Mastering Large-Scale Systems

Building Resilient, Scalable Architecture with UPI Payment Systems

Session Agenda

- Fundamentals of Large-Scale Systems
- System Design Principles
- Core Components Architecture
- Scalability Techniques
- Resiliency & Fault Tolerance
- Consistency & Data Management
- Performance Optimization
- Monitoring & Observability
- Security in Large-Scale Systems
- UPI Payment System Case Study

Why Large-Scale Systems?

20B+
UPI Transactions/Month

99.99%
Uptime Required

~10K
Transaction per second

500M+
Active Users

$500B+
Worth Trx per Month

Fundamentals of Large-Scale Systems

Understanding Core Concepts & Challenges

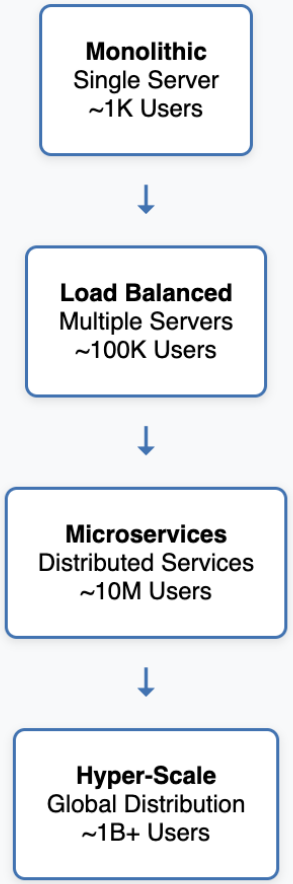
What Defines Large-Scale?

- **Core Pillers:** Performance, Scalability, Reliability, Resilience, Secure
- **Volume:** Millions of users, billions of requests daily
- **Velocity:** High throughput with low latency requirements
- **Variety:** Multiple data types, formats, and sources
- **Geographic Distribution:** Multiple data centers globally
- **Complexity:** Hundreds of microservices

Key Challenges

- Handling massive concurrent load
- Ensuring high availability (99.99%+)
- Maintaining data consistency
- Managing system complexity
- Cost optimization at scale

Scale Evolution Journey



System Design Principles

Foundation for Building Robust Large-Scale Systems

Core Design Principles

🎯 Scalability

- Horizontal scaling (scale-out)
- Vertical Scaling (Scale-Up)
- Auto-scaling based on metrics
- Load balancing & distribution
- Microservices architecture

🛡️ Reliability

- Fault tolerance & error handling
- Redundancy across all layers
- Graceful degradation strategies
- Disaster recovery planning
- Data backup & recovery

⚡ Performance

- Low latency design
- High throughput optimization
- Caching at multiple layers
- Database query optimization
- CDN for global performance

🔒 Security

- Defense in depth strategy
- Zero trust security model
- End-to-end encryption
- Authentication & authorization
- Regular security audits

Additional Critical Principles

📊 Observability

- Comprehensive monitoring
- Distributed tracing
- Centralized logging
- Real-time alerting

🔄 Consistency

- Eventual consistency model
- ACID where required
- Data synchronization
- Conflict resolution

🔗 Loose Coupling

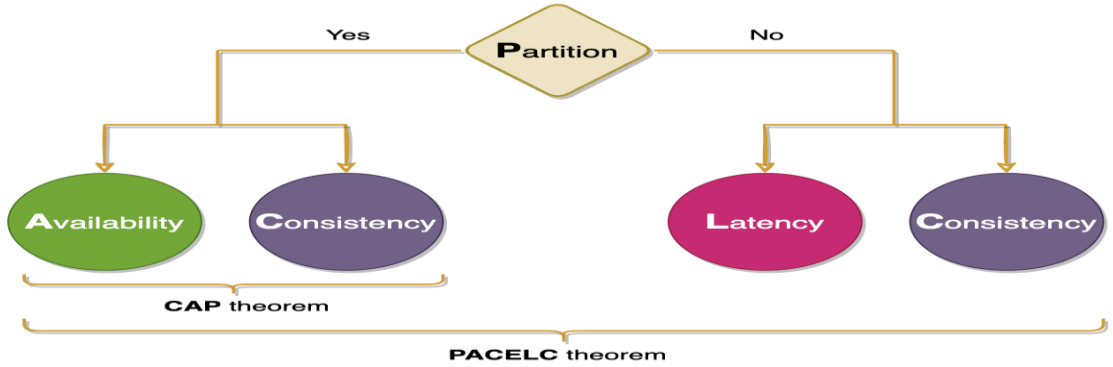
- Service independence
- API-driven communication
- Event-driven architecture
- Database per service

System Design Principles

Foundation for Building Robust Large-Scale Systems

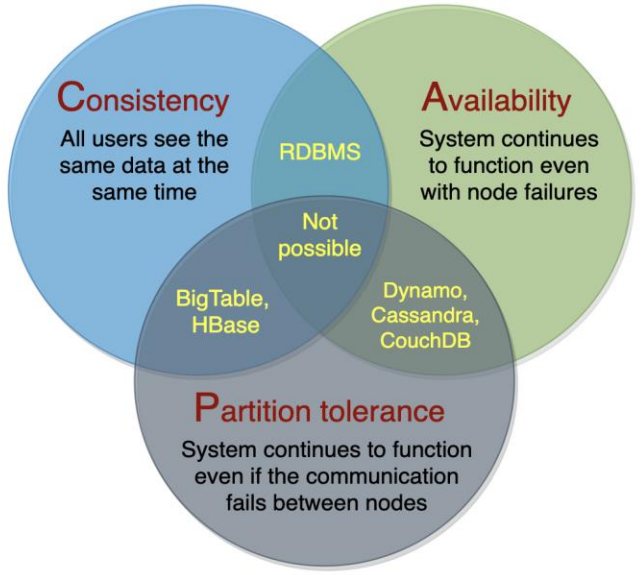
CAP Theorem & Trade-offs

System Type	Consistency	Availability	Partition Tolerance
Banking/UPI Core	✅ Strong (CP)	⚠️ Moderate	✅ Required
Social Media	⚠️ Eventual (AP)	✅ High	✅ Required
E-commerce Catalog	⚠️ Eventual (AP)	✅ High	✅ Required
Analytics/Reporting	✅ Strong (CP)	⚠️ Lower	✅ Required



