CS 4530 Software Engineering

Module 14: Principles and Patterns of Cloud Infrastructure

Adeel Bhutta, Joydeep Mitra and Mitch Wand

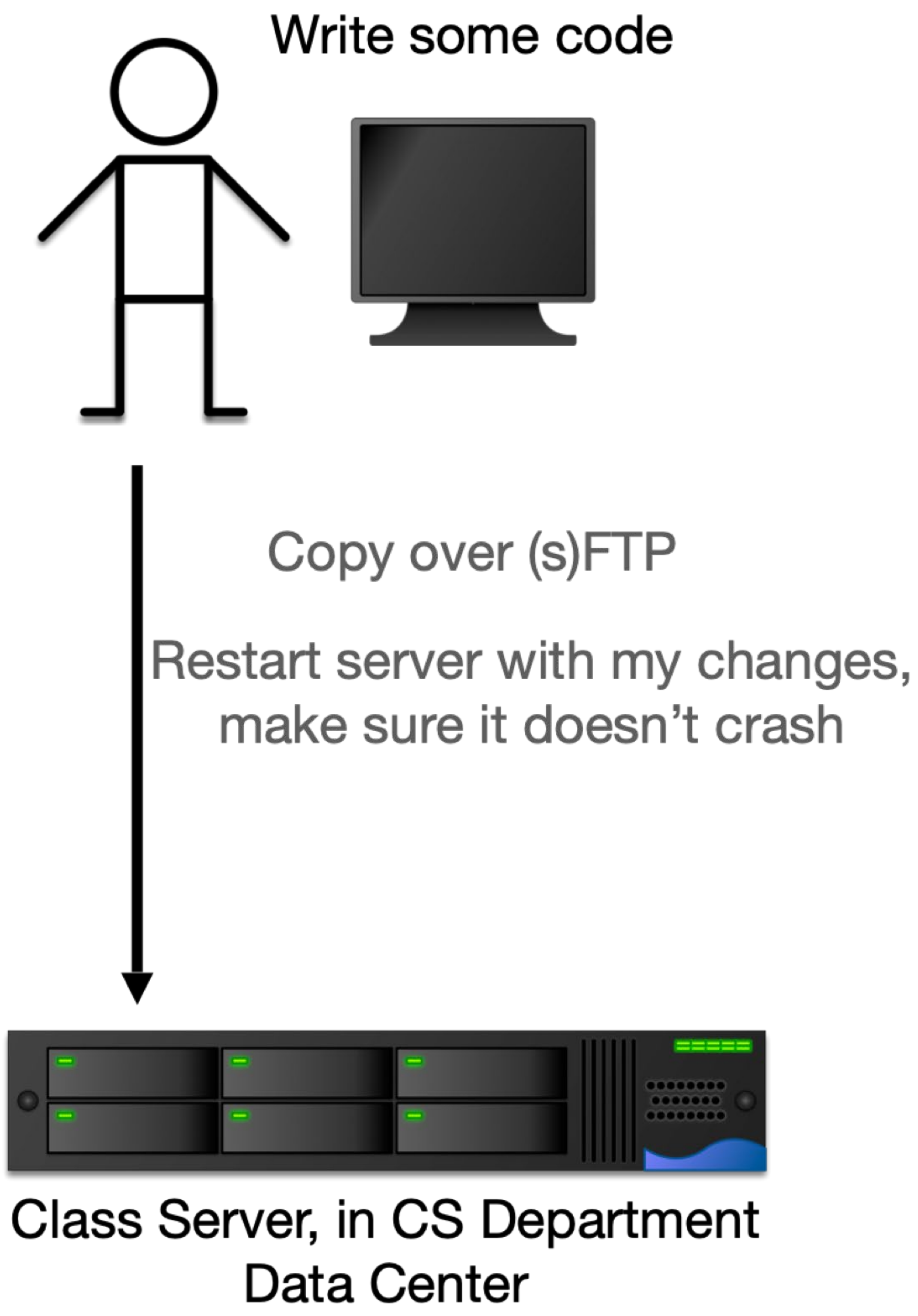
Khoury College of Computer Sciences

Learning objectives for this lesson

- By the end of this lesson, you should be able to…
  - Explain what “cloud” computing is and why it is important
  - Explain why shared infrastructure is important in cloud computing
  - Describe the difference between virtual machines and containers
  - Discuss trade-offs that you might consider for self or vendor-managed platforms

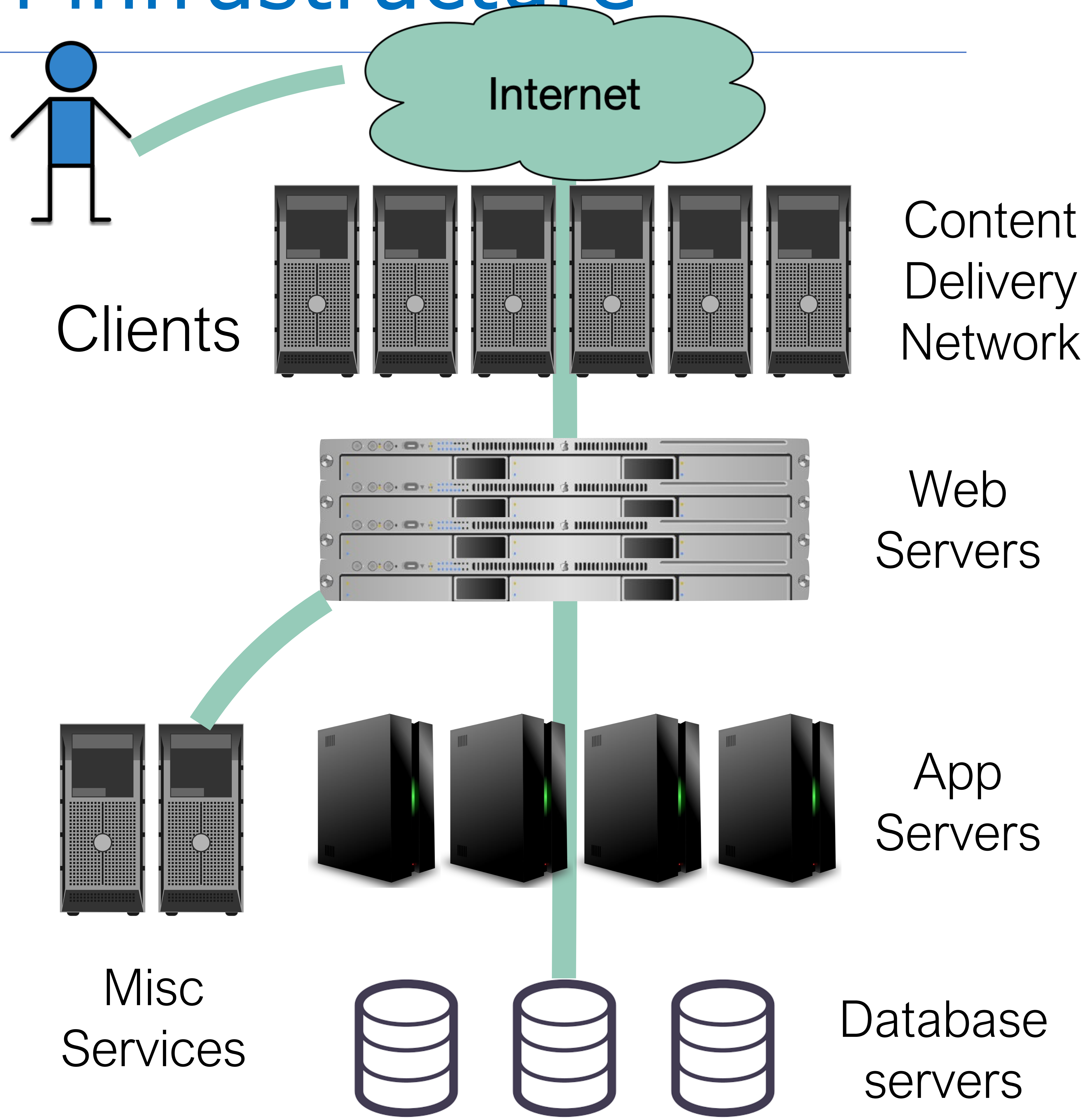
How to deploy web apps?

- What we need:
  - A server that can run our application
  - A network that is configured to route requests from an address to that server
- Questions to think about:
  - What software do we need to run besides our application code? (Databases, caches, etc?)
  - Where does this server come from? (Buy/Borrow?)
  - Who else gets to use this server? (Multi-tenancy or exclusive?)
  - Who maintains the server and software? (Updates OS, libraries, etc?)



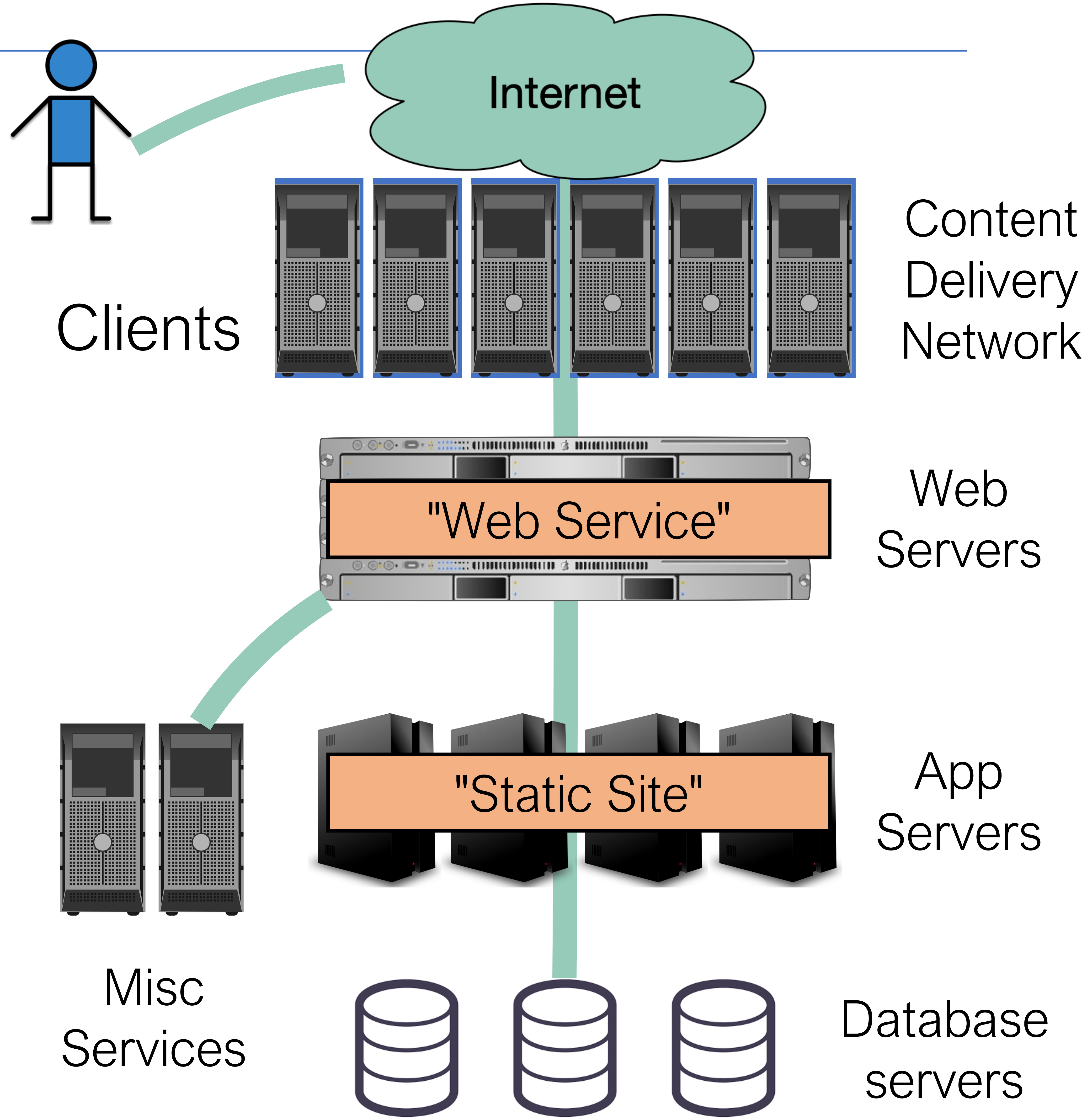
Many apps rely on common infrastructure

- Content delivery network: caches static content "at the edge" (e.g. cloudflare, Akamai)
- Web servers: Speak HTTP, serve static content (eg REACT), load balance between app servers (e.g. haproxy, traefik)
- App servers: Run our backend application (e.g. nodejs)
- Misc services: Logging, monitoring, firewall
- Database servers: Persistent data

