Q1. Design a system to process billions of events per day.

I would start by clarifying requirements such as data volume, latency, and SLAs. I would use Kafka for ingestion to handle high throughput, Spark Structured Streaming for scalable processing, and store data in a data lake using Parquet for efficient analytics. The system scales horizontally, supports replay, and separates compute from storage to control cost.

Q2. Design a real-time analytics pipeline.

I would ingest events through Kafka, process them in near real time using Spark Structured Streaming, apply aggregations and enrichments, and store results in a low-latency warehouse or serving layer. I would define freshness SLAs and monitor lag to ensure timely dashboards.

Q3. Design a batch processing data pipeline.

I would schedule batch jobs using an orchestrator, read data from distributed storage, process it using Spark for parallelism, validate data quality, and load curated tables for reporting. Batch pipelines prioritize correctness and cost efficiency over latency.

Q4. Design a Lambda architecture.

I would separate the system into a batch layer for historical accuracy and a speed layer for real-time processing. The batch layer recomputes full datasets periodically, while the streaming layer provides low-latency views.

Q5. Design a Kappa architecture.

I would use a single streaming pipeline where all data flows through Kafka. Both real-time and historical views are built by replaying events from the log, simplifying maintenance.

Q6. Design an event ingestion system.

I would expose SDKs or APIs for producers, validate schemas at ingestion, and publish events to partitioned Kafka topics to enable horizontal scalability.

Q7. Design a clickstream data pipeline.

I would collect client events, ingest them through Kafka, enrich and clean them using streaming jobs, store them in partitioned Parquet files, and expose them via analytics tools.

Q8. Design a log aggregation system.

I would collect logs using agents, stream them through Kafka, process and index them for search, alerting, and long-term storage.

Q9. Design a metrics collection system.

I would collect lightweight metrics, aggregate them in streaming pipelines, and store them in a time-series optimized datastore for monitoring and alerts.

Q10. Design an ETL system for large-scale data.

I would extract data from multiple sources, transform it using distributed compute, validate schema and quality, and load it into analytical storage optimized for queries.

Q11. Design a Kafka-based ingestion pipeline.

I would partition topics by key, use consumer groups for parallelism, and ensure idempotent or transactional writes downstream.

Q12. Design a low-latency streaming system.

I would minimize transformations, avoid unnecessary shuffles, use efficient serialization, and keep state in memory where possible.

Q13. Design a system to handle late-arriving data.

I would use event-time processing with watermarking, allowing late data within defined thresholds and handling extreme lateness via backfills.

Q14. Design event-time based processing.

I would rely on event timestamps rather than processing time to ensure accurate aggregations, especially when data arrives out of order.

Q15. Design a system for out-of-order events.

I would buffer and reorder events within a bounded window and rely on watermarking to finalize results.

Q16. Design a backfill and replay mechanism.

I would replay historical data from Kafka offsets or stored files and reprocess it using the same pipeline logic for consistency.

Q17. Design a dead-letter queue strategy.

I would route invalid or failed records to a separate topic, store them for analysis, and enable controlled reprocessing.

Q18. Design exactly-once processing.

I would use idempotent producers, checkpointing, and transactional sinks to ensure records are processed once even during failures.

Q19. Design at-least-once processing.

I would allow retries and rely on downstream deduplication to ensure no data loss.

Q20. Design a streaming deduplication system.

I would assign unique event IDs and maintain state to detect and drop duplicates.

Q21. Design a high-throughput ingestion system.

I would scale horizontally using partitions, stateless consumers, and backpressure handling.

Q22. Design a multi-producer ingestion platform.

I would enforce data contracts, quotas, and isolation to protect the platform from noisy producers.

Q23. Design a schema-evolution friendly pipeline.

I would use schema registries and enforce backward-compatible schema changes.

Q24. Design a fault-tolerant streaming pipeline.

I would implement retries, checkpointing, monitoring, and automated recovery mechanisms.

Q25. Design a scalable data ingestion framework.

I would build reusable, configuration-driven ingestion components that can scale independently and support multiple data sources.