



# **Enterprise Architecture**

Intro



Who are you?



Who I am



# What we'll learn

- Several Architectures and trade-offs
- From Monolith to Micro services & reactive programming
  - Circuit breakers, gateways, ES,CQRS , Sagas, ..

Distributed systems concepts with use cases:

- Persistence (Cassandra)
- Config and Service Discovery (Consul)
- Secret Management (Vault)
- Queue/log (Kafka)
- Cluster Deployment (Kubernetes)
- Basics of Platform engineering
- If you're still alive!
  - Real time processing

# You'll need

- Unix based OS (e.g. Linux)
- In Microservices, you can pick different stacks!
  - **Java Path:** Java 8+, Spring framework (cloud,data, batch, web services,...)
  - NodeJS, Go , ...
  - You're an explorer?! → Scala and Lagom
- Time, we'll watch several videos!



# Optional Books

- Free Book:
  - Microservices: From Design to Deployment from Chris Richardson and Floyd Smith
    - <https://www.nginx.com/blog/microservices-from-design-to-deployment-ebook-nginx/>
- Building Microservices | Sam Newman
- Beyond the 12 Factor App | Kevin Hoffman
- Domain driven design distilled | Vaughn Vernon.



# Grading

- 10% final
- 10% Midterm
- 20% labs (individual Mini projects), quiz ,..
- 50% group project
- Note: Understanding is our highest priority→  
I'll downgrade a grade letter up to 10% if you  
don't show a solid understanding, or good  
participation
  - *By continuing in this class you agree to these terms..*