Essential Design Patterns

- **Database per Service:** Data isolation
- **API Gateway:** Single entry point
- **CQRS:** Separate read/write models
- **Event Sourcing:** Audit trail & replay
- **Saga Pattern:** Distributed transactions
- **Circuit Breaker:** Prevent cascading failures
- **Bulkhead:** Resource isolation
- **Strangler Fig:** Legacy migration



CAP Theorem

🎯 Design Philosophy for Large Scale

Core Principles

- **Design for Failure:** Assume components will fail
- **Keep it Simple:** Complexity is the enemy of reliability
- **Automate Everything:** Reduce human error
- **Measure & Monitor:** You can't improve what you don't measure

Operational Excellence

- **Immutable Infrastructure:** Consistent deployments
- **Blue-Green Deployment:** Zero-downtime releases
- **Feature Flags:** Safe feature rollouts
- **Chaos Engineering:** Test system resilience

Core COmponents of Large Scale Systems

Complete component overview with connections and data flow

Internet / External Zone

Users
Web browsers, mobile apps

→

DNS
Domain name resolution

→

CDN
Global content delivery for static assets

|

3rd Party APIs
External services

Edge / Security Zone

WAF
Web application firewall

→

DDoS Protection
Attack mitigation

→

Reverse Proxy
SSL termination

→

API Gateway
Single API entry point

→

Load Balancer
Traffic distribution

Application Zone

Web Servers
Static content serving

→

App Servers
Business logic execution

↔

Cache Layer
Fast data access

|

Microservices
Independent services

↔

Message Queue
Async communication

|

Background Jobs
Async processing

|

Search Engine
Full-text search

|

Session Store
User sessions

Data Zone

Primary Database
Main RDBMS

→

Read Replicas
Read-only copies

|

NoSQL Database
Document/key-value store

|

Blob Storage
Object storage

|

Time Series DB
Metrics storage

|

Data Warehouse
Analytics store

Infrastructure Zone

Monitoring
System metrics

↔

Alerting
Real-time notifications

|

Logging
Centralized logs

↔

Distributed Tracing
Request tracing

|

CI/CD Pipeline
Automated deployment

|

Service Discovery
Dynamic lookup

|

Auto Scaling
Resource scaling

|

Secrets Mgmt
Secure credentials

🌐 Internet / External Zone

U Users & Clients

- Web browsers, mobile apps, desktop applications
- Different device types (mobile, tablet, desktop)
- Various network conditions and locations
- Authentication states (logged in/out)

C CDN (Content Delivery Network)

- Caches static content globally (images, CSS, JS)
- Reduces latency and server load
- Edge locations worldwide
- Examples: CloudFlare, AWS CloudFront

D DNS (Domain Name System)

- Translates domain names to IP addresses
- Load balancing through DNS routing
- Failover and geo-routing capabilities
- TTL settings for cache control

3 Third-Party Services

- Payment gateways (Stripe, PayPal)
- Social media APIs (Facebook, Google)
- External data providers
- Analytics services (Google Analytics)

Edge / DMZ Zone

WAF (Web Application Firewall)

- Filters malicious HTTP/HTTPS traffic
- Protection against OWASP Top 10 attacks
- SQL injection and XSS prevention
- Rate limiting and bot detection

DDoS Protection

- Mitigates distributed denial of service attacks
- Traffic scrubbing and filtering
- Automatic scaling during attacks
- Real-time threat intelligence

Load Balancer (ALB/NLB)

- Distributes traffic across multiple servers
- Health checks and failover
- SSL termination
- Layer 4 (TCP) or Layer 7 (HTTP) balancing

API Gateway

- Single entry point for all API requests
- Authentication and authorization
- Rate limiting and throttling
- Request/response transformation

Reverse Proxy

- Routes client requests to backend servers
- SSL termination and certificate management
- Compression and caching
- Examples: Nginx, HAProxy, Envoy

⚙️ Application Zone

W Web Servers

- Serve static content and handle HTTP requests
- Examples: Apache, Nginx, IIS
- Handle concurrent connections
- Request routing and load distribution

A Application Servers

- Execute business logic and dynamic content
- Process API requests and database queries
- Session management and state handling
- Examples: Tomcat, Node.js, Django

M Microservices

- Loosely coupled, independently deployable services
- Domain-driven design principles
- Service-to-service communication (REST/gRPC)
- Independent scaling and technology choices

Q Message Queues

- Asynchronous communication between services
- Decoupling of producers and consumers
- Guaranteed delivery and ordering
- Examples: RabbitMQ, Apache Kafka, AWS SQS

C Cache Layer

- In-memory data storage for fast access
- Reduces database load and latency
- TTL-based expiration strategies
- Examples: Redis, Memcached, Hazelcast

S Search Engine

- Full-text search and analytics
- Indexing and real-time search capabilities
- Faceted search and aggregations
- Examples: Elasticsearch, Solr, Amazon CloudSearch

B Background Jobs/Workers

- Process long-running or scheduled tasks
- Email sending, report generation, cleanup
- Job queues and retry mechanisms
- Examples: Sidekiq, Celery, AWS Lambda

K Container Orchestration

- Deploy and manage containerized applications
- Auto-scaling and self-healing
- Service discovery and networking
- Examples: Kubernetes, Docker Swarm, ECS

🗄️ Data Zone

D Primary Database

- ACID-compliant relational database
- Handles writes and critical reads
- Examples: PostgreSQL, MySQL, Oracle
- Master in master-slave replication

R Read Replicas

- Read-only copies of primary database
- Distributes read load across multiple instances
- Eventually consistent with master
- Geographic distribution for latency

N NoSQL Database

- Schema-flexible, horizontally scalable
- Document, key-value, column, or graph storage
- Examples: MongoDB, Cassandra, DynamoDB
- Eventually consistent, partition tolerant

