Distributed systems design
Part II

Distributed systems design
-> **Distributed data store**

Distributed systems design
-> **Distributed data store**

- ACID
- NoSQL
  - MongoDB/Cassandra/Neo4J
- Distribution Models
- CAP
- Map/Reduce
- Hadoop
- Apache Spark

Характеристики современных приложений

- Веб (доступные по HTTP)
- Интерактивные
- Много клиентов (Броузер, Native clients)

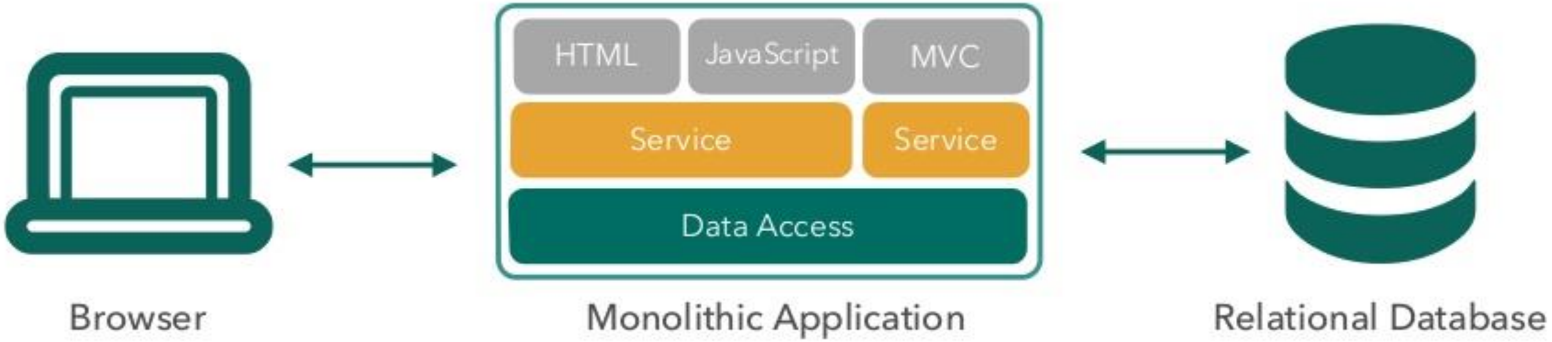
  => единое API
- Много пользователей (одновременных)
- Оооочень много пользователей
  - Cyber Monday

  ***=> оооочень мощные (т.е. дорогие) сервера* || много серверов**
- Много данных
- Оооочень много данных
  - YouTube, Facebook, …

  ***=> оооочень большие (т.е. дорогие) БД* || много БД**
- Новая функциональность/возможности на основе «Оооочень много данных»

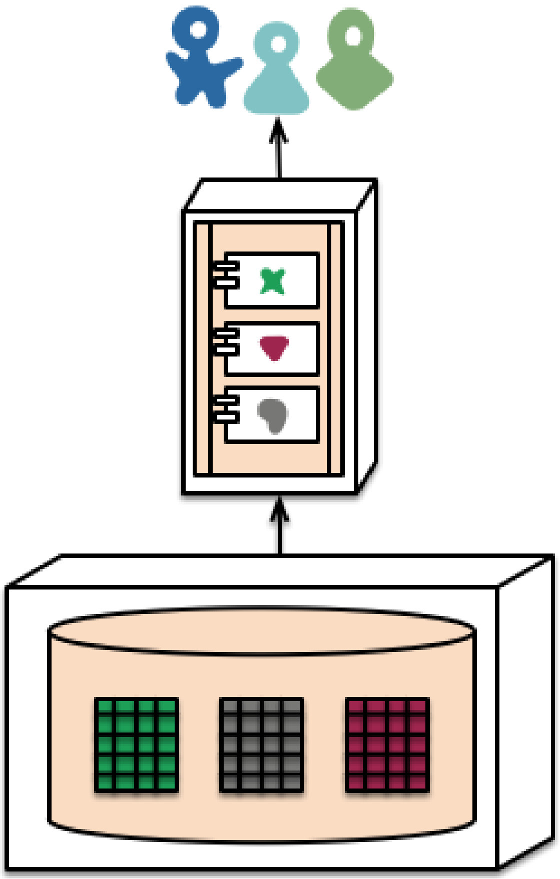
  => Big Data

Monolithic Architecture

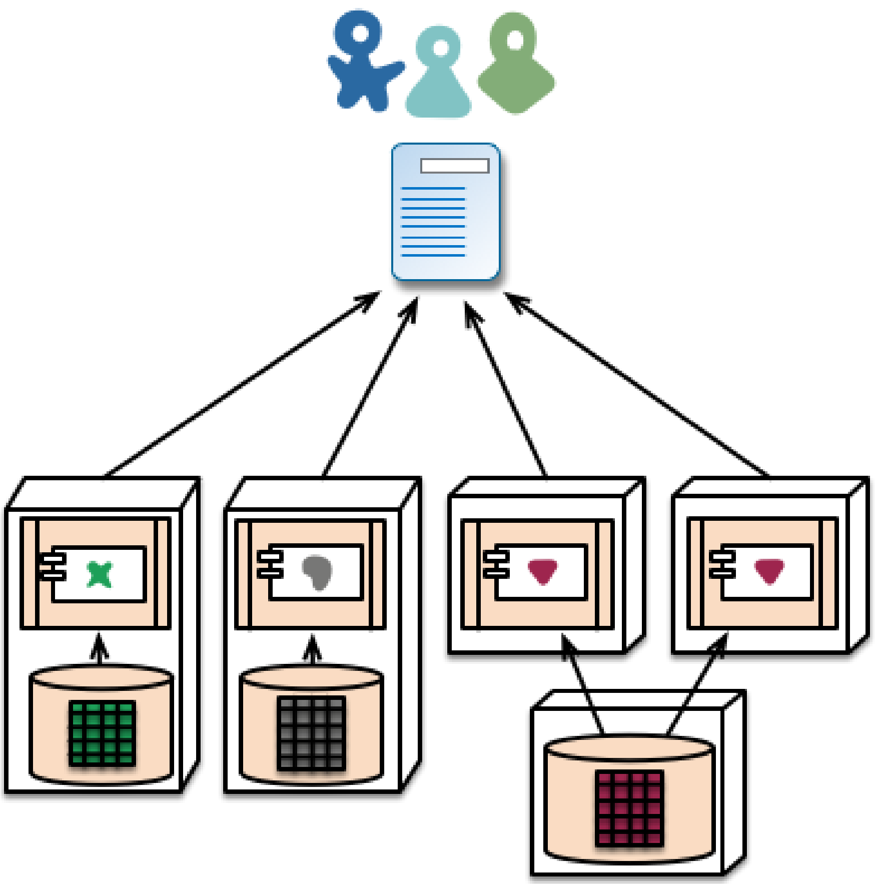


MICROSERVICES
LIGHTYEAR
SPACE RANGER
MICROSERVICES EVERYWHERE

Traditional DB vs Microservices DBs



monolith - single database



microservices - application databases

WHEN YOUR BOSS TELLS YOU
WE'RE CONVERTING TO MICROSERVICES
tex
made on imgur

Структура курса

- Part 1 – Theoretical
  - Distributed systems
- Part 2 – Practical
  - Microservices