Let's get started





How the customer explained it



How the Project Leader understood it



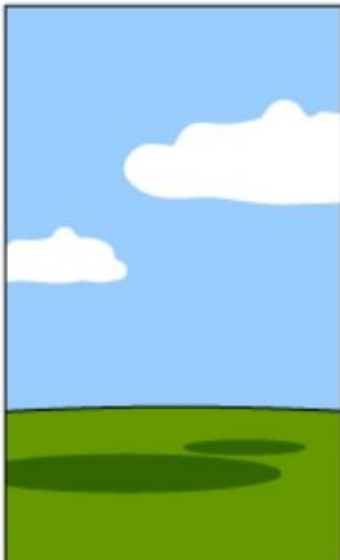
How the Analyst designed it



How the Programmer wrote it



How the Business Consultant described it



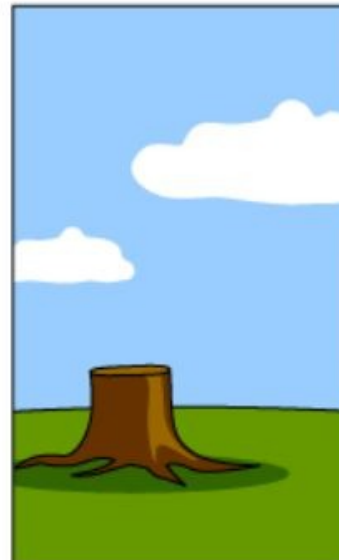
How the project was documented



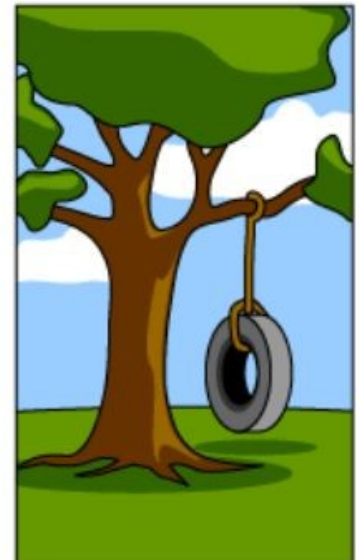
What operations installed



How the customer was billed

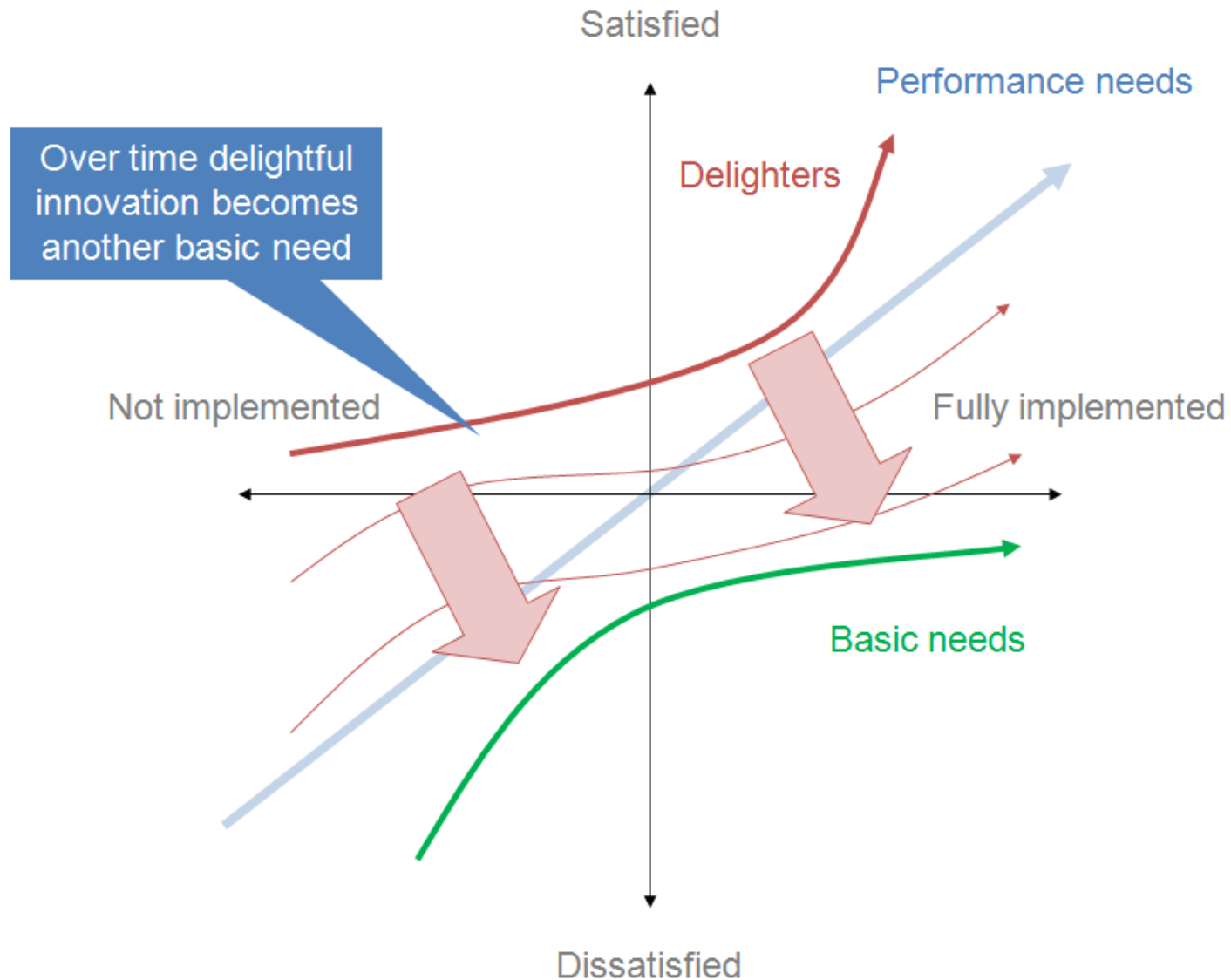


How it was supported



What the customer really needed


# The Kano Model





# The Kano Model

- Professor Noriaka Kano, a Japanese expert in customer satisfaction
  - Basic Features: Expected, frustrating when missing but not exciting when exists
  - Performance Features
  - Attractive Features: Going the extra mile!
- Read:
  - [https://articles.uie.com/kano\\_model/](https://articles.uie.com/kano_model/)



Can you think of the 3 types of features in  
a system ?



What's software Architecture ?



# From Wikipedia

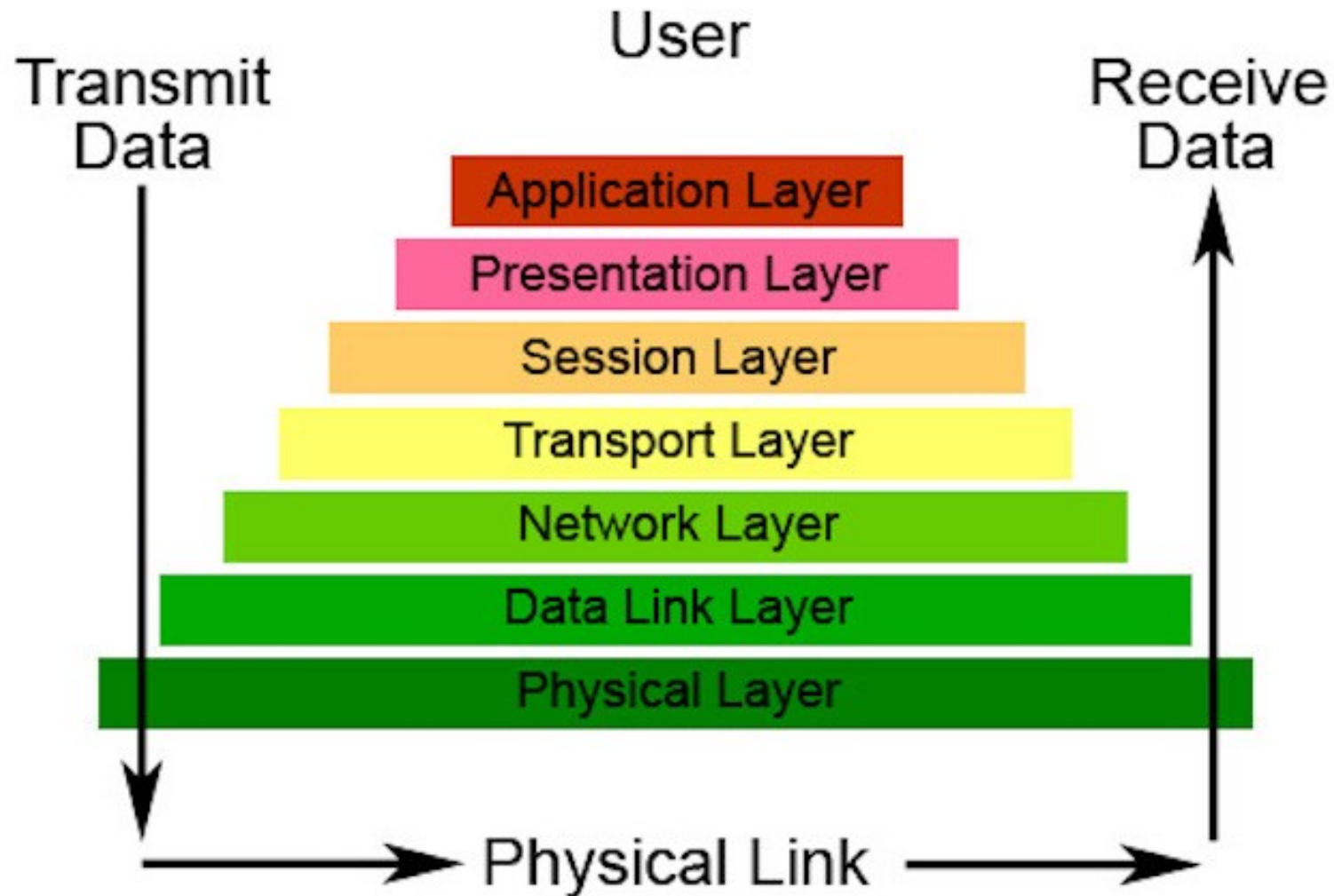
- Software architecture is about making fundamental structural choices that are costly to change once implemented.

[https://en.wikipedia.org/wiki/Software\\_architecture](https://en.wikipedia.org/wiki/Software_architecture)



What's the architecture behind the  
internet ?

# The Seven Layers of OSI





# 7 Layers of the OSI Model

## Application

- End User layer
- HTTP, FTP, IRC, SSH, DNS

## Presentation

- Syntax layer
- SSL, SSH, IMAP, FTP, MPEG, JPEG

## Session

- Synch & send to port
- API's, Sockets, WinSock

## Transport

- End-to-end connections
- TCP, UDP

## Network

- Packets
- IP, ICMP, IPsec, IGMP

## Data Link

- Frames
- Ethernet, PPP, Switch, Bridge

## Physical

- Physical structure
- Coax, Fiber, Wireless, Hubs, Repeaters



# Every arch makes compromises

- IP protocol is best effort routing (packet loss, out of order,...)
- The Internet is not secure by design!
- IPv4 isn't scalable ( NAT, IPv6 ,...)
  - But IP layer is implemented everywhere, and we can't change it easily!



# Architectural pattern

- Wiki:
  - A general, reusable solution to a commonly occurring problem in software architecture within a given context



Tell me some of the architectures that you  
know?



