Web Services

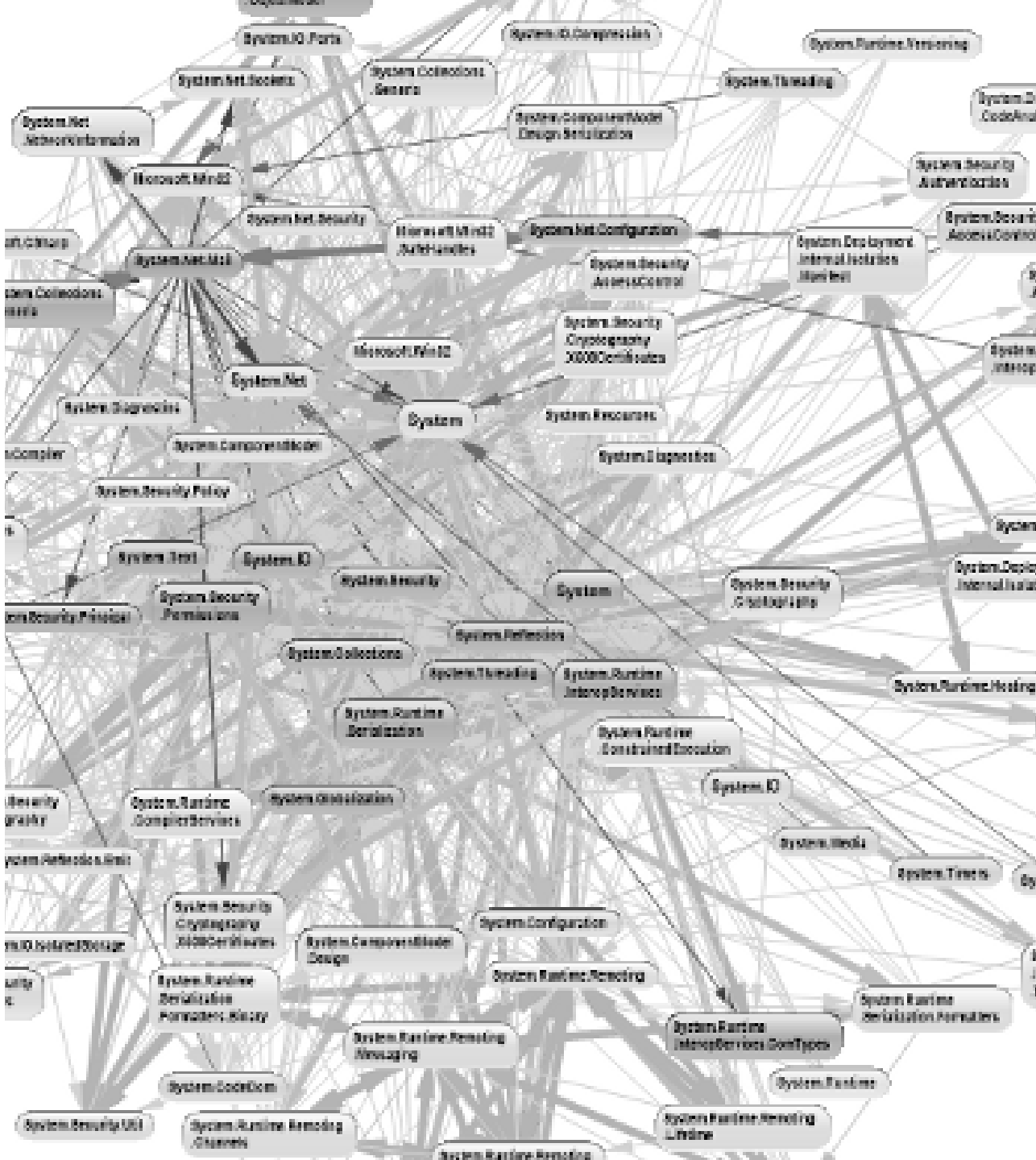
Agenda

- SOA
- API
- Restcountries API - *Code*
- Spotify API - *Code*
- ToDo FastAPI App - *Demo*

Why We Need Software Architecture

Architecture-less software becomes **unmanageable** with time and hence enhance the maintenance cost drastically with every new iteration.

As each and every change becomes costlier, this approach is termed as **Big Ball of Mud**



Architectural approaches

Over the years of evolutions in software design, developers have come up with different architectural approaches in order to avoid the issues of architecture less software design - **Big Ball of Mud**.

The most *famous* ones.

- *Layered Architecture*
- *Tiered Architecture*
- **Service Oriented Architecture (SOA)**

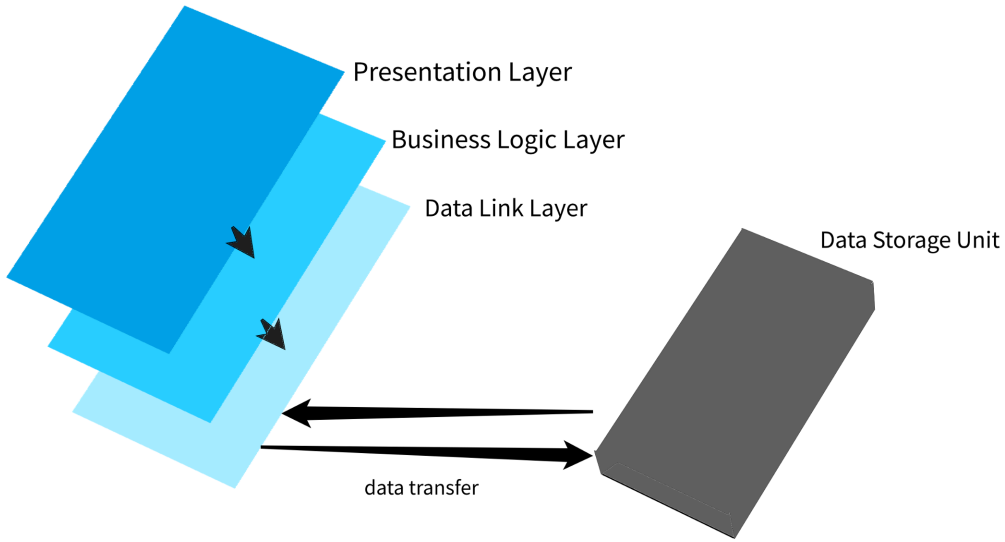
Layered Architecture

This approach works on principle of **separation of concerns**.

Software design is **divided into layer** laid over one another. Each layer performs a **dedicated responsibility**.

Architecture divides the software into the following layers

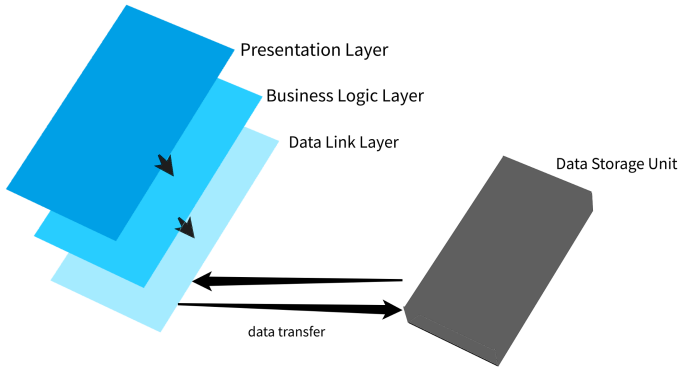
- **Presentation Layer**
- **Business Logic Layer**
- **Data Link Layer**



