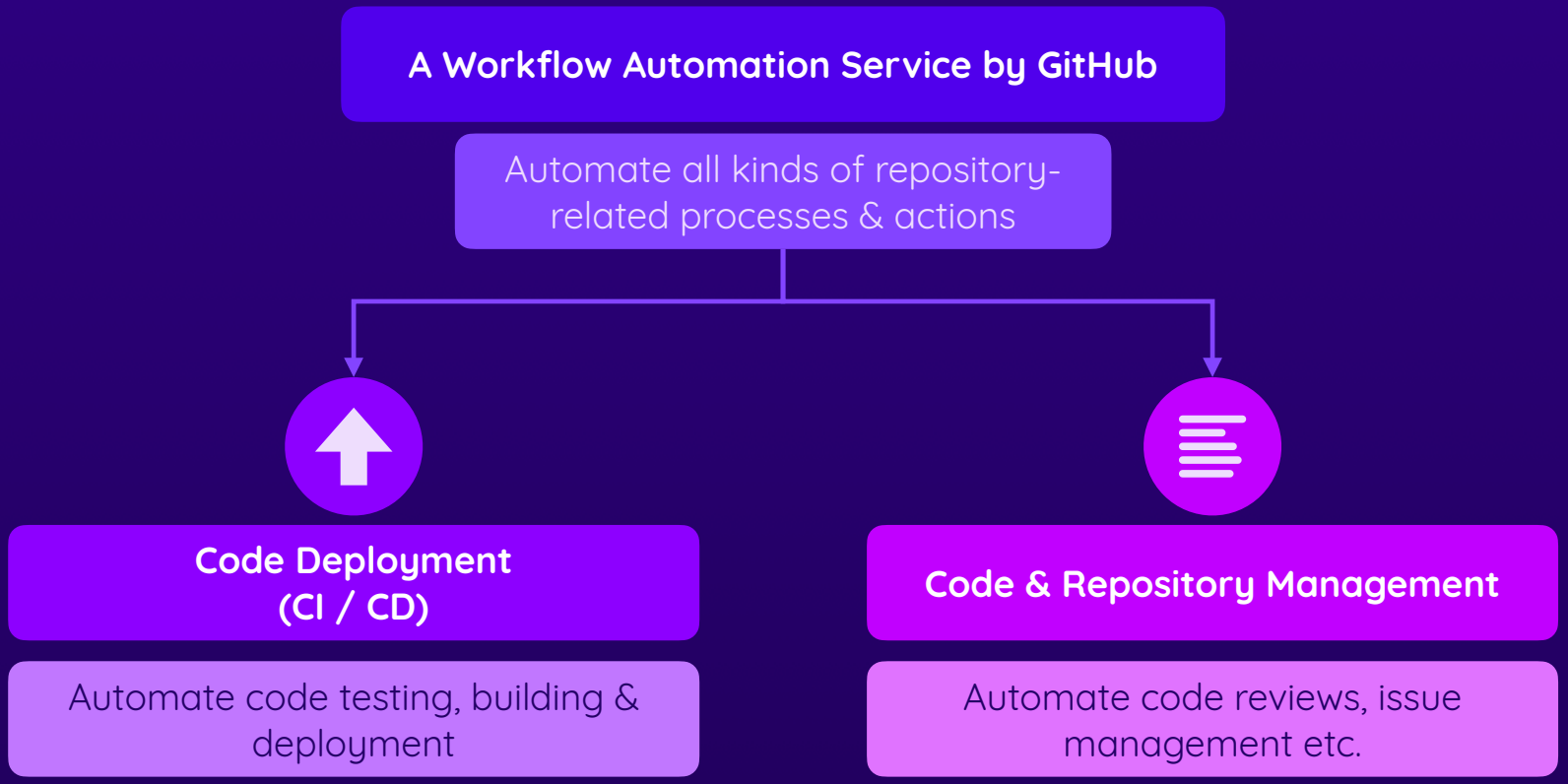
What Is “GitHub Actions”?