# Architectural patterns

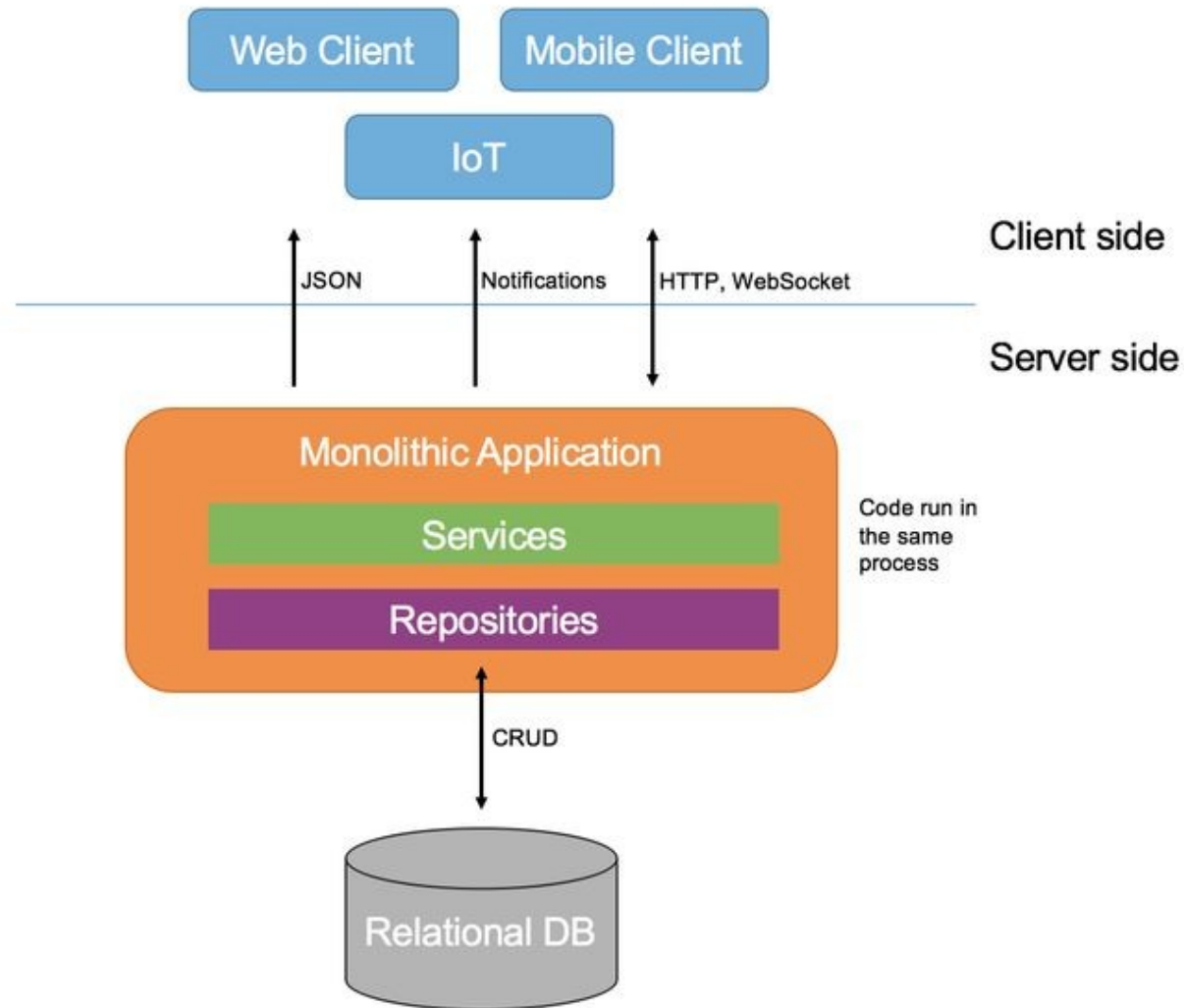
- Plugin/Micro kernel (browser plugins, IDE plugins,...)
- Layered (n-tier e.g. MVC)
- Event Driven (messages, Pub/Sub, ES,...)
- Services (SOA, Restful, Micro-services,...)
- Client-server vs Peer to peer
- Pipe and filter
- Serverless ..and a lot more!



# Let's watch

- Software Architecture | Architectural patterns  
| Architecture vs Design pattern
  - <https://www.youtube.com/watch?v=ITkL1oIMiaU>

# A sample Architecture



<https://www.pinterest.com/khoadinh290/software-architecture/>



# It's simple :)

- Simple to develop & test
- Simple to deploy. You just have to copy a jar/war to a server.
- Maybe run DB on a separate server.
- Data is consistent/transactions
- Simple method calls between different classes/modules



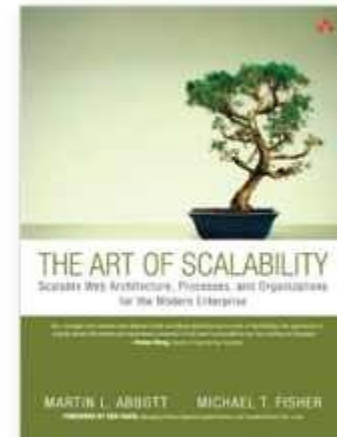
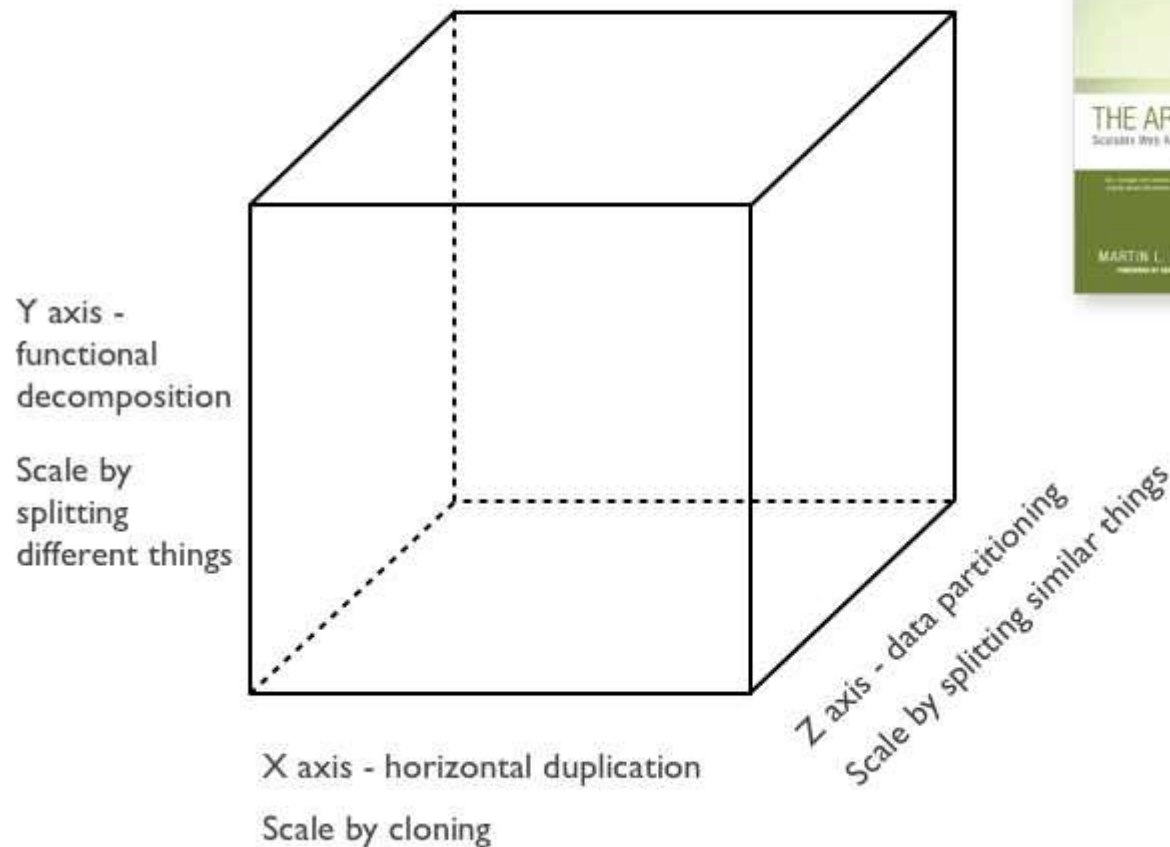


