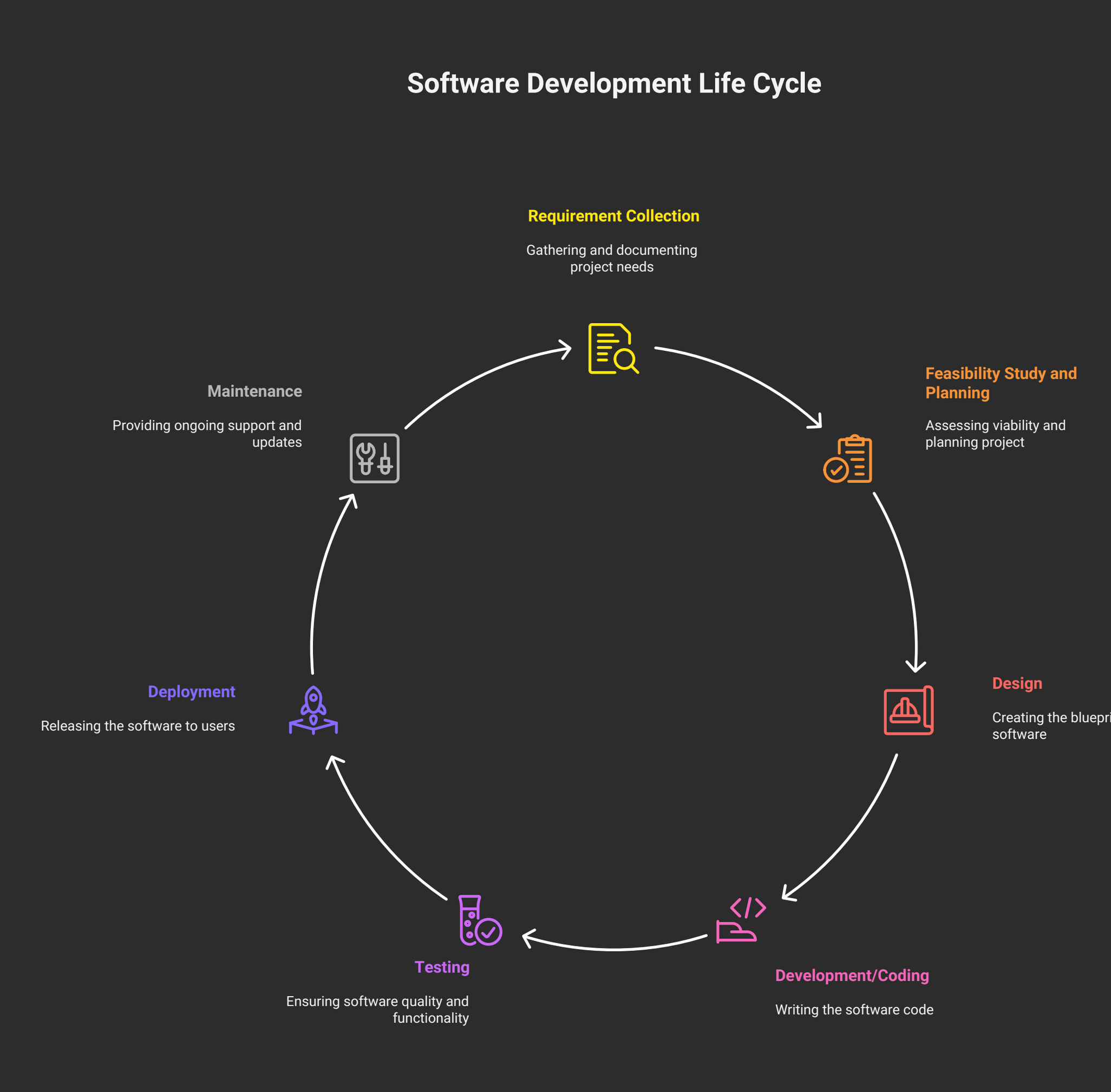
Software Development Life Cycle (SDLC)

Date: 22/05/2005

SDLC stands for **Software Development Life Cycle**. It is the process of developing a software application through different stages in a structured manner.

Stages of SDLC:

1. Requirement Collection
2. Feasibility Study and Planning
3. Design
4. Development/Coding
5. Testing
6. Deployment
7. Maintenance