W Data Warehouse

- Optimized for analytical queries (OLAP)
- Historical data storage and reporting
- ETL processes for data transformation
- Examples: Snowflake, Redshift, BigQuery

B Blob Storage

- Object storage for files, images, videos
- Highly scalable and durable
- CDN integration for global distribution
- Examples: AWS S3, Azure Blob, Google Cloud Storage

T Time Series Database

- Optimized for time-stamped data
- Metrics, logs, and IoT data storage
- High ingestion rates and compression
- Examples: InfluxDB, TimescaleDB, OpenTSDB

🔧 Infrastructure & Operations Zone

M Monitoring & Observability

- Metrics collection and visualization
- Application performance monitoring (APM)
- Examples: Prometheus, Grafana, DataDog
- SLI/SLO tracking and alerting

L Logging & Log Management

- Centralized log collection and analysis
- Log aggregation from multiple sources
- Examples: ELK Stack, Splunk, CloudWatch
- Log retention and compliance

T Distributed Tracing

- Track requests across microservices
- Performance bottleneck identification
- Examples: Jaeger, Zipkin, AWS X-Ray
- Service dependency mapping

C CI/CD Pipeline

- Automated build, test, and deployment
- Version control integration
- Examples: Jenkins, GitLab CI, GitHub Actions
- Blue-green and canary deployments

S Service Discovery

- Dynamic service registration and lookup
- Health checking and load balancing
- Examples: Consul, etcd, AWS Cloud Map
- DNS-based or API-based discovery

A Auto Scaling

- Dynamic resource scaling based on demand
- Horizontal and vertical scaling strategies
- CPU, memory, and custom metric triggers
- Cost optimization and performance

V Secrets Management

- Secure storage of passwords, API keys, certificates
- Encryption at rest and in transit
- Examples: HashiCorp Vault, AWS Secrets Manager
- Automatic rotation and access control

B Backup & Disaster Recovery

- Automated backup strategies and scheduling
- Point-in-time recovery capabilities
- Cross-region replication
- RTO/RPO requirements and testing

Database Technologies Guide

Complete overview of database types, use cases, and selection criteria

Relational (SQL) Databases

MySQL
Relational

Open-source, reliable, web-focused

Web apps, E-commerce, CMS

PostgreSQL
Relational

Advanced features, JSON support

Analytics, Complex queries, GIS

SQL Server
Relational

Microsoft enterprise database

Enterprise, BI, .NET apps

Oracle
Relational

Enterprise-grade, high performance

Mission-critical, Large enterprises

SQLite
Embedded

Lightweight, serverless

Mobile apps, Embedded systems

Document Databases

MongoDB
Document

Flexible JSON-like documents

Rapid development, Content management

CouchDB
Document

Multi-version concurrency control

Offline-first apps, Mobile sync

DocumentDB
Document

AWS managed MongoDB-compatible

Cloud applications, Content catalogs

Key-Value Stores

Redis
Key-Value

In-memory, extremely fast

Caching, Sessions, Real-time

DynamoDB
Key-Value

AWS managed, auto-scaling

Serverless, Gaming, IoT

Memcached
Cache

Distributed memory caching

Web acceleration, Query caching

Column Family

Cassandra
Column Family

Highly scalable, distributed

Big data, High-volume writes, IoT

HBase
Column Family

Hadoop-based wide column store

Big data analytics, Real-time access

Bigtable
Column Family

Google's NoSQL service

Low-latency, High-throughput analytics

Graph Databases

Neo4j
Graph

Native graph storage & processing

Social networks, Fraud detection

Neptune
Graph

AWS managed graph database

Identity graphs, Network security

ArangoDB
Multi-Model

Document, graph & key-value

Multi-model apps, Complex relationships

Database Type	Examples	Primary Use Cases	When to Choose (Selection Criteria)
Relational Databases (RDBMS)	MySQL, PostgreSQL, Oracle, SQL Server	Traditional applications (ERP, CRM, banking, ecommerce), Structured data with relationships, ACID compliance required	Data is structured & relational Need for transactions & consistency Mature tooling & SQL support
NoSQL – Document Stores	MongoDB, CouchDB, Firestore	Content management, Product catalogs, Mobile & web apps with dynamic schema	Flexible schema Need hierarchical or semistructured data High scalability & availability
NoSQL – KeyValue Stores	Redis, DynamoDB, Riak	Caching Session management Realtime leaderboards, gaming	Ultrafast read/write Simple keyvalue access pattern Low latency required
NoSQL – ColumnOriented	Apache Cassandra, HBase, ScyllaDB	Timeseries data IoT telemetry Logging & analytics at scale	Largescale data ingestion High write throughput Distributed & faulttolerant
NoSQL – Graph Databases	Neo4j, ArangoDB, Amazon Neptune	Social networks Fraud detection Recommendation engines	Complex relationships Query performance on connected data Traversing networks quickly
TimeSeries Databases	InfluxDB, TimescaleDB, Prometheus	IoT sensors Monitoring & observability Financial market data	Timestamped data Optimized for sequential writes & queries over time Realtime analytics
NewSQL Databases	CockroachDB, YugabyteDB, Google Spanner	Largescale OLTP (transactions) Cloudnative applications Global services	Need SQL + scalability High availability across regions ACID compliance at scale
Search Engines (as DBs)	Elasticsearch, OpenSearch, Solr	Fulltext search Log analytics Ecommerce search systems	Need text search & ranking Fast query over unstructured text Support for analytics & indexing
ObjectOriented Databases	db4o, ObjectDB, Versant	CAD/CAM systems AI & simulations Objectheavy applications	Tight integration with objectoriented programming Complex data objects
InMemory Databases	Redis (also keyvalue), Memcached, SAP HANA	Realtime analytics Gaming leaderboards Financial trading systems	Low latency is critical High throughput Volatile or cached data use cases
MultiModel Databases	ArangoDB, OrientDB, Cosmos DB	Applications needing multiple data models (graph + document + keyvalue) Unified APIs	Flexibility to use multiple paradigms Reduces need for polyglot persistence