Layered Architecture

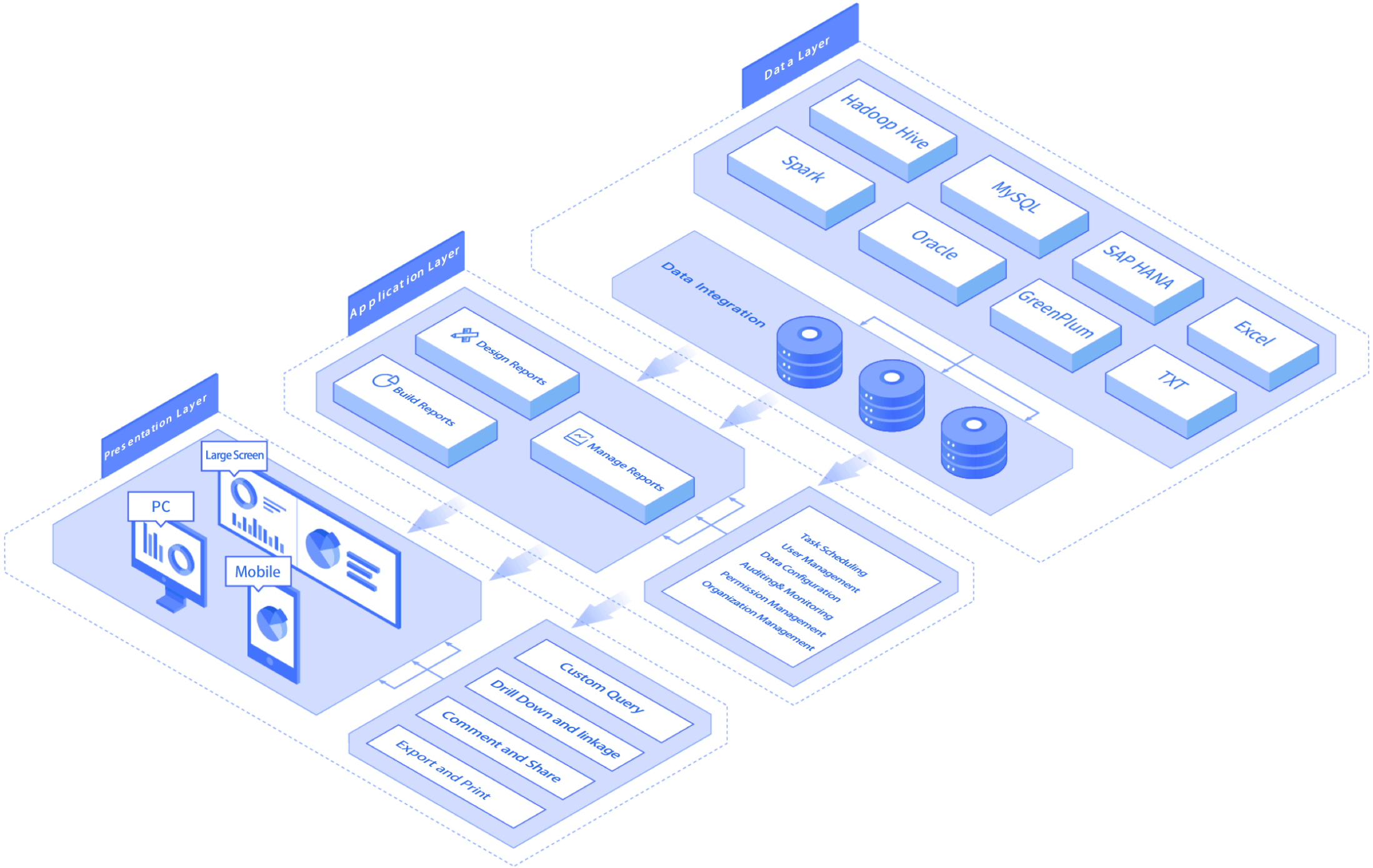
Advantages

- Simpler to implement
- Abstraction due to separation of concerns among layers
- Isolation between layers
- More manageable due to **low coupling**



Disadvantage

- Less scalability
- Monolith structure, lacking ease of modifications
- Data has to flow from each layer one after another


Data Layer
Hadoop Hive
Spark
MySQL
Oracle
SAP HANA
GreenPlum
Excel
TXT
Data Integration
Application Layer
Design Reports
Build Reports
Manage Reports
Presentation Layer
Large Screen
PC
Mobile
Task Scheduling
User Management
Data Configuration
Auditing& Monitoring
Permission Management
Organization Management
Custom Query
Drill Down and linkage
Comment and Share
Export and Print

Tiered Architecture

Divided the software into into **tiers** based on **client server communication principle**.

Can have **one**, **two** of **n-tiered** system separating the responsibilities among data provider and the consumer.

Single Tiered System

In this approach, **single system** is responsible to work **as client as well as server** and can offer ease of deployment eliminating the need of *Inter System Communications* (ISC).

This system are suitable **only for small scale single user application** and should not be used for multi user complex applications.

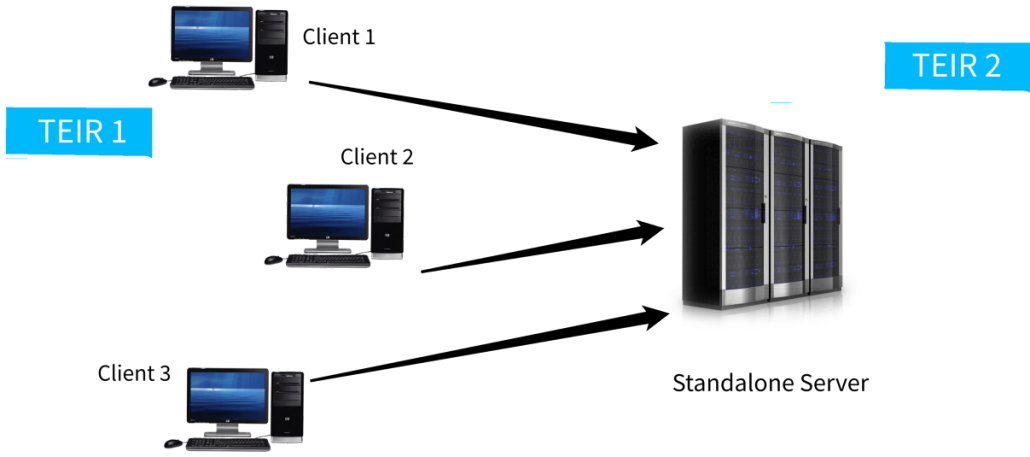
2-Tiered System

This system consist of **two physical machines**

- **server**
- **client**

It provides **isolation** among the **data management** operations and **data processing** and representation operations.

- *Client* holds **Presentation, Business Logic and Data link layer**.
- *Server* holds the **Data stores** such as Databases

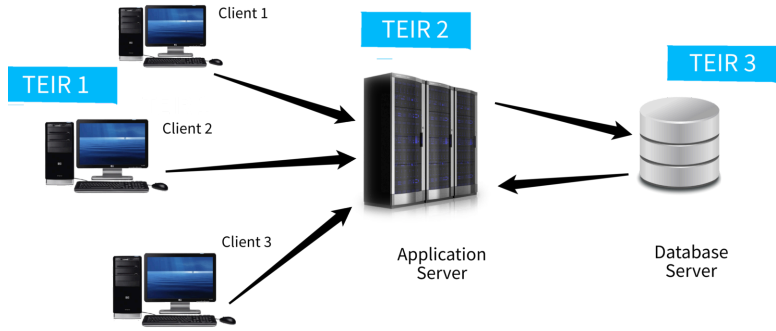


3-Tiered / n-Tiered System

Highly scalable both horizontally and vertically. Implementing n-tiered architecture is generally **costlier but offer high performance**. Hence it is preferred in **large complex software solutions**.

It can be **combined** with advanced **Service Oriented Architectural** style to generate highly sophisticated model.

