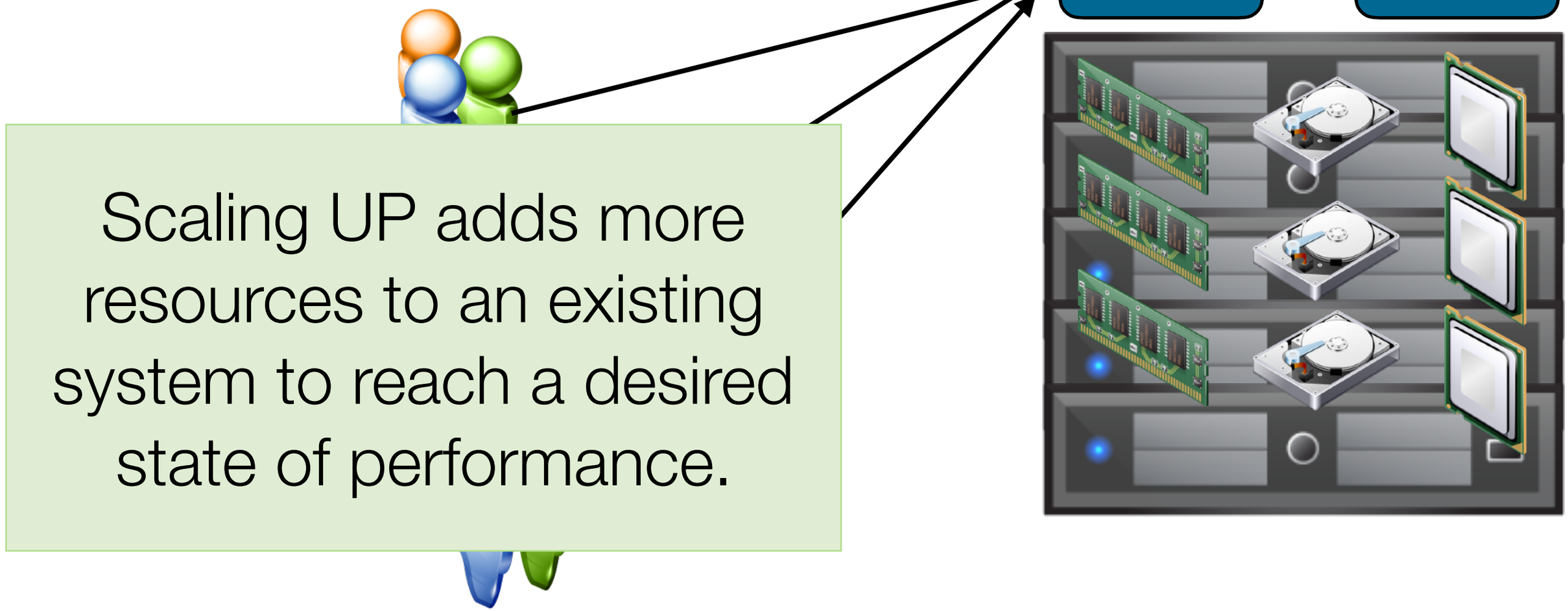
Scale Up

Let's just add more compute resources!