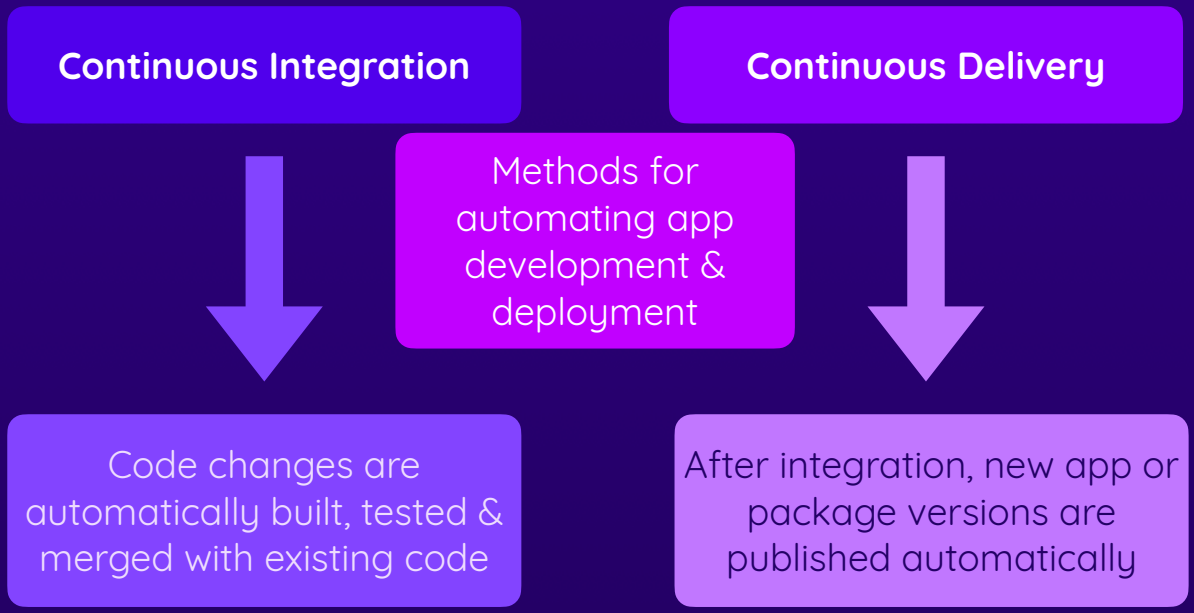


ACADEMIND

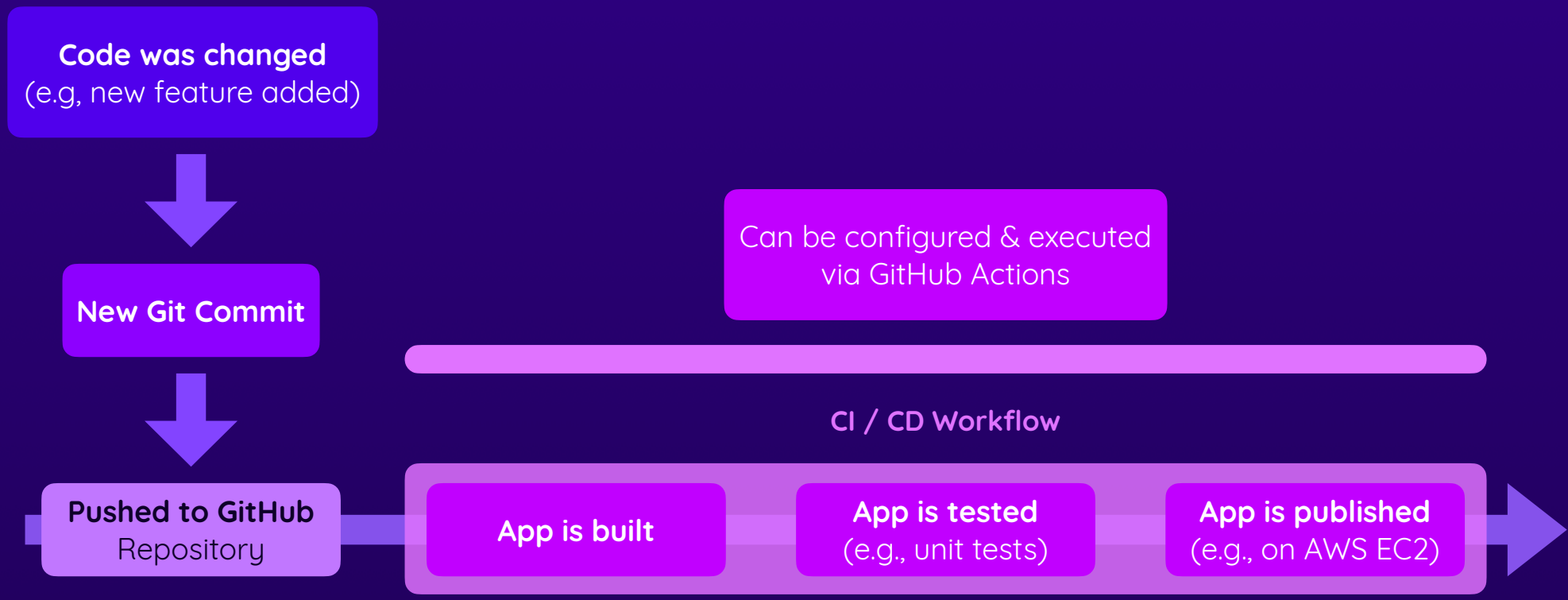
What is GitHub?

And what are “repositories”?