It is **recommended** to use this architecture when the software is **complex** and requires **performance** as well as **scaling** as it can be a costlier approach in terms of resources as well as time.



Difference between Layers and Tiers

Layer

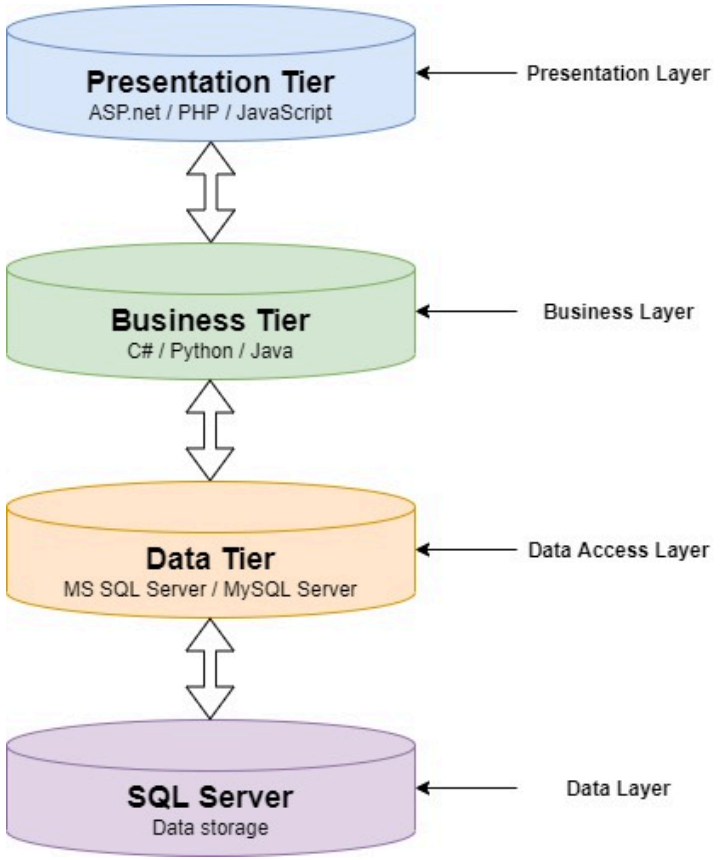
*Layers are the **logical** separation of code*

- Presentation Layer or *UI Layer*
- Business Layer or Business Logic Layer
- Data Access Layer and/or Data Layer

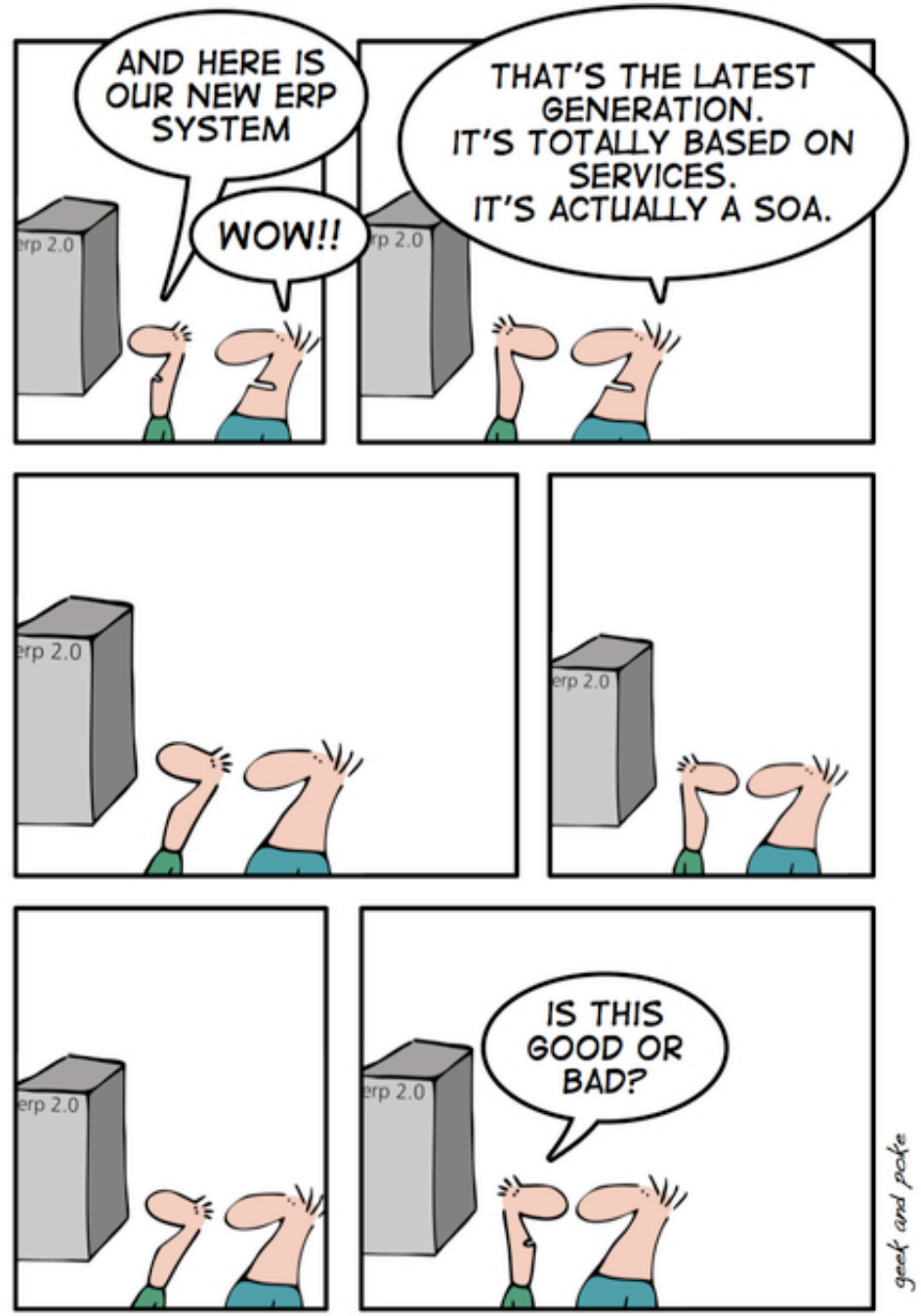
Tiers

*Tiers are the **physical** deployment of layers*

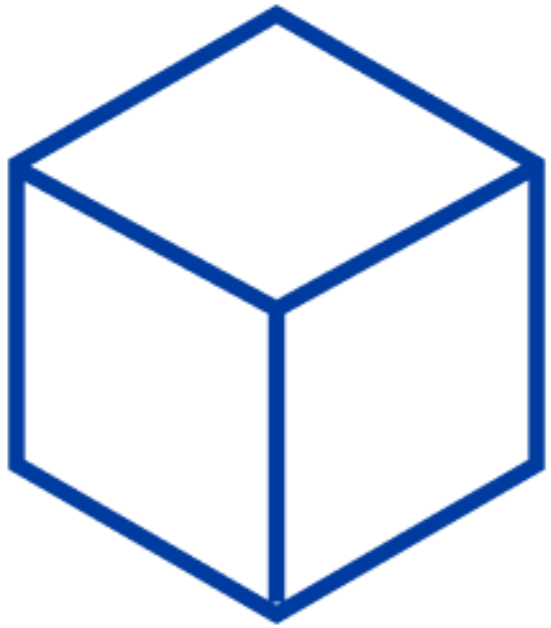
- Presentation Tier - *UI Tier*
- The Application Tier or Business Tier
- The Data Access Tier
- The Database Tier – *SQL Server, MySQL*