Scale Up

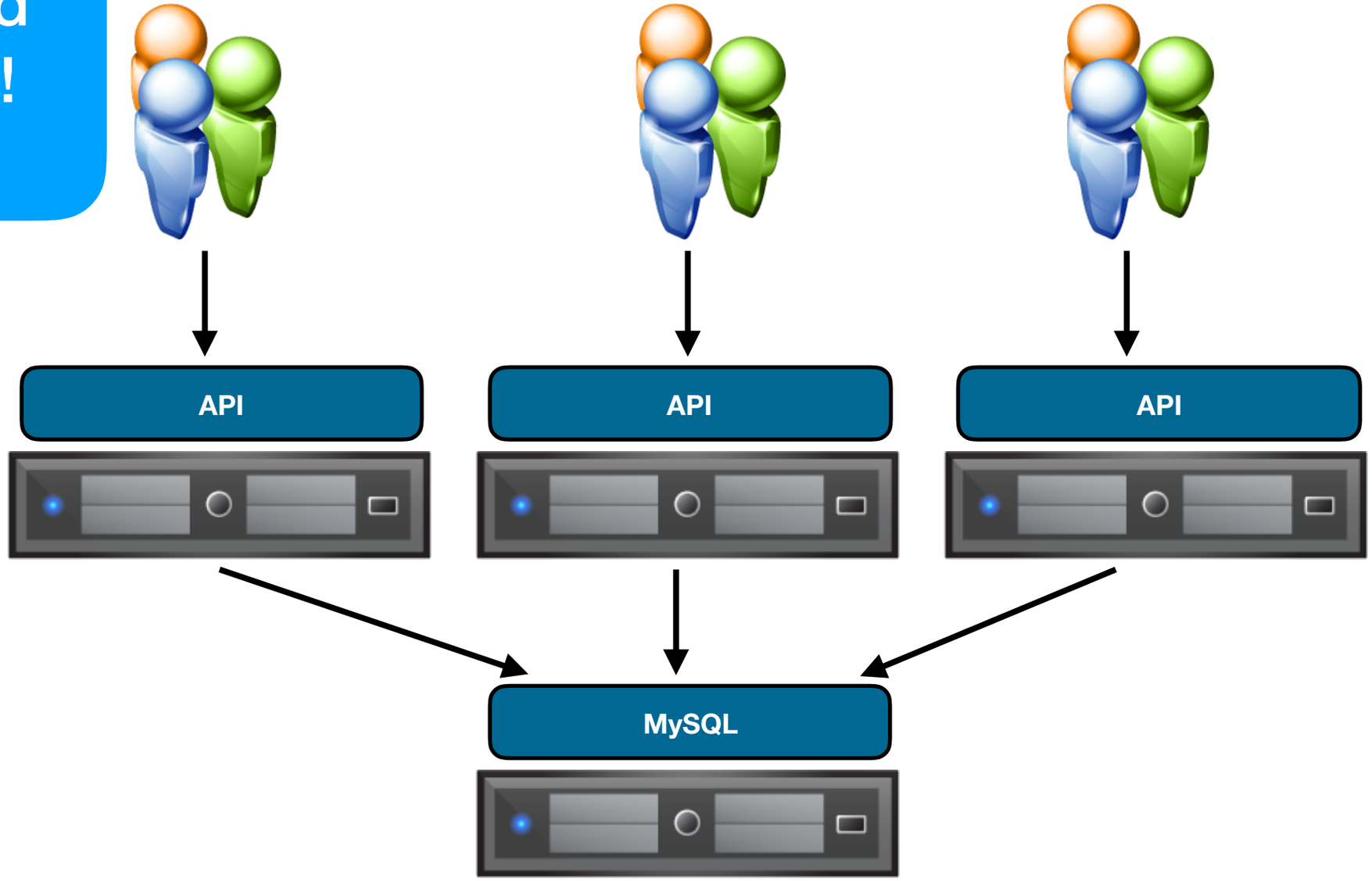
Let's just add more compute resources!



Scaling UP adds more resources to an existing system to reach a desired state of performance.

