CS6216 Advanced Topics in Machine Learning (Systems)

Cloud systems for AI

Yao LU

06 Nov 2024

National University of Singapore
School of Computing

From LLMs to the cloud

Chef
(LLM)

Restaurant
(serving systems)

Disney world
(cloud systems)

From serving to cloud systems:

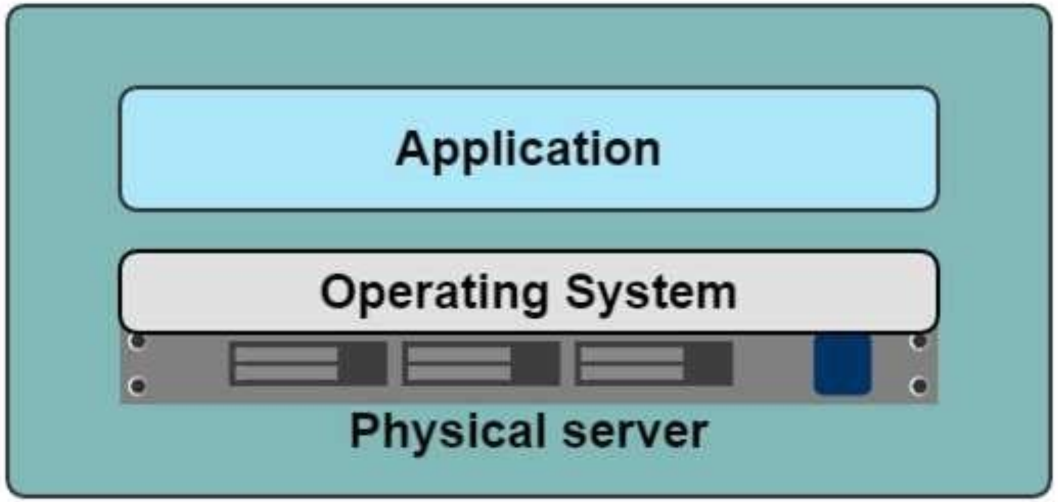
- **Multi-tenancy**: from scaling-up to scaling-out (models, users, applications, tasks etc.)
- **Operations of** large-scale, heterogeneous infrastructures

Outline

- Brief history of cloud computing
- Cloud native technologies
- Current practice and opportunities of AI on cloud

A history lesson

In the Dark Ages

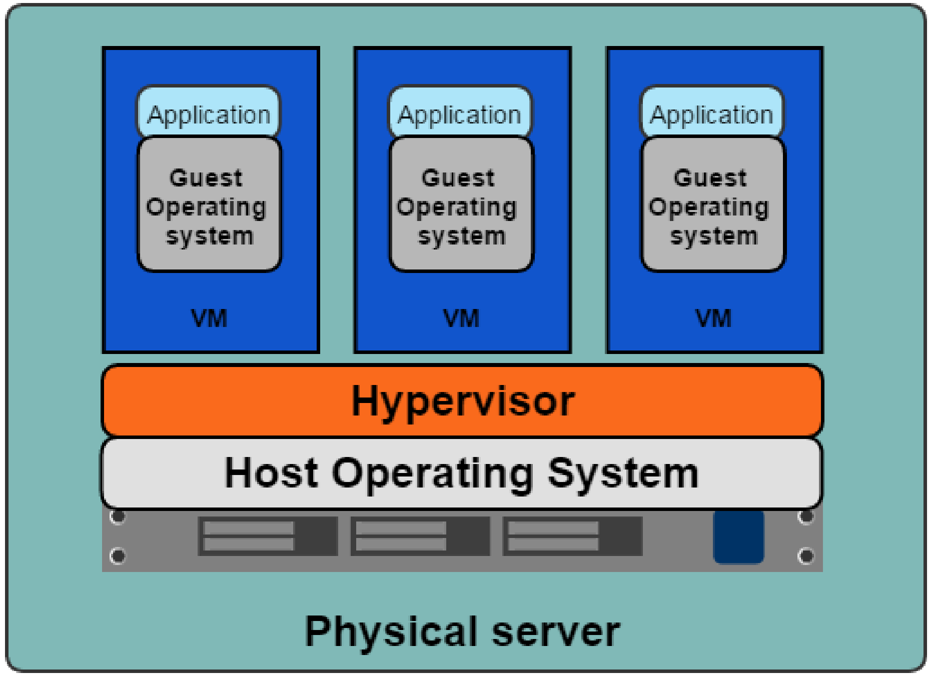


- Slow deployment times
- Huge costs & wasted resources
- Difficult to scale & migrate
- Vendor lock in

One application on one physical server

A history lesson

Hypervisor-based Virtualization



- One physical server can contain multiple applications
- Each application runs in a virtual machine (VM)

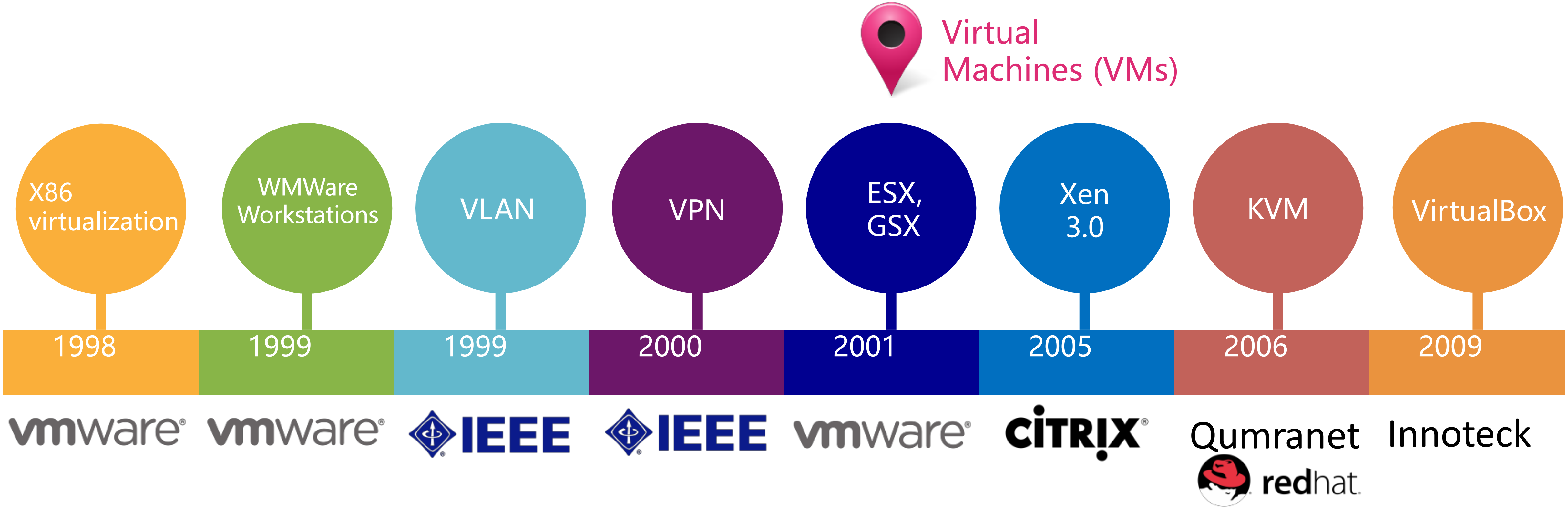
- Better resource pooling
  - One physical machine divided into multiple virtual machines
- Easier to scale
- VMs in the cloud
  - Rapid elasticity
  - Pay as you go model

