

# The World of Back-end Development

K. Yuttapichai (PnXGUIDE)  
[pxguide.dev](http://pxguide.dev)

# Who am I?

- Prospective PhD Student in Electrical and Computer Engineering (ECE) at Carnegie Mellon University – CMKL
- Working as a **Full-Stack Developer**
- Concentrate on Big Data Management and Intelligence Applications for Big Data
  - Also interested in Immersive Technologies, Game Design and Development, Deep Learning and Neural Networks
- Japanese animation, rhythm game , FEVER lover (also, a seal lover ❤)



# Agenda

Database  
Programming



API  
Development

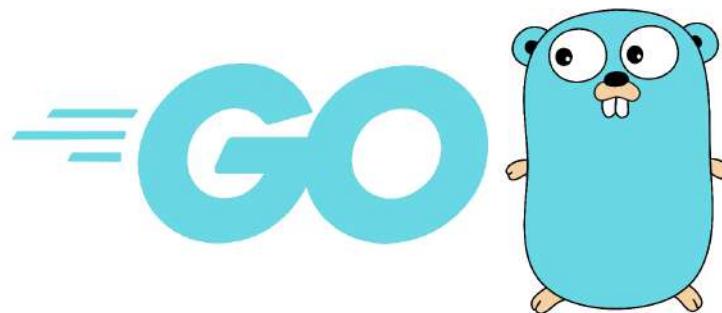
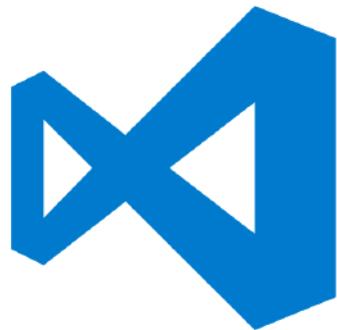


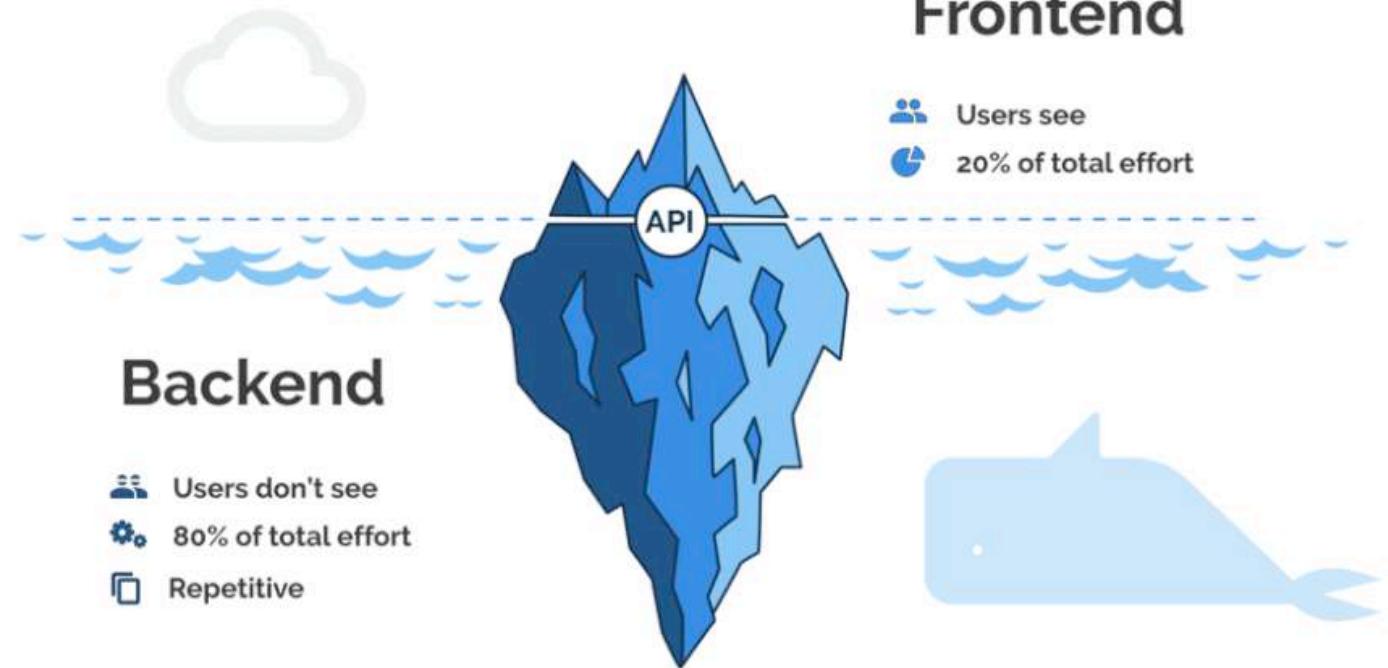
Other  
Practices



# Prerequisites

- Code editor (Visual Studio Code is recommended.)
- Go 1.14 (Golang)
- MariaDB Server version 10.5.7
- Postman



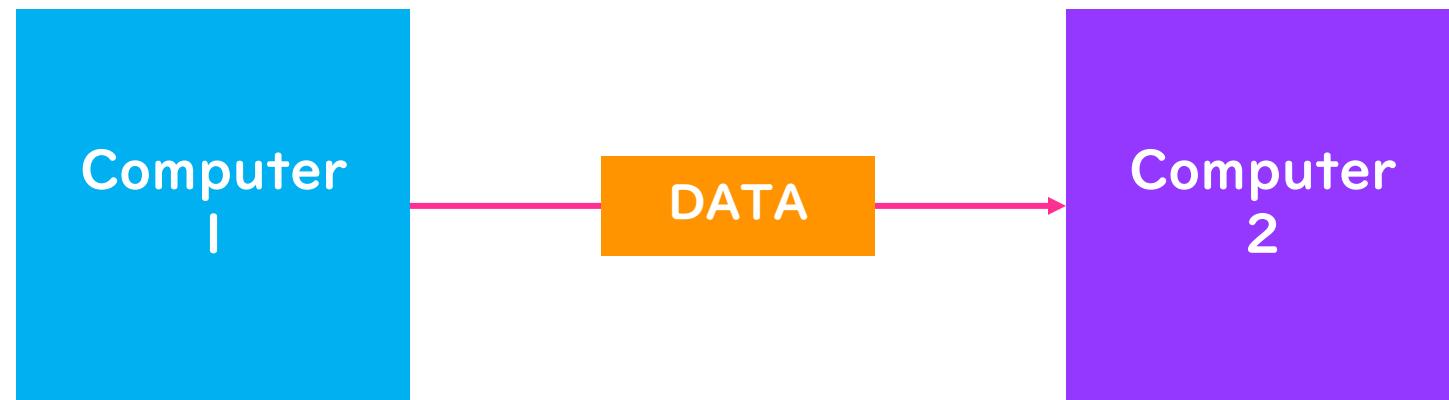


# Internet Fundamentals



# Data Transmission

- How do a computer send data to another computer?



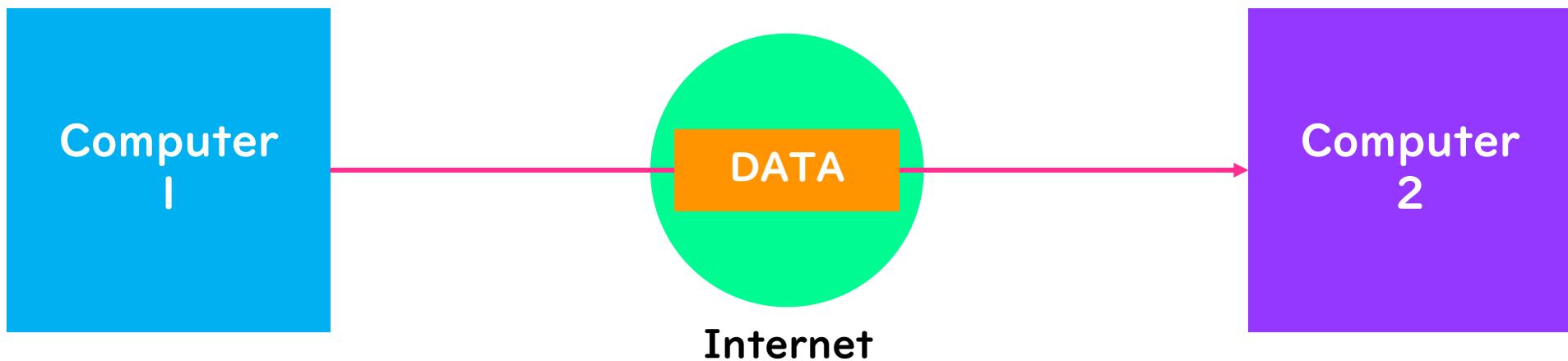
# Data Transmission

- Via hardware (portable storage devices)



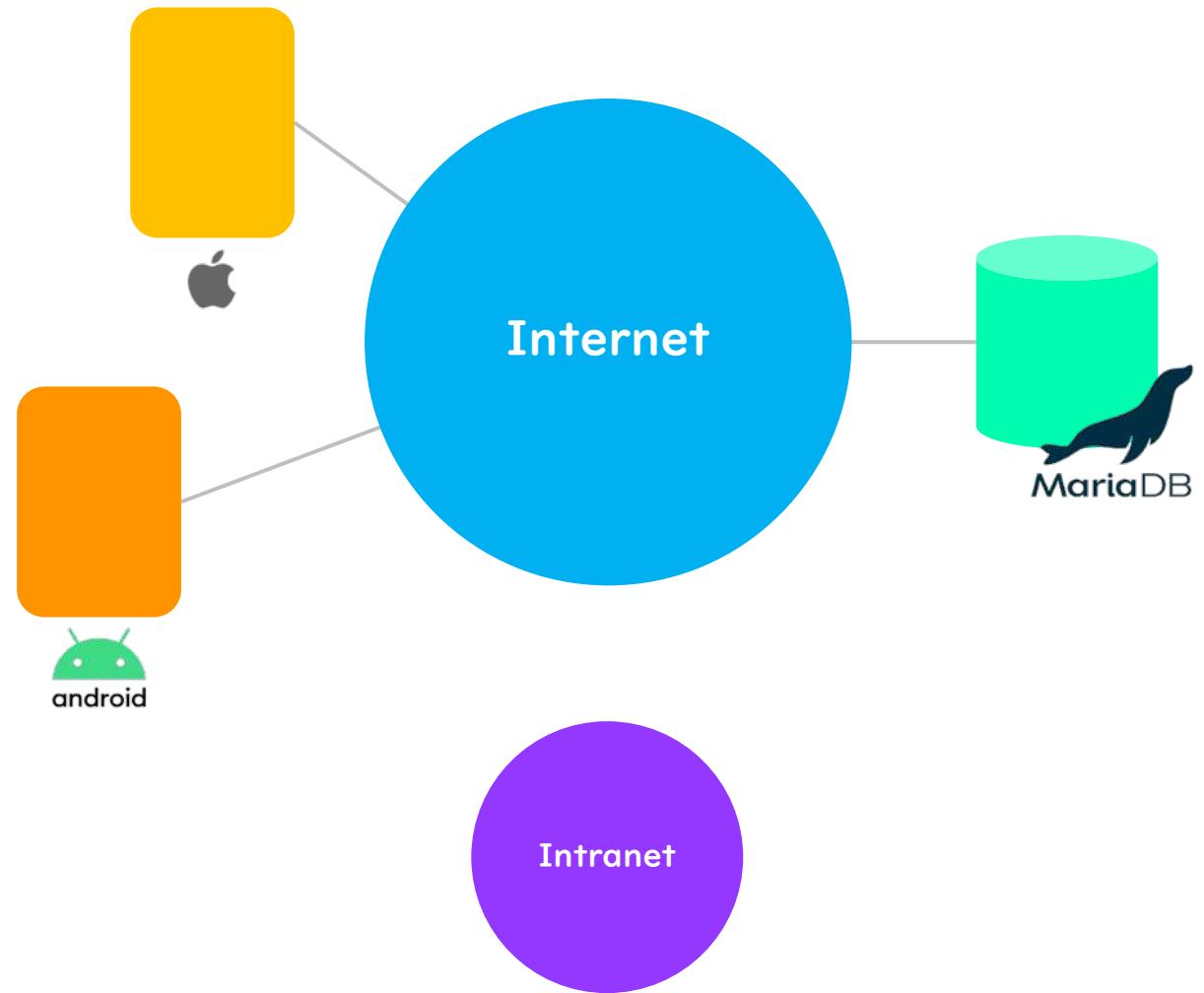
# Data Transmission

- Via networks (mostly internet)

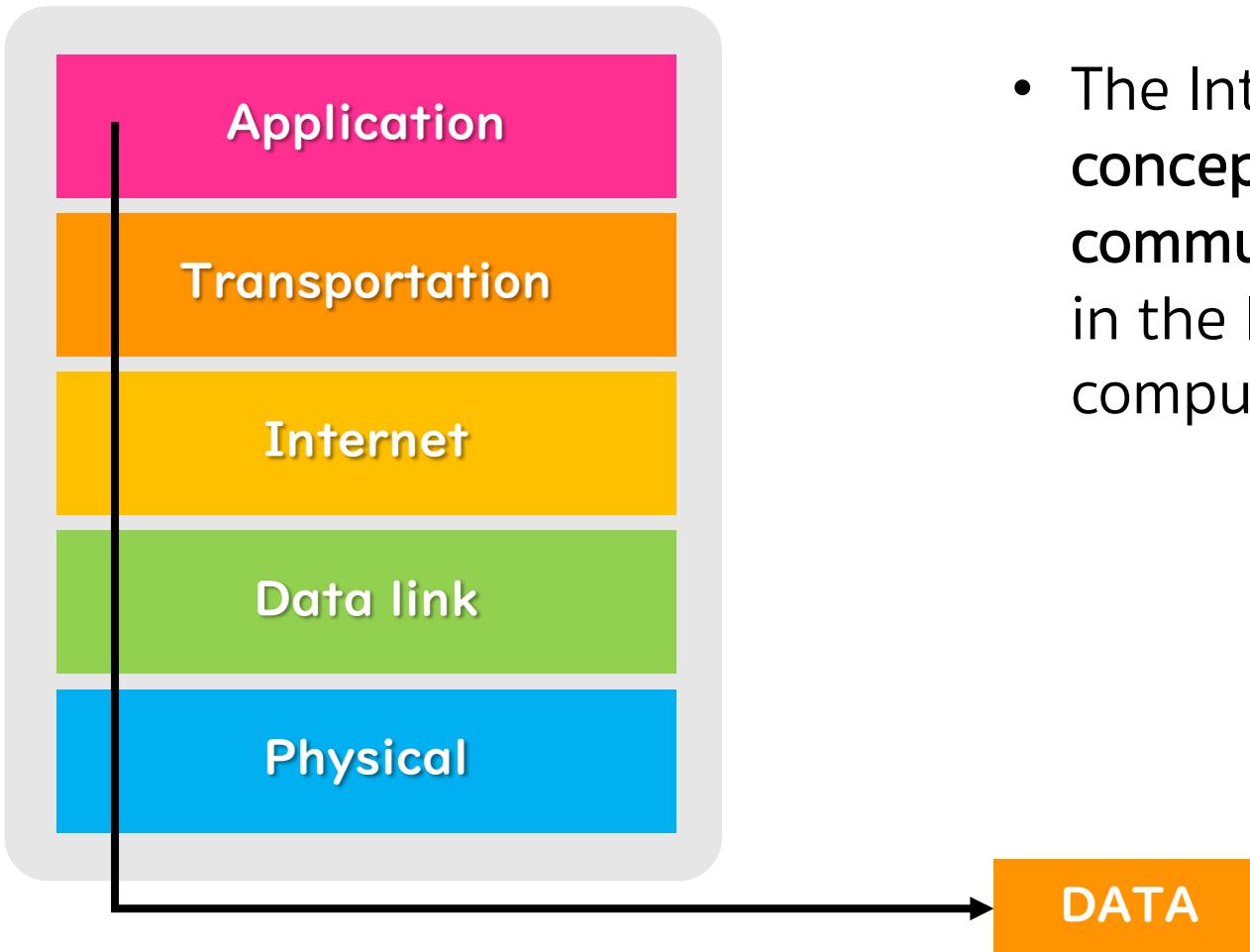


# Internet

- Internet is the global system of interconnected computer networks that uses the Internet protocol suite (TCP/IP) to communicate between networks and devices.

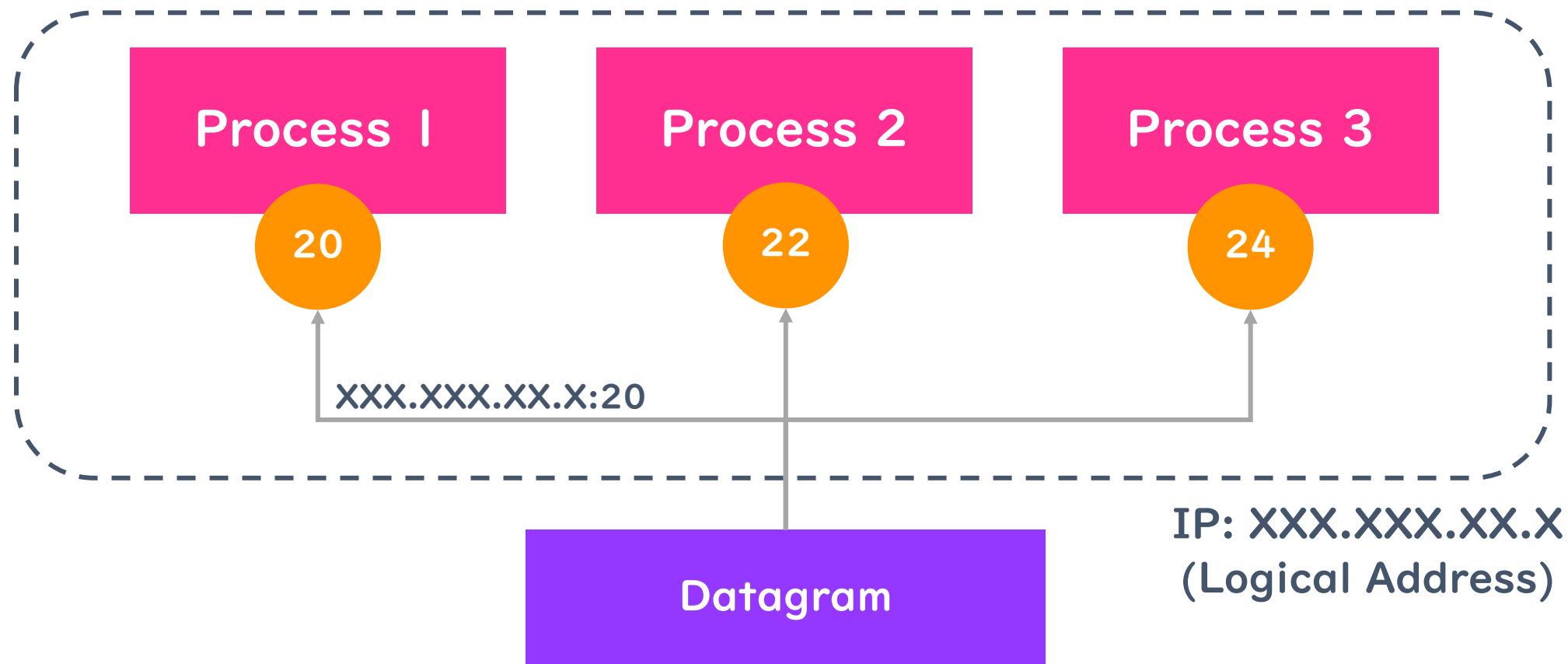


# Internet Protocol Suite



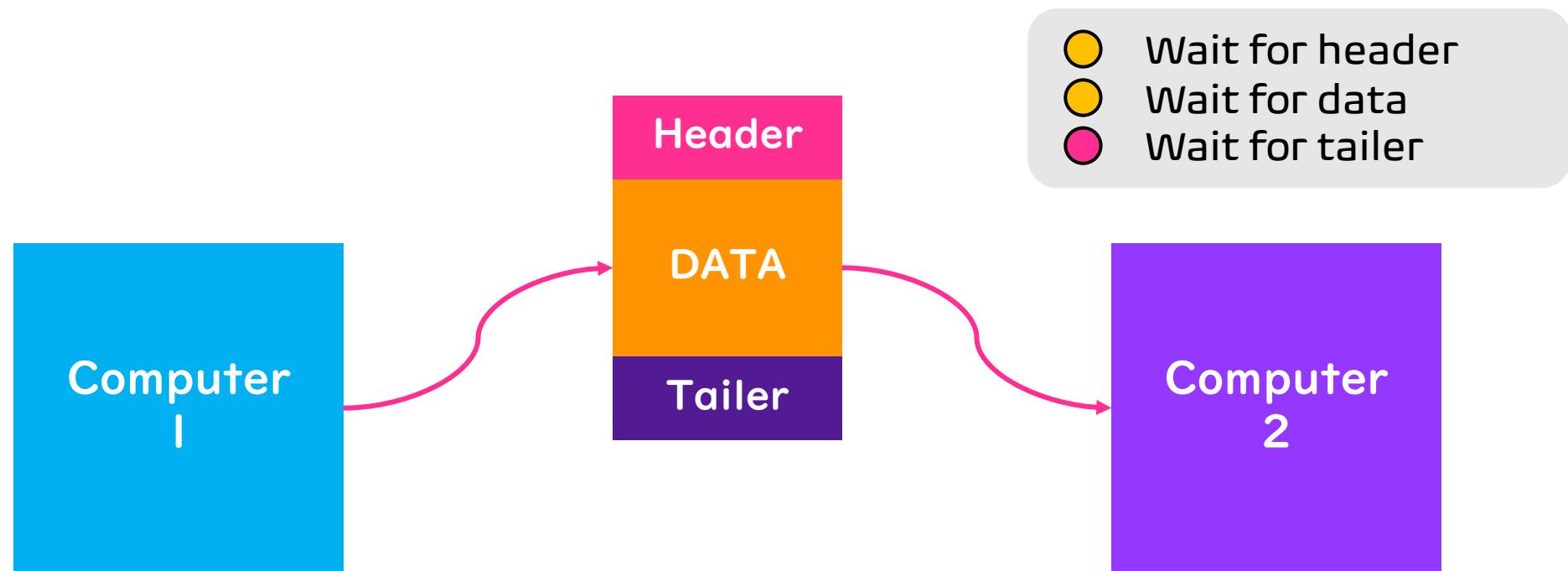
- The Internet protocol suite is **the conceptual model and set of communications protocols** used in the Internet and similar computer networks.

# Addressing



# Protocol

- A protocol is a standard set of rules that allow electronic devices to communicate with each other.



# Application-Layer Protocols

Protocol Name	Description
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol <b>Secure</b>
FTP	File Transfer Protocol
DHCP	Dynamic Host Configuration Protocol
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell

# Hypertext Transfer Protocol

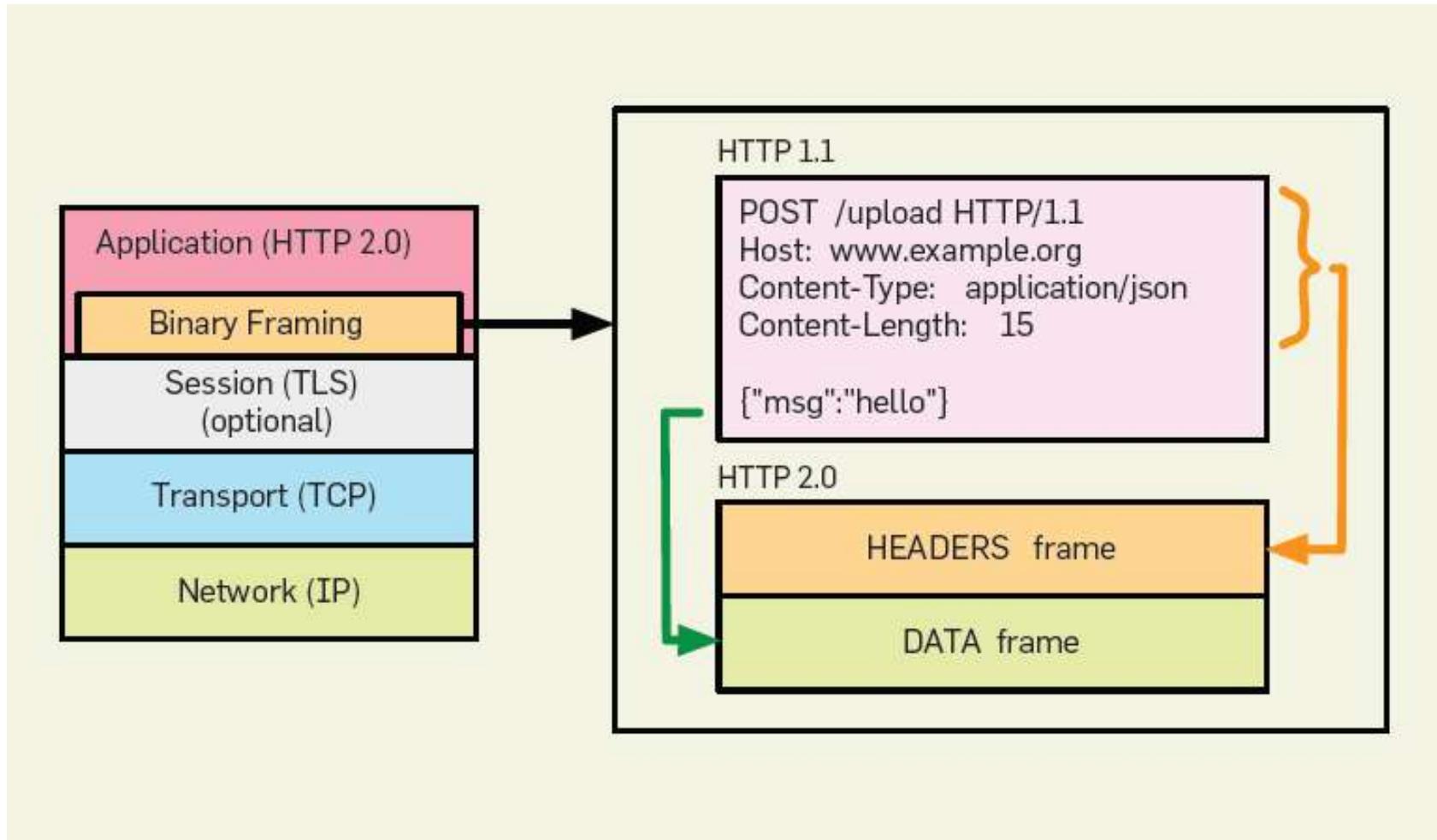
- The Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, hypermedia information systems.