# But

- What can we do to make our single node (app or database) serve:
  - A lot more reads? Any ideas?!
  - A lot more writes ?
  - Many locations around the globe ?
- What to do when the node fails ?

# We need to scale

## 3 dimensions to scaling

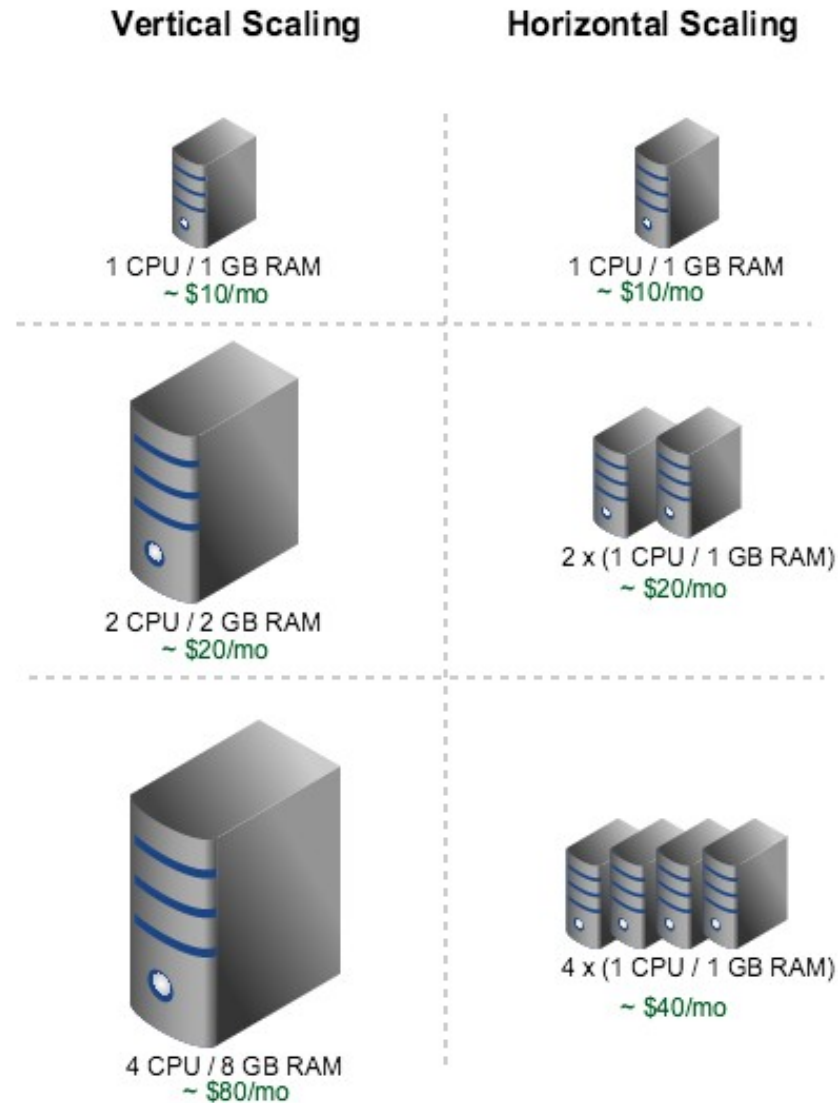




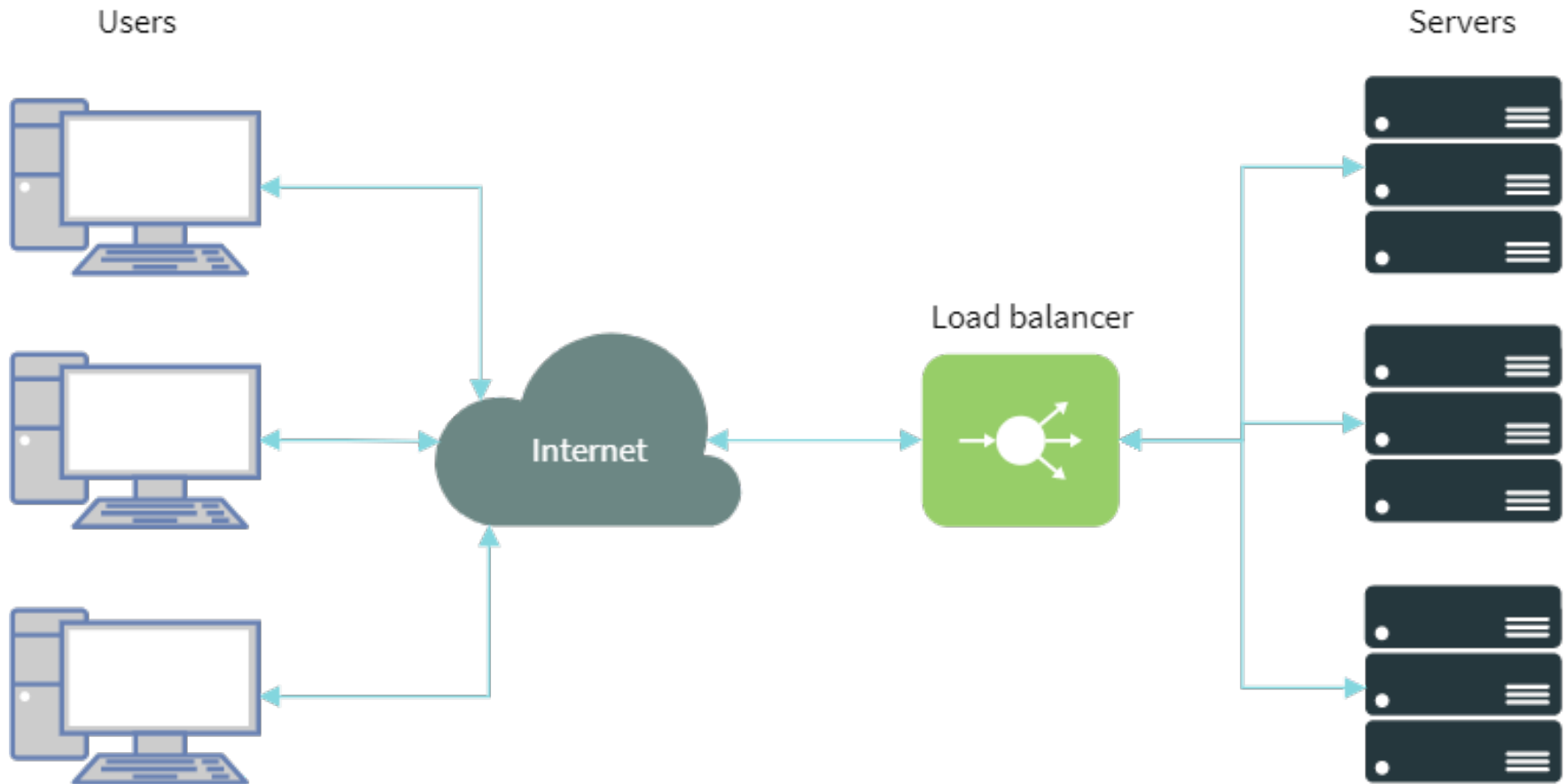
# Scaling axis

- Let's read:
  - <https://akfpartners.com/growth-blog/scaling-your-systems-in-the-cloud-akf-scale-cube-explained>