1. Requirement Collection

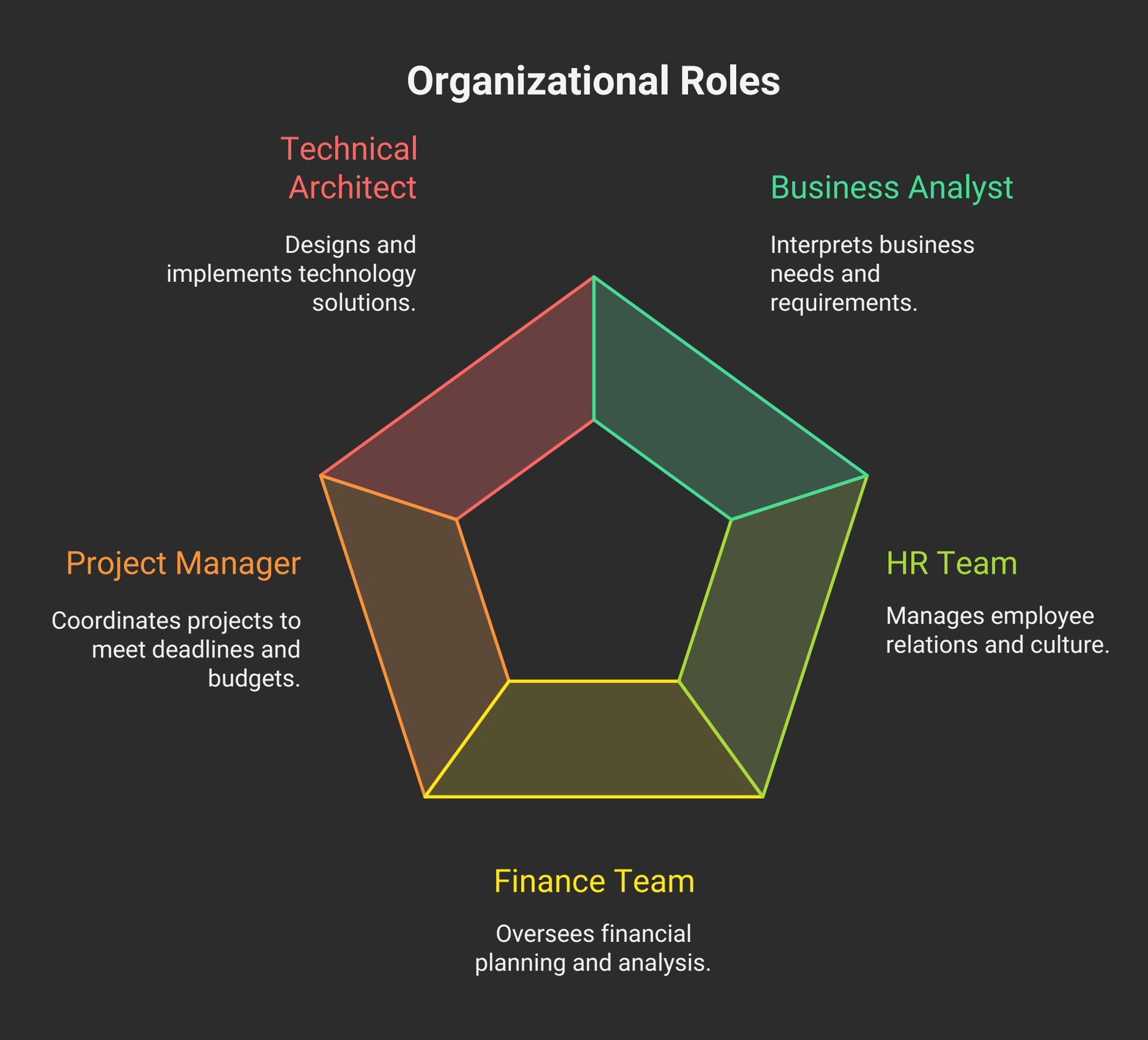
In this phase, the requirements are collected from the client and documented in a **Software Requirement Specification (SRS)** document. This task is typically handled by a **Business Analyst** from the organization.

2. Feasibility Study and Planning

In this stage, a group of professionals from the organization analyzes whether the project is feasible from different perspectives (technical, financial, operational).

This team typically includes:

- Business Analyst
- HR Team
- Finance Team
- Project Manager
- Technical Architect



3. Designing the Application

- **High-Level Design (HLD):**
  - Designs the core structure and architecture.
  - Defines modules and their interactions.
  - Handled by: **Technical Architect**
- **Low-Level Design (LLD):**
  - Describes each component, user interfaces, database schemas, and internal logic.

4. Development / Coding

Based on the finalized design, developers begin writing code for the application.

- Handled by: **Developers**

5. Testing

After development, **Test Engineers** verify that every feature works as expected.

- Includes: Unit Testing, Integration Testing, System Testing, UAT (User Acceptance Testing)

6. Deployment

Once confirmed bug-free, the application is deployed to the client's environment or server.

7. Maintenance

Post-deployment, updates, bug fixes, and improvements are done by maintenance teams.

> **Note:** Deployment and maintenance are usually handled by different specialized teams.

Software Process Implementation Concepts (SPIC)

To implement SPIC, various software development models are used:

- Waterfall Model
- Spiral Model
- V-Model
- Prototype Model
- Agile Model

What is DevOps?

DevOps is a **practice or culture** that focuses on automating and integrating software development and IT operations.

Goal of DevOps:

To break the wall between development and operations teams, ensuring faster and more reliable software delivery.

DevOps supports:

- Continuous Integration (CI)
- Continuous Delivery (CD)
- Continuous Deployment

> DevOps complements the **Agile Model** for frequent and rapid software delivery.