SOA

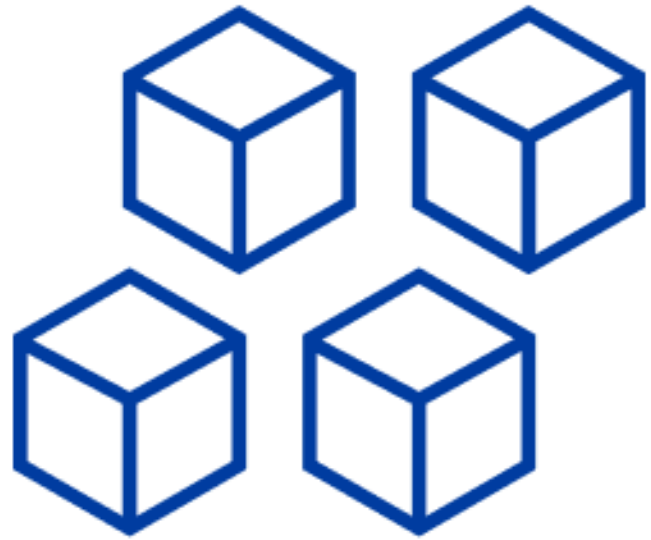


SOA FOR EVERYBODY



MONOLITHIC

Single unit



SOA

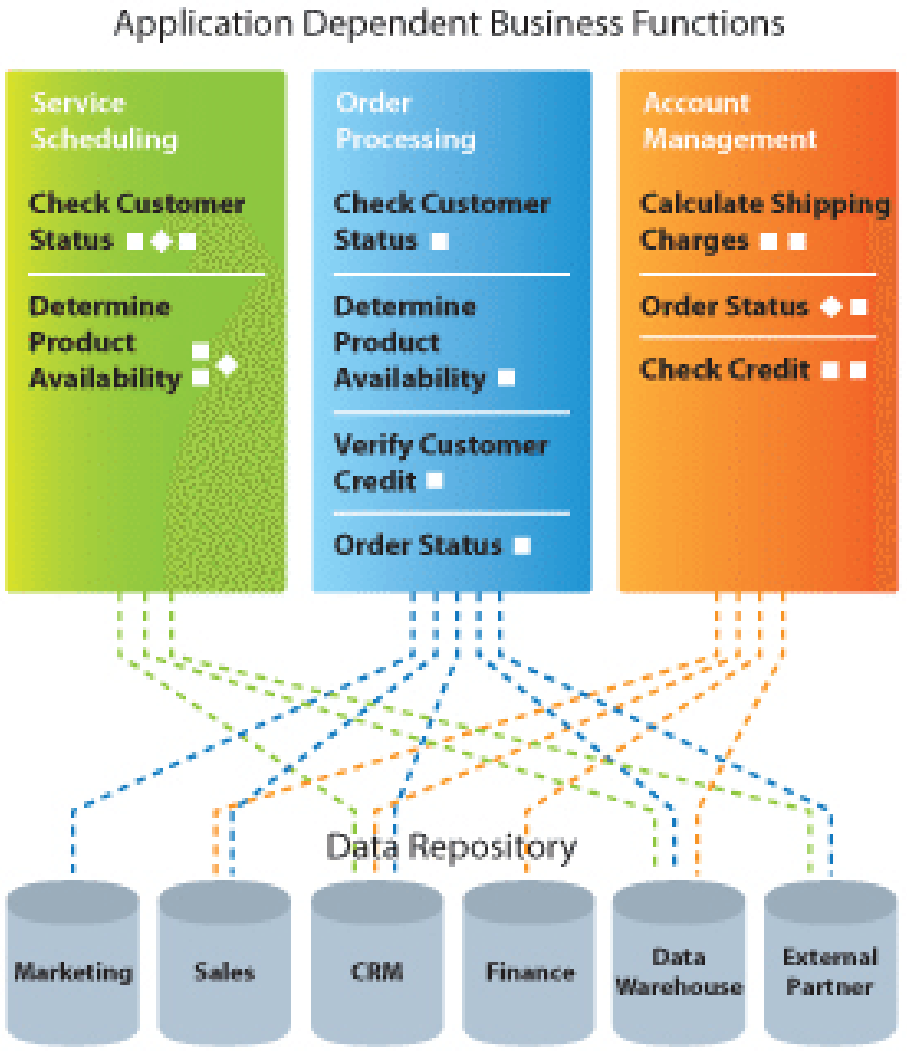
Coarse-grained

MICROSERVICES

Fine-grained

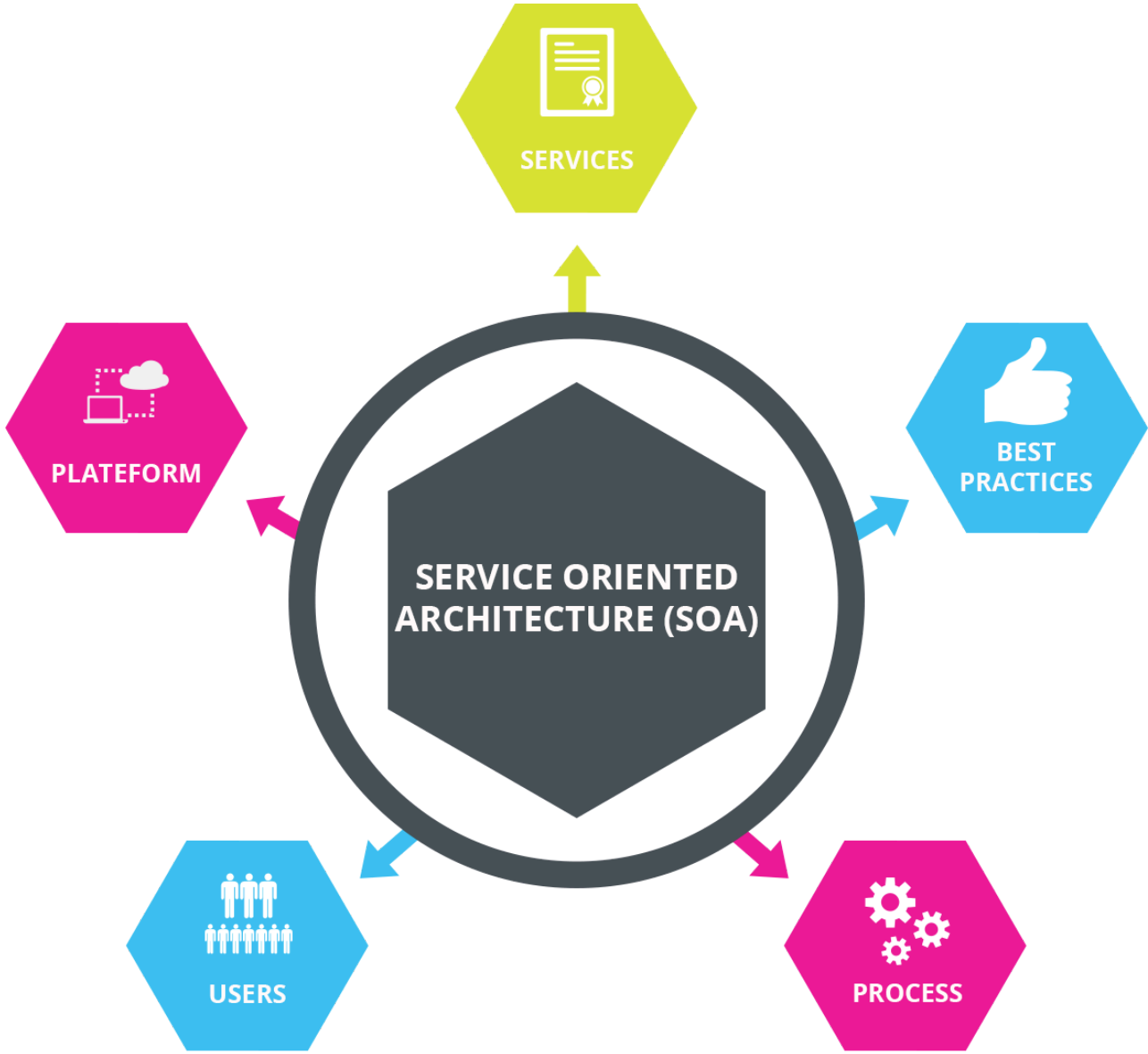
Before SOA

Closed - Monolithic - Brittle

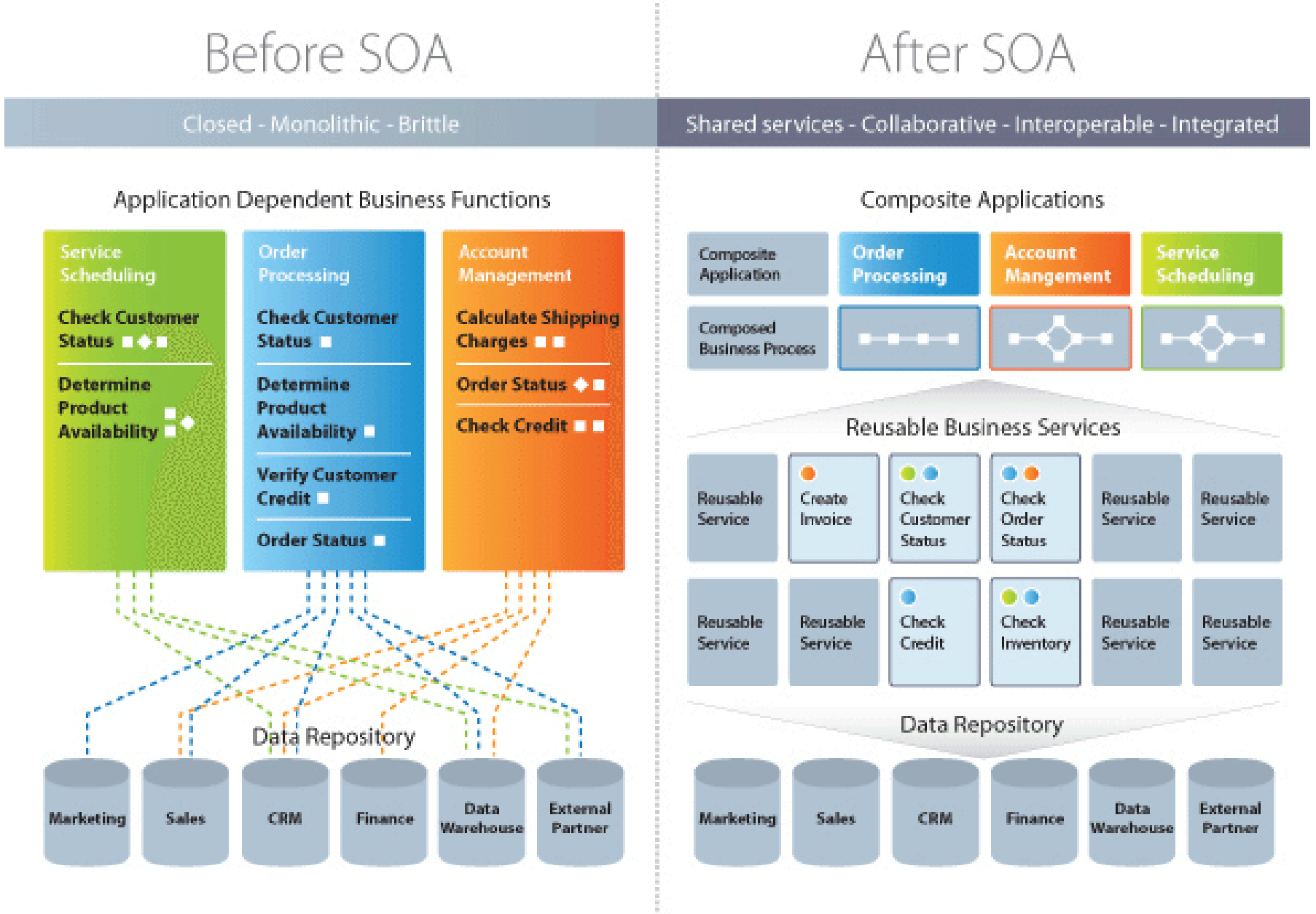


SOA - Service Oriented Architecture

SOA can be described as an **approach** to the **development process**, which, based on the business, leads to the development, acquisition and use of IT solutions as a set of business support, **reusable and flexible services**.



- SOA organize contexts in a **vertical way**
- **Multiples components** can be part of the **same service** providing multiples capabilities (operations)
- An SOA service is like a bounded context
- SOA fosters **reuse** and composition inside the **same domain**
- Each SOA service represents a group of **smaller components**
- In SOA, it is common to see all **services** using the **same technology** stack and the **same database technology**