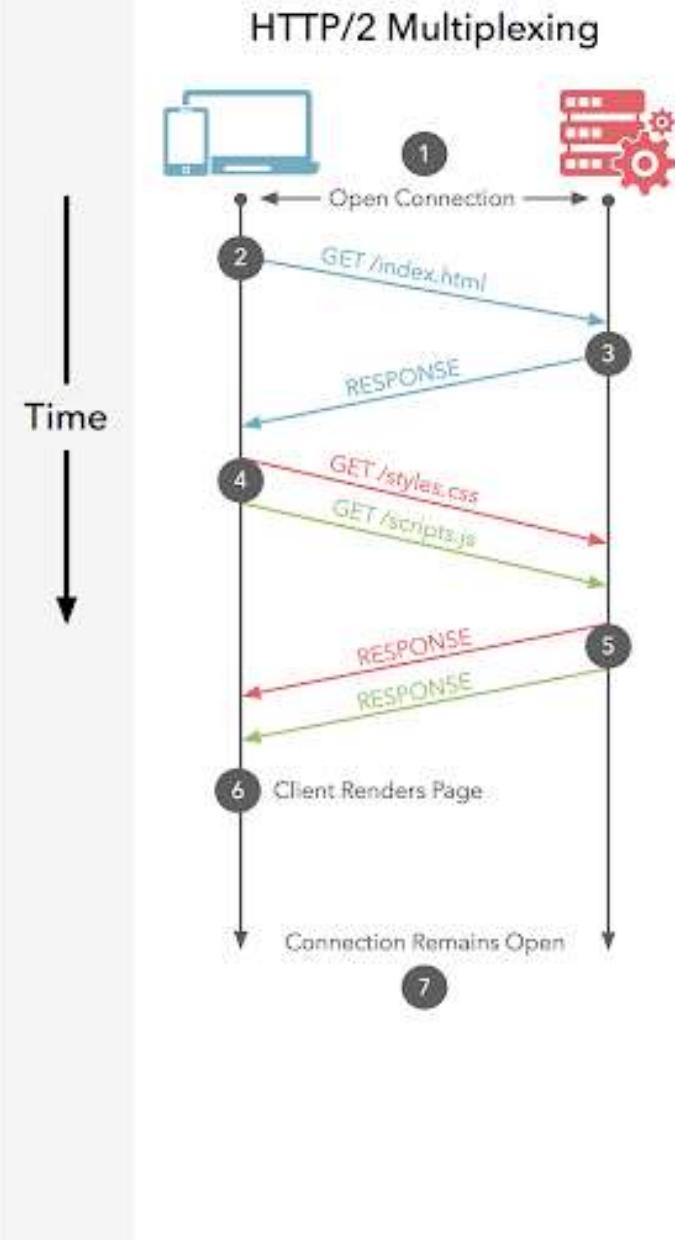
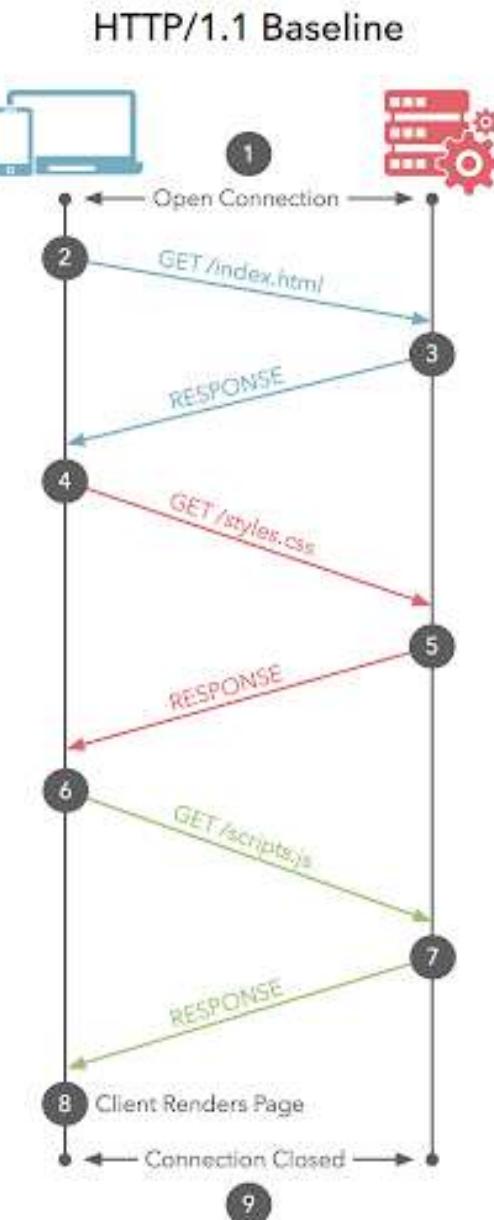
```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;... )... Firefox/51.0
Accept: text/html,application/xhtml+xml,...,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

The diagram illustrates the structure of an HTTP request message. The message is divided into three colored sections: Request headers (red), General headers (green), and Entity headers (blue). Arrows point from the labels to their respective sections.

- Request headers: The first section, colored red, contains headers specific to the request, such as Host, User-Agent, Accept, Accept-Language, Accept-Encoding, Connection, Upgrade-Insecure-Requests, Content-Type, and Content-Length.
- General headers: The second section, colored green, contains headers that apply to both requests and responses, such as Connection and Upgrade-Insecure-Requests.
- Entity headers: The third section, colored blue, contains headers related to the entity being transferred, such as Content-Type and Content-Length.





# Transport-Layer Protocols

Protocol Name	Description
TCP	Transmission Control Protocol (Connection Oriented)
UDP	User Datagram Protocol (Connectionless or Best Effort)

# TCP



# UDP



# Connection-Oriented vs. Connectionless

- Connection oriented protocols provide many data transmission supports, for instance, flow control, error control, congestion control, and many more.
  - However, the more supports the protocol gives, the **more overheads** the transmission emits.
- On the other hands, connectionless (or best-effort) protocols do not provide any transmission control.

# Flow Control vs. Congestion Control (Data Link)

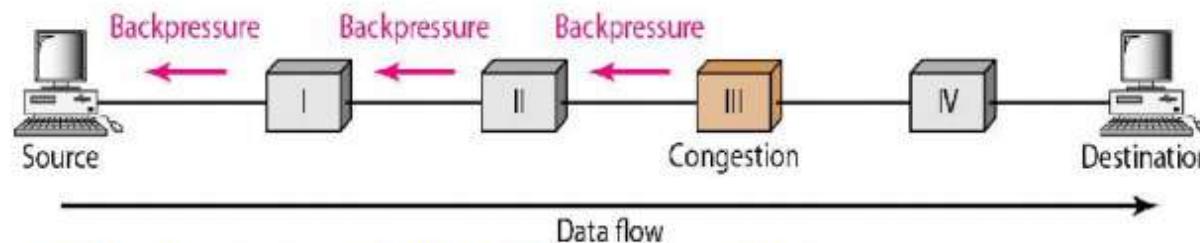
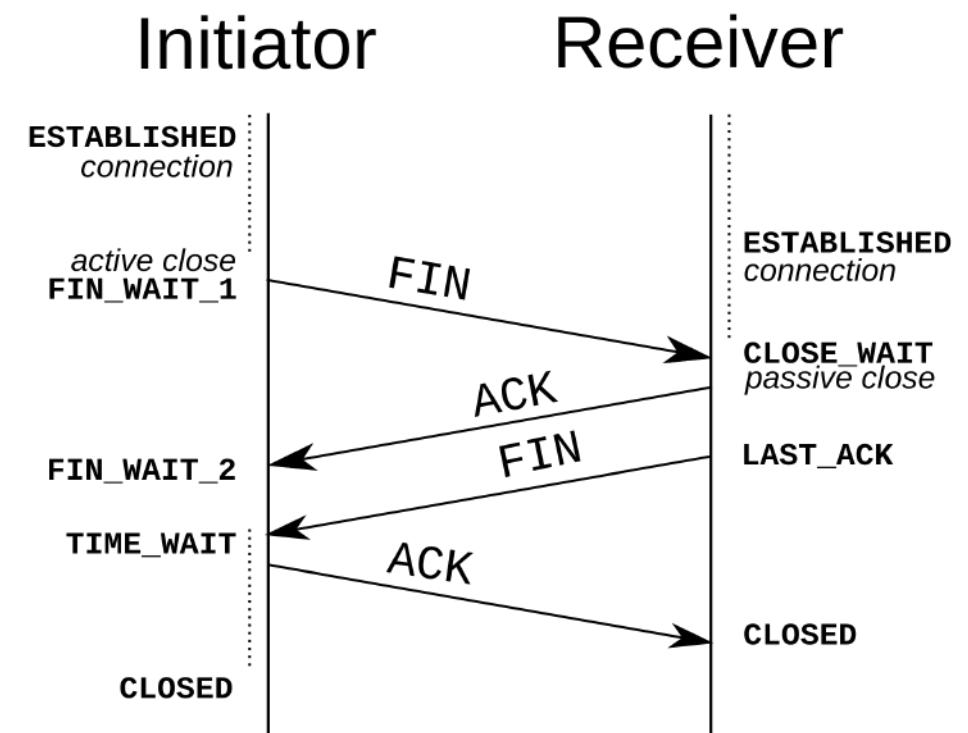
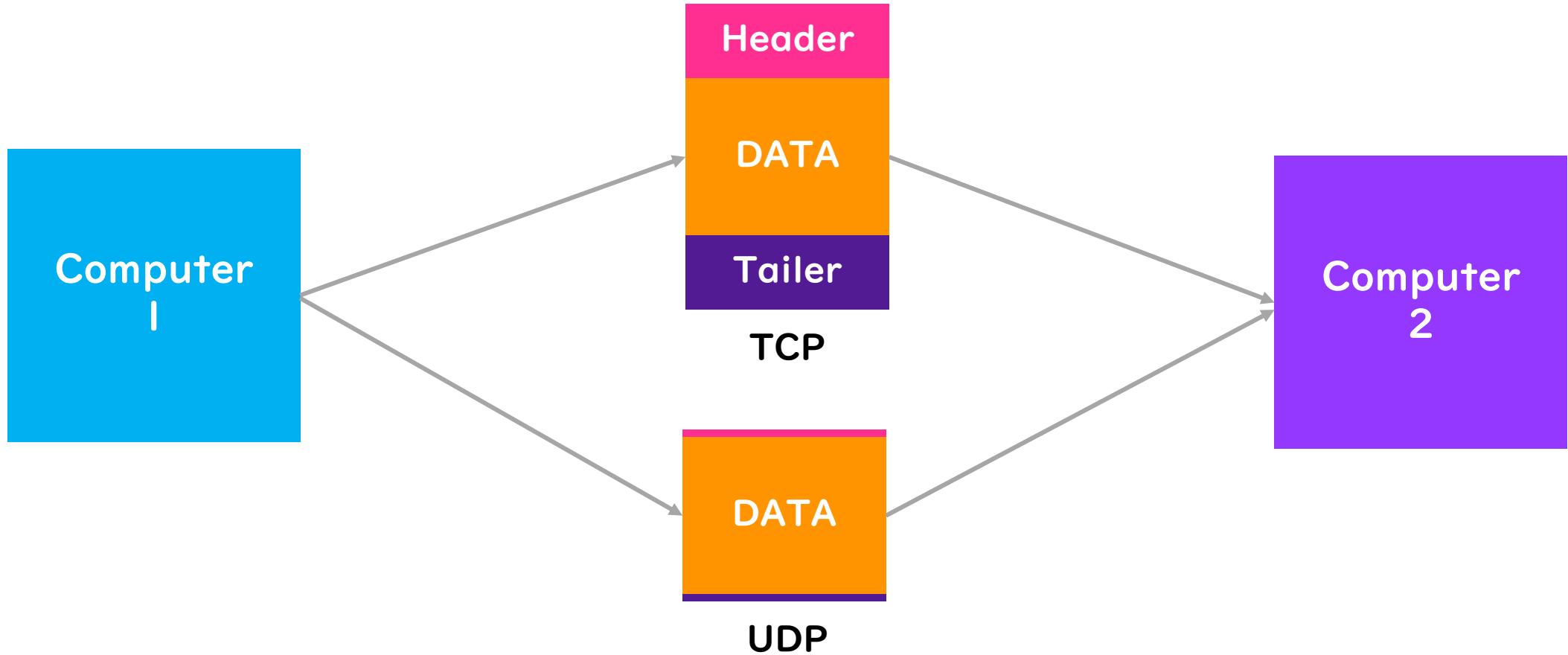


Figure 4.28 Backpressure method for alleviating congestion

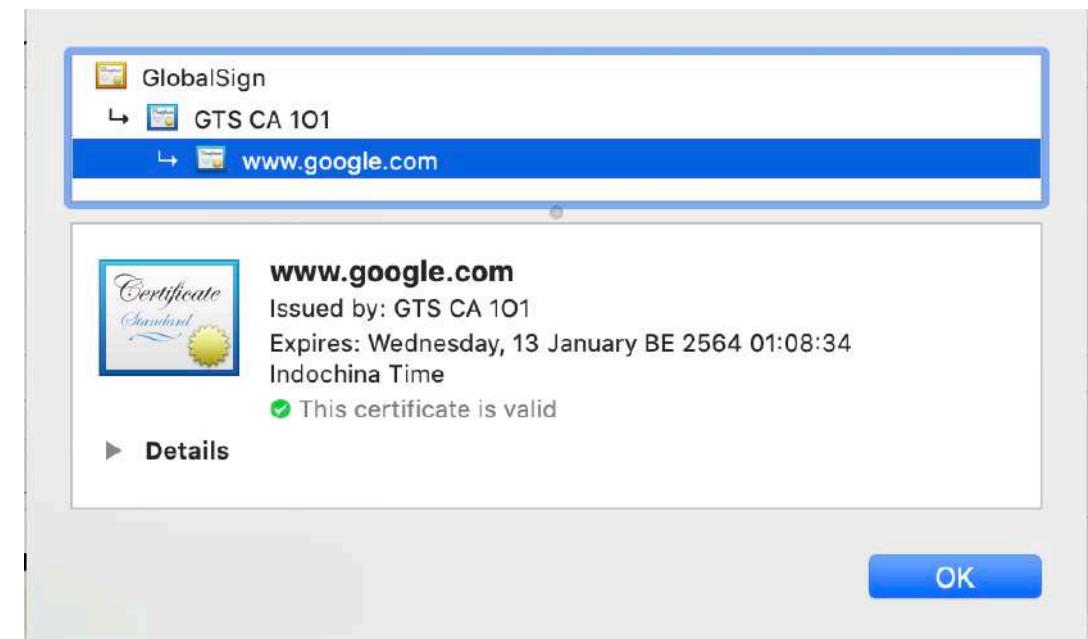
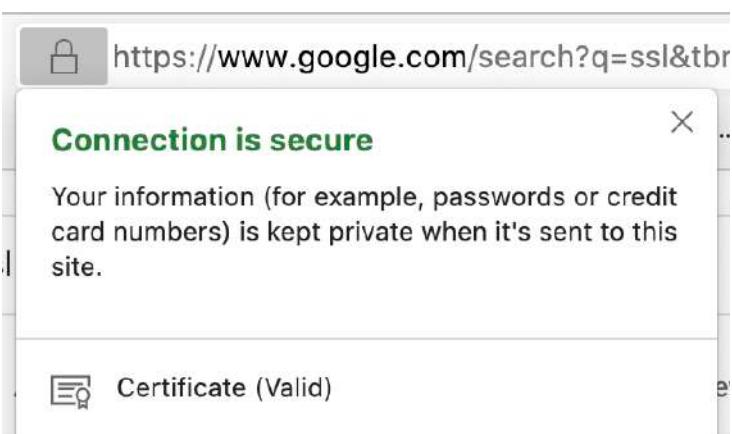


# Connection-Oriented vs. Connectionless



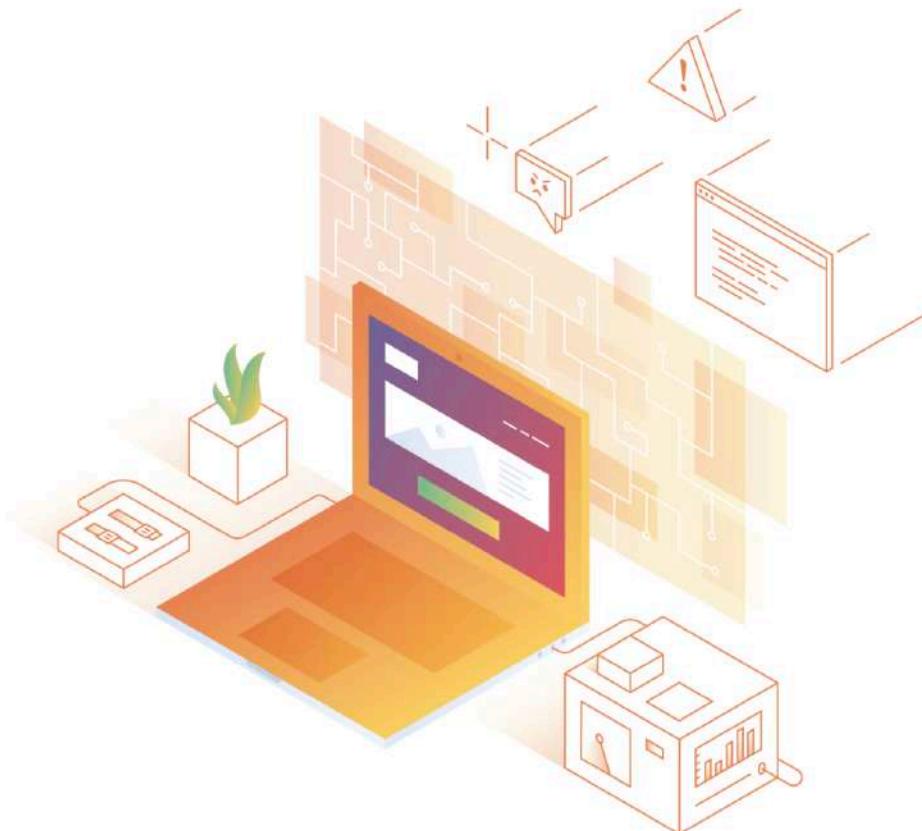
# Secure Socket Layer (SSL)

- Secure Sockets Layer (SSL) are cryptographic protocols designed to provide communications security over a computer network.



## Cloudflare Free SSL/TLS

Encrypting as much web traffic as possible to prevent data theft and other tampering is a critical step toward building a safer, better Internet. We're proud to be the first Internet performance and security company to offer SSL protection free of charge.

[Sign Up](#)[Contact Sales](#)

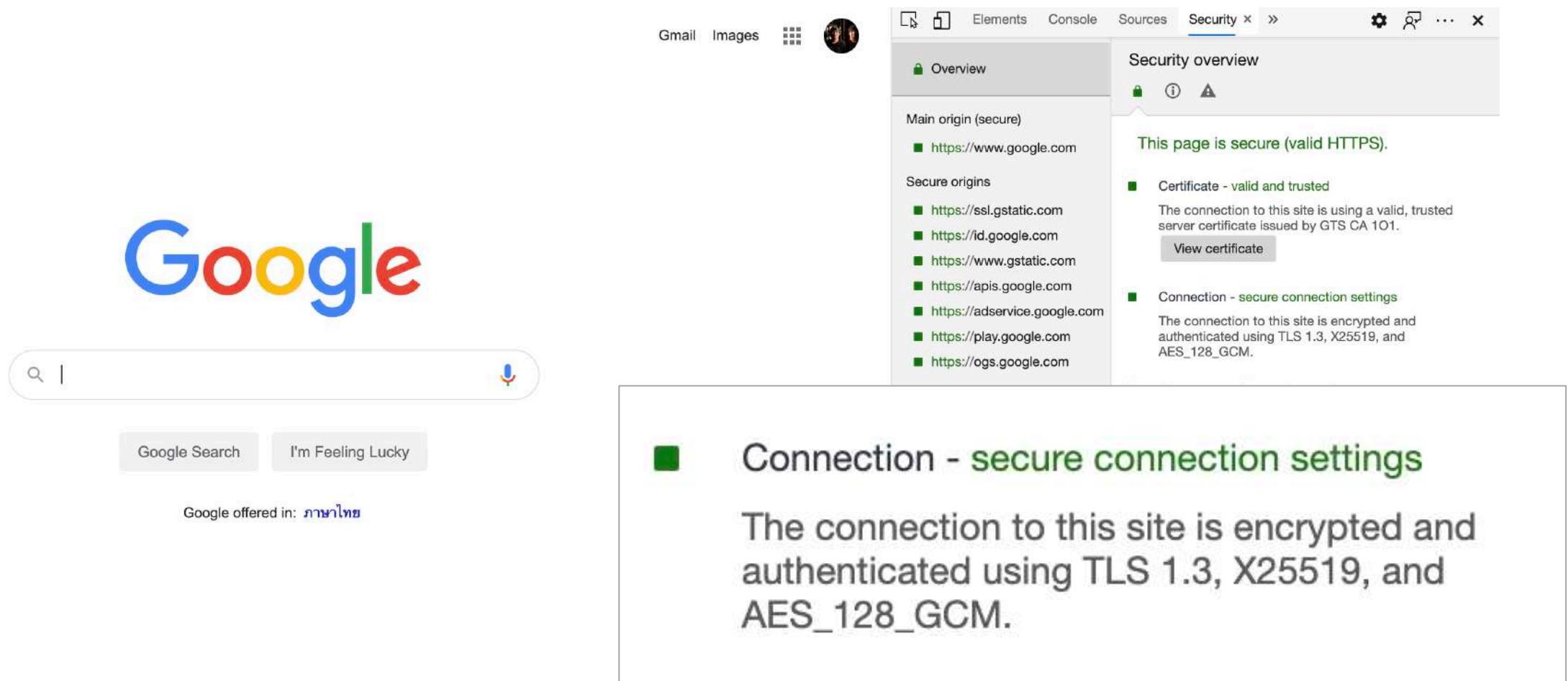
# HTTPS

- Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP).
- In HTTPS, the communication protocol is encrypted using **Transport Layer Security (TLS)** or, formerly, **Secure Sockets Layer (SSL)**.

### SSL and TLS protocols

Protocol	Published	Status
SSL 1.0	Unpublished	Unpublished
SSL 2.0	1995	Deprecated in 2011 ( <a href="#">RFC 6176</a> )
SSL 3.0	1996	Deprecated in 2015 ( <a href="#">RFC 7568</a> )
TLS 1.0	1999	Deprecated in 2020 <sup>[11][12][13]</sup>
TLS 1.1	2006	Deprecated in 2020 <sup>[11][12][13]</sup>
TLS 1.2	2008	
TLS 1.3	2018	

# Transportation Layer Security (TLS)



The screenshot shows a web browser window with the Google homepage loaded. The address bar shows "https://www.google.com". The browser's developer tools are open, specifically the "Security" tab under the "Elements" section. The "Overview" panel is selected, displaying information about the secure connection.

**Main origin (secure)**

- https://www.google.com

**Secure origins**

- https://ssl.gstatic.com
- https://id.google.com
- https://www.gstatic.com
- https://apis.google.com
- https://adservice.google.com
- https://play.google.com
- https://ogs.google.com

**This page is secure (valid HTTPS).**

**Certificate - valid and trusted**  
The connection to this site is using a valid, trusted server certificate issued by GTS CA 101.  
[View certificate](#)

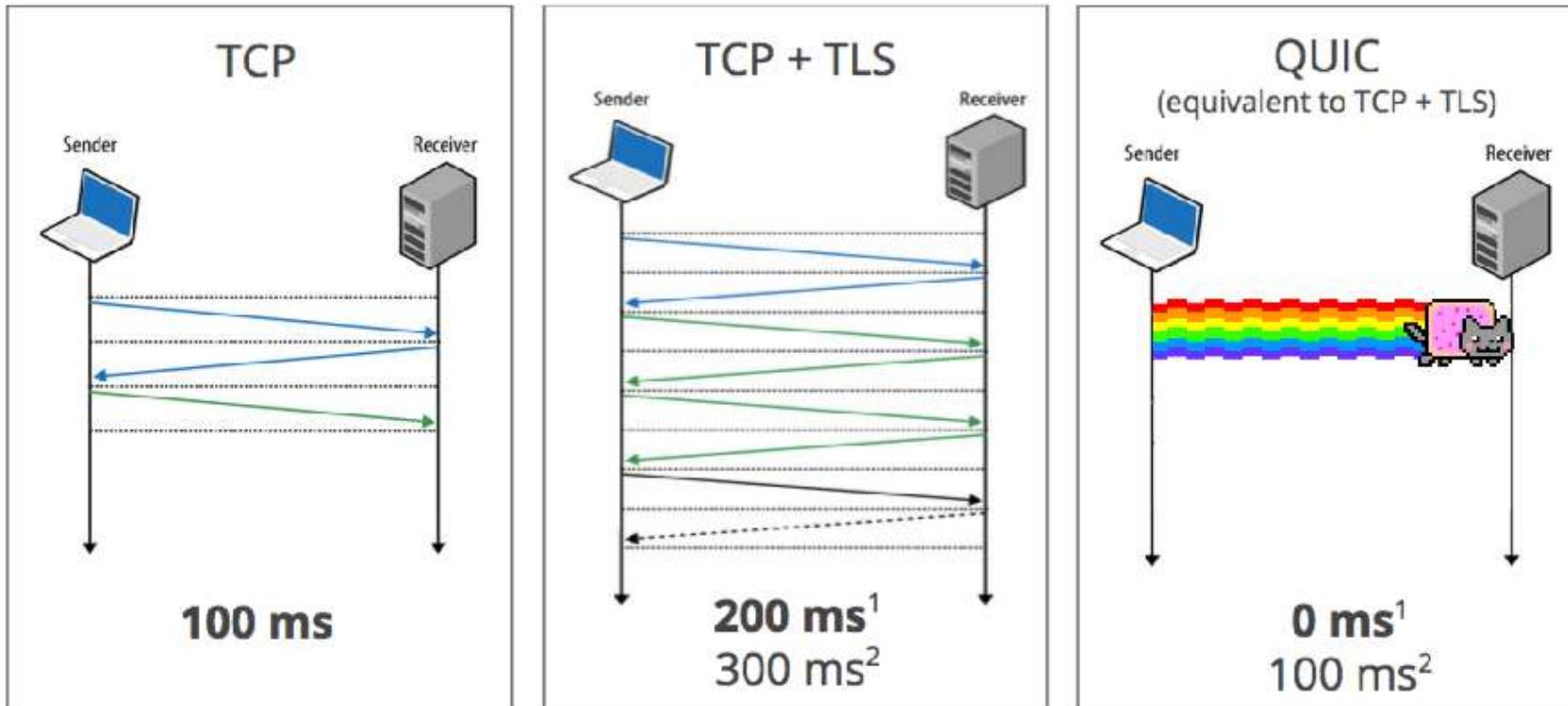
**Connection - secure connection settings**  
The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES\_128\_GCM.

■ **Connection - secure connection settings**

The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES\_128\_GCM.

# QUIC

## Zero RTT Connection Establishment



# QUIC

The screenshot shows a browser window displaying the Google homepage (<https://www.google.com>). A developer tools sidebar is open, specifically the 'Security' tab under the Network panel. The main content area displays a green lock icon and the text "This page is secure (valid HTTPS)". Below this, three items are listed: "Certificate - valid and trusted", "Connection - secure connection settings", and "Resources - all served securely". Each item has a descriptive text block and a "View certificate" button.

Google

Google Search I'm Feeling Lucky

Google offered in: ภาษาไทย

Elements Console Sources Network Performance Memory Application Security Lighthouse

Overview

Main origin (secure)

- https://www.google.com

Secure origins

- https://id.google.com
- https://www.gstatic.com
- https://apis.google.com
- https://adservice.google.com
- https://ogs.google.com

Unknown / canceled

- https://lh3.googleusercontent.com
- https://fonts.gstatic.com

Security overview

This page is secure (valid HTTPS).

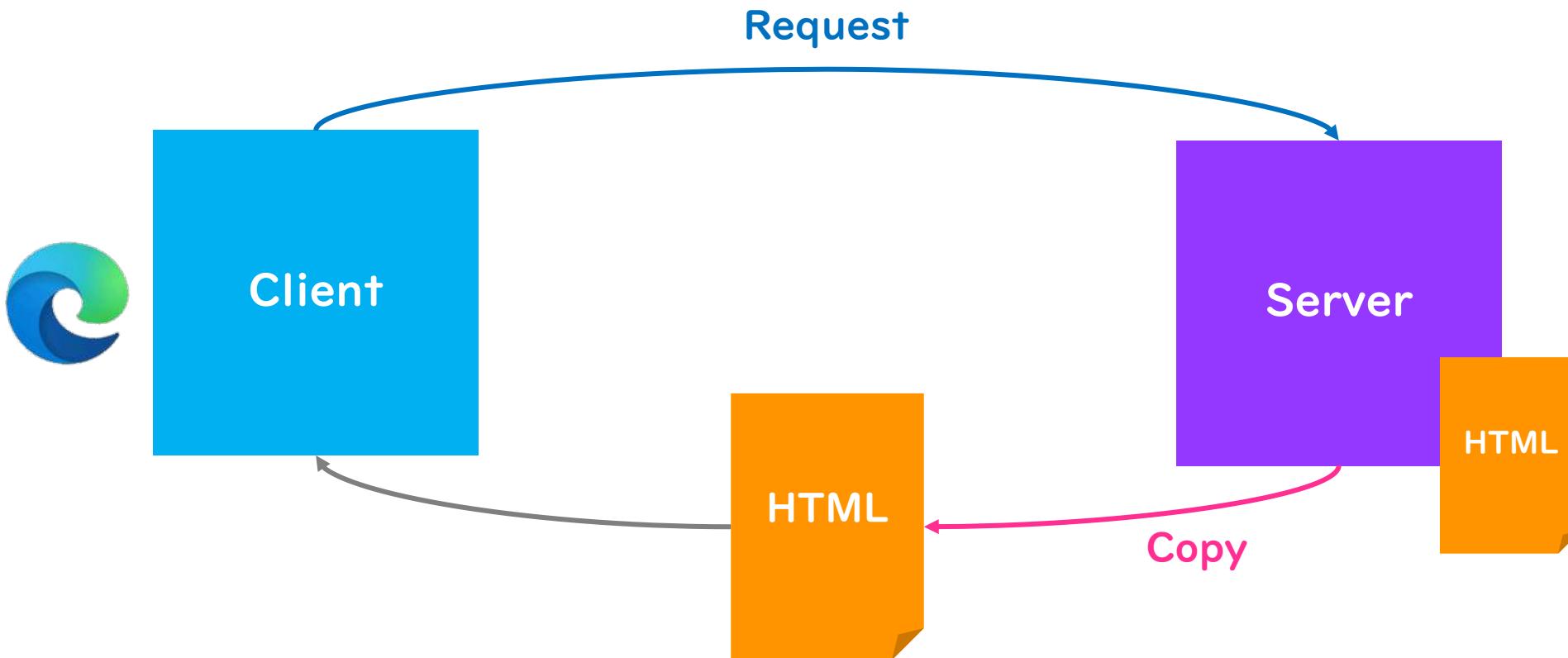
- Certificate - valid and trusted  
The connection to this site is using a valid, trusted server certificate issued by GTS CA 1O1.  
View certificate
- Connection - secure connection settings  
The connection to this site is encrypted and authenticated using QUIC, X25519, and AES\_128\_GCM.
- Resources - all served securely  
All resources on this page are served securely.

# Introduction to Web Documents

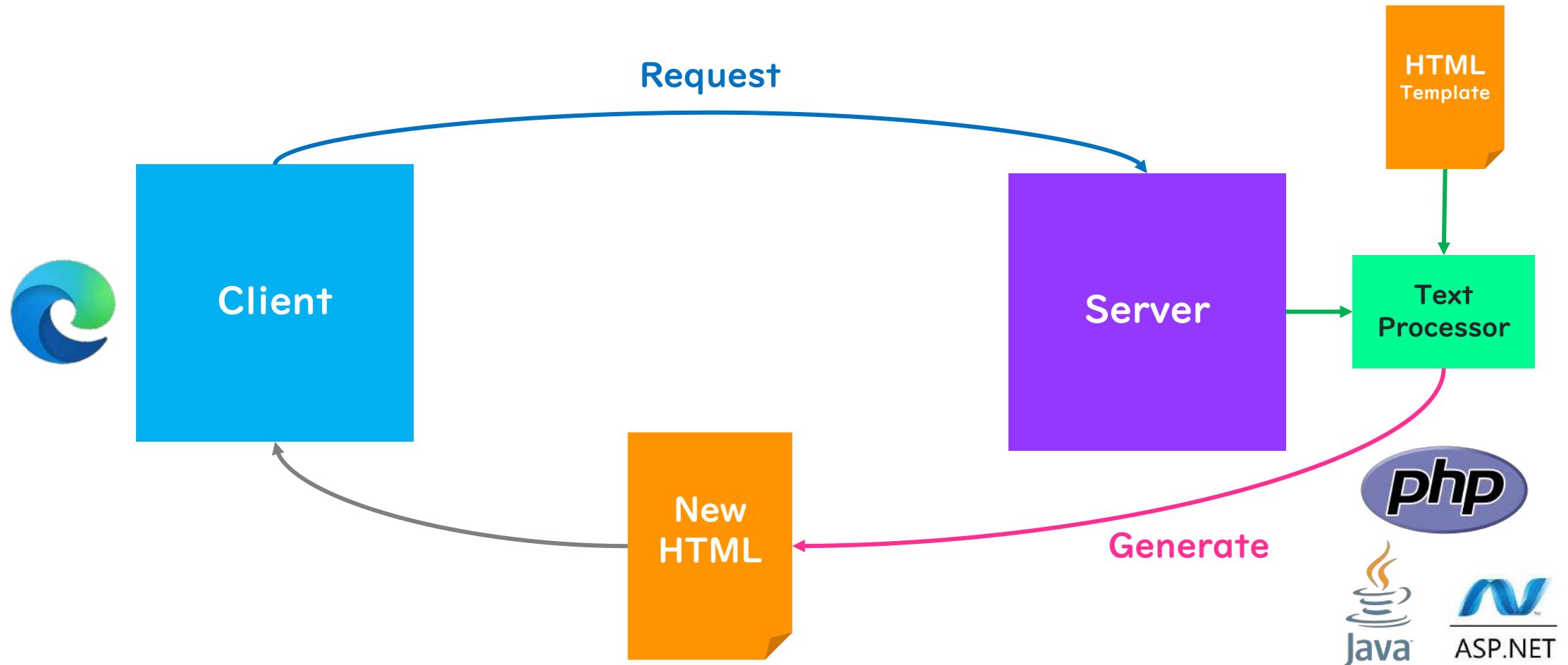
# Web Documents

- A document is the name of a message sent over HTTP. It's the data unit for the web. A document moves on the internet and is understood by the web client (browser mainly).
- There are 3 main types of web document in the meantime.

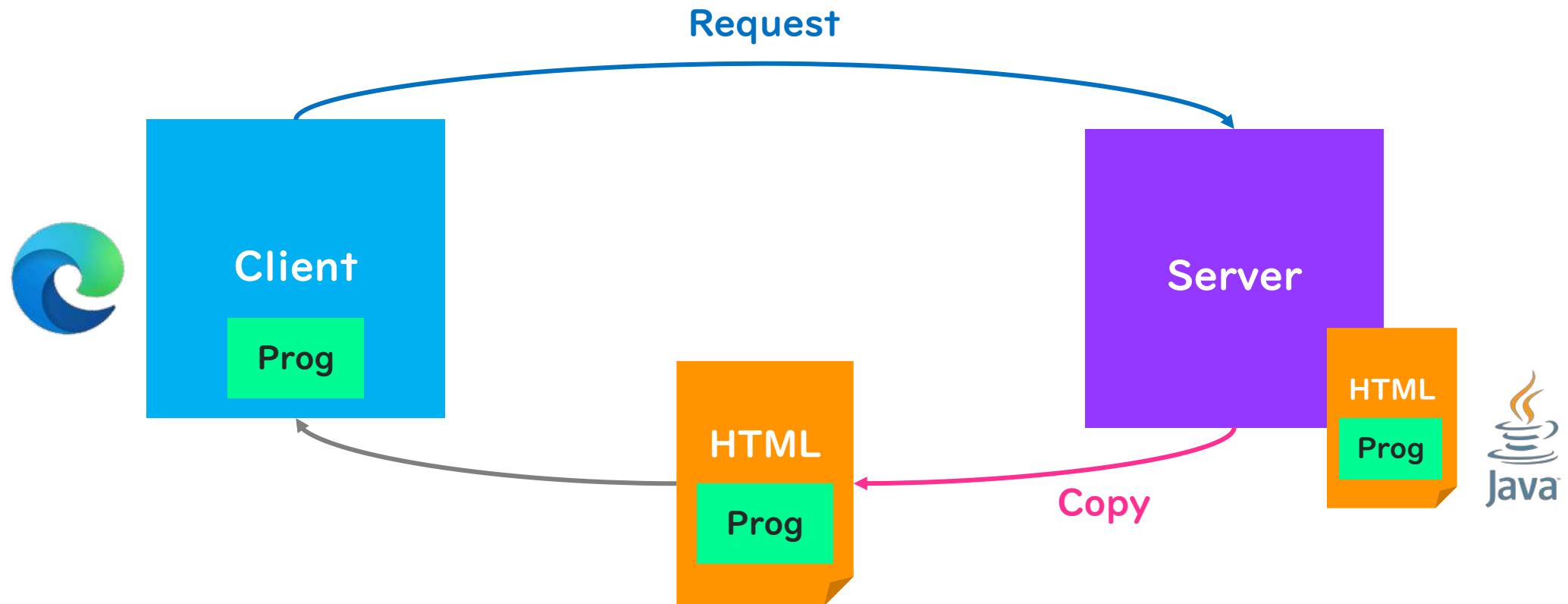
# Static Document



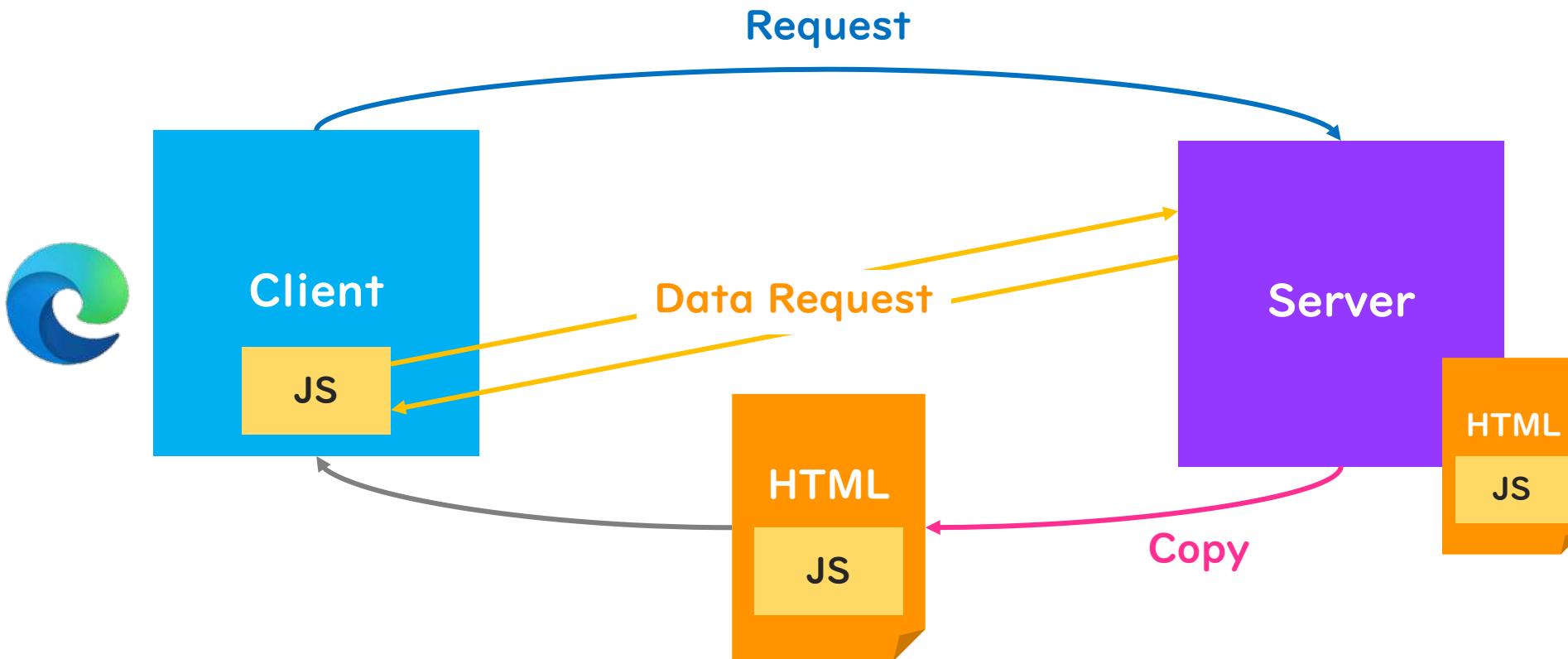
# Dynamic Document (Server-side)



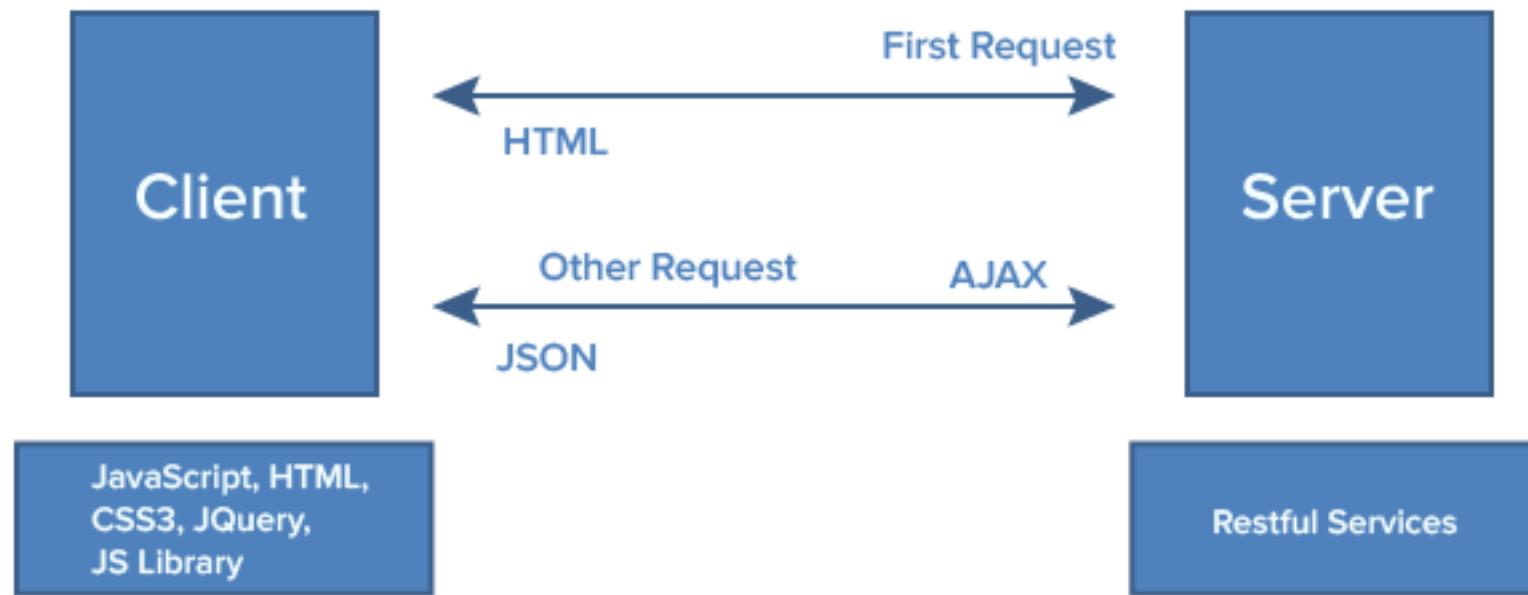
# Active Document



# Client-Side Document

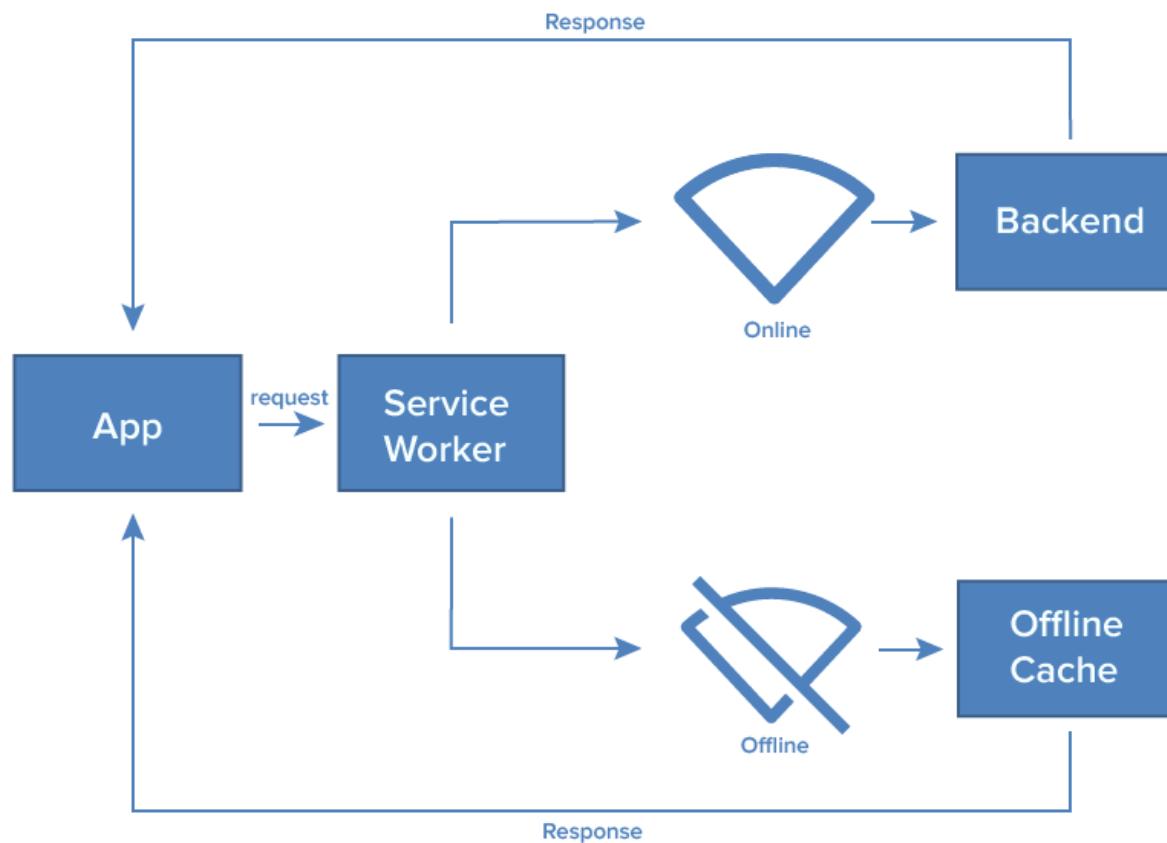


# Single Page Application (SPA)

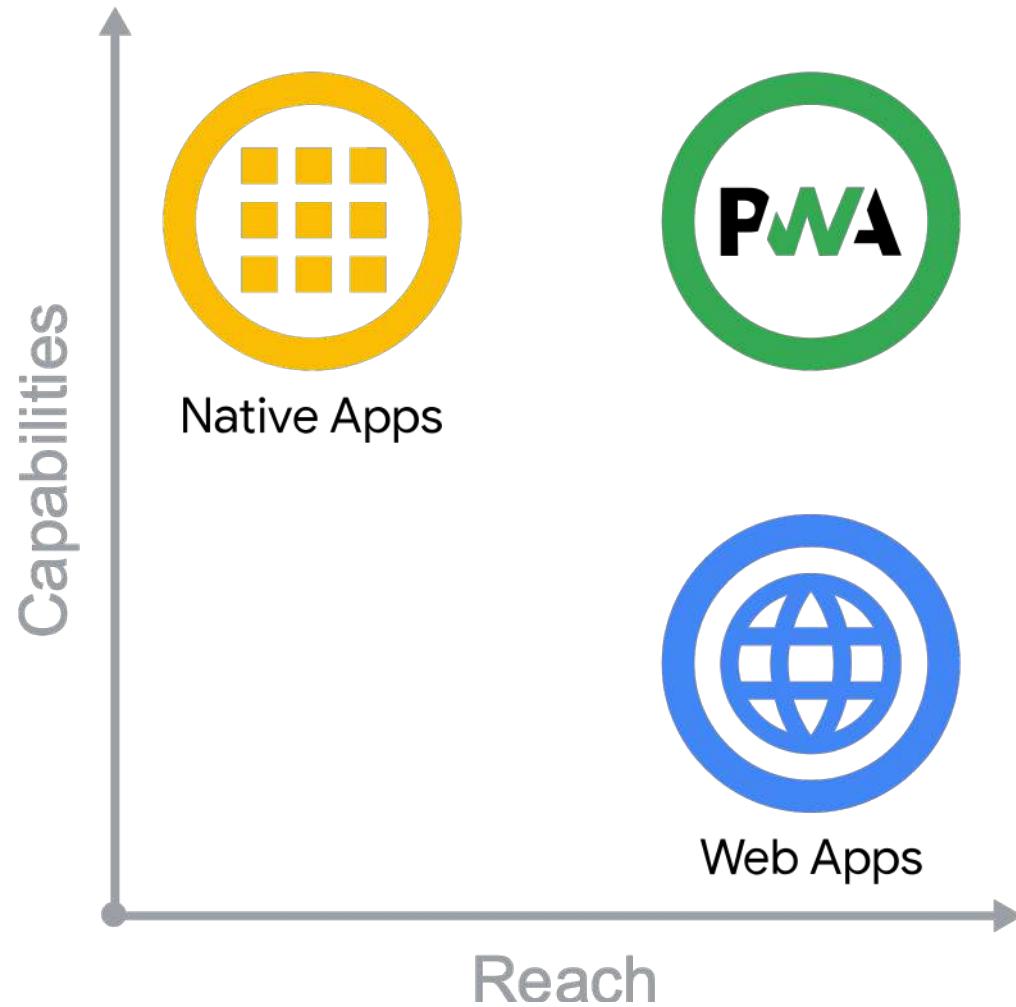


<https://www.simicart.com/blog/pwa-vs-spa>

# Progressive Web Application (PWA)



<https://www.simicart.com/blog/pwa-vs-spa>



<https://webdev.imgix.net/what-are-pwas>

# Go Programming Basics



# Why Go?

Black Lives Matter. [Support the Equal Justice Initiative.](#)

 Documents Packages The Project Help Blog Play Search 

Go is an open source programming language that makes it easy to build **simple, reliable, and efficient** software.



 [Download Go](#)

Binary distributions available for Linux, macOS, Windows, and more.

**Try Go** [Open in Playground ↗](#)

```
// You can edit this code!
// Click here and start typing.
package main

import "fmt"

func main() {
    fmt.Println("Hello, 世界")
}
```

Hello, World!   

<https://golang.org/>

## Results

Updated on 2020-11-05

<https://github.com/the-benchmarkator/web-frameworks>

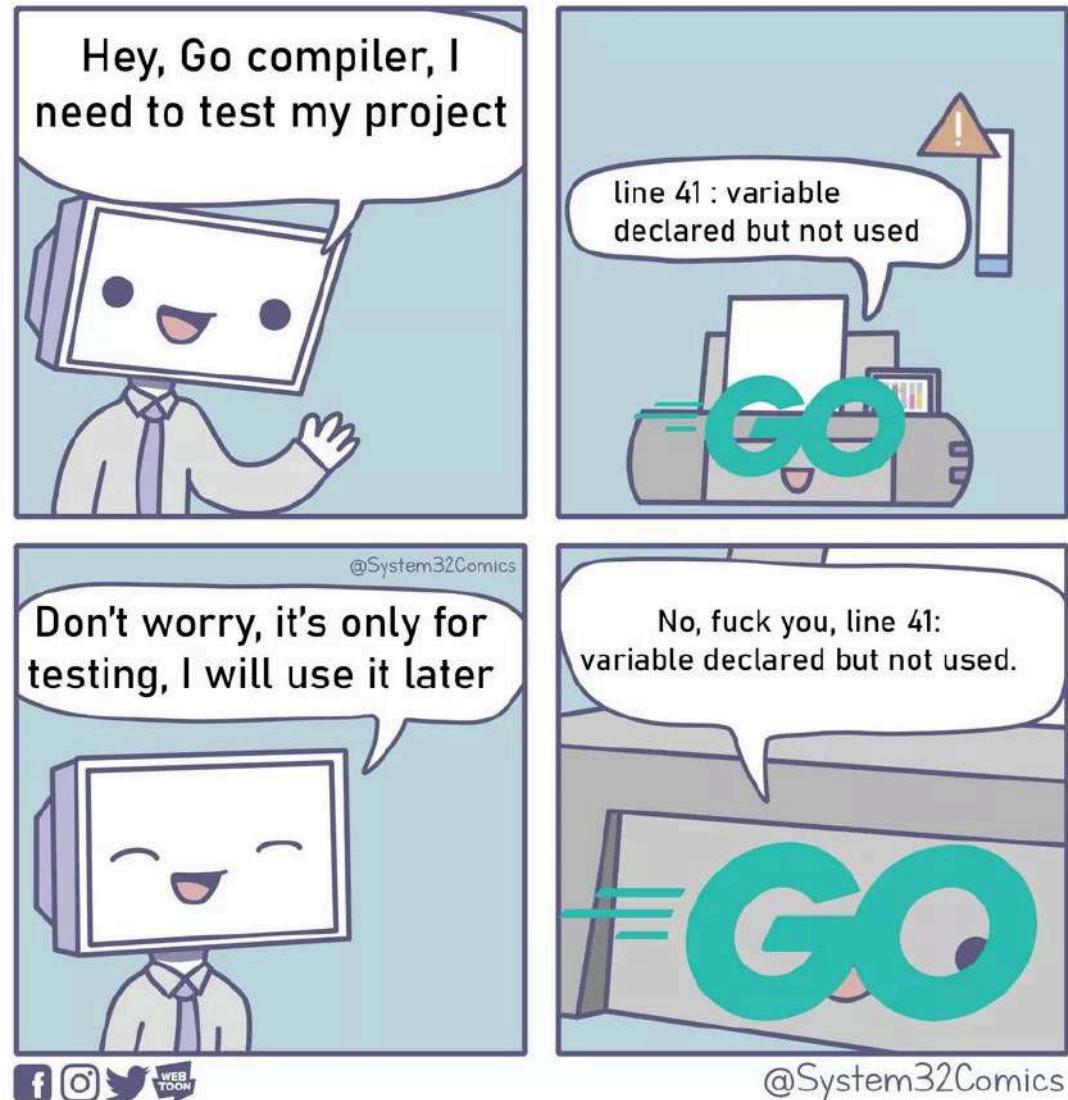
Benchmarking with wrk

- Threads : 8
- Timeout : 8
- Duration : 15s (seconds)

Sorted by max req/s on concurrency

	Language	
1	go (1.15)	fiber (2.0)
2	go (1.15)	fasthttp (1.16)
3	php (7.4)	mark (1.1)
4	go (1.15)	gearbox (1.1)
5	go (1.15)	gorouter-fasthttp (4.4)
6	go (1.15)	router (1.3)
7	nim (1.2)	httpbeast (0.2)
8	go (1.15)	atreugo (0.4)
9	java (11)	rapidoid (0.4)
10	nim (1.2)	whip (0.2)
11	java (11)	jooby (2.8)
12	javascript (12.18)	sifrr (0.0)
13	java (11)	light-4j (2.0)
14	nim (1.2)	jester (0.4)
15	kotlin (1.4)	kooby (2.8)

	Language	Framework	Speed (64)	Speed (256)	Speed (512)
1	go (1.15)	fiber (2.0)	169 728	181 862	183 418
2	go (1.15)	fasthttp (1.16)	169 038	183 813	184 384
3	php (7.4)	mark (1.1)	168 006	172 705	172 622
4	go (1.15)	gearbox (1.1)	167 604	177 253	177 806
5	go (1.15)	gorouter-fasthttp (4.4)	167 276	181 708	179 071
6	go (1.15)	router (1.3)	166 242	180 922	179 163
7	nim (1.2)	httpbeast (0.2)	166 147	199 087	201 467
8	javascript (12.18)	sifrr (0.0)	154 386	194 178	195 460
9	java (11)	light-4j (2.0)	153 823	193 652	198 295
10	nim (1.2)	jester (0.4)	150 971	178 514	181 225
11	kotlin (1.4)	kooby (2.8)	146 865	186 795	189 715



<https://www.jesuisundev.com/en/understand-go-in-5-minutes/>

## Hello, 世界

Welcome to a tour of the [Go programming language](#).

The tour is divided into a list of modules that you can access by clicking on [A Tour of Go](#) on the top left of the page.

You can also view the table of contents at any time by clicking on the [menu](#) on the top right of the page.

Throughout the tour you will find a series of slides and exercises for you to complete.

You can navigate through them using

"previous" or [PageUp](#) to go to the previous page,

"next" or [PageDown](#) to go to the next page.

The tour is interactive. Click the [Run](#) button now (or press Shift + Enter) to compile and run the program on a remote server. The result is displayed below the code.

These example programs demonstrate different aspects of Go. The programs in the tour are meant to be starting points for your own experimentation.

Edit the program and run it again.

When you click on [Format](#) (shortcut: Ctrl + Enter), the text in the editor is formatted using the [gofmt](#) tool. You can switch syntax highlighting on and off by clicking on the [syntax](#) button.

When you're ready to move on, click the [right arrow](#) below or type the [PageDown](#) key.

< 1/5 >

```
hello.go
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello, 世界")
7 }
```

Reset Format Run



# Go's 'Hello World'

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

# Init Function

```
func init() {  
    fmt.Println("Init! 1")  
}  
  
func init() {  
    fmt.Println("Init! 2")  
}
```

- Init function runs before the main function.
- One program may declare more than one **init** functions.
  - Run sequentially

# Primitive Data Type

```
bool
string
int int8 int16 int32 int64
uint uint8 uint16 uint32 uint64 uintptr
byte // alias for uint8
rune // alias for int32
      // represents a Unicode code point
float32 float64
complex64 complex128
```

<https://tour.golang.org/basics/11>

# Slice

- An array has a fixed size. A slice, on the other hand, is a dynamically-sized, flexible view into the elements of an array.
- In practice, slices are much more common than arrays.

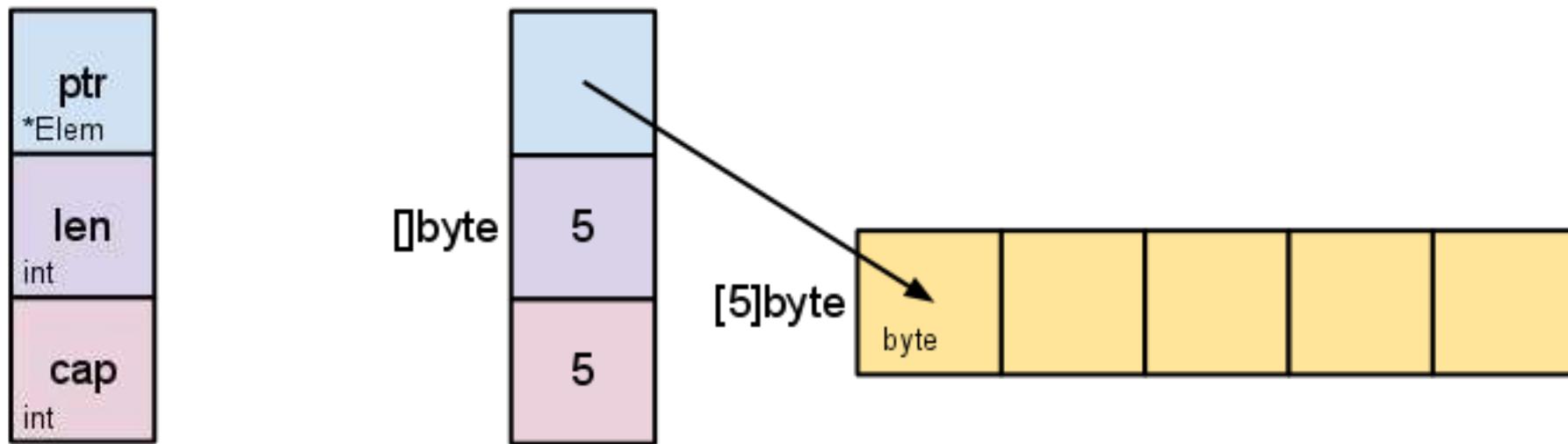
# Slice

