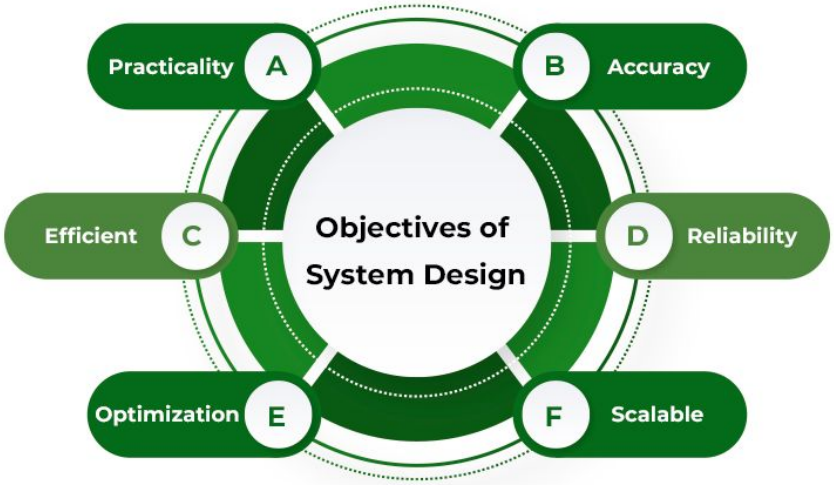
Mastering System Design

Section 11: The System Design Blueprint

The 4-Step System Design Approach: From Problem Definition to Final Solution

What is System Design?

- Building scalable, maintainable systems
- Balancing trade-offs between performance, cost, and complexity
- The importance of clear architecture and thoughtful planning



The 4-Step System Design Process

- Step 1: Understanding the Problem & Defining the Scope
  - Functional Requirements: Define the core features (e.g., URL shortening, redirection).
  - Non-Functional Requirements: Consider performance, scalability, reliability, security.
  - Constraints: Account for time, budget, regulatory, and technical limitations.
- Step 2: Estimating Scale & Identifying Bottlenecks
  - Estimating Traffic: Analyze peak load, traffic patterns, and user growth.
  - Identifying Bottlenecks: Pinpoint critical components (e.g., database, CPU, network) that may face performance issues as traffic increases.
  - Capacity Planning: Estimate storage, bandwidth, and compute requirements to handle growth.
- Step 3: High-Level Design: Services, APIs & Communication
  - Core Services: Break down the system into essential services (e.g., URL shortener, analytics).
  - API Design: Define public-facing endpoints (e.g., POST /shorten, GET /redirect).
  - Communication Patterns: Choose synchronous vs. asynchronous (e.g., REST APIs, WebSockets).
  - Service Interaction: Define how services communicate (e.g., via APIs, message queues).
- Step 4: Making Tech & Infra Decisions Strategically
  - Tech Stack Decisions: Choose between SQL/NoSQL, data stores, and caching solutions (e.g., Redis).
  - Scalability & Availability: Implement load balancing, autoscaling, and replication.
  - Performance Considerations: Focus on latency, throughput, and caching hot data.
  - Trade-offs: Weigh cost vs. performance, simplicity vs. complexity in tech choices (e.g., Redis for caching vs. traditional databases).

Conclusion - The Blueprint for Success

- Always start with a clear understanding of requirements and constraints
- Use scale and bottleneck analysis to guide architectural decisions
- Create a high-level design that balances performance, cost, and complexity
- Make informed infrastructure and tech choices to ensure scalability and availability