Before SOA
After SOA
Closed - Monolithic - Brittle
Shared services - Collaborative - Interoperable - Integrated
Application Dependent Business Functions
Service Scheduling
Check Customer Status
Determine Product Availability
Order Processing
Check Customer Status
Determine Product Availability
Verify Customer Credit
Order Status
Account Management
Calculate Shipping Charges
Order Status
Check Credit
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner
Composite Applications
Composite Application
Order Processing
Account Mangement
Service Scheduling
Composed Business Process
Reusable Business Services
Reusable Service
Create Invoice
Check Customer Status
Check Order Status
Reusable Service
Reusable Service
Reusable Service
Reusable Service
Check Credit
Check Inventory
Reusable Service
Reusable Service
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner

API

What is an API?

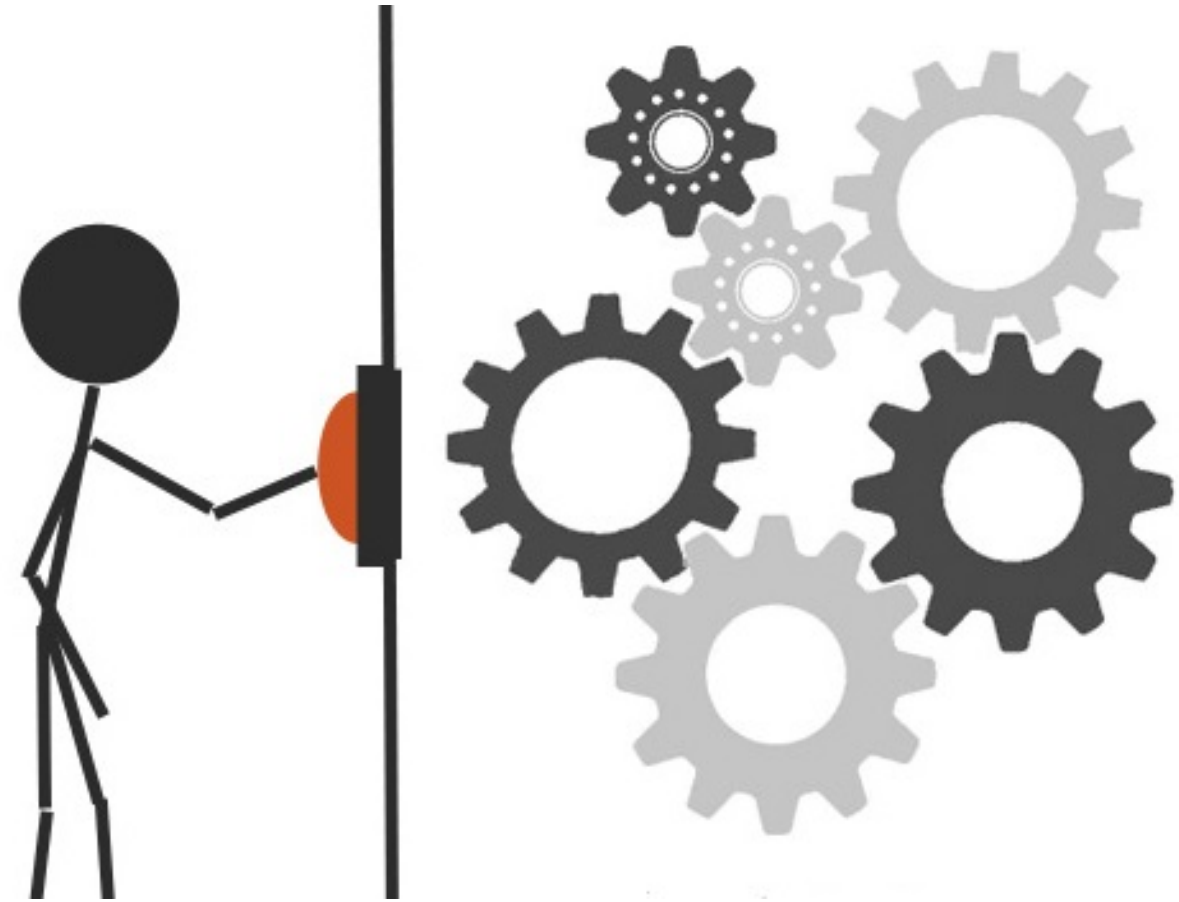
Find the best, in your opinion, description of an API

