Introduction to DevOps

Prof. Manjushree K
Assistant Professor
Dept. of CSE
BNMIT, Bengaluru

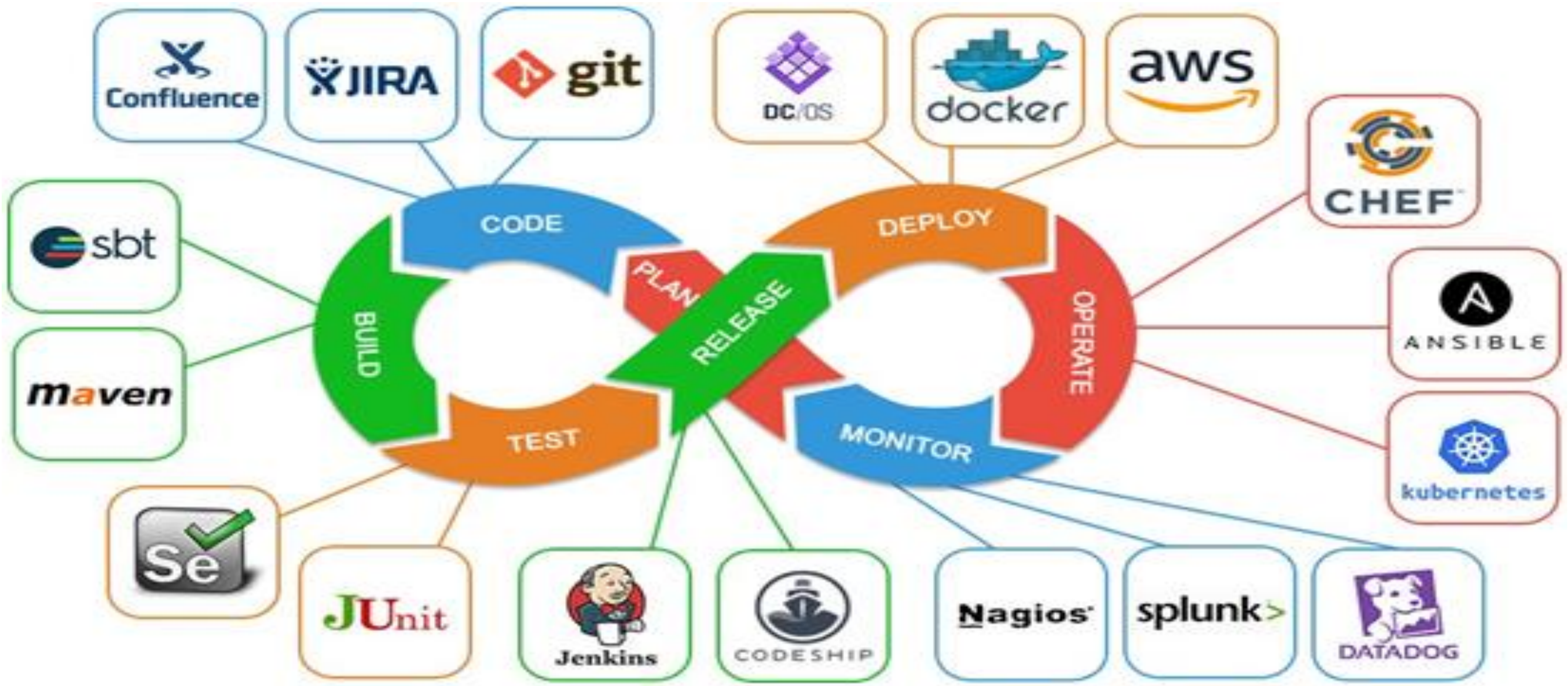
Introduction

- **Definition:** It is a **process** or **culture** or **approach** or **mindset** or **practice** that will be followed in an organization or company for better delivery of application or software product.
- Devops is all about

1. Improving Delivery
2. Automation (automated the process)
3. Quality (quality is maintained)
4. Monitoring (continuous monitoring if automation is failed or compromised)
5. Testing

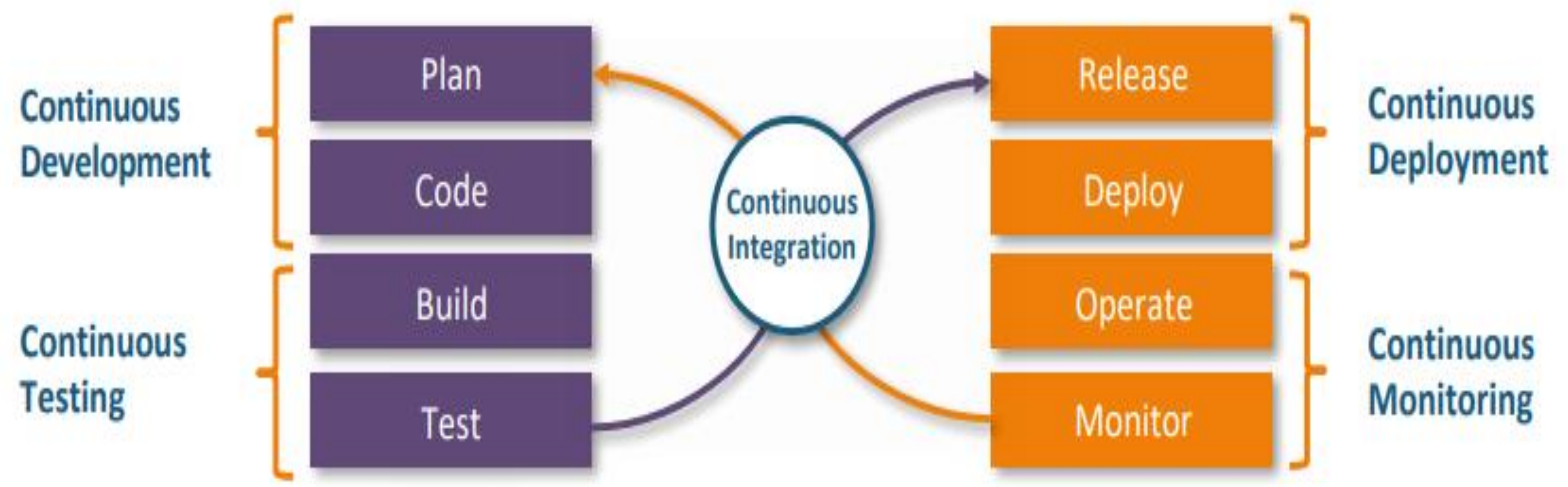
- End goal of any company is to delivery of the application

Devops Tools



DevOps is a set of Practices and tools designed to shorten the life cycle of software development process.

Cont…


Continuous Development
Continuous Testing
Plan
Code
Build
Test
Continuous Integration
Release
Deploy
Operate
Monitor
Continuous Deployment
Continuous Monitoring

DEVelopment + OPerations = DEVOPS

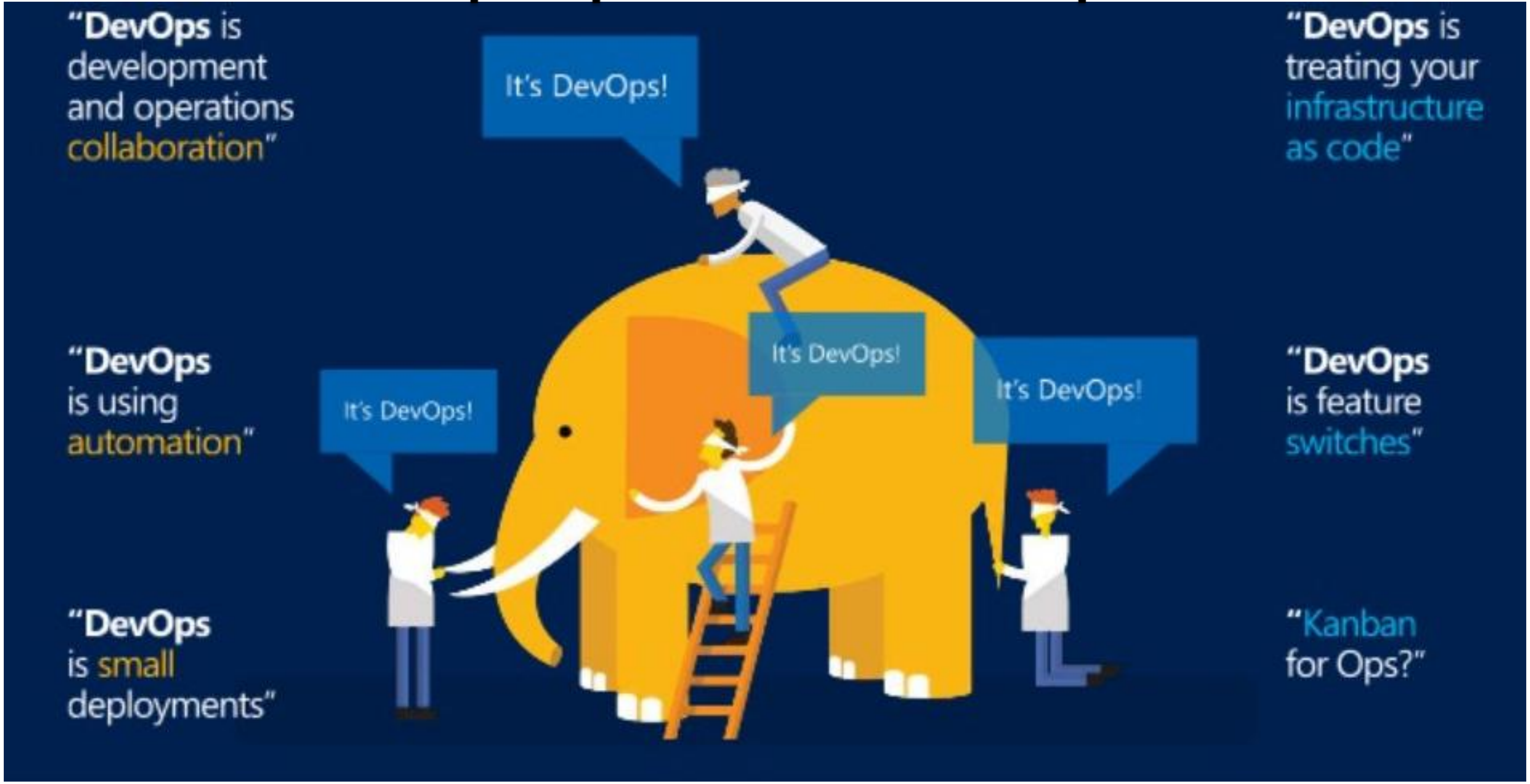
DEVelopment

- Gather Requirements
- Design
- Code
- Test
- Prepare working Software

OPerationS

- **Deploy Software into production**
- Monitor live issues
- First level of defence
- Upgrade production with new releases

What people think DevOps is ?


"DevOps is development and operations collaboration"
"DevOps is using automation"
"DevOps is small deployments"
"DevOps is treating your infrastructure as code"
"DevOps is feature switches"
"Kanban for Ops?"
It's DevOps!
It's DevOps!
It's DevOps!
It's DevOps!

Each person insists **"It's DevOps!"**, but they are only seeing **one piece** of the full picture.

Cont…

- What it *means* in the context of DevOps
- DevOps is not one tool, one practice, or one process. It is the combination of many practices and mindsets working together.
- Like the blind men who each think the elephant is something different, people in organizations often define DevOps only by the part they interact with.

What they see	Why it's only one part of DevOps
Automation	Automation helps, but DevOps also requires culture, feedback loops, collaboration.
Small deployments	True, but deployment size alone doesn't define DevOps.
Infrastructure as Code	Important for consistency, but DevOps is broader.
Collaboration	Essential, yet still not the whole picture.
Feature switches	A technique used in continuous delivery, not DevOps by itself.
Kanban	A workflow tool, helpful but not the essence of DevOps.

