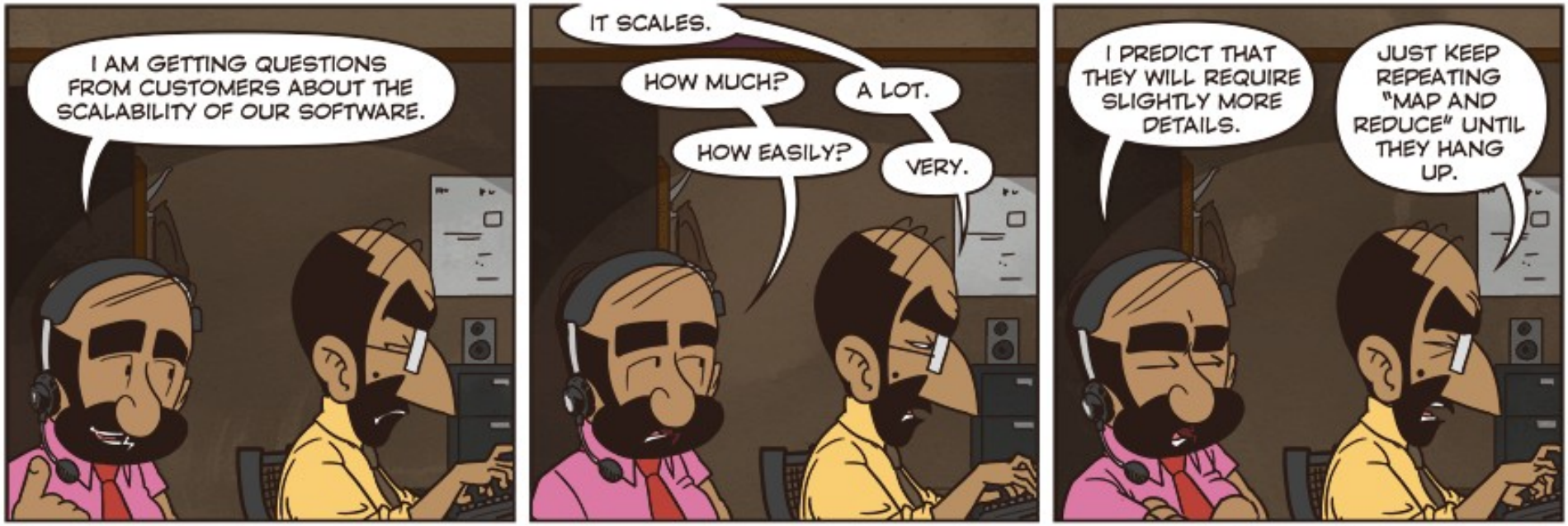
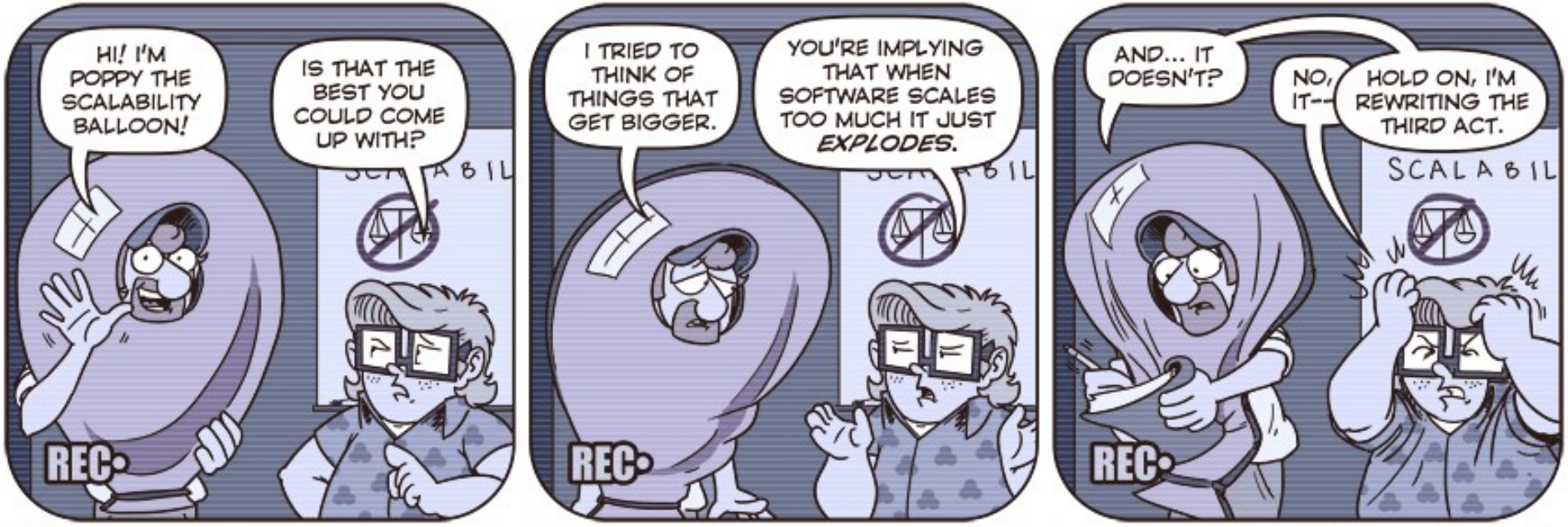


Barnes • Southworth • Zugale

Architectural Elements for Scalability, HA & FT

Scalability

- Ability to handle growth



Scalability

- Ability to handle growth
- Can be measured as the ability to maintain latency in the face of increased
  - frequency of requests
  - number of users
  - volume of exchanged data
  - distance between clients and servers
- Latency components:
  - network latency
  - server latency

- Elasticity:
  - ability to dynamically adapt and scale resources up or down, based on demand

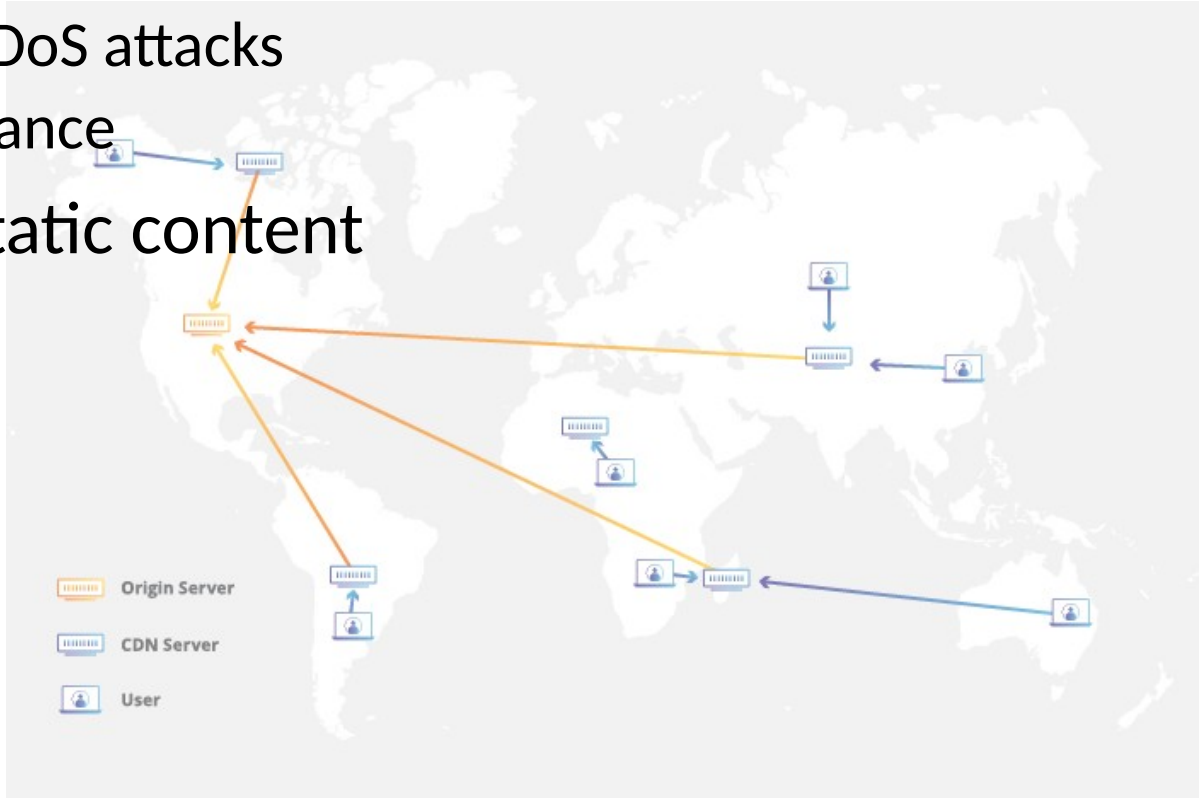
  → optimized resource utilization: maintain performance, and manage costs

Mitigate network latency (under increased load)

- (Data compression)
- (Network protocols optimization)
- Content Delivery Network
- Fog and Edge Computing

Content Delivery Network (CDN)

- Network of geographically distributed server replicas + DNS to route the client to the closest mirror
- Additional benefits:
  - Load balanced between servers (reduces server latency)
  - Defense against DoS attacks
  - Better fault tolerance
- Better suited to static content

