Welcome!

DevOps

by

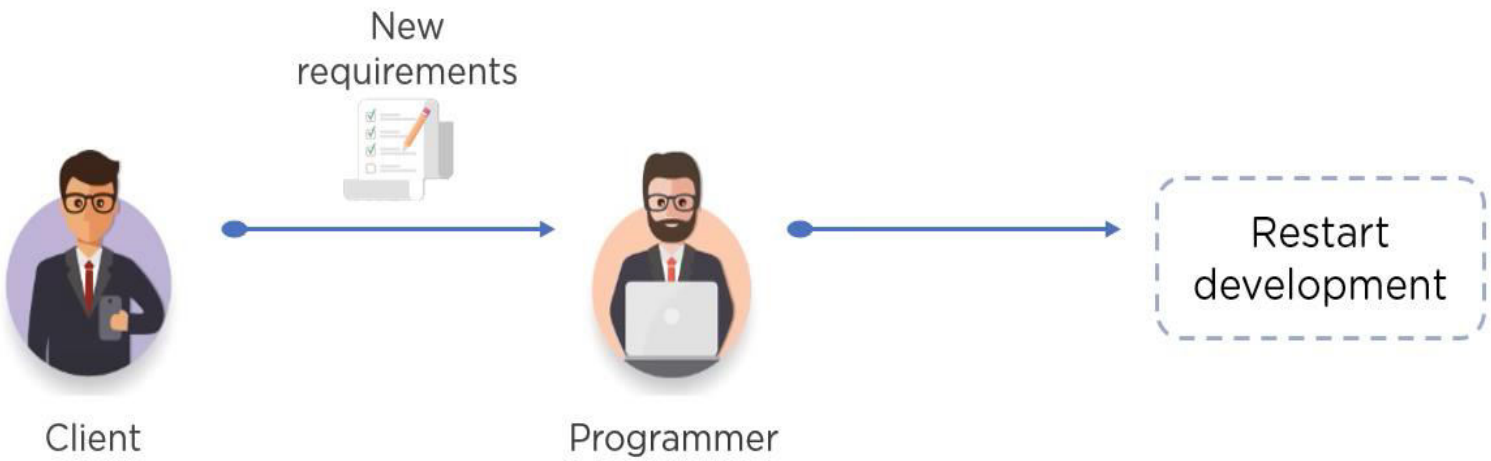
Girish Godbole

DevOps

Waterfall Model

Disadvantage of waterfall model

Any new requirements from the client will restart the development cycle

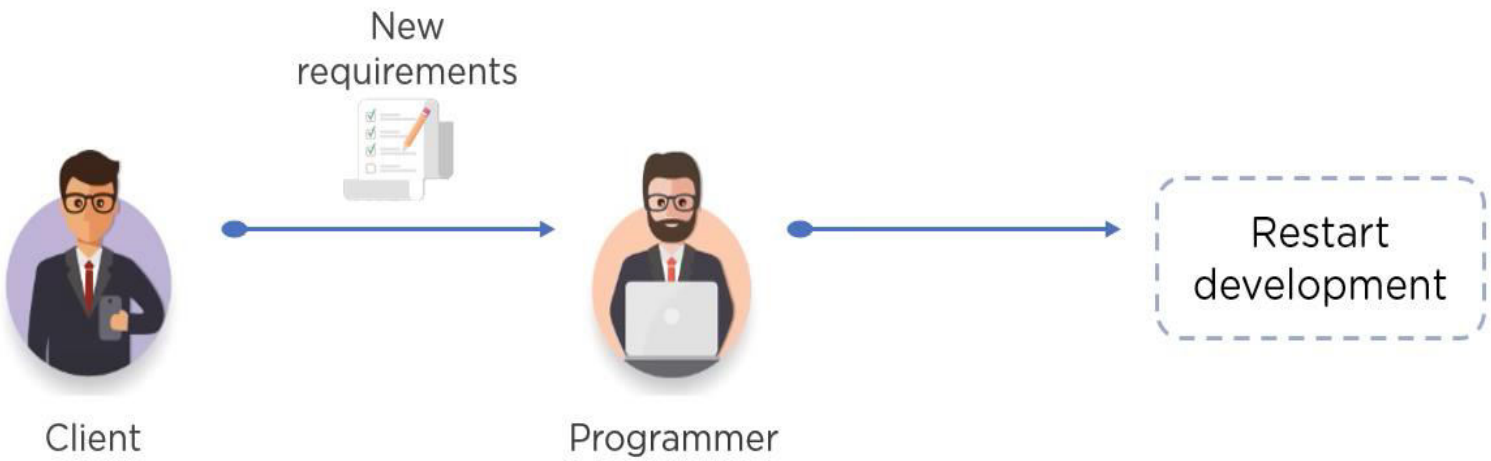


DevOps

Waterfall Model

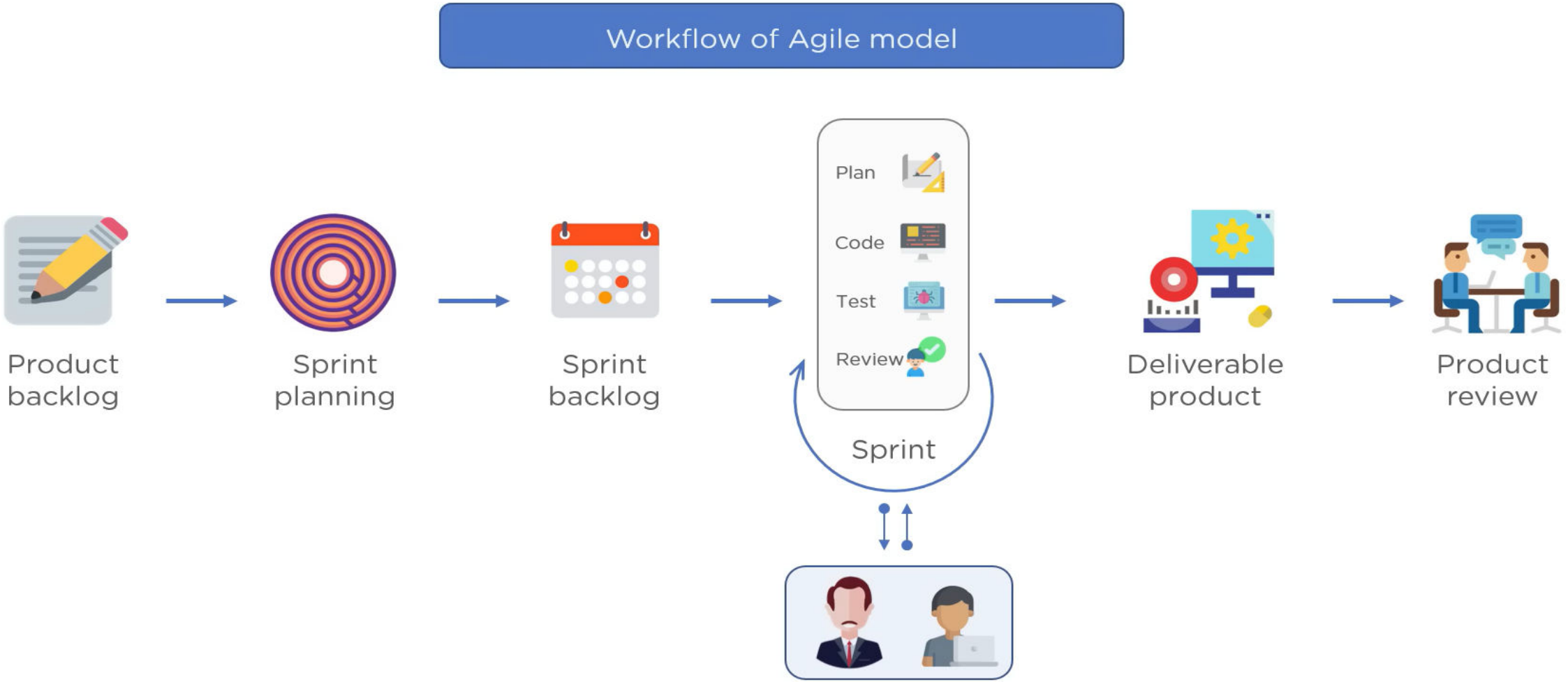
Disadvantage of waterfall model

Any new requirements from the client will restart the development cycle



DevOps

Agile Model



DevOps

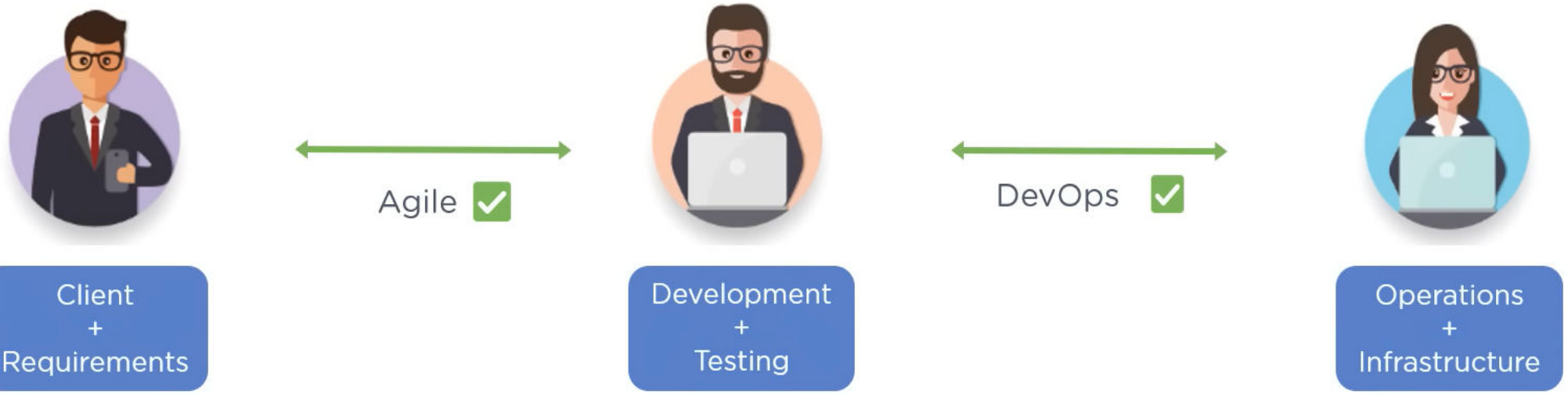
Agile Model

When the product fails in production servers, the operations team are clueless and send product back to the development team

DevOps

What is DevOps?

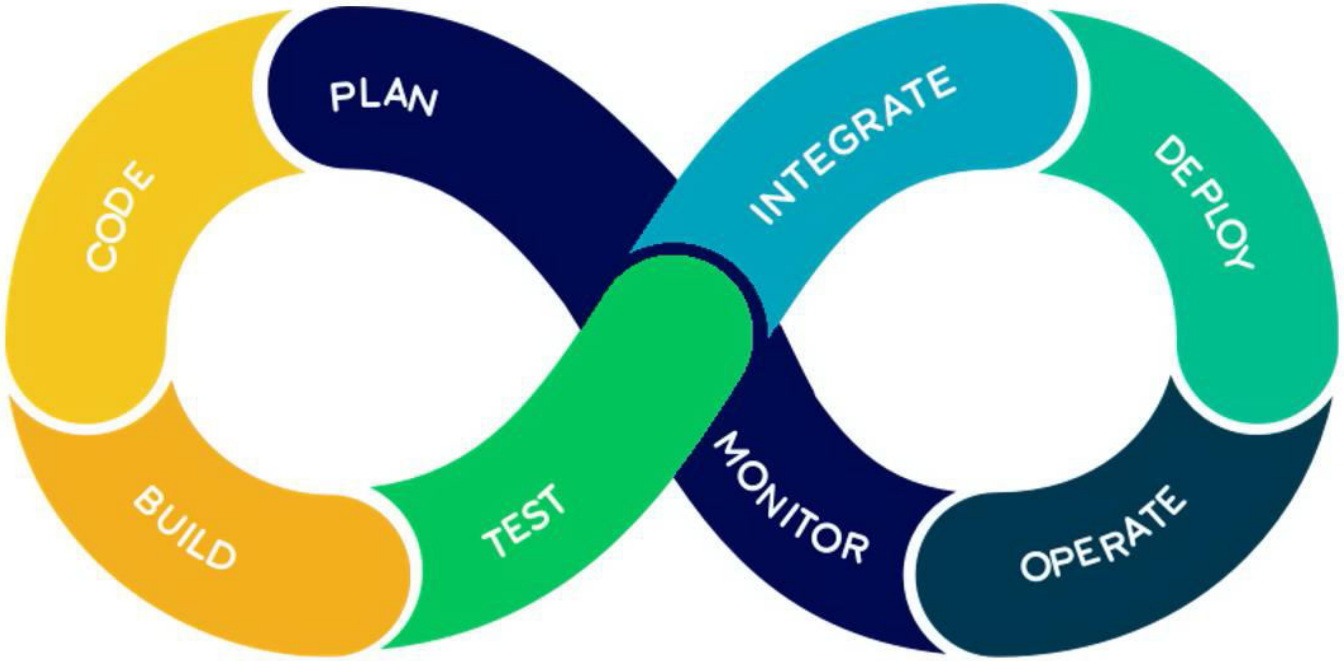
DevOps addressed the gap between Developers and Operations



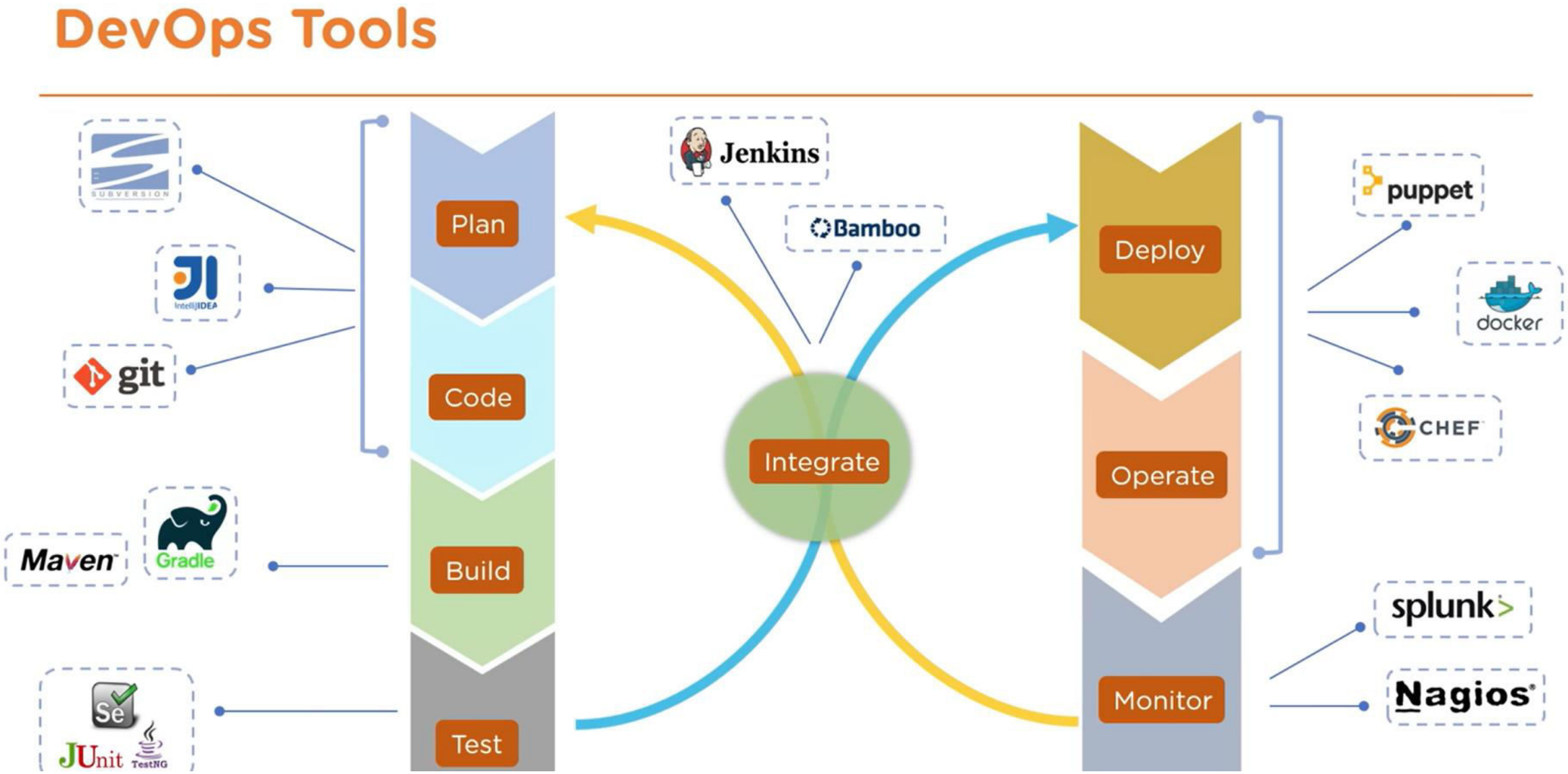
DevOps

DevOps Phases

According to DevOps practices, the workflow in software development and delivery is divided into 8 phases

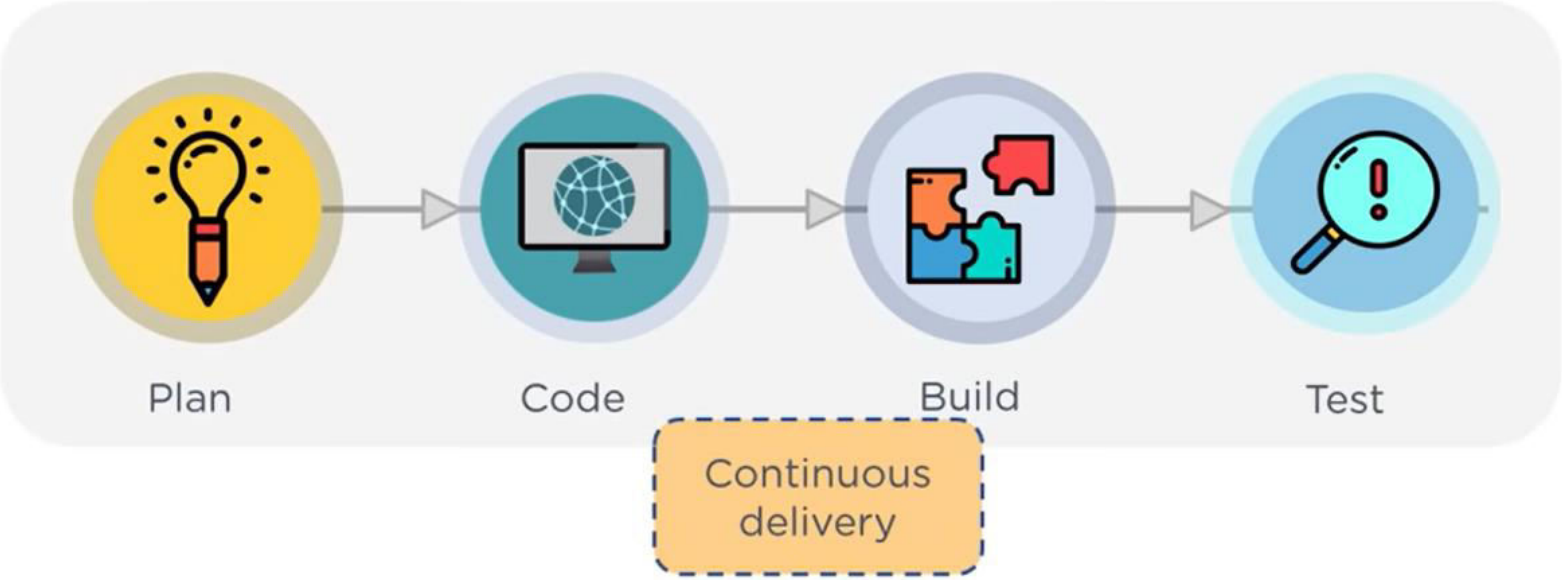


DevOps


DevOps Tools
Plan
Code
Build
Test
Integrate
Deploy
Operate
Monitor
Jenkins
Bamboo
puppet
docker
CHEF
splunk>
Nagios
git
Maven
Gradle
JUnit
TestNG

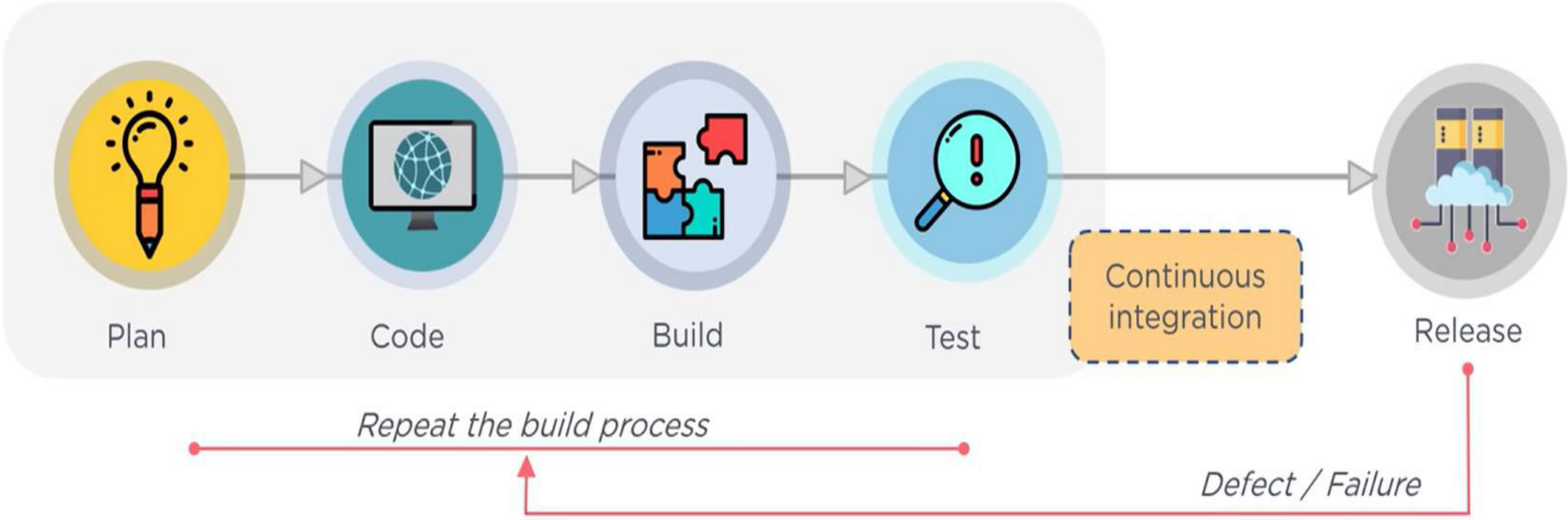
DevOps

Continuous Delivery

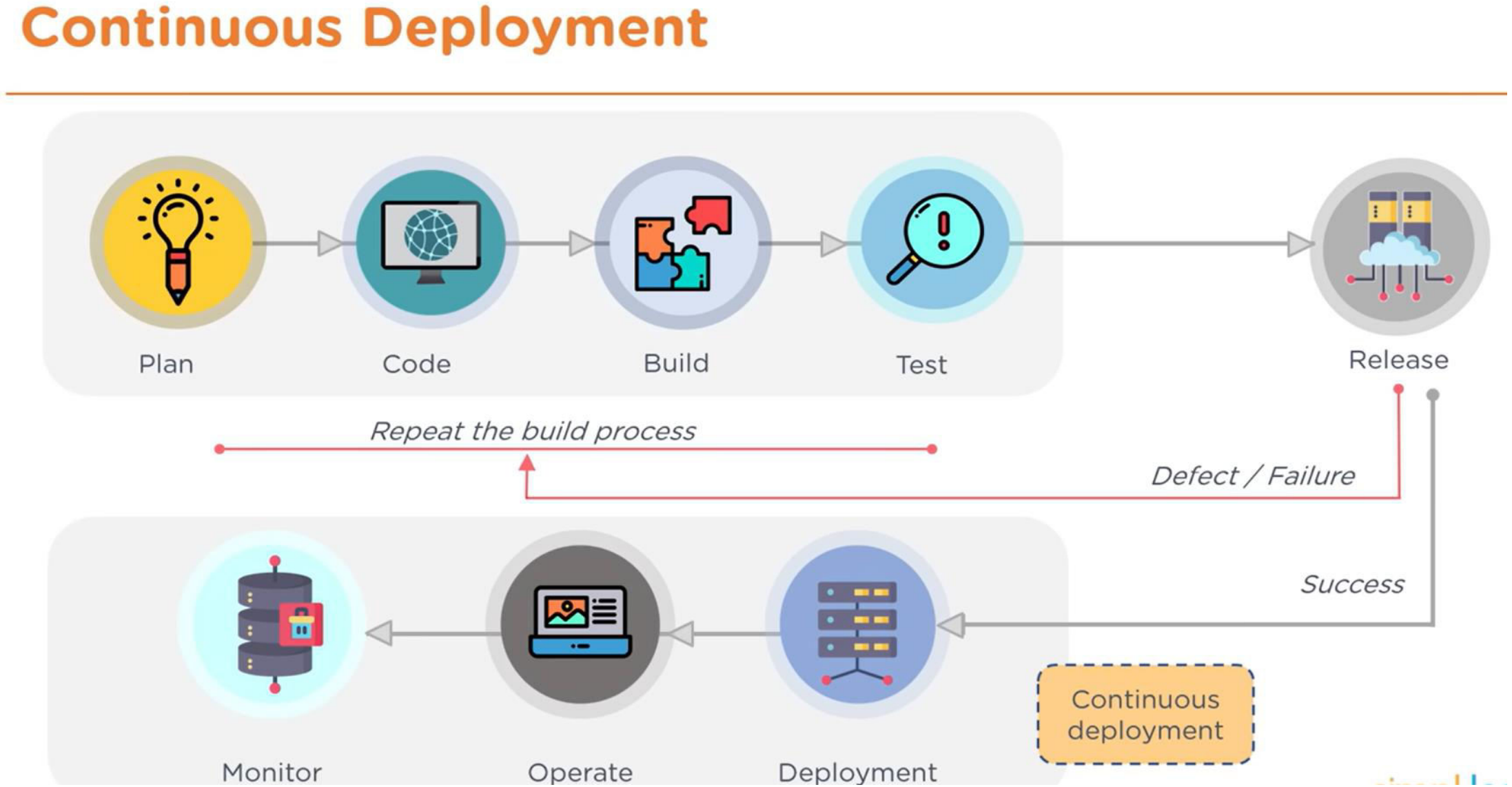


DevOps

Continuous Integration

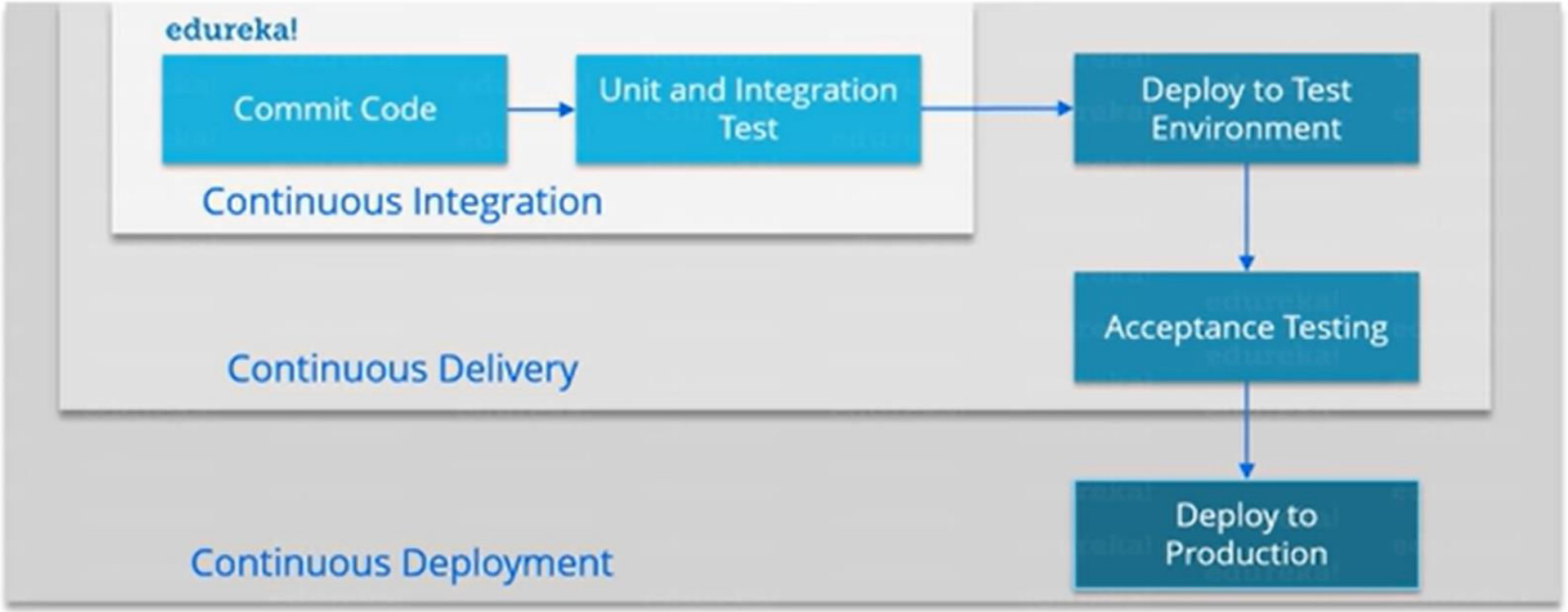


DevOps

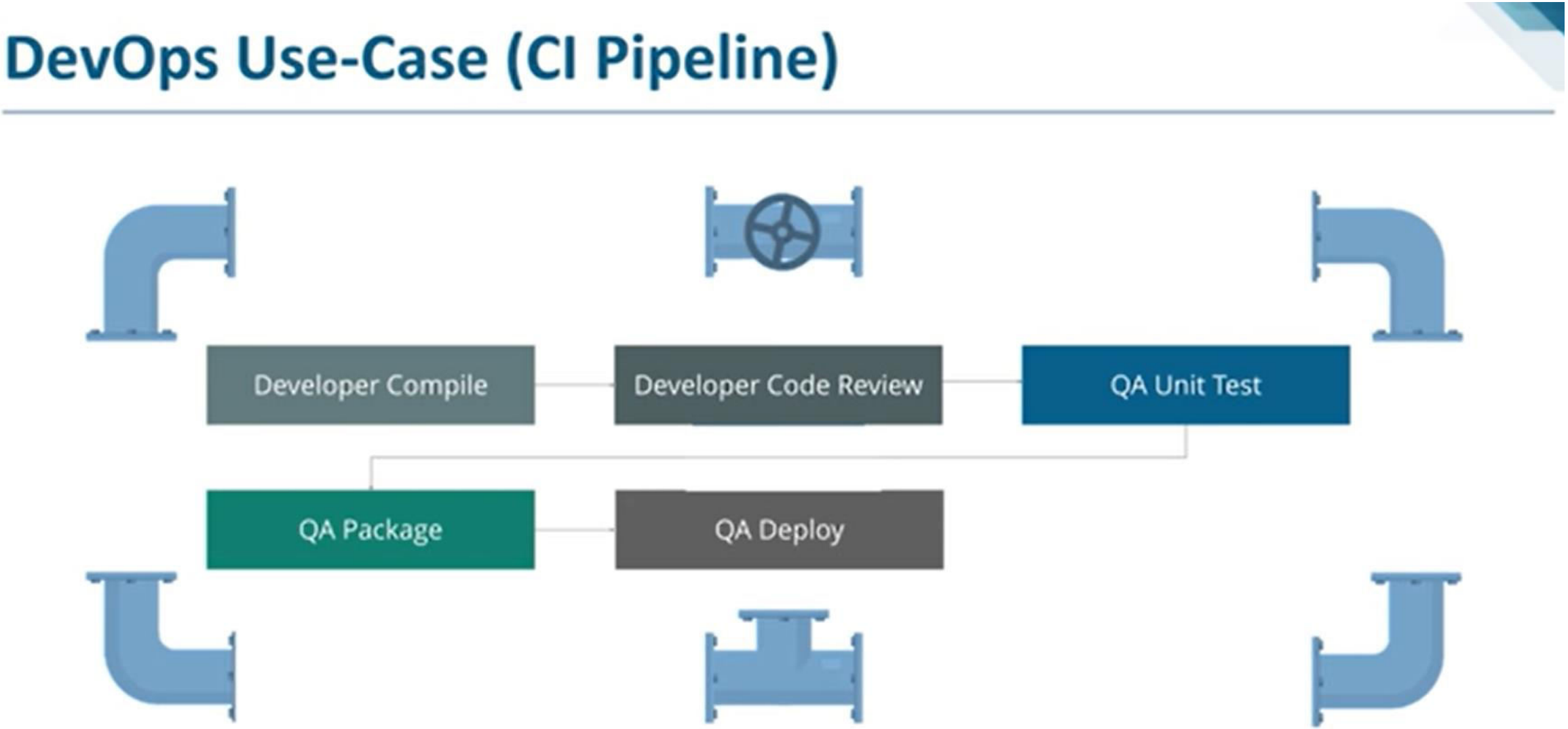

Continuous Deployment
Plan
Code
Build
Test
Release
Repeat the build process
Defect / Failure
Success
Monitor
Operate
Deployment
Continuous deployment

DevOps

CI/ CD


edureka!
Commit Code
Unit and Integration Test
Deploy to Test Environment
Continuous Integration
Acceptance Testing
Continuous Delivery
Deploy to Production
Continuous Deployment

DevOps


DevOps Use-Case (CI Pipeline)
Developer Compile
Developer Code Review
QA Unit Test
QA Package
QA Deploy

DevOps

DevOps Advantages

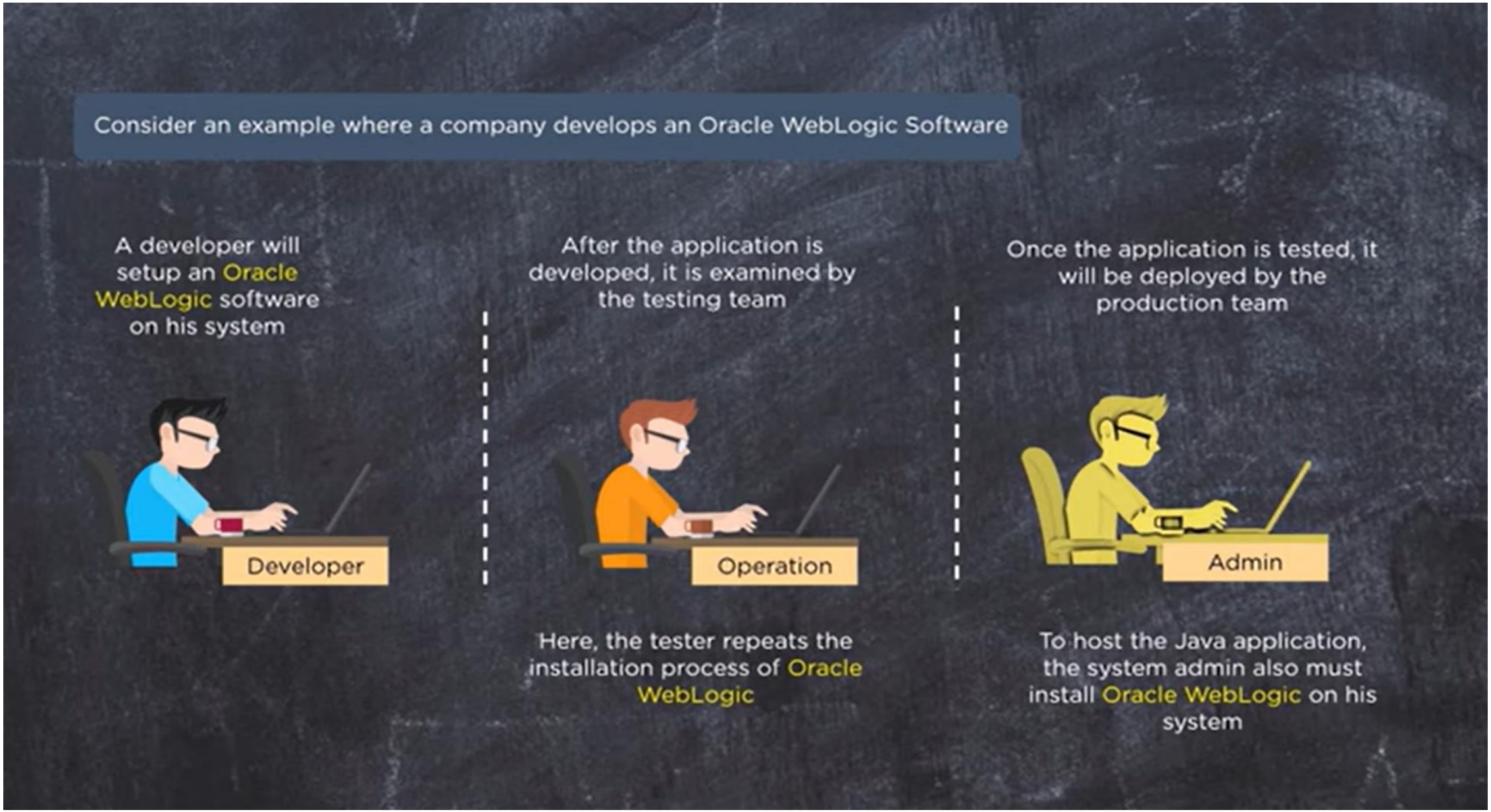
Time taken to create and deliver software is reduced

Complexity of maintaining an application is reduced

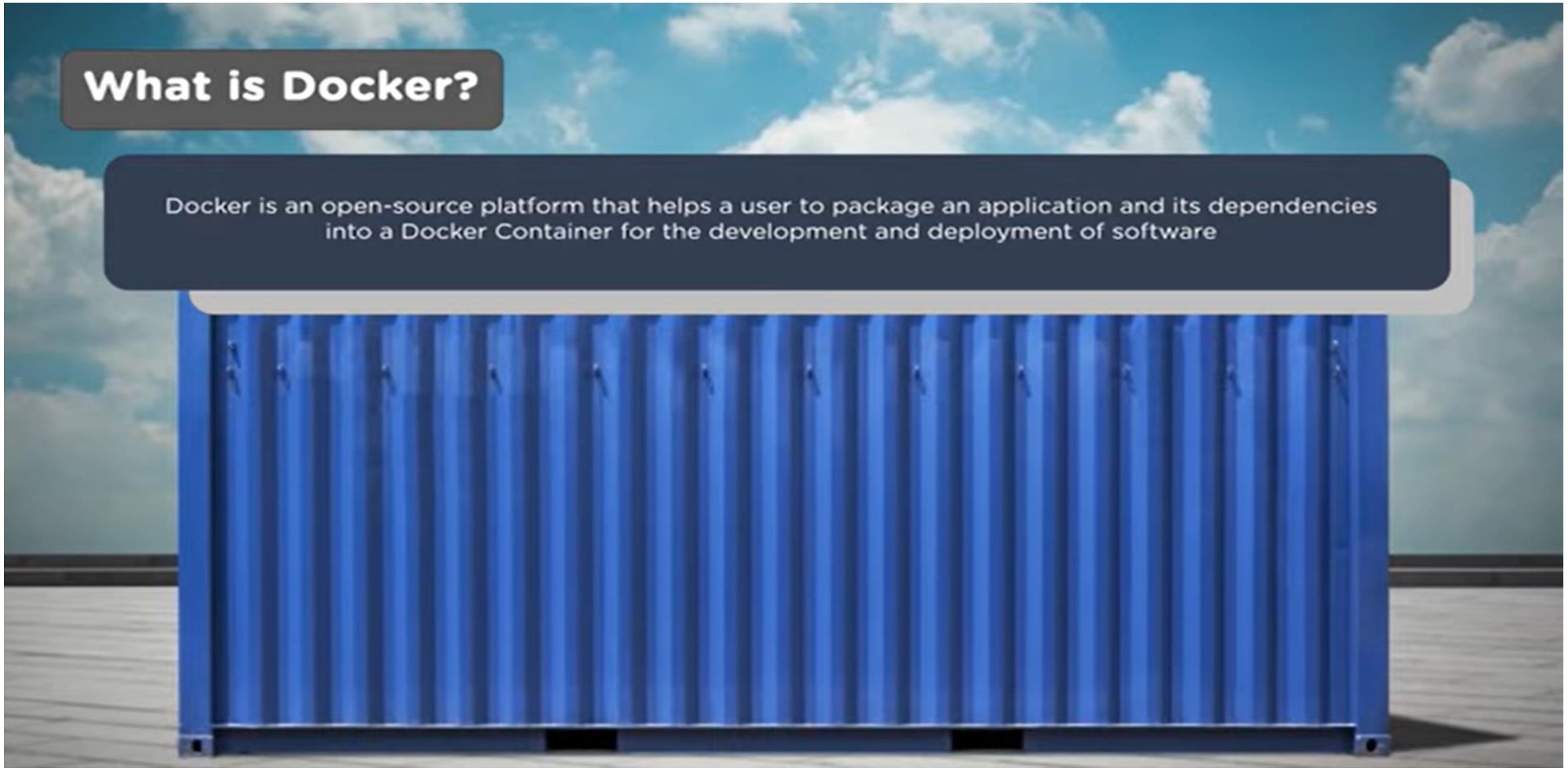
Improved collaboration between developers and operations team

Continuous integration and delivery ensure faster time to market

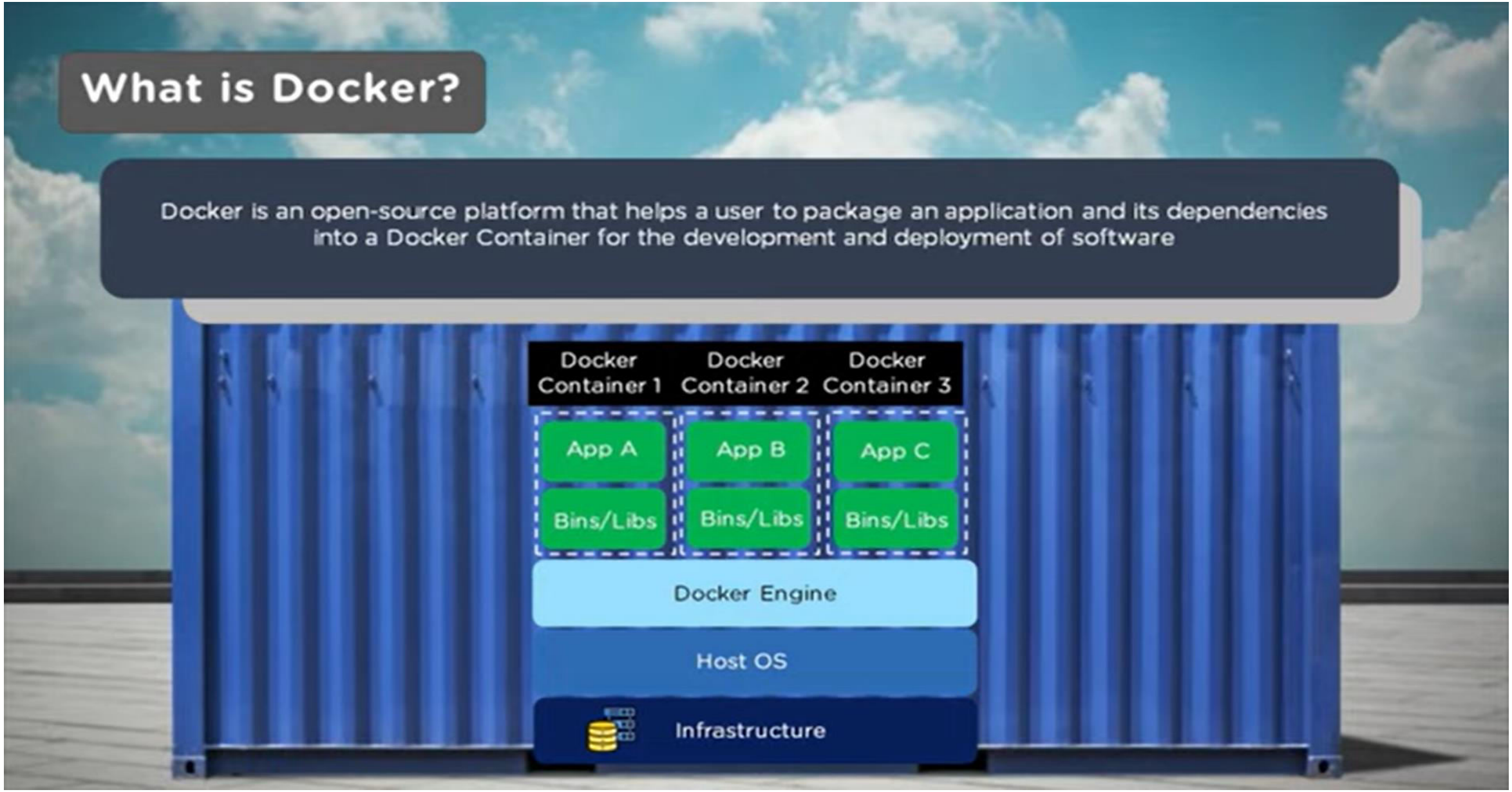
Docker


Consider an example where a company develops an Oracle WebLogic Software
A developer will setup an Oracle WebLogic software on his system
Developer
After the application is developed, it is examined by the testing team
Operation
Here, the tester repeats the installation process of Oracle WebLogic
Once the application is tested, it will be deployed by the production team
Admin
To host the Java application, the system admin also must install Oracle WebLogic on his system

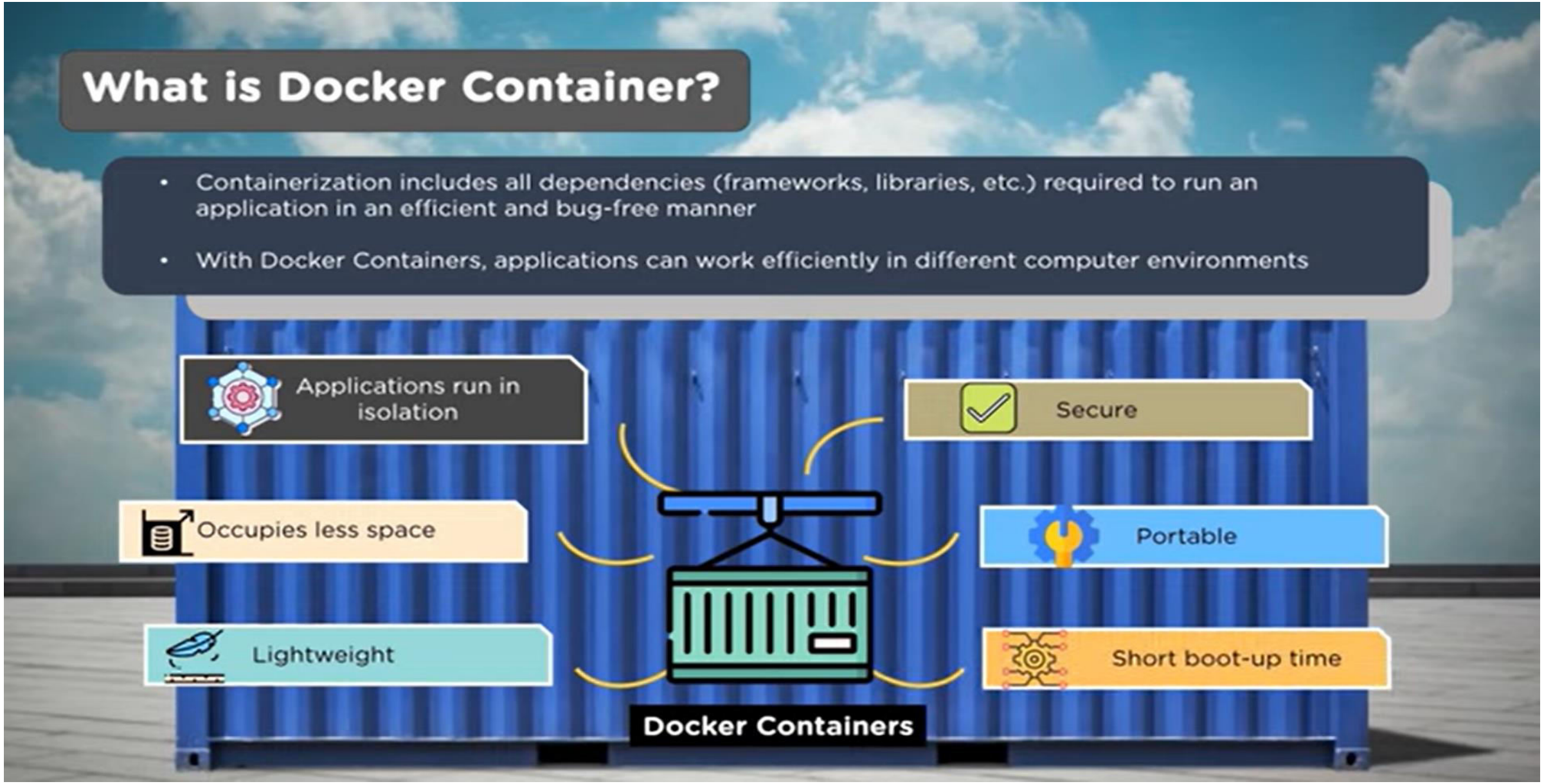
Docker


What is Docker?
Docker is an open-source platform that helps a user to package an application and its dependencies into a Docker Container for the development and deployment of software

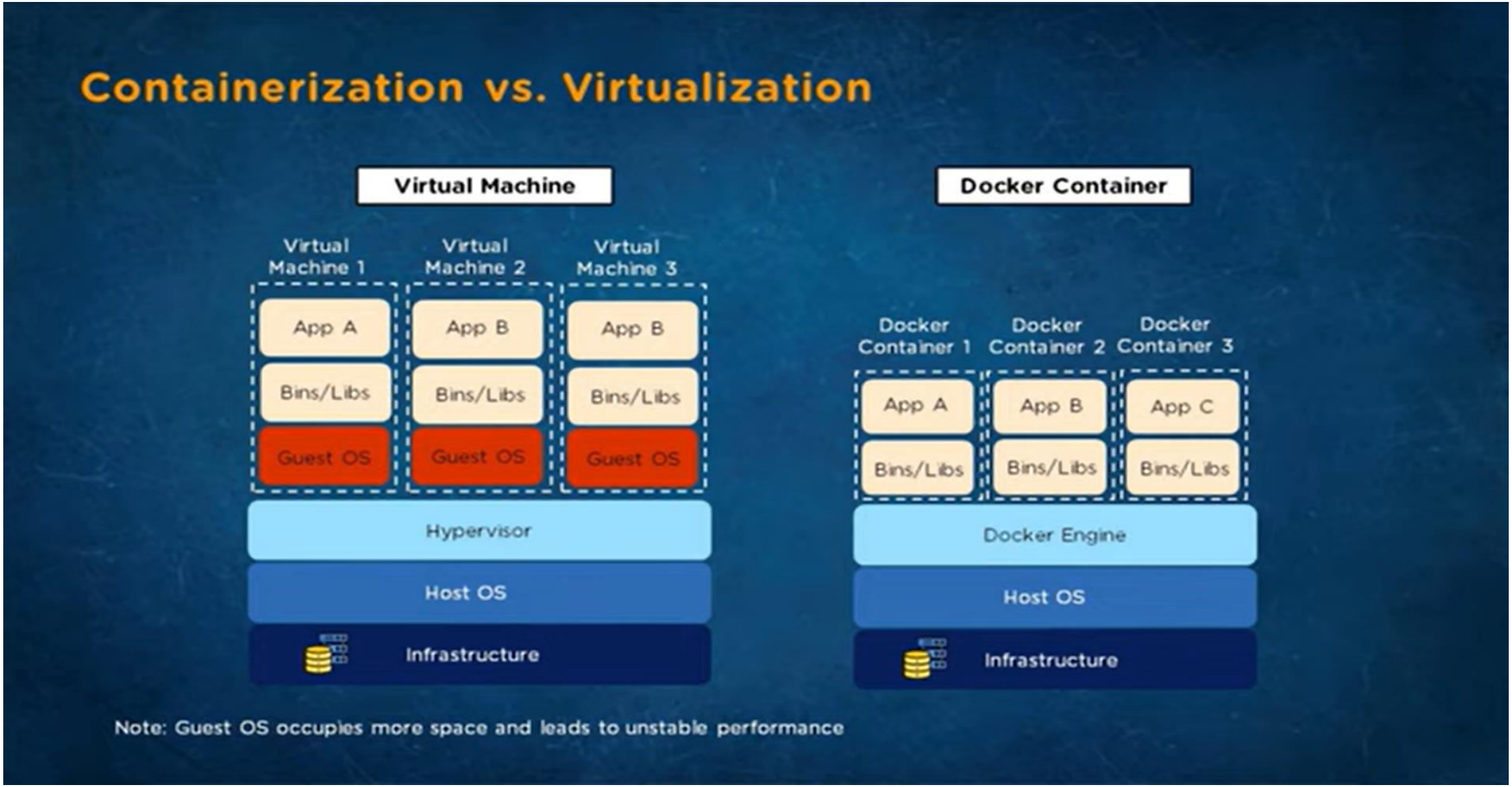
Docker


What is Docker?
Docker is an open-source platform that helps a user to package an application and its dependencies into a Docker Container for the development and deployment of software
Docker Container 1
Docker Container 2
Docker Container 3
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Infrastructure

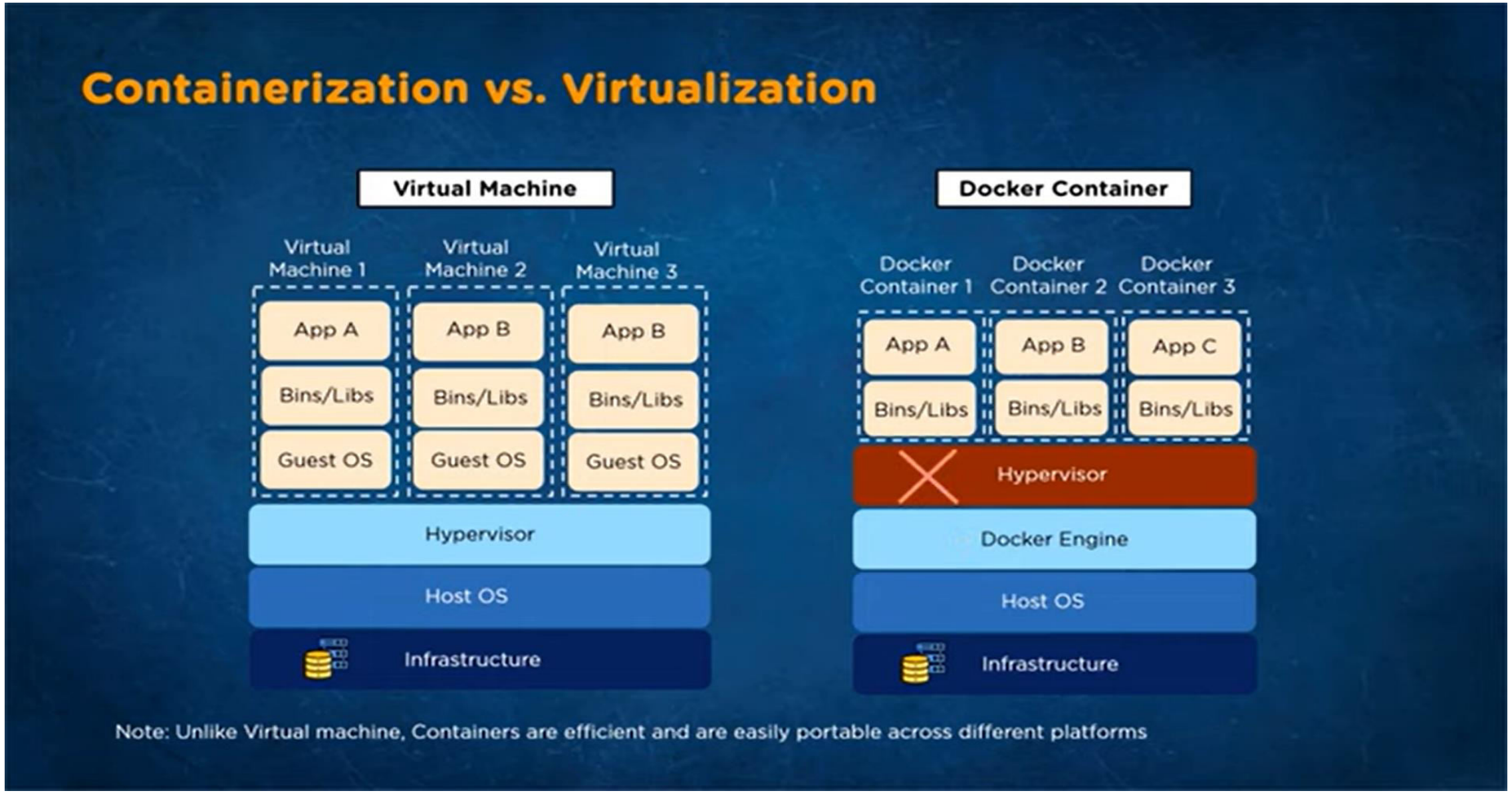
Docker


What is Docker Container?
Containerization includes all dependencies (frameworks, libraries, etc.) required to run an application in an efficient and bug-free manner
With Docker Containers, applications can work efficiently in different computer environments
Applications run in isolation
Secure
Occupies less space
Portable
Lightweight
Short boot-up time
Docker Containers

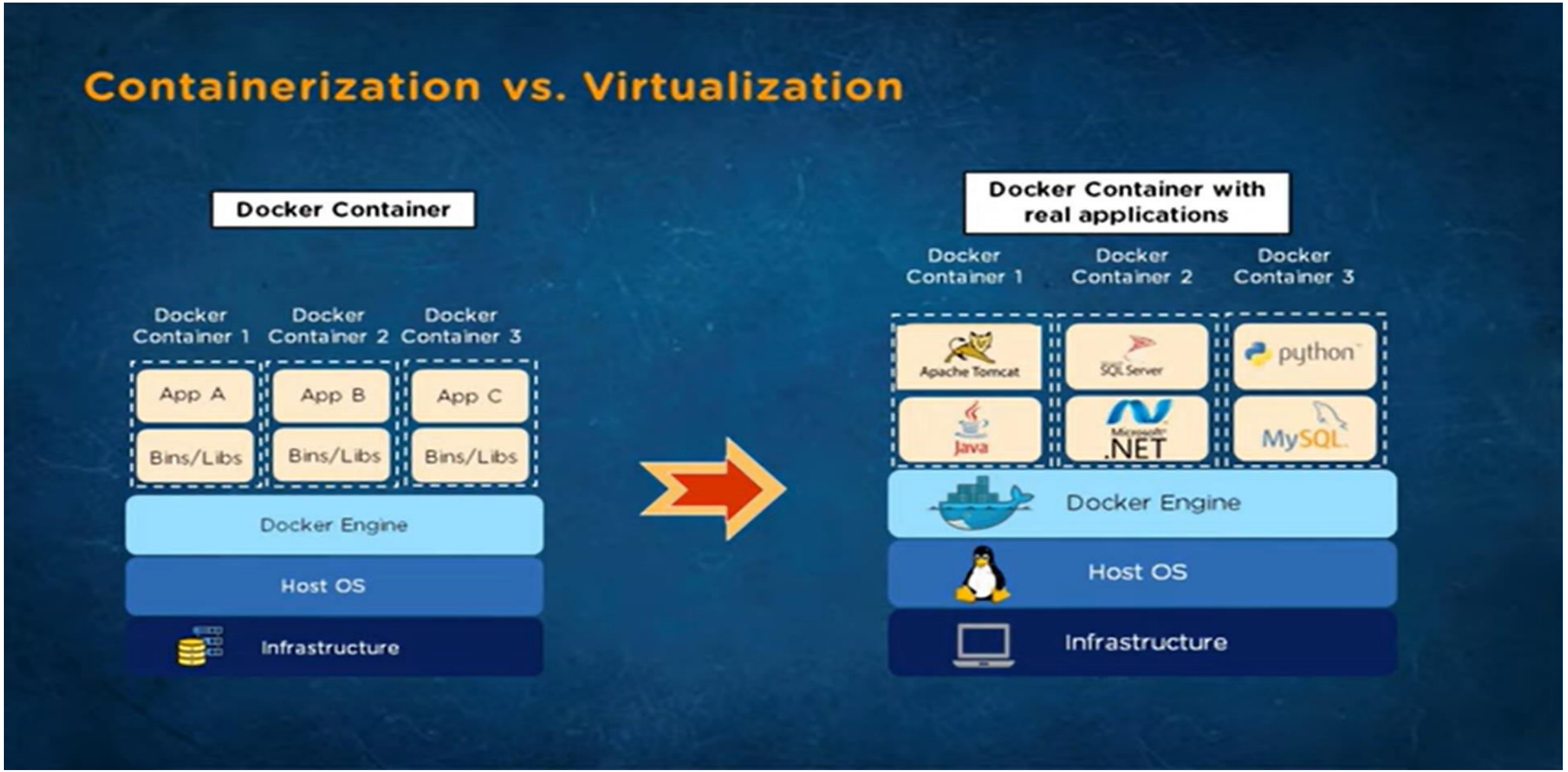
Docker


Containerization vs. Virtualization
Virtual Machine
Virtual Machine 1
Virtual Machine 2
Virtual Machine 3
App A
App B
App B
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host OS
Infrastructure
Docker Container
Docker Container 1
Docker Container 2
Docker Container 3
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Infrastructure
Note: Guest OS occupies more space and leads to unstable performance

