20 System Design Terms Every Developer Must Know

Master these essential concepts to level up your engineering skills!

Scalability

What is it?

The capability of a system to handle growth in users, data, or traffic

Why it matters

Prevents crashes during growth and ensures consistent performance as demand increases

Types

- Vertical (scale up)
- Horizontal (scale out)

Availability

Definition

Measure of system uptime and accessibility, often expressed as "five nines" (99.999%)

Five Nines

99.999% uptime means only 5.26 minutes of downtime per year

Achieved Through

Redundancy, failover mechanisms, and proper monitoring

Reliability

Consistent Performance

System performs as expected without failures

Measured By

Mean Time Between Failures (MTBF)

Building Blocks

Redundancy, testing, and error handling

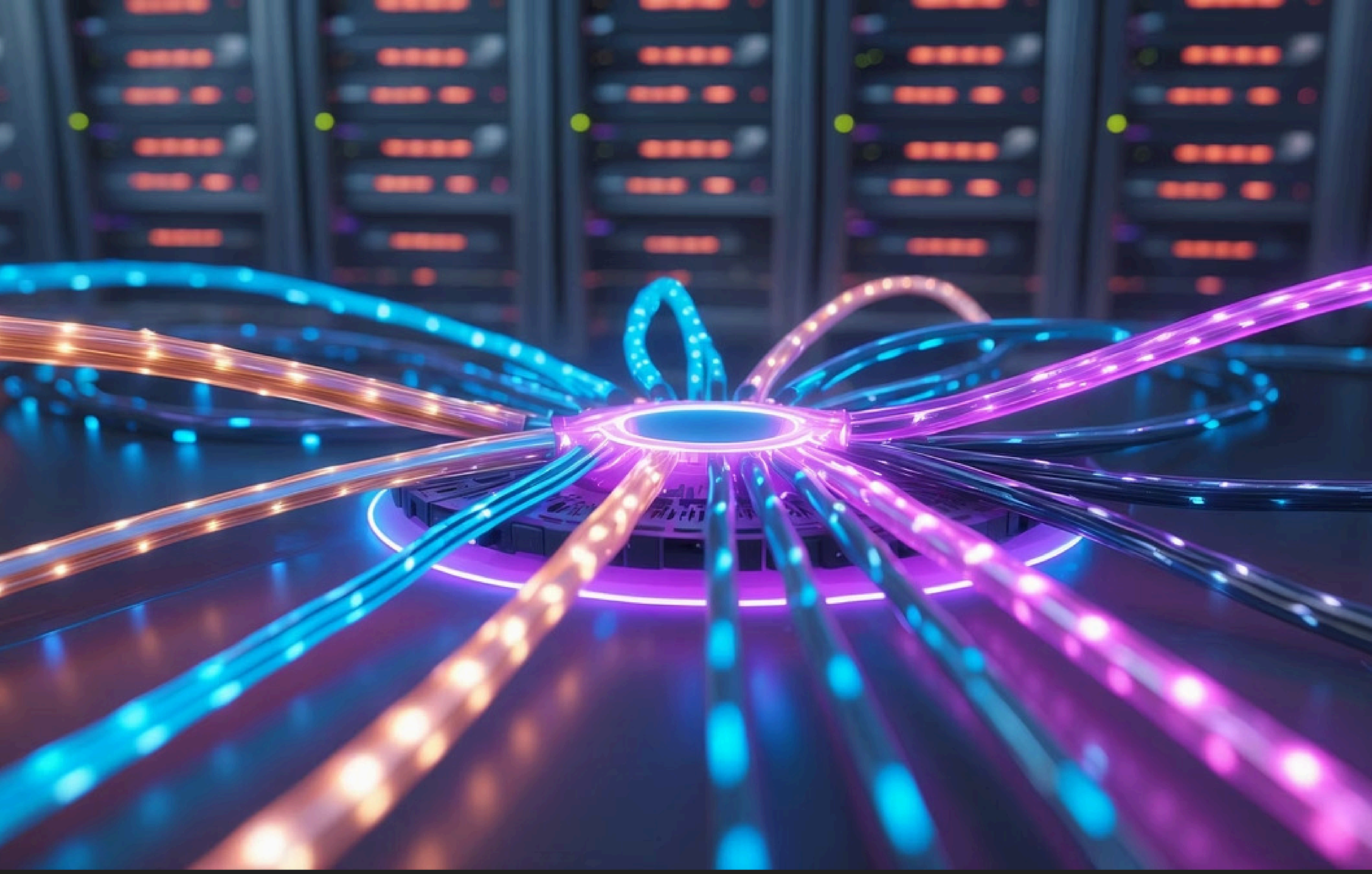
Latency

Request sent

Time delay

Response received

Time delay between request and response in a system, typically measured in milliseconds