# Horizontal scaling (X-Axis)



# Load Balancer

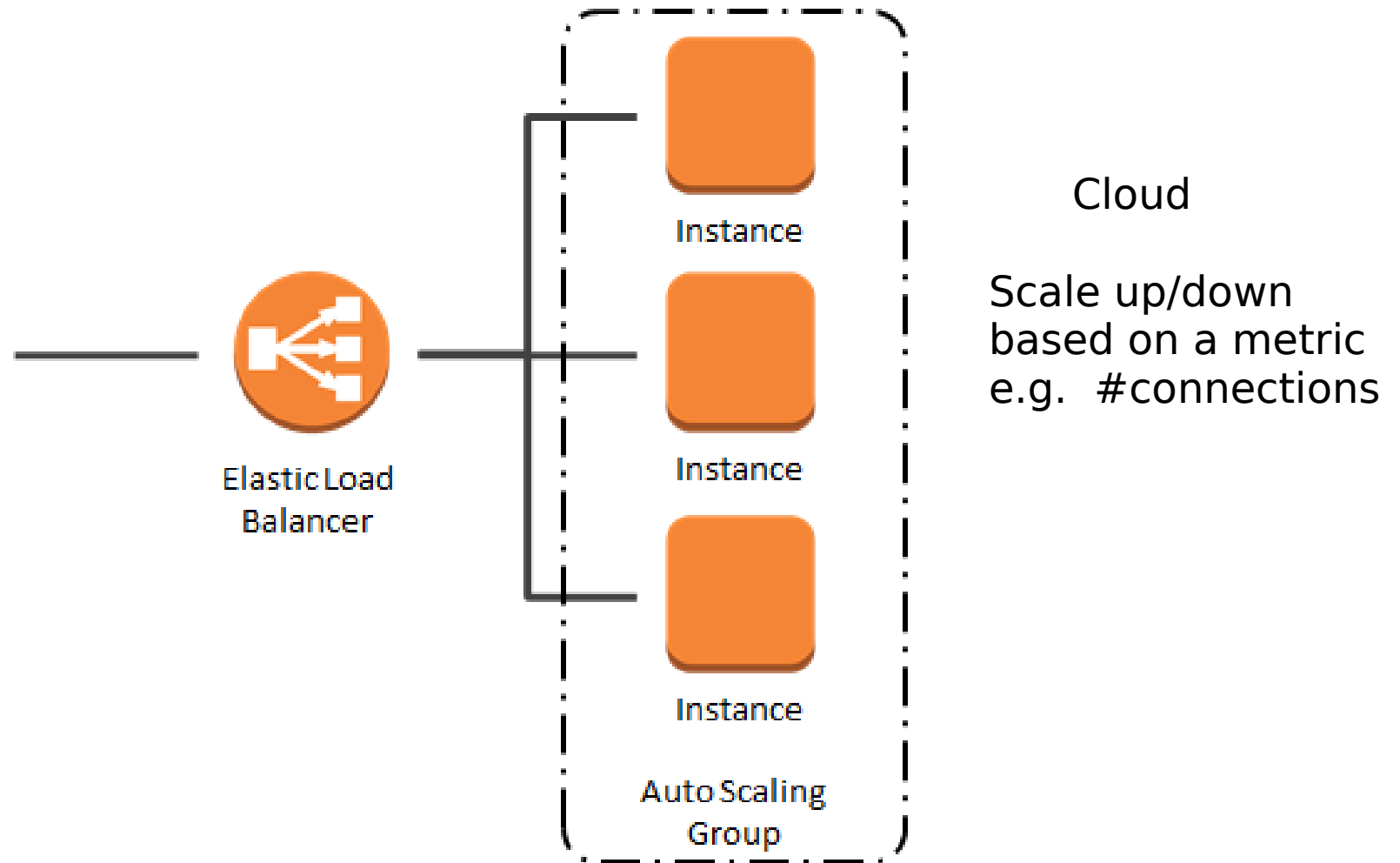




# Load Balancer

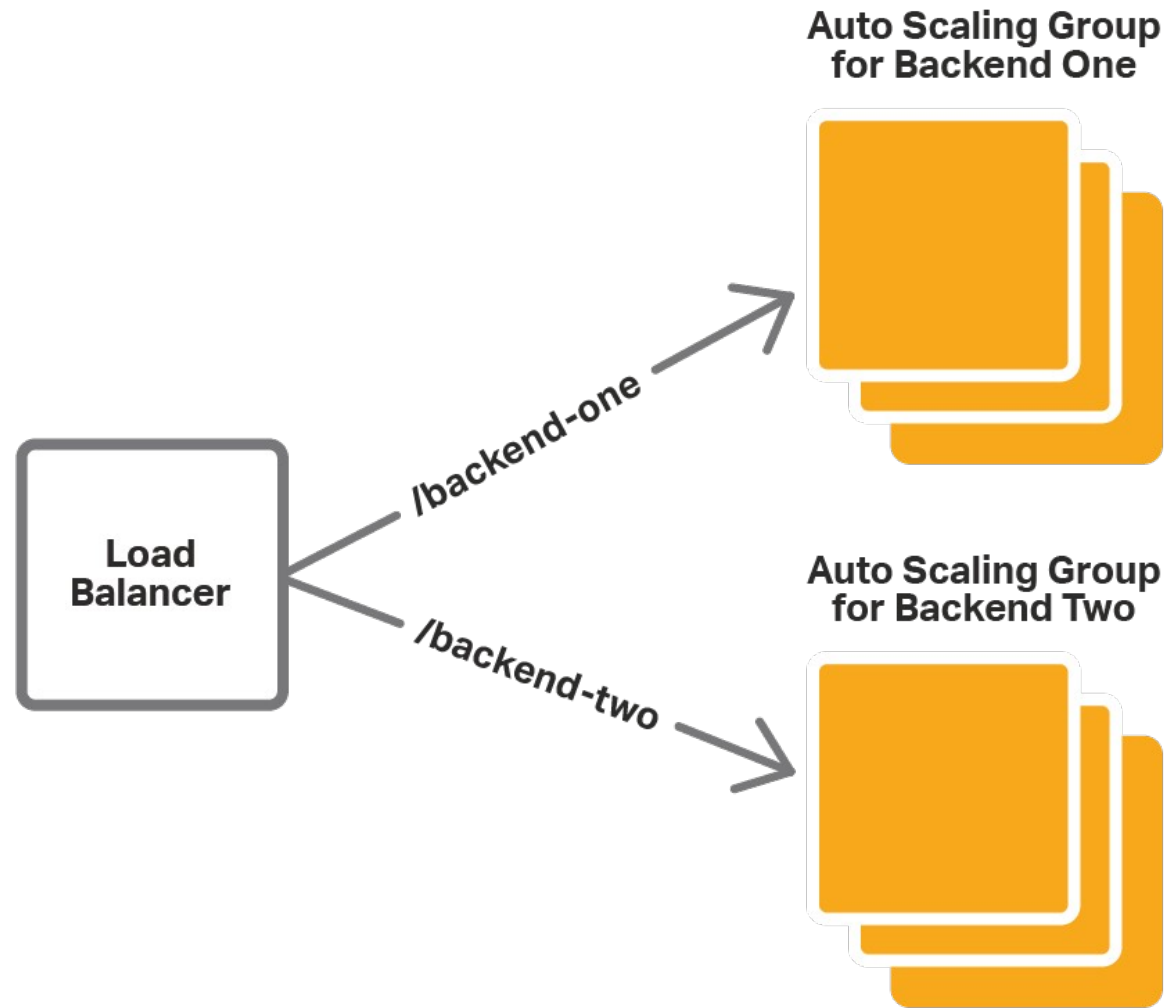
- How many servers do I need ?
- The traffic load changes a lot (during the day or seasons like black Friday/Christmas ,.. )
