

Low-Code Meets Public Code: Automating with Minimal Engineering

Simplifying software delivery in the era of automation



Anoosha Cherukuri

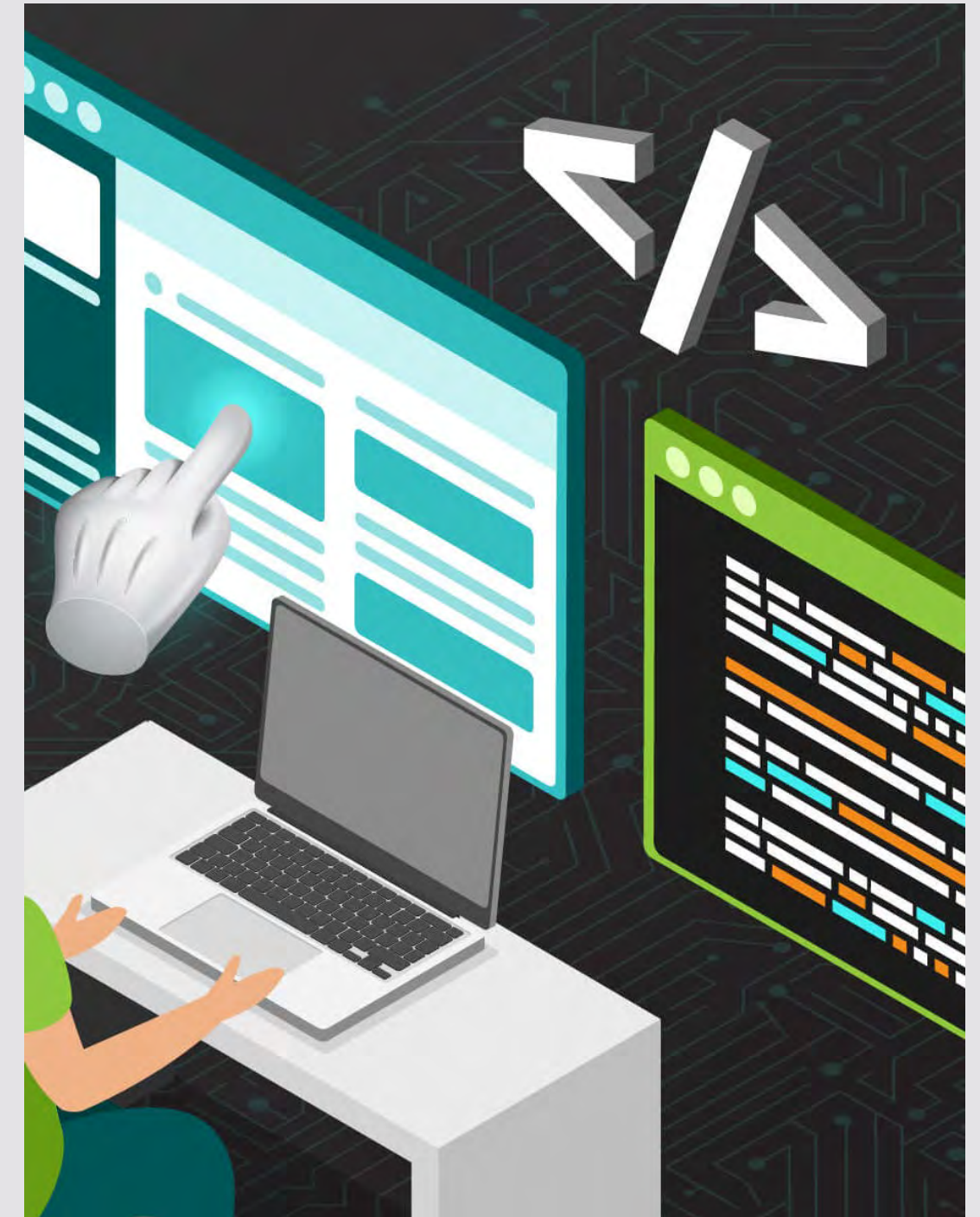
Table of Content

- Introduction
- Understanding Low-Code
- Public Code Principles
- Architectural Patterns
- Governance & Engineering Standards
- Integration in Practice
- Case Studies / Real-World Scenarios
- Challenges & Pitfalls
- Future Directions
- Conclusion
- Thank you