Throughput

Definition

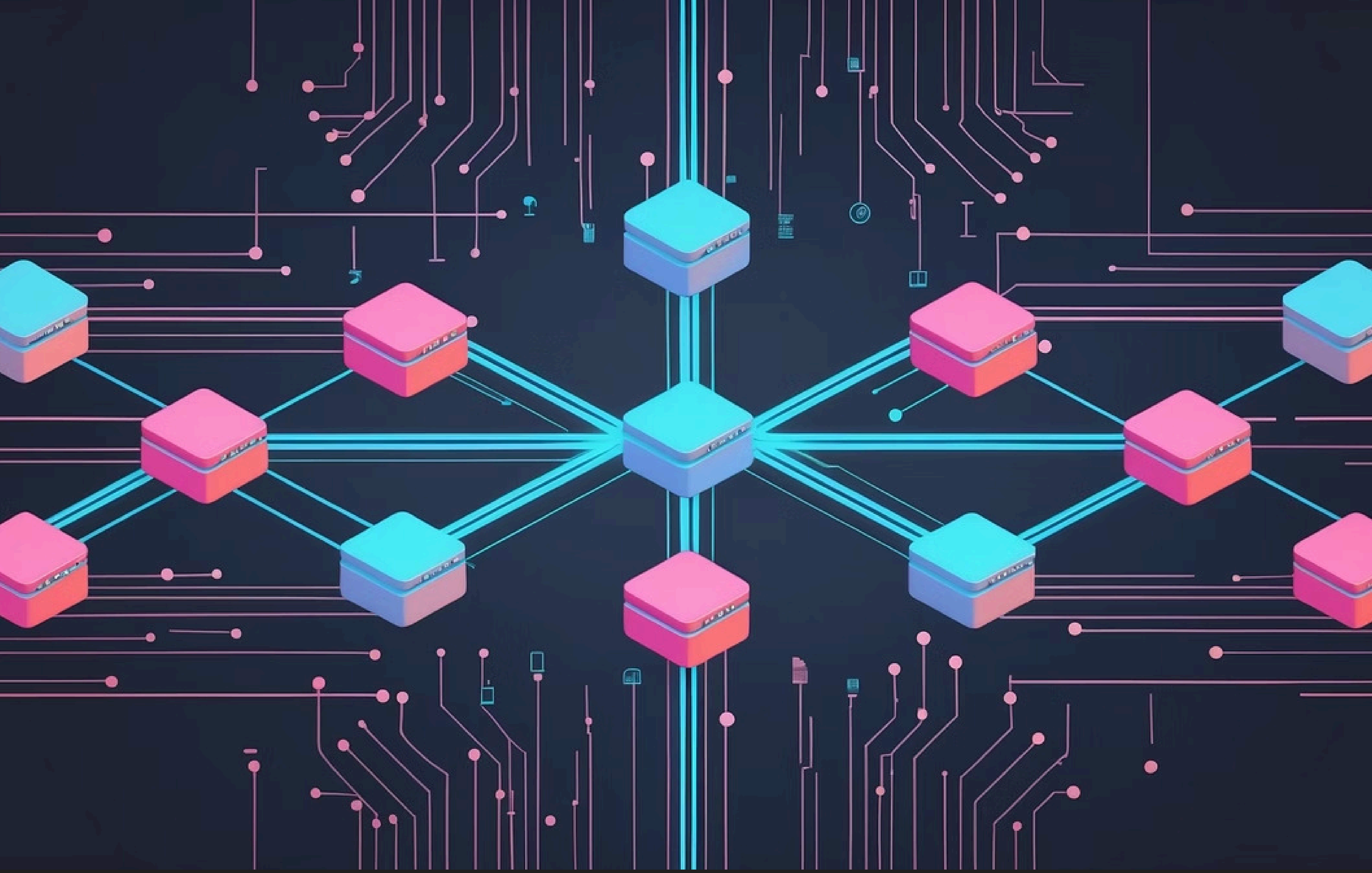
The number of operations a system can handle per unit of time