Come up with some examples of API's

What is an API?

API stands for **Application Programming Interface**

But what is a *Interface*?



Interfaces

Every device you use has some kind of **interface**.

We use these interfaces to get the device to **do the thing we want**.

We **don't need to understand** the underlying functionality.

Abstraction

API's provide a layer of abstraction for the user.

Abstraction **hides everything but what is relevant** to the user, making it *simple* to use.

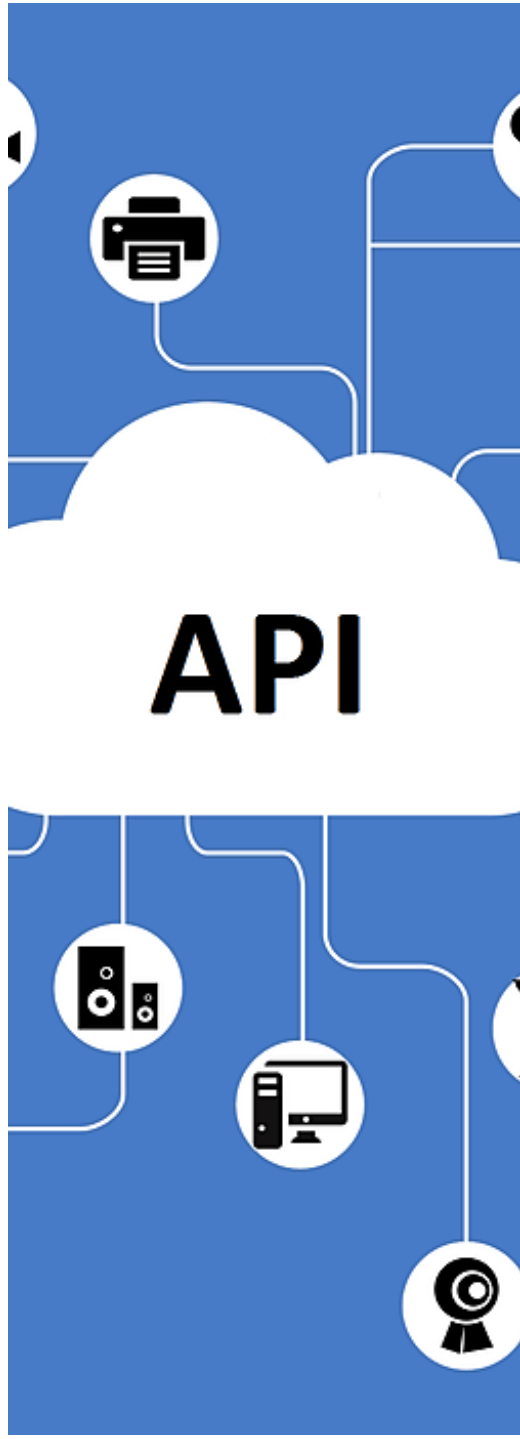
An API is how applications talk to each other

API - Application Programming Interface

API is a software intermediary that allows two applications to talk to each other.

You can ask an API for **data**, and they API will return what you want, usually in the form of **JSON** or **XML**. You can then use the data in your application.

Every time you use an app like Facebook, send an SMS, or check the weather on your phone, you're using an API.



API's as a way to serve your customers

Some companies are packaging API's as products.

- Weather Underground sells access to its weather data API
    - www.wunderground.com
- e-conomic has an API where the customers can access there data
    - www.e-conomic.com

When a company offers an API to their customers, it just means that they've built a set of dedicated URLs that return pure data responses — meaning the responses won't contain the kind of presentational overhead that you would expect in a graphical user interface like a website.

OpenAPI

There is the **OpenAPI Specification** (*OAS*), a technical specification that describes certain APIs, and there is the **OpenAPI Initiative** (*OAI*), an organization that enables specifications like OAS to thrive.

www.openapis.org

What is the difference between a Web service and an API?

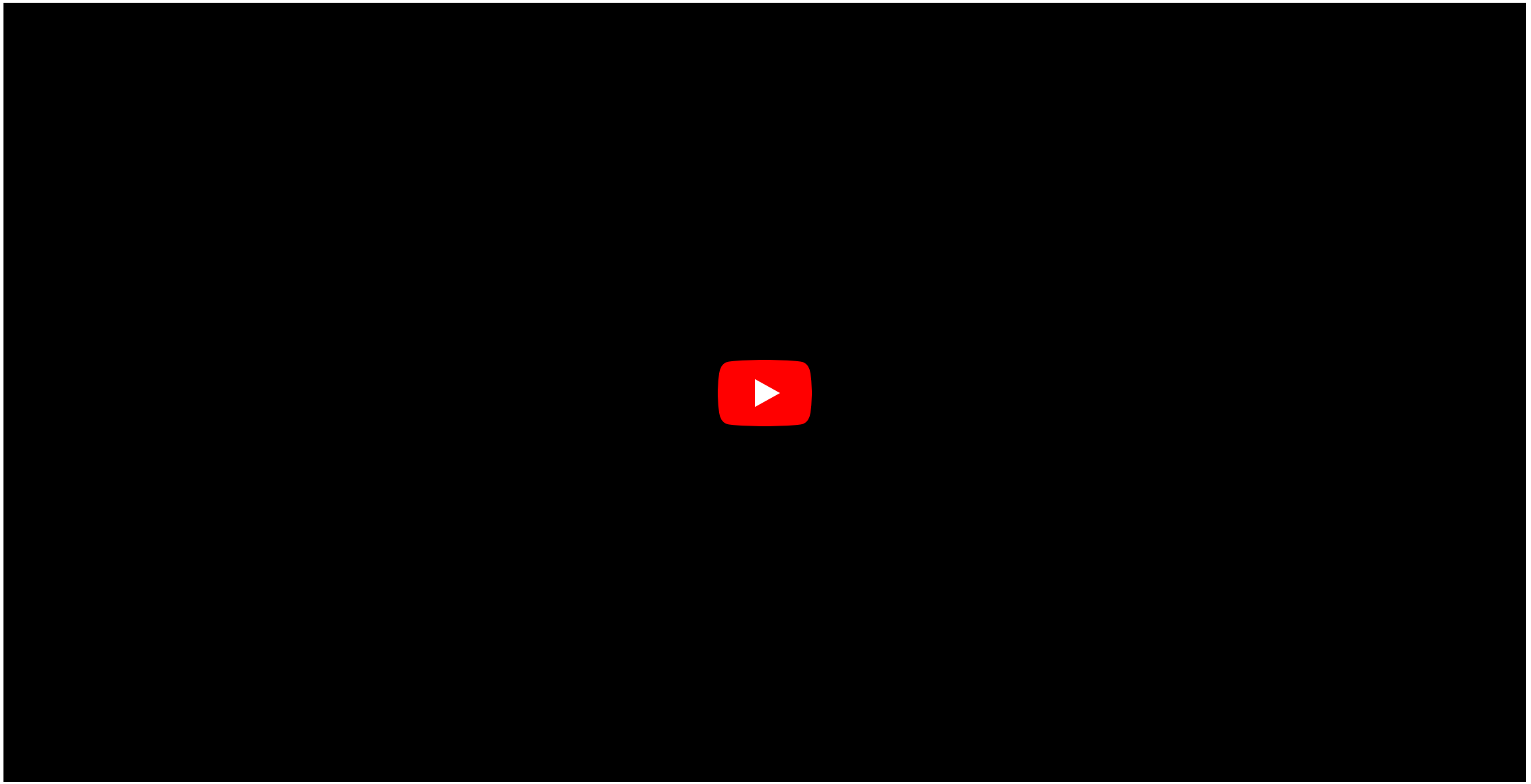
An **API** is an **interface** that allows you to build on the data and functionality of another application, while a **web service** is a **network-based resource** that fulfills a **specific task**.