Scale Out

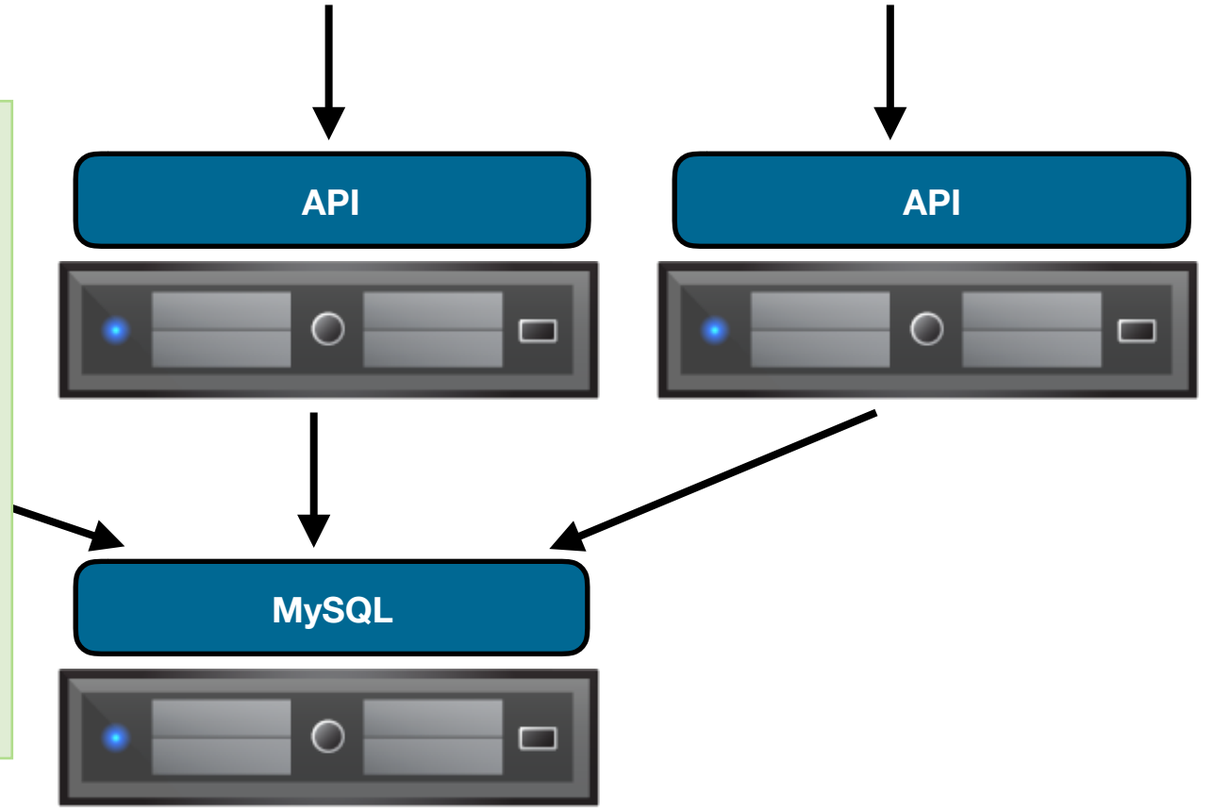
Let's just add more nodes!