Структура курса

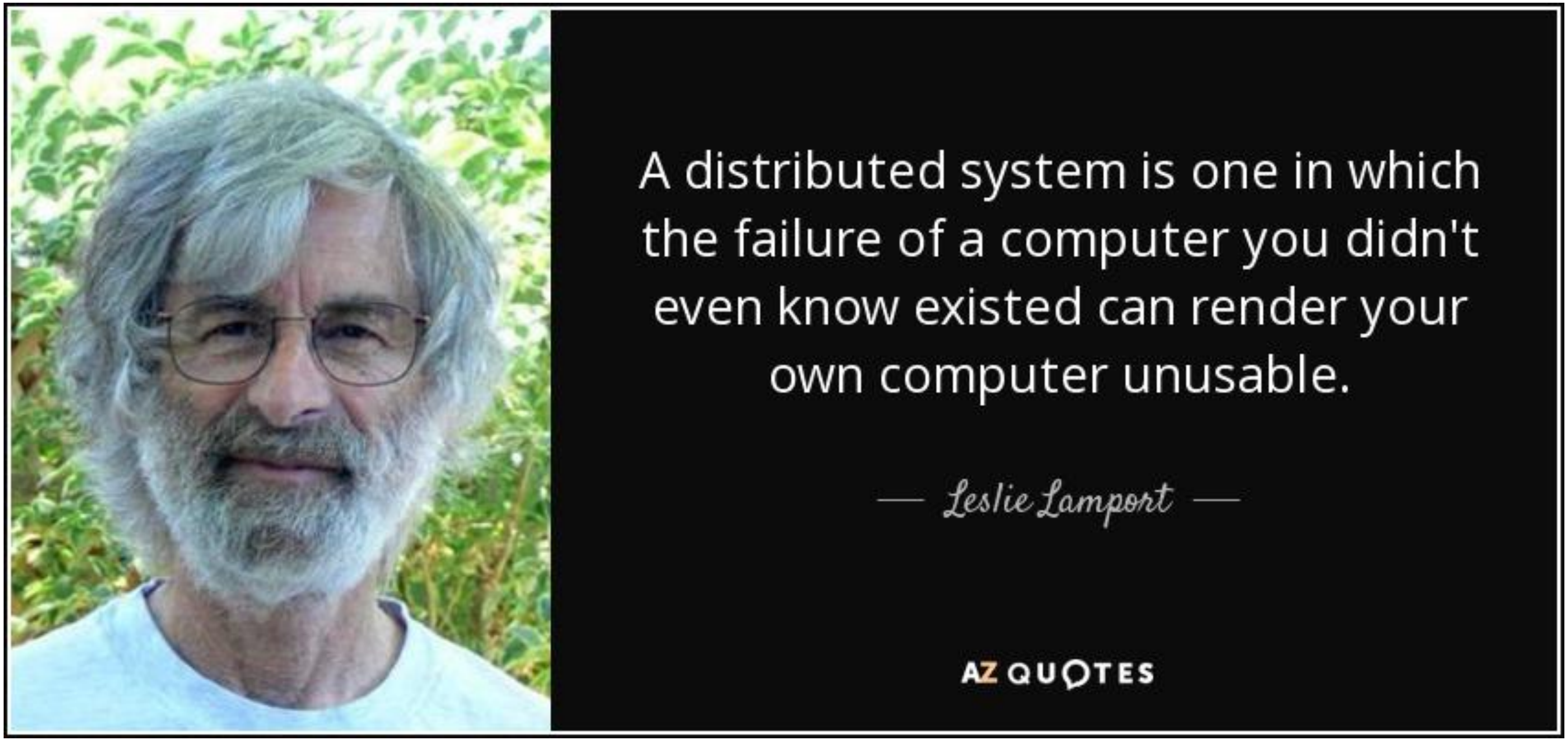
- Traditional (Web) Application architecture
- Distributed systems and scalability rules
- App/Business/Service Layer

Traditional (Web) Application architecture

- Application Layers
  - Repository/Persistence
  - Service/Business
  - Web/Presentation/View Layer
  - Domain/Business
- Distributed transactions: 2PC, 3PC
- Cost of scale

Distributed systems and scalability rules

- Parallel computing vs Distributed computing. Design and architecture principles
- Split-brain problem. Consistency
- Replication, Sharding (Partitioning)
- CAP theorem
- Consensus problem. Byzantine Generals problem
- Consensus protocols: Paxos, Raft, …