Docker


Containerization vs. Virtualization
Virtual Machine
Virtual Machine 1
Virtual Machine 2
Virtual Machine 3
App A
App B
App B
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Host OS
Infrastructure
Docker Container
Docker Container 1
Docker Container 2
Docker Container 3
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Hypervisor
Docker Engine
Host OS
Infrastructure
Note: Unlike Virtual machine, Containers are efficient and are easily portable across different platforms

Docker


Containerization vs. Virtualization
Docker Container
Docker Container 1
Docker Container 2
Docker Container 3
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Infrastructure
Docker Container with real applications
Docker Container 1
Docker Container 2
Docker Container 3
Apache Tomcat
SQL Server
python
Java
.NET
MySQL
Docker Engine
Host OS
Infrastructure

Docker

Containerization vs. Virtualization

Virtual machine

Example: Bungalow

- Amenities (binary and library) cannot be shared with neighbours (applications)
- Cannot have multiple tenants (application)

Containers

Example: Apartment

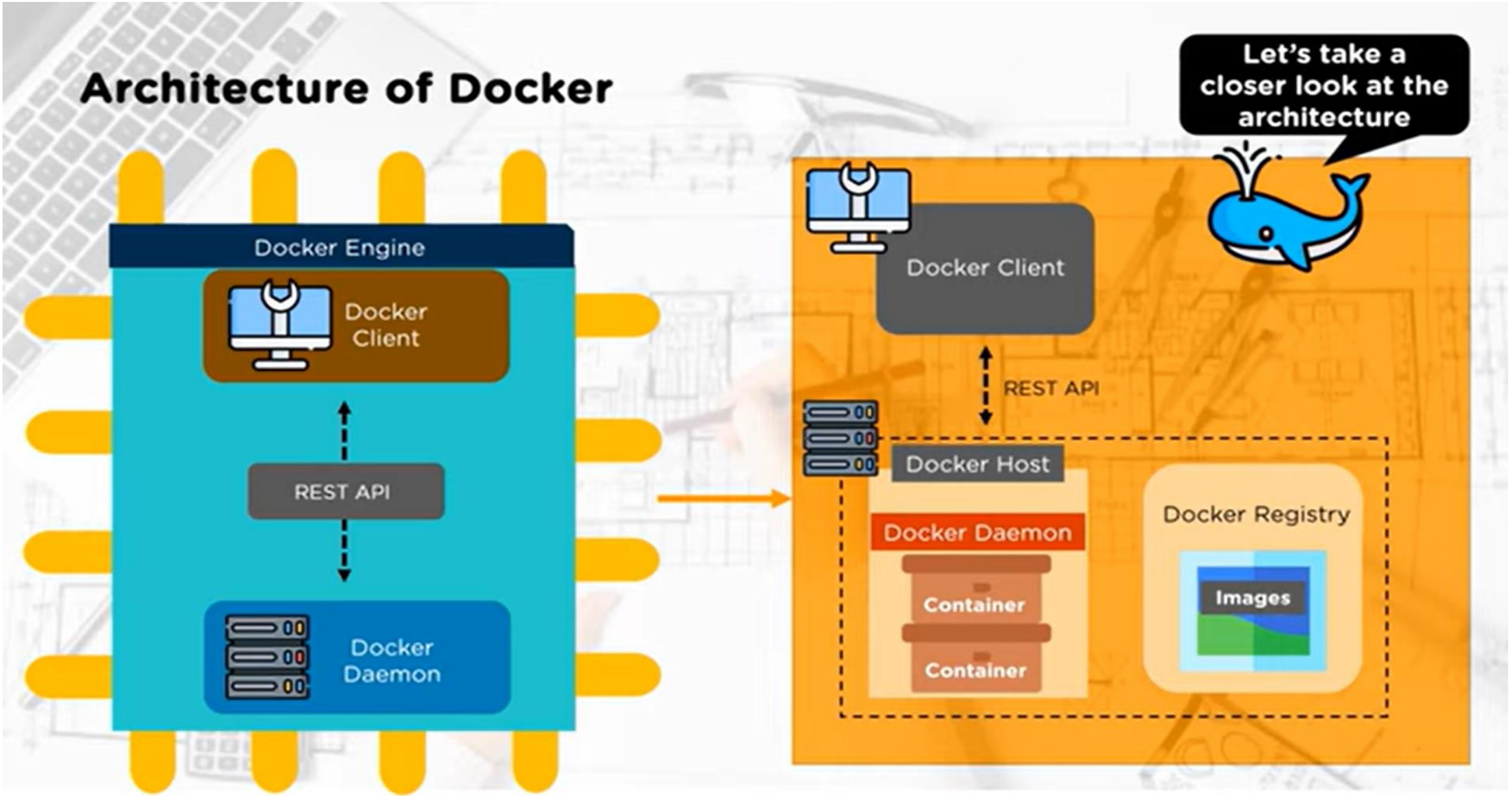
- Most amenities (binary and library) are shared with neighbours (applications)
- Can have multiple tenants (Applications)

Docker

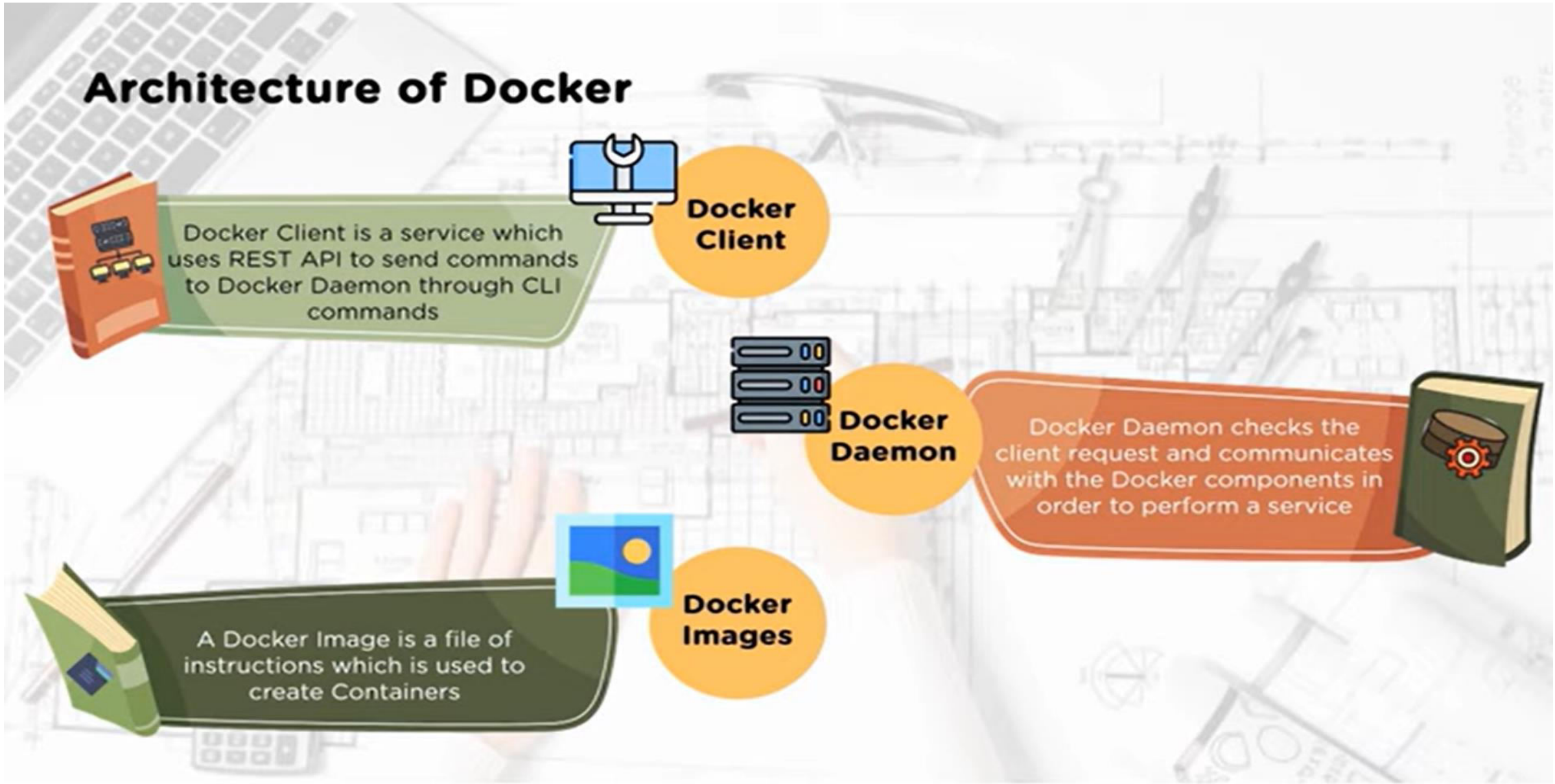
Containerization vs. Virtualization

	DOCKER CONTAINERS	VIRTUAL MACHINE
Space allocation	Data volumes can be shared and reused among multiple containers	Data volumes cannot be shared
Performance	They have a better performance as they are hosted in a single Docker engine	Running multiple virtual machines leads to unstable performance
Efficiency	High efficiency	Low efficiency
Capability	It can run multiple containers on a system	It can run only a limited number of VMs on a system
Portability	Easily portable across different platforms	Compatibility issues while porting across different platforms

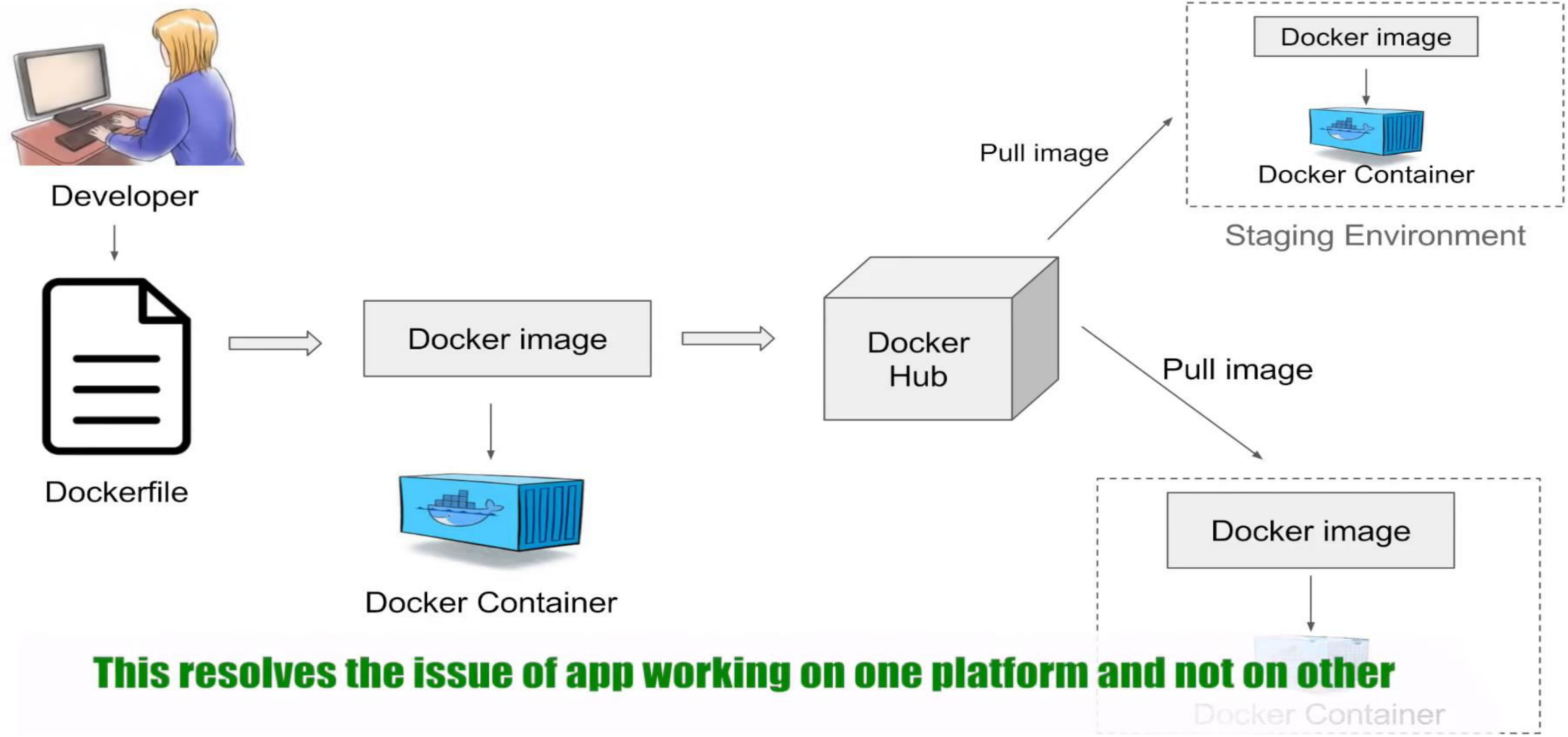
Docker


Architecture of Docker
Docker Engine
Docker Client
REST API
Docker Daemon
Let's take a closer look at the architecture
Docker Client
REST API
Docker Host
Docker Daemon
Container
Container
Docker Registry
Images

Docker


Architecture of Docker
Docker Client
Docker Client is a service which uses REST API to send commands to Docker Daemon through CLI commands
Docker Daemon
Docker Daemon checks the client request and communicates with the Docker components in order to perform a service
Docker Images
A Docker Image is a file of instructions which is used to create Containers

Docker


Developer
Dockerfile
Docker image
Docker Container
Docker Hub
Pull image
Docker image
Docker Container
Staging Environment
Pull image
Docker image
Docker Container
This resolves the issue of app working on one platform and not on other

Docker

What is Docker?

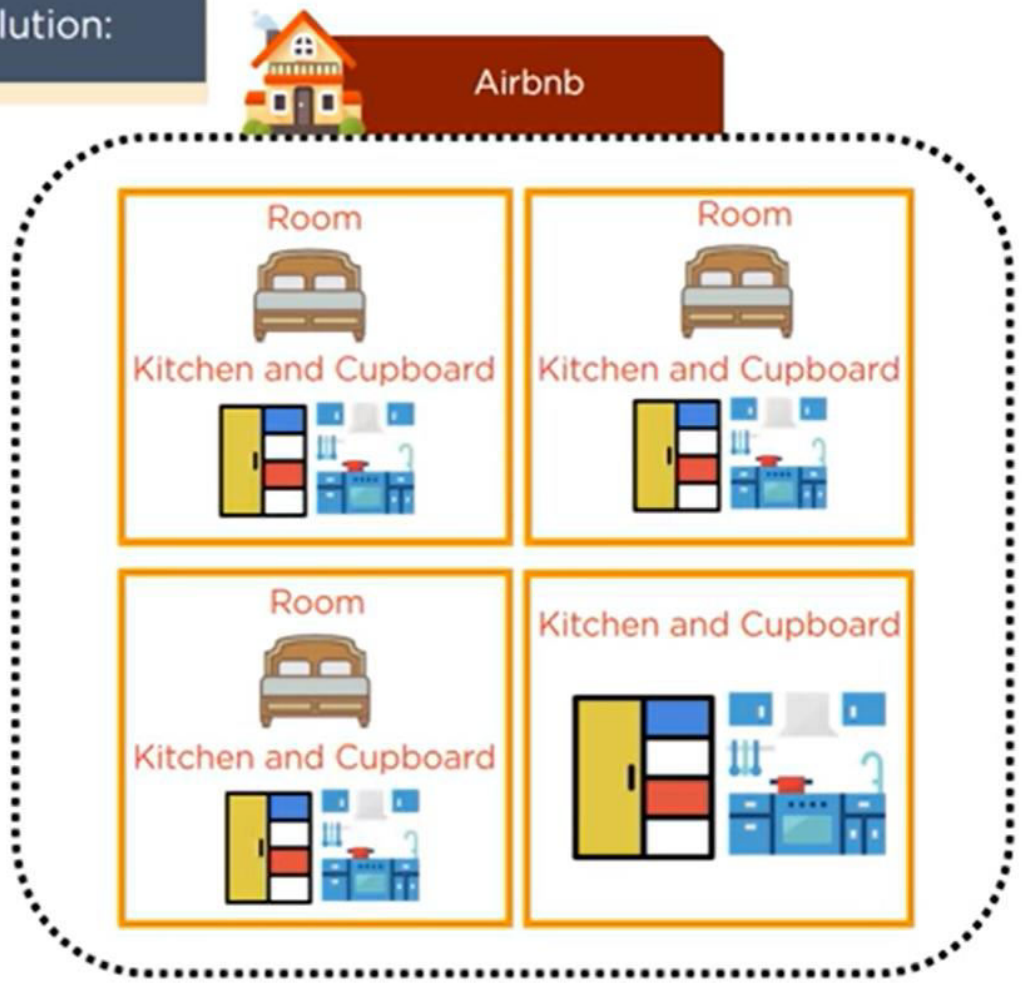
Let's take an example where you plan to rent a house in Airbnb

Because every individual has different preferences when it comes to the cupboard and the kitchen usage

Docker

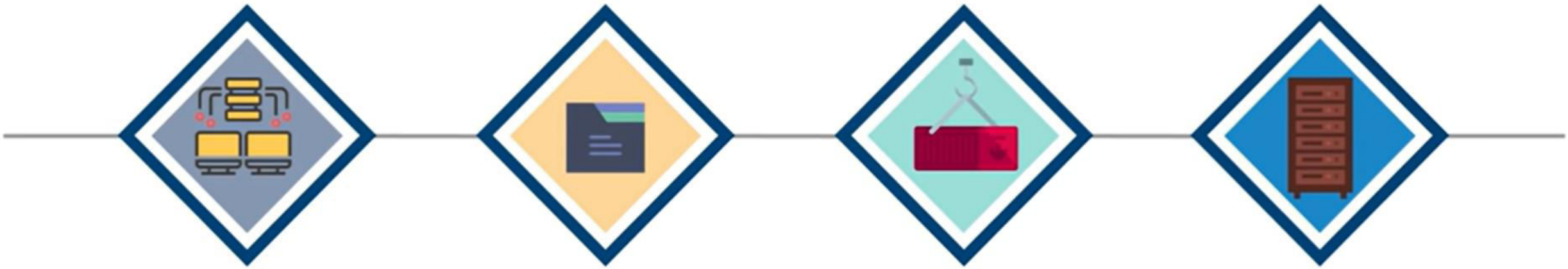
What is Docker?

Solution:



Docker

Components of Docker



Docker Client and Server

Docker Images

Docker Containers

Docker Registry

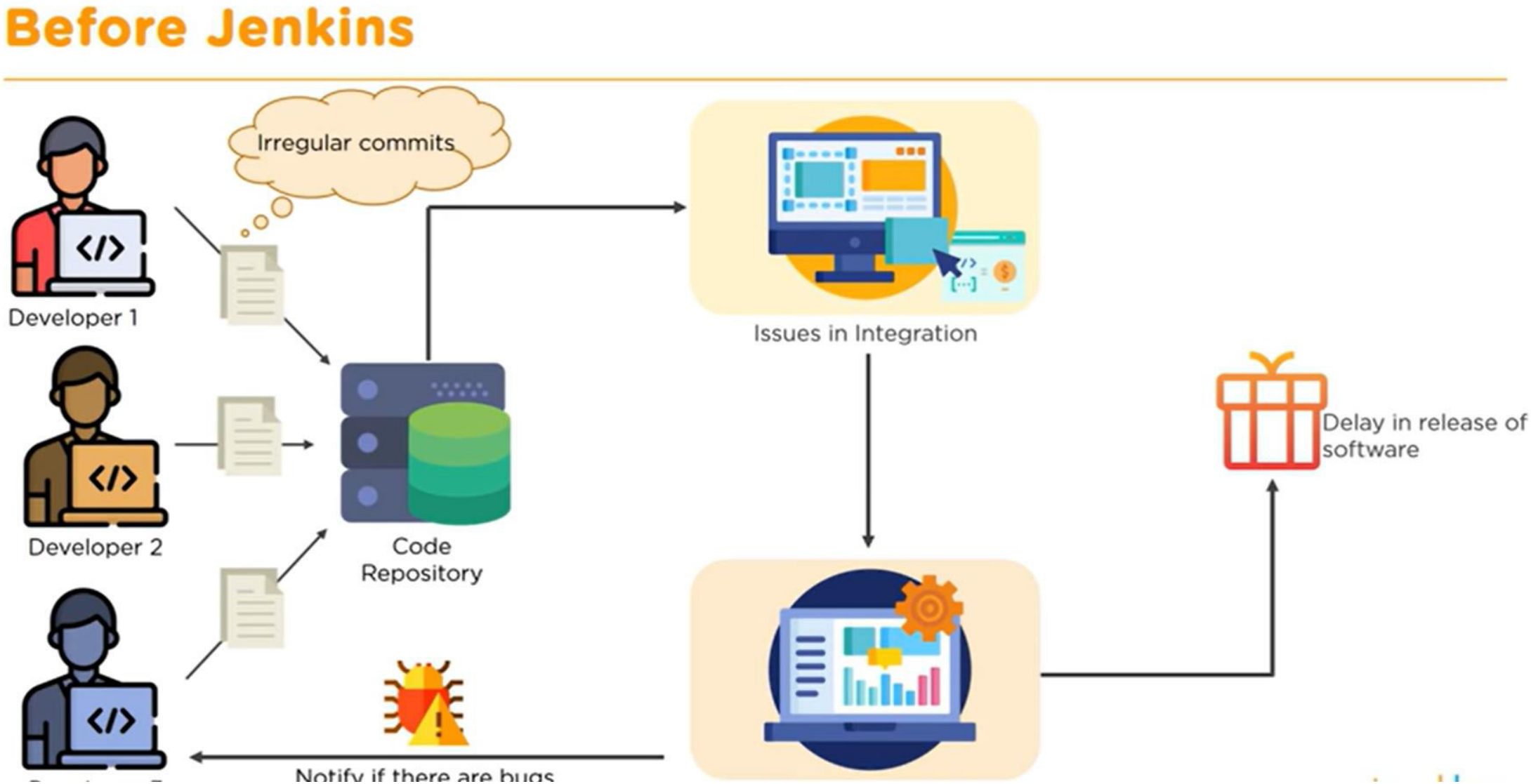
Jenkins

What's in it for you?

- Before Jenkins
- Issues before Jenkins
- What is Jenkins?
- What is Continuous Integration?

- Continuous Integration Tools
- Features of Jenkins
- Jenkins Architecture
- Jenkins Case Study

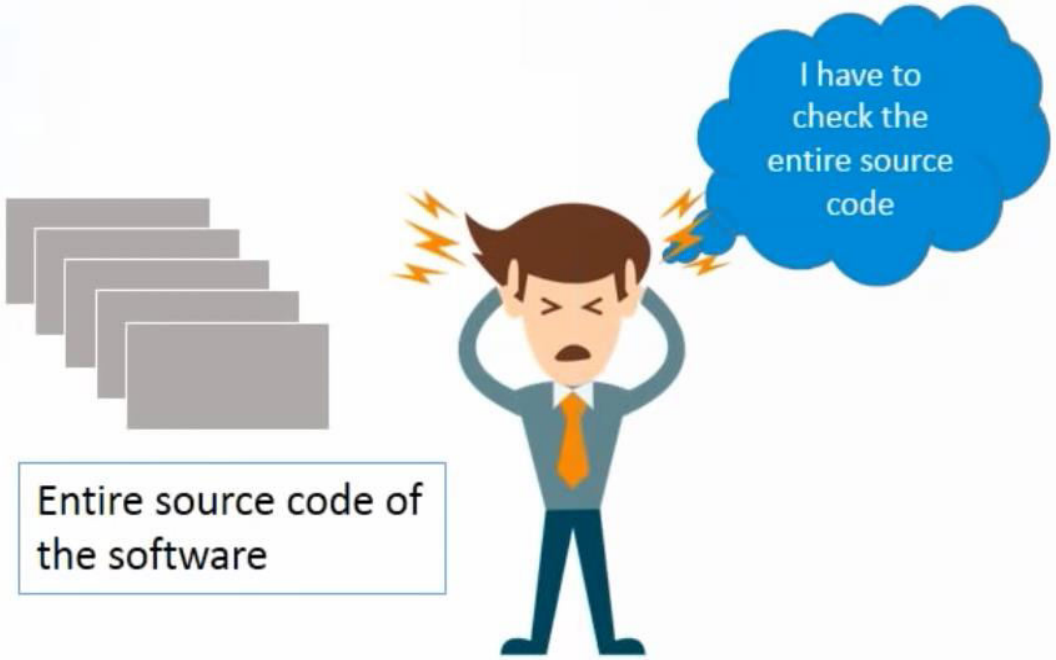
Jenkins


Before Jenkins
Irregular commits
Developer 1
Developer 2
Code Repository
Issues in Integration
Delay in release of software
Notify if there are bugs

Jenkins

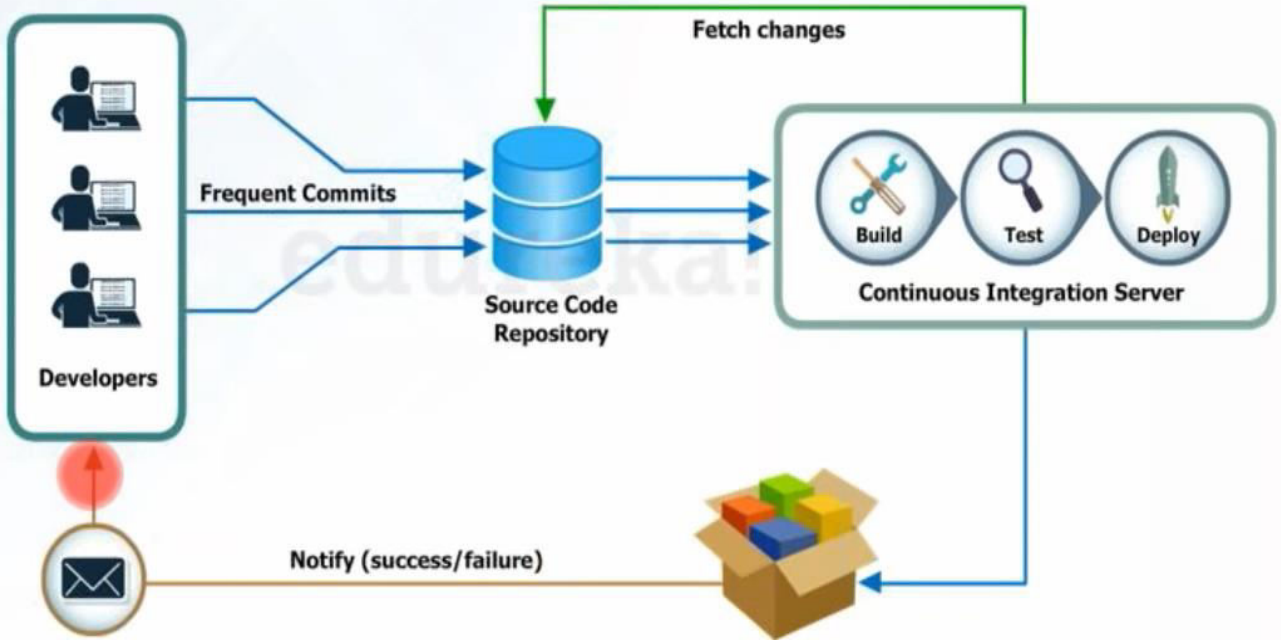
Developers have to wait till the complete software is developed for the test results.

If the test fails then locating and fixing bugs is very difficult. Developers have to check the entire source code of the software.



Jenkins

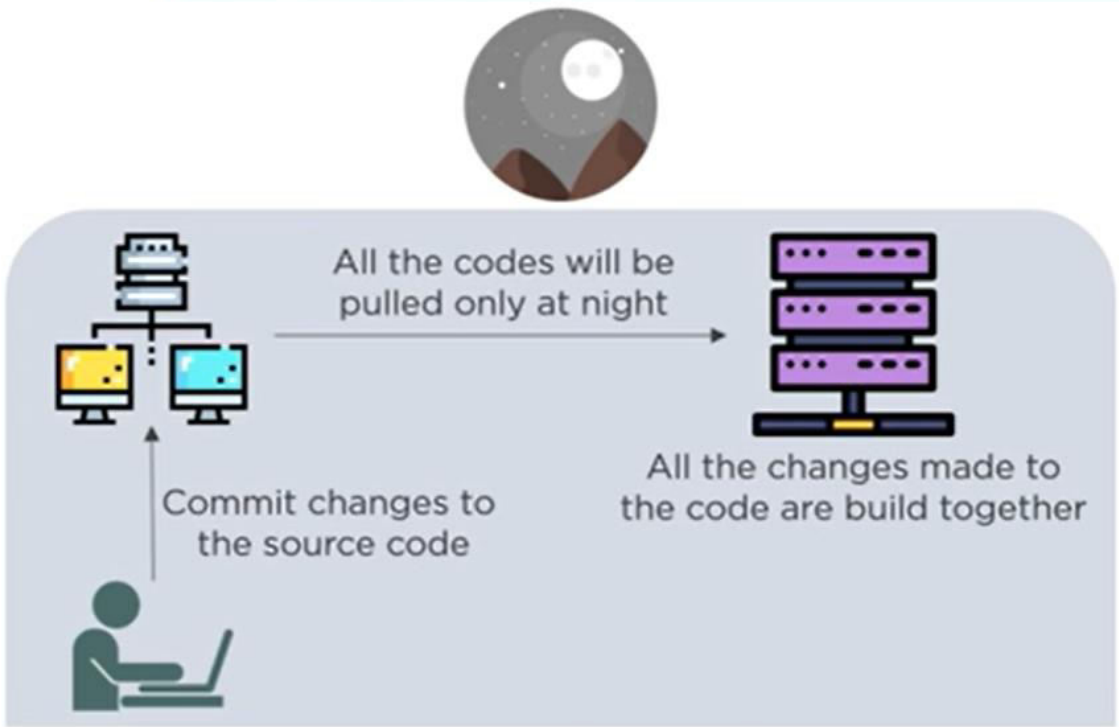
- Since after every commit to the source code an auto build is triggered and then it is automatically deployed on the test server
- If the test results shows that there is a bug in the code then the developers only have to check the last commit made to the source code
- This also increases the frequency of new software releases
- The concerned teams are always provided with the relevant feedback



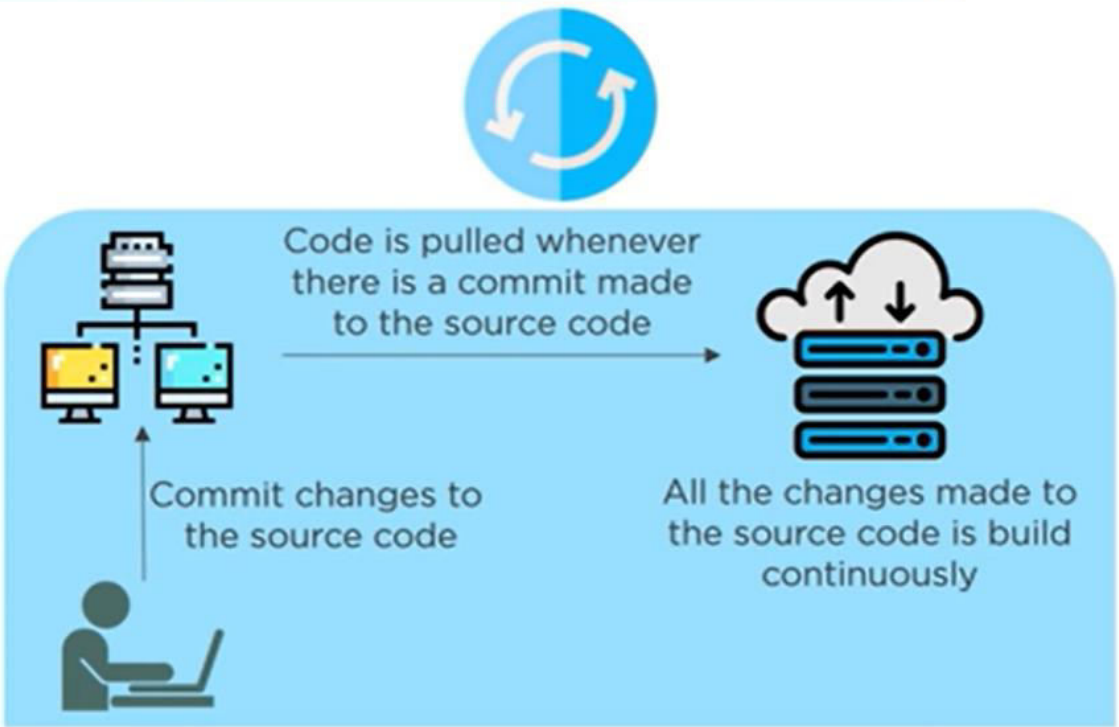
Jenkins

What is Jenkins?

Jenkins is a Continuous Integration tool that allows continuous development, test and deployment of newly created codes



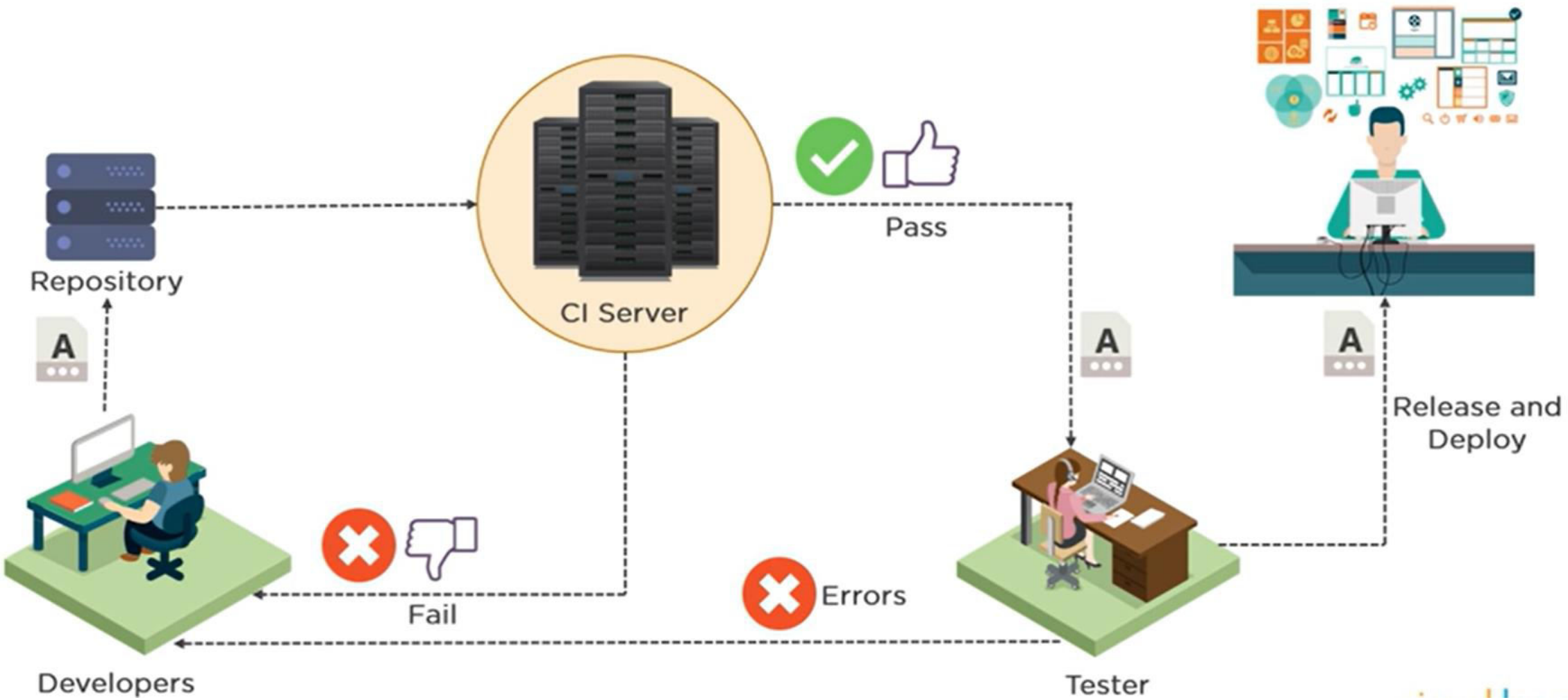
Nightly build and integration



Continuous build and Integration

Jenkins

What is Continuous Integration?

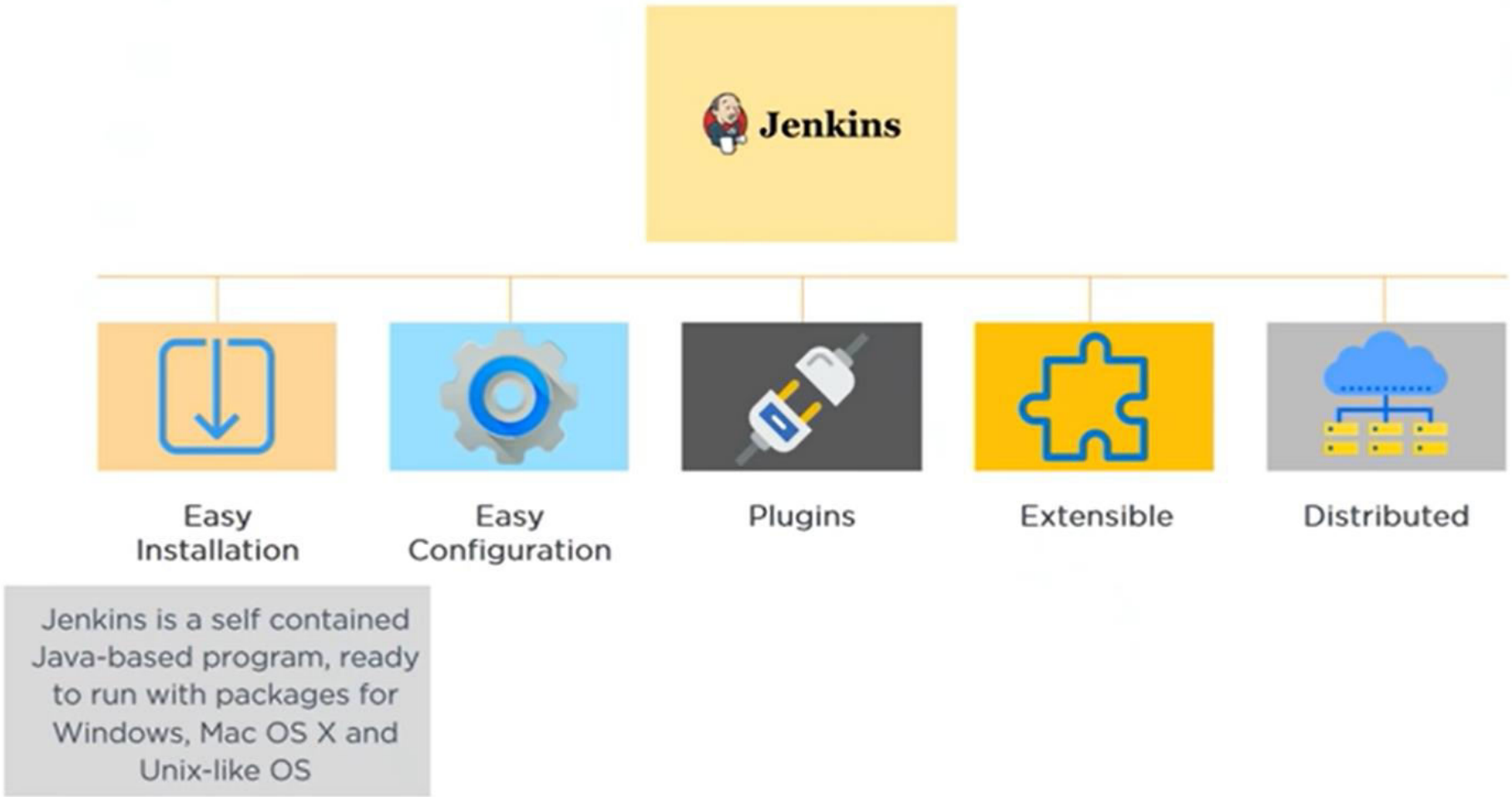


Jenkins

Continuous Integration Tools

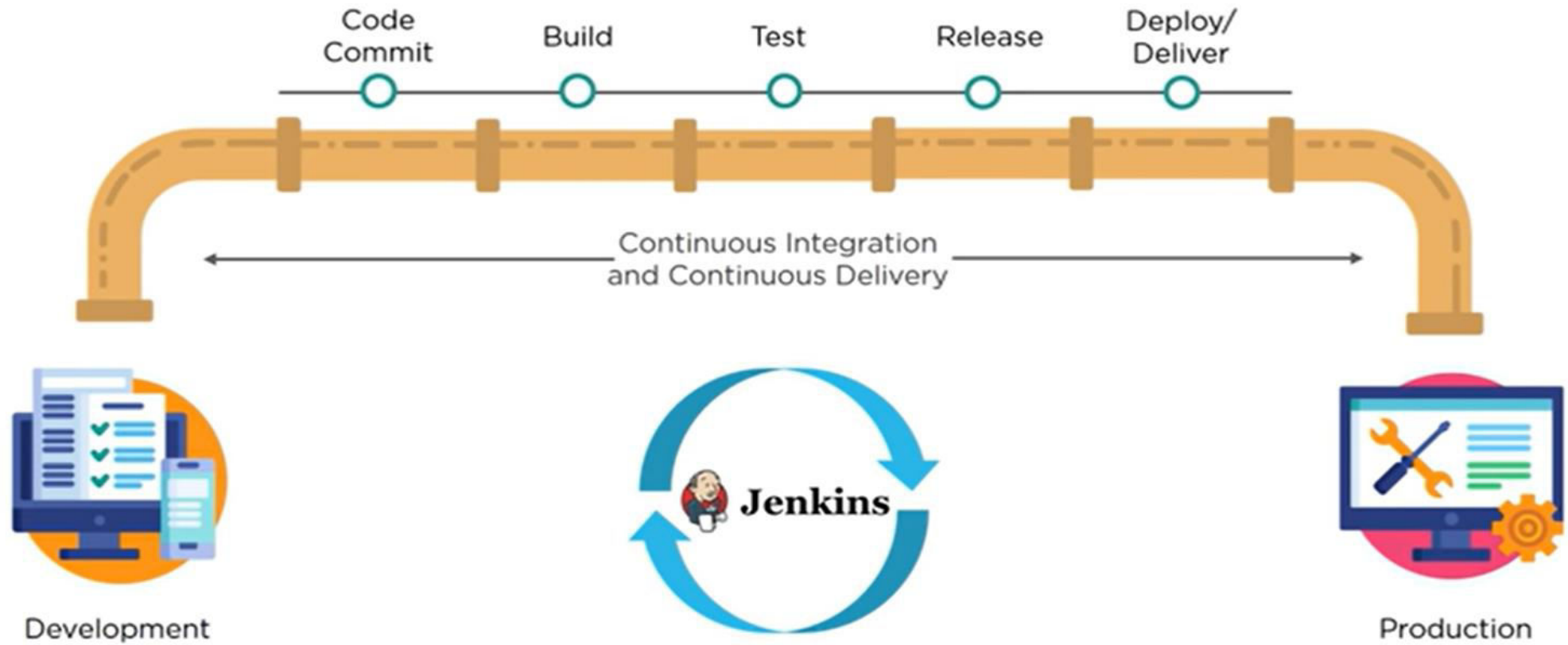
Jenkins

Features of Jenkins



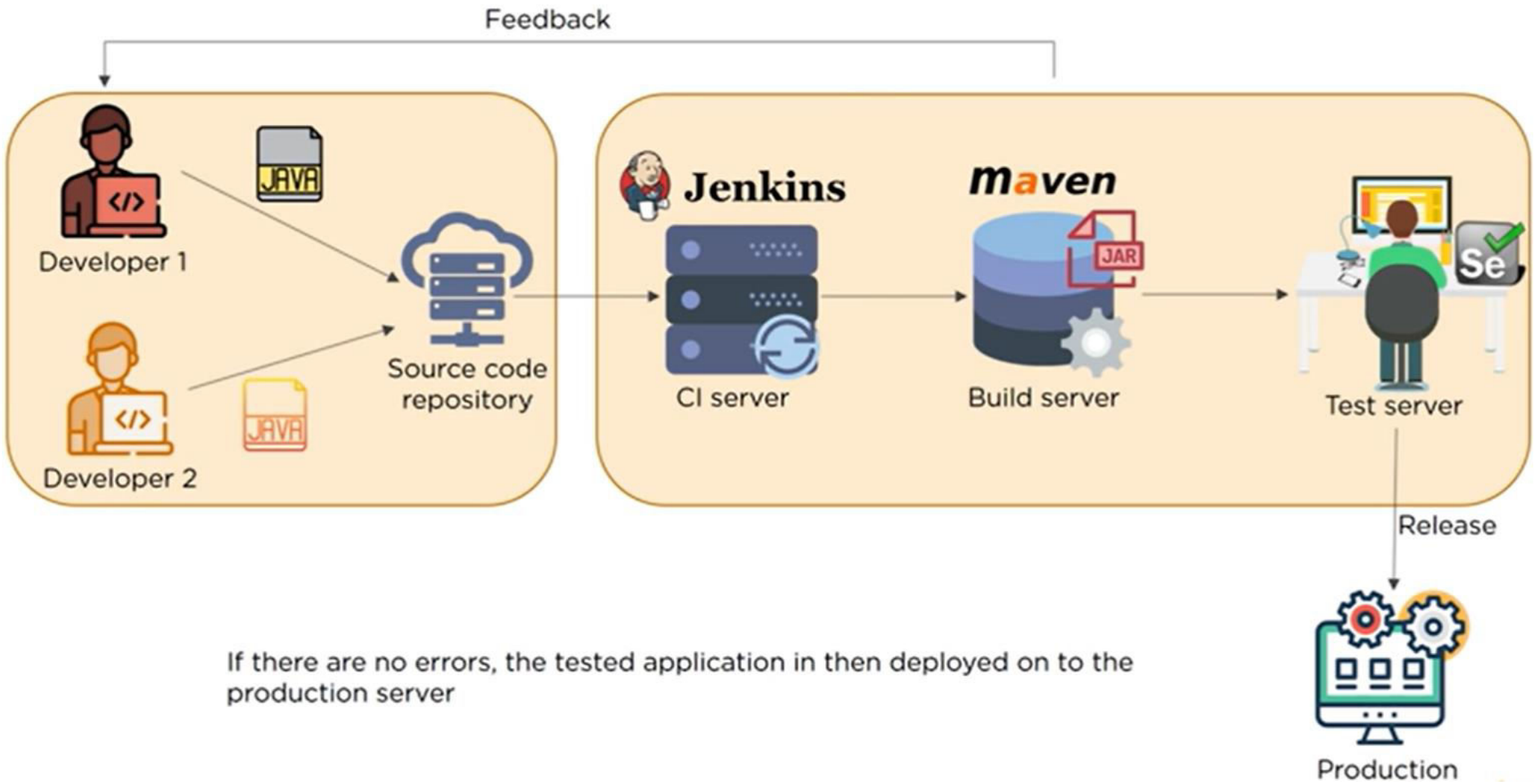
Jenkins

Jenkins Pipeline



Jenkins

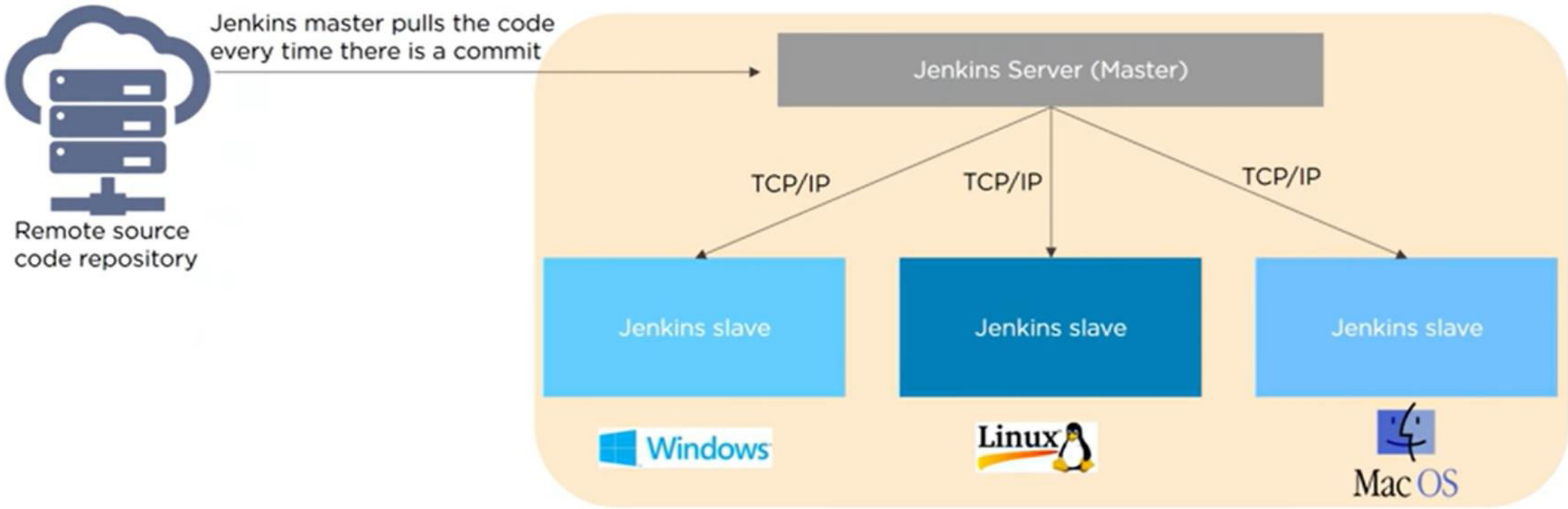
Jenkins Architecture



If there are no errors, the tested application in then deployed on to the production server

Jenkins

Jenkins Master-Slave Architecture



- Jenkins master distributes its workload to all the slaves
- On request from Jenkins master, the slaves carry out builds and tests and produce test reports

Jenkins

Jenkins Case Study

Automotive systems are becoming more sophisticated and complex, so the focus for automotive manufacturers is shifting from hardware to software

Many vehicle features, capabilities and performance enhancements are being implemented in software

Jenkins

Jenkins Case Study

BOSCH

BOSCH found a growing need to help its software engineers produce and deliver higher quality software faster

CHALLANGE

Manage and streamline the development of increasingly complex automotive software by adopting CI and CD practices to shorten the entire development and delivery process

Jenkins

Jenkins Case Study

BOSCH

CloudBees
The Enterprise Jenkins Company

RESULTS

- 3 day build process reduced to less than 3 hours
- Large scale deployment kept on track by expert support
- Visibility and transparency improved with Jenkins Operations support

YAML

YAML is a human-readable data serialization language that is often used for writing configuration files. Depending on whom you ask, YAML stands for yet another markup language or YAML ain't markup language (a recursive acronym), which emphasizes that YAML is for data, not documents.

Many developers consider it easier to learn than JSON because it's written using natural language. YAML is a superset of JSON. It was developed around the same time to handle more kinds of data and offer a more complex but still readable syntax.

YAML is a popular programming language because it is designed to be easy to read and understand. It can also be used in conjunction with other programming languages. Because of its flexibility and accessibility

YAML

Differences

XML

```xml
<People>
    <Person>
        <name>Bob</name>
        <age>21</age>
        <address>Boca Raton, Florida(FL), 33432</address>
    </Person>
</People>
```

JSON

```json
{
    "People": [
        {
            "name": "Bob",
            "age": 21,
            "address": "Boca Raton, Florida(FL), 33432"
        }
    ]
}
```

YAML

```yaml
People:
  - name: Bob
    age: 21
    address: Boca Raton, Florida(FL), 33432
```

YAML

Key Value Pairs

Key Value Pair

```
Animal: Dog
Fruit: Apple
Meal: "Vegetable Lasagna"
```

YAML

Arrays / Lists

Arrays / Lists

```yaml
Drinks:
  - Smoothie
  - Water
Meals:
  - "Vegetable Lasagna"
  - "Tomato Soup"
  - Burger
```

YAML

Directories / Maps

Directories / Maps

```yaml
Burger:
  Calories: 200
  Price: 15
Smoothie:
  Calories: 80
  Price: 5
```

YAML

Combining everything together

Arrays with Directories

```yaml
Drinks:
  - Smoothie:
      Calories: 80
      Price: 5
Meals:
  - Burger:
      Calories: 200
      Price: 15
```

YAML

Spacing

Key Value Pair

```yaml
Animal: Dog
Fruit: Apple
Meal: "Vegetable Lasagna"
```

Directories / Maps

```yaml
Burger:
  Calories: 200
  Price: 15
Smoothie:
  Calories: 80
  Price: 5
```

Arrays / Lists

```yaml
Drinks:
  - Smoothie
  - Water
Meals:
  - "Vegetable Lasagna"
  - "Tomato Soup"
  - Burger
```

Lab

YAML

Comments

Arrays with Directories

```yaml
# This is single line comment.
Drinks:
  - Smoothie:
      Calories: 80
      Price: 5
Meals:
  - Burger:
      Calories: 200
      Price: 15
```

Kubernetes

Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management. Originally designed by Google, the project is now maintained by the Cloud Native Computing Foundation. The name Kubernetes originates from Greek, meaning 'helmsman' or 'pilot'.

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications

Kubernetes automates operational tasks of container management and includes built-in commands for deploying applications, rolling out changes to your applications, scaling your applications up and down to fit changing needs, monitoring your applications, and more—making it easier to manage applications.

Kubernetes is the most popular container orchestration platform and has become an essential tool for DevOps teams. Application teams can now deploy containerized applications to Kubernetes clusters, which can run either on-premises or in a cloud environment.

Kubernetes creates and manages containers on the cloud-based server systems. Kubernetes helps DevOps teams to reduce the burden of infrastructure by letting containers operate on different machines/environments without breakdowns

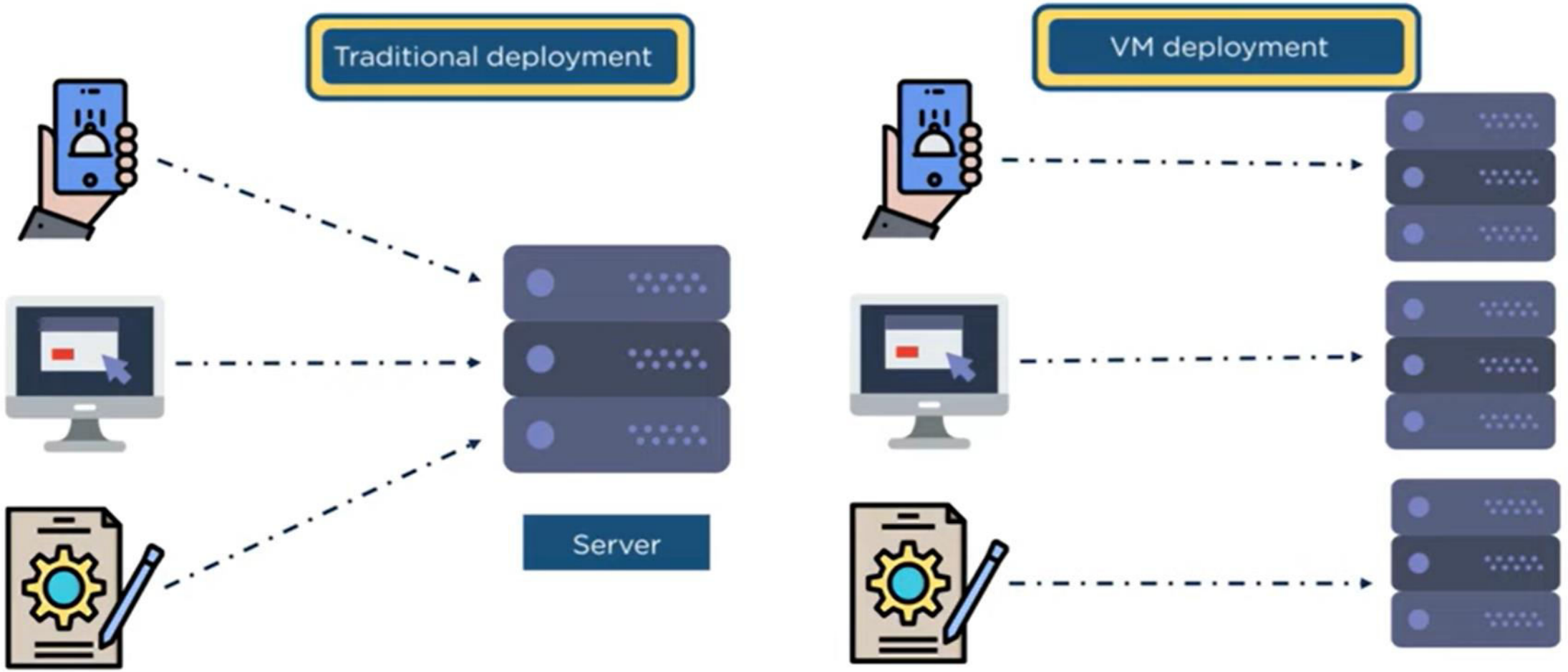
Kubernetes

What's in it for you?