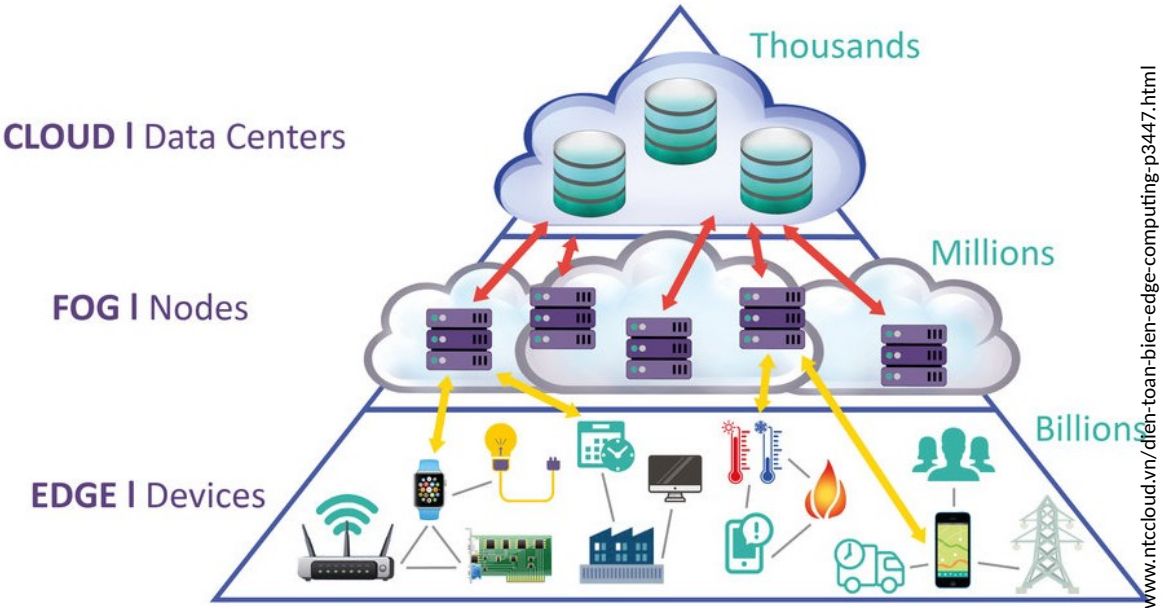


https://www.cloudflare.com/learning/cdn/what-is-a-cdn/

Fog computing and edge computing

- IoT objects generate a lot of data
- Transmitting them to the cloud for processing and storage is problematic
  - Too much traffic
  - Too much latency

- Idea: Leverage the middle layers between the cloud and objects



https://www.ntccloud.vn/dien-toan-bien-edge-computing-p3447.html

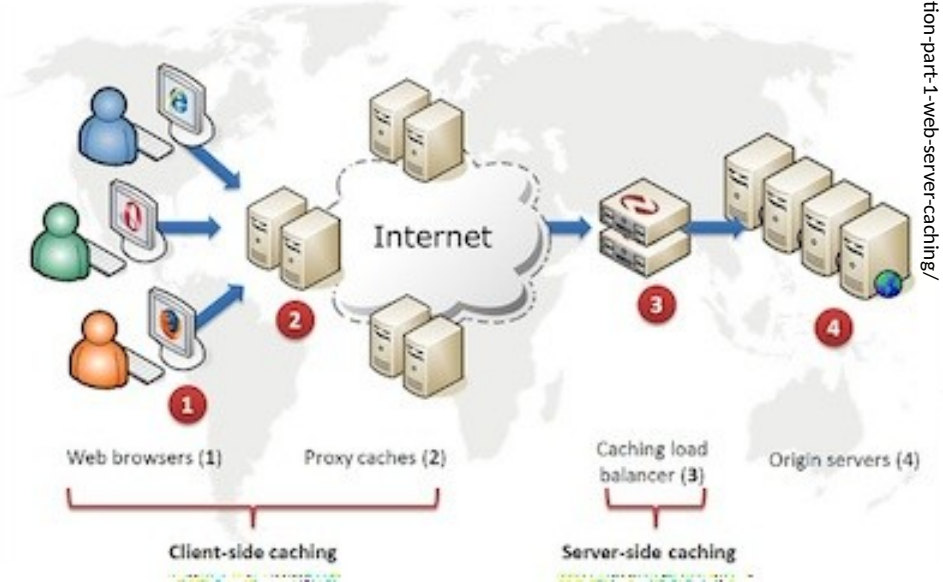
Mitigate server latency (under increased load)

- Cache
- Vertical scaling
- Horizontal scaling: replication + load balancing

Caches

- Principle:
  - Keep the most frequently read data close to the user and serve them without requesting the original server
- Objectives:
  - Reduce latency
  - Reduce servers load
  - [increase fault tolerance]
- At different levels of

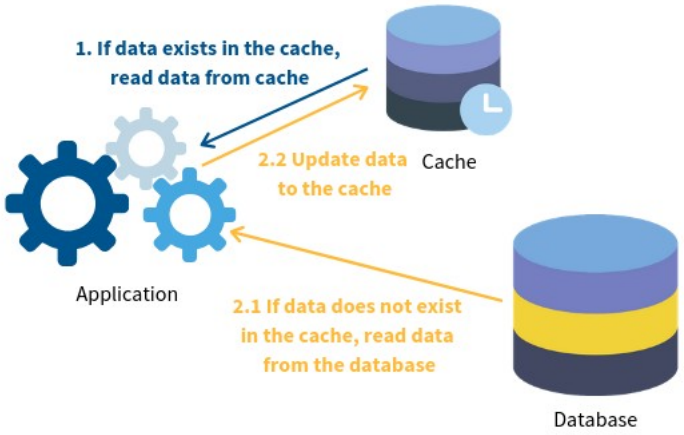
the architecture



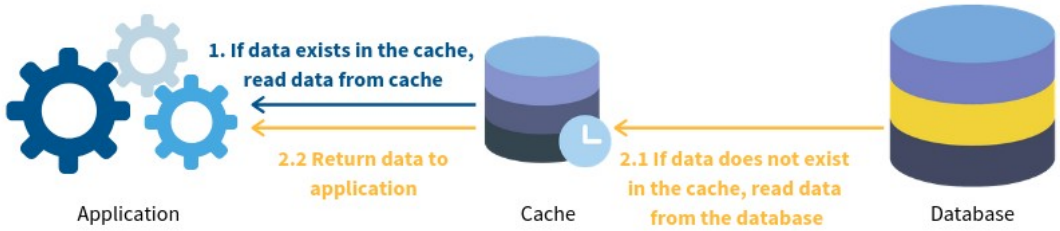
https://loadstorm.com/2011/06/web-application-performance-optimization-part-1-web-server-caching/

DB Caching Strategies

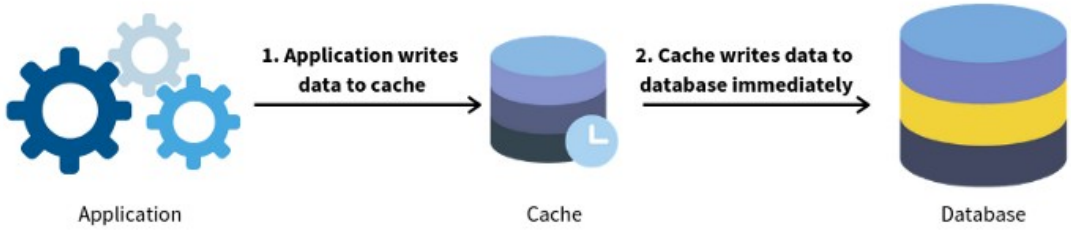
Cache Aside



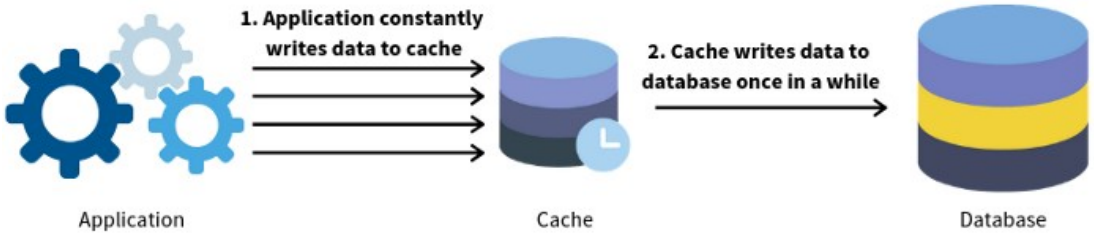
Read Through



Write Through



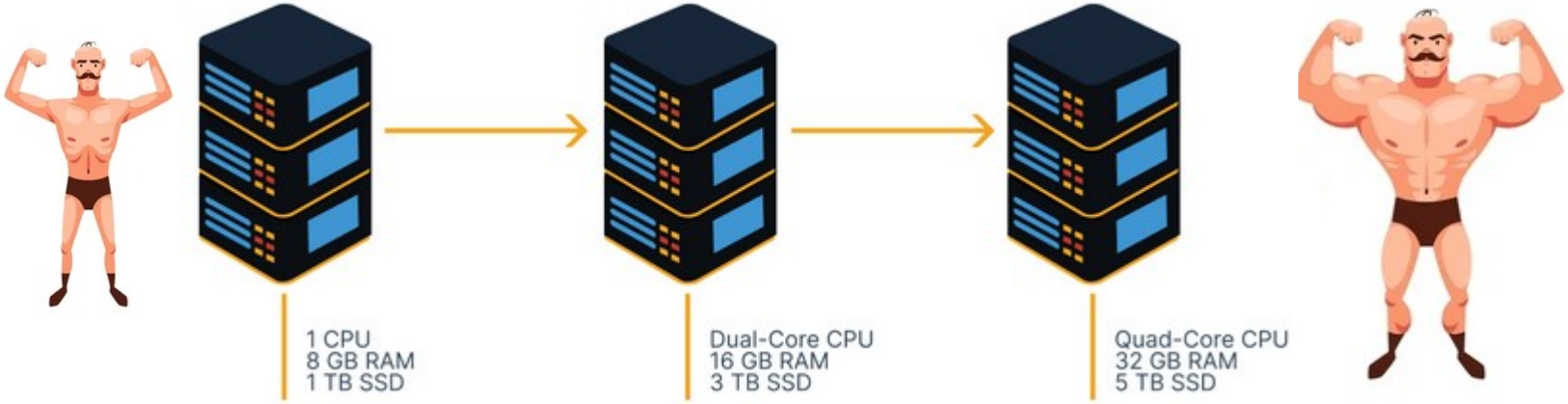
Write Behind



https://blog.bluzelle.com/things-you-should-know-about-database-caching-2e8451656c2d

Vertical scaling (scale up)