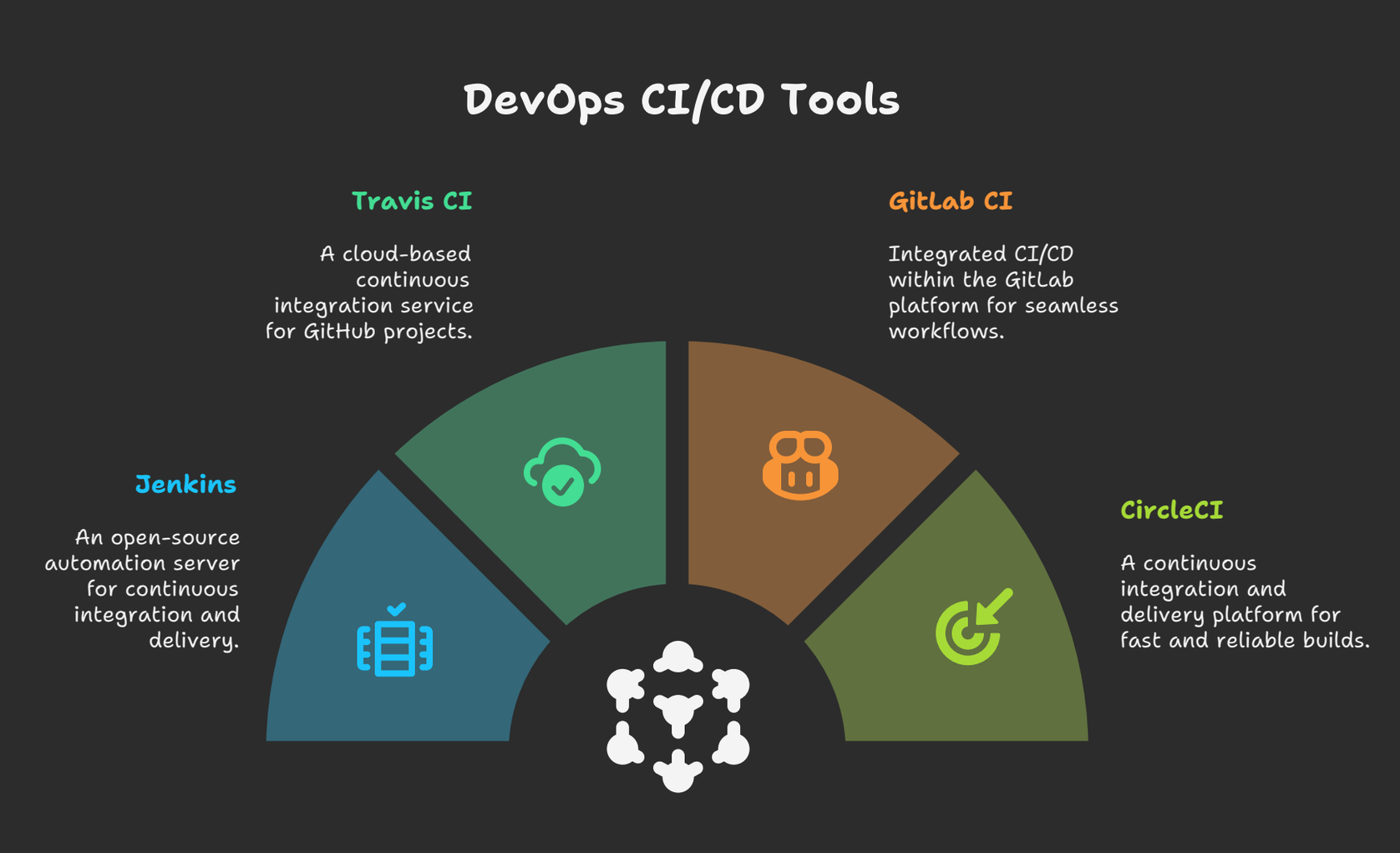
Common DevOps Tools and Components

1. Version Control Tools

- Git
- GitHub
- GitLab
- Bitbucket

2. CI/CD Tools (Continuous Integration / Continuous Deployment)