- Before Kubernetes
- What is Kubernetes?
- Benefits of Kubernetes
- Kubernetes Architecture and Working

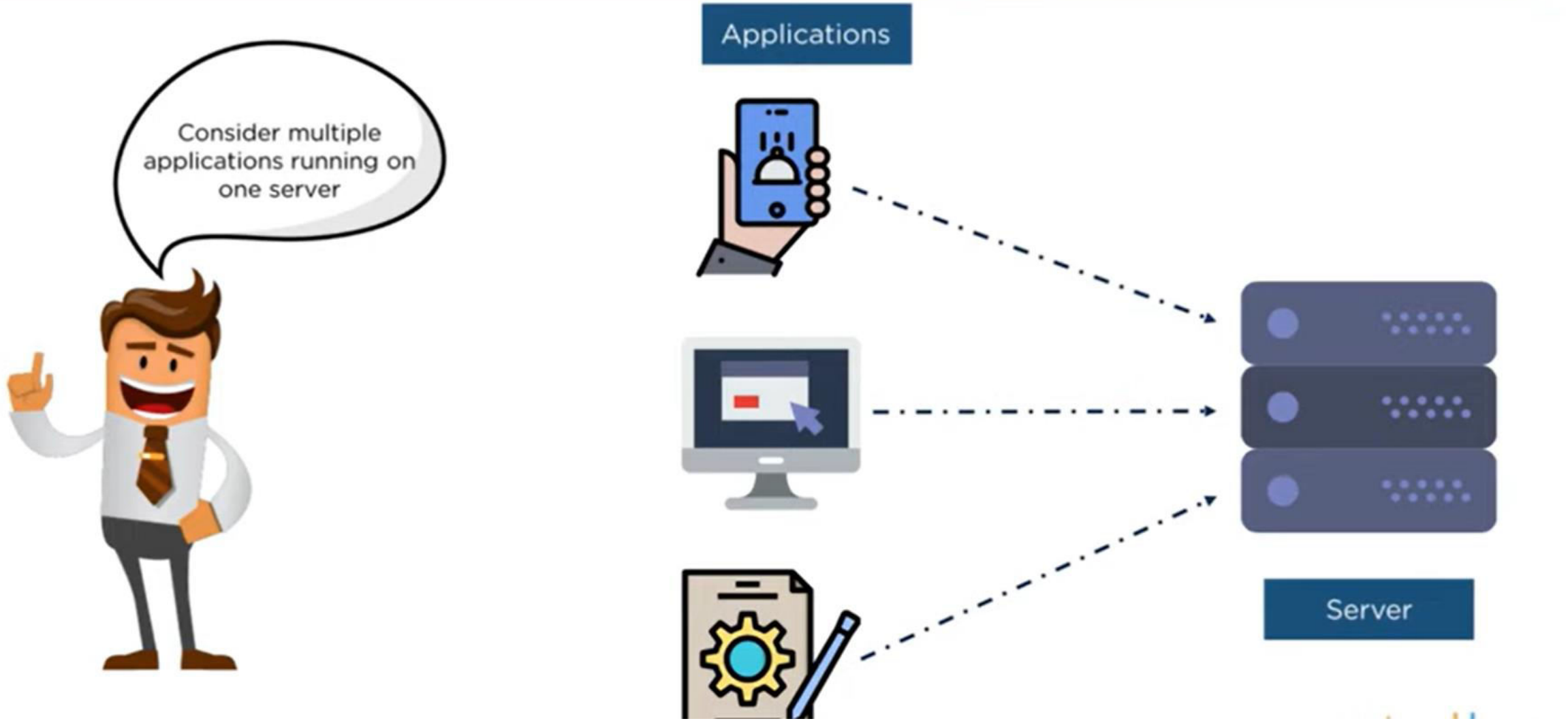
Kubernetes

Before Kubernetes



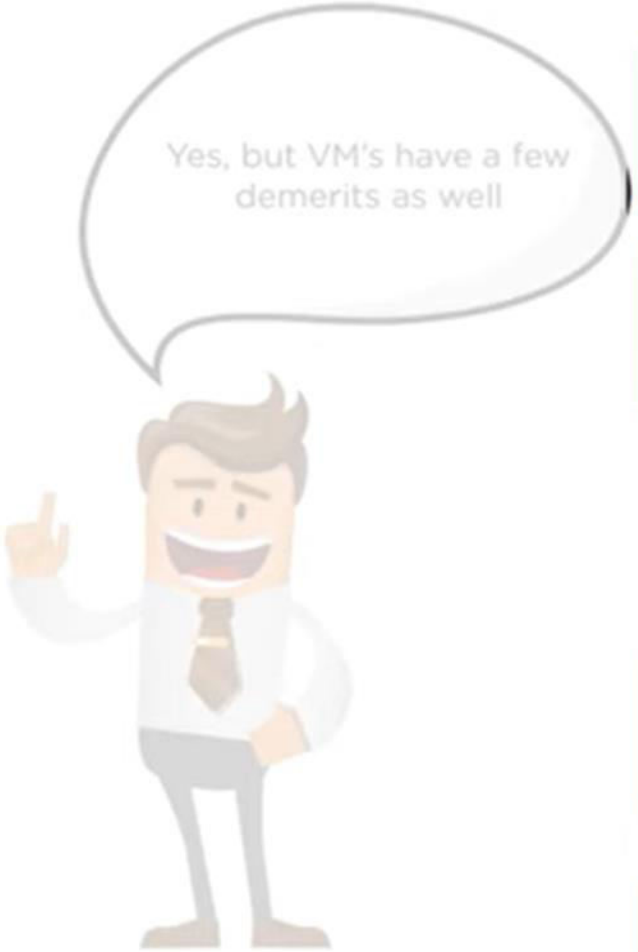
Kubernetes

Before Kubernetes – Traditional Deployment



Kubernetes

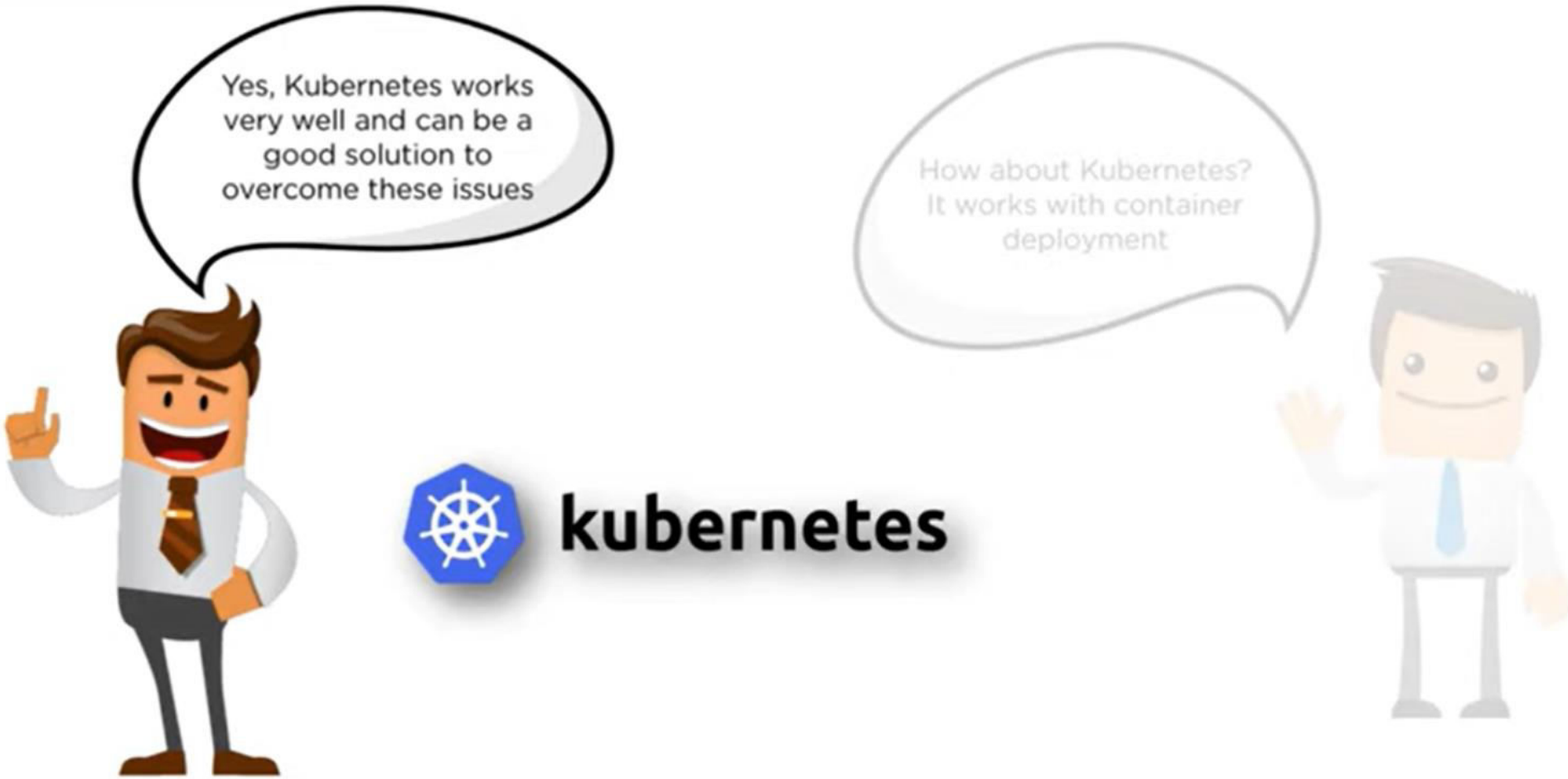
Before Kubernetes – Virtualization Deployment



High cost of implementation	Hardware and software components are required by the VM's that increases the cost of implementation
Security risk	The chances of data breach while using virtualization is common
Availability issue	Assets are unavailable most of the time because the resources are being used by other applications
Limitations and restrictions	Not every application or server can work in a virtualization environment. Some may require hybrid machines to work with
Time-consuming	It is a very time-consuming process. This is because extra steps are followed to generate the desired results

Kubernetes

After Kubernetes



Kubernetes

Virtual Machine vs Kubernetes

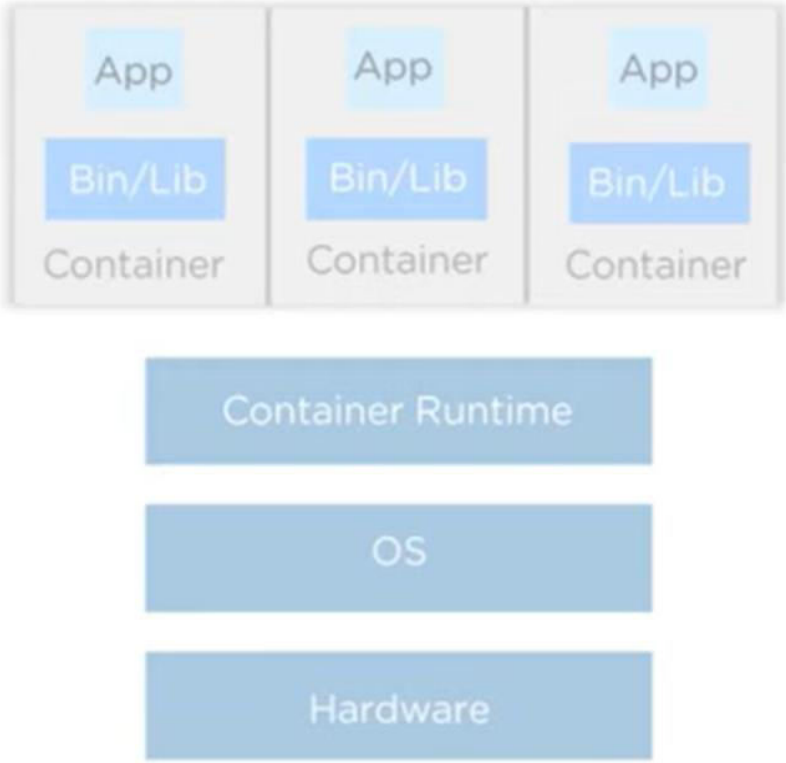
Major differences are:

Kubernetes

Virtual Machine vs Kubernetes

VM – They have less isolation when compared to Kubernetes, and hence, security risks are higher

Kubernetes - They have better isolation properties to share the OS among various applications

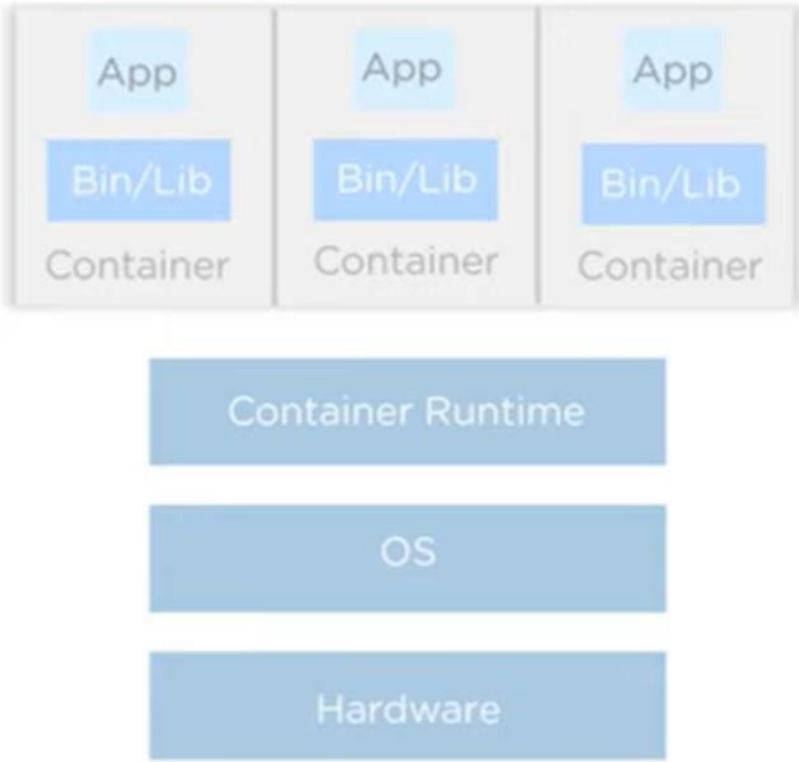


Kubernetes

Virtual Machine vs Kubernetes

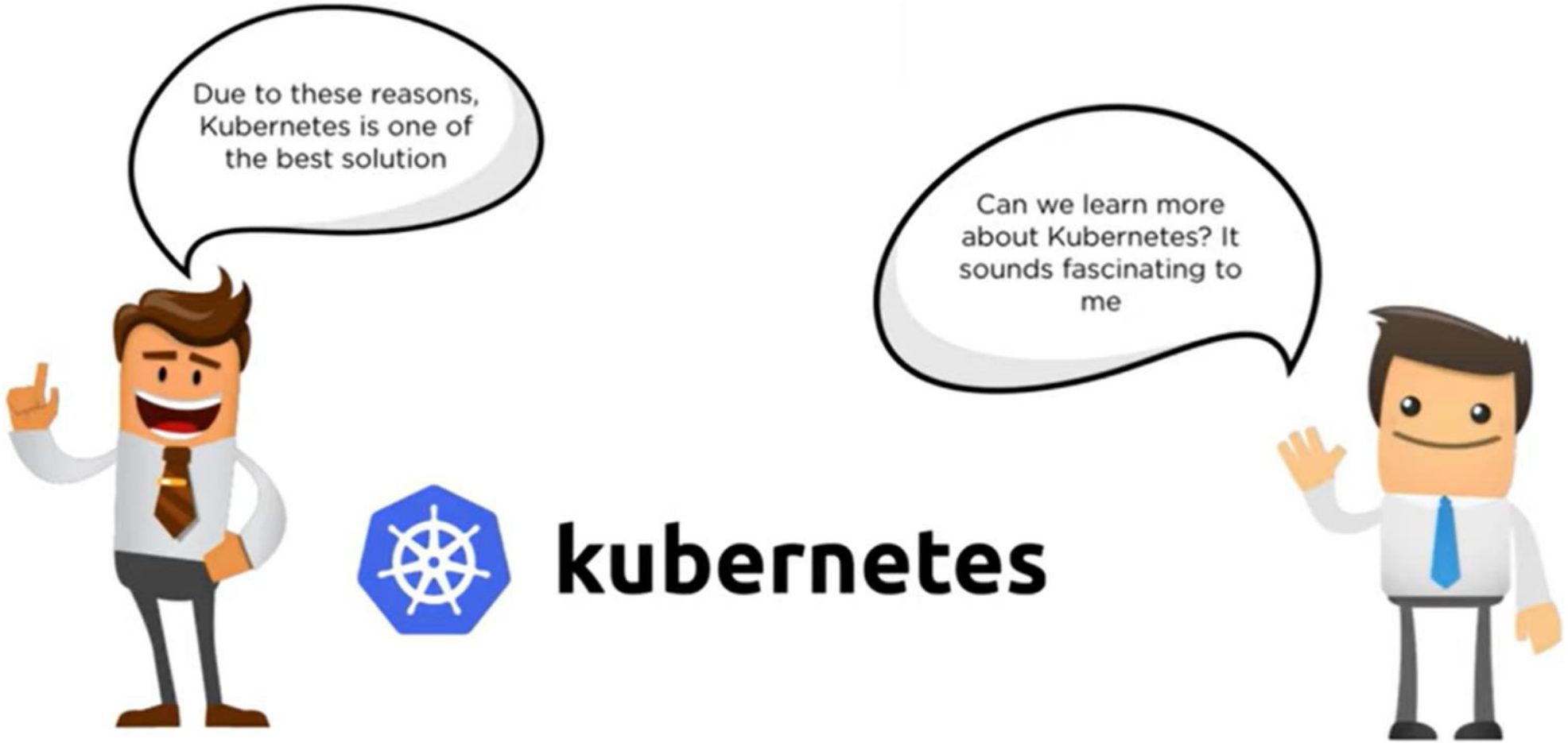
VM – They are portable but not as much as Kubernetes

Kubernetes – They are easily portable and could work on any platform or environment



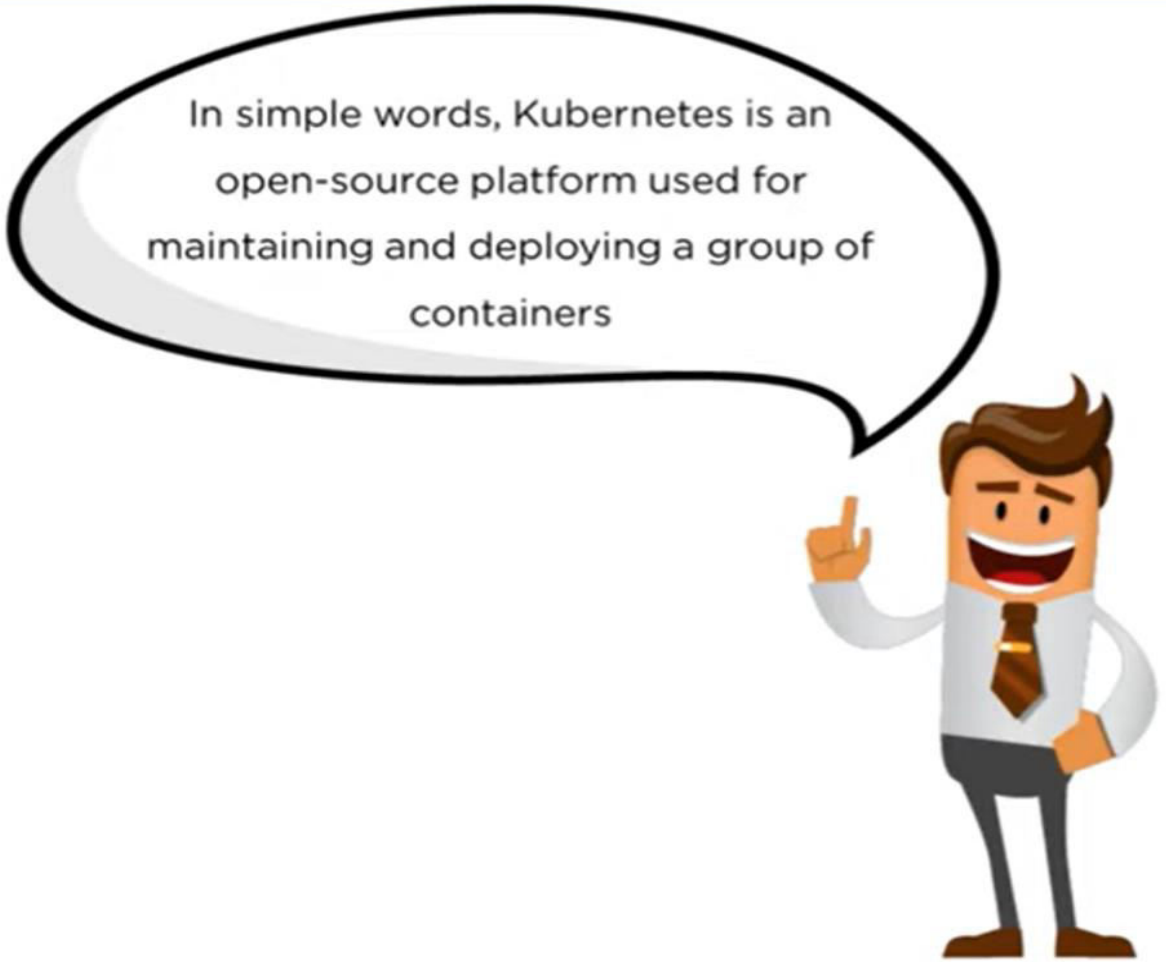
Kubernetes

Kubernetes Era



Kubernetes

What is Kubernetes?



kubernetes

Kubernetes

What is Kubernetes?

In practice, Kubernetes is most commonly used alongside Docker for better control and implementation of containerized applications

Note: Containerized Applications means bundling an application together with all its files, libraries, and packages required for it to run reliably and efficiently on different platforms

Kubernetes

Did you know?

Kubernetes was introduced as a project at Google, and it was a successor of Google Borg

It was started in 2014 because architecture of Kubernetes made it convenient for applications to run on the cloud

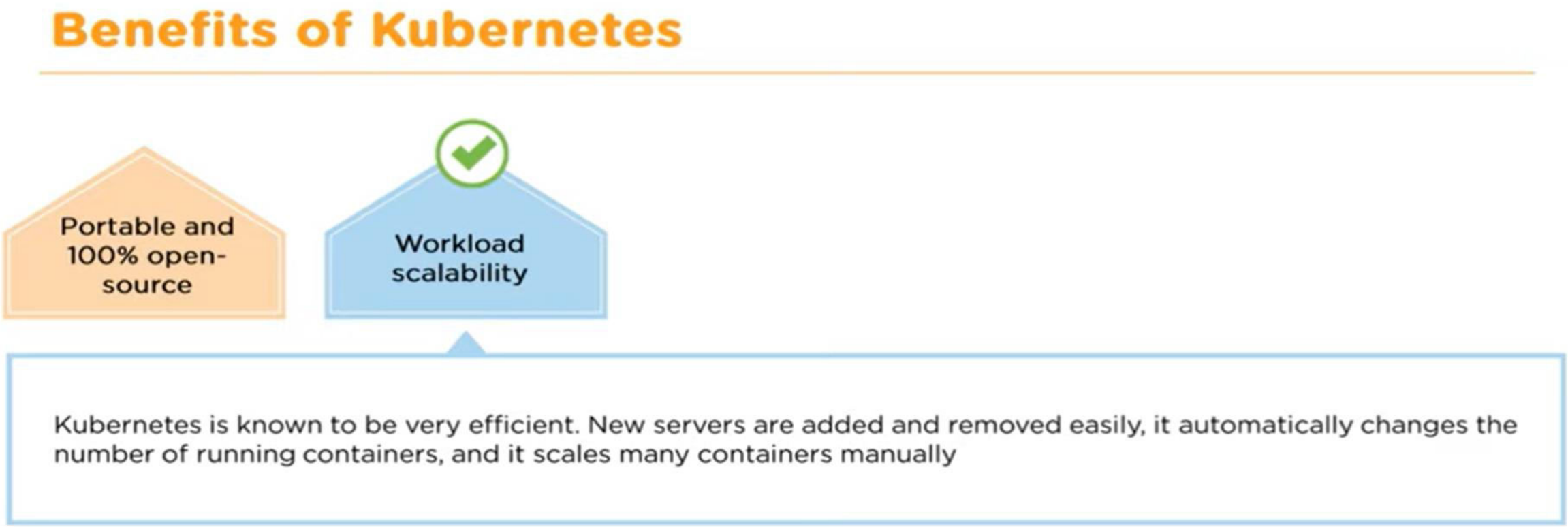
Cloud-Native Computing Foundation has currently maintained Kubernetes

Kubernetes

Benefits of Kubernetes

Kubernetes is compatible across several platforms. It is 100% open source project and thus, provides greater flexibility

Kubernetes


Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
Kubernetes is known to be very efficient. New servers are added and removed easily, it automatically changes the number of running containers, and it scales many containers manually

Kubernetes

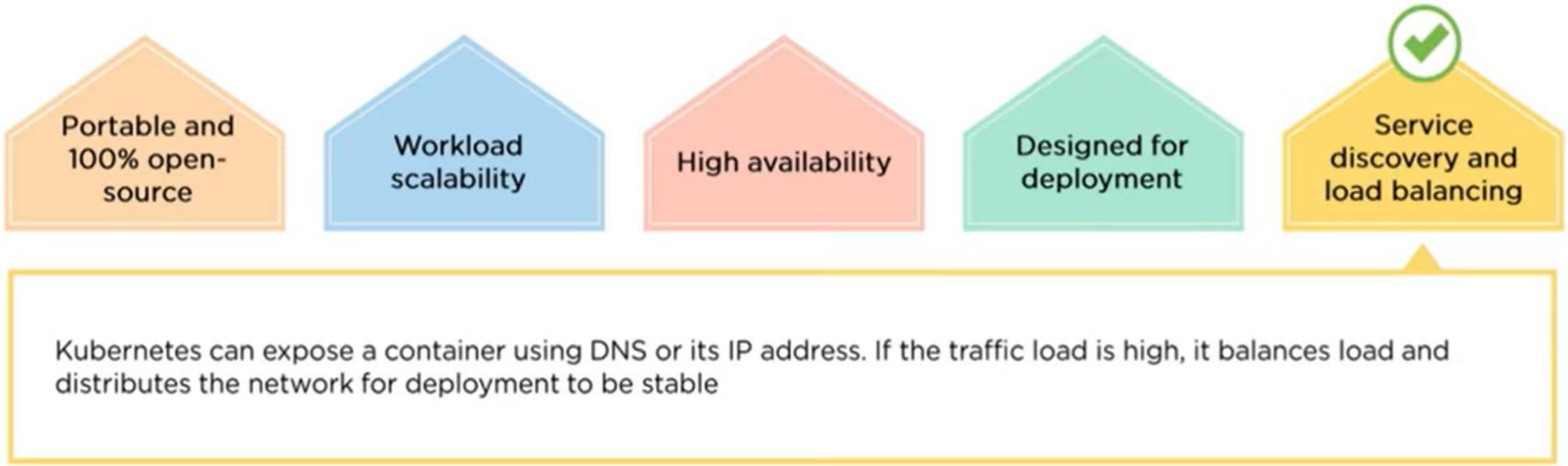
Benefits of Kubernetes

Kubernetes

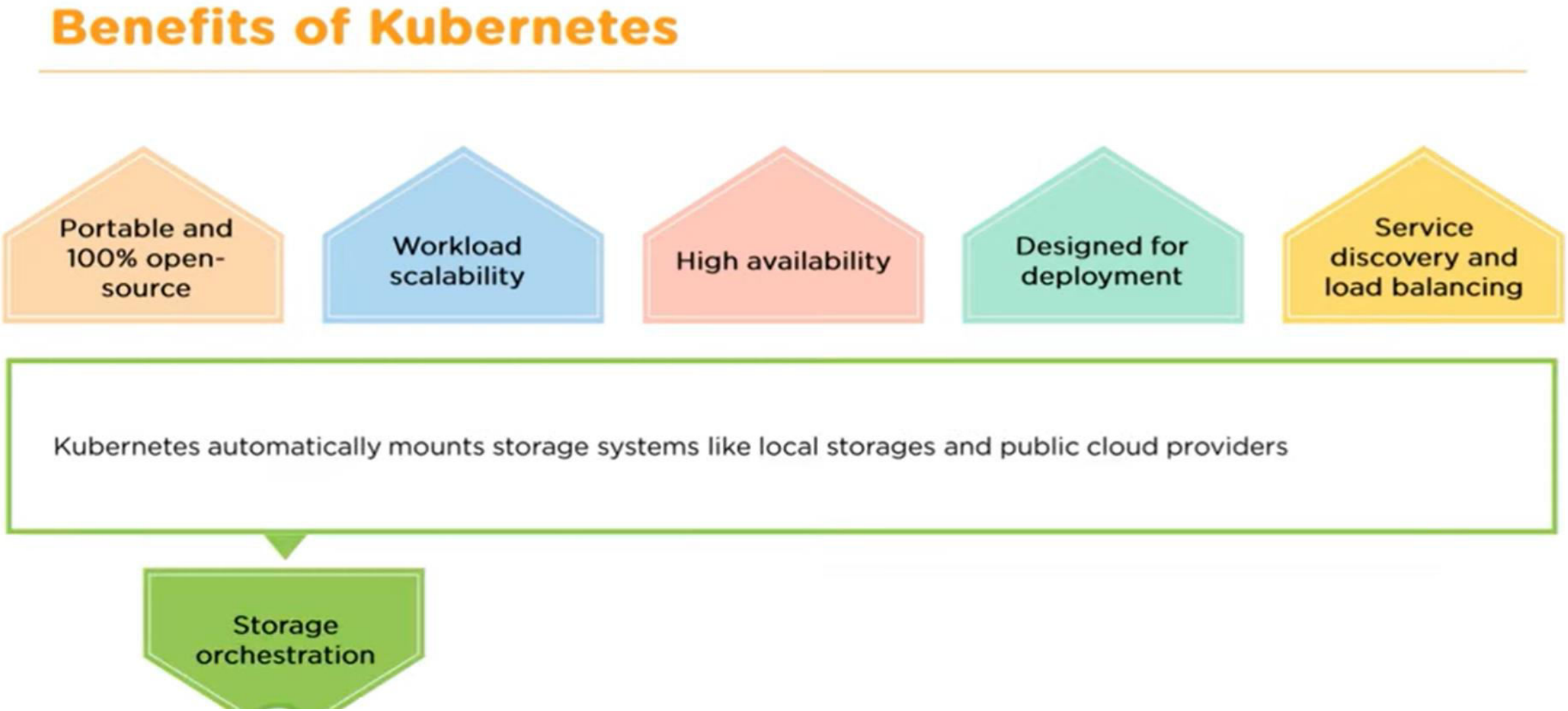
Benefits of Kubernetes

Kubernetes

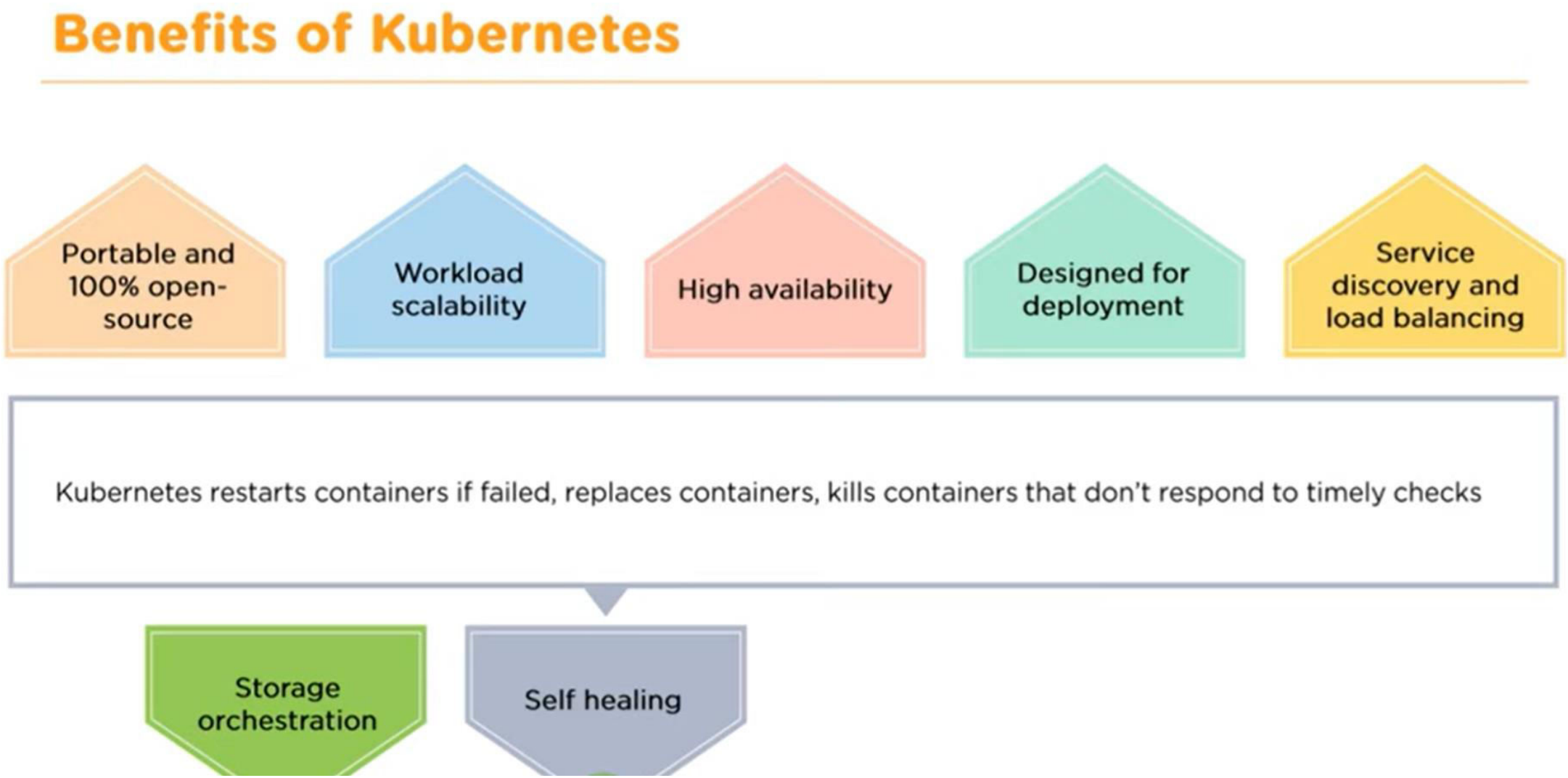
Benefits of Kubernetes



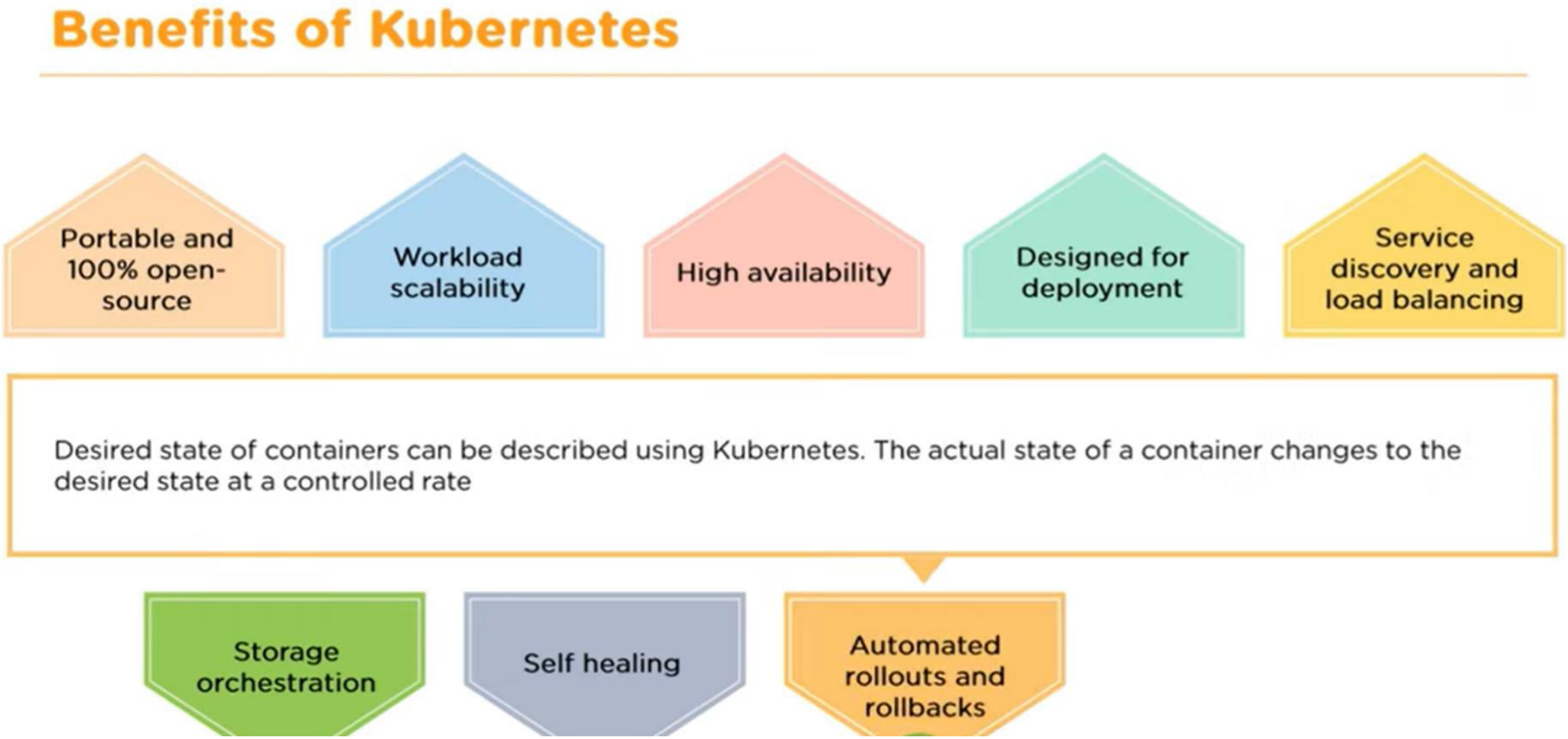
Kubernetes


Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
High availability
Designed for deployment
Service discovery and load balancing
Kubernetes automatically mounts storage systems like local storages and public cloud providers
Storage orchestration

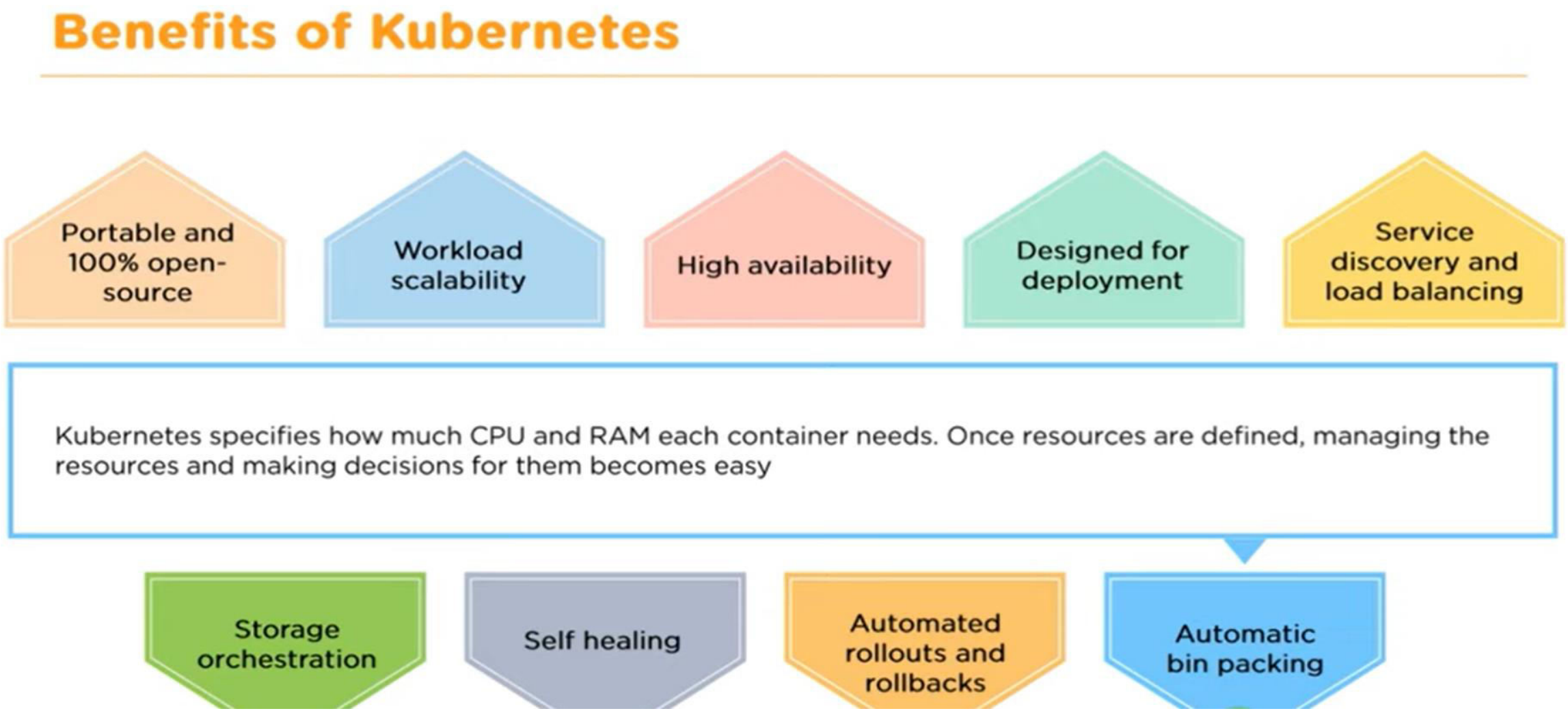
Kubernetes


Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
High availability
Designed for deployment
Service discovery and load balancing
Kubernetes restarts containers if failed, replaces containers, kills containers that don't respond to timely checks
Storage orchestration
Self healing

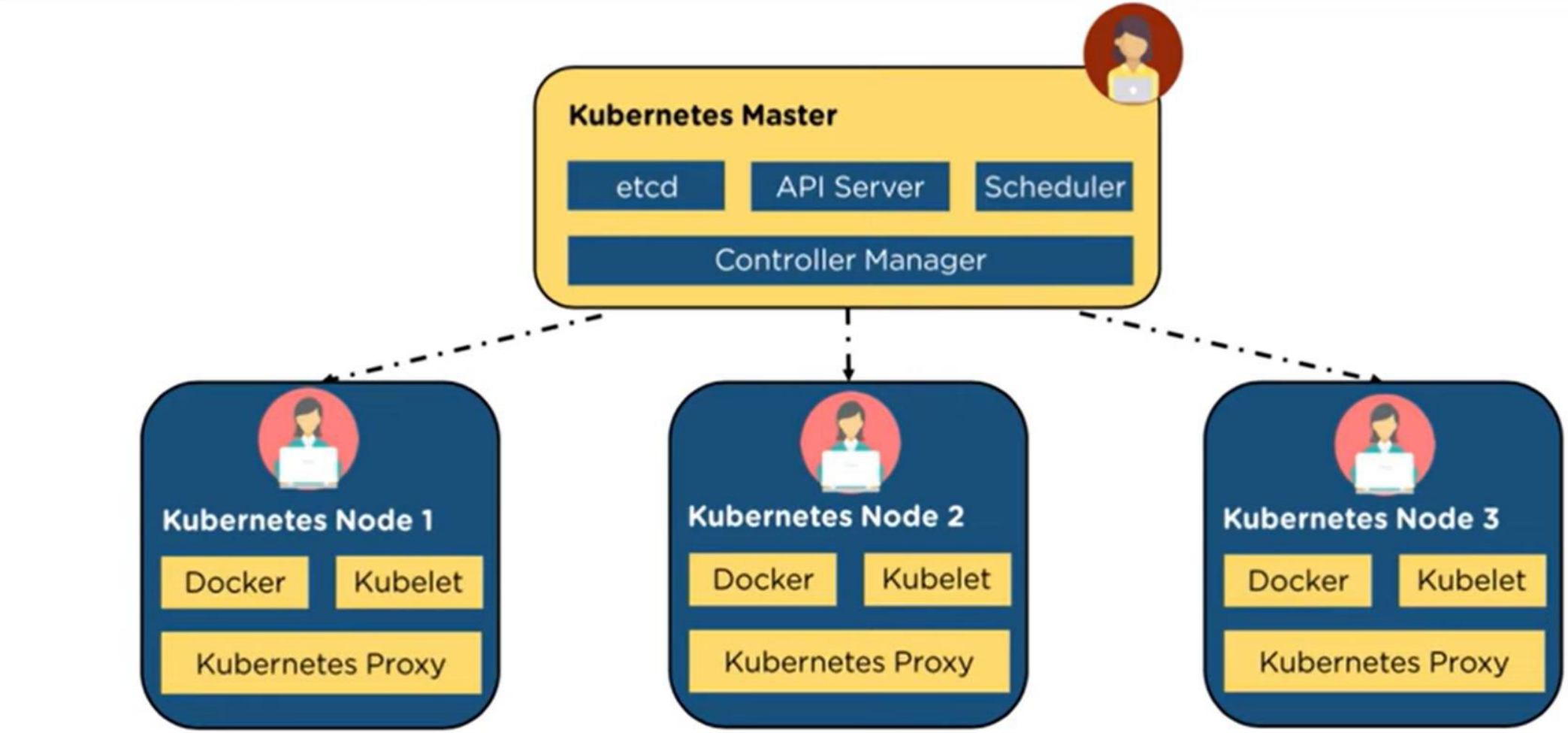
Kubernetes


Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
High availability
Designed for deployment
Service discovery and load balancing
Desired state of containers can be described using Kubernetes. The actual state of a container changes to the desired state at a controlled rate
Storage orchestration
Self healing
Automated rollouts and rollbacks

Kubernetes

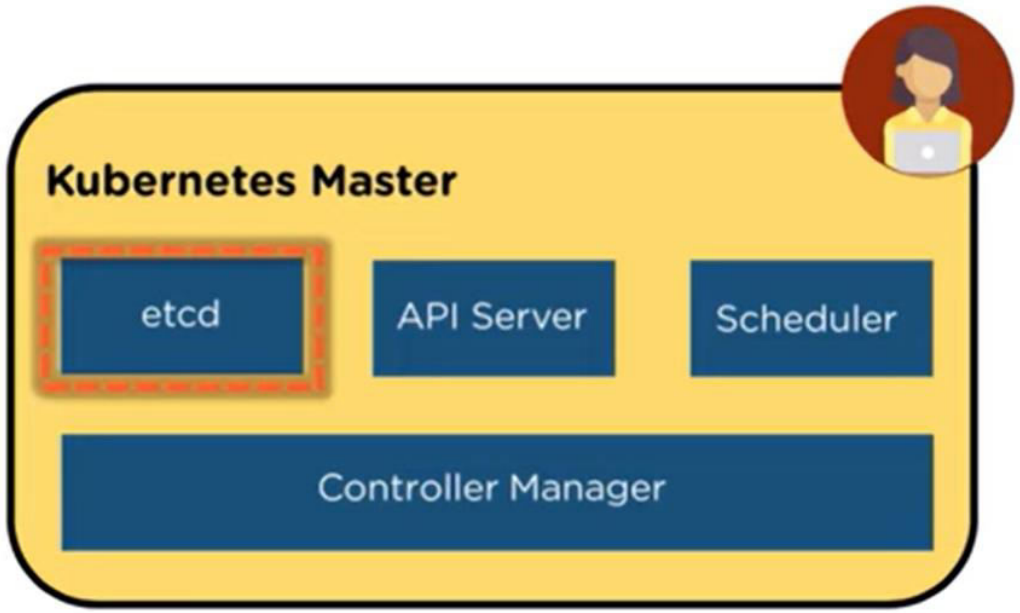

Benefits of Kubernetes
Portable and 100% open-source
Workload scalability
High availability
Designed for deployment
Service discovery and load balancing
Kubernetes specifies how much CPU and RAM each container needs. Once resources are defined, managing the resources and making decisions for them becomes easy
Storage orchestration
Self healing
Automated rollouts and rollbacks
Automatic bin packing

Kubernetes


Kubernetes Cluster Architecture
Kubernetes Master
etcd
API Server
Scheduler
Controller Manager
Kubernetes Node 1
Docker
Kubelet
Kubernetes Proxy
Kubernetes Node 2
Docker
Kubelet
Kubernetes Proxy
Kubernetes Node 3
Docker
Kubelet
Kubernetes Proxy

Kubernetes

Kubernetes Master Components



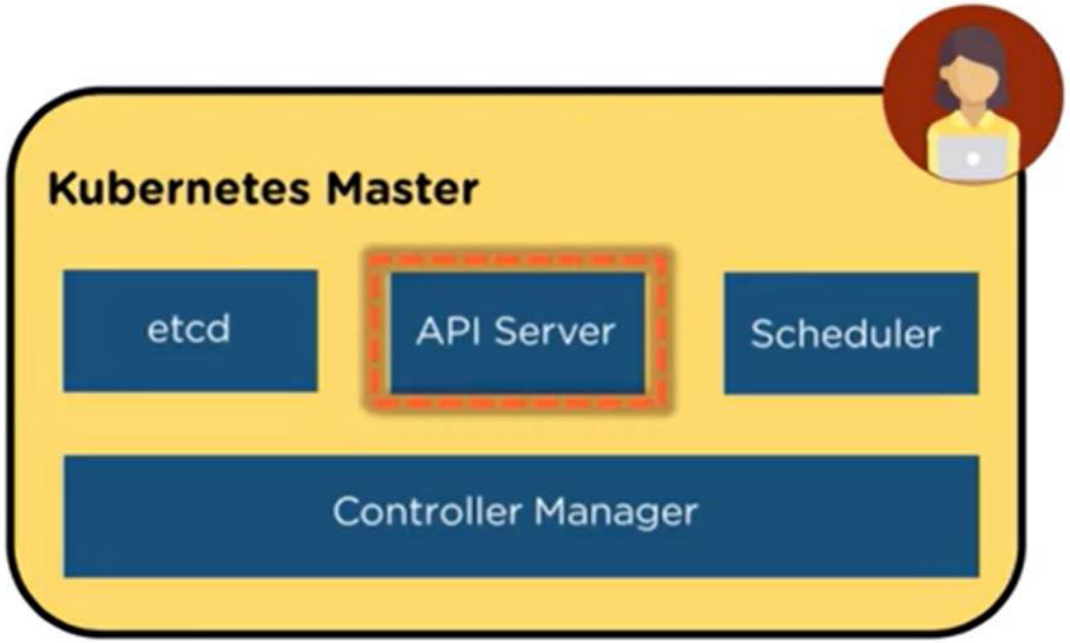
etcd

- Stores the configuration information required by nodes in the cluster
- Key-value is available and distributed across multiple nodes
- Accessible only by API Server as it may contain some sensitive information

Note - A node is a working machine in Kubernetes; it can be a VM, cloud, and so on

Kubernetes

Kubernetes Master Components

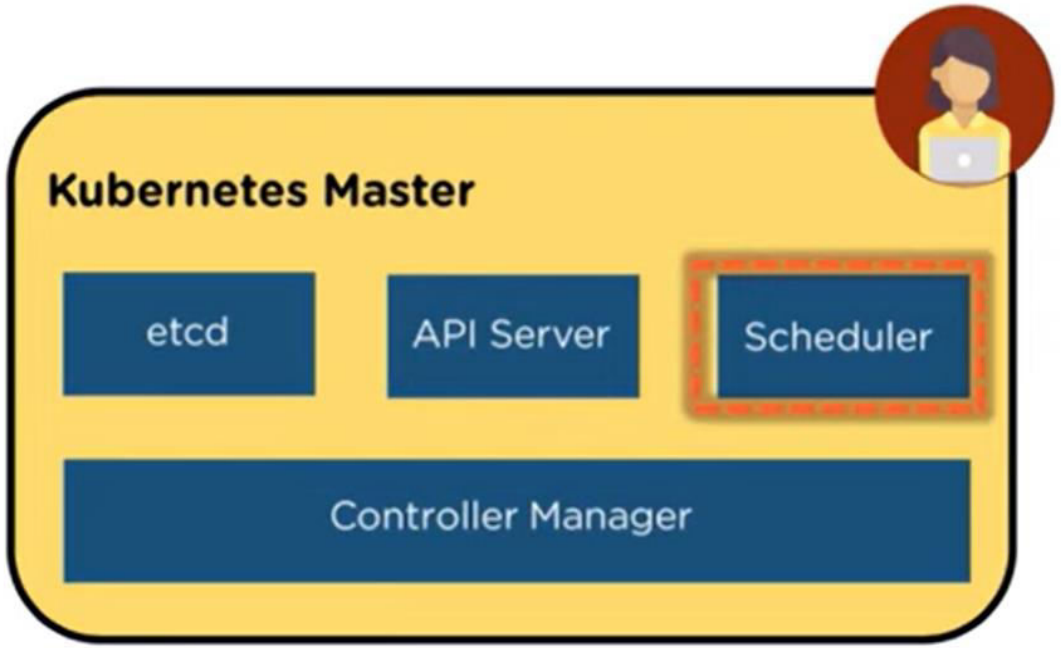


API Server

- It is a means to provide operations on the cluster
- It implements an interface so that different tools and libraries can communicate effectively
- This component interacts with the worker nodes and gives them the require information

Kubernetes

Kubernetes Master Components

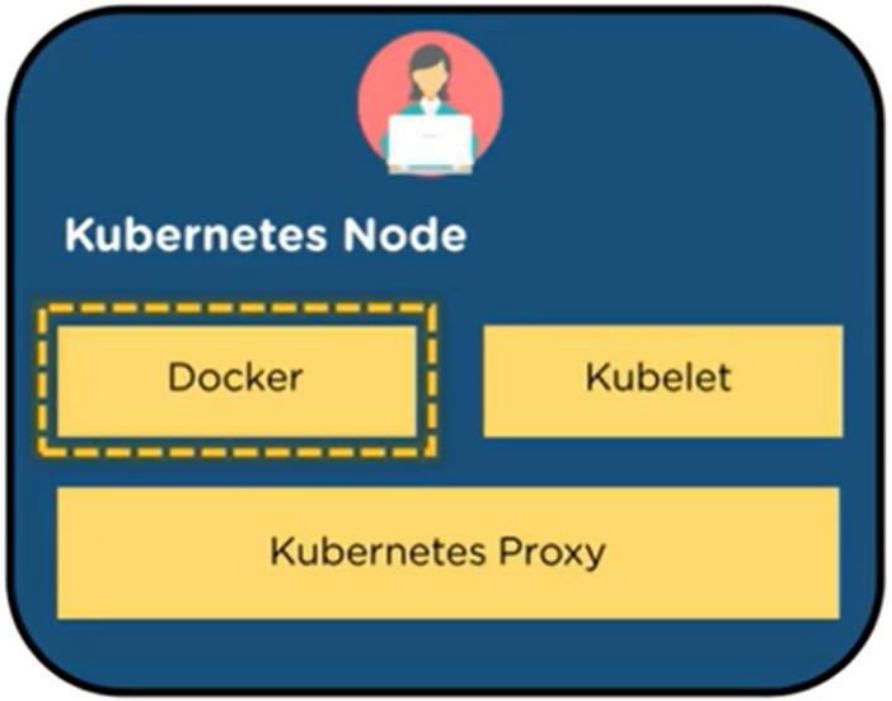


Scheduler

- It is a key component of Kubernetes Master
- It is a service that is responsible for the dispersing workload in the master node
- It also tracks the utilization of workload on cluster nodes and then places the workload onto the resources that are available

Kubernetes

Kubernetes Worker Components

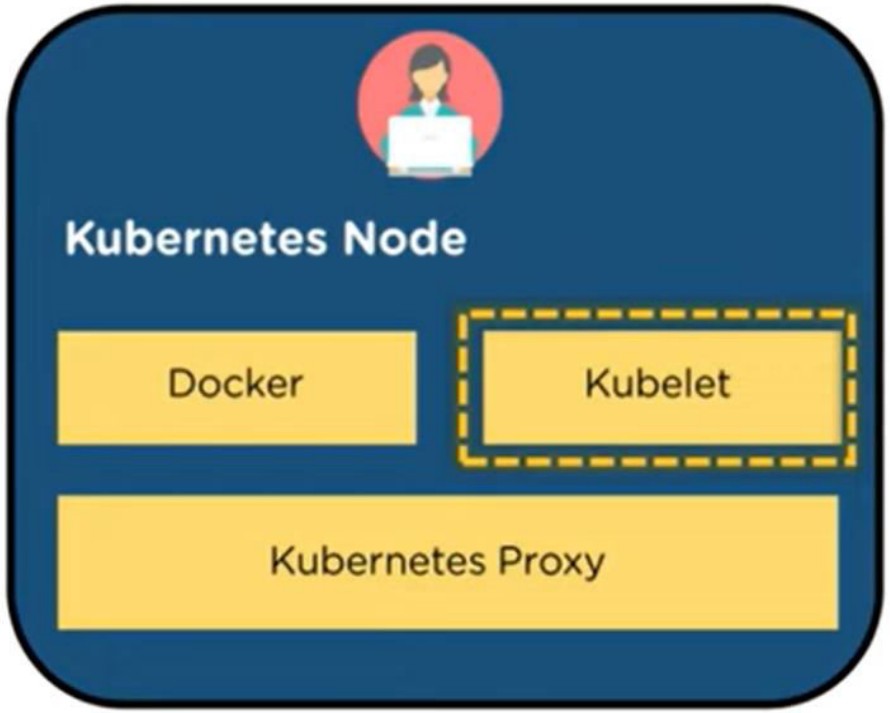


Docker

- It helps in running the applications in an isolated but lightweight operating environment
- Docker is referred to as container runtime and container runtime is used to run and manage a containers lifecycle

Kubernetes

Kubernetes Worker Components

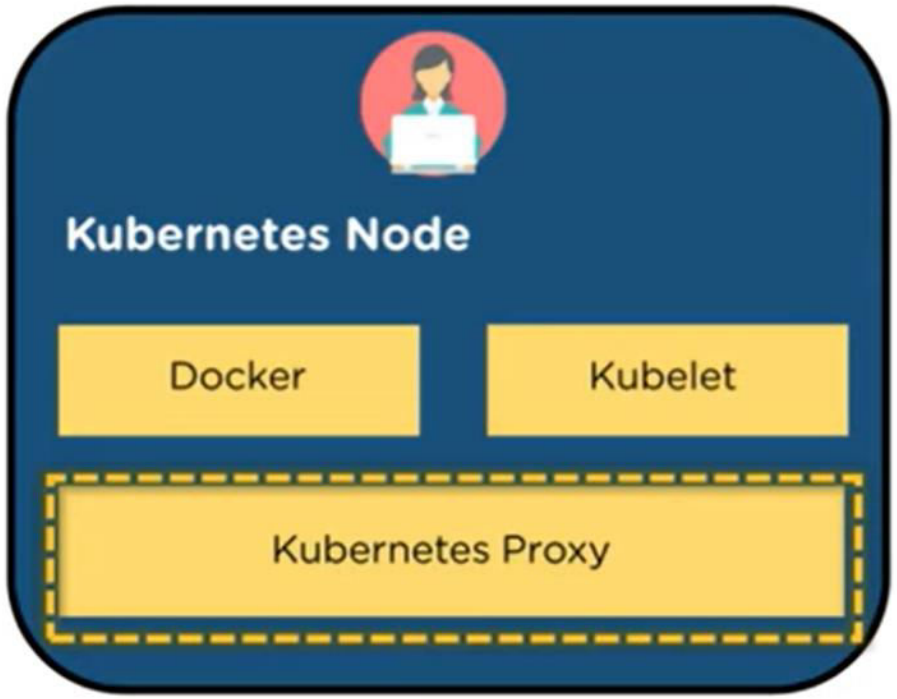


Kubelet

- It is a service responsible for conveying information to and fro to the control plane service (API Server)
- It interacts with etcd to read the configuration details. It communicates with the master node to get commands and work efficiently

Kubernetes

Kubernetes Worker Components

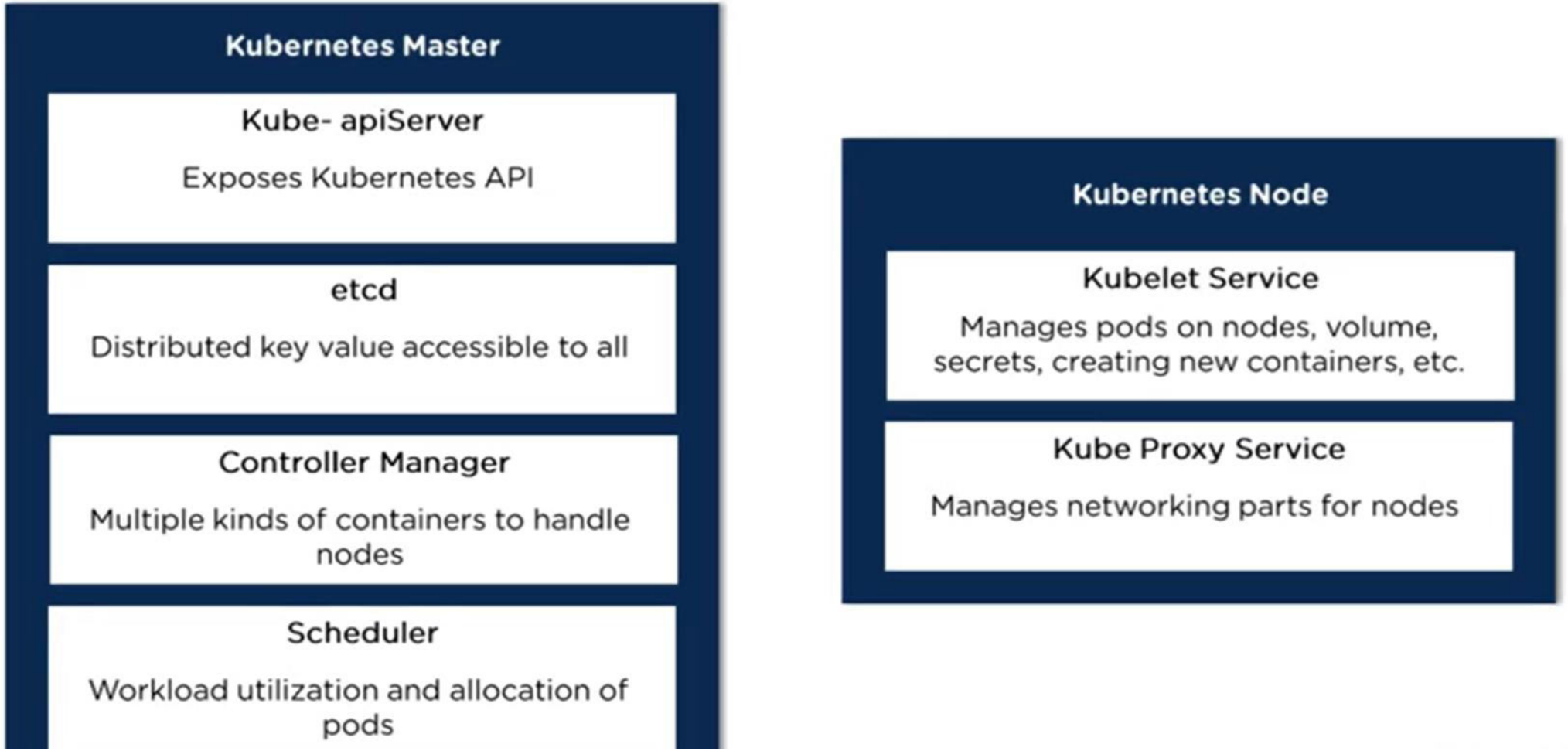


Kubernetes Proxy

- It is a proxy service that runs on every node and helps in making services available to the external host. It helps in forwarding request to assigned containers
- It performs primitive load balancing. It manages pods on nodes, volumes, secrets, creation of new containers, and health check-ups

Kubernetes

Kubernetes Cluster Structure (RECAP)



Microservices

Microservice is a service-based application development methodology. In this methodology, big applications will be divided into smallest independent service units. Microservice is the process of implementing Service-oriented Architecture (SOA) by dividing the entire application as a collection of interconnected services, where each service will serve only one business need.

