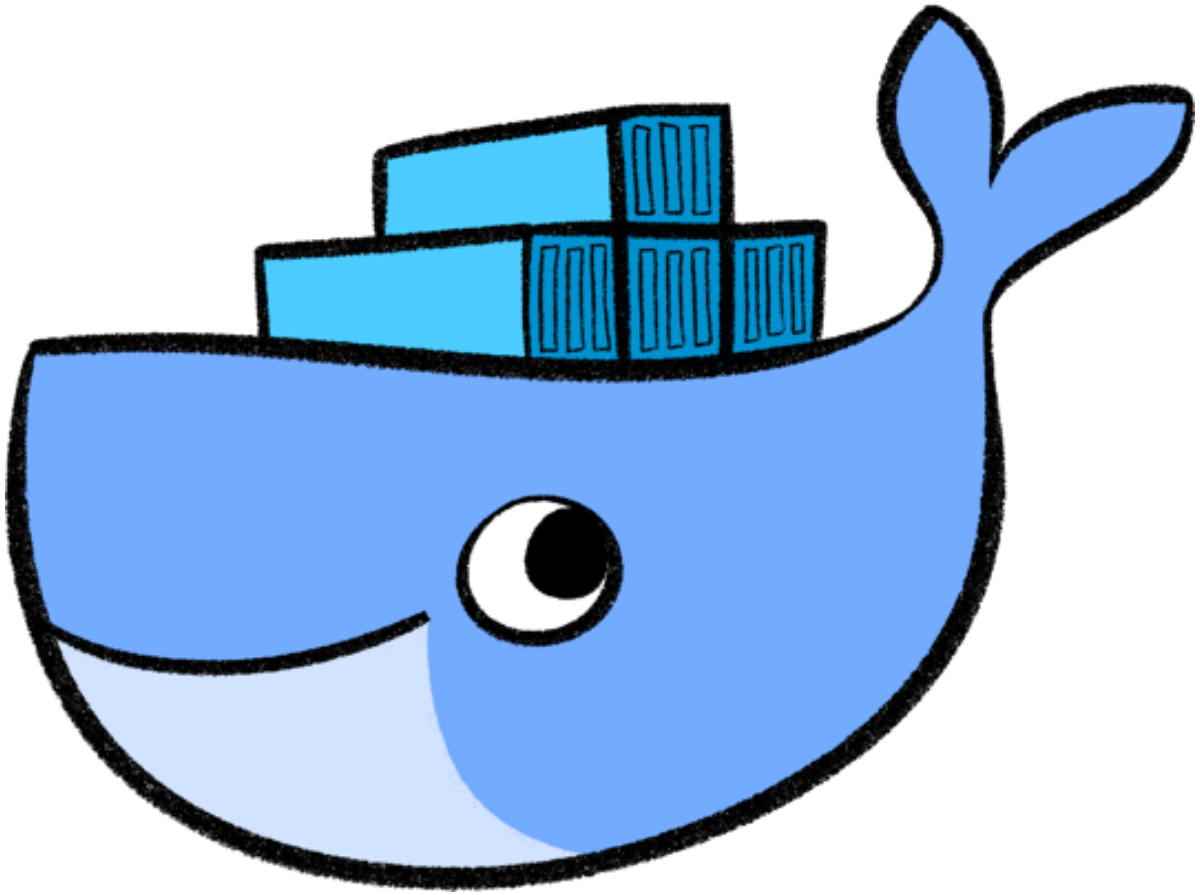
Scaling Dynamic Pages



Slides by Andrew DeOrio

Scaling road map

- Last time: scaling static pages
- Today: scaling dynamic pages
- Next time: scaling storage
  - Database
  - Media uploads

Agenda

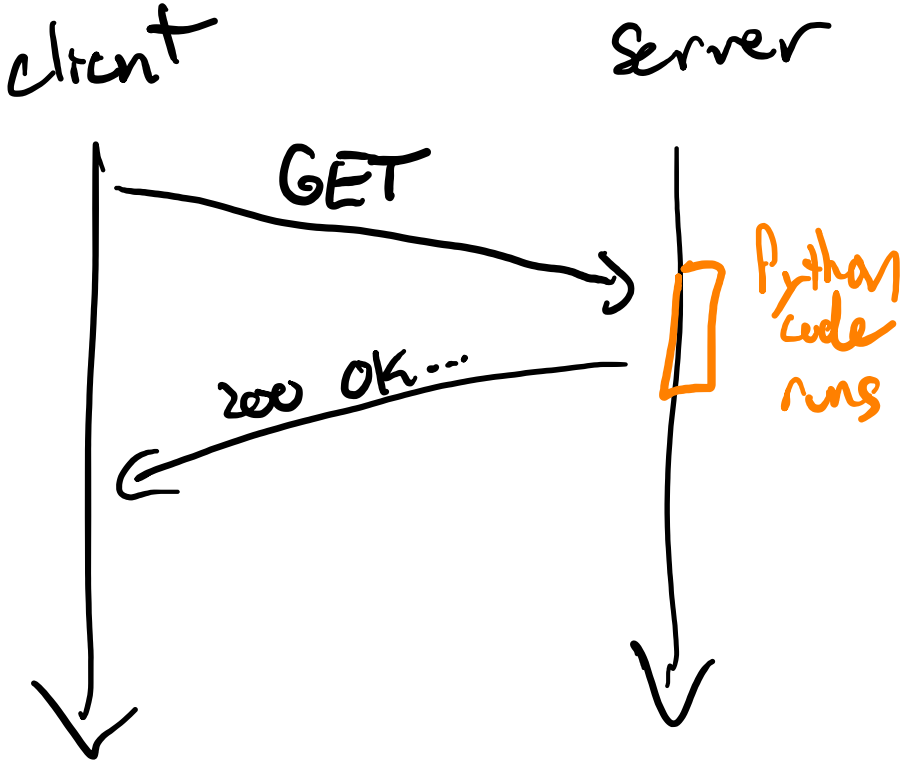
- Multi-process, multi-thread and asynchronous
- Round robin DNS and load balancing
- Hardware virtualization
- Containerization

Server-side dynamic pages

- Server-side dynamic pages: Server response is the output of a function.
- What happens when two clients make two requests at the same time?
- What about many clients?