Measured In

Requests per second (RPS), transactions per second (TPS)

Optimization

Achieved through parallelization and efficient algorithms



Load Balancing

1

User Requests

Traffic from multiple sources

2

Load Balancer

Distributes workload

3

Server Pool

Handles requests efficiently

Caching

Request data

Check if data exists in cache

Cache hit

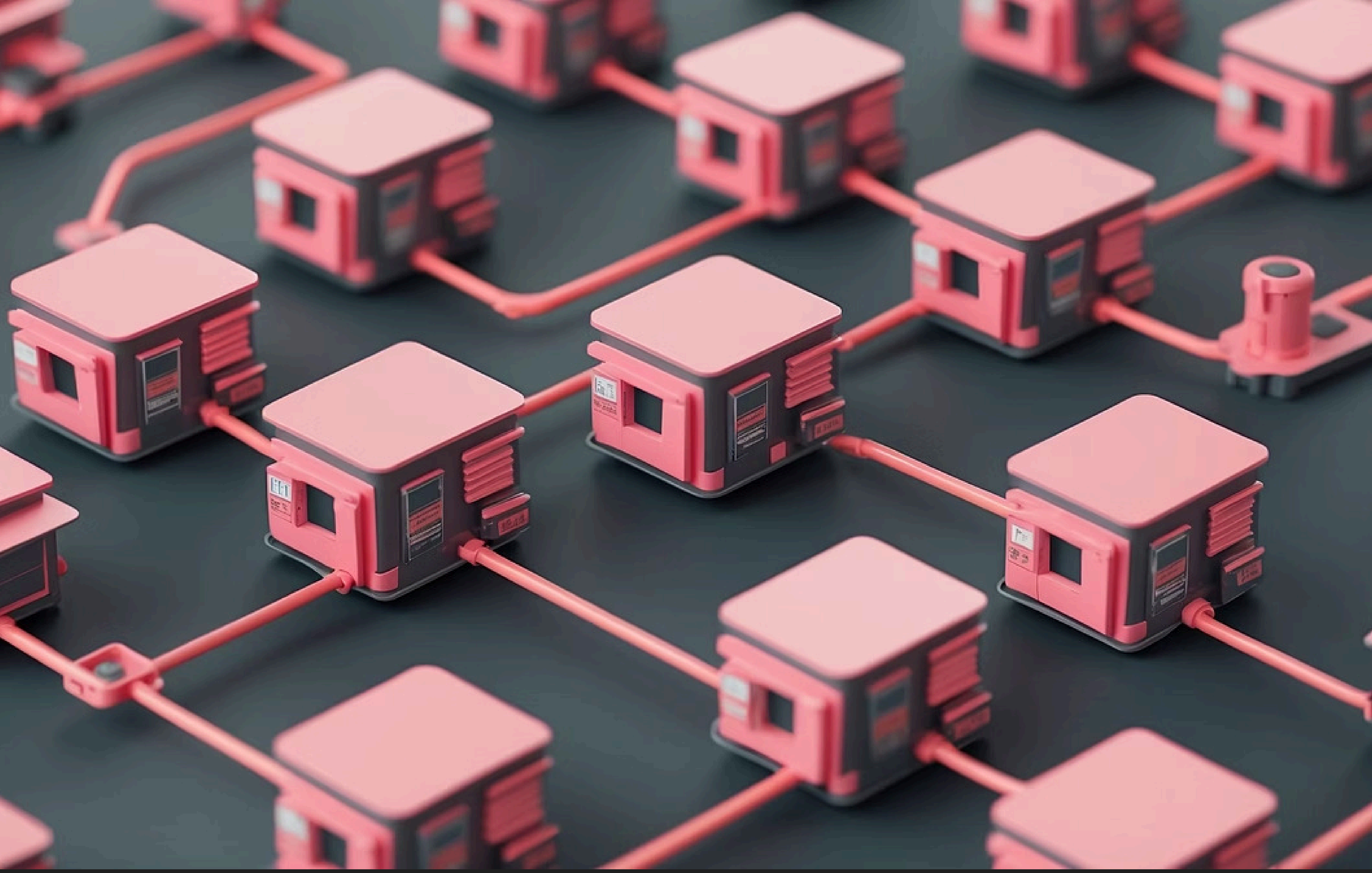
Return data from cache

Cache miss

Fetch from source and store in cache

Invalidate

Update when data changes



Microservices

Independent Components

Applications built as independent services

Advantages

Scalability, maintainability, and team autonomy

Challenges

Network complexity and distributed system debugging

API Gateway

Single entry point

Unified access for client requests

Request routing

Directs traffic to appropriate services

Security & monitoring

Authentication, rate limiting, and analytics

Database Sharding

What is it?