Yes, there's **overlap between the two**:

- **All** web services are API's
- **Not all** API's are web services
- Web services require a network. APIs can be on- or offline, web services must use a network
- **Web services** are usually associated with **SOA**
- **API's are protocol agnostic**. API's can use any protocols or design styles - **Web services** use SOAP, REST, UDDI, XML-RPC

FastAPI

FastAPI is a modern, **fast** (*high-performance*), **web framework** for **building APIs** with Python 3.7+ based on standard Python type hints.



Flask <> FastAPI

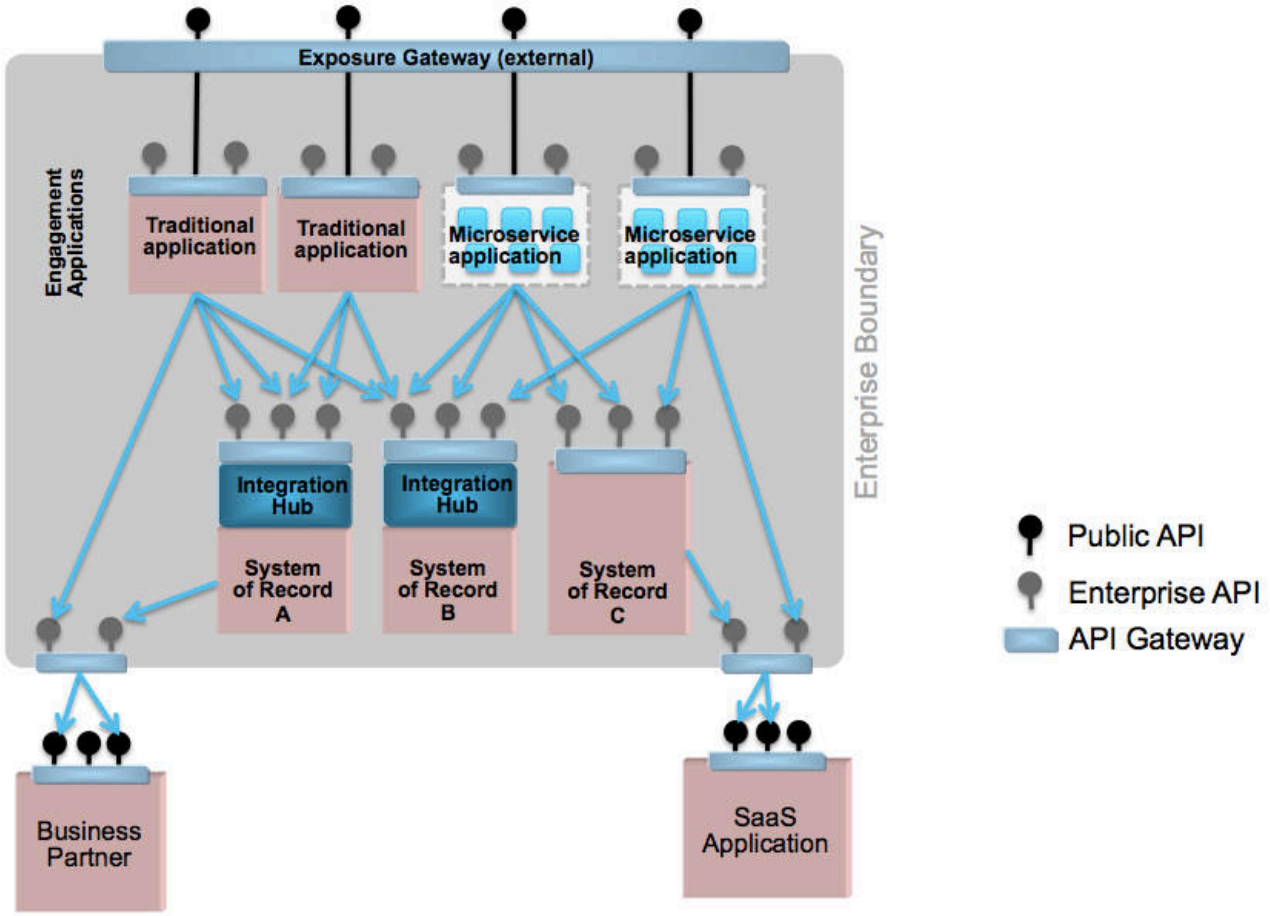
When to use Flask?

- To develop web applications
- To develop quick prototypes

When to use FastAPI?