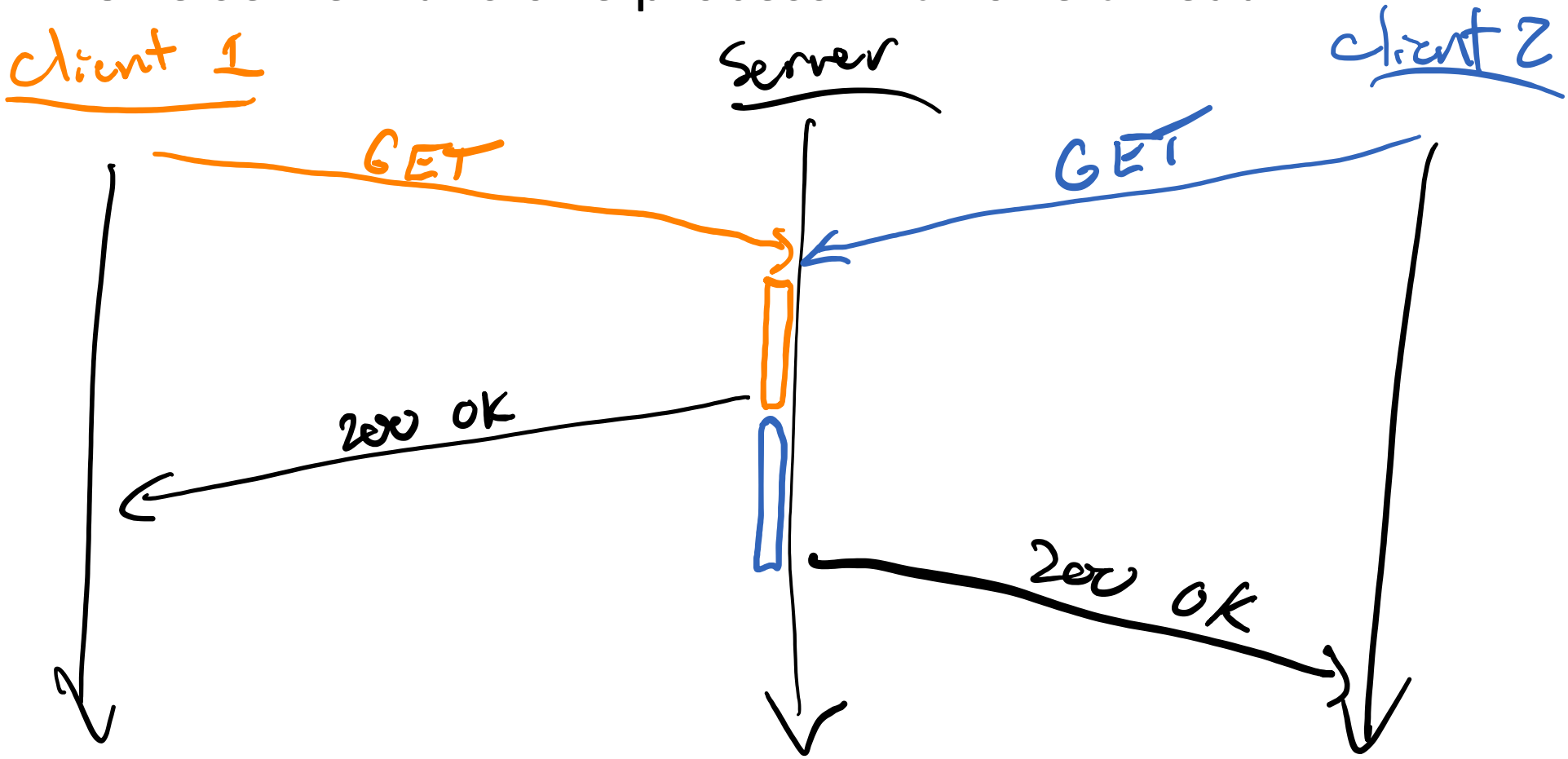
1 server, 1 process, 1 request

- One server runs one process with one thread
  - Example: Flask development server
  - One response at a time



1 server, 1 process, 2 requests

- One server runs one process with one thread

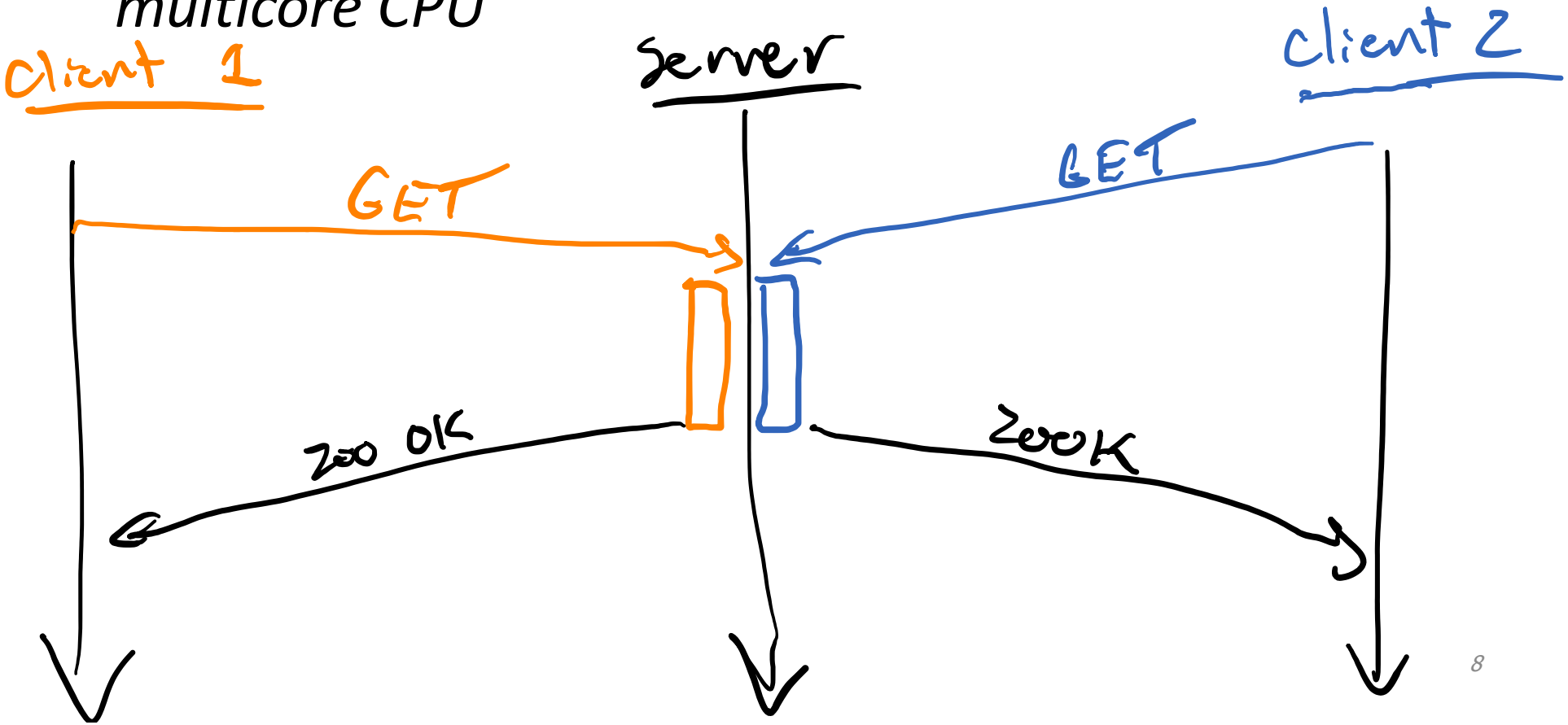


Limitations of 1 process and 1 thread

- Problem: simultaneous requests go one at a time
- Solution 1: Asynchronous programming
  - Limited solution
  - Illusion of parallelism, still runs on one CPU core
- Solution 1: Multiple threads on one CPU core
  - Limited solution
  - Illusion of parallelism, still runs on one CPU core
- Solution 3: Multiple processes on one CPU core
  - Limited solution
  - Illusion of parallelism, still runs on one CPU core

1 server, 2 processes, 2 requests

- Problem: simultaneous requests go one at a time
- Solution 4: multiple processes or threads on a *multicore CPU*