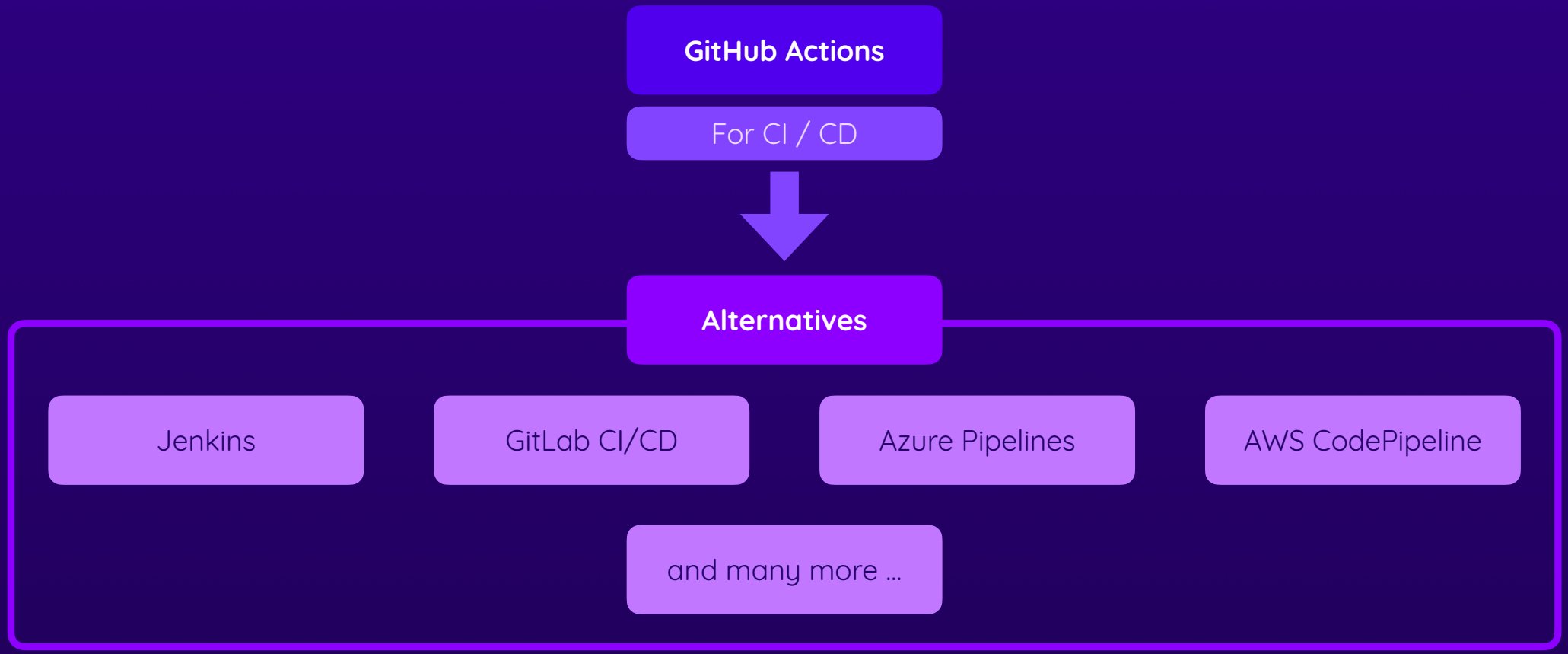
What's CI / CD?


Continuous Integration
Continuous Delivery
Methods for automating app development & deployment
Code changes are automatically built, tested & merged with existing code
After integration, new app or package versions are published automatically

A Typical CI / CD Workflow



GitHub Actions Alternatives



ACADEMIND

What is Git?

ACADEMIND

What Is GitHub?

A cloud Git repository & services provider

Store & manage Git repositories

ACADEMIND

What Is Git?

A (free) version control system

A tool for managing source code changes

Save code snapshots **(“commits”)**

Work with alternative code versions **(“branches”)**

Move between branches & commits

With Git, you can easily roll back to older code snapshots or develop new features without breaking production code.

What Is GitHub?

A cloud Git repository & services provider

Store & manage Git repositories

Cloud Git repository storage ("push" & "pull")

Backup, work across machines & work in teams

Public & private, team management & more

Code management & collaborative development

Via "Issues", "Projects", "Pull Requests" & more

Automation & CI / CD

Via **GitHub Actions**, GitHub Apps & more

About This Course

Learn GitHub Actions From The Ground Up

Video-based Explanations

- Watch the videos—at your pace
- Recommendation: Watch all videos in the provided order
- Repeat videos as needed

Practice & Experiment

- Pause videos and practice on your own
- Build up on course examples & feel free to experiment
- Build your own demo projects & workflow examples

Learn & Grow Together

- Help each other in the course Q&A section
- Dive into our (free!) community

ACADEMIND

Git & GitHub Crash Course

The Very Basics

- Working with Git: Setup & Key Commands
- Working with GitHub: Creating & Using Repositories
- Using Git with GitHub

ACADEMIND

What Is Git?

A (free) version control system

A tool for managing source code changes

Save code snapshots **(“commits”)**

Work with alternative code versions **(“branches”)**

Move between branches & commits

With Git, you can easily roll back to older code snapshots or develop new features without breaking production code.

