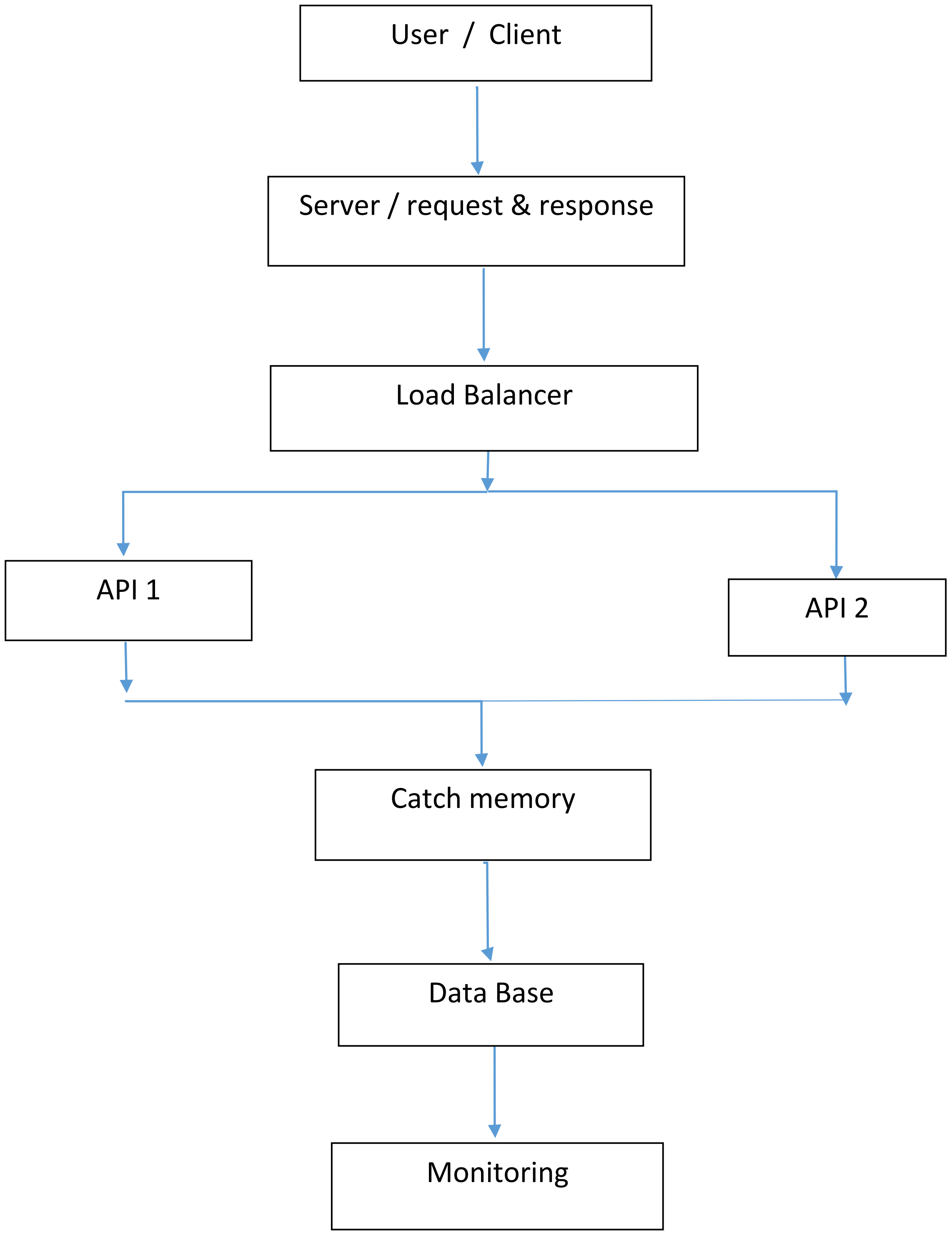
Task2: Concurrent System Design



Key Components

Component	Description
User / Client	The users or external systems that send requests to the optimization backend.
Server (Request & Response Layer)	The entry point that receives client requests, processes them, and prepares responses.
Load Balancer	It distributes incoming API requests across multiple server instances to ensure no single server gets overwhelmed.
API Servers (API 1, API 2, etc.)	These servers handle the business logic, optimization algorithms, and return optimized responses.
Cache Memory (e.g., Redis or Memcached)	A fast-access memory storage that saves frequent optimization results to serve repeated queries quickly.
Database (SQL/NoSQL)	A robust storage system that keeps persistent records such as container moves, user queries, and optimization history.
Monitoring System	Constantly checks system health, server loads, request failures, and other metrics to trigger alerts if anything goes wrong.

Handling Failures (Humanized Points)

1. If there is too much traffic, the load balancer automatically spreads the load across servers. This keeps the system smooth even during rush hours.
2. In case an API server crashes, monitoring tools quickly detect it and shift the user requests to other healthy servers without any manual effort.
3. If the database becomes slow or unreachable, the system smartly uses cached results from fast memory storage like Redis. This prevents users from experiencing delays.
4. If the cache memory itself fails, the system doesn't crash. Instead, it switches to pulling fresh data from the database, maintaining the service.
5. Whenever there's a sudden increase in users or requests, the backend auto-scales itself by creating more API instances to meet the demand.
6. If the entire backend faces issues, the monitoring system triggers emergency alerts. Meanwhile, the system tries to serve users in a basic, limited mode so that critical operations are still possible.