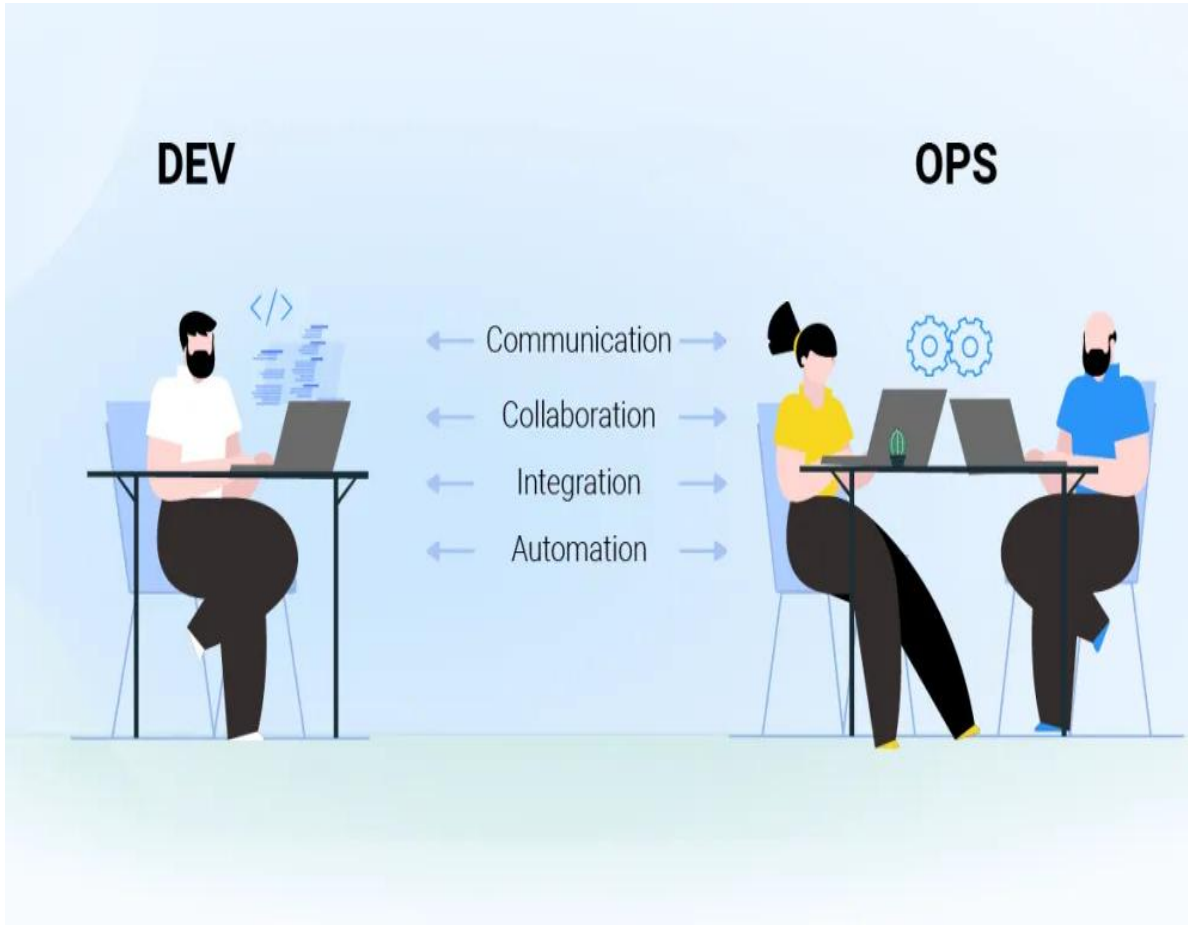
Cont…

- You cannot understand DevOps by focusing on just:
- tools
- automation
- deployment frequency
- infrastructure
- processes
- All these together—plus continuous improvement and that is DevOps.
- So what is DevOps really?
- A complete definition would be:
- DevOps is a cultural and technical movement that improves how development and operations work together to deliver software faster, more reliably, and with continuous feedback.

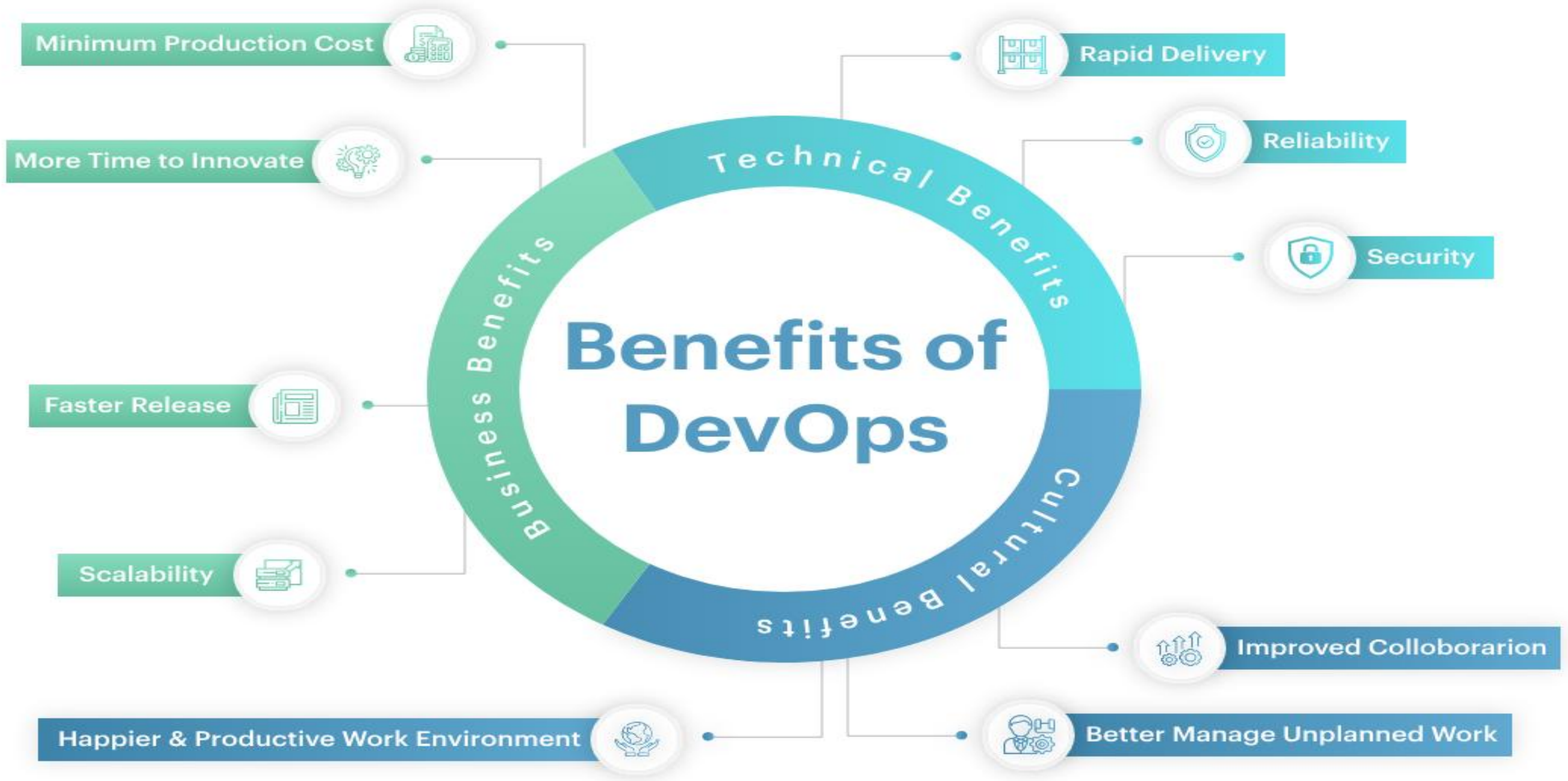
DevOps Is.....

A Professional cultural movement/philosophy/mindset emphasizing on..

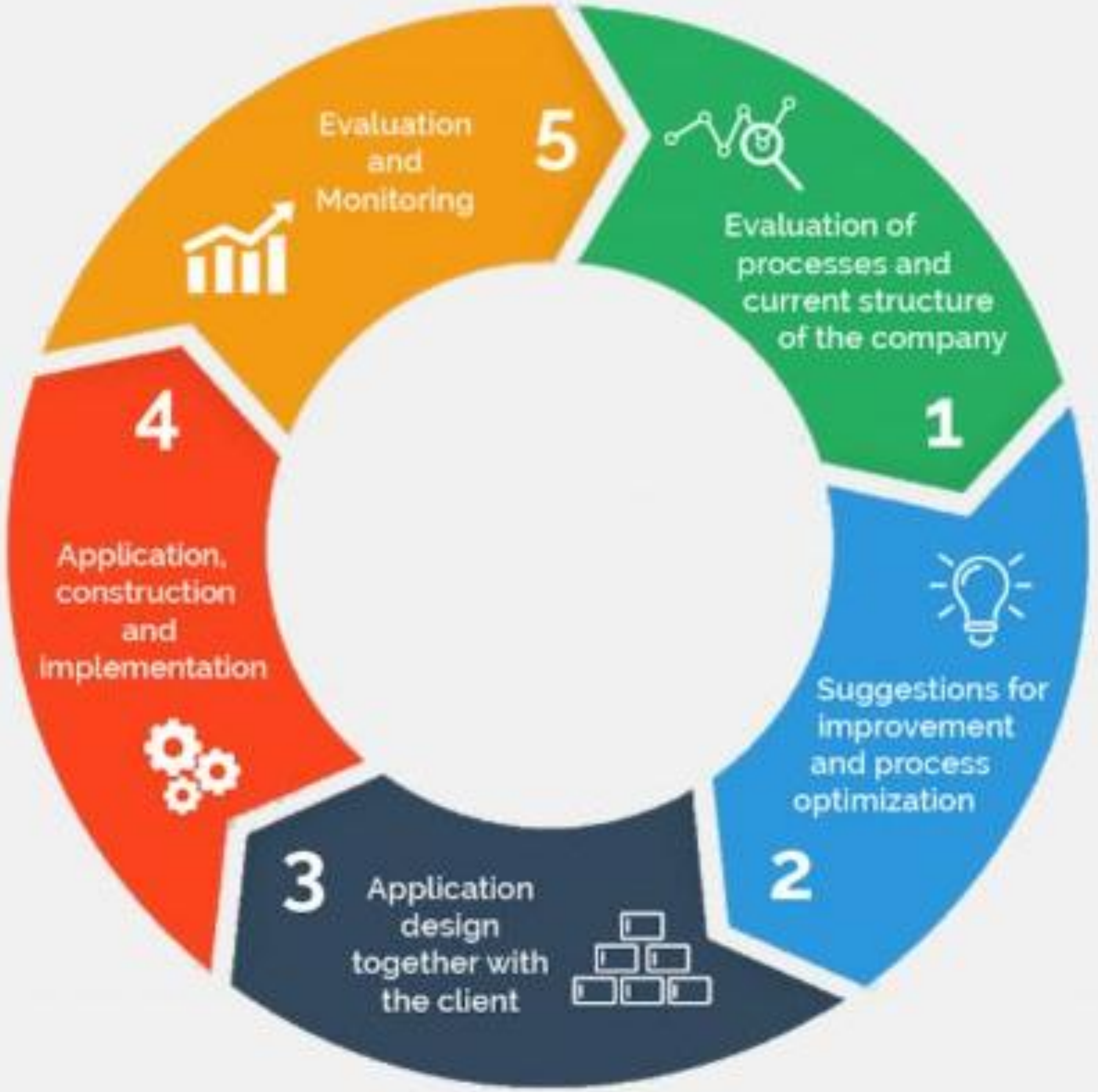
- Continuous Collaboration between Dev and Ops
- Automated CI/CD pipeline working in Small batches, with short lead-time and low failure-rates
- Agile Practices applied to whole software development cycle



Benefits of DevOps



What is AGILE Software Development?

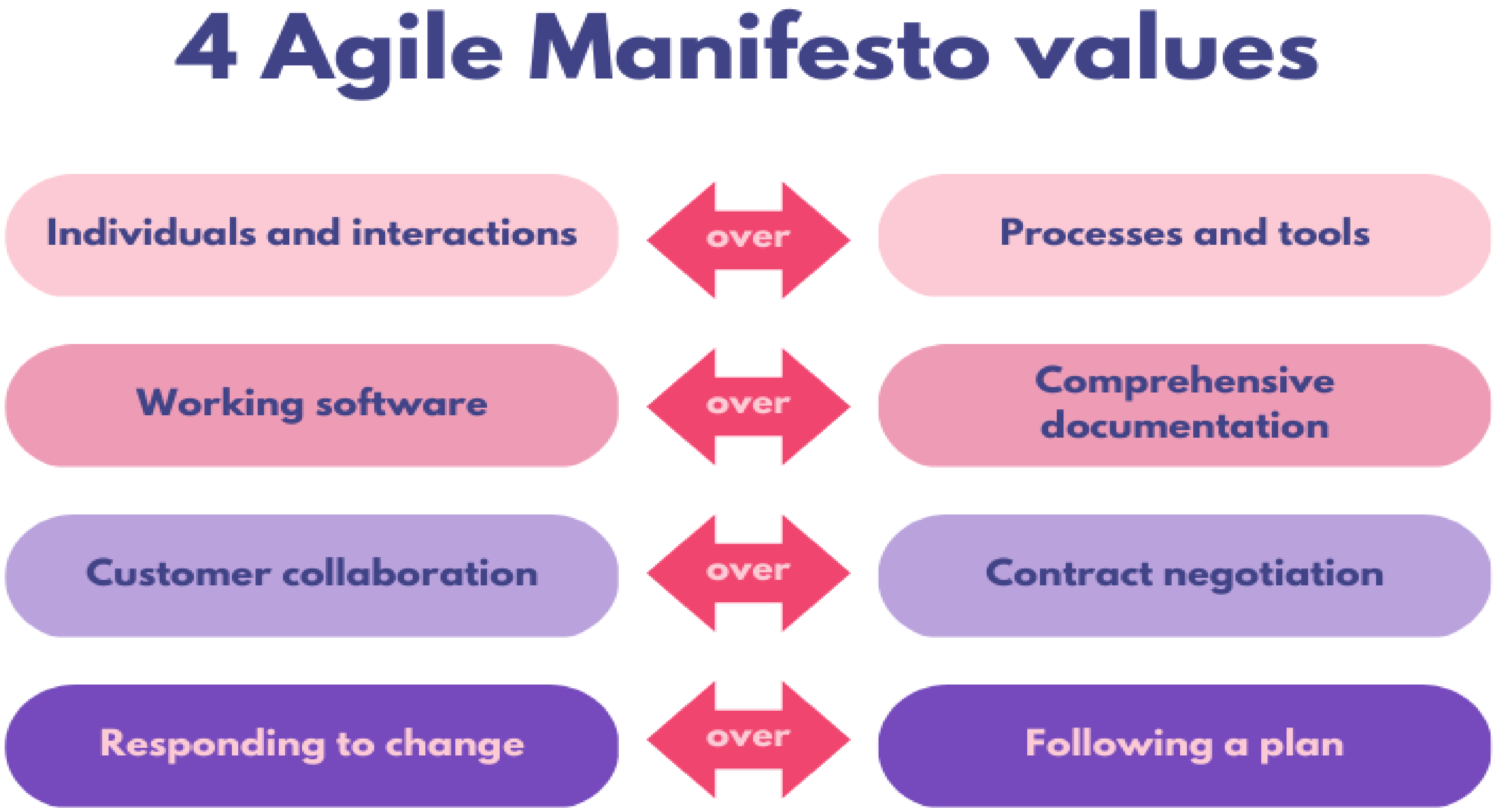


5
Evaluation and Monitoring
1
Evaluation of processes and current structure of the company
2
Suggestions for improvement and process optimization
3
Application design together with the client
4
Application, construction and implementation

Agile Development

- Agile development is a project management methodology that values individuals and interactions over processes and tools.
- All life cycle features will be developed in one month or 2-3 weeks.
- Instead of delivering at the end of year it is being released every month or within the time constraint.
- Delivered very quickly
- More focus on development rather than operation

Cont…



4 Agile Manifesto values

Individuals and interactions	over	Processes and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

Cont...

