

9 Technological Outlook

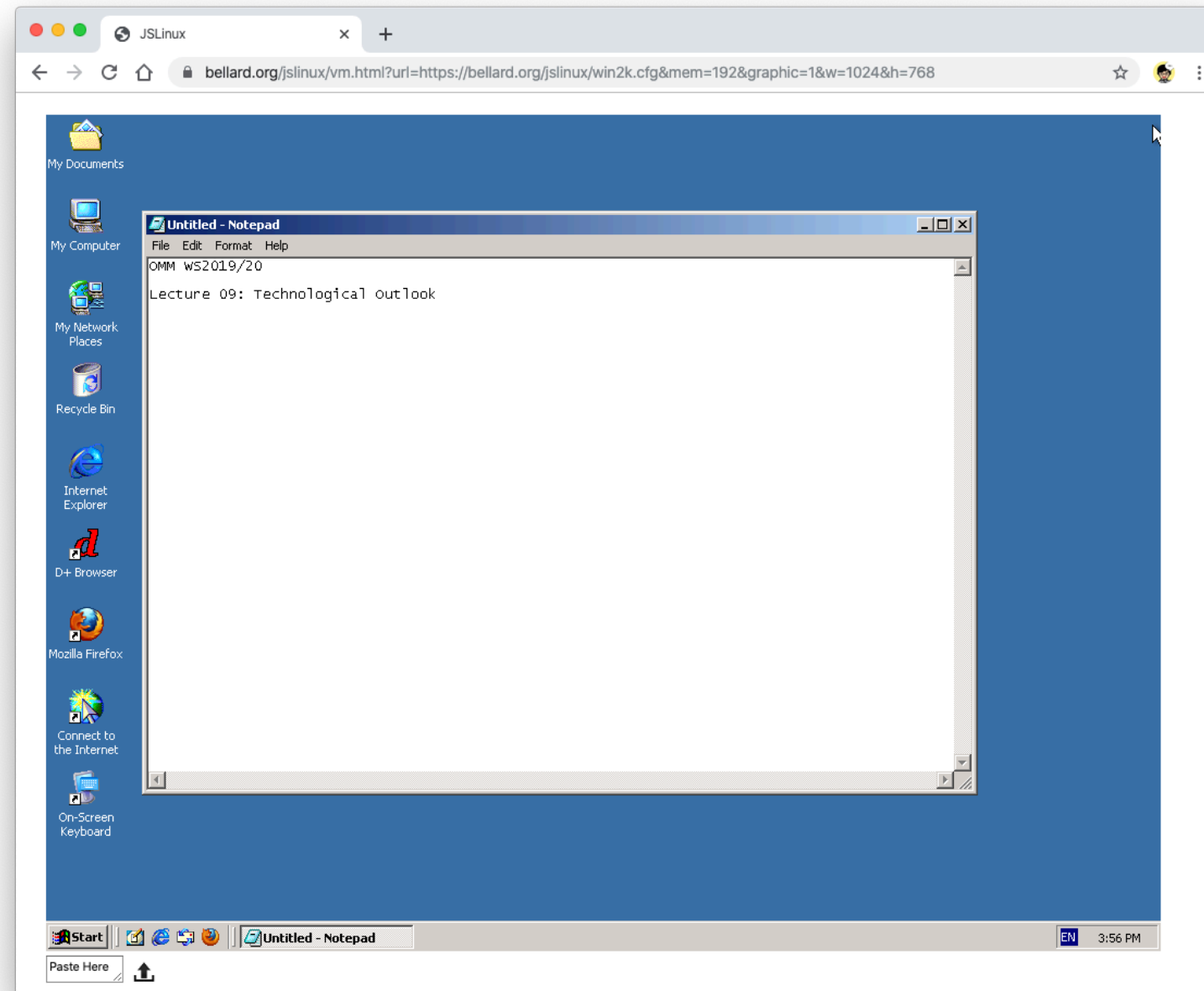
- 9.1 Web Application on Bare Metal: WebAssembly
- 9.2 Container Orchestration and Cloud Native
- 9.3 Other Selected Trends: Go, HTTP/3
- 9.4 Conclusion

Literature:

<https://webassembly.org>

Example: Windows 2000 in a Browser

- Can you imagine how the OS is running in a browser?



Windows 2000 runs in a browser, <https://bellard.org/jslinux/>

WebAssembly (WASM)



The screenshot shows the W3C News page. The main heading is "W3C RECOMMENDS WEBASSEMBLY TO PUSH THE LIMITS FOR SPEED, EFFICIENCY AND RESPONSIVENESS". The date "5 December 2019" is highlighted with an orange box. The article text states: "The [WebAssembly Working Group](#) has published today the three WebAssembly specifications as W3C Recommendations, marking the arrival of a new language for the Web which allows code to run in the browser." The article includes two bullet points: one for the [WebAssembly Core Specification](#) and another for the [WebAssembly Web API](#). The page also features a sidebar with navigation links like "MAIL, NEWS, BLOGS, PODCASTS, AND TUTORIALS" and a search bar for "Search W3C News".

<https://www.w3.org/blog/news/archives/8123>

Evolution of Front-end Engineering

- ECMAScript evolves a lot
 - The first formal draft submitted to ECMA (ECMAScript 1.1, 1997)
 - “Strict mode” is introduced (ECMAScript 5, 2009)
 - Massive changes to the language (ECMAScript 6, 2015)
 - Latest version: ECMAScript 2019 (version 10)
- Business is becoming much more complex
 - HTML/CSS/JS in the beginning
 - jQuery addresses pain points better manipulating DOMs and AJAX
 - Frameworks (Phase 1): Knockout / Backbone / AngularJS
 - Tooling: NodeJS/NPM/Babel/Webpack ...
 - Frameworks (Phase 2): React/Angular/Vue
- JavaScript, as a dynamic typed language, is the only language for front-end web development

JavaScript: The Good Parts



https://www.reddit.com/r/ProgrammerHumor/comments/621qrt/javascript_the_good_parts/

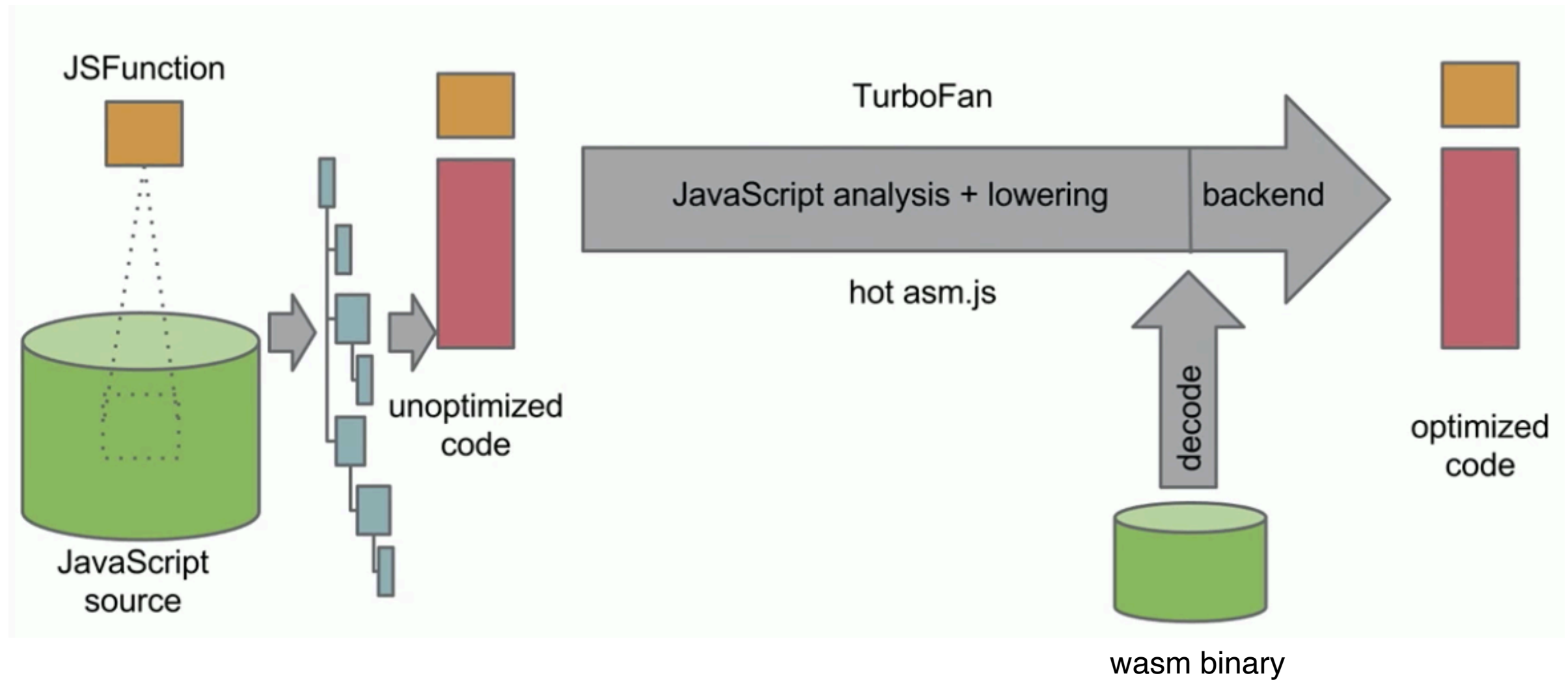
Early Attempts: ASM.js and NaCl

- JavaScript deduces types in runtime
- ASM.js by Mozilla
 - A subset of JavaScript to avoid type inconsistency and garbage collection
 - Proved that languages can be transpiled to JavaScript and run in the browser

```
function f(i) {  
    i = i|0;  
    return (i + 1)|0;  
}
```

- NativeClient (NaCl) by Google
 - but never implemented except Chrome
 - Dropped in 2017