Scalability Techniques

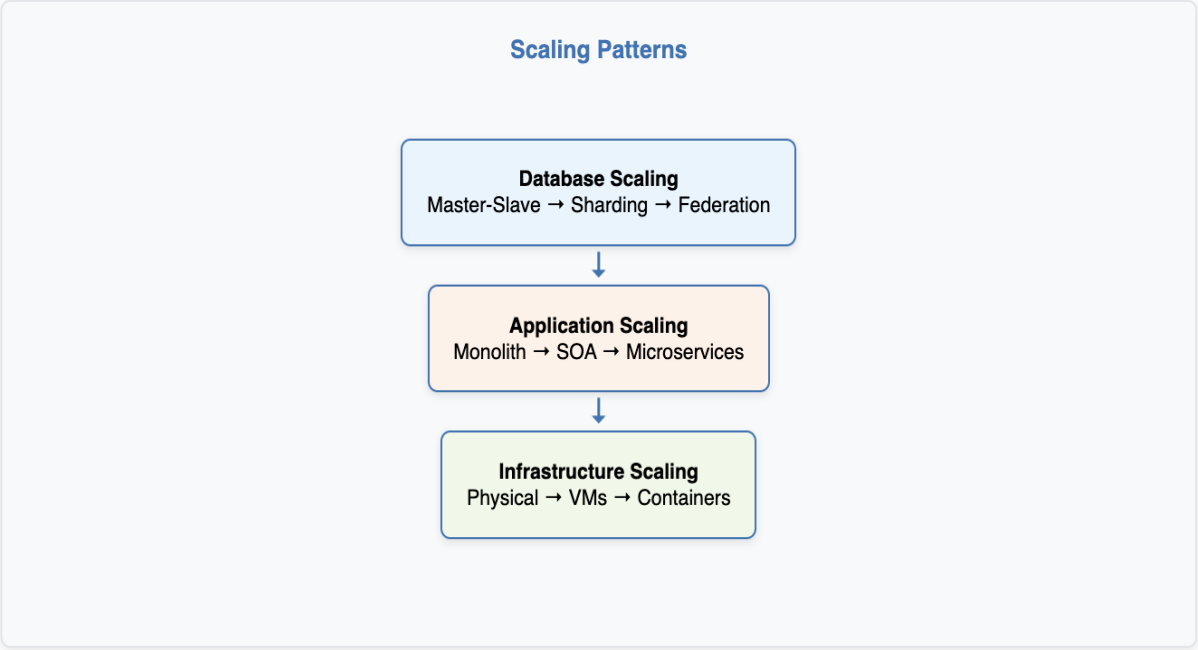
Strategies for Handling Growing Load

Horizontal Scaling

- **Load Balancing:** Distribute traffic across servers
- **Database Sharding:** Partition data across nodes
- **Read Replicas:** Scale read operations
- **CDN:** Global content distribution

Vertical Scaling

- **CPU Scaling:** More powerful processors
- **Memory Scaling:** Increased RAM capacity
- **Storage Scaling:** Faster, larger disks
- **Network Scaling:** Higher bandwidth



Caching Strategies

Cache Type	Location	Use Case	Pros	Cons
Browser Cache	Client Side	Static Assets	Reduces server load	Cache invalidation issues
CDN	Edge Locations	Global Content	Low latency globally	Cost for high traffic
Application Cache	Server Memory	Database Queries	Fast access	Memory limitations
Database Cache	Database Server	Query Results	Automatic management	Limited control

Resiliency & Fault Tolerance

Building Systems That Handle Failures Gracefully

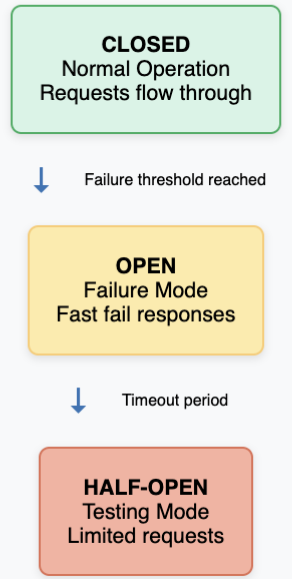
Fault Tolerance Patterns

- **Circuit Breaker:** Prevent cascading failures
- **Retry Pattern:** Handle transient failures
- **Timeout Pattern:** Avoid hanging requests
- **Bulkhead Pattern:** Isolate critical resources
- **Health Checks:** Monitor service health

Disaster Recovery Strategy

- **RTO:** Recovery Time Objective
- **RPO:** Recovery Point Objective
- **Backup Strategy:** 3-2-1 Rule
- **Multi-Region:** Geographic redundancy

Circuit Breaker Pattern



Error Handling Strategies

Graceful Degradation

- Reduced functionality
- Essential features only
- User-friendly messages

Fail-Fast

- Quick failure detection
- Immediate error response
- Resource conservation

Failover

- Automatic switching
- Backup systems ready
- Seamless user experience

Consistency & Data Management

Managing Data Across Distributed Systems

Consistency Models

- **Strong Consistency:** All nodes see same data simultaneously
- **Eventual Consistency:** System becomes consistent over time
- **Weak Consistency:** No guarantees when all nodes consistent
- **Causal Consistency:** Causally related operations ordered

Database Scaling Patterns

- **Read Replicas:** Scale read operations
- **Sharding:** Horizontal partitioning
- **Federation:** Split databases by function
- **Denormalization:** Optimize for reads

ACID vs BASE

ACID (Traditional)	BASE (NoSQL)
Atomicity All or nothing	**Basically Available** System remains operational
Consistency Valid state always	**Soft state** May change over time
Isolation Concurrent safety	**Eventual consistency** Becomes consistent eventually
Durability Permanent storage	