```
a := make([]int, 5) // len(a)=5
b := make([]int, 0, 5) // len(b)=0, cap(b)=5

b = b[:cap(b)] // len(b)=5, cap(b)=5
b = b[1:] // len(b)=4, cap(b)=4

q := []int{2, 3, 5, 7, 11, 13} //slice literal
fmt.Println(q)
```

# Slice



# Slice

- <https://blog.golang.org/slices-intro>

# Abstract Data Type

```
type Rect struct {
    width, height float64
}

func (r Rect) Area() float64 {
    return r.width * r.height
}

func (r Rect) Perim() float64 {
    return 2 * r.width + 2 * r.height
}
```

Rect

width  
height

Area()  
Perim()

# Interface

```
type Shape interface {
    Area() float64
    Perim() float64
}

func measure(s Shape) {
    fmt.Println(s.Area(), s.Perim())
}
```

# **However, there is no OOP in Go!**

**(Please be appreciative that there are also no OOP problems in Go!)**

# Scope

```
v := "outer"
fmt.Println(v)
{
    v := "inner"
    fmt.Println(v)
    {
        fmt.Println(v)
    }
}
{
    fmt.Println(v)
}
fmt.Println(v)
```

## Block Scope!

(Also static)

```
outer
inner
inner
outer
outer
```

<https://medium.com/golangspec/scopes-in-go-a6042bb4298c>

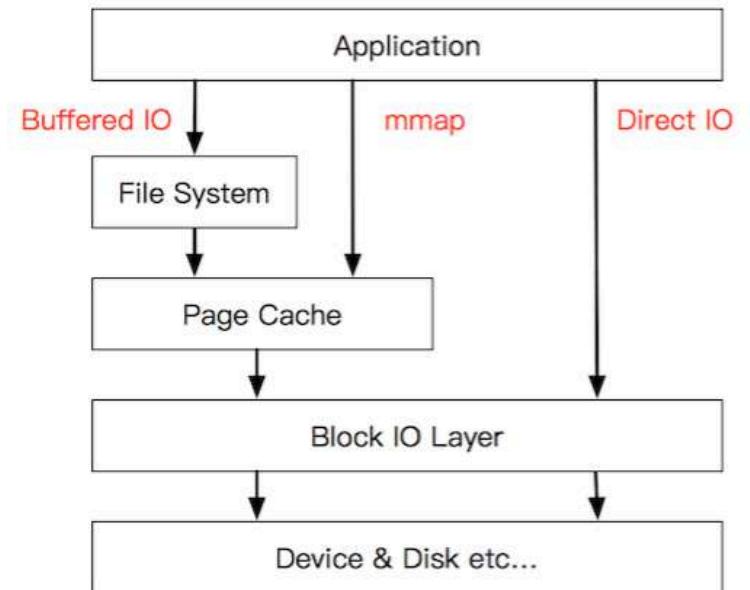
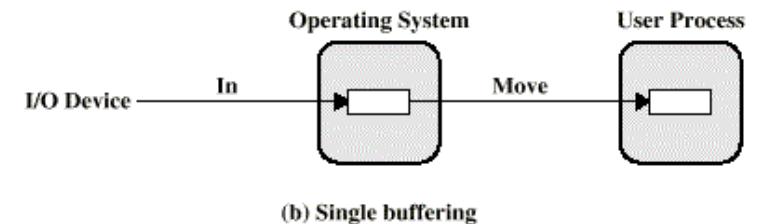
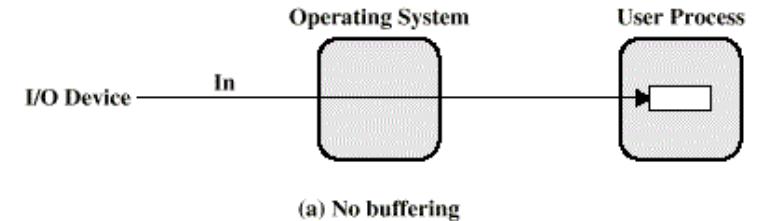
# Stdin Input

```
reader := bufio.NewReader(os.Stdin)
myStr, _ := reader.ReadString('\n')
```

## Buffered I/O

```
var myInt int
fmt.Scanf("%d\n", &myInt)
```

## Unbuffered I/O (Blocking)



# Selective Statement

```
if myStr == "Hi!" {  
    fmt.Println("Hello!")  
} else {  
    fmt.Println("Bye!")  
}
```

- Go's switch does not fall into other cases as its default behavior (like other languages).
  - Use `fallthrough`

```
switch myStr {  
case "A":  
    fmt.Println("AA")  
case "B":  
    fmt.Println("BB")  
    fallthrough  
case "C":  
    fmt.Println("CC")  
}
```

# Iterative Statement

```
for k < 5 {  
    k--  
    fmt.Println(k)  
}
```

For as a while loop

```
for j := 0; j < myInt; j++ {  
    fmt.Print("*")  
}
```

Typical for loop

# Function

```
func addShape(a Shape, b Shape) (float64, string) {  
    return (a.Area() + b.Area()), "ok"  
}
```

## Multiple Returns

# Function Type

```
type subtract func(a int, b int) int

...
var b subtract = func(a int, b int) int {
    return a - b
}

fmt.Println(b(4, 2))
```

# First-Class Function

Anonymous  
Function

Higher-Order  
Function

Closure

# Anonymous Function

- Function without a name (anonymous)

```
a := func() {
    fmt.Println("Anonymous Function")
}

a()

func(str string) {
    fmt.Println("prog > ", str)
}("something")
```

# Higher-Order Function

- Function as a parameter

```
func myHigherFunc(someFunc func()) {  
    someFunc()  
    someFunc()  
}  
  
func myFunc() {  
    fmt.Println("Hello!")  
}  
  
myHigherFunc(myFunc)
```

# Closure

```
func factorial() func(n uint) uint {  
    var y uint = 1  
    return func(n uint) uint {  
        if n > 1 {  
            y = n * y  
        } else {  
            return 1  
        }  
        return y  
    }  
}
```

I've used closures to do things like:

255

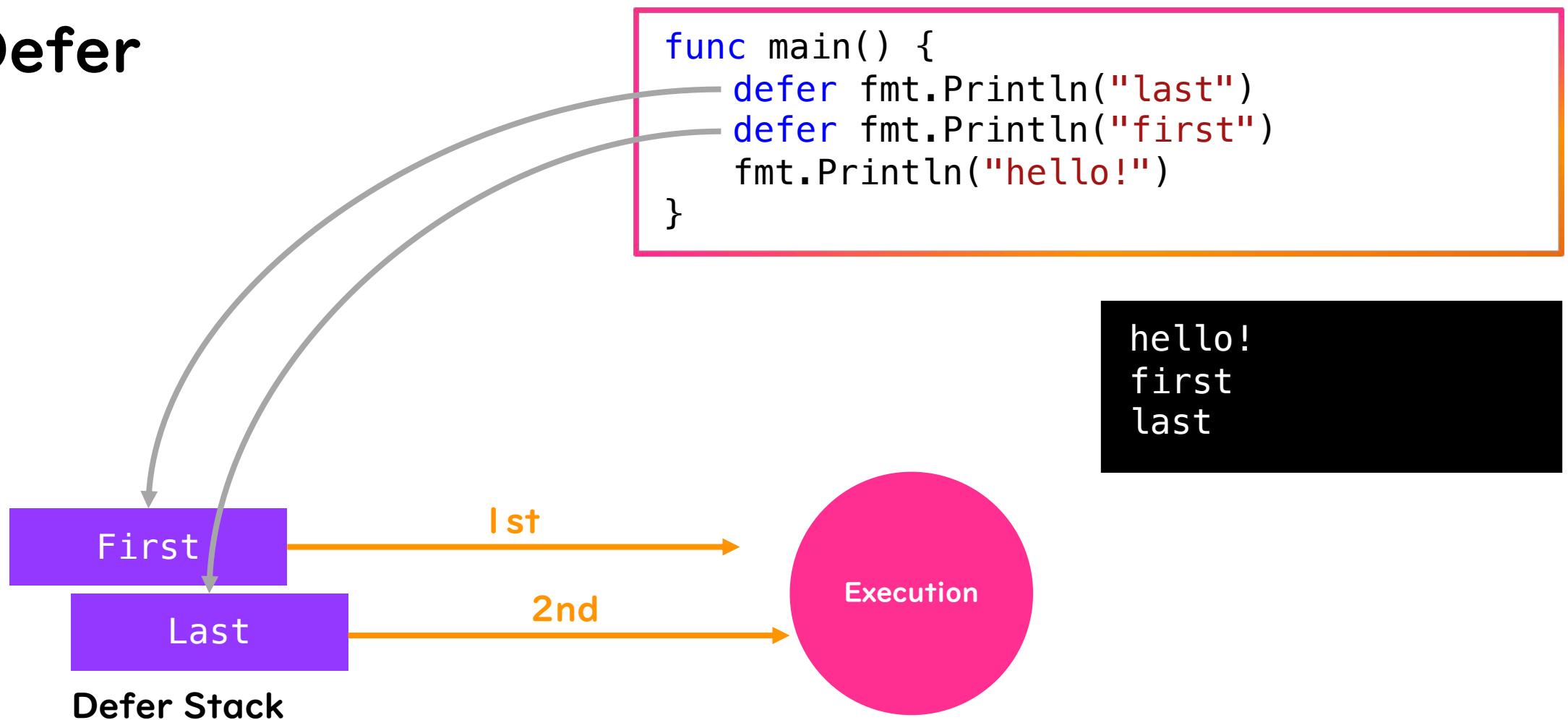
```
a = (function () {  
    var privatefunction = function () {  
        alert('hello');  
    }  
  
    return {  
        publicfunction : function () {  
            privatefunction();  
        }  
    }  
})();
```



As you can see there, `a` is now an object, with a method `publicfunction()` (`a.publicfunction()`) which calls `privatefunction`, which only exists inside the closure. You can **NOT** call `privatefunction` directly (i.e. `a.privatefunction()`), just `publicfunction()`.

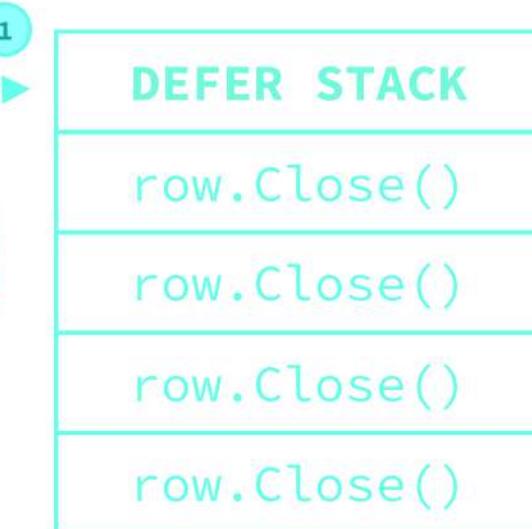
Its a minimal example but maybe you can see uses to it? We used this to enforce public/private methods.

# Defer



# Defer

```
func() {  
    for {  
        row, err := db.Query("SELECT ...")  
        if err != nil {  
            ..  
        }  
        defer row.Close()  
        ..  
    }  
    // deferred funcs run here  
}
```



2

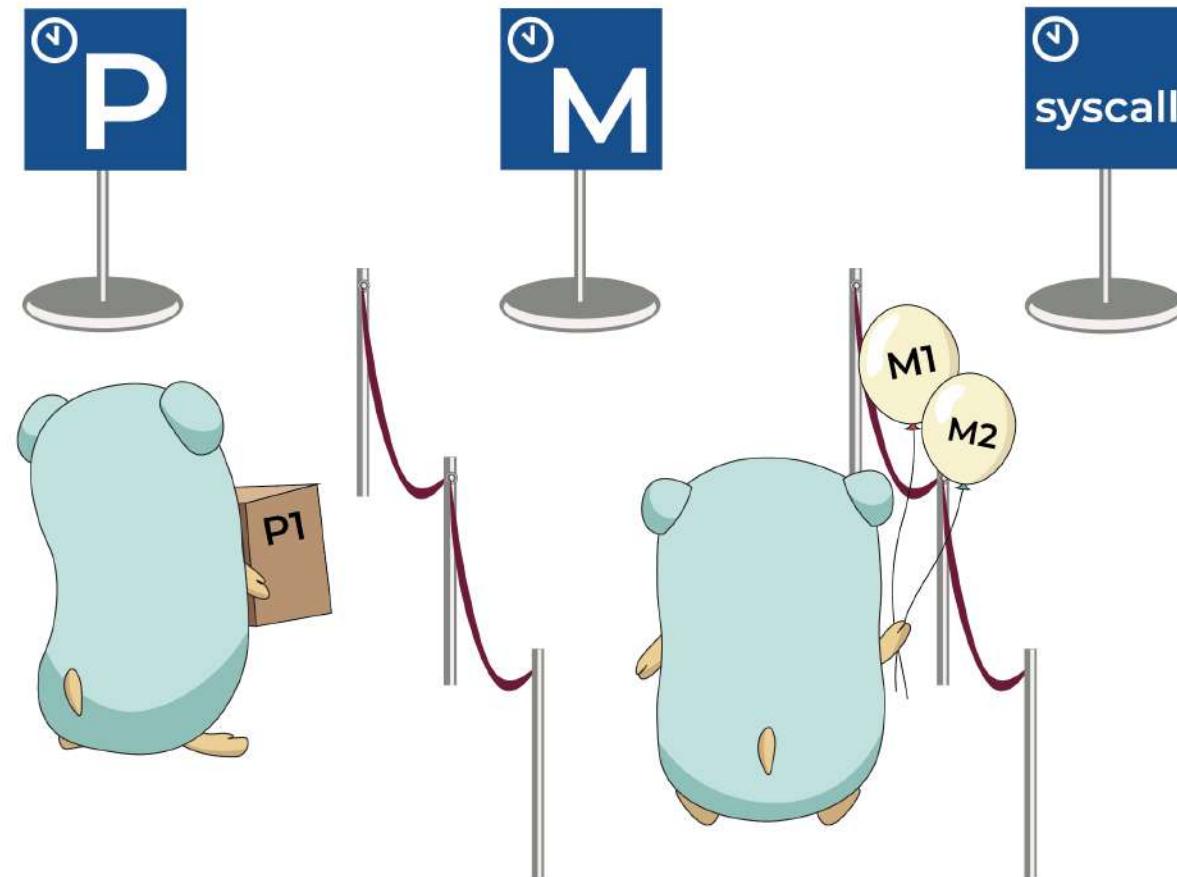
<https://blog.learnprogramming.com/gotchas-of-defer-in-go-1-8d070894cb01?gi=7213d37bd899>

# Panic and Recover

```
defer func {
    errMsg := recover()
    if errMsg == "INTEGER_OVERFLOW" {
        log.Println("Panic handled!")
    }
}()

if myInt > 100 {
    panic("INTEGER_OVERFLOW")
}
```

# Goroutine

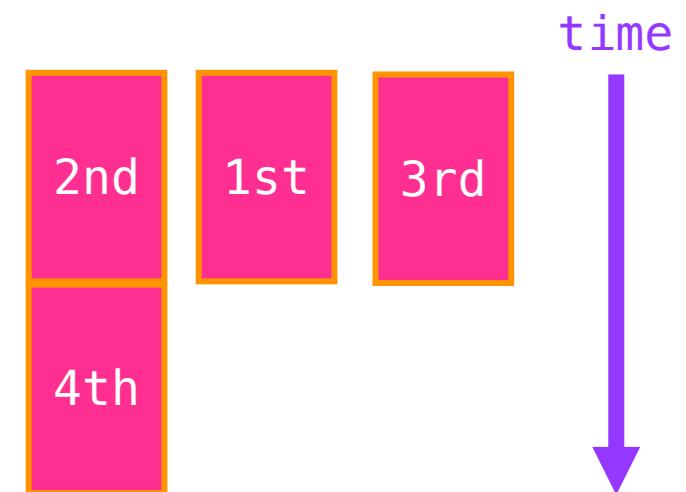


# Goroutine

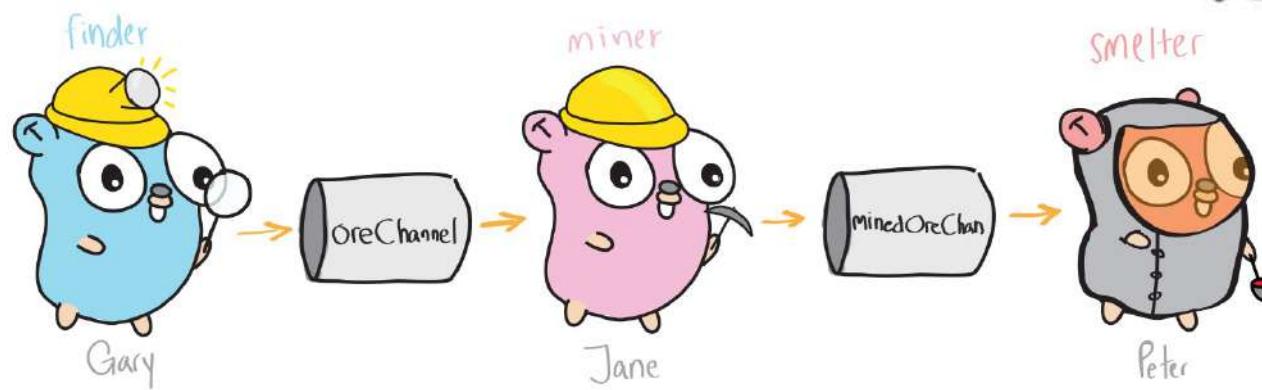
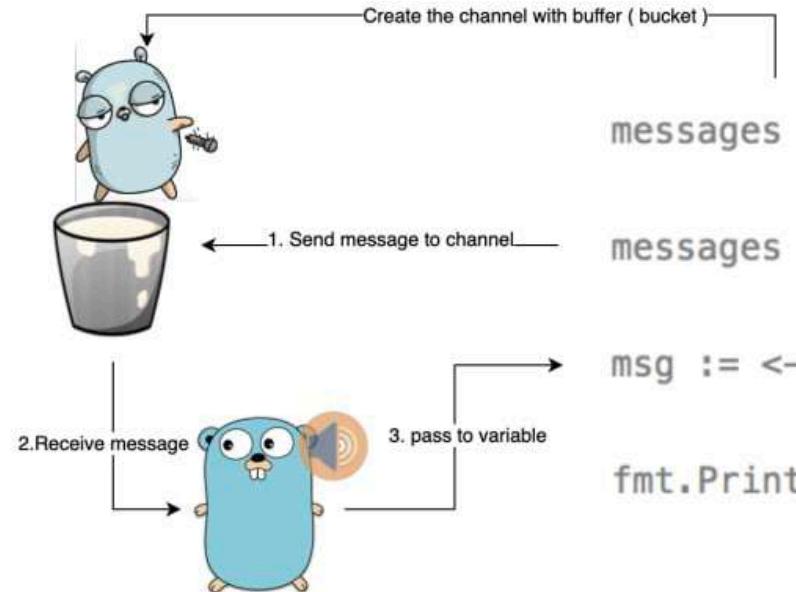
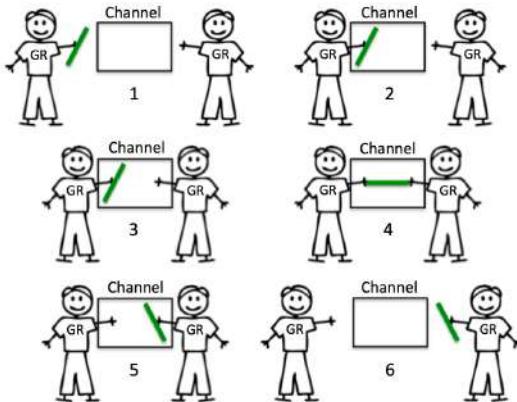
```
go myFunc()  
myFunc()  
go myFunc()  
myFunc()  
  
time.Sleep(time.Second)  
  
func myFunc() {  
    a += 1  
    fmt.Println(a)  
}
```

One possible output

```
1  
3  
2  
4
```



# Channel

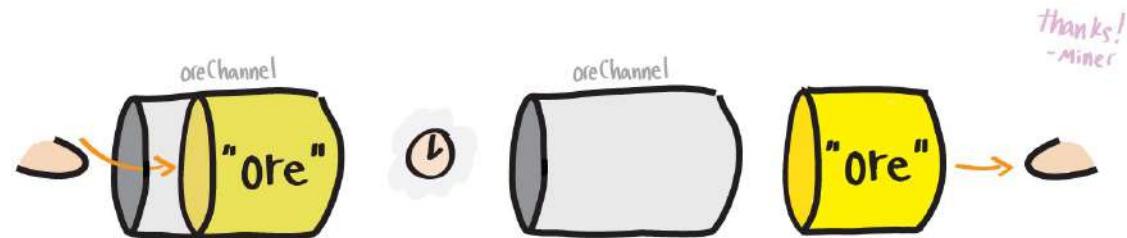


<https://dev.to/ahmedash95/understand-golang-channels-and-how-to-monitor-with-grafana-154>

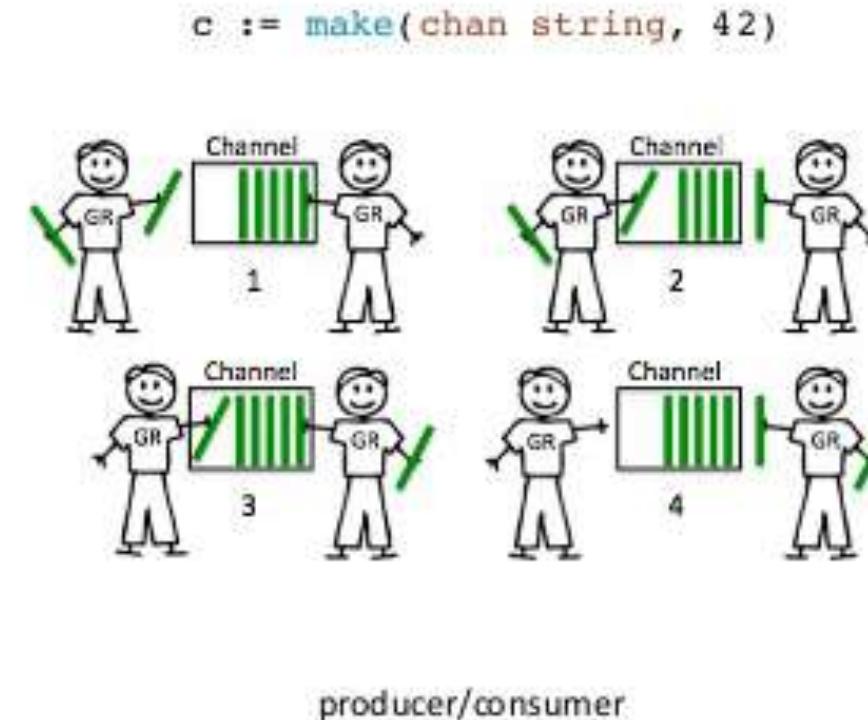
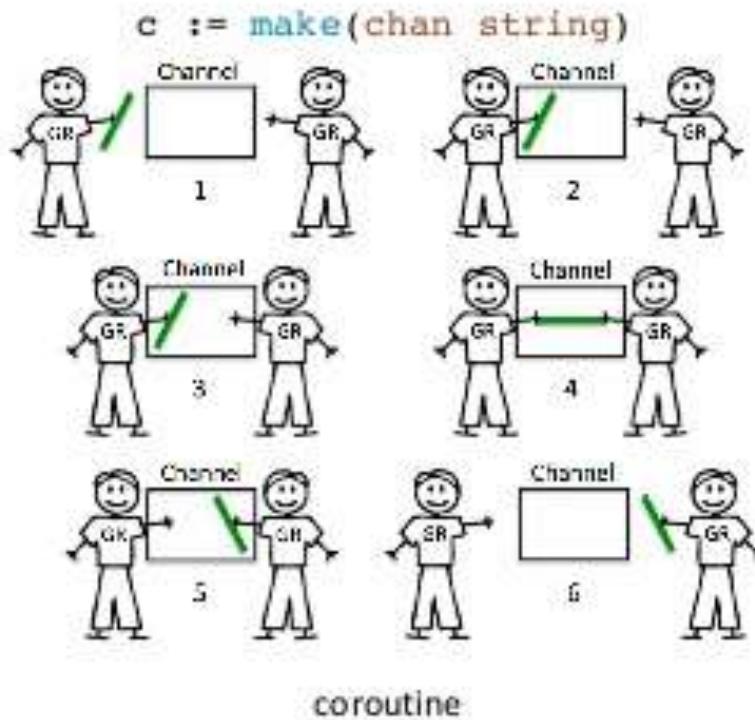
# Channel

```
channel := make(chan uint)
go func() {
    time.Sleep(time.Second)
    channel <- 5
}()

// wait until there is a value in the channel
fmt.Println(<-channel)
```



# Channel



# Package

```
package main

import (
    "fmt"
    "./utils"
)

func main() {
    a, b := 10, 2
    fmt.Println(utils.Add(a, b))
    fmt.Println(utils.Subtract(a, b))
    fmt.Println(utils.Multiply(a, b))
    fmt.Println(utils.Divide(a, b))
}
```

⌄ utils

-[calculator.go](#)

-[main.go](#)

# Pointer

```
var i uint32 = 5  
  
// * = referencing, & = address  
poi := &i  
fmt.Println(i, poi, *poi)  
  
*poi = 10  
fmt.Println(i, poi, *poi)  
  
var j uint32 = 20  
poi = &j  
fmt.Println(i, poi, *poi)
```

# Pointer

- No pointer arithmetic in Go

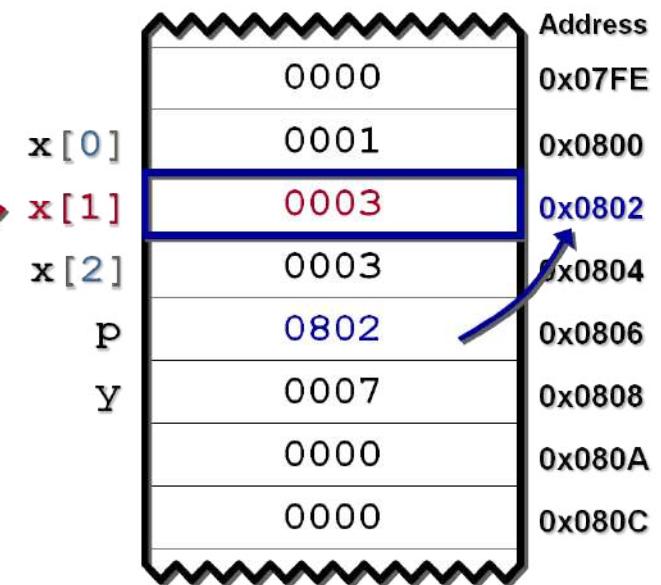
```
p := Point{10, 20}
      10 | 20  Point
```

```
pp := &Point{10, 20}
      *Point
      10 | 20  Point
```

```
{
    int x[3] = {1,2,3};
    int y;
    int *p = &x;

    y = 5 + * (p++);
    y = 5 + (*p)++;

}
```



# The Big Pictures of Databases

# Why do we need databases?

- Can't we use a typical file system to store data?

# Why do we need databases?

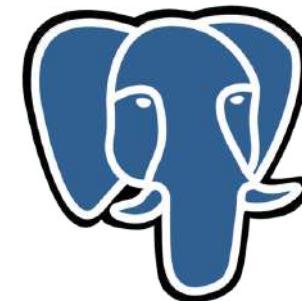
- Database consists of many data structures and algorithms which are utilized for storing data efficiently.
  - It is very quick and easy to find information.
  - It is easy to add new data and to edit or delete old data.
  - And many more...

# What is database?

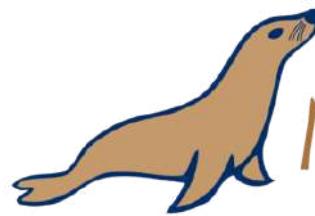
- A database is an organized collection of data, generally stored and accessed electronically from a computer system.
- Where databases are more complex, they are often developed using formal design and modeling techniques.

# Database Management System (DBMS)

- A database management system (DBMS) is a software package designed to define, manipulate, retrieve and manage data in a database.



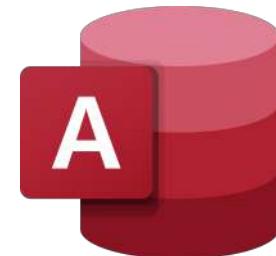
PostgreSQL



MariaDB



Microsoft®  
SQL Server®



# DBMS Feature

- Codd proposed the following functions and services a fully-fledged general-purpose DBMS should provide:

Data Storage

Transaction Control

Concurrent Control

Recovery Control

Data Integrity and  
Constraint

Security

# Types of DBMS

- There are many types of databases today; there are about 2 – 4 big types of databases, which are:
  - **Relational DBMS**
  - Object-Oriented DBMS
  - **Non-Relational DBMS (NoSQL)**
  - NewSQL

# Relational DBMS (RDBMS)

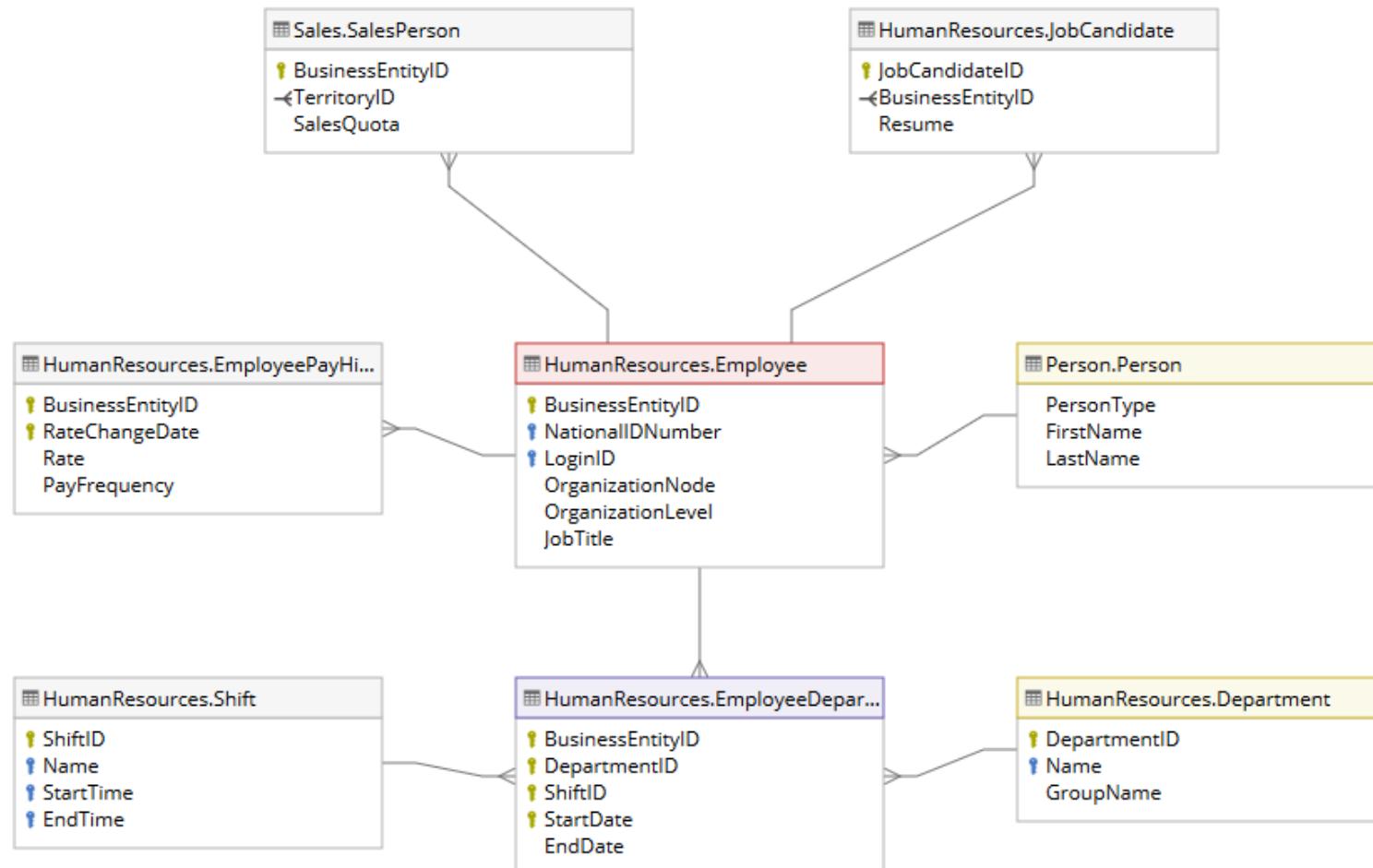
- A relational database is a digital database based on the **relational model of data**, as proposed by E. F. Codd in 1970.

SQL term	Relational database term	Description
<i>Row</i>	<i>Tuple</i> or <i>record</i>	A data set representing a single item
<i>Column</i>	<i>Attribute</i> or <i>field</i>	A labeled element of a tuple, e.g. "Address" or "Date of birth"
<i>Table</i>	<i>Relation</i> or <i>Base relvar</i>	A set of tuples sharing the same attributes; a set of columns and rows
<i>View</i> or <i>result set</i>	<i>Derived relvar</i>	Any set of tuples; a data report from the RDBMS in response to a <i>query</i>

# Schema

- The database schema of a database is its structure described in a formal language supported by the database management system (DBMS).
- The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases).

# Schema



Generated with [Dataedo](#)

# Entity-Relationship Diagram

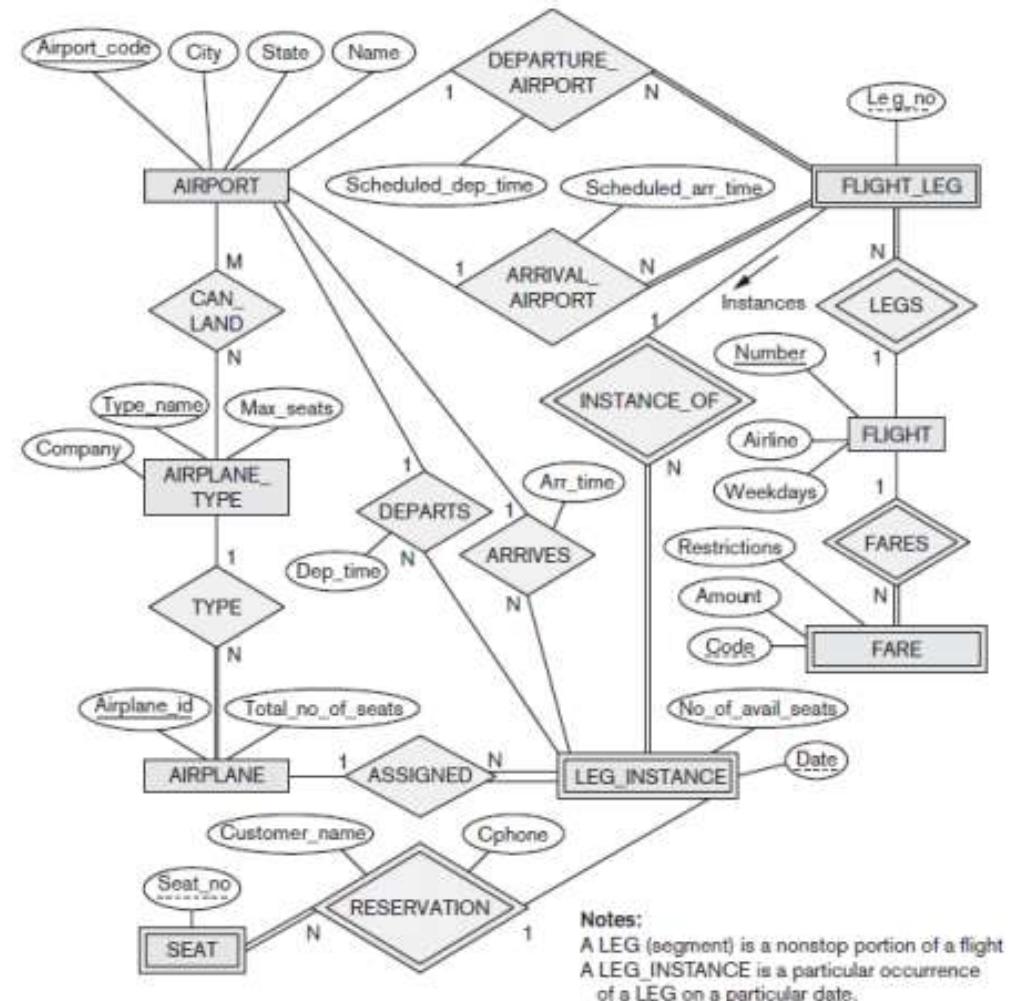
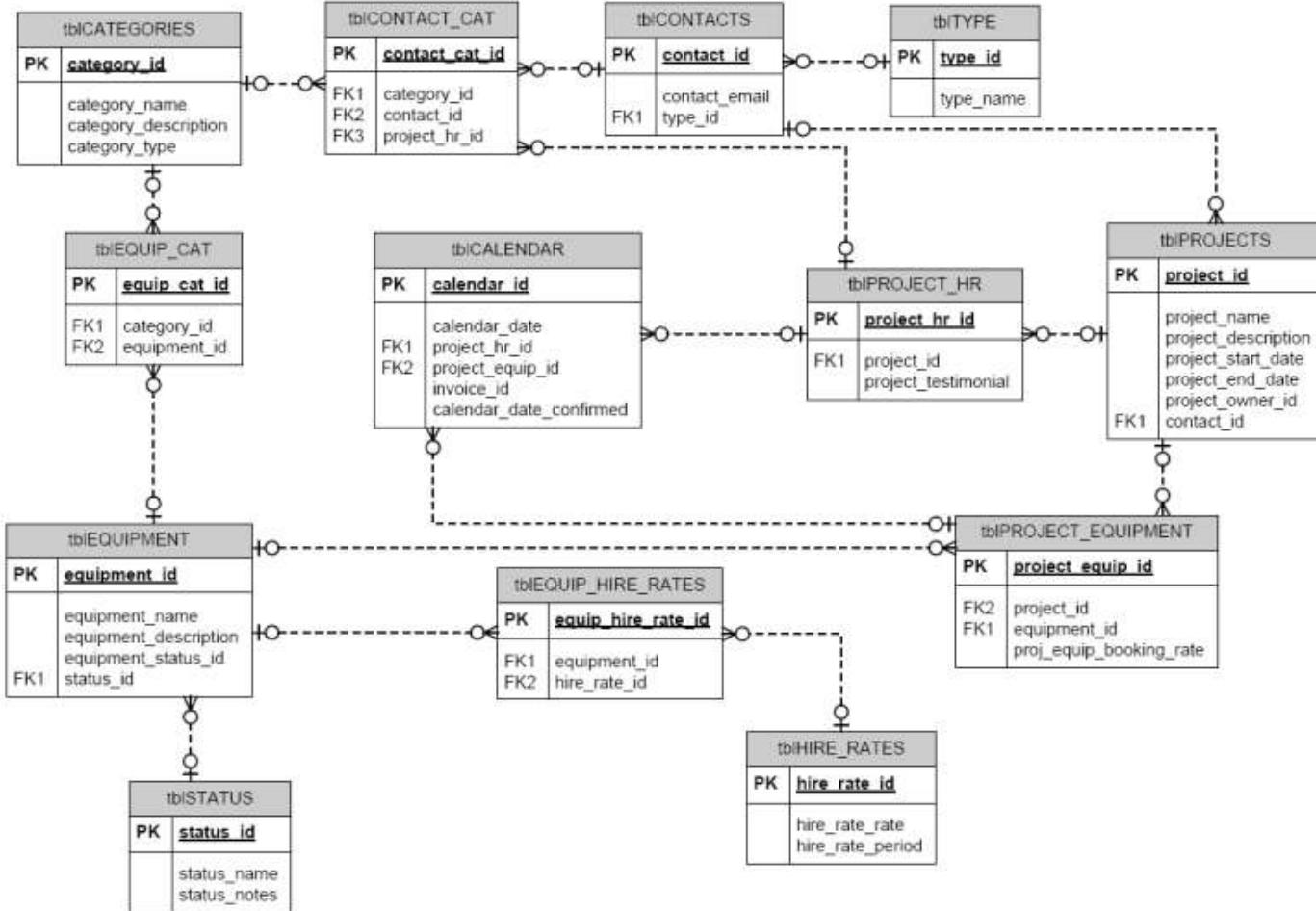


Figure 1: E-R Diagram for an Airline Booking Application

# Primary Key

- A primary key is "which attributes identify a record", and in simple cases are simply a single attribute: a unique id.
  - Citizen ID
  - Student ID
  - Employee ID

# Foreign Key

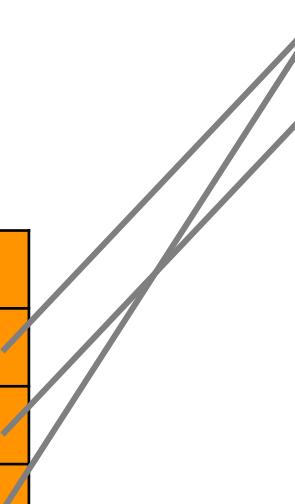
- A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables.

# How the keys work?

Student ID	Name	Class Name	Class Teacher
1	Alice	Rainbow	Kuro
2	Bob	Gradient	Kami
3	Cassandra	Rainbow	Kuro

# How the keys work?

Student ID	Name	Class ID
1	Alice	1
2	Bob	2
3	Cassandra	1



Class ID	Name	Teacher
1	Rainbow	Kuro
2	Gradient	Kami

- The keys make our data redundant.
  - However, interpretation of the data is harder.

# Data Integrity

- RDBMSs also give us a help for checking the data stored using user-defined constraints.
  - Entity integrity (for Primary Key)
  - Referential integrity (for Foreign Key)
  - Domain integrity
  - User-defined integrity

# Database Join

- Natural join ( $\bowtie$ ) is a binary operator that is written as  $(R \bowtie S)$  where R and S are relations.
  - The result of the natural join is the set of all combinations of tuples in R and S that are equal on their common attribute names.

Employee		
Name	Empld	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Mary	1257	Human Resources

Dept	
DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Employee $\bowtie$ Dept			
Name	Empld	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

# Structured Query Language (SQL)

- SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS).