A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

— Leslie Lamport —

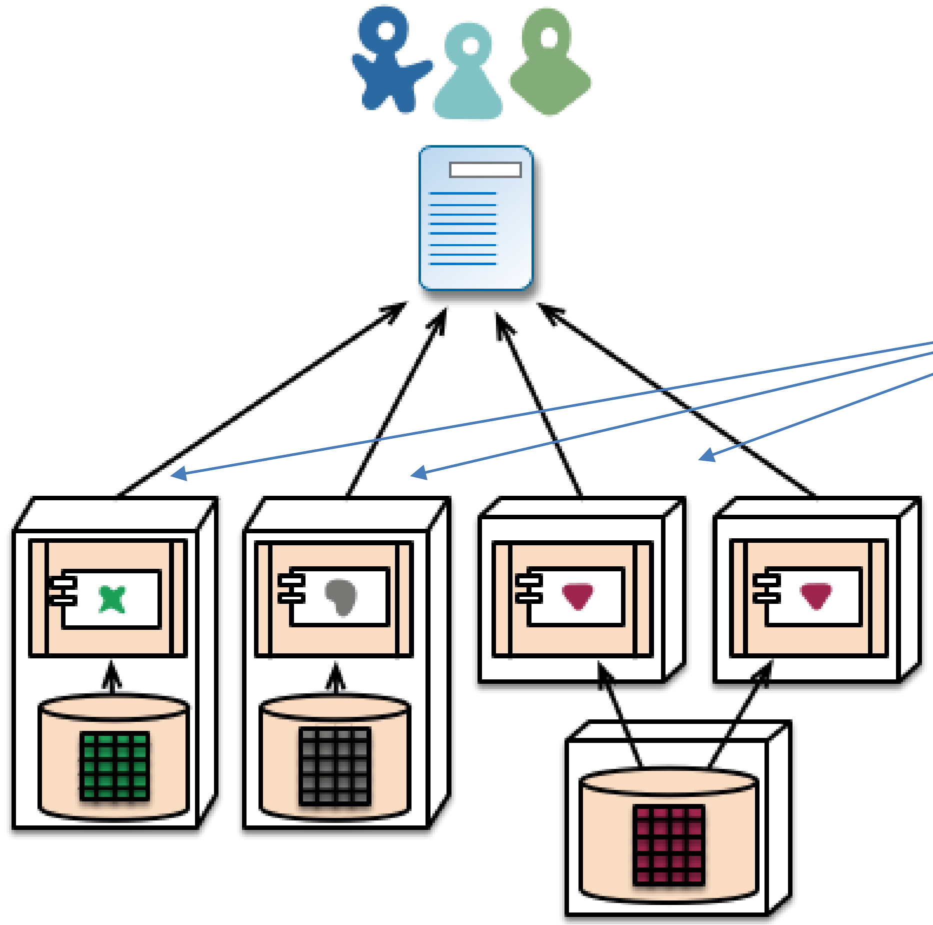
App/Business/Service Layer

- Stateless services
- Microservice architecture
- Distributed cache. In-Memory Data Grid
- Distributed Computing
- Messaging
- CQRS
- Batching

Practical tasks

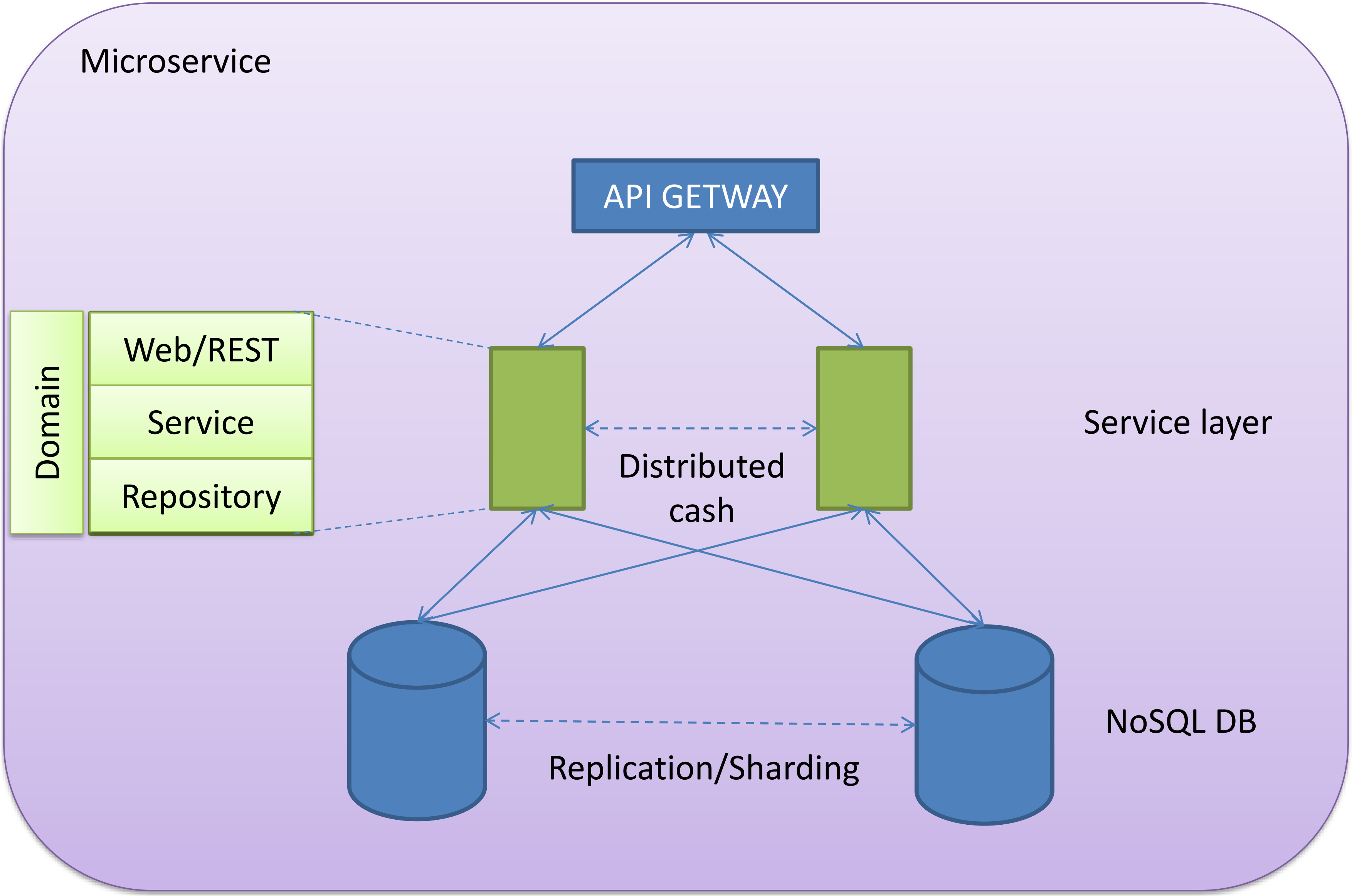
- GIT
- Distributed transactions (2 phase commit)
- Distributed cache (Hazelcats)
- Distributed computing (Hazelcats)
- Message broker (JMS, ActiveMQ, RabbitMQ)
- …