Microservice is a small, loosely coupled distributed service. Microservice architecture evolved as a solution to the scalability, independently deployable, and innovation challenges with Monolithic architecture
It allows you to take a large application and decompose or break it into easily manageable small components with narrowly defined responsibilities. It is considered the building block of modern applications. Microservices can be written in a variety of programming languages, and frameworks, and each service act as a mini-application on its own. Microservice can be considered as the subset of SOA(Service Oriented Architecture).

A microservice typically implements a set of distinct features or functionality. Each microservice is a mini-application that has its own architecture and business logic. For example, some microservices expose an API that's consumed by other microservices or by the application's clients, such as third-party integrations with payment gateways and logistics.

Microservices

Deployment Patterns - There are a few patterns available for deploying microservices. These include the following:
Service Instance per Host
Service Instance per Container
Service instance per Virtual Machine.
Multiple service instances per host

Benefits of Microservices

Small Modules – The application is broken into smaller modules that are easy for developers to code and maintain.
Easier Process Adaption – By using microservices, new Technology & Process Adaption becomes easier. You can try new technologies with the newer microservices that we use.
Independent scaling – Each microservice can scale independently via X-axis scaling (cloning with more CPU or memory) and Z-axis scaling (sharding), and Y-axis scaling (functional decomposition) based on their needs.
Removes dependency – Microservice eliminates long-term commitment to any single technology stack.
Unaffected – Large applications remain largely unaffected by the failure of a single module.
DURS – Each service can be independently DURS (deployed, updated, replaced, and scaled).
Increased Security: –Microservices enable data separation. Each service has its own database, making it harder for hackers to compromise your application.
Open Standards: –APIs enable developers to build their microservices using the programming language and technology they prefer.

Microservices

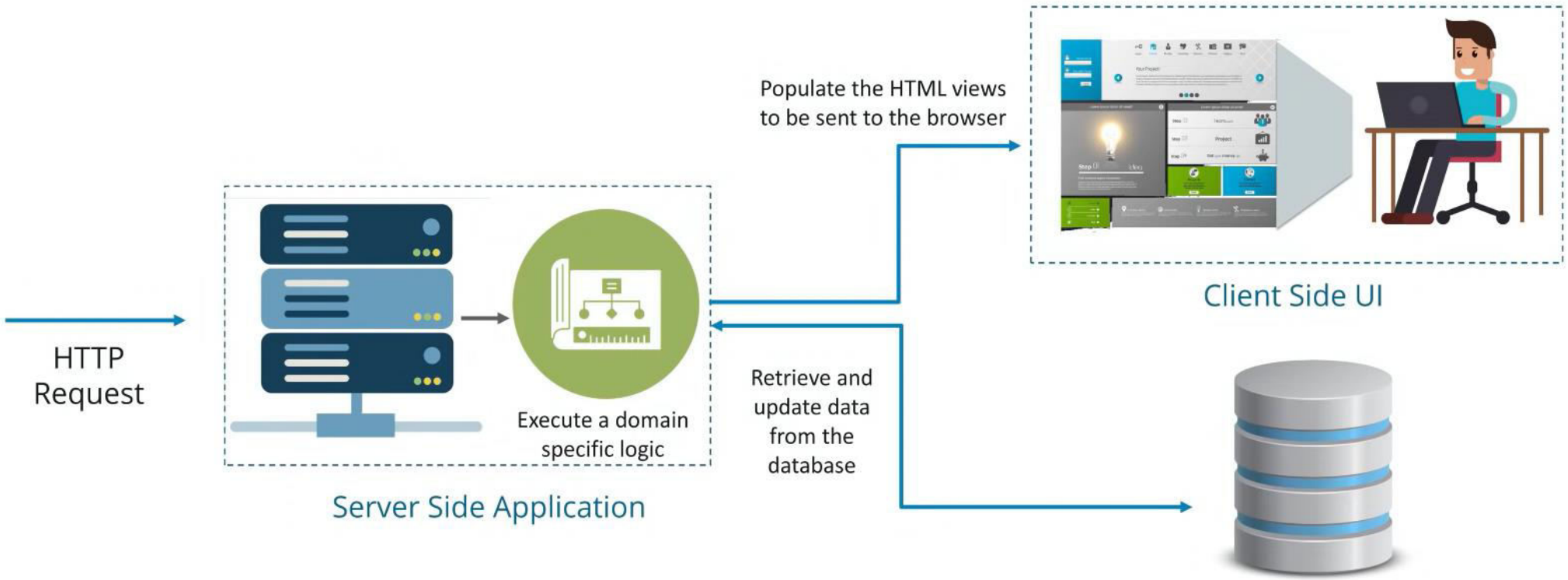
Limitations of Microservices

Configuration Management – As it becomes granular the headache comes with configuring the services and monitoring those. You need to maintain configurations for hundreds of components across environments.
Debugging – Tracking down the service failure is a painstaking job. You might need to look into multiple services across different components. Centralized Logging and Dashboards are essential to make it easy to debug problems.
Automation – Because there are a number of smaller components instead of a monolith, you need to automate everything – Builds, Deployment, Monitoring, etc.
Testing – Needs a greater effort for end-to-end testing as it needs all the dependent services to be up and running.
Coordination – While handling requests across multiple independent services there is a requirement for proper workflow management

Microservices

Before Microservices – Monolithic Architecture

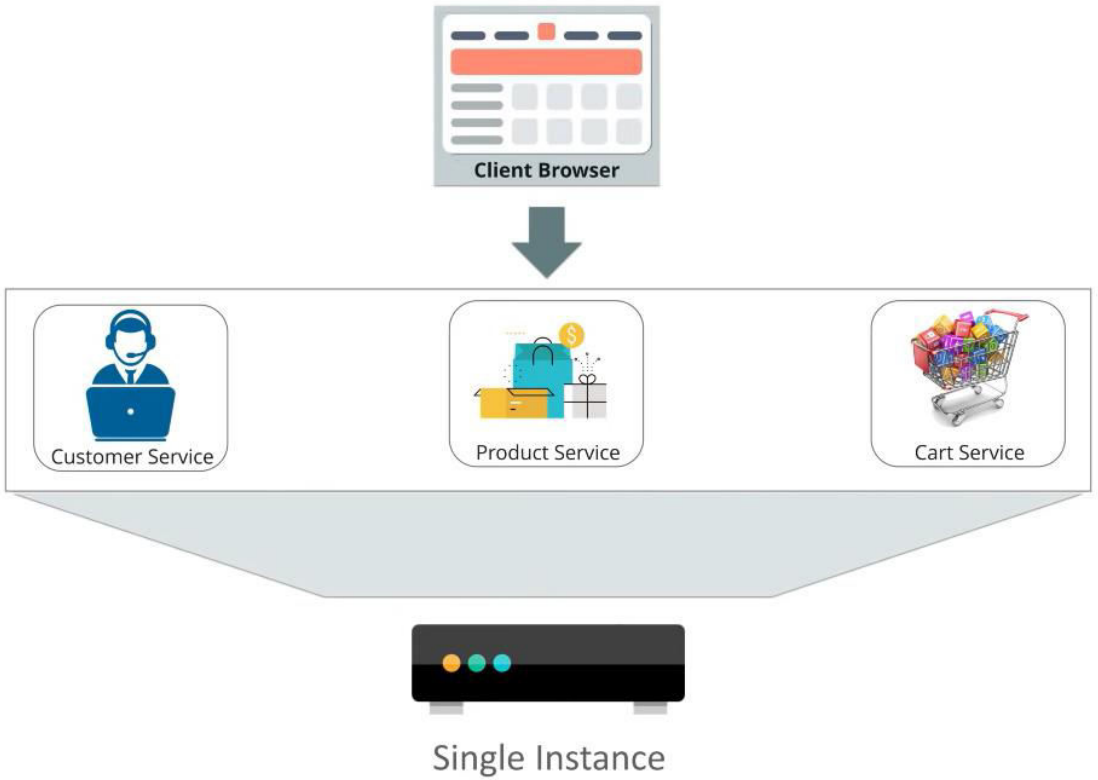
Monolithic Architecture is like a big container wherein all the software components of an application are assembled together and tightly packaged



Microservices

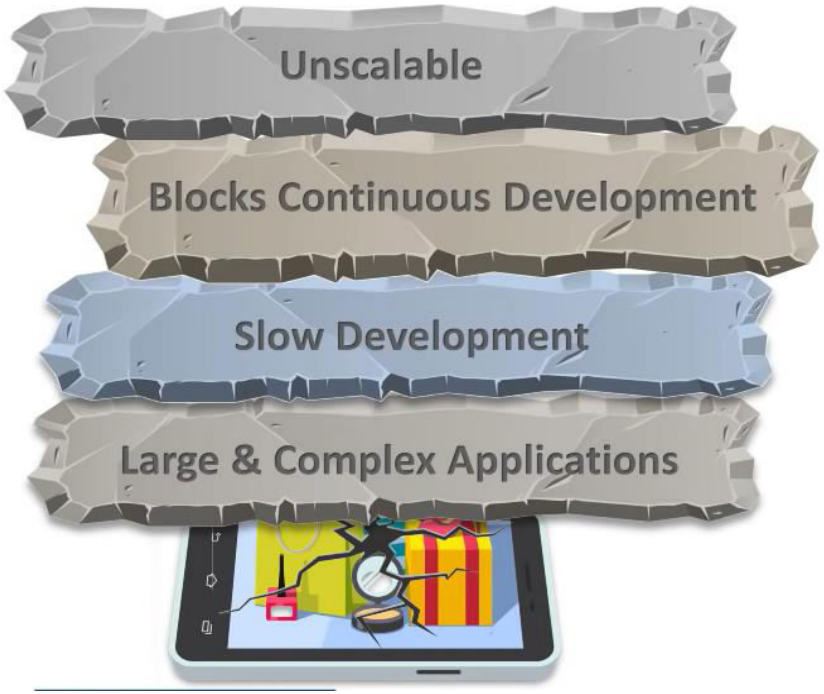
Monolithic Architecture - Example

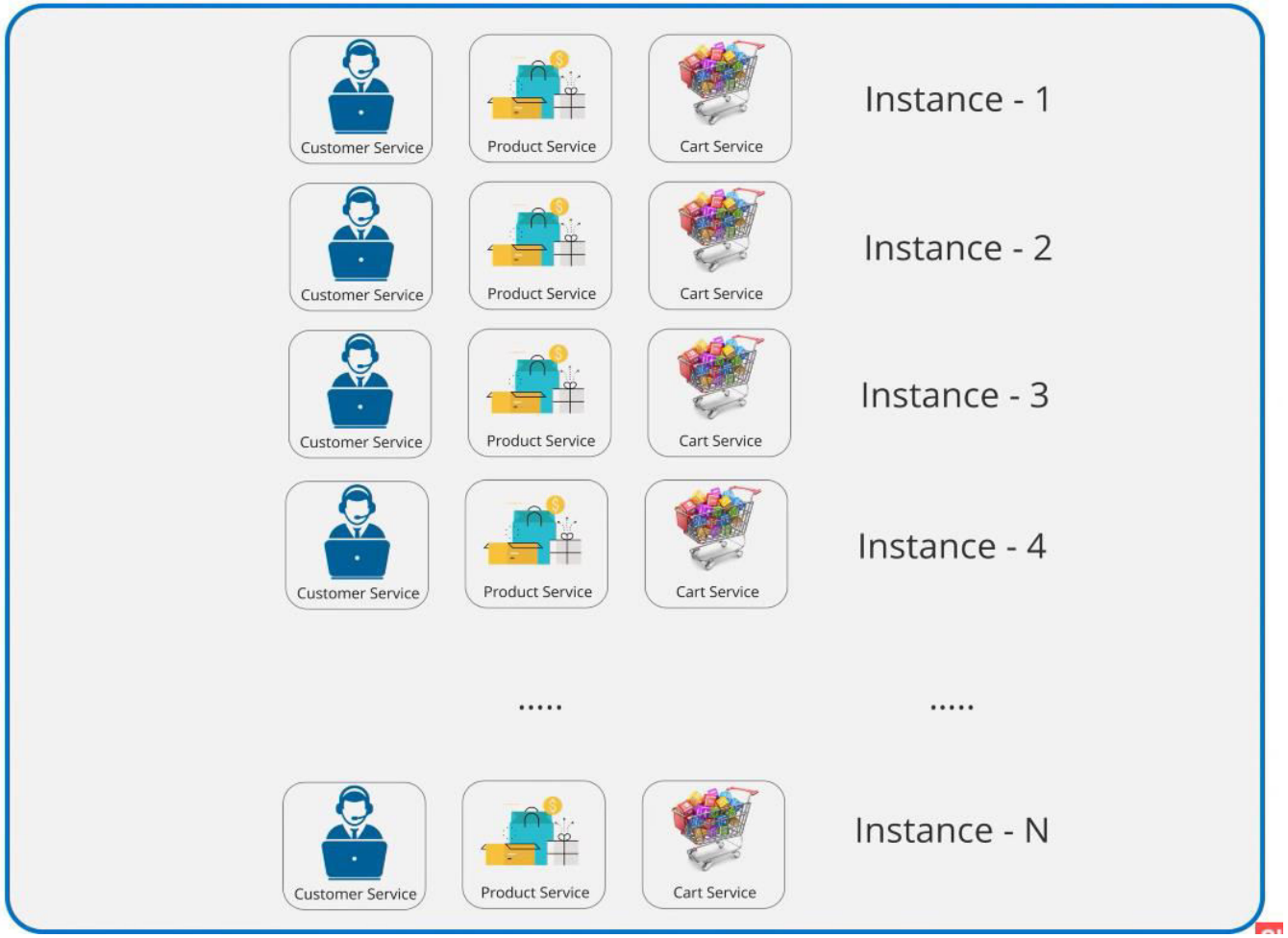
Let’s take a classic use case of an E-Commerce Application



Microservices

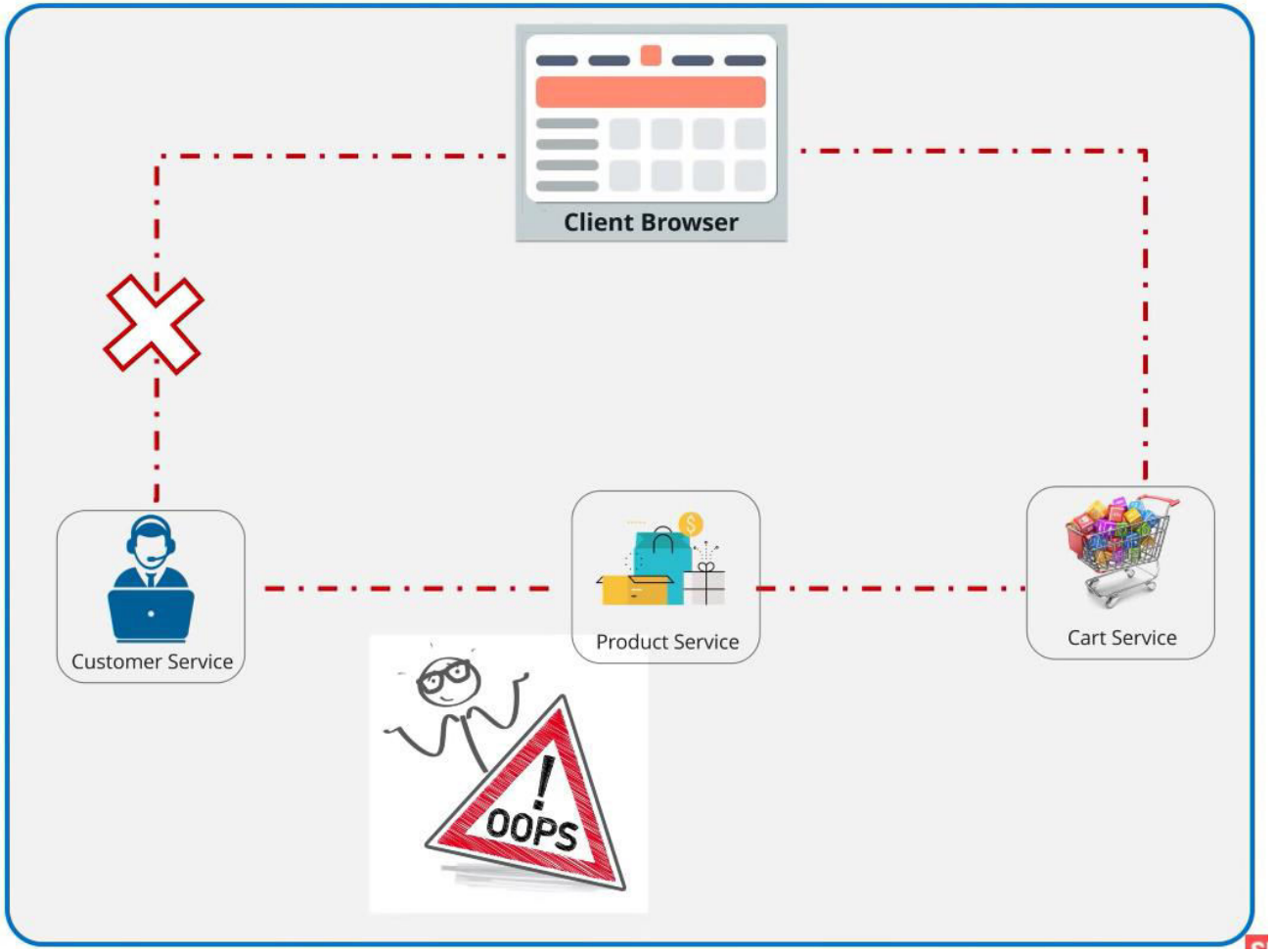
Monolithic Architecture - Challenges





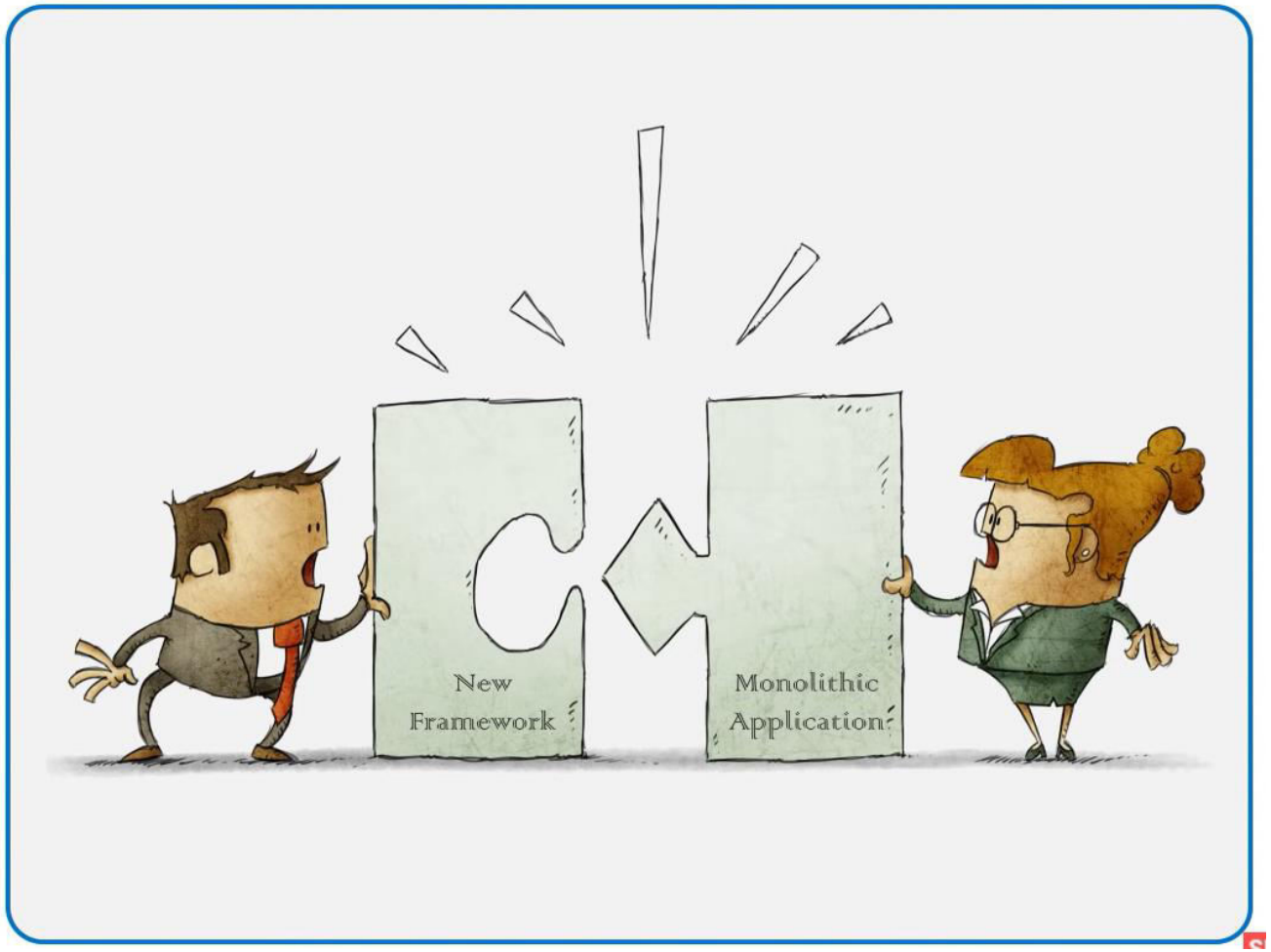
Microservices

Monolithic Architecture - Challenges



Microservices

Monolithic Architecture - Challenges



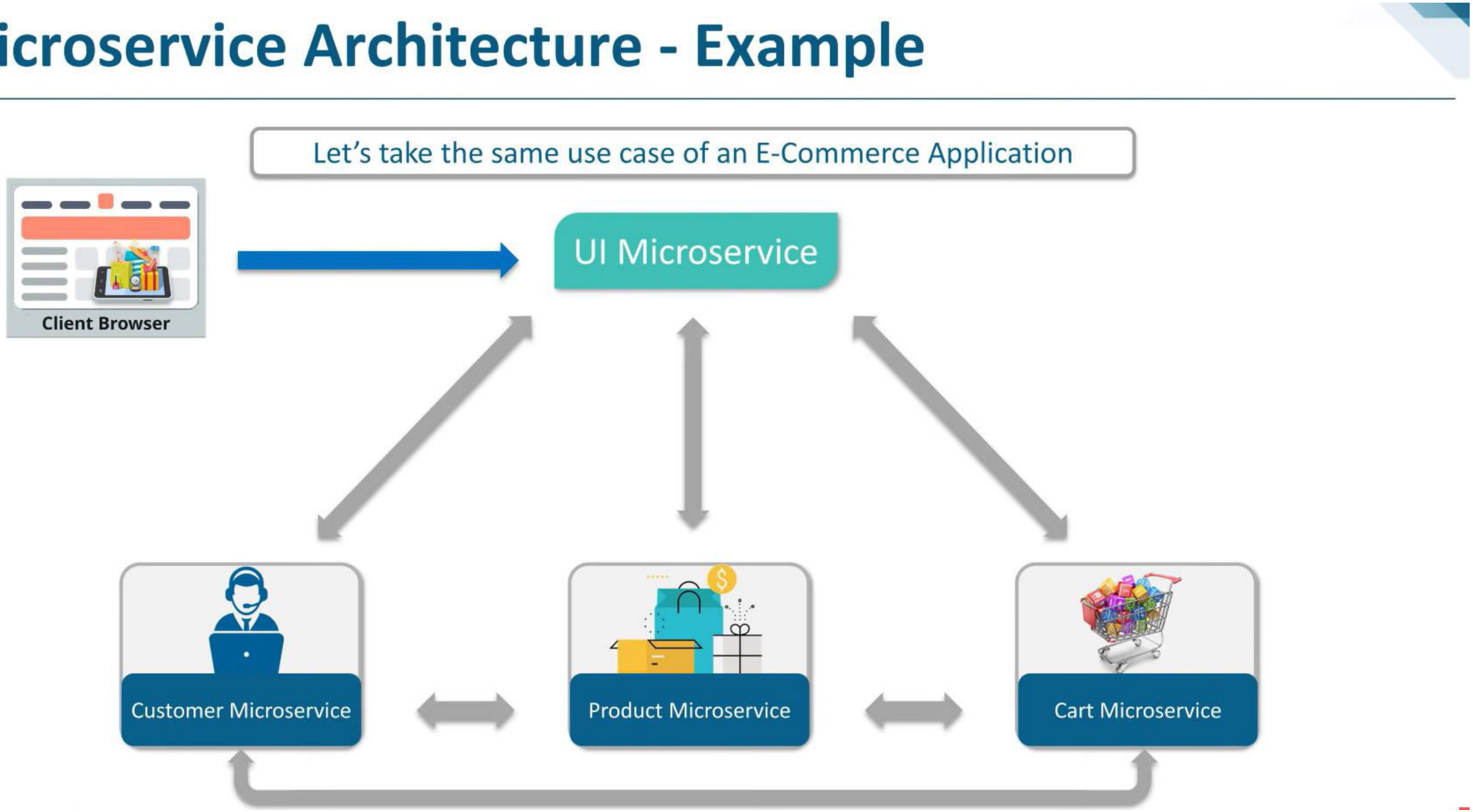
Microservices

What Is Microservice Architecture?

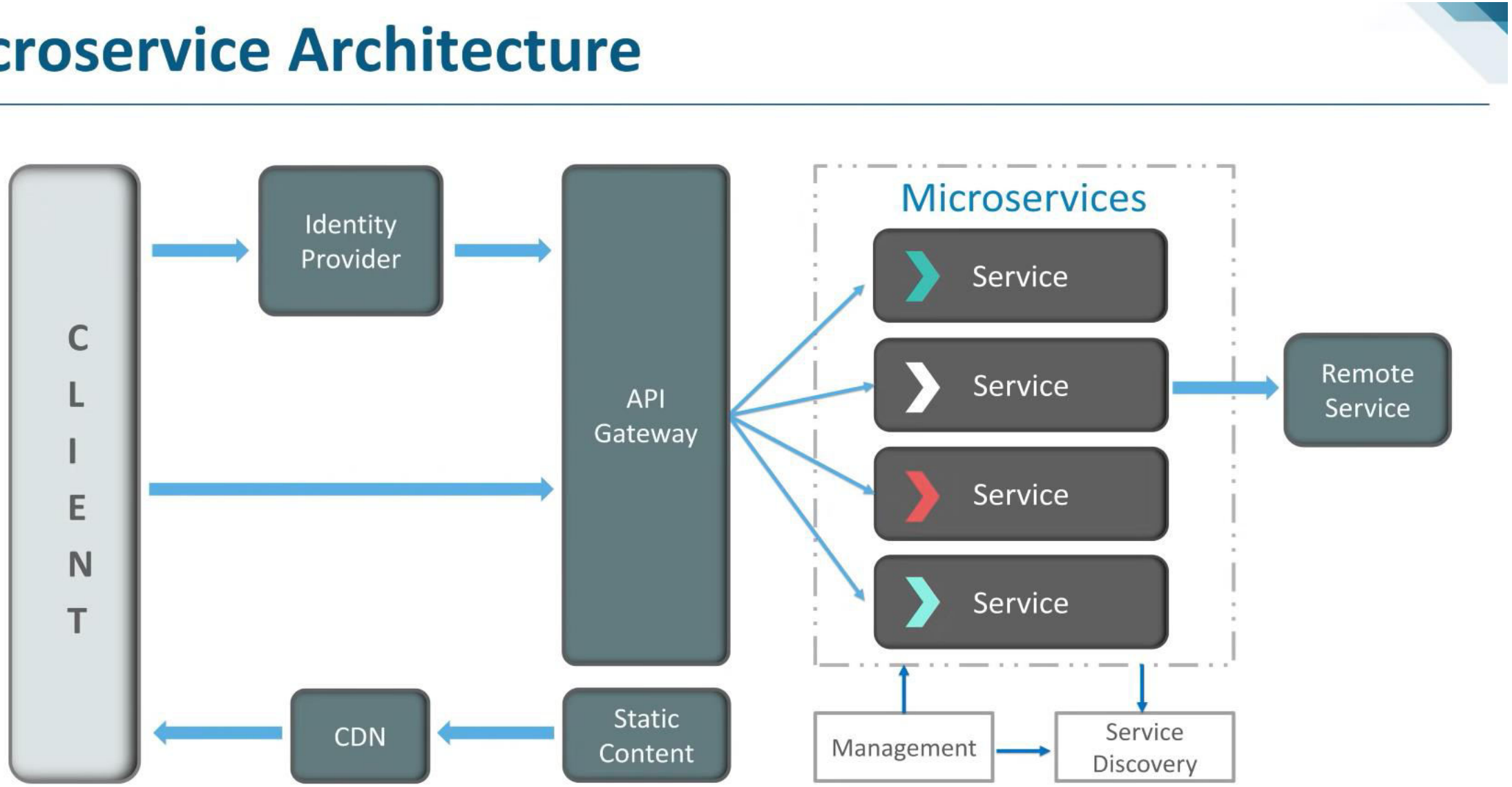
Microservices, aka *Microservice Architecture*, is an **architectural style** that structures an application as a **collection of small autonomous services**, modelled around a **Business Domain**

In Microservice Architecture, each service is **self-contained** and implements a **single Business capability**

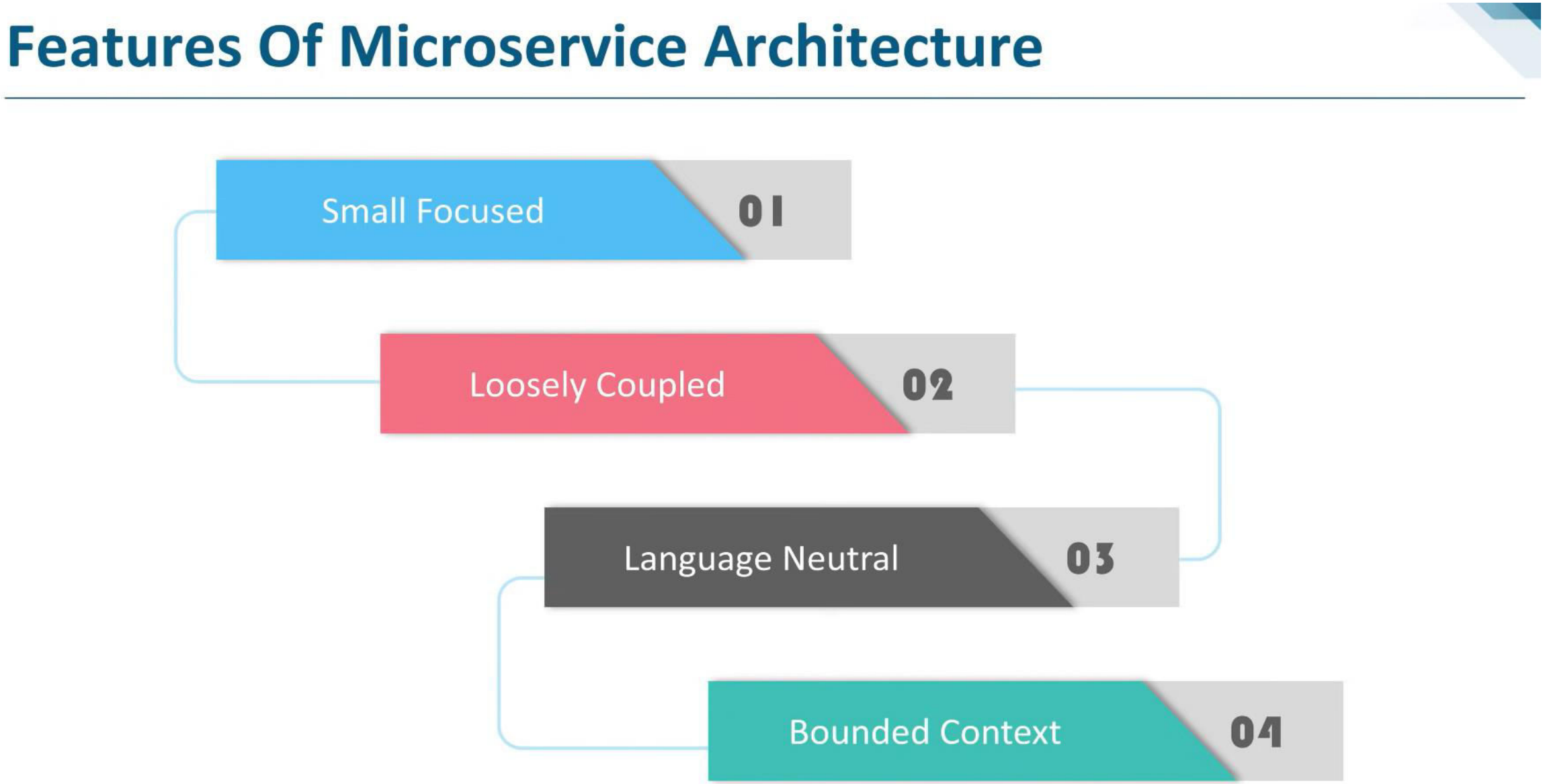
Microservices



Microservices

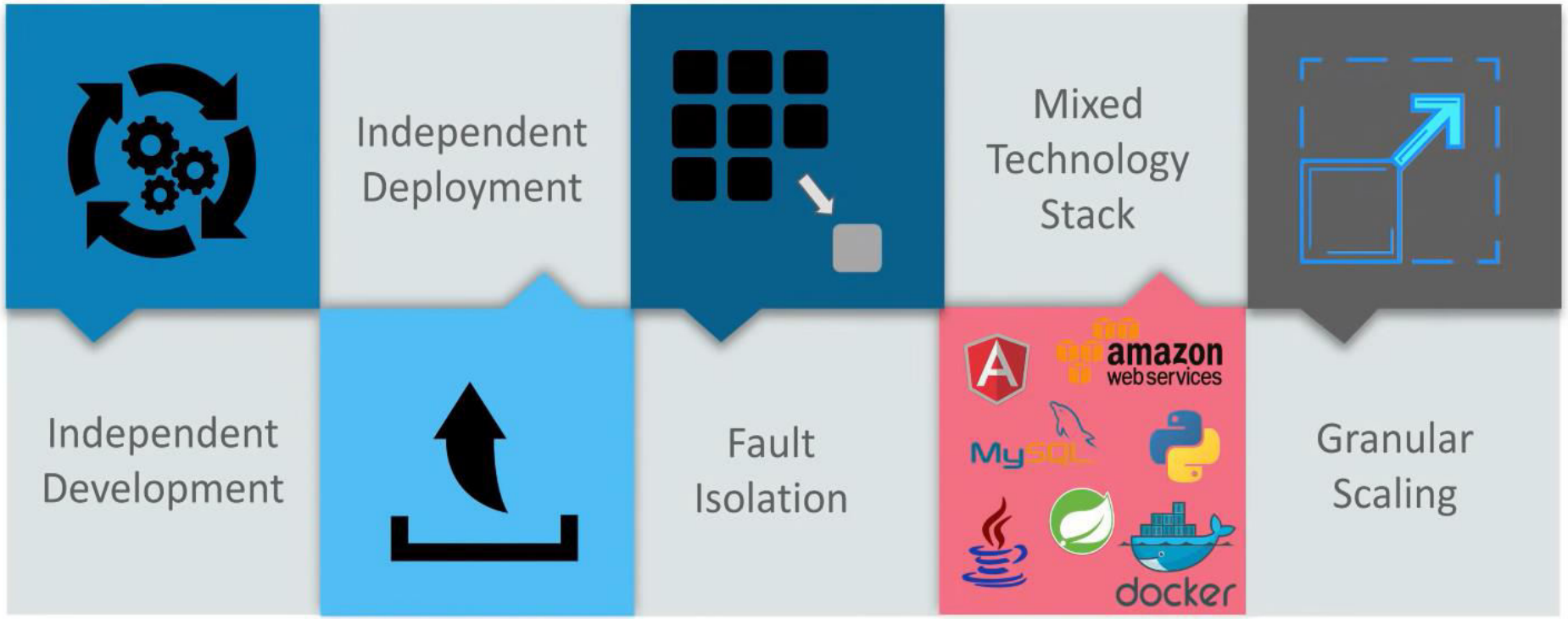


Microservices



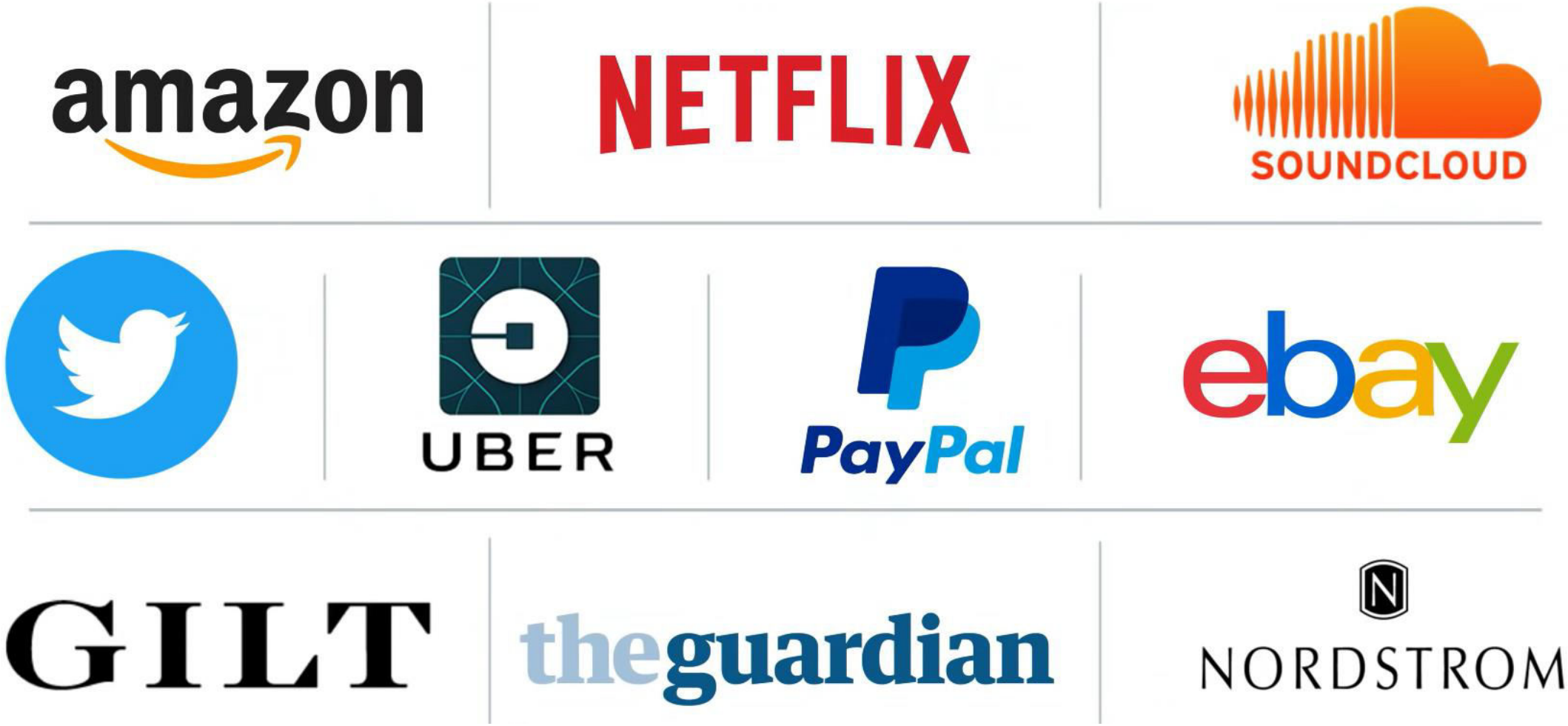
Microservices

Advantages Of Microservice Architecture

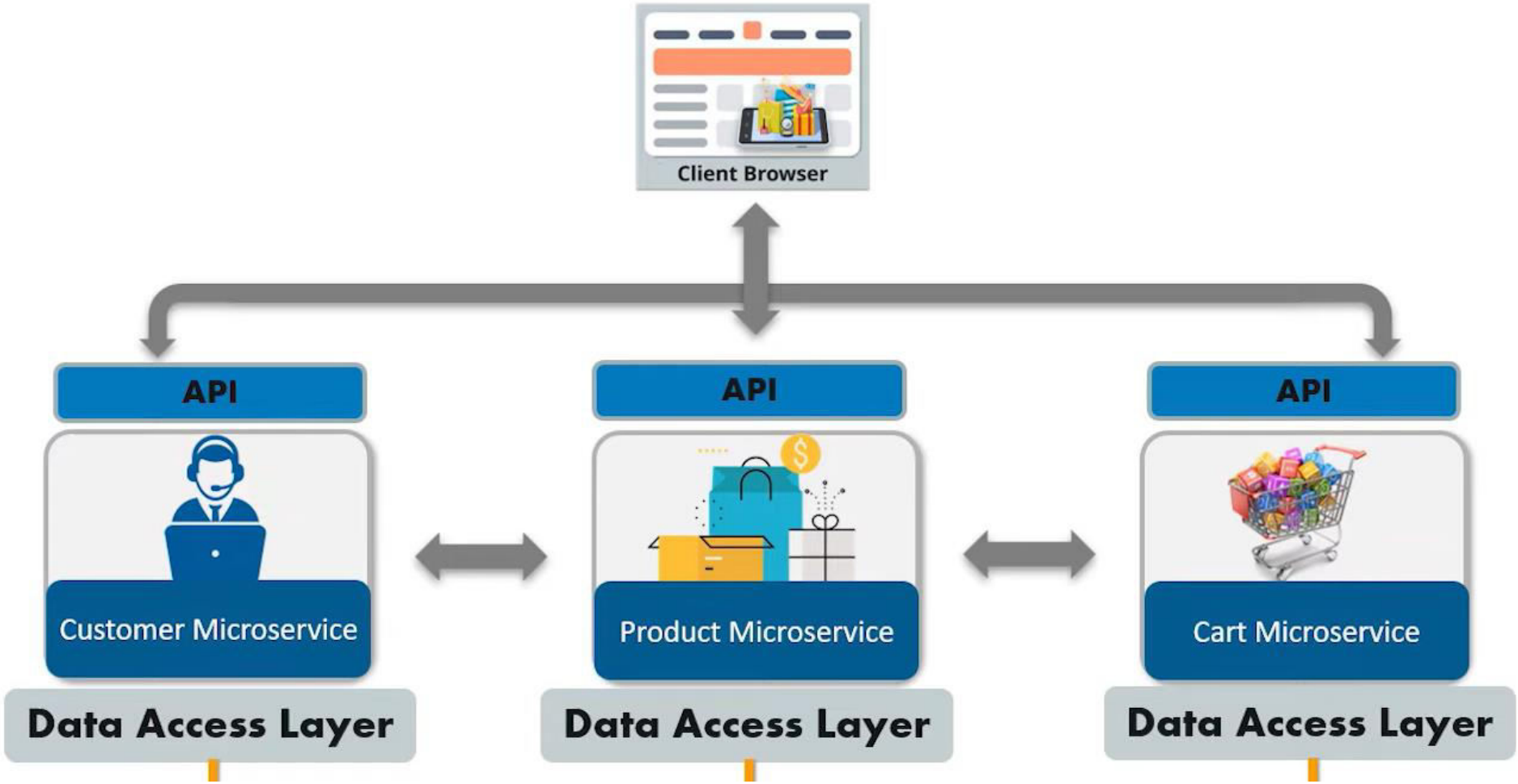


Microservices

Companies Using Microservices



Microservices



Microservices

MICROSERVICES

An architectural style through which, you can build applications in the form of small autonomous services.

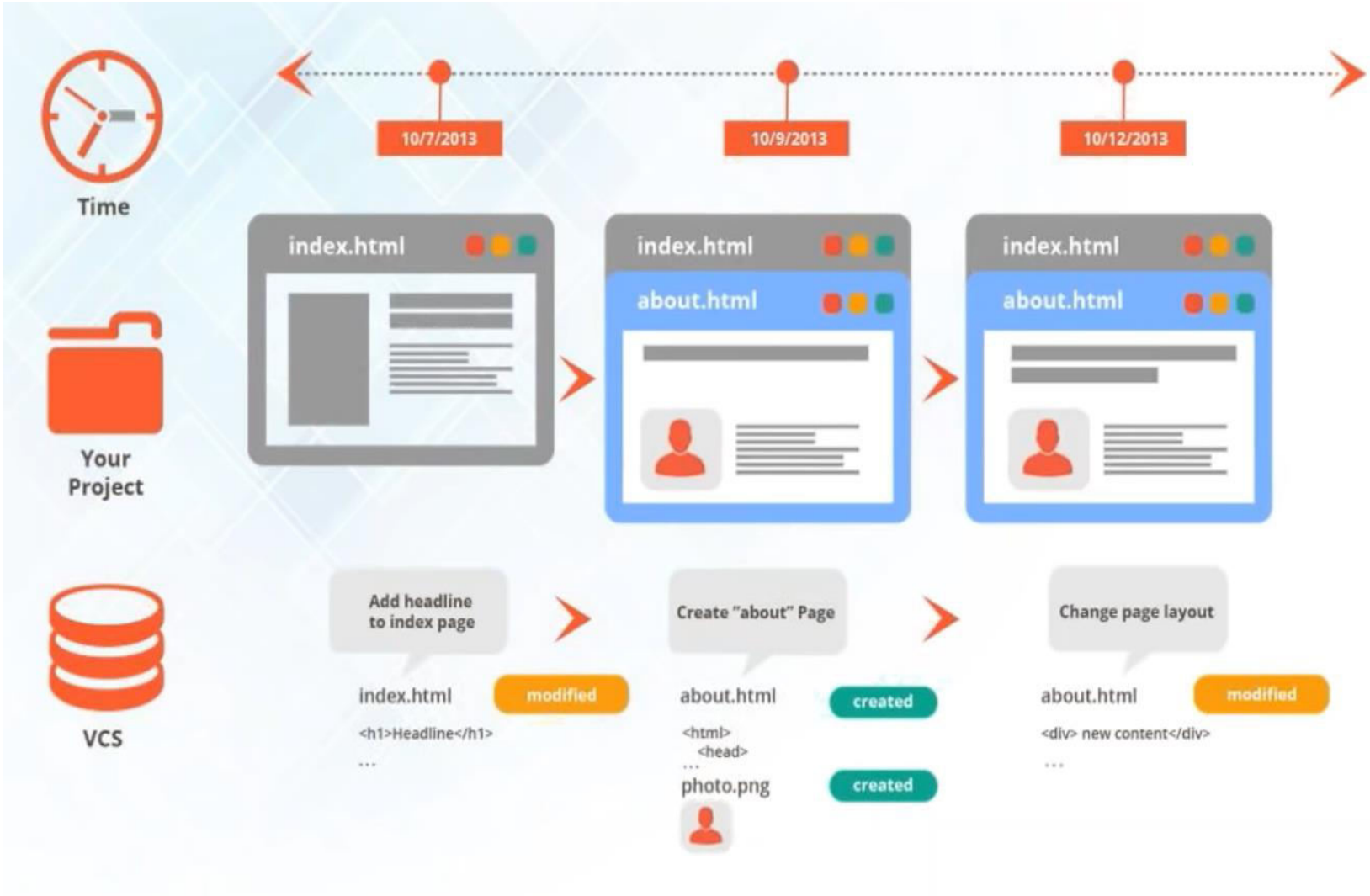
A set of procedures and functions which allow the consumer to use the underlying service of an application.

API

Git & GitHub

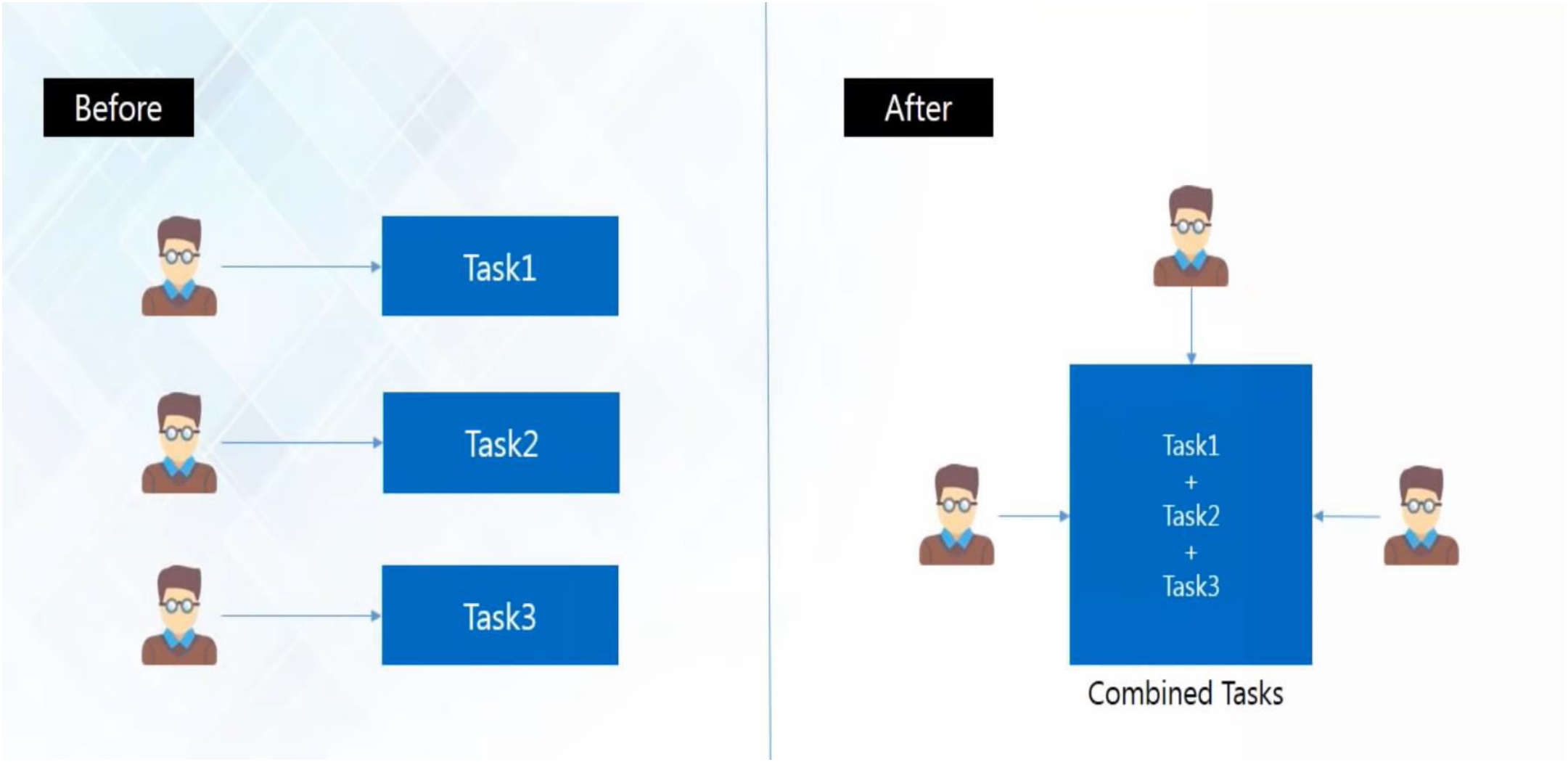
- Version Control – What & Why?
- Version Control Tools
- GitHub & Git
- Case Study: Dominion Enterprises
- Git Features
- Git Operations & Commands

Git & GitHub



- Version control is the management of changes to documents, computer programs, large web sites, and other collections of information.
- These changes are usually termed as "versions".

Git & GitHub

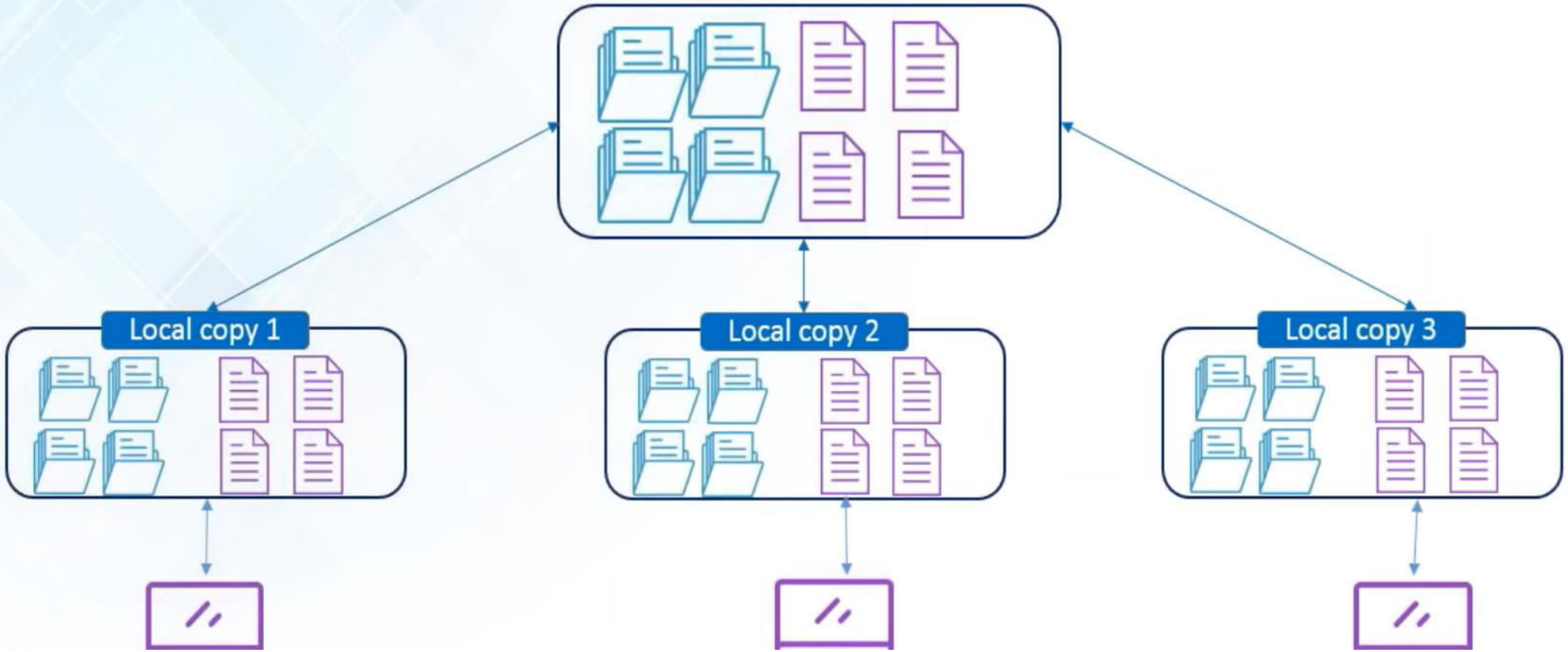

Before
Task1
Task2
Task3
After
Task1
+
Task2
+
Task3
Combined Tasks

Git & GitHub

- Snapshots of all versions are properly documented and stored.
- Versions are also named accurately.

Git & GitHub

In any case if your central server crashes, a backup is always available in your local servers.



Git & GitHub

When you change version -

- VCS provides you with proper description
- What exactly was changed
- When it was changed

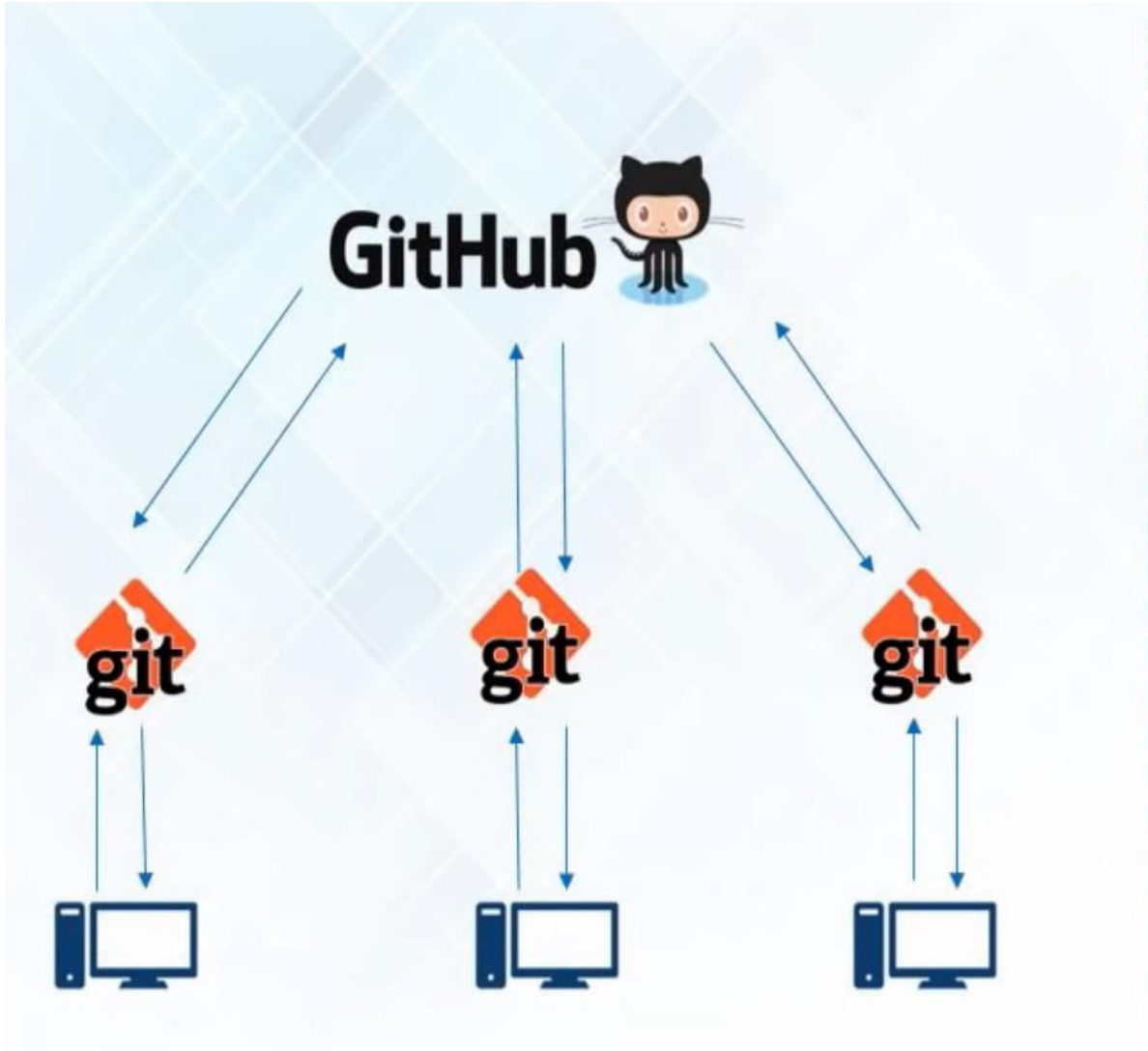
And hence, you can analyze how your project evolved between versions.

