Kubernetes Introduction & Motivation

Welcome!

Agenda

- Motivation for Kubernetes
- The Problem: Modern Application Delivery
- Containers & Microservices
- Kubernetes Fundamentals
- Key Concepts & Architecture
- Features & Benefits
- Industry Adoption
- Summary & Next Steps

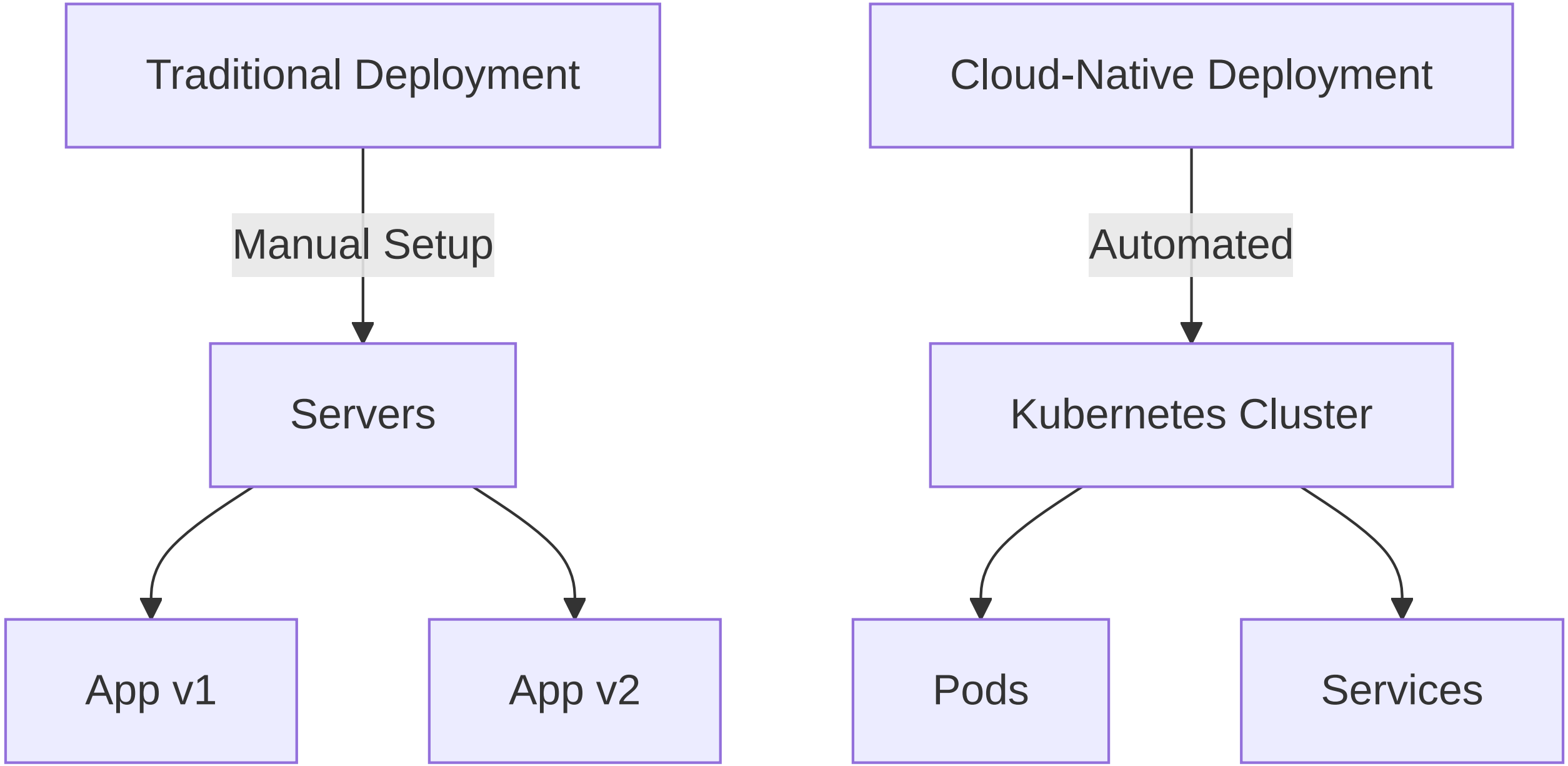
Motivation for Kubernetes

Why do we need Kubernetes?

- Rapid growth of cloud-native applications
- Need for scalable, resilient, and portable infrastructure
- Complexity of application deployment and management

Traditional Deployment Challenges

- Manual configuration and setup
- Difficult scaling and updates
- Environment drift
- Inefficient resource utilization


Traditional Deployment
Manual Setup
Servers
App v1
App v2
Cloud-Native Deployment
Automated
Kubernetes Cluster
Pods
Services