Scale Out

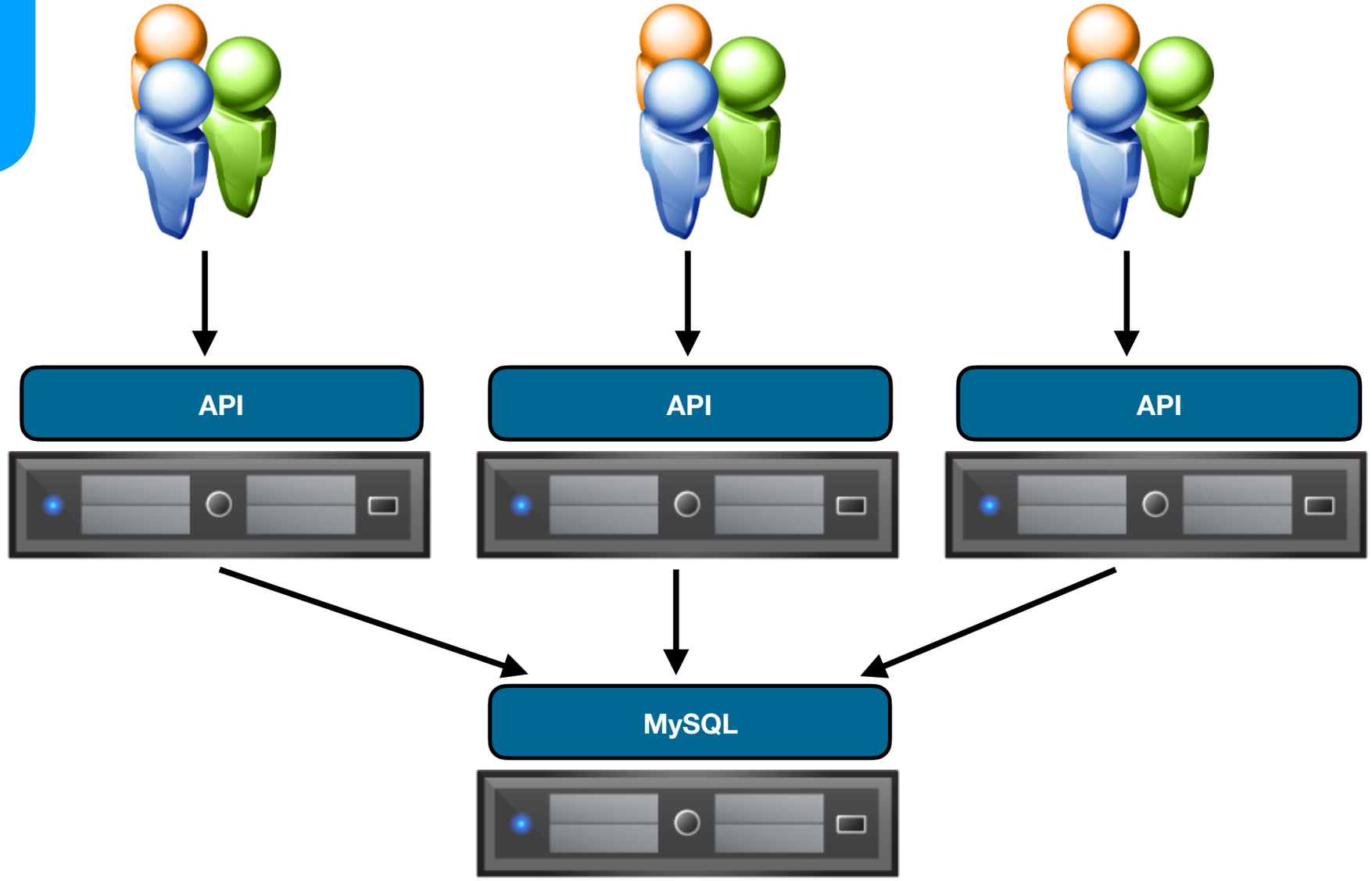
Let's just add more nodes!

Scaling OUT increases the number of processing nodes to increase processing power.



Distributed Monolith

What problems arise from this?