Data Management Strategies

Database per Service
Microservice isolation

Event Sourcing
Audit trail & replay

CQRS
Separate read/write models

Data Lake
Raw data storage

Data Warehouse
Structured analytics

Stream Processing
Real-time data

Performance Optimization

Techniques for High Performance at Scale

Application Level Optimization

- **Code Optimization:** Efficient algorithms & data structures
- **Memory Management:** Garbage collection tuning
- **Connection Pooling:** Reuse database connections
- **Asynchronous Processing:** Non-blocking operations
- **Lazy Loading:** Load data when needed

Database Optimization

- **Indexing:** B-tree, Hash, Bitmap indexes
- **Query Optimization:** Execution plan analysis
- **Partitioning:** Horizontal/vertical splits
- **Materialized Views:** Pre-computed results

Performance Metrics

TPS
Transactions Per Second

P95
95th Percentile Latency

QPS
Queries Per Second

CPU%
Resource Utilization

Optimization Priorities

1. Identify bottlenecks first
2. Optimize critical path
3. Cache frequently accessed data
4. Optimize database queries
5. Scale horizontally when needed

Performance Testing Strategy

Unit Testing
Component level

→

Load Testing
Expected traffic

→

Stress Testing
Breaking point

→

Spike Testing
Sudden traffic surge

Monitoring & Observability

Visibility into System Health & Performance

Three Pillars of Observability

- **Metrics:** Quantitative measurements over time
- **Logs:** Discrete events with context
- **Traces:** Request journey through system

Key Monitoring Areas

- **Infrastructure:** CPU, Memory, Disk, Network
- **Application:** Response time, Throughput, Errors
- **Business:** Transaction volume, Revenue impact
- **User Experience:** Page load time, Error rates

Alerting Strategy

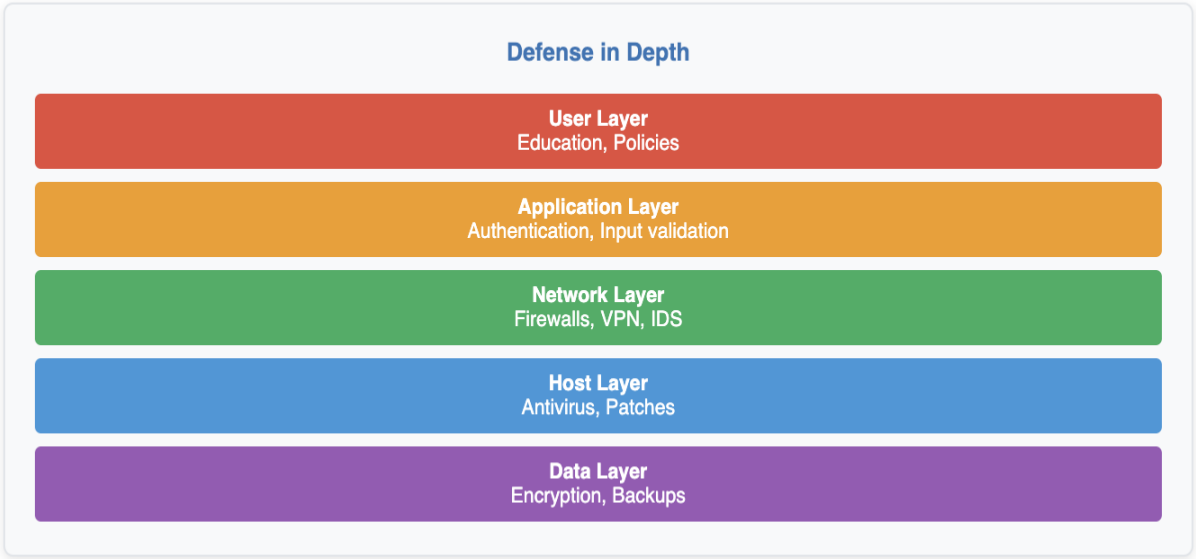
Alert Type	Trigger Condition	Response Time	Example
Critical	System down, Data loss	< 5 minutes	Payment service unavailable
High	Performance degradation	< 15 minutes	Response time > 2 seconds
Medium	Resource utilization high	< 1 hour	CPU usage > 80%
Low	Trends, Capacity planning	< 24 hours	Disk space trending up

Security in Large-Scale Systems

Multi-layered Security Approach

Security Layers

- **Perimeter Security:** WAF, DDoS protection
- **Network Security:** VPC, Subnets, Security groups
- **Application Security:** Authentication, Authorization
- **Data Security:** Encryption at rest & in transit
- **Infrastructure Security:** Hardened OS, Patches



Authentication & Authorization

- **Multi-factor Authentication:** Something you know/have/are
- **OAuth 2.0 / OpenID Connect:** Delegated authorization
- **JWT Tokens:** Stateless authentication
- **RBAC:** Role-based access control