The Evolution of Application Delivery

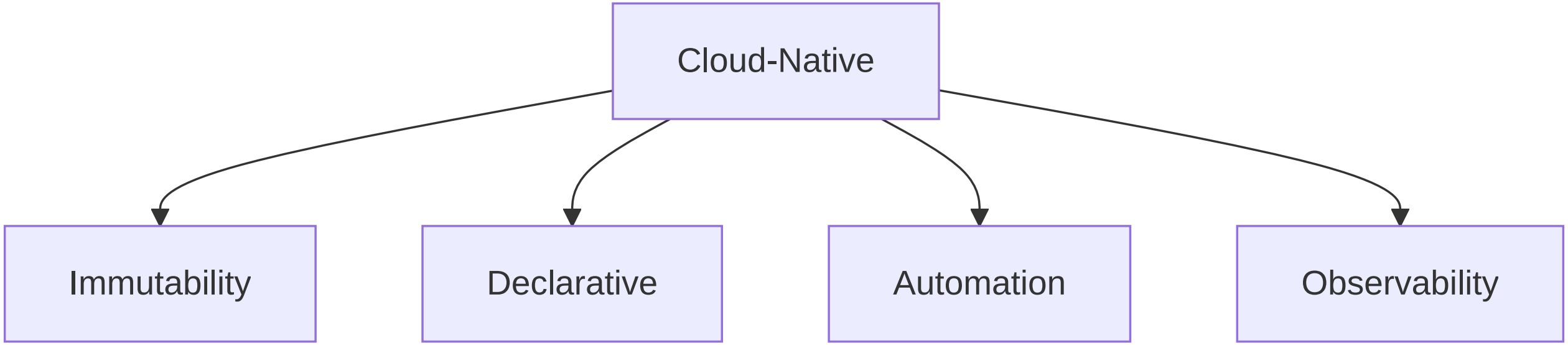
- **Physical servers** → Virtual machines → Containers → Orchestrators

What is Cloud-Native?

- Applications designed for scale, resilience, and automation
- Microservices, containers, DevOps

Principles of Cloud-Native

- Immutability
- Declarative configuration
- Automation
- Observability


Cloud-Native
Immutability
Declarative
Automation
Observability

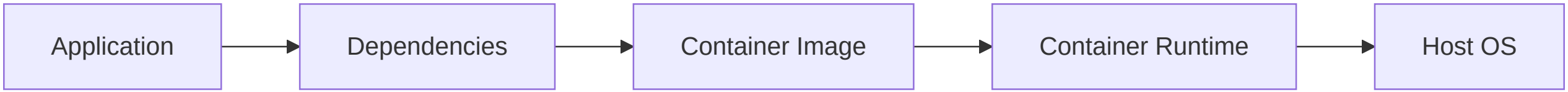
Containers & Their Advantages

What is a Container?

- Lightweight, portable, consistent runtime environment
- Encapsulates application and dependencies

Benefits of Containers

- Portability
- Efficiency
- Isolation
- Scalability


Application
Dependencies
Container Image
Container Runtime
Host OS

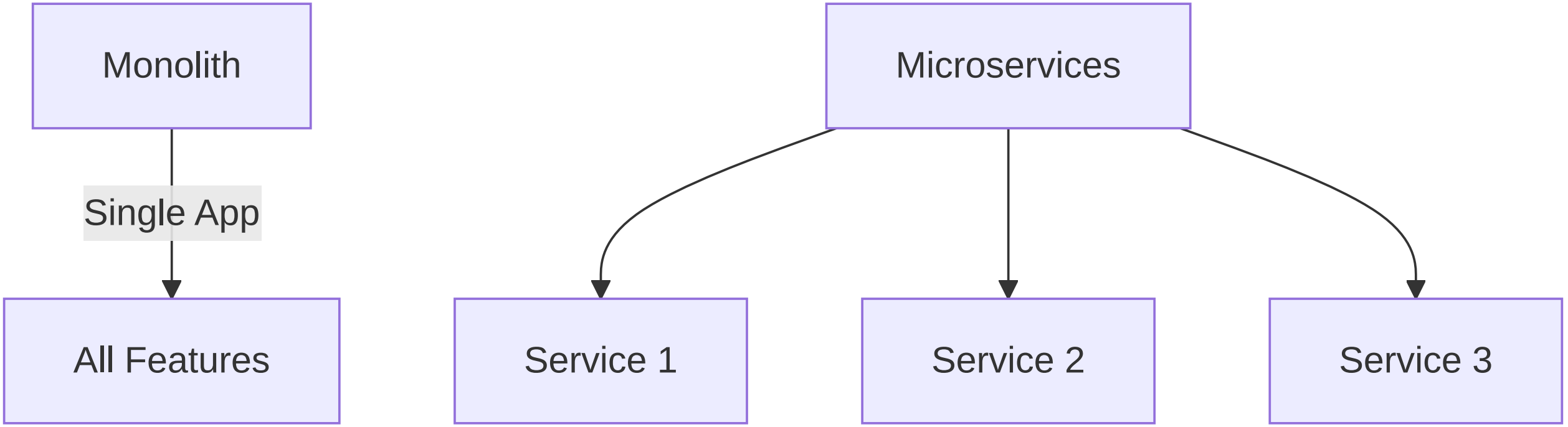
Microservices Architecture

What are Microservices?

- Decompose applications into small, independent services
- Each service: single responsibility

Microservices vs. Monolith

- Easier scaling and updates
- Better fault isolation
- Complex networking and orchestration


Monolith
Single App
All Features
Microservices
Service 1
Service 2
Service 3

Why Not Just Docker?