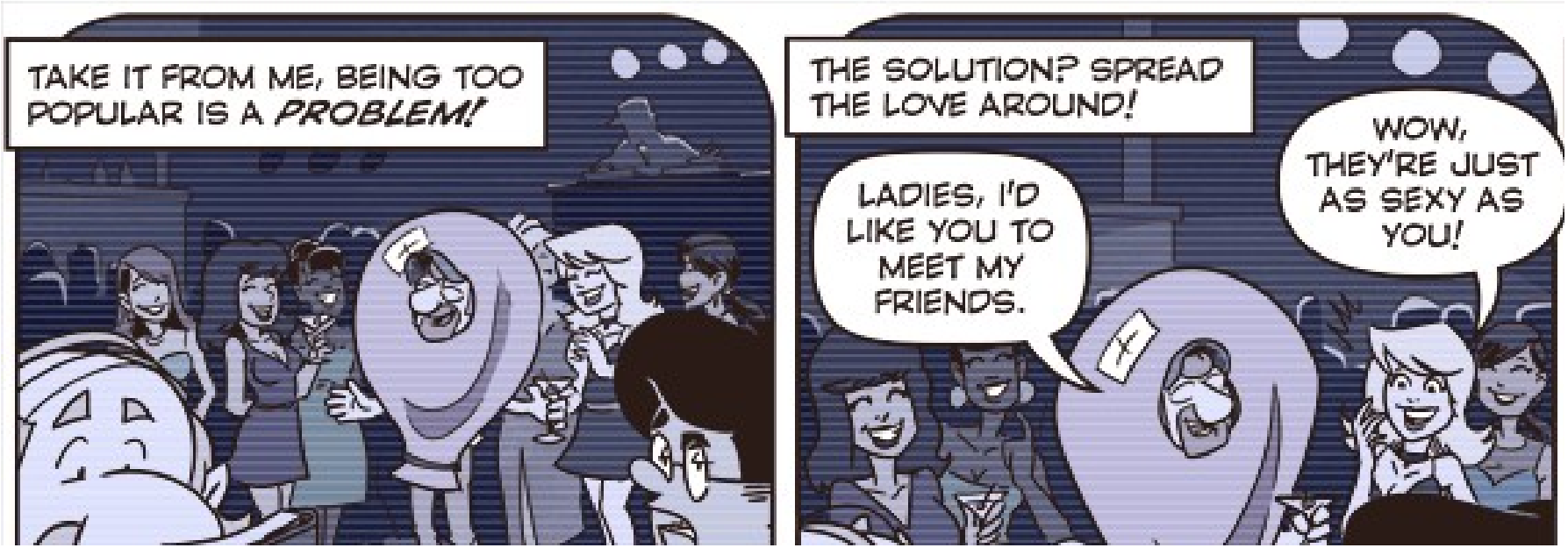
- Improve or replace the existing server: increase the capacity of a single server by adding resources (more powerful CPUs, larger amount of RAM, or expanded storage)



- a single node handles the entire workload, relies on multi-threading to process multiple concurrent requests

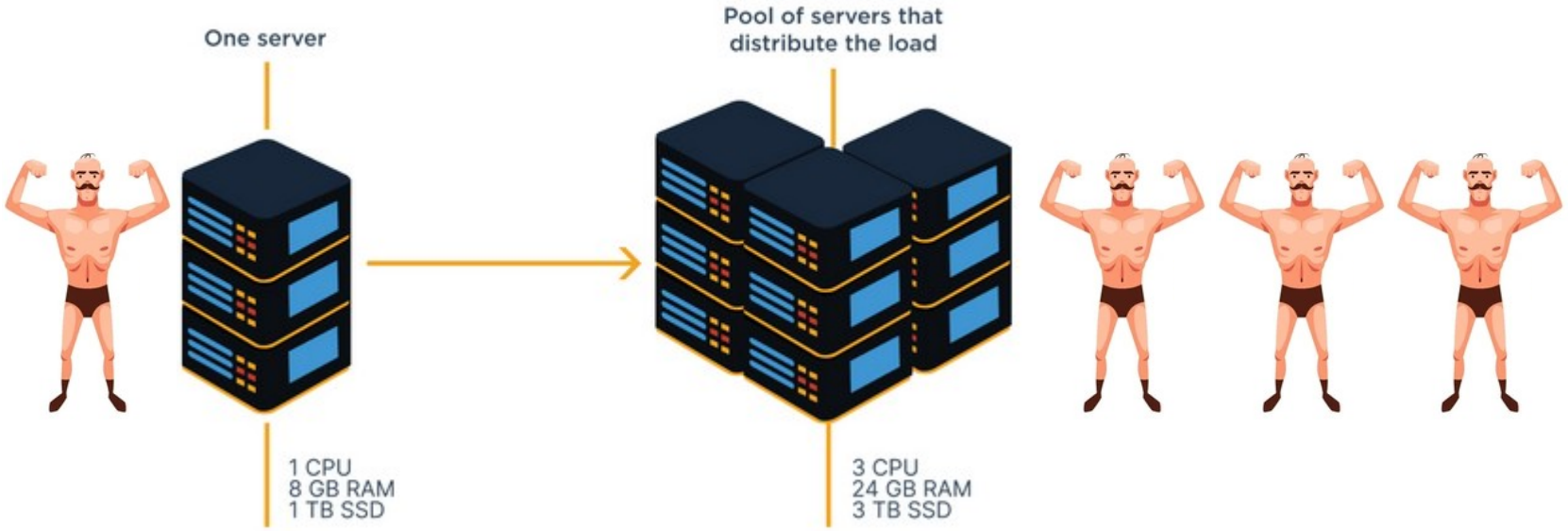
Horizontal scaling (scale out)

- Add nodes to the infrastructure and distribute the load among nodes



Horizontal scaling (scale out)

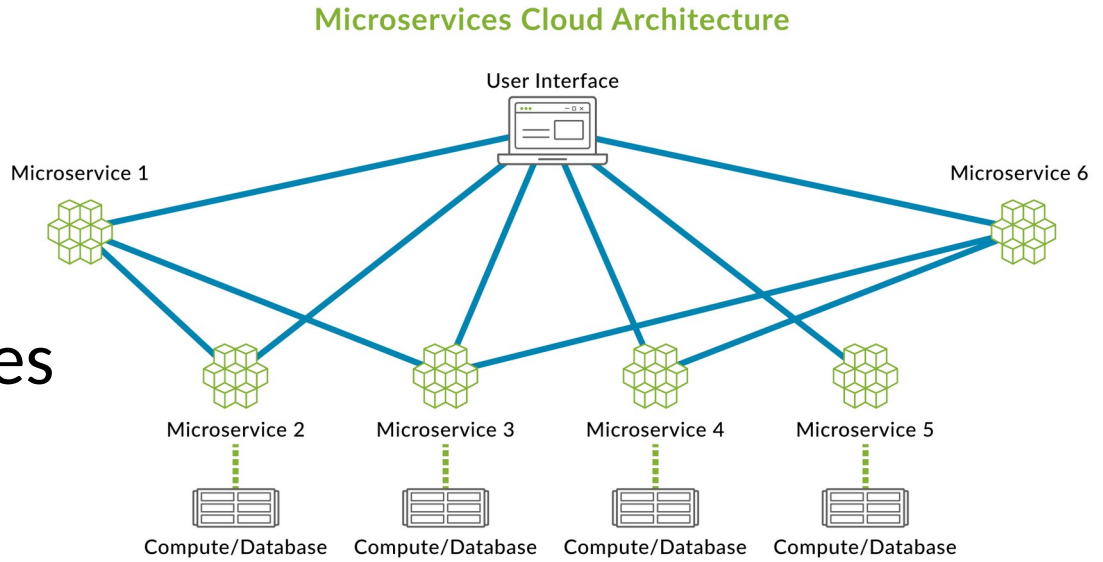
- Add nodes to the infrastructure and distribute the load among nodes



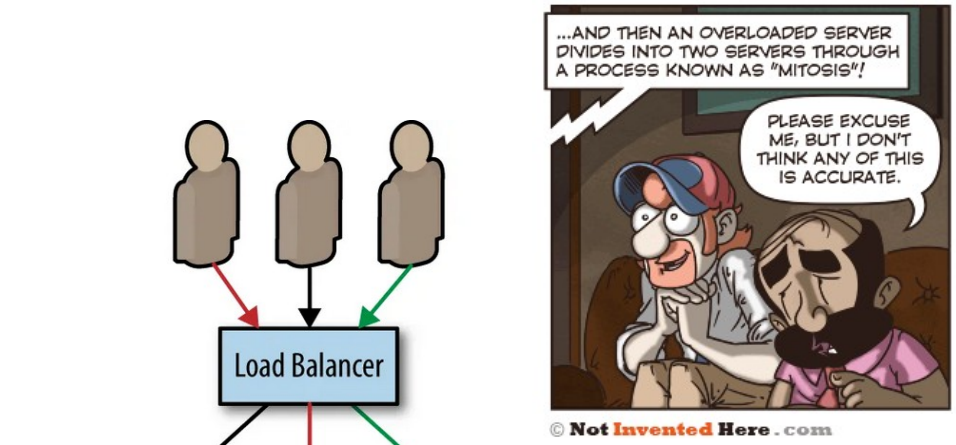
- Requirements:
  - Compliant software architecture
  - Load balancing mechanisms

Software architecture for horizontal scaling

- Distribute functions across multiple nodes → micro-services + middleware



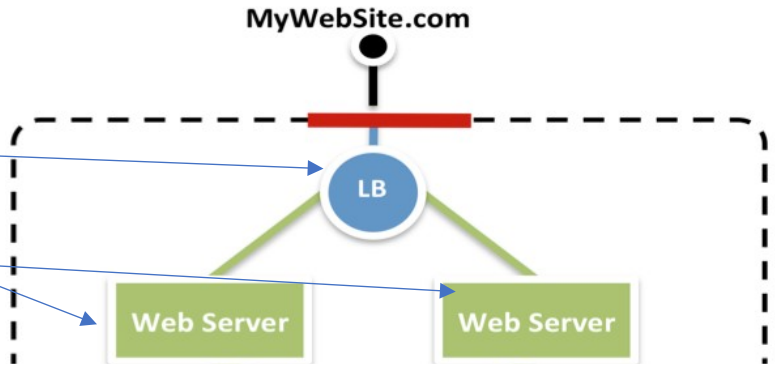
- As load increases, duplicate nodes → stateless services