Git Repositories

Git features can be used in projects with Git repositories

A repository is a folder used by Git to track all changes of a given project → Git commands require a repository in a project

Create Git repositories via `git init` → Only required once per folder / project

Some projects initialize Git for you → e.g., React projects

Working with Commits (Code Snapshots)

Create Commits

`git add <file(s)>`

Stage changes for next commit

+

`git commit`

Create a commit that includes all staged changes

Move between Commits

`git checkout <id>`

Temporarily move to another commit

Working with Commits (Code Snapshots)

Create Commits

`git add <file(s)>`

Stage changes for next commit

+

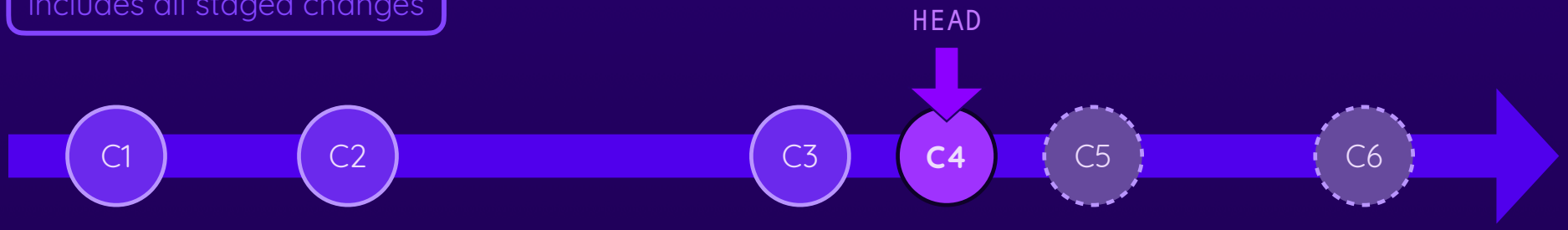
`git commit`

Create a commit that includes all staged changes

Move between Commits

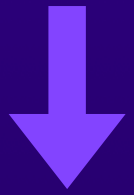
`git checkout <id>`

Temporarily move to another commit



Understanding Staging

Staging controls which changes are part of a commit



With staging, you can make sure that not all code changes made are added to a snapshot

If all changes should be included, you can use `git add .` to include all files in a Git repository

Working with Commits (Code Snapshots)

Create Commits

`git add <file(s)>`

Stage changes for next commit

+

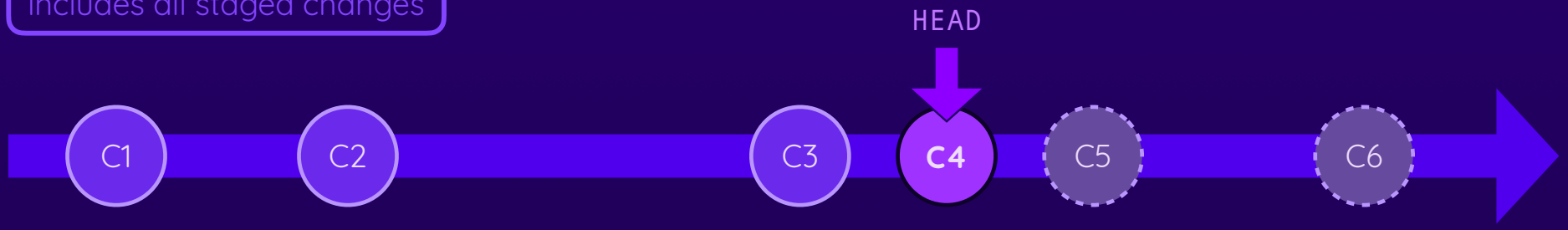
`git commit`

Create a commit that includes all staged changes

Move between Commits

`git checkout <id>`

Temporarily move to another commit



Working with Commits (Code Snapshots)

Create Commits

`git add <file(s)>`

Stage changes for next commit

+

`git commit`

Create a commit that includes all staged changes

Move between Commits

`git checkout <id>`

Temporarily move to another commit

Undo Commits

`git revert <id>`

Revert changes of commit by creating a new commit

Working with Commits (Code Snapshots)

Create Commits

`git add <file(s)>`

Stage changes for next commit

+

`git commit`

Create a commit that includes all staged changes

Move between Commits

`git checkout <id>`

Temporarily move to another commit

Undo Commits

`git revert <id>`

Revert changes of commit by creating a new commit

Working with Commits (Code Snapshots)

Create Commits

`git add <file(s)>`

Stage changes for next commit

+

`git commit`

Create a commit that includes all staged changes

Move between Commits

`git checkout <id>`

Temporarily move to another commit

Undo Commits

`git revert <id>`

Revert changes of commit by creating a new commit

`git reset --hard <id>`

Undo changes by deleting all commits since <id>

Working with Commits (Code Snapshots)