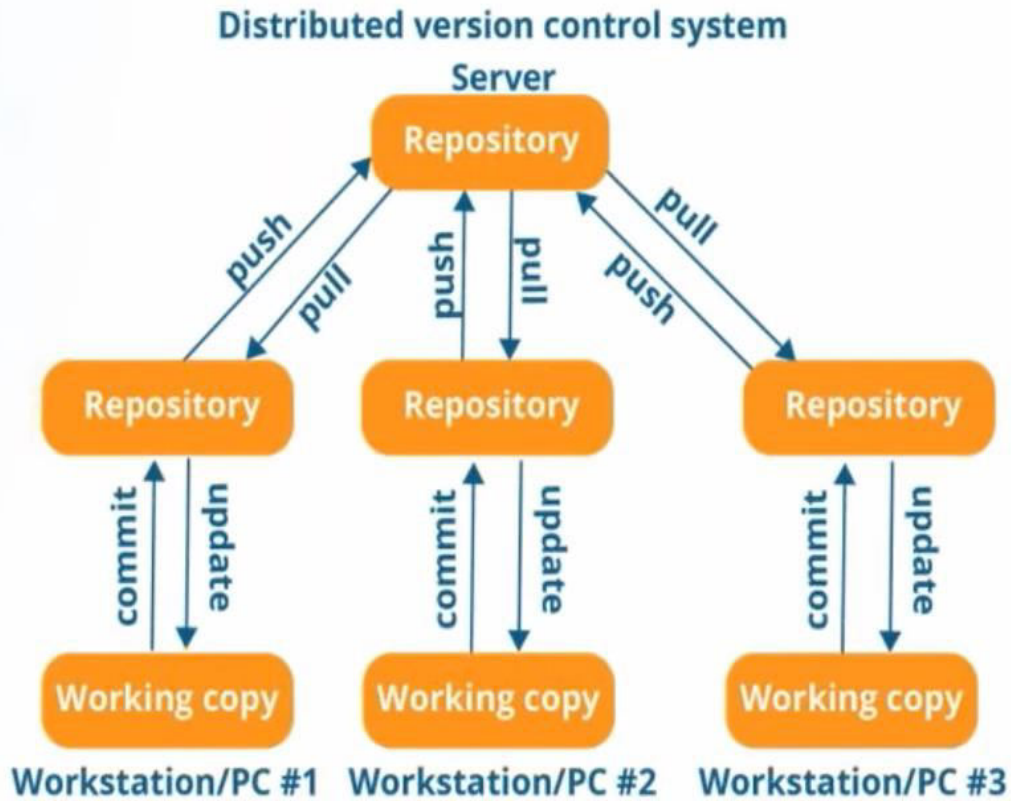
Git & GitHub

Git & GitHub


100
75
50
25
Average
Nov 6, 2011
Jul 21, 2013
Apr 5, 2015
Jan 16,2017
Git
SVN
Mercurial
CVS

Git & GitHub

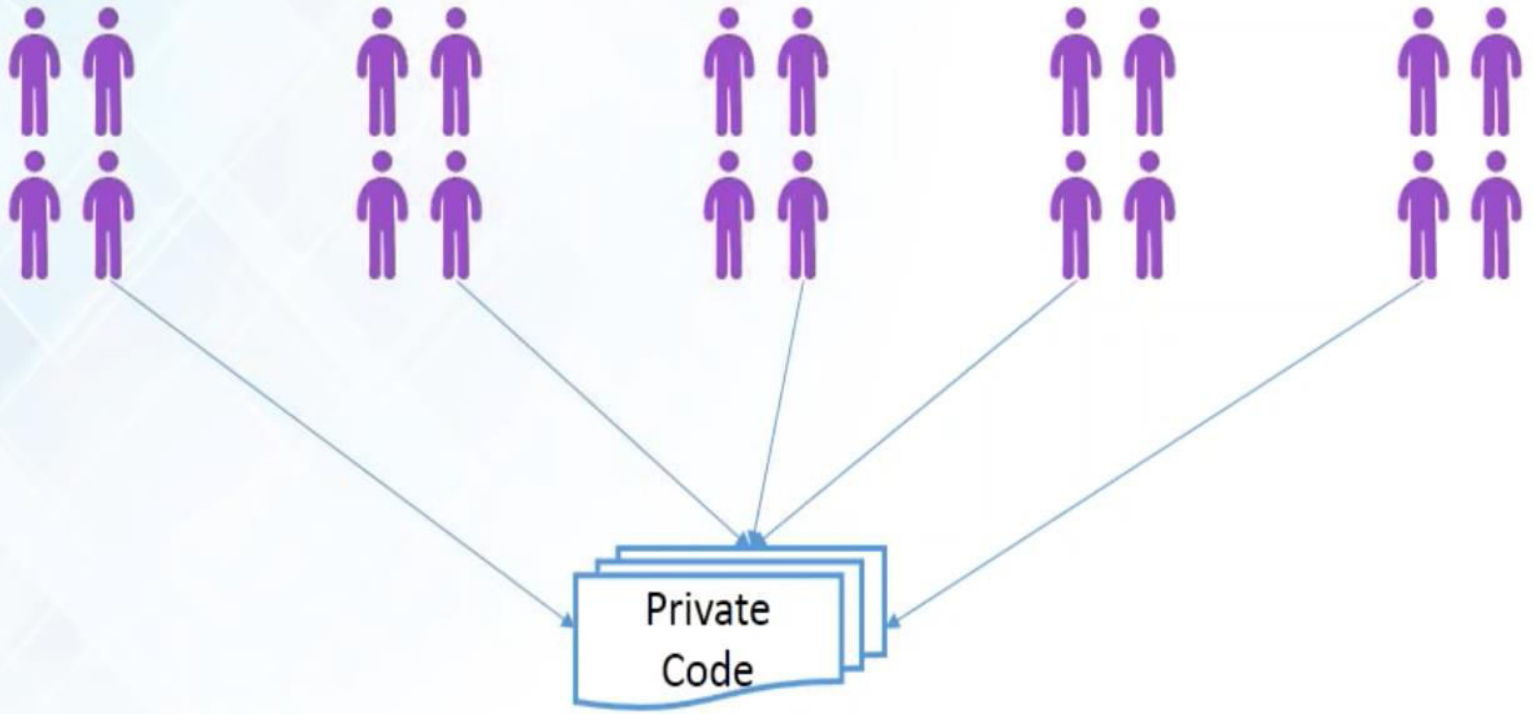

GitHub
git
git
git


Distributed version control system
Server
Repository
push
pull
push
pull
pull
push
Repository
Repository
Repository
commit
update
commit
update
commit
update
Working copy
Working copy
Working copy
Workstation/PC #1
Workstation/PC #2
Workstation/PC #3

Git & GitHub

Problem Statement:

Each team has its own goals, projects, and budgets and they also have Unique needs and workflows

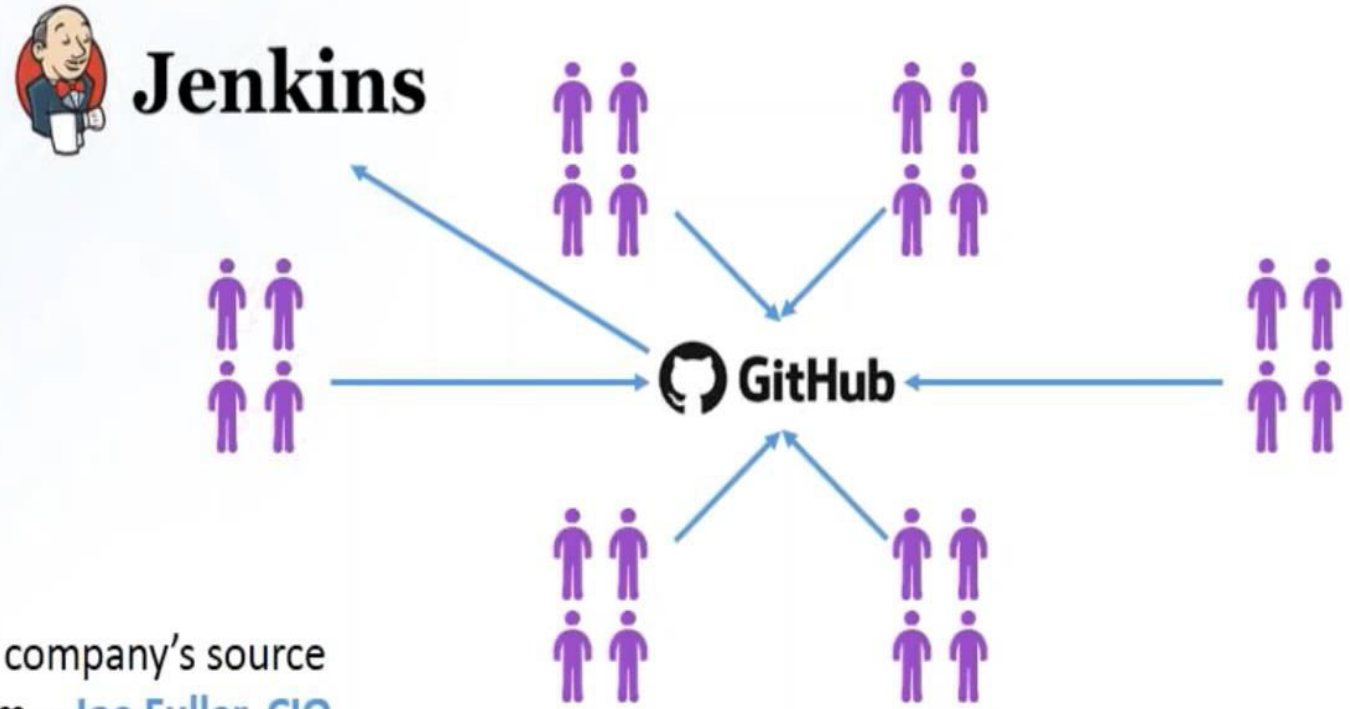


They wanted to make private code “publicly” to make their work more transparent across the company

Git & GitHub

Reason for using GitHub as the solution:

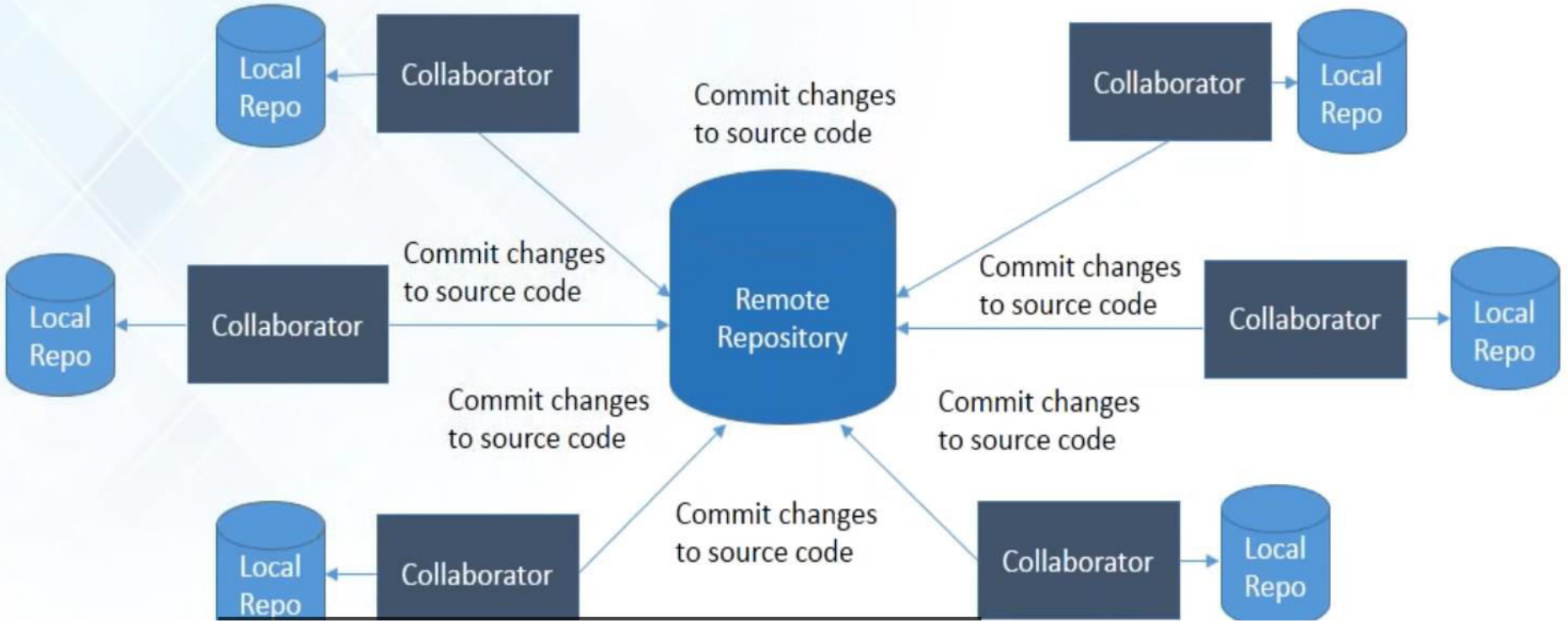
- ❑ They noticed that few of the teams were already using GitHub. Adopting a familiar platform has also made onboarding easier for new employees.
- ❑ Having all of their code in one place makes it easier for them to collaborate on projects.



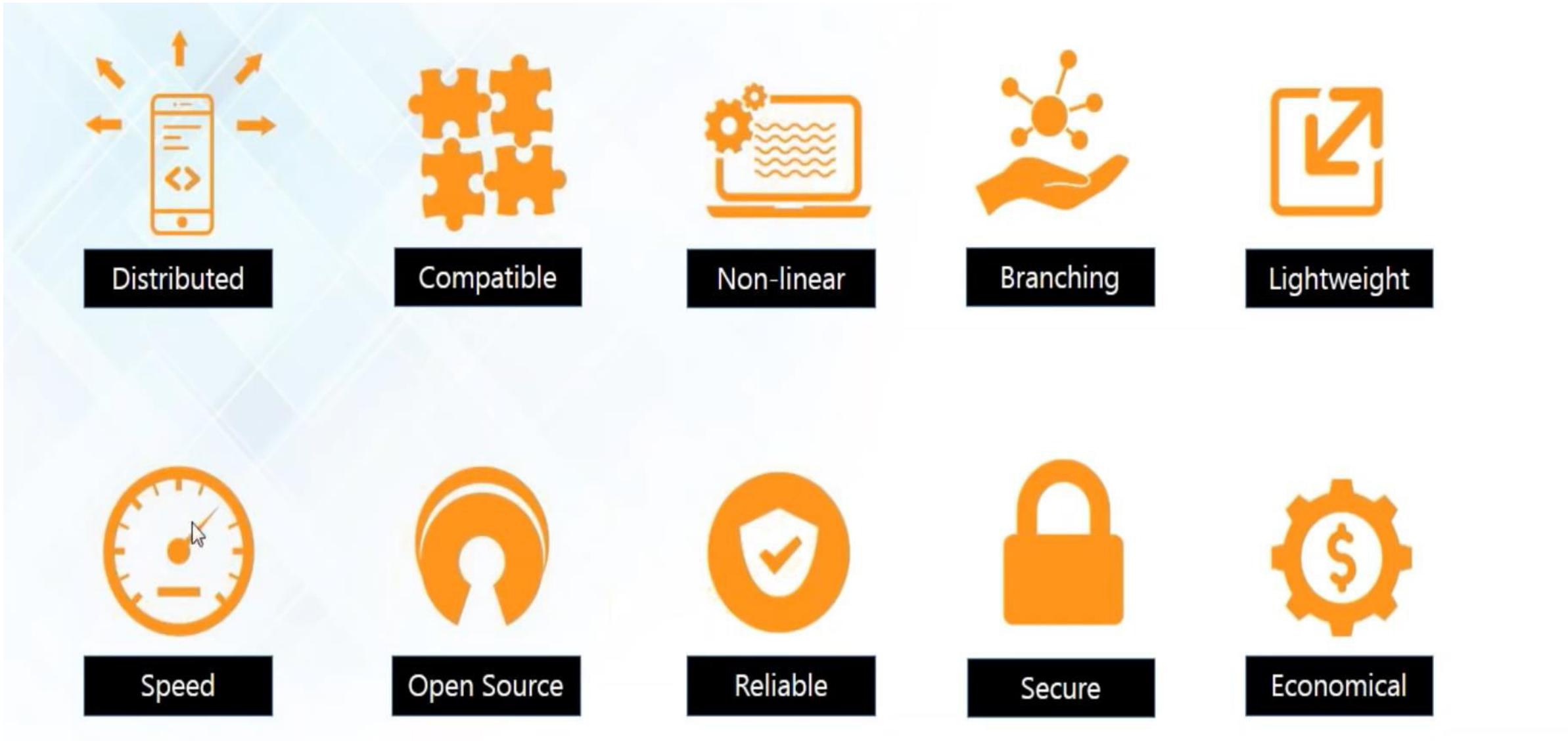
GitHub Enterprise has allowed us to store our company's source code in a central, corporately controlled system. - Joe Fuller, CIO

Git & GitHub

Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software.



Git & GitHub


Distributed
Compatible
Non-linear
Branching
Lightweight
Speed
Open Source
Reliable
Secure
Economical

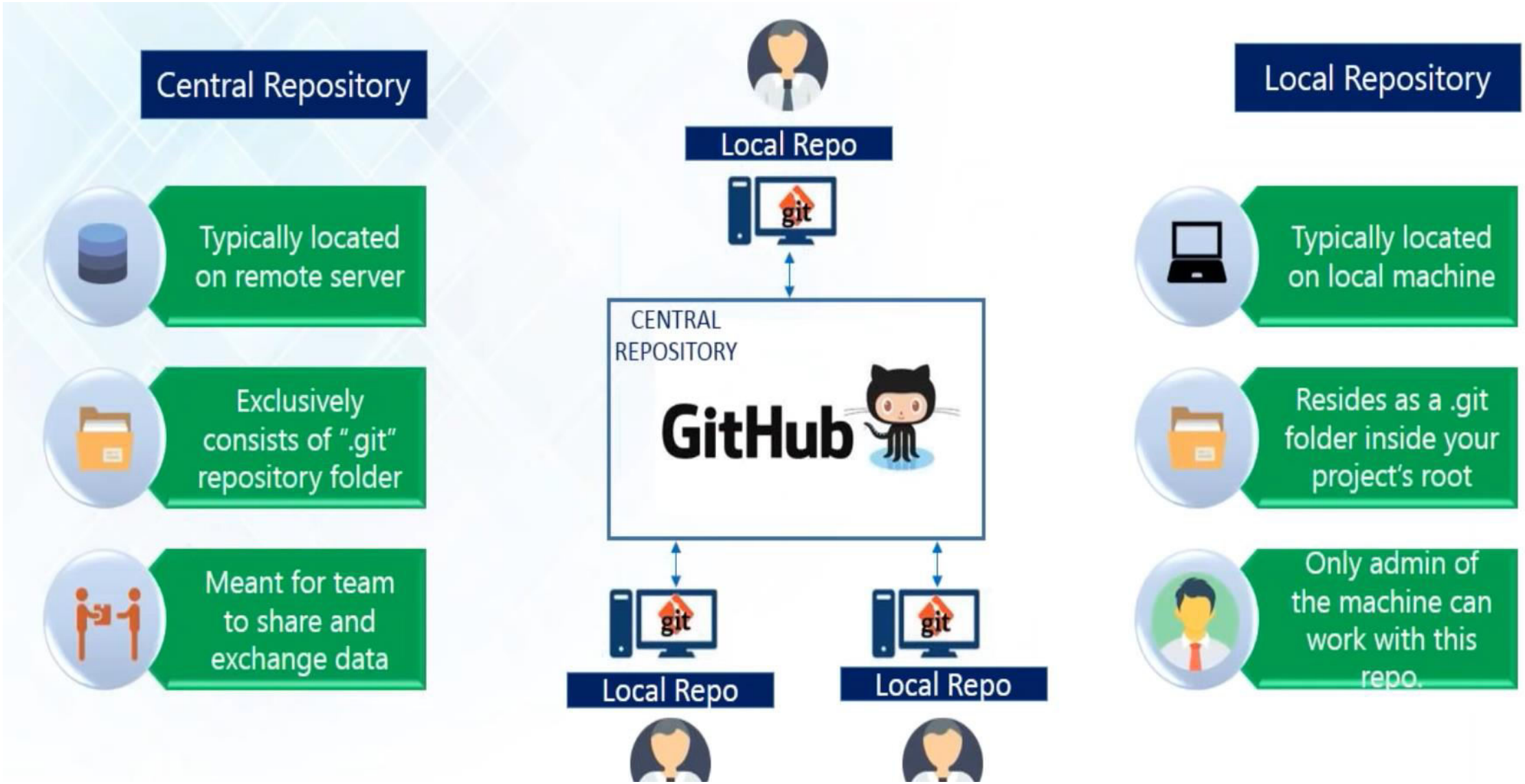
Git & GitHub

A directory or storage space where your projects can live. It can be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, image files, you name it, inside a repository.

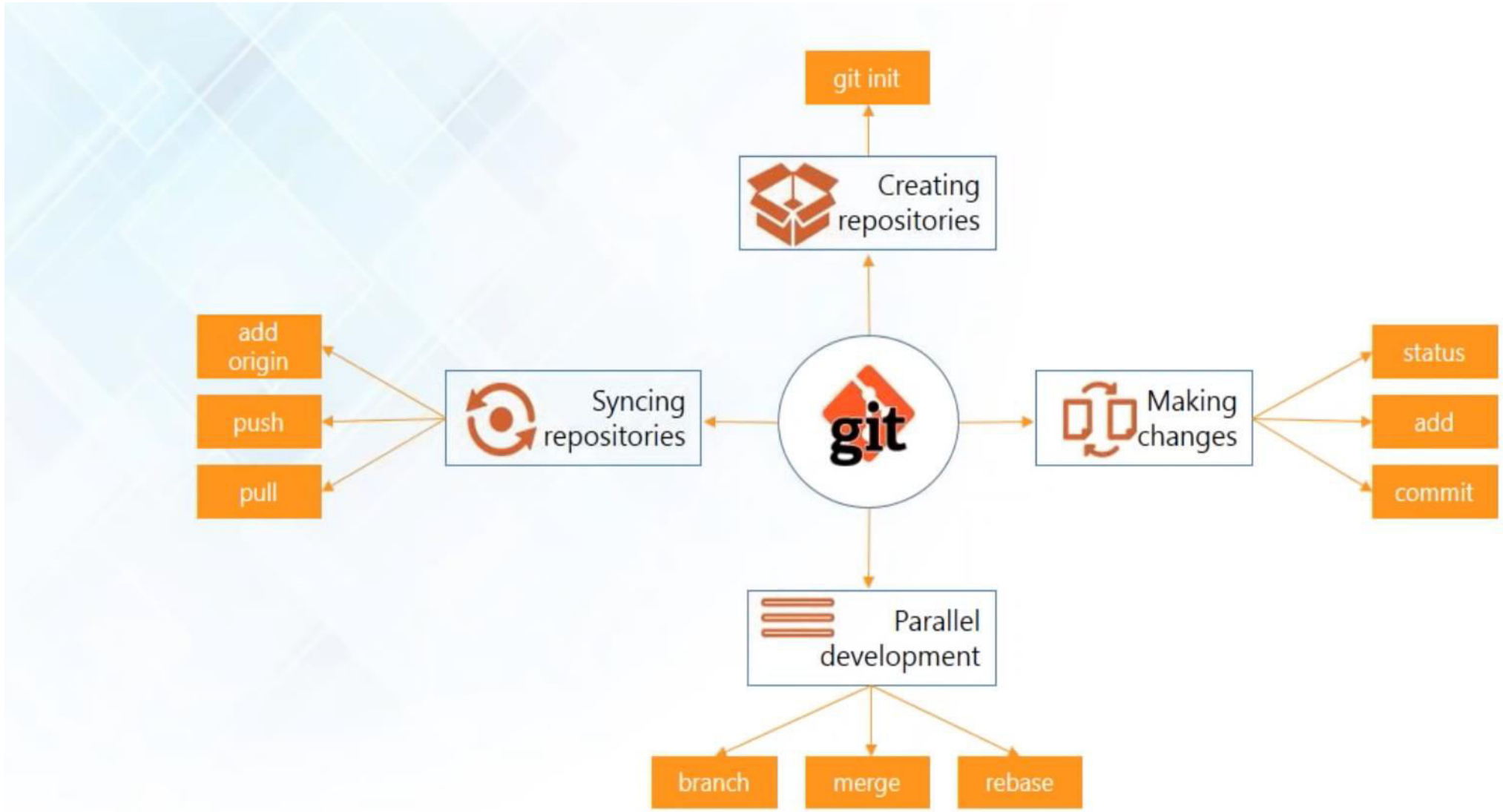
There are two types of repositories:

1. Central Repository
2. Local Repository

Git & GitHub


Central Repository
Typically located on remote server
Exclusively consists of ".git" repository folder
Meant for team to share and exchange data
Local Repo
git
CENTRAL REPOSITORY
GitHub
Local Repo
Local Repo
Local Repository
Typically located on local machine
Resides as a .git folder inside your project's root
Only admin of the machine can work with this repo.

Git & GitHub


git init
Creating repositories
add origin
push
pull
Syncing repositories
git
Making changes
status
add
commit
Parallel development
branch
merge
rebase

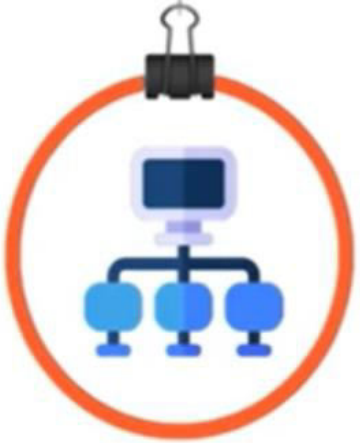
Git & GitHub

Git is a distributed version control tool which is used for managing source code

It's a software tool

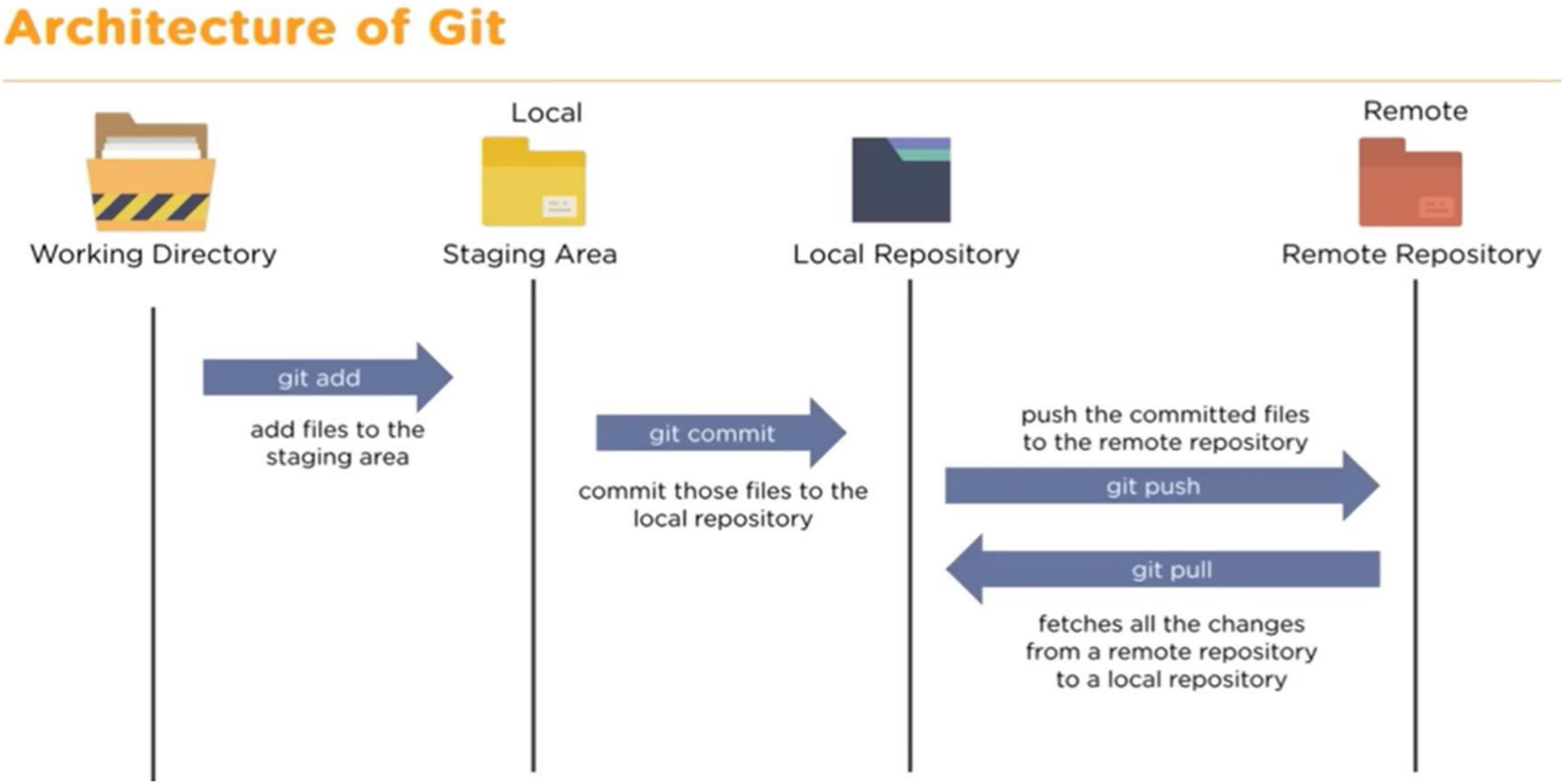
It is used to track changes in the source code

Multiple developers can work together



Supports non-linear development

Git & GitHub


Architecture of Git
Local
Remote
Working Directory
Staging Area
Local Repository
Remote Repository
git add
add files to the staging area
git commit
commit those files to the local repository
push the committed files to the remote repository
git push
git pull
fetches all the changes from a remote repository to a local repository

WELCOME

Git &GitHub

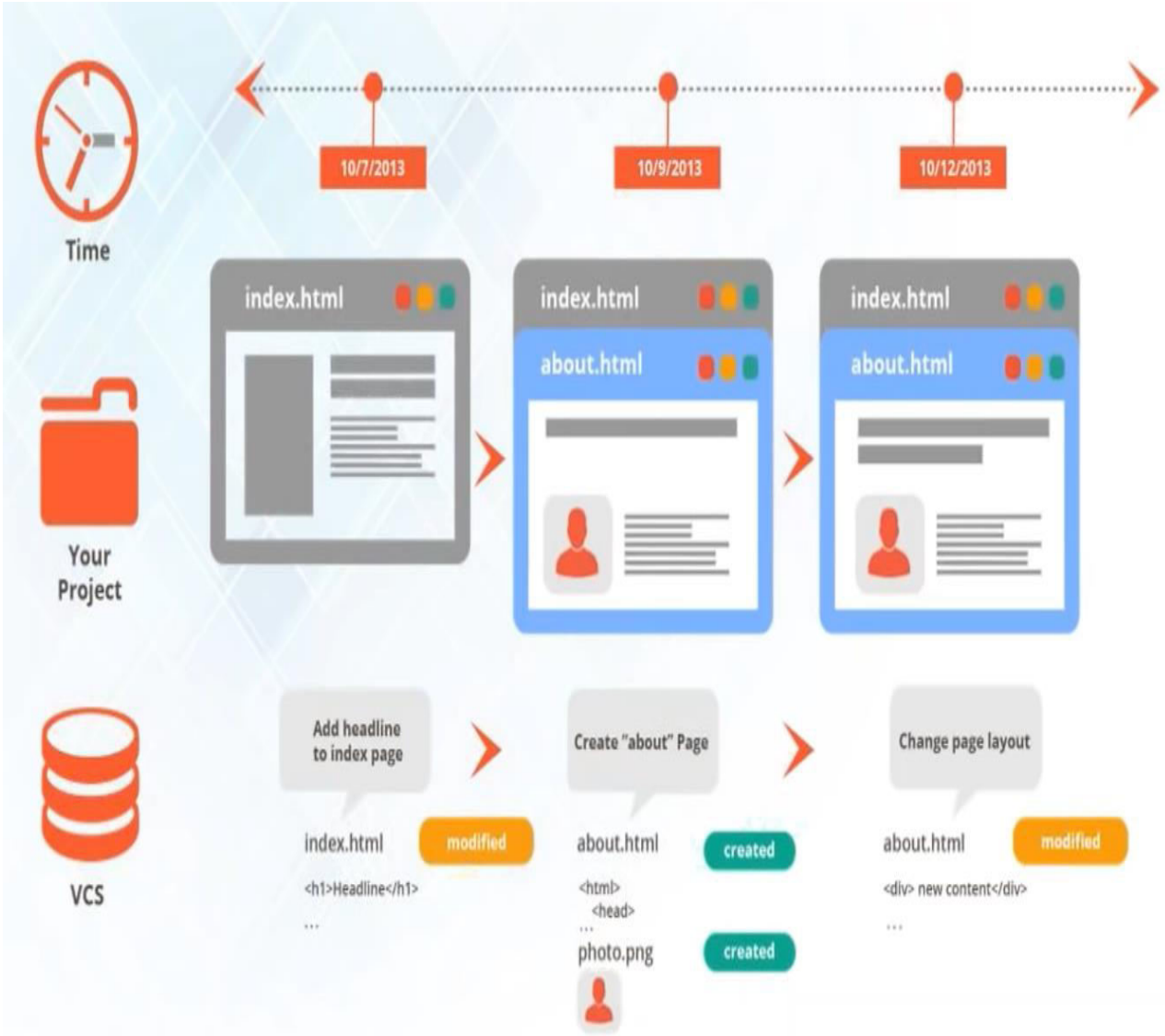
by

Girish Godbole

Git & GitHub

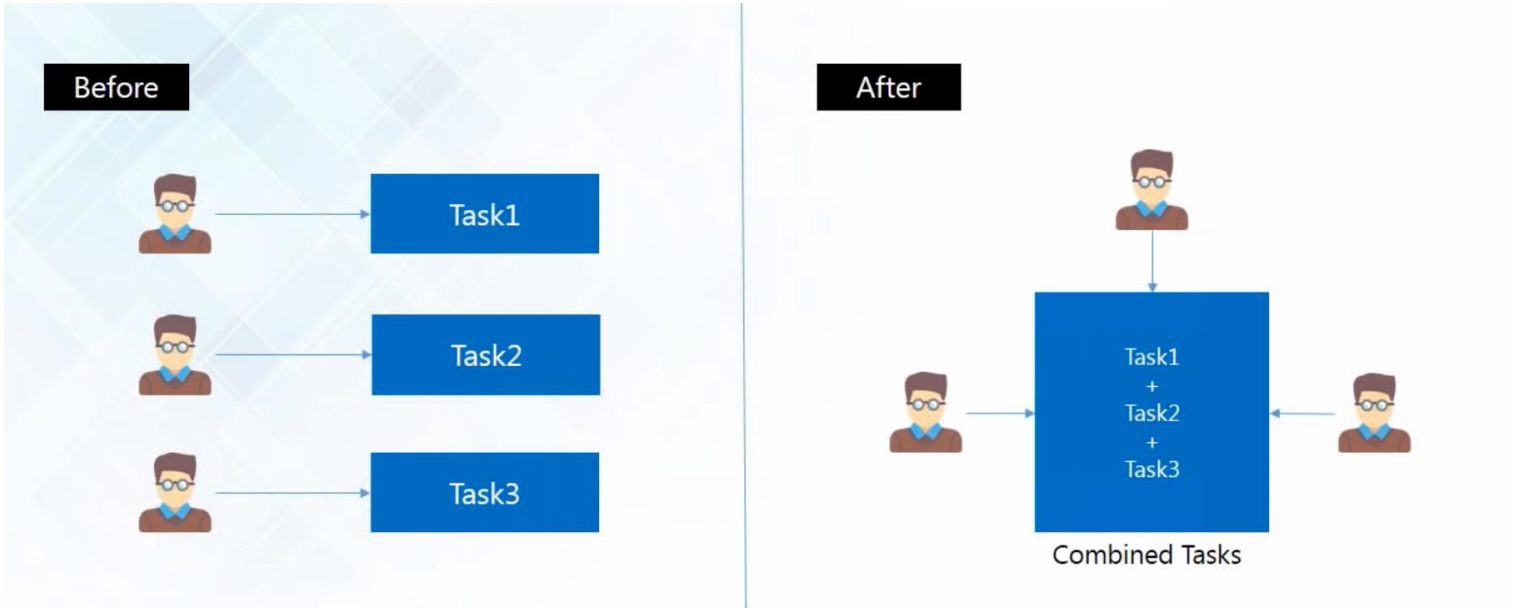
- Version Control – What & Why?
- Version Control Tools
- GitHub & Git
- Case Study: Dominion Enterprises
- Git Features
- Git Operations & Commands

Git & GitHub



- Version control is the management of changes to documents, computer programs, large web sites, and other collections of information.
- These changes are usually termed as "versions".

Git & GitHub

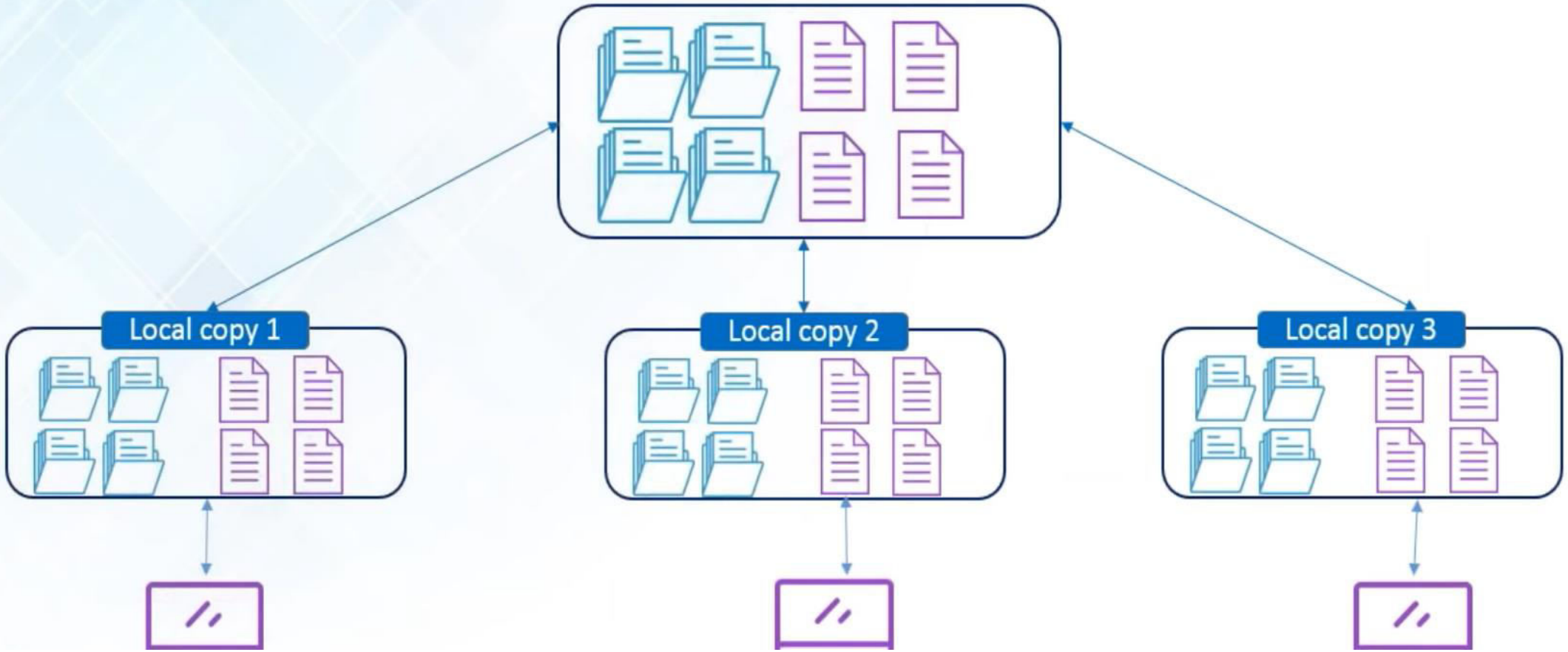

Before
Task1
Task2
Task3
After
Task1
+
Task2
+
Task3
Combined Tasks

Git & GitHub

- Snapshots of all versions are properly documented and stored.
- Versions are also named accurately.

Git & GitHub

In any case if your central server crashes, a backup is always available in your local servers.



Git & GitHub

When you change version -

- VCS provides you with proper description
- What exactly was changed
- When it was changed

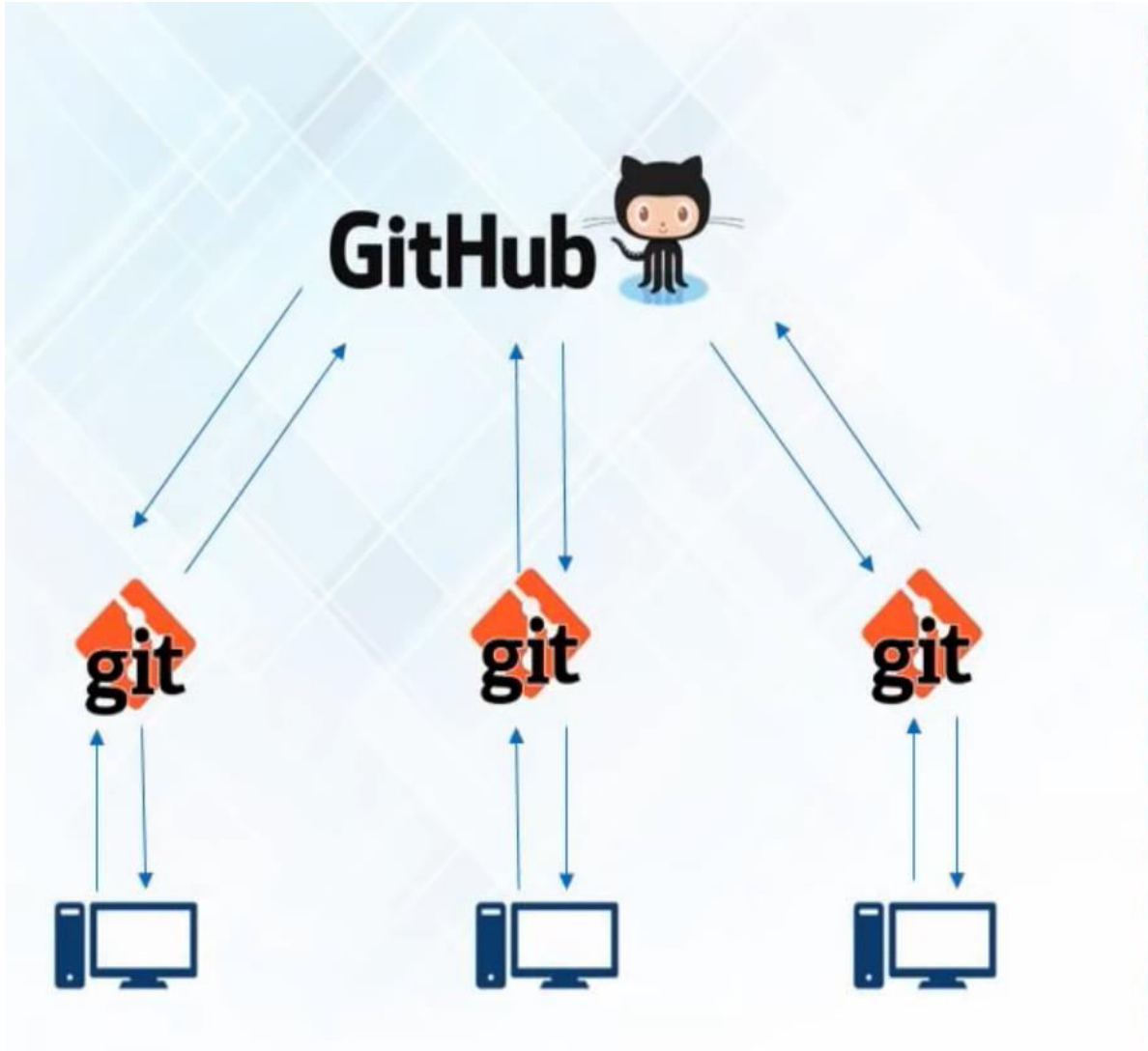
And hence, you can analyze how your project evolved between versions.

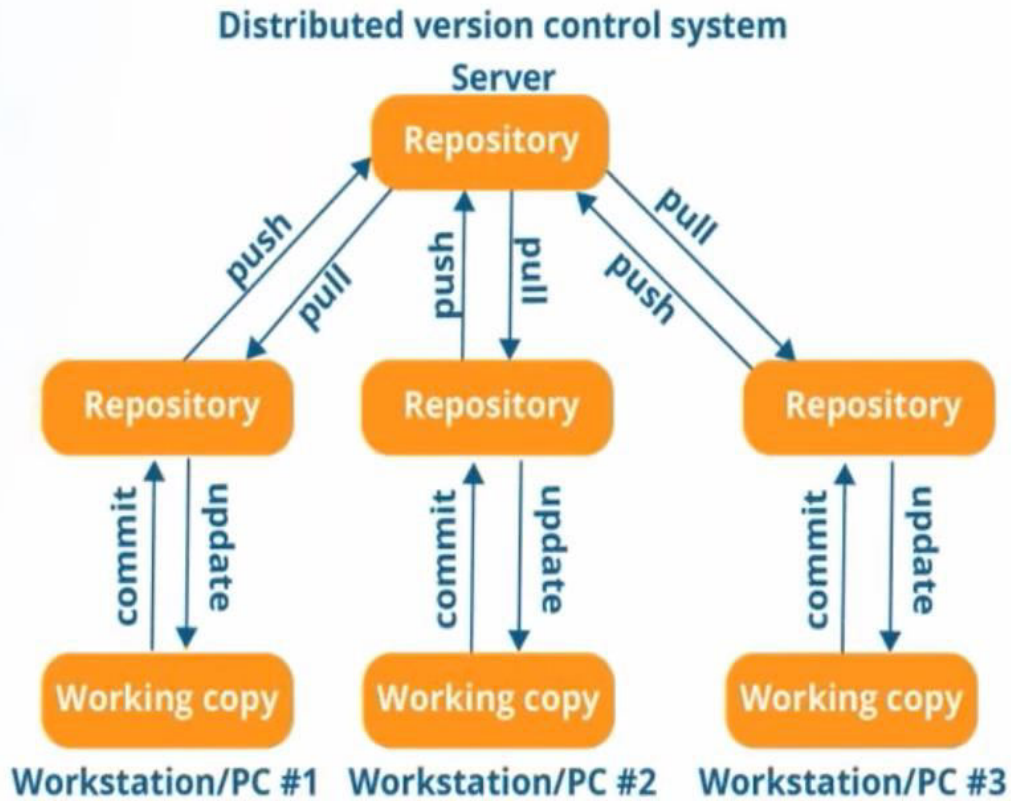
Git & GitHub

Git & GitHub


100
75
50
25
Average
Nov 6, 2011
Jul 21, 2013
Apr 5, 2015
Jan 16,2017
Git
SVN
Mercurial
CVS

Git & GitHub

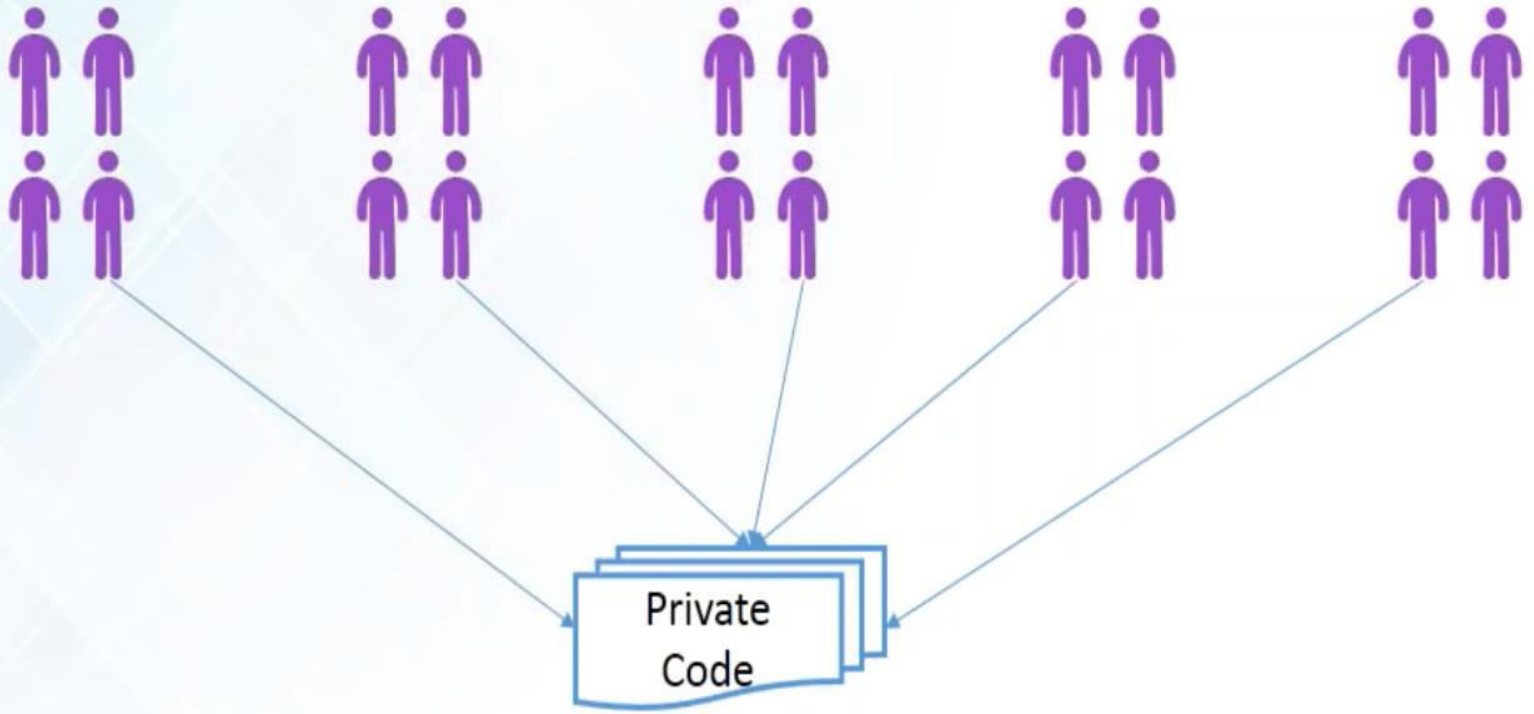

GitHub
git
git
git


Distributed version control system
Server
Repository
push
pull
push
pull
pull
push
Repository
Repository
Repository
commit
update
commit
update
commit
update
Working copy
Working copy
Working copy
Workstation/PC #1
Workstation/PC #2
Workstation/PC #3

Git & GitHub

Problem Statement:

Each team has its own goals, projects, and budgets and they also have Unique needs and workflows

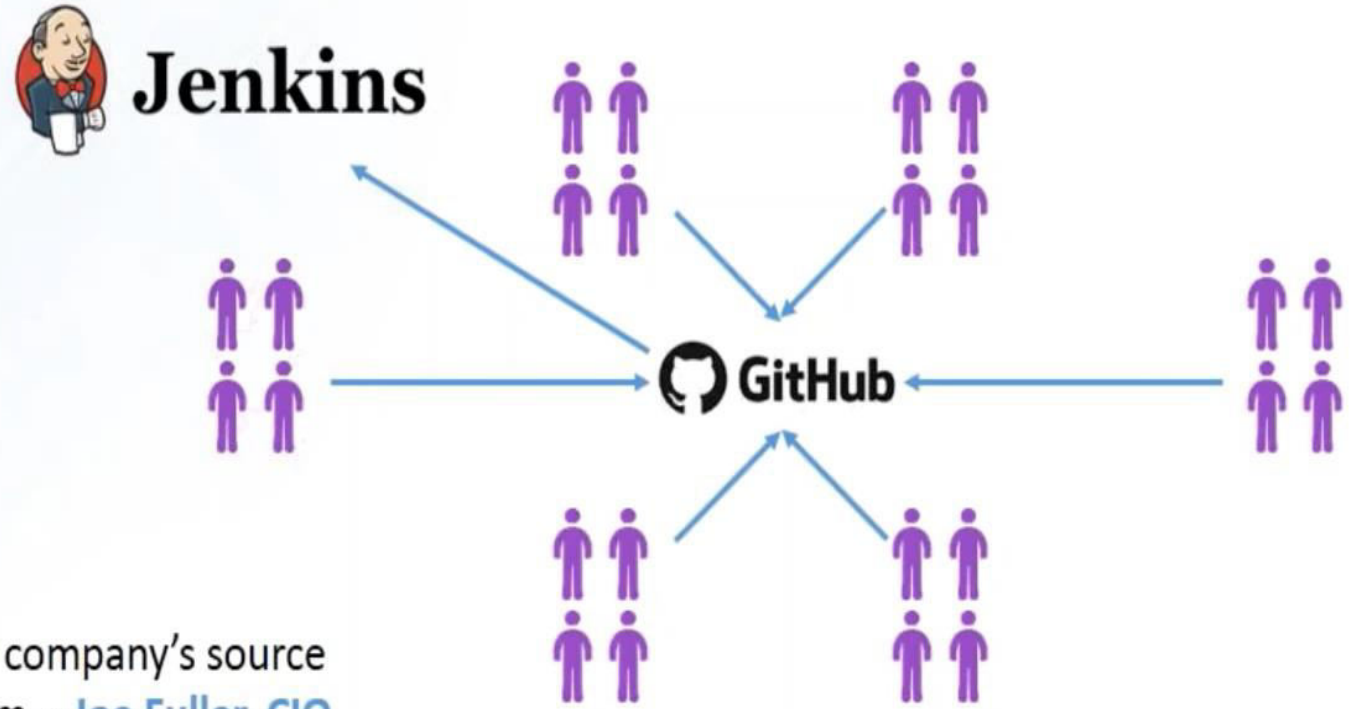


They wanted to make private code “publicly” to make their work more transparent across the company

Git & GitHub

Reason for using GitHub as the solution:

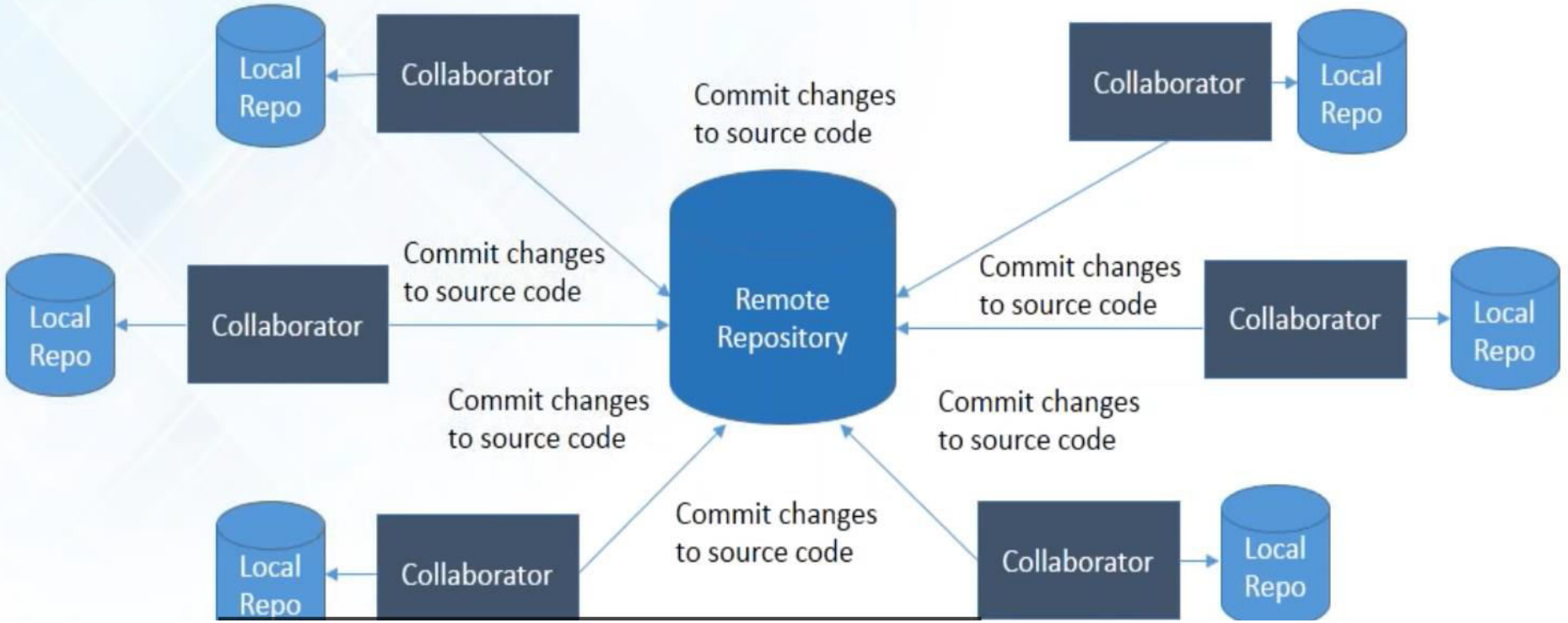
- ❑ They noticed that few of the teams were already using GitHub. Adopting a familiar platform has also made onboarding easier for new employees.
- ❑ Having all of their code in one place makes it easier for them to collaborate on projects.



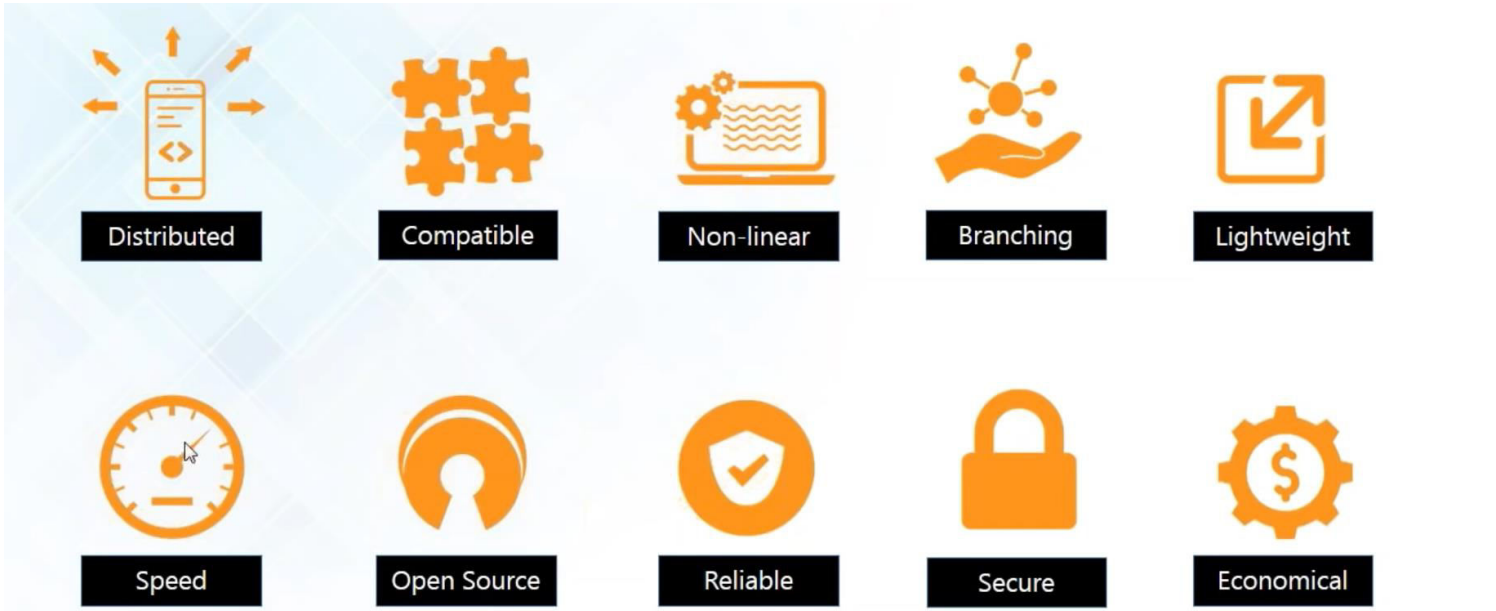
GitHub Enterprise has allowed us to store our company's source code in a central, corporately controlled system. - Joe Fuller, CIO

Git & GitHub

Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software.