Project

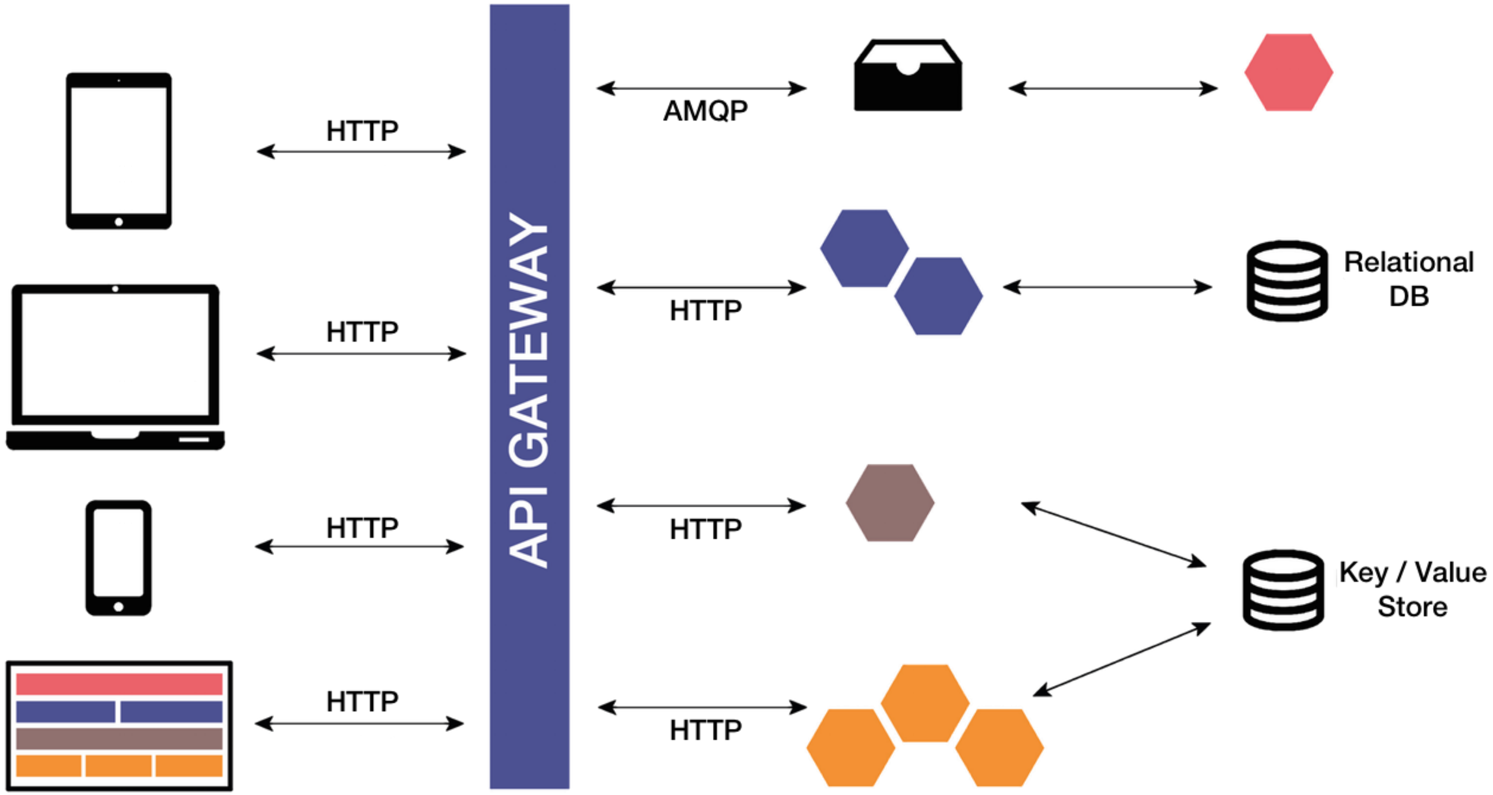


Microservices

Project


Microservice
API GETWAY
Domain
Web/REST
Service
Repository
Service layer
Distributed
cash
NoSQL DB
Replication/Sharding

Project

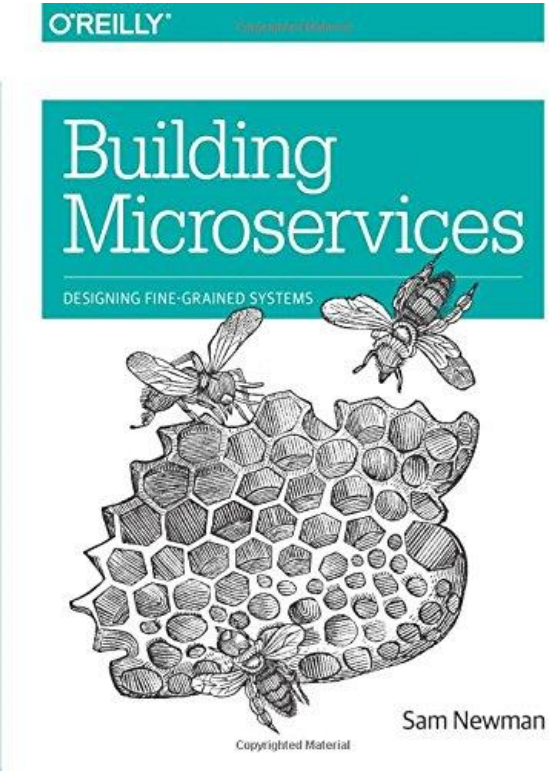

HTTP
HTTP
HTTP
HTTP
API GATEWAY
AMQP
HTTP
HTTP
HTTP
Relational DB
Key / Value Store

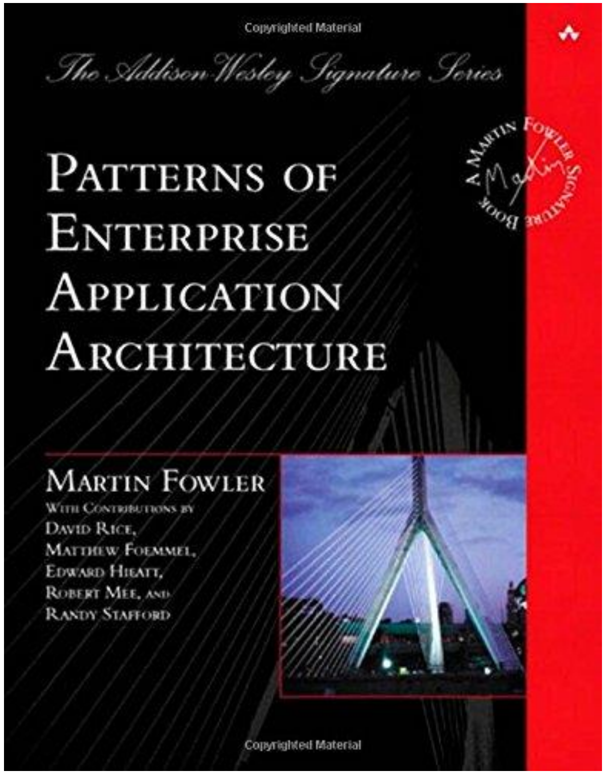
Project

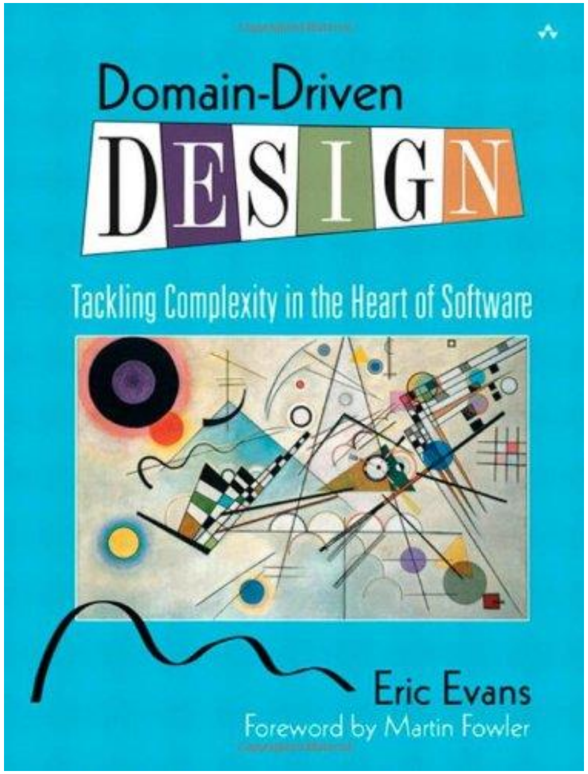
- Team: ~5-6 человек
- Source code repository (GitHub, BitBucket, ..)