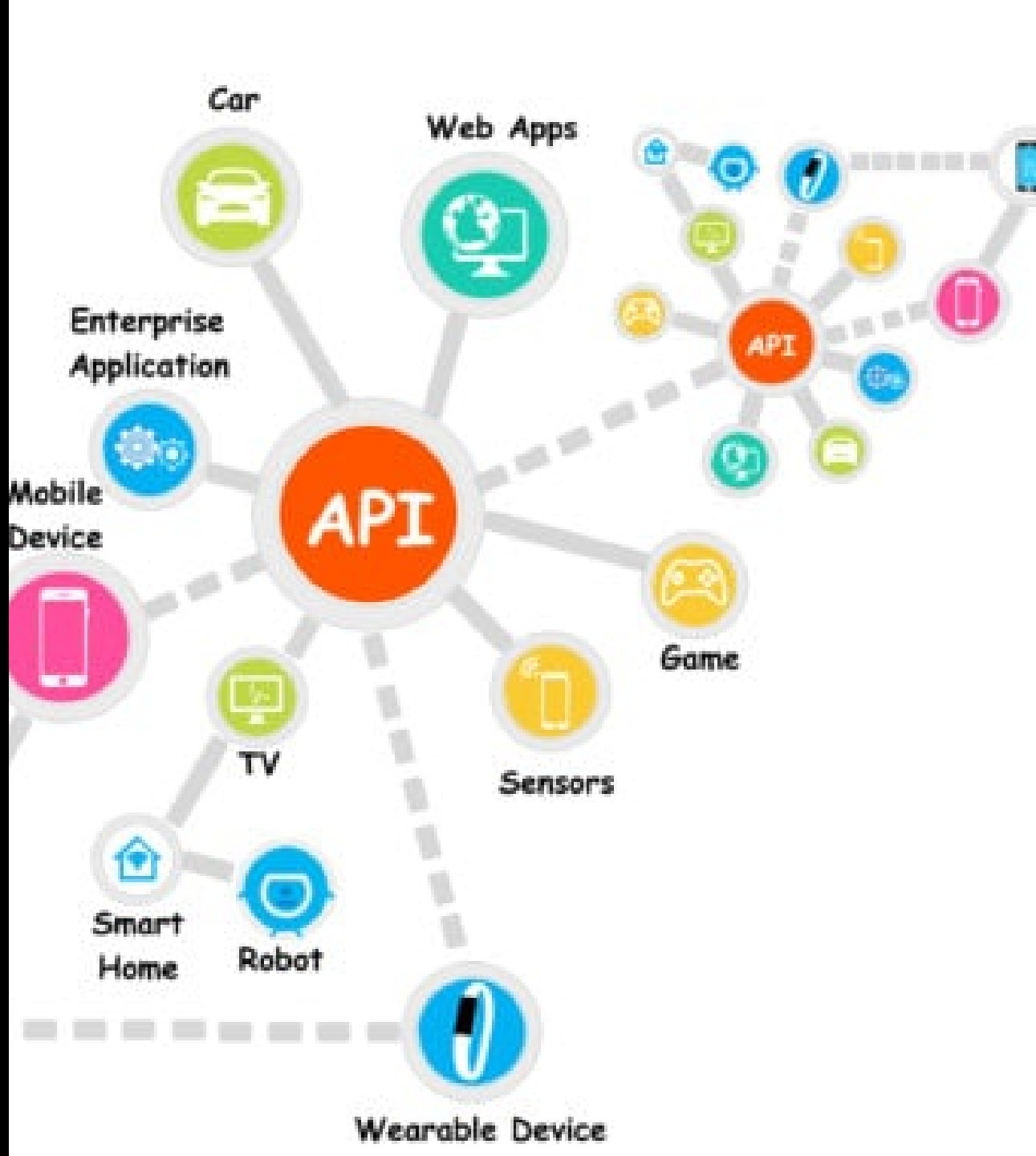
- To develop APIs from scratch
- To lower the number of bugs and errors in code

Microservices, SOA, and API's combined



https://developer.ibm.com/tutorials/1601_clark-trs

API DEMO


Car
Web Apps
Enterprise Application
API
Mobile Device
API
Game
TV
Sensors
Smart Home
Robot
Wearable Device

API - restcountries

- Python file - .py
- Jupyter Lab - .ipynb

https://restcountries.eu

python-restcountries https://pypi.org/project/python-restcountries/

```
[ ]: # Install
!pip install python-restcountries
```

```
[6]: # From restcountries import RestCountryApi as rapi
from restcountries import RestCountryApiV2 as rapi

# Get Denmark info
country_list = rapi.get_countries_by_name('Denmark')
```

```
[7]: # Print information
country = country_list[0]
print(country.name)
print(country.capital)
print(country.calling_codes)
print(country.population)
print(country.flag)
print(country.languages)
```

```
Denmark
Copenhagen
['45']
5717014
https://restcountries.eu/data/dnk.svg
[{'iso639_1': 'da', 'iso639_2': 'dan', 'name': 'Danish', 'nativeName': 'dansk'}
```

https://pypi.org/project/python-restcountries

Spotify API

Spotify provides software and app developers access to some of their data about users, playlists, and artists through a Web API.

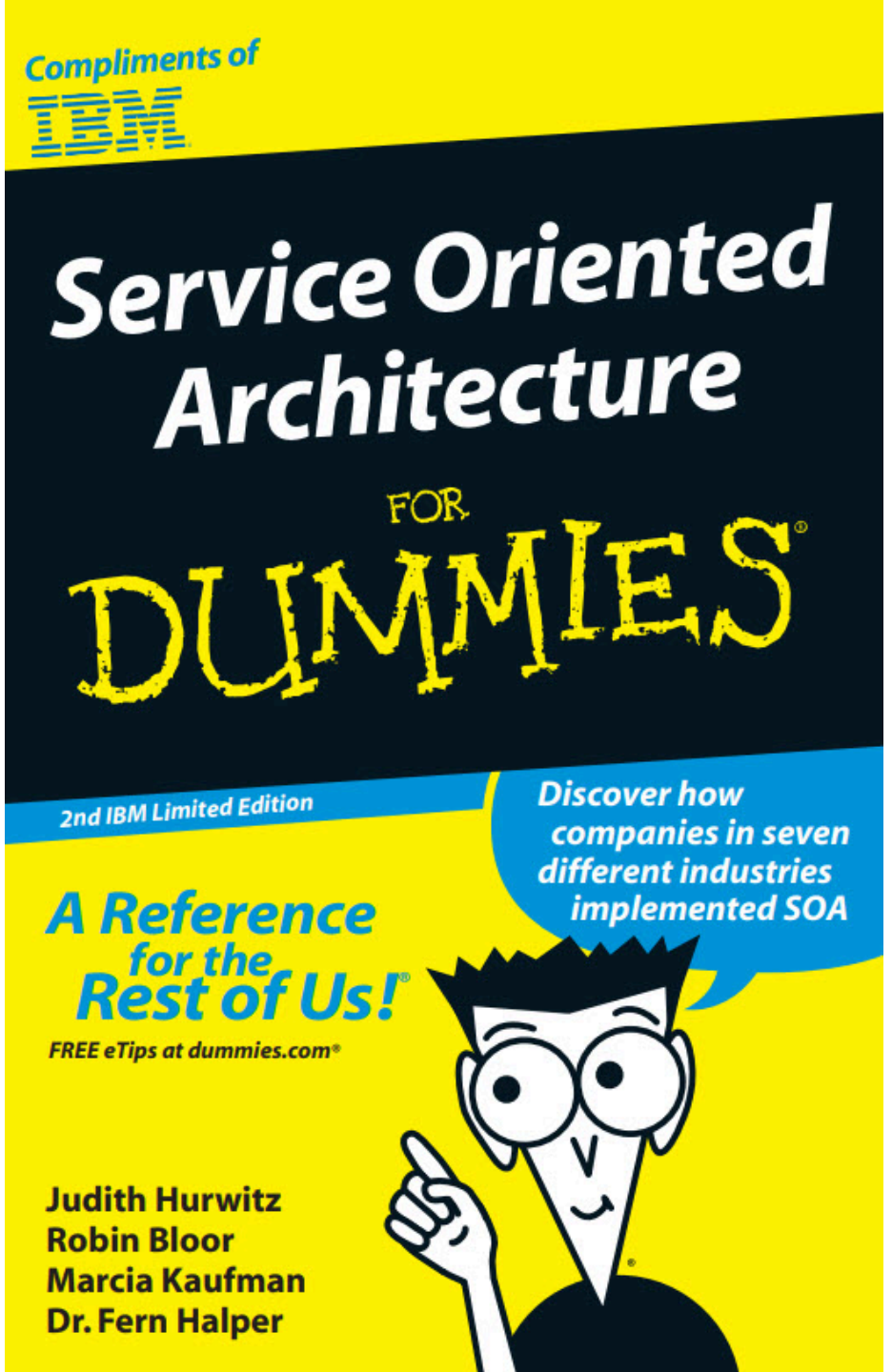
- Spotify_API_Spotipy.pdf
- Jupyter Lab Code .ipynb
- Python Code .py

Newscatcher

- Demo GitHub Reporitory
- https://newscatcherapi.com

IBM - SOA

SOA for Dummies



Links

- https://martinfowler.com/microservices
- https://www.ibm.com/cloud/learn/soa
- https://morioh.com/p/422b616d71a2
- https://fastapi.tiangolo.com/