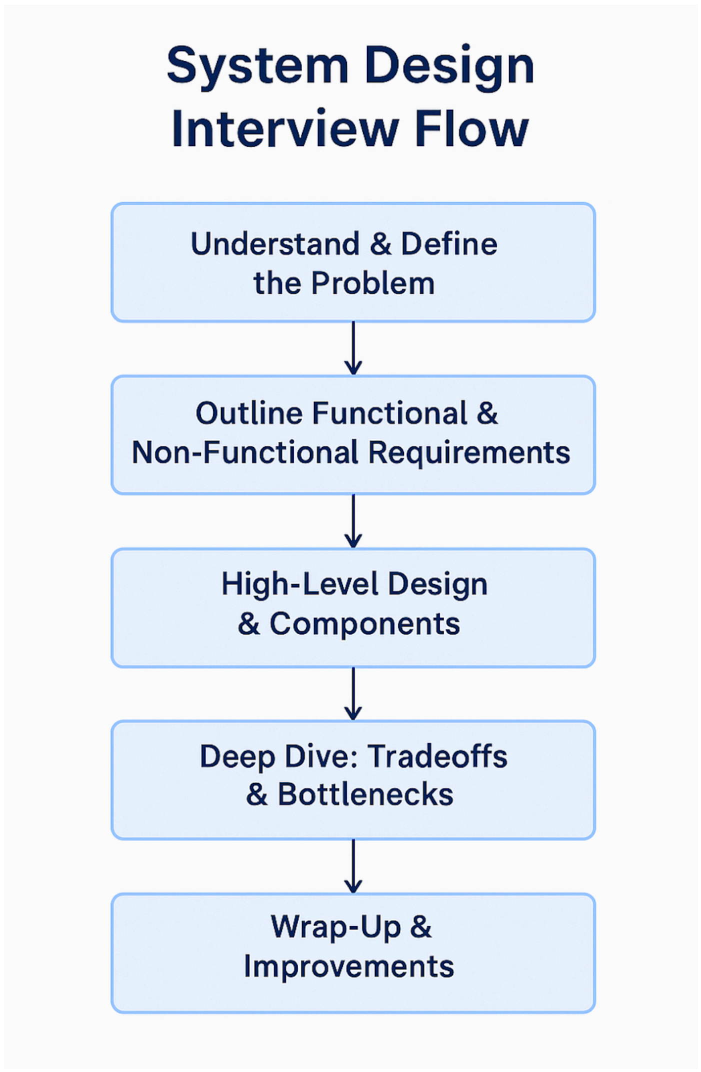
System Design Prep - By Kartikey (Meta | Google | Amazon)


System Design Interview Flow
Understand & Define the Problem
Outline Functional & Non-Functional Requirements
High-Level Design & Components
Deep Dive: Tradeoffs & Bottlenecks
Wrap-Up & Improvements

📘 System Design Cheat Sheet

🚀 *Real-Time Systems*

• Messaging System (WhatsApp): Delivery guarantees, offline support, scalable infra
• Notification System (Push/SMS/Email): Token mgmt, retries, multi-channel delivery
• Ride Sharing (Uber/Lyft): ETA, geo-matchmaking, surge logic
• Uber Matching: Rider-driver matching, surge pricing, location updates
• Webhook Callback: Reliable delivery, retries, exactly-once guarantees

💰 *Financial & Commerce Systems*

• Payment System (Stripe): Idempotent flows, retry handling, PCI compliance
• Ticket Booking System (BookMyShow): Atomic seat locks, payment consistency, race condition handling
• E-Commerce Platform (Amazon): Catalog, cart, order mgmt, scalability

• Ad Click Aggregator (Google Ads): High-throughput ingestion, deduplication, analytics
• Ad Click Analytics: Windowed aggregation, Kafka ingestion, low-latency queries

🔍 *Discovery & Social Systems*

• NewsFeed System (Instagram): Timeline construction, fanout, ranking
• Trending Hashtags (Twitter): Sliding window top-K, approx. counting, real-time updates
• Search Autocomplete (Google): Trie lookups, ranking, typo-tolerance
• Voting System (Reddit): Idempotent votes, fraud prevention, tallying
• Coding Platform (LeetCode): Problem mgmt, submission flow, eval infra

📦 *Infra & Platform Components*

• Distributed Cache (Redis/Memcached): Fast R/W, cache invalidation, eviction strategies
• Rate Limiter (API Gateway): Token/leaky bucket, TTL, distributed enforcement
• Amazon Lockers (IoT Logistics): Order to locker mapping, real-time status, auth
• Web Crawler (Googlebot): URL queuing, deduplication, rate limiting
• Task Scheduler (Cron): Recurring jobs, retries, cron parsing
• Task Management Tool (Jira): CRUD, background jobs, notifications
• Log Analysis (ELK Stack): Real-time ingestion, indexing, anomaly detection
• Image Hosting (Imgur/S3): Scalable storage, deduplication, CDN integration
• URL Shortener (Bitly): Unique code generation, high-QPS redirect, link analytics

---

📚 Mock Interview Flow + Questions (Deep Dive)

⏱️ 0–5 mins: Understand & Define the Problem

- Clarify Requirements: What does the system do? What's out-of-scope?
- Target Users & Scale: Is it for 1M or 100M users? Concurrency matters.
- Access Patterns: Read-heavy or write-heavy? Real-time vs batch?
- Example: For a ride-sharing app, clarify scope (payments? ratings? live tracking?)