- It costs money to buy servers , also what if we rent → cloud ?

# Auto-scaling



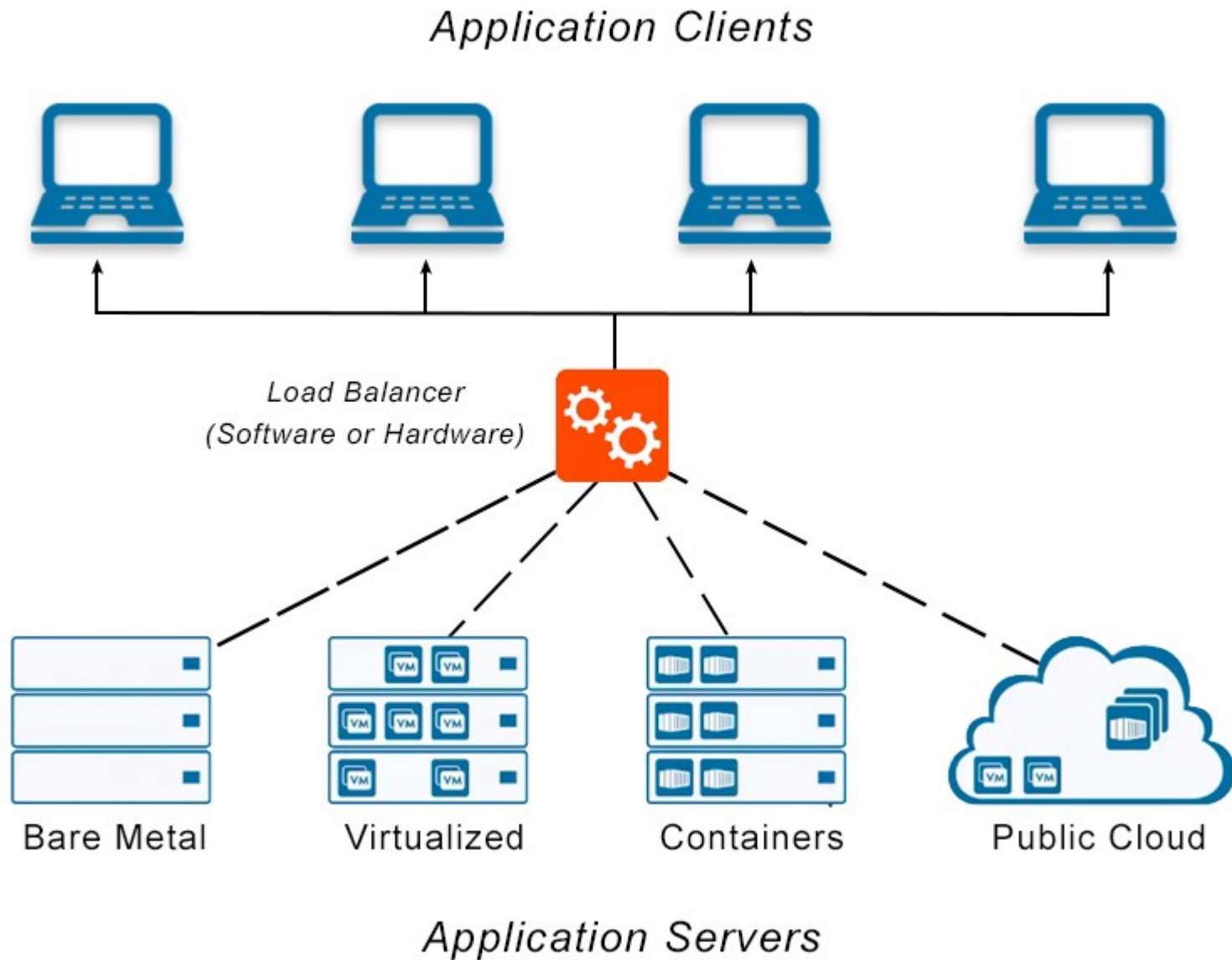
<https://aws.amazon.com/blogs/aws/elb-connection-draining-remove-instances-from-service-with-care/>