Microsoft Azure

amazon web services™

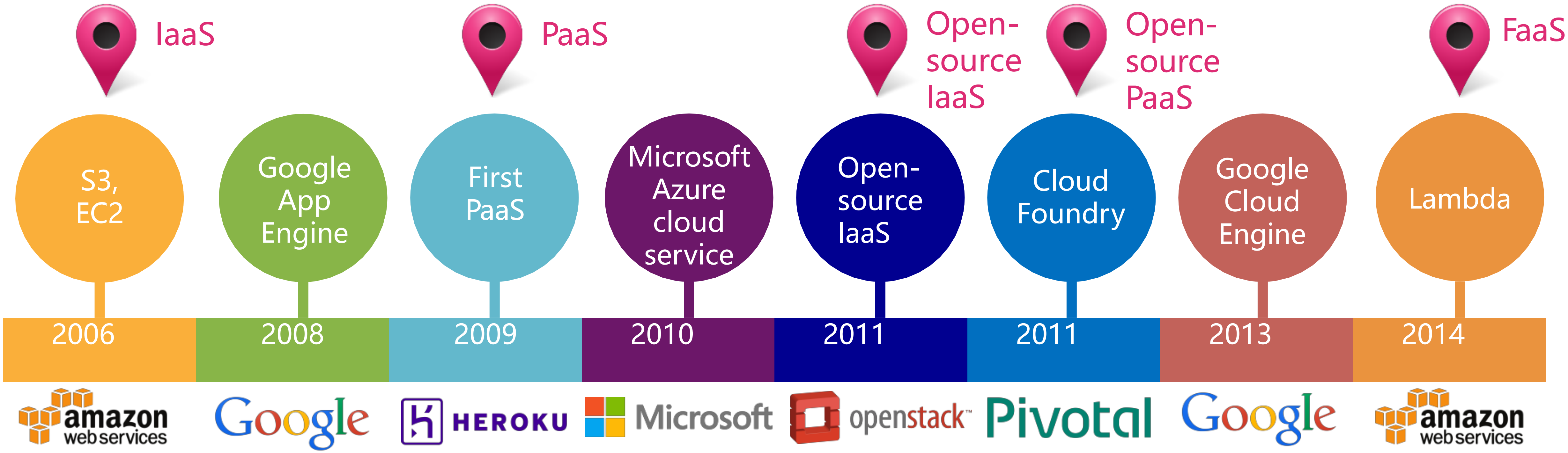
vmware®

Brief history of cloud computing



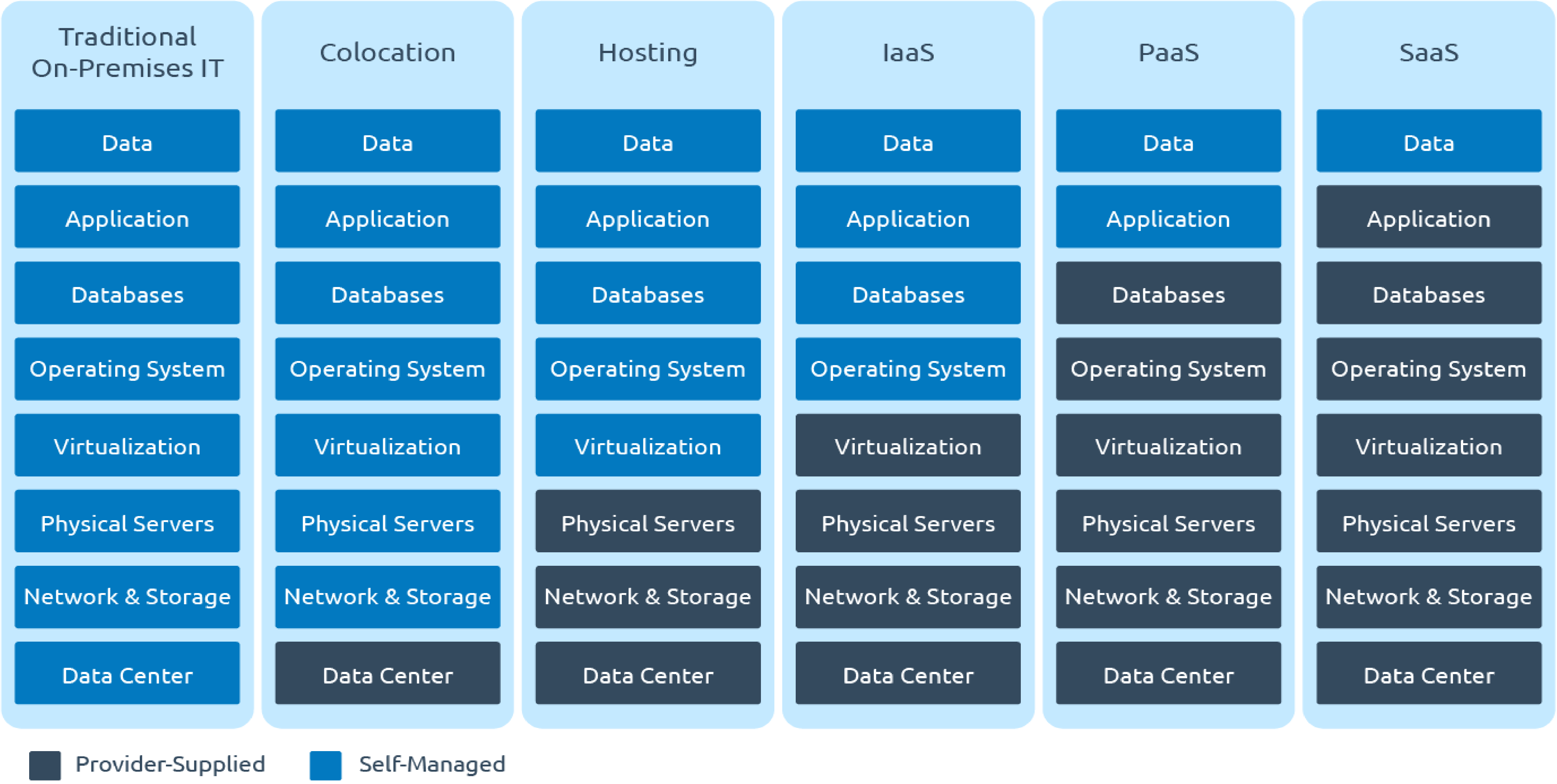
Mature of virtualization: no.1 important technology

Cloud computing offerings



	2005		2017
	$5,000,000		$5,000
2005	Team of engineers	**2017**	An engineer
	Months of development		Weeks development

Cloud computing offerings



However,

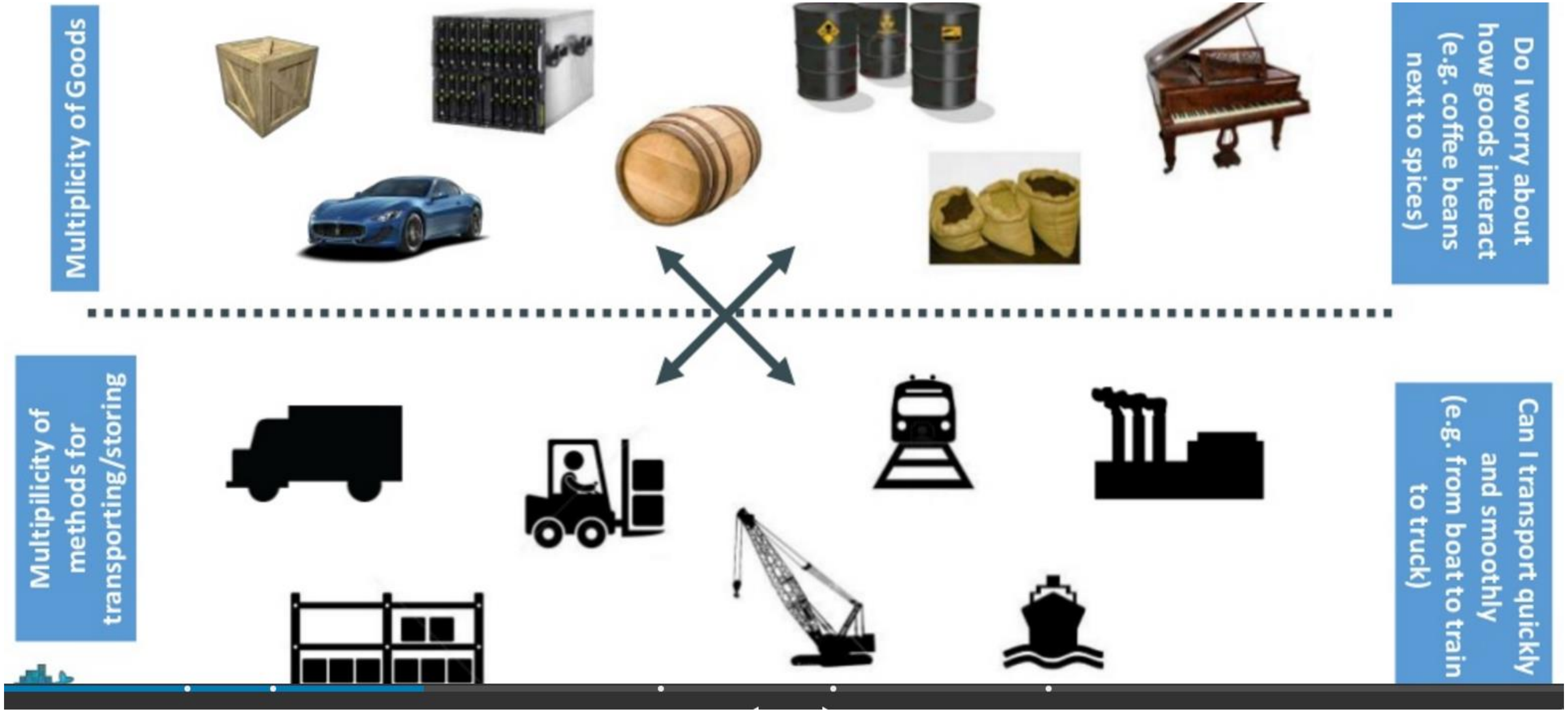
- Each VM stills requires
  - CPU allocation
  - Storage
  - RAM
  - An entire guest operating system
- The more VMs you run, the more resources you need
- Guest OS means wasted resources
- Application portability not guaranteed


Looking for all kinds of solutions…

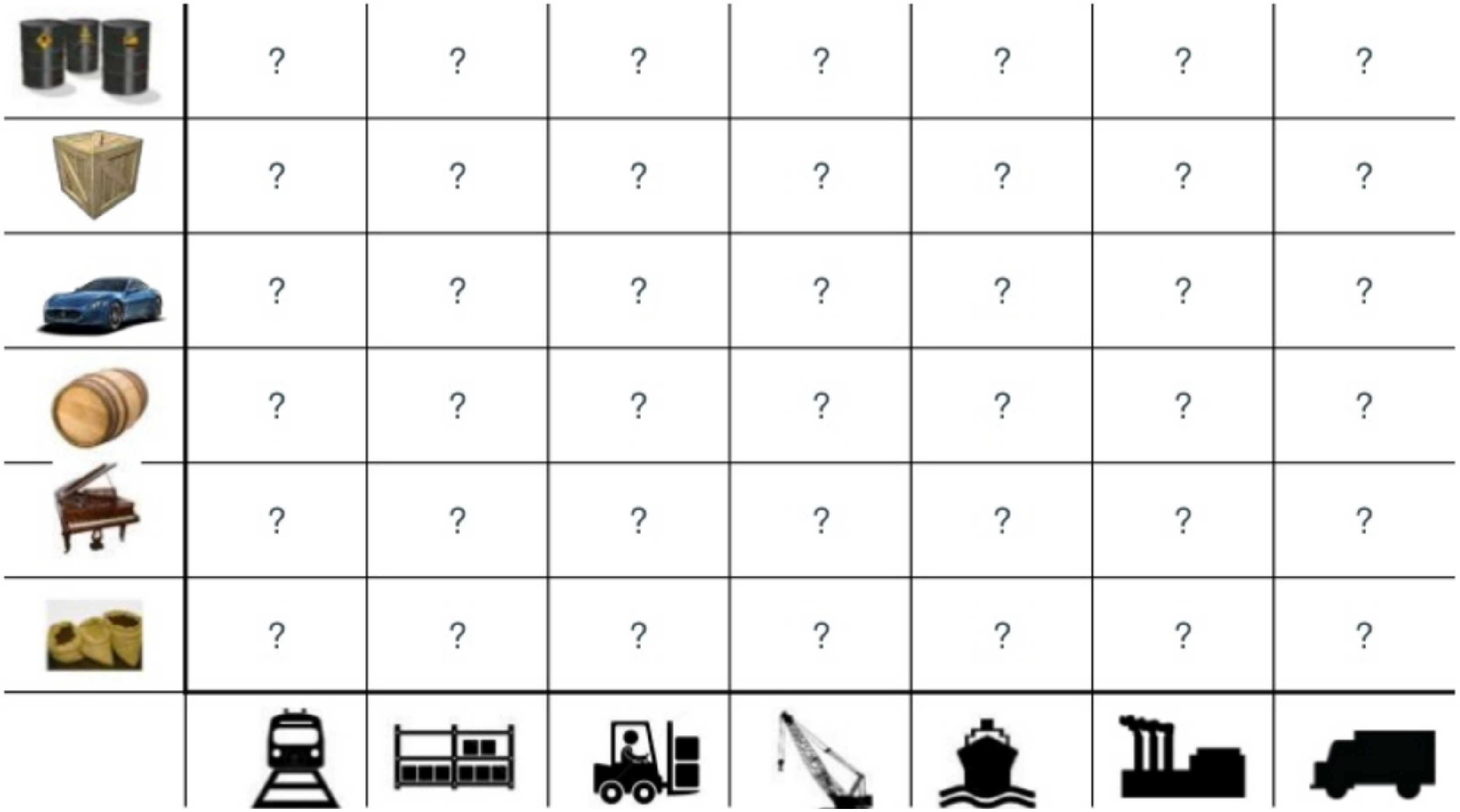
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Too many to consider