- Jenkins
- Travis CI
- GitLab CI
- CircleCI



3. Build Tools

- Maven
- Gradle
- Ant
- Yarn

4. Package Managers

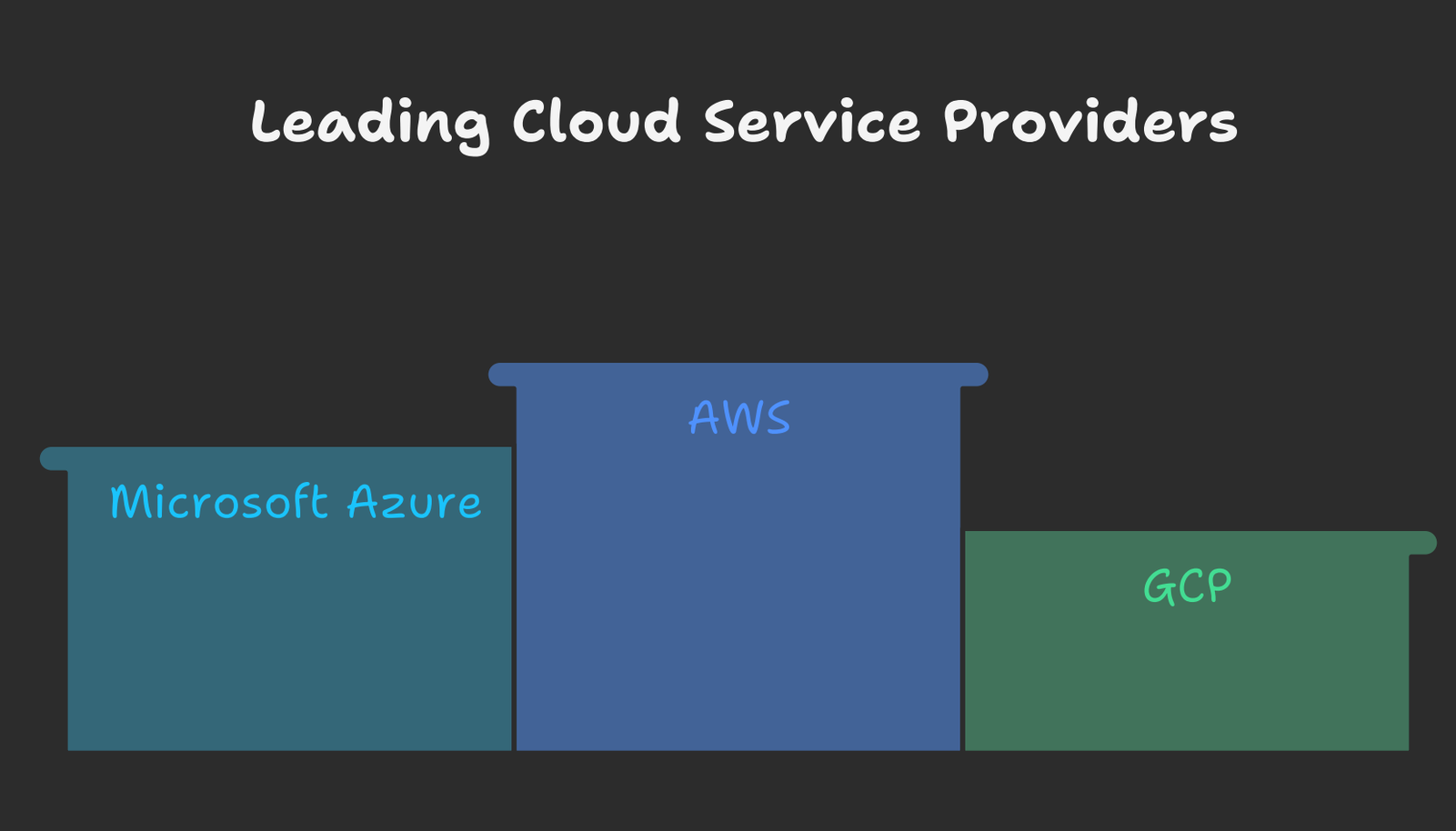
- pip (Python)
- npm (Node.js)