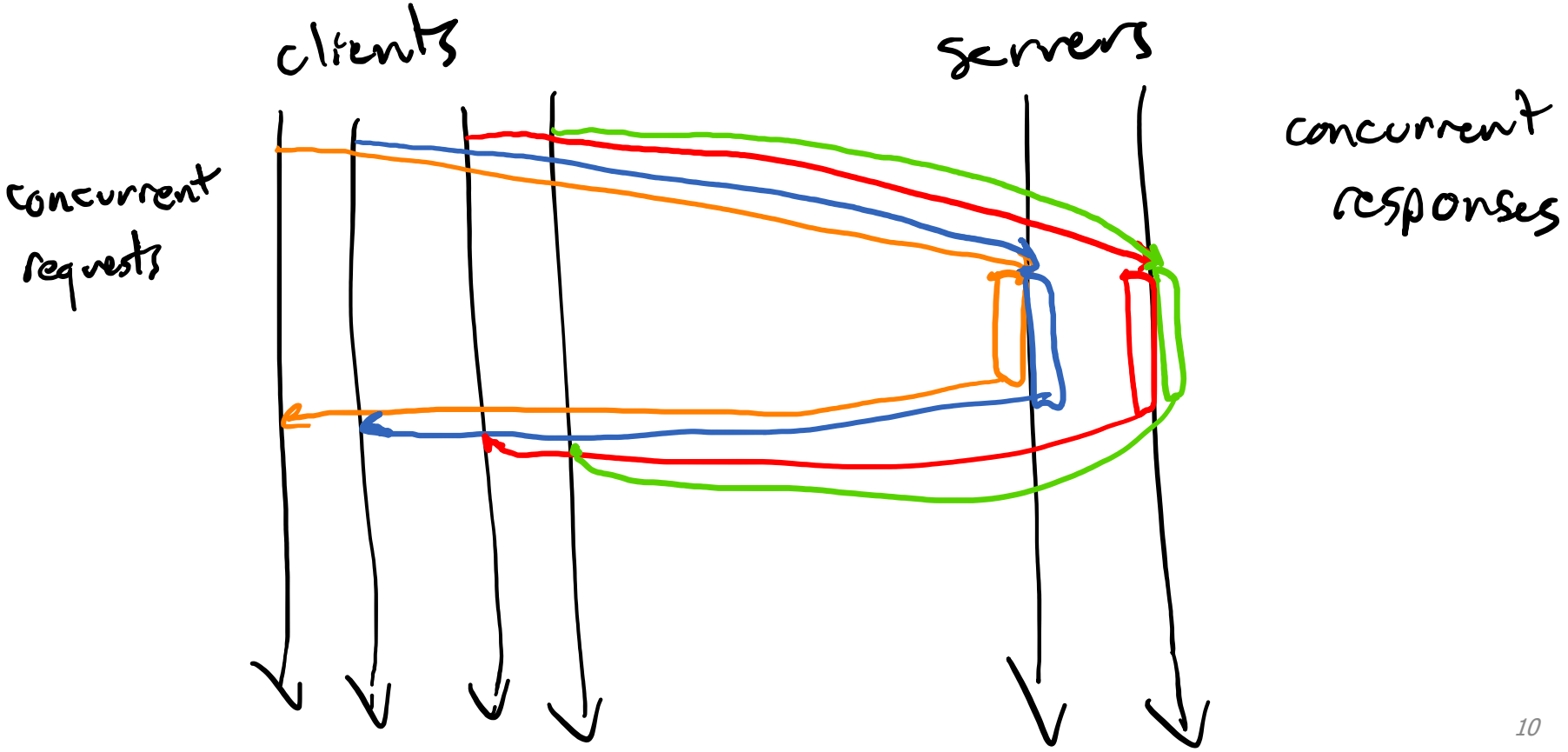


Multiprocessing with multithreading or asynchronous programming

- Can we combine multiprocessing with multithreading?
  - Yes
  - Example: Apache with mod_wsgi
- Can we combine multiprocessing with asynchronous programming?
  - Yes
  - Example: Gunicorn
- Can we combine multiprocessing with multithreading *and* asynchronous programming?
  - Yes
  - Example: Gunicorn

Many servers

- Problem: one multicore server is not enough
- Solution: buy or rent more servers



Agenda

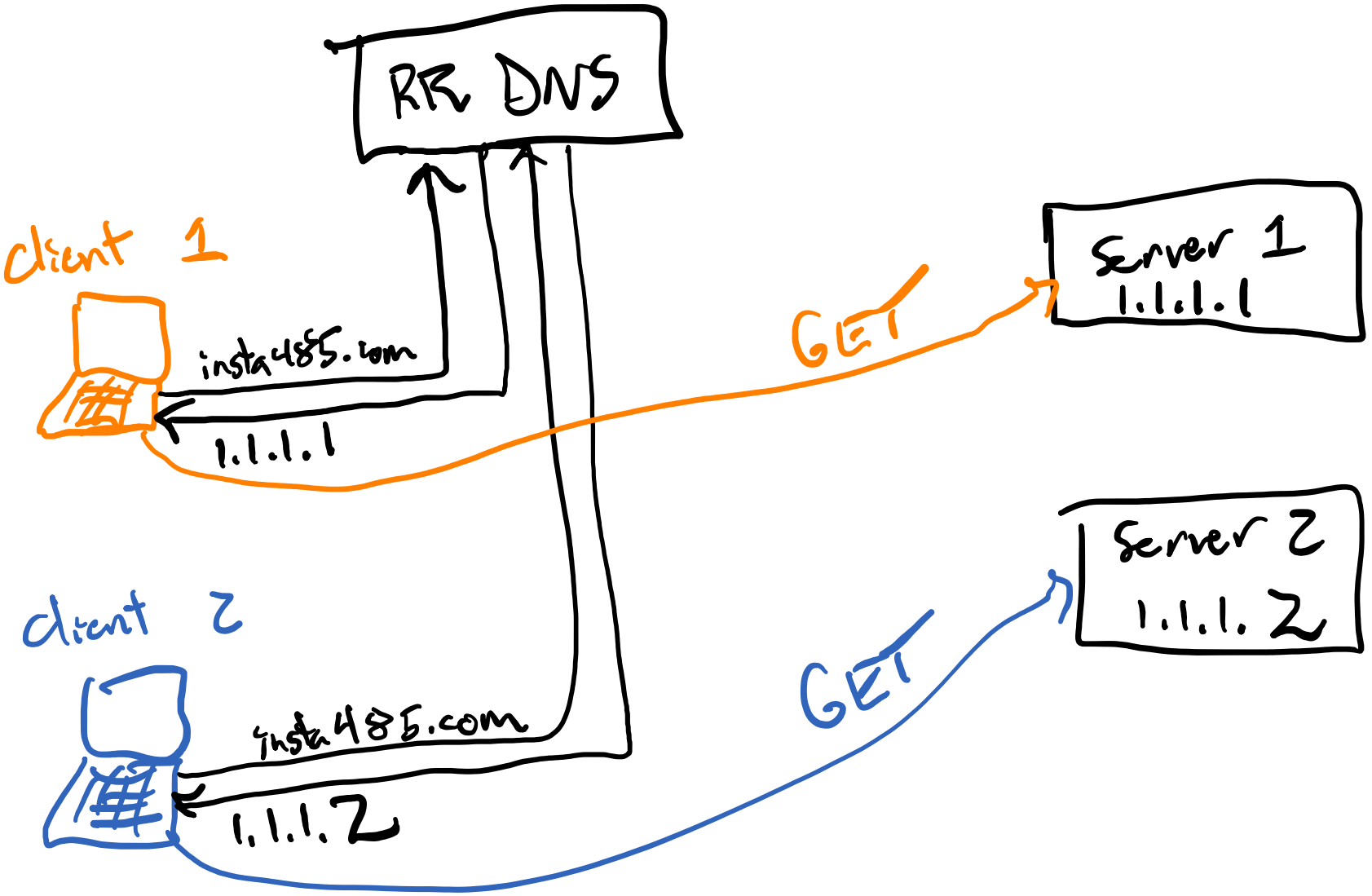
- Multi-process, multi-thread and asynchronous
- **Round robin DNS and load balancing**
- Hardware virtualization
- Containerization

Problems with many servers

- Problem: Two users make a request. Which server should respond?