Git & GitHub


Distributed
Compatible
Non-linear
Branching
Lightweight
Speed
Open Source
Reliable
Secure
Economical

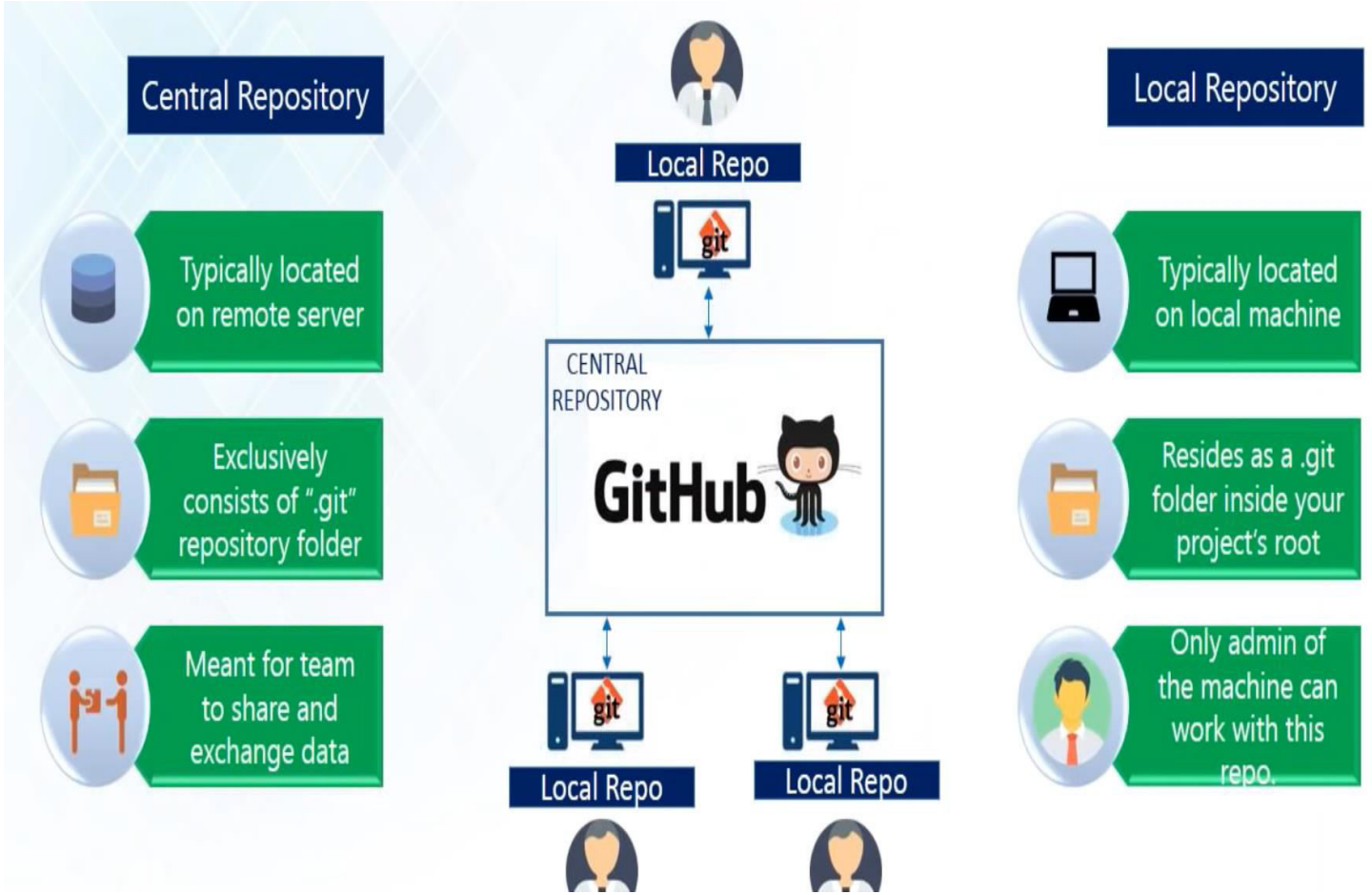
Git & GitHub

A directory or storage space where your projects can live. It can be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, image files, you name it, inside a repository.

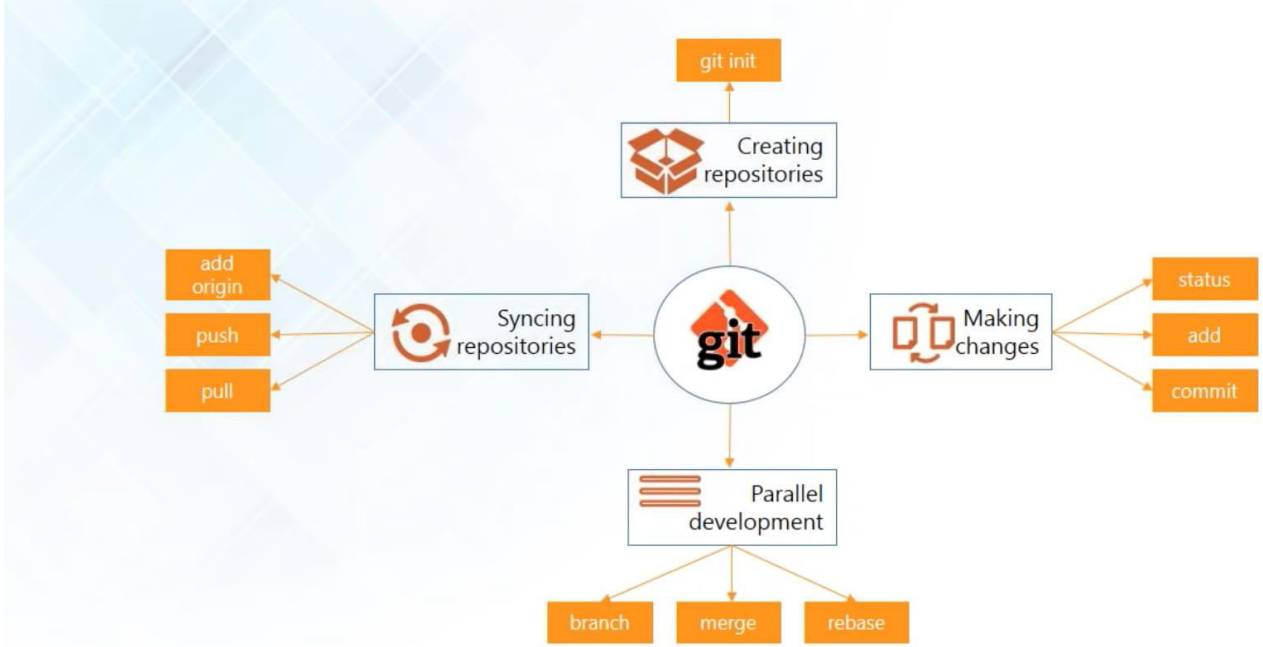
There are two types of repositories:

1. Central Repository
2. Local Repository

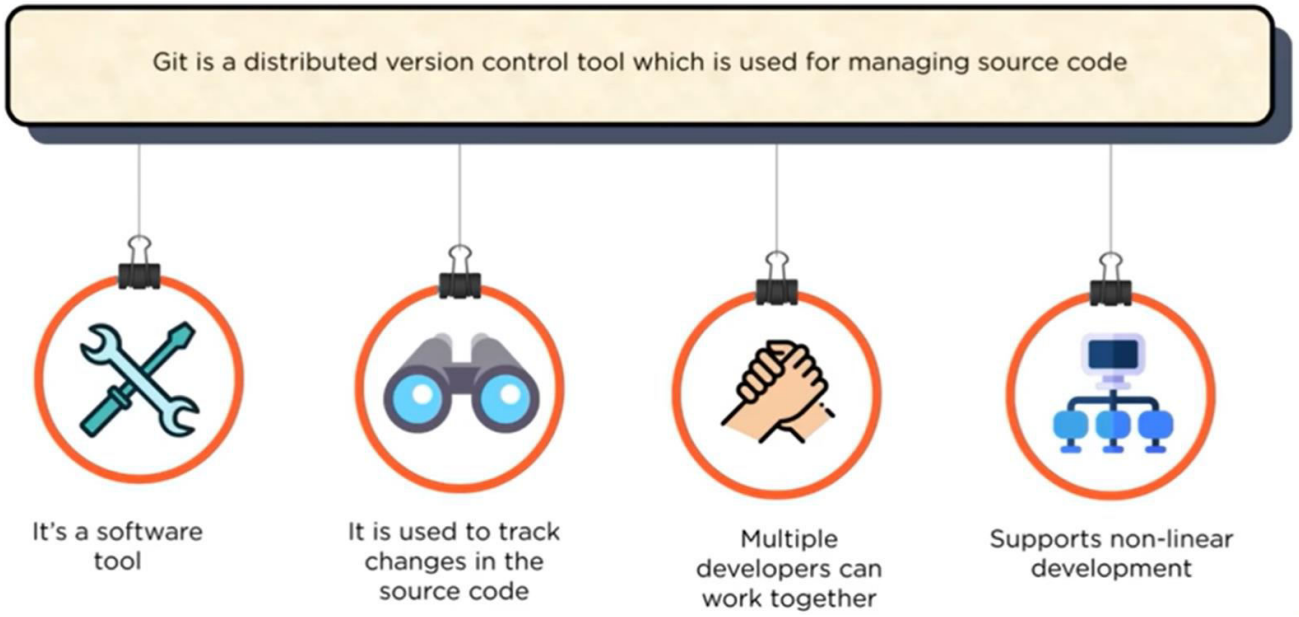
Git & GitHub


Central Repository
Typically located on remote server
Exclusively consists of ".git" repository folder
Meant for team to share and exchange data
Local Repo
git
CENTRAL REPOSITORY
GitHub
Local Repo
Local Repo
Local Repository
Typically located on local machine
Resides as a .git folder inside your project's root
Only admin of the machine can work with this repo.

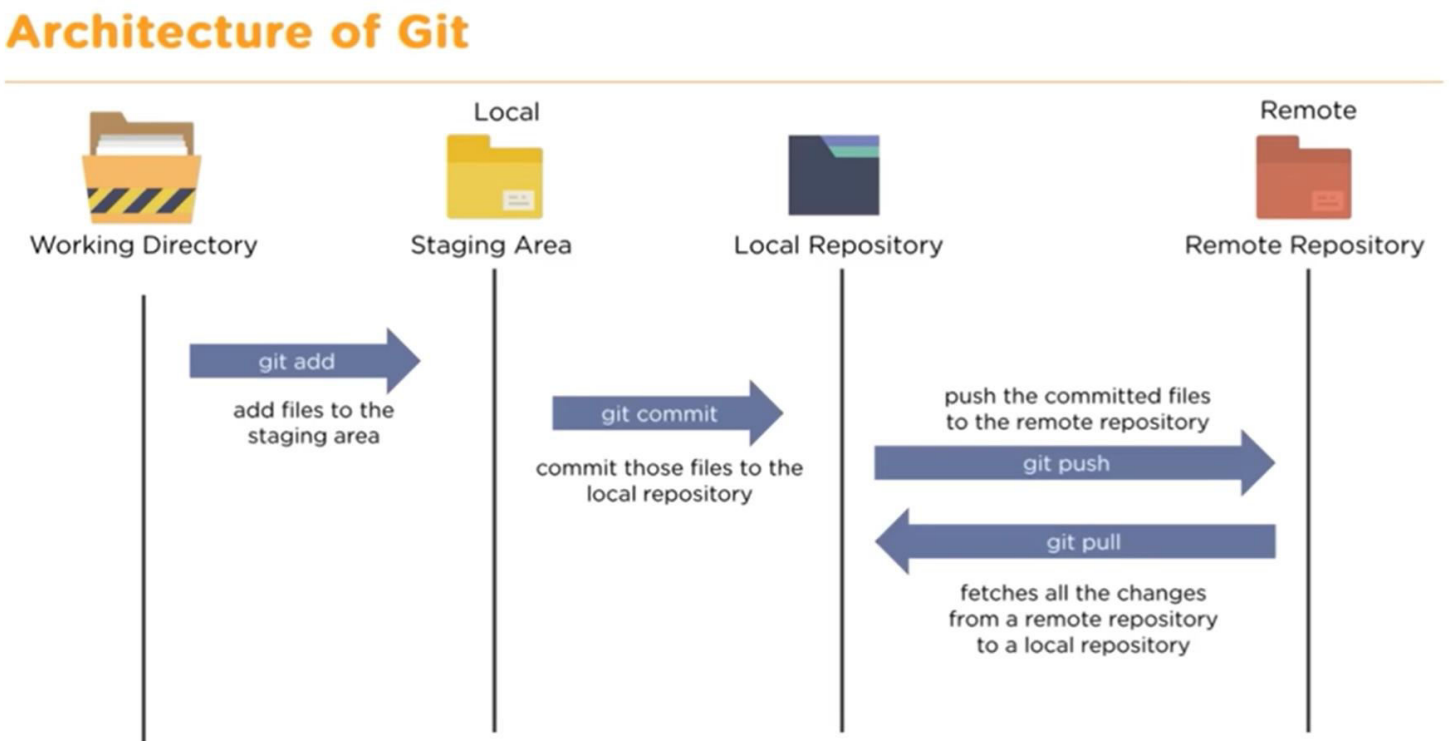
Git & GitHub


git init
Creating repositories
add origin
push
pull
Syncing repositories
git
Making changes
status
add
commit
Parallel development
branch
merge
rebase

Git & GitHub


Git is a distributed version control tool which is used for managing source code
It's a software tool
It is used to track changes in the source code
Multiple developers can work together
Supports non-linear development

Git & GitHub


Architecture of Git
Local
Remote
Working Directory
Staging Area
Local Repository
Remote Repository
git add
add files to the staging area
git commit
commit those files to the local repository
push the committed files to the remote repository
git push
git pull
fetches all the changes from a remote repository to a local repository

Software Engineering Best Practices

by

Girish Godbole

Program Agenda

1. Software Engineering - An Overview
2. Software Development Models
3. Software Requirements Engineering
4. Software Design and Development
5. Software Project Management(SPM)
6. Software Testing Best Practices

Software Engineering - What

Software Engineering - An Overview

Software -

“A set of executable program in an application along with associated libraries and documentations”

“A set of instructions, data or programs used to operate computers and execute specific tasks.”

Engineering -

“On the other hand, is all about developing products, using well-defined, scientific principles and methods.”

Software Engineering -

“Software engineering is the systematic, disciplined and quantifiable approach to the design, development, testing, implementation, and maintenance of application software and computer systems software.”

Broadly, software engineering can be divided into two categories:

Application Engineering - Application engineering is the process to develop software applications for businesses and organizations

Systems Engineering - System engineering refers to the coordination of the development and maintenance of computer systems for businesses and organizations.

Software Engineering - Importance

Large software – In our real life, it is quite more comfortable to build a wall than a house or building. In the same manner, as the size of the software becomes large, software engineering helps you to build software.

Scalability- If the software development process were based on scientific and engineering concepts, it is easier to re-create new software to scale an existing one.

Adaptability: Whenever the software process was based on scientific and engineering, it is easy to re-create new software with the help of software engineering.

Cost- Hardware industry has shown its skills and huge manufacturing has lower the cost of the computer and electronic hardware.

Dynamic Nature– Always growing and adapting nature of the software. It depends on the environment in which the user works.

Quality Management: Offers better method of software development to provide quality software products.

The importance of software engineering is growing day by day as people require a wide range of applications to efficiently manage their businesses and increase productivity. Software engineering helps to handle complex and big projects by applying the principle of modularity. It divides complex projects into different modules and allows developers to work independently on each module. Software engineering itself is a process of developing a perfect plan for software development. Therefore, it reduces the time and cost required to develop software.

Software Engineering - Challenges

Critical Challenges

- In safety-critical areas such as space, aviation, nuclear power plants, etc. the cost of software failure can be massive because lives are at risk.
- Increased market demands for fast turnaround time.
- Dealing with the increased complexity of software need for new applications.
- The diversity of software systems should be communicating with each other.

Characteristics of Good Software

Every software must satisfy the following attributes:

- Operational
- Transitional
- Maintenance

Operational

This characteristic let us know about how well software works in the operations which can be measured on:

- Budget, Efficiency, Usability, Dependability, Correctness, Functionality, Safety, Security

Transitional

This is an essential aspect when the software is moved from one platform to another:

- Interoperability, Reusability, Portability, Adaptability

Maintenance

This aspect talks about how well software has the capabilities to adapt itself in the quickly changing environment:

- Flexibility, Maintainability, Modularity, Scalability

Software Engineering -Objectives

- **Maintainability –** It should be feasible for the software to evolve to meet changing requirements.
- **Efficiency –** The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
- **Correctness –** A software product is correct if the different requirements as specified in the SRS document have been correctly implemented.
- **Reusability –** A software product has good reusability if the different modules of the product can easily be reused to develop new products.
- **Testability –** Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria
- **Reliability –** It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
- **Portability –** In this case, the software can be transferred from one computer system or environment to another.
- **Adaptability –** The software allows differing system constraints
- **Interoperability** – Capability of 2 or more functional units to process data cooperatively.

Software Engineering - Classification of Software

Types of Project /Classification of Software

- Web based project development
- Standalone/Desktop based programs/applications
- Jobs/Schedules
- Enterprise projects (ERP systems)
- Database oriented applications
- Reporting/Analytics applications
- Artificial Intelligence Software
- Scientific Software
- Embedded Software
- Cloud based projects

Software Engineering – Model Selection

Which Model to select ?

Before selecting any model, there are few questions which should be addressed other than looking into the differences in the approach of any model. These questions will not just help to find the model but will also play a role in finding the best according to your demands. These questions are:

- What is the size of the project you are going to work on?
- Do you need a prototype?
- Are you using a packaged solution?
- How flexible is your team?
- How much will your customer participate in the process?
- Does your project manager have experience?

Above mentioned important queries need to be answered before going any step ahead.

Software Development Life Cycle –SDLC Model

Software Development Life Cycle –SDLC Model

SDLC –

1. A framework that describes the activities performed at each stage of a software development project.

2. Is a systematic process for building software that ensures the quality and correctness of the software built.

Every phase of the SDLC life Cycle has its own process and deliverables that feed into the next phase.

Why

- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

Software Development Life Cycle –SDLC Model

Phase 1: Requirement collection and analysis

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study

This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

- **Economic:** Can we complete the project within the budget or not?
- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software
- **Schedule:** Decide that the project can be completed within the given schedule or not.

Software Development Life Cycle –SDLC Model

Phase 3: Design

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture. This design phase serves as input for the next phase of the model. There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design(LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

Software Development Life Cycle –SDLC Model

Phase 4: Coding

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Software Development Life Cycle –SDLC Model

Phase 6: Installation/Deployment

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance

Once the system is deployed, and customers start using the developed system, following 3 activities occur

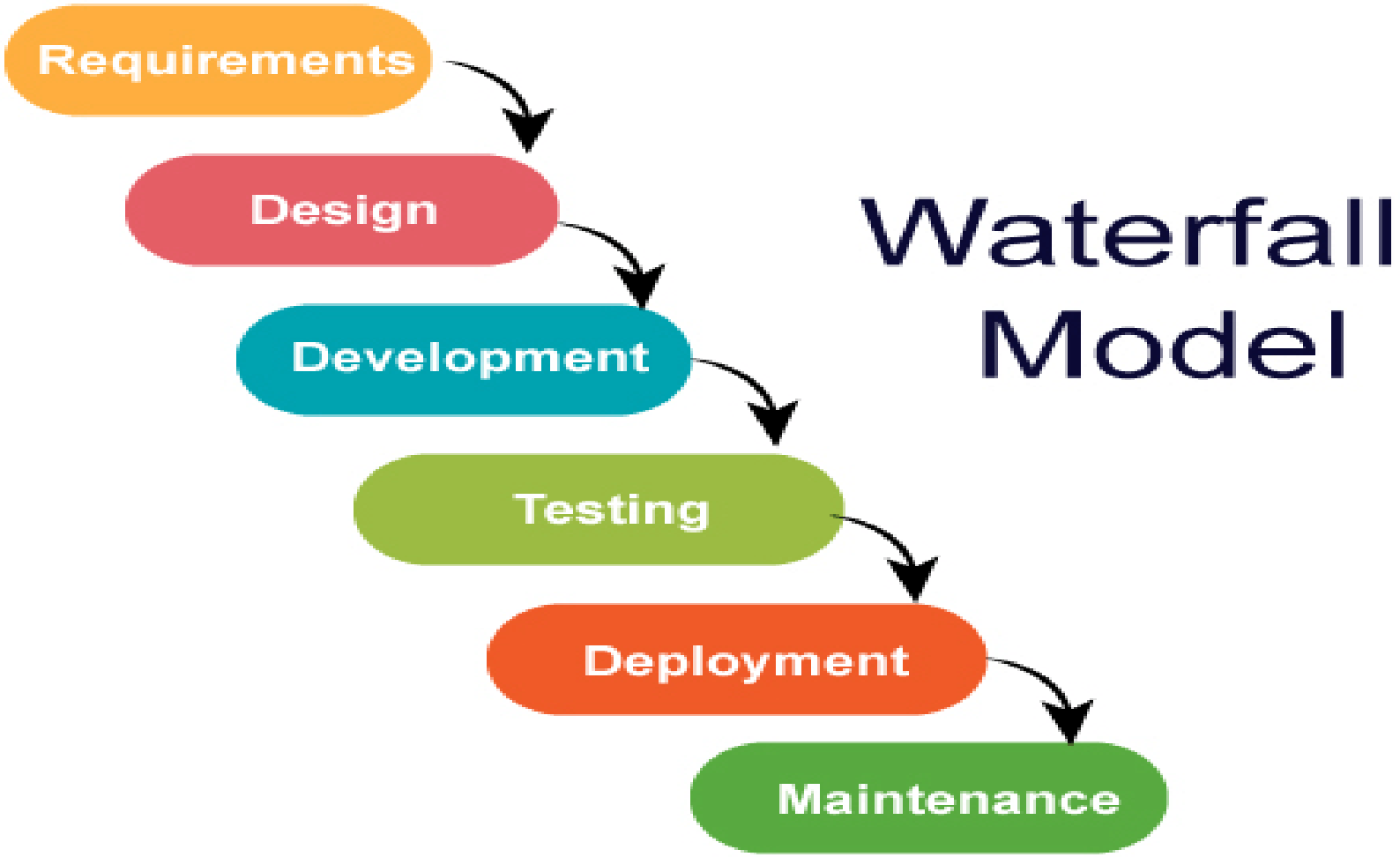
Bug fixing – bugs are reported because of some scenarios which are not tested at all

Upgrade – Upgrading the application to the newer versions of the Software

Enhancement – Adding some new features into the existing software

The focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

SDLC – Waterfall Model



SDLC – Waterfall Model

Waterfall Strengths

Easy to understand, easy to use
Provides structure to inexperienced staff
Milestones are well understood
Sets requirements stability
Good for management control (plan, staff, track)
Works well when quality is more important than cost or schedule

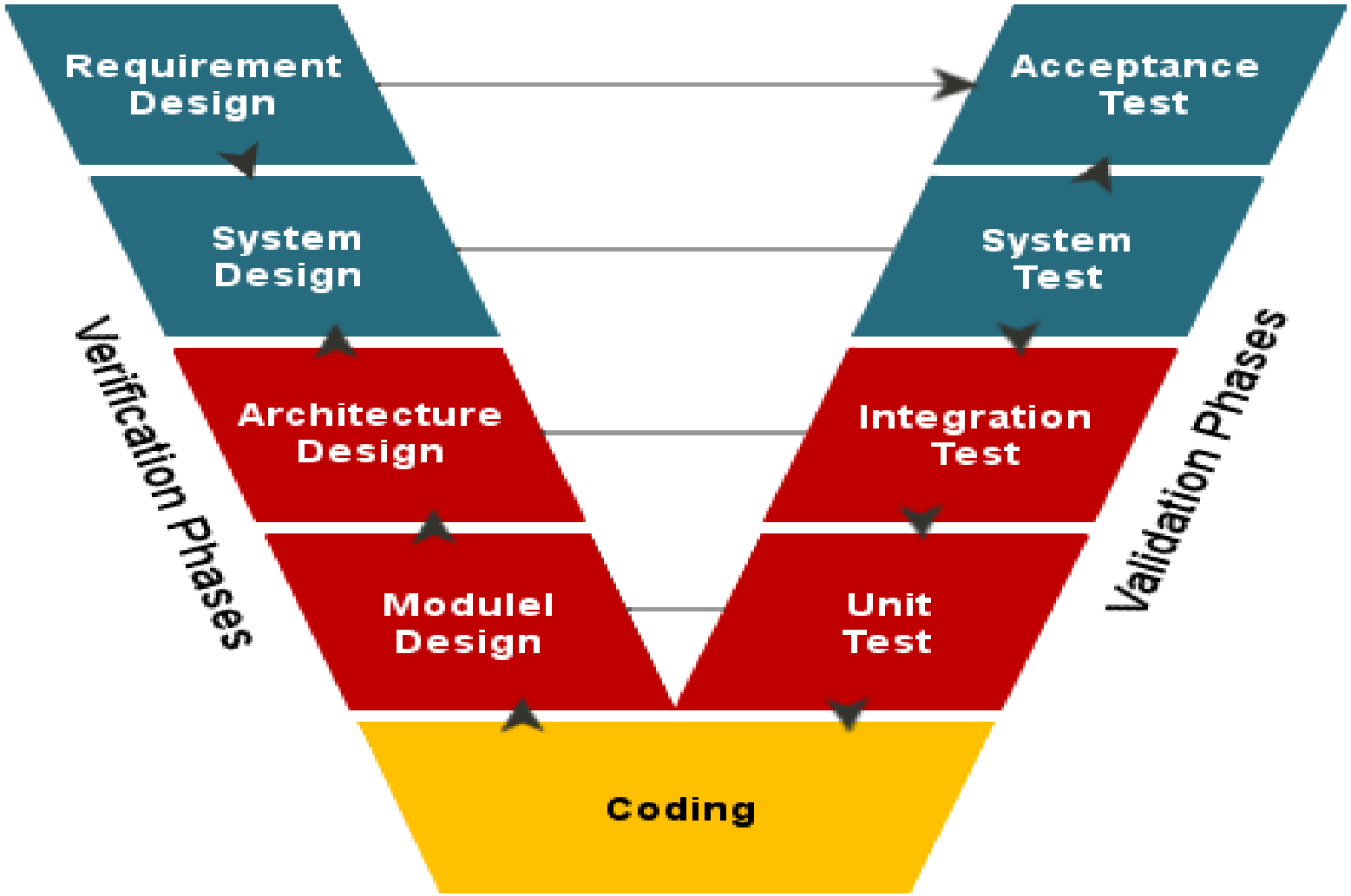
Waterfall Deficiencies

All requirements must be known upfront
Deliverables created for each phase are considered frozen – inhibits flexibility
Not ideal for long, complex, or ongoing projects where more flexibility is required
Does not reflect problem-solving nature of software development – iterations of phases
Integration is one big bang at the end
Little opportunity for customer to preview the system (until it may be too late)

When to use

Requirements are very well known
Product definition is stable. Technology is understood
New version of an existing product. Porting an existing product to a new platform.
Useful for small, one-time projects with few requirements and short timelines because it is simple and can be quickly configured

SDLC – V Model


Requirement Design
System Design
Architecture Design
Modulel Design
Coding
Unit Test
Integration Test
System Test
Acceptance Test
Verification Phases
Validation Phases

SDLC – V Model

V Model Strengths
Emphasize planning for verification and validation of the product in early stages of product development
Each deliverable must be testable
Project management can track progress by milestones
Easy to use

V Model Weakness
Does not easily handle concurrent events
Does not handle iterations or phases
Not ideal for long, complex, or ongoing projects
Does not easily handle dynamic changes in requirements
Does not contain risk analysis activities

When to use
Excellent choice for systems requiring high reliability – hospital patient control applications
All requirements are known up-front
When it can be modified to handle changing requirements beyond analysis phase
Solution and technology are known

Verification Vs. Validation

Verification	Validation
Are we building the system, right?	Are we building the right system?
Verification process includes checking of documents, design, code and program	Validation process includes testing and validation of the actual product.
Verification is the static testing.	Validation is the dynamic testing.
Verification uses methods like reviews, walkthroughs, inspections and desk-checking	Validation uses methods like black box testing, white box testing and non-functional testing.
Verification does not involve code execution	Validation involves code execution.
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
Cost of errors caught in Verification is less than errors found in Validation.	Cost of errors caught in Validation is more than errors found in Verification.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.

Static Testing Vs. Dynamic Testing

Static Testing	Dynamic Testing
It is performed in the early stage of the software development.	It is performed at the later stage of the software development.
In static testing whole code is not executed.	In dynamic testing whole code is executed.
Static testing prevents the defects.	Dynamic testing finds and fixes the defects.
Static testing is performed before code deployment.	Dynamic testing is performed after code deployment.
Static testing is less costly.	Dynamic testing is highly costly.
Static Testing involves checklist for testing process.	Dynamic Testing involves test cases for testing process.
It includes walkthroughs, code review, inspection etc.	It involves functional and nonfunctional testing.
It generally takes shorter time.	It usually takes longer time as it involves running several test cases.
It can discover variety of bugs.	It expose the bugs that are explorable through execution hence discover only limited type of bugs.
Static Testing may complete 100% statement coverage in comparably less time.	While dynamic testing only achieves less than 50% statement coverage.
Verification	Validation

Software Engineering – Iterative Model

In a practical software development project, the old waterfall model is not going to be a big help. It has a lot of errors in it when it comes to bigger and complex projects. So, Iterative can be a good alternative. In the Iterative model, the development begins by specifying and implementing part of the software which you will review later to identify the further requirements. According to the iterative model, you can make software by using some of the software specifications and develop the first version of the software. And later, if you or your client realize that you need some modification than it can be easily made using a new iteration.

The process of Iterative model is cyclic, once the initial planning is complete, few of the phases are kept repeating repeatedly, with the completion of each cycle incrementally improving and iterating on the software.

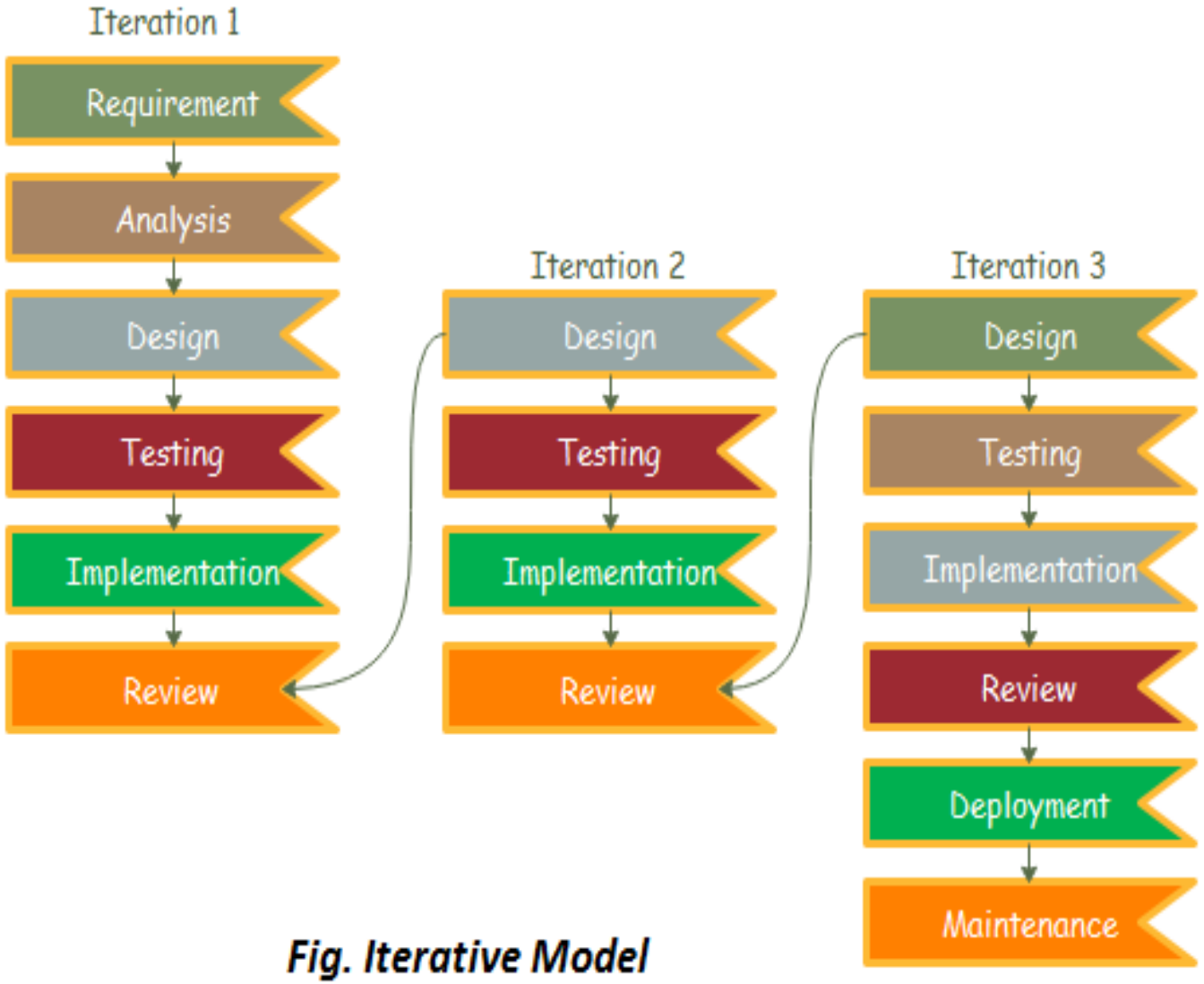


Fig. Iterative Model

Software Engineering – Iterative Model

Advantage(Pros) of Iterative Model:

1. Testing and debugging during smaller iteration is easy.
2. A Parallel development can plan.
3. It is easily acceptable to ever-changing needs of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.

Disadvantage(Cons) of Iterative Model:

1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.

The Iterative model can pose more risks with frequent changes, unknown costs and resource requirements, and uncertain deadlines

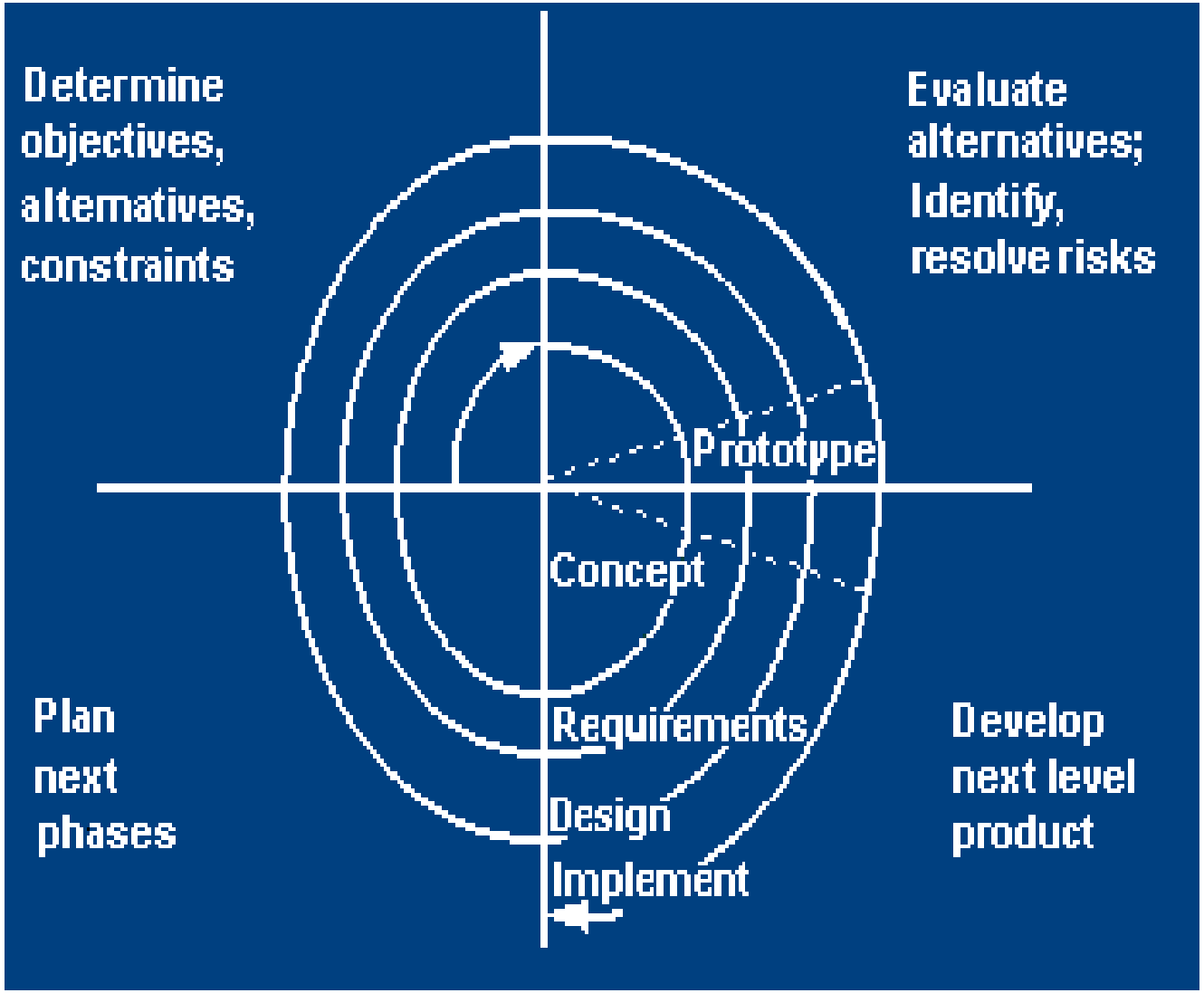
When to use the Iterative Model?

1. When requirements are defined clearly and easy to understand.
2. When the software application is large.
3. New technology involved
4. Being learned by the development team
5. Big project with high-risk features and goals will change later on.

SDLC – Spiral Model

Spiral Model is a risk-driven software development process model. It is a combination of waterfall model and iterative model.
Spiral Model helps to adopt software development elements of multiple process models for the software project based on unique risk patterns ensuring efficient development process.
Each phase of spiral model in software engineering begins with a design goal and ends with the client reviewing the progress.

The development process in Spiral model in SDLC, starts with a small set of requirement and goes through each development phase for those set of requirements.
The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.



SDLC – Spiral Model

Spiral Model Strengths

Provides early indication of insurmountable risks, without much cost
Users see the system early because of rapid prototyping tools
Critical high-risk functions are developed first
The design does not have to be perfect
Users can be closely tied to all lifecycle steps
Early and frequent feedback from users
Cumulative costs assessed frequently

Spiral Model weakness

Time spent for evaluating risks too large for small or low-risk projects
Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
The model is complex, Risk assessment expertise is required
Spiral may continue indefinitely, Developers must be reassigned during non-development phase activities
May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

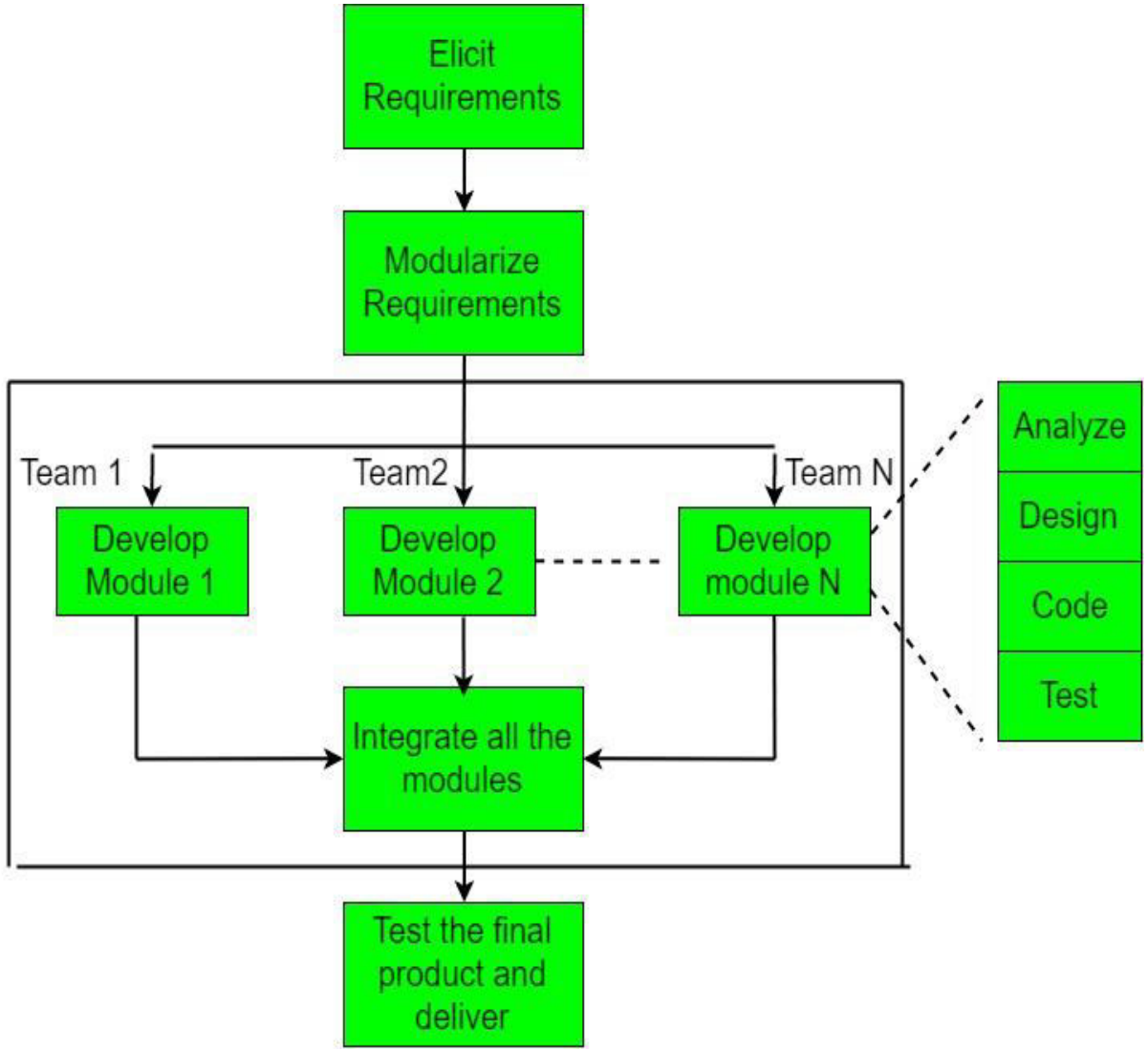
When to use

A Spiral model in software engineering is used when project is large
When creation of a prototype is applicable and appropriate
When costs and risk evaluation is important, Significant changes are expected (research and exploration)
For medium to high-risk projects.
Long-term project commitment unwise because of potential changes to economic priorities
Users are unsure of their needs, Requirements are complex, New product line

Software Engineering – RAD Model

RAD is completely different from any other types of engineering because changes can be made almost instantaneously, and it even gives the edge to make amendments at the end of the development process. The key benefit of the rapid application development approach is fast turn out of the project and it really attracts the developers as it is working in a fast-paced environment which is ideal for software development. By minimizing the planning stage and maximizing the prototype development, the rapid pace is achieved in the **RAD Model**

By this approach, it has become easy for the stakeholders and the project managers to accurately measure progress and communicate in real time to focus on changes and take care of error as soon as possible. That brings efficiency, faster development, and effective communication.



Software Engineering – RAD Model

Advantages of Rapid Application Development

- RAD breaks the project into smaller manageable tasks.
- Clients get the best version in a shorter period.
- This model is flexible for change.
- It increases the reusability of features, It reduced development time.
- Regular communication and constant feedback increases the efficiency of the design and build process.

Disadvantages of Rapid Application Development

- In the hurry of completing everything timely, it is very common to leave something important unattended.
- No so cost friendly, depending on the iterations
- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.
- On the high technical risk, it's not suitable.

When to use RAD Model?

- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known, When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

Software Engineering – Agile vs RAD Model

Agile model	RAD model
The Agile model does not recommend developing prototypes but emphasizes the systematic development of each incremental feature at the end of each iteration.	The central theme of RAD is based on designing quick and dirty prototypes, which are then refined into production quality code.
Agile projects logically break down the solution into features that are incrementally developed and delivered.	The developers using the RAD model focus on developing all the features of an application by first doing it badly and then successively improving the code over time.
The Agile team only demonstrate completed work to the customer after each iteration.	Whereas RAD teams demonstrate to customers screen mock up and prototypes, that may be based on simplifications such as table lookup rather than actual computations.
Agile model is not suitable for small projects as it is difficult to divide the project into small parts that can be incrementally developed.	When the company has not developed a almost similar type of project, then it is hard to use RAD model as it is unable to reuse the existing code.

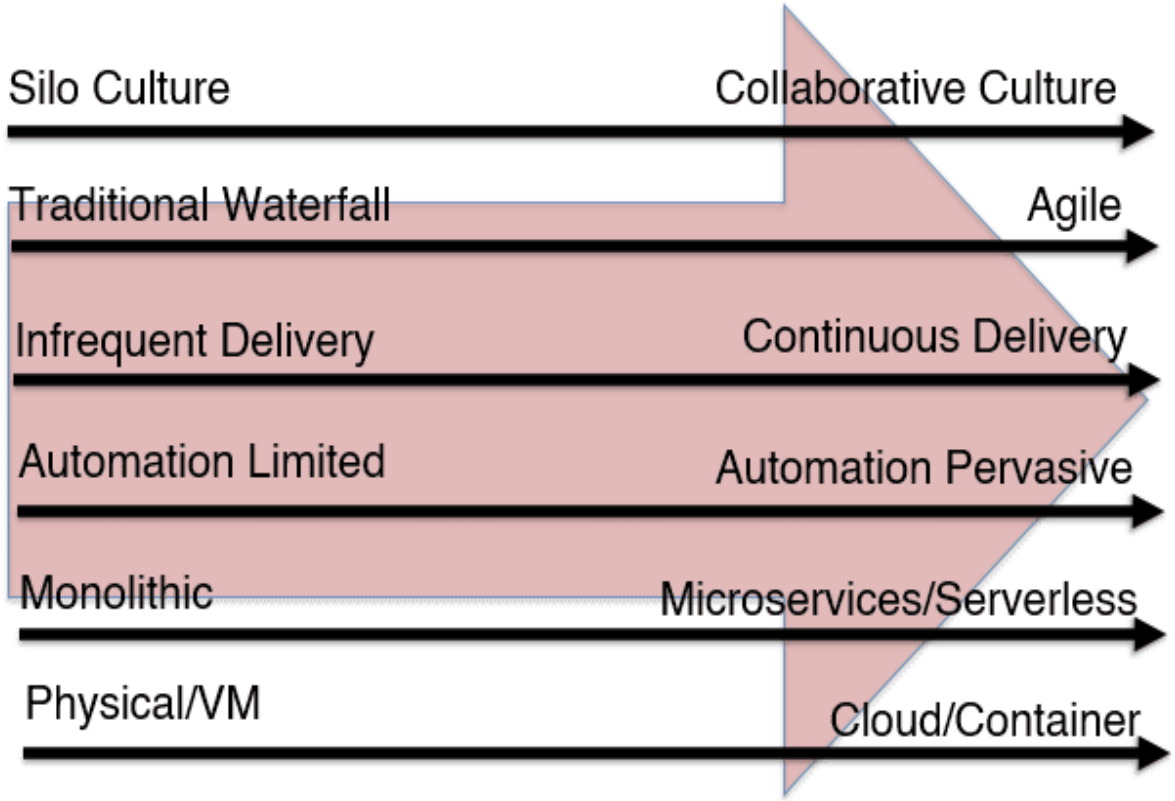
The Shift to Agile Software Development

Every model was created with the purpose of improving the software development and delivery process. Today, every software development model works well for specific types of projects. However, older manual models, like Waterfall, are quickly becoming a thing of the past.

IT teams, and businesses at large, must move faster and more effectively to deliver software, please their end-users, and keep up with the competition. A faster, repeatable, and more secure software development process is anchored in automation.

This level of automation and speed is not attainable with many models. As a result, the Agile methodology has become increasingly popular.

Companies are adopting enabling technologies and processes like continuous integration, continuous delivery, release automation, and DevOps.



Requirement Engineering

What is Requirement ?

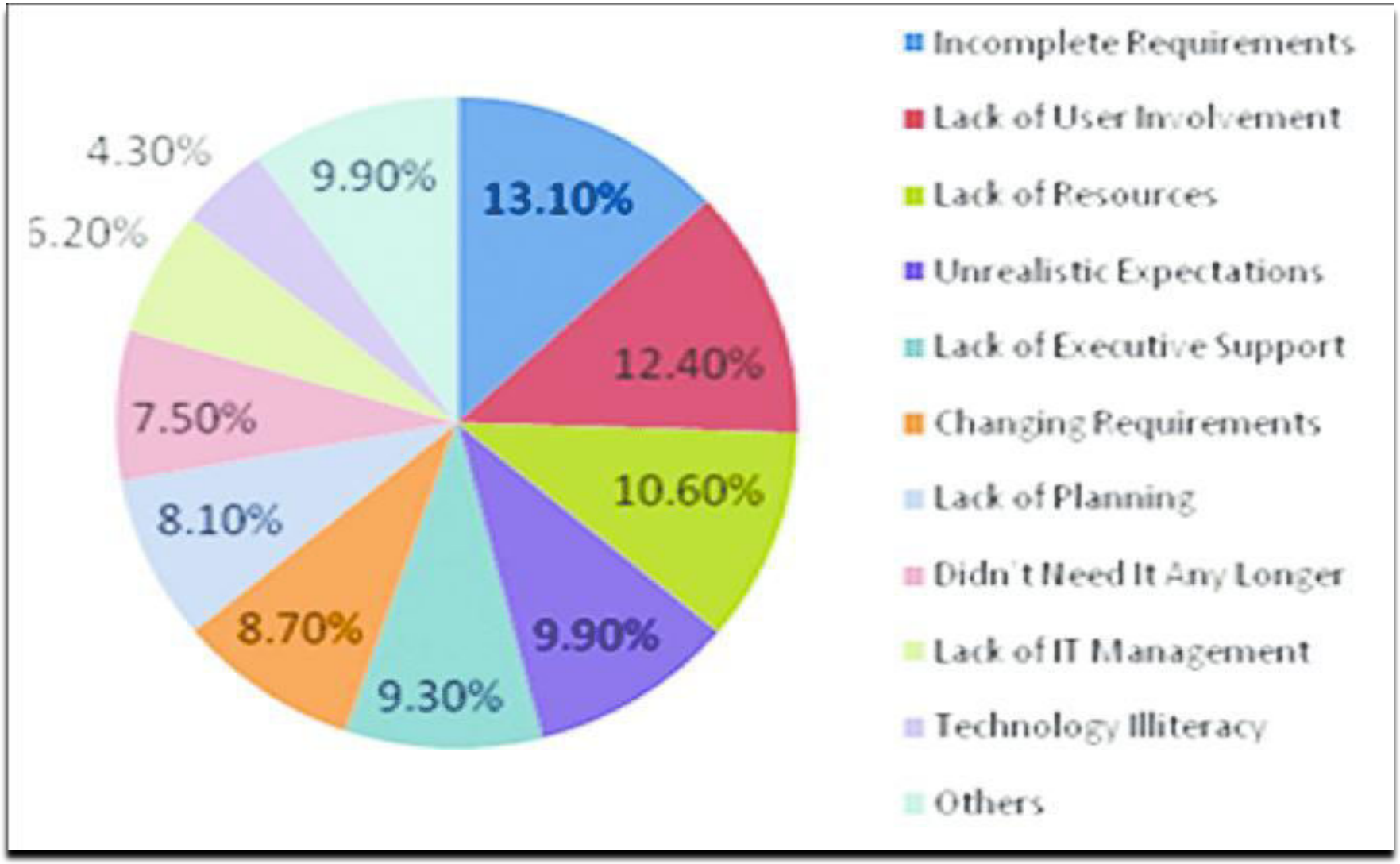
(1) A condition or capability needed by a user to solve a problem or achieve an objective.
(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

Type of Requirements –

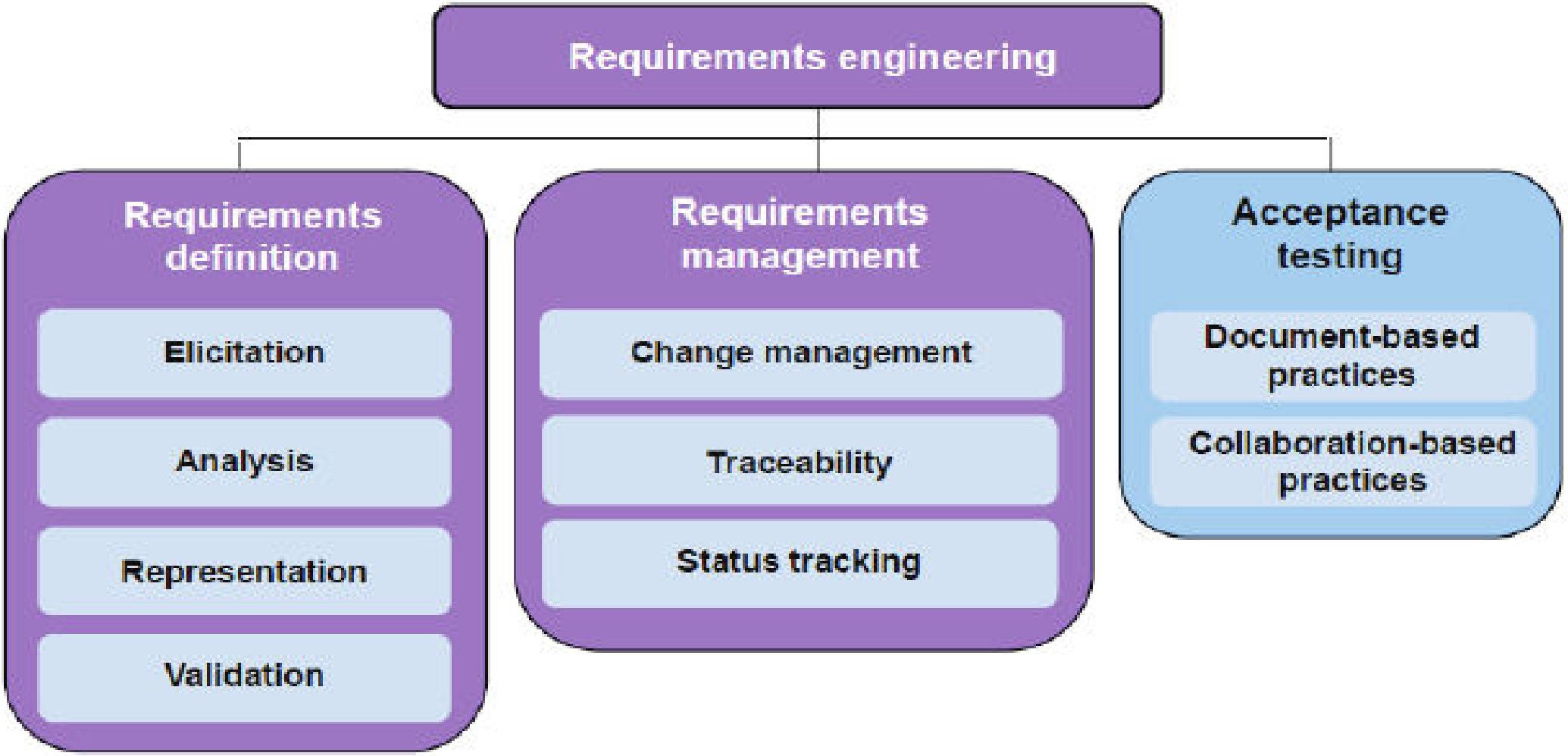
Business Requirements
Customer Requirements
Functional Requirements
Non-functional Requirements
Quality / Testing Requirements
Technology requirement Requirements
Implementation/Transition Requirements

Requirement Engineering

Why do Projects fail ? – Current Survey



Requirement Engineering



Use Case Elements

Explanation of Use Case Contents

This section will provide explanations for each element of the Use Case.

Name of Use Case: Provide a short name for the use case which should lend itself to the objective of the system.

Description: This section should provide a description of both the reason for using the use case and the expected outcome of the use case.

Actors: Actors may be primary or secondary. Primary actors are the people who will be initiating the system described in the use case. Secondary actors are those will participate in the completion of the use case.

Precondition: This section should describe any conditions that must be true or activities that must be completed prior to executing the use case.

Postcondition: This section should describe the state of the system at the conclusion of the use case. Postconditions may include conditions for both successful and unsuccessful execution of the use case.

Use Case Elements

Explanation of Use Case Contents

Flow: This section should describe all actions of the user and the expected system responses for planned normal execution of the use case. The description should be sequential and provide adequate detail to understand all user actions and system responses.

Alternative Flows: Many use cases have varying or special extensions or conditions which are separate from the main flow but also necessary. Alternative flows are usually the result of options or exceptions built into the use case which may alter the primary flow.

Exceptions: When use cases are executed, there may be various conditions which result in errors. This section should describe any errors that may result during use case execution and how the system will react or respond to those errors.

Requirements: This section should describe any non-functional or special requirements for the system as the use case is executed. These requirements may consist of legal or regulatory requirements, quality standards, or organizational requirements that are outside of the functional requirements the system is expected to perform.