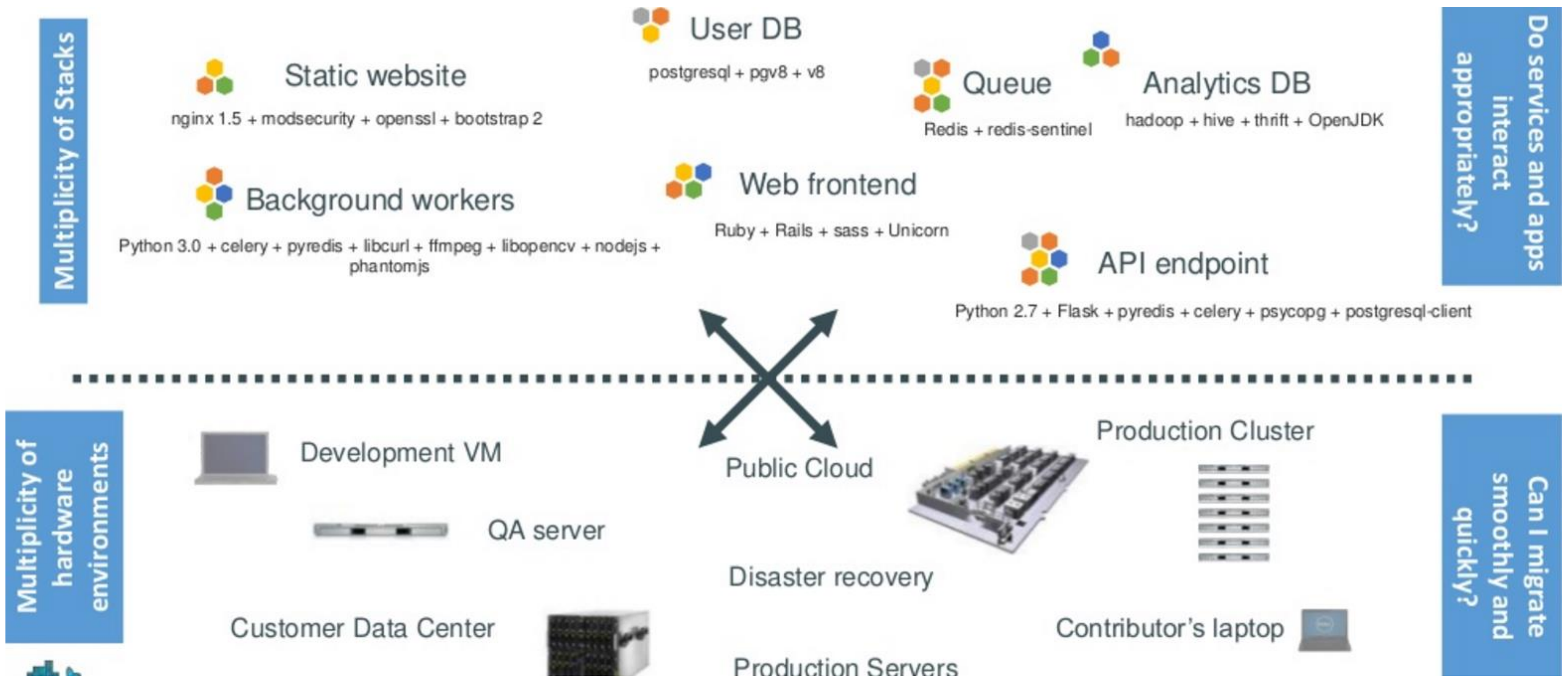
An analogy: cargo transportation



What are the possibilities

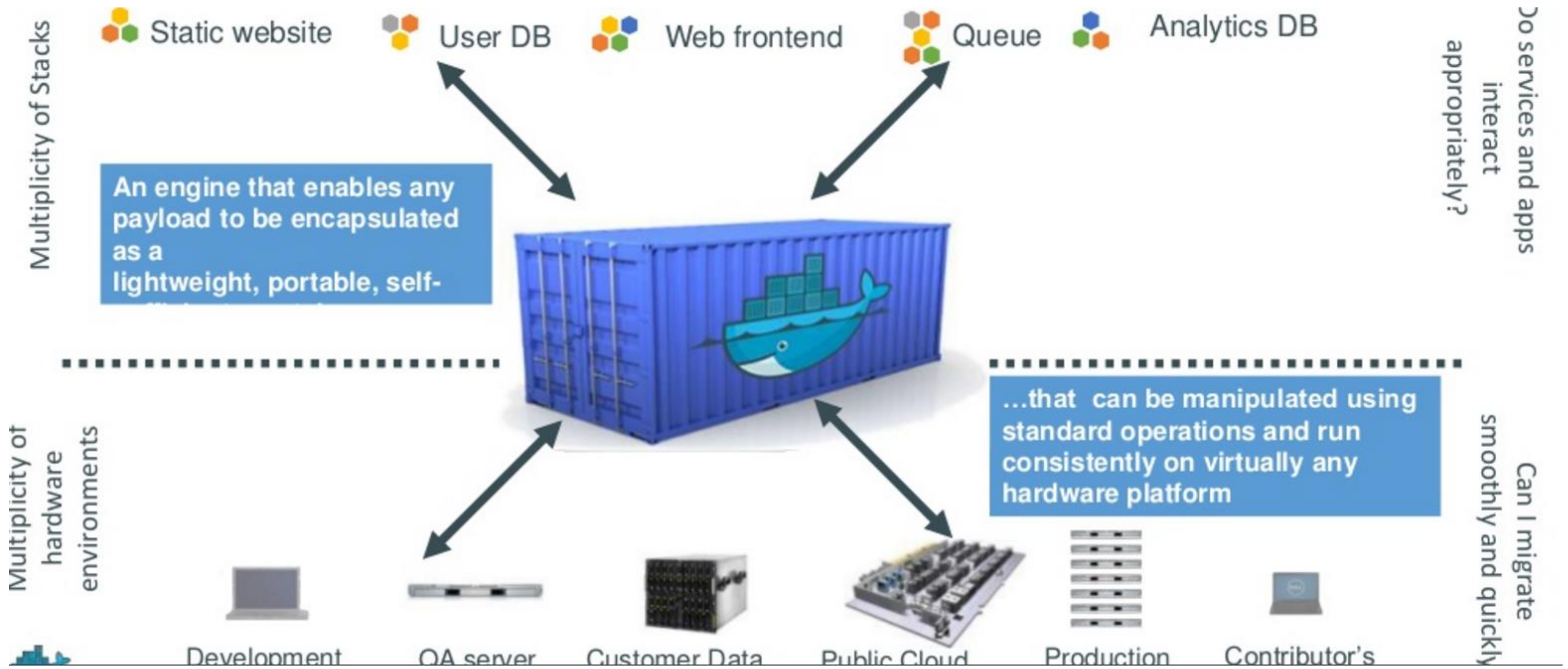


The challenge continued



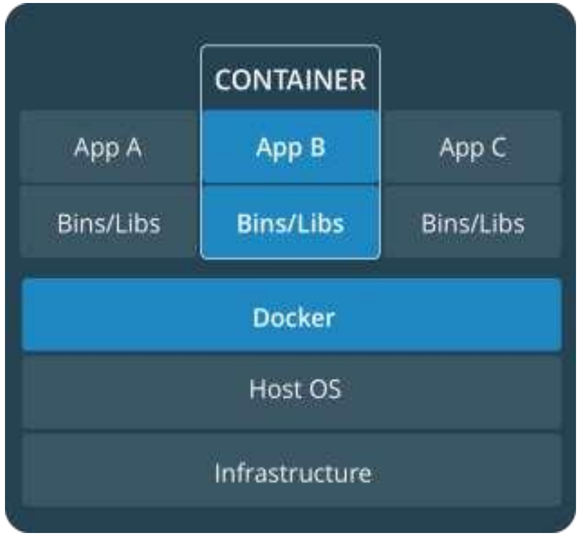
Shipping containers

Container for code?

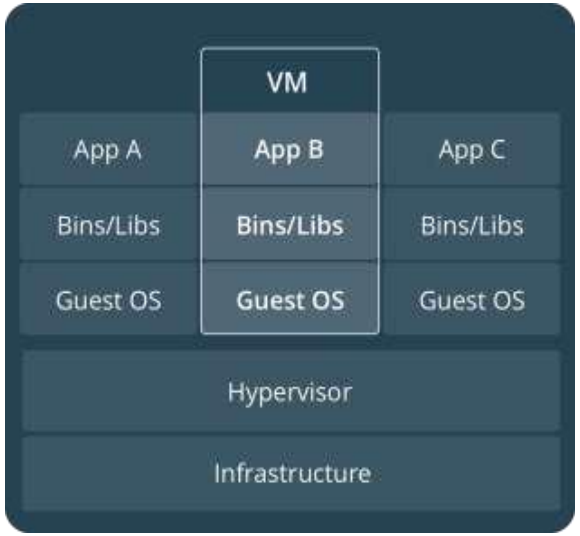


- **Speed:** share the same OS kernel. No OS to boot = applications online in seconds
- **Portability:** Standardized software packaging. Less dependencies between process layers = ability to move between infrastructure & OS
- **Efficiency:** Less OS overhead & improved VM density

Comparing containers and VMs

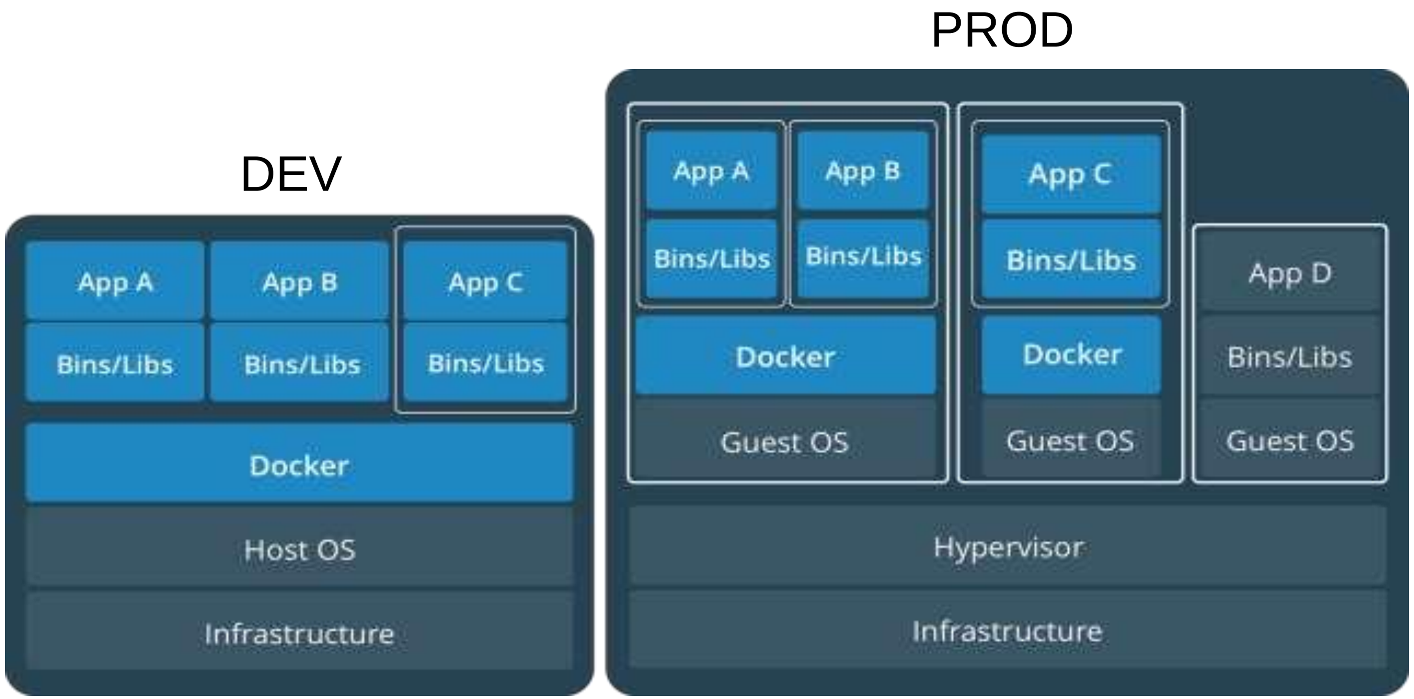


Containers are an app level construct



VMs are an infrastructure level construct to turn one machine into many servers

Containers and VMs together



Containers and VMs together provide a tremendous amount of flexibility for IT to optimally deploy and manage apps.