V8 Pipeline Design + WASM



Source: [Compiling for the Web with WebAssembly \(Google I/O '17\)](#)



WebAssembly (WASM)

- Binary instruction format
 - a low-level virtual machine standard for web application
 - Memory safe execution environment sandbox
- W3C WebAssembly Working Group, Community Group
- The “fourth” language for web development
- Benefits
 - Speed: (Near) native
 - Portability: Extreme Low-level
 - Flexibility: Get rid of JavaScript only



<https://caniuse.com/#search=wasm>

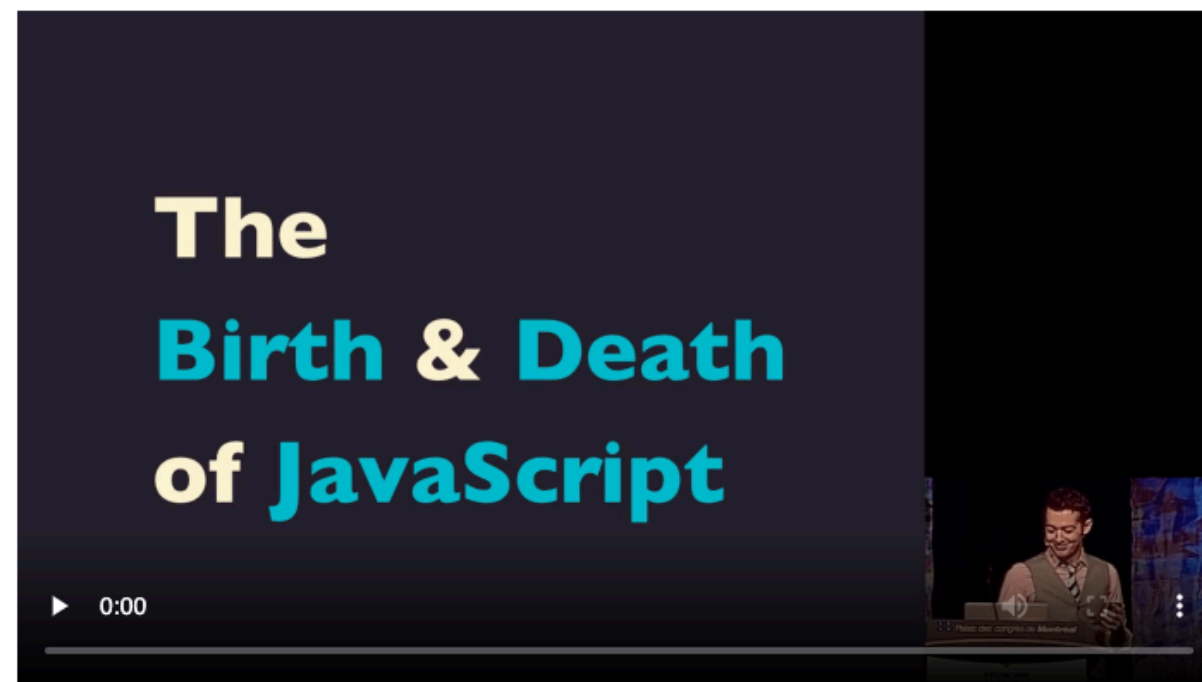
Discussion

- What makes asm.js and NaCl failed?
- Do you think JavaScript will die in the near future?

☰ DESTROY ALL SOFTWARE

The Birth & Death of JavaScript

A talk by Gary Bernhardt from PyCon 2014



<https://www.destroyallsoftware.com/talks/the-birth-and-death-of-javascript>

9 Technological Outlook

- 9.1 Web Application on Bare Metal: WebAssembly
- 9.2 Container Orchestration and Cloud Native
- 9.3 Other Selected Trends: Go, HTTP/3
- 9.4 Conclusion

Literature:

Newman S. Building microservices: designing fine-grained systems. O'Reilly Media, Inc. 2015 Feb 2.

<https://kubernetes.io>

<https://cncf.io>

Example: Occamy Remote Desktop Streaming

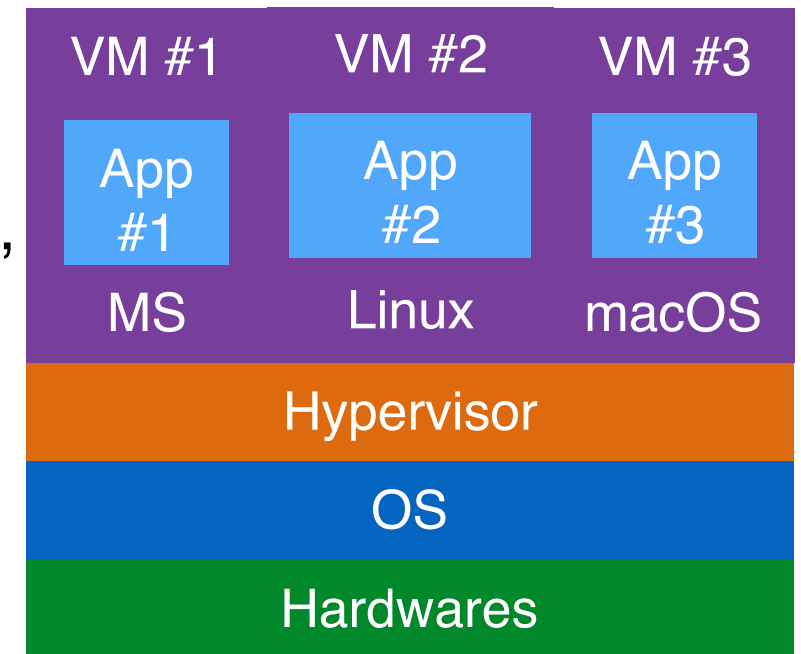
The image shows a terminal window on the left and a web browser on the right. The terminal window displays the output of a `make run` command, which sets up a Docker container for Occamy. A red box highlights the initial setup steps: `Creating network "docker_occamy_network" with driver "bridge"`, `Creating rdp ... done`, `Creating ssh ... done`, `Creating vnc ... done`, and `Creating occamy ... done`. The terminal also shows the container's configuration, including the user ID (1000), group ID (0), and the VNC resolution (1280x1024). The web browser on the right shows the Occamy interface, which features a yellow 3D character standing on the shoulders of two white 3D characters. The browser's network tab is open, showing a list of requests, including `connect?token=eyJh...`.

```
# changkun at ou-lmu.wintermute.medeia in ~/dev/occamy
$ make run
cd docker && docker-compose up
Creating network "docker_occamy_network" with driver "bridge"
Creating rdp ... done
Creating ssh ... done
Creating vnc ... done
Creating occamy ... done
Attaching to ssh, rdp, vnc, occamy
vnc    USER_ID: 1000, GROUP_ID: 0
vnc    nss_wrapper location: /usr/lib/libnss_wrapper.so
vnc    ----- update chromium-browser.init -----
vnc    ... set window size 1280 x 1024 as chrome window size!
rdp    *** Running /etc/rc.local...
vnc    ----- change VNC password -----
vnc    ----- start noVNC -----
rdp    *** Booting runit daemon...
vnc    ----- start VNC server -----
vnc    remove old vnc locks to be a reattachable container
vnc    start vncserver with param: VNC_COL_DEPTH=24, VNC_RESOLUTION=1280x1024
vnc    ...
rdp    *** Runit started as PID 8
rdp    tail: unrecognized file system type 0x794c7630 for '/var/log/syslog'. please report this to bug-coreutils@gnu.org. reverting to polling
[GIN-debug] [WARNING] Creating an Engine instance with the Logger and Recovery middleware already attached.
occamy [GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.
occamy   - using env:      export GIN_MODE=release
occamy   - using code:    gin.SetMode(gin.ReleaseMode)
occamy [GIN-debug] GET    /static/*filepath    --> github.com/gin-gonic/gin.(*RouterGroup).createStaticHandler.func1 (3 handlers)
occamy [GIN-debug] HEAD  /static/*filepath    --> github.com/gin-gonic/gin.(*RouterGroup).createStaticHandler.func1 (3 handlers)
occamy [GIN-debug] GET    /api/v1/ping         --> github.com/changkun/occamy/server.(*proxy).Ping-fm (3 handlers)
occamy [GIN-debug] POST   /api/v1/login        --> github.com/appleboy/gin-jwt/v2.(*GINJWTMiddleware).LoginHandler-fm (3 handlers)
occamy [GIN-debug] GET    /api/v1/connect      --> github.com/changkun/occamy/server.(*proxy).serveWS-fm (4 handlers)
occamy [GIN-debug] GET    /debug/pprof/        --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] GET    /debug/pprof/cmdline --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] GET    /debug/pprof/profile --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] POST   /debug/pprof/symbol  --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] GET    /debug/pprof/symbol  --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] GET    /debug/pprof/trace   --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] GET    /debug/pprof/block   --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] GET    /debug/pprof/goroutine --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] GET    /debug/pprof/heap    --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] GET    /debug/pprof/mutex   --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy [GIN-debug] GET    /debug/pprof/threadcreate --> github.com/changkun/occamy/server.profile.func1.1 (3 handlers)
occamy time="2019-12-17T19:41:59Z" level=info msg="occamy-proxy: starting at http://0.0.0.0:5636..."
vnc    start window manager
vnc    ...
vnc    ----- VNC environment started -----
vnc
```