- Option 1: Round robin DNS

Round Robin DNS diagram



Round Robin DNS description

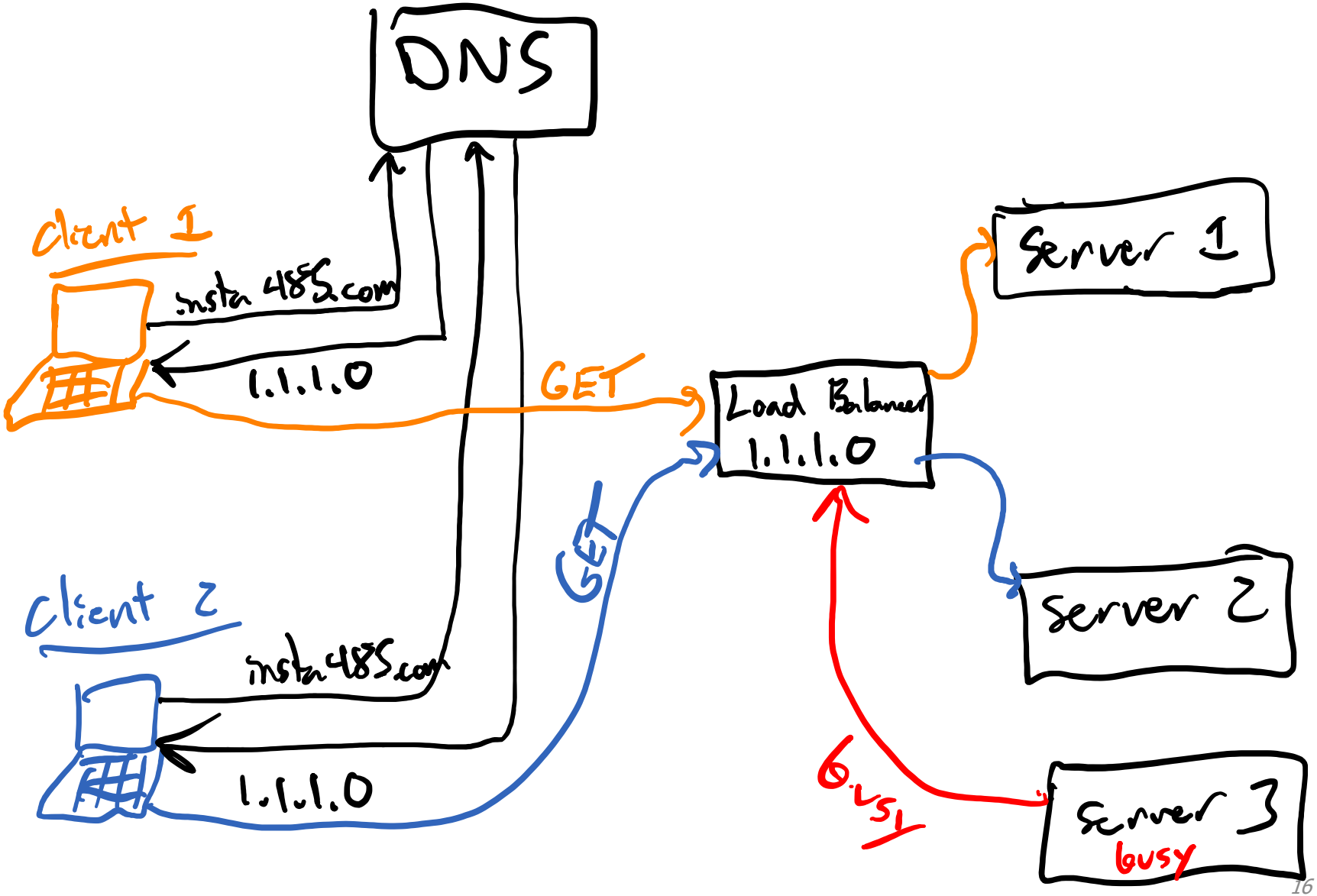
- Modified DNS server responds with different IPs to different dynamic pages web servers
- Rotating list of IPs
- Usually in the same datacenter

Problem with Round Robin DNS

- Problem: round robin DNS doesn't consider server load. One request might be more time consuming than another