Load balancing

How to balance the load between

Replicas of the group of servers

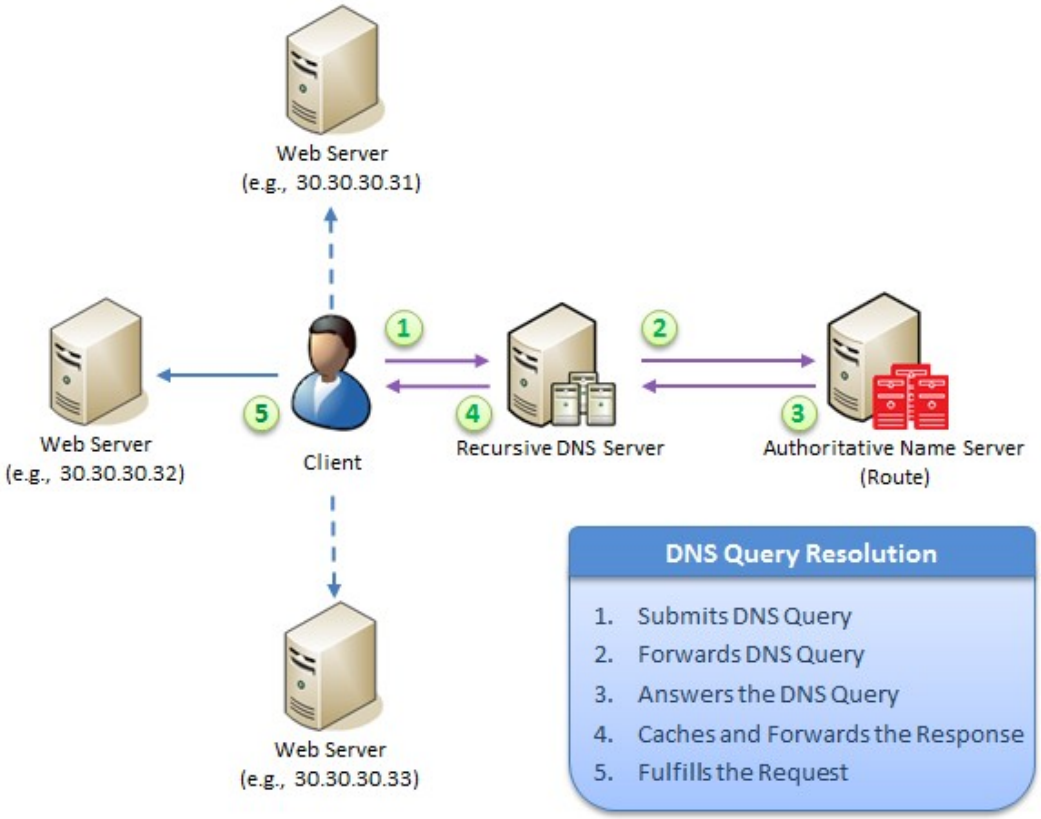


https://docplayer.net/4134029-Software-as-a-service-saas-on-aws-business-and-architecture-overview.html

Hardware load balancer

- Solution in decline
  - non-virtualizable → not deployable in a cloud
  - not elastic → must be oversized
  - must be duplicated for HA
  - more expensive than a subscription to a cloud load balancing service

DNS-based load balancing

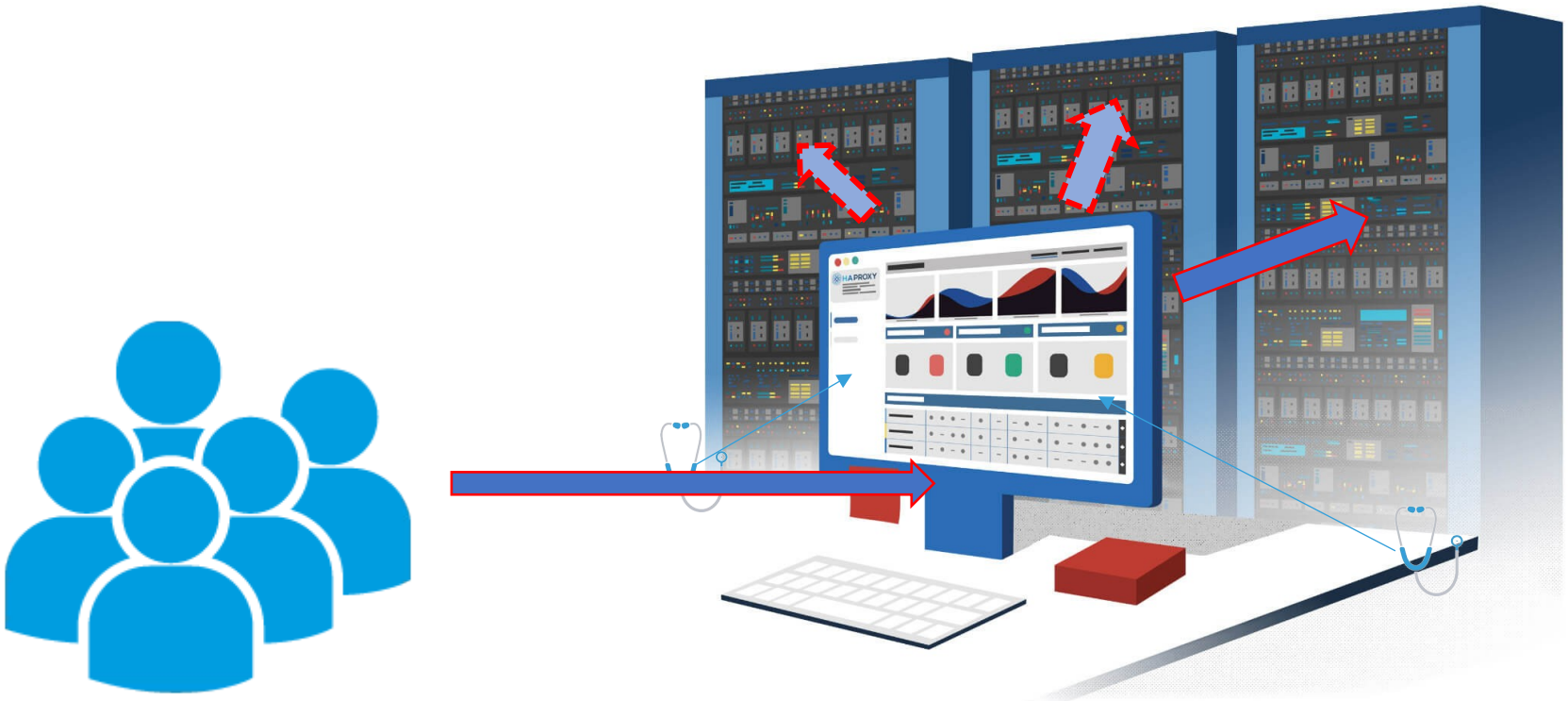


- Does not take into account the load of the servers
- Low reactivity to failures