<https://github.com/changkun/occamy>

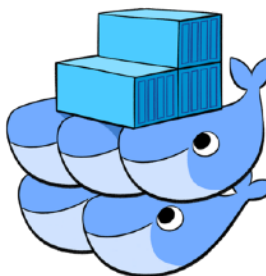
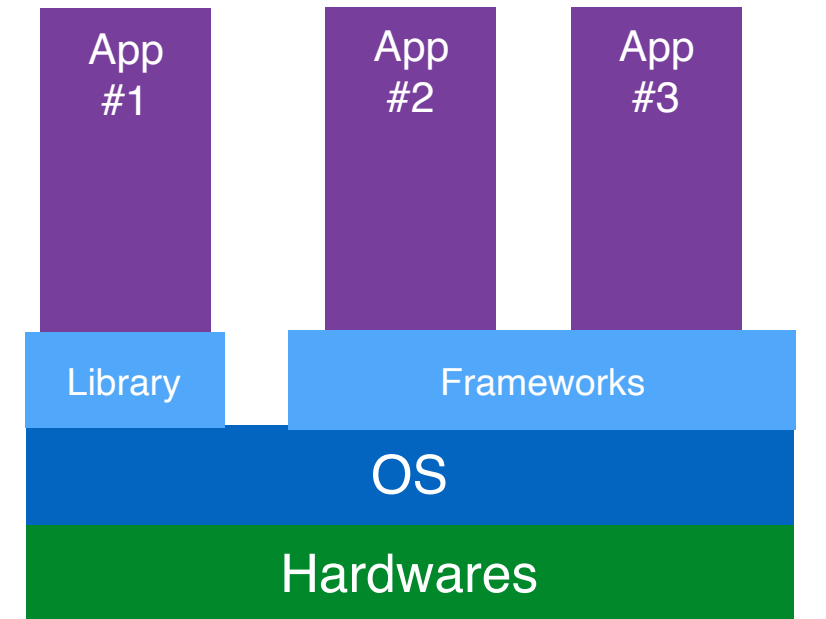
Virtualization (2000-2010)

- Windows 2000 (NT) Server introduce “Active Directory”
 - All servers centralized in a single domain
- Virtualization & OS-level resources isolation
 - Virtual machines (VMs) over the operating system
 - Debugging different platforms
 - Enables programmable hardware resource management automation
- Related tech.: VMware Workstation, vSphere, Hyper-V, QEMU, Xen, KVM...
- Products offer the ability of virtualization requires better managements
 - Infrastructure-as-a-Service (IaaS)
 - AWS by Amazon (2006)
 - Azure by Microsoft (2008)
 - OpenStack (2010)
- But VMs are expensive for lightweight applications



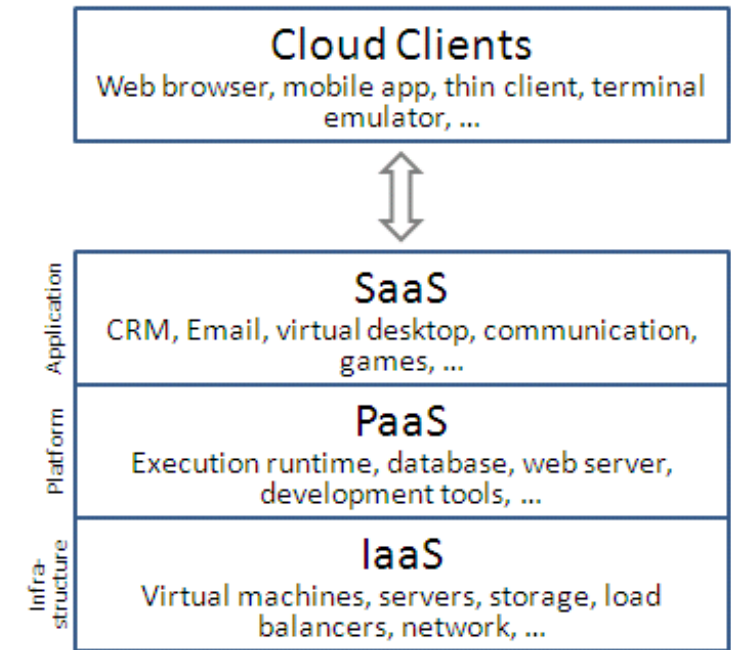
Containerization (2010-2015)

- Platform-as-a-Service (PaaS)
 - Cloud Foundry (2010) Foundation (2014)
 - OpenShift (2011)
- Docker (2013):
 - Encapsulate simple, friendly, and easy to use
 - **Resolve issues of packaging and delivery**
 - Based on LXC, Cgroups, and Namespace
 - Process-level hardware resources isolation
- Operations eventually require platform-level orchestration utilities
 - Apache Mesos: Marathon (2013) offers large-scale cluster management
 - Docker Swarm (2014) uses Docker APIs for container orchestration
 - Kubernetes initiated by Google in 2014 and releases in 2015 rescues CoreOS (a major competitor of Docker) and RedHat (early contributor of Docker) in the container market



Serverless (2015-today)

- Open Container Initiative (OCI)
 - Container image spec and runtime spec
- Cloud Native Computing Foundation (CNCF)
 - Cloud native standardizing incubating applications and best practices of creating cloud native applications
- Serverless != No server
 - is an ideology for eliminating hardware and operation details
 - Cloud Native is a set of standards and infrastructures to achieve serverless
 - Today: Serverless \approx Container Runtime (e.g., Docker) + Kubernetes