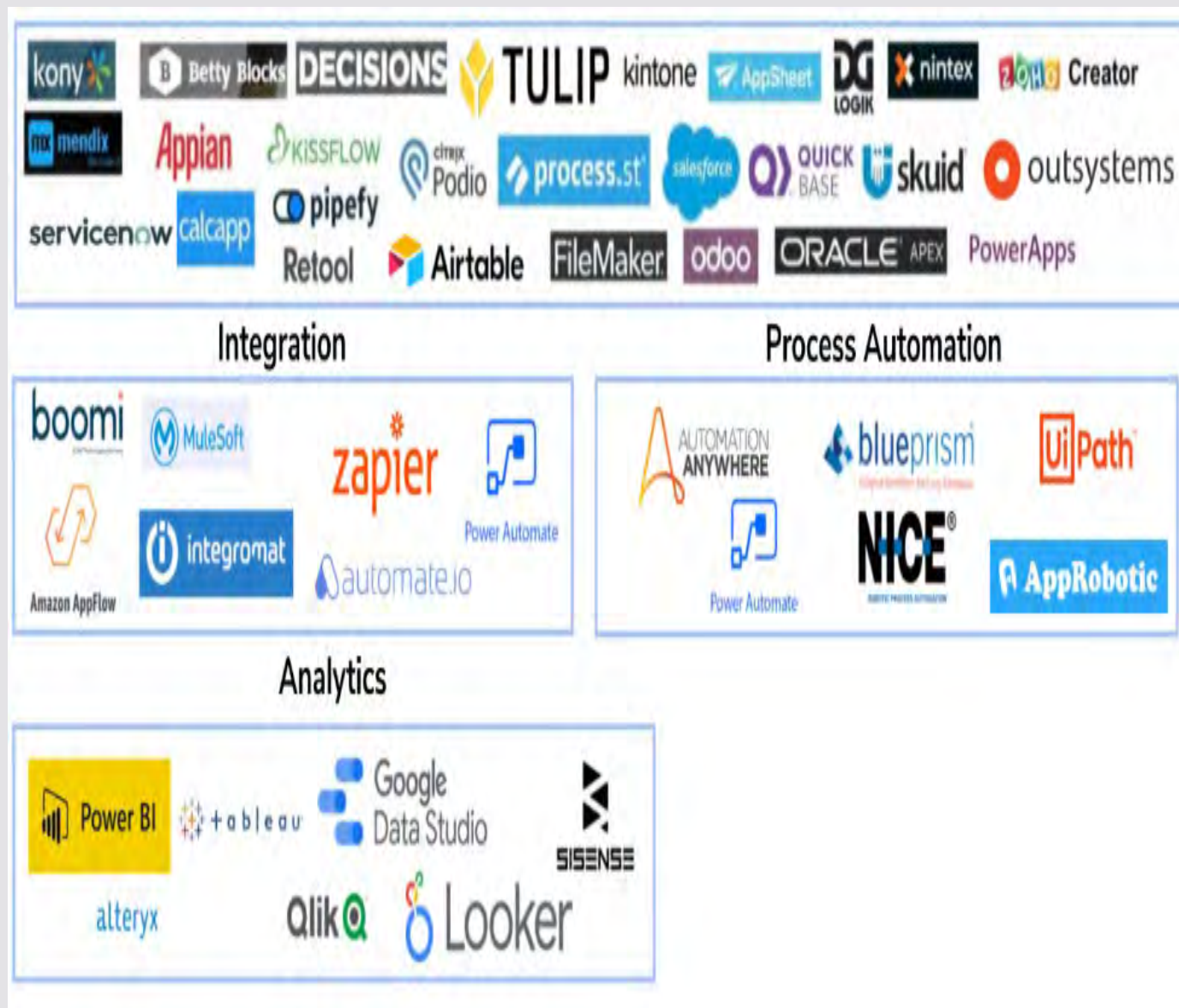


Introduction

- Traditional coding is often slow, costly, specialist-driven
- Demand for faster, adaptive digital solutions
- Low-Code accelerates delivery with visual tools
- Public Code ensures transparency & collaboration
- Together both low code and public code is scalable, compliant, efficient automation.