https://docs.vdms.com/cdn/Content/Route/Administration/LB_How_Does_It_Work.htm

Solfware load balancer

- Probe-based decisions
- Smarter balancing than with DNS
- Layer 4, Layer 7

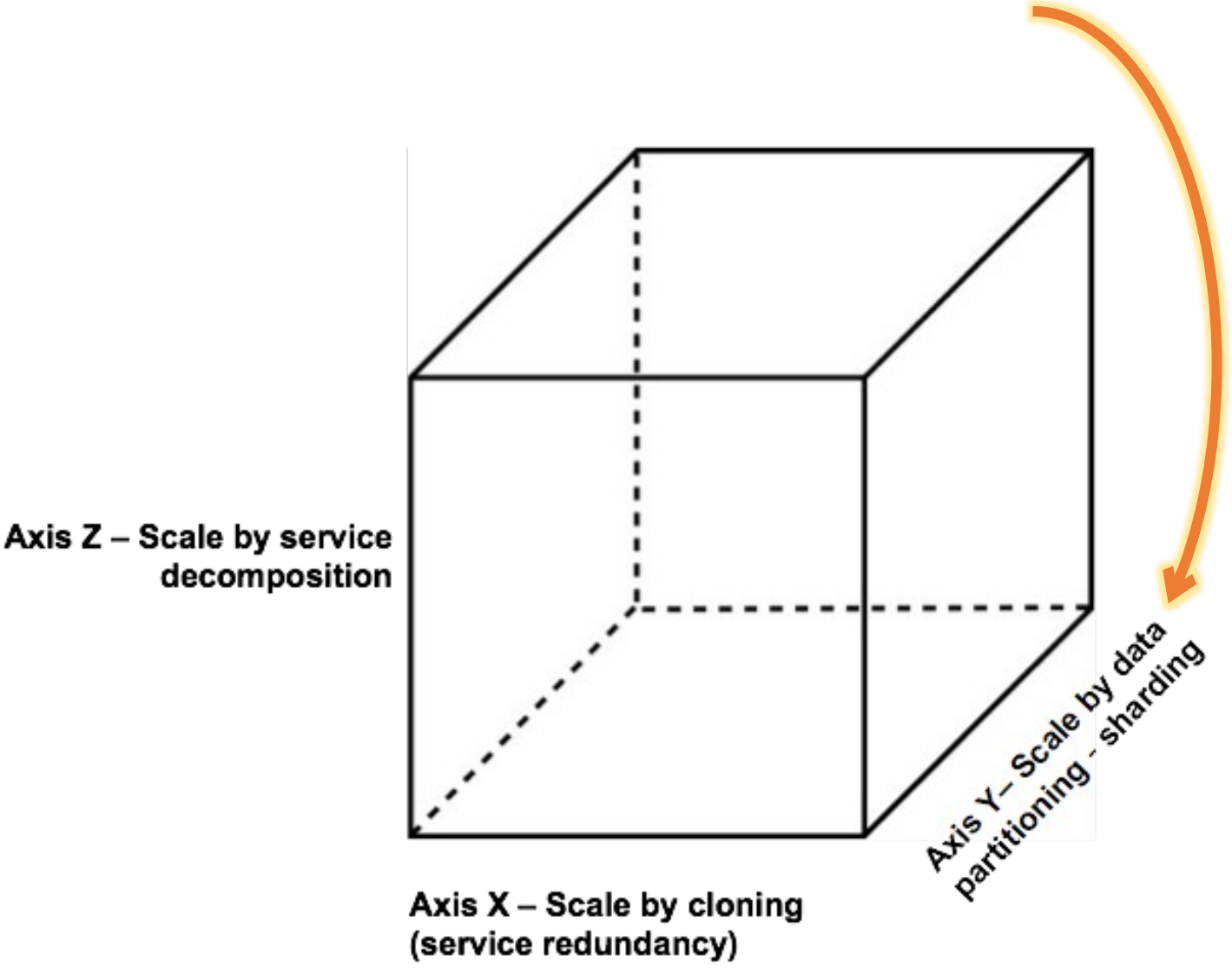


https://www.haproxy.com/fr/homepage/

Horizontal / Vertical Scalability

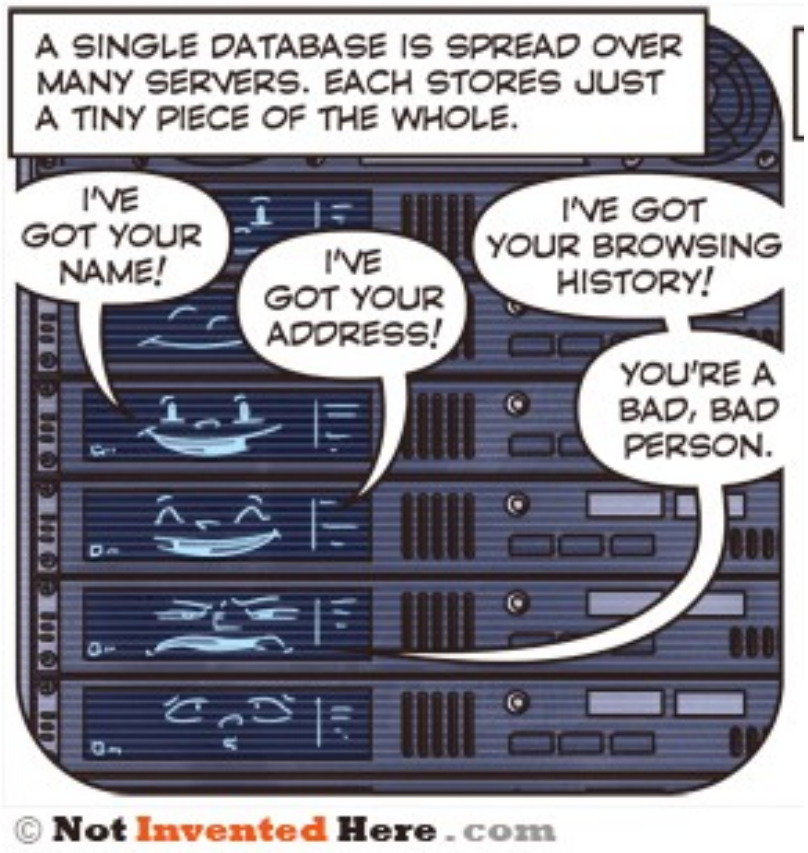
	Vertical (*scale up*)	Horizontal (*scale out*)
principle	increase in server power (CPU, memory, I/O) 	multiplication of resources and distribution of treatments 
Extensibility	Limited	Important
Elasticity	None	Yes
Influence on the code	None	Redesign potentially needed to make it distributed and stateless
Configuration	Easy	Complex (LB, middleware…)
upgrades	Often requires temporary downtimes	transparent
Un availability risk	High	Low

The 3rd dimension of scaling



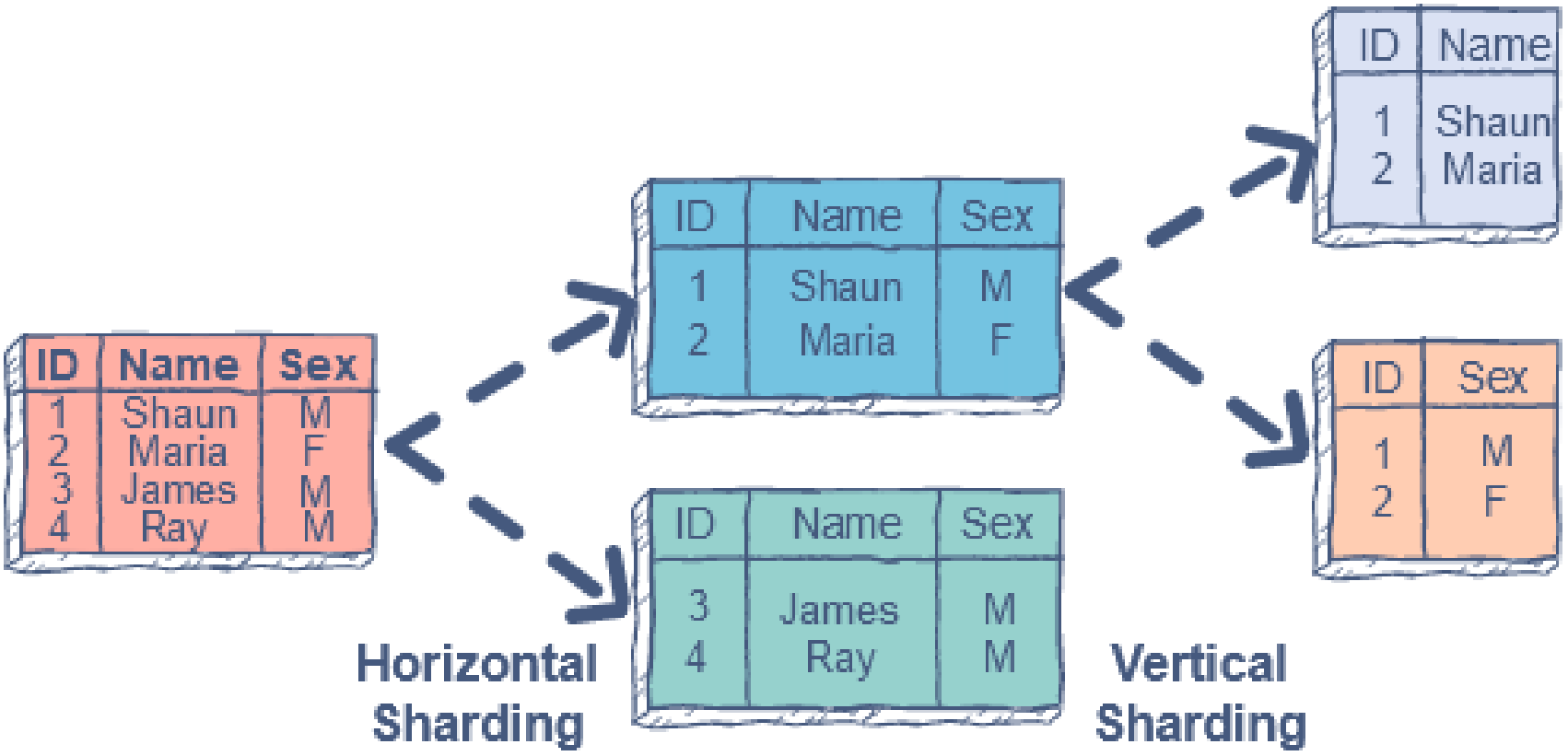
Data partitioning

- divide large databases into smaller datasets hosted on different servers



Data partitioning

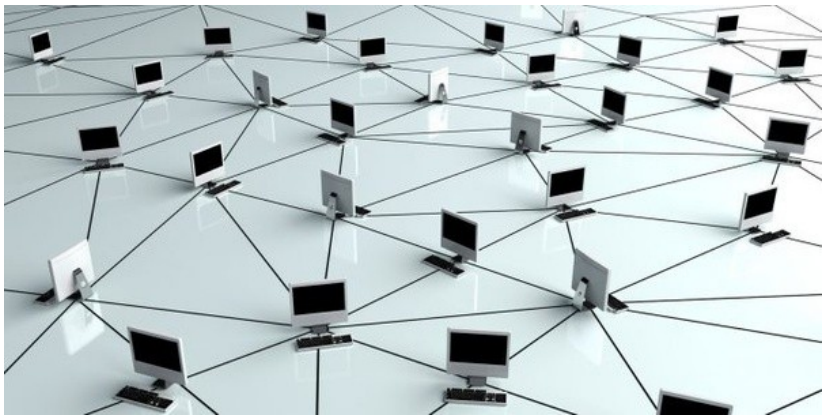
- divide large databases into smaller datasets hosted on different servers

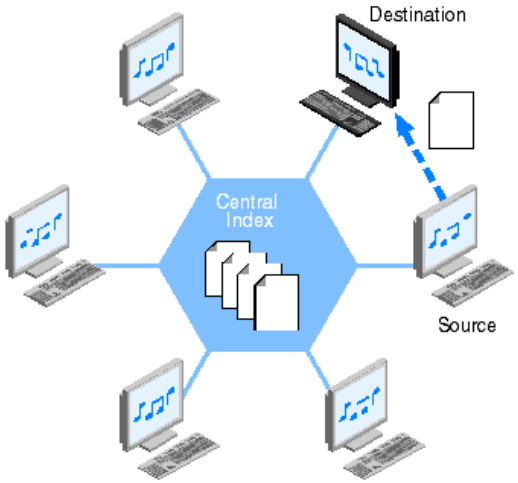


https://www.educative.io/answers/what-is-database-sharding

Peer-to-Peer (P2P) Architecture

- Each node can both
  - be a consumer of services offered by others
  - provides services to others
- Variants :
  - “pure": resource discovery mechanism
  - hybrid: with a central server to connect (but then direct exchanges between peers)





https://pubs.opengroup.org/architecture/togaf8-doc/arch/chap31.html

Advantages of P2P : example of file sharing

- “natural” scale up

  the more a file is requested, the more it is available

- robust

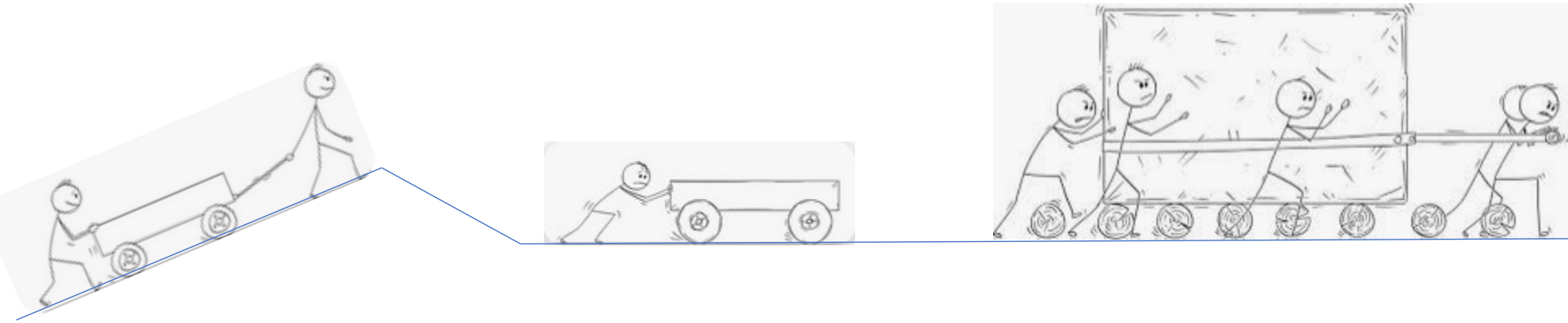
  (apart from single point of failure if there is central connection server)

