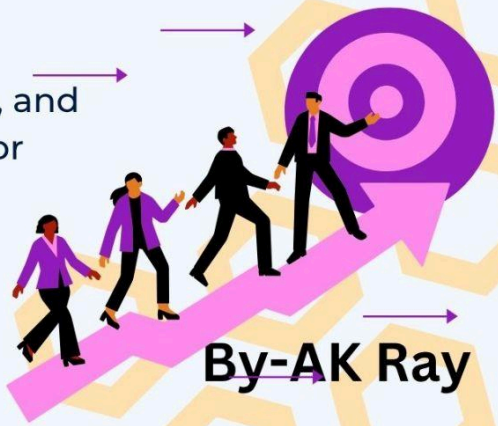
SYSTEM DESIGN & ARCHITECTURE

Design Principles, Patterns, and Real-World Architectures for Modern Systems



By-AK Ray

Message to Developers !!

Hey Techies,

Over the years, I've seen many talented developers build great code but struggle when it comes to *designing systems*. Writing code is an art — but designing a scalable, reliable, and maintainable system is *architecture*. It's what transforms a good developer into a great engineer.

In this book, I've curated the most essential **system design principles, patterns, and real-world concepts** that are not only *commonly asked in interviews* but also *crucial for building production-grade systems*. Each topic is simplified, structured, and supported with practical insights drawn from real projects — the kind of lessons that go beyond textbooks and straight into what industry experts use every day.

Whether you're preparing for a **System Design interview**, aiming to **improve your architectural thinking**, or simply **want to understand how large-scale systems are built**, this book is for you. Don't just read it — *walk through it, sketch the diagrams, imagine the data flow,* and see how each decision impacts scalability, performance, and reliability.

System design is not about memorizing answers — it's about understanding *why* and *how* things work together. Once you master that mindset, you'll design solutions that don't just work — they *last*.

Keep learning. Keep building.
And most importantly — keep *designing with intent*.

– Amitesh Kumar Ray

Who Is This Book For?

This book is for **every software professional and learner** who wants to design better systems.

If you're a **Software Engineer**, it will help you go beyond coding and understand how real systems are built, scaled, and maintained.

If you're a **Designer or Architect**, it will sharpen your ability to make informed design choices and balance scalability, performance, and simplicity.

If you're an **Engineering Manager**, it will give you a framework to guide your team in making sound architectural decisions.

And if you're an **Engineering Student**, it will bridge the gap between classroom concepts and real-world software design.

> **If you want to design a BETTER system, you must know what's inside this book.**

– Amitesh Kumar Ray

About the Author

Amitesh Kumar Ray is a seasoned **Software Engineer** with over a decade of experience in building scalable, high-performance systems. He currently serves as a **Lead SDE at M2P Fintech**, where he leads product design and architecture initiatives, guiding teams to create resilient and distributed enterprise systems.

With deep expertise in **Java, Spring Boot, Microservices, and System Architecture**, Amitesh has been instrumental in transforming large-scale applications into modern, scalable solutions. His practical, hands-on approach to engineering reflects years of experience designing systems from the ground up.

A passionate **author and mentor**, Amitesh has written **15+ programming books** on topics including Java, functional programming, microservices, and prompt engineering. He has conducted **100+ tech sessions and workshops**, helping developers bridge the gap between theory and real-world application.

He holds a **certification in Software Design and Architecture** from the **University of Alberta** and is currently **pursuing an 11-month Advanced AI and ML Certificate Program from IIT Madras (IITM)**, reflecting his commitment to continuous learning and innovation.

> *"True engineering lies not just in writing code, but in designing systems that evolve gracefully over time."*

Connect with him on LinkedIn to explore his latest work and insights on modern software design and architecture.

– Amitesh Kumar Ray

Chapter 1: Fundamentals of System Design

1.1 Fundamentals of System Design
1.2 Scalability
1.3 Availability
1.4 Reliability
1.5 Latency
1.6 Throughput
1.7 Performance vs Efficiency
1.8 Fault Tolerance
1.9 Resiliency and Recovery
1.10 CAP Theorem
1.11 ACID vs BASE
1.12 Durability and Consistency Models
1.13 Data Partitioning and Replication Basics

Chapter 2: Core Infrastructure Components

2.1 Load Balancer (L4/L7)
2.2 CDN (Content Delivery Network)
2.3 Proxy Server & Reverse Proxy
2.4 API Gateway
2.5 Message Queue (Kafka, RabbitMQ, SQS)
2.6 Rate Limiting and Throttling
2.7 Circuit Breaker & Bulkhead Pattern
2.8 Service Discovery (Eureka, Consul, Zookeeper)
2.9 Caching (Client, Server, Distributed)
2.10 Cache Strategies: Read-Through, Write-Through, Write-Back
2.11 Consistent Hashing
2.12 Heartbeats & Health Checks
2.13 Checksum and Data Validation
2.14 Consensus Algorithms (Raft, Paxos, ZAB)
2.15 Gossip Protocol
2.16 Bloom Filter
2.17 Leader Election
2.18 Distributed Locking