Splitting data across multiple databases to improve performance

Benefits

Improved query performance and horizontal scaling

Challenges

Complex joins, potential for hotspots, rebalancing

Eventual Consistency

Data changed

Update made to one node

Propagation

Changes spread gradually

Consistency

All nodes eventually updated

Data storage model where changes propagate gradually across the system

CAP Theorem

1 Consistency

All nodes see the same data at the same time

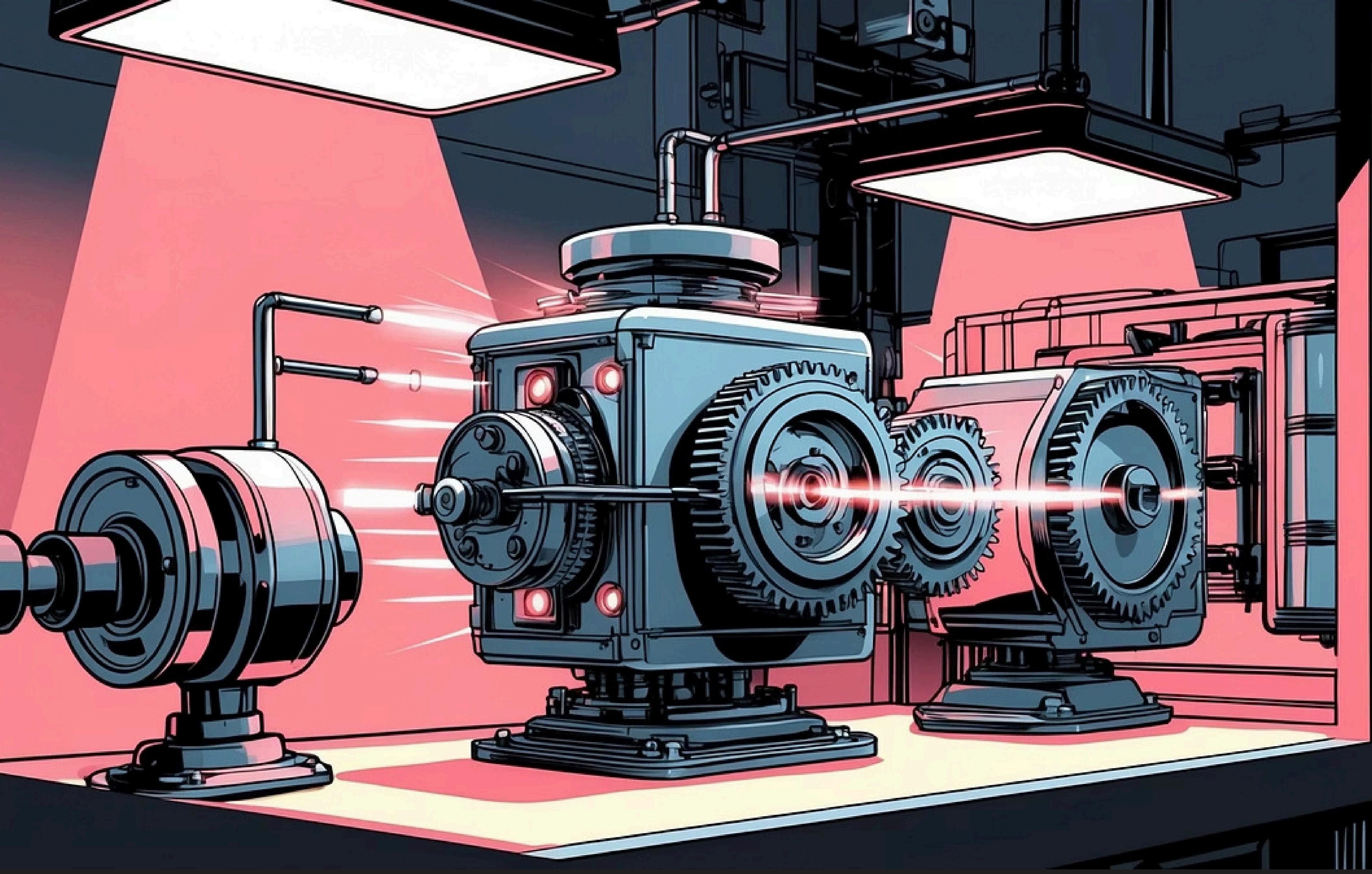
Availability

Every request receives a response

Partition Tolerance

System works despite network failures

You can only guarantee two of these three properties in a distributed system



Fault Tolerance

Definition

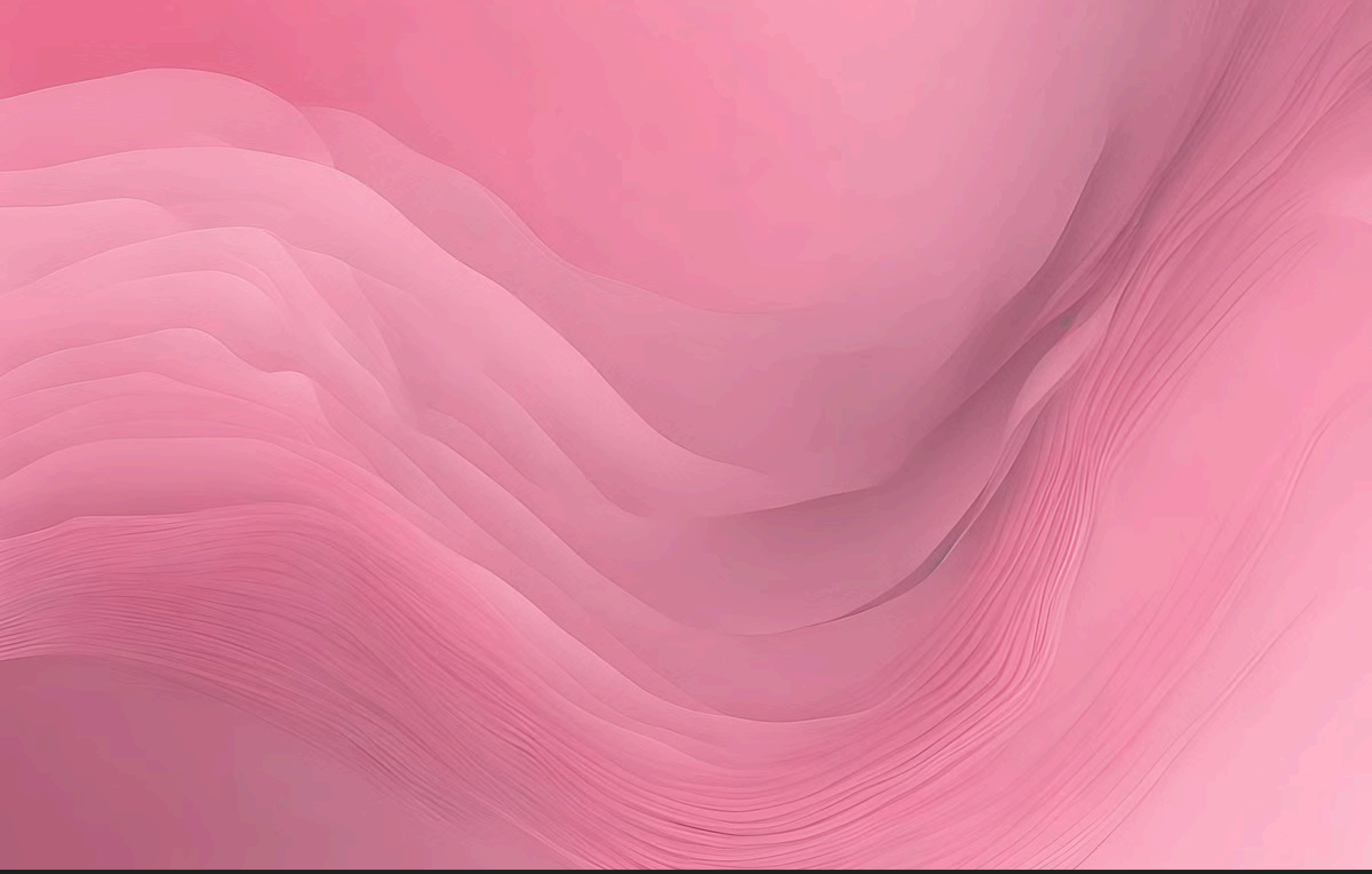
System's ability to continue functioning despite component failures