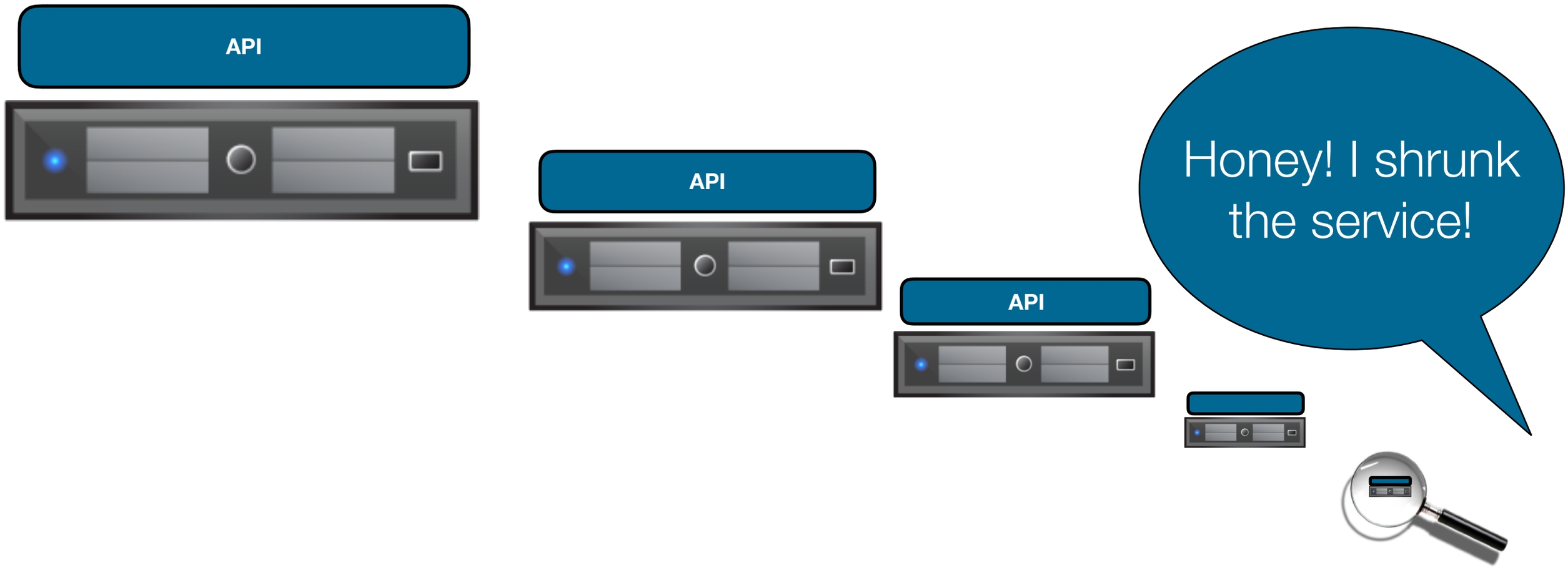
Monolithic Challenges

- **Scalability:** Need to possibly use both up and out to reach performance goals. Hard to scale things like databases.
- **Reliability:** A fault/memory leak in a single place can crash the entire app.
- **Orchestration:** You need to rebuild and deploy the entire application every time you make a change.
- **Code Complexity:** Code turns into spaghetti due to too many things happening at once. Hard to refactor features.
- **Upgradeability**: Moving to newer tech stacks requires converting the entire app at once.

There must be a better way….

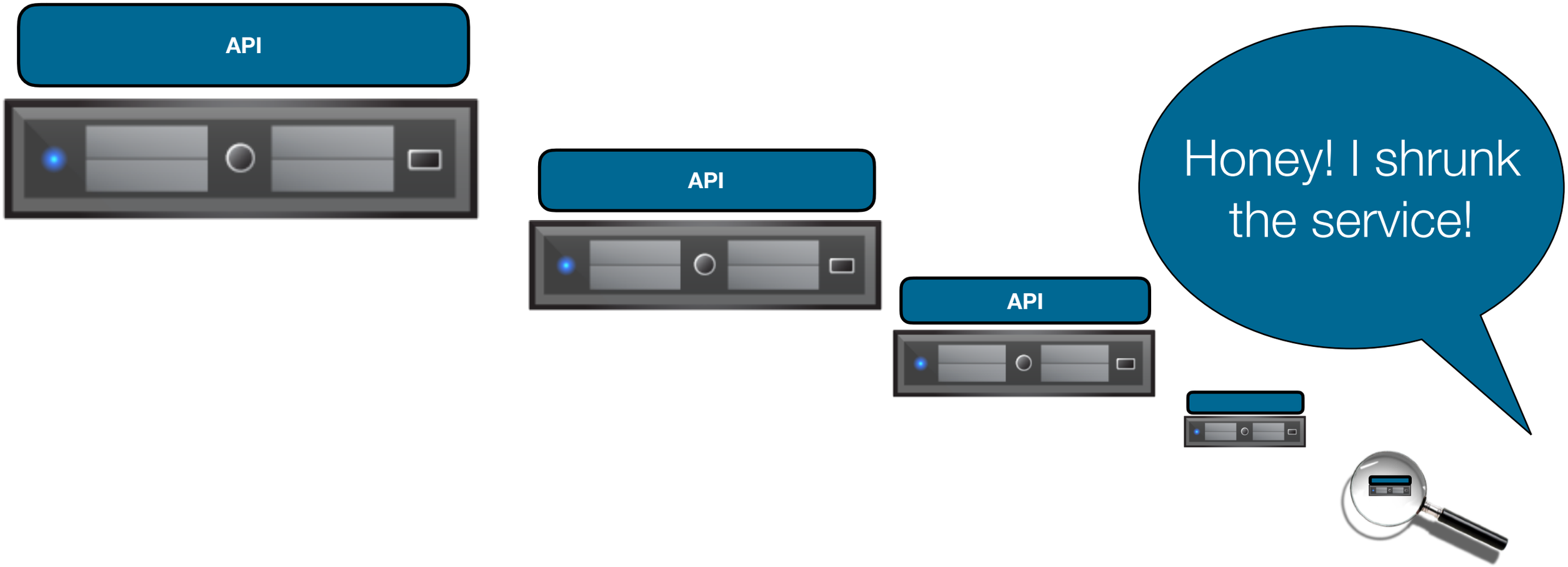
What is a Microservice?