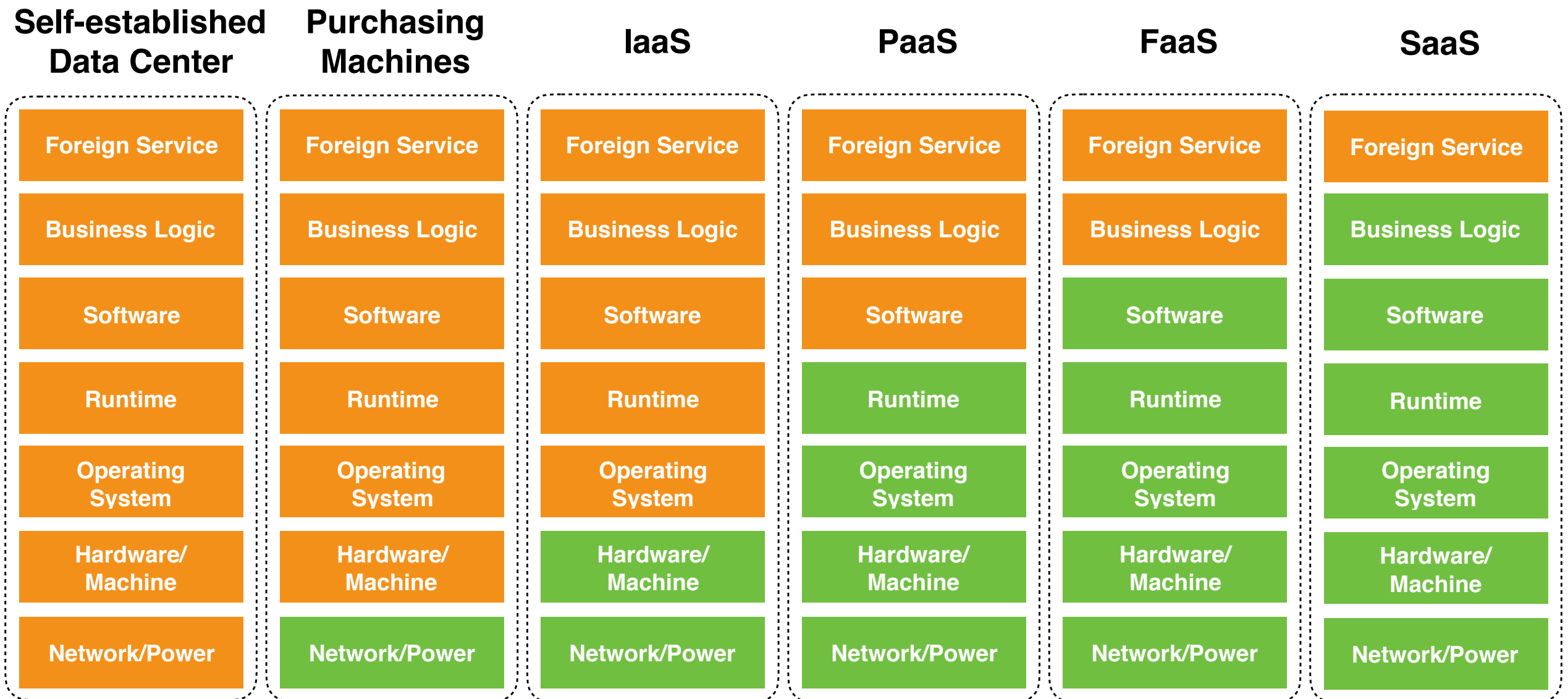


Source: Wikipedia

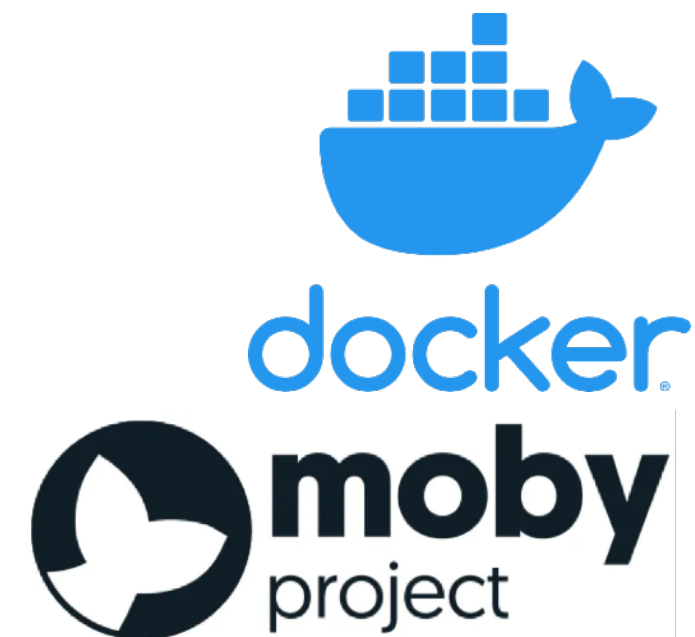
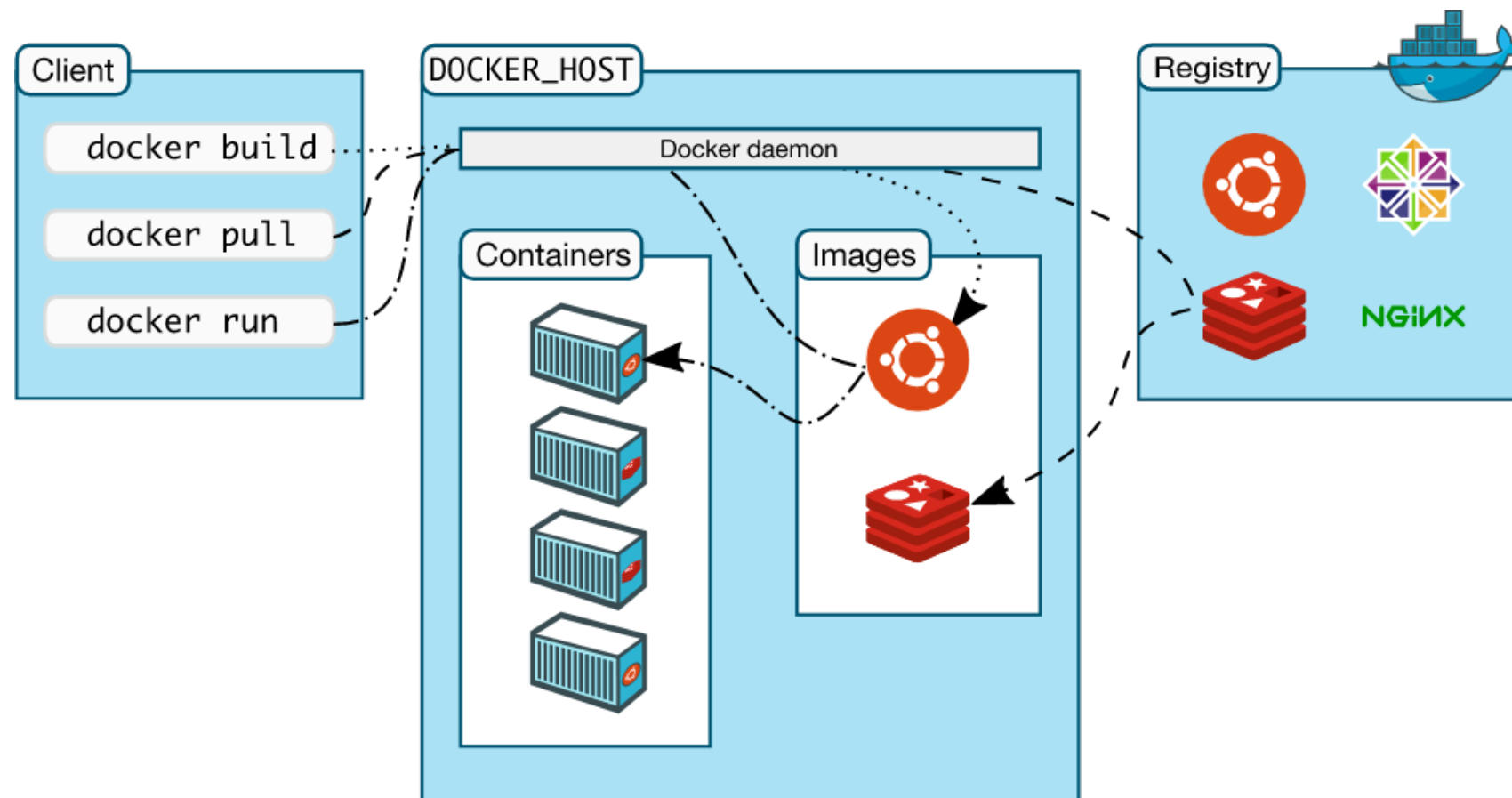
Cloud Computing Terminologies



- Building a data center is prohibitively expensive
- Computing resource business is feasible



Docker (now Moby) Core Concept and Architecture



Source: <https://docs.docker.com/engine/docker-overview/>

The Rise and Fall of Docker, Inc. (former dotCloud)

- 2013 - A PaaS startup *dotCloud* open sourced their product *Docker*
 - Gathering developers and building community shapes its early success
 - Changed the company to Docker and branded the name of Docker
- 06/2014 - Google announced the Kubernetes project
- 12/2014 - Docker announced Docker Swarm project
 - 250 Million investments from Goldman Sachs, Greylock Partners, Sequoia Capital, etc.
- 06/2015 - Docker, CoreOS, Google, and RedHat initiated OCI
 - Docker donated *libcontainer* as *RunC* for container standardization
- 07/2015 - Kubernetes 1.0 release, Google & Linux Foundation launched CNCF
- 2016 - Docker, Inc. accounted for the abandonment of Docker Swarm
- 2017 - Rename *Docker* project to *Moby* at *Dockercon17*
 - Docker announce Kubernetes support
- 2018 - Solomon Hykes (the CTO of Docker) announces his resignation

Discussion

- What did you learn from the rise and fall of Docker Inc.?
 - Think about the balance of building a successful product and make profits
 - Think about the developer community
- Where should Container-as-a-Service (CaaS) be placed in:
 - IaaS > PaaS > FaaS > SaaS

Kubernetes



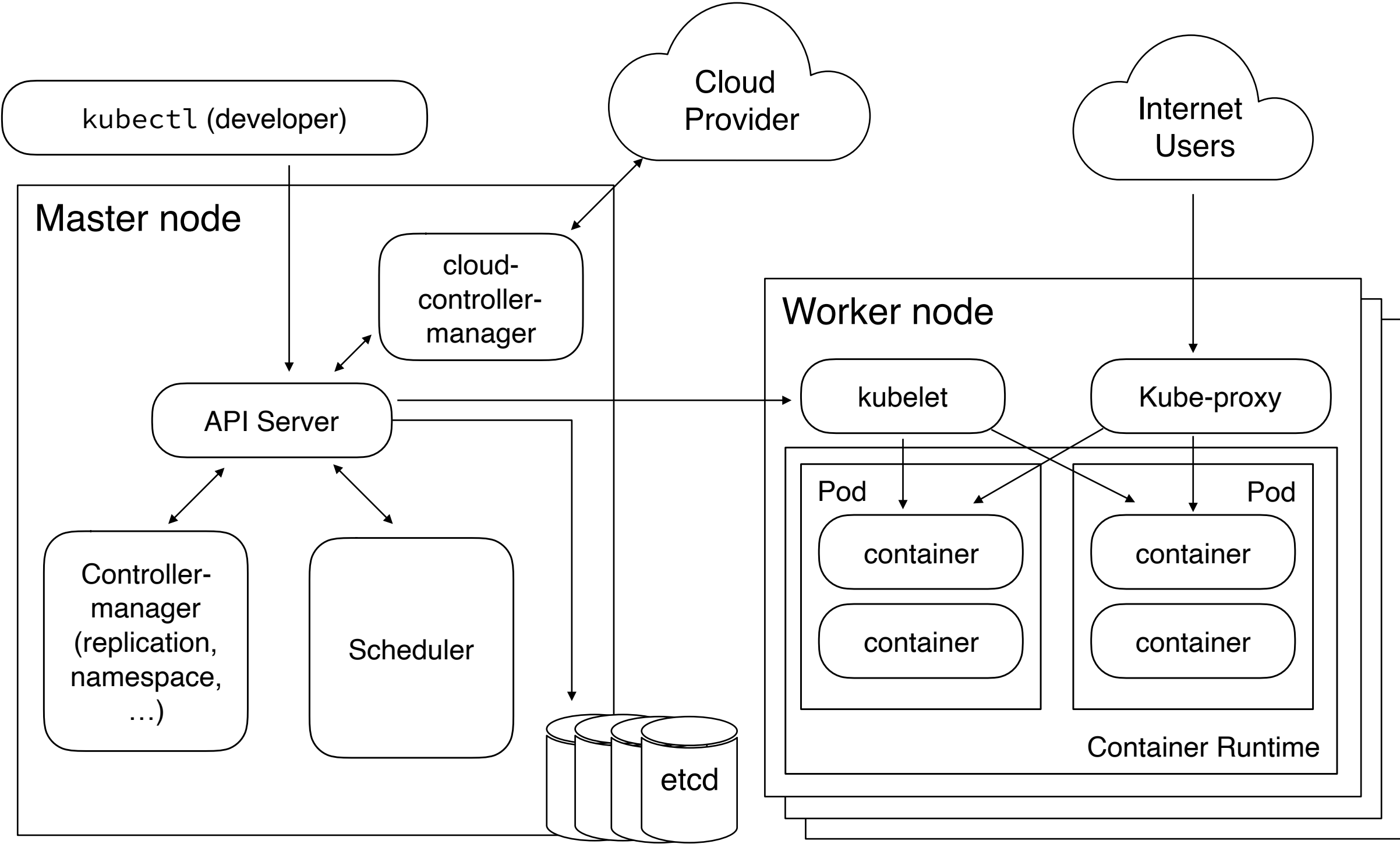
- *Kubernetes* (greek for governor, helmsman, captain)
 - Open-source container orchestration system
 - Originally designed by Google, maintained by CNCF since 1.0 release
 - Aim to provide “platform for automating deployment, scaling and operations of application containers across clusters of hosts
- Declarative YAML-based configuration
 - `kubectl apply -f deployment.yaml`

Kubernetes Core Concepts

- Pod
 - The smallest deployable object in Kubernetes
 - Encapsulates multiple application's containers, storage resources, a unique network IP, and options that govern how the containers should run
- Controllers
 - Control loop
 - » for { if actual state != desired state then do orchestrate }
 - » The desired state is defined in a YAML configuration file
 - Kind: Deployments
 - » horizontal scaling (e.g., rolling update)

```
apiVersion: apps/v1 deployment.yaml
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  selector:
    matchLabels:
      app: myapp
  replicas: 2
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myapp:latest
        ports:
        - containerPort: 80
```