- Microservice architecture
- 2-3 different DBs (NoSQL, Relational DB)
- Availability (DB layer & Service layer)
- Messaging

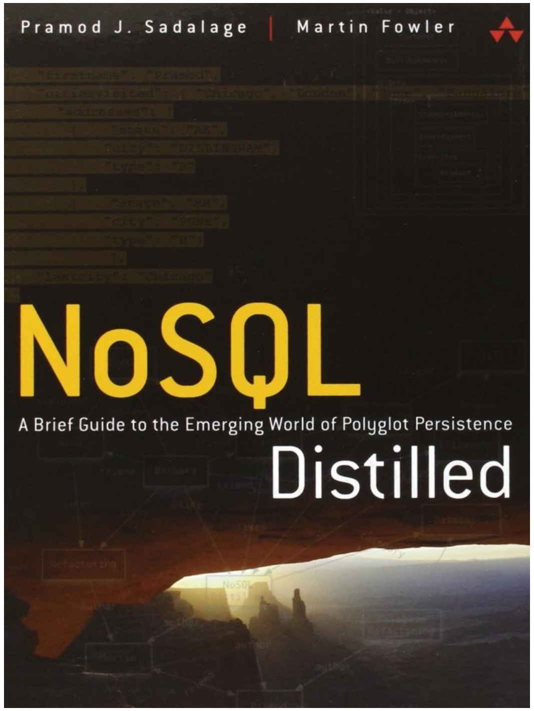
Books

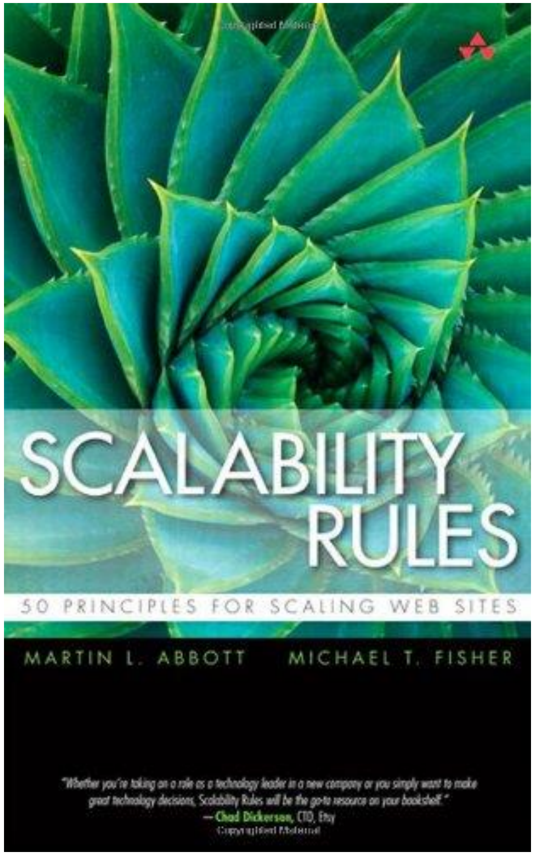

O'REILLY
Building Microservices
DESIGNING FINE-GRAINED SYSTEMS
Sam Newman
Copyrighted Material

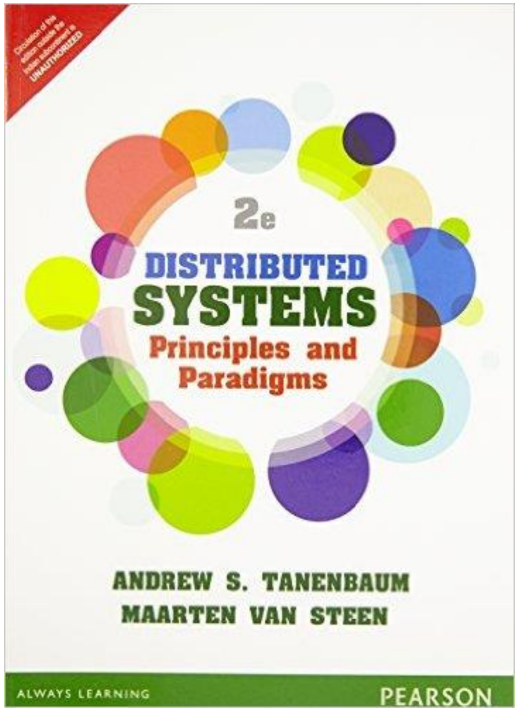

Copyrighted Material
The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Patterns of Enterprise Application Architecture
Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford
Copyrighted Material


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Books


Pramod J. Sadalage | Martin Fowler
NoSQL
A Brief Guide to the Emerging World of Polyglot Persistence
Distilled


SCALABILITY RULES
50 PRINCIPLES FOR SCALING WEB SITES
MARTIN L. ABBOTT MICHAEL T. FISHER
"Whether you're taking on a role as a technology leader in a new company or you simply want to make great technology decisions, Scalability Rules will be the go-to resource on your bookshelf."
—Chad Dickerson, CTO, Etsy
Copyrighted Material


Circulation of this edition outside the Indian subcontinent is UNAUTHORIZED
2e
DISTRIBUTED SYSTEMS
Principles and Paradigms
ANDREW S. TANENBAUM
MAARTEN VAN STEEN
ALWAYS LEARNING
PEARSON

Архитектура приложений

- Клиентские приложения

- Клиент-серверная архитектура (тонкий и толстый клиент)