# Can Be layered





# In Real life

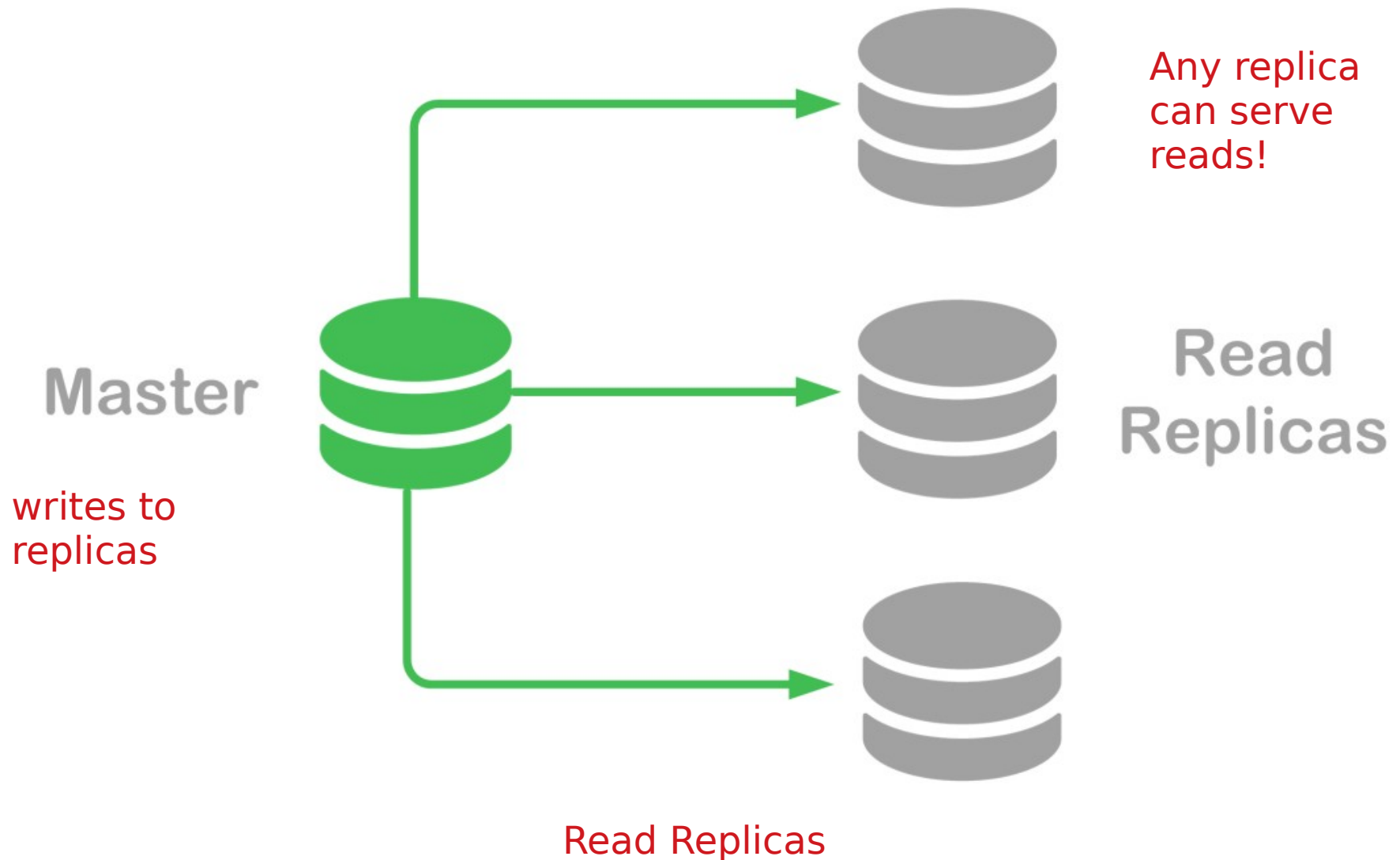




# Wait a minute

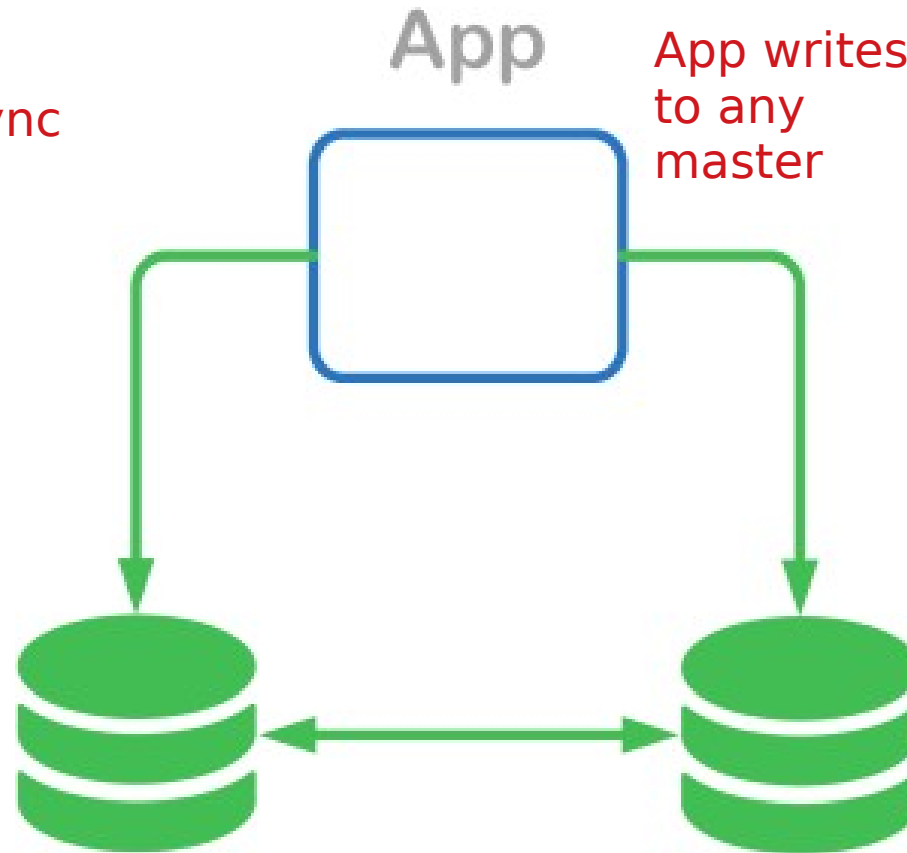
- Horizontal scaling behind LB is fine for stateless services (API servers)
- What about database server ?
  - We must scale it , but LB won't work :(