Create Commits

`git add <file(s)>`

Stage changes for next commit

+

`git commit`

Create a commit that includes all staged changes

Move between Commits

`git checkout <id>`

Temporarily move to another commit

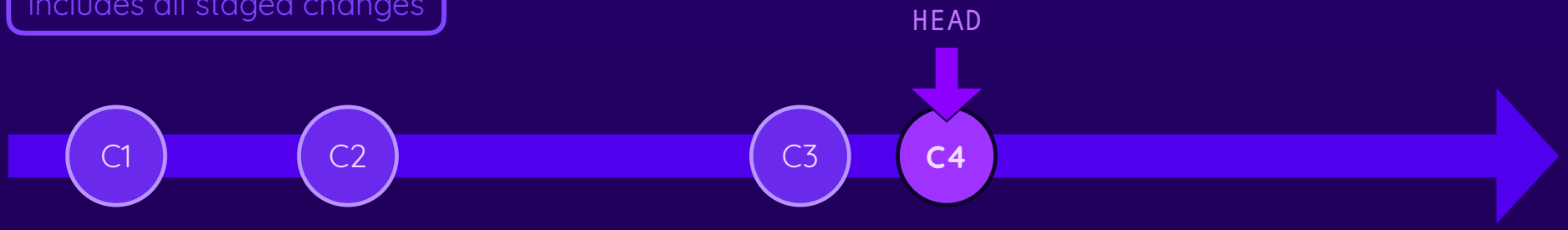
Undo Commits

`git revert <id>`

Revert changes of commit by creating a new commit

`git reset --hard <id>`

Undo changes by deleting all commits since <id>

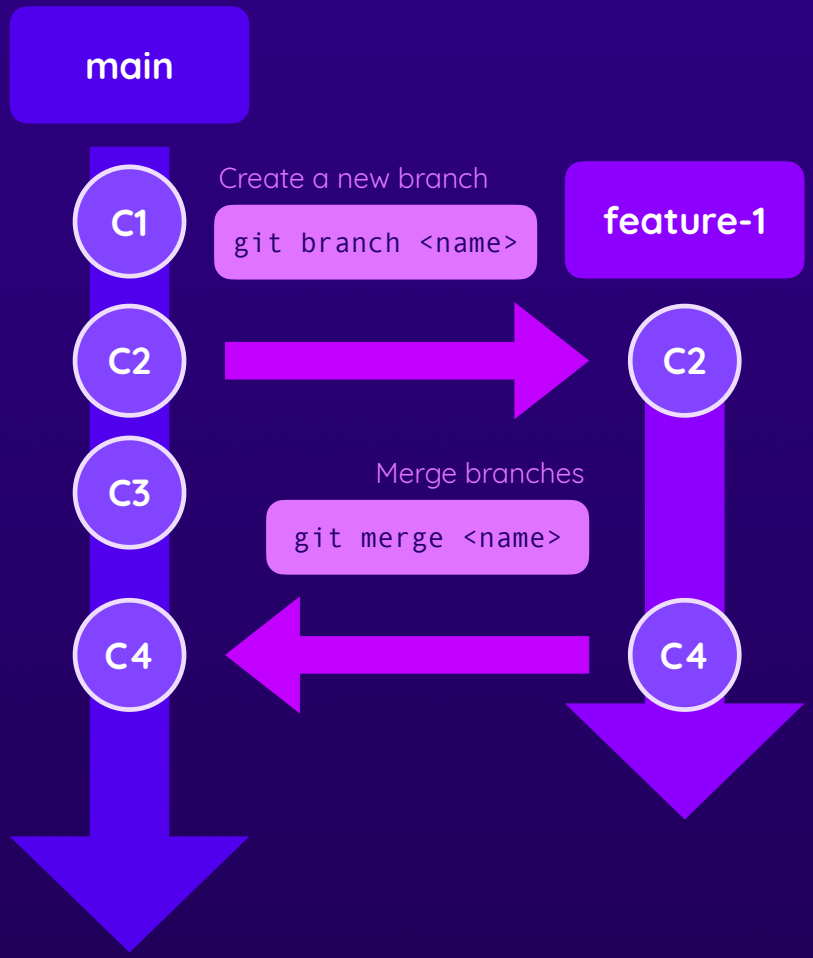


ACADEMIND

Key Commands

`git init`	**Initialize a Git repository** (only required once per project)
`git add <file(s)>`	**Stage code changes** (for the next commit)
`git commit -m "..."`	**Create a commit for the staged changes** (with a message)
`git status`	**Get the current repository status** (e.g., which changes are staged)
`git log`	**Output a chronologically ordered list of commits**
`git checkout <id>`	**Temporarily move back to commit <id>**
`git revert <id>`	**Revert the changes of commit <id> (**by creating a new commit)
`git reset <id>`	**Undo commit(s) up to commit <id> by deleting commits**

Understanding Git Branches



What Is GitHub?