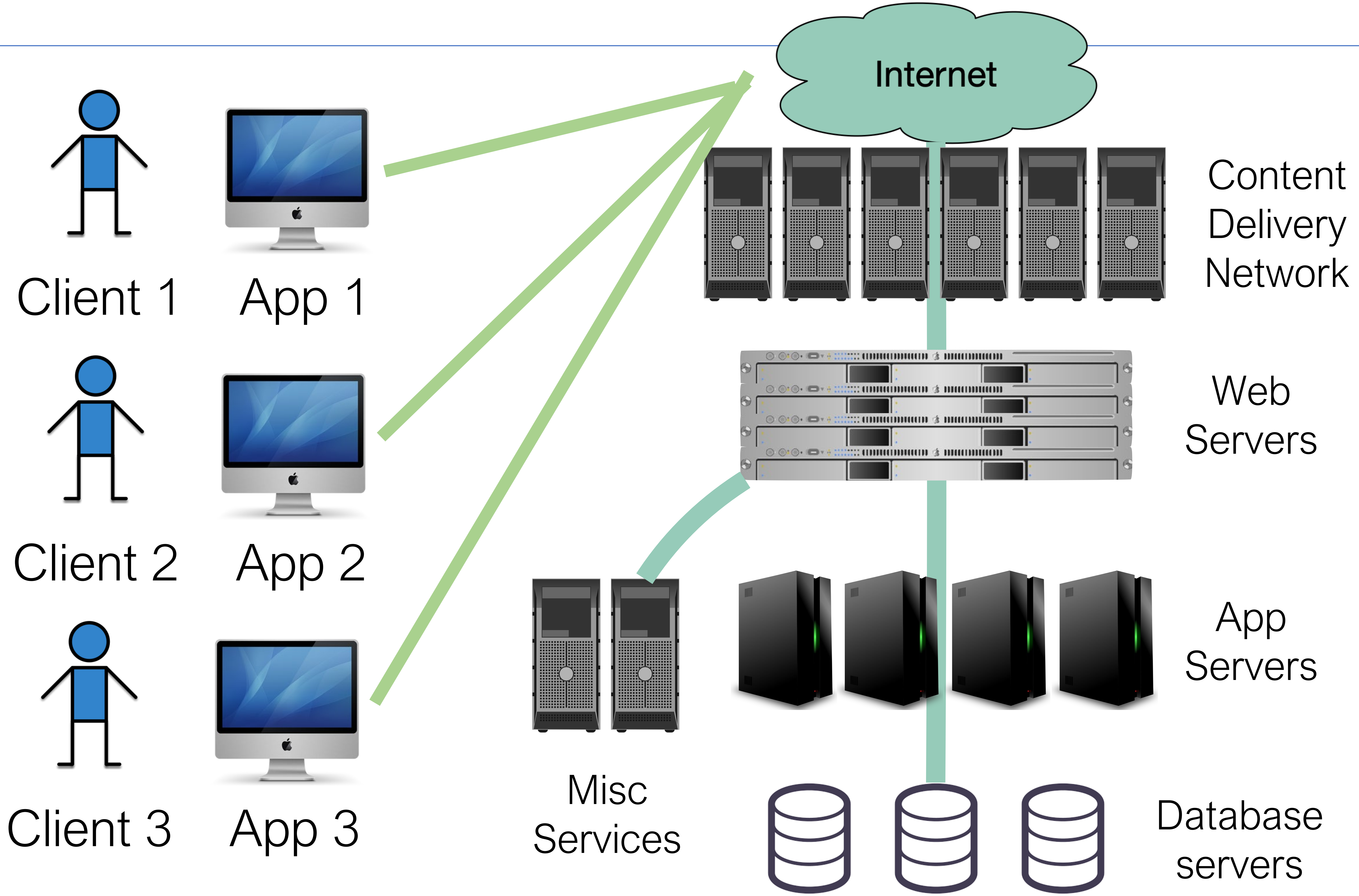


Render.com terminology

- Content delivery network: caches static content “at the edge” (e.g. cloudflare, Akamai)
- Web servers: Speak HTTP, serve static content (eg REACT), load balance between app servers (e.g. haproxy, traefik)
- App servers: Run our backend application (e.g. nodejs)
- Misc services: Logging, monitoring, firewall
- Database servers: Persistent data

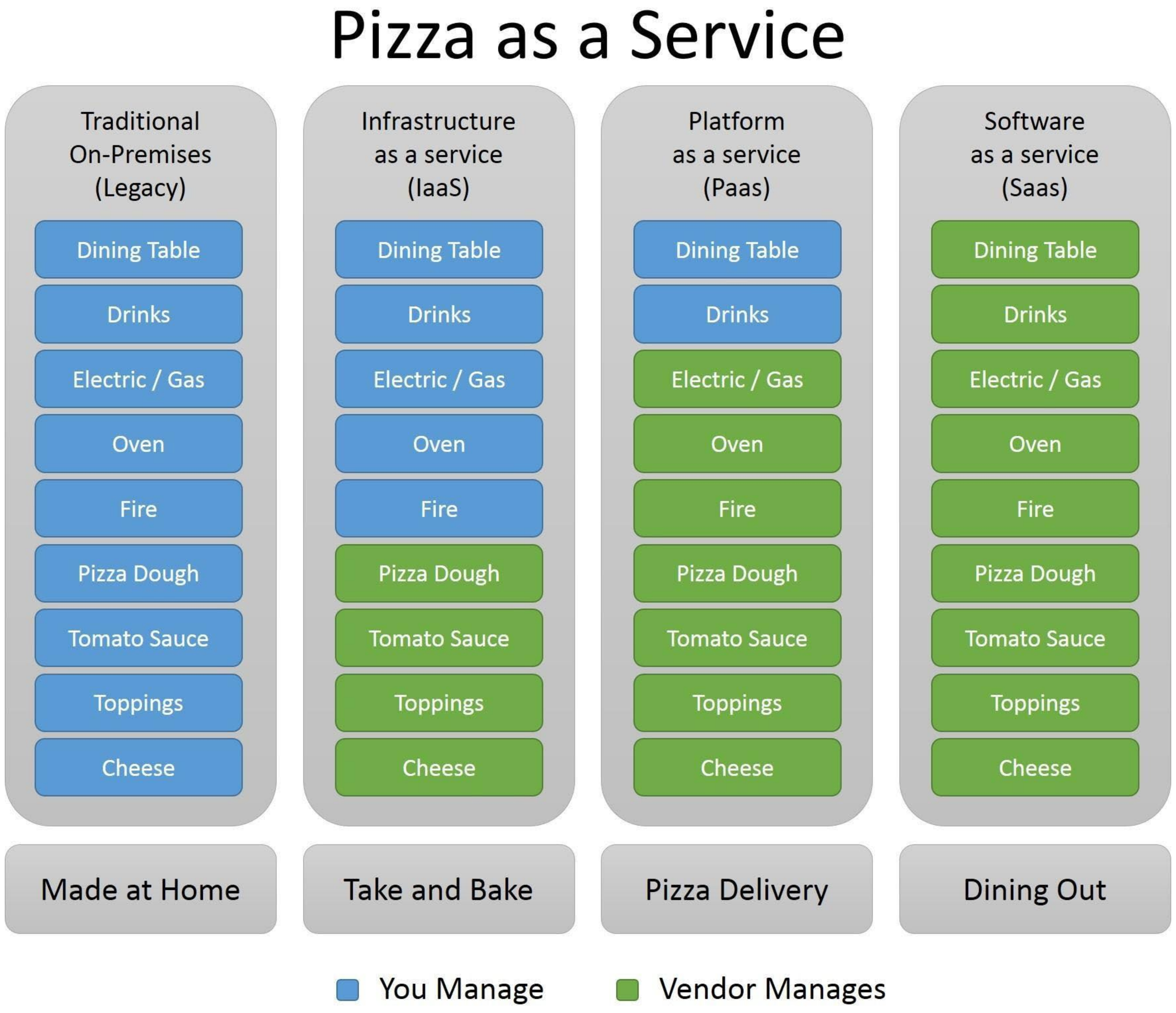


What parts of this infrastructure can be shared across different clients?



Shared infrastructure analogy: Pizza

- Four ways to get pizza: Make yourself, take and bake, delivery, dine out
- Vendor manages different levels of the stack, achieving economies of scale
- When would you choose one over the other?



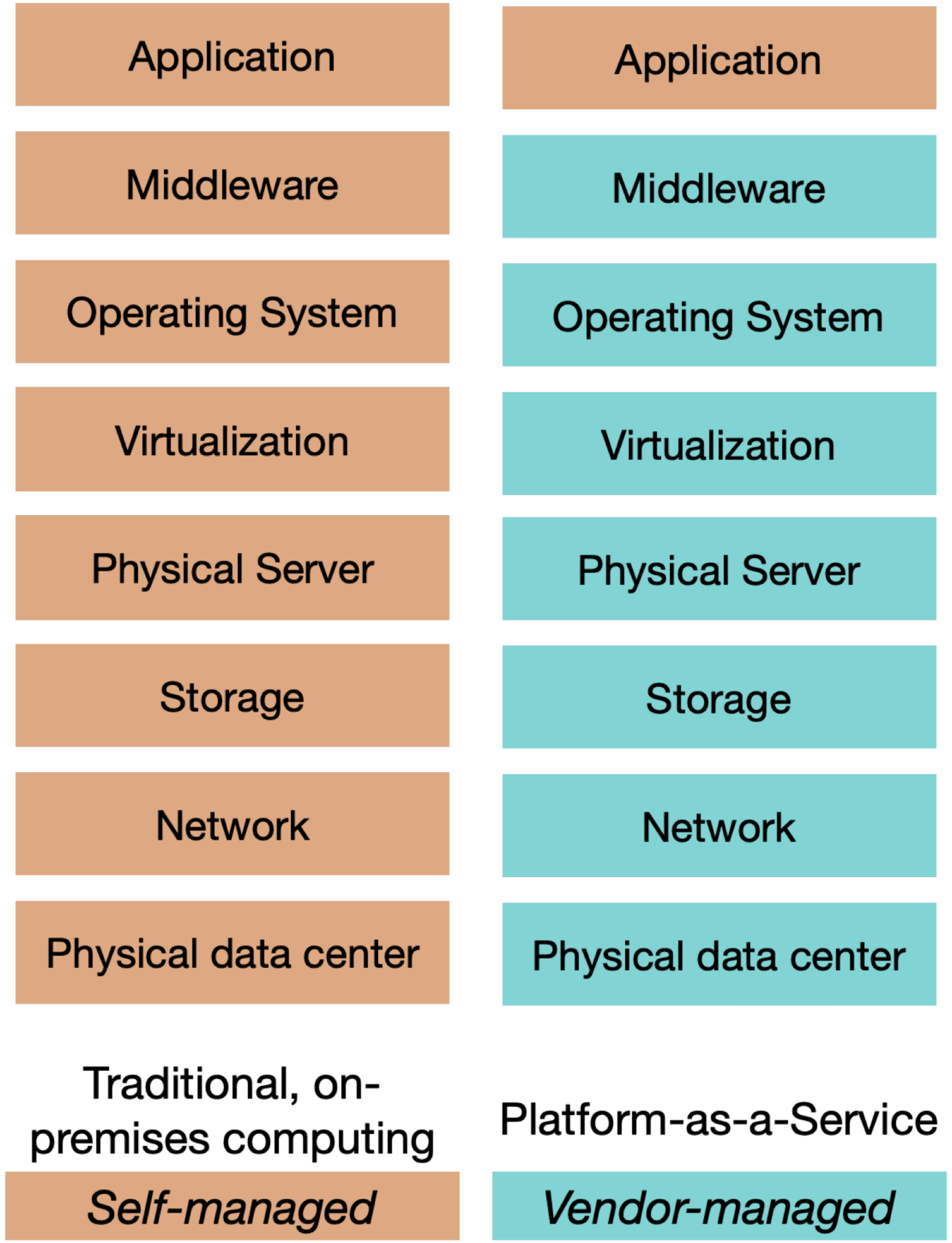
Pizza as a Service — by Albert Barron (unlicensed?)