Use Case Diagram -Notations

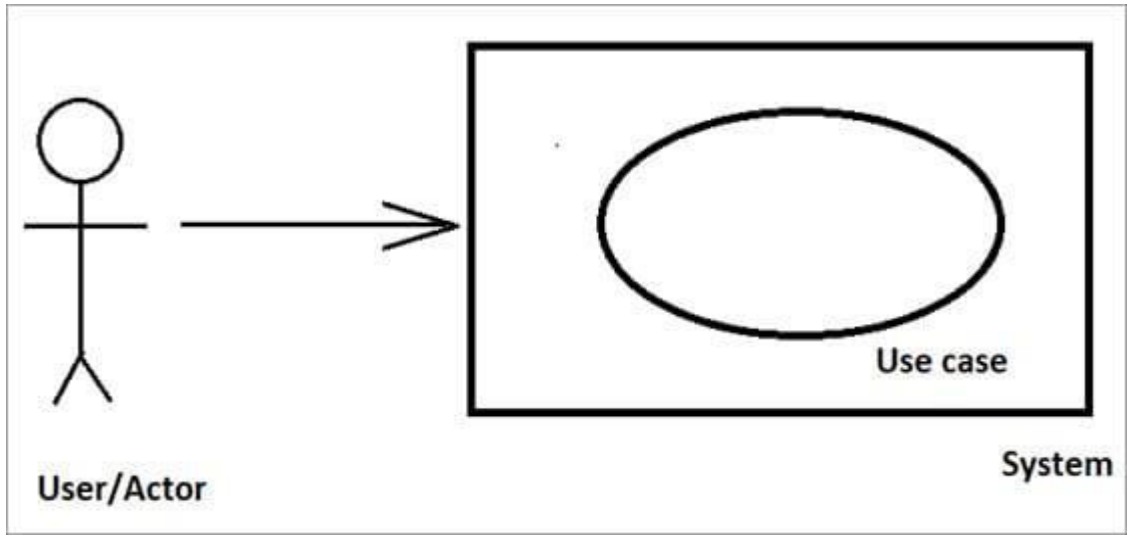
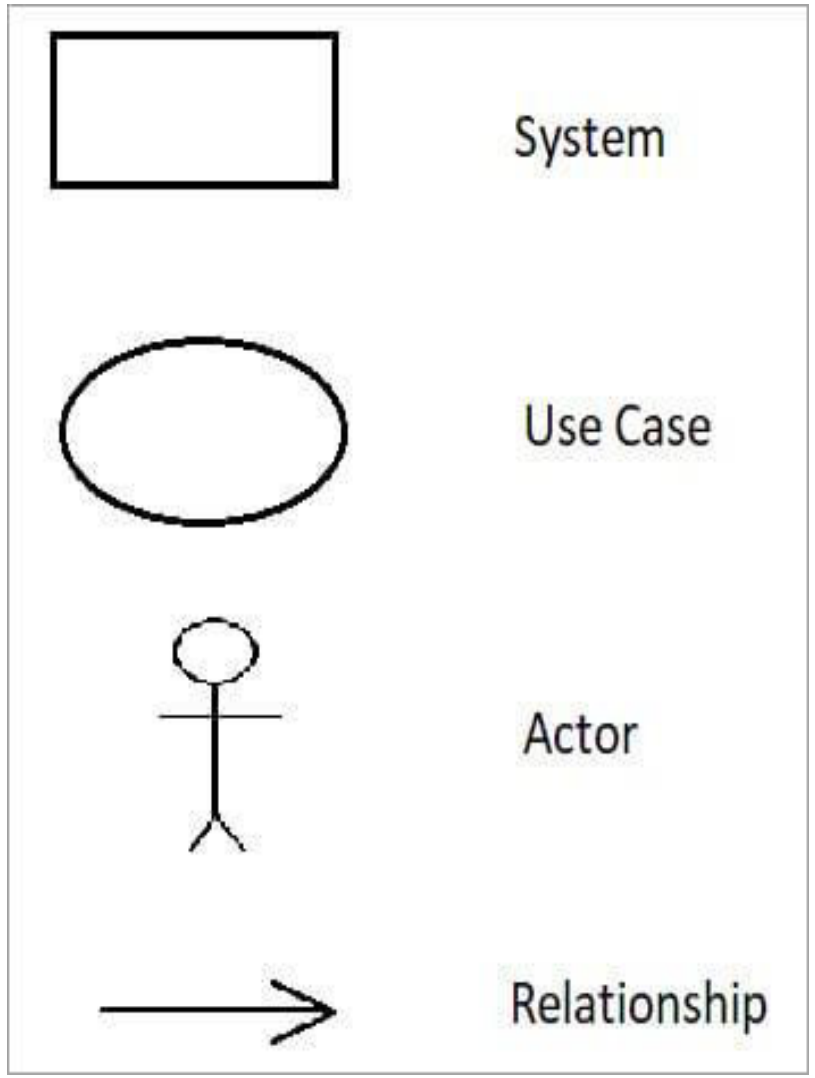


Fig No: UC 01

As shown in the Fig No: UC 01 it represents a diagram where Rectangle represents a ‘System’, oval represent a ‘Use Case’, Arrow represents a ‘Relationship’ and the Man represents a ‘User/Actor’. It Shows a system/application, then it shows the organization/people who interact with it and shows the basic flow of ‘What the system does?’

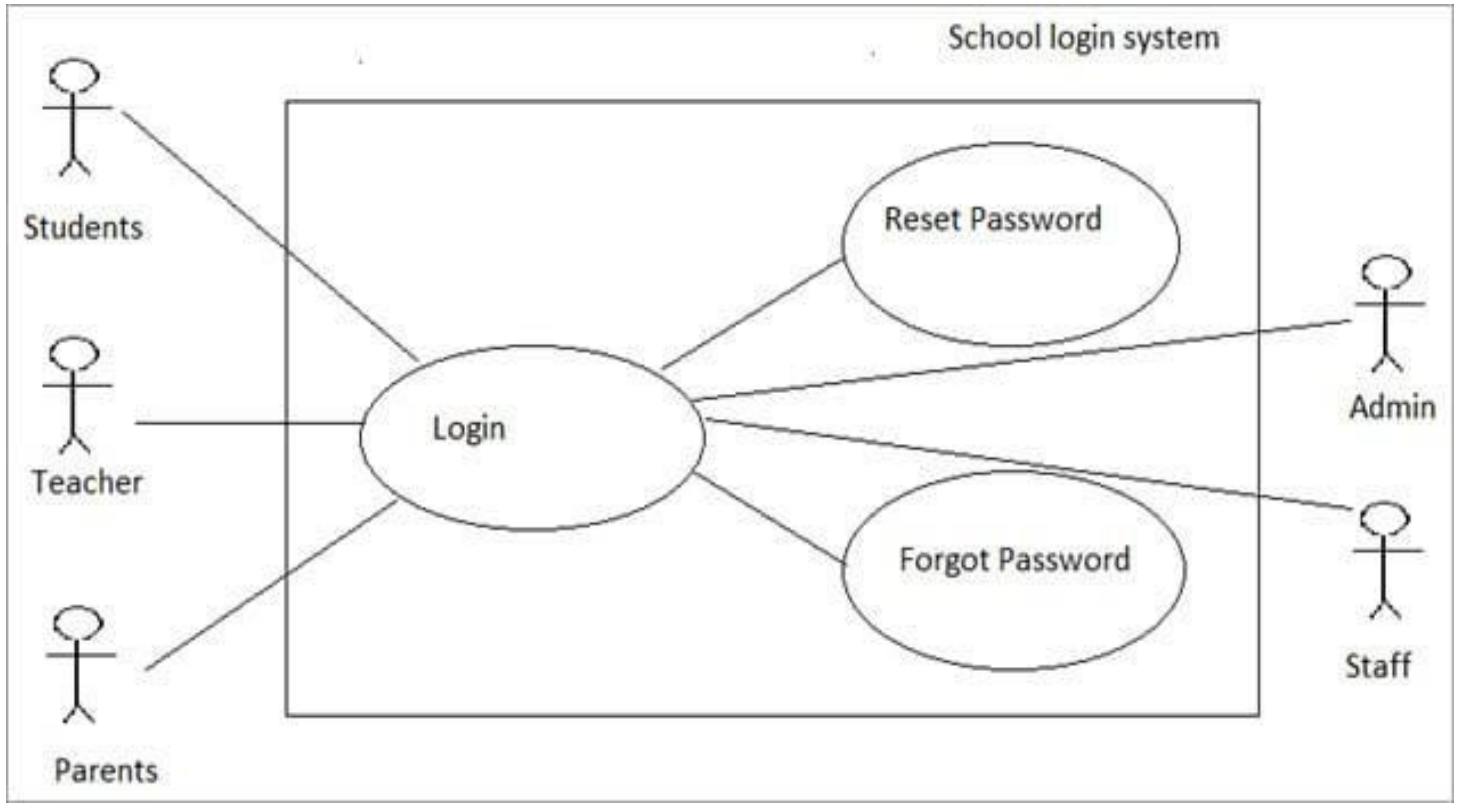


Use case diagram for login

This is the Use case diagram of ‘Login’ case. Here, we have more than one actor, they are all placed outside the system. Students, teachers, and parents are considered as primary actors. That is why they all are placed on the left side of the rectangle.

Admin and Staff are considered as secondary actors, so we place them on the right side of the rectangle. Actors can log in to the system, so we connect the actors and login case with a connector.

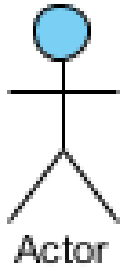
Other functionality found in the system are Reset Password and Forgot password. They are all related to login case, so we connect them to the connector.



Use Case Diagram -Notations

Actor

•Someone interacts with use case (system function).
•Named by noun.
•Actor plays a role in the business
•Similar to the concept of user, but a user can play different roles
•For example:
- A prof. can be instructor and also researcher
- plays 2 roles with two systems



•Actor triggers use case(s).
•Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).

Use Case

•System function (process - automated or manual)
•Named by verb + Noun (or Noun Phrase).
•i.e. Do something
•Each Actor must be linked to a use case, while some use cases may not be linked to actors.

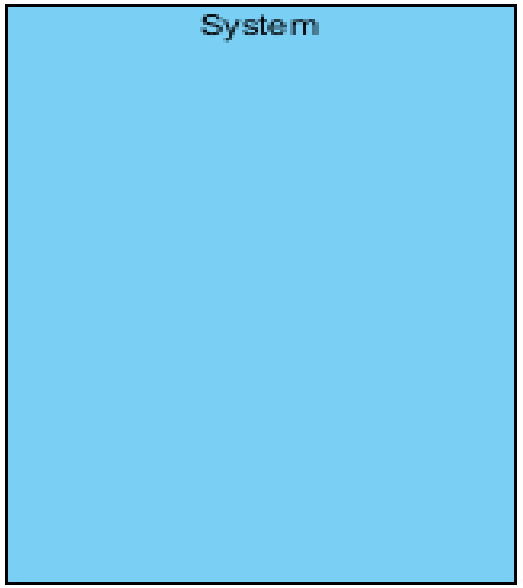
Use Case Diagram -Notations

Communication Link

The participation of an actor in a use case is shown by connecting an actor to a use case by a solid link.
Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.

Boundary of system

The system boundary is potentially the entire system as defined in the requirements document.
For large and complex systems, each module may be the system boundary.
For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting
can form a system boundary for use cases specific to each of these business functions.
The entire system can span all of these modules depicting the overall system boundary

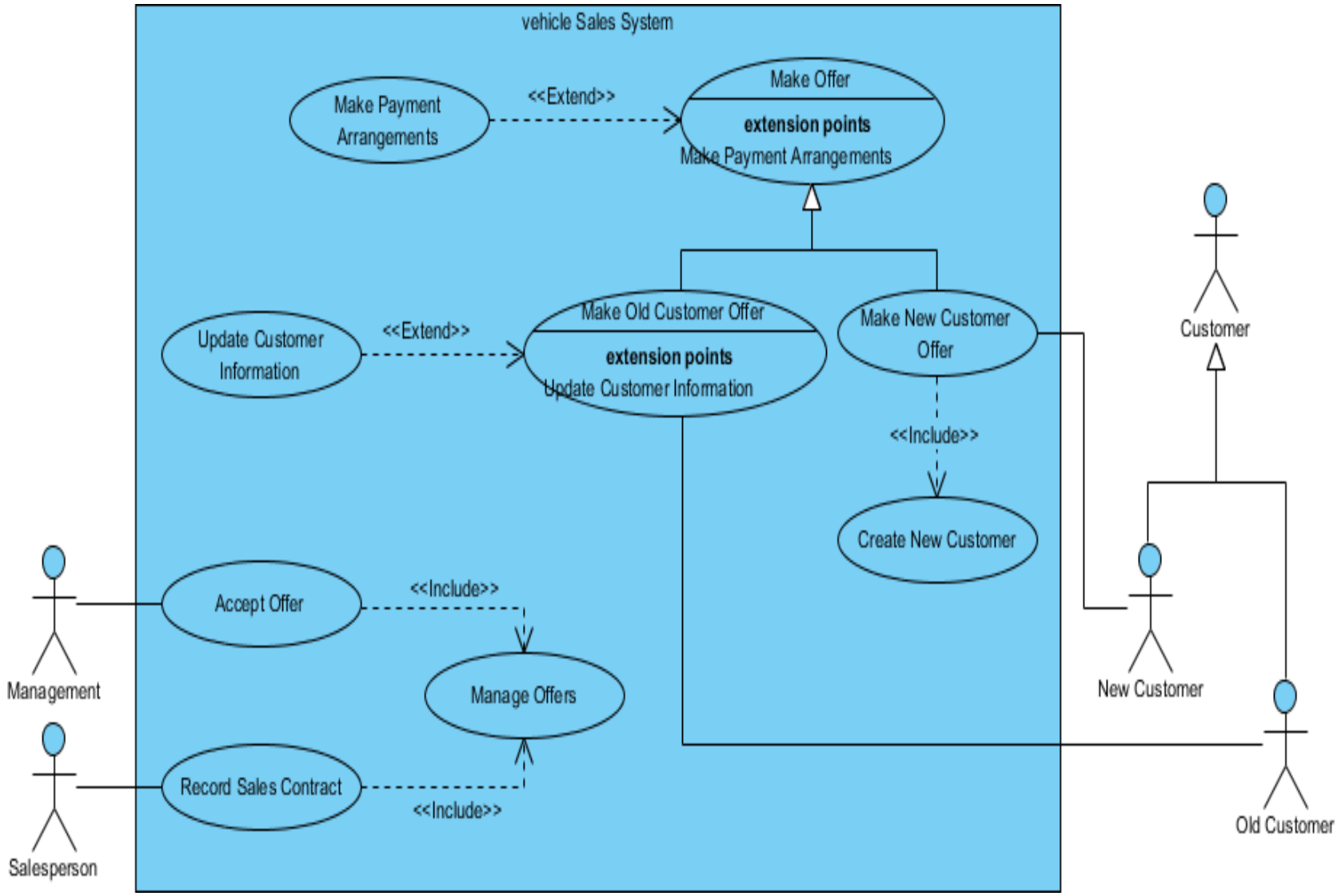


Use Case Example

Login' to a 'School Management System.

Use Case Name	Login	
Use case Description	A user login to System to access the functionality of the system.	
Actors	Parents, Students, Teacher, Admin	
Pre-Condition	System must be connected to the network.	
Post -Condition	After a successful login a notification mail is sent to the User mail id	
Main Scenarios	Serial No	Steps
	1	Enter username Enter Password
	2	Validate Username and Password
	3	Allow access to System
Extensions	1a	Invalid Username System shows an error message
	2b	Invalid Password System shows an error message
	3c	Invalid Password for 4 times Application closed

Use Case Model - Vehicle Sales Systems



Note that:

While a use case itself might drill into a lot of detail about every possibility, a use-case diagram is often used for a higher-level view of the system as blueprints.

It is beneficial to write use cases at a coarser level of granularity with less detail when it's not required.

Requirement Engineering

Software Requirement Specification (SRS) / Requirement Specification

A **software requirement specification** (**SRS**) is a comprehensive information/description of a product/system to be developed with its functional and non-functional requirements.

The software **requirement specification** (**SRS**) is developed based on the agreement between customer and supplier. It may include the use cases of how a user is going to interact with the product or software system. SRS helps to reduce the time and effort to develop software. To develop the software system we should clearly understand the Software requirements. To achieve this we need to continuously communicate with clients to gather all related information and requirements.

Characteristics of good SRS Document

- **Concise**: The SRS document should be unambiguous, consistent, and complete.
- **Structured**: A well-structured SRS document is easy to understand and modify.
- **Black-Box View**: SRS should only specify what the system should do and restrict from stating how to do.
- **Conceptual Integrity**: Conceptual integrity in the SRS helps the reader to easily understand it.
- **Traceable**: Traceability means it can tell which code part corresponds to which design component, which test case corresponds to which requirement and which design component corresponds to which requirement etc.
- **Verifiable**: All requirements of the system should be verifiable. This means it should be possible to tell whether or not the requirement has been met as specified in the SRS.

Software Design & Architectural Engineering

Software Design and its Activities

The purpose of Design phase in the Software Development Life Cycle is to produce a solution to a problem (transforming the client requirements) given in the SRS(Software Requirement Specification) document. The output of the design phase is Software Design Document (SDD).

3 Software Design Levels

1) Architectural Design
2) Preliminary (High-Level) Design
3) Detailed Design

Difference between Analysis and Design

The aim of analysis is to understand the problem with a view to eliminate any deficiencies in the requirement specification such as incompleteness, inconsistencies, etc.

The aim of design is to produce a model that will provide a seamless transition to the coding phase, i.e. once the requirements are analyzed and found to be satisfactory, a design model is created which can be easily implemented.

Software Design & Architectural Engineering

Software Design and its Activities

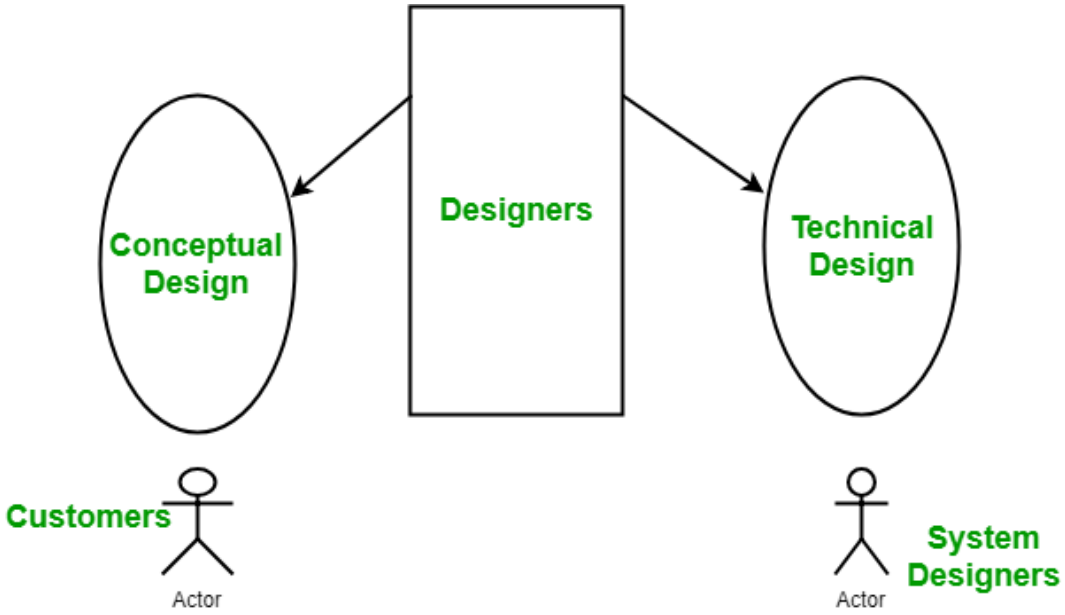
The purpose of Design phase in the Software Development Life Cycle is to produce a solution to a problem (transforming the client requirements) given in the SRS(Software Requirement Specification) document. The output of the design phase is Software Design Document (SDD).

Conceptual Design of the system:

- Written in simple language i.e., customer understandable language.
- Detailed explanation about system characteristics.
- Describes the functionality of the system.
- It is independent of implementation.
- Linked with requirement document.

Technical Design of the system:

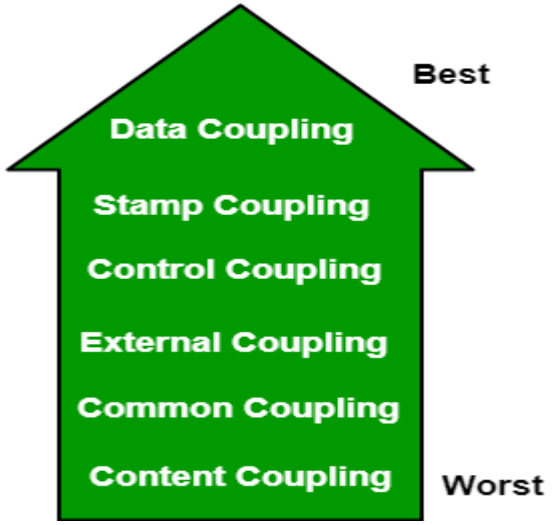
- Hardware component and design.
- Functionality and hierarchy of software components.
- Software architecture
- Network architecture
- Data structure and flow of data.
- I/O component of the system.
- Shows interface.



Software Design & Architectural Engineering

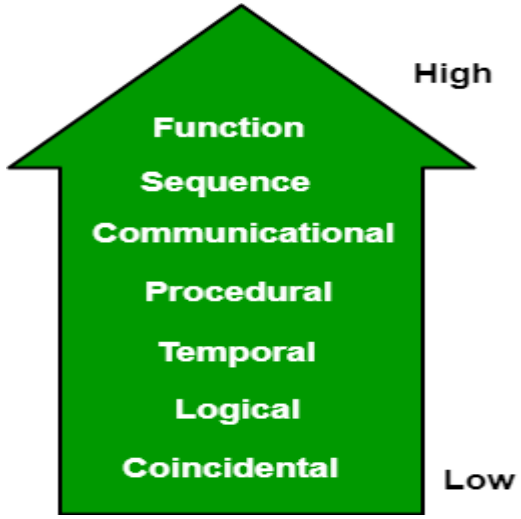
Modularization: Modularization is the process of dividing a software system into multiple independent modules where each module works independently. There are many advantages of Modularization in software engineering. Some of these are given below:

- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.



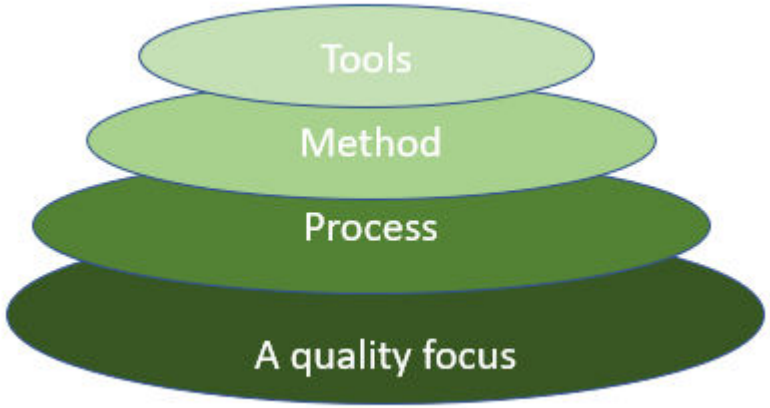
Coupling: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.

Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



Software Design & Architectural Engineering

Layering: Software engineering is a fully layered technology, to develop software we need to go from one layer to another. All the layers are connected, and each layer demands the fulfillment of the previous layer



Characteristics of good design

The definition of “a good software design” can vary depending on the application being designed. Most researchers and software engineers agree on a few desirable characteristics that every good software design for a general application must possess. The characteristics are listed below:

Correctness: A good design should correctly implement all the functionalities identified in the SRS document.

Understandability: A good design is easily understandable.

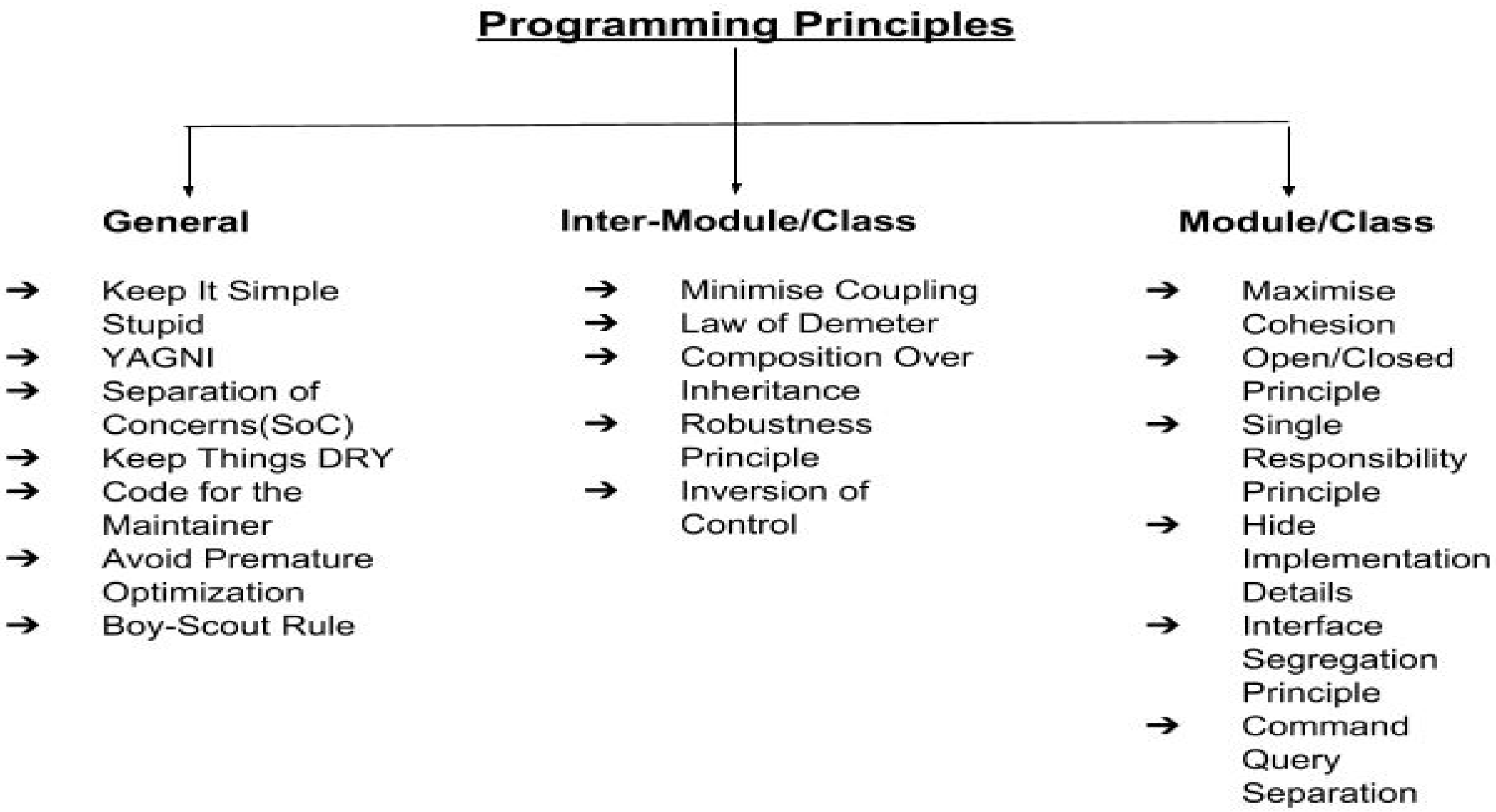
Efficiency: It should be efficient.

Maintainability: It should be easily amenable to change.

Software Design & Architectural Engineering

COMPARISON FACTORS	FUNCTION ORIENTED DESIGN	OBJECT ORIENTED DESIGN
Abstraction	The basic abstractions, which are given to the user, are real world functions.	The basic abstractions are not the real-world functions but are the data abstraction where the real-world entities are represented.
Function	Functions are grouped together by which a higher-level function is obtained.	Function are grouped together on the basis of the data they operate since the classes are associated with their methods.
execute	carried out using structured analysis and structured design i.e, data flow diagram	Carried out using UML
State information	In this approach the state information is often represented in a centralized shared memory.	In this approach the state information is not represented is not represented in a centralized memory but is implemented or distributed among the objects of the system.
Approach	It is a top-down approach.	It is a bottom-up approach.
Begins basis	Begins by considering the use case diagrams and the scenarios.	Begins by identifying objects and classes.
Decompose	In function-oriented design we decompose in function/procedure level.	We decompose in class level.
Use	This approach is mainly used for computation sensitive application.	This approach is mainly used for evolving system which mimics a business or business case.

Programming Principles



Programming Principles

- **Pseudocode**
  - Pseudocode is **an informal way of programming description that** does not require any strict programming language syntax or underlying technology considerations.
  - It is used for creating an outline or a rough draft of a program. Pseudocode summarizes a program's flow, but excludes underlying details

  - E.g.

    Get 10,000 records from DB

    Filter records based on Category

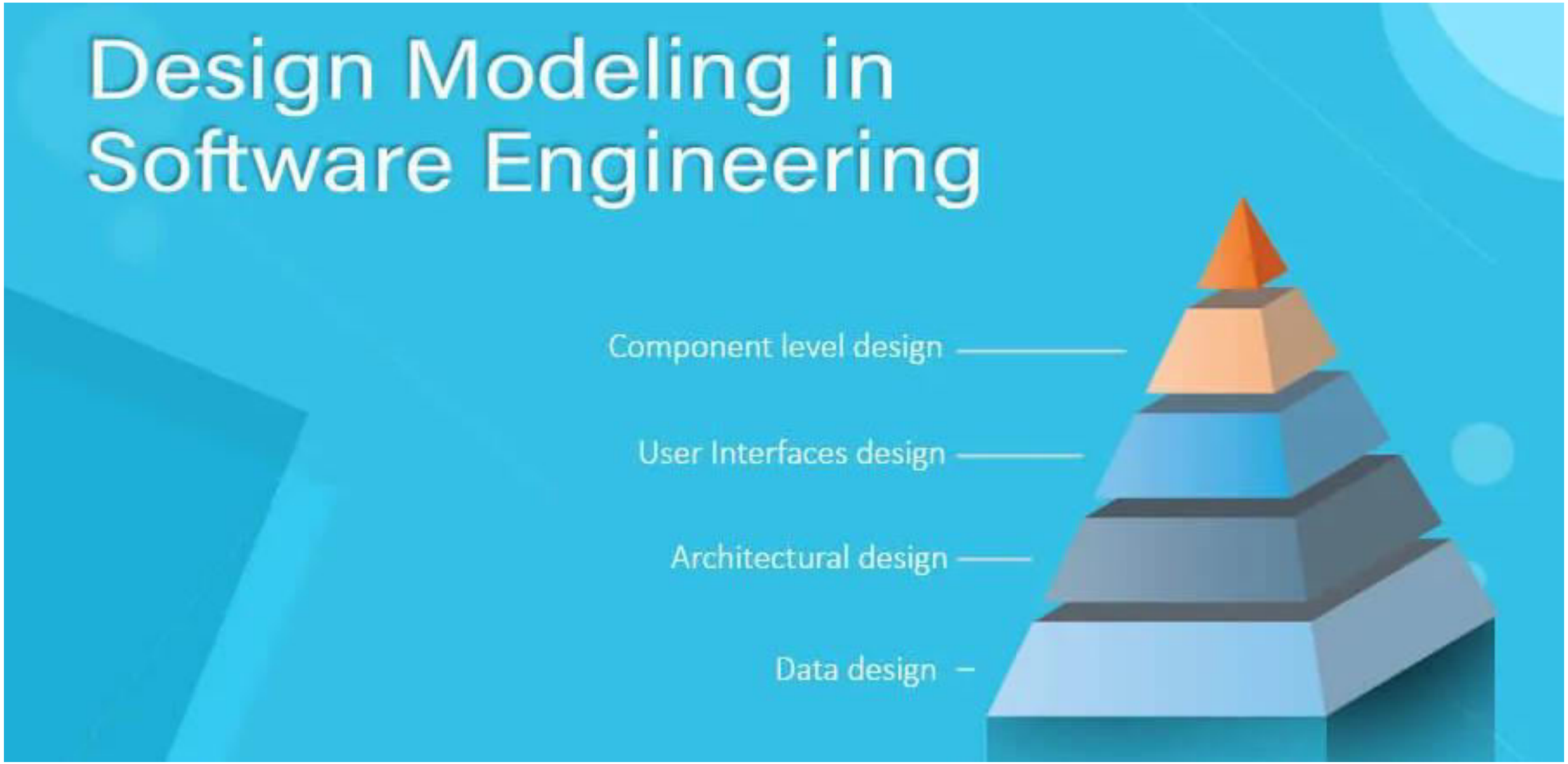
    For loop: Process each record for business algorithm

    Business requirement: Check if each record is created during certain period.

    If Yes, mark the record as invalid.

    End For Loop

UML - Unified Modeling Language (UML)



UML - Unified Modeling Language (UML)

This is not a programming language, it is rather a visual language. We use UML diagrams to portray the **behavior and structure** of a system

1.Structural Diagrams – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
Describe system's architecture, entities and their properties.
2.Behavior Diagrams – Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.
Describe how the system responds and behaves in different scenarios.

UML

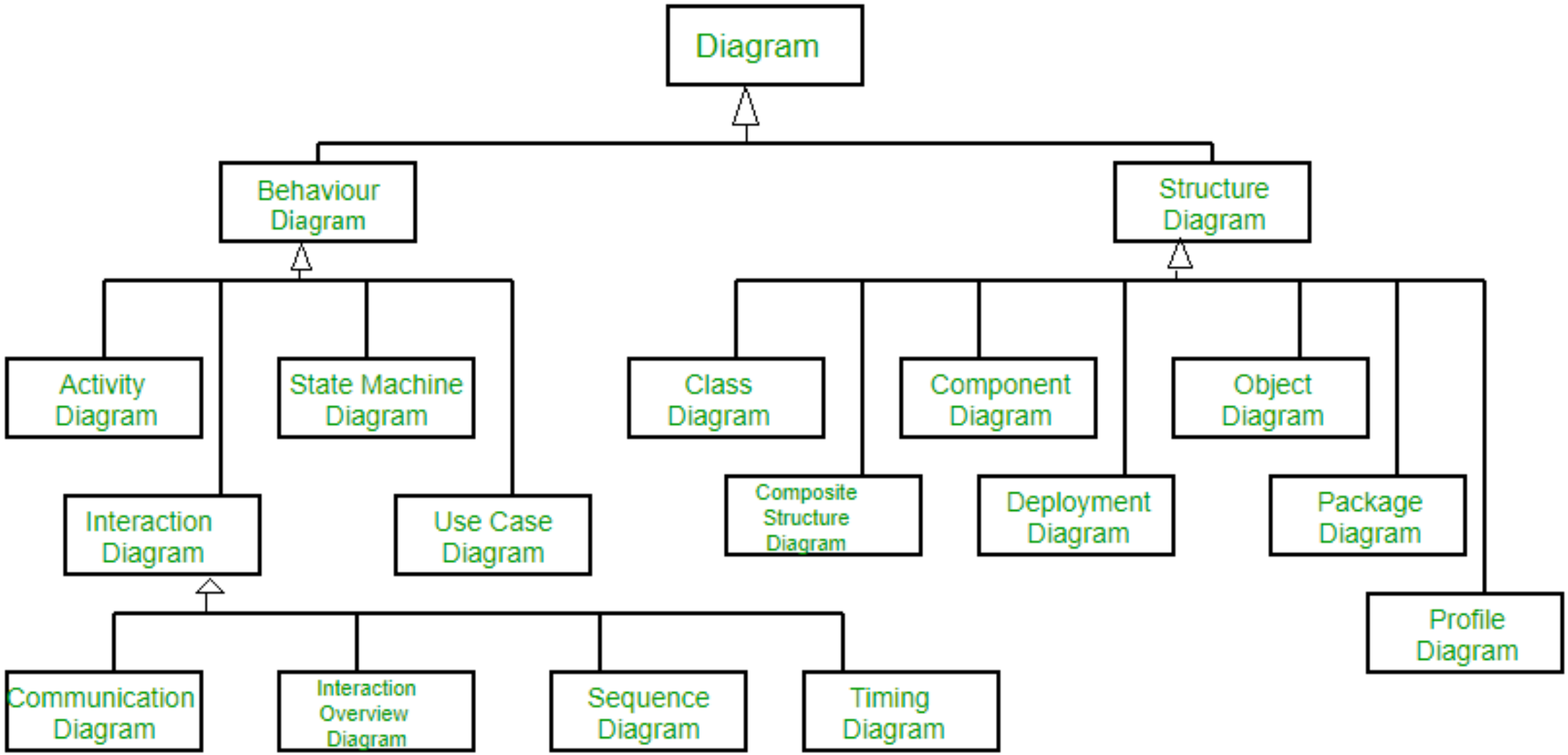


Diagram
Behaviour Diagram
Structure Diagram
Activity Diagram
State Machine Diagram
Interaction Diagram
Use Case Diagram
Class Diagram
Component Diagram
Object Diagram
Composite Structure Diagram
Deployment Diagram
Package Diagram
Profile Diagram
Communication Diagram
Interaction Overview Diagram
Sequence Diagram
Timing Diagram

CMM-Capability Maturity Model

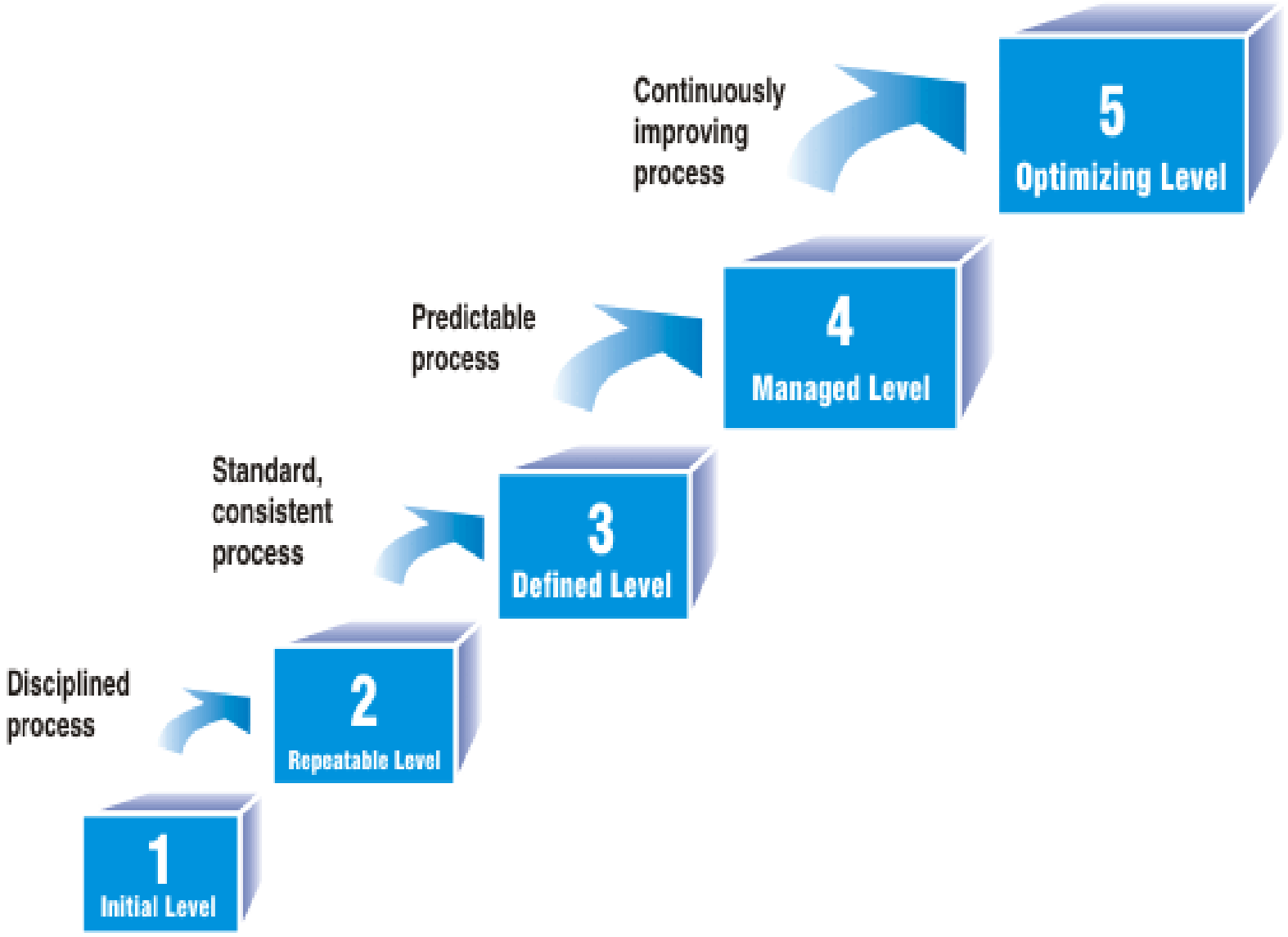
CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.

It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.

It also provides guidelines to further enhance the maturity of the process used to develop those software products.

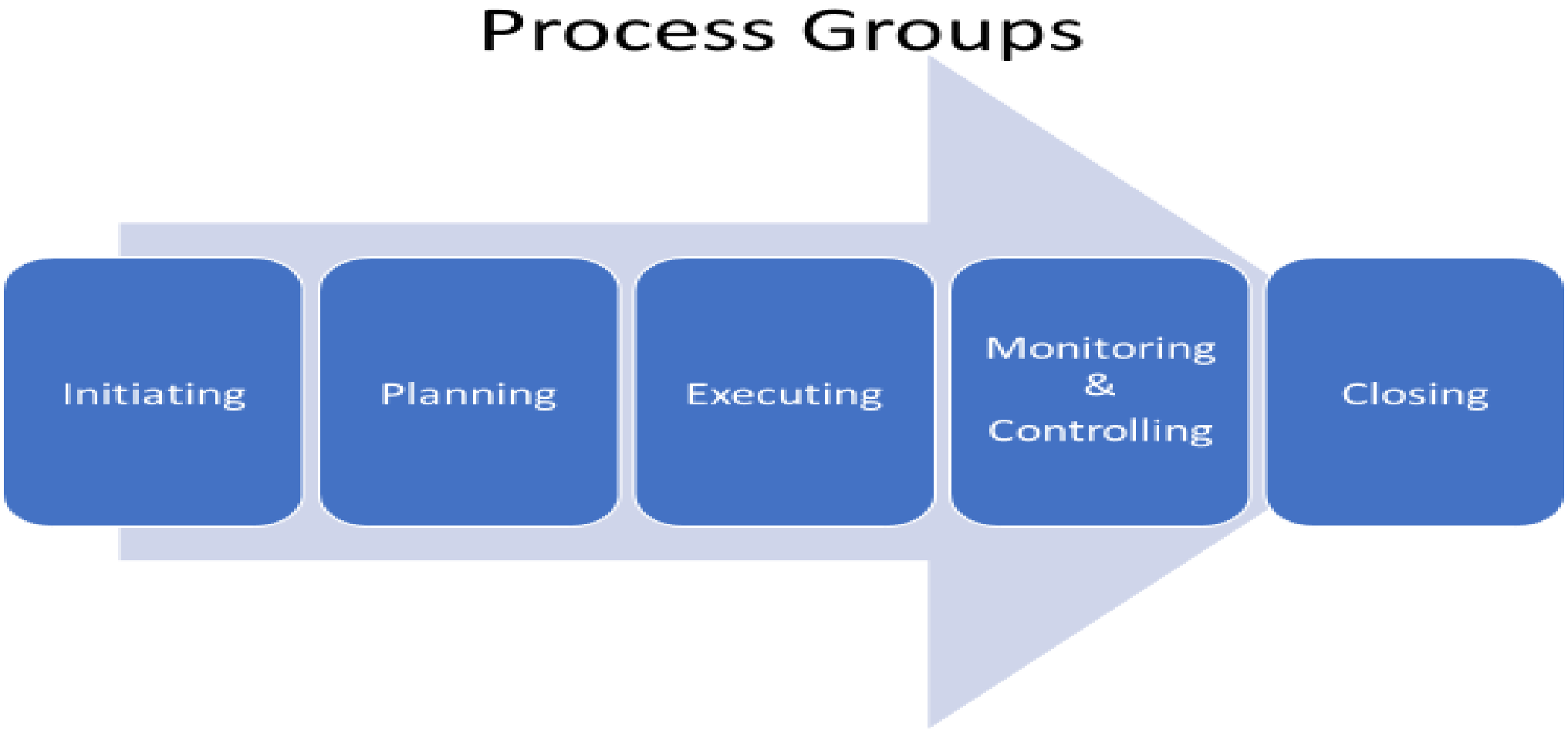
It is based on profound feedback and development practices adopted by the most successful organizations worldwide.

This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.



Software Project Management(SPM)

Project Management is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements.



Software Estimation Measures

Estimation of the size of the software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:

- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

Software Configuration Management - What

Software Configuration Management(SCM)

SCM is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.

The primary goal is to increase productivity with minimal mistakes.

SCM is part of cross-disciplinary field of configuration management, and it can accurately determine who made which revision.

Configuration management is considered a subset of systems management, a process for keeping servers, systems, and software functioning consistently within a set of established parameters.

The process ensures the system, and its resources perform as expected, despite updates, additions, and deletions. So, configuration management ensures that all the devices in your network infrastructure march to the same beat, keeping everyone in line.

Software Configuration Management - Features

Enforcement:
With enforcement feature execution daily, ensures that the system is configured to the desired state.

Cooperating Enablement:
This feature helps to make the change configuration throughout the infrastructure with one change.

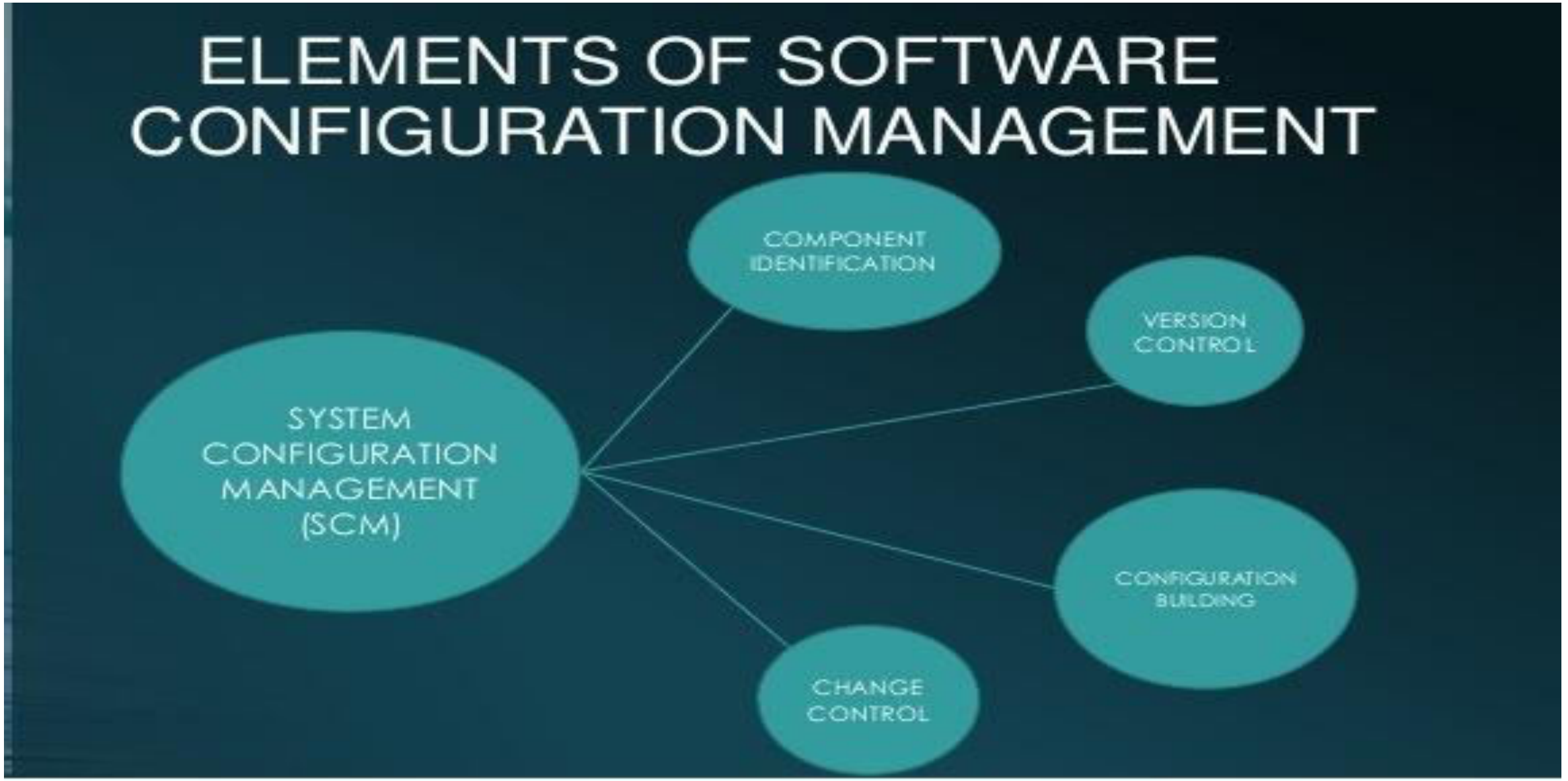
Version Control Friendly:
With this feature, the user can take their choice of version for their work.

Enable Change Control Processes:
As Software Configuration Management tools are version control and textual friendly, we can make changes in code. Changes can be made as a merge request and send for review.

Tasks in SCM process

1. Planning and Identification Process
2. Version Control Process or Baselines
3. Change Control Process
4.Configuration Release Process
5. Configuration Auditing Process
6. Review and Status Reporting Process

Software Configuration Management - Elements



SDLC – Agile Methodologies

Agile - “quick to move or drive”

Agile Software development methodology.

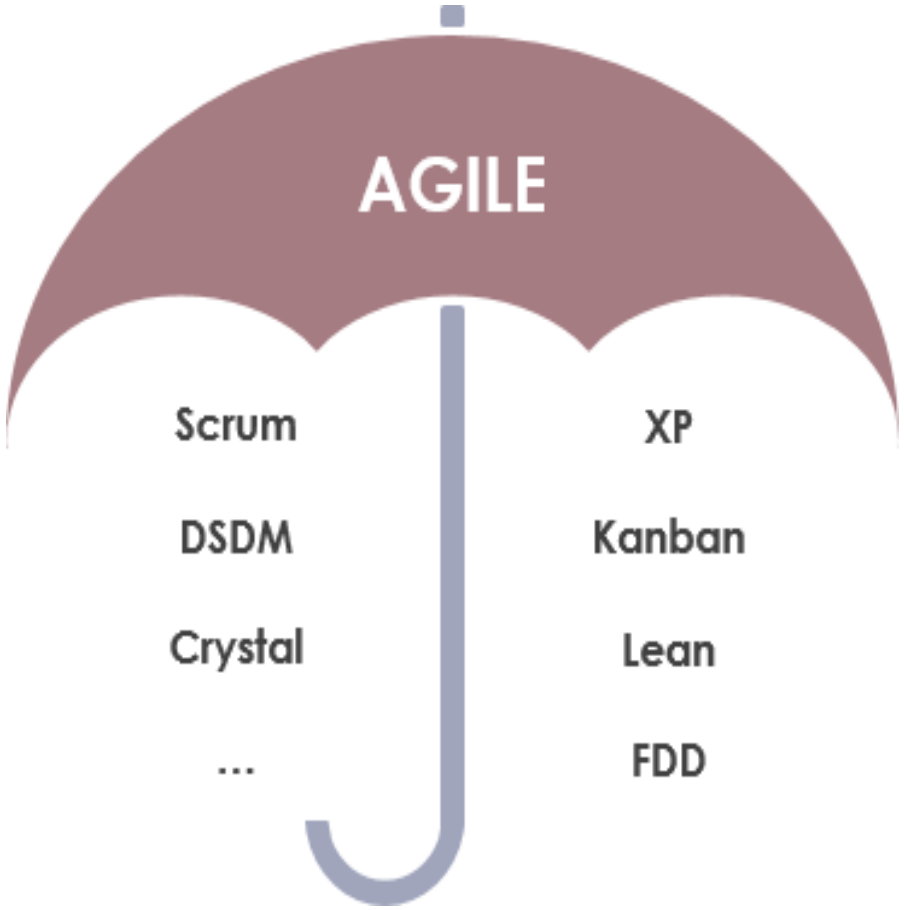
1. Adaptive Software Development (ASD)
2. Crystal Methods
3. Dynamic Systems Development Method (DSDM)
4. Extreme Programming (XP)
5. Feature-Driven Development (FDD)
6. Lean Software Development (LSD)
7. SCRUM
8. Kanban

Agile – The Manifesto

The agile software development emphasizes on four core values.

Individual and team interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

Agile Methods and Practices

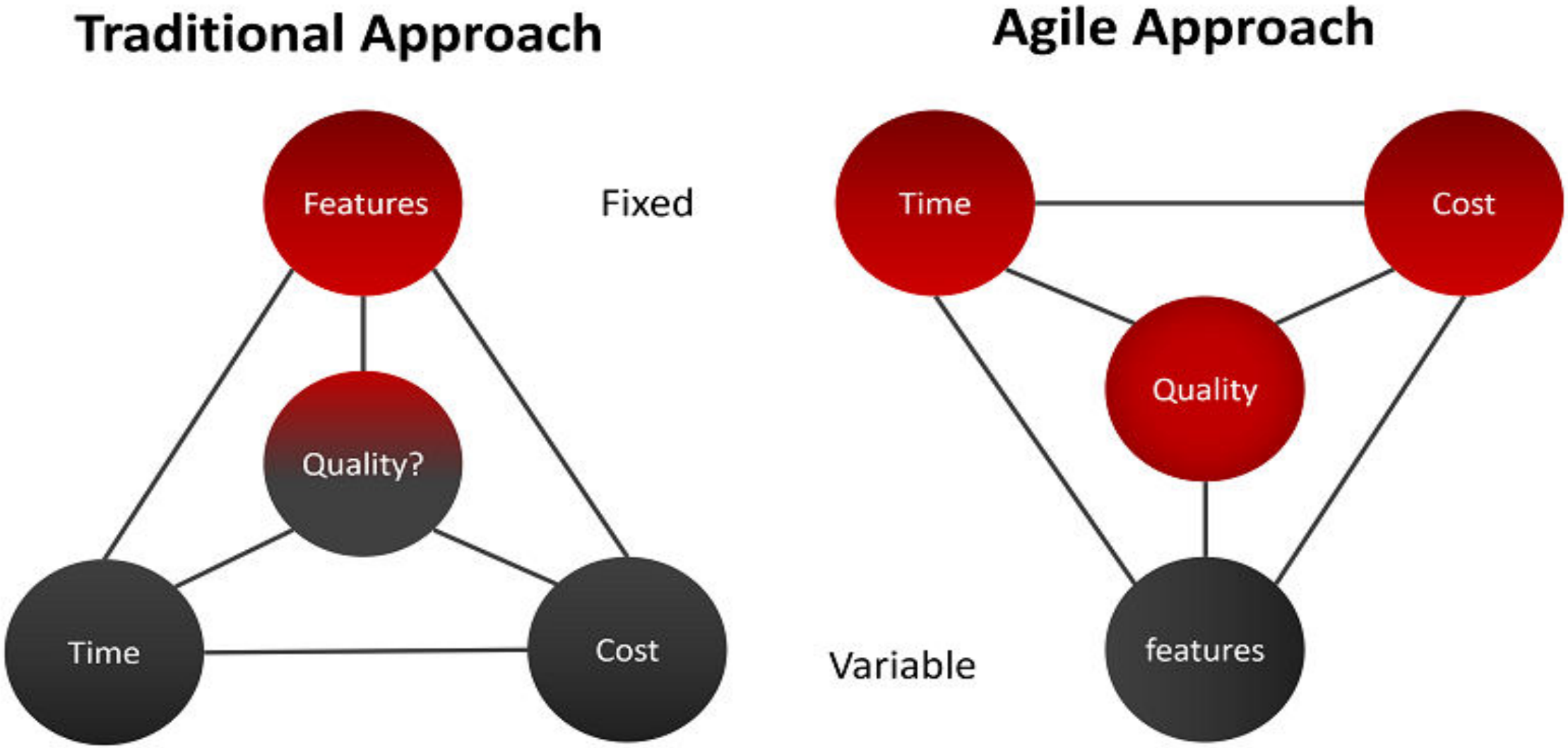


- Scrum
- Crystal Methodologies
- DSDM (Dynamic Software Development Method)
- Feature driven development (FDD)
- Lean software development
- Extreme Programming (XP)

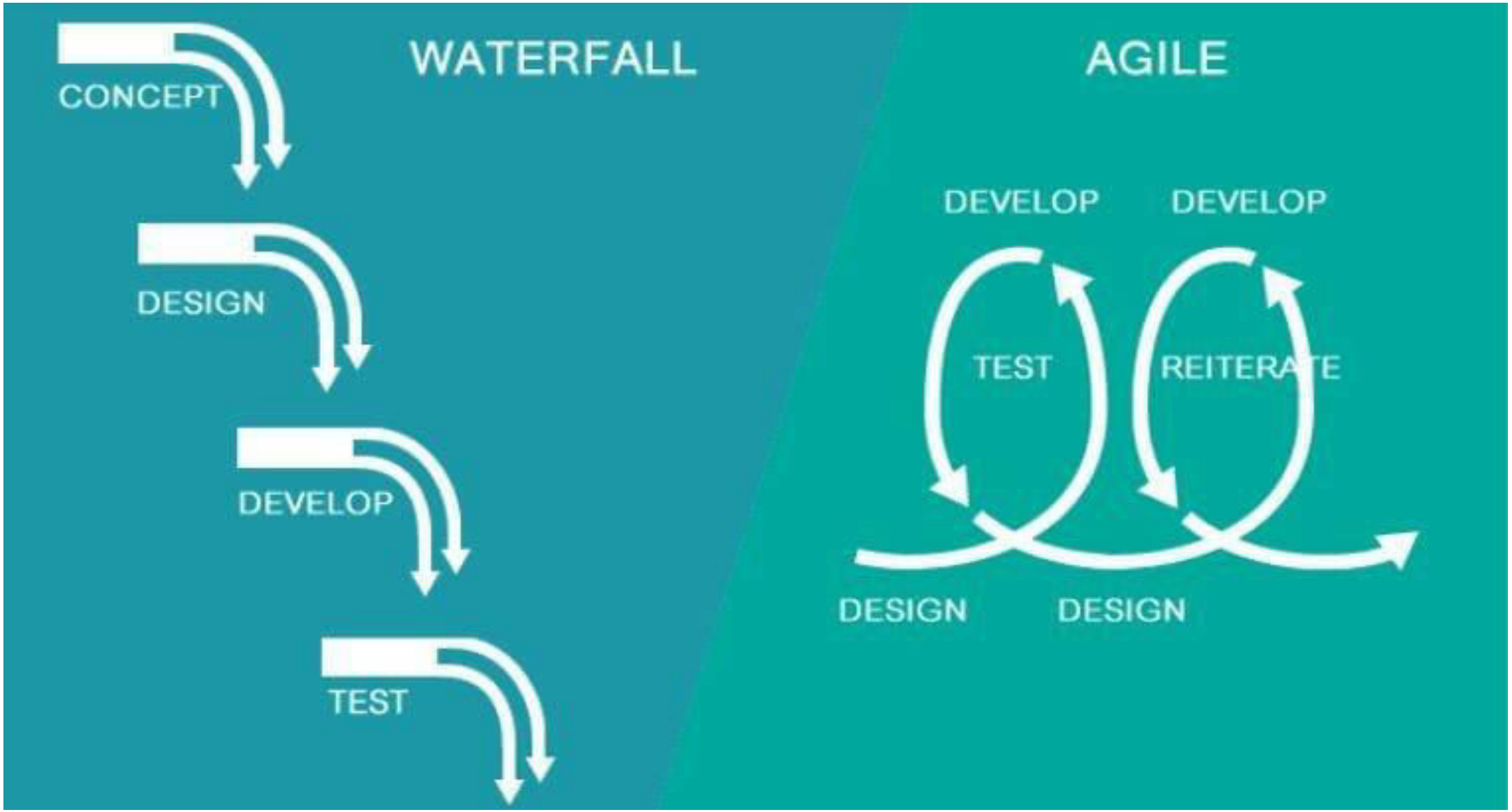
Agile – The Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Businesspeople and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile – Approach



Traditional vs Agile Approach



Agile Methods and Practices

Extreme Programming (XP) –
Extreme Programming (XP) also emphasizes speed and continuous delivery. Like Scrum, XP enables closely-knit teams to deliver working software increments at frequent intervals, usually every 1-3 weeks. It relies on customers to communicate the most useful features of a software product and developers to work towards implementing that feedback.

The five basic component of Extreme Programming are:
1.Communication
2.Simplicity
3.Feedback
4.Respect
5.Courage

Extreme Programming allow changes in their set timelines.
In Extreme Programming(XP), teamwork for 1-2 weeks only.
Extreme Programming emphasizes strong engineering practices
In Extreme Programming, team must follow a strict priority order or pre-determined priority order
Extreme Programming(XP) can be directly applied to a team. Extreme Programming is also known for its **Ready-to-apply** features.

Agile Methods and Practices

Feature-Driven Development (FDD) –
Feature-Driven Development, or FDD, provides a framework for product development that starts with an overall model and gets progressively more granular. Like other Agile methodologies, FDD aims to deliver working software quickly in a repeatable way. It uses the concept of “just enough design initially” (JEDI) to do so, leveraging two-week increments to run “plan by feature, design by feature, build by feature” iterations.
Organizations that practice Agile like Feature-Driven Development for its feature-centric approach and its scalability.

Five Step Process -

- Developing an overall model
- Building the feature list
- Planning by feature
- Designing by feature
- Building by feature

FDD is amazing for big projects and is quite scalable and prone to get achieve success.
FDD is very effective in helping with complex projects that are in a critical situation.

Agile Methods and Practices

Lean Software Development –

Lean software development is more flexible than Scrum or XP, with fewer strict guidelines, rules, or methods. Lean is based on a set of principles developed to ensure value and efficiency in production in the mid 20th century and has evolved into the software setting. Lean relies on five principles of Lean management:

1.Identify value
2.Value stream mapping
3.Create continuous workflow
4.Create a pull system
5.Continuous improvement

Lean particularly emphasizes eliminating waste. In the context of software development, that includes cutting out wasted time and unproductive tasks, efficiently using team resources, giving teams and individuals decision-making authority, and prioritizing only the features of a system that deliver true value.