Security Best Practices

Data Protection

- Encryption everywhere
- Data classification
- Access logging
- Data masking

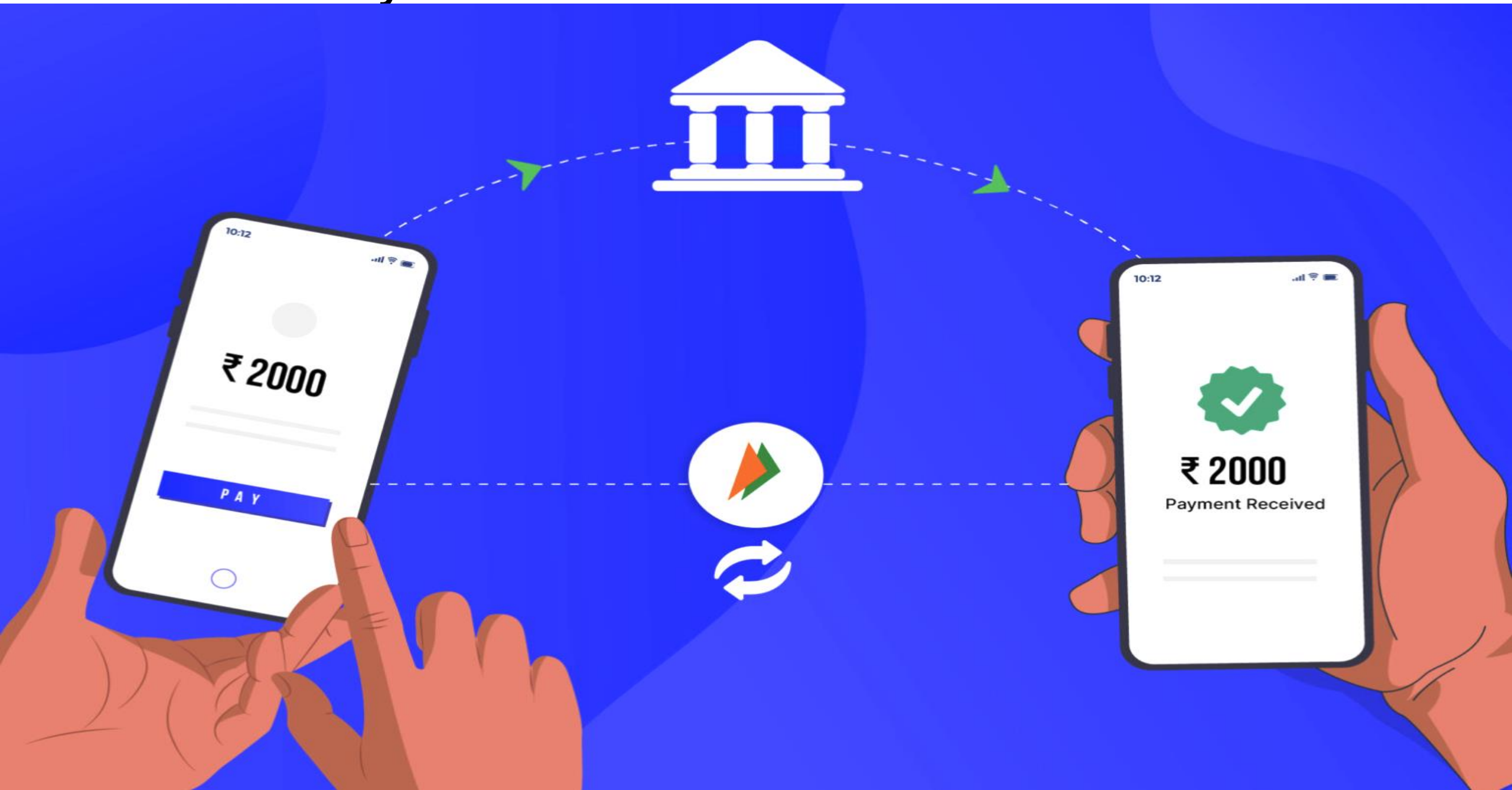
API Security

- Rate limiting
- Input validation
- HTTPS only
- API versioning

Compliance

- PCI DSS
- GDPR/Privacy
- SOC 2
- Regular audits

UPI Case Study



Functional Requirements – Core Features

FR1: User Registration & KYC
Users must be able to register with valid bank accounts and complete KYC verification

FR2: Virtual Payment Address (VPA) Creation
System shall allow users to create unique payment addresses like user@bankname

FR3: Fund Transfer via VPA
Enable instant money transfer using recipient's virtual payment address

FR4: QR Code Payment
Support payment through scanning merchant or personal QR codes

Functional Requirements – Transaction Management

FR5: Account Balance Inquiry
Users should be able to check their linked bank account balances

FR6: Transaction History
Maintain complete transaction logs with timestamps and status updates

FR7: Payment Request (Collect)
Allow users to send payment requests to other UPI users

FR8: Transaction Limits Management
Enforce daily/monthly transaction limits as per RBI guidelines

Functional Requirements - Security & Authentication

FR9: Multi-Factor Authentication
Implement MPIN, biometric, and device-based authentication mechanisms

FR10: Transaction PIN Validation
Require secure PIN entry for all financial transactions

FR11: Fraud Detection & Prevention
Real-time monitoring for suspicious activities and automatic blocking

FR12: Session Management
Secure session handling with automatic timeout and re-authentication

Functional Requirements - Integration & Services

FR13: Bank Account Linking
Support multiple bank account integration through NPCI infrastructure

FR14: Merchant Payment Gateway
Enable businesses to accept UPI payments through APIs and SDKs

FR15: Bill Payment Integration
Support utility bill payments, mobile recharges, and subscription services

FR16: Notification Services
Send real-time SMS/push notifications for all transaction activities

Non-Functional Requirements - Performance

NFR1: Response Time
Transaction processing must complete within 5 seconds under normal load

NFR2: Throughput Capacity
System shall handle minimum 10K concurrent transactions per second

NFR3: System Availability
Maintain 99.9% uptime

NFR4: Scalability
Architecture must scale horizontally to support 500M+ registered users

Non-Functional Requirements - Security

NFR5: Data Encryption
All sensitive data must be encrypted using AES-256 encryption standards

NFR6: Regulatory Compliance
Comply with RBI, PCI-DSS, and Indian IT Act security regulations

NFR7: Audit Trail
Maintain immutable logs for all system activities and transactions

NFR8: Access Control
Role-based access control with least privilege principle implementation

Non-Functional Requirements - Reliability & Quality

NFR9: Disaster Recovery
Complete system recovery within 4 hours of catastrophic failure

NFR10: Data Consistency
ACID properties maintained across all financial transactions

NFR11: Cross-Platform Compatibility
Support Android, iOS, and web platforms with consistent functionality

NFR12: Error Handling
Graceful degradation with meaningful error messages and automatic retry

UPI System Capacity Estimation - Volume & Storage

Metric	Value
Total Transactions	20 Billion
Total Transaction Amount	$500B per month
Daily Average Transactions	500 Million
Estimated Annual Transactions	500M*365
Peak Daily Transactions	800 Million
Transactions Per Hour	10K * 60 * 60
Transactions Per Second (TPS)	10K

Capacity Estimations

Metric	Value
Sample Payload Size	~373 bytes
Daily Storage Requirement	200 GB
Monthly Storage Requirement	6 TB
Annual Storage Requirement	75 TB