What is Low-Code?



- **A modern development approach** that uses visual, drag-and-drop interfaces instead of extensive manual coding.
- **Reduces technical complexity**, allowing faster application design, workflow automation, and system integration.
- **Bridges business and IT teams**, enabling both developers and non-technical users to actively participate in solution building.
- **Accelerates delivery cycles**, moving projects from concept to deployment in days or weeks instead of months.
- **Enhances scalability and adaptability**, making it easier to respond to evolving business needs and market demands.



What is Public Code?

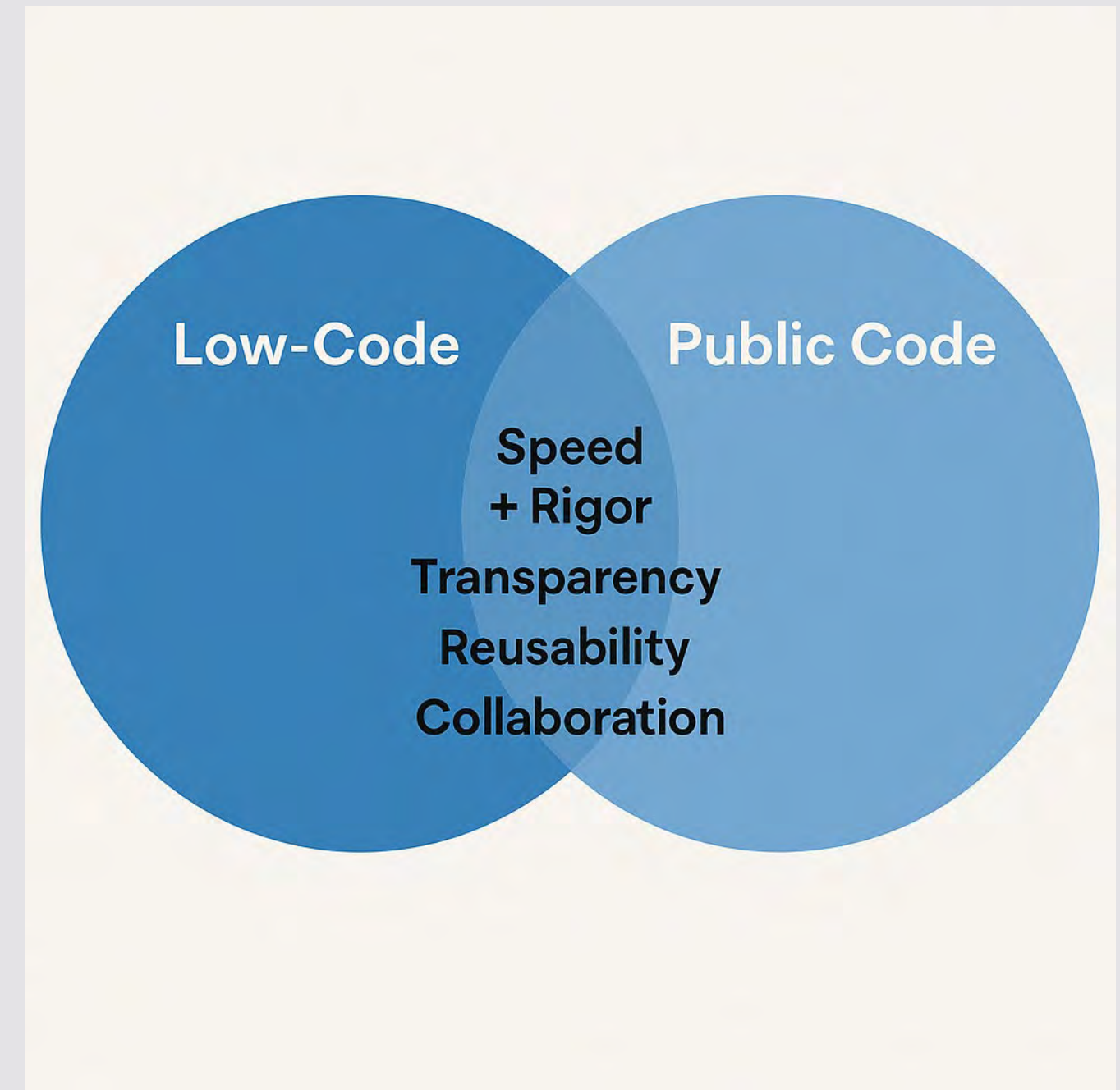
Definition: Public Code refers to software developed openly using transparent standards, shared repositories, and reusable components.