A cloud Git repository & services provider

Store & manage Git repositories

Cloud Git repository storage ("push" & "pull")

Backup, work across machines & work in teams

Public & private, team management & more

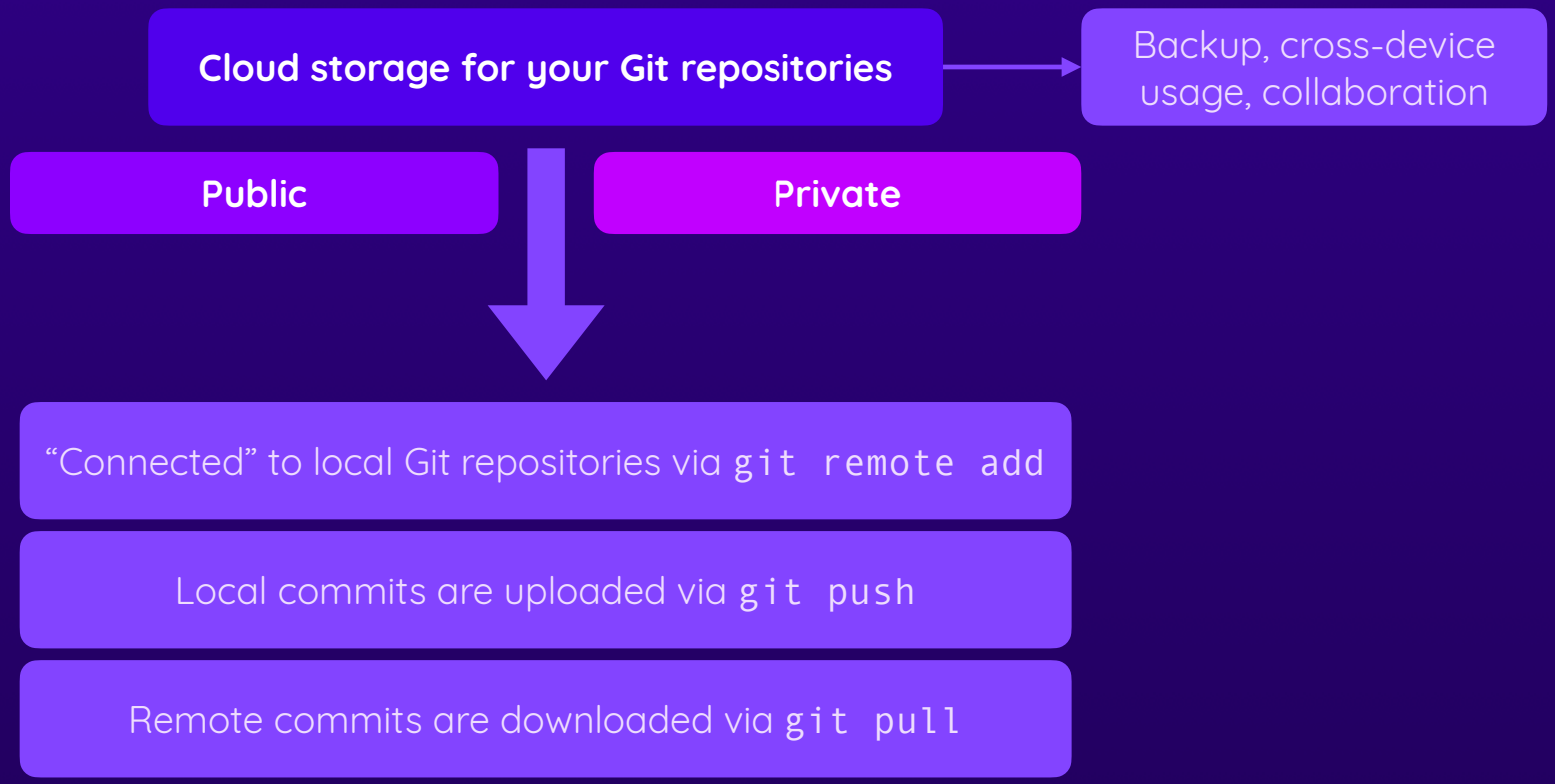
Code management & collaborative development

Via "Issues", "Projects", "Pull Requests" & more

Automation & CI / CD

Via **GitHub Actions**, GitHub Apps & more

GitHub Repositories



Forking & Pull Requests

Repository Forking

Creates a standalone copy of a repository

Can be used to work on code without affecting the original repository

Pull Requests

Requests merging code changes into a branch

Can be based on a forked repository or another branch from the same repository

Pull requests allow for code reviews before merging changes

ACADEMIND

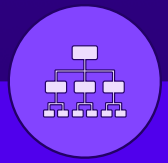
GitHub Actions: Fundamentals

Key Building Blocks & Usage

- Understanding the Key Elements
- Working with Workflows, Jobs & Steps
- Building an Example Workflow

ACADEMIND

Key Elements



Workflows

- Attached to a GitHub repository
- Contain one or more **Jobs**
- Triggered upon **Events**