Chapter 3: Database and Storage Design

3.1 Databases (RDBMS, NoSQL)
3.2 SQL vs NoSQL
3.3 DB Indexing and Query Optimization
3.4 Database Sharding and Partitioning
3.5 Database Replication
3.6 Normalization vs Denormalization
3.7 Read Replica and Write Master Design
3.8 Transaction Management and Isolation Levels
3.9 Distributed Transactions
3.10 Data Modeling Techniques
3.11 Caching with Databases (Redis, Memcached)
3.12 Event Sourcing & CQRS (for complex data workflows)

Chapter 4: System Design Trade-offs

4.1 Vertical vs Horizontal Scaling
4.2 Strong vs Eventual Consistency
4.3 Stateful vs Stateless Design
4.4 Synchronous vs Asynchronous Communication
4.5 Batch vs Stream Processing
4.6 Long Polling vs WebSockets vs Server-Sent Events
4.7 Read-Through vs Write-Through vs Write-Back Caching
4.8 REST vs RPC (gRPC, Thrift)
4.9 SQL vs NoSQL Trade-offs
4.10 Normalization vs Denormalization
4.11 TCP vs UDP
4.12 Push vs Pull Architecture
4.13 Throughput vs Latency Trade-off
4.14 Cost vs Performance Optimization

Chapter 5: Architectural Patterns

5.1 Client–Server Architecture
5.2 Layered (N-tier) Architecture
5.3 Microservices Architecture
5.4 Monolithic Architecture (and evolution to microservices)
5.5 Event-Driven Architecture
5.6 Serverless Architecture
5.7 Peer-to-Peer Architecture
5.8 Service-Oriented Architecture (SOA)
5.9 Hexagonal (Ports and Adapters) Architecture
5.10 Domain-Driven Design (DDD)
5.11 CQRS and Event Sourcing Pattern
5.12 Saga Pattern (Transaction Management in Microservices)
5.13 Bulkhead & Circuit Breaker Pattern
5.14 Sidecar and Ambassador Patterns
5.15 Strangler Fig Pattern (for legacy migration)
5.16 API Gateway Pattern

Chapter 6: Distributed Systems & Communication

6.1 Fundamentals of Distributed Systems
6.2 Distributed Caching
6.3 Consensus & Coordination (Zookeeper, etcd)
6.4 Gossip Protocol in Node Communication
6.5 Consistent Hashing for Node Distribution
6.6 Quorum Reads/Writes
6.7 Idempotency and Retry Mechanisms
6.8 Leader Election and Replication
6.9 Eventual Consistency in Distributed Systems
6.10 Heartbeats, Health Checks, Failover Mechanisms
6.11 Split-Brain Handling
6.12 Distributed Logging and Monitoring
6.13 Message Ordering and Exactly-once Semantics

Chapter 7: Performance, Reliability, and Observability

7.1 Fault Detection and Recovery
7.2 Redundancy and Failover Mechanisms
7.3 Circuit Breaker and Bulkhead Design
7.4 Graceful Degradation
7.5 Autoscaling (Reactive Scaling)
7.6 Load Shedding and Backpressure
7.7 Observability (Logs, Metrics, Traces)
7.8 Monitoring Stack (Prometheus, Grafana, ELK, OpenTelemetry)
7.9 Alerting and Incident Management
7.10 SLO, SLI, and SLA
7.11 Blue-Green and Canary Deployment Patterns
7.12 Capacity Planning and Cost Optimization

Chapter 8: Implementation in Modern Tech Stack

8.1 Designing Scalable Systems with Java and Spring Boot
8.2 Microservices with Spring Cloud (Eureka, Config Server, Gateway, Feign)
8.3 Resilience with Resilience4j and Hystrix
8.4 Containerization with Docker
8.5 Kubernetes for Orchestration and Autoscaling
8.6 Service Mesh (Istio, Linkerd)
8.7 CI/CD Pipeline (Jenkins, GitHub Actions, ArgoCD)
8.8 SonarQube for Code Quality and Static Analysis
8.9 Distributed Tracing (Sleuth, Zipkin, Jaeger)
8.10 Security and Authentication (OAuth2, JWT, mTLS)
8.11 Infrastructure as Code (Terraform, Helm)

Chapter 9: Real-World System Design Examples

9.1 Design a URL Shortener
9.2 Design a Social Media Feed System
9.3 Design a Ride-Sharing Platform (Uber)
9.4 Design a Chat Application (WhatsApp)
9.5 Design a Video Streaming Platform (YouTube/Netflix)
9.6 Design an E-Commerce System
9.7 Design a Payment Gateway
9.8 Design a Notification/Alert System
9.9 Design a Logging and Monitoring System
9.10 Design a Distributed File Storage (Google Drive / Dropbox Style)

A Note at the End

You've reached the final page — but this isn't the end. It's the beginning of your journey as a system thinker.

System design isn't about memorizing patterns — it's about understanding *why* things work and *how* to make them better. Every line of code, every design choice, and every trade-off teaches you something deeper about building systems that last.

Keep learning. Keep experimenting. Keep designing.
Because true engineers don't just build systems — they **create impact**.

> *"Don't just aim to be a better coder. Aim to be a better designer of solutions."*

– Amitesh Kumar Ray