Shared infrastructure creates economies of scale

- At the physical level:
  - Multiple customers' physical machines in the same data center
  - Save on physical costs (centralize power, cooling, security, maintenance)
- At the physical server level:
  - Multiple customers' virtual machines in the same physical machine
  - Save on resource costs (utilize marginal computing capacity – CPUs, RAM, disk)
- At the application level:
  - Multiple customer's applications hosted in same virtual machine
  - Save on resource overhead (eliminate redundant infrastructure like OS)
- "Cloud" is the natural expansion of multi-tenancy at all levels

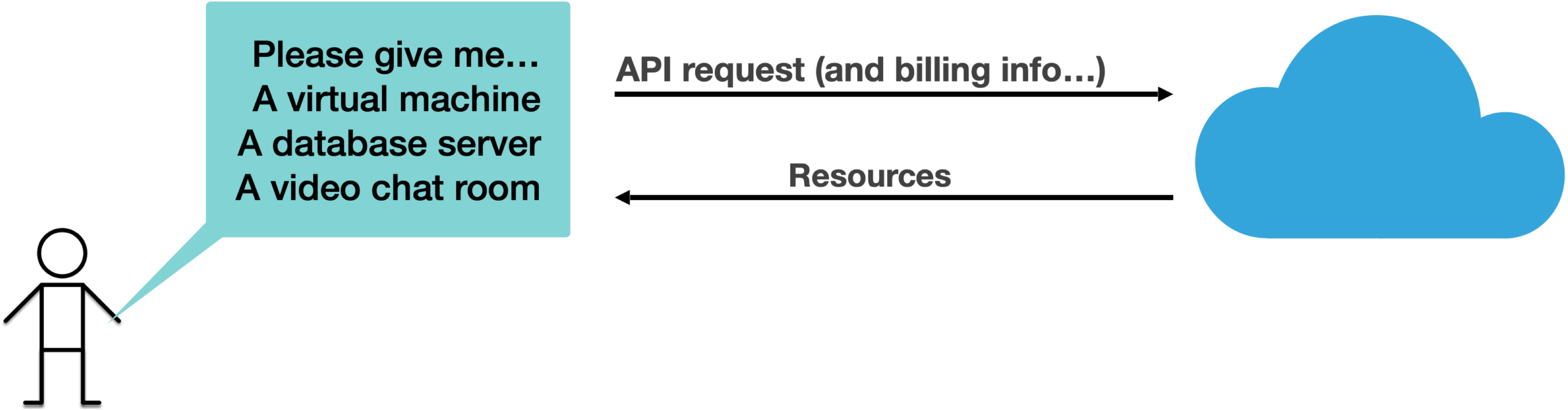
What is the infrastructure that can be shared?

- Our apps run on a “tall stack” of dependencies
- Old style: this full stack is self-managed
- Cloud providers offer products that manage parts of that stack for us:
  - “Infrastructure as a service”
  - “Platform as a service”
  - “Software as a Service”
  - Collectively called “X as a Service”



Cloud services gives on-demand access to infrastructure, “as a service”

- Vendor provides a service catalog of “**X as a service**” abstractions that provide infrastructure as a service
- API allows us to provision resources on-demand
- Transfers responsibility for managing the underlying infrastructure to a vendor

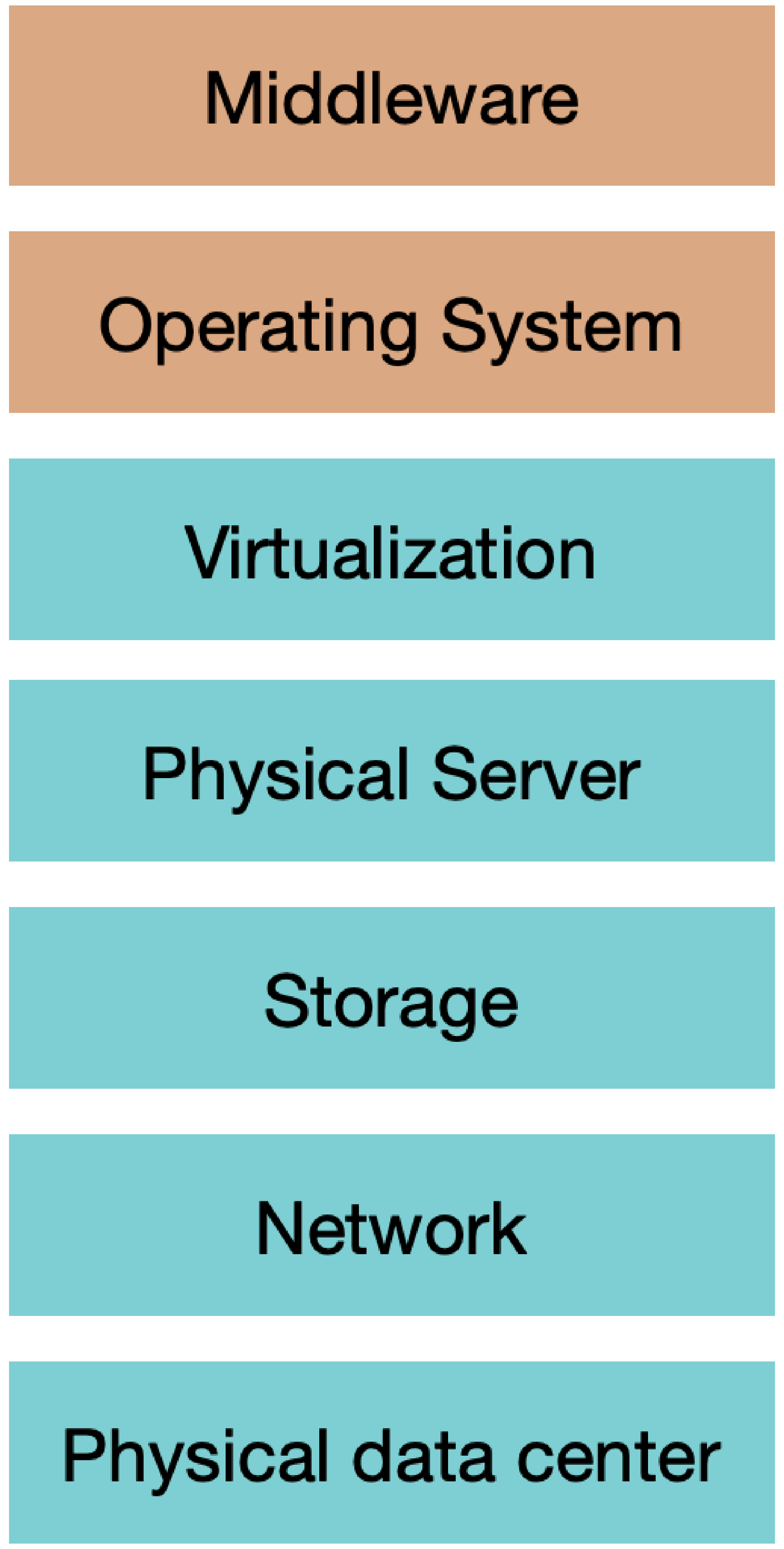


Cloud infrastructure scales elastically

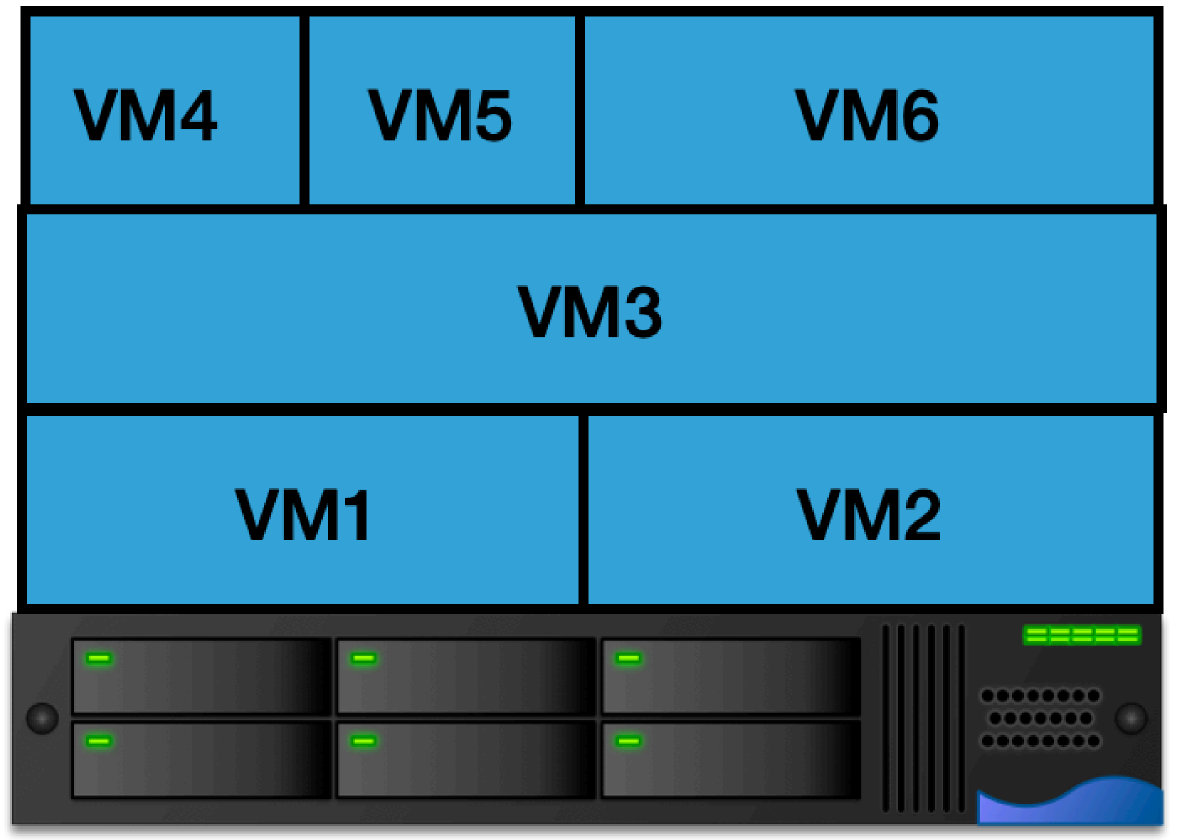
- “Traditional” computing infrastructure requires capital investment
  - “Scaling up” means buying more hardware, or maintaining excess capacity for when scale is needed
  - “Scaling down” means selling hardware, or powering it off
- Cloud computing scales elastically:
  - “Scaling up” means allocating more shared resources
  - “Scaling down” means releasing resources into a pool
  - Billed on consumption (usually per-second, per-minute or per-hour)