Kubernetes Architecture



<https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture/architecture.md>

Cloud Native Computing Foundation (CNCF)

Cloud native technologies *empower* organizations to *build and run scalable applications* in modern, dynamic environments such as public, private, and hybrid clouds. **Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.**

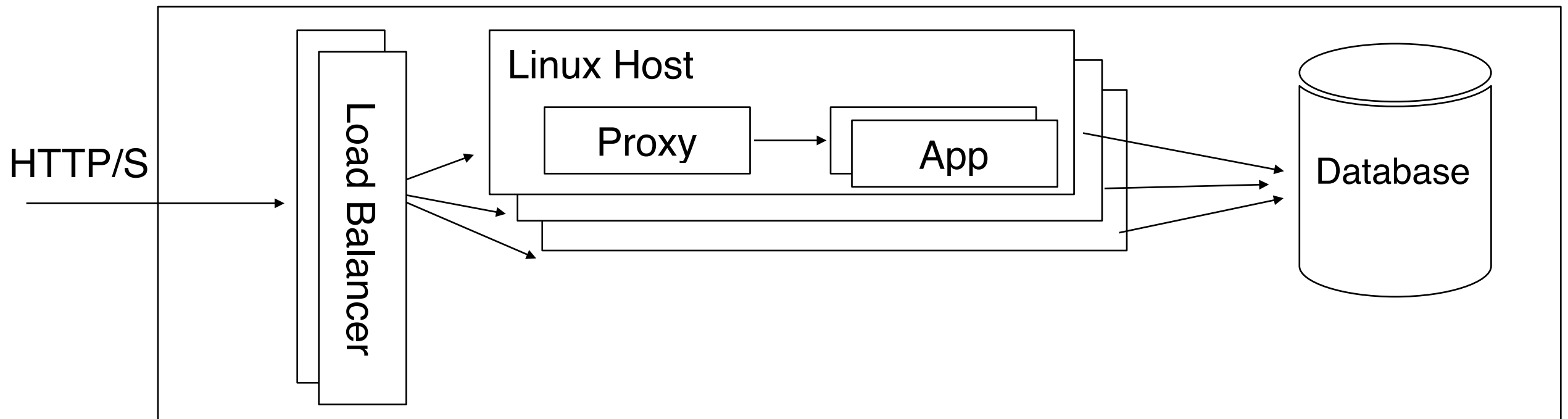
CNCF Cloud Native Definition v1.0 <https://github.com/cncf/toc/blob/master/DEFINITION.md>

- Is a Linux Foundation project
 - Linux Foundation was founded by non-profit Open Source Development Labs (OSDL) and Free Standards Group (FSG)
- Announced with Kubernetes 1.0 in 2015
 - Operational control handed over to the community in 2018
- Hosts critical components of the global technology infrastructure
 - Microservices architecture!



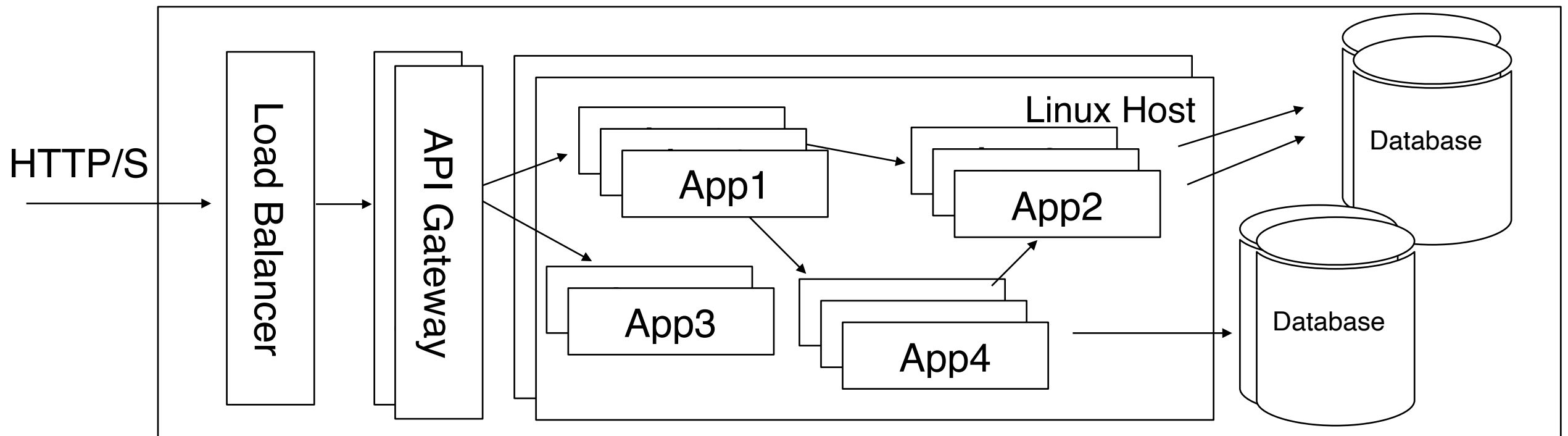
Monolith Architecture

- Monolithic code base: contributes to a single big codebase
- Monolithic database and everything tightly coupled architecture
 - Massive conflicts
 - Crash at once
 - Sticky connections



Microservice Architecture

- “...the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.” — Martin Fowler
- Separation of concerns: Modularity, encapsulation
- Scalability: Horizontally scaling, workload partitioning
- Virtualization & elasticity: Automated operations, on demand provisioning



Microservice Metaphors

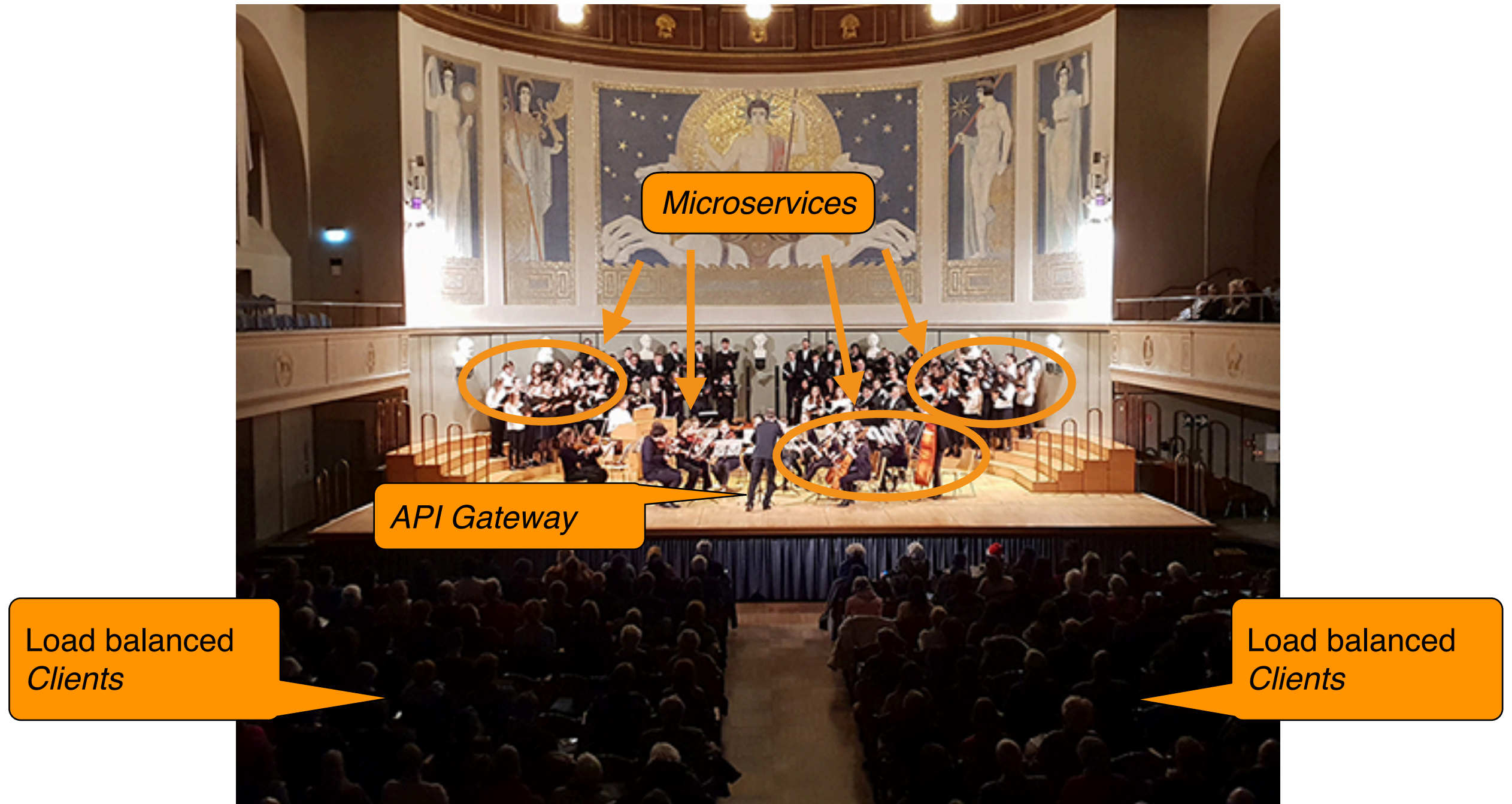
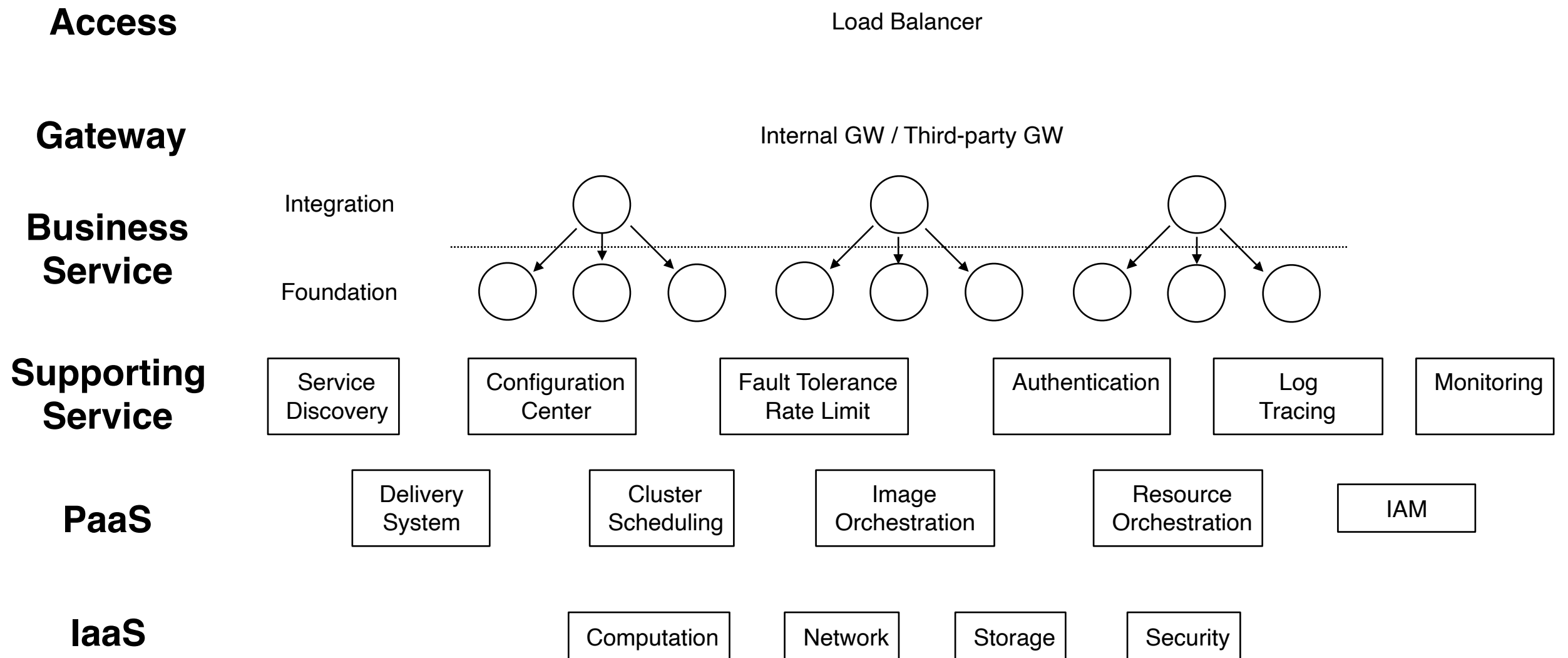

