Core Principles:

- Openness → code is publicly accessible and reviewable
- Transparency → design and decisions are documented and auditable
- Reusability → code can be adapted across projects and organizations

Why It Matters:

- Ensures trust, compliance, and accountability in public sector and enterprise IT
- Reduces duplication of effort by enabling collaboration and knowledge sharing
- Supports long-term maintainability through community-driven development



Architectural Patterns

Hybrid Workflows: Low-code for orchestration, public code for core business logic.

Ensures speed with maintainability

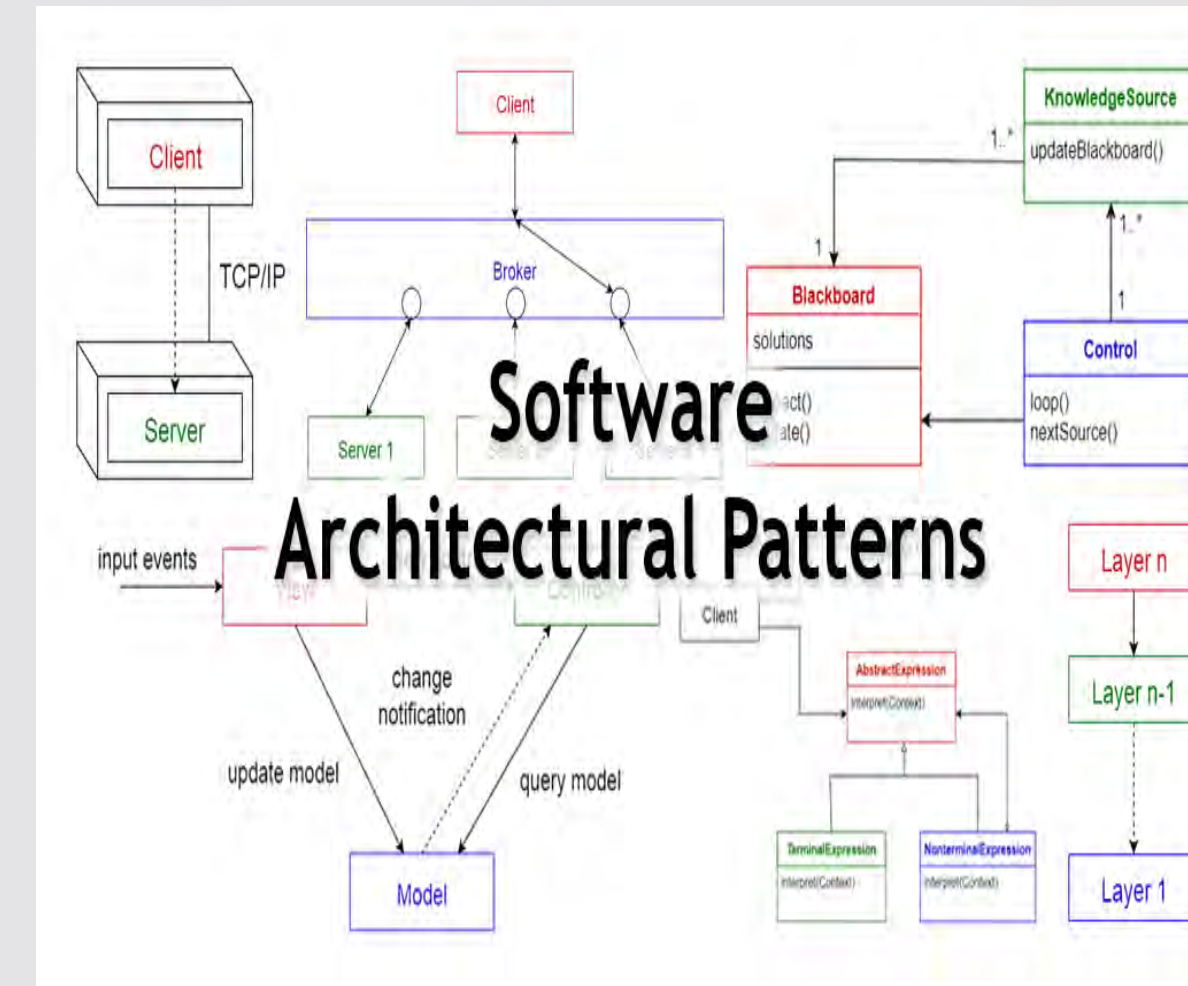
API-First Design: Use connectors and service APIs instead of direct database calls.