5. Server Configuration Tools

- Ansible
- Puppet
- Chef
- SaltStack

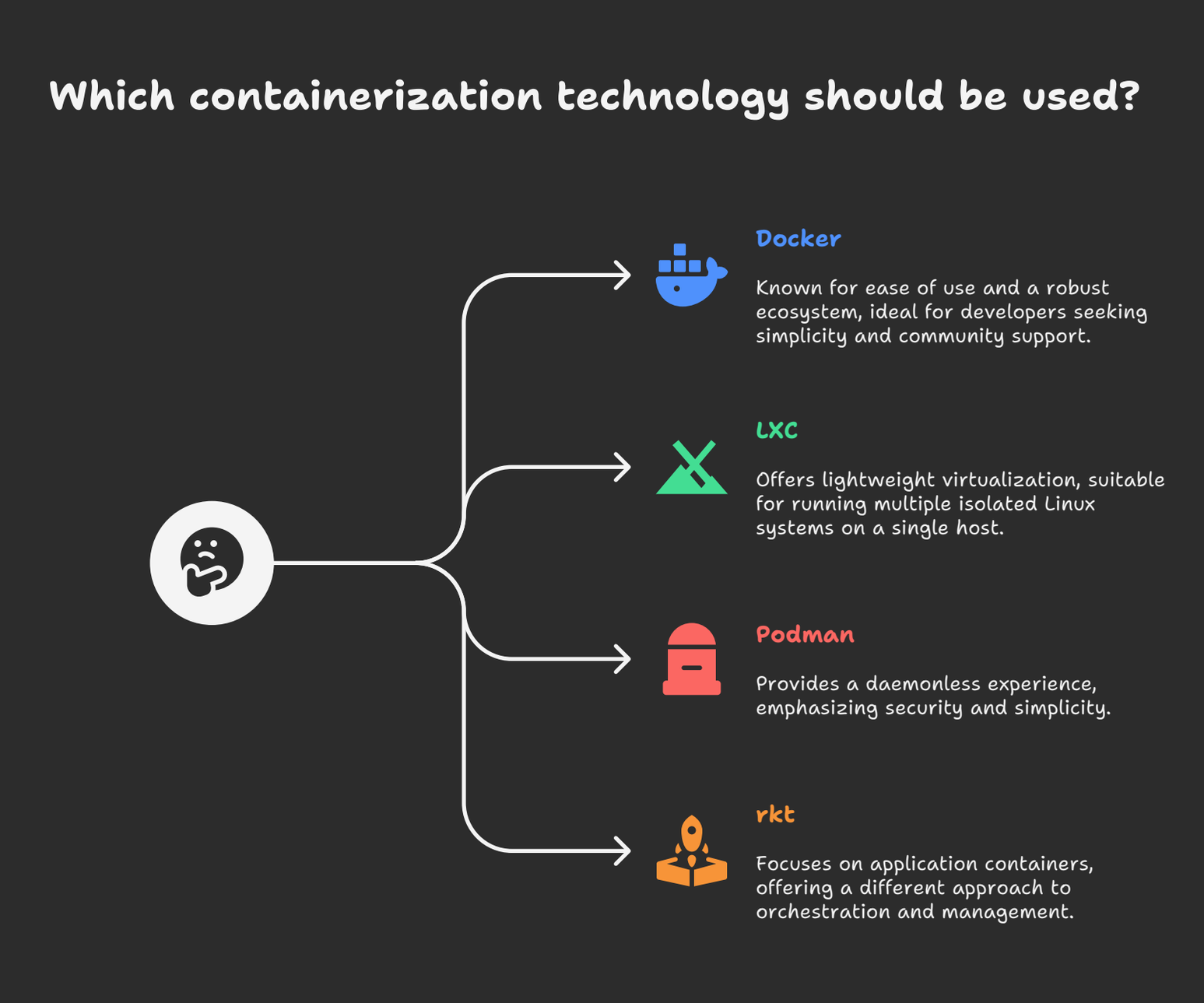
6. Cloud Computing Platforms

- AWS (Amazon Web Services)
- Microsoft Azure
- GCP (Google Cloud Platform)