- Solution: Load balancer

Load balancer diagram



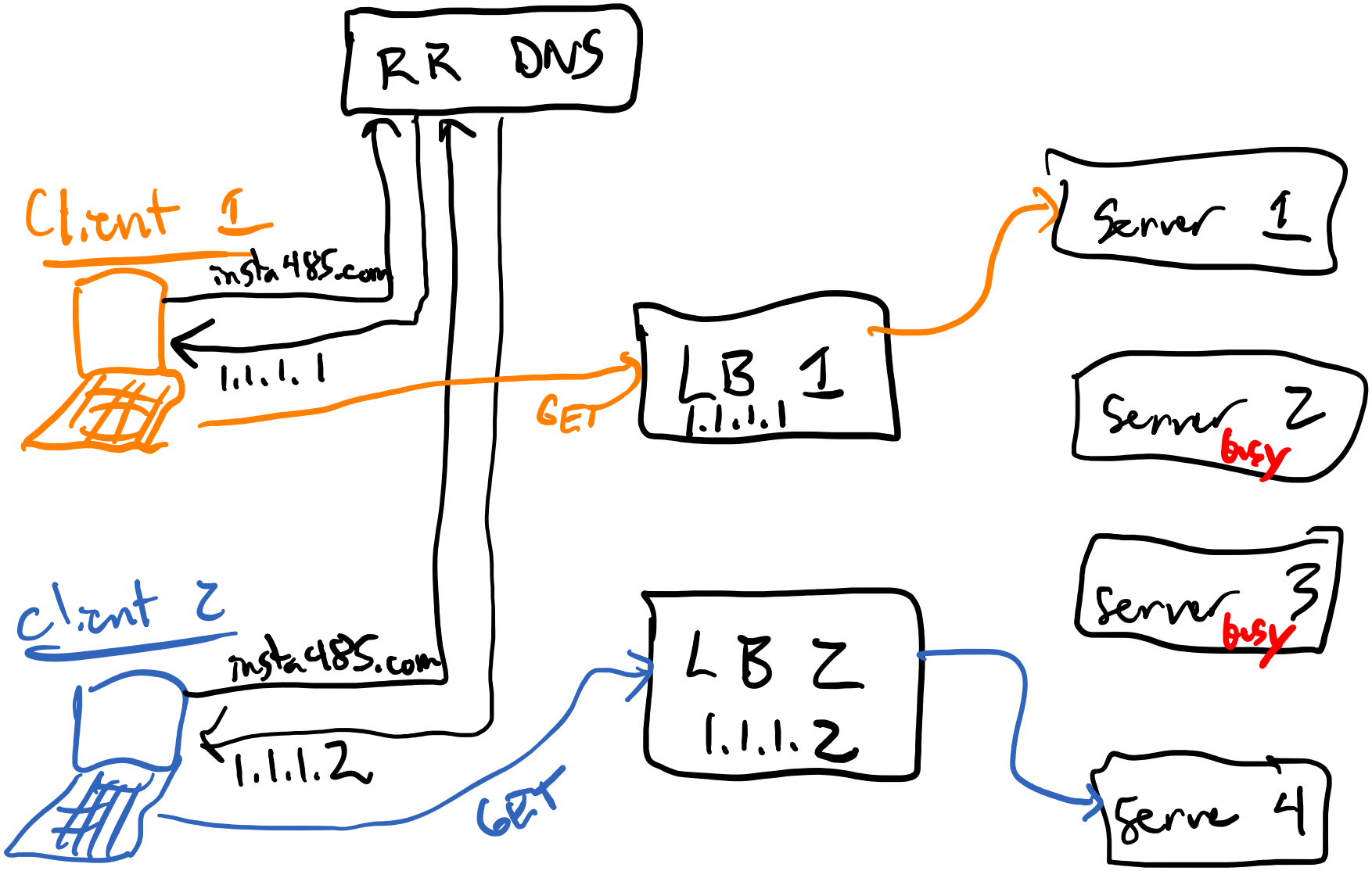
Load balancer description

- Load balancer, AKA proxy server
- "Middleman" forwards requests to backend servers
- End user doesn't need to know about them

Problem with load balancer

- Problem: Load balancer is a single point of failure
- Solution: Round robin DNS to multiple load balancers

Load balancer with Round Robin DNS



RR DNS and load balancing via PaaS

- Round robin DNS and load balancing via PaaS
  - AWS Application Load Balancer (ALB)
  - Google Cloud Load Balancing
  - Azure Load Balancer

Agenda

- Multi-process, multi-thread and asynchronous
- Round robin DNS and load balancing
- **Hardware virtualization**
- Containerization

Scaling servers

- We have a bunch of servers running our server-side dynamic pages code
  - Example: Python/Flask via Gunicorn
- Rented from an IaaS provider

Datacenter problems

- IaaS provider has many different computers
  - Some old, some new
  - Some have more CPUs, some less
  - Some have more memory, some less
- Problem 1: Energy efficiency
  - Wasteful to run the same small server-side dynamic pages program on each server. Difficult to customize for each server.

Datacenter problems

- Programs have different requirements
  - Linux kernel versions, installed libraries, etc.
  - Example: load balancer, server-side dynamic pages server, static pages server
- Problem 2: Diverse environments
  - How to customize the OS and environment for each program?

Datacenter problems

- IaaS provider has many customers
- Problem 3: Security and isolation
  - How to securely separate different customer's programs from each other?

Datacenter problems

- Sometimes many users visit our site, sometimes few
  - Variable workload
- Problem 4: Scaling
  - How to add more servers when load is high, and remove them when load is low?

Hardware virtualization

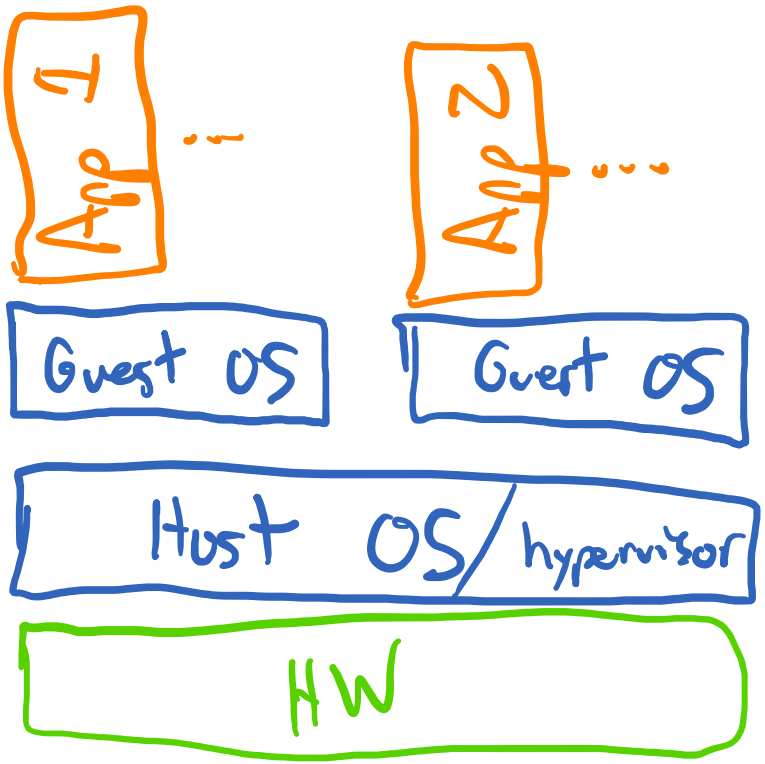
- Problem 1: Energy efficiency
- Problem 2: Diverse OS environments
- Problem 3: Security and isolation
- Problem 4: Scaling
- Solution: Hardware Virtualization

Not hardware virtualization

- Without hardware virtualization, one physical computer runs one operating system

Hardware virtualization

- With hardware virtualization, one physical computer runs multiple operating systems

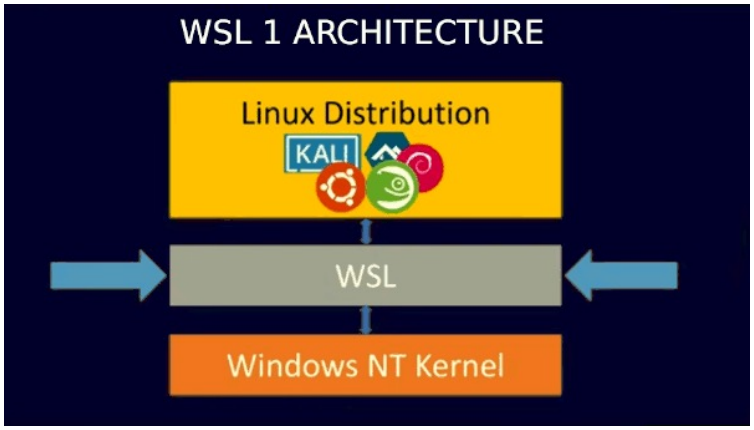


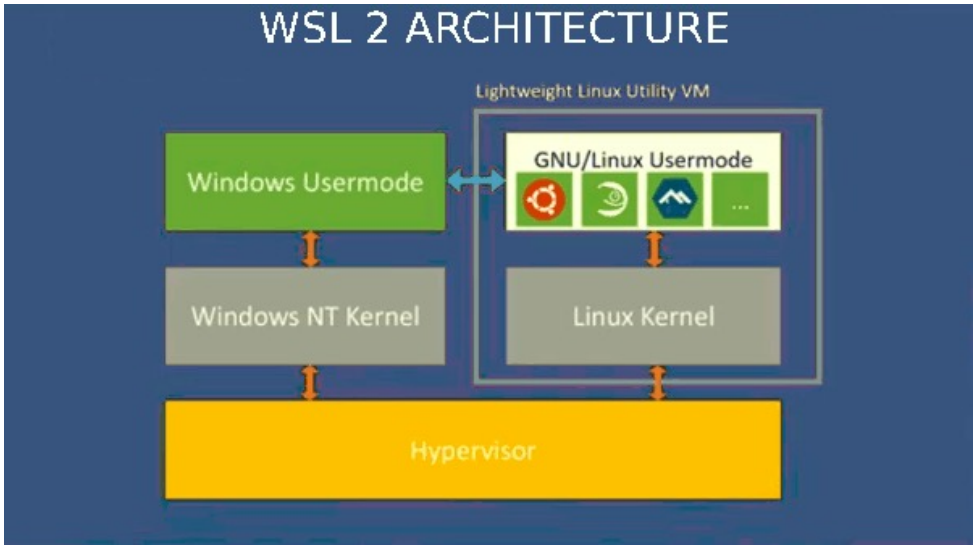
Hardware virtualization terminology

- Host: physical computer running OS and virtualization software
- Guest: operating system(s) being run as “programs”, not on dedicated physical hardware
- Hypervisor: virtualization software runs guests on host
  - VirtualBox, VMWare, et al.
- Hardware emulation
  - Host shares physical resources like network, disk, etc. with guests via virtual hardware