- Agile development focuses on creating working software quickly, collaborating with customers frequently, and being able to adapt to changes easily.
- This methodology is especially beneficial for projects that are complex or have uncertain requirement

How does Agile Development work?

Agile process down into three main stages:

1. Preparation
2. Sprint planning
3. Sprint

Cont…

1. Preparation: In the preparation stage, the product owner creates a backlog of features they want to include in the final product. This is known as the product backlog. Then, the development team estimates how long each feature will take to build.

2. Sprint planning: The sprint planning meeting is where the team decides which features from the product backlog they are going to work on during the sprint.

A sprint is a set period (usually two weeks) during which the development team must achieve a specific goal. The team also decides how many of each type of task they can complete during the sprint.

For example, the team may decide they can complete three coding tasks, two testing tasks, and one documentation task during the sprint. This information is then added to the sprint backlog.

Cont…

- During the sprint, the team works on completing the tasks in the sprint backlog. They may also come across new issues to address. If this happens, they will add these issues to the product backlog and prioritize them accordingly. At the end of the sprint, the development team should have completed all features in the sprint backlog.

- if not, the team will carry them over to the next sprint. The team then holds a sprint review meeting where they demo completed features to the product owner and stakeholders. They also discuss what went well during the sprint and how they could improve their next one.

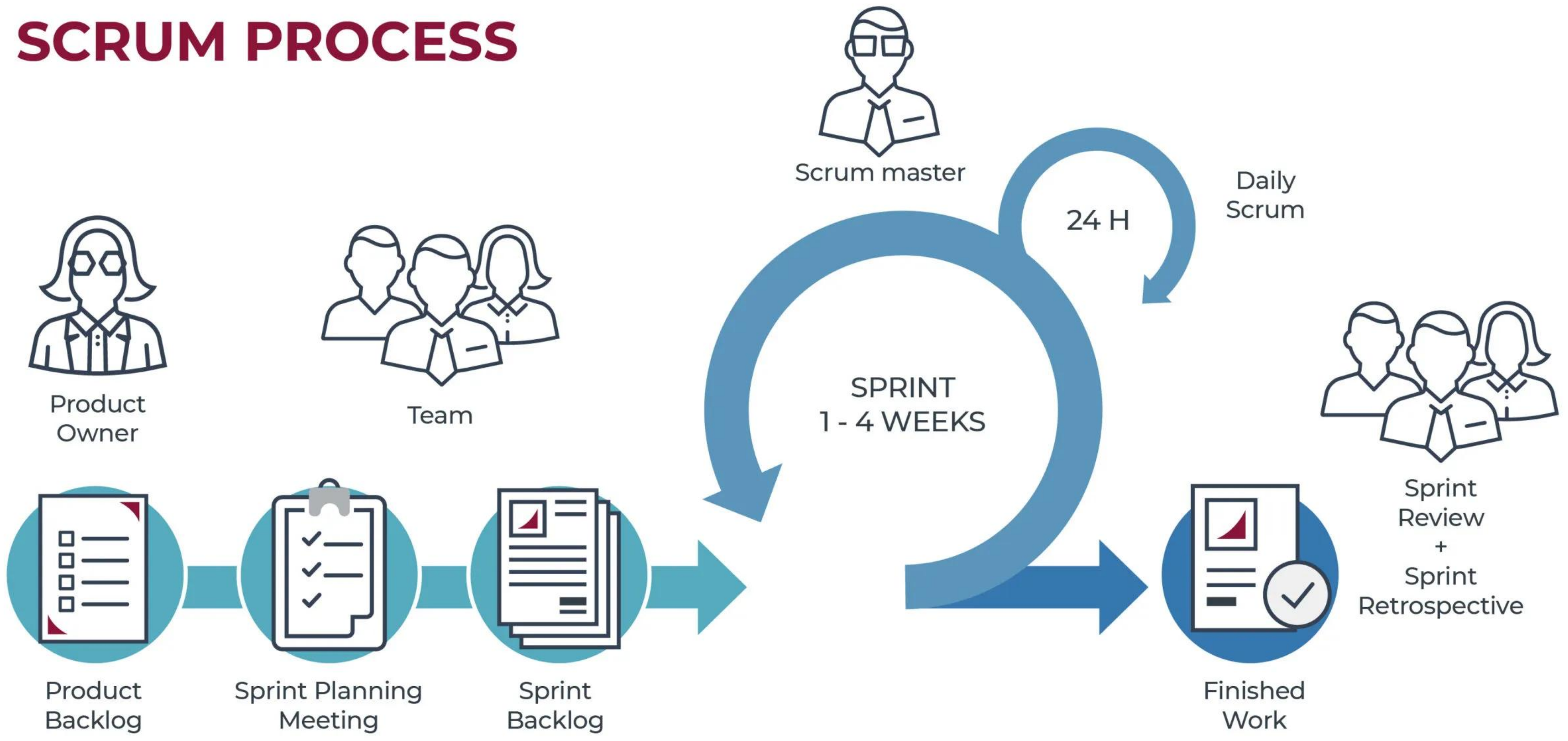
- Finally, the team holds a retrospective meeting, where they reflect on what went well and what didn't go so well during the sprint. They then create a plan of action for addressing these issues in future sprints.

Cont…

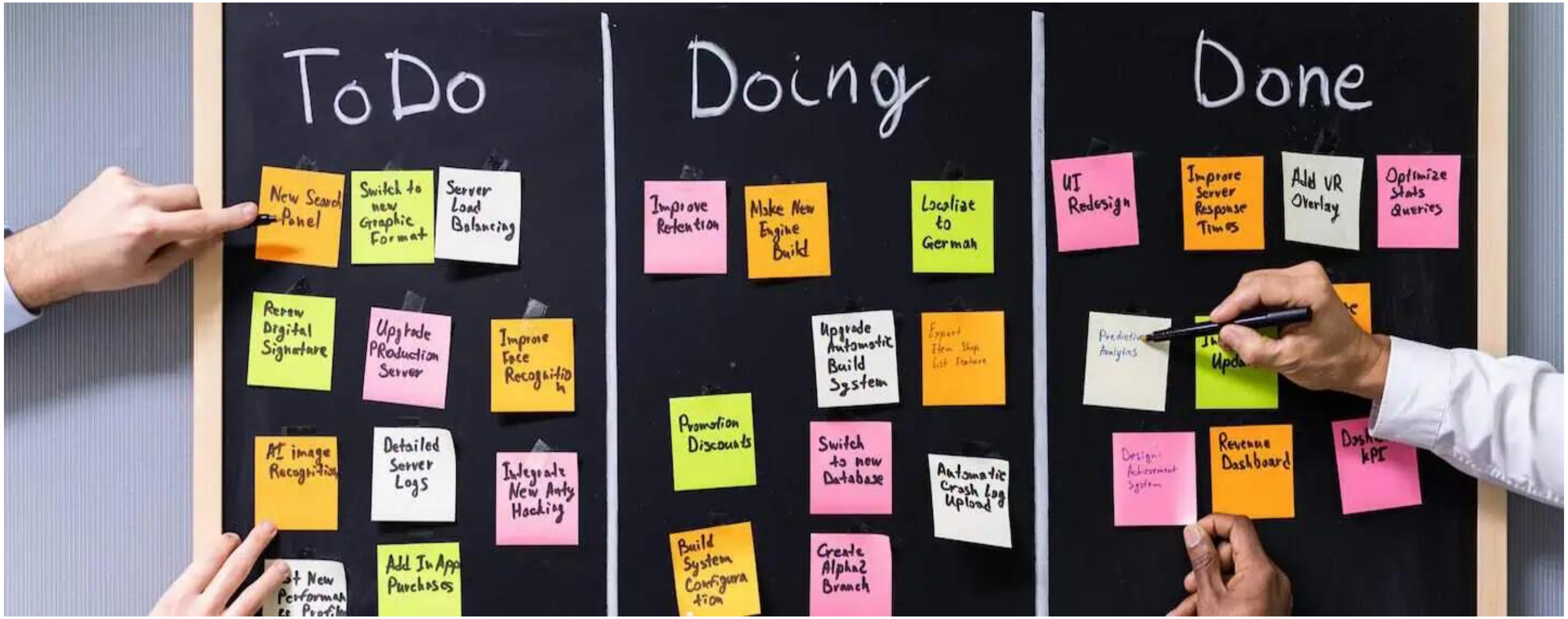
- **Why is Agile Development important?**
- Agile development is important because it helps to ensure that development teams complete projects on time and within budget. It also helps to improve communication between the development team and the product owner.

- Additionally, Agile development methodology can help reduce the risks associated with complex projects