Infrastructure as a Service: Virtual Machines

- Virtualize a single large server into many smaller machines
- Each VM runs its own OS
- OS limits resource usage and guarantees per-VM quality
- Administration responsibilities separated for physical machine vs virtual machines
- Examples:
- Cloud: Amazon EC2, Google Compute Engine, Azure
- On-Premises: VMWare, Proxmox



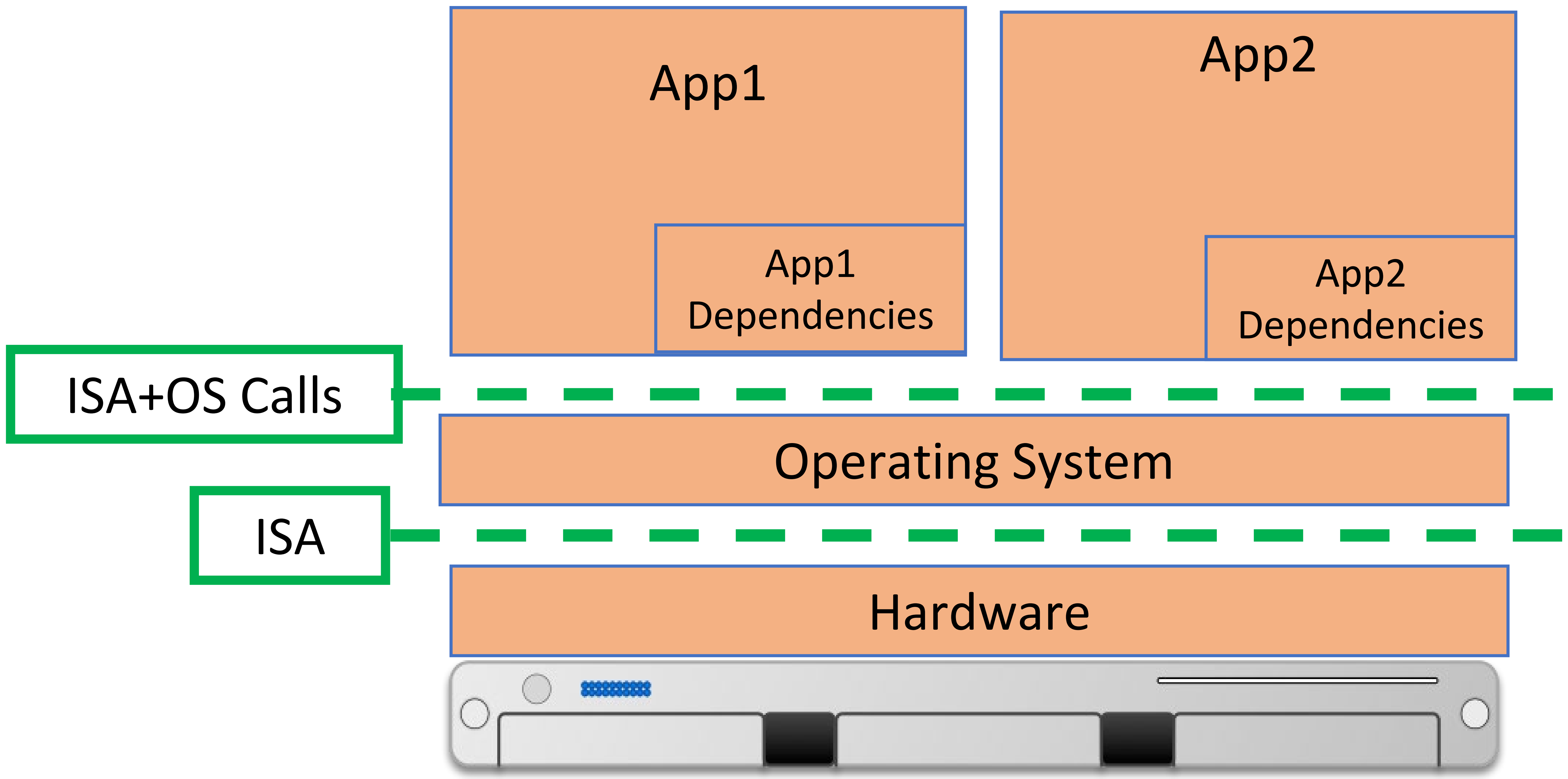
IaaS



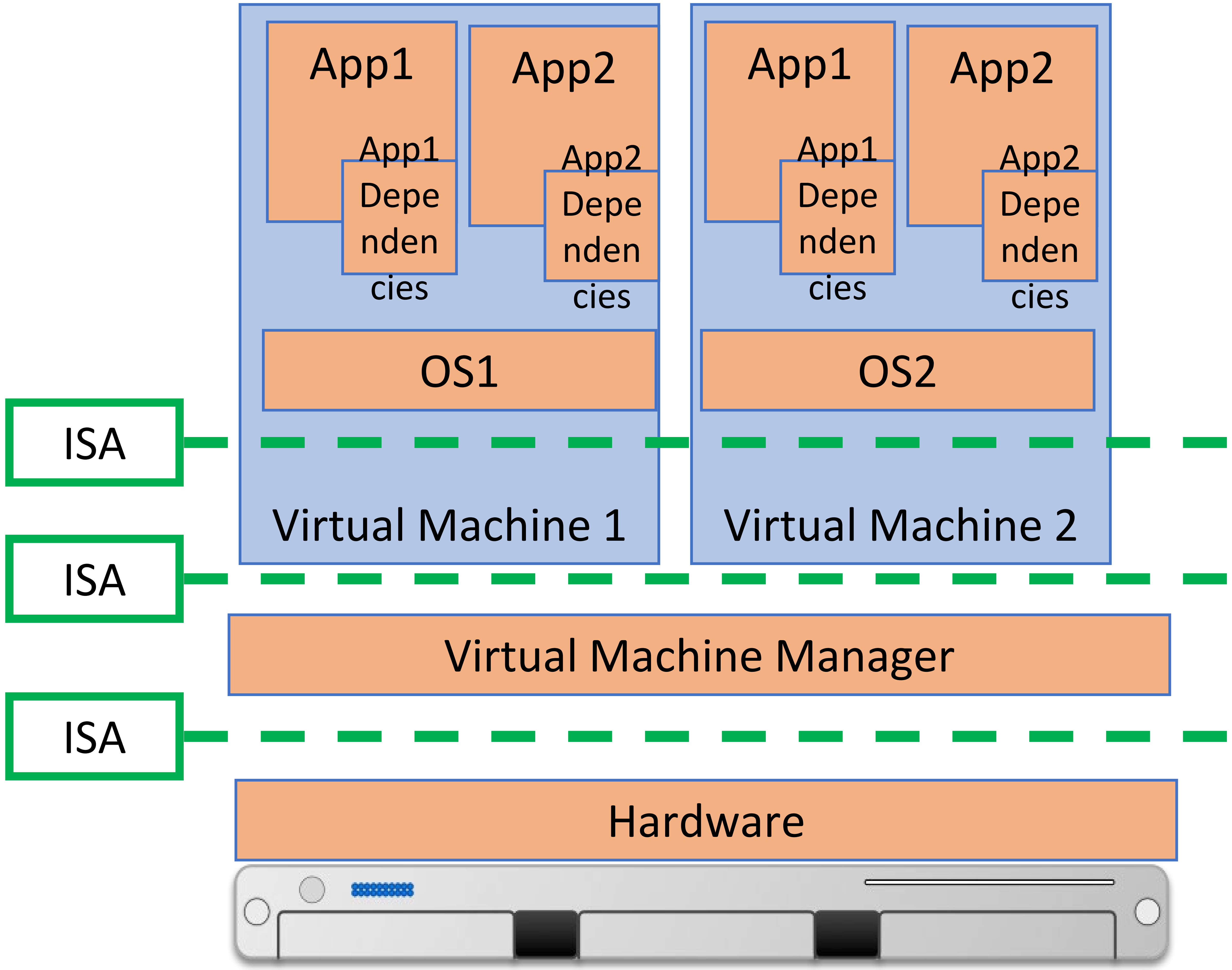
Abstracted physical machine

The operating system allows several apps to share the underlying hardware

- The "instruction set" is an abstraction of the underlying hardware
- The operating system presents the same abstraction + OS calls.



A virtual machine layer allows several different operating systems to share the same hardware



Virtual Machines facilitate multi-tenancy

- Multi-Tenancy
  - Multiple customers sharing same physical machine, oblivious to each other
- Decouples application from hardware
  - virtualization service can provide “live migration” transparent to the operating system, maximizing utilization
- Faster to provision and release
  - VM v. physical machines == ~mins v. ~hours (days?)

Virtual Machines to Containers

- Each VM contains a full operating system
- What if each application could run in the same (overall) operating system? Why have multiple copies?
- Advantages to smaller apps:
  - Faster to copy (and hence provision)
  - Consume less storage (base OS images are usually 3-10GB)

Containers run layered images, reducing storage space

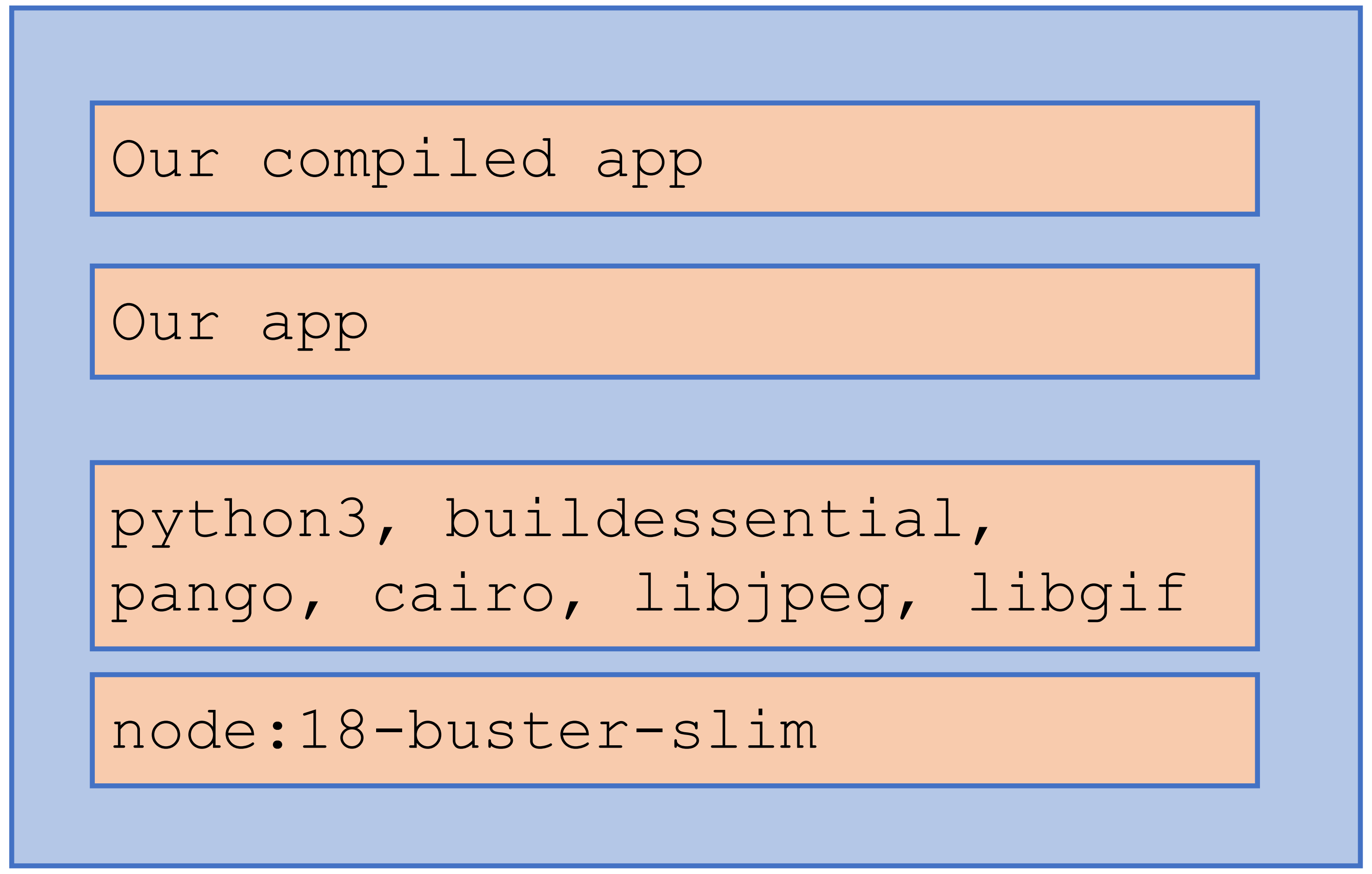
- Images are defined programmatically as a series of “build steps” (e.g. Dockerfile)
- Each step in the build becomes a “layer”
- Built layers can be shared and cached
- To run a container, the layers are linked together with an “overlay” filesystem

```
FROM node:18-buster-slim
RUN apt-get update && apt-get install python3
build-essential libpango1.0-dev libcairo2-dev
libjpeg-dev libgif-dev -y

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY ./ /usr/src/app

RUN npm ci
RUN npm run build
CMD [ "npm", "start" ]
```