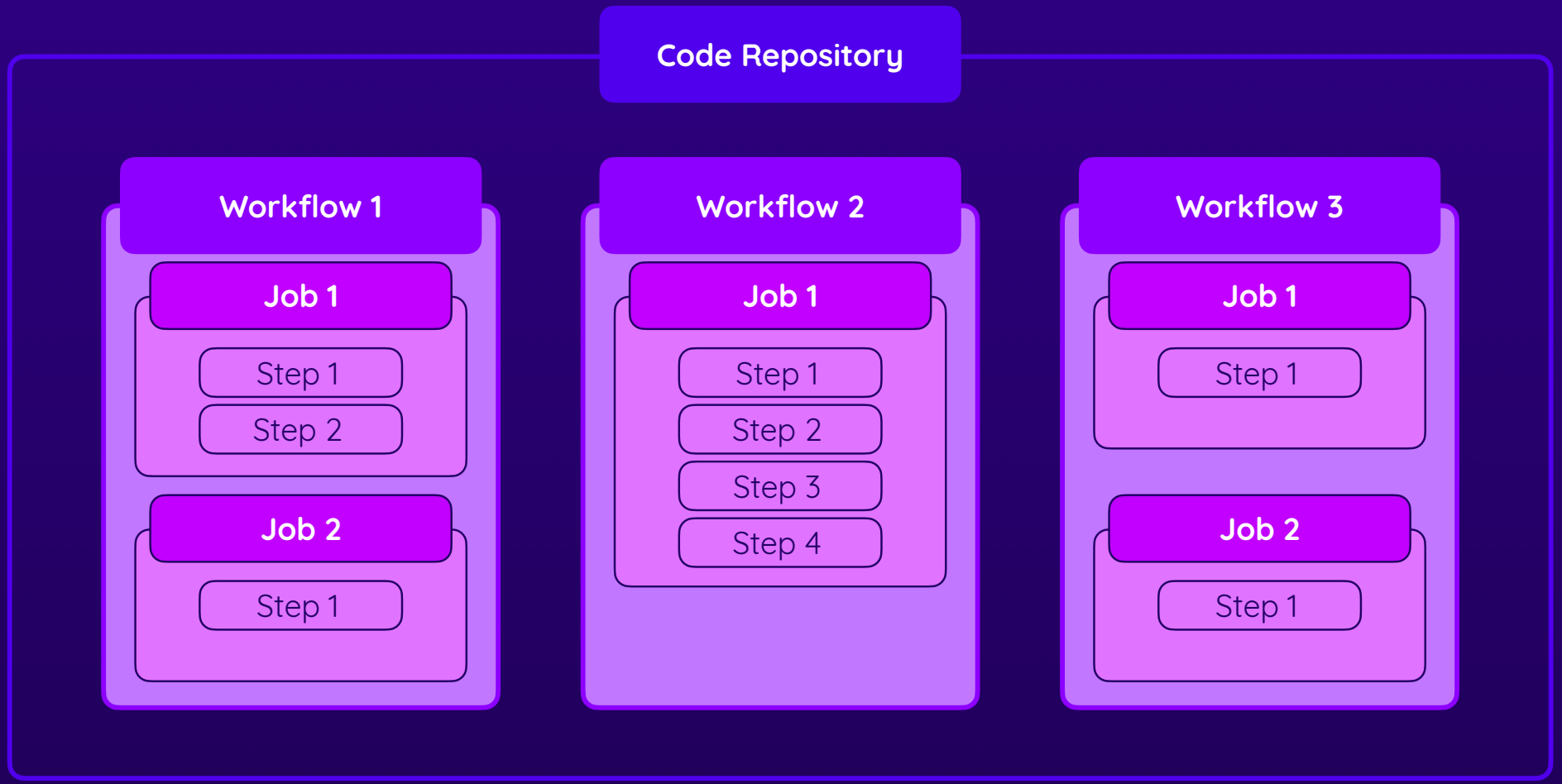
Jobs

- Define a **Runner** (execution environment)
- Contain one or more **Steps**
- Run in parallel (default) or sequential
- Can be conditional

Steps

- Execute a **shell script** or an **Action**
- Can use custom or third-party actions
- Steps are executed in order
- Can be conditional

Workflows, Jobs & Steps



Events (Workflow Triggers)

Repository-related

push
Pushing a commit

pull_request
Pull request action (opened, closed, ...)

create
A branch or tag was created

fork
Repository was forked

issues
An issue was opened, deleted, ...

issue_comment
Issue or pull request comment action

watch
Repository was starred

discussion
Discussion action (created, deleted, ...)

Many More!

Other

workflow_dispatch
Manually trigger workflow

repository_dispatch
REST API request triggers workflow

schedule
Workflow is scheduled

workflow_call
Can be called by other workflows

What Are Actions?