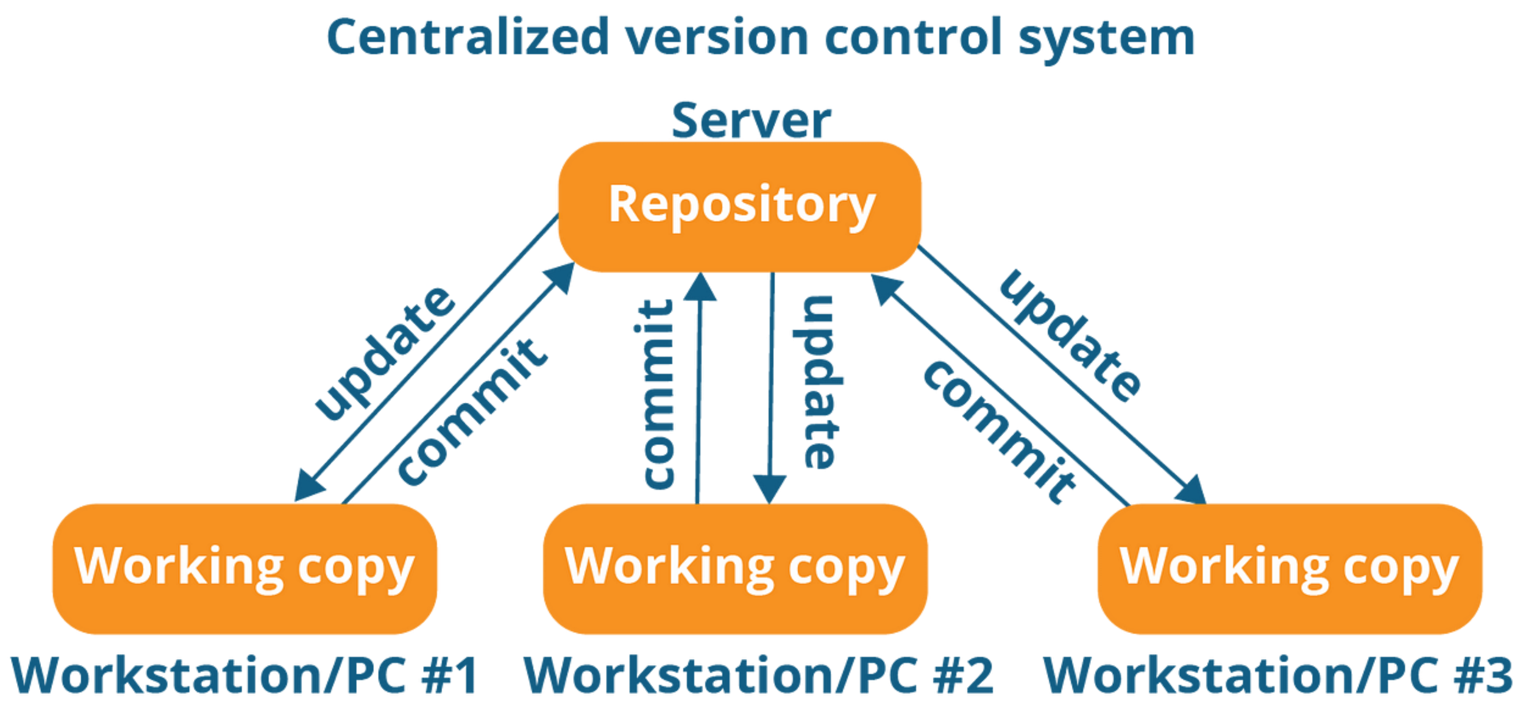
SCRUM PROCESS



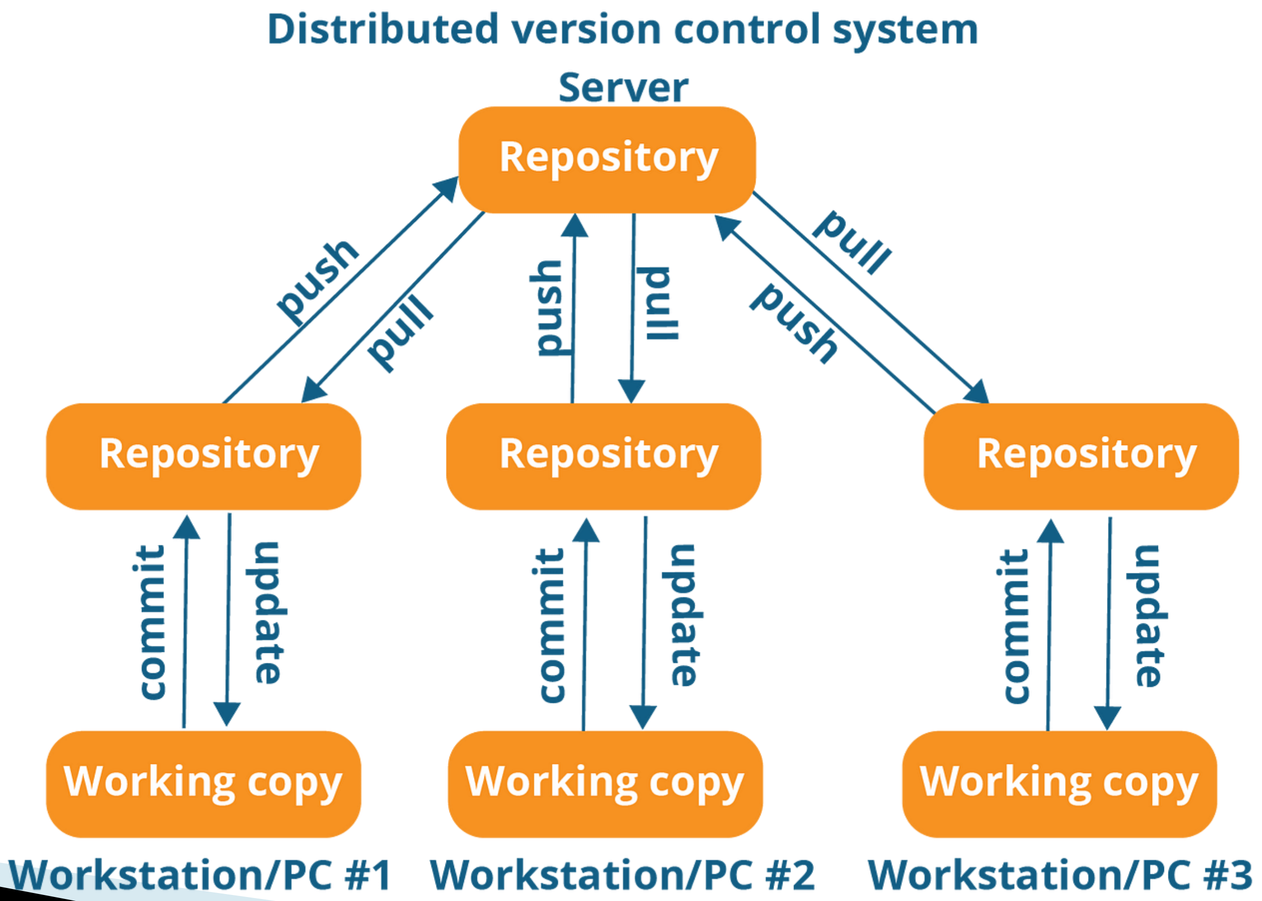
kanban


ToDo
Doing
Done
New Search Panel
Switch to new Graphic Format
Server Load Balancing
Renew Digital Signature
Upgrade Production Server
Improve Face Recognition
AI image Recognition
Detailed Server Logs
Integrate New Anty Hacking
Add In App Purchases
Improve Retention
Make New Engine Build
Localize to German
Upgrade Automatic Build System
Promotion Discounts
Switch to new Database
Automatic Crash Log Upload
Build System Configuration
Create Alpha2 Branch
UI Redesign
Improve Server Response Times
Add VR Overlay
Optimize Stats Queries
Predictive Analytics
Revenue Dashboard

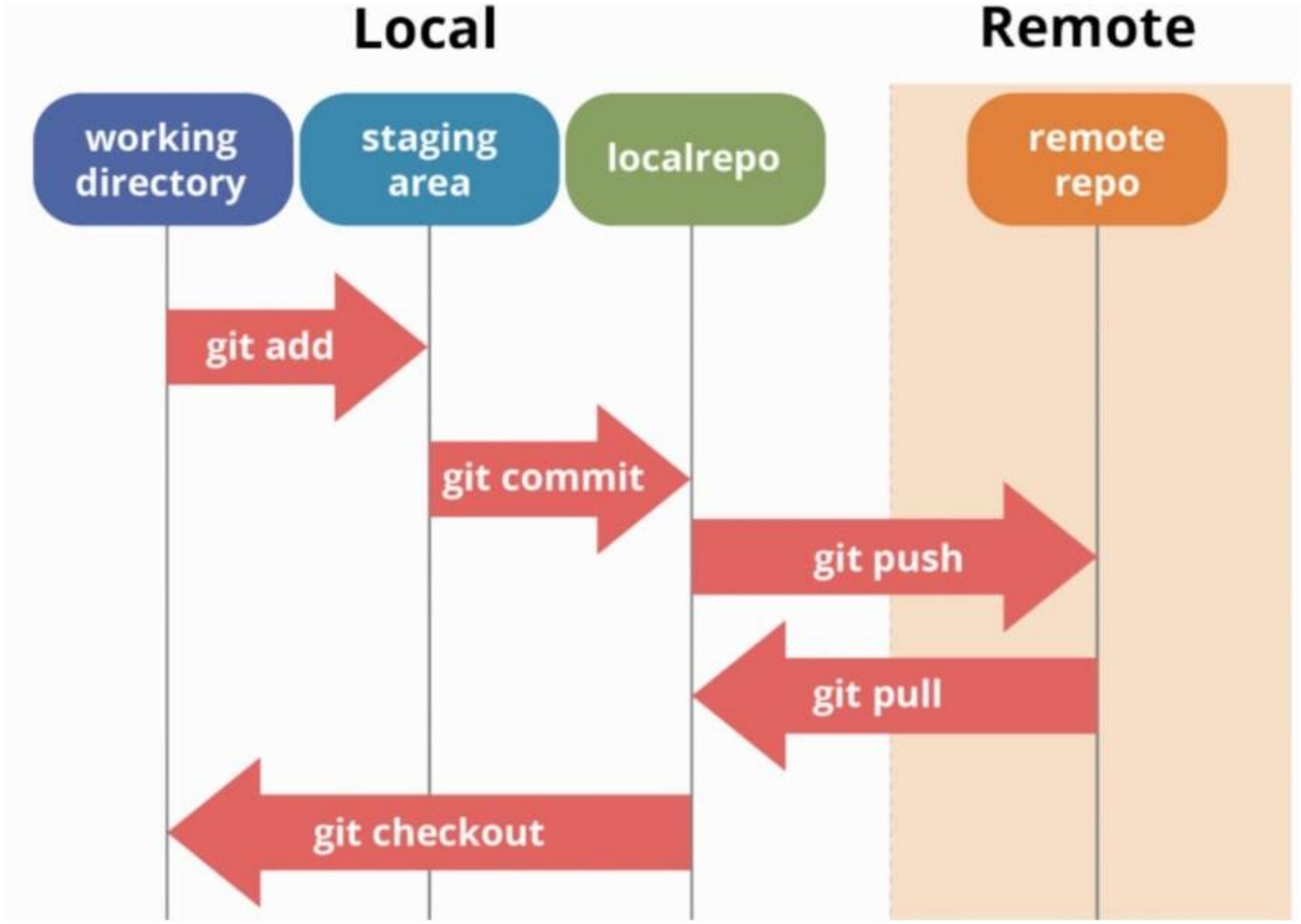
Centralized Source Code Management Systems



Distributed Version Control System



Git workflow



Git Commands

Git: configurations

```
$ git config --global user.name "FirstName LastName"
$ git config --global user.email "your-email@email-provider.com"
$ git config --list
```

Git: starting a repository

```
$ git init
$ git status
```

Git: staging files

```
$ git add <file-name>
$ git add <file-name> <another-file-name> <yet-another-file-name>
$ git add .
$ git add --all
$ git add -A
$ git rm --cached <file-name>
$ git reset <file-name>
```

Git: committing to a repository

```
$ git commit -m "Add three files"
```

Git Command Cont…

Going to a particular version

Git checkout <commitID> --*/filename

Git checkout master --*/filename

Correcting the Mistakes

1) Edited and saved

git resotre

2) Edited and added to staging area

git restore --staged

3) Edited, added to Staging area and edited again

git restore --worktree

4) Edited and committed

git restore –soft HEAD^

git restore –hard HEAD^

Git Command Cont….

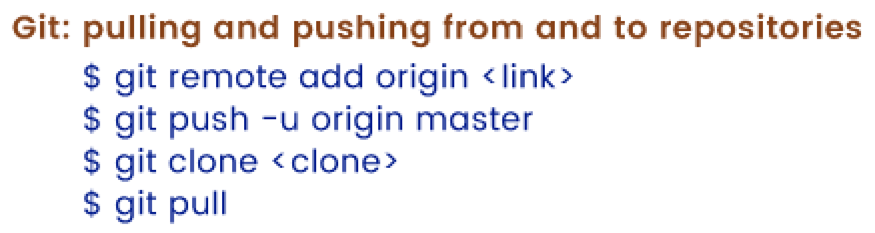
Git logging

```
git log –p –2 ( last two commit with difference )
git log –stat ( along with summary of changes )
git log ––pretty=oneline
git log –pretty=format:"%h –%an, %ar:%s"
git log –S <funcation_name>
```

Explore other log options on your own….(grep, since, until, author..)

Working with remote repo

Setting and working with Github



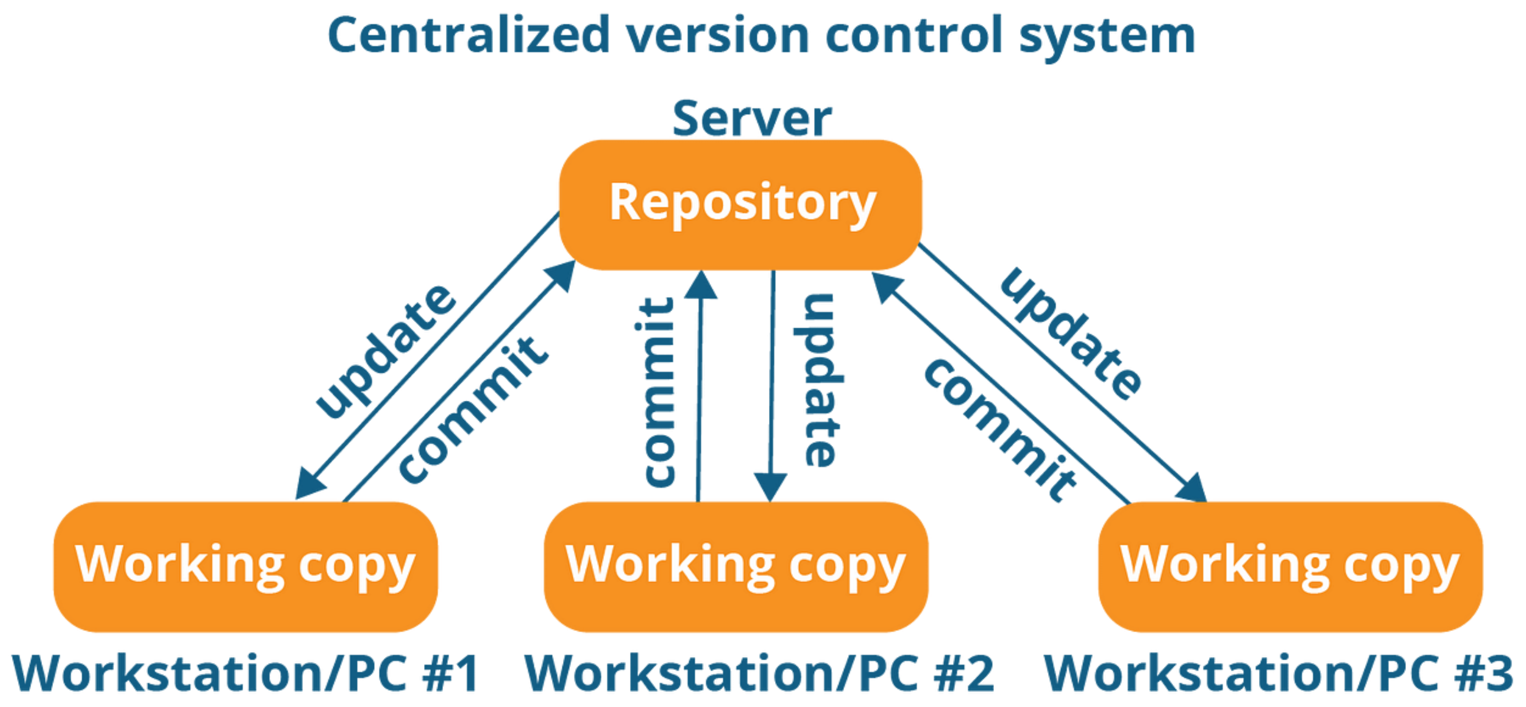
Git: pulling and pushing from and to repositories

```
$ git remote add origin <link>
$ git push -u origin master
$ git clone <clone>
$ git pull
```