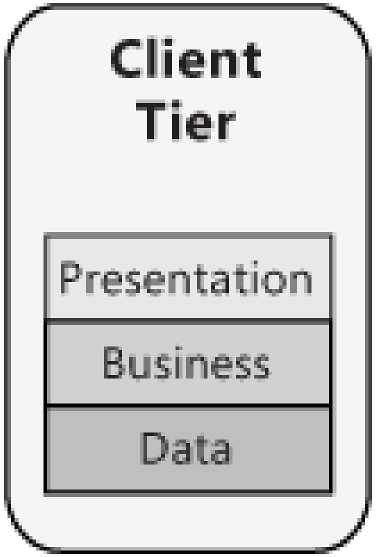
- Трех и многоуровневая архитектура

- Веб-приложения

Application Layers

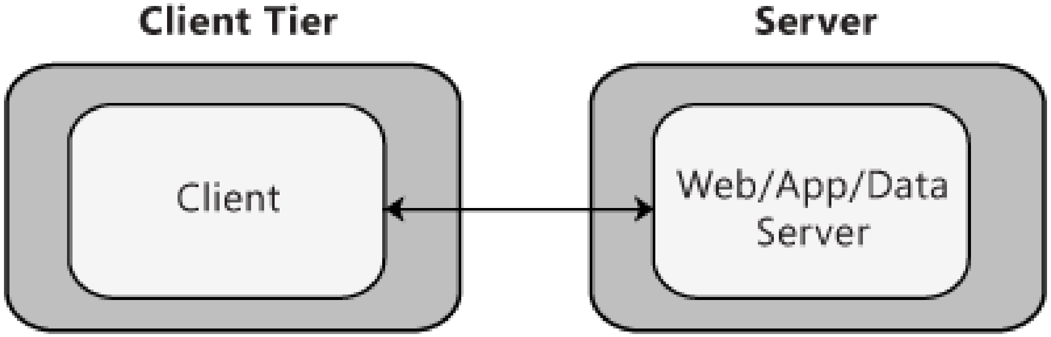
- Presentation
- Business
- Data

Stand-alone Deployment



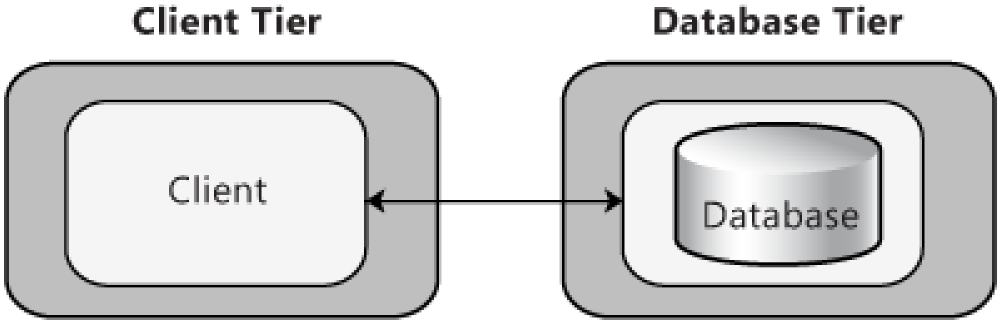
Client–server model

- The client–server model of computing is a distributed computing structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients
- The *client–server* characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services.

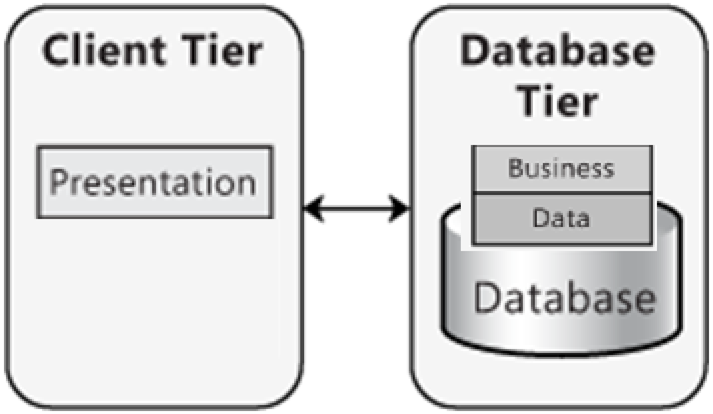


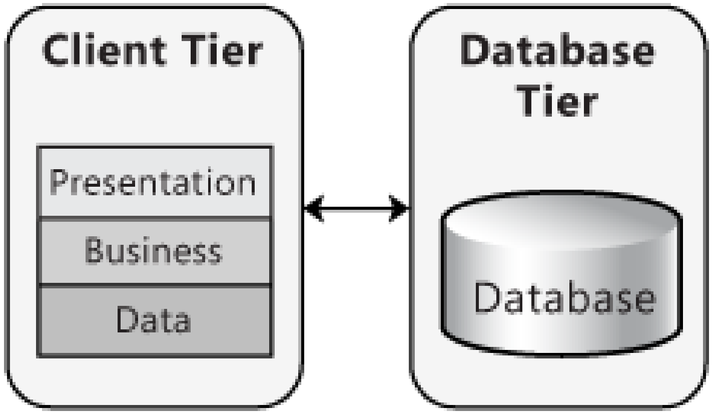
Client/Server

- Segregates the system into two applications, where the client makes requests to the server.
- In many cases, the server is a database with application logic represented as stored procedures.



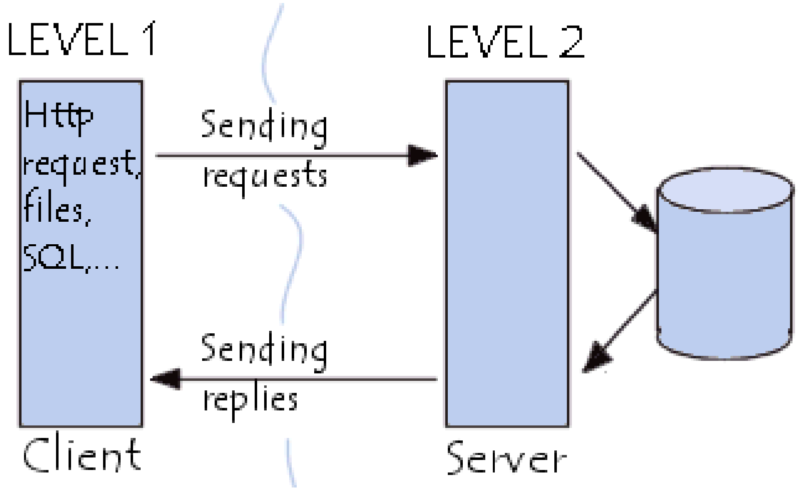
Thin Client vs Thick Client Architecture





Client and server communication

- Clients and servers exchange messages in a request-response messaging pattern: The client sends a request, and the server returns a response