Tangent: WSL2

- Another use for hardware virtualization: Windows Subsystem for Linux 2 (WSL2)
- Run Linux operating system on top of windows
  - Better compatibility with Linux programs



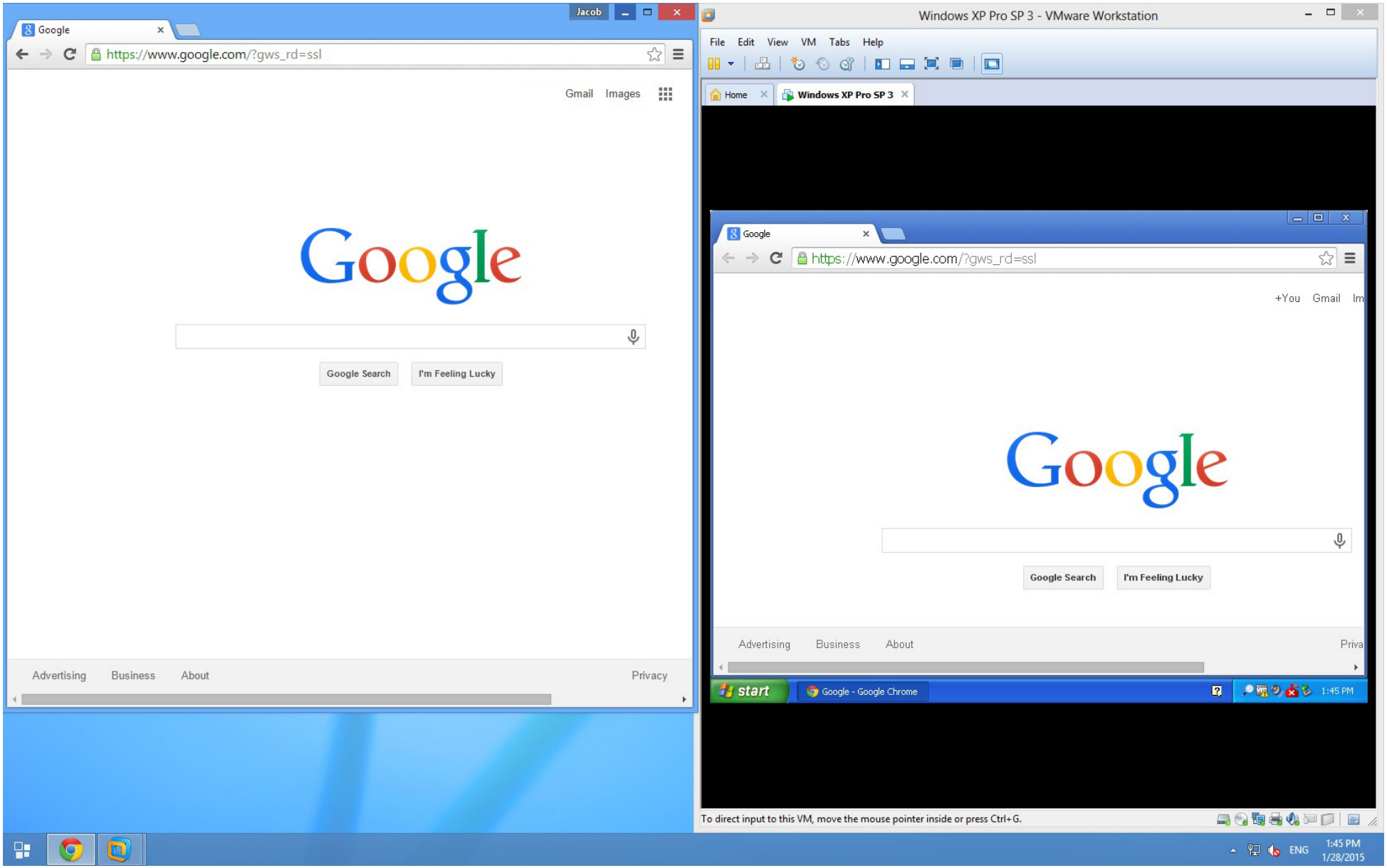


Images from https://fossbytes.com/what-is-windows-subsystem-for-linux-wsl/

Tangent: Remote proctoring software

- Remote proctoring software monitors your webcam, mic, mouse movement, how many browser windows you have open, etc.
- Easy to circumvent with a virtual machine
  - Run proctoring software inside VM
  - Take your exam inside VM
  - Do anything you want outside the VM
- But they do virtual machine detection!
  - Usually they check the name of some driver, e.g., "VirtualBox Webcam". Just change it in the Windows Registry Editor.
- Further reading: https://jakebinstein.com/blog/on-knuckle-scanners-and-cheating-how-to-bypass-proctortrack/#VirtualMachine

Tangent: Remote proctoring software



Hardware virtualization advantages

- Energy efficiency
  - Run every physical machine at maximum efficiency
- Diverse environments
  - Each service gets its own preferred OS, libraries, etc.
- Security and isolation
  - Compromised guest OS won’t affect other guest VMs
- Scaling
  - Add more VMs as more users hit your site
  - "Add another computer" is now "start a VM"

Hardware virtualization advantages

- Replication for correctness
  - Development VM has exact same guest OS, SW, libraries, packages as production VM
  - Catch bugs in development, not production
- Replication for disaster recovery
  - Easy to relocate VM
  - If there's a power failure at your East data center, replicate VMs to your Midwest data center

Hardware virtualization disadvantages

- Memory
  - Guest includes a complete OS including drivers, binaries, and libraries
- Slow start
  - 10s ~ 100s to boot a VM

Hardware virtualization IaaS

- Hardware virtualization via IaaS
- When you just rent servers, that's IaaS
- (When you rent servers with preinstalled, preconfigured software, that's PaaS)

- AWS Elastic Compute Cloud (EC2)
- Google Cloud Compute Engine
- Microsoft Azure Virtual Machines