```
SELECT *
FROM some_table
WHERE x > y;
```

# Data Definition Language (DDL)

- Data definition or data description language (DDL) is a syntax for creating and modifying database objects such as tables, indexes, and users.
- Common examples of DDL statements include CREATE, ALTER, and DROP.

# Data Manipulation Language (DML)

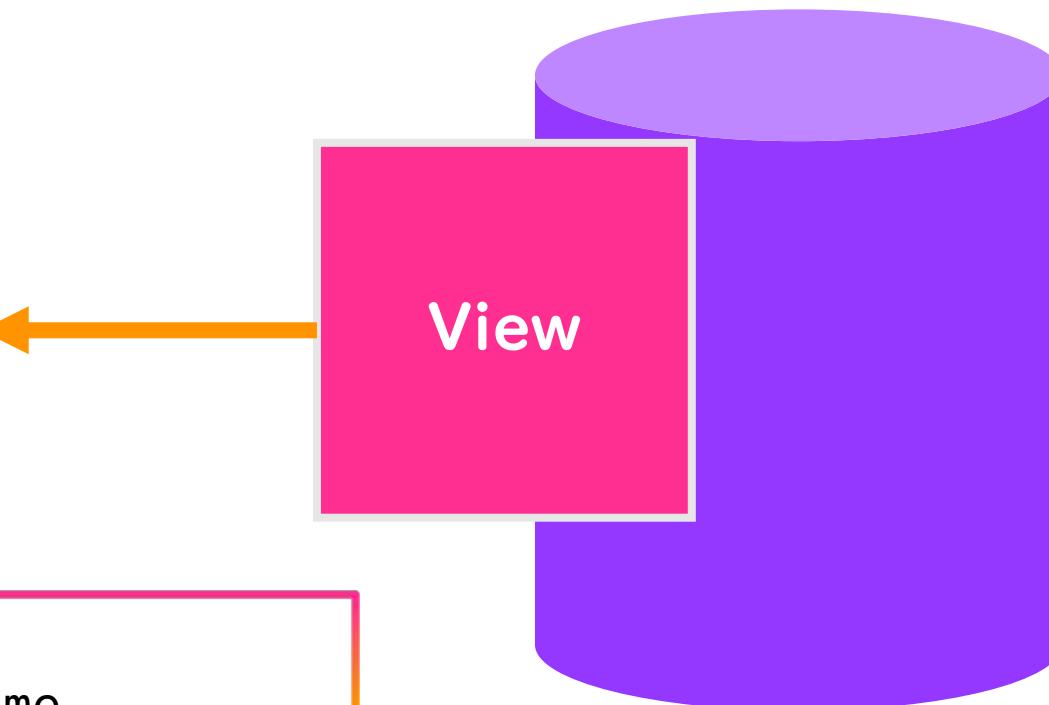
- A data manipulation language (DML) is a computer programming language used for adding (inserting), deleting, and modifying (updating) data in a database.

# SELECT statement

```
SELECT [column_name]*
FROM [table_name]*
WHERE [condition]
GROUP BY [column_name]*
HAVING [aggregated_condition]
ORDER BY ([column_name] [option])*;
```

# View

Student ID	Name
1	Alice
2	Bob
3	Cassandra



```
CREATE VIEW some_view
    SELECT student_id, name
    FROM students;
```

# Practice

- Let's get to know more about SQL commands
  - CREATE TABLE
    - PRIMARY KEY
    - FOREIGN KEY
  - INSERT
  - SELECT
    - JOIN
  - UPDATE
  - DELETE

# ACID

- RDBMSs provide the properties called ACID.
  - Atomicity
    - Transactions cannot be irreducible.
    - All steps either need to be **all done or nothing happens.**
  - Consistency
    - Any data written to the database must be valid according to all defined rules.
  - Isolation (Database locking)
  - Durability (Recovery control)

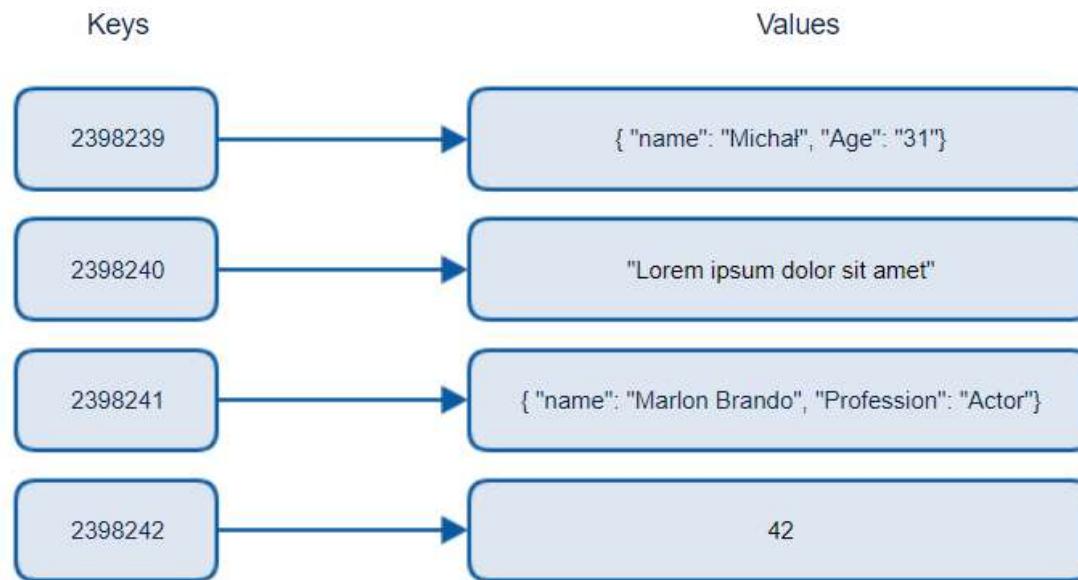
# Non-Relational DBMS (NoSQL)

- Wide column: Accumulo, Cassandra, Scylla, HBase.
- Document: Apache CouchDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, eXist-db, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB
- Key-value: Aerospike, Apache Ignite, ArangoDB, Berkeley DB, Couchbase, Dynamo, FoundationDB, InfinityDB, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, SciDB, SDBM/Flat File dbm, ZooKeeper
- Graph: AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J, OrientDB, Virtuoso

Type	Notable examples of this type
Key-value cache	Apache Ignite, Couchbase, Coherence, eXtreme Scale, Hazelcast, Infinispan, Memcached, Redis, Velocity
Key-value store	ArangoDB, Aerospike, Couchbase, Redis
Key-value store (eventually consistent)	Oracle NoSQL Database, Dynamo, Riak, Voldemort
Key-value store (ordered)	FoundationDB, InfinityDB, LMDB, MemcacheDB
Tuple store	Apache River, GigaSpaces
Object database	Objectivity/DB, Perst, ZopeDB
Document store	ArangoDB, BaseX, Clusterpoint, Couchbase, CouchDB, DocumentDB, eXist-db, IBM Domino, MarkLogic, MongoDB, Qizx, RethinkDB, Elasticsearch
Wide Column Store	Amazon DynamoDB, Bigtable, Cassandra, Scylla, HBase, Hypertable
Native multi-model database	ArangoDB, Cosmos DB, OrientDB, MarkLogic

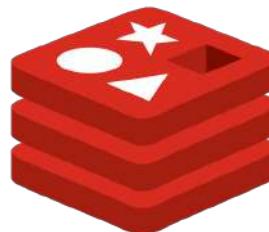
# Key/Value Store

- A key–value database, or key–value store, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, and a data structure more commonly known today as a dictionary or hash table.

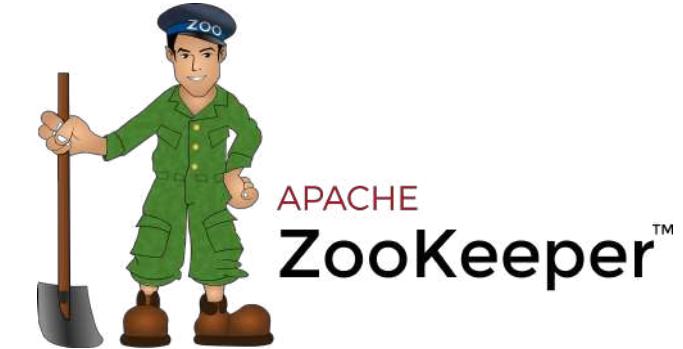


# Key/Value Store

- No schema!

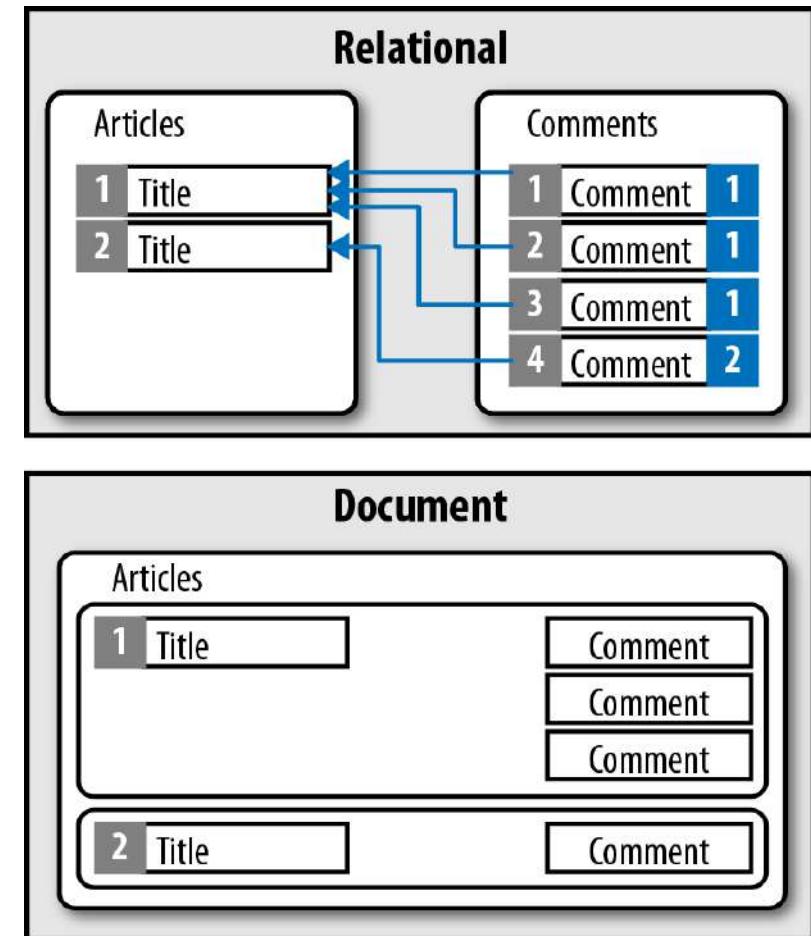


redis



# Document-Oriented Database

- A document-oriented database, or document store, is a computer program designed for storing, retrieving and managing document-oriented information, also known as semi-structured data.

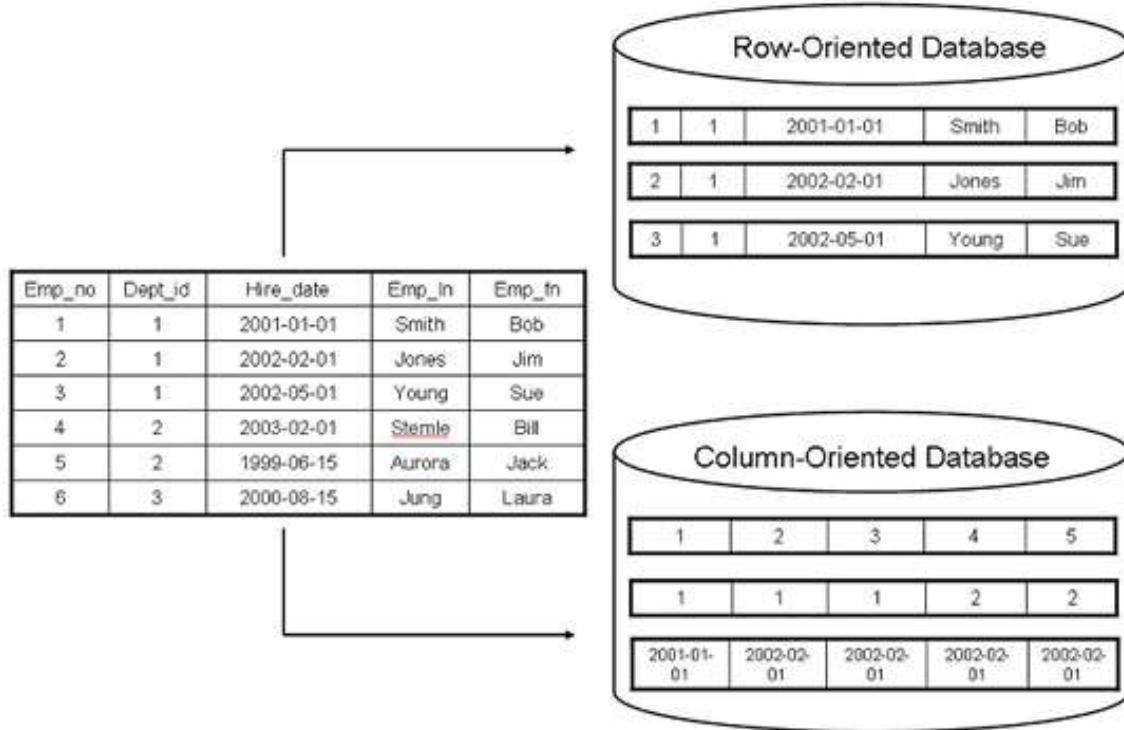


# Column-Oriented Database

- A column-oriented DBMS or columnar DBMS is a database management system (DBMS) that stores data tables by column rather than by row.



# Column-Oriented Database



This diagram compares the storage of a single product (chair) across four databases: a Row-Oriented Database, a Column-Oriented Database, and two intermediate tables.

**Row-Oriented Database:** Shows a single table with 3 rows and 4 columns. A dashed line separates this from the other structures.

sID	product	location	available
1	chair	Boston	15
2	chair	Ohio	6
3	chair	Denver	9

**Column-Oriented Database:** Shows three separate tables corresponding to the columns of the Row-Oriented database. A dashed line separates these from the Row-Oriented database.

sID	product
1	chair
2	chair
3	chair

sID	location
1	Boston
2	Ohio
3	Denver

sID	available
1	15
2	6
3	9

# Graph Database

- A graph database (GDB) is a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data.



# BASE

- BASE stands for
  - Basic Availability
    - The database appears to work most of the time.
  - Soft-state
    - Stores don't have to be write-consistent, nor do different replicas have to be mutually consistent all the time.
  - Eventual consistency
    - Stores exhibit consistency at some later point (e.g., lazily at read time).

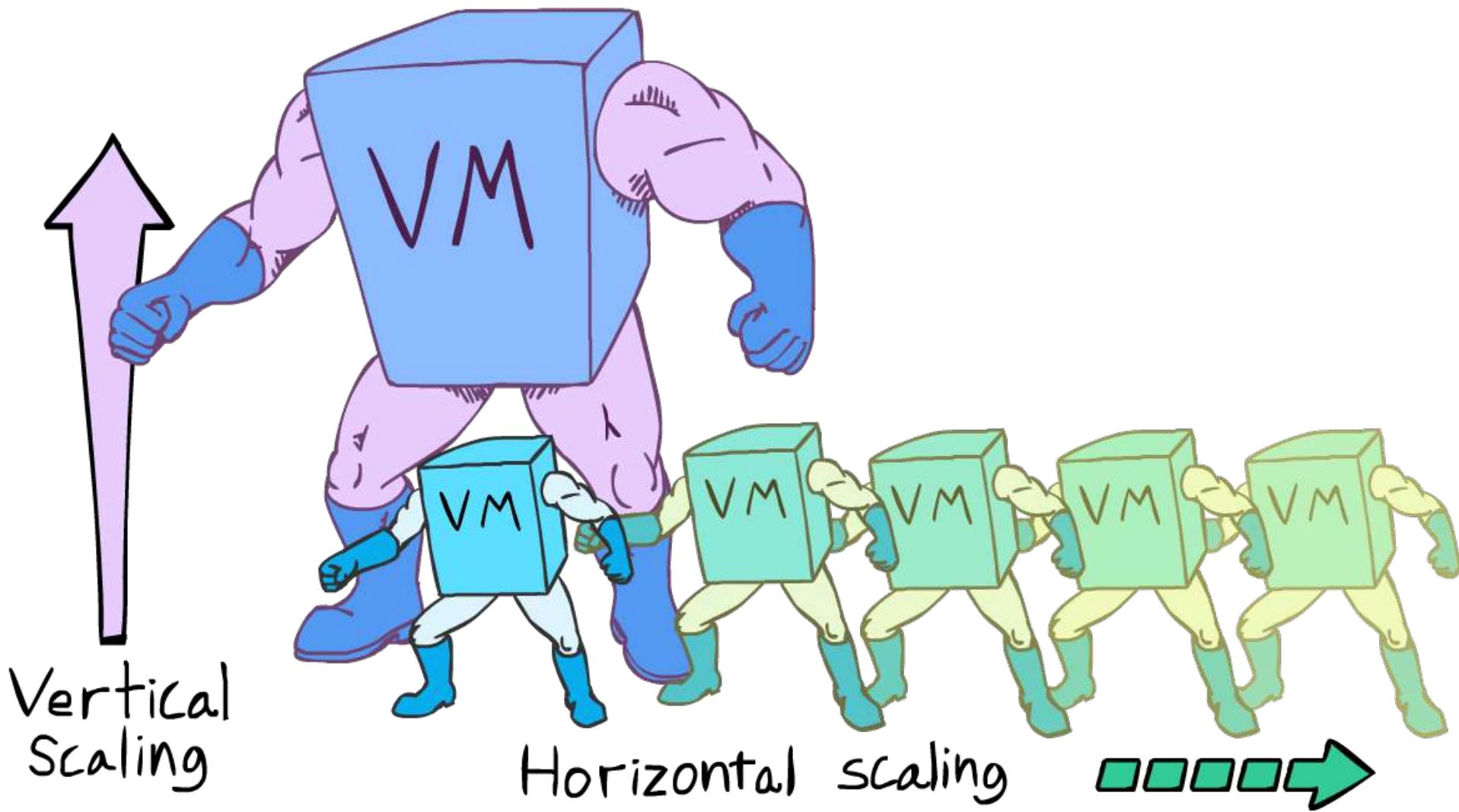
# ACID vs. BASE

- There's no right answer to whether your application needs an ACID versus BASE consistency model.
  - Developers and data architects should select their data consistency trade-offs on a case-by-case basis – not based just on what's trending or what model was used previously.

# Database Scaling

*Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth.*

— [Wikipedia](#)



# Vertical Scaling (Scale up)

- Suppose you have a database server with 10GB memory and it has exhausted.
  - Now, to handle more data, you buy an expensive server with memory of 2TB. Your server can now handle large amounts of data.
- This is called Vertical Scaling. It is buying a single expensive and bigger server.

# Horizontal Scaling (Scale out)

*“In pioneer days they used oxen for heavy pulling, and when one ox couldn’t budge a log, they didn’t try to grow a larger ox. We shouldn’t be trying for bigger computers, but for more systems of computers.”*

–Grace Hopper

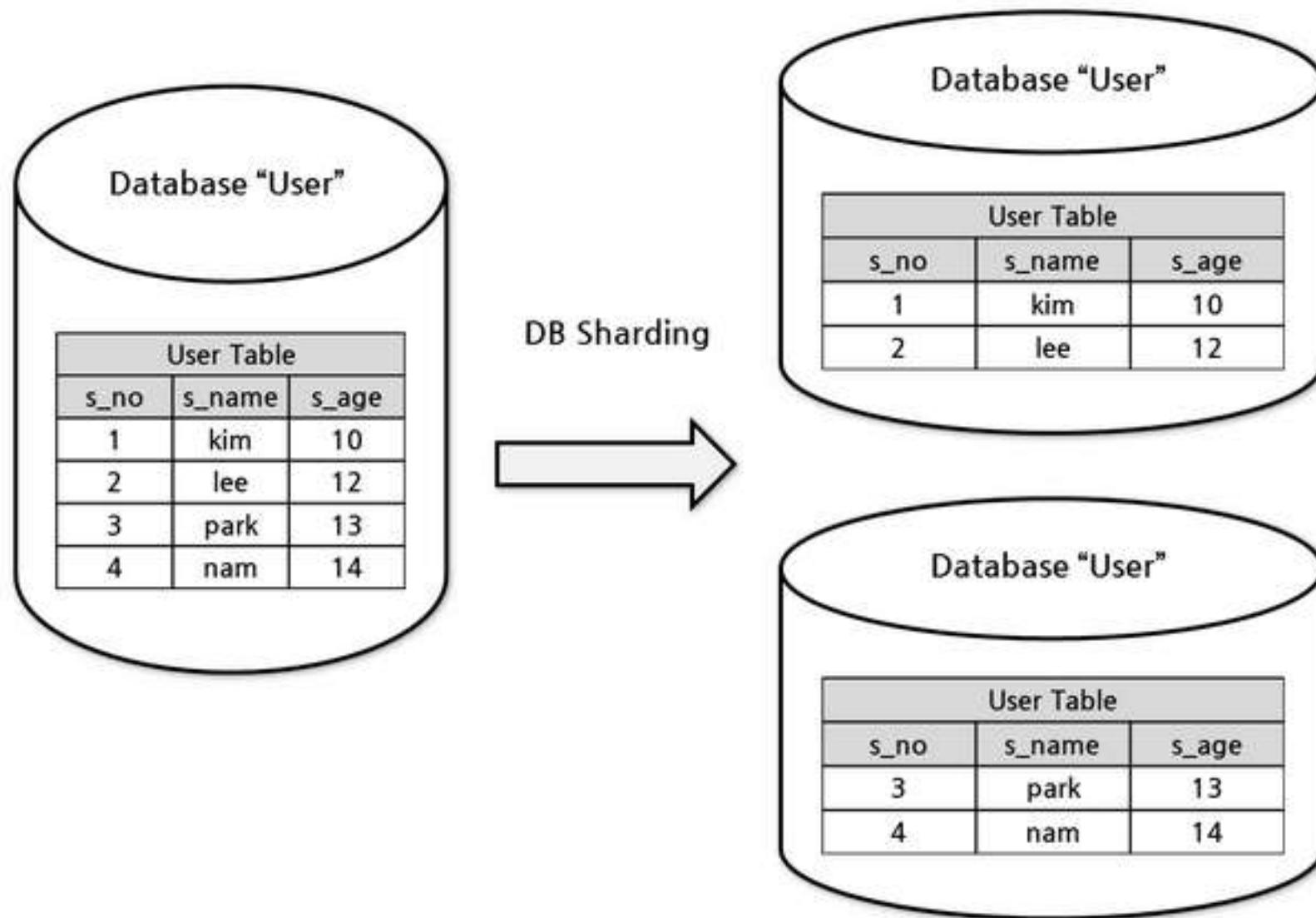
# Horizontal Scaling (Scale out)

- Horizontal Scaling, as the image depicts is scaling of the server horizontally by adding more machines.
- It divides the data set and distributes the data over multiple servers, or shards.

# Sharding

- A database shard, or simply a shard, is a horizontal partition of data in a database or search engine.
- Each shard is held on a separate database server instance, to spread load.

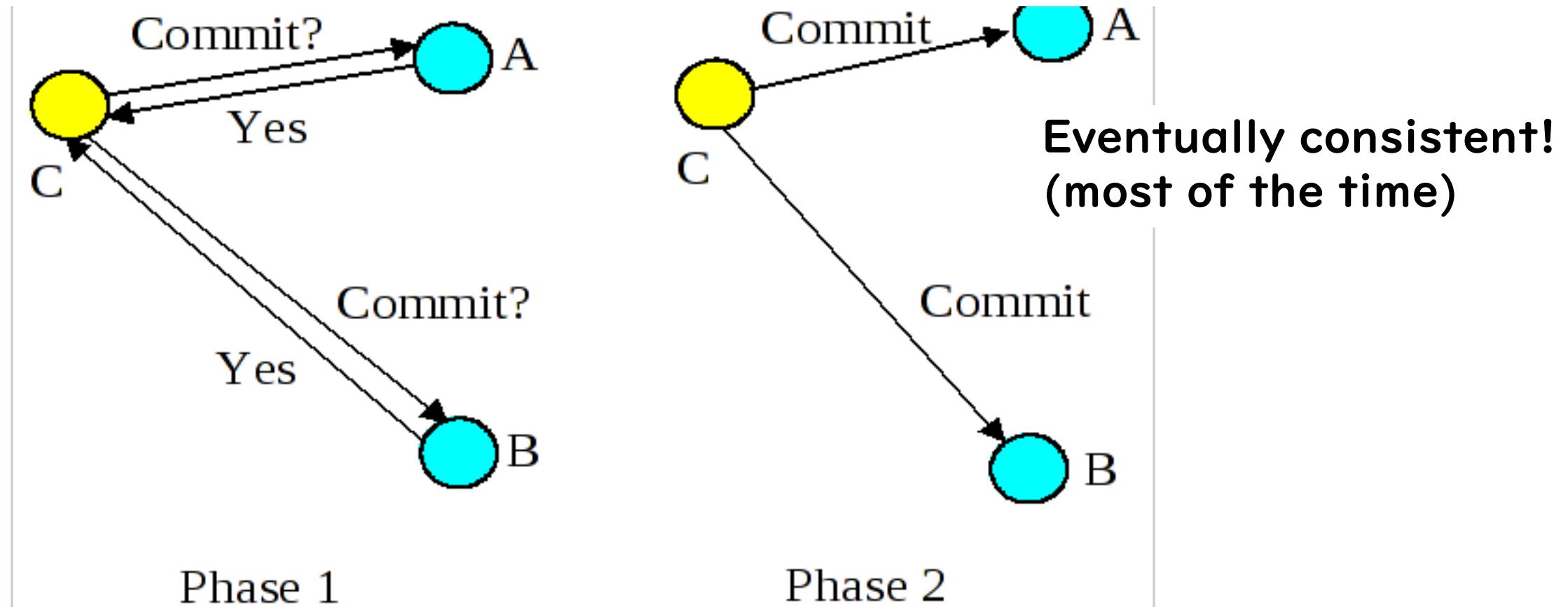
# Sharding



# Distributed Database

- A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

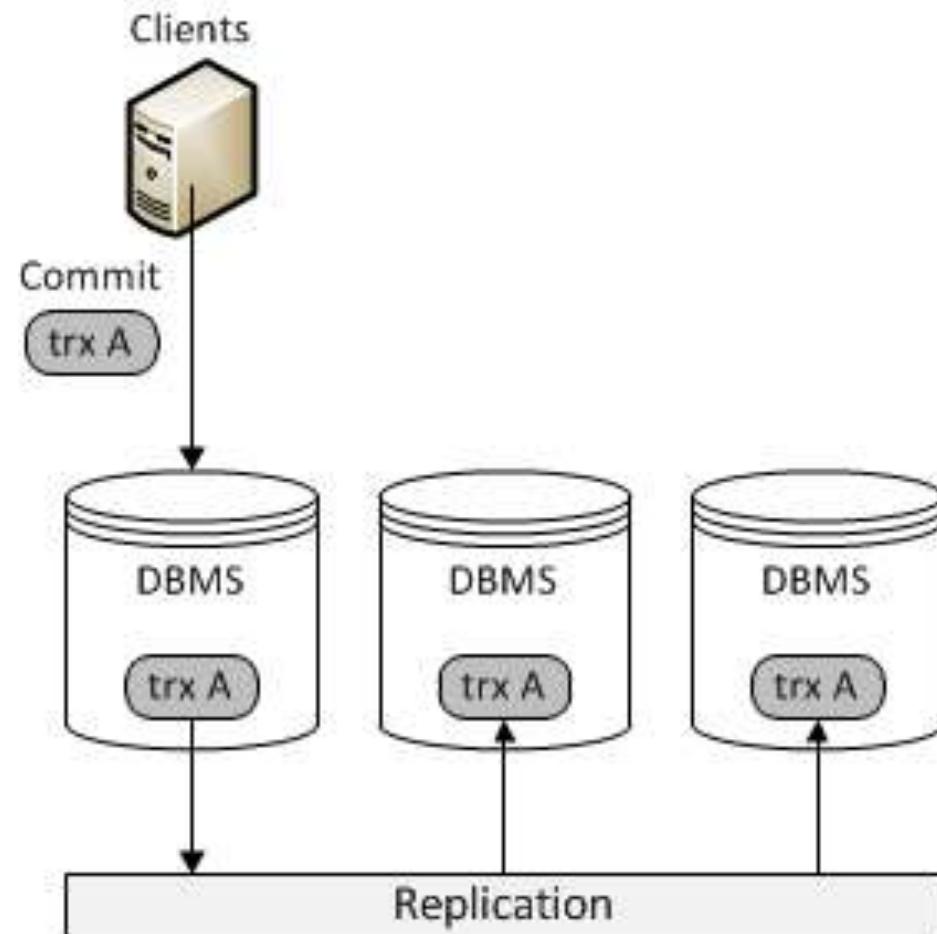
# Two Phase Commit (2PC)



# Replication

- Replication in computing involves sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility.

# Replication



<https://galeracluster.com/library/documentation/tech-desc-introduction.html>

# CAP Theorem

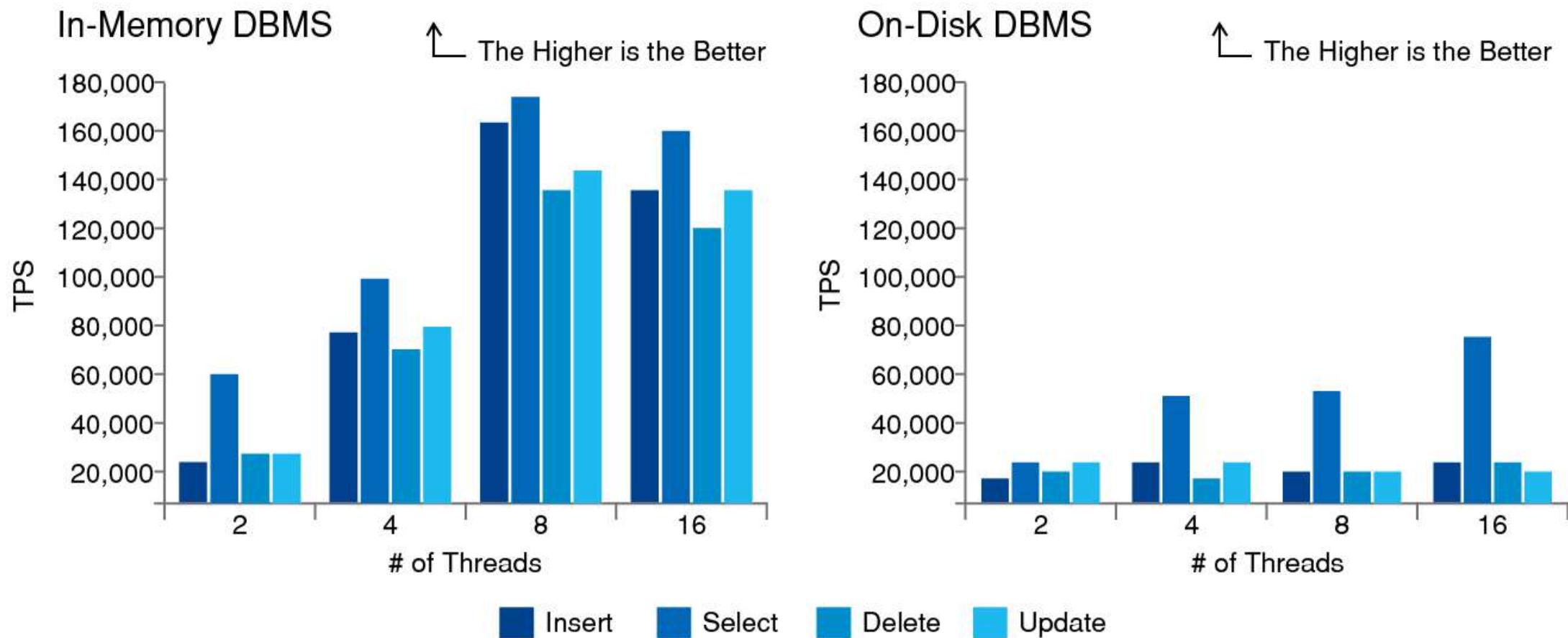
- The CAP theorem states that it is impossible for a **distributed data store** to simultaneously provide more than two out of the following three guarantees:
  - Consistency
  - Availability
  - Partition tolerance

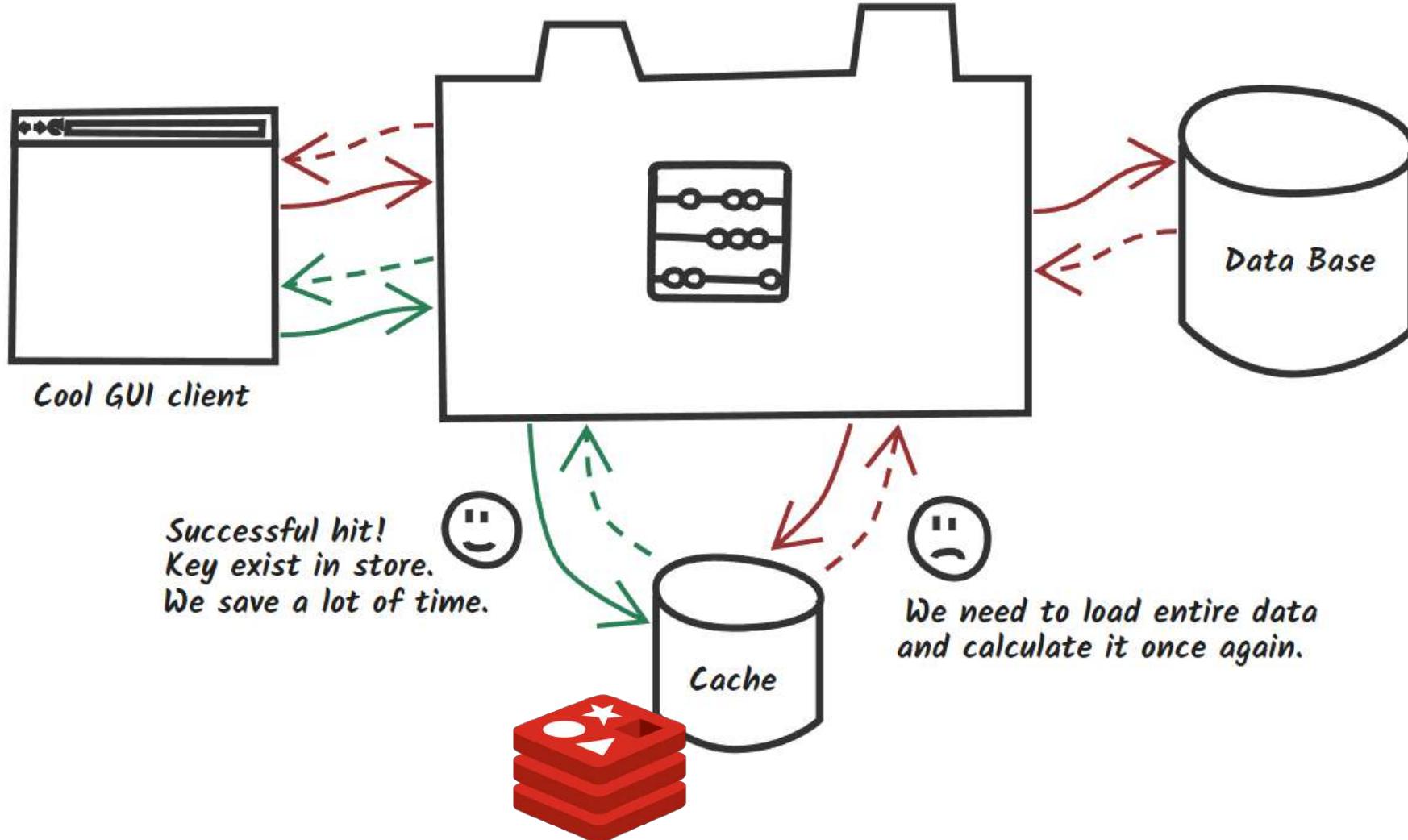
# CAP Theorem is misleading!

- Thinking of CAP as a “You-Pick-Two” theorem is misguided and dangerous.
- <https://medium.com/@sumitsethia94/consistency-or-availability-of-databases-do-you-really-understand-cap-8ecf2b3bb099>

# In-Memory Database

- An in-memory database is a type of purpose-built database that relies primarily on memory for data storage, in contrast to databases that store data on disk or SSDs.





# Workshop I – Create your database

- Create a table namely `products` with the following attributes
  - `product_id` as integer
  - `product_name` as 30 vary characters
  - `product_quantity` as integer
  - `product_price` as decimal(7, 4)
- Make `product_id` a primary key



# Database Programming using Go

# Go Package ‘sql’

- Package sql provides a generic interface around SQL (or SQL-like) databases.
- The sql package must be used in conjunction with a database driver.
  - <https://github.com/golang/go/wiki/SQLDrivers>

# Go Package ‘sql’

- <http://go-database-sql.org/>

# Database Driver

- A database driver is a computer program that implements a protocol (ODBC or JDBC) for a database connection.
  - Open Database Connectivity (ODBC) is a standard application programming interface (API) for accessing database management systems (DBMS).
  - Java Database Connectivity (JDBC)

# Importing a database driver

```
import (
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
)
```

- You need to install the database driver first.
  - go get "github.com/go-sql-driver/mysql"
- P.S. The command uses for installing mysql/mariadb drivers only.

# Connecting to a database

```
func main() {
    db, err := sql.Open("mysql",
        "user:password@tcp(127.0.0.1:3306)/hello")
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()
}
```

# Check the connection status

```
err = db.Ping()  
if err != nil {  
    // do something here  
}
```

# Executing Queries

```
rows, err := db.Query("select id, name from users where id = ?", 1)
if err != nil {
    log.Fatal(err)
}
defer rows.Close()

for rows.Next() {
    err := rows.Scan(&id, &name)
    if err != nil {
        log.Fatal(err)
    }
    log.Println(id, name)
}
```

# CRUD

- CRUD stands for Create, Read, Update, Delete
- These four operations usually use when we program with databases.

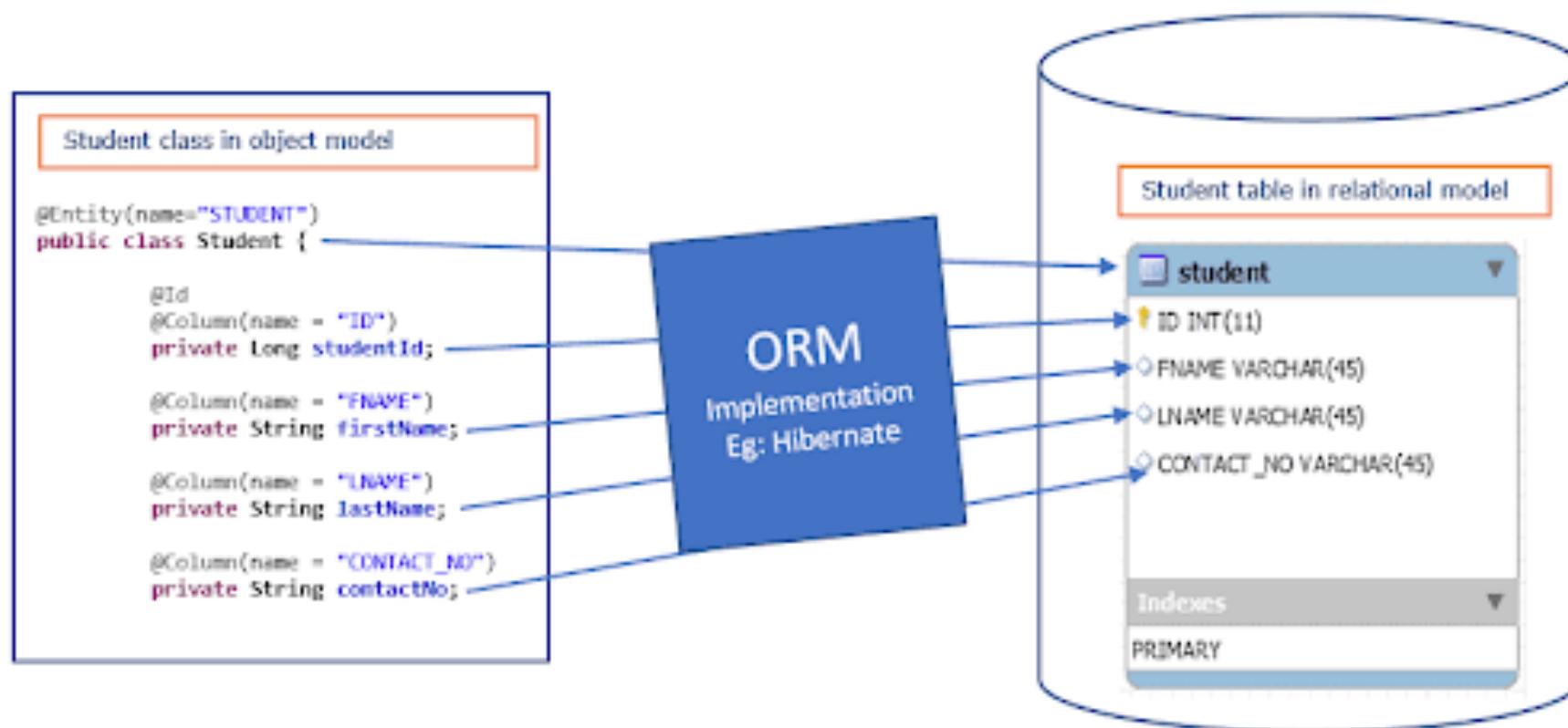
# Workshop 2 – CRUD the database

- Write CRUD functions in Go
  - Read all products
  - Insert a product
  - Update a product (by using product\_key)
  - Delete a product (by using product\_key)

# Object-Relational Mapping (ORM)

- Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages.

# Object-Relational Mapping (ORM)



<https://javabydeveloper.com/orm-object-relational-mapping/>

▲ THe ORM is an implementation detail of the Repository. The ORM just makes it easy to access the db tables in an OOP friendly way. That's it.

13

▼ The repository abstract persistence access, whatever storage it is. That is its purpose. THe fact that you're using a db or xml files or an ORM doesn't matter. The Repository allows the rest of the application to ignore persistence details. This way, you can easily test the app via mocking or stubbing and you can change storages if it's needed. TOday you might use MySql, tomorrow you'll want to use NoSql or Cloud Storage. Do that with an ORM!



Repositories deal with Domain/Business objects (from the app point of view), an ORM handles db objects. A business objects IS NOT a db object, first has behaviour, the second is a glorified DTO, it only holds data.

**Edit** You might say that both repository and ORM abstract access to data, however the devil is in the details. The repository abstract the access to **all storage concerns**, while the ORM abstract access to a **specific RDBMS**

In a nutshell, Repository and ORM's have DIFFERENT purposes and as I've said above, the ORM is always an implementation detail of the repo.

You can also check [this post](#) about more details about the repository pattern.

share improve this answer follow

add a comment

edited Sep 4 '18 at 7:51

 Prashant Kumar  
25 ● 1 ● 1 ● 7

answered Apr 15 '12 at 8:48

 MikeSW  
SAPIENS ● 15.1k ● 3 ● 33 ● 50

[https://stackoverflow.com/questions/10155517/  
repository-pattern-vs-orm](https://stackoverflow.com/questions/10155517/repository-pattern-vs-orm)

# The fantastic ORM library for Golang

```
$ go get -u gorm.io/gorm
```



STARS

22K

FORKS

2.5K

CONTRIBUTORS

224

FOLLOWERS

2.9K

JINZHU

1.5K

V2 RELEASE NOTE

V1.20.7

```
type Product struct {
    gorm.Model
    Code string
    Price uint
}

func main() {
    db, err := gorm.Open(sqlite.Open("test.db"), &gorm.Config{})
    if err != nil {
        panic("failed to connect database")
    }

    // Migrate the schema
    db.AutoMigrate(&Product{})

    // Create
    db.Create(&Product{Code: "D42", Price: 100})

    // Read
    var product Product
    db.First(&product, 1) // find product with integer primary key
    db.First(&product, "code = ?", "D42") // find product with code D42

    // Update - update product's price to 200
    db.Model(&product).Update("Price", 200)
    // Update - update multiple fields
    db.Model(&product).Updates(Product{Price: 200, Code: "F42"}) // non-zero fields
    db.Model(&product).Updates(map[string]interface{}{"Price": 200, "Code": "F42"})