????



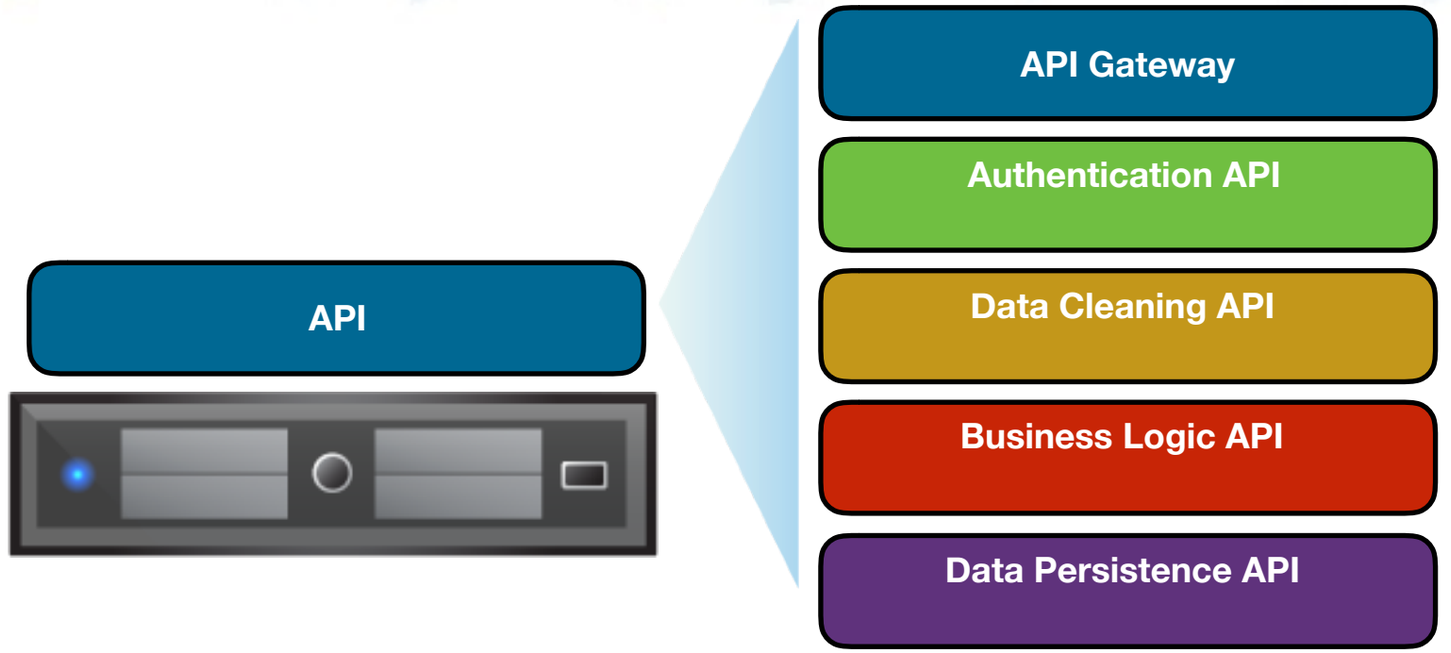
What is a Microservice?