Agenda

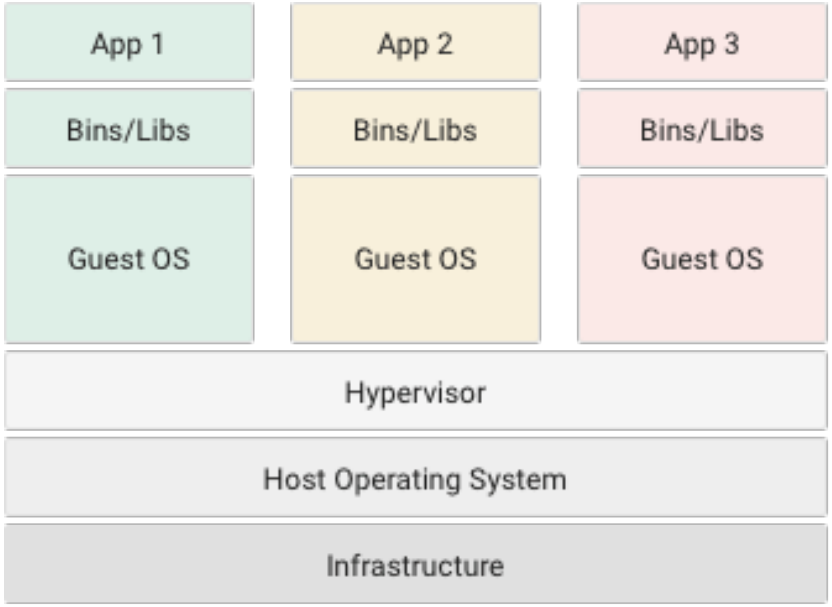
- Multi-process, multi-thread and asynchronous
- Round robin DNS and load balancing
- Hardware virtualization
- **Containerization**

Containerization

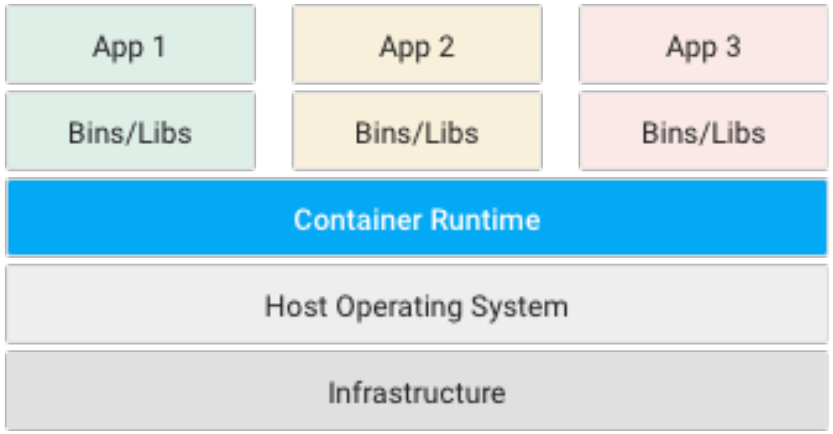
- Problem: Virtualization has high overhead
  - Memory
  - Slow start
- Solution: Containerization
- Key idea: share OS, with its binaries and libraries

Containerization

- Container includes application code and dependencies
- Container does *not* include OS



Virtual Machines



Containers

Example: Virtual machine

- Install virtualization manager, e.g., VirtualBox
- Create guest virtual machine
  - Install OS on guest VM
  - Install Python and Gunicorn on guest VM
- Copy Flask program `insta485` from host to guest
- SSH into guest
- Run program on guest

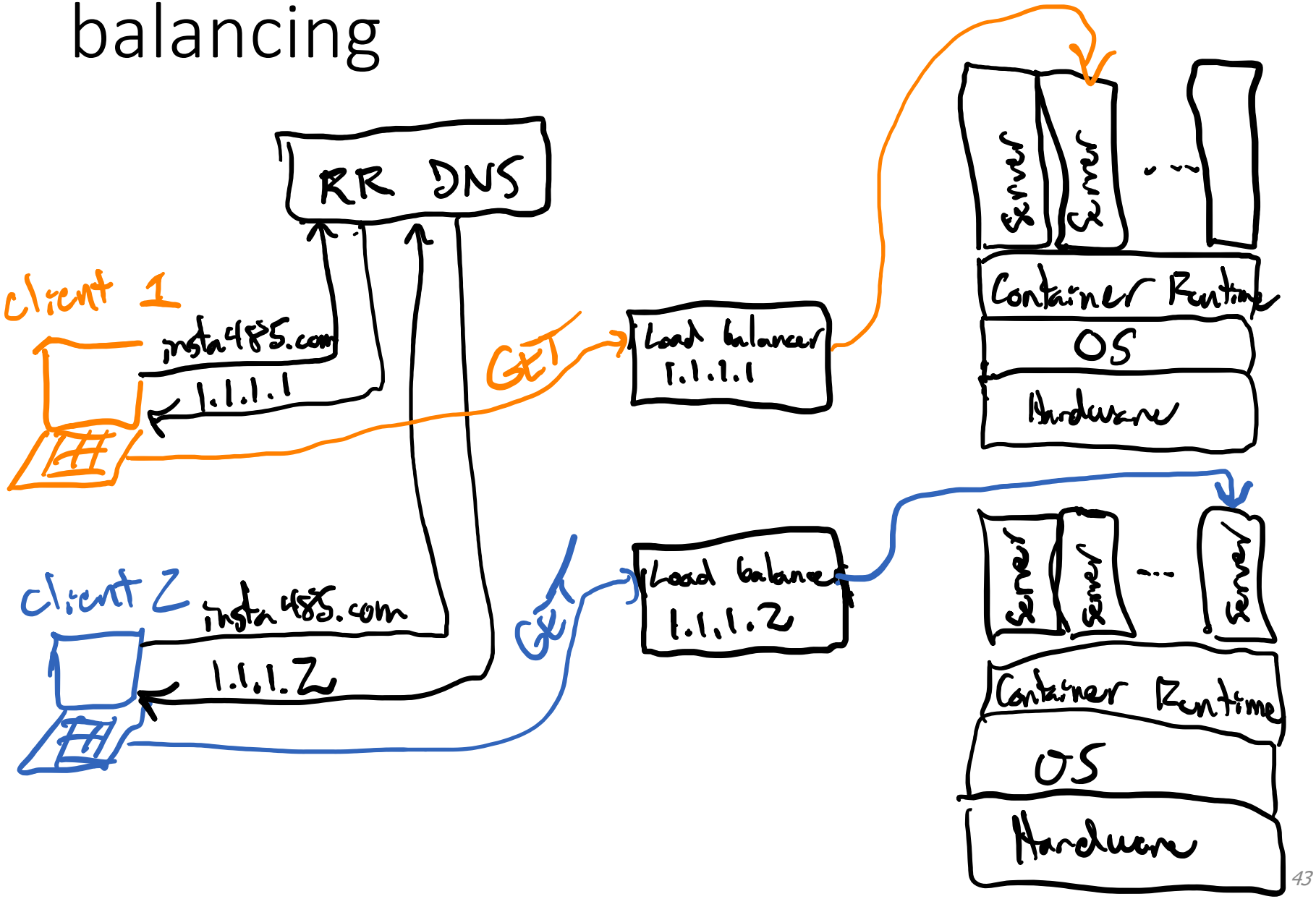
```
$ gunicorn insta485:app
```

- Stop guest VM

Example: container

- Install container runtime (Docker)
- Create Docker image
  - Copy Flask program `insta485` into container
  - Configure container to execute server
  - `$ docker build`
- Run Docker container from host
- `$ docker run insta485`

Containers, round robin DNS and load balancing



Containerization advantages

- Lower memory footprint
- Faster start-up time 0.1 ~ 1 s

Containerization disadvantages

- All containers that run on a Linux host must be designed to run on Linux
  - Can’t run Windows containers on a Linux host
- A security vulnerability in the OS will impact every container on that host machine

Virtualization vs. containerization

Virtualization

- Run multiple OSs on a single physical machine
- Abstraction of hardware
- Replication
- Isolation
- High mem overhead
- Slow startup
- Stateful