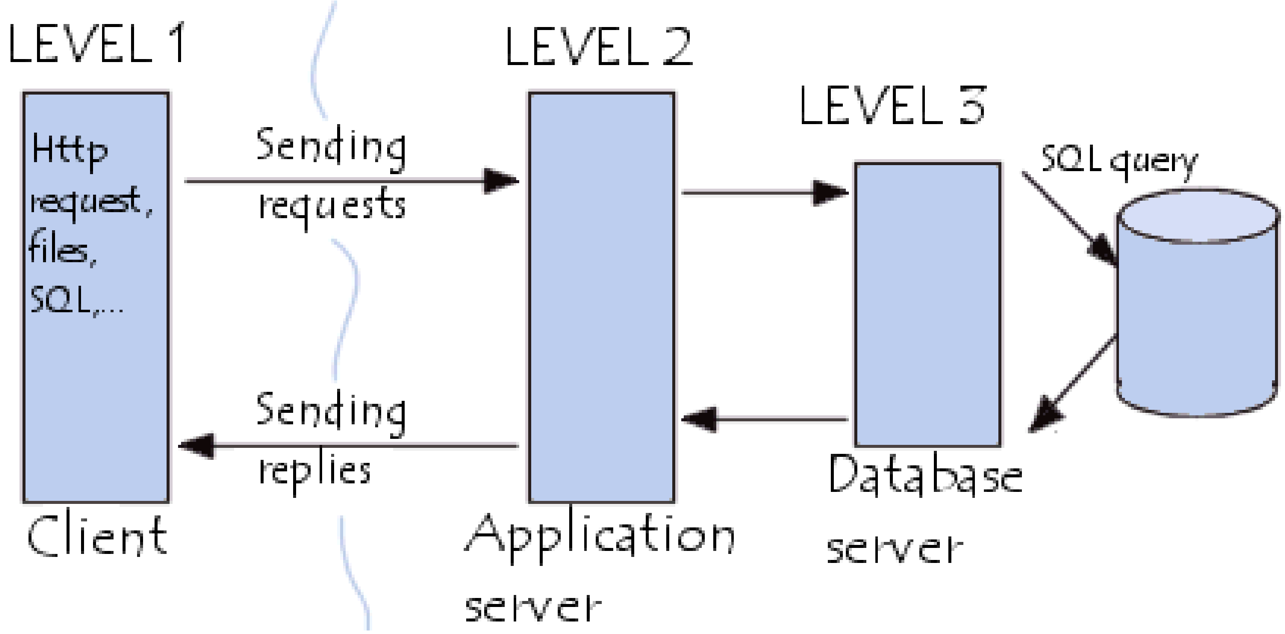
Client and server communication

- Clients and servers exchange messages in a request-response messaging pattern: The client sends a request, and the server returns a response
  - Synchronous communication
  - Asynchronous communication
- The language and rules of communication are defined in a communications protocol.

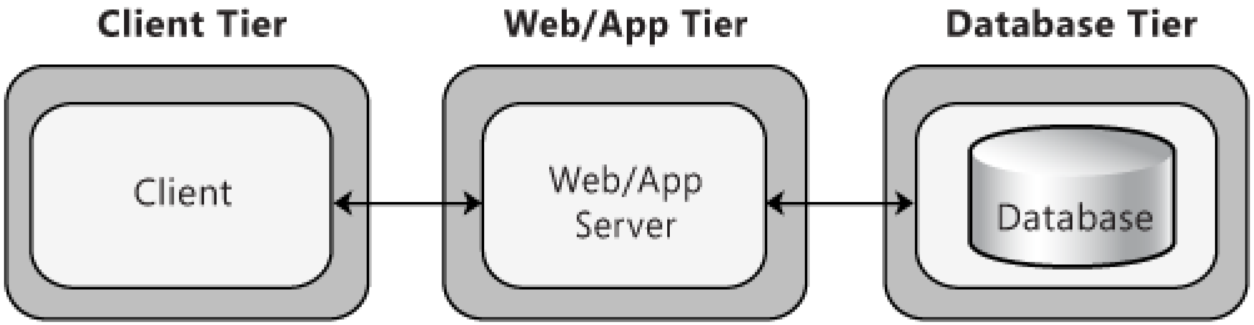
Client/Server Architectural Style

- **Client-Queue-Client systems**
  - This approach allows clients to communicate with other clients through a server-based queue. Clients can read data from and send data to a server that acts simply as a queue to store the data. This allows clients to distribute and synchronize files and information. This is sometimes known as a *passive queue* architecture.
- **Peer-to-Peer (P2P) applications**
  - Developed from the Client-Queue-Client style, the P2P style allows the client and server to swap their roles in order to distribute and synchronize files and information across multiple clients. It extends the client/server style through multiple responses to requests, shared data, resource discovery, and resilience to removal of peers.
- **Application servers**
  - A specialized architectural style where the server hosts and executes applications and services that a thin client accesses through a browser or specialized client installed software. An example is a client executing an application that runs on the server through a framework such as Terminal Services.