Storage Growth Pattern

Daily: 200 GB → **Monthly:** 6 TB → **Annual:** 75 TB

Network Bandwidth Calculation

API Call Size	2 KB	
API Calls per Transaction	5	(initiate, process, notify)
Total API Calls per Day	2.5 Billion	
Data Transfer per Day	5 TB	(2.5 Billion API calls × 2 KB)
Peak API Calls per Day	4 Billion	
Peak Data Transfer per Day	8 TB	(4 Billion API calls × 2 KB)

UPI System Capacity Estimation - Infrastructure

Server Requirements

Metric	Value
Assumed Request Processing Time	50 ms
Peak TPS	10K
Throughput (Requests per Second)	500,500 RPS
API Gateway Servers Required	~300 servers
Backend Servers Required	~1800 servers

Server Assumptions

Gateway: ~1,000 RPS/server
Backend: ~200 RPS/server

Infrastructure Summary

8,000
Peak TPS

75 TB
Annual Storage

200 GB/day
Peak Bandwidth

~2100
Total Servers

Key Capacity Planning Insights

Peak Load
8000 TPS peak transactions
500,500 RPS total requests
4B API calls per peak day

Storage Growth
200 GB/day storage requirement
6 TB/month monthly growth
75 TB/year annual storage

Infrastructure
~300 API Gateway servers
~1800 Backend servers
~2100 Total server capacity

