8 System Design patterns that will help you solve 99% of SD interview questions

Chandra Shekhar Joshi
Engineering career coach

01

Most candidates fail SD interviews not because they don’t know SD concepts, but because they don’t have a mental library of proven real world design patterns to pull from.

#1

Event driven architecture for multiple operations

Where to use:
When a single action needs to trigger multiple, loosely coupled follow-up operations.

How to use:

- Publish an event to a broker (Kafka, SNS, RabbitMQ)
- Have independent consumers process their part asynchronously
- Decouple producers from consumers to improve scalability and maintainability

Examples:

- E-commerce checkout → payment processing, inventory update, email confirmation
- Social media post → notify followers, update timelines, send push alerts

#2

Prepare upfront for heavy, complex reads

Where to use:

- Read-heavy workloads with thousands of TPS
- Expensive computations that would cause high latency if done on-demand

How to use:

- Precompute and store results in caches or materialized views
- Use background jobs or streaming pipelines to refresh data
- Trade some freshness for massive performance gains

Examples:

- Facebook news feed
- Twitter tweet delivery
- E-commerce product recommendations
- Search results pages
- Financial trading dashboards
- Ride-sharing ETA calculations

#3

Avoid distributed transactions (if possible)

Where to use:
When multi-step operations span multiple services or databases, but strict atomicity isn’t a hard business requirement.

How to use:

- Sequence operations so each step can commit independently
- Use clever UX + Change Data Capture (CDC) to sync state
- Apply the Transactional Outbox pattern for RDBMS-to-RDBMS sync
- Use eventual consistency when strong consistency isn’t essential

Examples:

- E-commerce product catalog updates
- Financial transaction processing with ledger reconciliation
- Customer profile data sync
- Order processing in high-volume system

#4

Chunk large files

Where to use:
When uploading, processing, or distributing files too large for a single transfer or where network reliability is a concern.

How to use:
- Split files into smaller chunks
- Upload/process chunks independently (parallel if possible)
- Reassemble on the server or client after processing

Examples:
- Video streaming (Netflix, YouTube)
- Cloud storage uploads (Google Drive, Dropbox)
- Peer-to-peer sharing (BitTorrent)

#5

Prevent race conditions in distributed systems

Where to use:
When multiple processes or services update the same resource concurrently, risking inconsistent or incorrect results.

How to use:

- Use Redis atomic operations for speed and simplicity
- If Redis is not an option, apply RDBMS row-level locking or equivalent concurrency controls
- Be ready to discuss sharding and scaling implications when using RDBMS

Examples:

- Preventing double booking in ticketing systems
- Maintaining accurate inventory in flash sales
- Avoiding duplicate payment processing
- Managing concurrent leaderboard updates

#6

Cache aside (lazy loading)

Where to use:
Read-heavy systems where data freshness can tolerate slight delays.

How to use:

- App reads from cache first
- On a miss → fetch from DB → populate cache with TTL
- Use eviction and refresh strategies to control memory usage

Examples:

- Product detail pages
- User profiles
- Game leaderboard reads

#7

Idempotency & de-duplication

Where to use:
When an operation might be triggered multiple times (e.g., retries, network glitches, client bugs) and must be processed exactly once.

How to use:

- Generate idempotency keys per request
- Maintain processed logs/tables to prevent reprocessing
- Apply deduplication windows for event streams

Examples:

- Preventing double charges in payments
- Avoiding duplicate orders in e-commerce
- Processing webhook events from Stripe, PayPal, or GitHub exactly once

#8

Circuit breakers & exponential backoff (with jitter)

Where to use:
When calling slow, unreliable, or overloaded dependencies and you need to prevent cascading failures.

How to use:

- Generate idempotency keys per request
- Maintain processed logs/tables to prevent reprocessing
- Apply deduplication windows for event streams

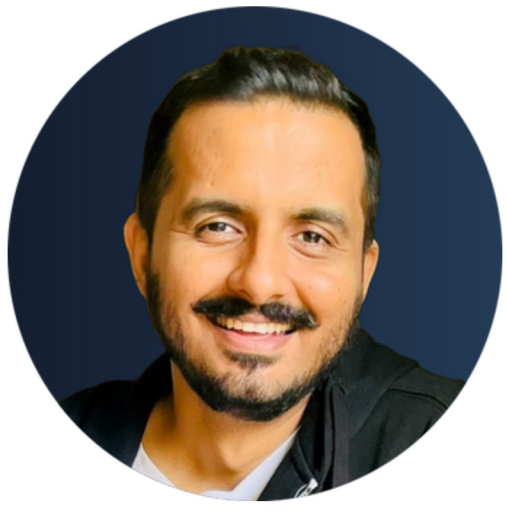
Examples:

- Handling payment gateway downtime
- Calling ML inference APIs under load
- Making requests to rate-limited 3rd-party APIs

That’s a wrap

Here is the list of real world design pattern

1. Event-driven architecture for multiple operations
2. Prepare upfront for heavy, complex reads
3. Avoid distributed transactions (if possible)
4. Chunk large files
5. Prevent race conditions in distributed systems
6. Cache-aside (lazy loading)
7. Idempotency and de-duplication
8. Circuit breakers and exponential backoff (with jitter)



Need help with mid-senior level SDE, EM interview prep?

DM me COACH