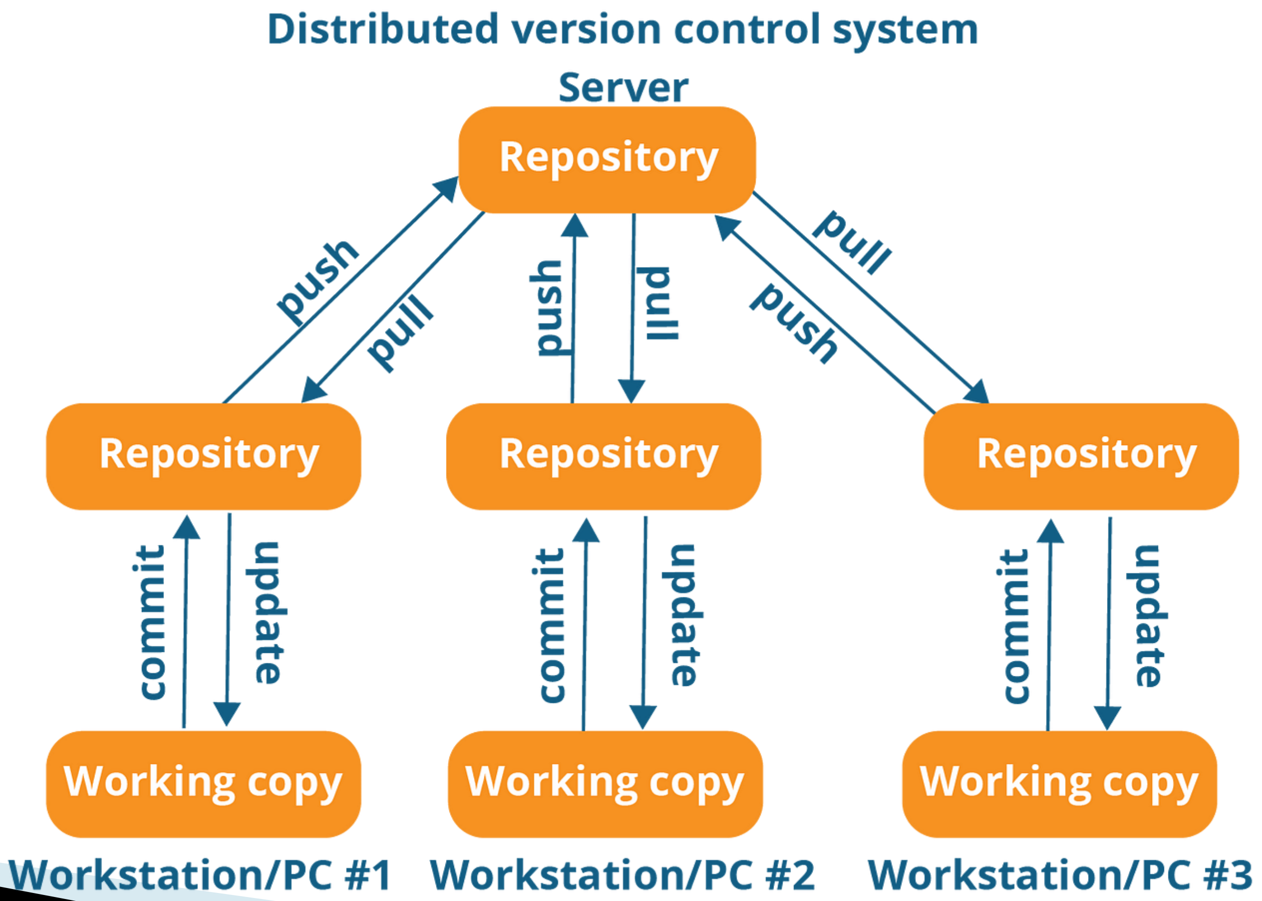
Quick topics…

- Version control tool Migration
  - With and without History
- Version Naming
  - Major Number
  - Minor Number
  - Bug Fixes
  - Build Number
- Hosted Git Servers to manage Corporate code
- Gerrit plug in for code review

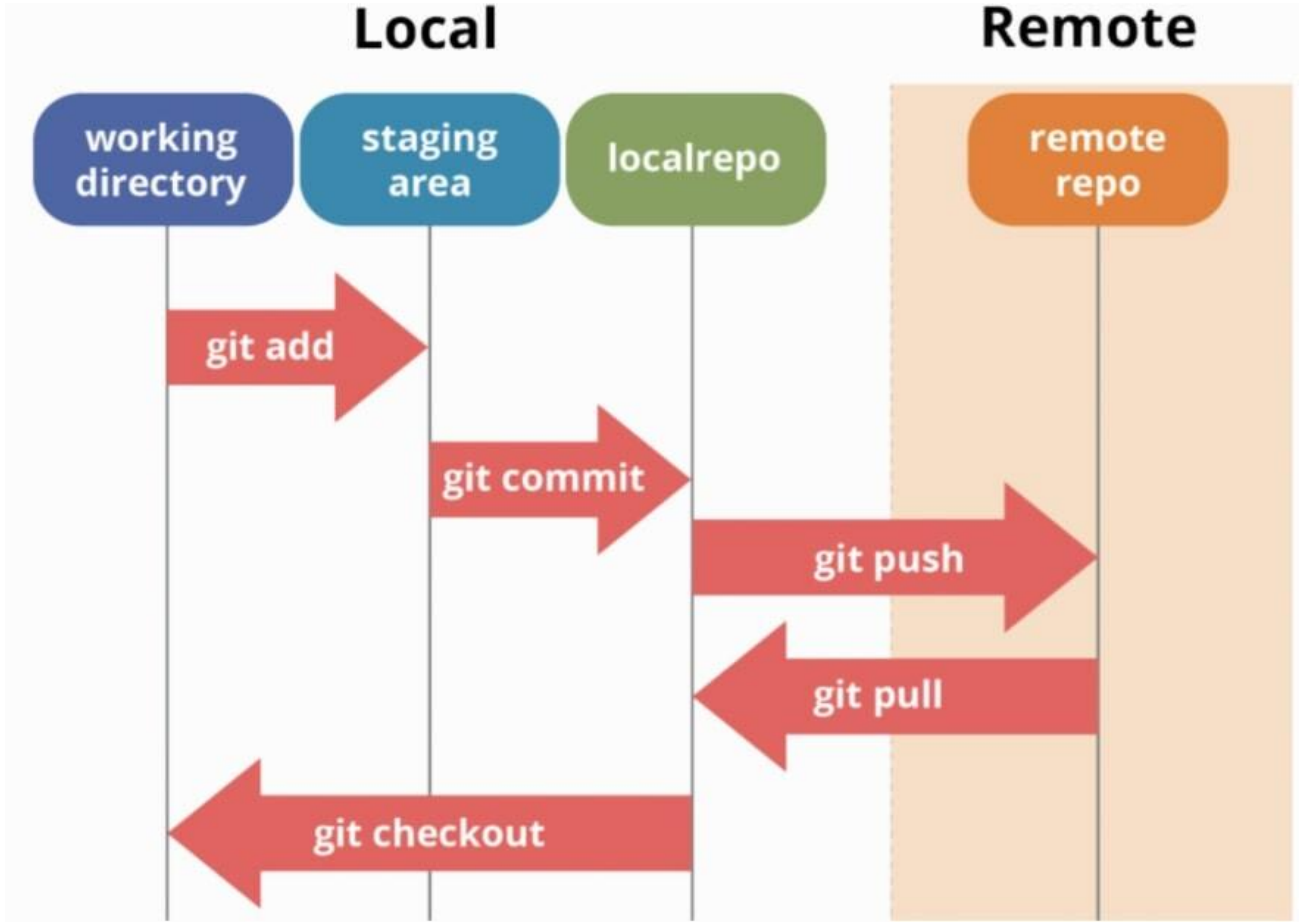
Centralized Source Code Management Systems



Distributed Version Control System



Git workflow



Git Commands

Git: configurations

```
$ git config --global user.name "FirstName LastName"
$ git config --global user.email "your-email@email-provider.com"
$ git config --list
```

Git: starting a repository

```
$ git init
$ git status
```

Git: staging files

```
$ git add <file-name>
$ git add <file-name> <another-file-name> <yet-another-file-name>
$ git add .
$ git add --all
$ git add -A
$ git rm --cached <file-name>
$ git reset <file-name>
```

Git: committing to a repository

```
$ git commit -m "Add three files"
```

Git Command Cont…

Going to a particular version

Git checkout <commitID> --*/filename

Git checkout master --*/filename

Correcting the Mistakes

1) Edited and saved

git resotre

2) Edited and added to staging area

git restore --staged

3) Edited, added to Staging area and edited again

git restore --worktree

4) Edited and committed

git restore –soft HEAD^

git restore –hard HEAD^

Git Command Cont….

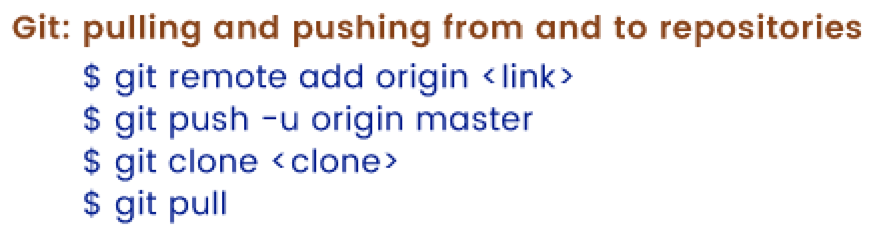
Git logging

```
git log –p -2 ( last two commit with difference )
git log –stat ( along with summary of changes )
git log --pretty=oneline
git log –pretty=format:"%h -%an, %ar:%s"
git log –S <funcation_name>
```

Explore other log options on your own….(grep, since, until, author..)

Working with remote repo

Setting and working with Github



Git: pulling and pushing from and to repositories

```
$ git remote add origin <link>
$ git push -u origin master
$ git clone <clone>
$ git pull
```

Quick topics…

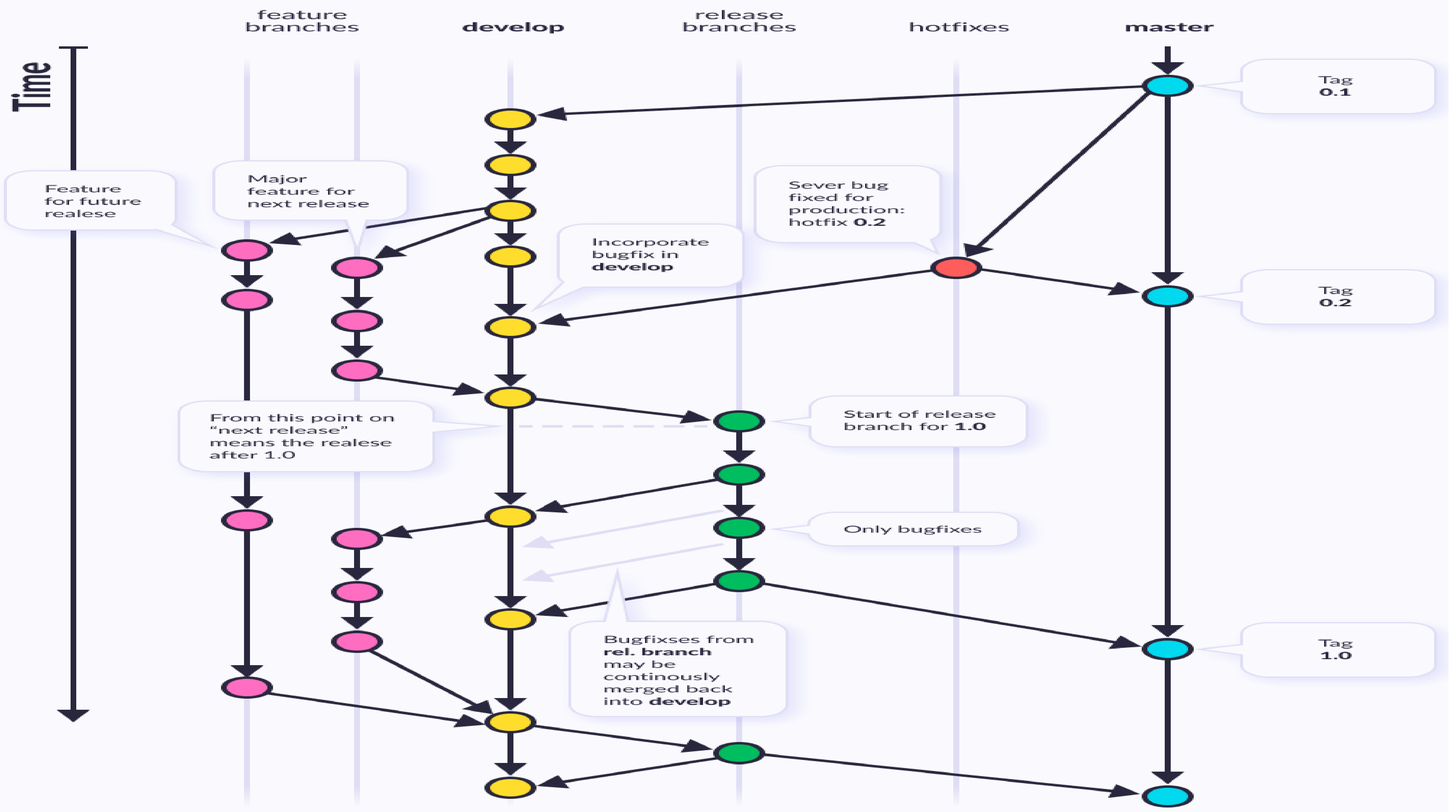
- Version control tool Migration
  - With and without History
- Version Naming
  - Major Number
  - Minor Number
  - Bug Fixes
  - Build Number
- Hosted Git Servers to manage Corporate code
- Gerrit plug in for code review