Limitations of Docker Alone

- No built-in orchestration
- Manual networking
- Manual scaling and recovery

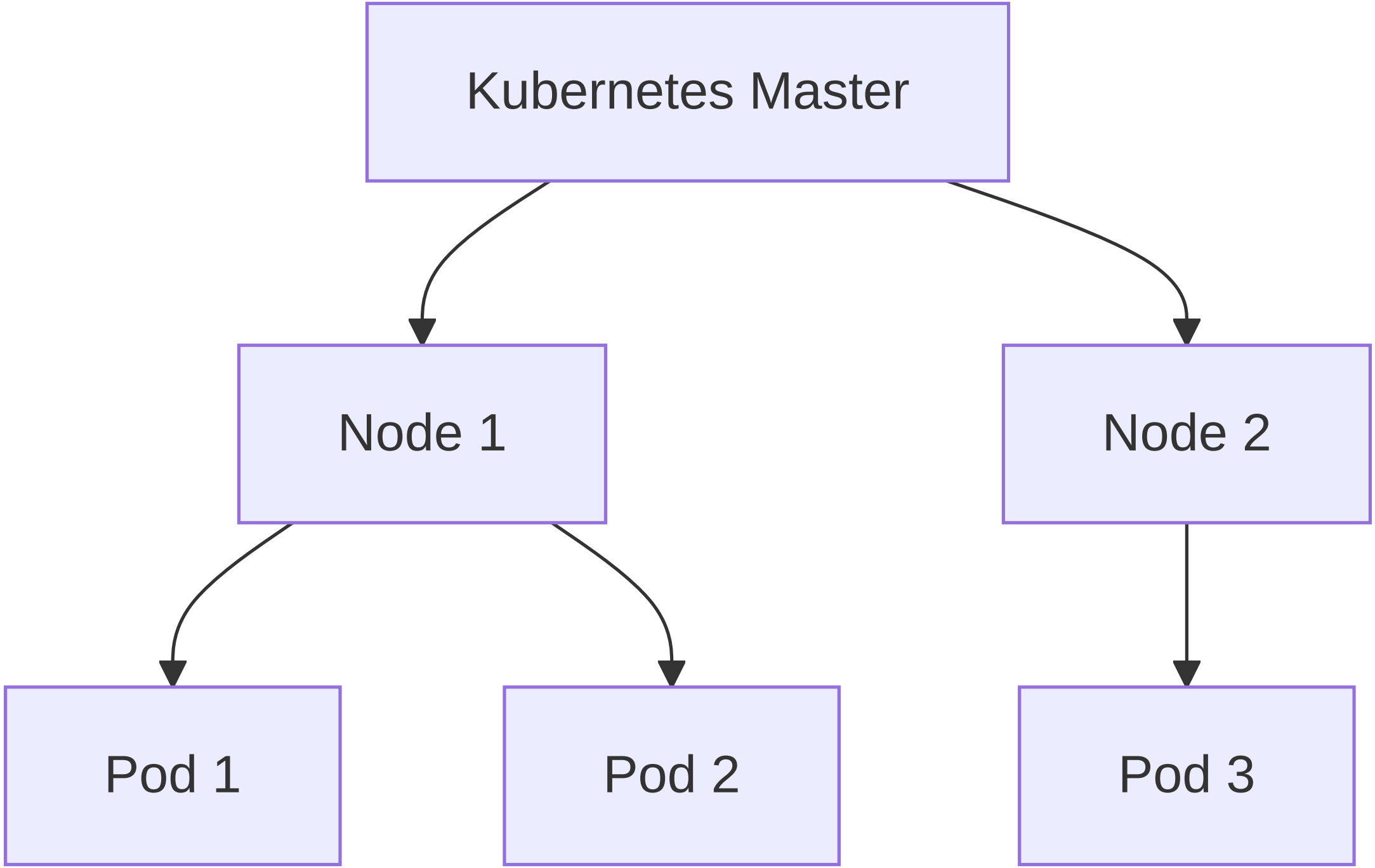
The Need for Orchestration

- Automate deployment, scaling, and management
- Monitor and heal applications

Kubernetes: What & Why

What is Kubernetes?

- Open-source container orchestration platform
- Automates deployment, scaling, and management


Kubernetes Master
Node 1
Node 2
Pod 1
Pod 2
Pod 3

Why Kubernetes?

- Declarative infrastructure
- Self-healing capabilities
- Portable and extensible