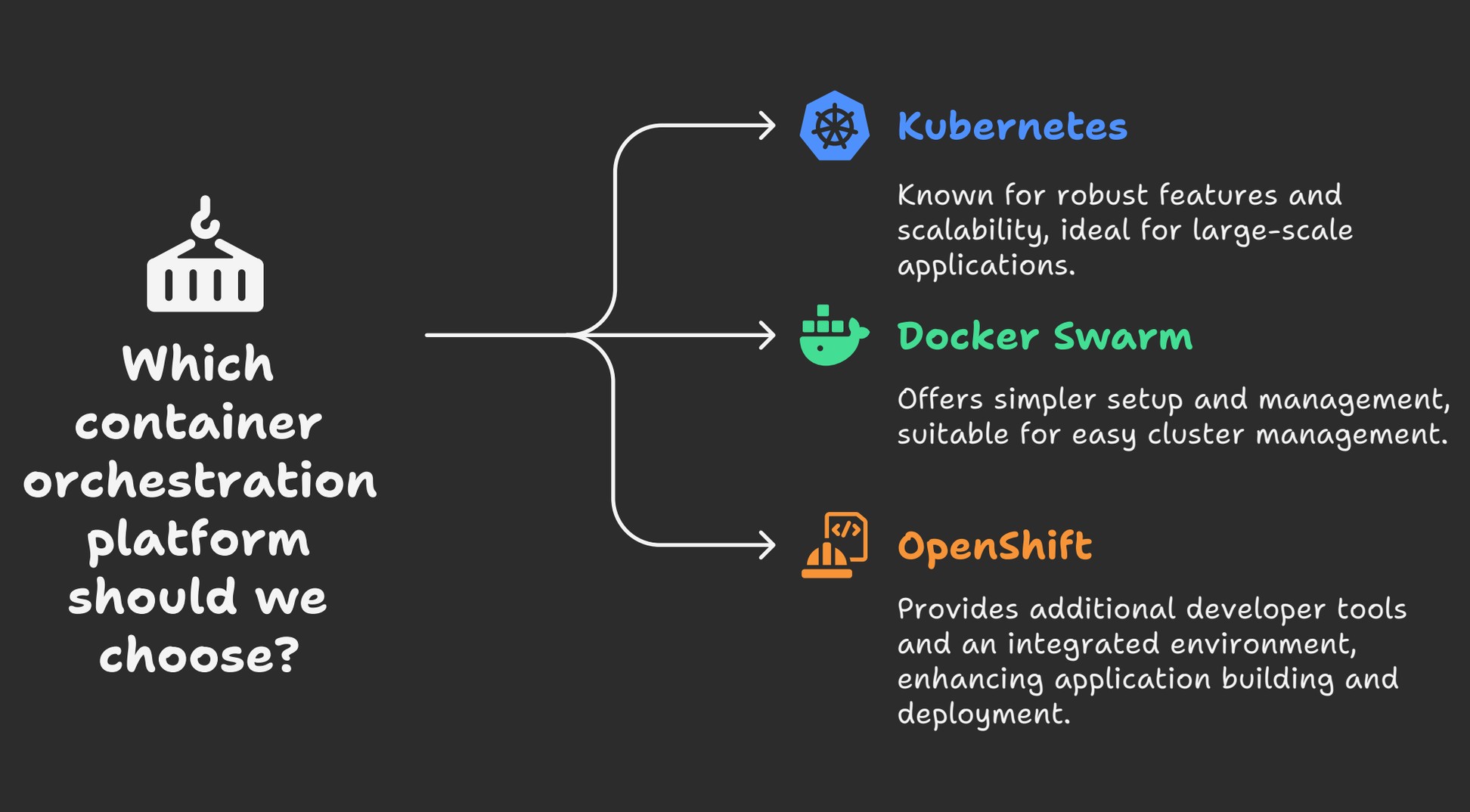
7. Containerization Tools

- Docker
- Linux Containers (LXC)
- Podman
- rkt (Rocket)