Redundancy

Duplicate critical components to eliminate single points of failure

Recovery Time

Minimize downtime through quick failover mechanisms



Circuit Breaker

Closed State

Normal operation, requests flow through

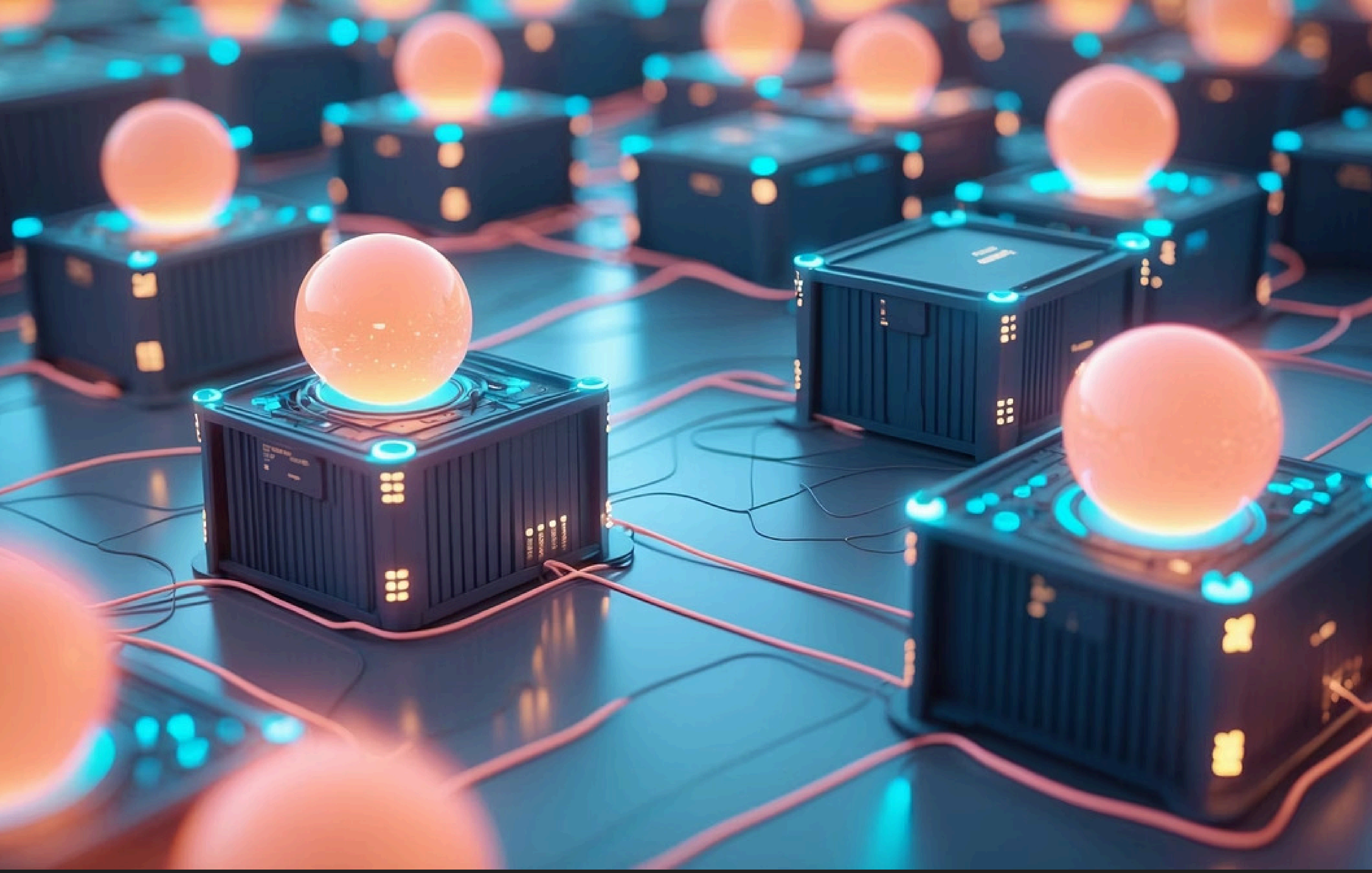
Open State

Failure threshold reached, requests rejected immediately

Half-Open State

Limited requests allowed to test system recovery

Prevents cascading failures in distributed systems by "breaking the circuit" when dependencies fail