Cloud native technologies

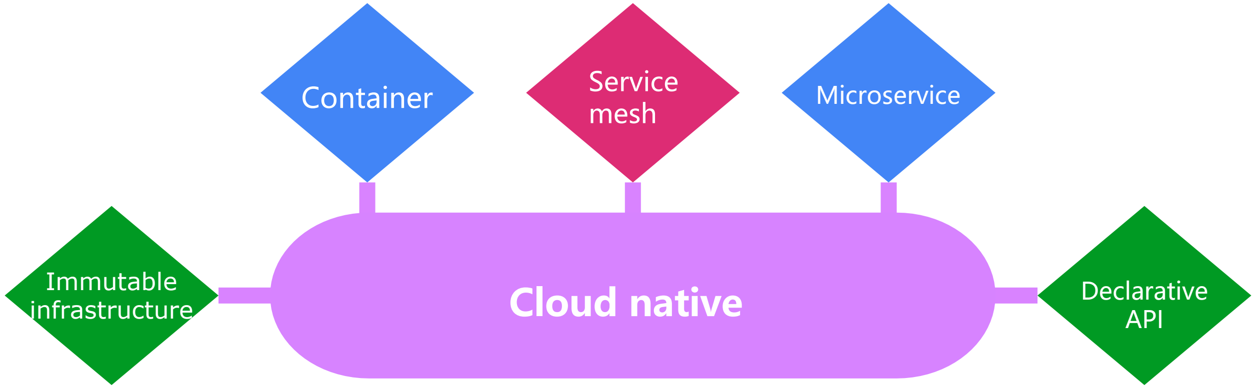
Definitions by Cloud Native Computing Foundation (CNCF) :

CLOUD NATIVE
COMPUTING FOUNDATION

- Cloud native practices empower organizations to develop, build, and deploy workloads in computing environments (public, private, hybrid cloud) to meet their organizational needs at scale in a programmatic and repeatable manner. It **is characterized by loosely coupled systems that interoperate in a manner that is secure, resilient, manageable, sustainable, and observable.**

- Cloud native technologies and architectures typically consist of some combination of containers, service meshes, multi-tenancy, microservices, immutable infrastructure, serverless, declarative APIs etc.

- Combined with robust automation, cloud native practices allow organizations to make high-impact changes frequently, predictably, with minimal toil and clear separation of concerns.



Rise of containers and Kubernetes (K8s)

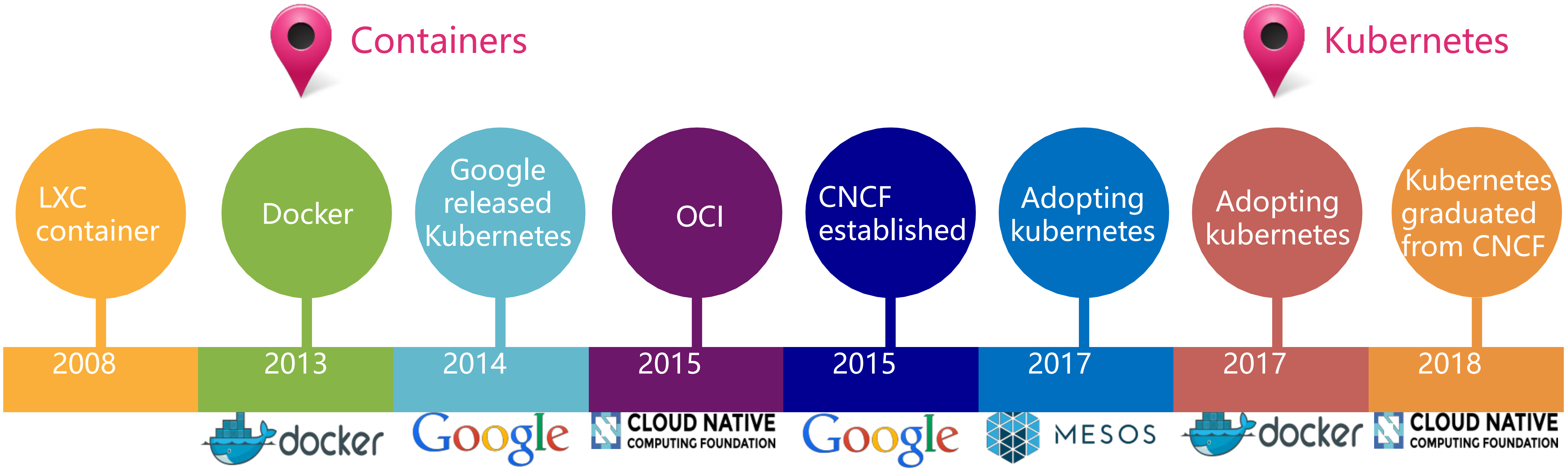
Containers

Kubernetes

- 2008 — LXC container
- 2013 — Docker
- 2014 — Google released Kubernetes
- 2015 — OCI
- 2015 — CNCF established
- 2017 — Adopting kubernetes
- 2017 — Adopting kubernetes
- 2018 — Kubernetes graduated from CNCF

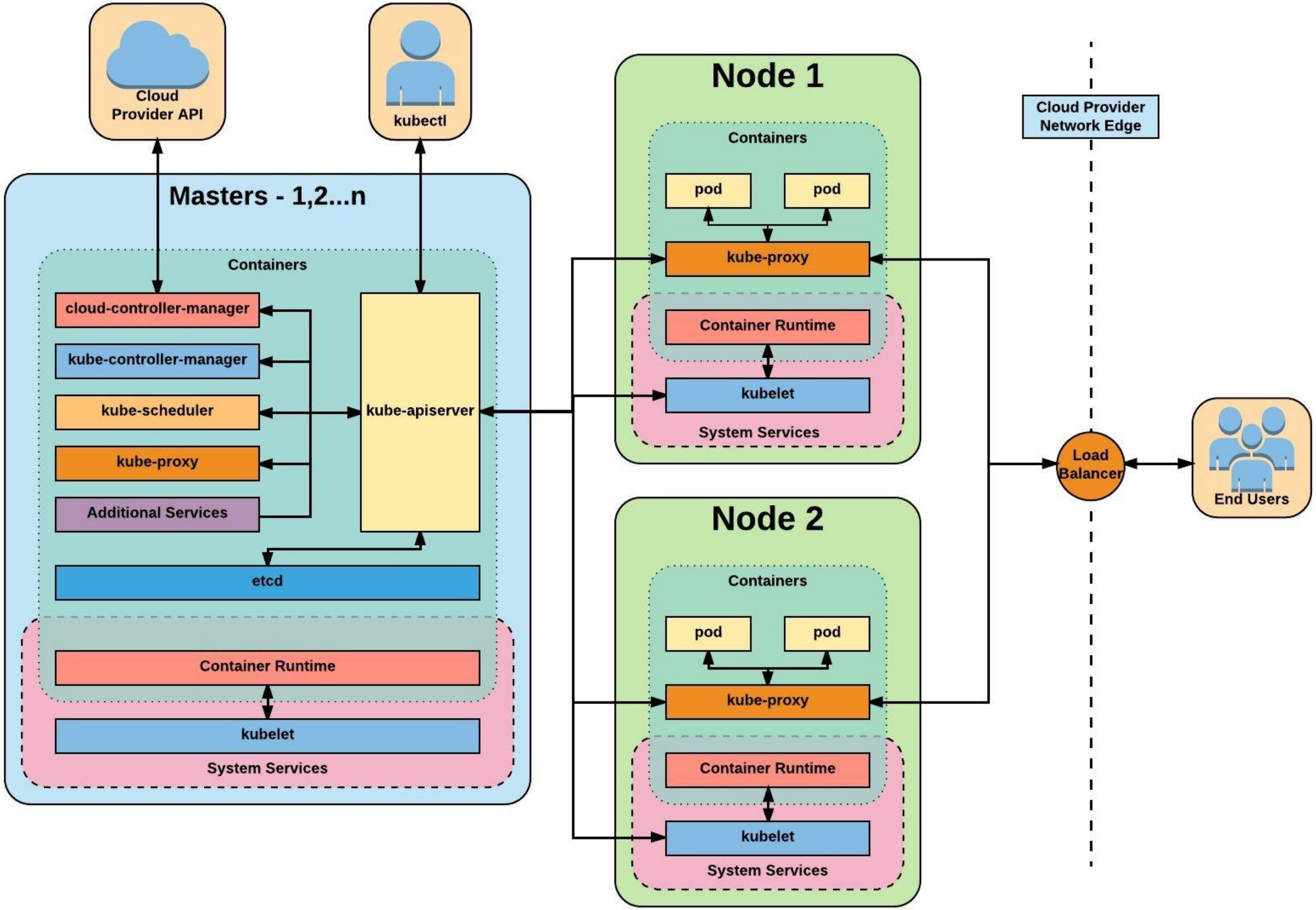
Kubernetes or **K8s** is a project spun out of Google as a open source next-gen container scheduler

Rise of containers and Kubernetes (K8s)