Image taken from <https://www.musikwissenschaft.uni-muenchen.de/musikpraxis/collegium/eindruecke/index.html>

Technologies in Microservices



Microservices Governance and Technologies

- Providers   Microsoft Azure   Alibaba Cloud
- Provisioning
 - Automation & Configuration  ANSIBLE  KubeEdge
 - Container Registry  DOCKER  Google Container Registry  Amazon ECR
 - Security & Compliance  alcide
 - Key Management  Teleport  ORY / Hydra
- Runtime
 - Storage  ALLUXIO  ioceph
 - Container runtime  containerd  runc  gVisor
 - Network  CNI  OvS Open vSwitch
- Observability and Analysis
 - Monitoring  Prometheus
 - Logging  elastic  logstash
 - Tracing  JAEGER

Microservices Governance and Technologies

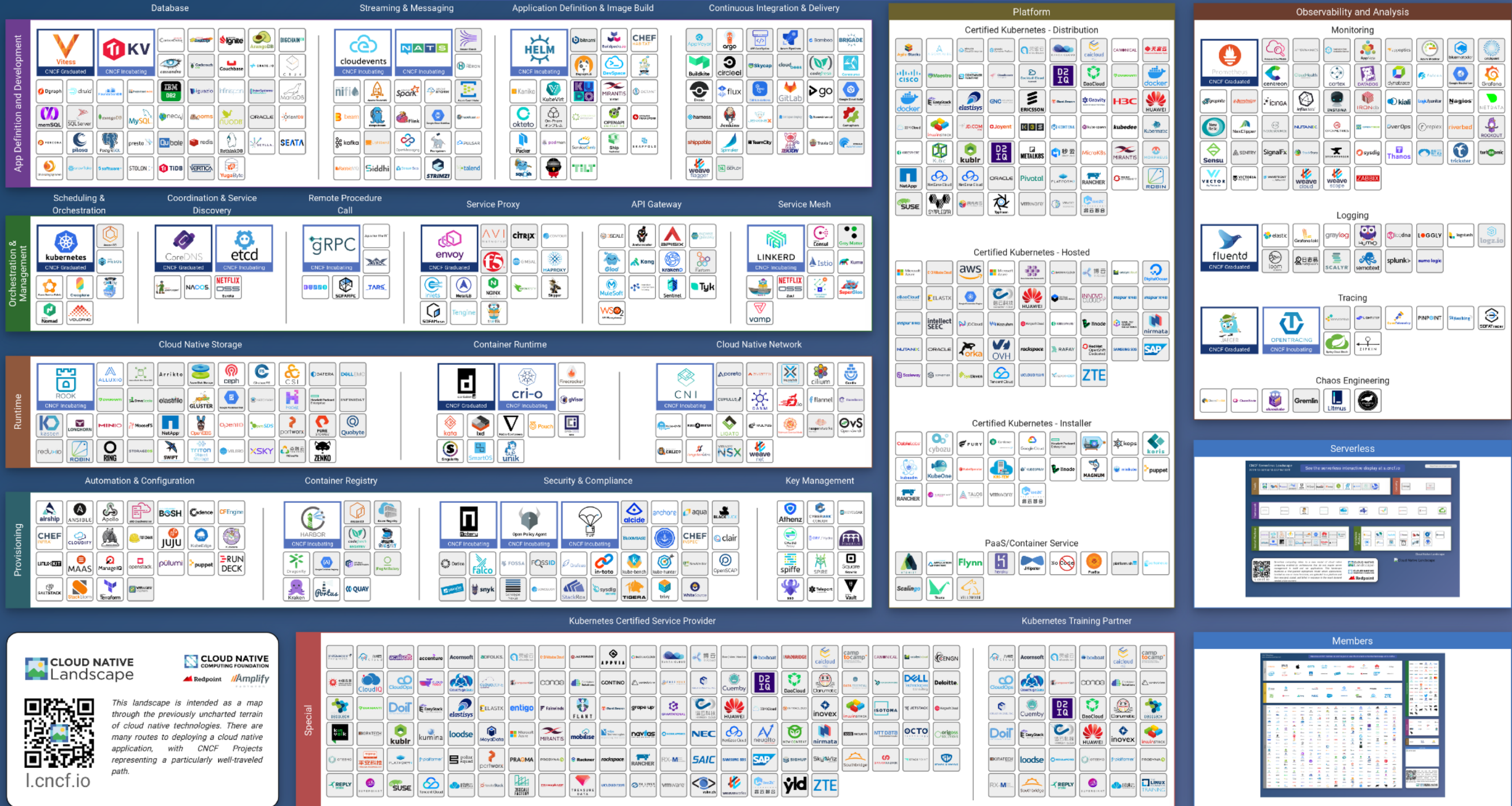
- Orchestration
 - Scheduling  **kubernetes** 
 - Coordination & service discovery  **etcd**  **CoreDNS**  **APACHE ZooKeeper™**
 - PRC  **gRPC**  **NGINX**  **traefik**
 - Service proxy  **Gloo**  **krakend**
 - API gateway  **Istio**  **Consul**
 - Service mesh
- Developments
 - Database  **KV**  **redis**  **mongoDB**
 - Streaming  **APACHE Spark™**  **Flink**  **kafka**
 - Image build  **HELM**  **podman**
 - continues integration & delivery  **Jenkins**  **Travis CI** 
- Front-end   

CNCF Cloud Native Landscape

CNCF Cloud Native Landscape
2019-12-04T04:10:42Z 96133ff

Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io

Greyed logos are not open source



CLOUD NATIVE Landscape

CLOUD NATIVE COMPUTING FOUNDATION
Redpoint Amplify



l.cncf.io

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.

Special

Discussion

- Does microservice always better than monolithic?
 - Think about building your personal website with microservice architecture
- When do you want to choose microservices architecture?