    // Delete - delete product
    db.Delete(&product, 1)
}
```

# Brief Overview of Software Architectural Patterns

Outer Architecture

API  
Gateway

Service  
Router

API/Service  
Registry

Analytics

Security,  
Access mgt.

Microservices Cluster

Microservices Cluster

Microservices Cluster

Data



Messaging Channels

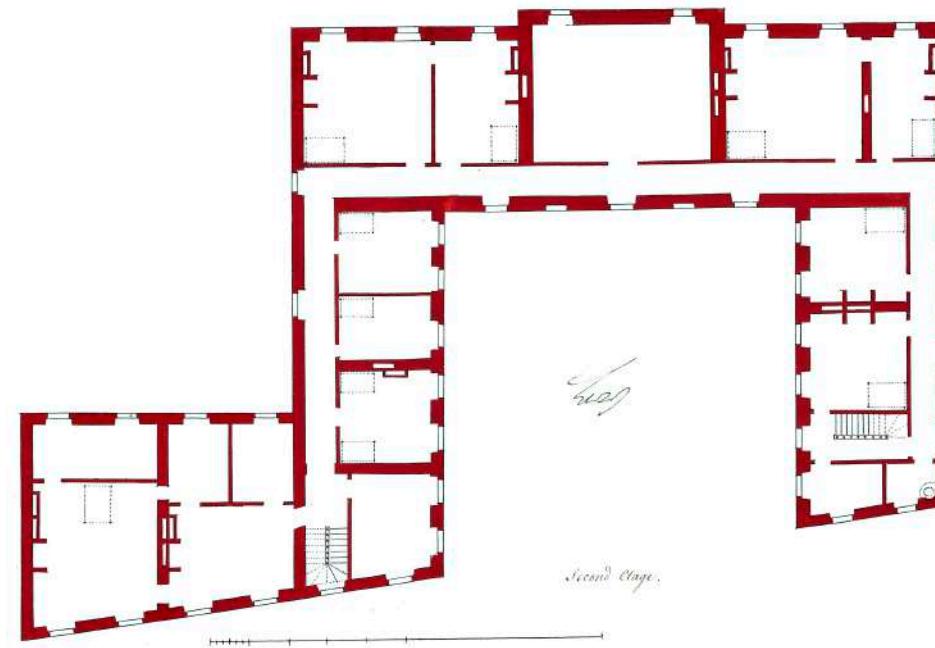
DevOps

Cloud Native

Cloud Native

# Architecture

- Architecture is both the process and the product of planning, designing, and constructing buildings or other structures.



# Software Architecture

- Software architecture refers to the fundamental structures of a software system and the discipline of creating such structures and systems.
- The architecture of a software system is a metaphor, analogous to the architecture of a building.

# Software Architecture vs. Software Design

- The architecture supports non-functional requirements while design supports functional requirements of a system.
- Architecture focus on inter-module dependencies, accessing other modules while the design is within a module.
- <https://medium.com/@arnavgupta180/software-design-vs-software-architecture-e2b90c06bbb5>

# Architectural Component

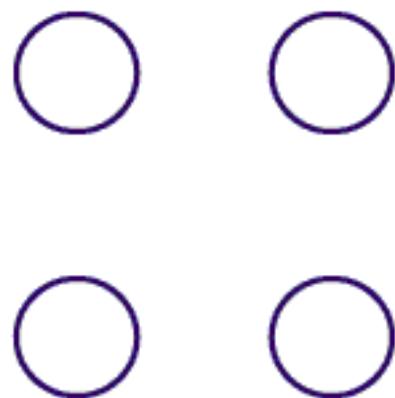
- Software Component
  - Data storage
  - Data access logic
  - Application logic
  - Presentation logic
- Hardware Component
  - Computers (clients, servers)
  - Networks

# Cohesion and Coupling

- Coupling is the degree of interdependence between software modules.
- Cohesion defines to the degree to which the elements of a module belong together.

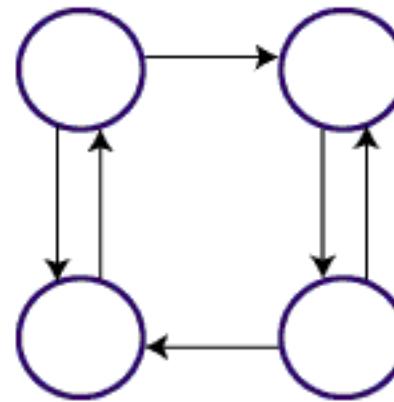
# Cohesion and Coupling

Module Coupling



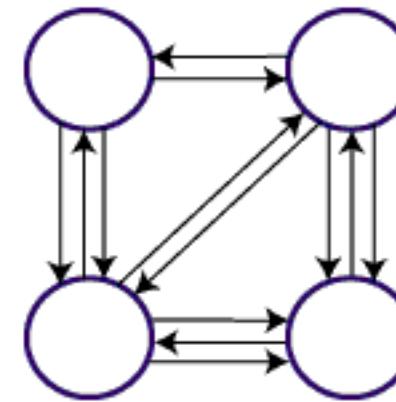
Uncoupled: no  
dependencies

(a)



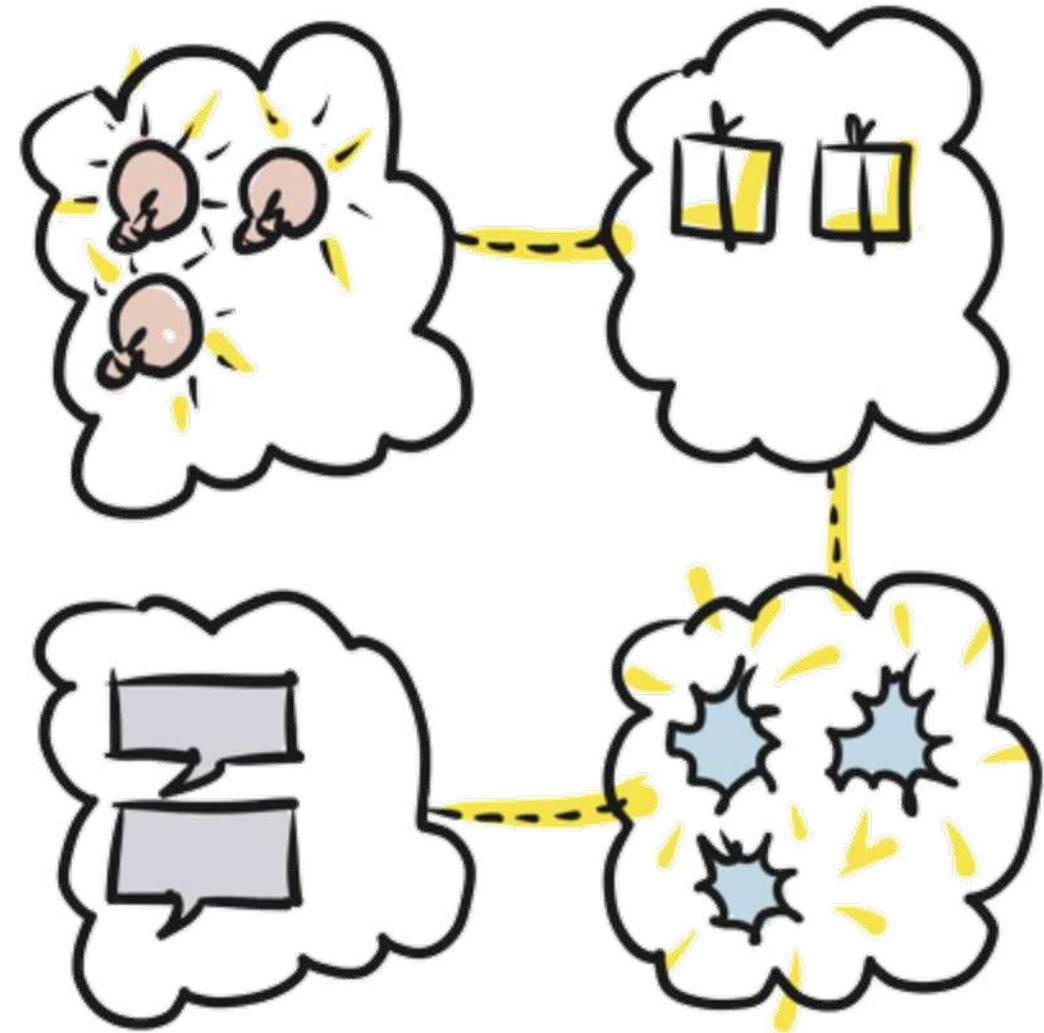
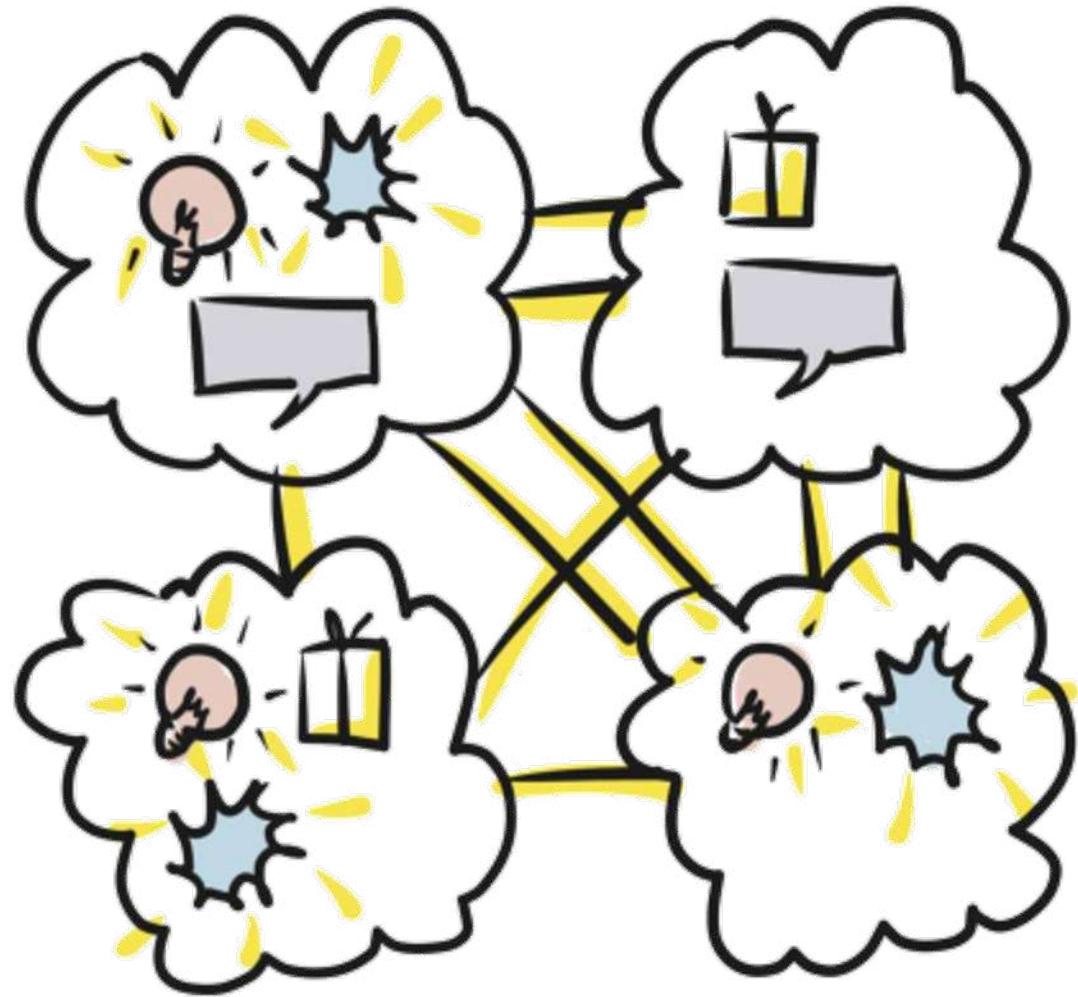
Loosely Coupled:  
Some dependencies

(b)



Highly Coupled:  
Many dependencies

(c)

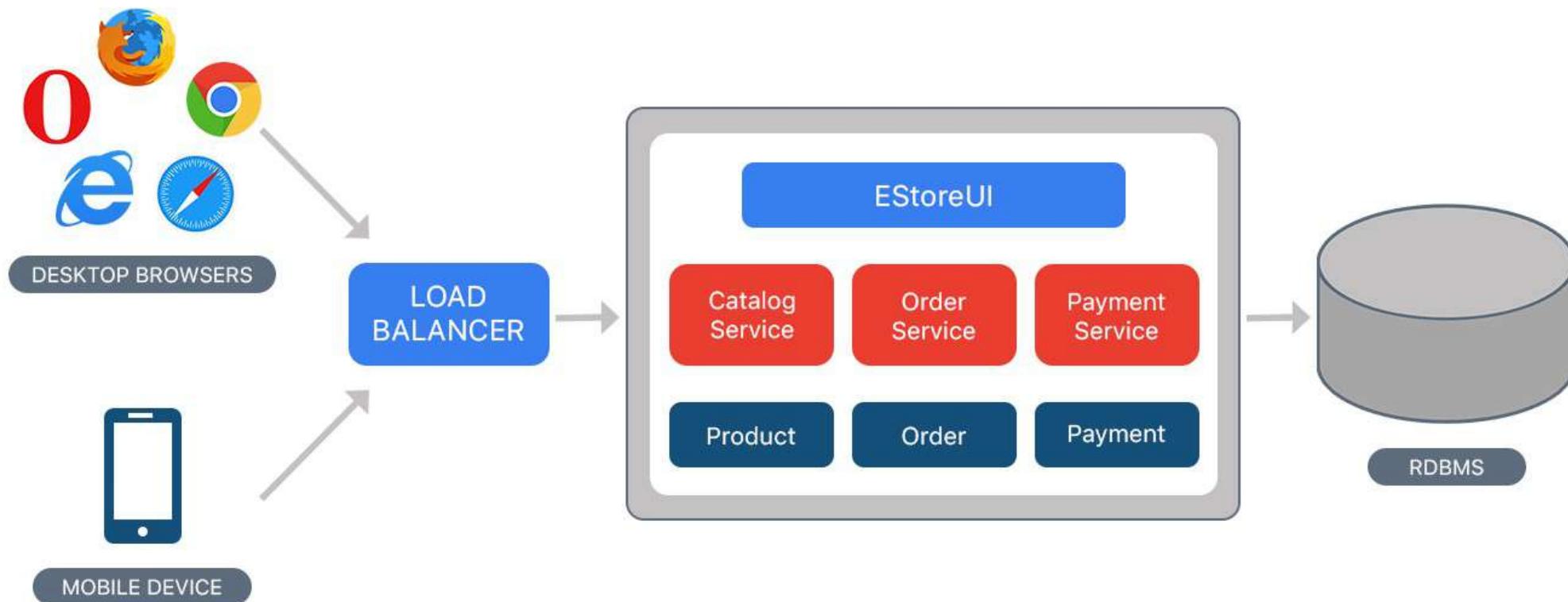


*High Cohesion,  
Low Coupling*

# Monolithic Architecture

- Monolith means composed all in one piece.
- The Monolithic application describes a single-tiered software application in which different components combined into a single program from a single platform.

# Monolithic Architecture



# Benefits of Monolithic

- Simple to develop
- Simple to test
- Simple to deploy
- Simple to scale horizontally by running multiple copies behind a load balancer

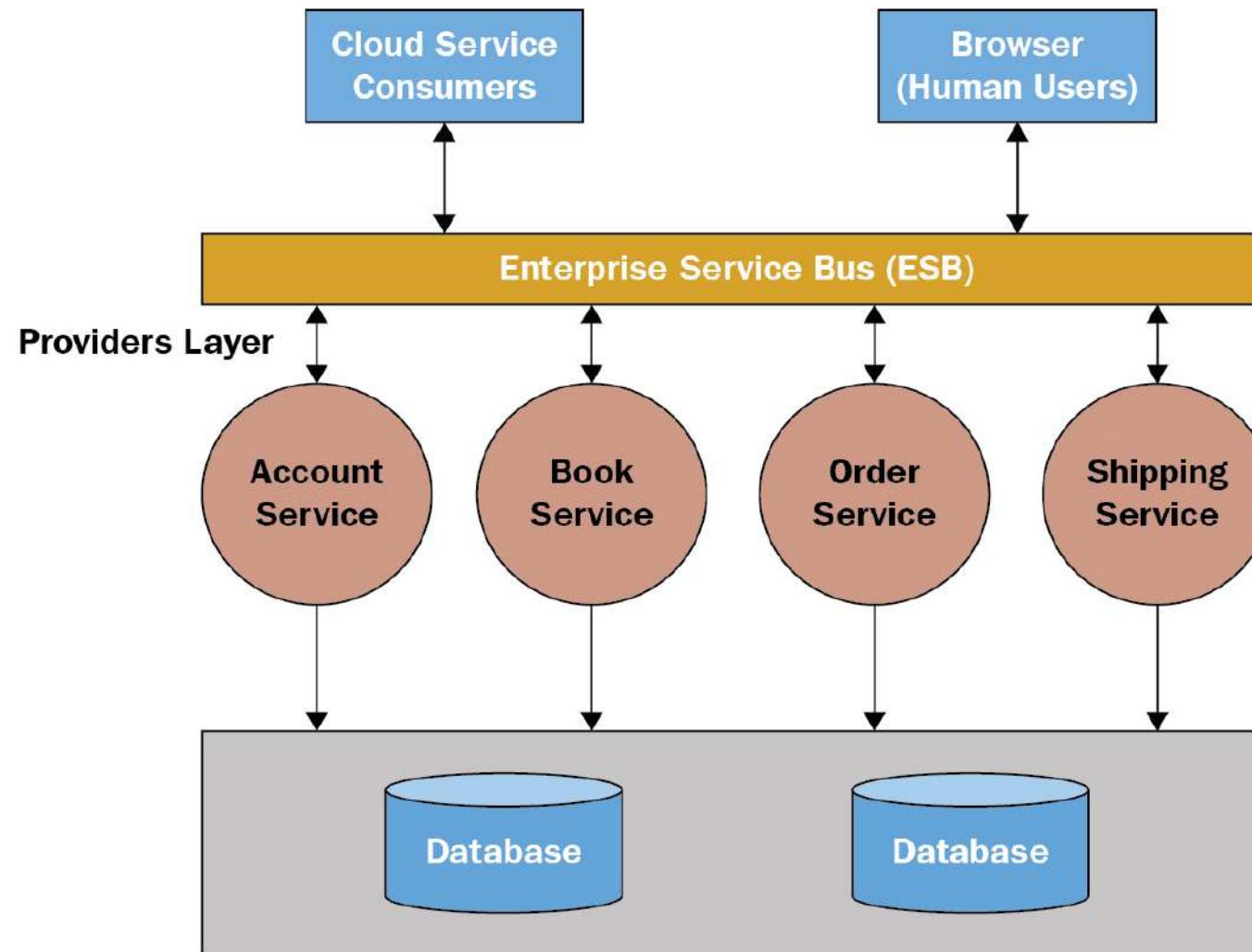
# Drawbacks of Monolithic

- Maintenance
- The size of the application can slow down the start-up time.
- You must redeploy the entire application on each update.
- Monolithic applications can also be challenging to scale
- Reliability
- Difficult to adopting new and advance technologies.

# Service Oriented Architecture (SOA)

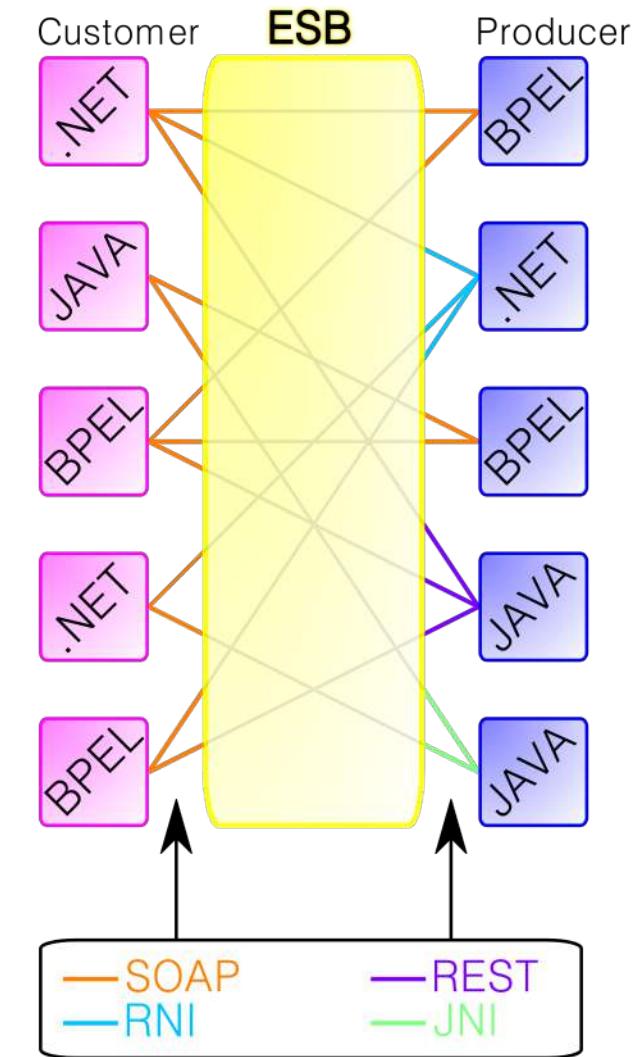
- Service-oriented architecture (SOA) is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.

## Consumers Layer



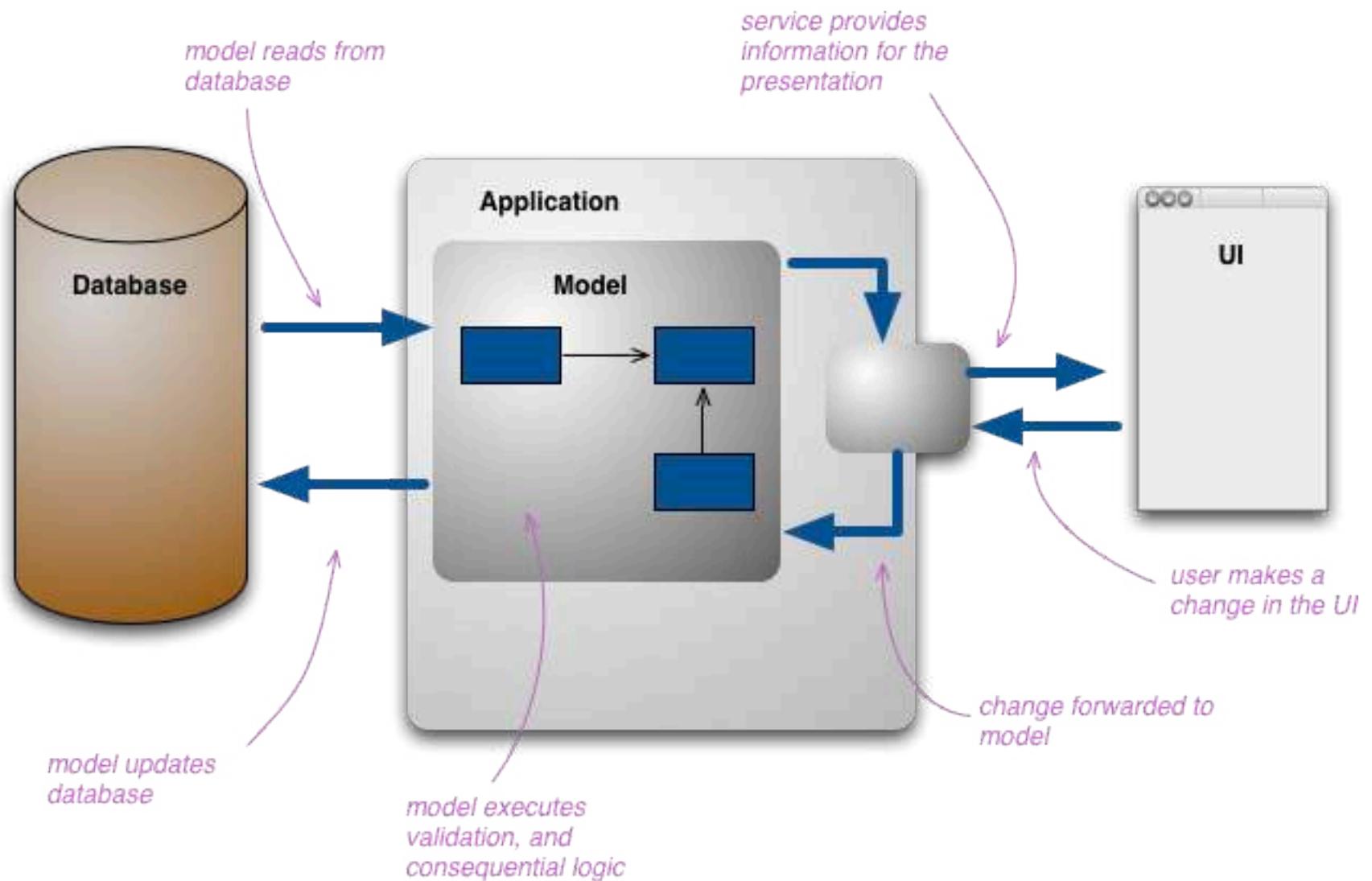
# Enterprise Service Bus (ESB)

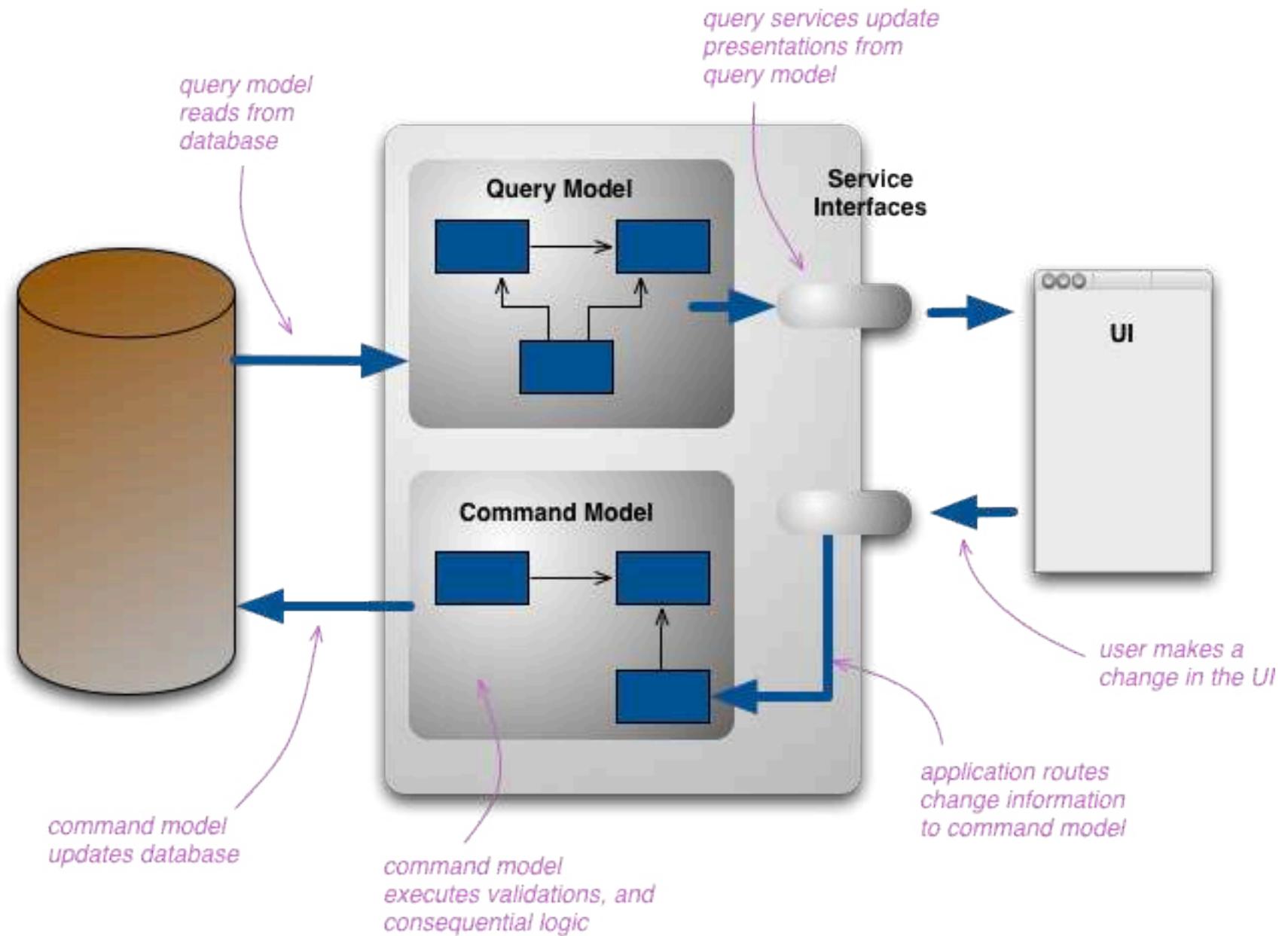
- An enterprise service bus (ESB) implements a communication system between mutually interacting software applications in a service-oriented architecture (SOA).



# Command Query Responsibility Segregation

- The Command and Query Responsibility Segregation (CQRS) pattern separates read and update operations for a data store.



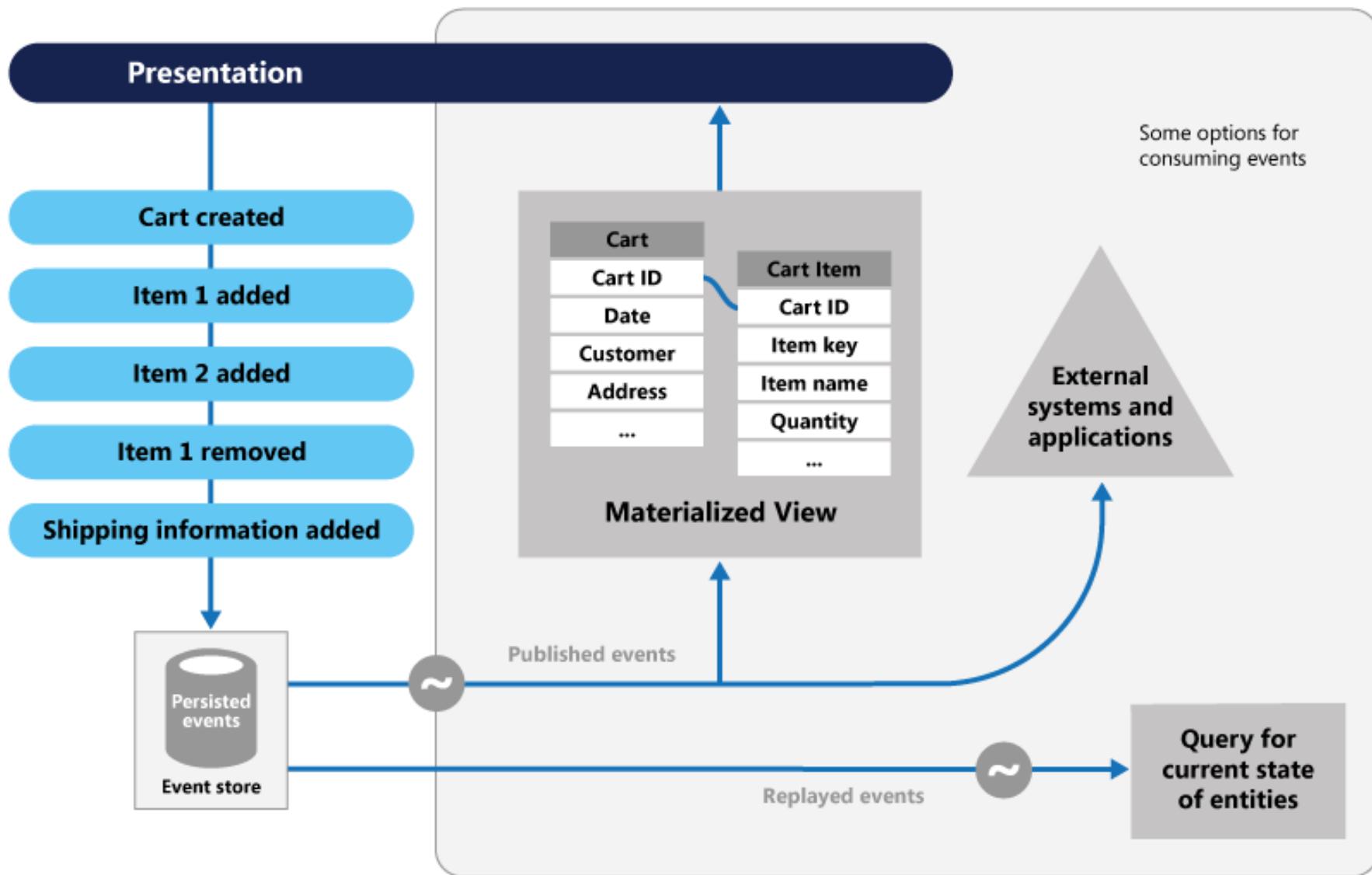


# Command Query Responsibility Segregation

- This pattern has the following benefits:
  - Supports multiple denormalized views that are scalable and performant
  - Improved separation of concerns
  - Necessary in an event sourced architecture
- This pattern has the following drawbacks:
  - Increased complexity
  - Potential code duplication
  - Replication lag/eventually consistent views

# Event Sourcing

- The Event Sourcing pattern defines an approach to handling operations on data that's driven by a sequence of events, each of which is recorded in an append-only store.
- <https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing>



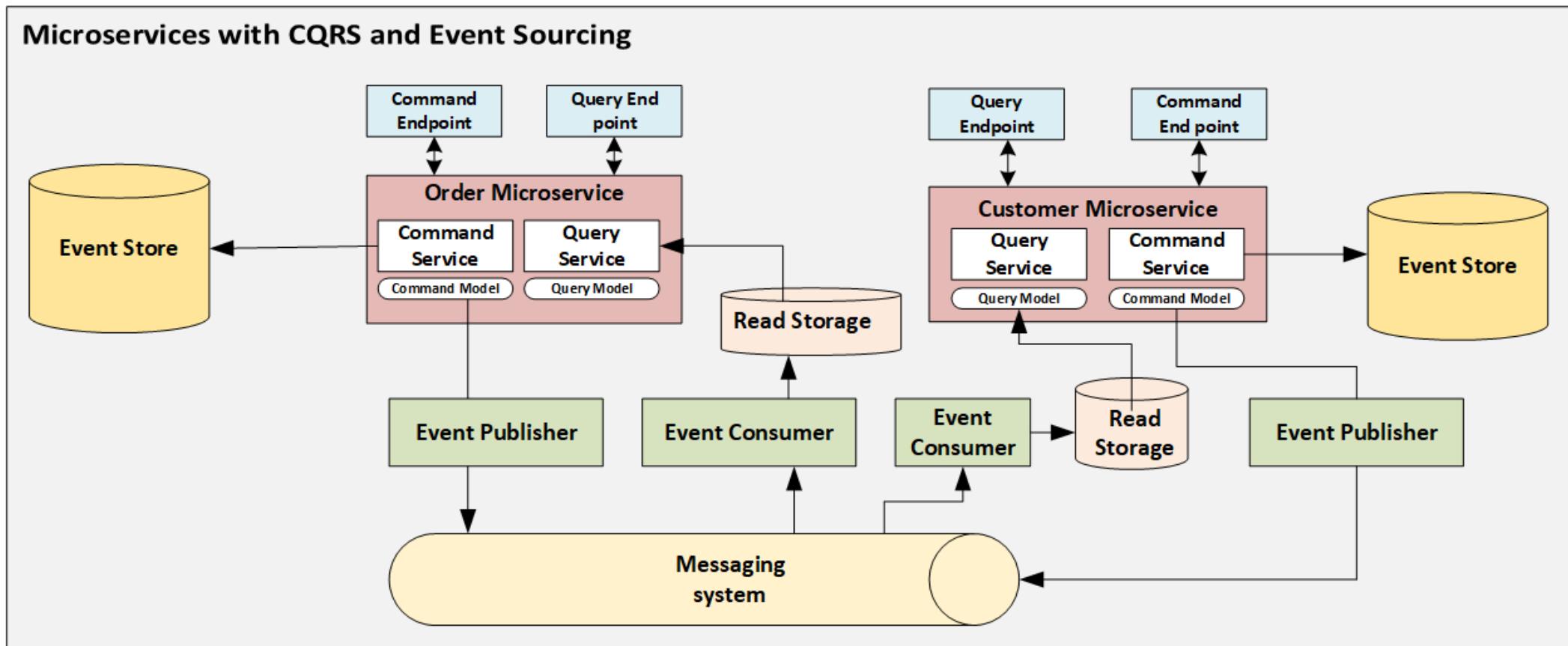
# Event Sourcing

- Improve performance due to asynchronous actions
  - recorded for handling at the appropriate time
- Prevent concurrent updates
- Audit trail
- Replay events

# Event Sourcing

- Only use when **eventually consistency** is acceptable in your system!

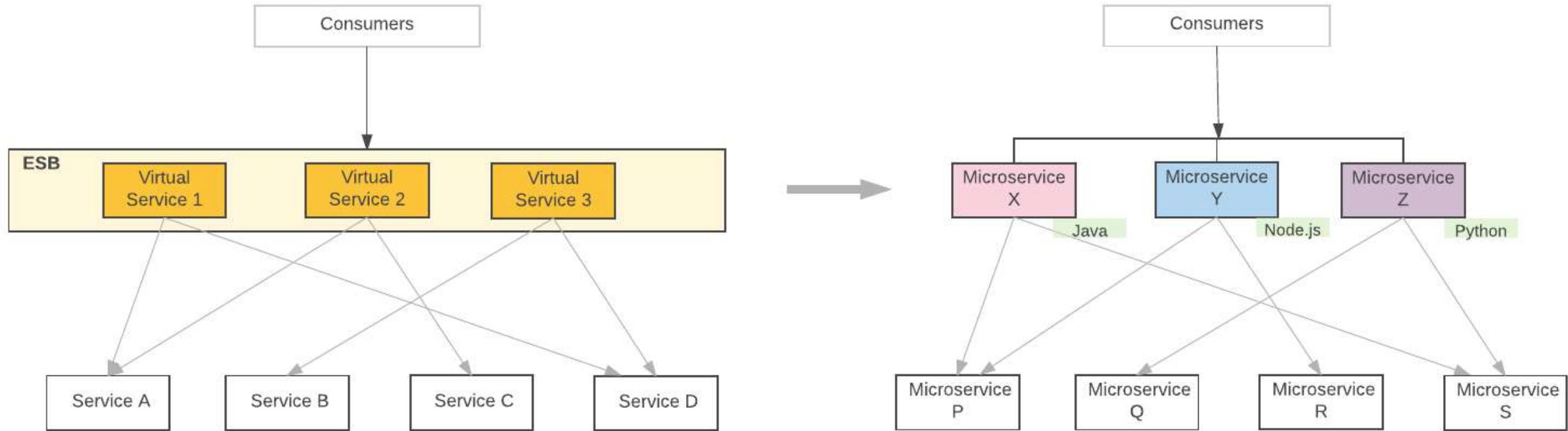
# CQRS with Event Sourcing



# Microservice

- Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small team

# Microservice



# Microservice

- The microservice architecture is not a silver bullet.
  - It has several drawbacks.
- Moreover, when using this architecture there are numerous issues that you must address.

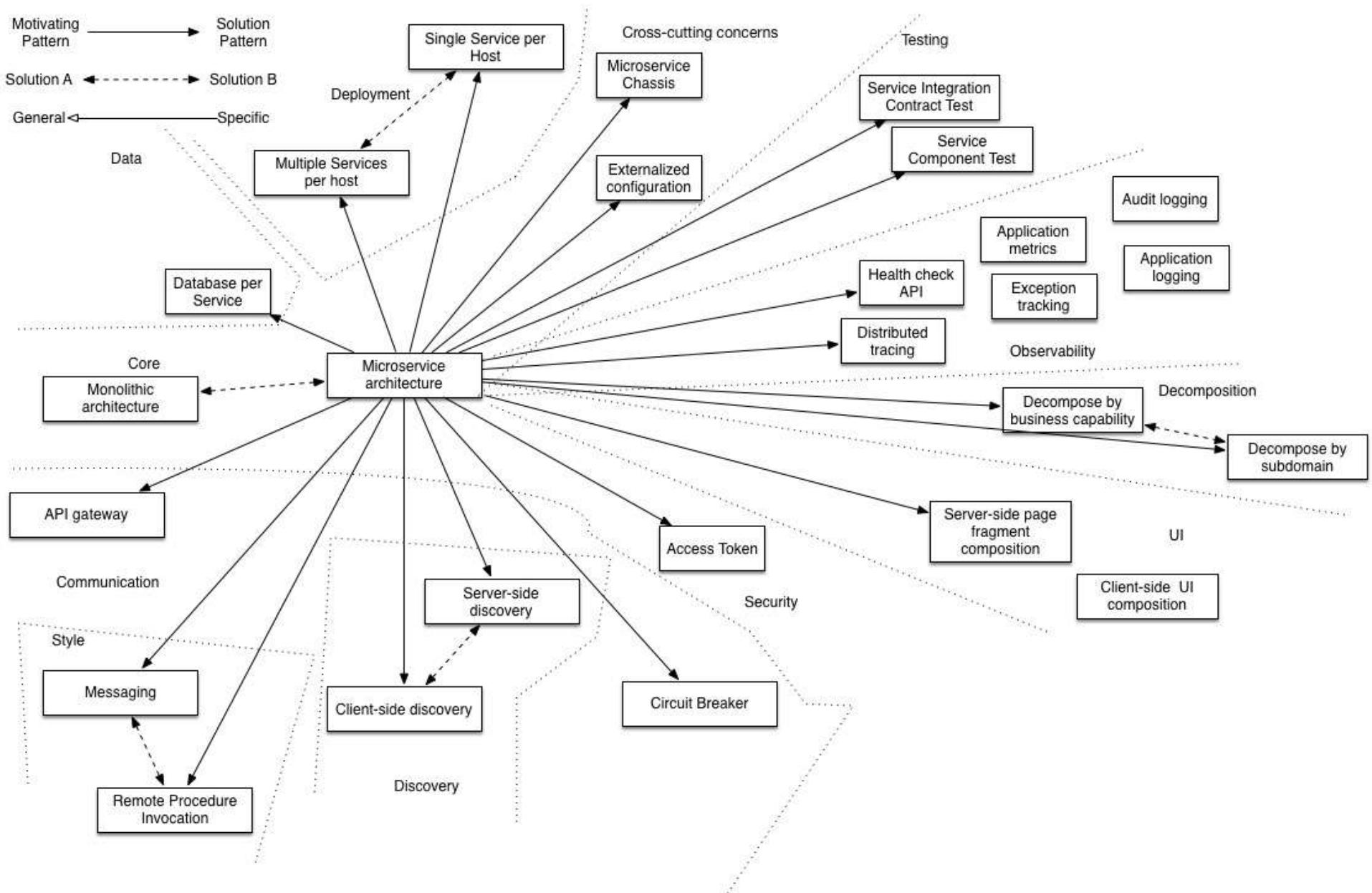
# Microservice

## Forces

- There is a team of developers working on the application
- New team members must quickly become productive
- The application must be easy to understand and modify
- You want to practice continuous deployment of the application
- You must run multiple instances of the application on multiple machines in order to satisfy scalability and availability requirements
- You want to take advantage of emerging technologies (frameworks, programming languages, etc)

# Microservice

- Microservice architecture usually uses **CQRS** and **API composition** pattern for solving querying issues due to **Database per service** pattern.

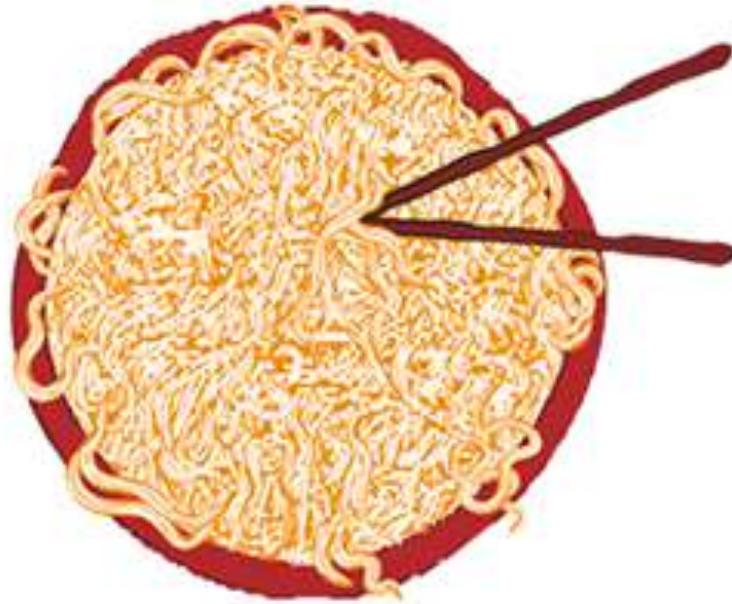


# Microservice

- <https://microservices.io/>

### 1990s and earlier

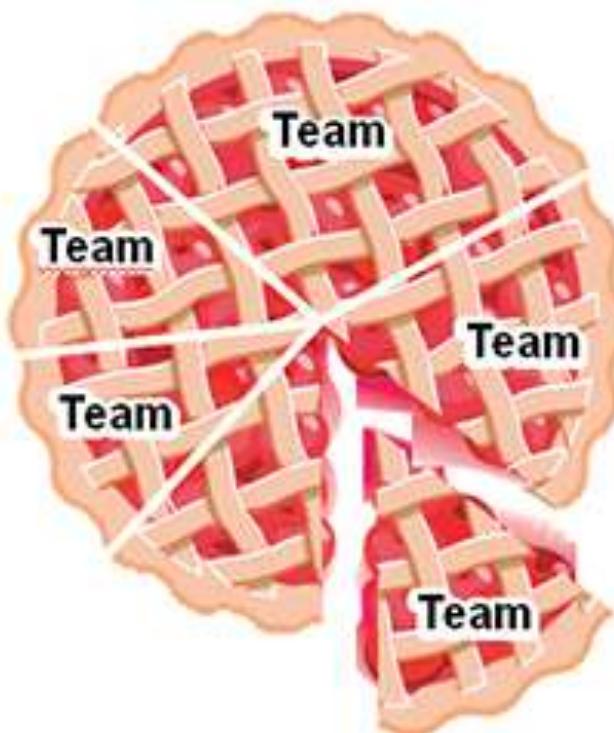
Pre-SOA (monolithic)  
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

### 2000s

Traditional SOA  
Looser coupling



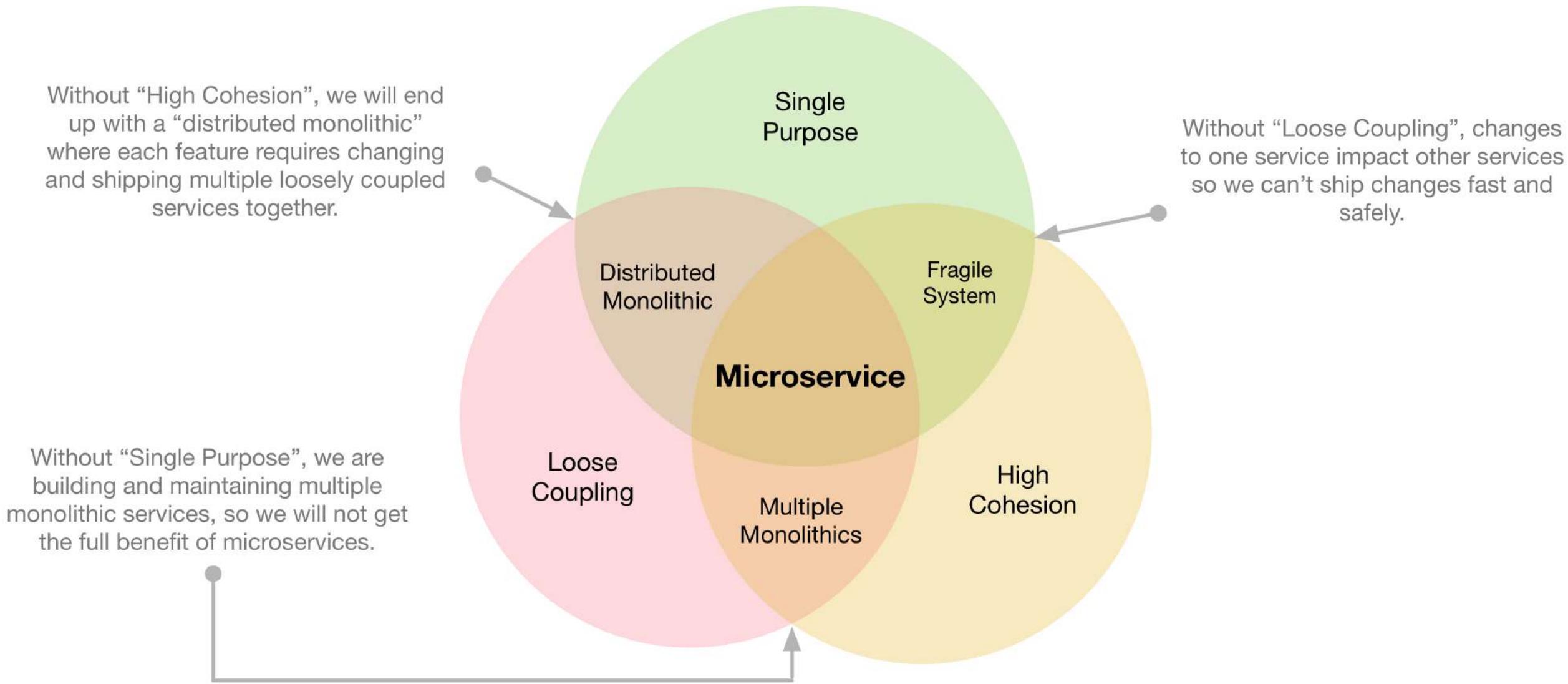
Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

### 2010s

Microservices  
Decoupled



Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.



# Microservice

- The most complex challenge in realizing microservice architecture is not building the services themselves, but the communication between services.

# Software Design Principles

# The pasta theory of programming



### Spaghetti code

Unstructured and hard-to-maintain code caused by lack of style rules or volatile requirements. This architecture resembles a tangled pile of spaghetti in a bowl.



### Lasagna code

Source code with overlapping layers, like the stacked design of lasagna. This code structure makes it difficult to change one layer without affecting others.



### Ravioli code

Isolated bits of code that resemble ravioli. These are easy to understand individually but—taken as a group—add to the app's call stack and complexity.



### Pizza code

A codebase with interconnected classes or functions with unclear roles or responsibilities. These choices result in a flat architecture, like toppings on a pizza.

### Spaghetti Sample