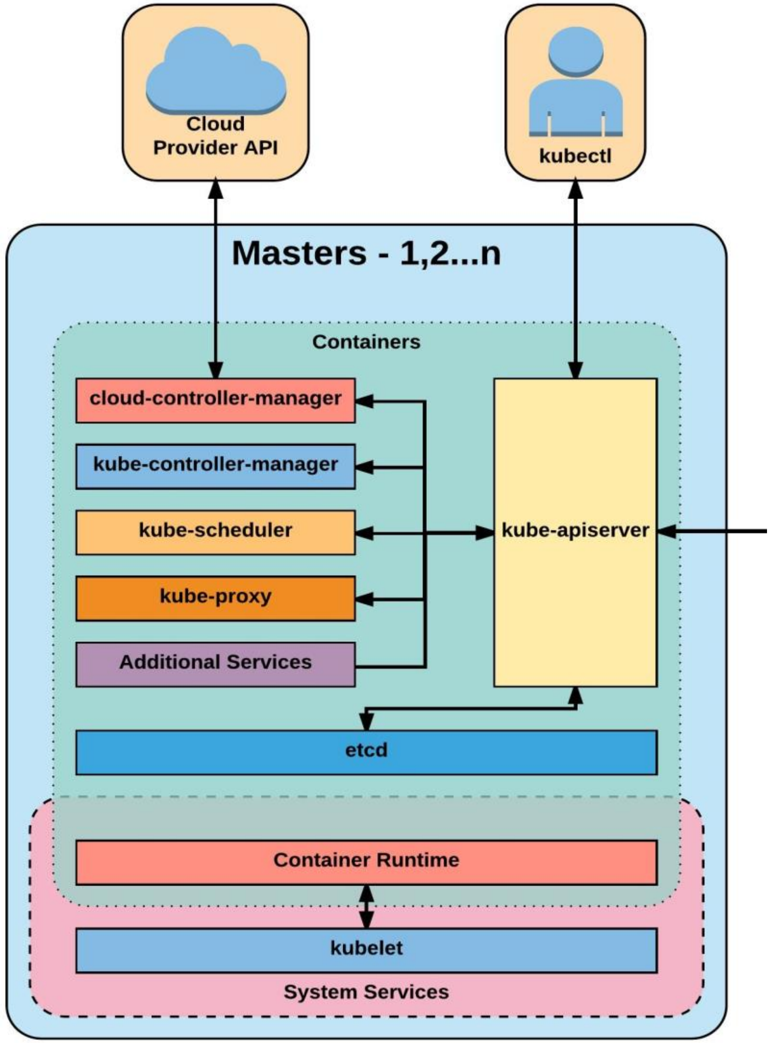


- K8s is an orchestration tool for managing distributed services or containerized applications across a distributed cluster of nodes.
- K8s follows a **client-server architecture with a master and worker nodes**. Core concepts in Kubernetes include pods, services (logical pods with a stable IP address) and deployments (a definition of the desired state for a pod or replica set).
- K8s **users define rules** for how container management should occur, and then K8s handles the rest

Architecture overview

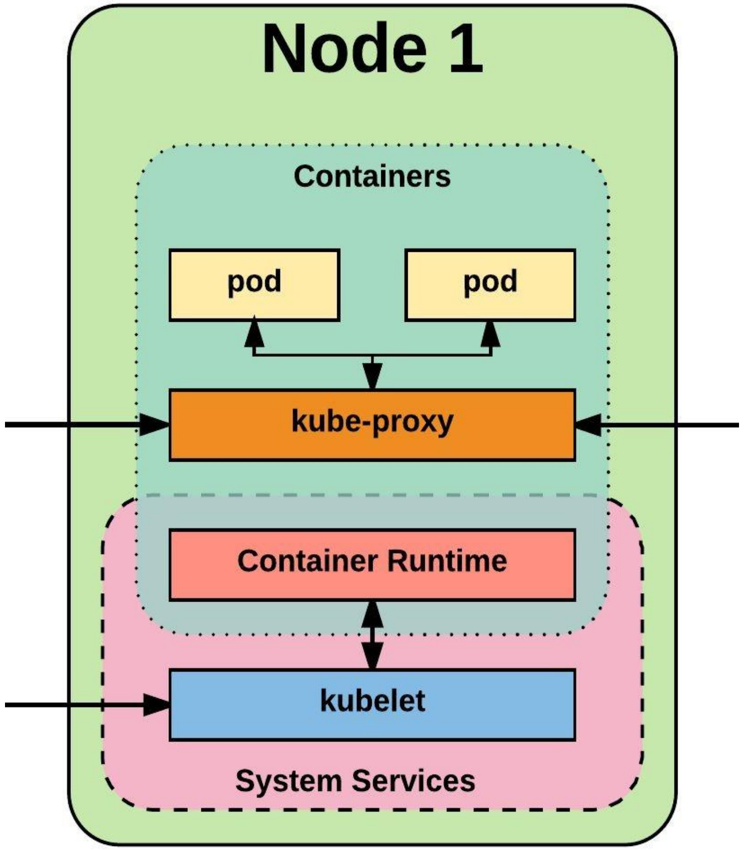


Master components



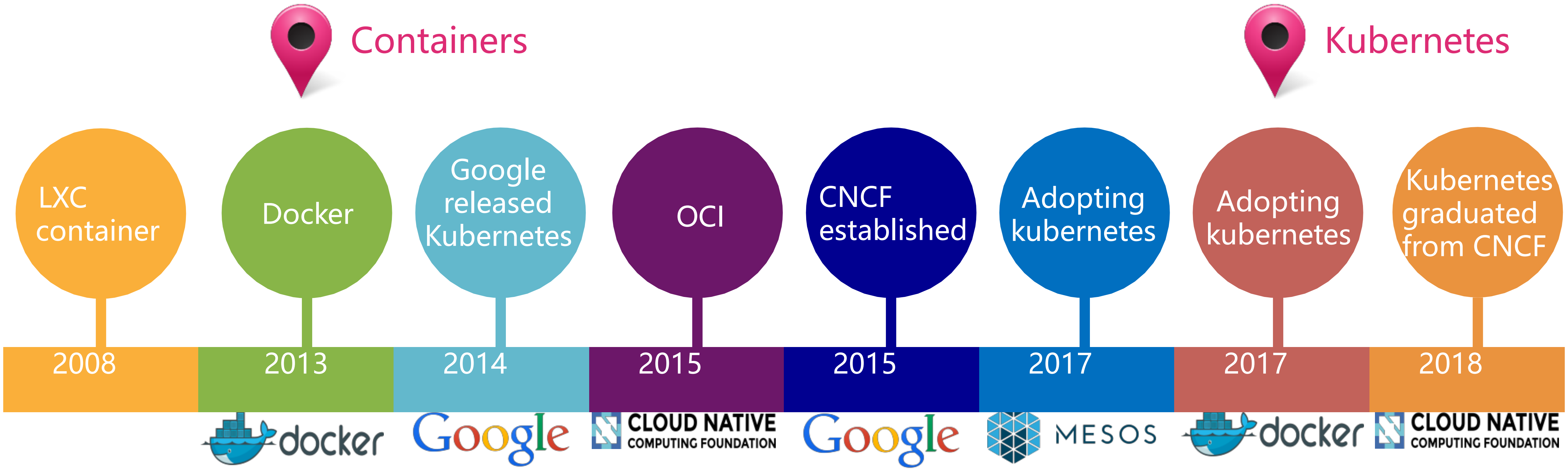
- **Kube-apiserver:** provides REST interface into the K8s control plane and datastore.
- **Etcd:** the cluster datastore; providing a strong, consistent and highly available key-value store used for persisting cluster state
- **Kube-controller-manager:** manages all core component control loops; monitors and steers the cluster towards the desired state.
- **Cloud-controller-manager:** provides cloud-provider specific knowledge and integration capability.
- **Kube-scheduler:** evaluates workload resource requirements and place it on a matching resource.

Node components



- **kubelet:** node agent for managing pod lifecycle on its host.
- **kube-proxy:** managing the network rules on each node and performs connection forwarding or load balancing.
- **container runtime:** executes and manages containers.

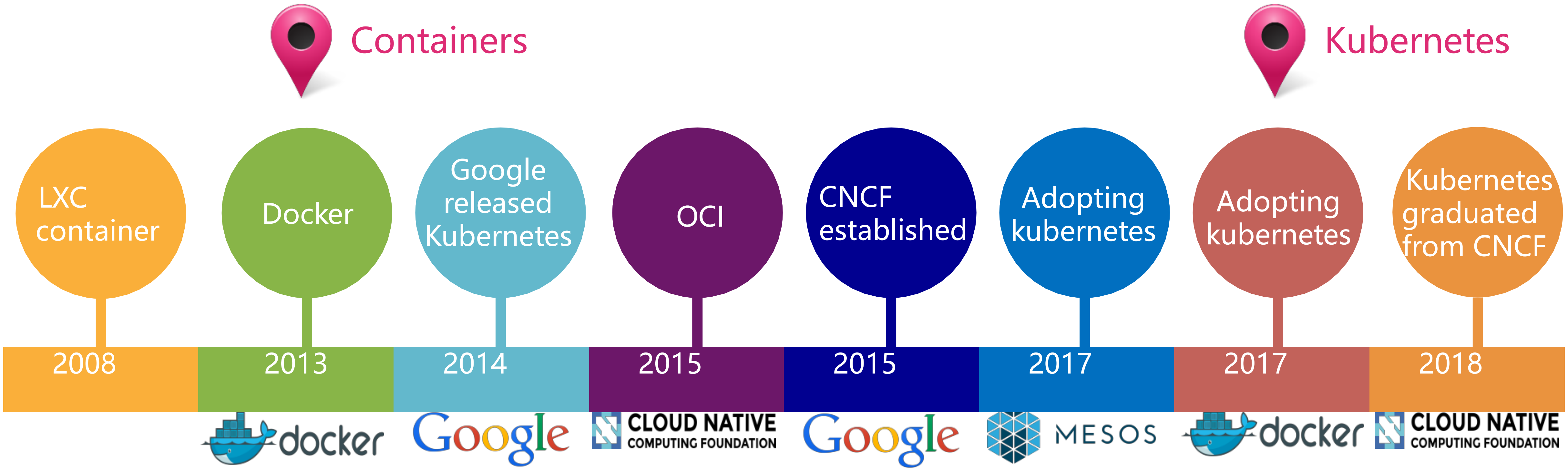
Rise of containers and Kubernetes (K8s)



Advantages of using K8s in reliable & efficient software deployment

- **Velocity:** fast to deploy while maintaining availability by immutable infrastructures & declarative configurations
- **Scaling:** fast and auto scaling of software and develop team
- **Infrastructure abstraction:** applications-infrastructure separation & portability
- **Efficiency:** lower costs of running a server, develop/deploy/test software

Rise of containers and Kubernetes (K8s)



Cloud evolvement in the last two decades

- From physical machines to virtual machines to containers
- Different offerings: IaaS, PaaS, SaaS, Caas, FaaS on Public /private / hybrid cloud
- Kubernetes becoming standard
- High-available service on low-available hardware

Outline

- Brief history of cloud computing
- Cloud native technologies
- Current practice and opportunities of AI on cloud

Placement and load balancing (PLB)

Question: put 18KB into [A: 10KB | B: 20KB | C: 19KB | D: 25KB | E: 30KB]