Agile Methods and Practices

Kanban –
Kanban focuses on helping teams work together more effectively to enable continuous delivery of quality products. Kanban is unique, however, for offering a highly visual method for actively managing the creation of products.

The Kanban Agile methodology relies on six fundamental practices:
1.Visualize the workflow
2.Limit work in progress
3.Manage flow
4.Make process policies explicit
5.Implement feedback loops
6.Improve collaboratively

Kanban achieves these practices using a Kanban board. The Kanban board facilitates the visual approach to Agile using columns to represent work that is To Do, Doing, and Done. This Agile methodology improves collaboration and efficiency and helps define the best possible team workflow.
To Do, In Progress, In Review, and Done (or Completed).
The most important metrics are lead time and cycle time. Each of these metrics measures the average amount of time it takes for tasks to move through the board.

Agile – Scrum Vs Kanban

	Scrum	Kanban
Cadence	Regular fixed length sprints (i.e., 2 weeks)	Continuous flow
Release methodology	At the end of each sprint	Continuous delivery
Roles	Product owner, scrum master, development team	No required roles
Key metrics	Velocity	Lead time, cycle time, WIP
Change philosophy	Teams should not make changes during the sprint.	Change can happen at any time
Approach	A structured agile approach	Continuous improvement, flexible processes

Agile Methods and Practices

Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) rounds out our list of well-known Agile methodologies. DSDM originated in the 1990s to provide a common industry framework for rapid software delivery. Today, it has matured into a comprehensive

Agile methodology that revolves around:
- Business needs and value
- Active user involvement
- Empowered teams
- Frequent delivery
- Integrated testing
- Stakeholder collaboration

The DSDM framework is particularly useful for prioritizing requirements. It also mandates that rework is to be expected, so any development changes must be reversible. DSDM relies on sprints, similar to other Agile methodologies, and is often used in conjunction with approaches like Scrum and XP.

Agile Methods and Practices

Crystal -
The Crystal Agile methodology focuses more on the interactions of the people involved in a project versus the tools and techniques of development. A lightweight model, Crystal emphasizes interaction, people, community, skills, communications, and talents.

Crystal categorizes projects based on three criteria:
1.Team size
2.System criticality
3.Project priorities

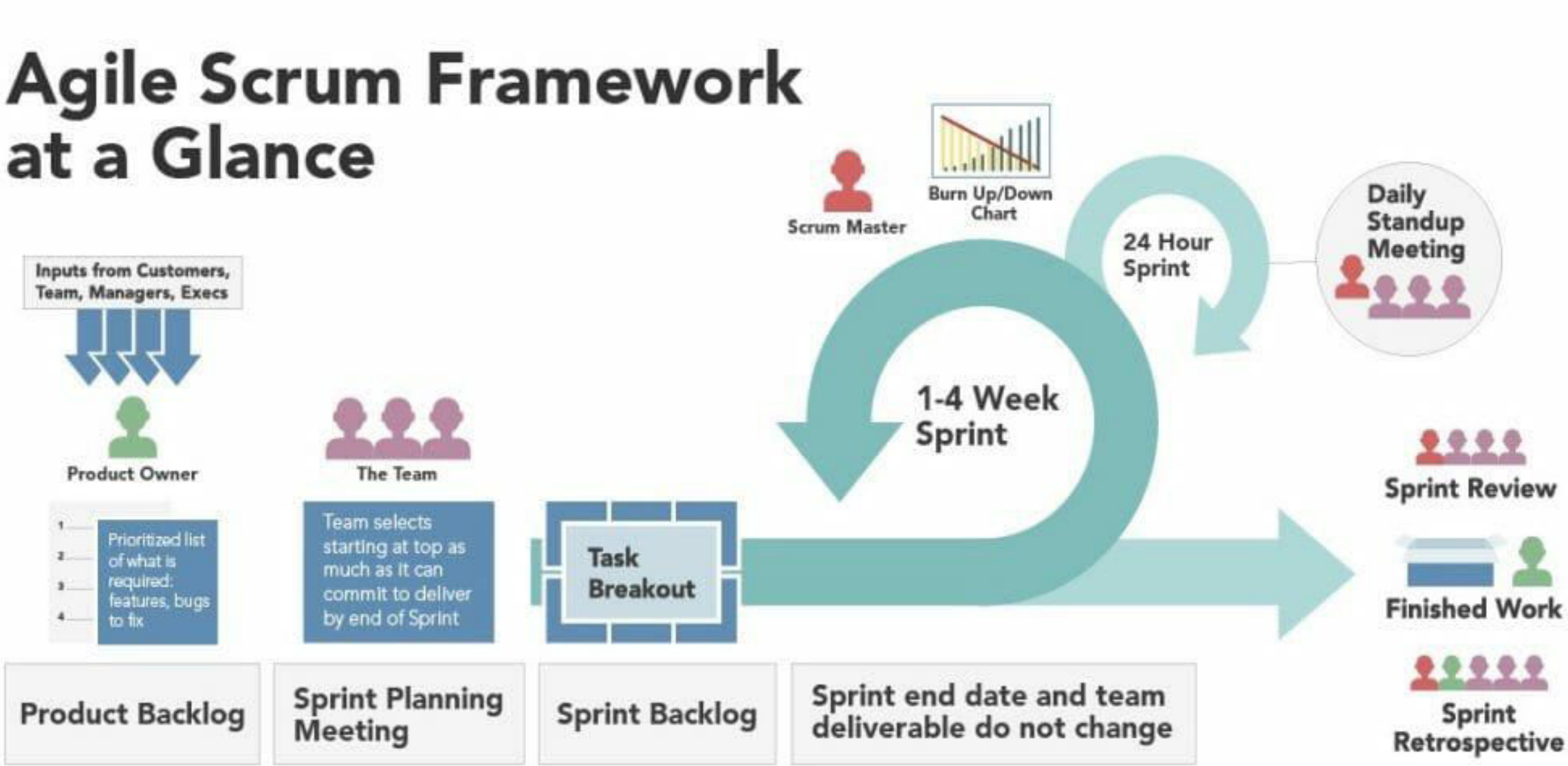
The approach is similar to other Agile methodologies in its attention to early and often delivery of software, high involvement of users, and removal of red tape. Crystal's assertion that every project is unique, however, has led to its reputation as one of the most flexible Agile methodologies.

Agile Methods and Practices

Scrum

SCRUM is an agile development method which concentrates specifically on how to manage tasks within a team-based development environment. Basically, Scrum is derived from activity that occurs during a rugby match. Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members). Agile and Scrum consist of three roles, and their responsibilities are explained as follows:

Agile – Scrum Framework



Agile Concepts –
I know Agile concepts.

All set for Applied Agile.

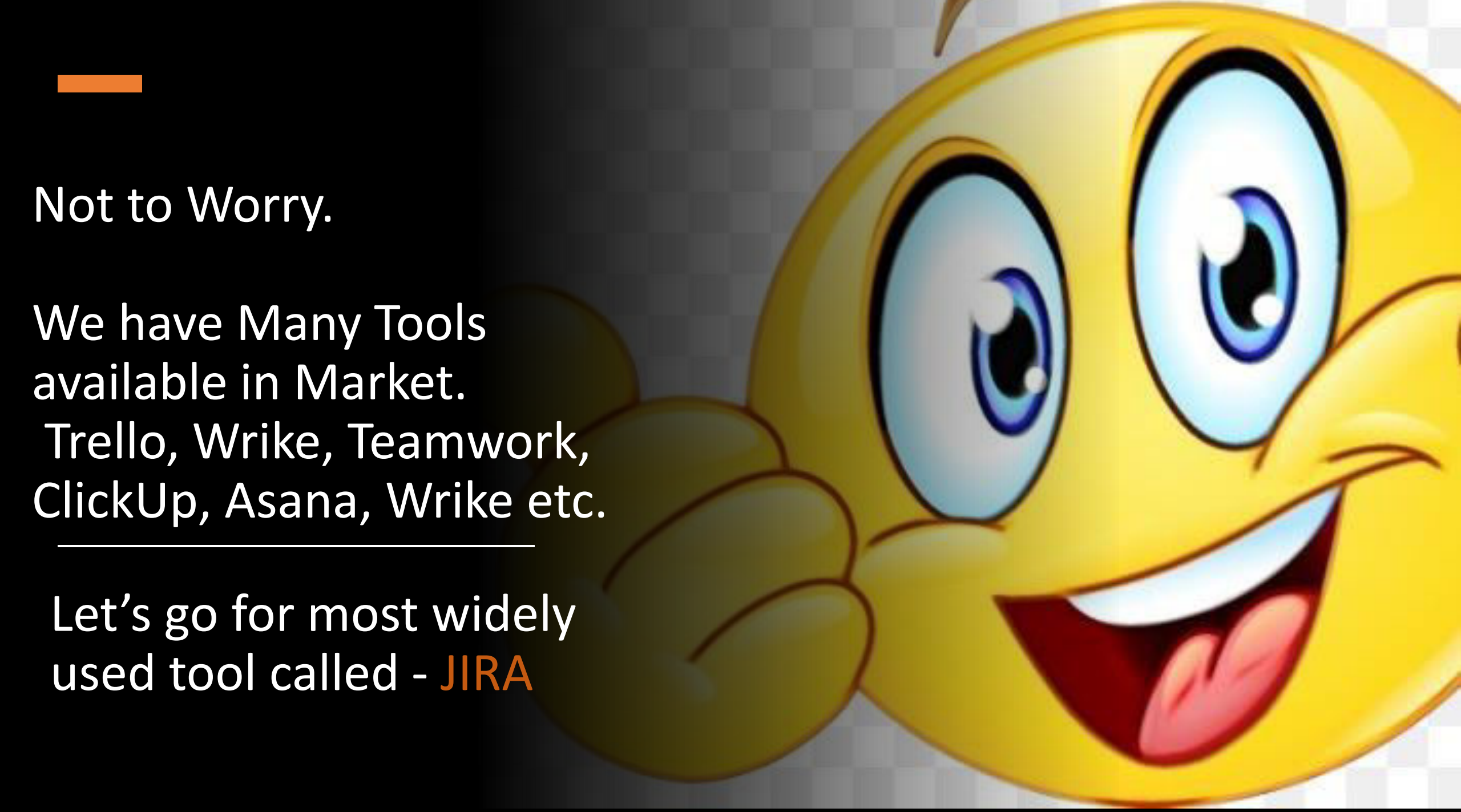
- How do I apply my agile skills ?
- How do I Manage ?

Which Tool will help me ?

Not to Worry.

We have Many Tools available in Market.
Trello, Wrike, Teamwork, ClickUp, Asana, Wrike etc.

Let’s go for most widely used tool called - JIRA

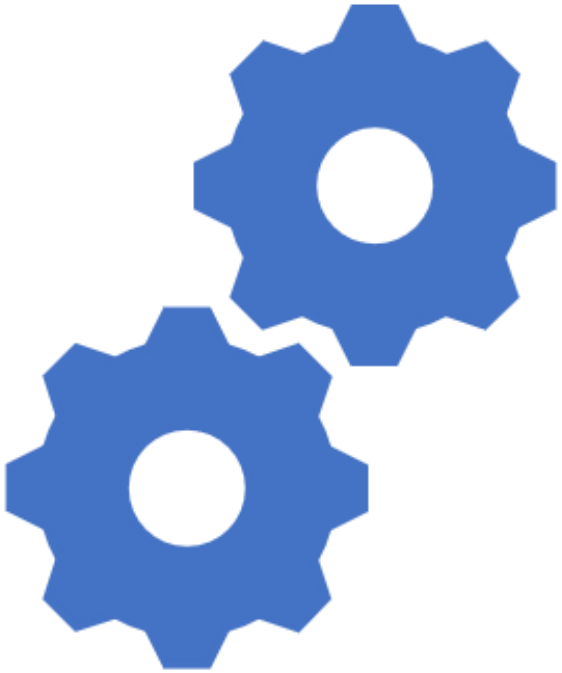


What is JIRA?

JIRA is powerful Work management Tool.

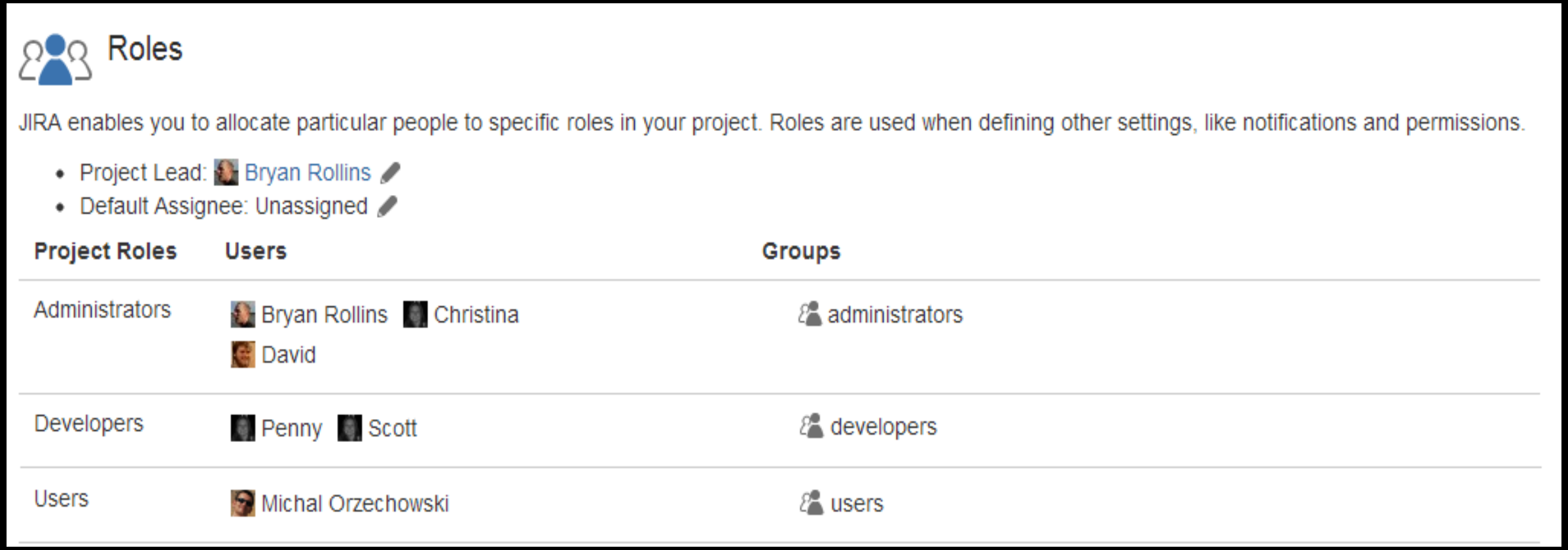
Jira is great at helping your team plan and track all the work.

It can help teams work faster, communicate more effectively due to its well-managed workflow, Mapping and issue tracking ability with minimal effort.



Jira – Default Roles

- Administrators (people who administer a given project).
- Developers (people who work on issues in each project).
- Users (people who log issues on a given project).



Roles

JIRA enables you to allocate particular people to specific roles in your project. Roles are used when defining other settings, like notifications and permissions.

- Project Lead: Bryan Rollins
- Default Assignee: Unassigned

Project Roles	Users	Groups
Administrators	Bryan Rollins, Christina, David	administrators
Developers	Penny, Scott	developers
Users	Michal Orzechowski	users

Jira - Activities

Configure Boards

Configure and Generate Reports

Get Active Sprint Data

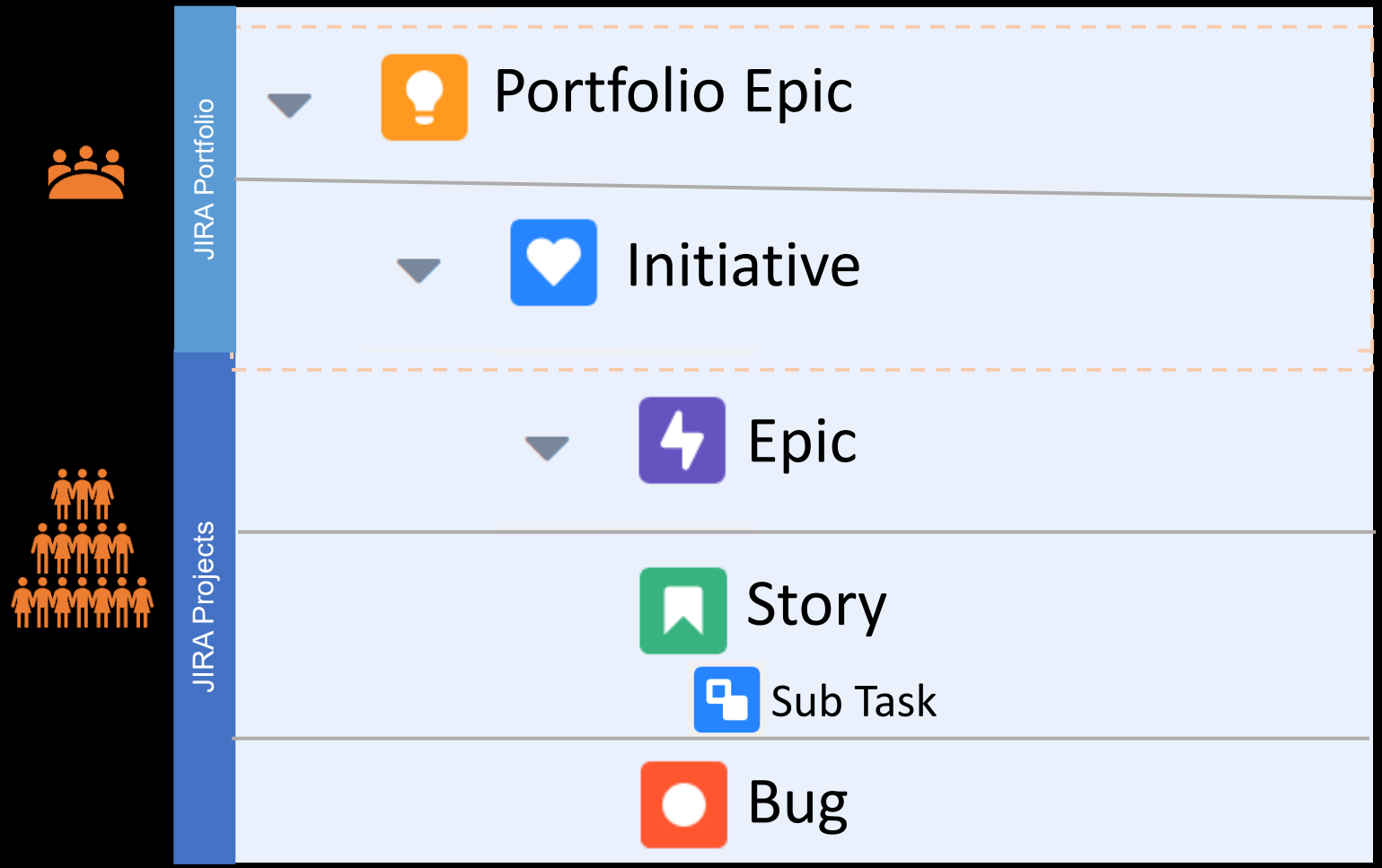
Manage Backlog

Plan and Manage Sprints

Plan and Manage Stories (User Story, Bug)

Setup Project

JIRA Hierarchy and Story definition



Portfolio Epic and Initiative set by PMO and/or Lead BA/Product Owner

Epics set by Asset BA/ Product Owner

User Story Types:

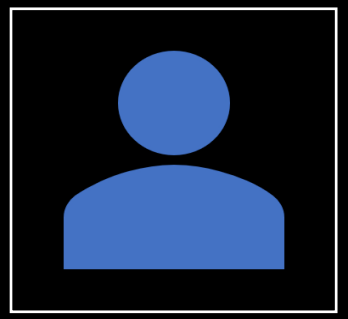
- Development Story [BA]
- Technical Debt Story [Dev Lead]
- Automated Test story [Dev/QA]
- QA Development story [QA]
- SIT Test story [E2E QA Lead]
- UAT Test story [BA]

No Spike or Chore Issue types

Plan and Manage Sprints

How to create Sprint

How to Start Sprint

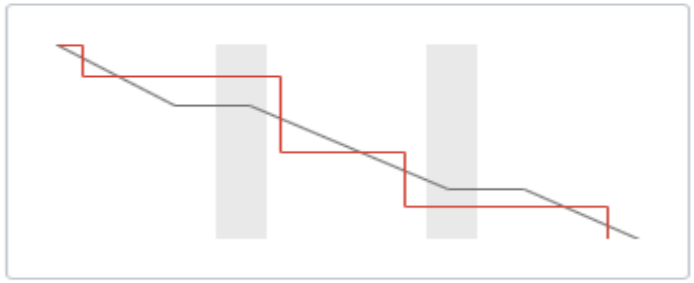


How to Manage Sprint

How to Manage Sprint Backlog

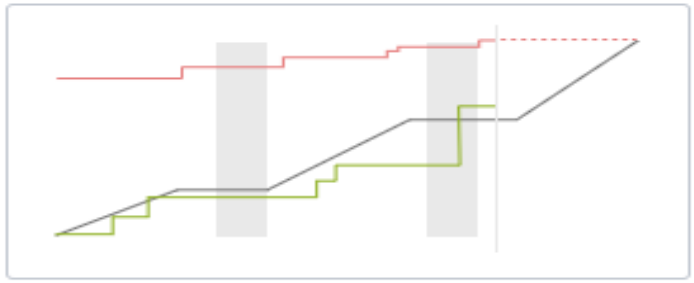
Jira – Configure and Generate Reports

Agile



Burndown Chart

Track the total work remaining and project the likelihood of achieving the sprint goal. This helps your team manage its progress and respond accordingly.

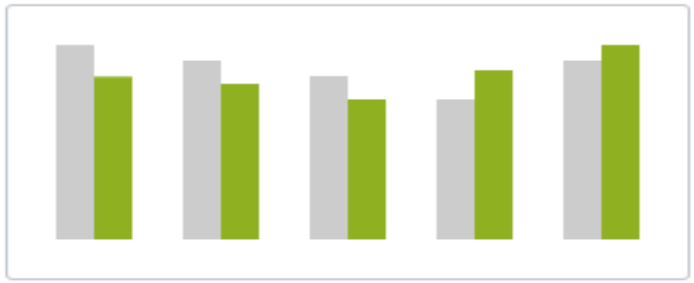


Burnup Chart

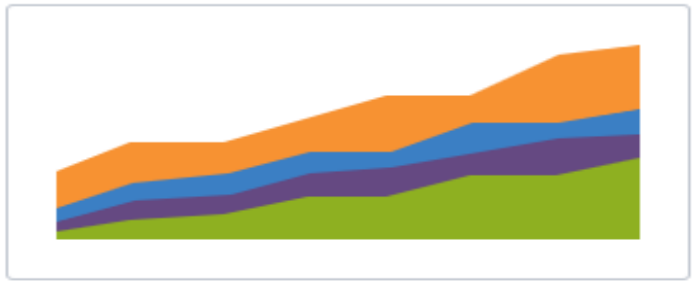
Track the total scope independently from the total work done. This helps your team manage its progress and better understand the effect of scope change.

Sprint Report

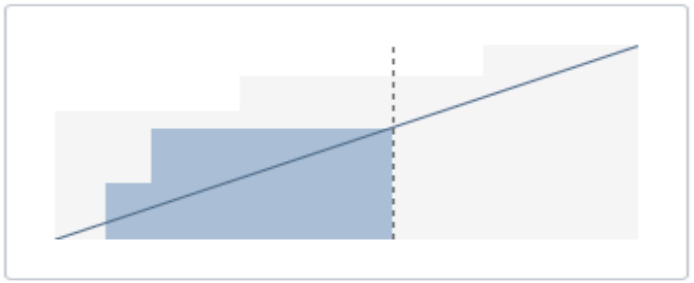
Understand the work completed or pushed back to the backlog in each sprint. This helps you determine if your team is overcommitting or if there is excessive scope creep.



Velocity Chart



Cumulative Flow Diagram



Version Report

Software Testing– An Introduction

Software testing is a process of checking software applications and products is Bugs/Defect free to ensure their performance is efficient. And whether the actual software product matches expected requirements.

It is more important toa demonstrate that the software is not doing what it is not supposed to do.

Testing in software engineering is a fundamental process of creating reliable – and usable – software products

The goal of software testing is to find errors, gaps, or missing requirements in comparison to the actual requirements

Testing is a DESTRUCTIVE PROCESS : A CREATIVE DESTRUCTION.

Testing Need a Negative approach

Successful testing requires a methodical approach

Establishing the proper “Test to Break” mental attitude has a profound effect on testing success.

Software Testing - Principles

7 Principles Of Software Testing

Testing Shows the Presence of Defects -
Every application or product is released into production after enough testing by different teams or passes through different phases like System Integration Testing, User Acceptance Testing, and Beta Testing etc.
So, *have you ever seen or heard from any of the testing team that they have tested the software fully and there is no defect in the software*? Instead of that, every testing team confirms that the software meets all business requirements, and it is functioning as per the needs of the end user.
In the software testing industry, no one will say that there is **no defect** in the software, which is quite true as testing cannot prove that the software is error-free or defect-free.
However, the objective of testing is to find more and more hidden defects using different techniques and methods.
Testing can reveal undiscovered defects and if no defects are found then it does not mean that the software is defect free.

Early Testing –
Testers need to get involved at an early stage of the Software Development Life Cycle (SDLC). Thus, the defects during the requirement analysis phase or any documentation defects can be identified. The cost involved in fixing such defects is very less when compared to those that are found during the later stages of testing.

Software Testing - Importance

Testing is important because software bugs could be expensive or even dangerous.

Software testing is required to identify errors >> Reduce flaws in the component or system.
Software testing is required to Increase the overall quality of the system >> Gain customer. confidence, Satisfaction.
Software testing is required to check software adaptability >> To accelerate software development and adding new features.
Software testing is required to avoid risks.
Software testing is required to avoid extra costs >> To optimise business.
Software testing is required to determining Software performance.
Software testing is required to determining Software Security.
Software testing is required to check the reliability of the software.
Software testing is required to make sure that the final product is user friendly.

Software Testing - Principles

Exhaustive Testing is Not Possible –
It is not possible to test all the functionalities with all valid and invalid combinations of input data during actual testing. Instead of this approach, testing of a few combinations is considered based on priority using different techniques. Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application.

Testing is Context-Dependent –
There are several domains available in the market like Banking, Insurance, Medical, Travel, Advertisement etc. and each domain has several applications. Also, for each domain, their applications have different requirements, functions, different testing purpose, risk, techniques etc.
Different domains are tested differently, thus testing is purely based on the context of the domain or application.
For Example, testing a banking application is different than testing any e-commerce or advertising application. The risk associated with each type of application is different, thus it is not effective to use the same method, technique, and testing type to test all types of application.

Defect Clustering –
Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

Software Testing - Principles

Pesticide Paradox -
Pesticide Paradox principle says that if the same set of test cases are executed again and again over the period then these set of tests are not capable enough to identify new defects in the system.
In order to overcome this “Pesticide Paradox”, the set of test cases needs to be regularly reviewed and revised. If required a new set of test cases can be added and the existing test cases can be deleted if they are not able to find any more defects from the system.

Absence of Error -
If the software is tested fully and if no defects are found before release, then we can say that the software is 99% defect free. But what if this software is tested against wrong requirements? In such cases, even finding defects and fixing them on time would not help as testing is performed on wrong requirements which are not as per needs of the end user.
For Example, suppose the application is related to an e-commerce site and the requirements against “Shopping Cart or Shopping Basket” functionality which is wrongly interpreted and tested. Here, even finding more defects does not help to move the application into the next phase or in the production environment.

Quality Concepts – Cost of Quality

COQ -- Cost of Quality

Prevention Cost

Money required to prevent errors and to do the job right the first time. money spent on establishing methods and procedures

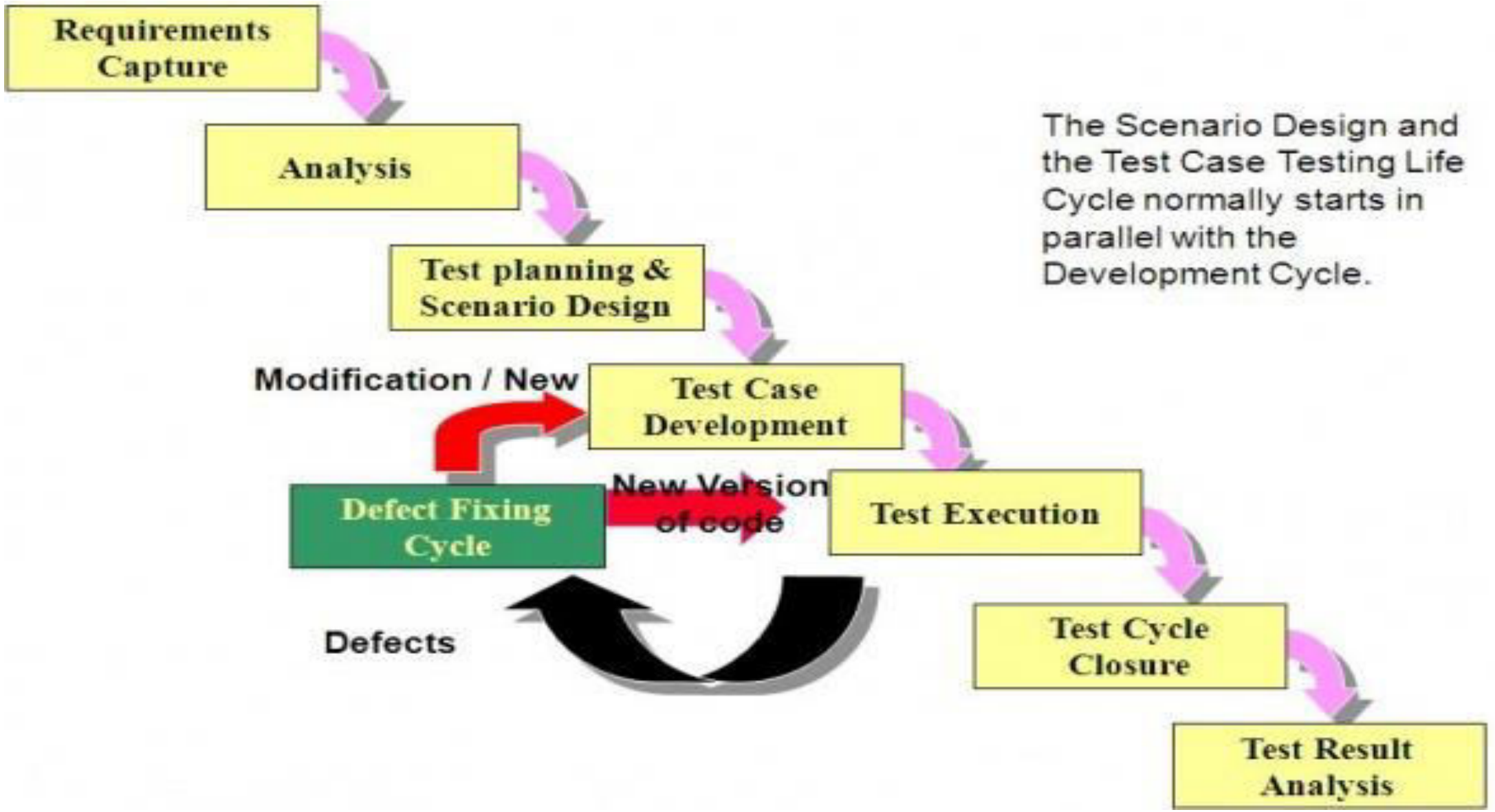
Appraisal Cost

Money spent to review completed products against requirements. Appraisal includes the cost of inspections, testing, and reviews. This money is spent after the product is built but before it is shipped to the user or moved into production.

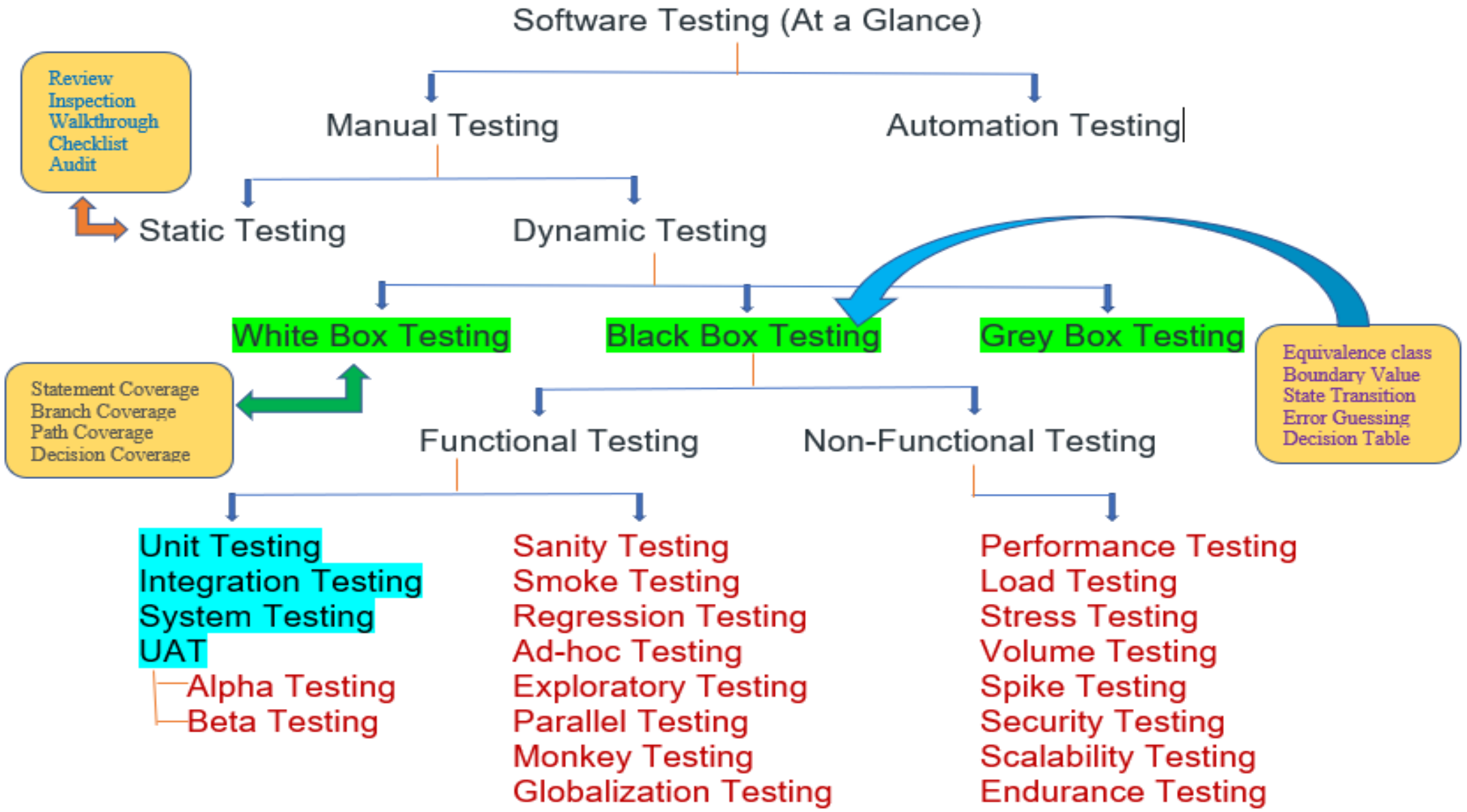
Failure Cost

All costs associated with defective products that have been delivered to the user or moved into production.

Software Testing Life Cycle



Software Testing – At a Glance

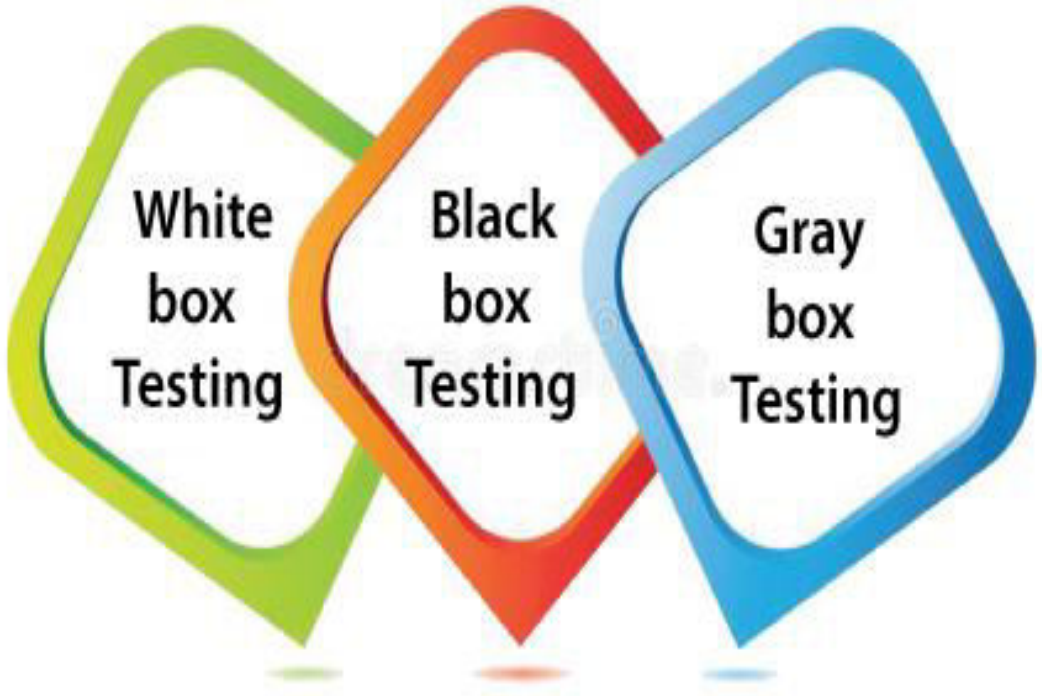


Software Testing Methodologies

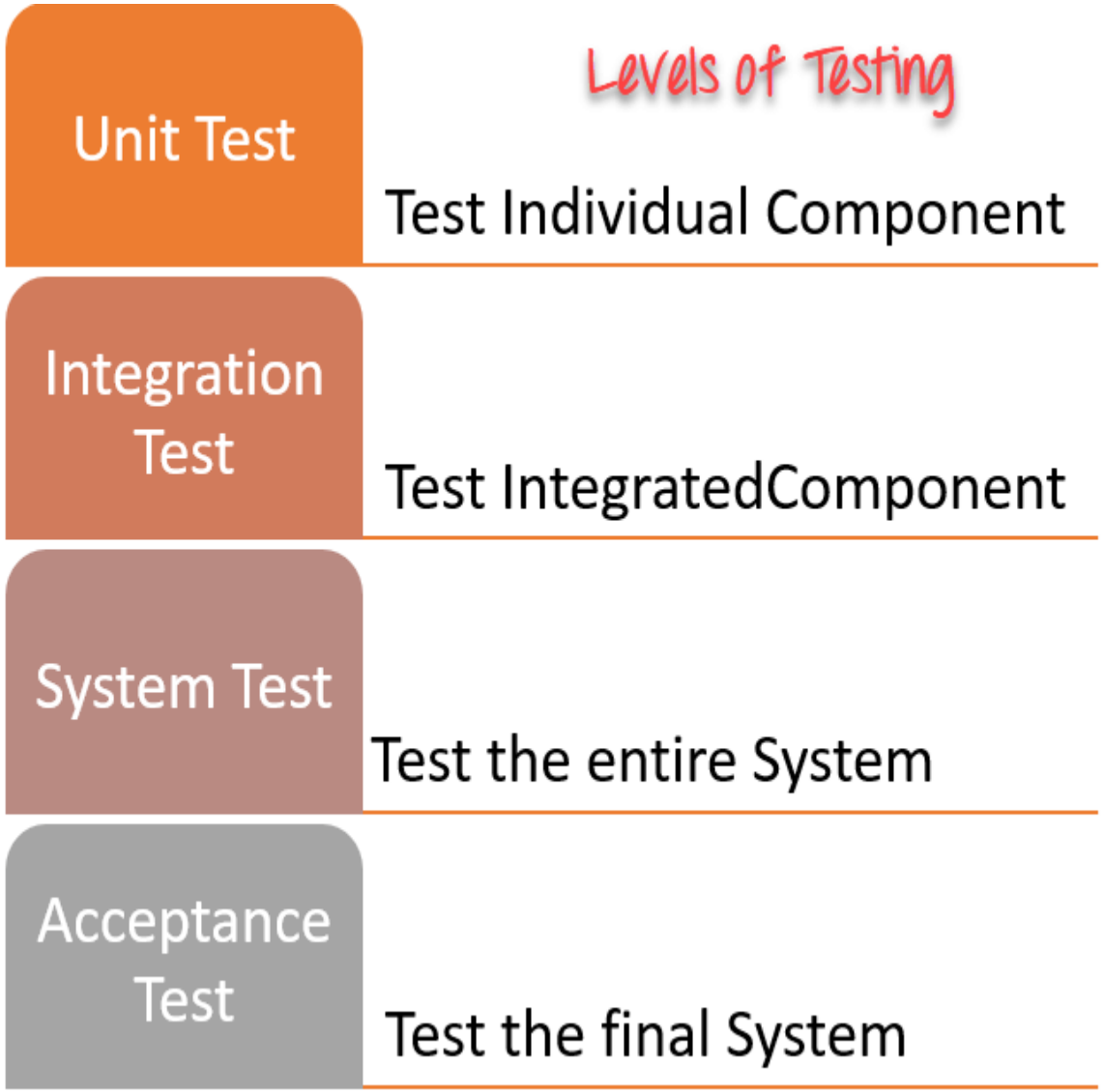
White-box testing
The white box testing is done by Developer, where they check every line of a code before giving it to the Test Engineer. Since the code is visible for the Developer during the testing, that's why it is also known as White box testing.

Black box testing
The black box testing is done by the Test Engineer, where they can check the functionality of an application or the software according to the customer /client's needs. In this, the code is not visible while performing the testing; that's why it is known as black-box testing.

Gray Box testing
Gray box testing is a combination of white box and Black box testing. It can be performed by a person who knew both coding and testing. And if the single person performs white box, as well as black-box testing for the application, is known as Gray box testing.



Levels of Software Testing



Unit Testing :
In this type of testing, errors are detected individually from every component or unit by individually testing the components or units of software to ensure that if they are fit for use by the developers. It is the smallest testable part of the software.

Integration Testing :
Two or more modules which are unit tested are integrated to test i.e., technique interacting components and are then verified if these integrated modules work as per the expectation or not and interface errors are also detected.

System Testing :
In system testing, complete and integrated Software's are tested i.e., all the system elements forming the system is tested as a whole to meet the requirements of the system.

Acceptance Testing :
It is a kind of testing conducted to ensure whether the requirement of the users are fulfilled prior to its delivery and the software works correctly in the user's working environment.

Types of Testing

Sanity Testing
Sanity testing is a stoppage to check whether testing for the build can proceed or not. The focus of the team during sanity testing process is to validate the functionality of the application and not detailed testing. Sanity testing is generally performed on build where the production deployment is required immediately like a critical bug fix.

Smoke Testing
This test is done to make sure that software under testing is ready or stable for further testing
It is called a smoke test as the testing an initial pass is done to check if it did not catch the fire or smoke in the initial switch on.

Regression Testing
Every time a new module is added leads to changes in the program. This type of testing makes sure that the whole system works properly even after adding or changing components to the complete program. (Impact)

Parallel Testing: Testing technique which has the purpose to ensure that a new application which has replaced its older version has been installed and is running correctly. It is conducted by the testing team.

Exploratory Testing: Black box testing technique performed without planning and documentation. It is usually performed by manual testers.

Ad-hoc Testing:

Types of Testing

Usability Testing: Testing technique which verifies the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component. It is usually performed by end users.

Mutation Testing: Method of software testing which involves modifying programs' source code or byte code in small ways in order to test sections of the code that are seldom or never accessed during normal tests execution. It is normally conducted by testers

End to End Testing
Like system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate. It is performed by QA teams:

Localization Testing

Types of Testing

Load Testing: Testing technique that puts demand on a system or device and measures its response. It is usually conducted by the performance engineers.

Stress Testing
In this, we give unfavorable conditions to the system and check how they perform in those conditions.

Performance Testing
It is designed to test the run-time performance of software within the context of an integrated system. It is used to test the speed and effectiveness of the program. It is also called load testing. In it we check, what is the performance of the system in the given load.

Volume Testing: Testing which confirms that any values that may become large over time (such as accumulated counts, logs, and data files), can be accommodated by the program and will not cause the program to stop working or degrade its operation in any manner. It is usually conducted by the performance engineer.

Recovery Testing: Testing technique which evaluates how well a system recovers from crashes, hardware failures, or other catastrophic problems. It is performed by the testing teams

Security Testing
Security testing unveils the vulnerabilities of the system to ensure that the software system and application are free from any threats or risks. These tests aim to find any potential flaws and weaknesses in the software system that could lead to a loss of data, revenue, or reputation per employees or outsides of a company.

Types of Testing

Scalability Testing: Part of the battery of non-functional tests which tests a software application for measuring its capability to scale up – be it the user load supported, the number of transactions, the data volume etc. It is conducted by the performance engineer.

Compatibility Testing: Testing technique that validates how well a software performs in a particular hardware/software/operating system/network environment. It is performed by the testing teams.

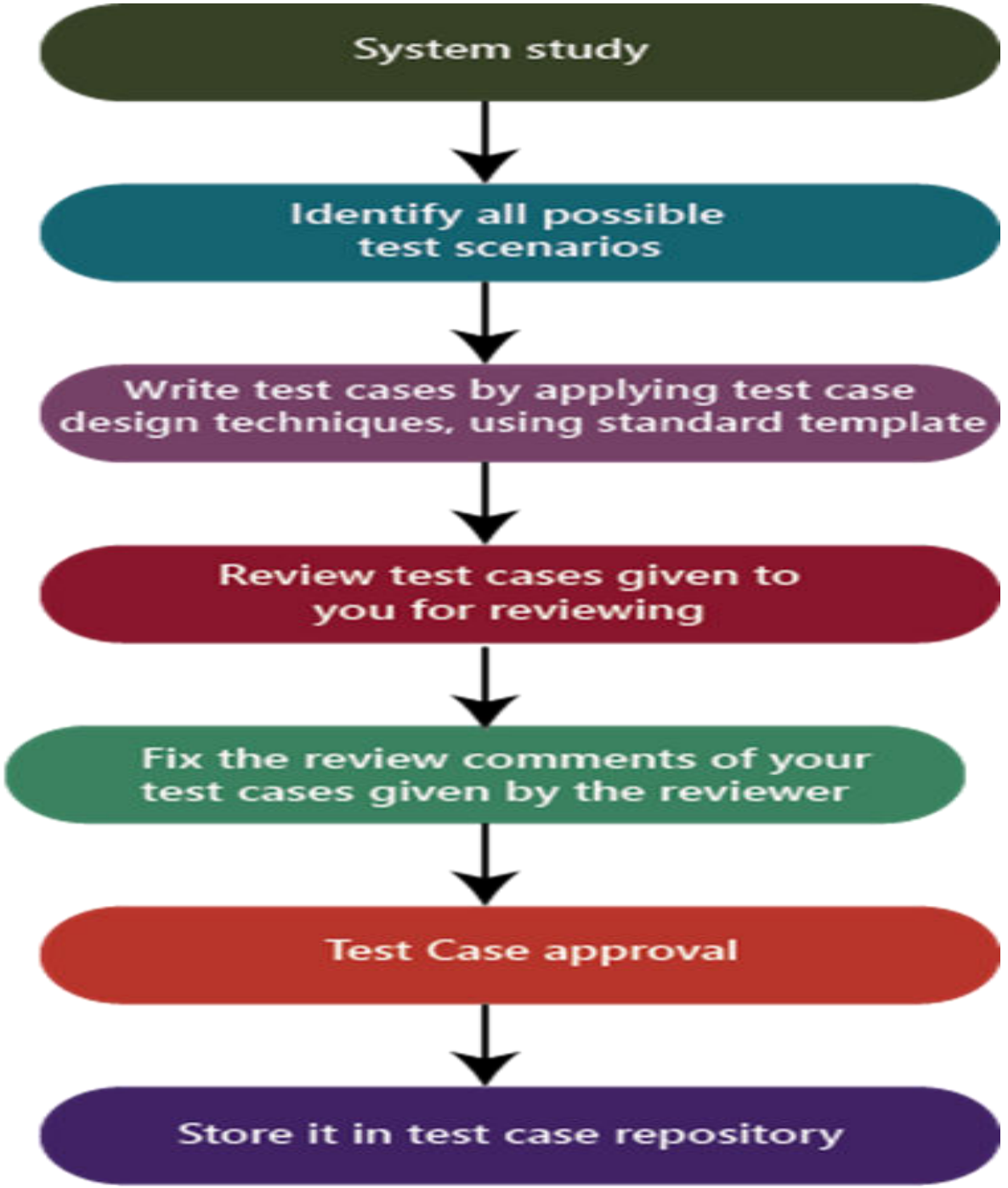
Penetration Testing: Testing method which evaluates the security of a computer system or network by simulating an attack from a malicious source. Usually, they are conducted by specialized penetration testing companies.

Acceptance Testing: Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system. It is usually performed by the customer.

Alpha Testing
Alpha Testing is a type of software testing performed to identify bugs before releasing the software product to the real users or pubic, It is a type of acceptance testing which is done before the product is released to end user and performed in controlled environment. It is typically done by QA people.

Beta Testing
The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for a limited number of users for testing in a real-time environment

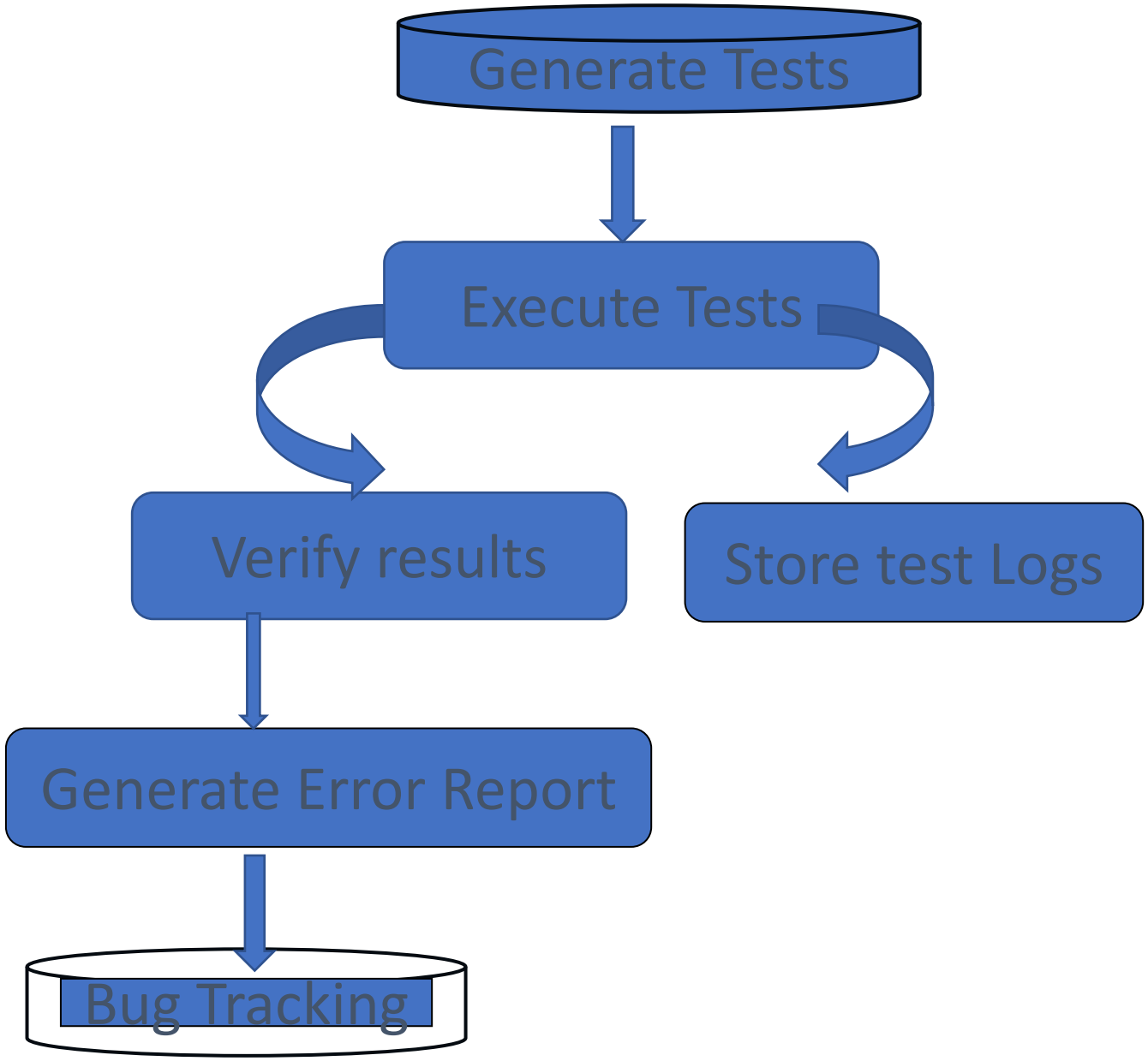
Test Case - Process



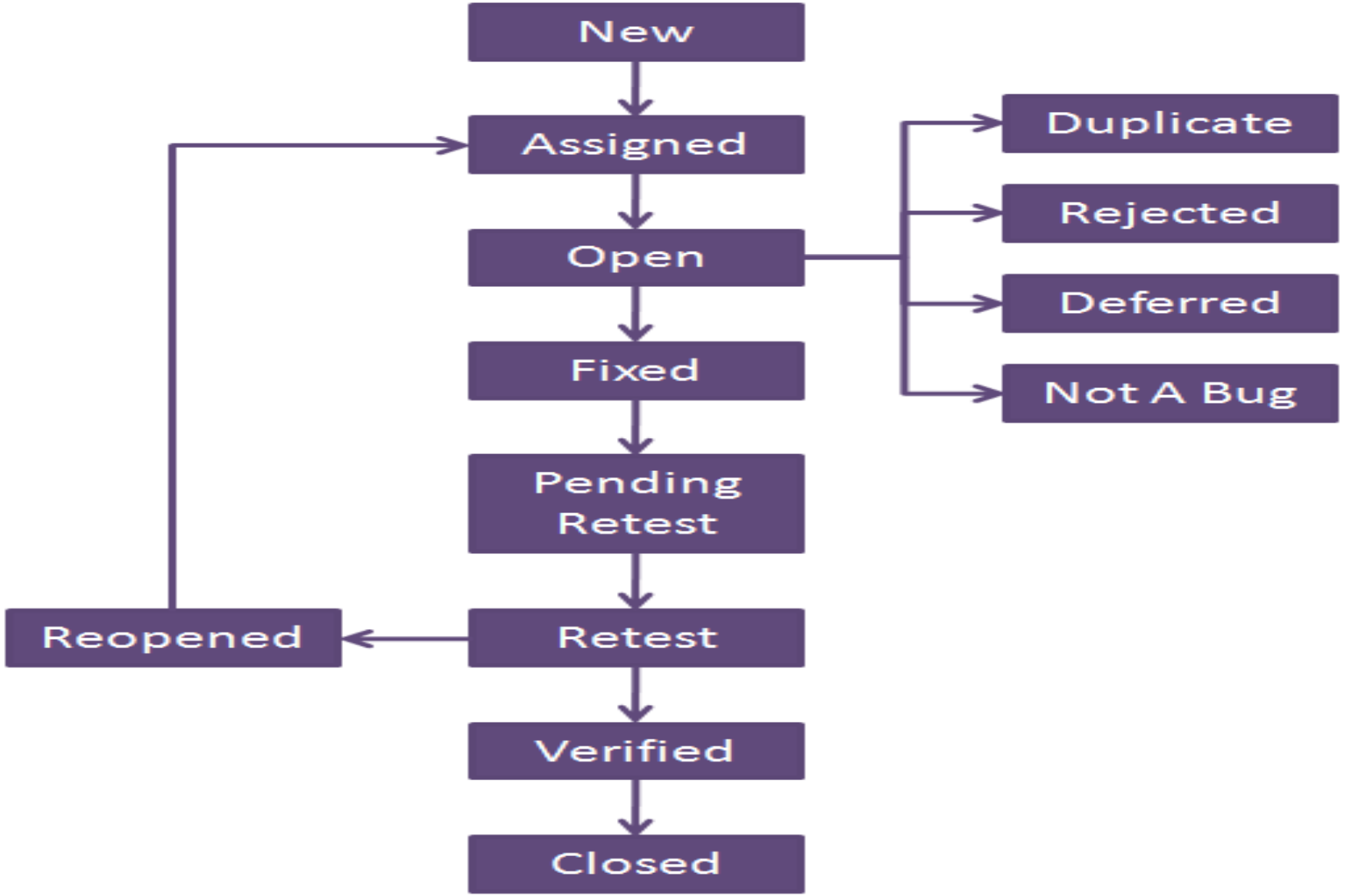
Best Practices To Write Good Test Case

- Easy to understand and execute
- Create Test Cases with End User's perspective
- Unique Test case Identifiers must be used. It allows us to track them easily.
- Prerequisites should be listed clearly. Helps to execute the test case without any issues.
- Test data should be defined to evaluate each functional area.
- Test case description should be concise.
- Test Steps should be in detail and clear.
- Specify the exact expected result.
- Position condition should be listed if any.
- Test cases should neither too simple nor too complex.
- Test cases must be distinctive. There should not be no repeated test cases.
- Test cases should be written by following test case design techniques
- Test cases must be comprehensible. So that any tester (even a newly appointed testers) can understand them by perusing once.
- Needs to provide clear environment details where we need to execute them.
- Test cases should be reusable & maintainable
- Get peer review.

Test Case –Execution Flow



Defect Life Cycle



Defect - Priority and Severity

Defect Priority provides a perspective for the order of the defect fixes. Priority in other words tells us , how soon the defect must be fixed.

For Example, the priority can be divided into P1, P2, P3,P4 and so on.
P1 – Fix the defect on highest priority, fix it before the next build
P2 – Fix the defect on high priority before the next test cycle
P3 – Fix the defect on moderate priority when time permits , before the release
P4- Postpone the defect for the next release or live with this defect.

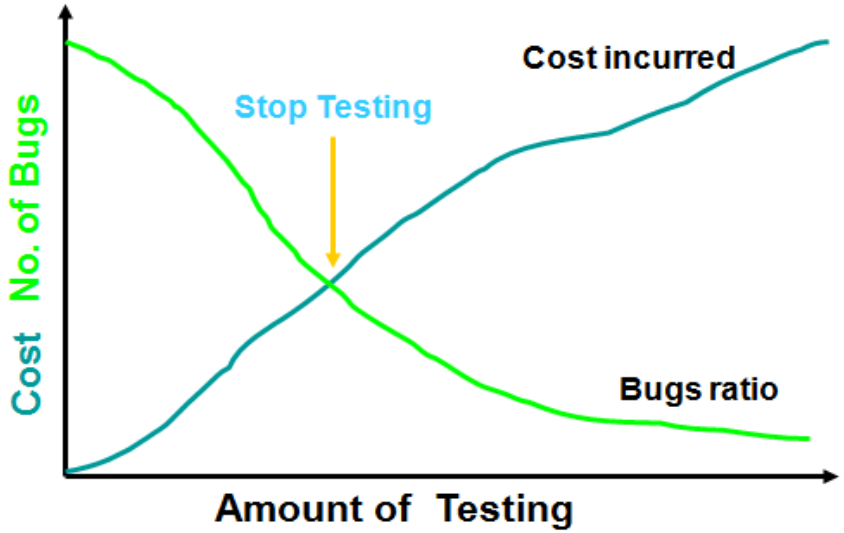
Defect Severity provides the perspective of the impact of that defect in product functionality.
Defect Severity levels can be:

S1,S2,S3, S4 or Extreme, Critical, important, Minor, Cosmetic
Extreme – Product Crashes or is unusable
Critical - Basic functionality of the product is not working.
Important- Extended functionality of the product not working
Minor – Product behaves differently
Cosmetic – GUI related

When to Stop Testing

Test Manager will consider the following factors

- Deadlines, e.g. release deadlines, testing deadlines;
- Test cases completed with certain percentage passed;
- Test budget has been depleted;
- Coverage of code, functionality or requirements reaches a specified point;
- Bug rate falls below a certain level
- Beta or alpha testing period ends



Thank You