# serve more reads?



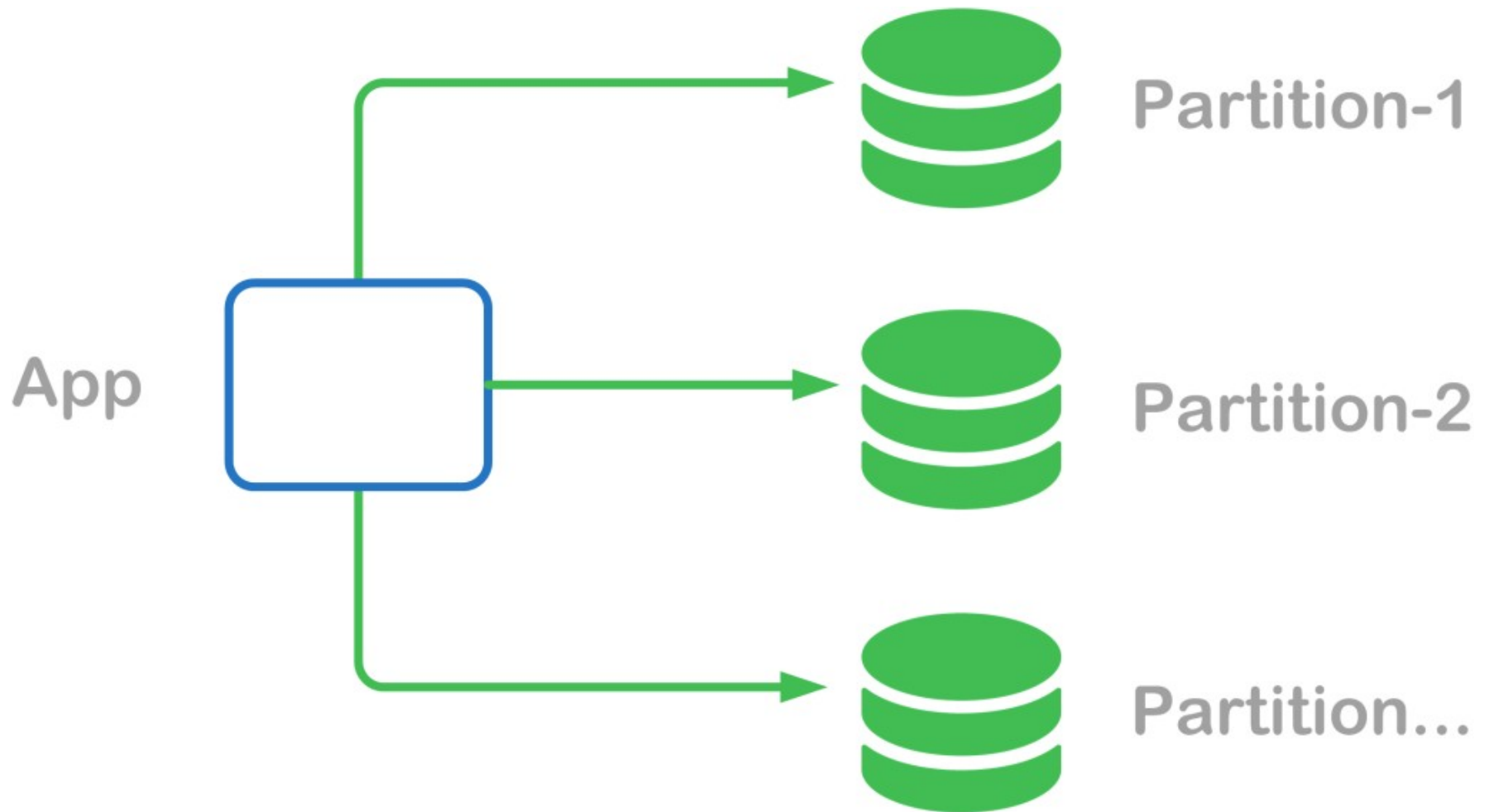
# Serve more writes ?

Collisions during sync  
Not Simple :(



Multi-Master

# Store more data



Partitioning



# Sharding

- Select an attribute to shard with
- A shard is determined by hash value
- Can cause skews
  - e.g. shard users by country\_code
    - Think China and sweden!



# Read more

- There are better options, we'll cover later
- Check:
  - <http://realscale.cloud66.com/database-server-scaling-strategies/>



# What we just did?

- We made several servers act like they are just one service to the client
  - Distributed system!





# No big deal

- Let's use more servers and connect them together..
  - What can go wrong?!



# Fallacies of Distributed computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.



# Fallacies of Distributed computing

- Got some time? , check
  - <https://medium.com/baseds/foraging-for-the-fallacies-of-distributed-computing-part-1-1b35c3b85b53>



# Our objectives

- Depends on the problem
  - Scale across the globe
  - High availability
  - Resiliency, disaster recovery and fail over
  - Low latency
  - Security
  - Cost
  - ....



# Many things to consider

- Computation and Storage
- Discovery and Communication
- Configuration management
- Storing secrets
- Deployment
- logging/monitoring

We need to understand  
our problem and make the  
right compromises !



# Let's watch

- Pat Helland | Kafka Summit 2017 Keynote (Standing on the Distributed Shoulders of Giants)
  - <https://www.youtube.com/watch?v=p9LBi11KR2c>