- **Placement**

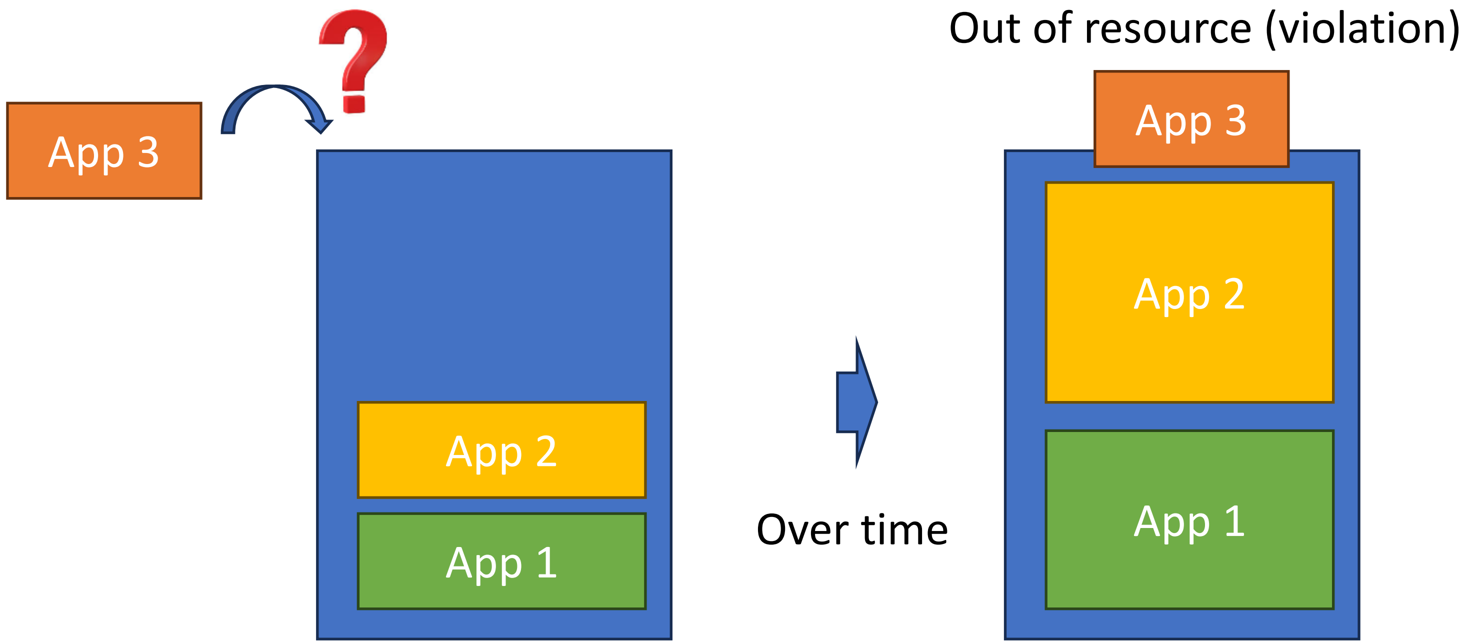
  The overall goal is to reduce violation to users' Service Level Agreements (SLAs), given that resource usages are **dynamic**.

  - **First Fit:** the first one that fits → B: 20KB
  - **Best Fit:** the one that just fits → C: 19KB
  - **Worst first:** the one that has the most resource → E: 30KB

  - **More advanced:**
    - **Multi-resource:** memory, disk, CPU, etc.
    - **More complex policies:** constrains, leave-one-out, leave-two-out and so on

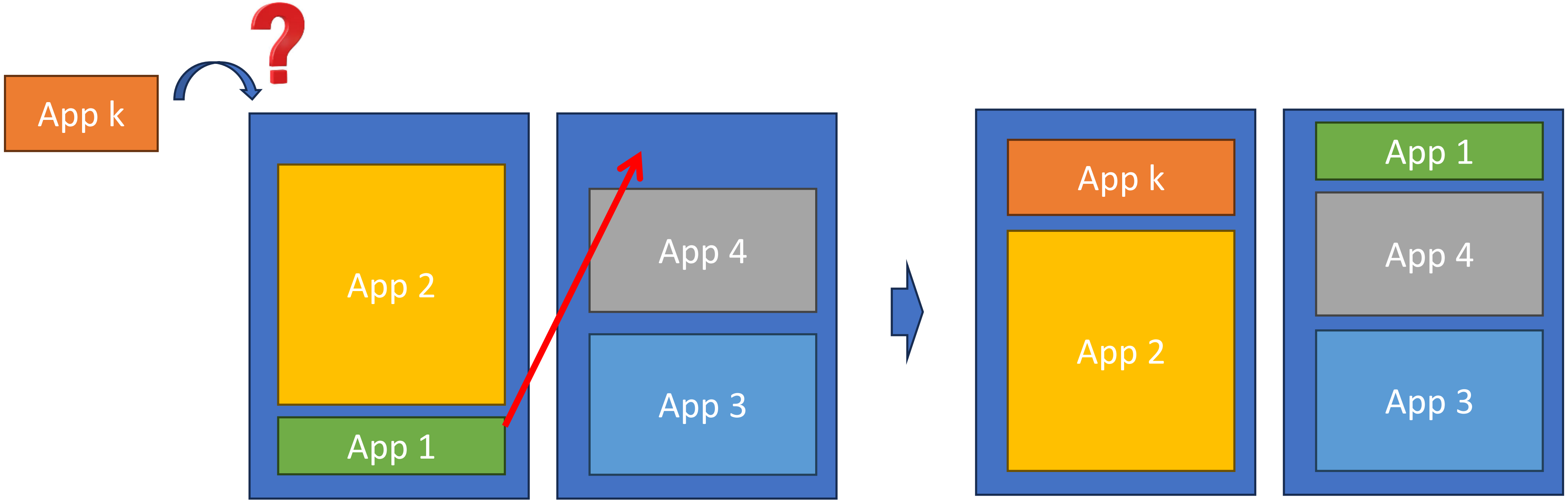
Placement and load balancing (PLB)

- **The real problem:** usages can change → no theoretical guarantee for optimal placement, since everything is data-driven



Placement and load balancing (PLB)

- **Load balancing:** migrate to “make” some room

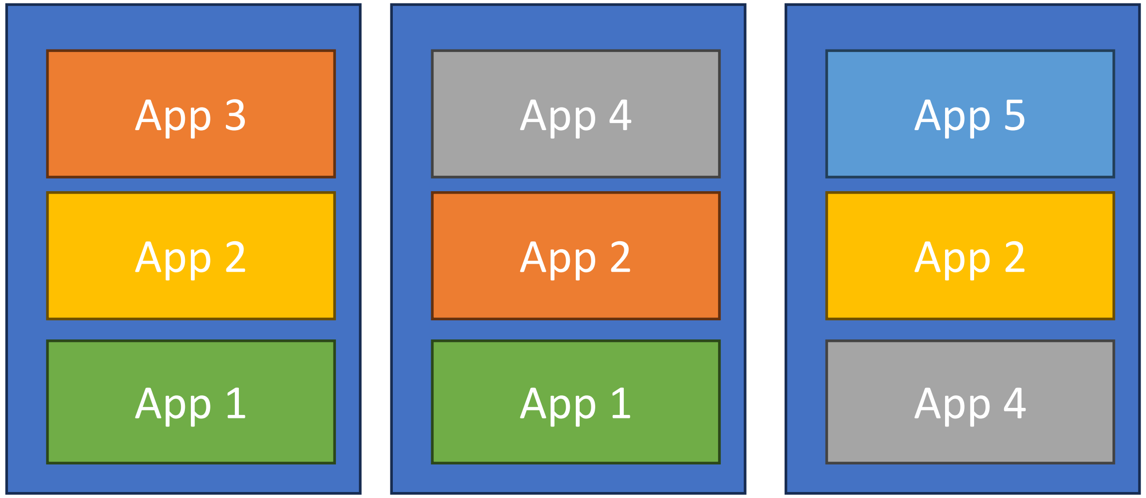


But migration is not free, often very expensive: stateful VMs, DBs, etc.

- Better migration mechanisms: cache compression, disaggregated memory etc.
- Resource usage prediction: placement & balancing based on predicted usages

Failovers

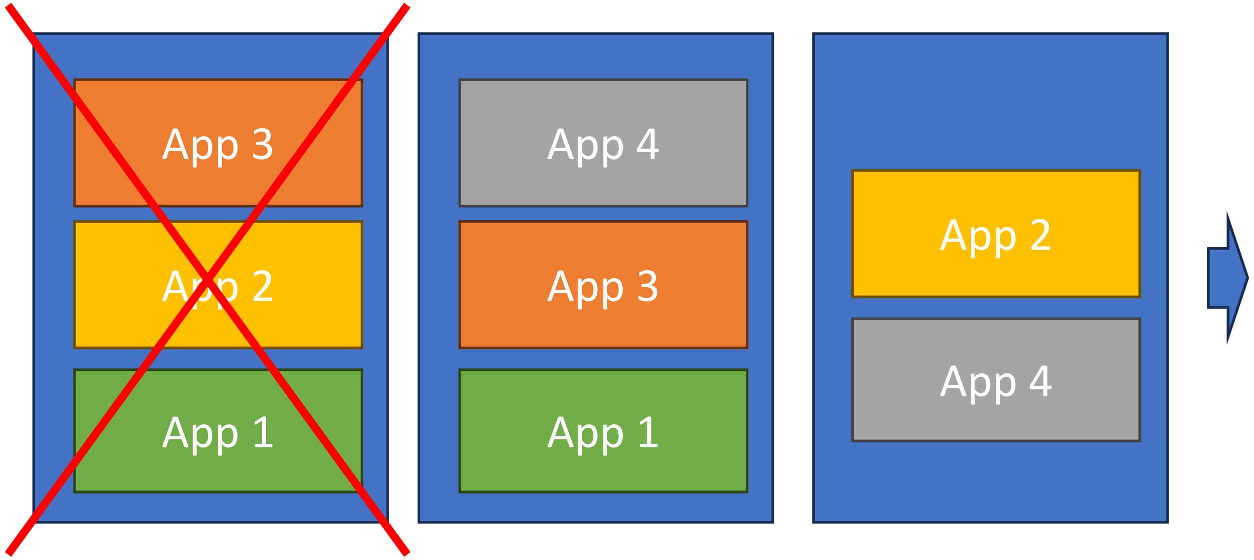
- Failovers ensure a robust & highly available service
  - Duplica on independent resources to avoid simultaneous failure



- Failovers are useful for hot software patching / updates
- Failovers can co-exist with PLB which makes it a lot more complex

Failovers

- **A fast failover** involves efficient context switch & recovery

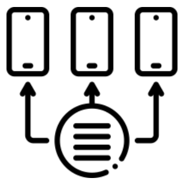


- Route requests to duplica
- Recover main from duplica
  - Reinstate using logs
  - Live migration

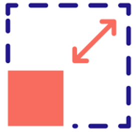
Failover due to:
- Unexpected: software/hardware failure
- Scheduled: Software update

Serverless computing

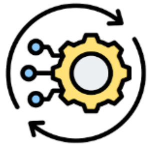
- Some "rewrap" of ideas, but many cloud-native techniques are the same underneath



No servers to provision or manage. User describes application; system finds out best provision.