Scaling Recommendations

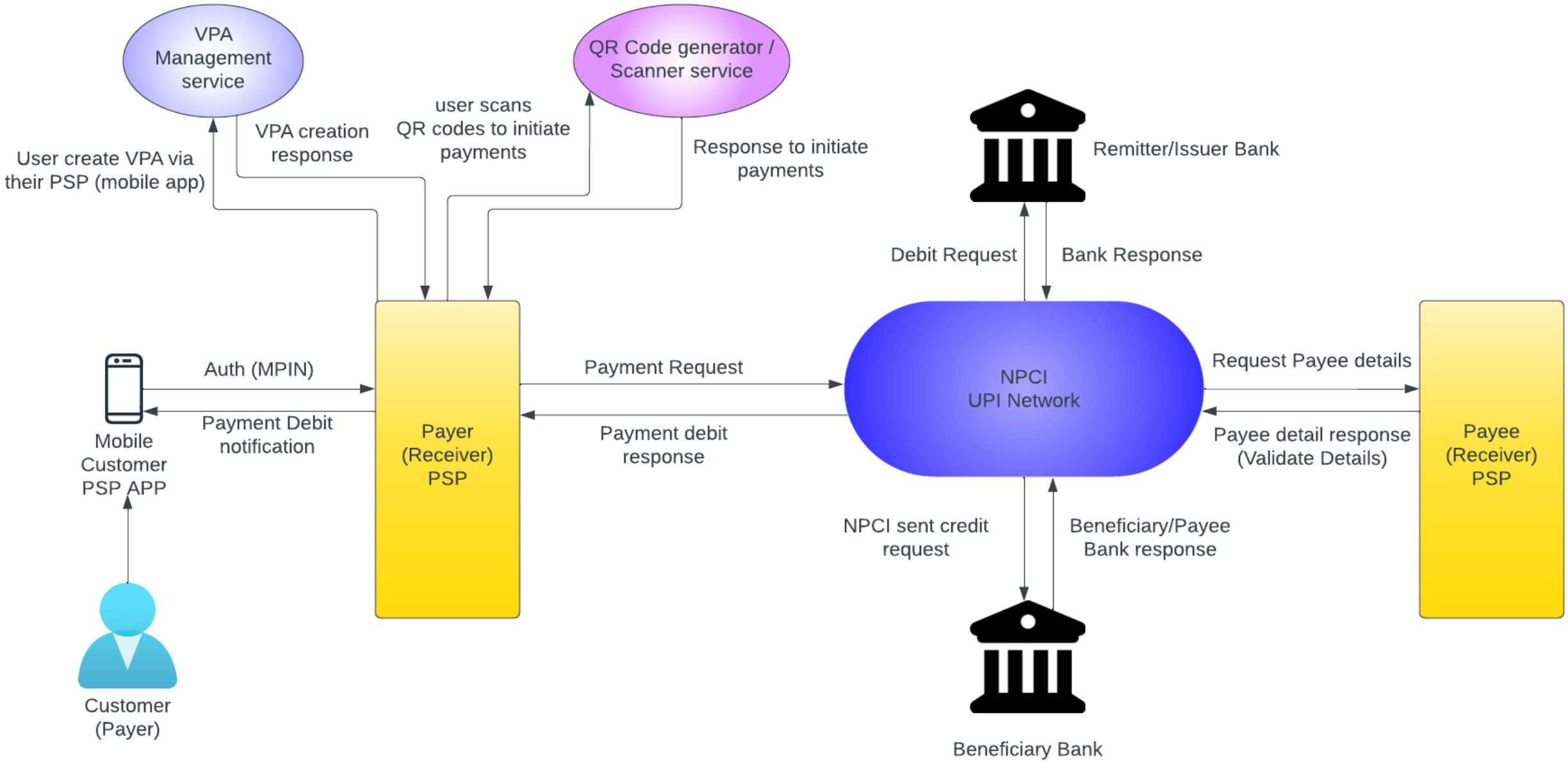
Auto-scaling:
30% buffer capacity

Load Balancing:
Multi-region deployment

Caching:
Redis clusters for hot data

Database:
Horizontal sharding strategy

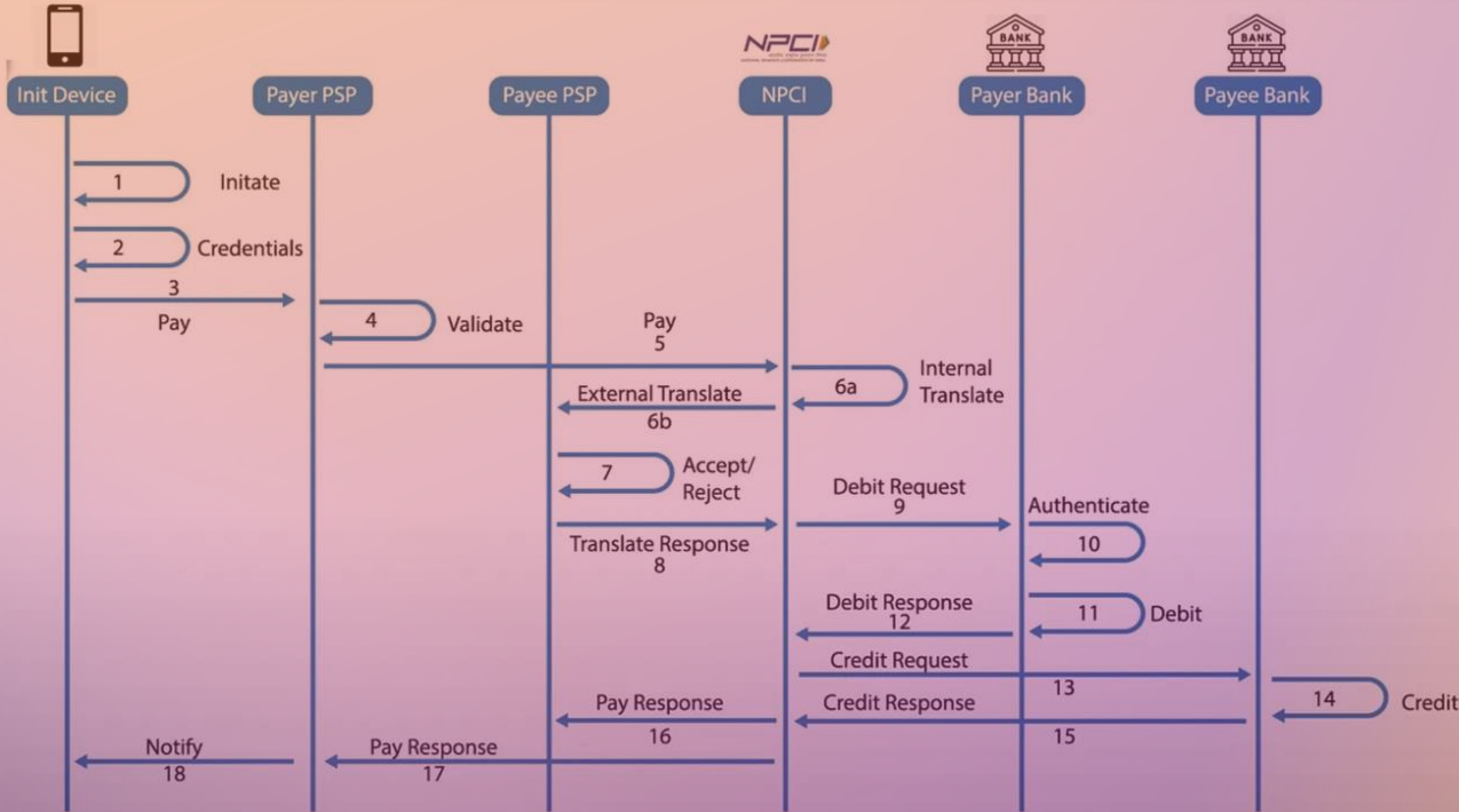
UPI High level Arch Diagram



Components

UPI Payment System Design - NPCI

- **Client Side Components**
  - **Mobile Application**
    To Initiate the request / receive payments
  - **Web Application**
    Interface for merchants to handle UPI payments
  - **User Authentication**
    OTP, biometric, or PIN for user authentication
- **Server Side Components**
  - **Payment Gateway Server**
    Handles UPI payment requests and responses
  - **Merchant Server**
    Server to handle transactions from the merchant side
- **Databases**
  - **Relational Databases (MYSQL)**
    User Database
  - **NoSQL Databases (TIGRIS)**
    Transactions Database, Merchant Database
- **Cache**
  - **In-Memory Cache (Redis)**
    For caching user session data, transaction status, and frequently accessed data to improve performance.
  - **Content Delivery Network (CDN)**
    To serve Static Javascript Bundle
- **Load Balancers**
  Elastic Load Balancer
- **Message Queues (Kafka)**
  For handling asynchronous communication between services, such as transaction processing and notification services.
- **API Gateways**
  Manages API requests, handles routing, authentication, and rate limiting.
- **Microservices**
  Transaction Service, Notification Service, Account Service, Fraud Detection Service
- **Service Discovery**
- **Configuration Management**
  Configuration Files
- **Monitoring and Logging**
  - **Monitoring**
    Kyndryl's Monitoring tools but Prometheus and Grafna can be used
  - **Logging**
    Kyndryl's logging mechanism but ELK Stack can be used
- **Authentication and authorization**
  - **Authentication**
    OAuth 2.0 / JWT
  - **Access Control**
    RBAC
- **Analytics and Reporting**
  Kyndryl's Analytics tools
- **Scalability and Reliability**
  - **Horizontal Scaling**
  - **Vertical Scaling**
  - **Redundancy**
  - **Backup and Disaster Recovery**
- **Networking**
  - **DNS**
    Domain name resolution
  - **Firewalls and Security Groups**
    VPC
- **DevOps and CI/CD**
  - **CI/CD**
    Ansible
  - **Infrastructure as Code (IaC)**
    Terraform
- **Data Processing**
  - **Real-Time Processing**
    Kyndryl's tools but Flink can be used


Init Device
Payer PSP
Payee PSP
NPCI
Payer Bank
Payee Bank
1
Initate
2
Credentials
3
Pay
4
Validate
Pay
5
6a
Internal
Translate
External Translate
6b
7
Accept/
Reject
Debit Request
9
Authenticate
10
Translate Response
8
Debit Response
12
11
Debit
Credit Request
13
14
Credit
Pay Response
16
Credit Response
15
Notify
18
Pay Response
17

The UPI settlement process