A system built out of many different components running in different processes and communicating over well defined API's.

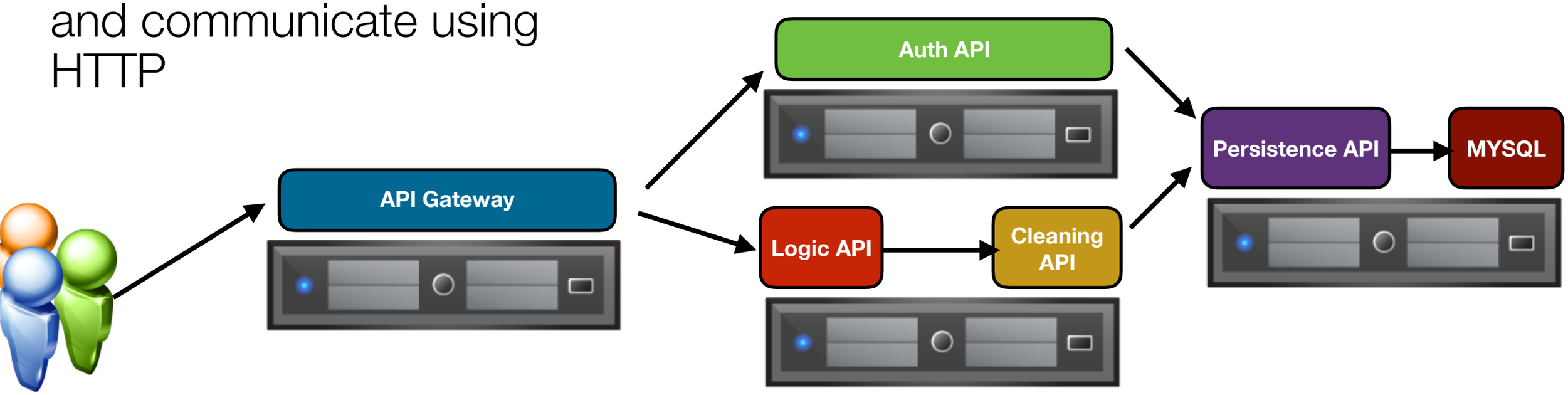


Microservices

Take your api and split it into smaller components based on function.



API's are then distributed across many machines and communicate using HTTP



Microservice Benefits?

????

Microservices Benefits?

- Source Code Complexity is lower. Each component responsible for smaller duty.

- Reliability. If a single service fails the rest of the application can keep running and even possibly recover.

- Services can be scaled independently and scaled up with tailored resources.

- Deploying changes only requires redeploying the single service. Can upgrade to new features independently.

- Development time is less. Ideal metric is every service can be redeployed in two weeks!

Challenges?

???

Microservices Challenges

Discovery: how to find a service you want?

Scalability: how to replicate services for speed?

Openness: how to agree on a message protocol?

Fault tolerance: how to handle failed services?

Data Consistency: how do you pass data around?

Coordination: how do you coordinate tasks?

The Big Picture: how do all the pieces fit together?

All distributed systems face these challenges, microservices just increases the scale and diversity…

Communication is Key

Microservices (and all distributed systems) need an easy way to communicate

Building custom protocols over raw sockets is messy

Most services are Request/Response

- Ask for a service or piece of data from server
- Send a piece of data to server

- What does this sound like?