Scales with usage. System expanse and shrinks automatically with actual usage.



Build-in availability and fault tolerance. System also provides safety belts at no cost to the users.

No pay for idle. Billing model – user pays only for actual usage; financial risks at the operator.

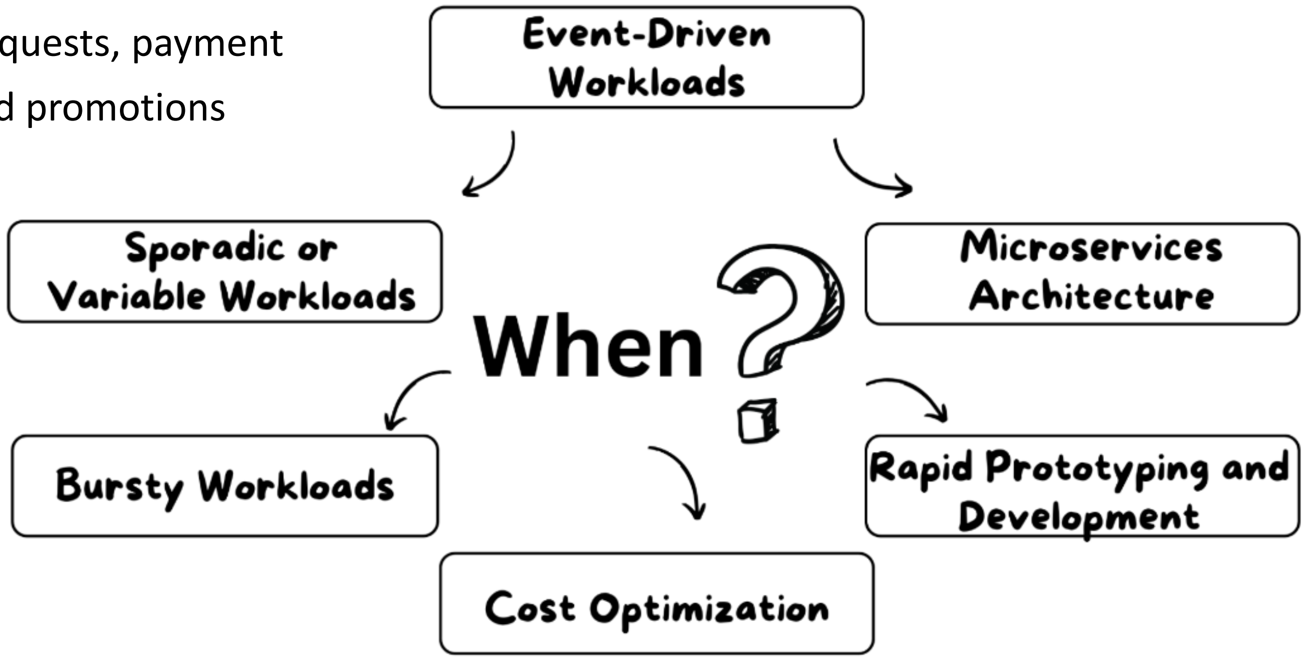
Containers & orchestrators

(New) Serverless functions & overscriptions

Serverless functions, or Function-as-a-service (FaaS)

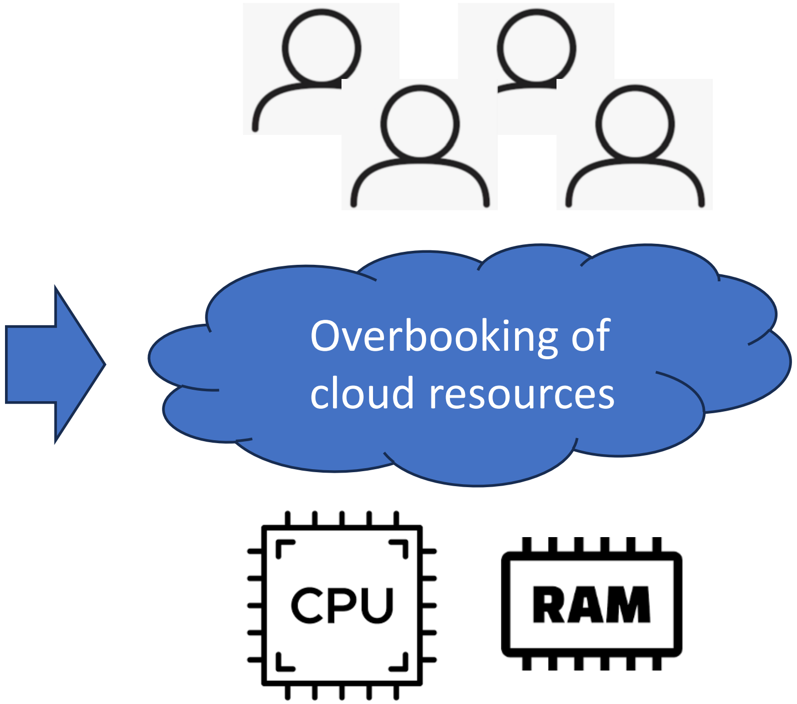
- FaaS is an example of serverless computing to simply deployment of event-driven function calls

- Examples
  - **Netflix:** media encoding, thumbnailing, content recommendation
  - **Airbnb:** user authentication, process booking requests, payment
  - **Coca-cola:** supply-chain operations, personalized promotions



Resource oversubscription

100 seats, sell 105 tickets



- (Almost) direct revenue boost , given the base at **$100B!**
- But still, new technologies needed

Resource oversubscription

- **Technology prerequisites:**
  - Virtualization to cut CPU/disk/memory into fine granularity
  - Quick allocation / migration
  - Multi-tenancy over shared resources

- **Key problem for oversubscription:**
  - Increase oversubscription rate, while reduce/prevent violations w/ user SLAs
    SLA can be latency of query, service availability, etc.

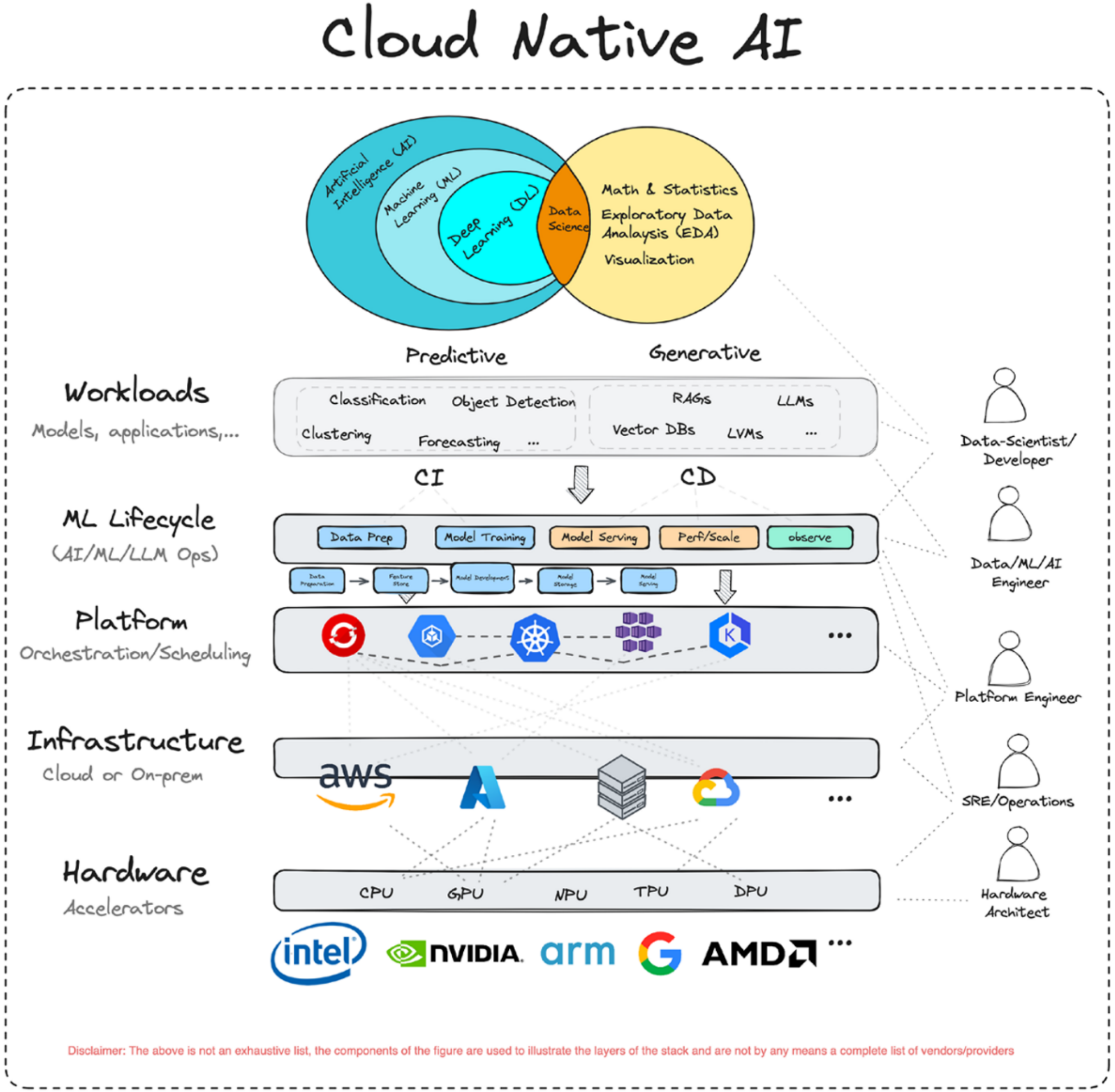
- **Mechanisms for an oversubscribed system:**
  - Similar PLB but at high resource usages

Outline

- Brief history of cloud computing
- Cloud native technologies
- Current practice and opportunities of AI on cloud

Practice and opportunities of AI on GPU cloud

- Cloud native technologies are open-box solutions for may AI use cases
- **Key idea:** containerizing your models