Branching

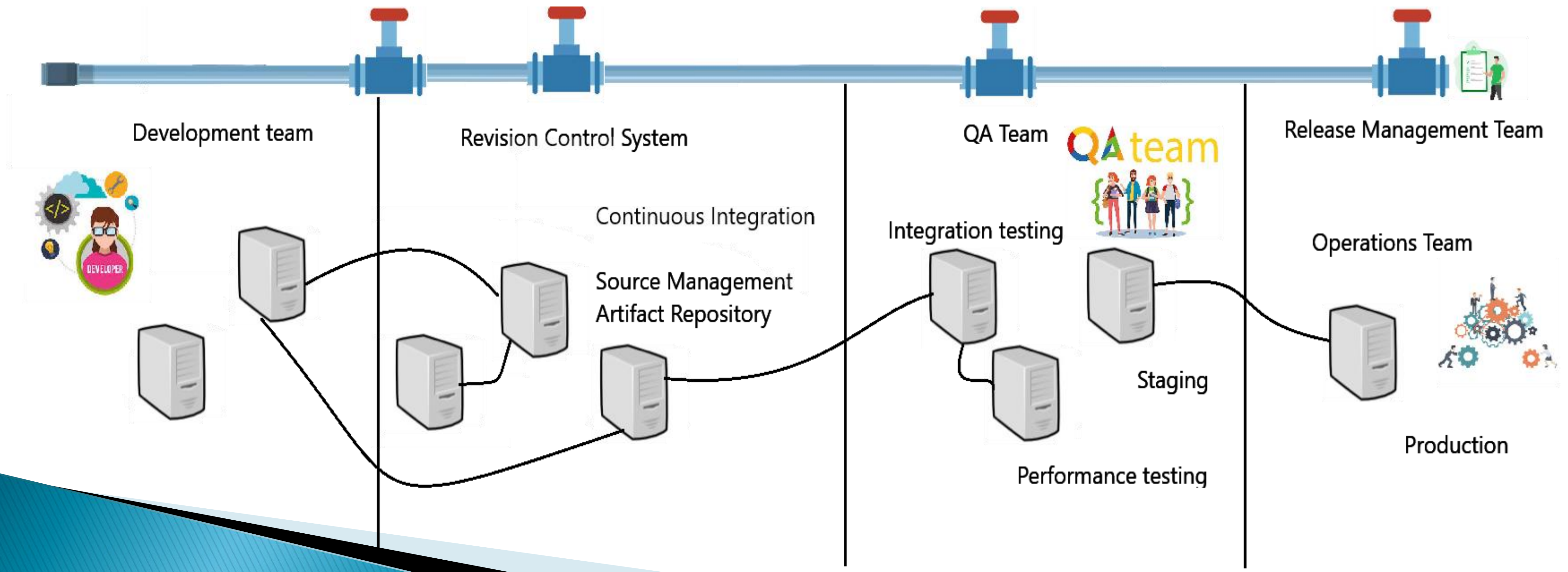
- Need for Branching

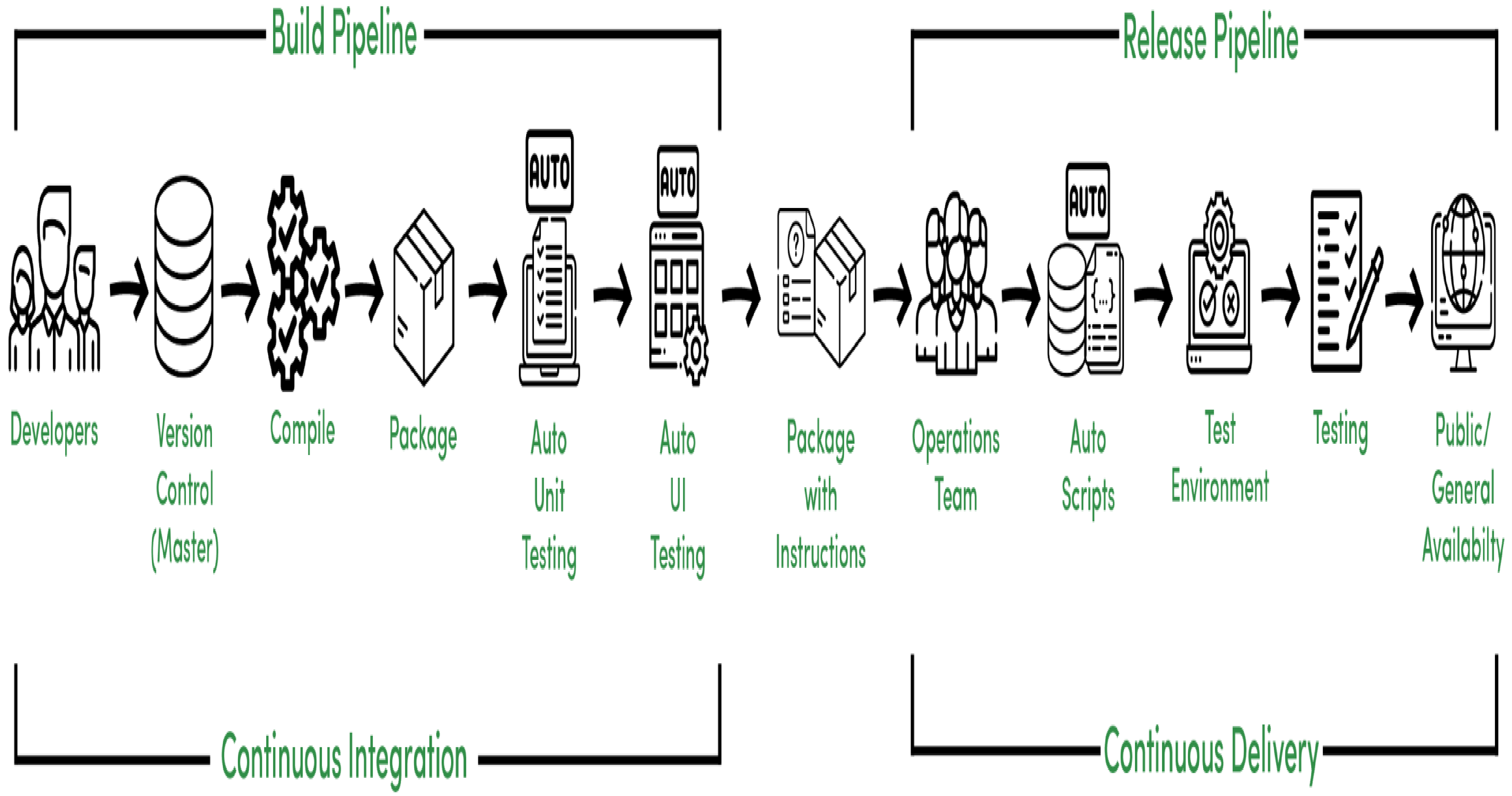
- Problem areas of Branching

- Branch Management
  - Porting of fixes
  - Feature Diversity


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future realese
Major feature for next release
Sever bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
From this point on "next release" means the realese after 1.0
Start of release branch for 1.0
Only bugfixes
Bugfixses from rel. branch may be continously merged back into develop
Tag 1.0

Continuous Delivery Pipeline




Build Pipeline
Release Pipeline
AUTO
AUTO
AUTO
Developers
Version Control (Master)
Compile
Package
Auto Unit Testing
Auto UI Testing
Package with Instructions
Operations Team
Auto Scripts
Test Environment
Testing
Public/ General Availabilty
Continuous Integration
Continuous Delivery

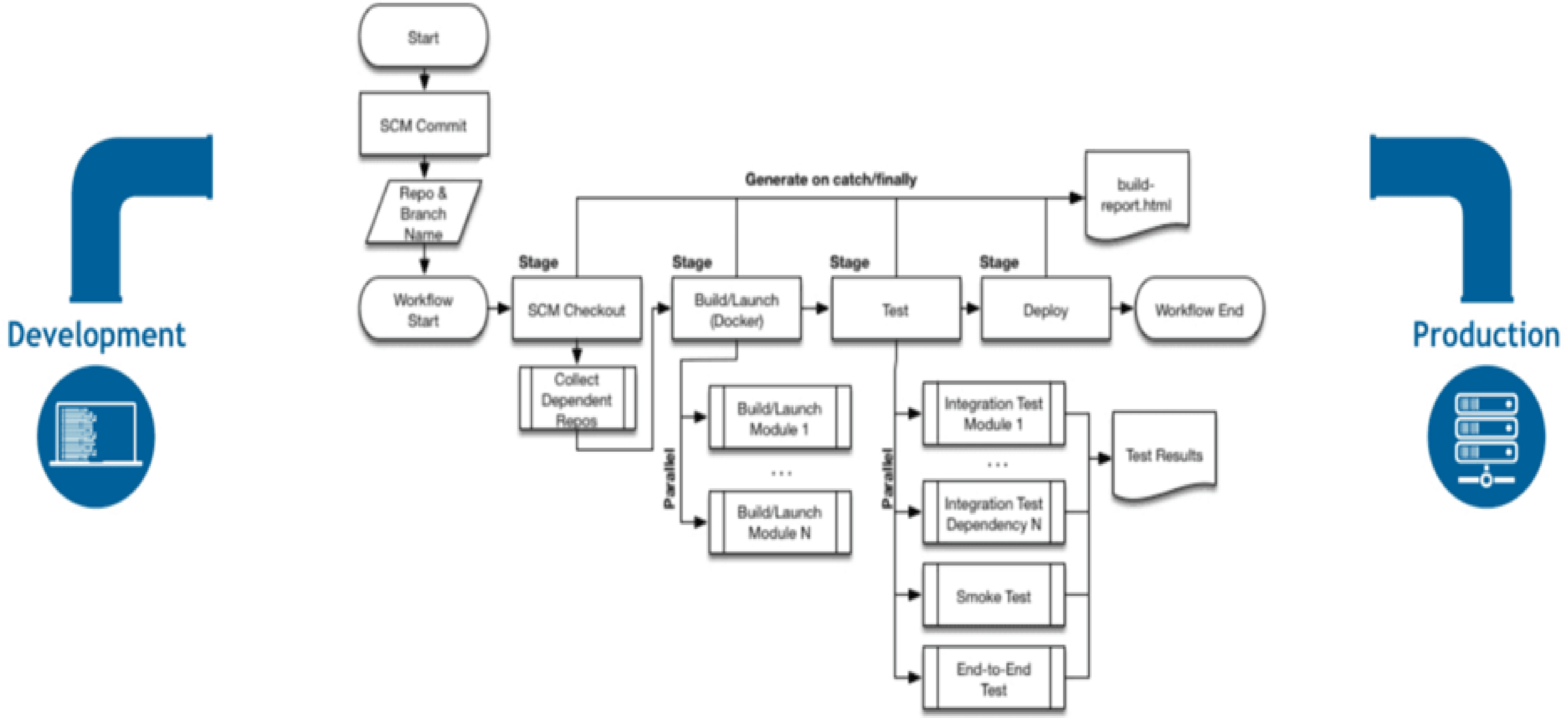
There is more to Build

- Build Systems
  - Maven
- Purpose of build
- Final output of build

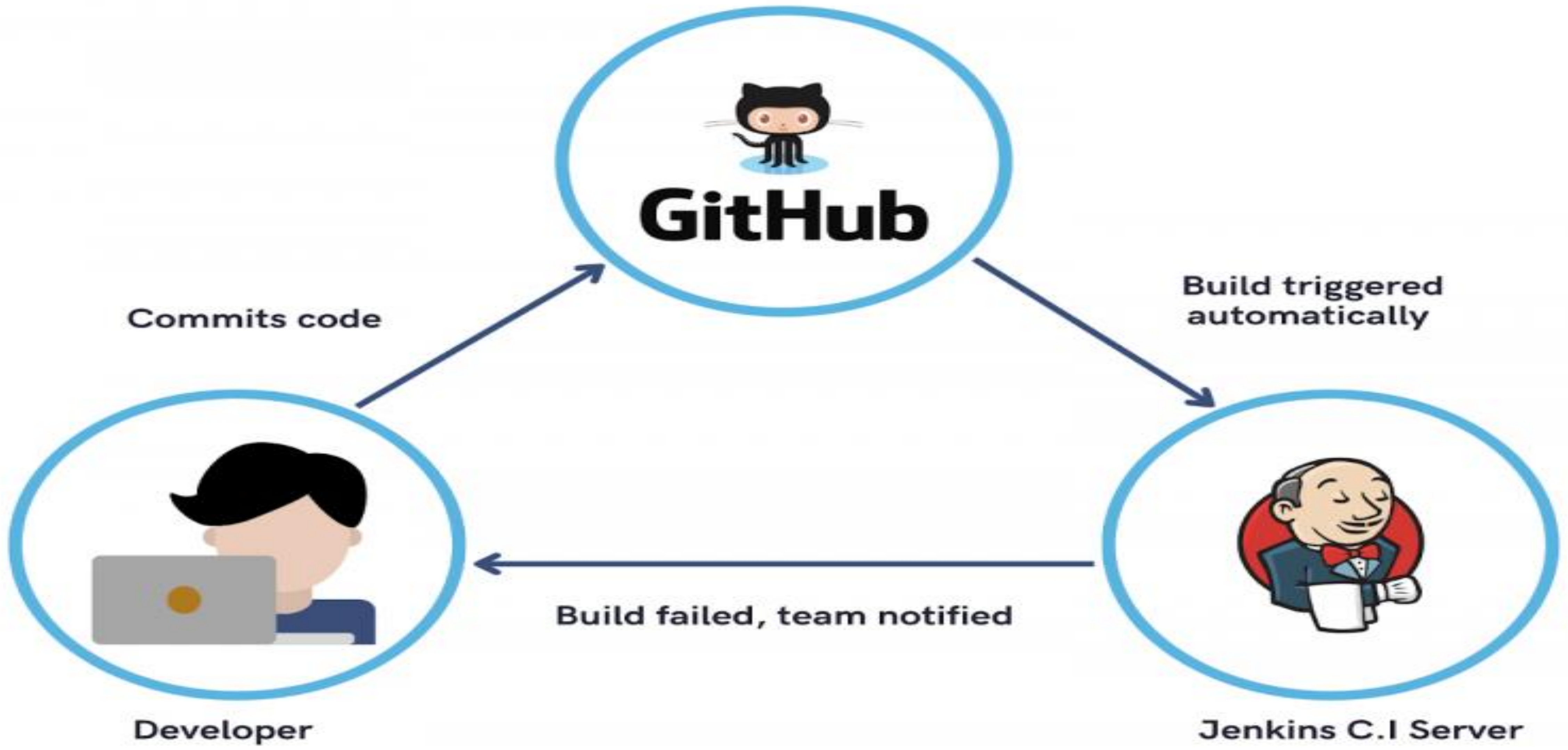
What is Jenkins ?