```
1.058 JMPL RBW_SWITCHING  
1.059 LABEL FREQUENCY_READOUT  
1.060 CALL Sub Frequency Readout  
1.061 CALL Sub Frequency Span  
1.062 CALL Sub Noise Sidebands  
1.063 JMPL SYSTEM_SIDEBANDS  
1.064 LABEL WIDE_OFFSETS  
1.065 CALL Sub Noise Wide Offsets  
1.066 JMPL SPURIOUS  
1.067 LABEL SYSTEM_SIDEBANDS  
1.068 CALL Sub System Sidebands  
1.069 CALL Sub Residual  
1.070 CALL Sub Display Switching  
1.071 JMPL SCALE_FIDELITY  
1.072 LABEL SWEEP_TIME  
1.073 CALL Sub Sweep Time  
1.074 JMPL WIDE_OFFSETS  
1.075 LABEL SCALE_FIDELITY  
1.076 CALL Sub Scale Fidelity  
1.077 CALL Sub Input Switching  
1.078 CALL Sub Reference Level  
1.079 JMPL FREQUENCY_RESPONSE  
1.080 LABEL RBW_SWITCHING  
1.081 CALL Sub Resolution BW Switching  
1.082 JMPL RESOLUTION_BANDWIDTH  
1.083 LABEL ABSOLUTE_AMPLITUDE  
1.084 CALL Sub Absolute Amplitude  
1.085 JMPL FREQUENCY_READOUT  
1.086 LABEL RESOLUTION_BANDWIDTH  
1.087 CALL Sub Resolution Bandwidth  
1.088 JMPL AVERAGE_NOISE  
1.089 LABEL FREQUENCY_RESPONSE  
1.090 CALL Sub Freq Response  
1.091 JMPL SWEEP_TIME  
1.092 LABEL AVERAGE_NOISE  
1.093 CALL Sub Displayed Average  
1.094 JMPL RESIDUAL_RESPONSES  
1.095 LABEL SPURIOUS  
1.096 CALL Sub TOI  
1.097 JMPL GAIN_COMPRESSION  
1.098 LABEL 2ND_HARMONIC  
1.099 CALL Sub 2nd Harmonic  
1.100 CALL Sub Other Spurs  
1.101 JMPL END  
1.102 LABEL GAIN_COMPRESSION  
1.103 CALL Sub GC  
1.104 JMPL 2ND_HARMONIC  
1.105 LABEL RESIDUAL_RESPONSES  
1.106 CALL Sub Residual Responses  
1.107 JMPL ABSOLUTE_AMPLITUDE  
1.108 LABEL END
```

### Structured Sample

```
1.058 JMPL ResolutionBandwidth  
1.059 LABEL ResolutionBandwidth_done  
1.060 JMPL ResolutionSwitching  
1.061 LABEL ResolutionSwitching_done  
1.062 JMPL DisplayedAverage  
1.063 LABEL DisplayedAverage_done  
1.064 JMPL ResidualResponses  
1.065 LABEL ResidualResponses_done  
1.066 JMPL AbsoluteAmplitude  
1.067 LABEL AbsoluteAmplitude_done  
1.068 JMPL FrequencyReadout  
1.069 LABEL FrequencyReadout_done  
1.070 JMPL FrequencySpan  
1.071 LABEL FrequencySpan_done  
1.072 JMPL NoiseSidebands  
1.073 LABEL NoiseSidebands_done  
1.074 JMPL SystemSidebands  
1.075 LABEL SystemSidebands_done  
1.076 JMPL Residual  
1.077 LABEL Residual_done  
1.078 JMPL Display Switching  
1.079 LABEL Display Switching_done  
1.080 JMPL ScaleFidelity  
1.081 LABEL ScaleFidelity_done  
1.082 JMPL InputSwitching  
1.083 LABEL InputSwitching_done  
1.084 JMPL ReferenceLevel  
1.085 LABEL ReferenceLevel_done  
1.086 JMPL FreqResponse  
1.087 LABEL FreqResponse_done  
1.088 JMPL SweepTime  
1.089 LABEL SweepTime_done  
1.090 JMPL NoiseWideOffsets  
1.091 LABEL NoiseWideOffsets_done  
1.092 JMPL TOI  
1.093 LABEL TOI_done  
1.094 JMPL GC  
1.095 LABEL GC_done  
1.096 JMPL 2ndHarmonic  
1.097 LABEL 2ndHarmonic  
1.098 JMPL OtherSpurs  
1.099 LABEL OtherSpurs
```

# SOLID Design Principles

- The SOLID principles of Object-Oriented Design include these five principles:
  - SRP – Single Responsibility Principle
  - OCP – Open/Closed Principle
  - LSP – Liskov Substitution Principle
  - ISP – Interface Segregation Principle
  - DIP – Dependency Inversion Principle

# The Twelve-Factor Application

- The twelve-factor app is a methodology for building software-as-a-service apps that:
- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

# The Twelve-Factor Application

- <https://12factor.net/>

# The Twelve-Factor Application

- I. Codebase
  - One codebase tracked in revision control, many deploys
- II. Dependencies
  - Explicitly declare and isolate dependencies
- III. Config
  - Store config in the environment
- IV. Backing services
  - Treat backing services as attached resources

# The Twelve-Factor Application

- V. Build, release, run
  - Strictly separate build and run stages
- VI. Processes
  - Execute the app as one or more stateless processes
- VII. Port binding
  - Export services via port binding
- VIII. Concurrency
  - Scale out via the process model

# The Twelve-Factor Application

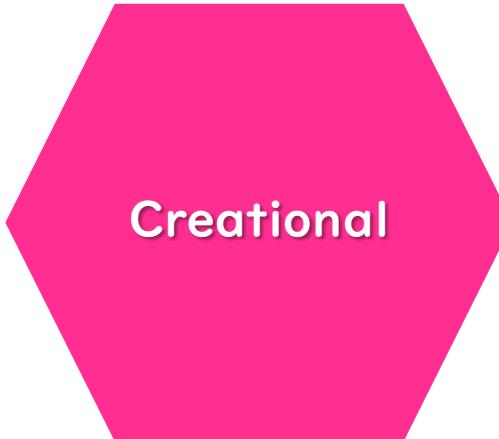
- IX. Disposability
  - Maximize robustness with fast startup and graceful shutdown
- X. Dev/prod parity
  - Keep development, staging, and production as similar as possible
- XI. Logs
  - Treat logs as event streams
- XII. Admin processes
  - Run admin/management tasks as one-off processes

# Software Design Patterns

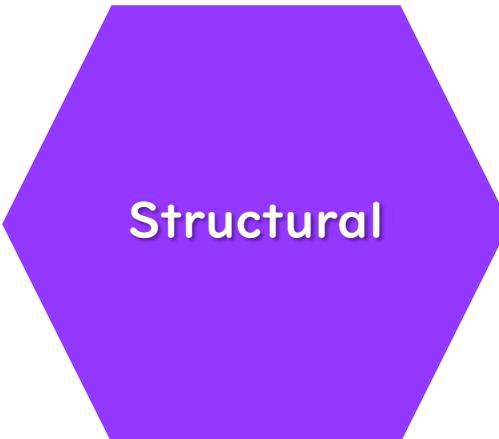
- A software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.

# Software Design Patterns

- Design patterns had originally been categorized into 3 sub-classifications based on kind of problem they solve.



Creation



Structur



Behavio

# Separation of Concerns

- Separation of concerns (SoC) is a design principle for separating a computer program into distinct sections such that each section addresses a separate concern.
- A concern is a set of information that affects the code of a computer program.

# Separation of Concerns



13



Look at a hospital, and think about all of the different roles that are involved in providing care to a patient: triage nurses, doctors, medical assistants, techs, clerical staff, cafeteria, etc.

Is there any one person that knows how *all* of those people get their jobs done? No, because it would be overwhelming. They have to separate out the different responsibilities into distinct roles and the touchpoints between those roles are very specific.

[share](#) [improve this answer](#) [follow](#)

edited Sep 2 '12 at 20:35



user34530

answered Dec 30 '10 at 13:26



RationalGeek

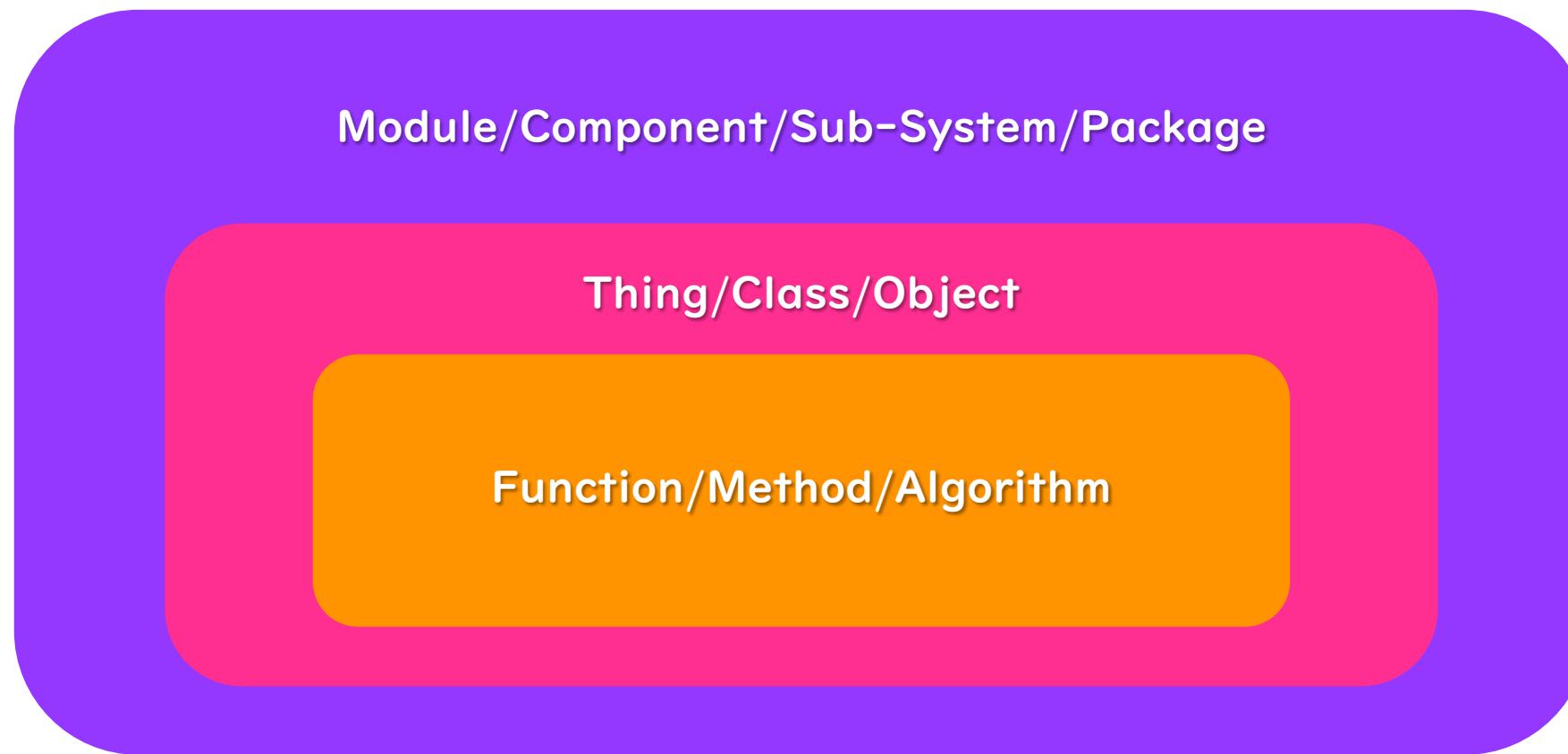
9,977 ● 7 ● 35 ● 55

[add a comment](#)

# Separation of Concerns

- <https://rkay301.medium.com/programming-fundamentals-part-5-separation-of-concerns-software-architecture-f04a900a7c50>

# Separation of Concerns



# Separation of Concerns

- The majority of programmers, unfortunately, do not observe separation of concerns:
  - At the level of Functions, they write **Spaghetti Code**
  - At the level of Things, they write **God Classes/Objects**
  - At the level of the entire architecture of their systems, they write **Monoliths**

# Message Passing Paradigms

# Distributed System

- A distributed system is a system whose components are located on different networked computers, which communicate and coordinate their actions by **passing messages** to one another.

# Message Passing

- There are many different types of implementations for the message passing mechanism, including pure HTTP, RPC-like connectors and message queues.

# Issues in Distributed System

- Transparency
- Fault Tolerance
- Single Point of Failure
- Concurrency

# Transparency

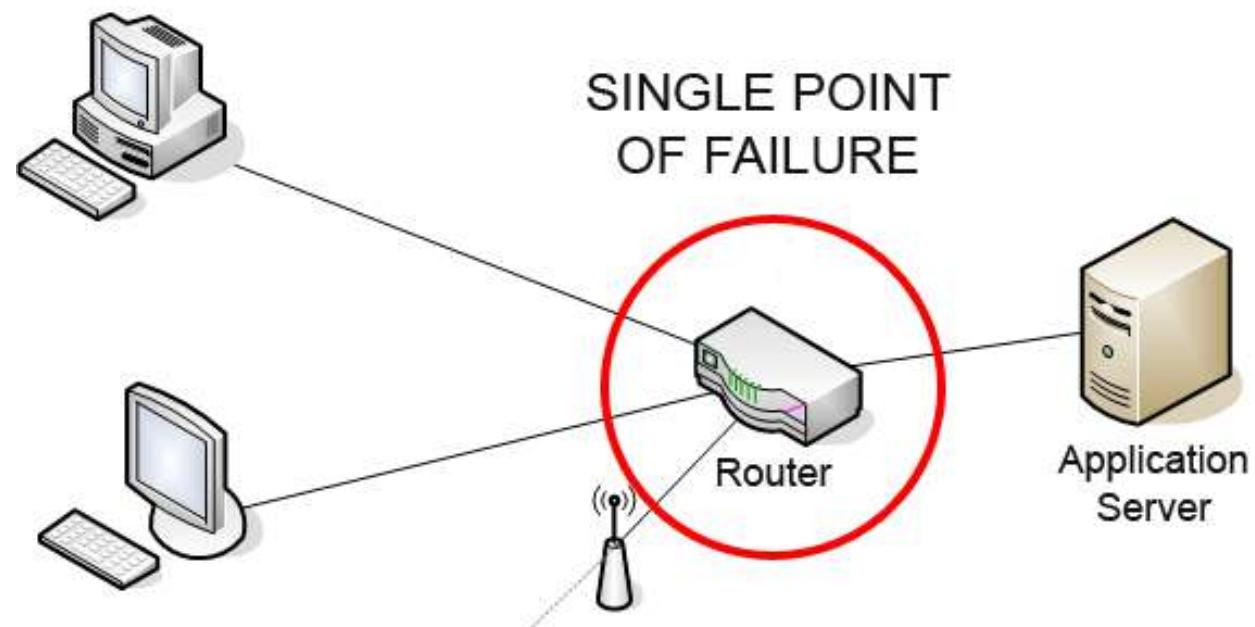
- Transparency in simple words is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components.
- For instance, access transparency, location transparency, failure transparency, concurrency transparency

# Fault Tolerance

- Fault tolerance refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail.

# Single Point of Failure

- A single point of failure (SPOF) is a part of a system that, if it fails, will stop the entire system from working.

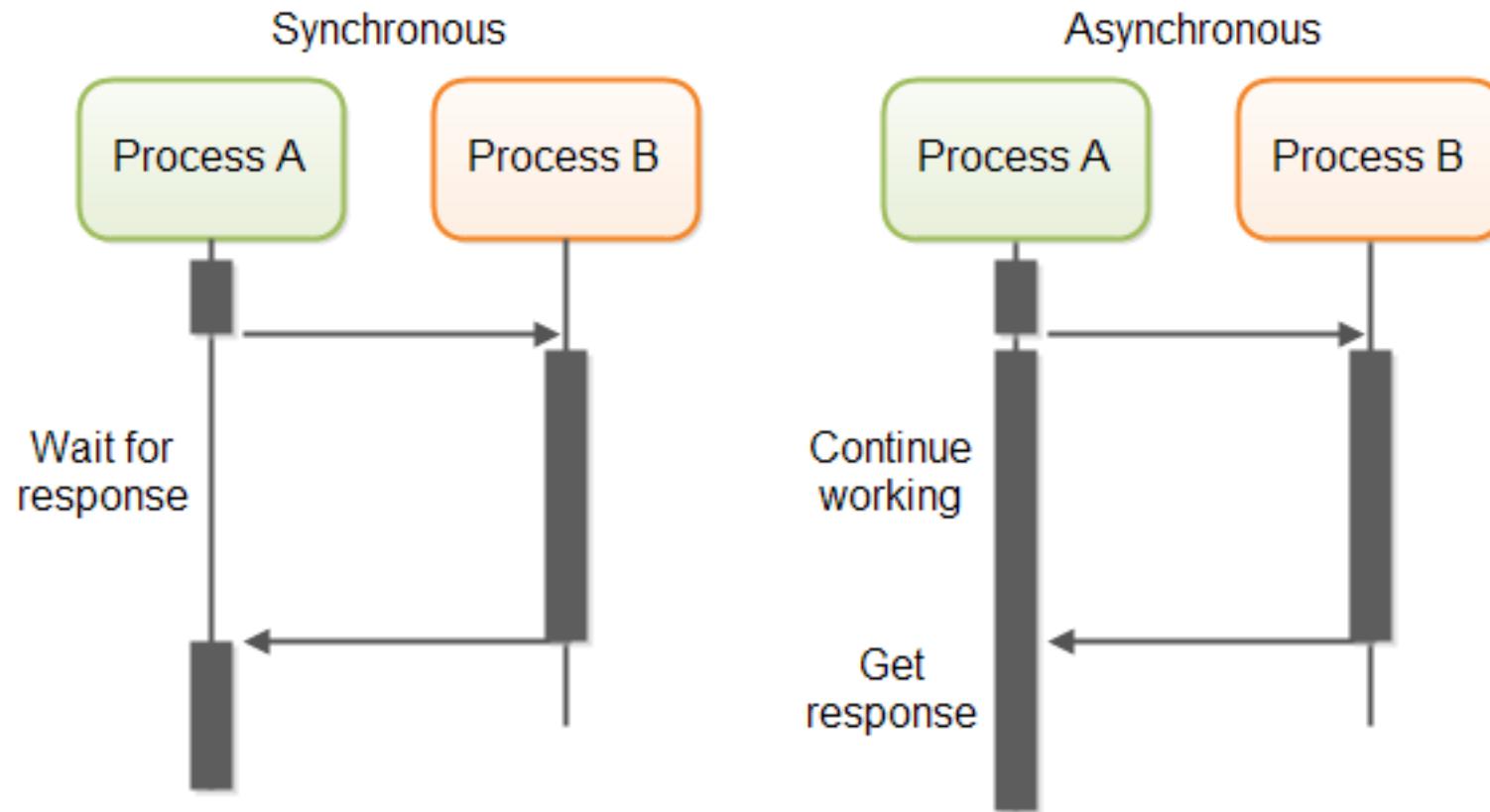


# Concurrency

- Concurrency might lead to several problems.
  - Race conditions
    - Usually solved by using **Synchronization**
  - Deadlocks
    - Processes are blocked
  - Livelocks
    - Processes run but make no progress
  - Starvation



# Synchronous vs. Asynchronous



# Semantic Interoperability

- Semantic interoperability is the ability of computer systems to exchange data with unambiguous, shared meaning.
- Semantic interoperability is a requirement to enable machine computable logic, inferencing, knowledge discovery, and data federation between information systems.

# XML

```
<Log date=May 05,2015>
  <message id=1>
    <participant id= Q12445678D></participant>
    <text> hi </text>
    <time>May 05,2015-16:45:20</time>
  </message>
  <message id=2>
    <participant id= Q12445678D></participant>
    <text> r u there? </text>
    <time>May 05,2015-16:45:26</time>
  </message>
  <message id=3>
    <participant id= F11445E211></participant>
    <text> hi </text>
    <time>May 05,2015-16:45:20</time>
  </message>
</Log>
```

# JSON



A screenshot of a web browser window displaying JSON data. The address bar shows the URL `date.jsontest.com`. The page content is a single JSON object:

```
{  
    "time": "07:21:25 AM",  
    "milliseconds_since_epoch": 1492240885824,  
    "date": "04-15-2017"  
}
```

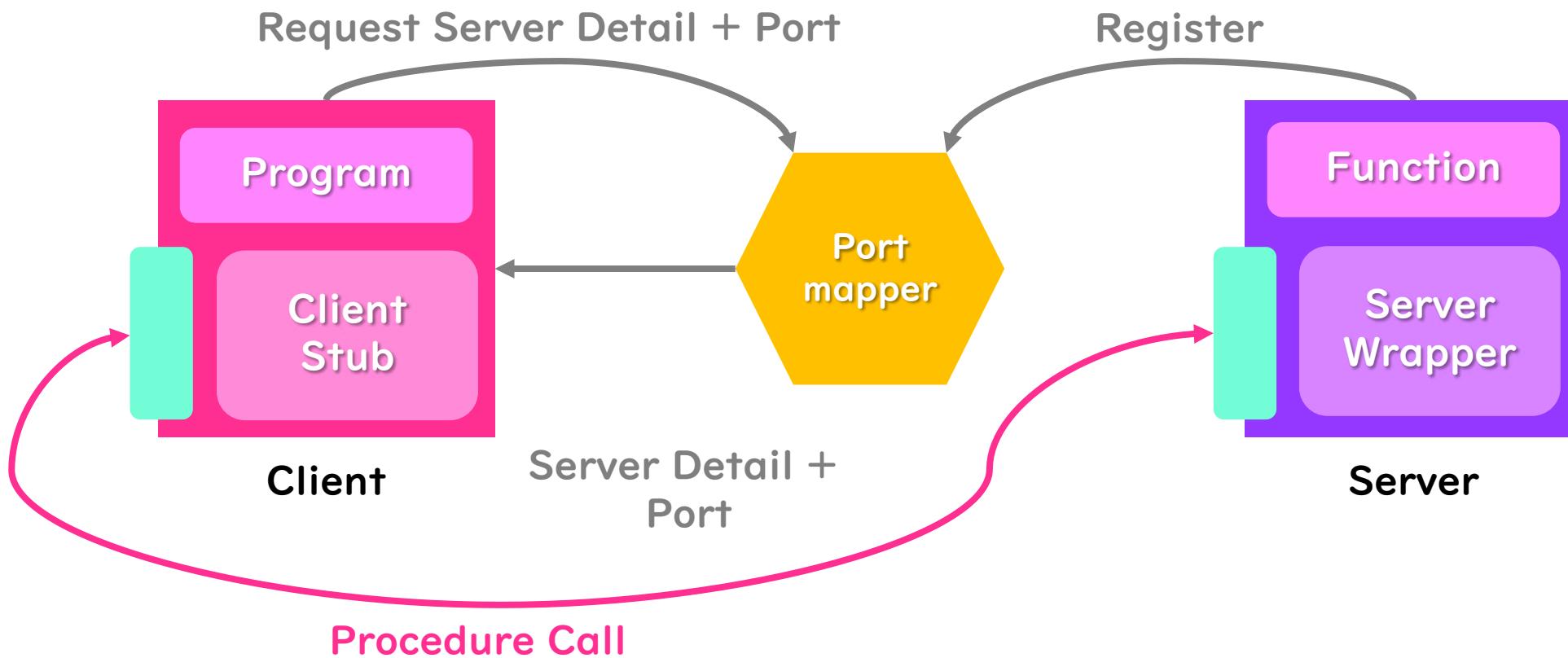
# Interface

- Software interfaces (programming interfaces) are the languages, codes and messages that programs use to communicate with each other and to the hardware.

# Remote Procedure Call (RPC)

- RPC is a request–response protocol.

# Remote Procedure Call (RPC)

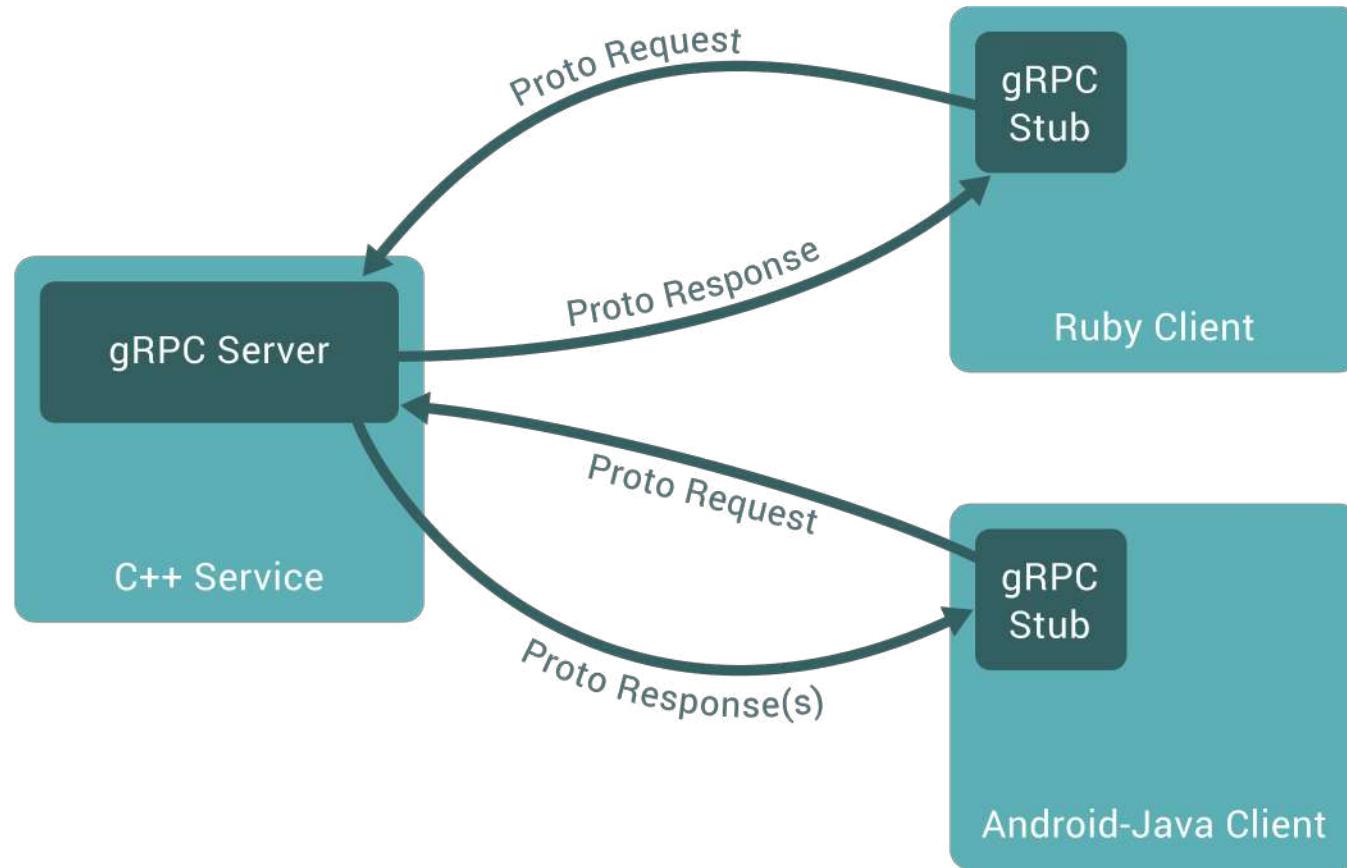


# gRPC

- gRPC is a modern open-source high performance RPC framework that can run in any environment.



# gRPC



# gRPC

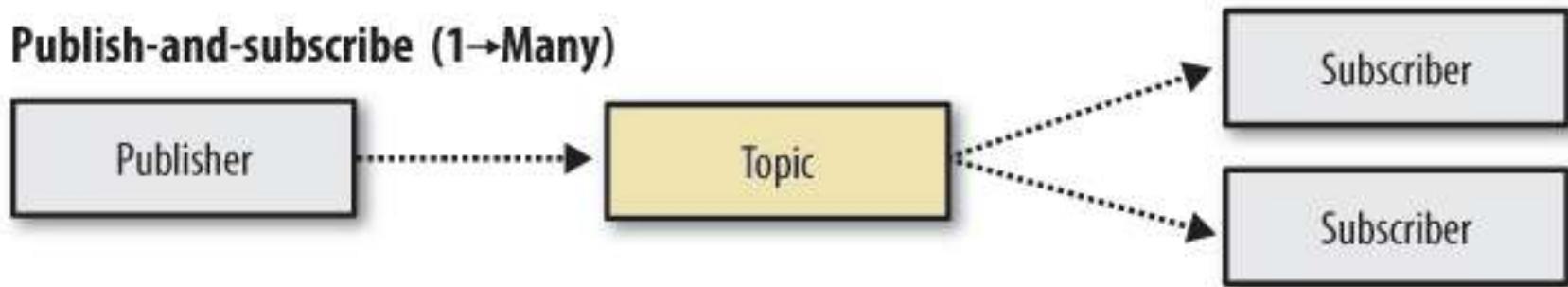
- gRPC uses **HTTP/2** which is, as you know, much faster than HTTP/1.1 used in REST by default.
- gRPC can use protocol buffers as both its Interface Definition Language (IDL) and as its underlying message interchange format.
- <https://grpc.io/docs/>

# Asynchronous Messaging Pattern

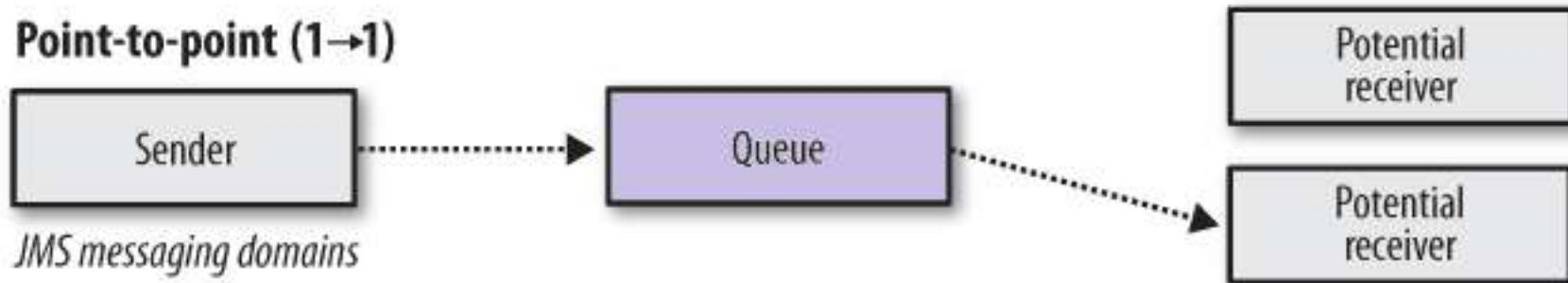
- Asynchronous messaging is a messaging scheme where message production by a producer is decoupled from its processing by a consumer.

# Queue vs. Pub/Sub

**Publish-and-subscribe (1→Many)**



**Point-to-point (1→1)**



# Subscriber Types

- An **ephemeral subscription**, where the subscription is only active as long the consumer is up and running.
  - Once the consumer shuts down, their subscription and yet-to-be processed messages are lost.
- A **durable subscription**, where the subscription is maintained as long as it's not explicitly deleted.
  - When the consumer shuts down, the messaging platform maintains the subscription, and message processing can be resumed later.

# Message Broker

- A message broker is an architectural pattern for message validation, transformation, and routing.
- It mediates communication among applications, minimizing the mutual awareness that applications should have of each other in order to be able to exchange messages, effectively implementing **decoupling**.



# Asynchronous Messaging Pattern

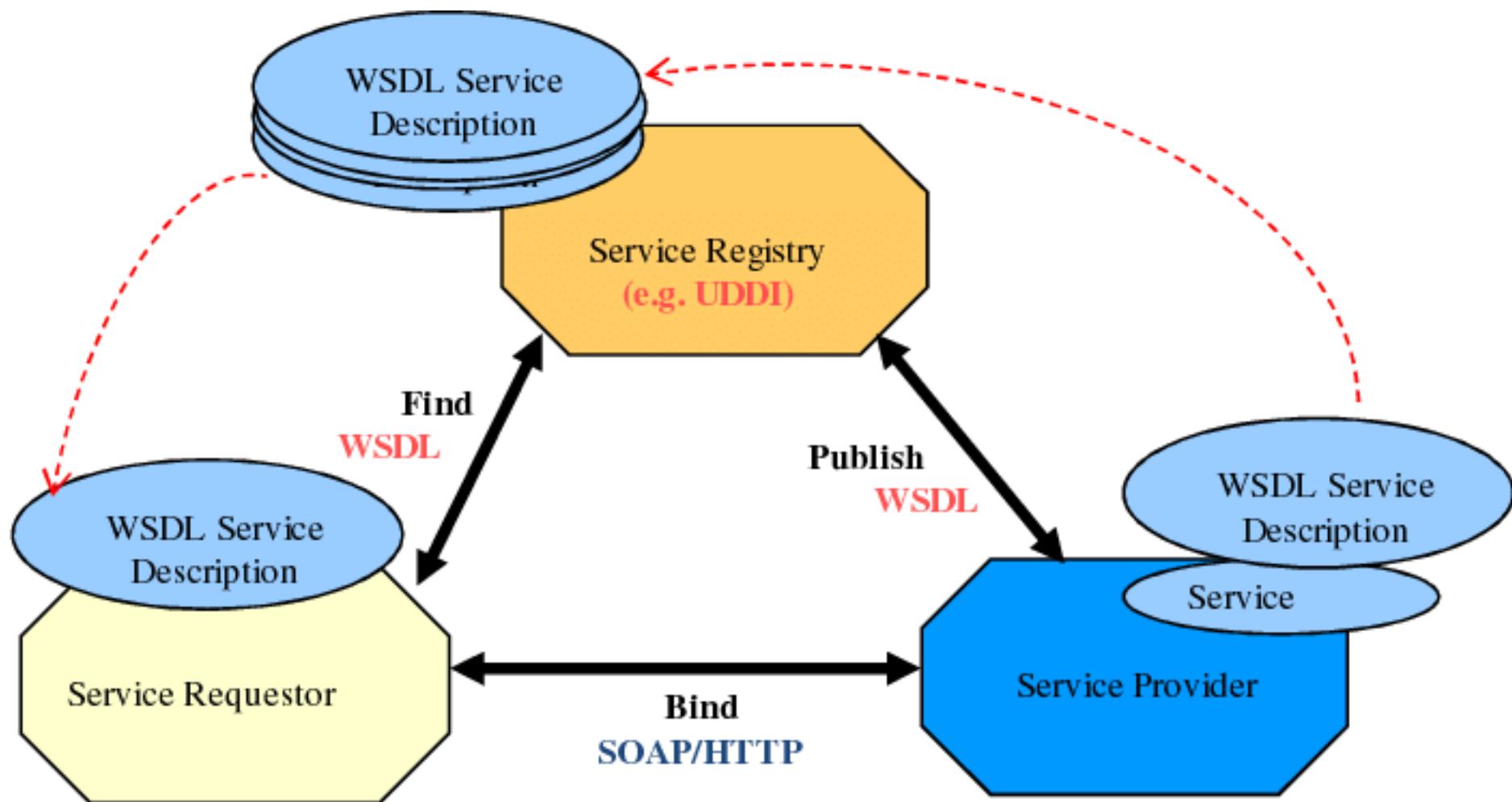
- <https://medium.com/better-programming/rabbitmq-vs-kafka-1ef22a041793>

# Web Service

- Web Service is a loosely coupling software component underlying the internet standard.
- Typically, web services use HTTP, FTP, SMTP for communication.

# Simple Object Access Protocol (SOAP)

- The Simple Object Access Protocol (SOAP) is a lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment.
- By combining SOAP-based requests and responses with a transport protocol, such as HTTP, the Internet becomes a medium for applications to publish database-backed Web services.



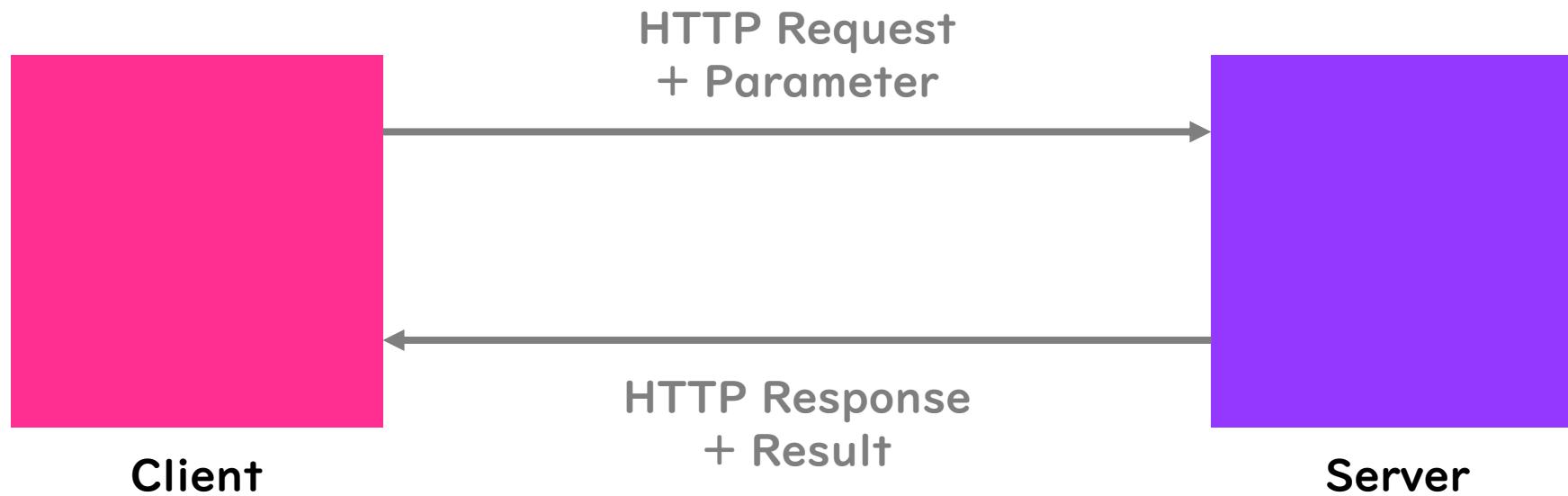
# Representational State Transfer (REST)

- Representational state transfer (REST) is a **software architectural style** that defines a set of constraints to be used for creating Web services.
- Web services that conform to the REST architectural style, called **RESTful Web services**, provide interoperability between computer systems on the internet.

# REST Constraints

- Client–server architecture
- Statelessness
- Cacheability
  - GET/PUT/DELETE
- Layered system
- Code on demand (optional)
- Uniform interface

# RESTful Web Service



# Application Programming Interface (API)

- An application programming interface (API) is a computing interface that defines interactions between multiple software intermediaries.

# Web Service vs. API

- API and Web service serve as a means of communication.
- The only difference is that a Web service **facilitates interaction between two machines over a network**.
- An API acts as an interface between **two different applications** so that they can communicate with each other.
- <https://medium.com/@programmerasi/difference-between-api-and-web-service-73c873573c9d>

# Web Service vs. API

- All web services are APIs, but not all APIs are web services.

# API Development using Go and MariaDB



# Go Package ‘http’

- http package is one of the default packages that Go prepared for you.
- You can just type the following code in your file.
  - import "net/http"
- <https://golang.org/pkg/net/http/>

# Creating a server