Message Queue

Producer

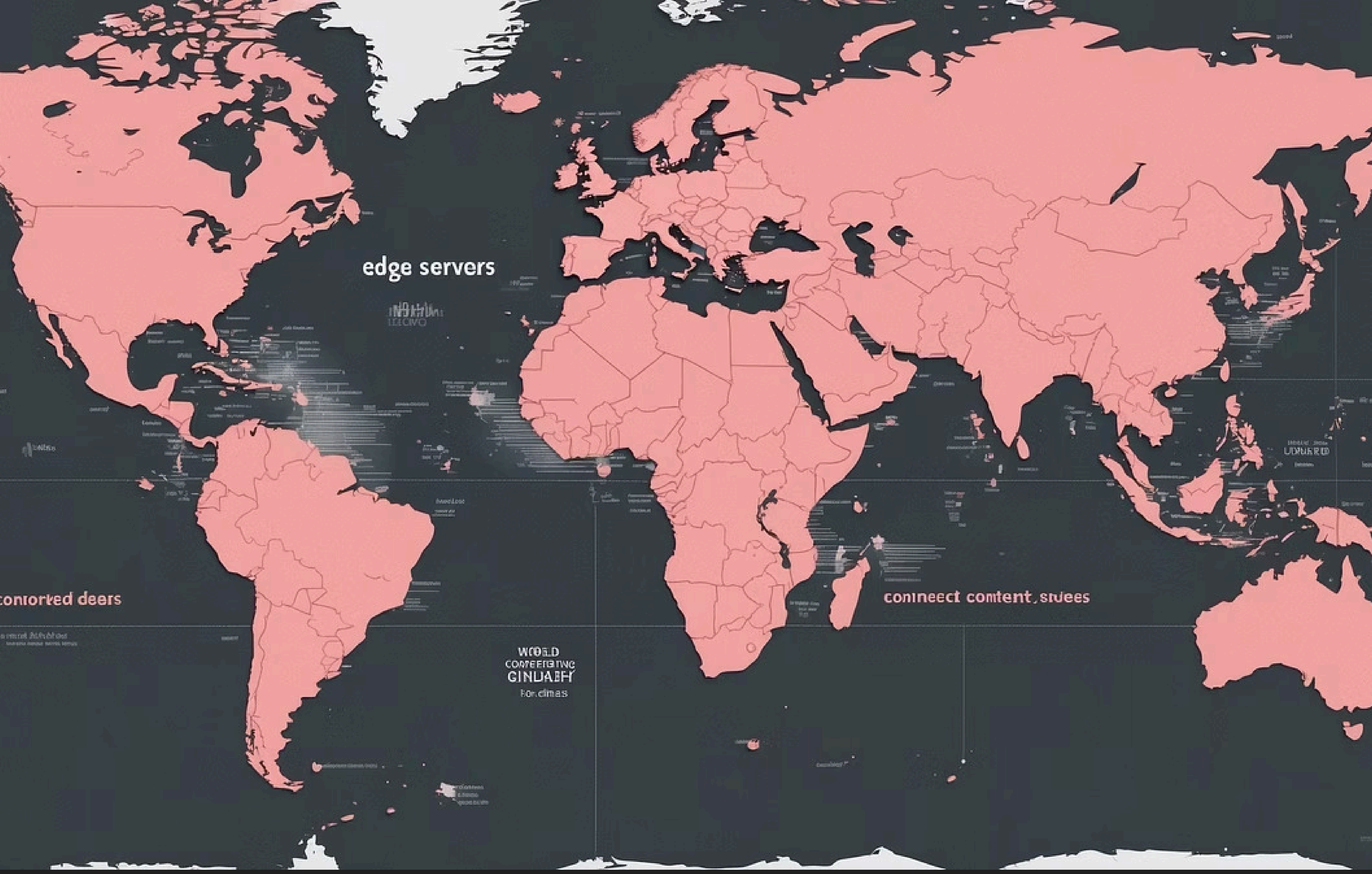
Creates messages

Queue

Stores messages

Consumer

Processes messages



Content Delivery Network (CDN)