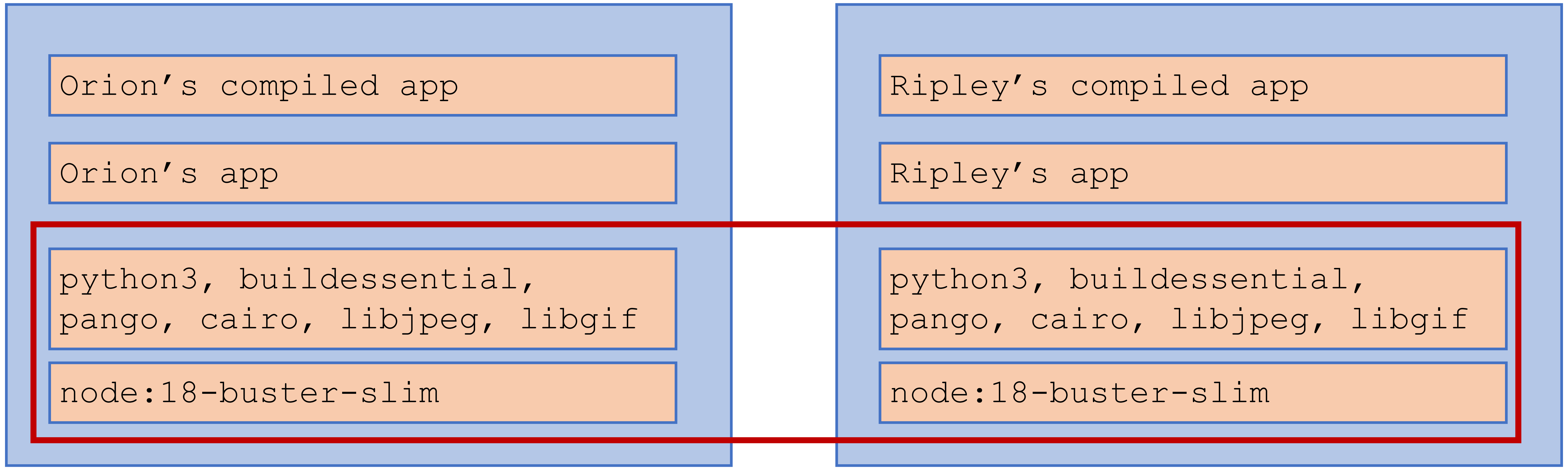
Example image specification (Dockerfile)



Example image, with layers shown

Containers run layered images, reducing storage space

- Many images may share the *same* lower layers (e.g. OS, NodeJS, some system dependencies)
- Layers are shared between images
- Multi-tenancy: *N* running containers only require *one* copy of each layer (they are read-only)



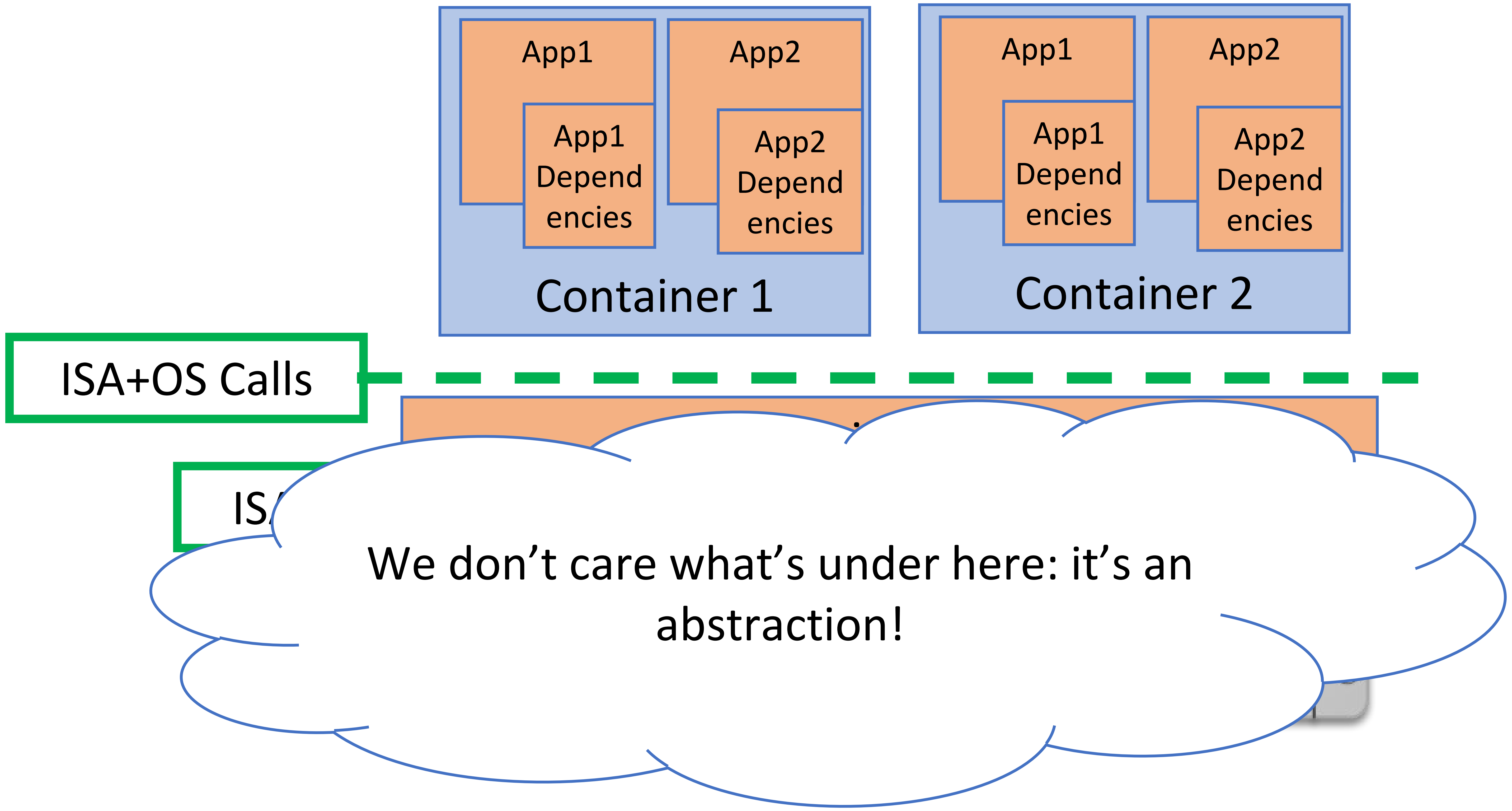
Two images, sharing two layers

A container contains your apps and all their dependencies

- Each application is encapsulated in a “lightweight container,” includes:
  - System libraries (e.g. glibc)
  - External dependencies (e.g. nodejs)
- “Lightweight” in that container images are smaller than VM images - multi tenant containers run in the OS
- Cloud providers offer “containers as a service” (Amazon AWS Fargate, Azure Kubernetes, Google Kubernetes)

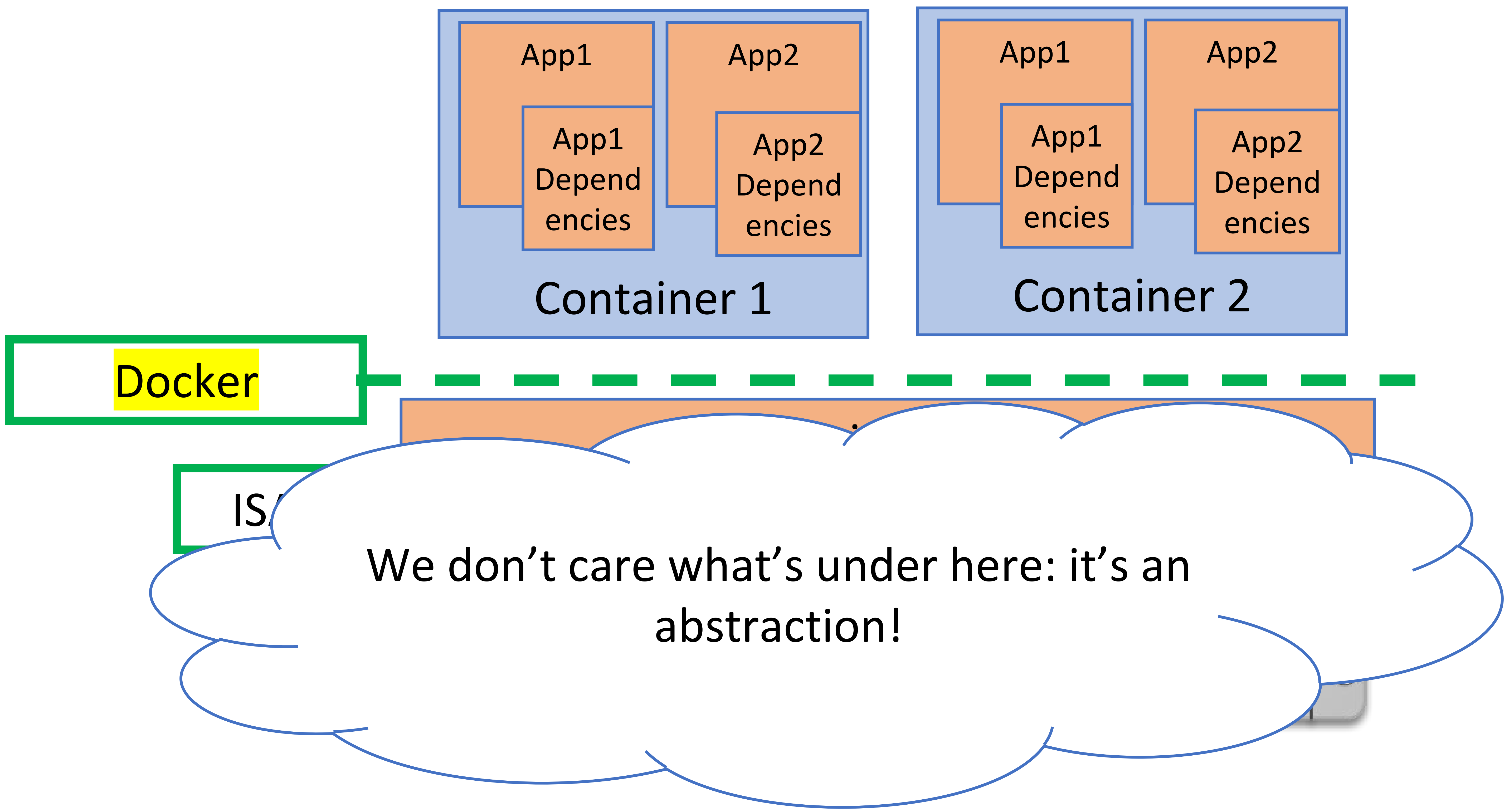
XaaS: Containers as a Service

- Vendor supplies an on-demand instance of an operating system
  - e.g.: Linux version NN
- Vendor is free to implement that instance in a way that optimizes costs across many clients.