- Segmented P2P file transfer (chunks) system:
  - cancels the asymmetry of ADSL
  - limits the effects of sudden disconnections from the supplier
  - a file can be globally available without anyone having the whole file

P2P challenges

- Peer discovery

  solutions:
  - central node
  - Distributed Hash Table
- Free-riders
  - solution: incentive mechanisms
- Peer volatility
- Trust
  - solution: redundancy and comparison

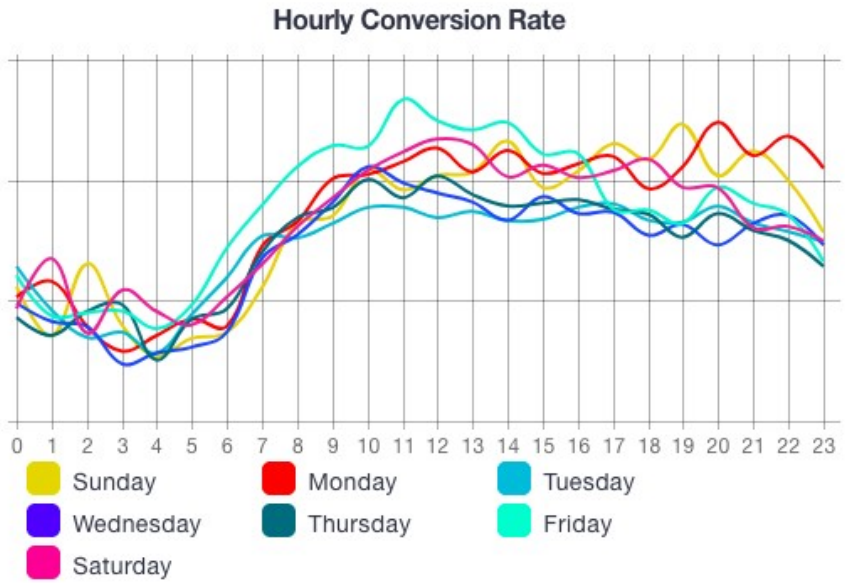


Elasticity

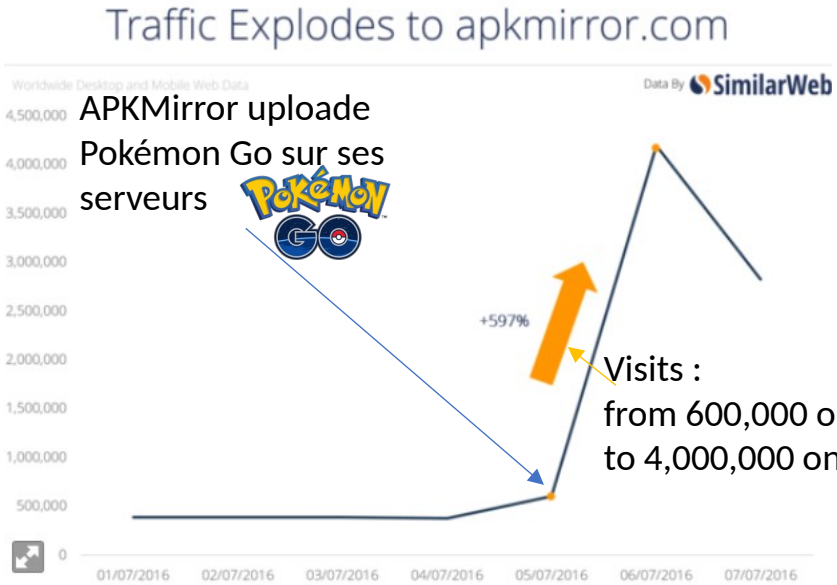
ability to dynamically adapt and scale resources up or down as needed

Load not always regular, nor predictable

- seasonal activity with foreseeable peaks of activity (daily, monthly, yearly…)
- rapid growth, little visibility (marketing effects and buzz…)



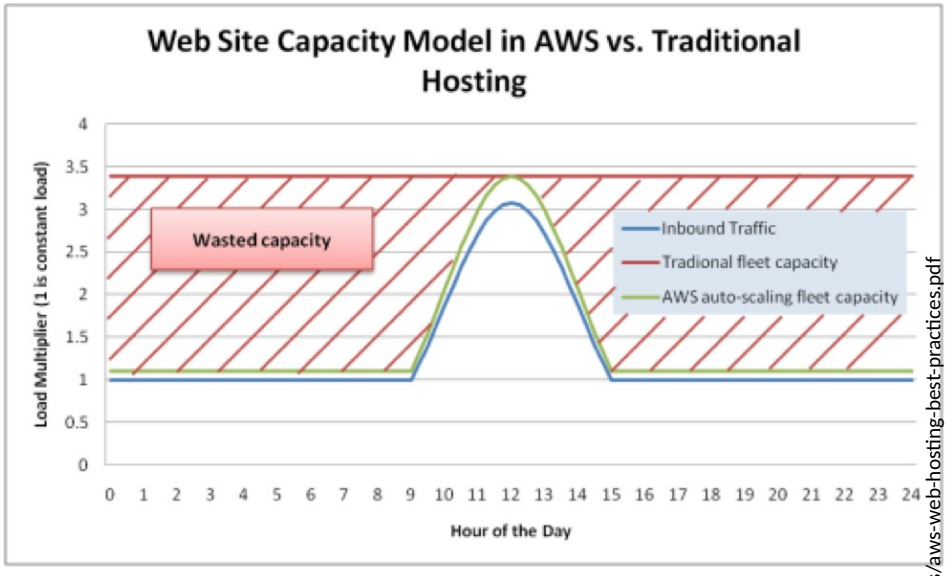
https://blog.workarea.com/trends-when-do-people-shop-online



https://www.pcworld.com/article/3092969/gotta-catch-em-all-5-facts-that-prove-pokmon-go-is-eating-the-world.html

Sizing

- Undersized infrastructure:
  - Cheap
  - Cannot handle the load
- Oversized infrastructure:
  - Absorb the load peaks easily
  - Expensive, lots of wasted resources
  - Difficult to predict how large is large enough



https://d1.awsstatic.com/whitepapers/aws-web-hosting-best-practices.pdf

Key elements for elasticity

- Dynamic scaling

  Resources can be added or removed as needed without interrupting the service

- Resource provisioning

  Ability to allocate additional resources when demand increases and de-allocate resources when demand decreases

- Automation

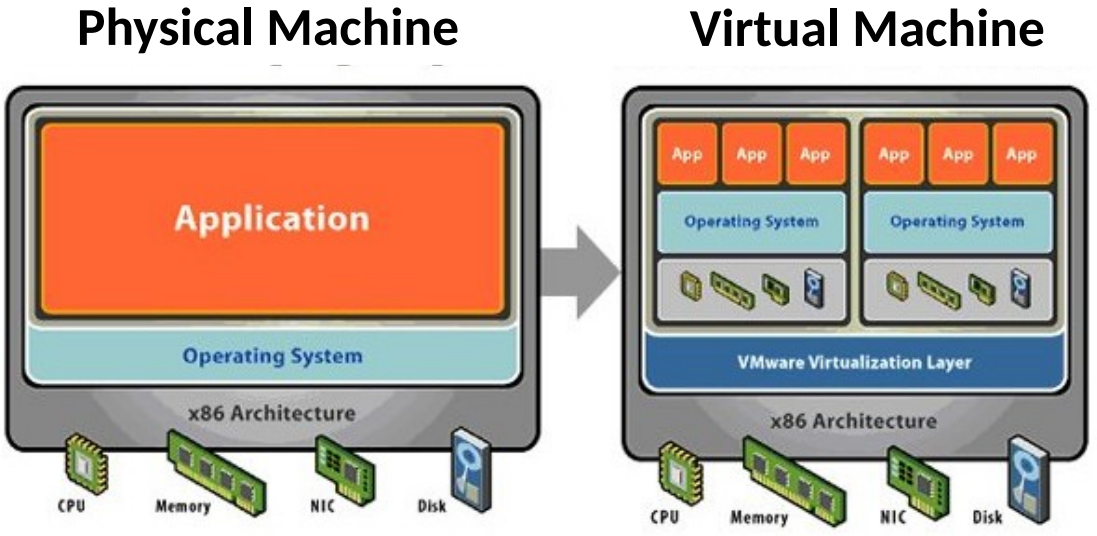
  to scale resources without manual intervention, ensuring a rapid and efficient response to changing workloads

Elasticity enabler technologies

- Virtualization of resources + automation

Virtual Machines

- Software emulation of a physical computer



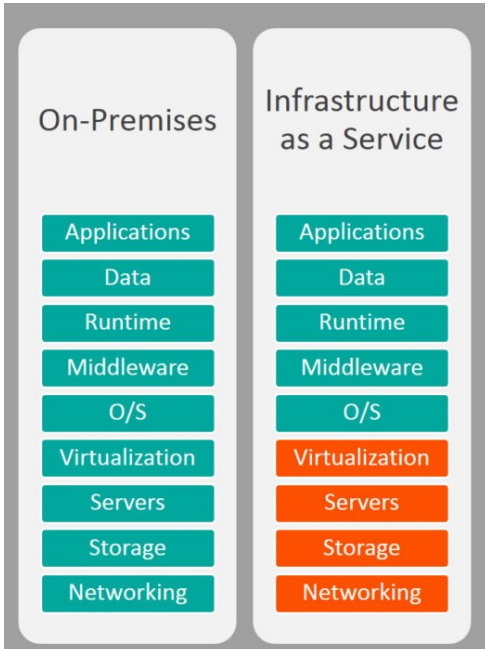
http://www.it-ns.com/virtualisation-et-storage.php

vmware vSphere · Microsoft Hyper-V · KVM · XPROXMOX · Xen · VirtualBox

Virtual Machines

- Enable vertical scaling: CPU, memory, and storage can be easily adjusted for a VM