HTTP

https://tools.ietf.org/html/rfc2616

Versatile request/response communication standard

Client sends:

Send HTTP Request - Write lines to socket

Method URL Protocol Version

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html, */*
Accept-Language: en-us
Accept-Charset: ISO-8859-1,utf-8
Connection: keep-alive
```

Header

blank line

Body (optional)

For POST and PUT method

Image Source: https://medium.com/from-the-scratch/http-server-what-do-you-need-to-know-to-build-a-simple-http-server-from-scratch-d1ef8945e4fa

HTTP

Server responds with

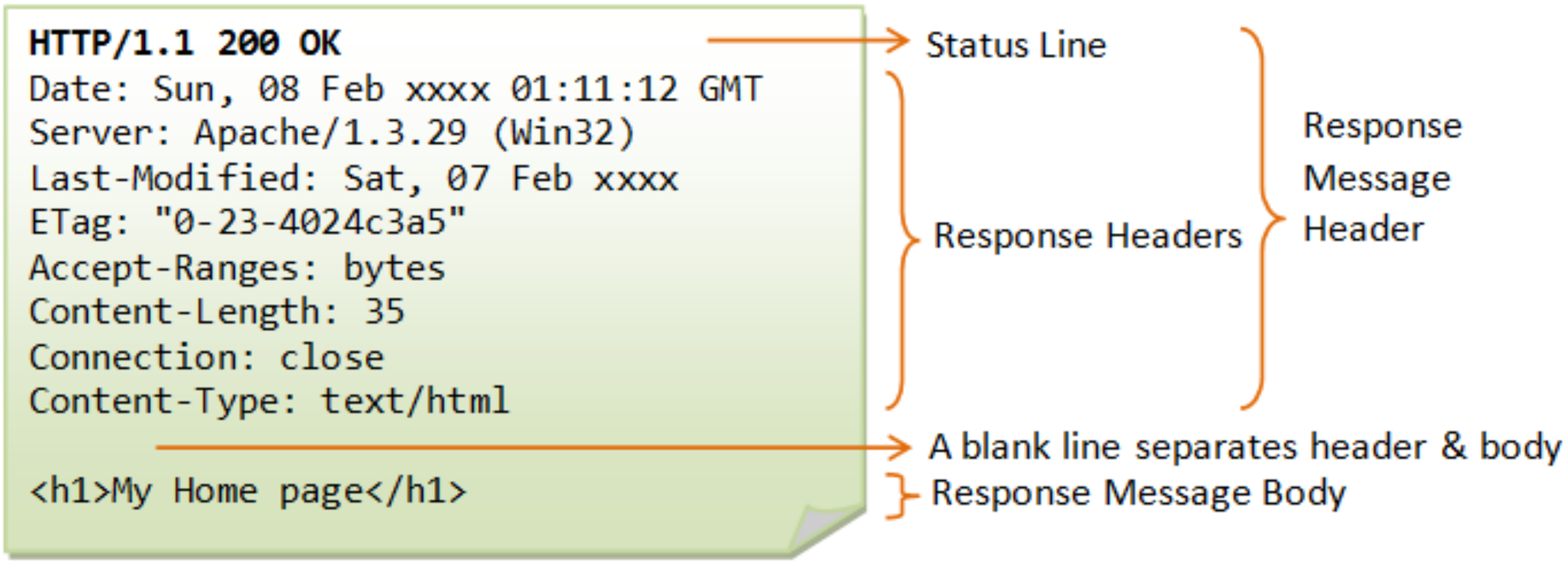


Image source: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

HTTP Request Types

Standard defines HTTP Methods

- **GET**
- HEAD
- **POST**
- **PUT**
- PATCH
- DELETE
- TRACE
- CONNECT

https://tools.ietf.org/html/rfc2616#section-9

RESTful Services

Representational **s**tate **t**ransfer (REST)

- Software architecture for web services
- Typically use HTTP as communication mechanism

Stateless

- No per-client state stored on server between requests
- Each request must contain all information needed to process it
- Application can still have state (e.g., a database), but each request should be treated independently

Let's REST!

We will build a RESTful service that interacts with another RESTful service