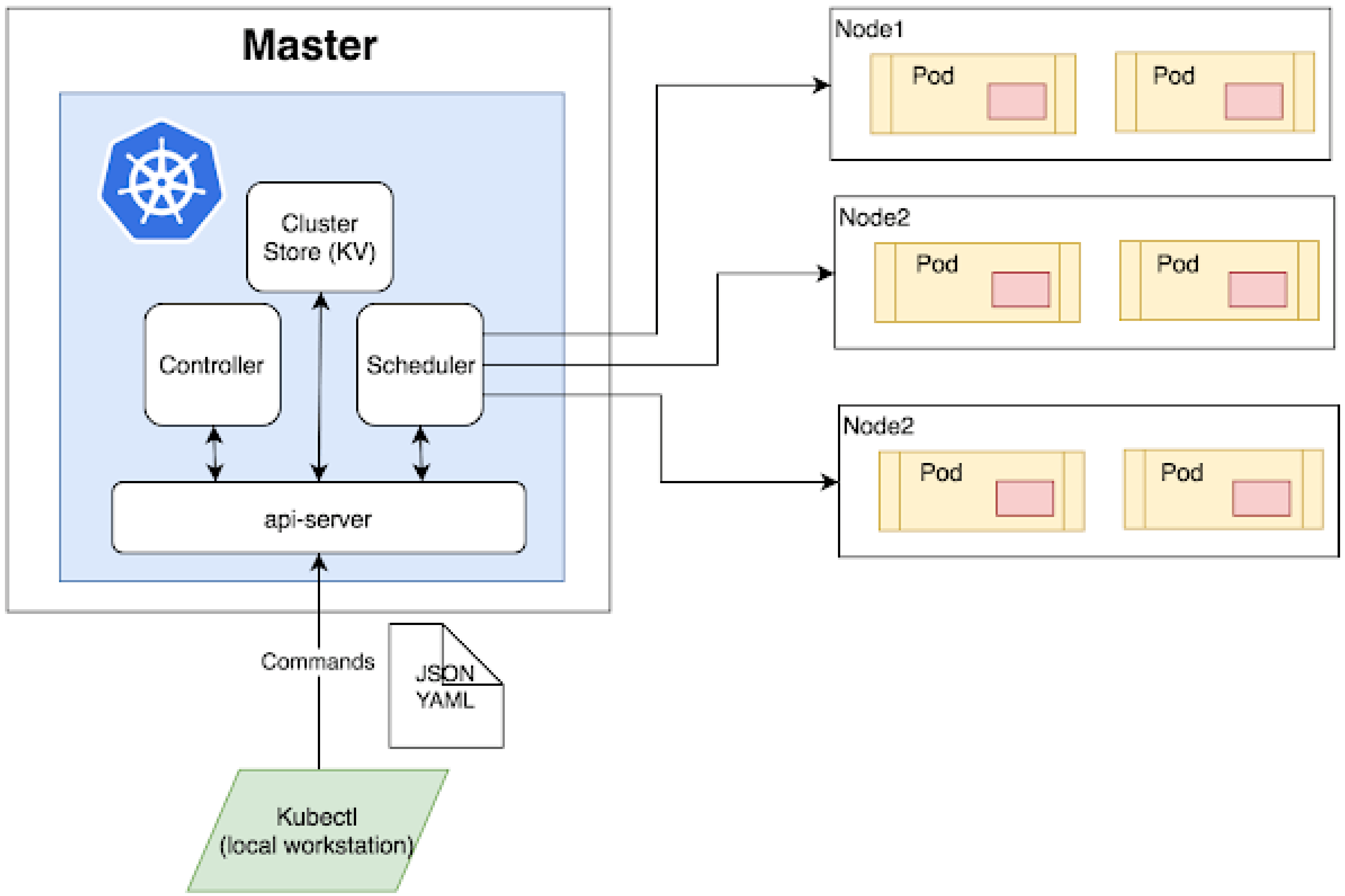
Docker is the prevailing container platform

- Docker provides a standardized interface for your container to use
- Many vendors will host your Docker container
- An open standard for containers also exists ("OCI")

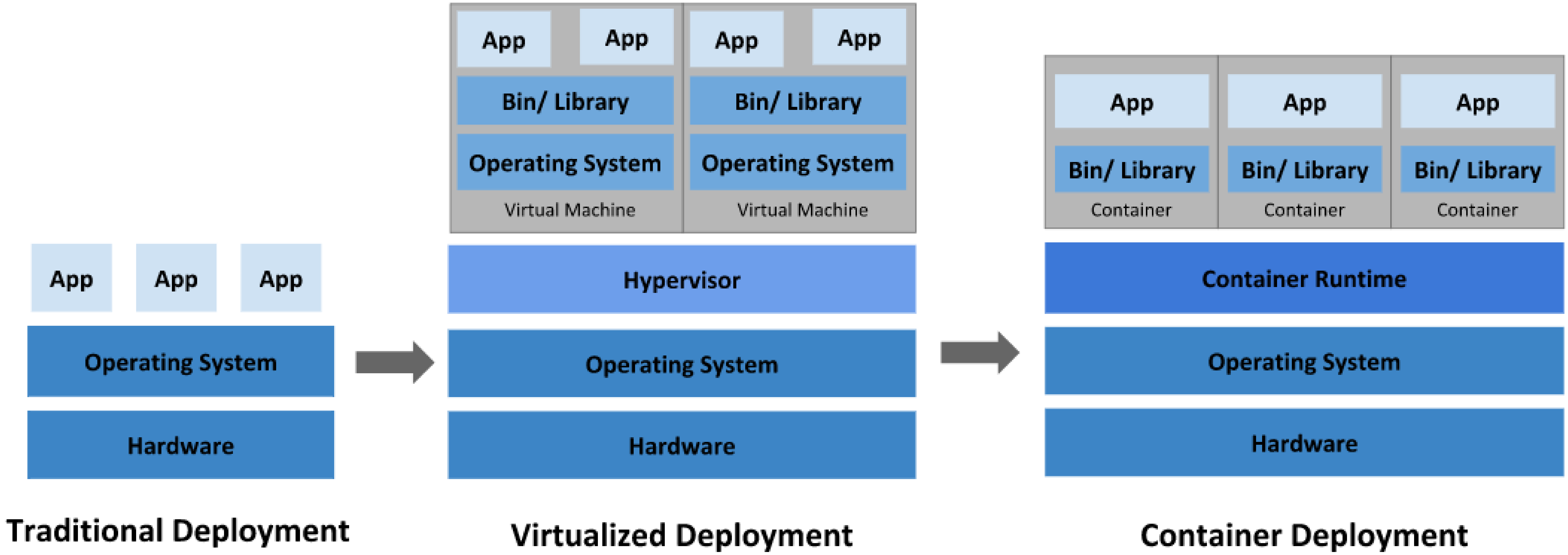


Kubernetes is an orchestration system for managing Docker containers at scale

- Containers alone are lightweight but **hard to manage at scale**.
- Kubernetes is an **open-source platform** for **automating deployment, scaling, and management** of containerized applications.



VMs and Containers



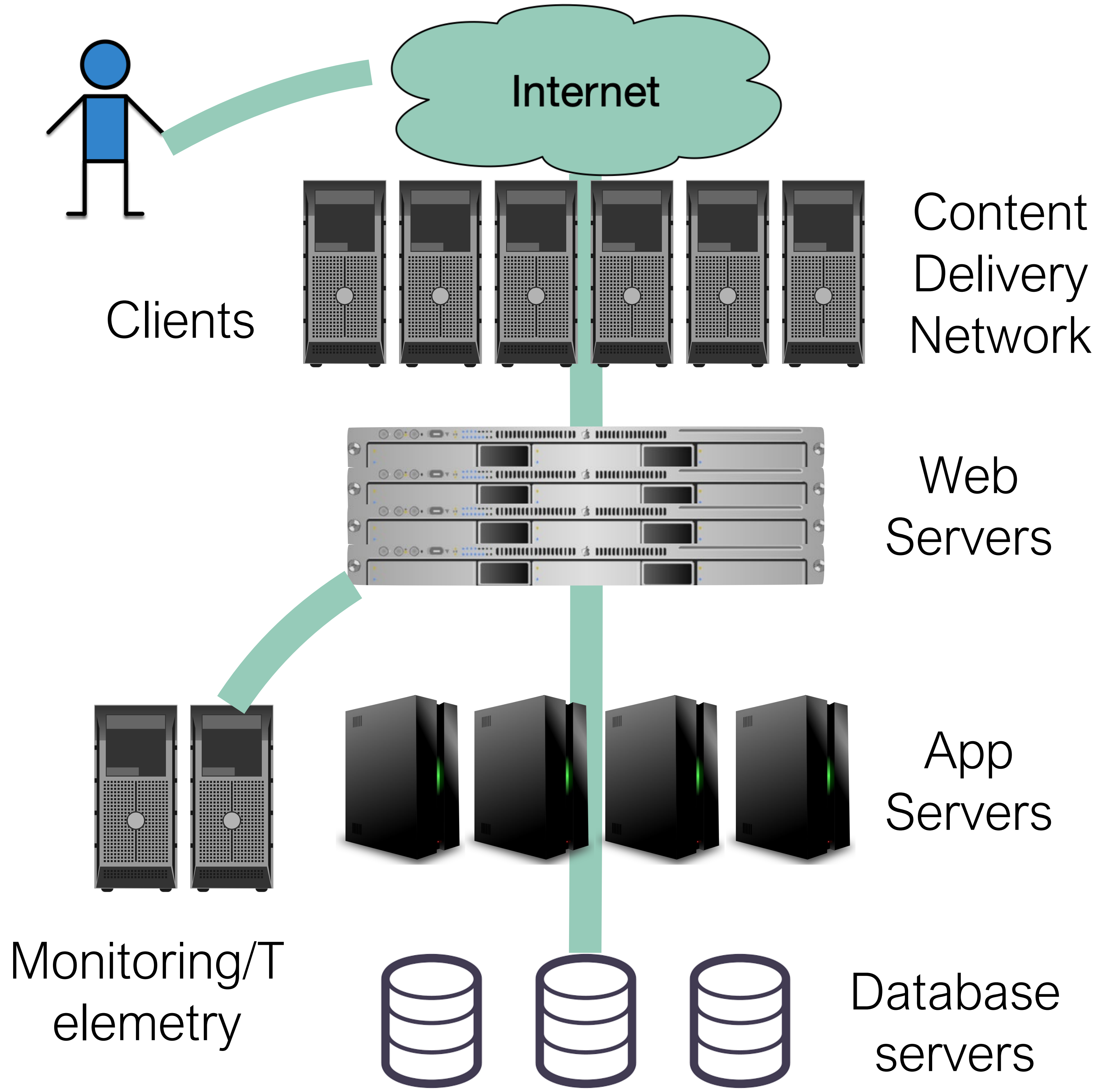
https://kubernetes.io/docs/concepts/overview/