Kubernetes History & Community

A Brief History

- Originated at Google (Borg system)
- Open-sourced in 2014
- CNCF stewardship

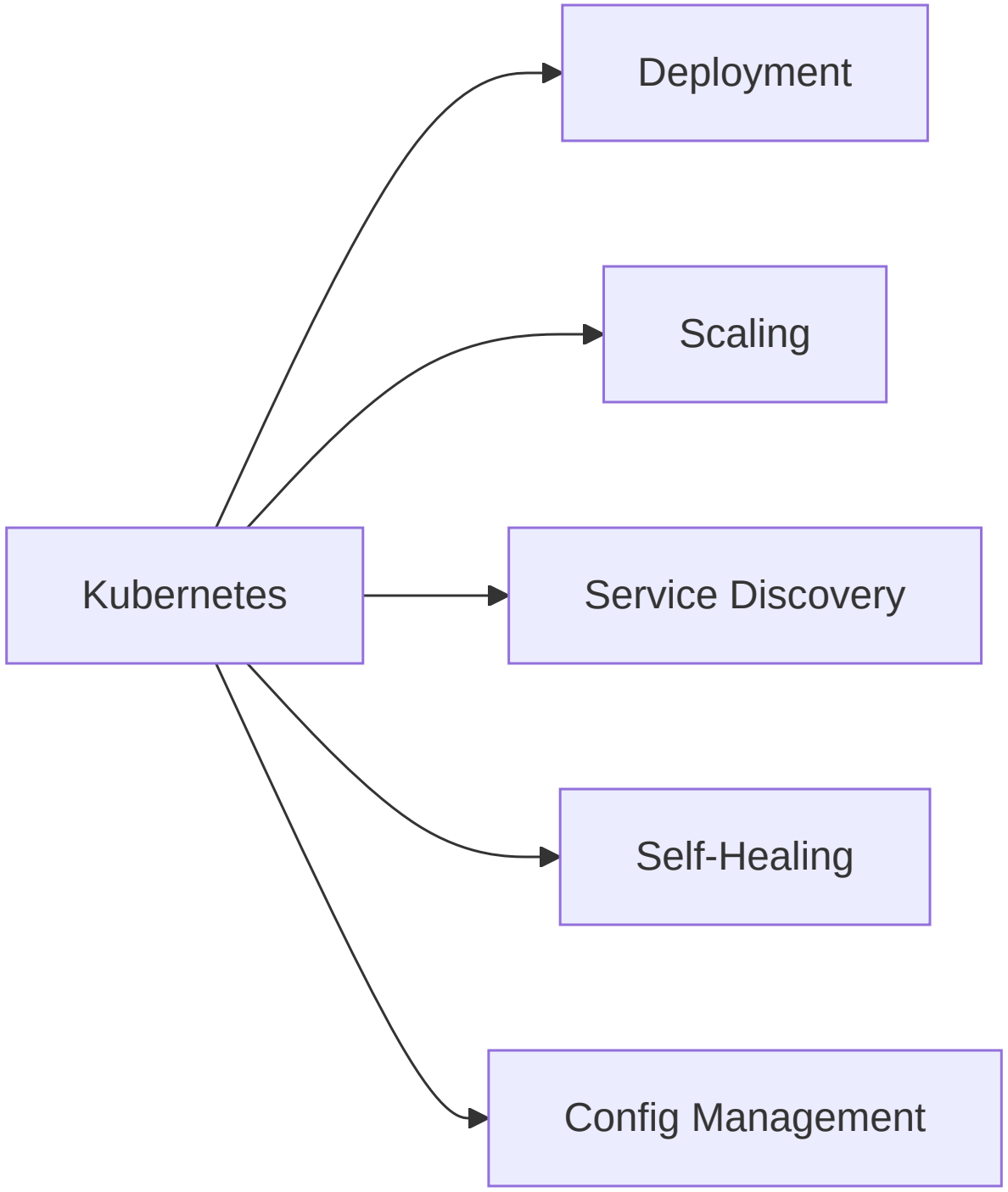
The Kubernetes Community

- Over 100,000 contributors
- Active development and innovation
- Rich ecosystem

Kubernetes Features Overview

Key Features

- Automated deployment
- Self-healing
- Horizontal scaling
- Service discovery
- Secret & config management


Deployment
Scaling
Kubernetes
Service Discovery
Self-Healing
Config Management

Kubernetes Architecture

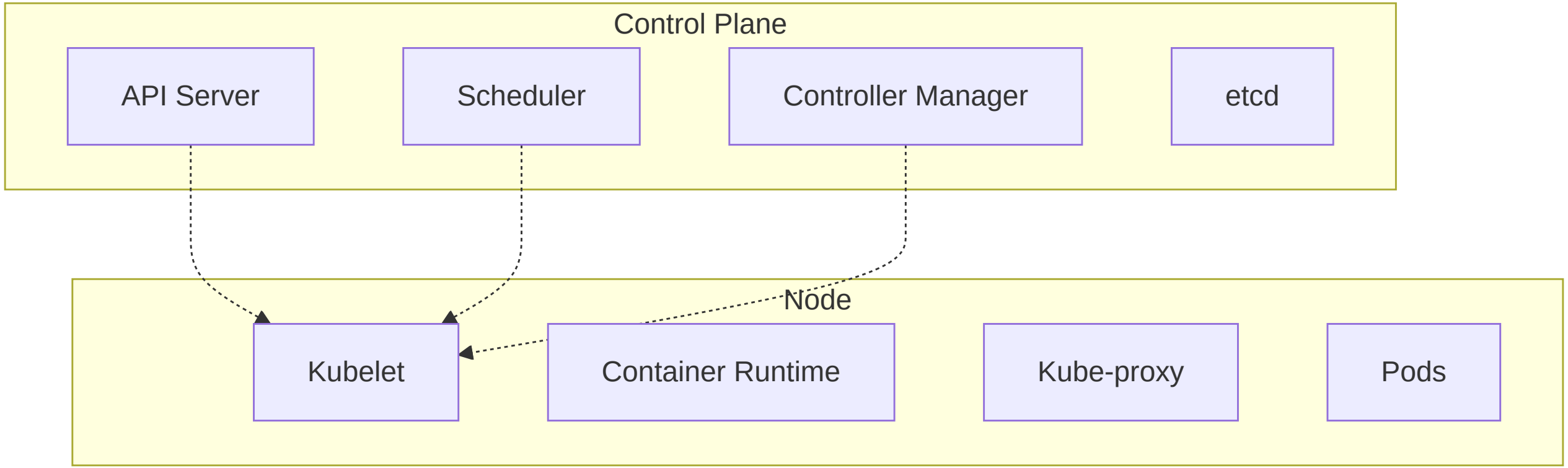
- Master components (control plane)
- Node components (worker nodes)
- Cluster resources