- Some horizontal scalability: buy more instances from the cloud provider (Iaas cloud model)

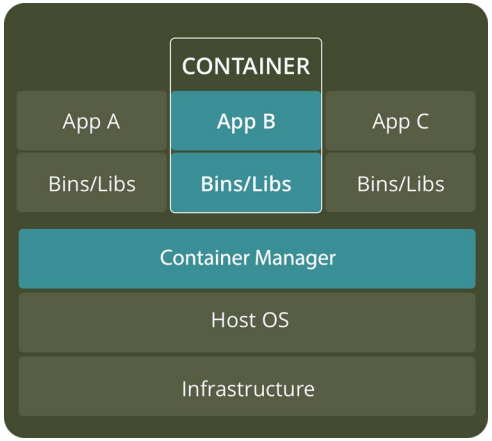


Managed by the user
Managed by the cloud provider

https://blog.dys.com/cloud-iaas-paas-saas/

Containers

- Containers share the host operating system's kernel but are isolated from each other, and encapsulate an application and its dependencies



https://www.backblaze.com/blog/vm-vs-containers/

Containers

- Lighter than VMs, offer faster deployment, well-suited for horizontal scaling
- CaaS cloud model

Container orchestrator

- tool or platform designed to automate the deployment, management, scaling, and operation of containerized applications
- Key functionalities:
  - Start, stop containers based on demand or on metrics
  - Monitor the containers and restart or replace unresponsive containers
  - Connect containers to the network
  - Logging (allow to track performance, diagnose issues…)

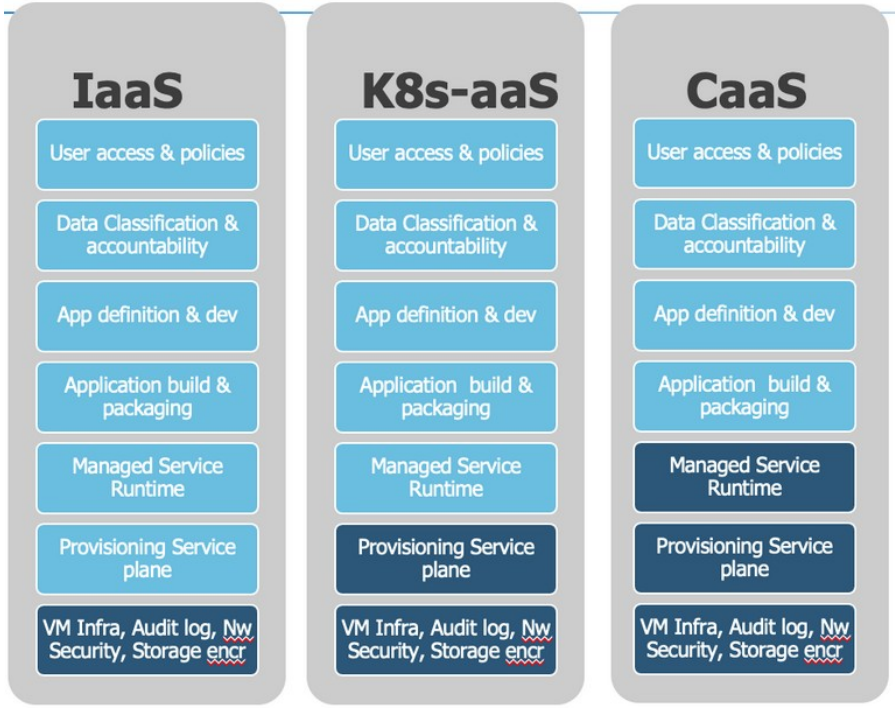
kubernetes

Amazon ECS

RED HAT OPENSHIFT Container Platform

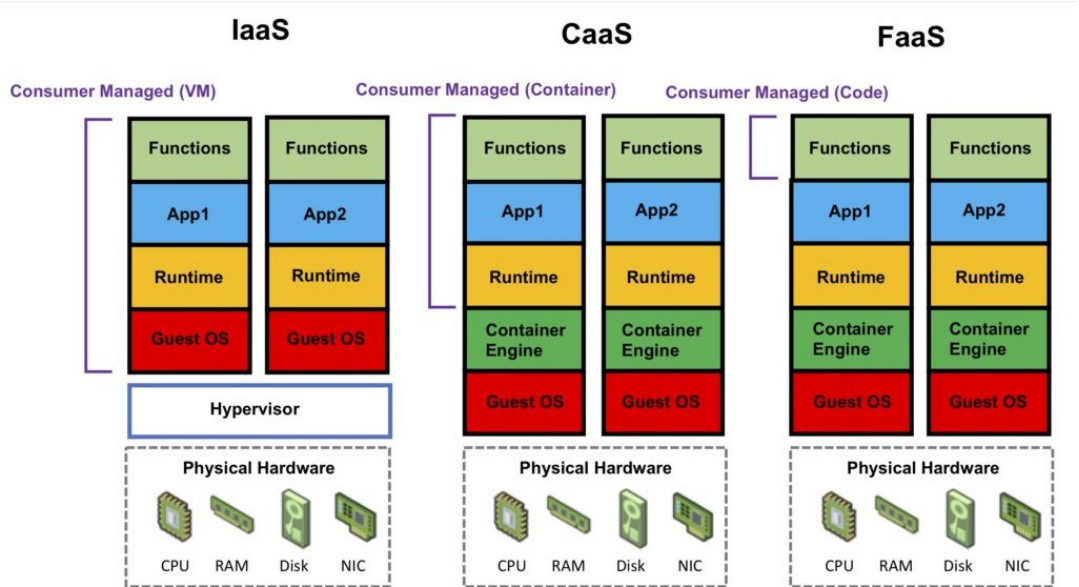
Container orchestrator

- Enable horizontal scaling: you can easily scale applications by adding or removing container instances
- KaaS cloud model which enables end users to deploy and manage Kubernetes clusters in on-demand and self-service mode



Function as a Service (FaaS)

- cloud model where individual stateless pieces of code are executed in response to events or triggers, the FaaS platform dynamically allocates resources to execute functions, and scales down when there's no activity
- highly cost-effective for sporadic workloads



aws Lambda

Azure Functions

Google Cloud Functions

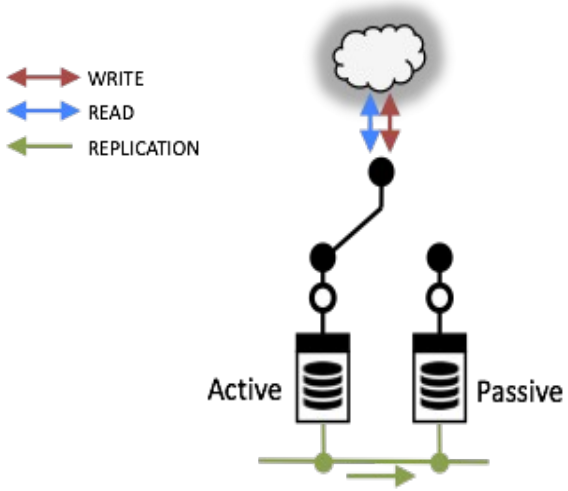
Resilience

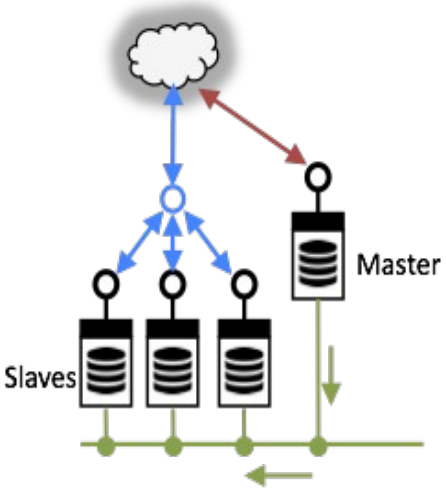
- Fault Tolerance:
  - continue operation, or at least gracefully degrade, even in the presence of failures
- High Availability:
  - minimize downtime, recover quickly from failures

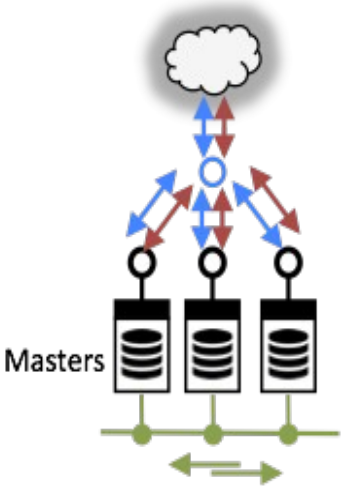
- A common strategy: redundancy and replication