3-Tier Architecture



N-Tier Architecture



Client Tier

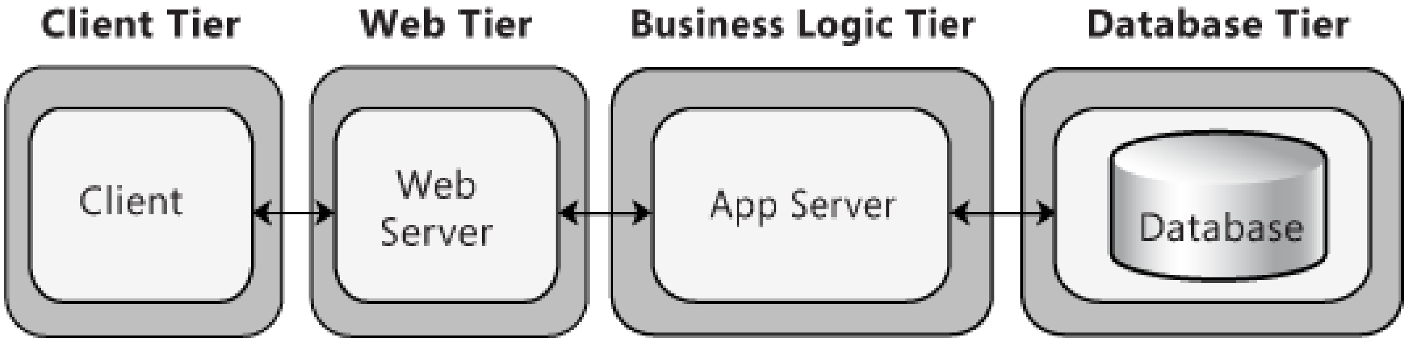
Web/App Tier

Database Tier

Client

Web/App Server

Database



Client Tier

Web Tier

Business Logic Tier

Database Tier

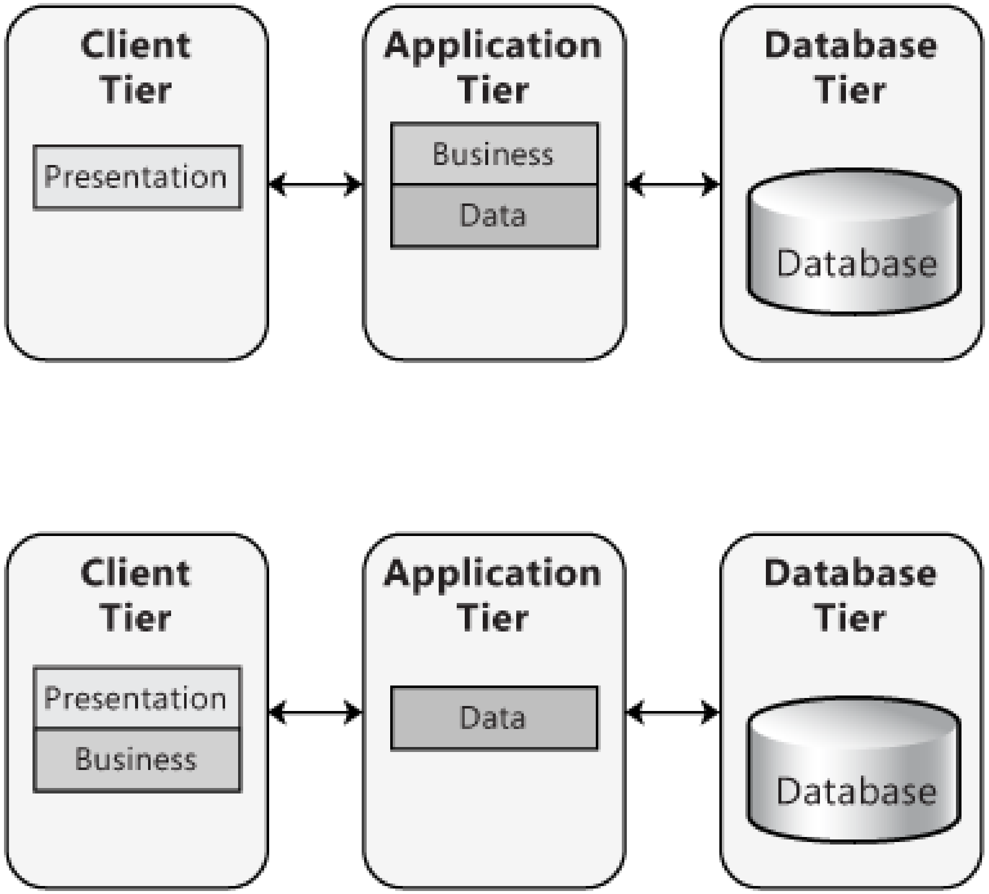
Client

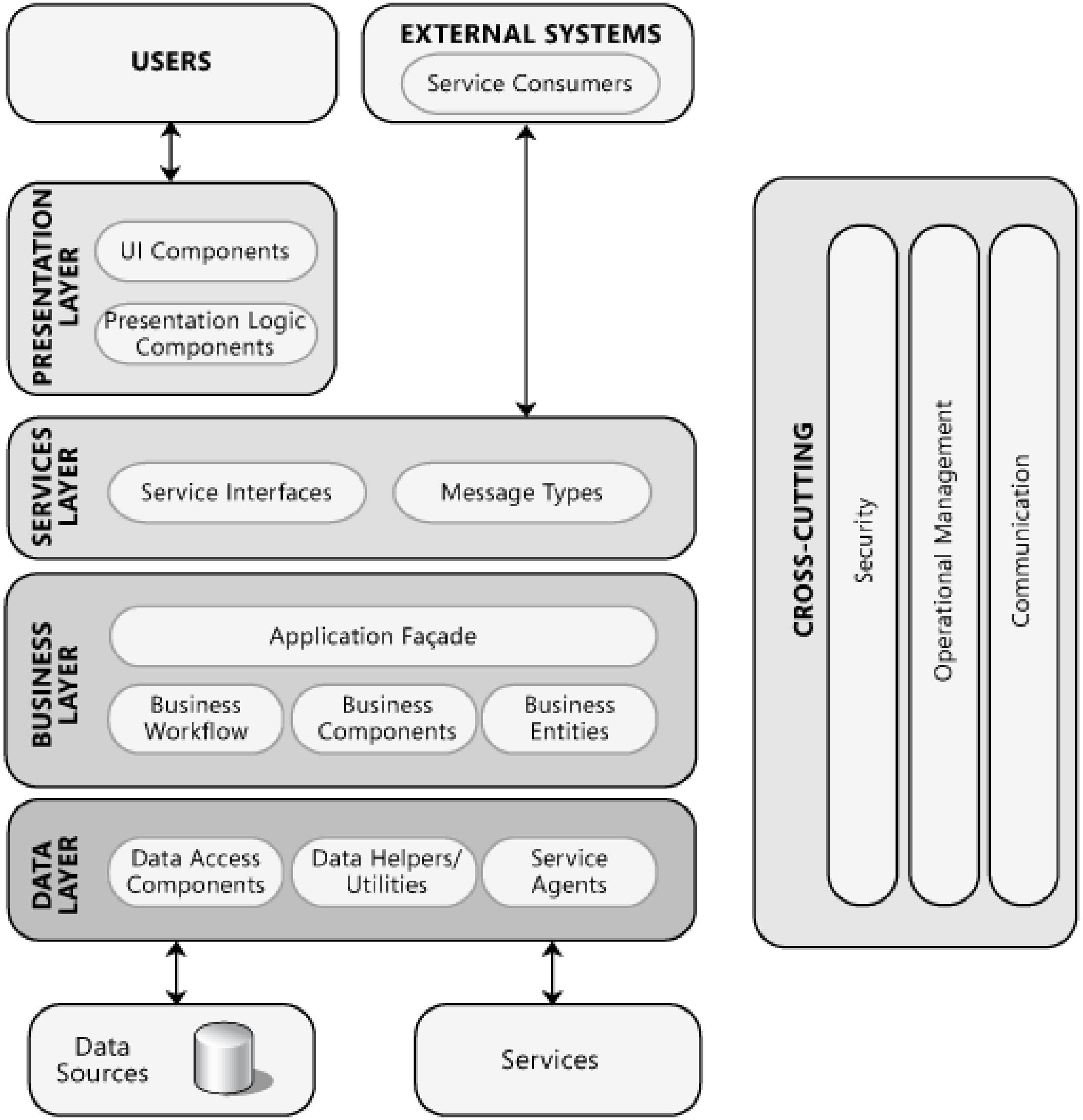
Web Server

App Server

Database

Thin vs Rich client




USERS
EXTERNAL SYSTEMS
Service Consumers
PRESENTATION LAYER
UI Components
Presentation Logic Components
SERVICES LAYER
Service Interfaces
Message Types
BUSINESS LAYER
Application Façade
Business Workflow
Business Components
Business Entities
DATA LAYER
Data Access Components
Data Helpers/ Utilities
Service Agents
Data Sources
Services
CROSS-CUTTING
Security
Operational Management
Communication