```
package main

import (
    "log"
    "net/http"
)

func main() {
    log.Fatal(http.ListenAndServe(":4000", nil))
}
```

main.go

# Creating a ping function

```
package controller

import (
    "fmt"
    "net/http"
)

func Ping(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Pong!")
}
```

controller/status.go

# Handling the ping function

```
func main() {
    http.HandleFunc("/ping", controller.Ping)

    log.Fatal(http.ListenAndServe(":4000", nil))
}
```

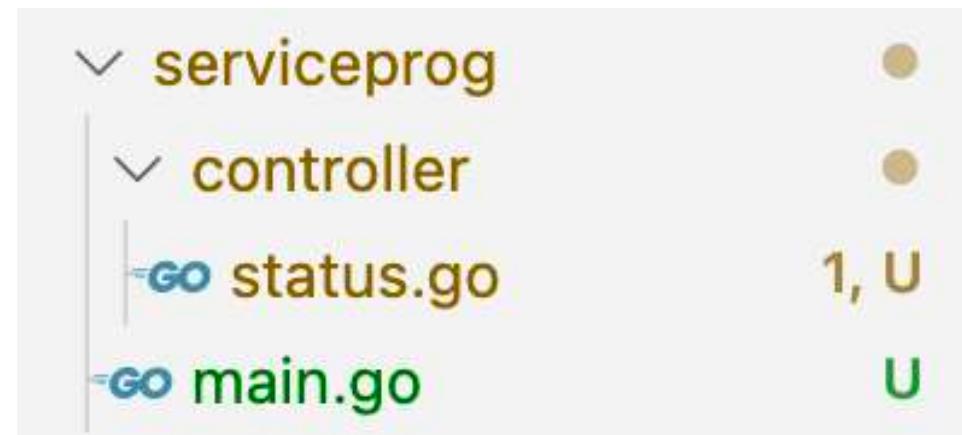
main.go

# Workshop 3 – Create a server program

- Using http package, create a server program and run it on port 4000.
- After that, make localhost:4000/hello looks like this!



# Current project structure



# Configuration File

- According to 12-factor app...
- Personally, I recommend **viper** package. (or maybe you can write your config file reader)
- go get [github.com/spf13/viper](https://github.com/spf13/viper)



# Separating the database configurations

```
dbconfig:  
  user: maria_dba  
  password: dbpassword  
  server: localhost  
  port: 3306  
  database: mydb
```

config/development.yaml

# Setting up a config reader

```
func Init() {  
    config = viper.New()  
    config.SetConfigType("yaml")  
    config.SetConfigName("dbconfig")  
    config.AddConfigPath("config/")  
  
    err := config.ReadInConfig()  
    if err != nil {  
        log.Fatal("error on parsing configuration file")  
    }  
}
```

config/config.go

# Setting up a config reader

```
import (
    "log"
    "github.com/spf13/viper"
)

var config *viper.Viper

...

func GetConfig() *viper.Viper {
    return config
}
```

config/config.go

# Setting up a config reader

```
package main  
  
...  
  
func init() {  
    config.Init()  
}
```

config/config.go

# Separating the database configurations

```
dbconfig:  
  user: maria_dba  
  password: dbpassword  
  server: localhost  
  port: 3306  
  database: mydb
```

config/development.yaml

# Connecting to database using the config file

```
func (d Database) NewConnection() (*sql.DB, error) {
    config := config.GetConfig()

    ...
    return db, err
}

func (d Database) CloseConnection(db *sql.DB) error {
    return db.Close()
}
```

database/database.go

# Connecting to database using the config file

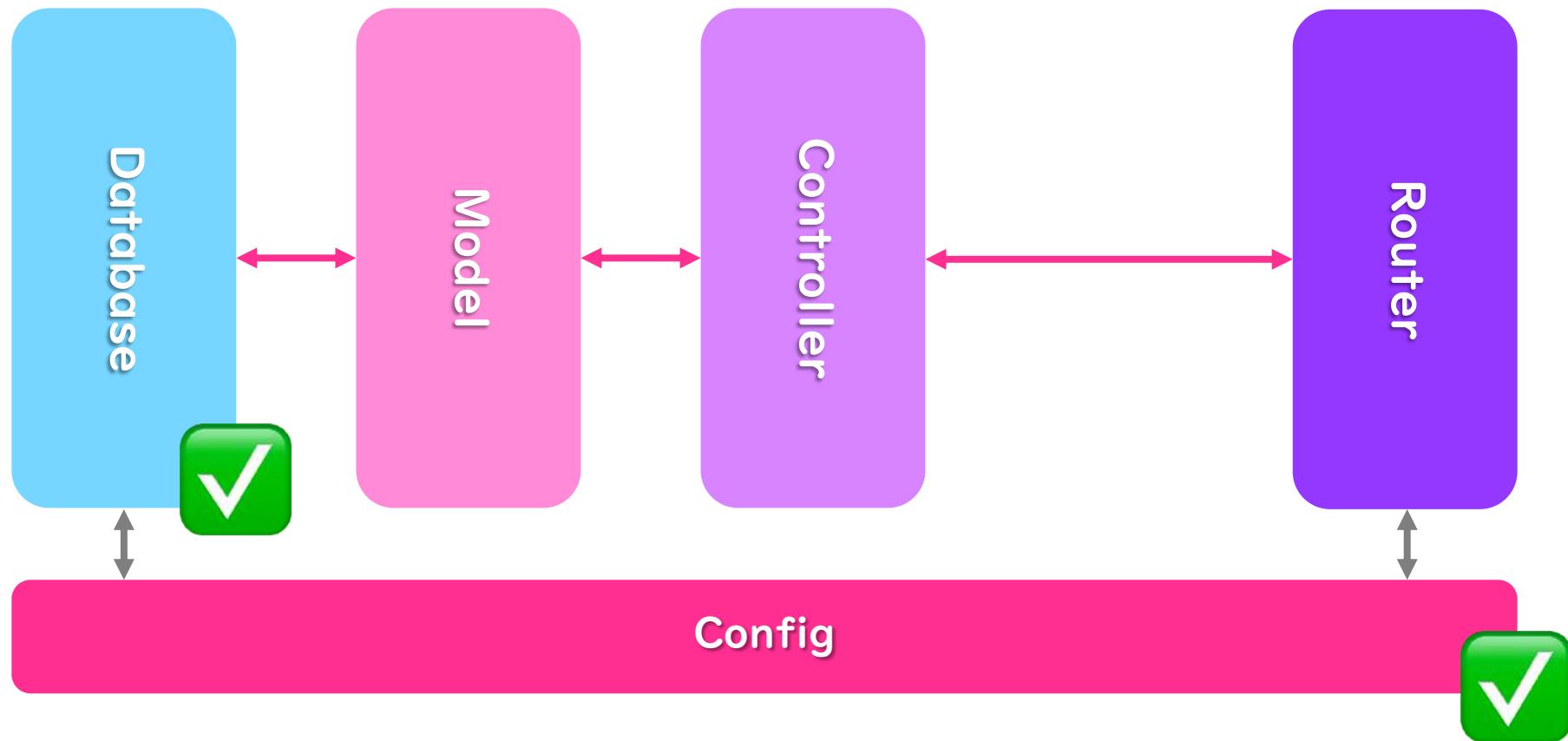
```
package database

...

type Database struct {
```

database/database.go

# Project Structure



# Creating the controller

```
package controller

...

type ProductController struct{}

func (pc ProductController) ReadAll(w
http.ResponseWriter, r *http.Request) {}
```

controller/product.go

# Creating the data model (form)

```
package form

type Product struct {
    ProductID      int      `json:product_id`
    ProductName    string   `json:product_name`
    ProductQuantity int     `json:product_quantity`
    ProductPrice   float64 `json:product_price`
}
```

form/product.go

# Creating the model

```
package model

...
type ProductModel struct{}

func (m ProductModel) Create(p form.Product) ([]Product,
error) {
    ...
}
```

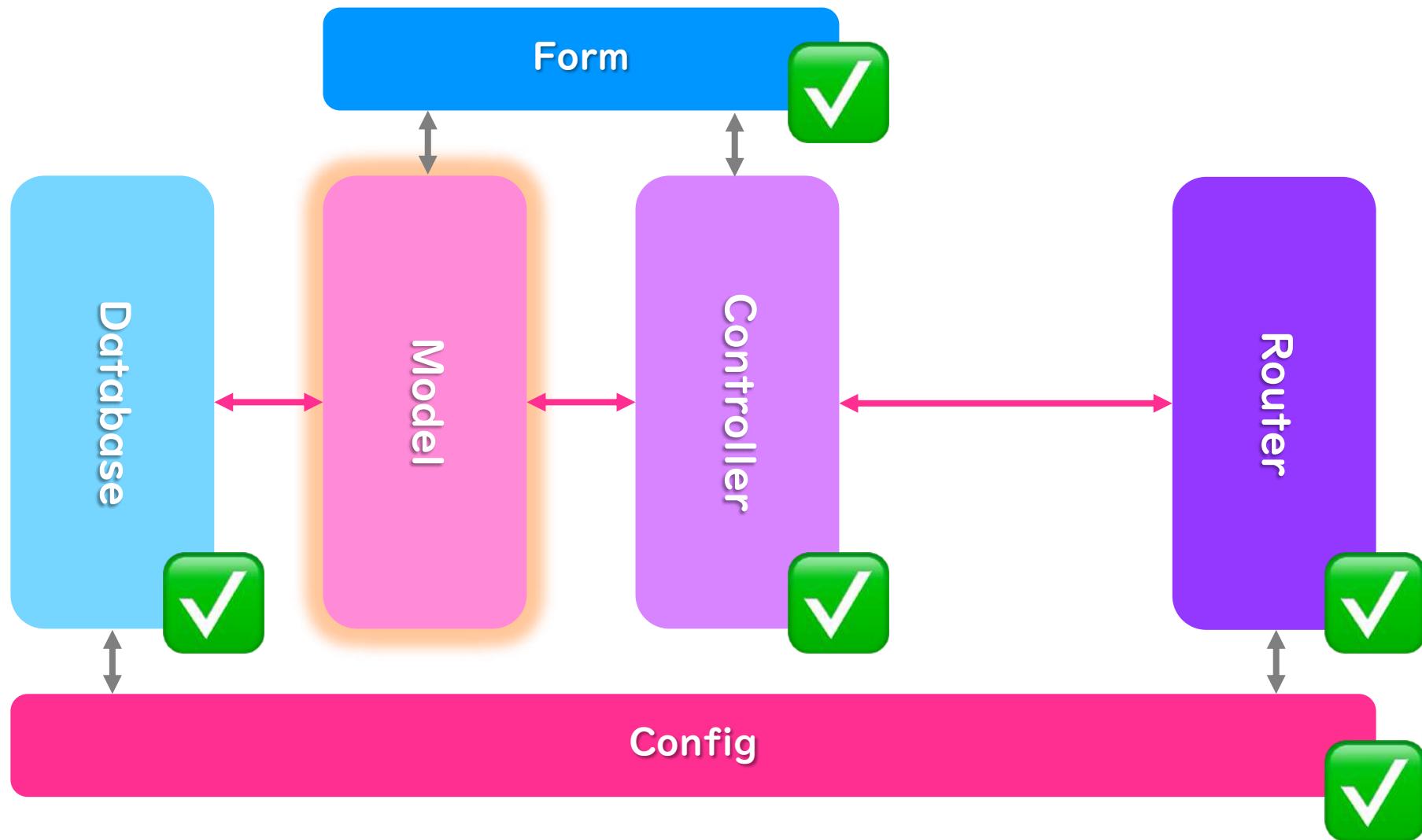
model/product.go

# Using the model

```
func (pc ProductController) ReadAll(w http.ResponseWriter,
r *http.Request) {
    m := model.ProductModel{}
    products, err := m.ReadAll()
    if err != nil {
        log.Println(err)
        w.WriteHeader(http.StatusInternalServerError)
        return
    }

    json.NewEncoder(w).Encode(products)
}
```

controller/product.go



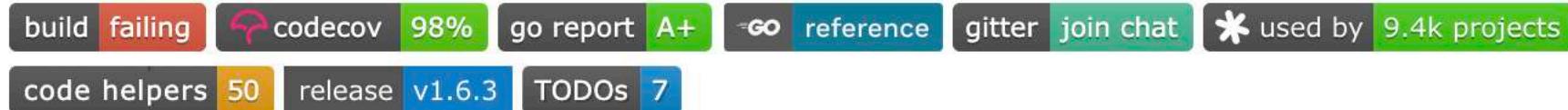
# Workshop 4 – Our first web service

- Let try to serve our first web service
- When you open the following URI, show all products in the database.
  - localhost:4000/products
- In other words, you should complete the ReadAll() function!

```
[ {"ProductID":1,"ProductName":"Apple","ProductQuantity":5,"ProductPrice":10} ]
```

# Go Package ‘gin-gonic’

## Gin Web Framework



Gin is a web framework written in Go (Golang). It features a martini-like API with performance that is up to 40 times faster thanks to [httprouter](#). If you need performance and good productivity, you will love Gin.



# Go Package ‘gin-gonic’

- go get -u github.com/gin-gonic/gin

# Gin-style Handler Function

```
func Ping(c *gin.Context) {  
    c.String(http.StatusOK, "Working!")  
}
```

controller/status.go

# Gin-style Handler Function

```
func (pc ProductController) ReadAll(c *gin.Context) {
    m := model.ProductModel{}
    products, err := m.ReadAll()
    if err != nil {
        log.Println(err)
        c.Status(http.StatusInternalServerError)
        return
    }
    c.JSON(http.StatusOK, products)
}
```

controller/product.go

# Update the configuration file

```
server:  
  port: 4000
```

config/development.yaml

# Refactor the server code

```
package server

import "../config"

func Init() {
    config := config.GetConfig()
    r := NewRouter()
    r.Run ":" + config.GetString("server.port")
}
```

server/server.go

# Refactor the server code

```
func NewRouter() *gin.Engine {  
    router := gin.New()  
  
    router.GET("/ping", controller.Ping)  
  
    productCrtl := controller.ProductController{}  
    router.GET("/products", productCrtl.ReadAll)  
  
    return router  
}
```

server/router.go

# Update the router

```
func (pc ProductController) ReadAll(c *gin.Context) {  
    m := model.ProductModel{}  
    products, err := m.ReadAll()  
    if err != nil {  
        log.Println(err)  
        c.Status(http.StatusInternalServerError)  
        return  
    }  
  
    c.JSON(http.StatusOK, products)  
}
```

controller/product.go

# Update the main file

```
package main

import (
    "./config"
    "./server"
)

func main() {
    config.Init()
    server.Init()
}
```

main.go

```
[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode  
in production.
```

- using env: `export GIN_MODE=release`
- using code: `gin.SetMode(gin.ReleaseMode)`

```
[GIN-debug] GET    /ping                      --> _/Users/pnx/Documents/  
Code/Teaching/CSINF-Backend/ginprog/controller.Ping (1 handlers)
```

```
[GIN-debug] GET    /products                 --> _/Users/pnx/Documents/  
Code/Teaching/CSINF-Backend/ginprog/controller.ProductController.ReadAll-  
fm (1 handlers)
```

```
[GIN-debug] Listening and serving HTTP on :4000
```

*Let's complete  
our services*

# Querying request parameters

```
func (pc ProductController) Read(c *gin.Context) {
    idParam, err := strconv.ParseInt(c.Query("id"), 10,
64)
    if err != nil ...

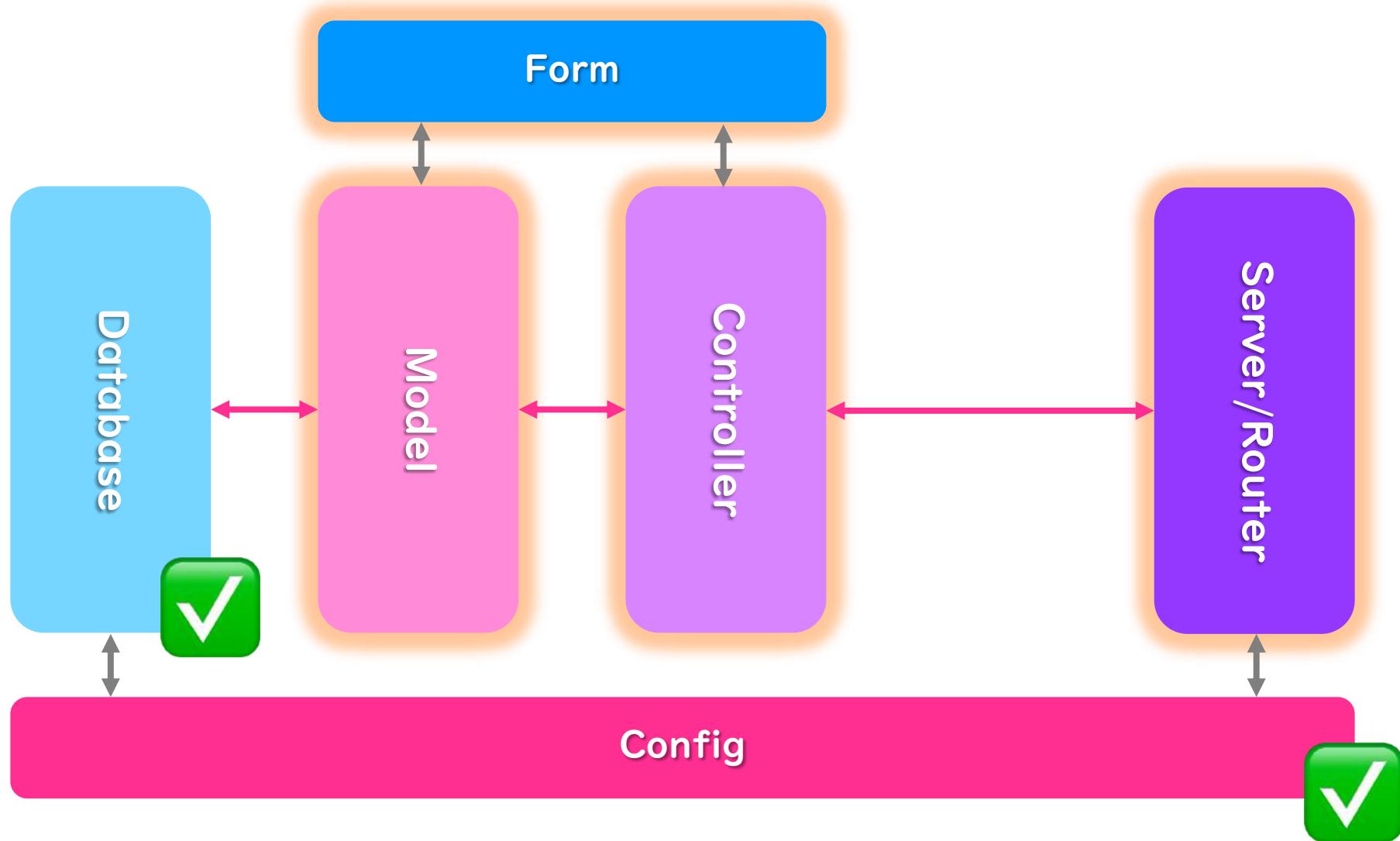
    product, err := pc.Model().Read(idParam)
    if err != nil ...

    c.JSON(http.StatusOK, product)
}
```

controller/product.go

# Querying request parameters

```
☆ localhost:4000/product?id=1
{"ProductID":1,"ProductName":"Apple","ProductQuantity":5,"ProductPrice":10}
```



# Workshop 5 – Upgrade the services

- Let's upgrade our net/http-based controller
  - Change it into gin-based controller
- Add Create/Update/Delete controllers/models

*You might encounter  
this problem...*

*(on production)*

One Origin Policy disallows reading the remote resource at <https://developer.mozilla.org>

Access to fetch at '[https://joke-api-strict-cors.appspot.com/r/localhost/random\\_joke](https://joke-api-strict-cors.appspot.com/r/localhost/random_joke)' from origin '<http://localhost:3000>' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

top Elements Console Sources Network Performance Memory Application Security Audits 1 Filter Default levels ▾  
led to load <https://stg-api.bazarservice.com/data/submitreview.json>: Response to preflight request at index isn't pass access control check. No 'Access-Control-Allow-Origin' header is present on the requested resource in <https://example.com:8081>, so therefore not allowed access.  
UR finished loading: OPTIONS "https://stg-api.bazarservice.com/data/submitreview.json". [Index]

1st response = await fetch("https://ionicframework.com");  
next to fetch at "<https://ionicframework.com/>" from origin "<http://localhost:8100>"  
locked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.  
Your response serves your needs, set the request's mode to 'no-cors' to fetch the resource.  
Caught (in process): TypeError: Failed to fetch  
Cross-Origin Read Blocking (CO-RB) blocked cross-origin response <https://ionicframework.com/>.  
See <https://www.chromestatus.com/feature/5629789524872758> for more  
caught (in promise) TypeError: Failed to fetch  
TypeError: Failed to fetch

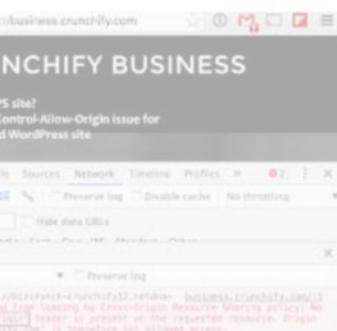
MDN

Top Five CORS Issues You Don't Want To ...  
[wirescript.now.sh](https://wirescript.now.sh)

CORS, What is it exactly? Is it really ...  
[medium.com](https://medium.com)

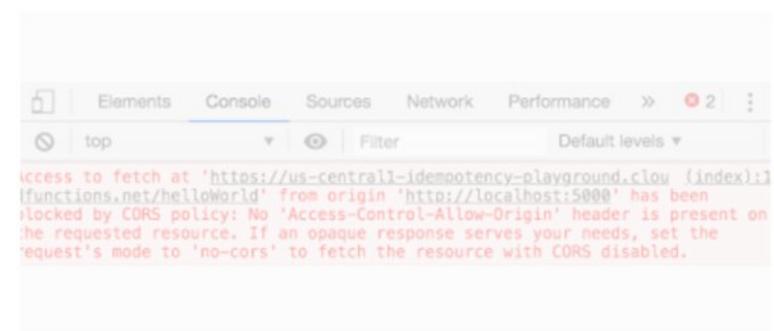
CORS Errors in Ionic Apps  
[fdezromero.com](https://fdezromero.com)

- ✖ Access to fetch at '[https://joke-api-strict-cors.appspot.com/r/localhost/:1/random\\_joke](https://joke-api-strict-cors.appspot.com/r/localhost/:1/random_joke)' from origin '<http://localhost:3000>' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.

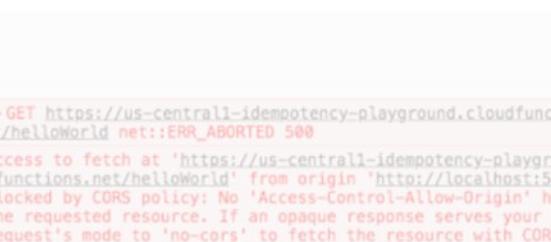


```
> const xhr = new XMLHttpRequest()
xhr.onreadystatechange = () => {
  if (xhr.readyState === 4) {
    xhr.status === 200 ? console.log(xhr.responseText) : console.error('error')
  }
}
xhr.open('GET', 'https://google.com')
xhr.send()
< undefined
➊ Failed to load https://google.com/: Redirect from 'https://google.com/' to 'bit [index]11851/www.google.it/4t8_rhcrdr-88e-3y6vbmM9CxswoMh' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost:1337' is therefore not allowed access.
➋ error
> XMLHttpRequest finished loading: GET "https://google.com/".
9M38718
```

The Web Platform Course  
[webplatformcourse.com](https://webplatformcourse.com)



CORS Error: Firebase Functions Walk ...  
[haha.world](https://haha.world)

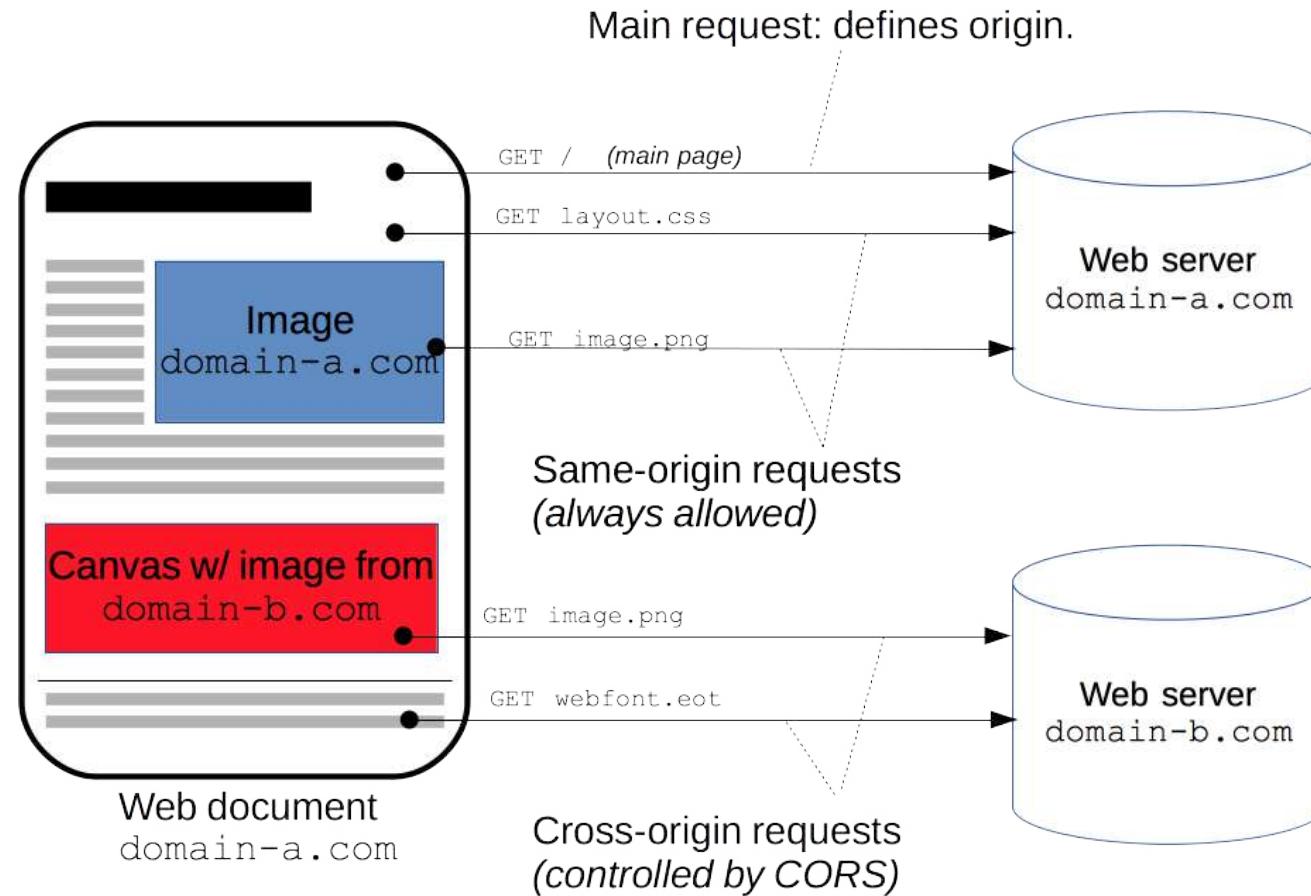


CORS Error: Firebase Functions Walk ...  
[haha.world](https://haha.world)

# Cross-Origin Resource Sharing

- Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any other origins (domain, protocol, or port) than its own from which a browser should permit loading of resources.
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

# Cross-Origin Resource Sharing



# Cross-Origin Resource Sharing

## How to fix the CORS “error”?

You have to understand that the CORS behavior is **not an error** — it's a mechanism that's working as expected in order to protect your users, you, or the site you're calling.

<https://medium.com/@baphemot/understanding-cors-18ad6b478e2b>

# Cross-Origin Resource Sharing

- CORS ไม่ใช่ปัญหา มันคือสิ่งที่ทุกคนต้องทำเพื่อความปลอดภัยของระบบ

# Setting up CORS middleware

```
func NewRouter() ... {  
    ...  
    router.Use(cors.Default())  
    ...  
}
```

Download and install it:

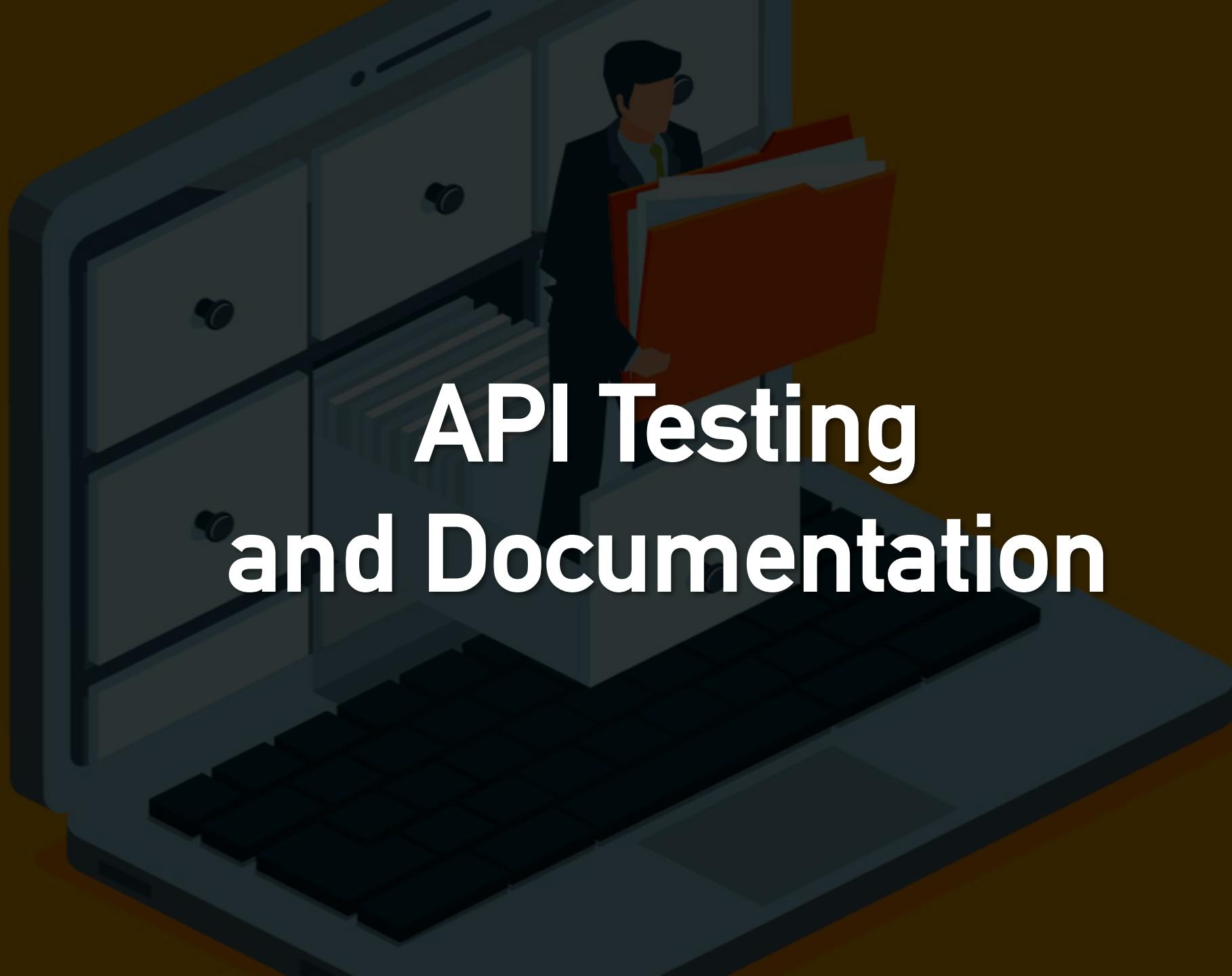
```
$ go get github.com/gin-contrib/cors
```

Import it in your code:

```
import "github.com/gin-contrib/cors"
```

*Don't imprudently config CORS!  
Please be careful...*

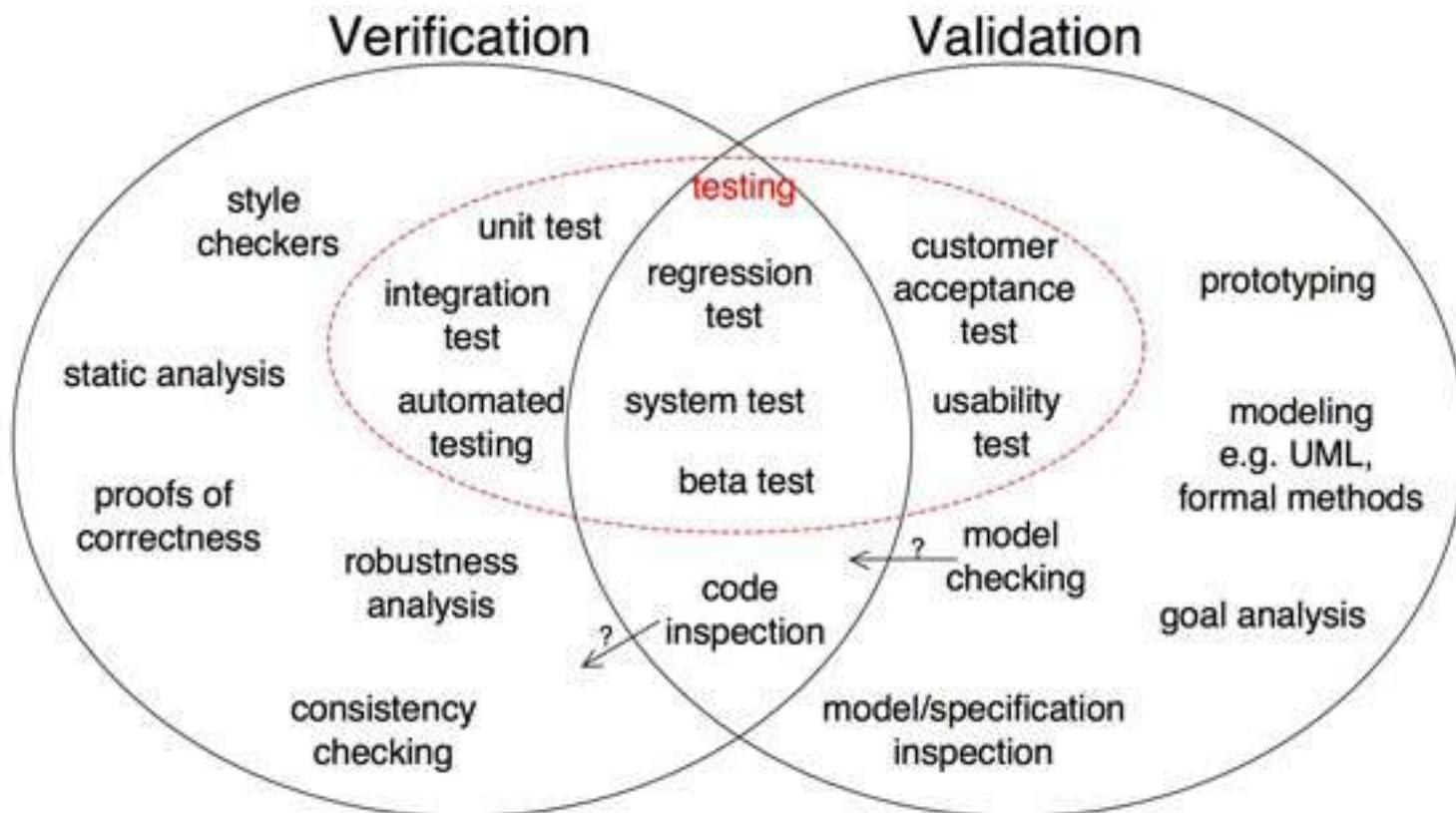
# API Testing and Documentation



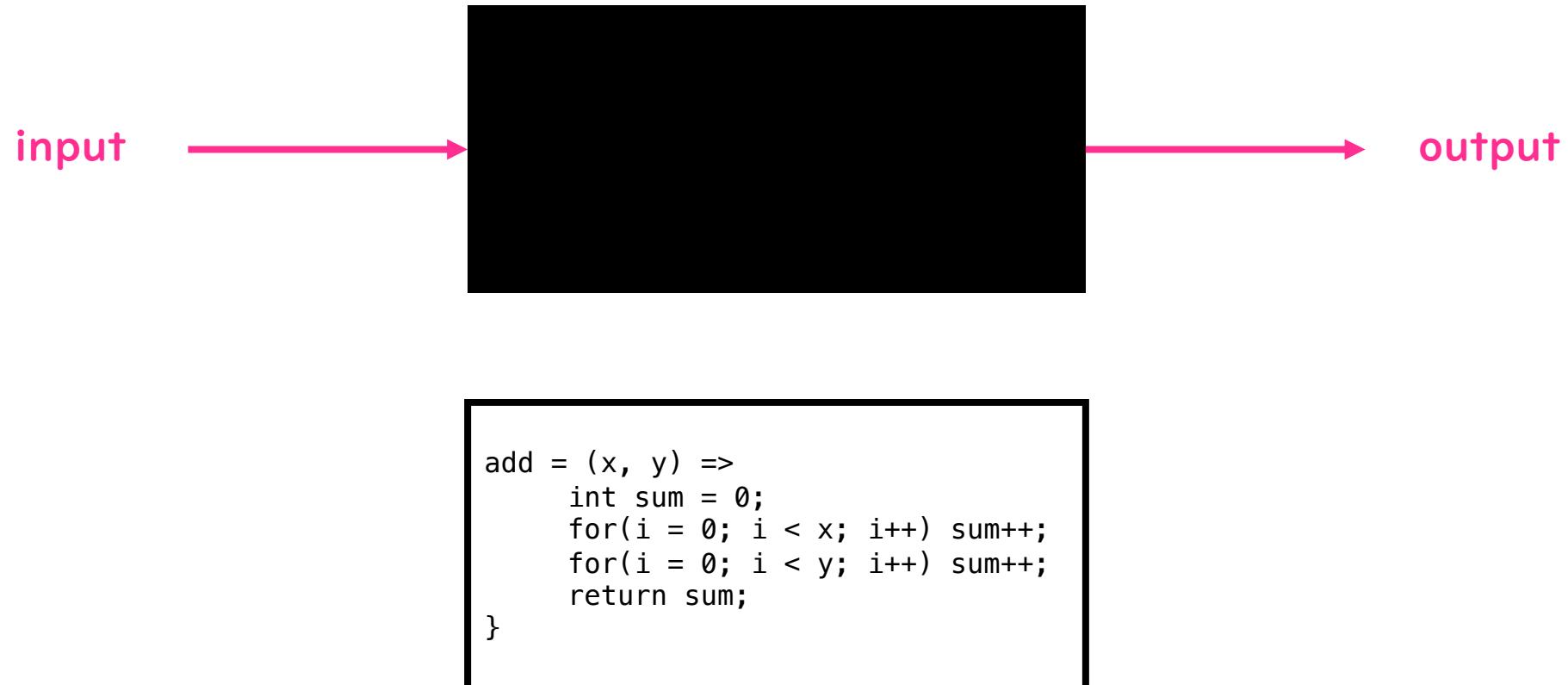
# Verification and Validation

- Verification simply means
  - “Are we implementing the system right?”
  - กำกับระบบถูกหรือเปล่า
  - $\text{add}(5, 5) = 10$  ถูกหรือไม่
- Validation also simply means
  - “Are we implementing the right system?”
  - กำกับระบบให้ถูกตาม Requirements
  - $\text{add}(x, y)$  คือการบวก

# Verification and Validation



# Black-box vs. White-box Testing



# Unit Testing

- Unit tests are typically automated tests written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.

# Unit Testing in Go

- In Go, you save unit tests inside separate files with a filename ending with \_test.go.
- Go provides go test command out of the box which executes these files and runs tests.

# Unit Testing in Go

The screenshot shows a Go development environment in VS Code. The left sidebar displays the file structure:

- OPEN EDITORS:
  - hello.go
  - hello\_test.go
- GROUP 1:
  - hello.go src/greeting
- GROUP 2:
  - hello\_test.go src/g...
- MAIN:
  - bin
  - pkg
  - src
    - github.com
    - greeting
      - hello\_test.go
      - hello.go
      - main.go

The main editor shows two files:

- hello.go** (main package):

```
1 package main
2
3 import "fmt"
4
5 // return hello greeting
6 func hello(user string) string {
7     if( len(user) == 0 ) {
8         return "Hello Dude!"
9     } else {
10        return fmt.Sprintf("Hello %v!", user)
11    }
12 }
```
- hello\_test.go** (testing package):