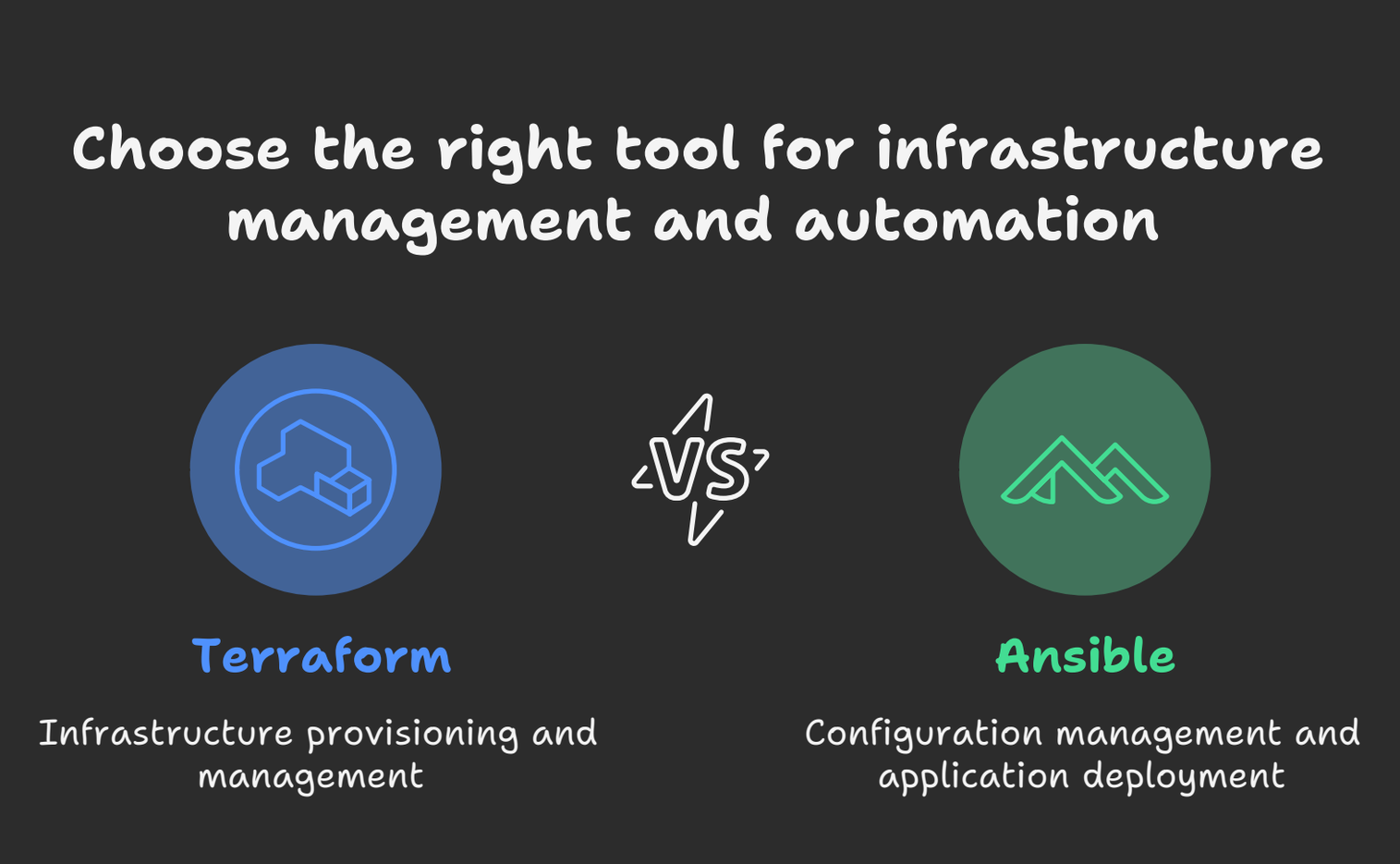
8. Container Orchestration Tools

- Kubernetes
- Docker Swarm
- OpenShift



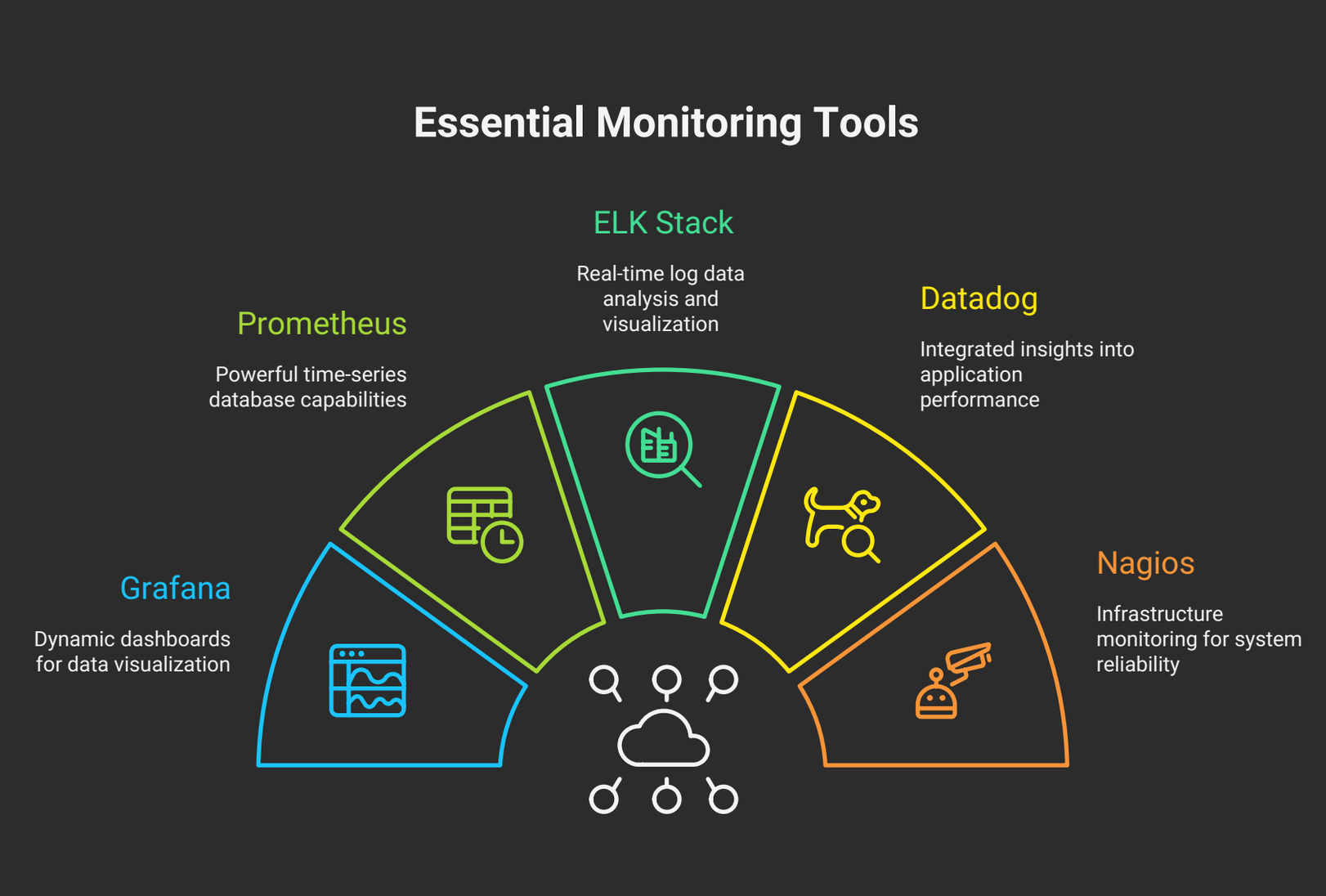
9. Cloud Management & Infrastructure as Code (IaC) Tools