Control Plane Components

- API Server
- Scheduler
- Controller Manager
- etcd

Node Components

- Kubelet
- Container runtime
- Kube-proxy


Control Plane
API Server
Scheduler
Controller Manager
etcd
Node
Kubelet
Container Runtime
Kube-proxy
Pods

Core Concepts

Cluster

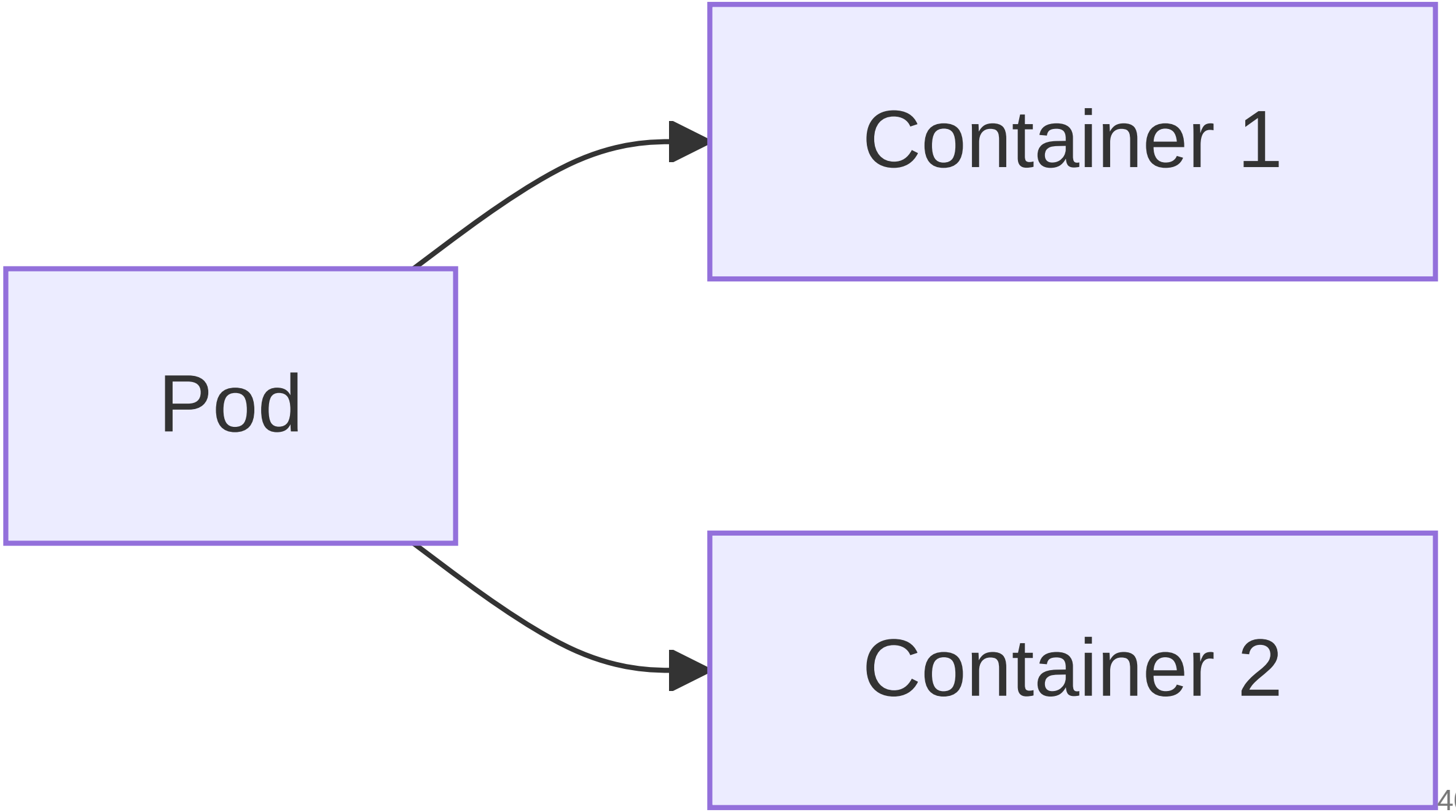
- A set of nodes managed by Kubernetes

Node

- Machine (VM or physical) running Kubernetes components

Pod

- Smallest deployable unit
- One or more containers


Container 1
Pod
Container 2

Service

- Abstracts access to pods
- Enables load balancing and discovery

Namespace

- Logical partitioning within a cluster

Declarative Infrastructure

Declarative vs. Imperative

- Declarative: specify desired state, Kubernetes reconciles
- Imperative: issue commands step-by-step