Tradeoffs between VMs and Containers

- Performance is comparable
- Each VM has a copy of the OS and libraries
  - Higher resource overhead
  - Slower to provision
  - Support for wider variety of OS's
- Containers are "lightweight"
  - Lower resource overhead
  - Faster to provision
  - Potential for compatibility issues, especially with older software
- Containers are often chosen for speed, efficiency and scalability (and cost benefits)

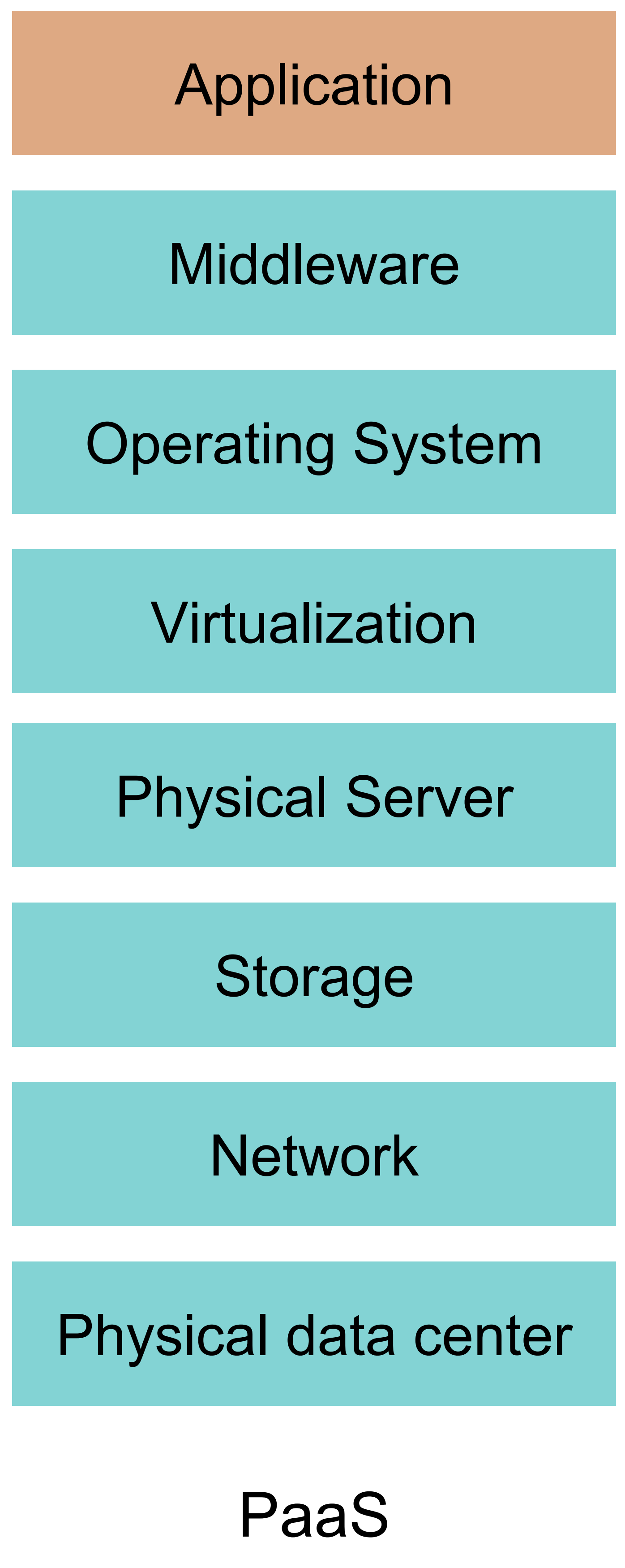
Platform-as-a-Service: vendor supplies OS + middleware

- Middleware is the stuff between our app and a user's requests:
  - Content delivery networks: Cache static content
  - Web Servers: route client requests to one of our app containers
  - Application server: run our handler functions in response to requests from load balancer
  - Monitoring/telemetry: log requests, response times and errors
- Cloud vendors provide managed middleware platforms too: "Platform as a Service"



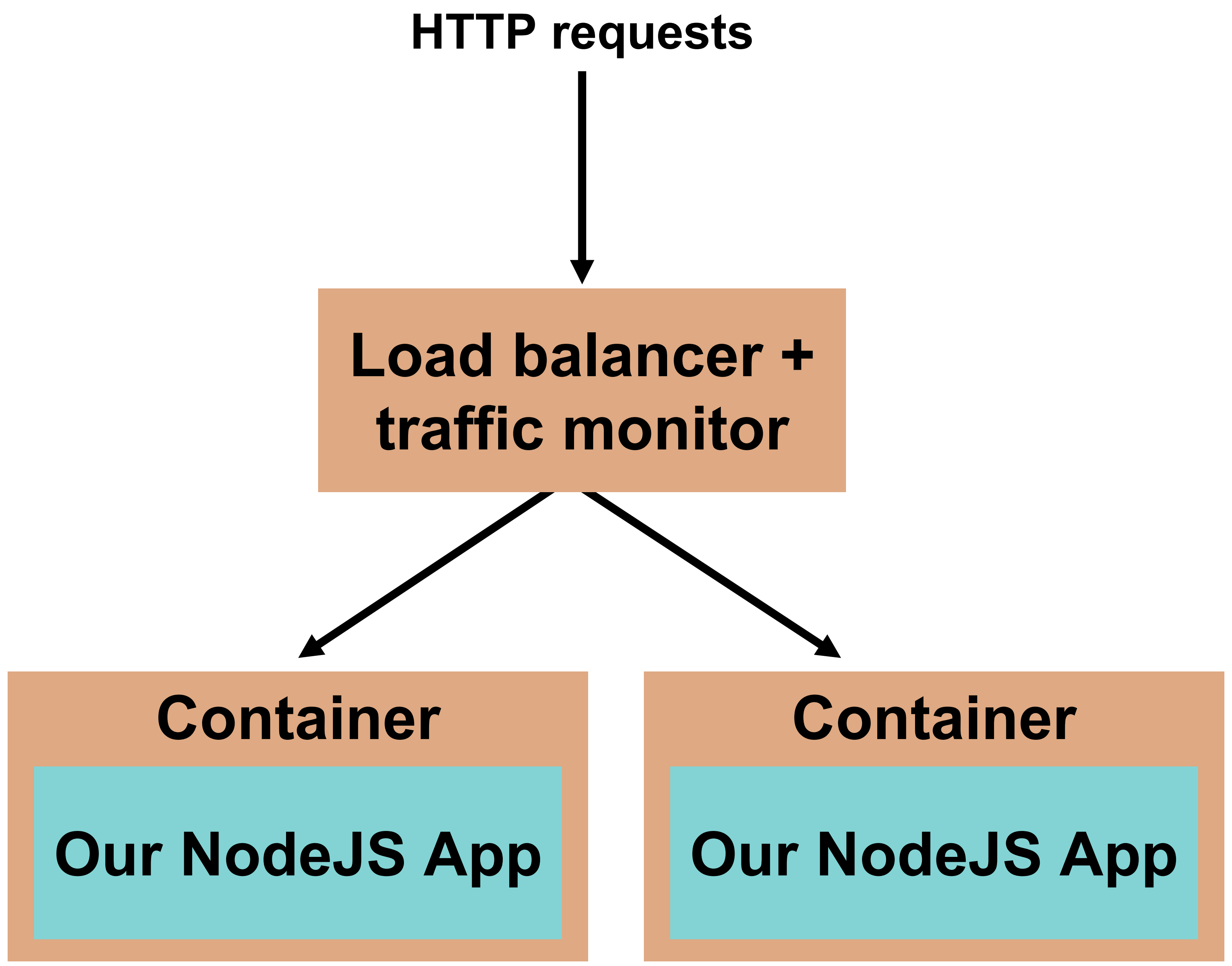
PaaS is often the simplest choice for app deployment

- **Platform-as-a-Service** provides components most apps need, fully managed by the vendor: load balancer, monitoring, application server
- Some PaaS run your app in a container: Heroku, AWS Elastic Beanstalk, Google App Engine, Railway, Vercel…
- Other PaaS run your apps as individual functions/event handlers: AWS Lambda, Google Cloud Functions, Azure Functions
- Other PaaSs provide databases and authentication, and run your functions/event handlers: Google Firebase, Back4App



PaaS in the style of Heroku runs containers

- Takes a web app as input
  - Provide an entry point to code, e.g. “npm start”, or optionally, a container specification
- Hosts web app at chosen URL, can scale resources up/down on-demand
  - Load balancer fully managed by Heroku, scaling transparent
  - Auto-scale down to use no resources, spins up container on reception of a request
  - Dashboard for monitoring/reporting
- Newcomers provide similar functionality (Vercel, Railway, etc)
- Host PaaS on-premises, too (Caprover)