Redundancy

- Active-passive, active-active redundancies…



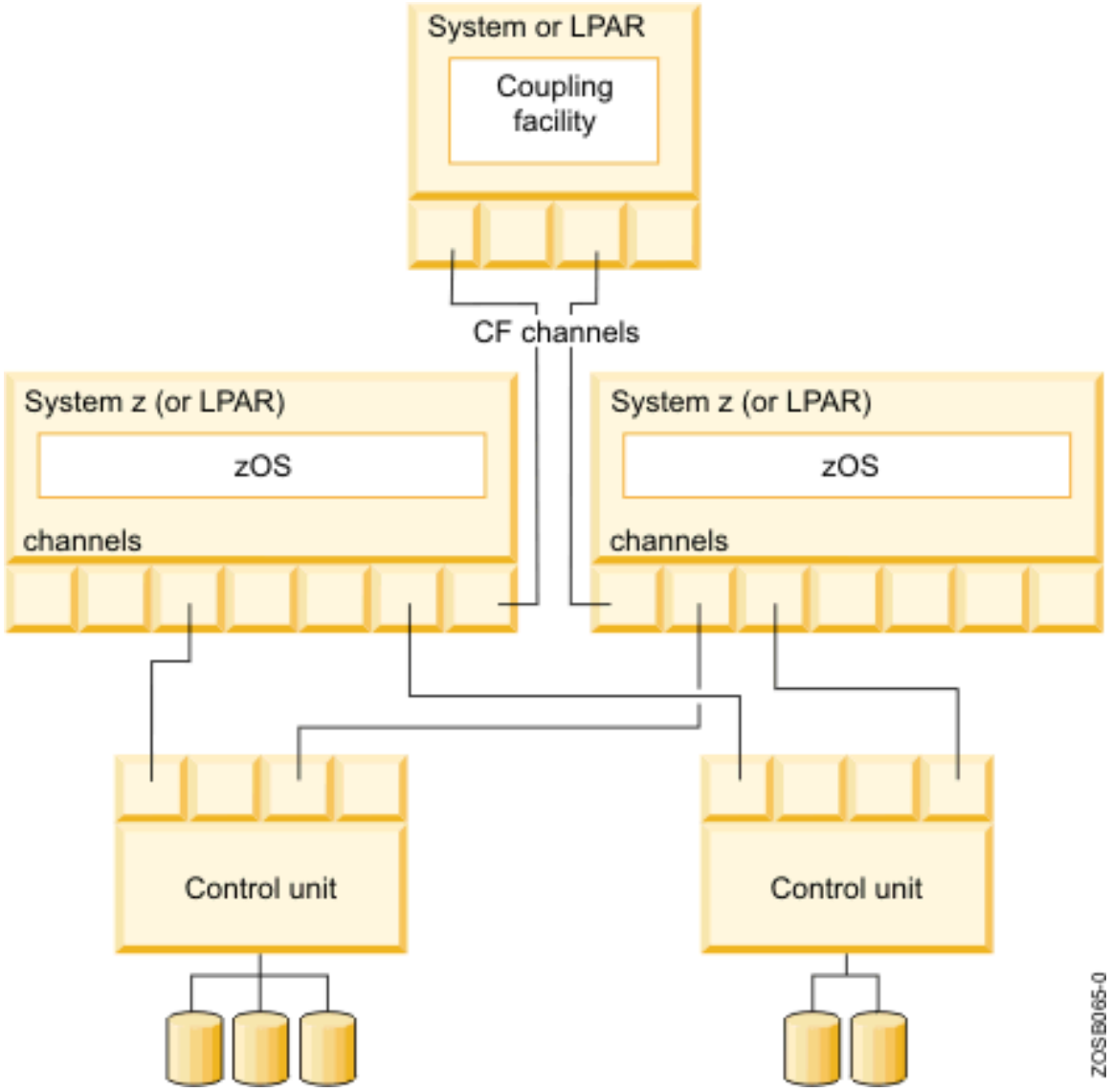




PAC theorem applies here…
(Impossible to ensure simultaneously
- Partition tolerance,
- Availability and
- Consistency)

- Geographical distribution
- Disk mirroring, redundant network connections, redundant power supply, etc.

Mainframes clustering

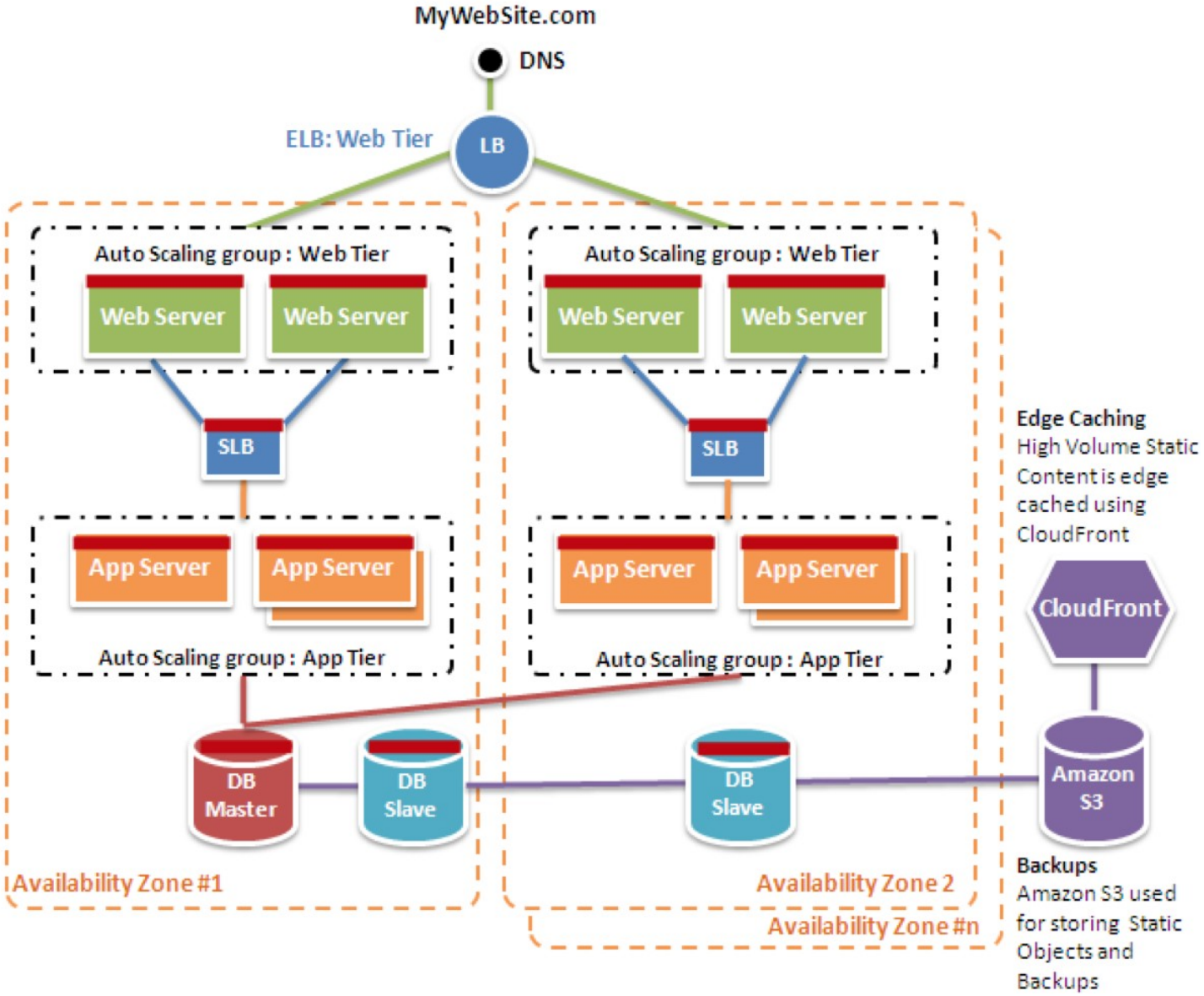


Service layer

Data layer

https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zmainframe/zconc_clusterPlSys.htm

Example of HA&FT Architecture



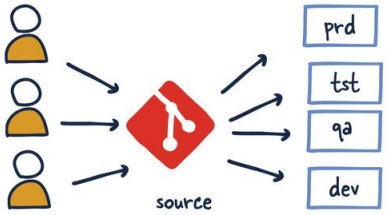
https://docplayer.net/4134029-Software-as-a-service-saas-on-aws-business-and-architecture-overview.html

Concluding Thoughts

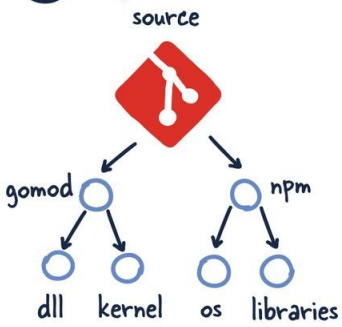
- Scaling calls for the seamless integration of both architectural and technical solutions
  - Requires a systemic approach: targeted optimization may not yield global benefits
  - Requires a multi-skilled team, with software design to commercial solutions experience

12 FACTOR APP REVISITED

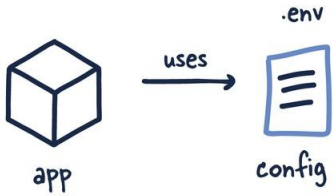
1 code base



2 dependencies

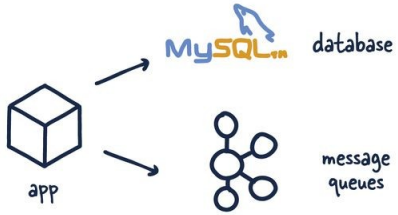


3 configuration

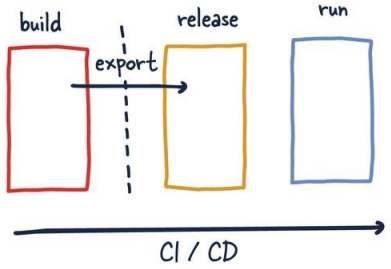


architecture notes

4 backing services



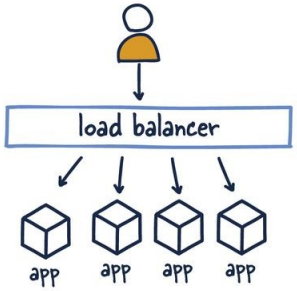
5 build, release, run



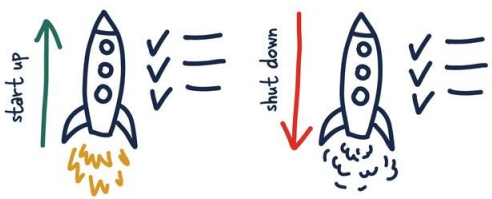
6 Processes

7 port binding

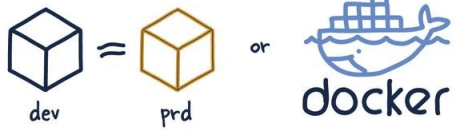
8 concurrency



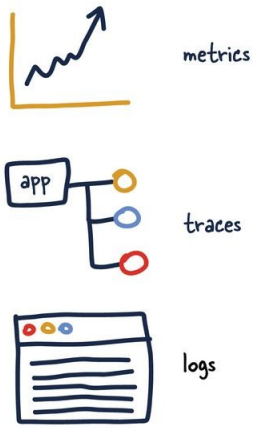
9 disposability



10 dev/prod parity



11 logs



12 admin processes