Configuration as Code

- YAML manifests define resources

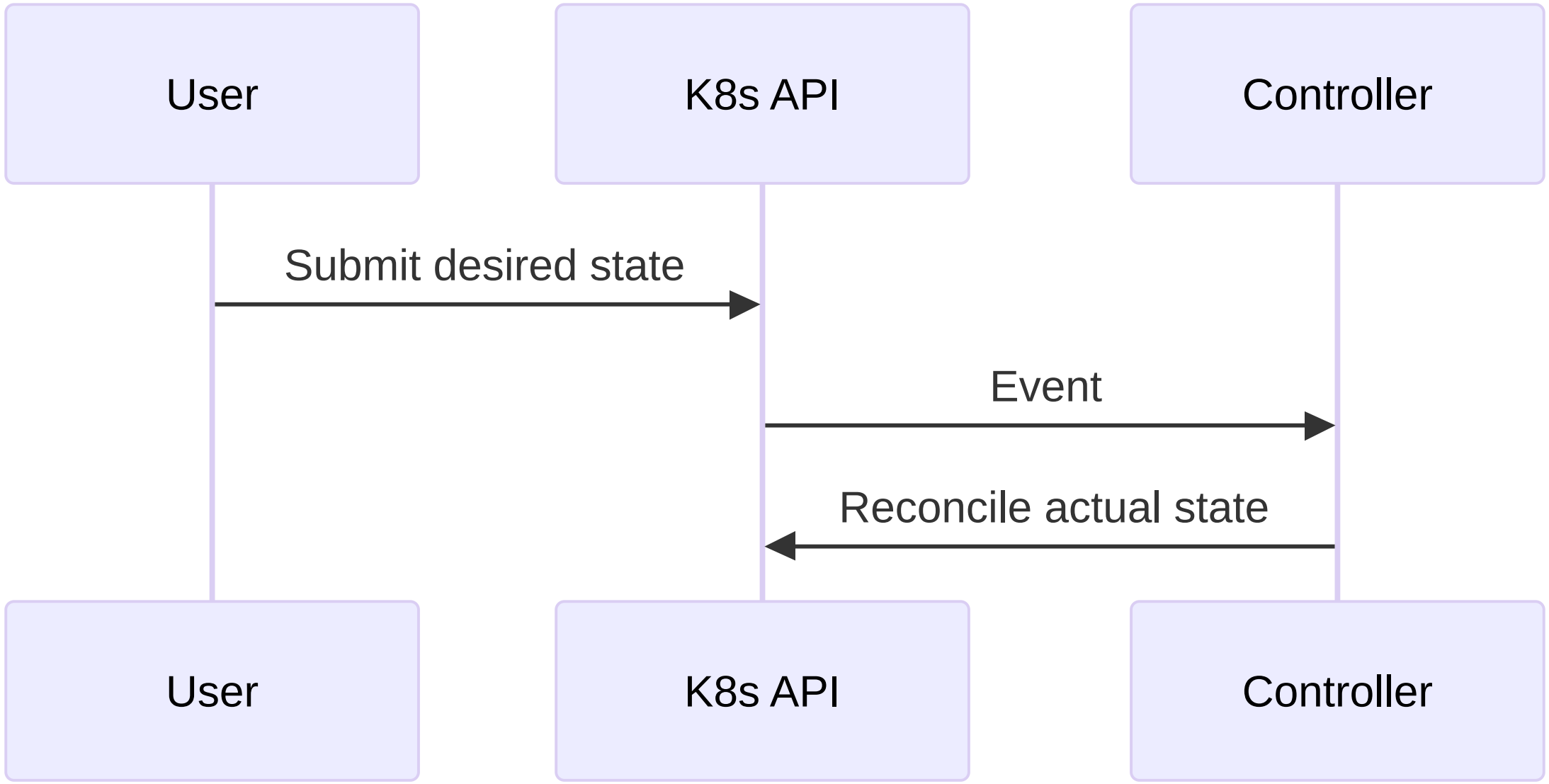
Desired State & Controllers

Desired State

- Expressed via resource definitions
- Kubernetes maintains the desired state

Controllers

- Manage resources to achieve desired state


User
K8s API
Controller
Submit desired state
Event
Reconcile actual state
User
K8s API
Controller