9 Technological Outlook

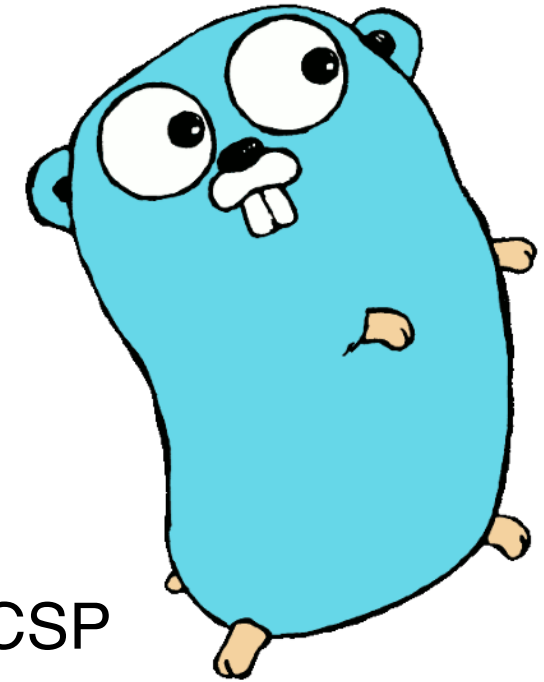
- 9.1 Web Application on Bare Metal: WebAssembly
- 9.2 Container Orchestration and Cloud Native
- 9.3 Other Selected Trends: Go, HTTP/3
- 9.4 Conclusion

Literature:

<https://golang.org>

<https://quicwg.org/base-drafts/draft-ietf-quic-http.html>

Go



- Open source programming language
 - Creators: Rob Pike, Ken Thompson, Robert Griesemer
 - Started almost simultaneously with V8
 - Start from C, inspired by Pascal family and Tony Hoare's CSP
- Key features: Simple, stable, fast compilation, built-in concurrency
 - 25 keywords, stable for 10 years since Go 1 releases
 - No cycle import, no function override
 - *Goroutines* as lightweight threads
 - *Channel* philosophy "Do not communicate by sharing memory, share memory by communicating"
- Support cross compilation and package modularization
- Must be formatted to pass compilation, only one style of coding
- Is de facto the language of cloud computing at present:
 - Kubernetes, etcd, Prometheus, Docker... are implemented by Go

Why Go? An Oversimplified Version

- Before Go

- C and Unix became dominant in research

- The desire for a higher-level language led to C++, e.g.

```
for(map<string, pair<string, string> >::const_iterator  
iter = p.begin(); iter != p.end(); ++p)
```

- C++ became the language of choice in parts of industry and in many research universities.

- Java arose as a clearer stripped-down C++

- By the late 1990s, a teaching language was needed that seemed relevant, and Java was chosen.

- C++03 brings more complex features, e.g.

```
for(const auto&& val: p)
```



<https://spf13.com/presentation/the-legacy-of-go/>

Why Go? Sophistication or Level of Abstraction

“Any given function template specialization F1 is eliminated if the set contains a second function template specialization whose function template is more specialized than the function template of F1 according to the partial ordering rules of 17.6.6.2. After such eliminations, if any, there shall remain exactly one selected function.” — *Working Draft, Standard for Programming Language C++ 16.4 Address of overloaded function*

- *“The reason I was enthusiastic about Go is because, at the same time we were starting on Go, I tried to read the C++ 0x proposed standard, that was the convincer for me.” — Ken Thompson*
- *“The code is harder to understand simply because it is using a more complex language” — Rob Pike*
- *“In Go (compare to C++), we’re trying to do a completely different approach, to take things out as much as we can, to reduce them to the bare bones, the absolute minimum that you need to build everything up.” — Robert Griesemer*

Go Design: Concurrency

- Concurrency is the ability to write your program as independently executing pieces. In Go, concurrency has three elements:
 - Grouting (execution): light-weight threads
 - » `go function(args)`
 - Channels (communication): Message passing and synchronization
 - » Send message: `ch <- value`
 - » Receive message: `dst := <- ch`
 - Select (coordination): managing channels concurrently
 - » `select {`
 - `case value := <- ch1: ...`
 - `case ch2 <- value: ...`
 - `}`

Example: A High Performance HTTP Server

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

func こんにちは_Gophers(w http.ResponseWriter, req *http.Request) {
    fmt.Fprintf(w, "こんにちは Gophers!\n")
}