Promotes modularity and scalability

Event-Driven Pipelines: Trigger workflows through events and queues. Supports retries, resilience, and audit trails

CI/CD Integration: Treat low-code artifacts as versioned code. Enable automated testing, deployment, and rollback

Observability & Governance: Build in logging, tracing, and monitoring. Maintain compliance and transparency in production flows



Governance & Engineering Standards

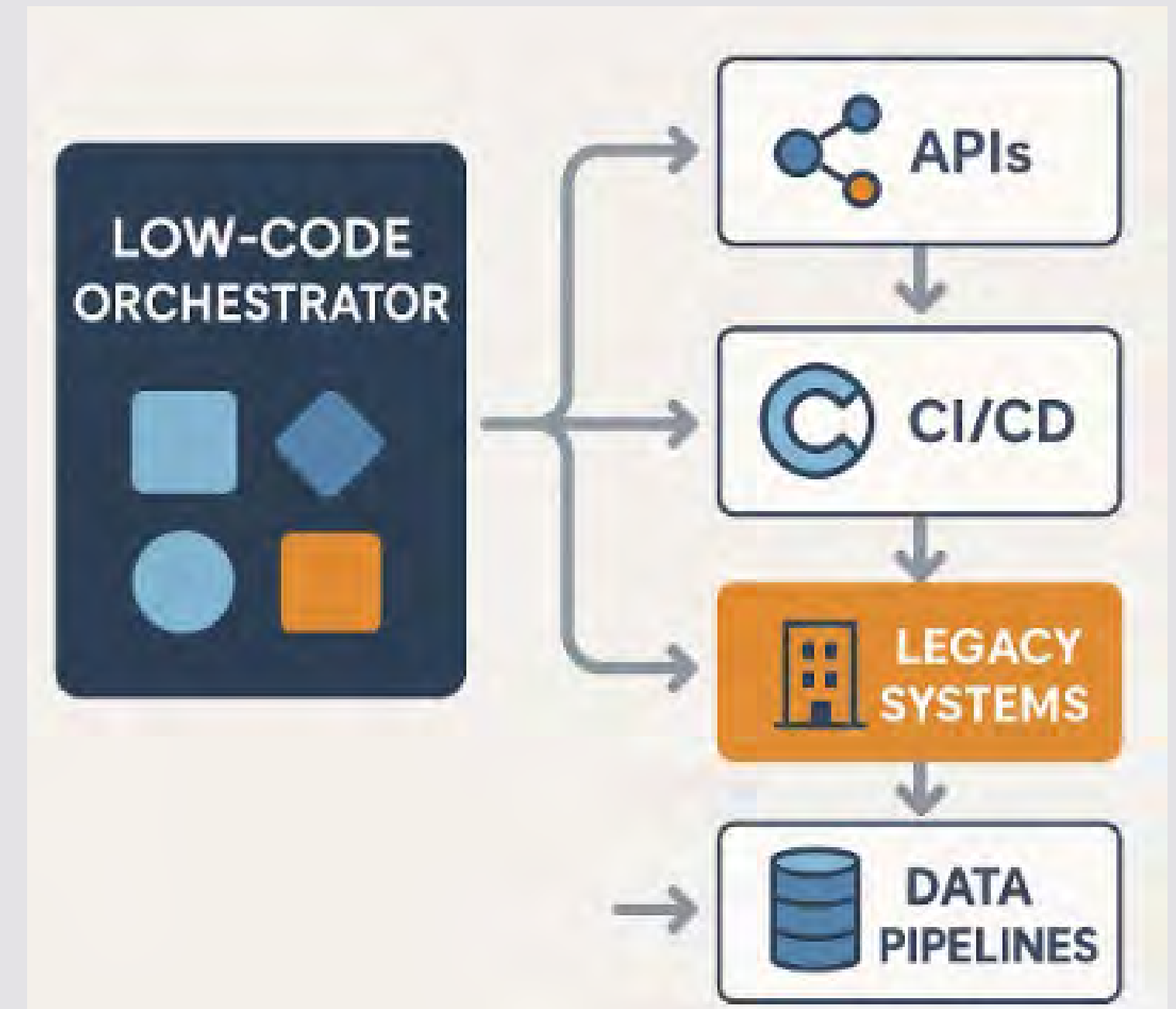


- **Version Control:** Track low-code flows, configs, and artifacts as code. Enable reviews, diffs, and rollback
- **Automated Testing:** Validate data quality, business logic, and integrations. Catch regressions early in pipelines
- **Security & Compliance:** Embed access controls, encryption, and audit trails. Ensure adherence to regulatory requirements
- **Reproducibility:** Guarantee consistent builds across environments
Support long-term auditability and maintainability



Integration in Practice

- **Data Pipelines & Automation:** Automate ingestion, transformation, and reporting with low-code workflows
- **DevOps & CI/CD Synergy:** Connect low-code apps to CI/CD pipelines for deployment and monitoring
- **Legacy Modernization:** Wrap legacy systems with APIs and orchestrate through low-code layers
- **Cross-Team Collaboration:** Enable non-developers to build flows while engineers ensure scalability and compliance



Case Studies / Real-World Scenarios

- **Government services:** Low-code workflows for permits, licensing, and citizen feedback