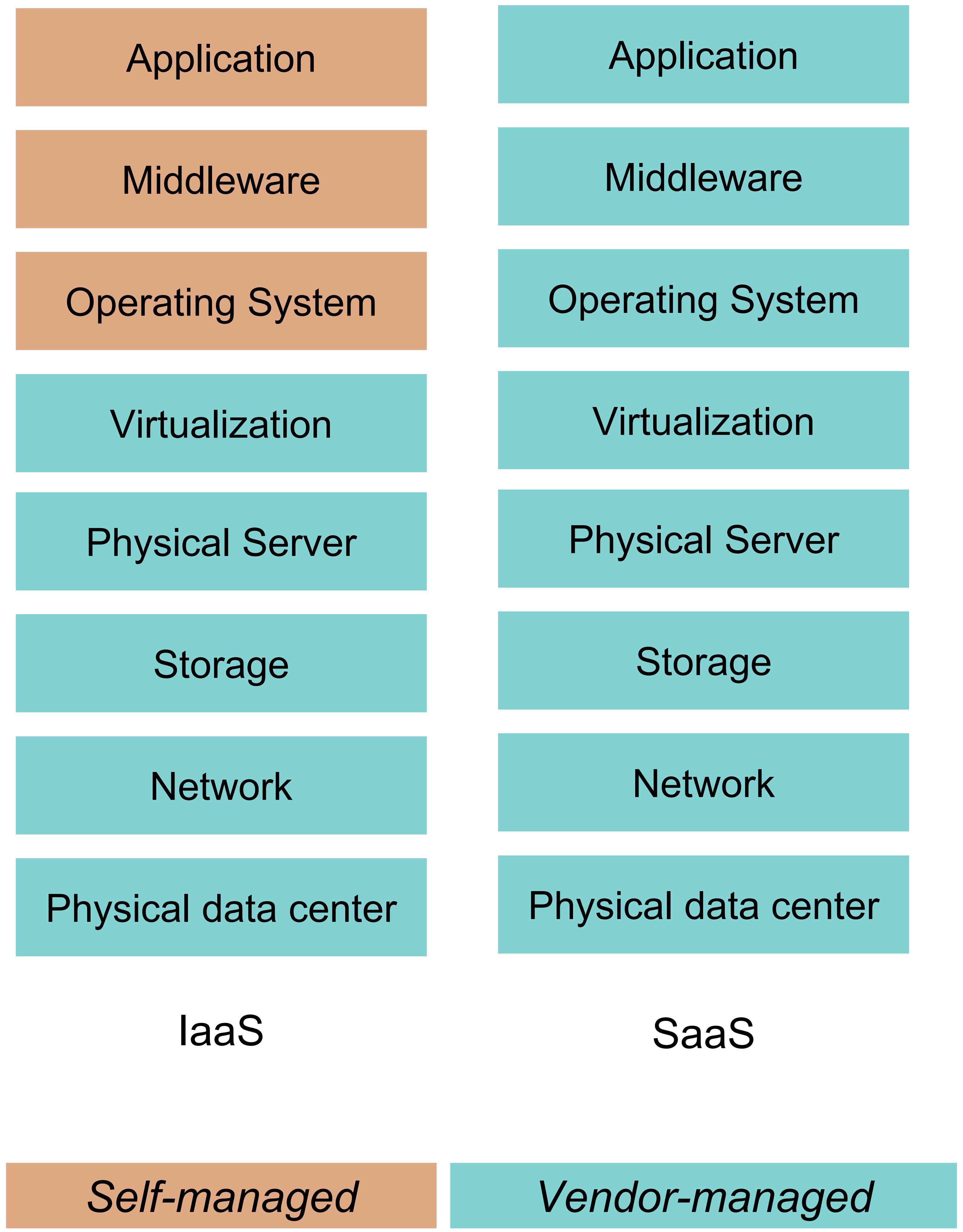


Render.com uses a mixed strategy for implementation

- Web services are run in containers on their app servers.
  - Load balancers, dashboards, etc. are shared by all users
  - Other services: eg Redis ("Software as a Service")
- "Static Web Sites" are hosted as static files on their CDN.
  - minimizes load times for users world-wide.

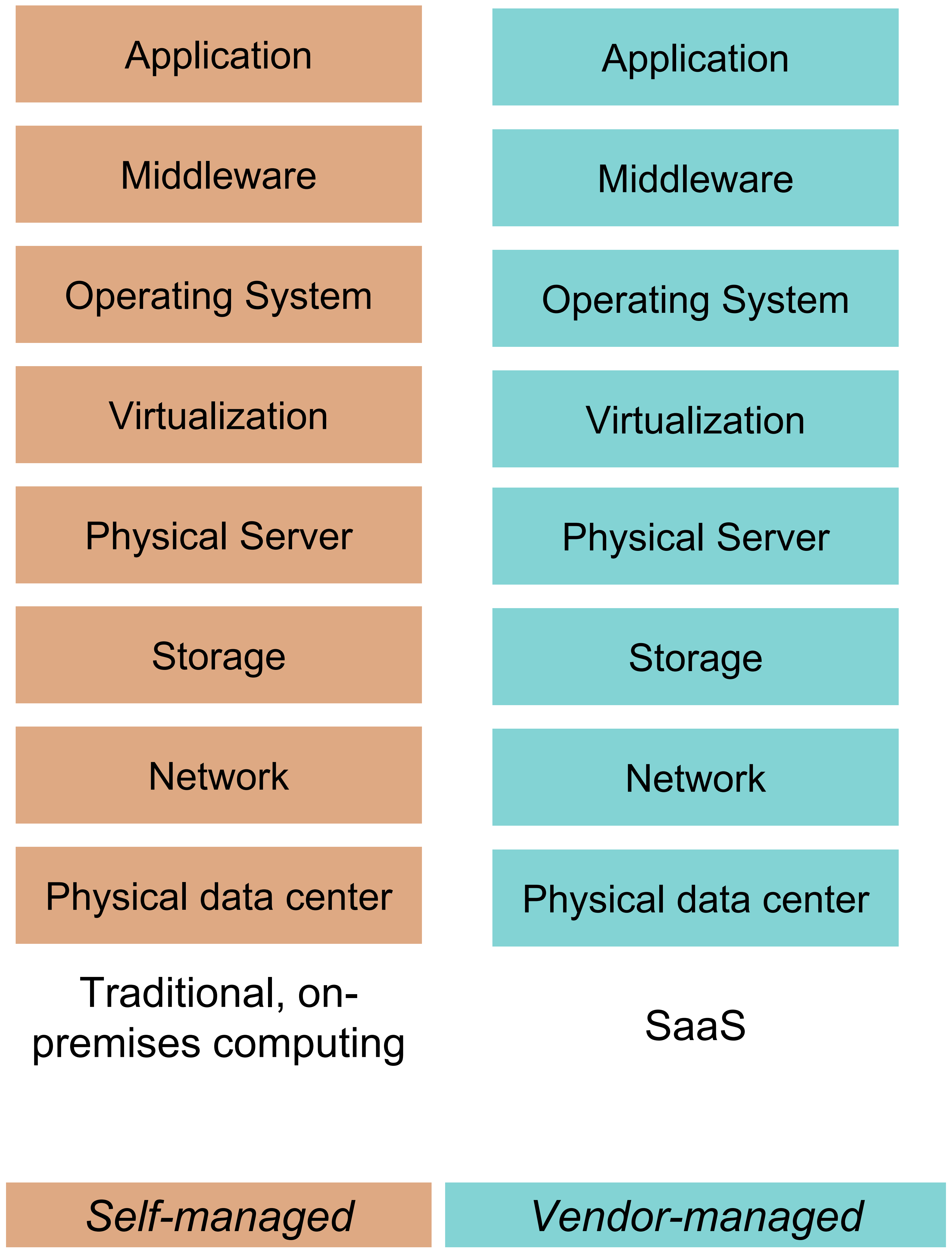
Software as a Service adds more vendor-managed apps

- Providers may also develop custom software offered only as a service
- Examples:
  - PostgreSQL (open source)
  - Twilio Programmable Video (proprietary chat)



Self-managed vs Vendor-managed Infrastructure Tradeoffs

- Consider who manages each tier in the stack
- Benefits to vendor-managed options:
  - More ways to reduce resource consumption, improve resource utilization
  - Less management burden
  - Less capital investment, more flexibility in scaling
- Benefits to self-managed options:
  - Greater flexibility to migrate between software platforms
  - More capital investment, potentially less operating expense



Cloud Infrastructure can be better for variable workloads

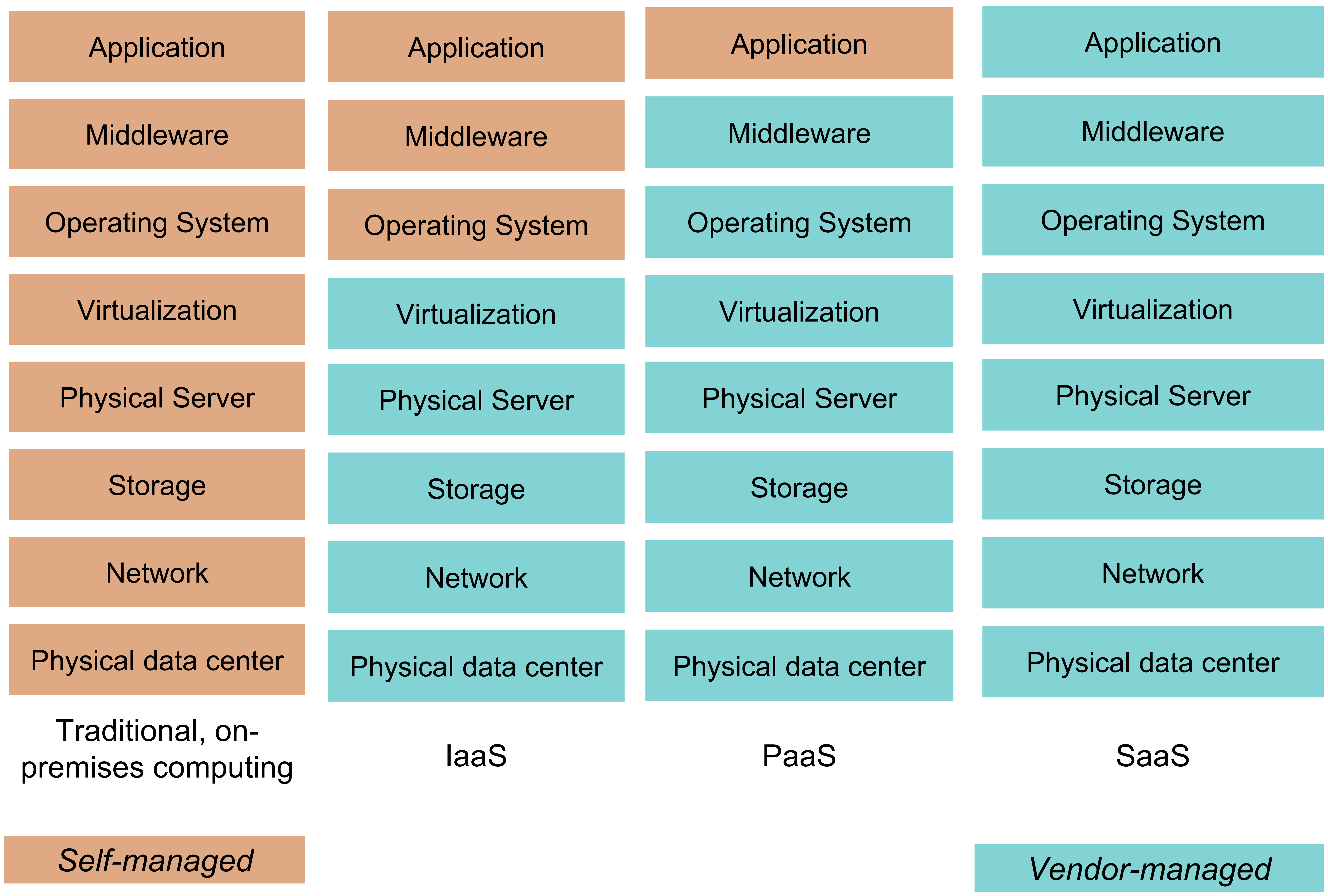
- Consider:
  - Does your workload benefit from ability to scale up or down?
  - Variable workloads have different demands over time (most common)
  - Constant workloads require sustained resources (less common)
- Example:
  - Need to run 300 VMs, each 4 vCPUs, 16GB RAM
- Private cloud:
  - Dell PowerEdge Pricing (AMD EPYC 64 core CPUs)
  - 7 servers, each 128 cores, 512GB RAM, 3 TB storage = $162,104
- Public cloud:
  - Amazon EC2 Pricing (M7a.xlarge instances, $0.153/VM-hour)
  - 10 VMs for 1 year + 290 VMs for 1 month: $45,792.90
  - 300 VMs for 1 year: $402,084.00

Public clouds are not the only option

- “Public” clouds are connected to the internet and available for anyone to use
  - Examples: Amazon, Azure, Google Cloud, DigitalOcean
- “Private” clouds use cloud technologies with on-premises, self-managed hardware
  - Cost-effective when a large scale of baseline resources are needed
  - Example management software: OpenStack, VMWare, Proxmox, Kubernetes
- “Hybrid” clouds integrate private and public (or multiple public) clouds
  - Effective approach to “burst” capacity from private cloud to public cloud

“X as a Service" offers several abstractions to choose from depending on your needs

- Vendor manages different levels of the stack, achieving economies of scale
- When would you choose one over the other?
- Explore some options at https://comparecloud.in/



Review

- You should now be able to…
  - Explain what "cloud" computing is and why it is important
  - Explain why shared infrastructure is important in cloud computing
  - Describe the difference between virtual machines and containers
  - Discuss trade-offs that you might consider for self or vendor-managed platforms