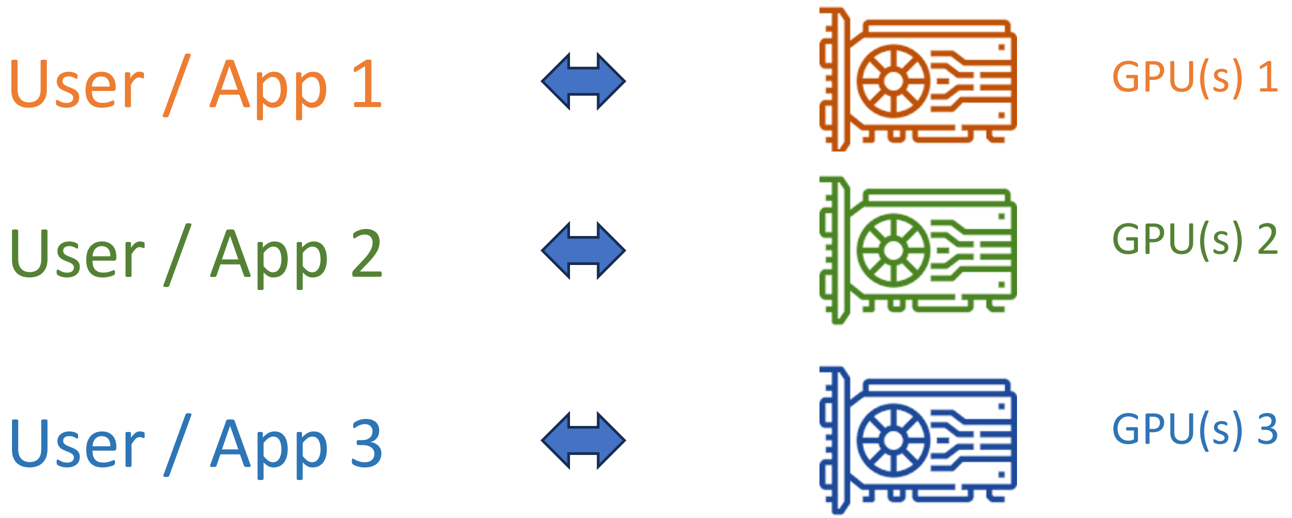


https://www.cncf.io/wp-content/uploads/2024/03/cloud_native_ai24_031424a-2.pdf

Practice and opportunities of AI on GPU cloud

- But, some cloud-native ideas couldn’t be applied
  - GPU virtualization and oversubscription
  - Fine-grained scheduling and operations
    - Each container is a big black-box

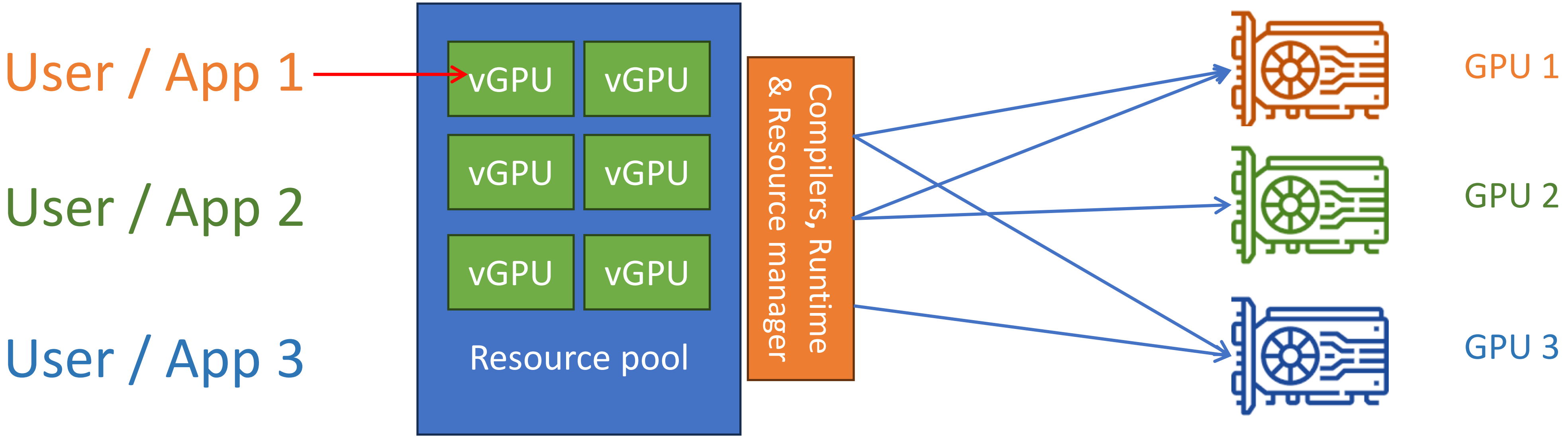
Resource oversubscription for GPUs?



Tight coupling between
resource specification and allocation

This means it’s hard to switch context / allocation, even when the resource is in idle.

Breaking the tight coupling between apps & allocation

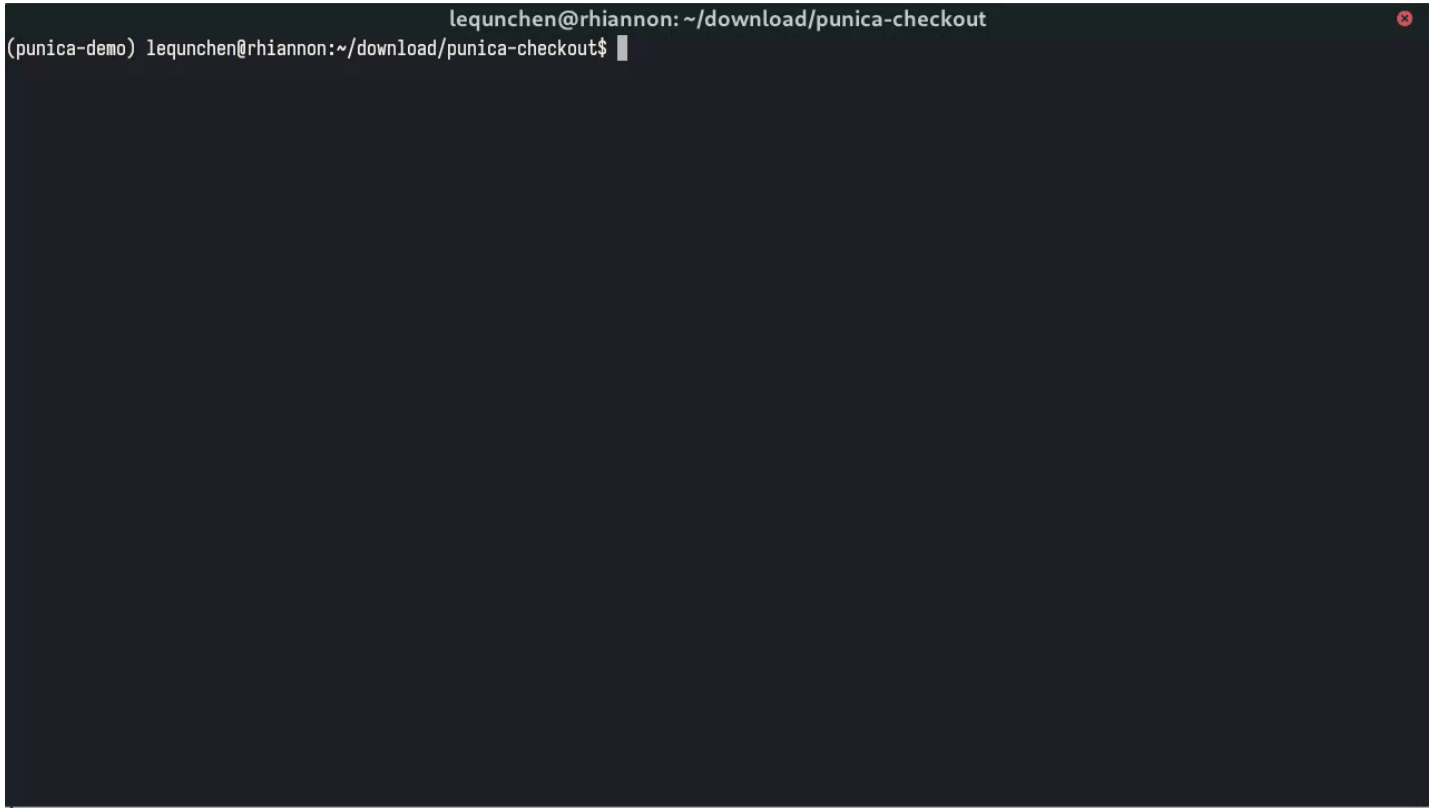


"Virtulizating" GPUs into thread and memory blocks,
But, big engineering challenges

Looking forward

- Multi-tenancy in AI workloads
  - Users, apps, tasks, models, adapters, prompts, …

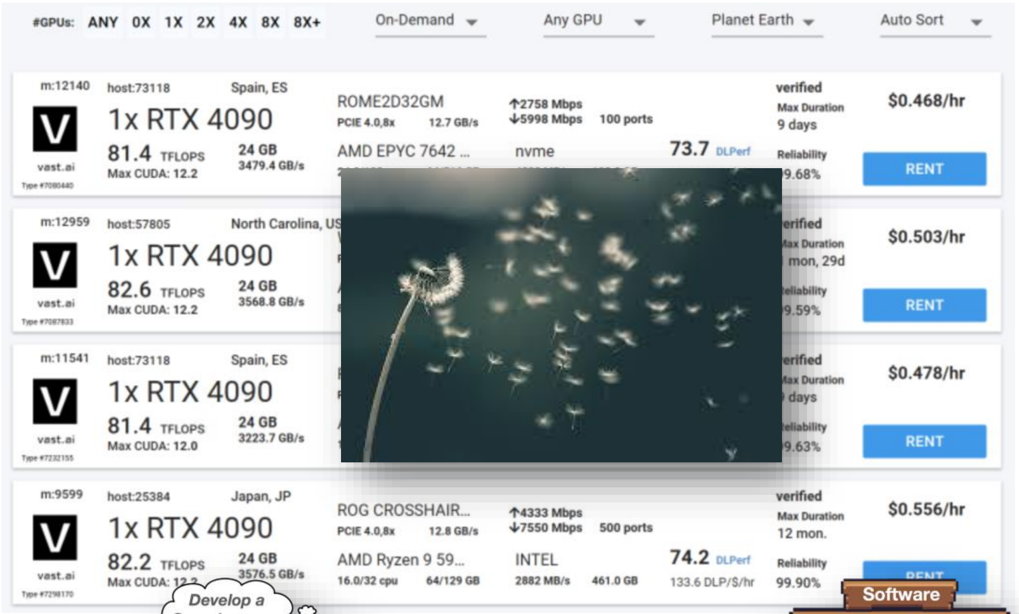
- Breaking the black-boxes
  - Co-design of AI and cloud systems
  - Cloud-native → AI-native



Punica: Multi-Tenant LoRA Serving

Looking forward

- New opportunities in cloud AI
  - **New cloud** with heterogeneous, ephemeral infra
    spot instances, intermittent/green power
  - **New services:** multi-agent AI, RAGs etc.
  - **New hardware & architecture:** RISC-V AI chips, AISC chips
  - **New applications:** AI-for-X



ChatDev

What we have covered & not covered

Covered

- MLsys foundations
- Automatic differentiation
- Hardware acceleration
- Parallelism and training techniques
- Transformers, attention and optimizations
- Serving LLMs
- Fine-tuning and alignment techniques
- AI for systems
- Application systems
- ML compilers
- Cloud systems for AI

Not covered

- Compute graph optimization
- Heterogeneous runtime
- Serving multi-modal models
- Serving mixture-of-experts models
- ML ops

- Many more..

Wish you all the best in your PhD / Masters journey!