- **Enterprise IT:** Rapid prototyping of internal dashboards and automation scripts
- **Healthcare & public sector:** Compliance-driven workflows with audit-ready pipelines
- **Outcome:** Faster delivery, reduced engineering effort, and improved transparency



Challenges & Pitfalls

- **Scalability limits:** Low-code solutions may struggle with complex, high-volume workloads
- **Vendor lock-in:** Dependency on proprietary platforms can reduce flexibility
- **Governance drift:** Lack of versioning and oversight leads to shadow IT risks
- **Skill balance:** Aligning citizen developers with professional engineers



Future Directions

- **Citizen developers + engineers:** collaborative models for sustainable automation
- **Open-source low-code platforms:** reducing vendor lock-in, enhancing transparency
- **AI-assisted low-code:** intelligent code generation and pipeline optimization
- **Public sector adoption:** aligning low-code with digital government and compliance mandates



Conclusion

Low-code accelerates delivery, but must be paired with engineering rigor

Public code principles ensure transparency, compliance, and reusability

Hybrid approaches balance speed, scalability, and governance

Sustainable automation requires collaboration between citizen developers and engineers



The background features a complex, abstract network of glowing blue lines and dots, resembling a molecular structure or a data network. The dots are of varying sizes and brightness, and the lines connect them in a non-linear fashion. The overall color palette is a range of blues, from deep navy to bright cyan.

THANK YOU