- Jenkins is a open source automation tool used to build and test software projects.
- Jenkins makes it easy for developers to integrate changes in the project.
- Jenkins helps CI with plug-in support.

Jenkins Workflow



Jenkins Continuous Integration



Learn to work with Git

- Remote Repo
- Branching

Create CI Pipeline with Jenkins

Automate Build pipeline with Jenkins

Basic test automation

Containerization of Java Project using Docker

Jenkins used with PYTHON

- Jenkins is widely used with Python for CI/CD, automating tasks.This is achieved either by running Python scripts within Jenkins build steps or by using Python client libraries to manage the Jenkins server itself.

- The primary way to use Python with Jenkins in a CI/CD pipeline is by executing Python scripts via shell or batch commands in a Jenkins job.

- The system where the Jenkins agent or controller runs must have Python and all necessary dependencies (e.g., pytest, pylint) installed and accessible in the system's PATH or a virtual environment.

- In the Jenkins job configuration's "Build Steps" section, you can add an "Execute shell" or "Execute Windows batch command" step and run your Python script using commands like python your_script.py or pip install -r requirements.txt.

- For complex, modern CI/CD workflows, a Jenkinsfile defines the pipeline stages as code, including steps to run Python commands.

Jenkins Build Triggers

jenkins build triggers are mechanisms that automatically initiate the execution of Jenkins jobs or pipelines based on specific events or conditions. They are essential for automating CI/CD workflows and reducing manual intervention.

The most common types of Jenkins build triggers include:

- **Manually Triggered:** "Build Now" button in the Jenkins UI.
- **Poll SCM (Source Code Management):** Jenkins periodically polls the SCM repository (e.g., Git, SVN) for changes at a specified interval using a cron-like syntax, If changes are detected, a build is triggered.
- **SCM Webhooks (e.g., GitHub hook trigger for GITScm polling):** This is a more efficient, real-time method than polling. When a change is pushed to the repository (e.g., GitHub) the SCM service sends a webhook (an HTTP POST request) to Jenkins, which then immediately triggers a build.
- **Build periodically:** This trigger runs a build at a specific time or fixed interval, regardless of whether changes have been made to the source code. It also uses a cron expression for scheduling.

- **Build after other projects are built:** This is used for creating multi-stage or chained pipelines. When one specified downstream project successfully completes its build, it automatically triggers an upstream project's build.

- **Trigger builds remotely (e.g., from scripts or external systems):** This allows external scripts, applications, or remote systems to trigger a Jenkins build by sending a specially crafted URL request with an authentication token.

- **Parameterized Trigger Plugin:** This plugin allows one build to trigger another (downstream) build while passing parameters to it, enabling more complex "function call"-like interactions between jobs.

- **AMQP or other message sources:** With the help of specific plugins, builds can be triggered by messages received from message brokers like AMQP or RabbitMQ, allowing integration with message-based systems

Configure SCM polling and GitHub webhook

- Configuring **SCM polling** and a **GitHub webhook** in Jenkins allows you to trigger builds automatically based on changes in your GitHub repository.

- **SCM Polling** works by periodically checking your GitHub repository for changes at a scheduled interval (e.g., every 15 minutes).

- A **GitHub Webhook** provides real-time triggers; GitHub sends an immediate notification (a POST request) to Jenkins whenever a specific event (like a code push) occurs.

Jenkins Declarative Pipeline

- A **Jenkins Declarative Pipeline** is a structured, simplified, and human-readable way to define continuous integration and continuous delivery (CI/CD) pipelines as code.
- Unlike the more flexible, Groovy-based Scripted Pipeline, Declarative Pipeline uses a predefined structure and specific directives, making it easier for new users to adopt and maintain.

All valid Declarative Pipelines must be enclosed within a pipeline block and contain four main required sections:

- **pipeline**: The top-level block that defines the entire pipeline.
- **agent**: Specifies where the pipeline, or a specific stage, will execute (e.g., agent any runs on any available agent).
- **stages**: Contains one or more stage directives.
- **stage**: Defines a section of the pipeline, such as "Build", "Test", or "Deploy", for visualization in the Jenkins UI.
- **steps**: Contains the individual tasks or steps to be executed within a stage.

Key Directives and Features

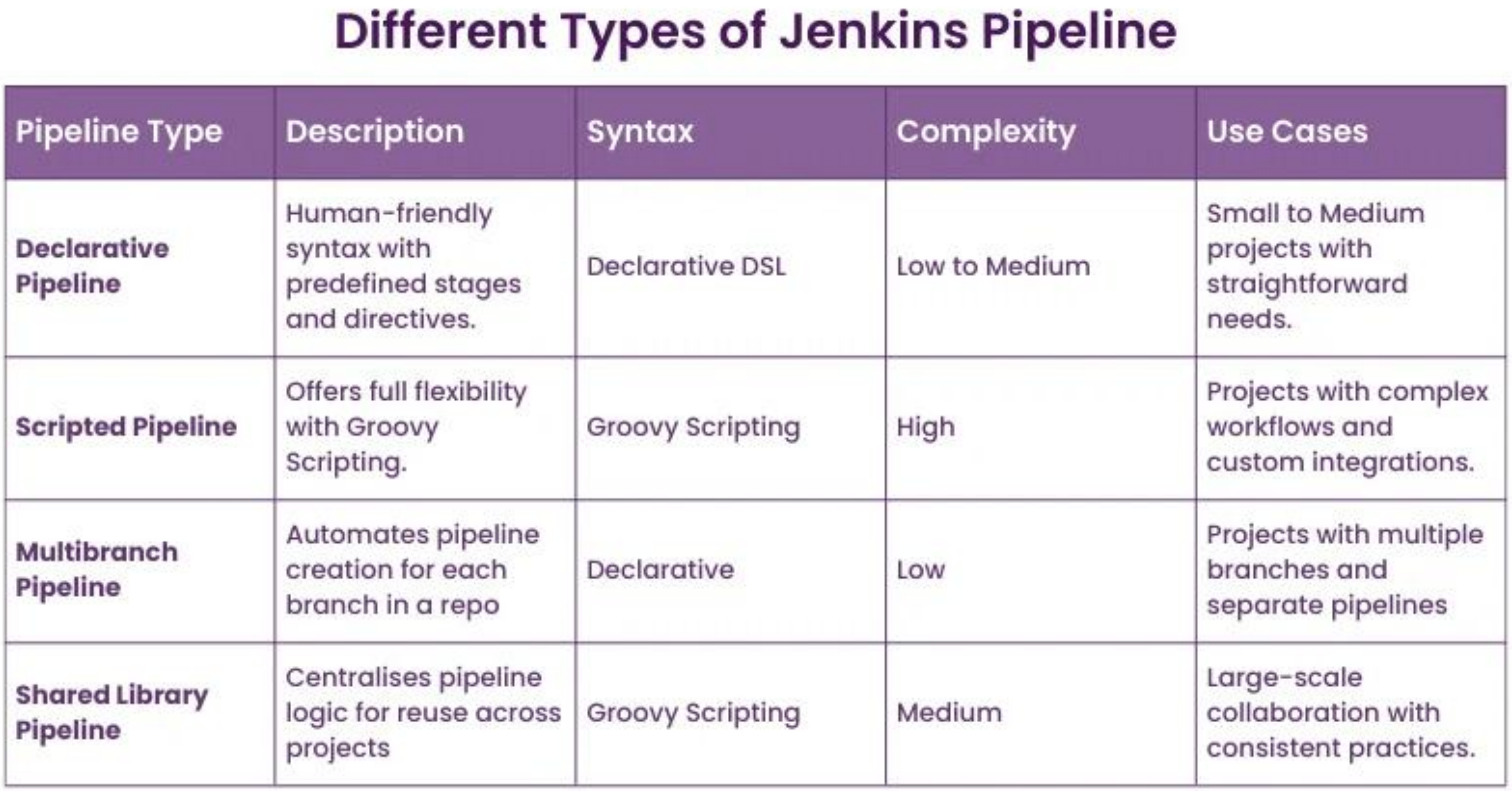
Declarative Pipelines offer several directives to manage workflow, environments, and conditions:

- **environment**
- **input**
- **options**
- **parameters**
- **post**
- **triggers**
- **when**
- **parallel**
- **tools**

Three types of pipelines in Jenkins

- Jenkins Pipelines offer a versatile and powerful solution for automating software delivery processes.

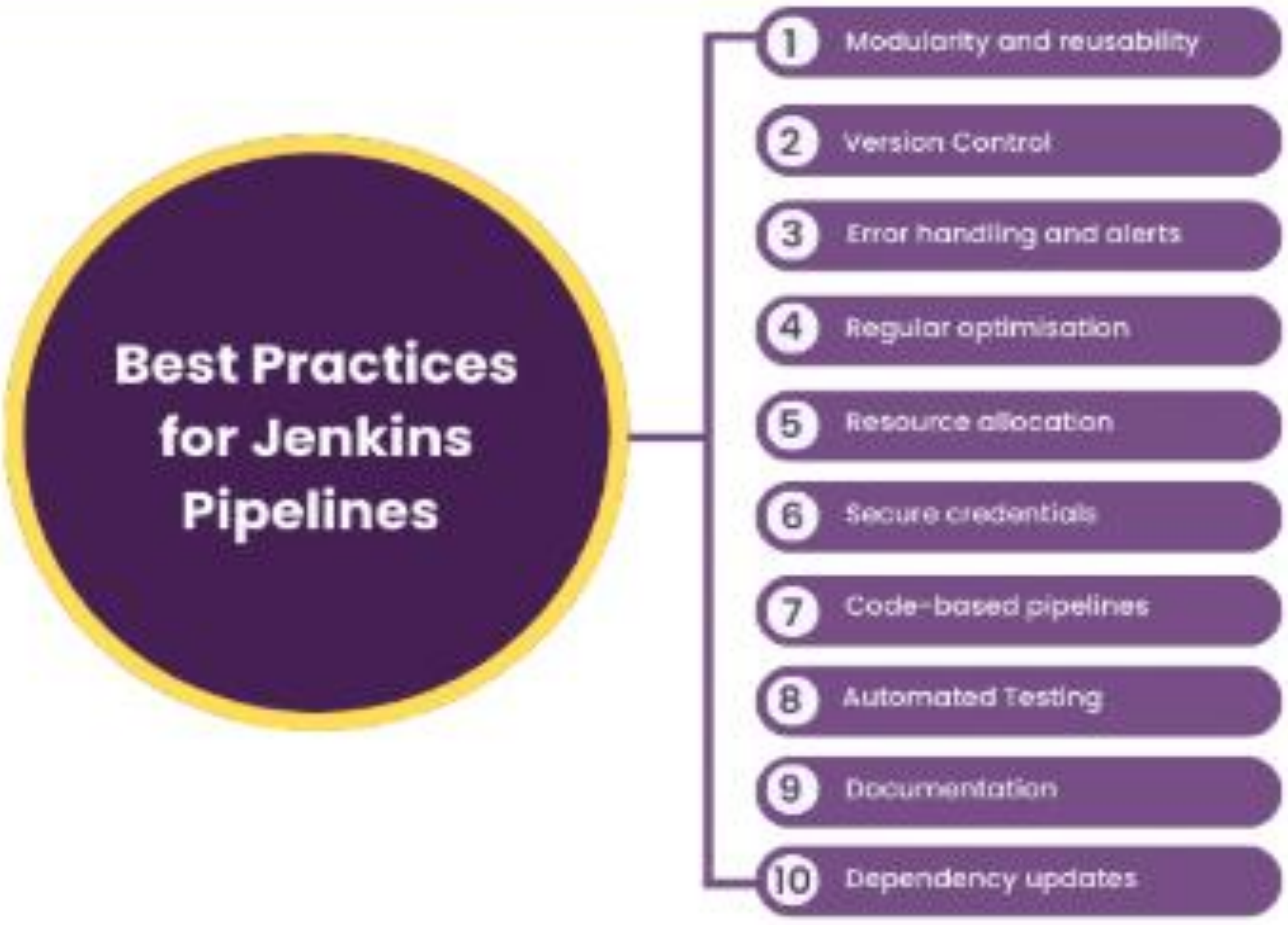
- By understanding the different Types of Pipeline in Jenkins Declarative, Scripted, Multibranch, and Shared Library – developers can tailor their workflows to project requirements.



Different Types of Jenkins Pipeline

Pipeline Type	Description	Syntax	Complexity	Use Cases
Declarative Pipeline	Human-friendly syntax with predefined stages and directives.	Declarative DSL	Low to Medium	Small to Medium projects with straightforward needs.
Scripted Pipeline	Offers full flexibility with Groovy Scripting.	Groovy Scripting	High	Projects with complex workflows and custom integrations.
Multibranch Pipeline	Automates pipeline creation for each branch in a repo	Declarative	Low	Projects with multiple branches and separate pipelines
Shared Library Pipeline	Centralises pipeline logic for reuse across projects	Groovy Scripting	Medium	Large-scale collaboration with consistent practices.

https://www.theknowledgeacademy.com/blog/types-of-jenkins-pipeline/



Best Practices for Jenkins Pipelines
1 Modularity and reusability
2 Version Control
3 Error handling and alerts
4 Regular optimisation
5 Resource allocation
6 Secure credentials
7 Code-based pipelines
8 Automated Testing
9 Documentation
10 Dependency updates

Jenkins Pipeline Visualization and Logs

- Jenkins provides built-in and plugin-based tools for visualizing pipeline flow and accessing logs to monitor the software delivery process.