Global Distribution

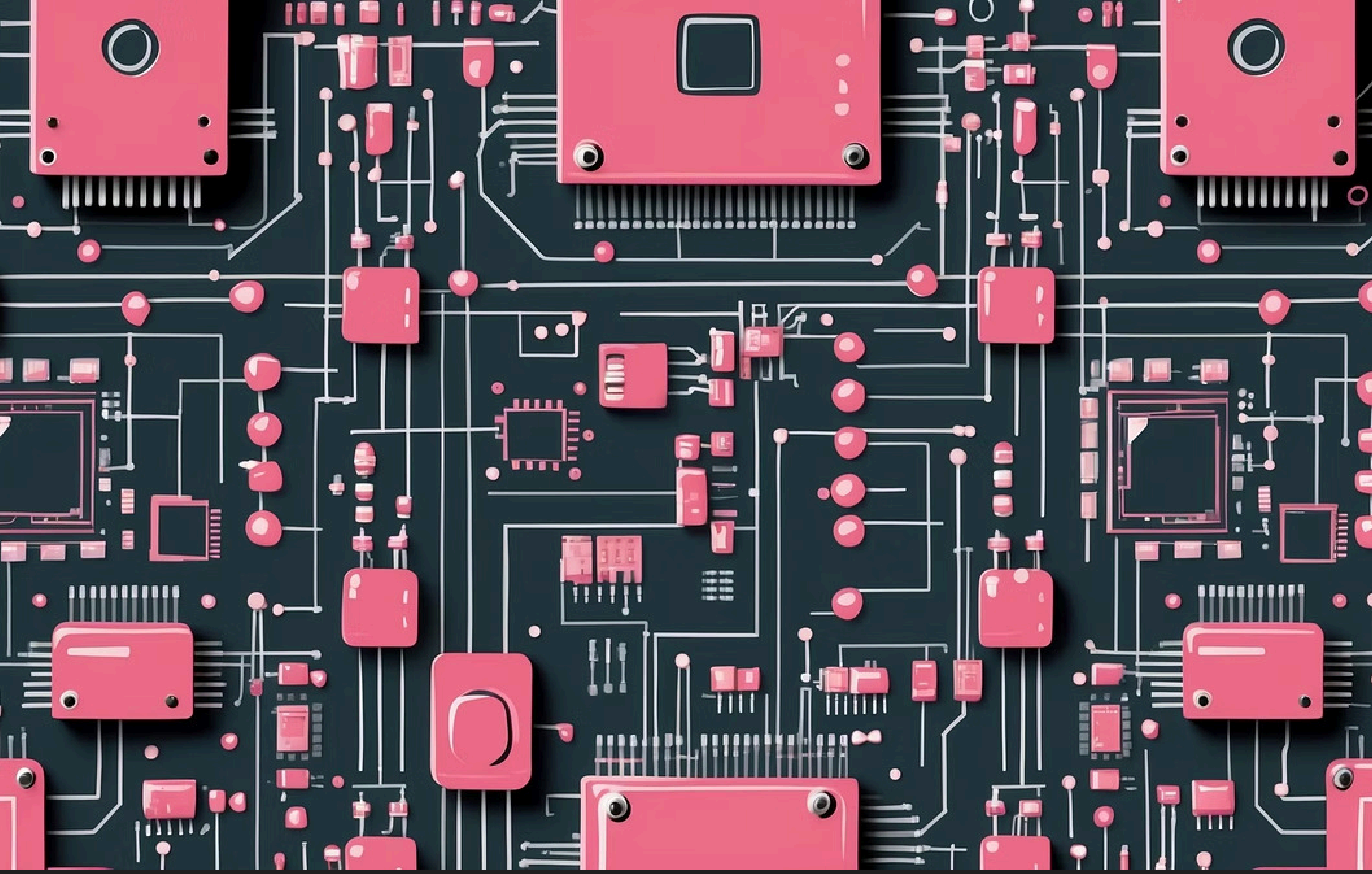
Servers located worldwide to reduce physical distance

Fast Delivery

Content served from nearest edge location to users

Protection

Provides DDoS protection and traffic offloading



Idempotence

What is it?

Operations produce the same result regardless of repetition

Examples

HTTP GET, PUT requests; setting a variable to a specific value

Why it matters

Critical for reliable systems that may retry operations

Rate Limiting

Protection

Prevent abuse and DoS attacks

Fair usage

Ensure resource sharing

System stability

Prevent overload

Controlling how many requests can be made in a given timeframe

Blue-Green Deployment

A deployment strategy that maintains two identical production environments, allowing for zero-downtime releases by switching traffic from the old version (blue) to the new version (green).

Tag a developer who needs to know these terms or share this post to help your team level up their system design knowledge!