func main() {
    http.HandleFunc("/", こんにちは_Gophers)
    err := http.ListenAndServe("localhost:12345", nil)
    if err != nil {
        log.Fatal("ListenAndServe: ", err)
    }
}
```

main.go

Packages

UTF8 by default

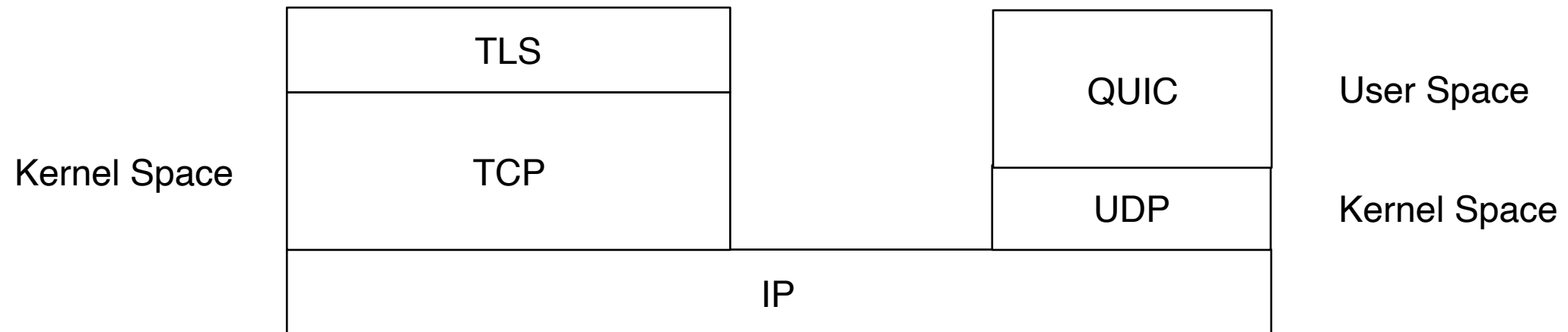
Fprintf direct to network connection

Truly concurrent and production ready

Error handling

HTTP/3

- Is the upcoming third major version of Hypertext Transfer Protocol
- Draft based on Request on Comments (RFC) draft, named “HTTP over QUIC, user space congestion control is used over UDP



- No public supports yet
 - Available on Chrome and Firefox latest beta

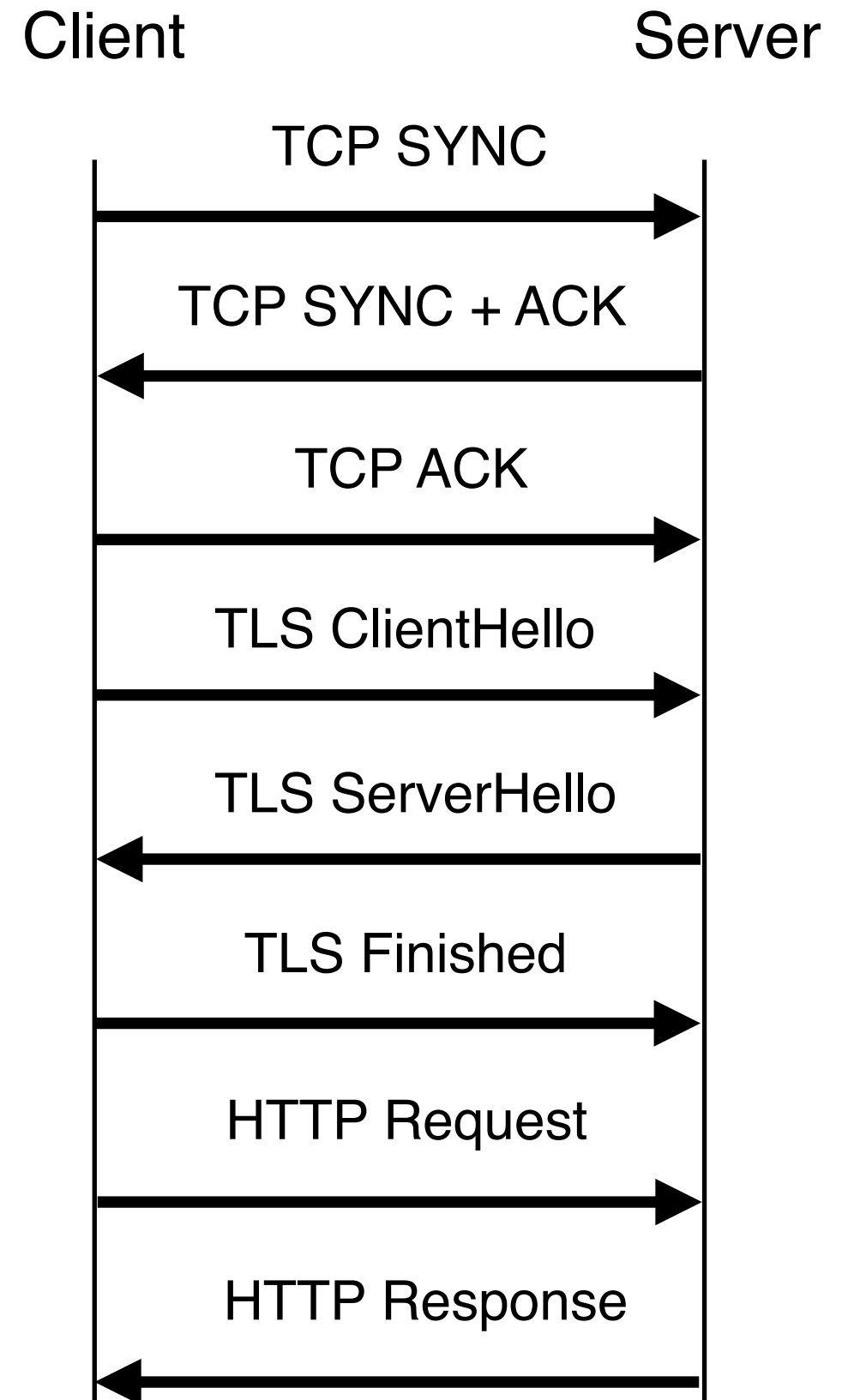
The screenshot shows the 'HTTP/3 protocol' status page. It includes a table of browser support across various platforms. The table has columns for browser names and rows for different operating systems. The data is as follows:

Browser	IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baic Brow
Current aligned	6-10	12-17	2-70	4-77	3.1-12.1	10-63	3.2-13.1		2.1-4.4.4	12-12.1				4-9.2		
Usage relative	11	18	71	78	13	64	13.2	all	76	46	78	68	12.12	10.1	1.2	7.1
Date relative		76	72-73	79-81	TP		13.3									

<https://caniuse.com/#feat=http3>

Evolution of HTTP

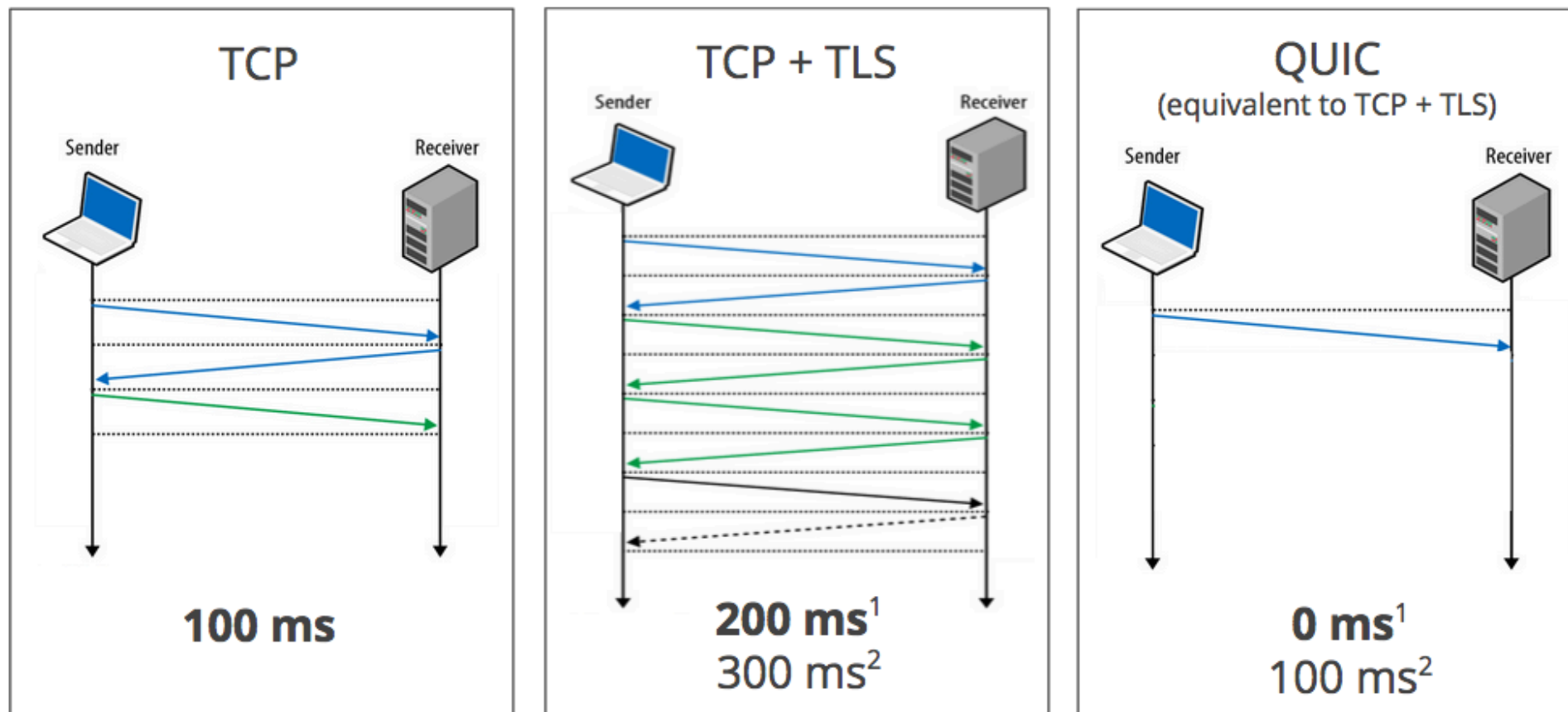
- HTTP/0.9 (1991)
- HTTP/1.0 (1996)
 - TCP connection is created for each request/response exchange between clients
 - All requests incur a latency penalty
- HTTP/1.1 (1997)
 - “keep-alive” connections that allow clients to reuse TCP connections
- HTTP/2.0 (2015)
 - Allow concurrently multiplex different HTTP exchanges onto the same TCP connection
- HTTP/3.0 (2018)



HTTP/3.0

- How communication is processed between two persons?

Zero RTT Connection Establishment



1. Repeat connection
2. Never talked to server before

<https://blog.chromium.org/2015/04/a-quick-update-on-googles-experimental.html>

9 Technological Outlook

- 9.1 Web Application on Bare Metal: WebAssembly
- 9.2 Container Orchestration and Cloud Native
- 9.3 Other Selected Trends: Go, HTTP/3
- 9.4 Conclusion

Literature:

Conway ME. How Do Committees Invent? Datamation magazine. 14(4), pp.28-31.

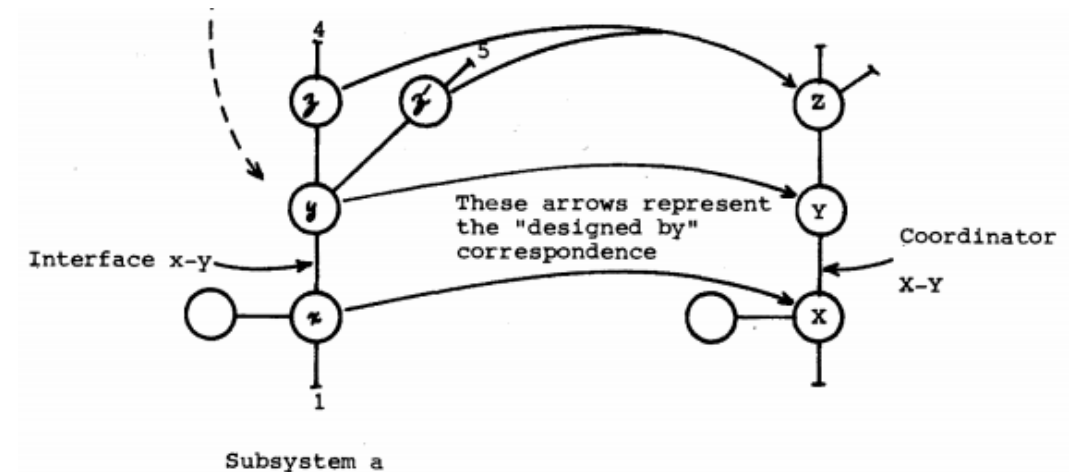
Brooks FP, No Silver Bullet. IEEE computer. 1987 Apr;20(4):10-9.

Brooks FP. The Mythical Man-Month, Anniversary ed. p. cm. 1995.

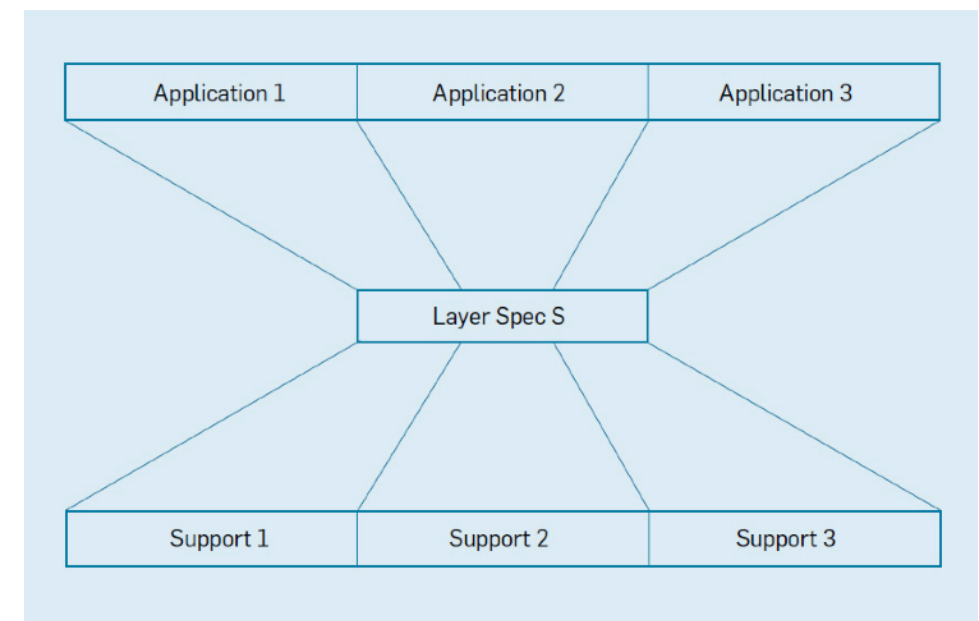
Micah Beck. 2019. On the hourglass model. Commun. ACM 62, 7 (June 2019), 48-57.

Architecture and Organizations

- Maintainability, reliability, and security are the most important (at scale)
- Monolith vs. Microservice
 - Web applications support any style
 - Stateless is the key to introduce redundancy (reliability)
 - Premature optimization is the root of all evil
- Conway's Law and Hourglass Model
 - The Conway's Law: *Organizations which design systems ... are constrained to produce designs which are copies of the communication structure of these organizations*
 - The Hourglass Model: *Logical weakness is critical to the development scalability*



Conway ME. How do committees invent. Datamation. 1968 Apr;14(4):28-31.



Micah Beck. 2019. On the hourglass model. Commun. ACM 62, 7 (June 2019), 48-57.

Take Away

- Open-source and developer community matters and are eating the world
- Future outlook:
 - Is hard to say, popularity != future
 - » But many experiences and lessons can help us make the prediction
 - Virtualization, containerization, and orchestration are hard
 - » See WebAssembly and CNCF landscape
 - Simplicity is complicated, but the clarity is worth the fight
 - » Compare JavaScript, TypeScript, C++, Rust, and also Go

Discussion (Time Permitting)

- When does a technical problem become an “organization” problem?
 - What is the general process of resolving the issue?
 - What is the root cause that technologies been revolutionized?
- How do you imagine WebAssembly changes the way of front-end developments?
 - Think about virtualization and containerization
- What could change if content distribution achieve (nearly) zero delay time?