```
1 import "testing"
2
3 // test hello function
4 func TestHello( t *testing.T ) {
5
6     // test for empty argument
7     emptyResult := hello("") // should return "Hello Dude!"
8
9     if emptyResult != "Hello Dude!" {
10        t.Errorf("hello(\"\") failed, expected %v, got %v", "Hello Dude!", emptyResult)
11    }
12
13    // test for valid argument
14    result := hello("Mike") // should return "Hello Mike!"
15
16    if result != "Hello Mike!" {
17        t.Errorf("hello(\"Mike\") failed, expected %v, got %v", "Hello Dude!", result)
18    }
19 }
```

The bottom terminal shows the test results:

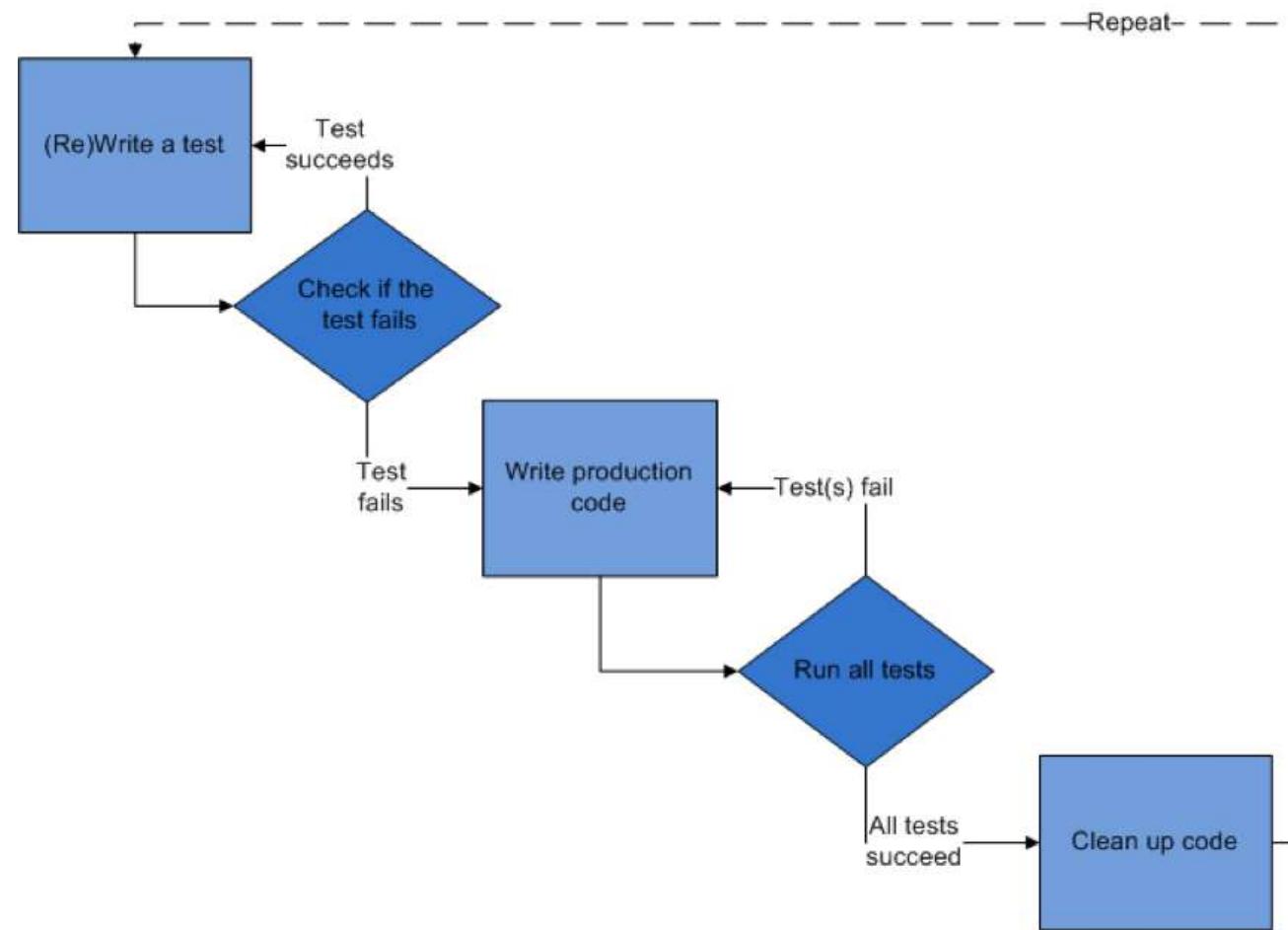
```
→ greeting go test
PASS
ok   greeting      0.010s
→ greeting
```

At the bottom, the status bar shows: JSONPath: Ln 21, Col 2 Tab Size: 4 UTF-8 LF Go 15.59MB 496 B Found 2 variables 😊 1

# Unit Testing in Go

- <https://medium.com/rungo/unit-testing-made-easy-in-go-25077669318>

# Test-Driven Development (TDD)



# Client URL (cURL)

- curl is a tool to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, MQTT, POP3, POP3S, RTMP, RTMPS, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET and TFTP).
- The command is designed to work without user interaction.

# Postman



POSTMAN

Product ▾

Use Cases ▾

Pricing

Enterprise

Explore

Learning Center

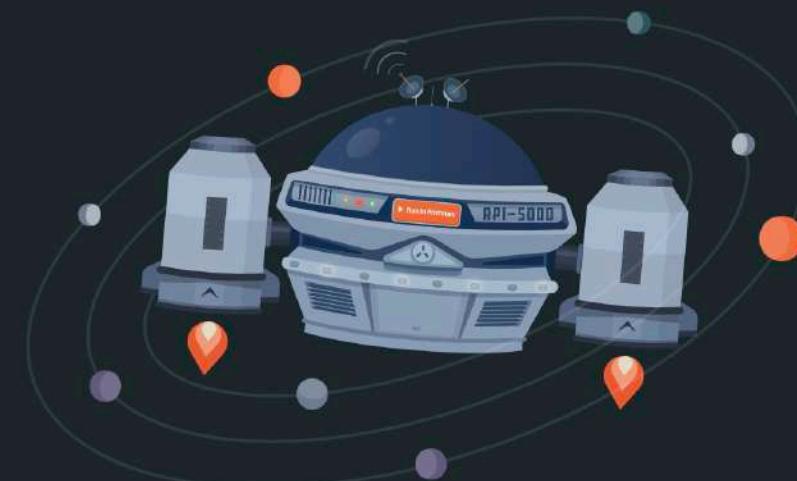
Launch Postman

## The Collaboration Platform for API Development

Simplify each step of building an API and streamline collaboration so you can create better APIs—faster.

Launch Postman

Learn More



# Postman Documentation

1. Select a collection to document

2. Write documentation

3. Next steps

X

Create a new collection

Select an existing collection

Create a new collection by adding individual requests. Use the "Description" column to document what each endpoint URL does. To add a request body or an example response body, click the (...) icon.

Method	Request URL	Description	...
GET	localhost:4000/products	Query all products	
GET	localhost:4000/product	Query the specific product	X
GET	URL	Description	

## GET localhost:4000/products

Comments 0

localhost:4000/products

Query all products

Example Request

Default

```
curl --location --request GET 'localhost:4000/products' \
--data-raw ''
```

Example Response

Body Headers (0)

No response body

This request doesn't return a response body.

## GET localhost:4000/product

Comments 0

localhost:4000/product?id

Query the specific product

Params

id

Example Request

Default

```
curl --location --request GET 'localhost:4000/product' \
--data-raw ''
```

Example Response

Body Headers (0)

No response body

This request doesn't return a response body.

# Workshop 6 – Documenting your APIs

- Document your API using Postman

# Session Management



# How do we create an application with log-in?

- Your application need to know who is active or not.
  - Maintaining each user state is quite challenging.

# Authentication and Authorization

- Authentication is the process of verifying who a user is.
- Authorization is the process of verifying what they have access to.

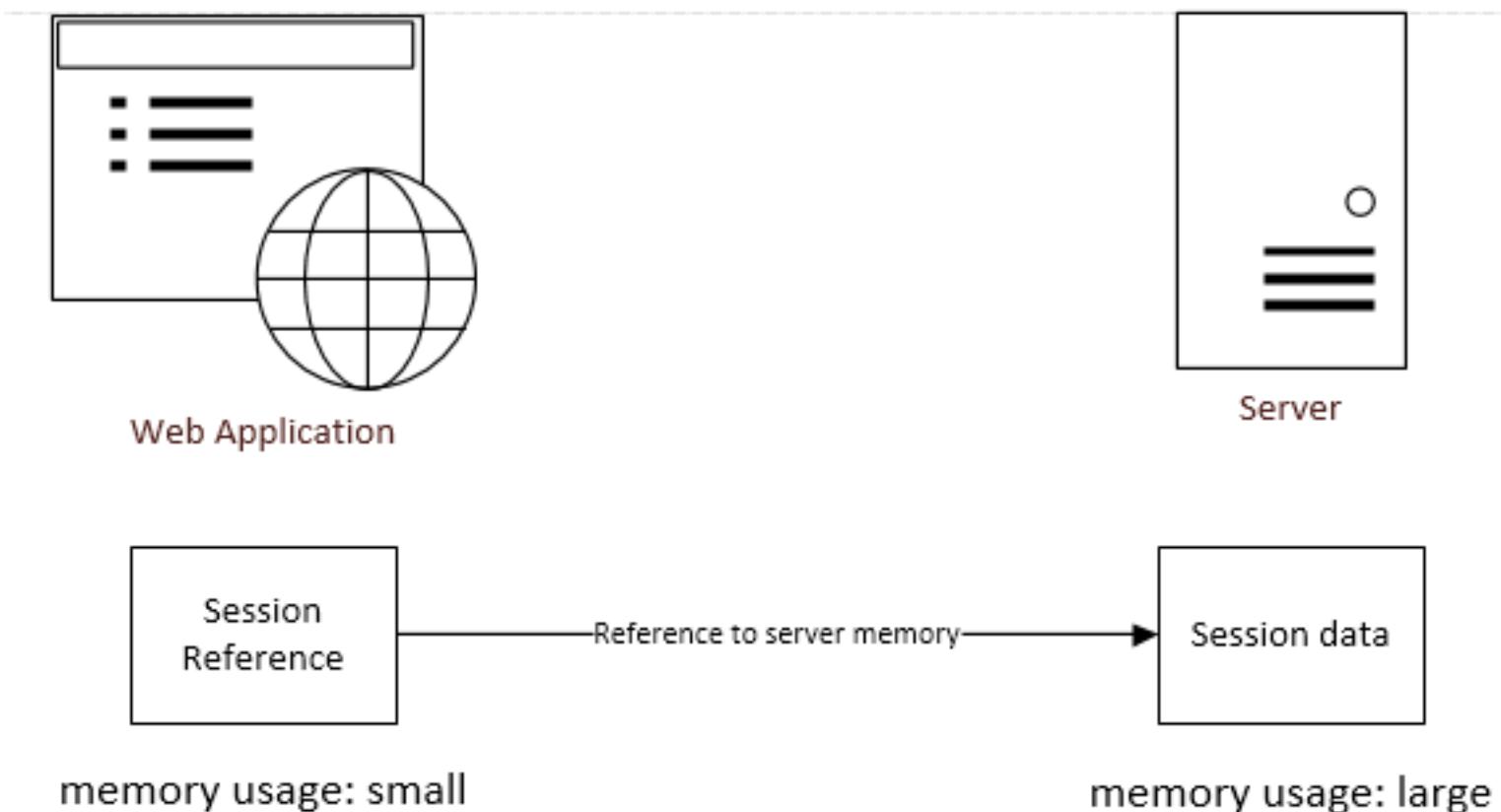
# Session

- A session is a temporary and interactive information interchange between two or more communicating devices, or between a computer and user (see login session).
- A session is established at a certain point in time, and then ‘torn down’ - brought to an end - at some later point.

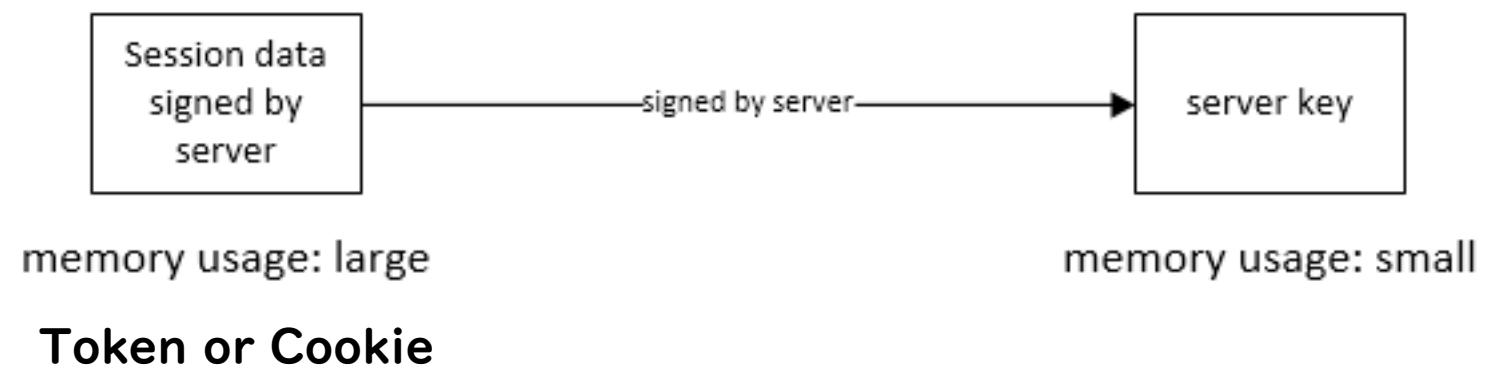
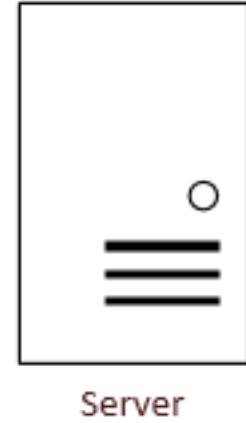
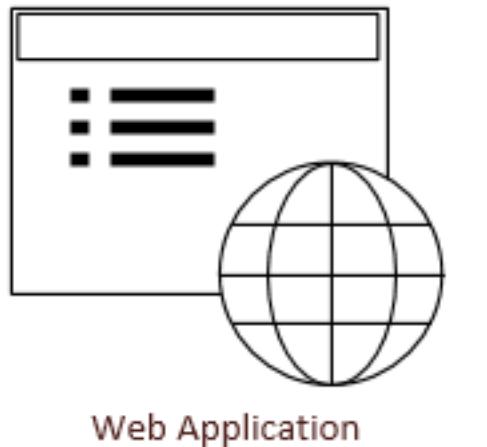
# Session

- There are two approaches to maintain users' sessions.
  - Stateful session
  - Stateless session
- <https://medium.com/@kennch/stateful-and-stateless-authentication-10aa3e3d4986>

# Stateful Session



# Stateless Session



# Session

- <https://www.openidentityplatform.org/blog/stateless-vs-stateful-authentication>

	<b>Stateful</b>	<b>Stateless</b>
Session information could be stolen	<input checked="" type="checkbox"/> It is impossible to steal session information from the session identifier because it is just an identifier associated with the session	<input type="checkbox"/> Session identifier contains all authentication information and it is possible to steal sensitive information, it is not encrypted.

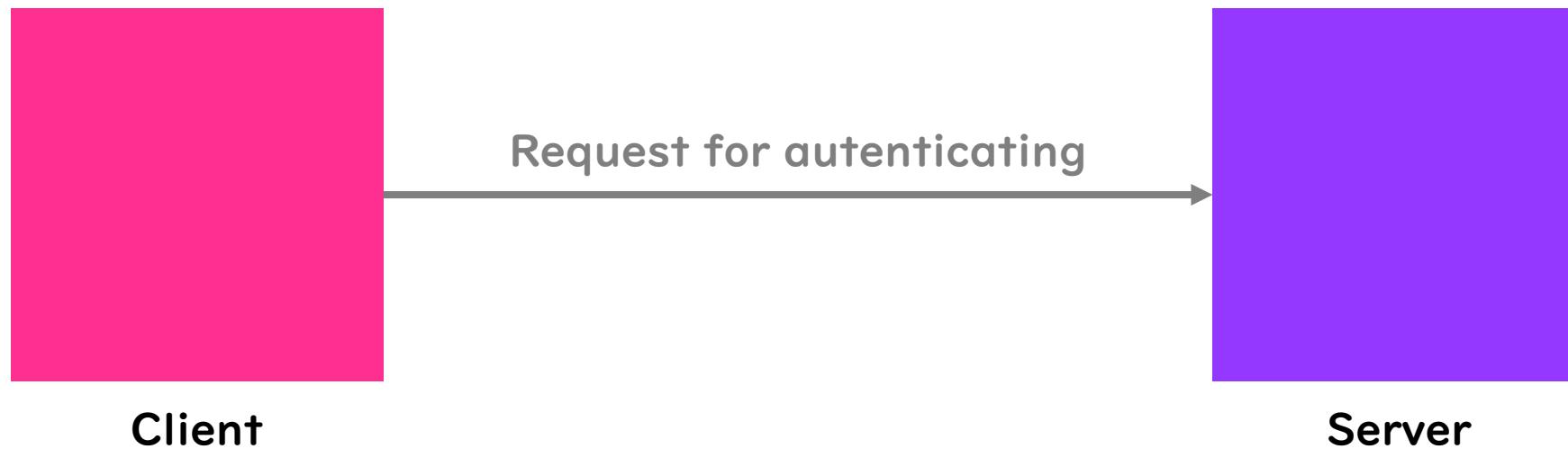
# Cookie-Based Authentication

- Cookie-based authentication is stateful.
  - This means that an authentication record or session must be kept both server and client-side.

# Cookie

- An HTTP cookie (web cookie, browser cookie) is a small piece of data that a server sends to the user's web browser.
- Cookies are mainly used for three purposes:
  - Session management
  - Personalization
  - Tracking
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>

# Cookie-Based Authentication



# Cookie-Based Authentication



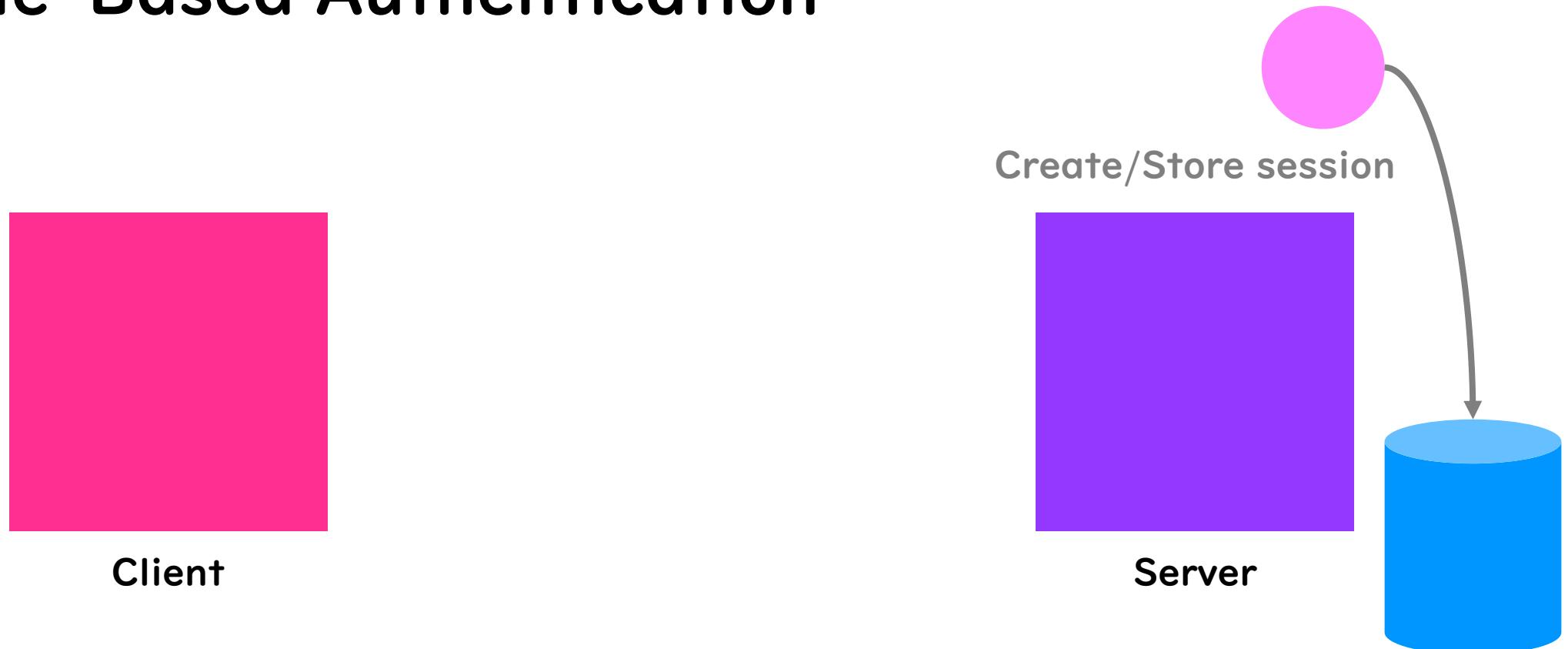
Client

Check client's sent credentials



Server

# Cookie-Based Authentication



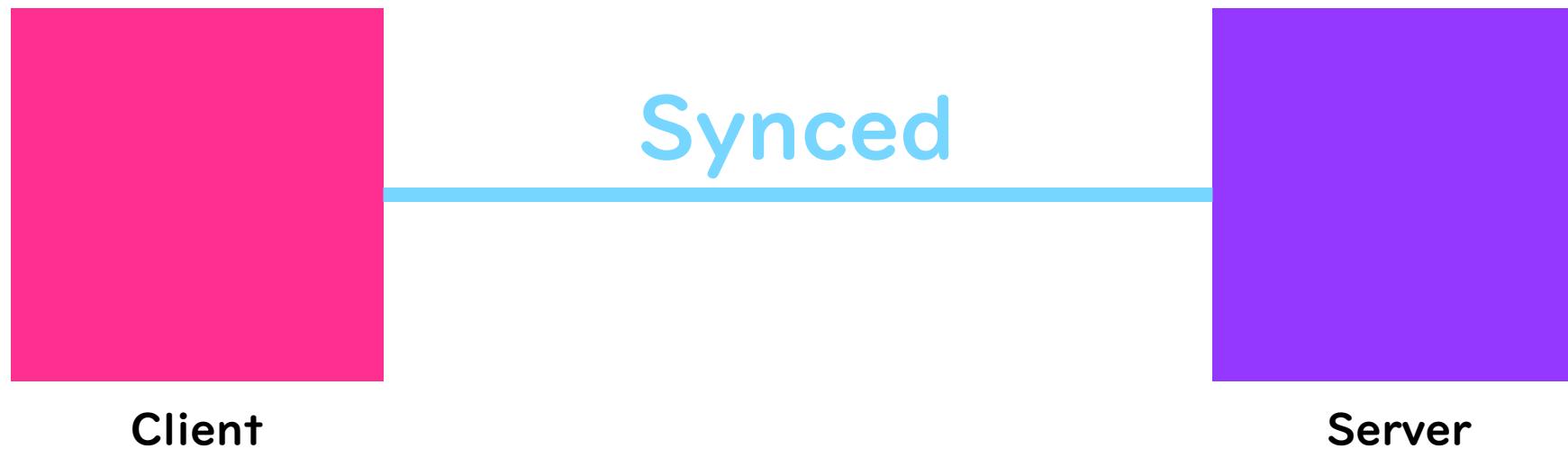
# Cookie-Based Authentication



# Cookie-Based Authentication



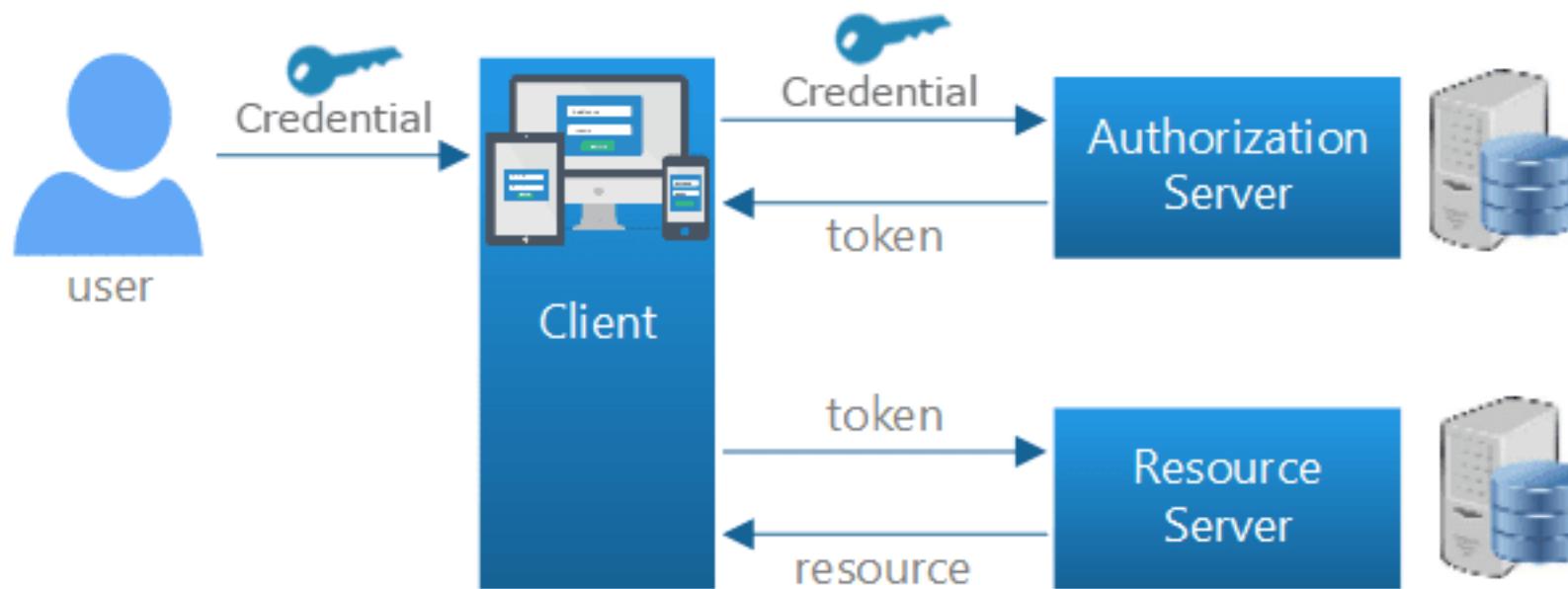
# Cookie-Based Authentication



# Token-Based Authentication

- Token-based authentication is a protocol which allows users to verify their identity, and in return receive a unique access token.
- The vulnerability of token-based authentication is the **expiry time!**
  - Tradeoff between user experience and security

# Token-Based Authentication



<http://www.dotnetawesome.com/2016/09/token-based-authentication-in-webapi.html>

# JSON Web Token (JWT)

- JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.

# JSON Web Token (JWT)

## Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SfIKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

## Decoded

EDIT THE PAYLOAD AND SECRET

### HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

### PAYOUT: DATA

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

### VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

# JSON Web Token (JWT)

- JWT can store some data.
  - However, these data is not invisible! (it can be decoded easily)
- Don't put any confidential data (esp. PII) in the JWT!

# Access and Refresh Token

- Access tokens carry the necessary information to access a resource directly.
- Refresh tokens carry the information necessary to get a new access token.
  - You don't need to login every time.

# Web Application Security

# Vulnerability

- In computer security, a vulnerability is a weakness which can be exploited by a threat actor, such as an attacker, to cross privilege boundaries within a computer system.

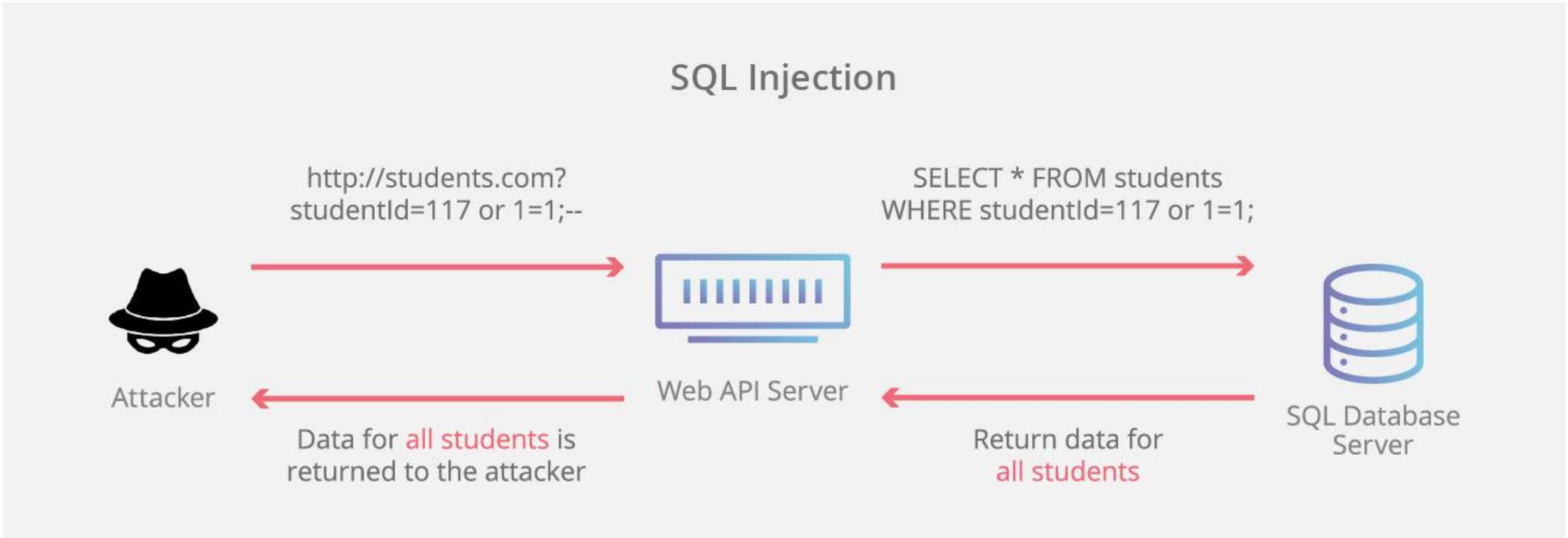
# OWASP Top 10

- OWASP Top 10 is the list of the 10 most common application vulnerabilities.
  - It also shows their risks, impacts, and countermeasures.
- <https://owasp.org/www-project-top-ten/>

# OWASP Top 10

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↗	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	↳	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↗	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	↳	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

# Injection



# Injection

## Is the Application Vulnerable?

An application is vulnerable to attack when:

- \* User-supplied data is not validated, filtered, or sanitized by the application.
- \* Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- \* Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- \* Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.

Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections, closely followed by thorough automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs. Organizations can include static source ([SAST](#)) and dynamic application test ([DAST](#)) tools into the CI/CD pipeline to identify newly introduced injection flaws prior to production deployment.

## How to Prevent

Preventing injection requires keeping data separate from commands and queries.

- \* The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).

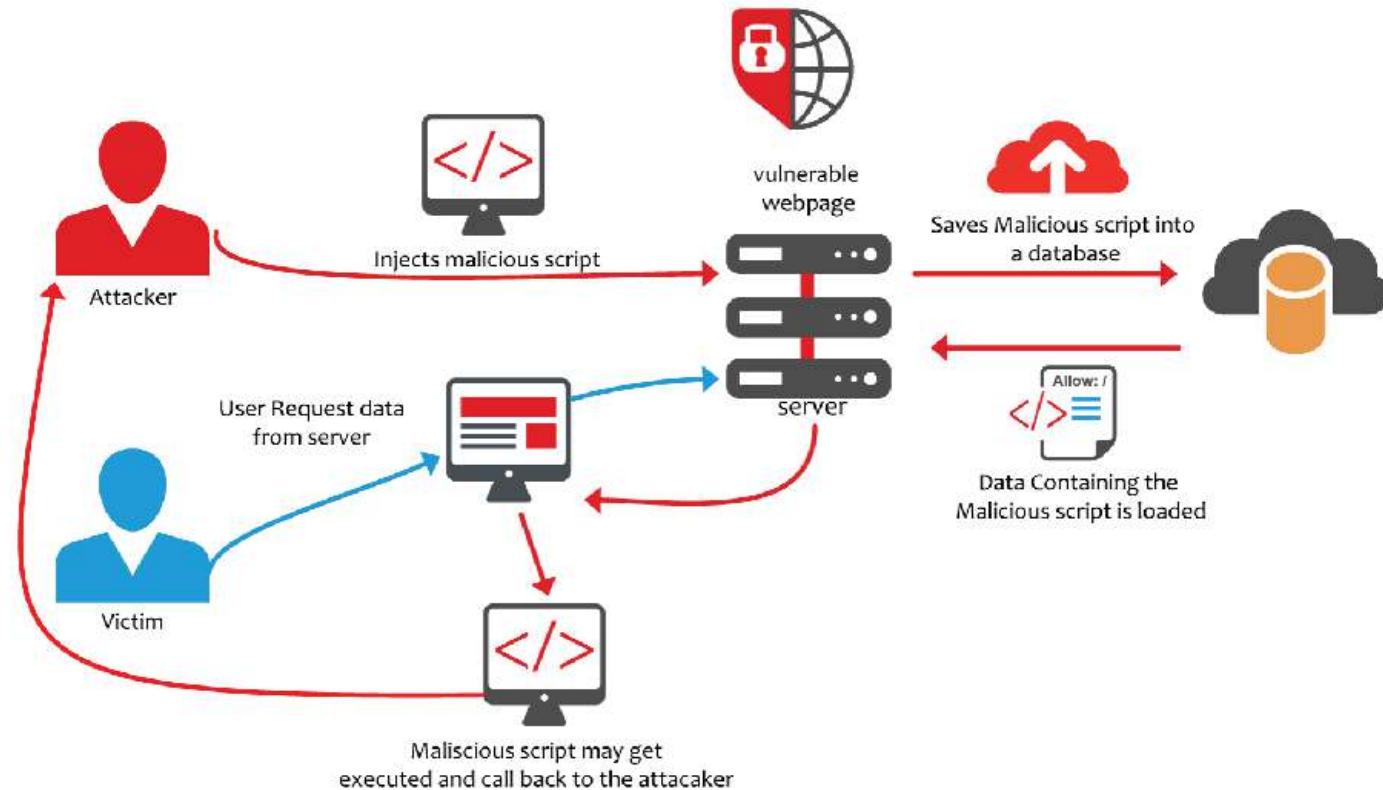
**Note:** Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().

- \* Use positive or “whitelist” server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
- \* For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.

**Note:** SQL structure such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.

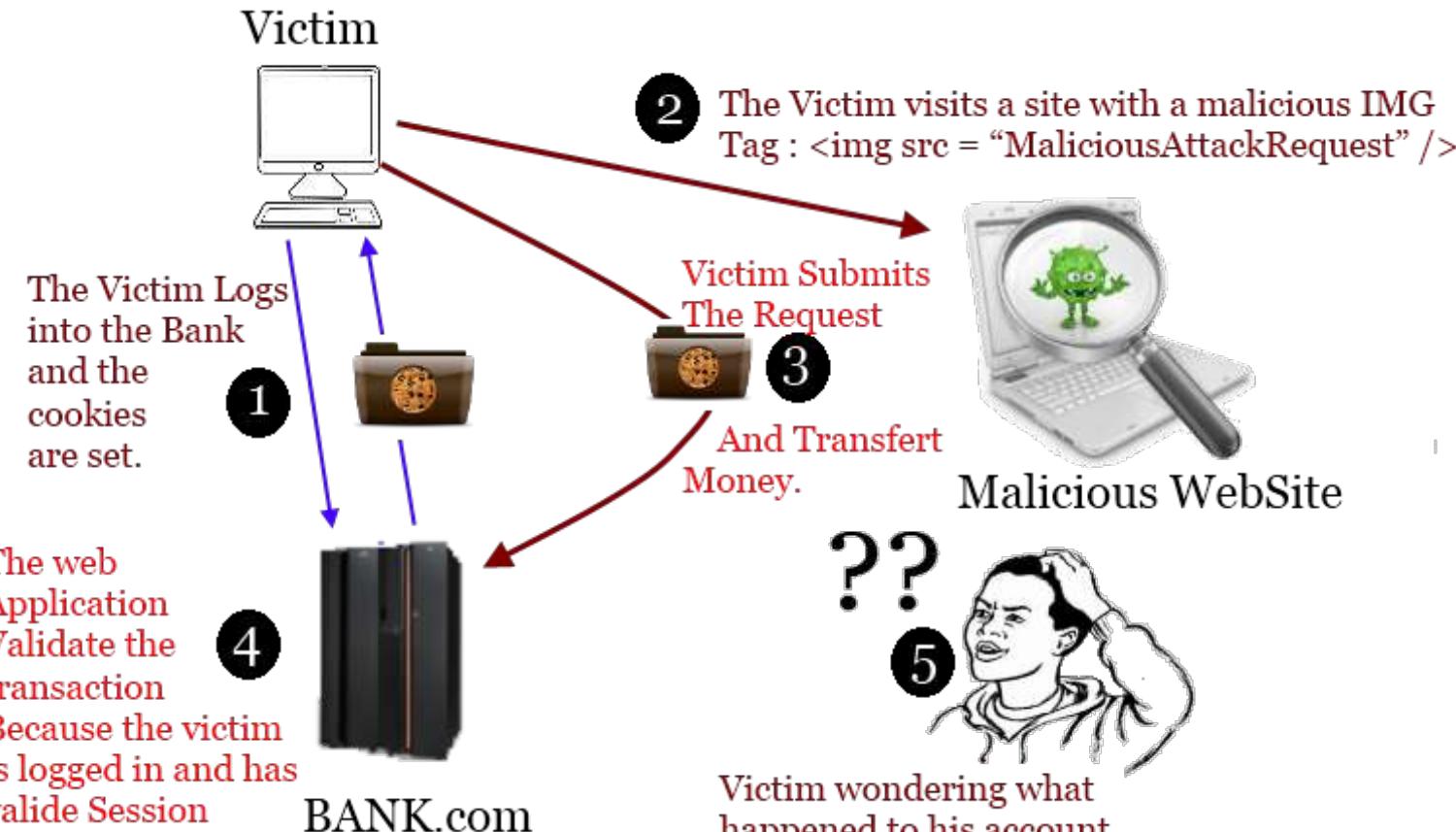
- \* Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

# Cross Site Scripting (XSS)



<https://blogs.sap.com/2015/12/17/xss-cross-site-scripting-overview-with-contexts/>

# Cross Site Request Forgery (CSRF)



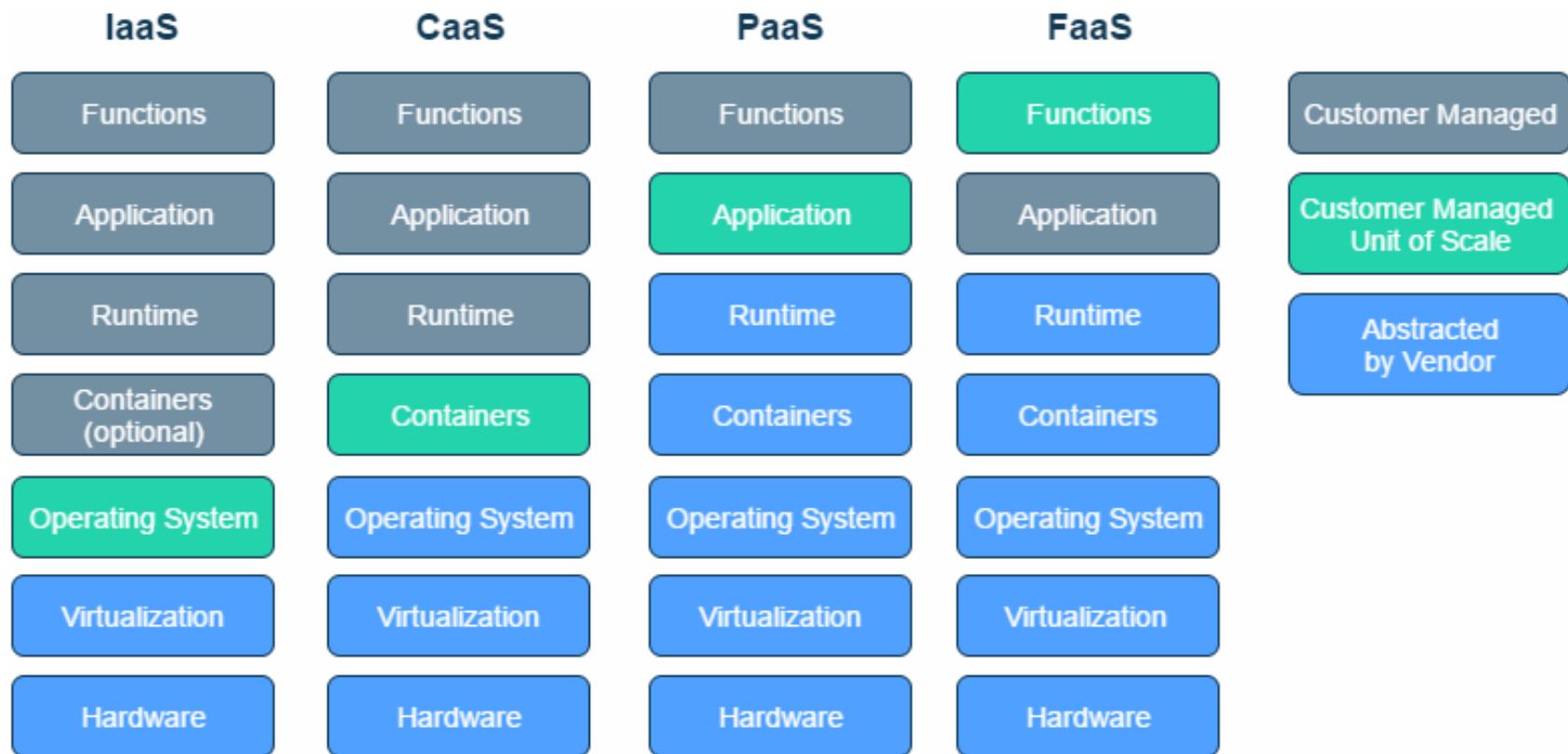
<http://www.avilashkumar.com/2013/02/csrf.html>

# Cloud Computing Basics

# What is Cloud?

- "The cloud" refers to servers that are accessed over the Internet, and the software and databases that run on those servers.
- Cloud servers are located in data centers all over the world.
- By using cloud computing, users and companies don't have to manage physical servers themselves or run software applications on their own machines.
- <https://www.cloudflare.com/learning/cloud/what-is-the-cloud/>

# Types of Cloud Computing



<https://serverless.zone/abstracting-the-back-end-with-faas-e5e80e837362>

# Cloud Provider



Google Cloud



HEROKU



# Workshop 7 – Deploy the web service

- Follow the link and try to deploy the server
  - <https://devcenter.heroku.com/articles/getting-started-with-go>



Next Step...

# Challenge

- Upgrade the web services using ORM (gorm)
- Create your databases and web services
- Find and try other Go backend frameworks

# Recommended Topics

- Other programming languages
  - Rust
  - Nim
- Foundations of computer systems
- Other databases
- API testing methodologies
- System scaling
- Security and reliability
- DevOps
  - Continuous integration / Continuous delivery (CI/CD)
  - Automated tools

A dark gray background featuring several translucent, overlapping wireframe geometric shapes, including cubes and spheres, which create a sense of depth and complexity.

**E N D**

# More References

- <https://martinfowler.com/>