Containerization

- Run multiple apps on a single virtual or physical machine
- Abstraction of OS
- Replication
- Isolation
- Low mem overhead
- Fast startup
- Stateless

Stateful vs. stateless

- Example server-side dynamic pages app that logs to /var/log/insta485.log

```
127.0.0.1 - [06/May/2021 16:50:09] "GET / HTTP/1.1" 200 -
127.0.0.1 - [06/May/2021 16:50:09] "GET /static/js/bundle.js HTTP/1.1" 200 -
127.0.0.1 - [06/May/2021 16:50:09] "GET /api/v1/posts/ HTTP/1.1" 200 -
```

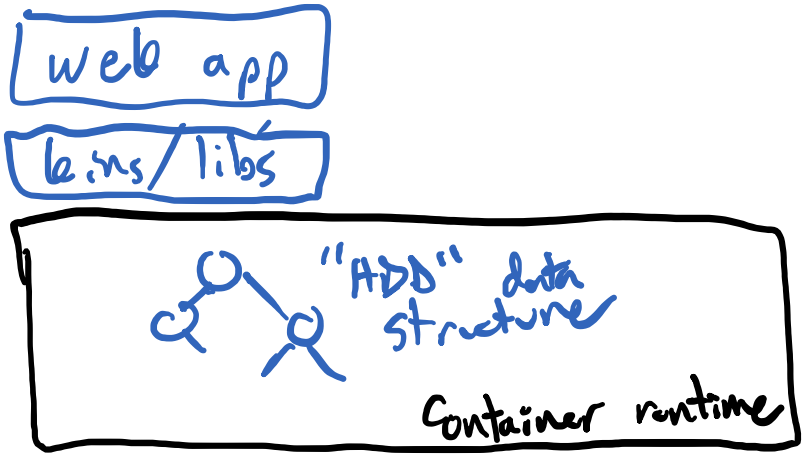
- Run it, make a few requests, restart
- What happens to the log in a virtual machine?
- What happens to the log in a container?

Stateful virtual machine example

- VMs have a virtual, persistent hard drive
  - Hard drive is a file on the host computer
  - VM writes to the HDD -> file modified
- Restart VM, file is restored, including changes

Stateless container example

- Container software layer simulates hard disk drive
  - Hard drive is a data structure in the container runtime
  - Container writes to the HDD -> data structure modified
- Restart container, *changes are lost*
  - This is a feature! Separation of storage and compute.

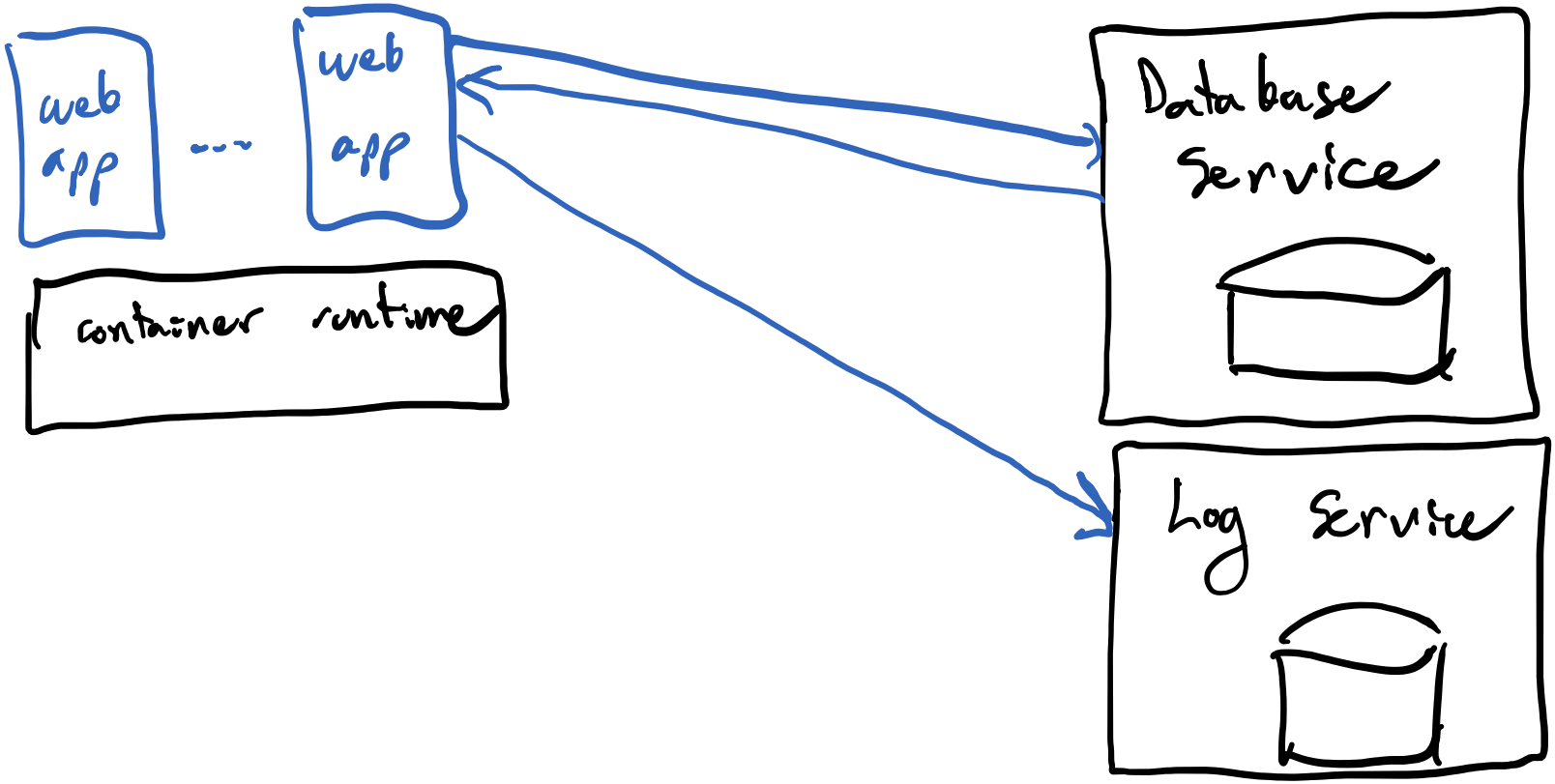


Service-oriented architecture (SOA)

- Service-oriented architecture (SOA)
- **Not SOA:** run entire web app on one machine
- **SOA:** Break up web app into services that communicate over a network
- Run different services on different machines
  - Static pages servers
  - Server-side dynamic pages servers
  - Database servers
  - Log storage servers
  - Etc.

Microservice architecture

- Microservice architecture is a service-oriented architecture with "smaller pieces"
- Common with containerization



Container execution via PaaS

- Rent servers running container runtime
  - E.g., Docker
- AWS FarGate
- Google Cloud Run
- Microsoft Azure Container Instances
- Sometimes called *serverless computing*

Check your understanding

- We talked about scaling server-side dynamic pages today
- How do we scale a client-side dynamic pages application?

Check your understanding

- How do we scale a client-side dynamic pages application?
  - Scale the JavaScript distribution with same techniques as static pages
    - bundle.js via CDN
  - Scale the REST API using same techniques as server-side dynamic pages
    - REST API server code in a container on PaaS

Scaling road map

- Last time: scaling static pages
- Today: scaling dynamic pages
- Next time: scaling storage
  - Database
  - Media uploads