⏱️ 5–15 mins: Functional & Non-Functional Requirements

- Functional Examples (for Uber): Request a ride, Match driver, Track ride, Pay, Rate
- Non-Functional Goals:
    - Latency: <100ms for critical APIs
    - Availability: 99.99%
    - Durability: No message loss
    - Scalability: Handle spikes with horizontal scale
- Capacity Planning:
    - 100M DAUs, 500k QPS, 80:20 read/write ratio

⏱️ 15–30 mins: High-Level Design & Components

- Sketch Architecture:
    - API Gateway → Service Layer → Cache → DB → Queue
    - Modular services: MatchingService, UserService, BillingService
- Database:
    - SQL: PostgreSQL/MySQL (for consistency)
    - NoSQL: MongoDB/Cassandra (for scale/flexibility)
- Caching:
    - Redis for hot reads, TTL strategies, invalidation
- Queues:
    - Kafka/SQS for async flows: billing, notifications
- API Design:
    - RESTful/GraphQL, idempotent POSTs, versioning

⏱️ 30–40 mins: Deep Dive — Tradeoffs, Bottlenecks, Scaling

- Data Consistency Models:
    - Strong (ticketing), Eventual (feeds), Causal (messaging)
- CAP Tradeoffs:
    - Choose CA, CP, or AP based on product goals
- Scaling:
    - Sharding by user/location, leader-follower replication
    - Load balancers with health checks
- Failures:
    - Circuit breakers, retry/backoff, dead letter queues
- Observability:
    - Logging (structured), metrics, alerts

⏱️ 40–45 mins: Wrap-Up & Future Enhancements

- Summarize the full system workflow
- Identify weak links and improvement paths
- Scope V2 features:
    - Multi-region active-active
    - Analytics pipelines
    - GDPR & PCI compliance

🧠 Mental Frameworks (Advanced)

🧩 Design Thinking

- Always start with user experience and intent
- APIs define contracts, versioning ensures evolution

⚙️ Scalability & Performance

- Design for 10x scale even if MVP
- Use event-driven async processing where possible
- Latency budgets per component (e.g. 30ms DB, 10ms cache)
- Optimize read-heavy vs write-heavy differently

🧠 System Reliability

- Think in terms of failure paths
- Add retries, rate limits, graceful degradation
- Secure every component (encryption, RBAC, audit logs)

💰 Cost + Maintainability

- Monitor cloud usage: storage, compute, transfer
- Favor managed services if ops is not differentiating

📅 7-Day System Design Mastery Plan

Day 1: URL Shortener + Ticket Booking
Goal: ID generation, atomic ops, consistency guarantees
Day 2: Messaging System + Payment Infra
Goal: Delivery guarantees, retries, PCI-compliance
Day 3: Rate Limiter + Cache System
Goal: High QPS handling, token buckets, eviction logic
Day 4: Newsfeed + Voting System
Goal: Timeline construction, ranking, idempotency
Day 5: E-Commerce Platform + Search Autocomplete
Goal: DB schema design, search latency reduction
Day 6: Uber Matching + Notification System
Goal: Geo-matching, mobile token handling
Day 7: Mock Interview + Design Review
Goal: Simulate a full 45-min interview, get feedback

🔁 System Design Tradeoffs Summary

Use this reference during interviews or system evaluations to justify architecture decisions.

Tradeoff Area	Option A	Option B
Consistency	Strong (ACID, strict correctness)	Eventual (faster, scalable)
Storage	SQL (structured, transactional)	NoSQL (flexible, scalable)
Communication	Synchronous (instant feedback)	Asynchronous (decoupled, resilient)
Latency Handling	Push (instant updates)	Pull (scalable, client-driven)
Scaling	Vertical (bigger machine)	Horizontal (more machines, sharding)
Data Access	Read-optimized (cache, replicas)	Write-optimized (batch, queues)
State Management	Centralized (single source of truth)	Distributed (high availability)
Queue Processing	At-most-once (fast)	At-least-once / Exactly-once (safe, retry logic)
Caching	Aggressive (low latency)	Minimal (high accuracy)
API Design	Monolith (easy to start)	Microservices (scalable, modular)