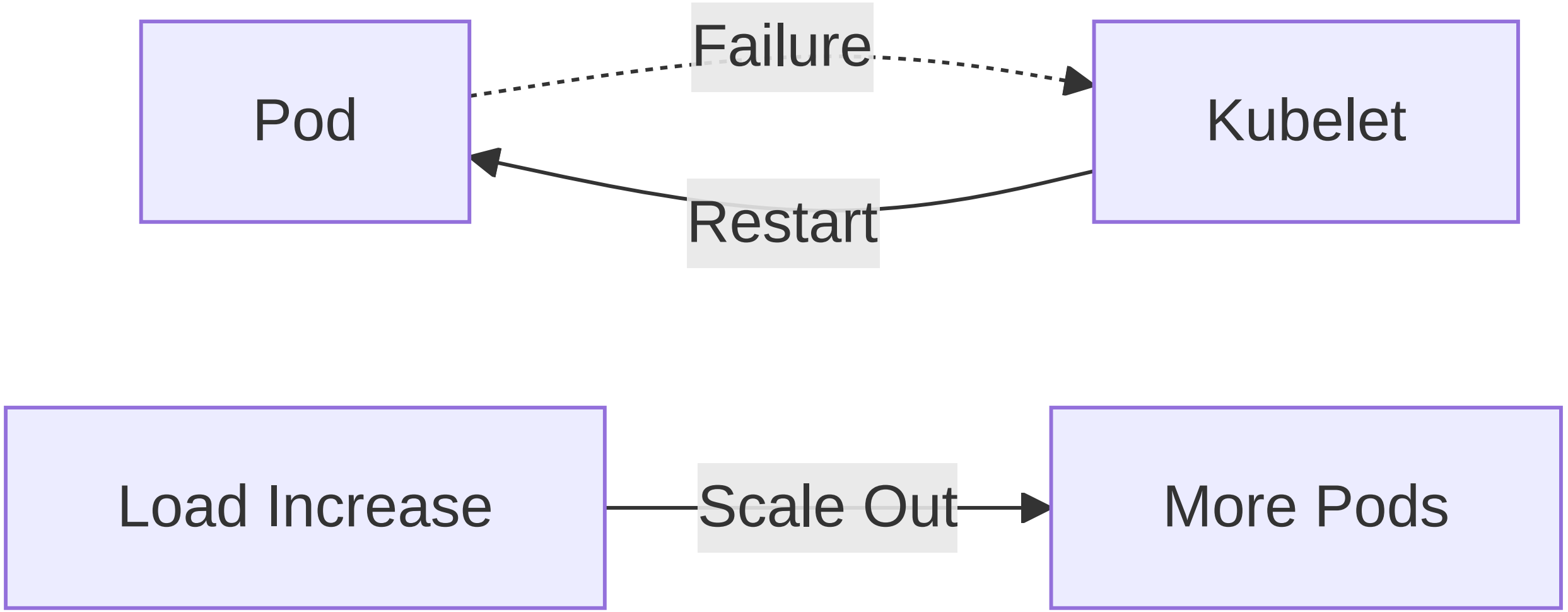
Scaling & Self-Healing

Horizontal Scaling

- Add more pods to handle load

Self-Healing

- Restart failed pods
- Replace unhealthy nodes


Pod
Failure
Kubelet
Restart
Load Increase
Scale Out
More Pods

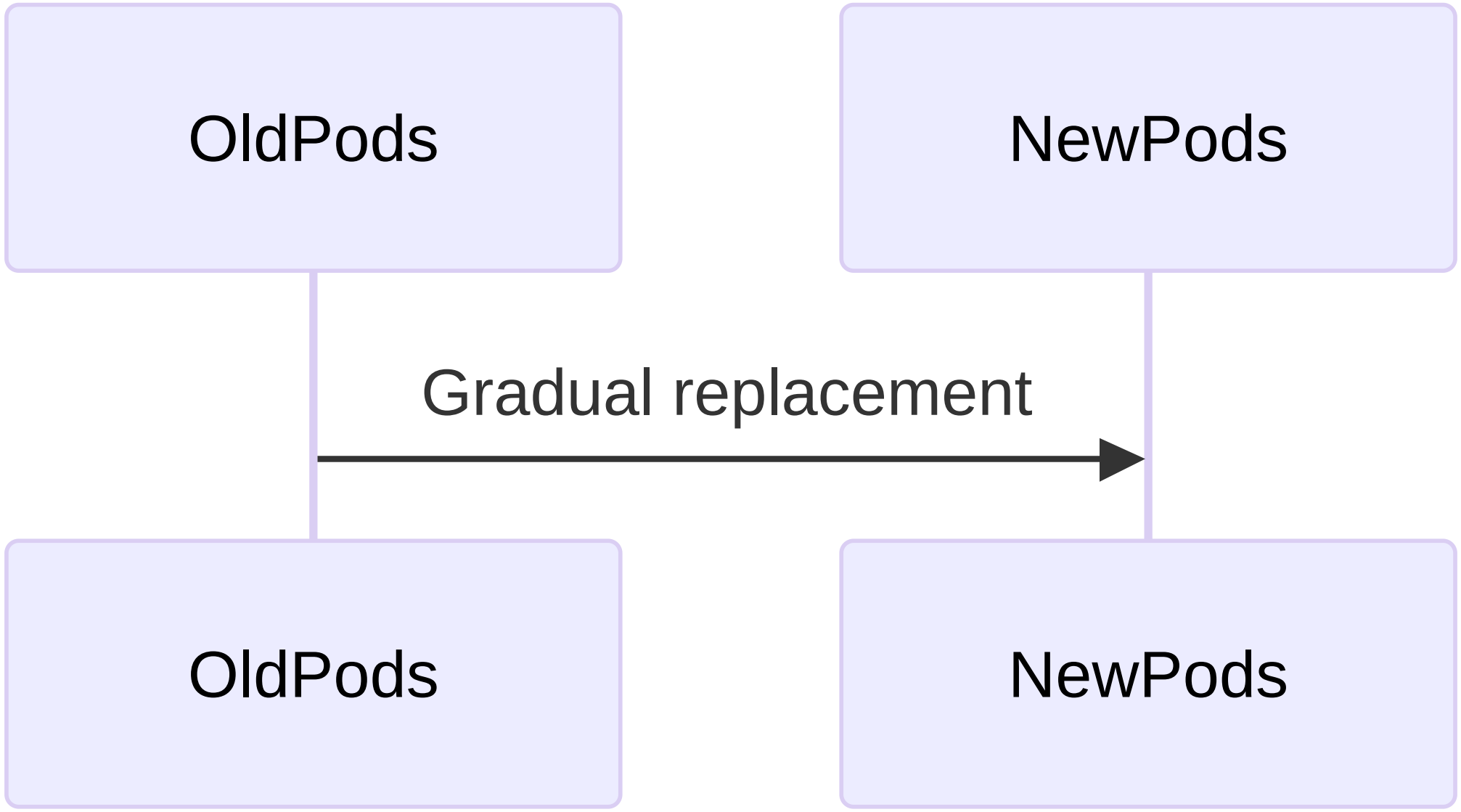
Rolling Updates & Rollbacks

Rolling Updates

- Update pods without downtime

Rollbacks

- Return to previous versions quickly


OldPods
NewPods
Gradual replacement
OldPods
NewPods

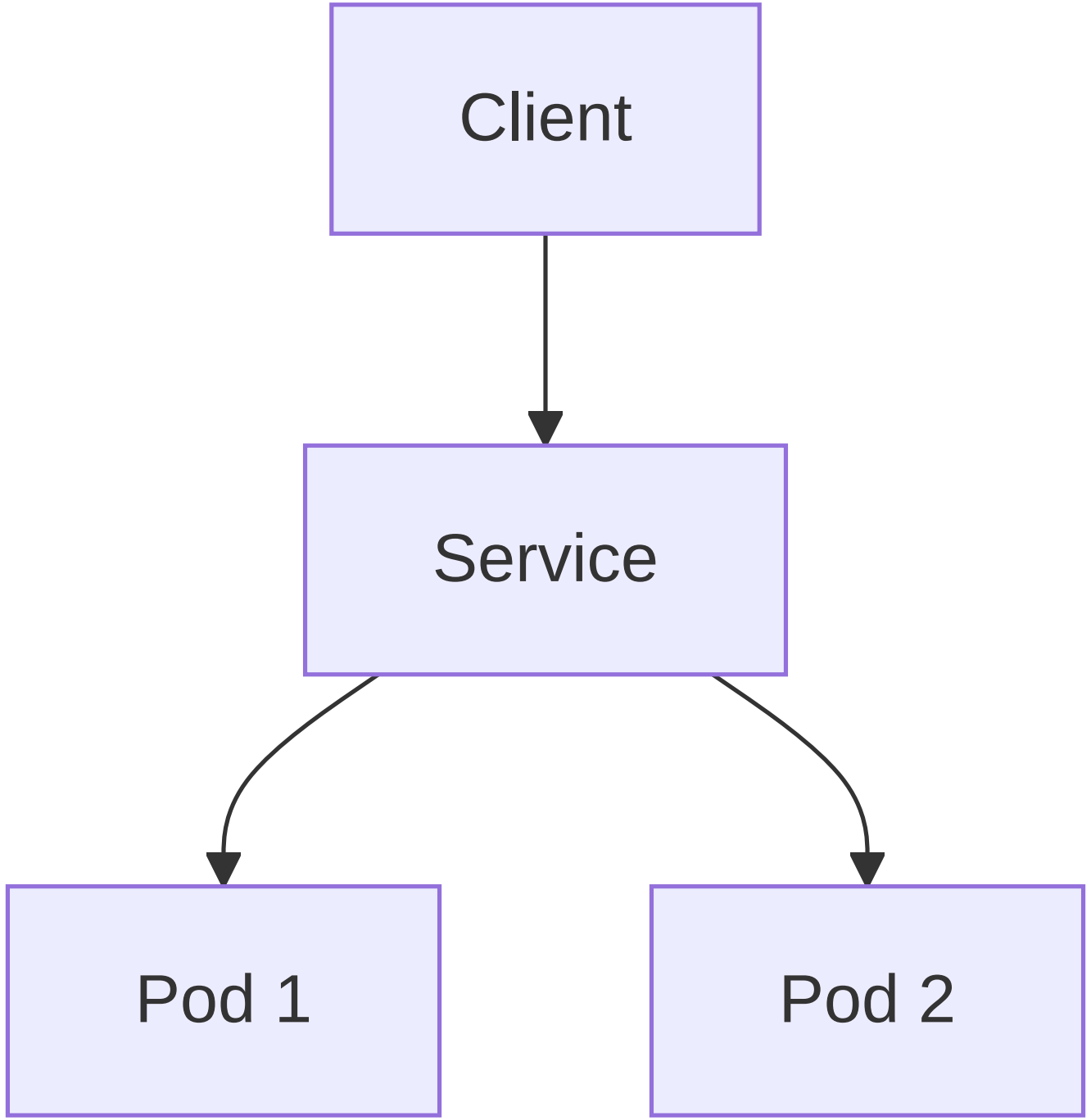
Networking & Service Discovery

Cluster Networking

- Pod-to-pod communication
- Service abstraction

Service Discovery

- Find and connect to services


Client
Service
Pod 1
Pod 2

Kubernetes vs. Alternatives

Other Orchestrators

- Docker Swarm
- Apache Mesos
- OpenShift

Kubernetes Advantages

- Large community
- Rich feature set
- Vendor-neutral

Cloud Providers & Managed Solutions

Managed Kubernetes

- GKE (Google)
- EKS (Amazon)
- AKS (Azure)
- Others

Industry Adoption

Who Uses Kubernetes?

- Enterprises: Google, Spotify, Adidas, etc.
- Startups & SMBs
- DevOps teams

Use Cases

- Web applications
- Big data & analytics
- CI/CD pipelines

Summary & Next Steps

Summary

- Kubernetes solves modern deployment challenges
- Powerful orchestration and automation
- Huge ecosystem and industry support

Next Steps

- Explore Kubernetes basics
- Hands-on with clusters
- Dive into core concepts

Thank You!

Questions?