Command ("run")

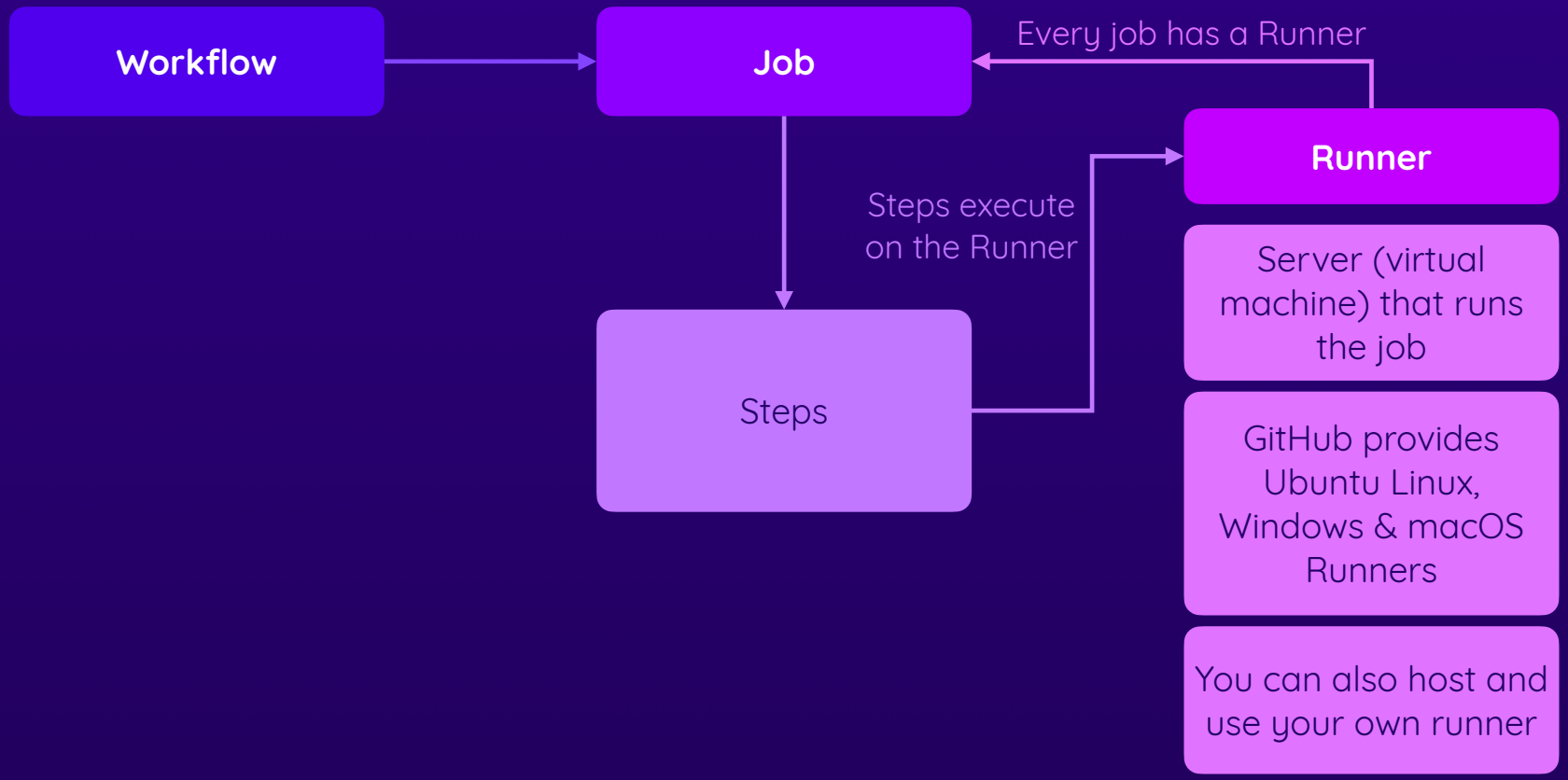
A (typically simple) shell command that's defined by you

Action

A (custom) application that performs a (typically complex) frequently repeated task

You can build your own Actions but you can also use official or community Actions

Job Runners


Workflow
Job
Every job has a Runner
Runner
Steps execute on the Runner
Steps
Server (virtual machine) that runs the job
GitHub provides Ubuntu Linux, Windows & macOS Runners
You can also host and use your own runner

Module Summary

Core Components

- Workflows: Define Events + Jobs
- Jobs: Define Runner + Steps
- Steps: Do the actual work

Defining Workflows

- .github/workflows/<file>.yml (on GitHub or locally)
- GitHub Actions syntax must be followed

Events / Triggers

- Broad variety of events (repository-related & other)
- Workflows have at least one (but possible more) event(s)

Runners

- Servers (machines) that execute the jobs
- Pre-defined Runners (with different OS) exist
- You can also create custom Runners

Workflow Execution

- Workflows are executed when their events are triggered
- GitHub provides detailed insights into job execution (+ logs)

Actions

- You can run shell commands
- But you can also use pre-defined Actions (official, community or custom)

Exercise Time!