- Key features include graphical stage views, real-time status indicators, and integrated console outputs for debugging.

Pipeline Visualization

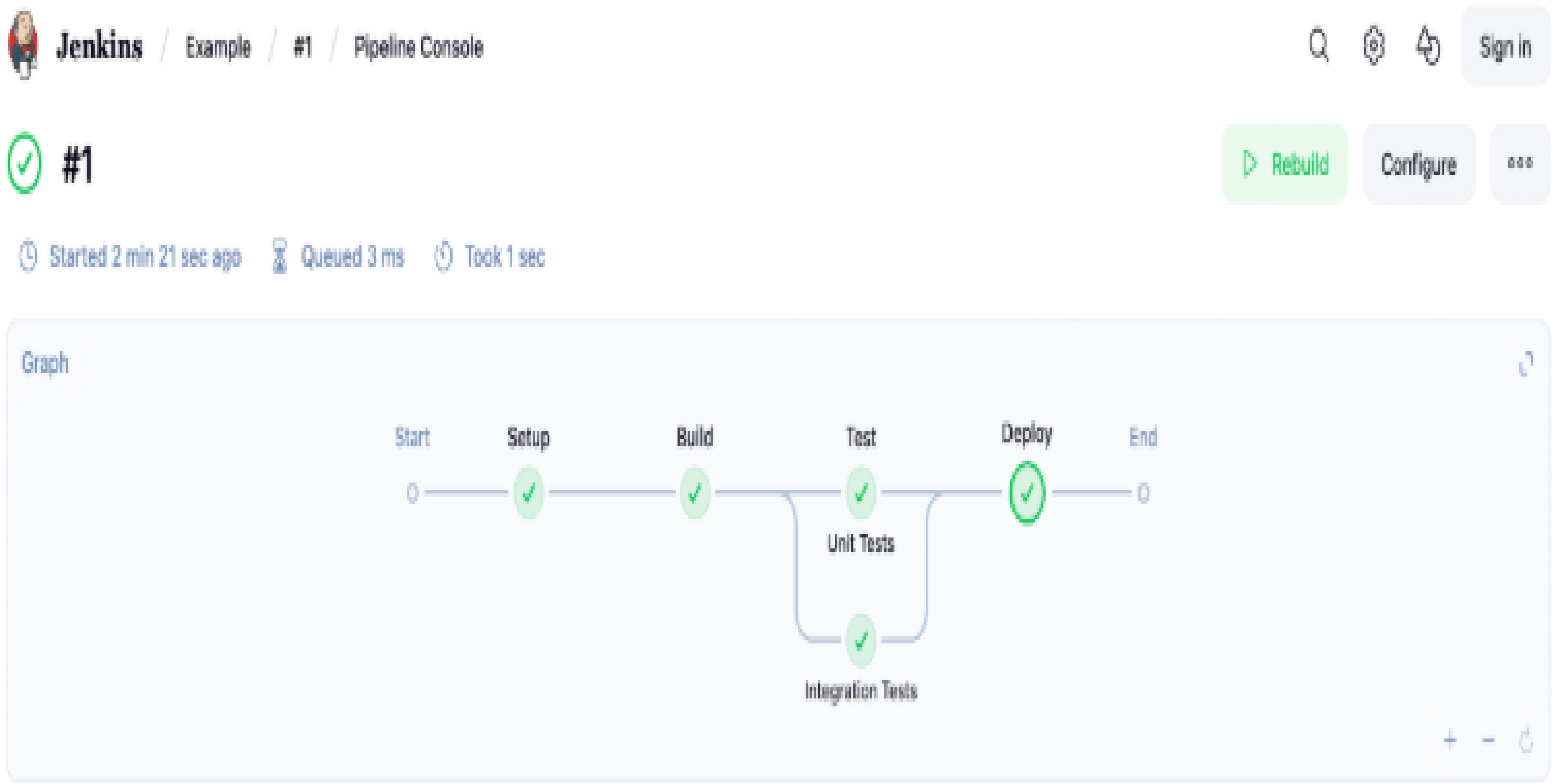
- Jenkins' primary method for visualizing pipelines is through plugins that display the build process as a series of interconnected stages and steps, with color-coding to indicate status (success, failure, in-progress).

Pipeline Graph View

- This is the current, recommended core plugin for visualizing pipelines in the modern Jenkins UI. It offers an interactive, nested graph view that combines the pipeline structure and logs into a single, streamlined interface. Users can pan, zoom, and access step details and logs without leaving the main view. It replaces older visualization methods like the **Blue Ocean** plugin (which is in maintenance mode) and other specialized plugins.

Stage View

- This feature, also provided by core plugins, displays an extended view of build history, showing each stage as a column and each build as a row. Hovering over a stage cell allows access to specific logs for that part of the pipeline.


Jenkins / Example / #1 / Pipeline Console

Sign in

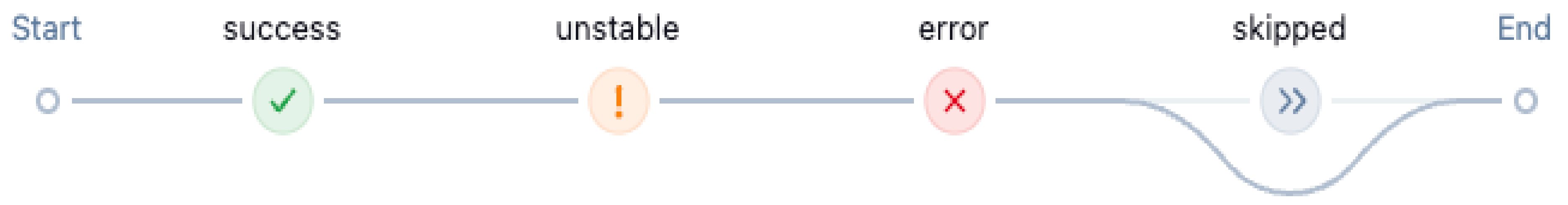
#1

Rebuild Configure

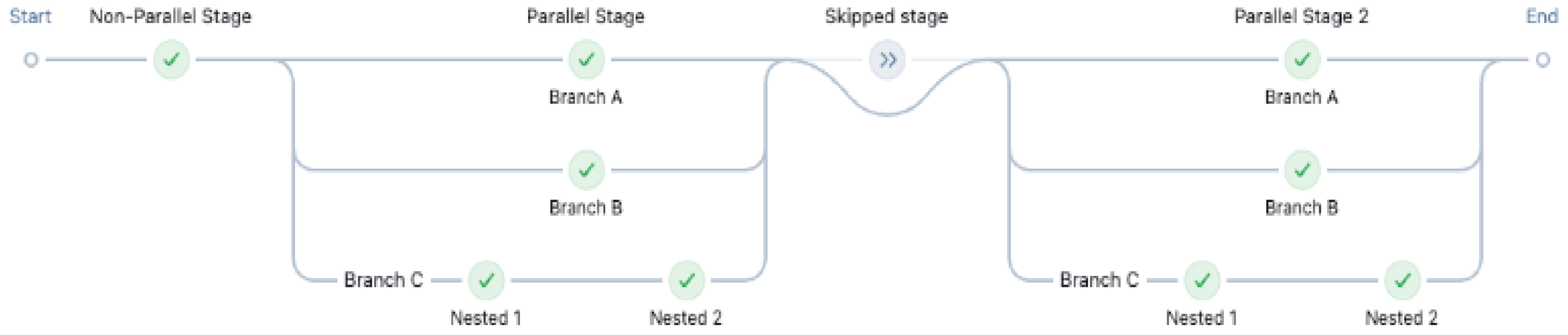
Started 2 min 21 sec ago Queued 3 ms Took 1 sec

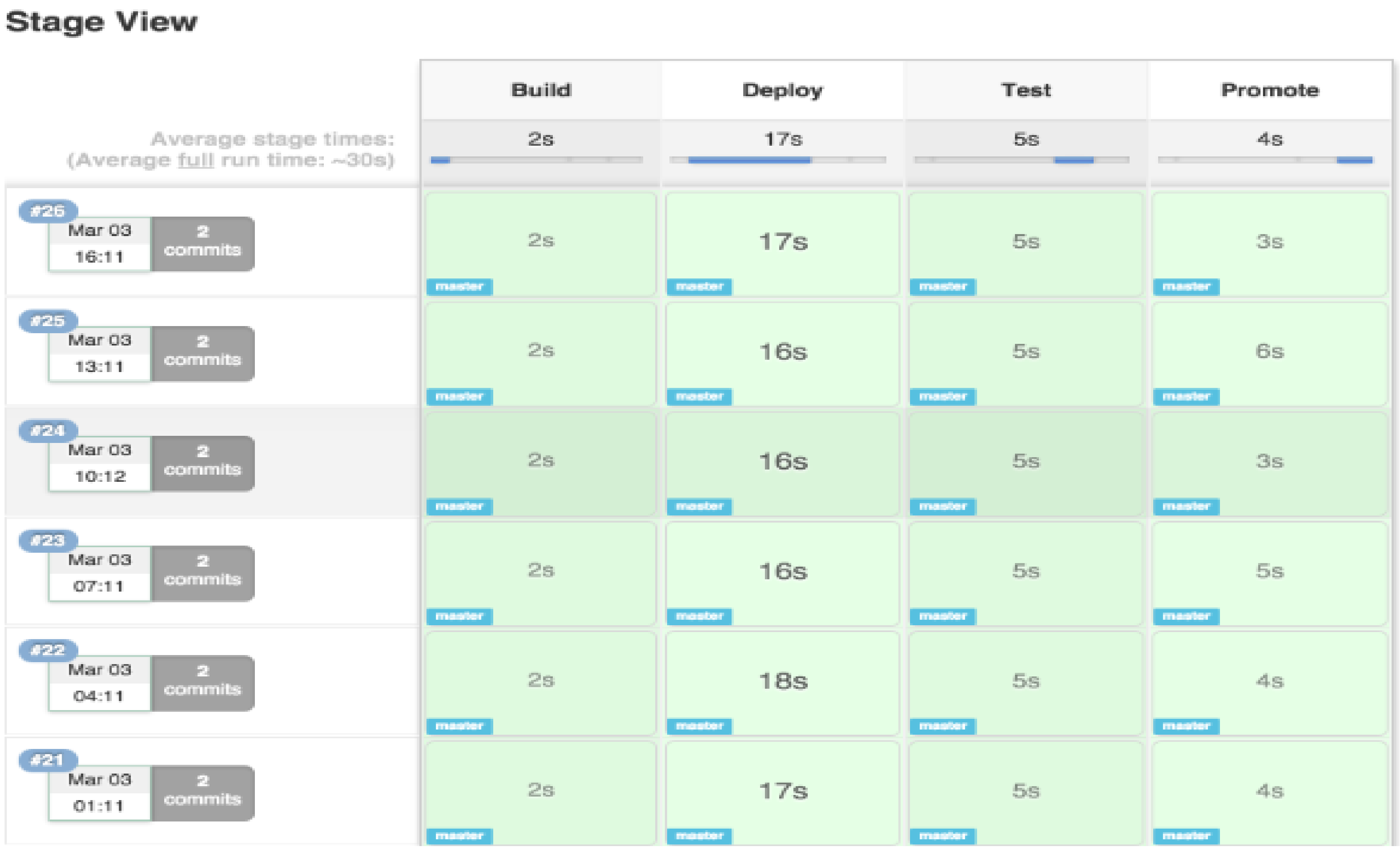
Graph

Basic pipeline:



Semi-complex pipeline:





Stage View

Average stage times: (Average full run time: ~30s)	Build	Deploy	Test	Promote
	2s	17s	5s	4s
#26 Mar 03 16:11 2 commits	2s master	17s master	5s master	3s master
#25 Mar 03 13:11 2 commits	2s master	16s master	5s master	6s master
#24 Mar 03 10:12 2 commits	2s master	16s master	5s master	3s master
#23 Mar 03 07:11 2 commits	2s master	16s master	5s master	5s master
#22 Mar 03 04:11 2 commits	2s master	18s master	5s master	4s master
#21 Mar 03 01:11 2 commits	2s master	17s master	5s master	4s master

Log Management and Access

- Logs are crucial for monitoring execution and debugging failures. Jenkins offers several ways to view and manage them:

- **Console Output** The classic Jenkins UI provides a raw "Console Output" for every build, which shows all log messages generated during the pipeline run in real-time. This is the most detailed view and is essential for deep troubleshooting.

- **Interactive Logs in Pipeline Graph View** The revamped Pipeline Graph View integrates logs directly within the visual interface. Clicking on a specific stage or step displays its associated logs, making it easier to pinpoint the source of an error.

- **System Logs** For server-level issues, administrators can access system logs via the **Manage Jenkins** > **System Log** page. This allows the creation of custom log recorders to filter and group relevant logs for specific components.

- **External Integrations** For advanced log analysis and long-term storage, Jenkins can integrate with external tools like the ELK stack (Elasticsearch, Logstash, Kibana), Splunk, or Datadog using relevant plugins. This enables advanced search, filtering, and visualization of logs outside of the core Jenkins interface.

Few Project Ideas....

- Build a Website to Manage
  - 21 days fitness event of the college
  - Sales data of Hyper Market
  - Daily Diet
  - Monthly Personal Expense
  - Savings and asset allocation

Iteration 1 : Build a website to manage the data

Iteration 2 : Build 2 Statistics / Reports around the data being managed.