- Terraform
- Ansible



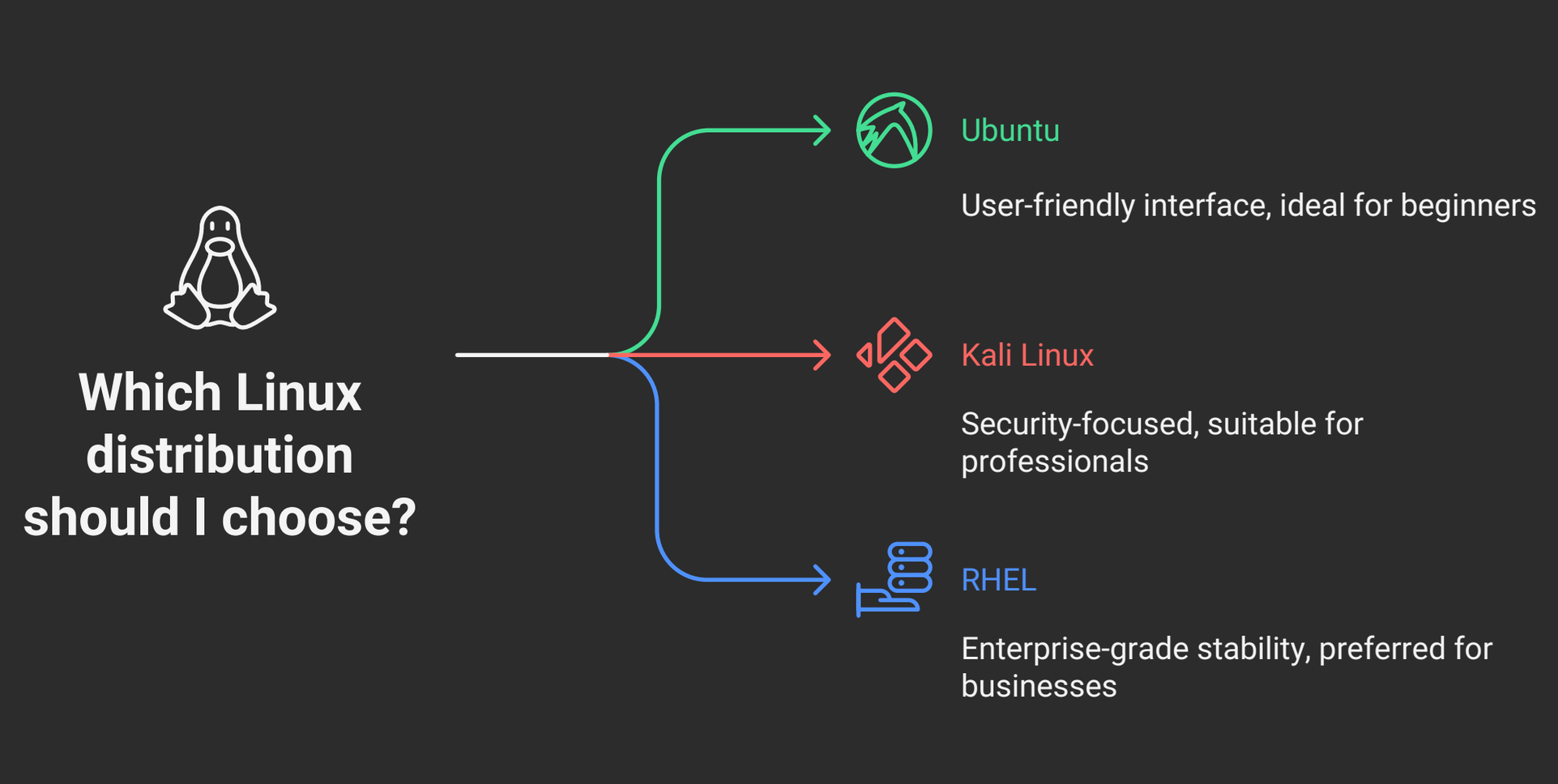
10. Monitoring & Logging Tools

- Grafana
- Prometheus
- ELK Stack (Elasticsearch, Logstash, Kibana)
- Datadog
- Nagios



11. Operating Systems Commonly Used in DevOps

- Ubuntu
- Kali Linux
- Red Hat Enterprise Linux (RHEL)



12. Scripting Languages

- Shell Scripting (Bash, Zsh)
- Python Scripting
- GoLang Scripting

What is Software?

Software is a **set of instructions/programs** that perform specific tasks on a computer. It is designed with specific logic and functionality and is created by developers.

Types of Software

1. **Application Software**
   - Designed to perform specific tasks for users.
   - Examples: MS Word, Excel, Chrome, WhatsApp, YouTube
2. **System Software**
   - Manages hardware and provides a platform to run application software.
   - Examples:
     - Operating Systems: Windows, Linux, macOS, Android
     - Device Drivers
     - Utility Programs

Architecture of Operating Systems

An **Operating System (OS)** acts as an interface between hardware and application software.

Features of an Operating System

- **Memory Management** – Manages RAM and memory allocation
- **Process Management** – Schedules and manages processes
- **Storage Management** – Controls file systems and disk storage
- **Device Management** – Manages input/output devices
- **User Interface** – Provides GUI or CLI for interaction

Why Linux?

- Linux is **open-source, secure, lightweight, and customizable**.
- Widely used in servers, DevOps, and cloud platforms.
- Most DevOps tools are **built on or for Linux environments**.

What is a Server?

A **server** is a powerful computer that delivers data, applications, or services to other computers (clients) over a network.

Typical server components:

- CPU
- RAM
- Storage
- Operating System (usually Linux-based)

Example:

If you run an application like Microsoft Word:

- The OS manages memory and CPU usage
- It allows file read/write access
- It handles input from mouse/keyboard and displays output on screen