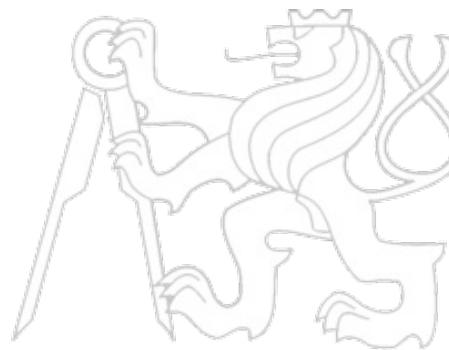


# Middleware Architectures 1

## Lecture 1: Information System Architectures

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • <https://vitvar.com>



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • <https://vitvar.com/lectures>



Modified: Sun Sep 21 2025, 20:05:30  
Humla v1.0

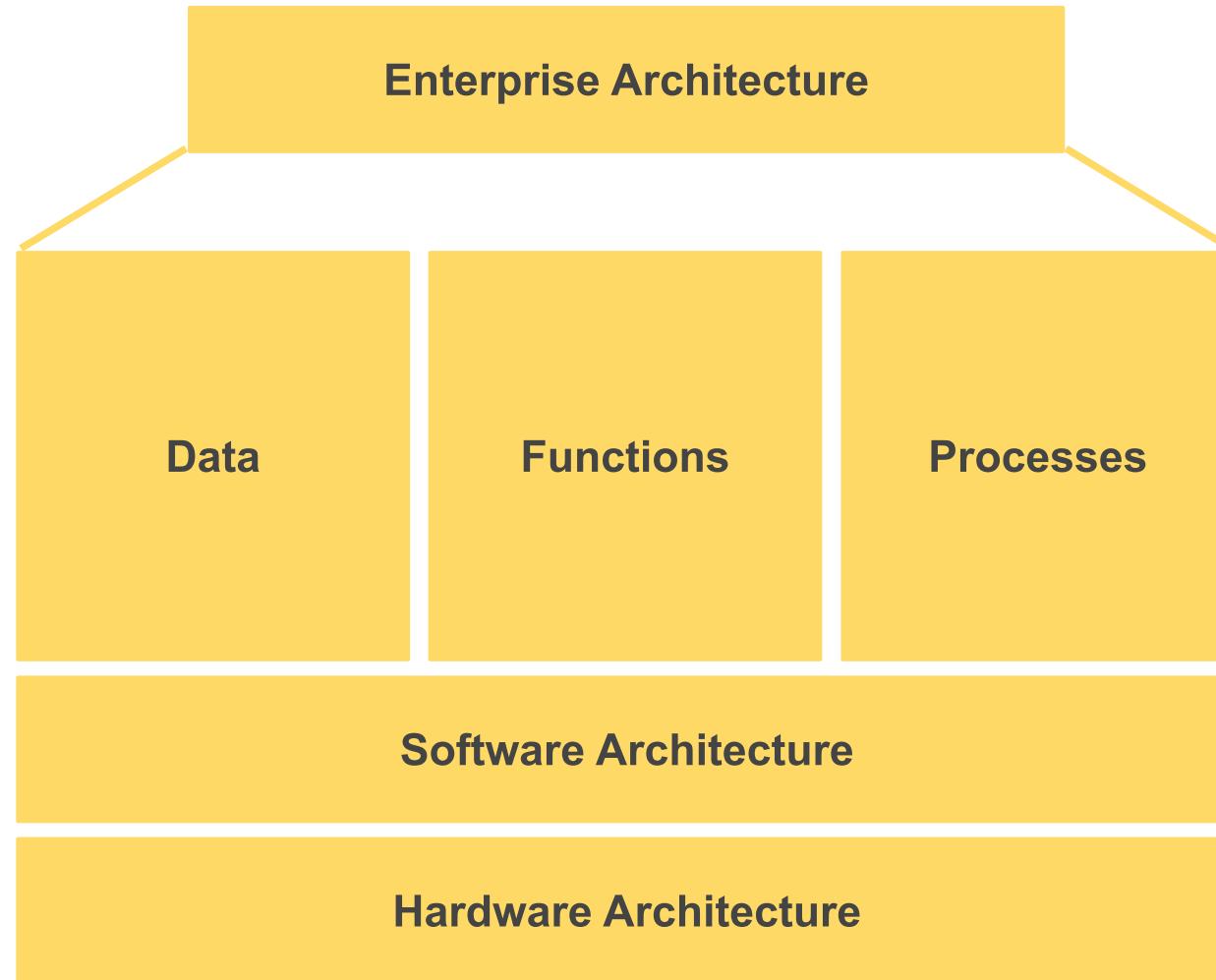
# Overview

- Architecture Overview
- Software Architecture

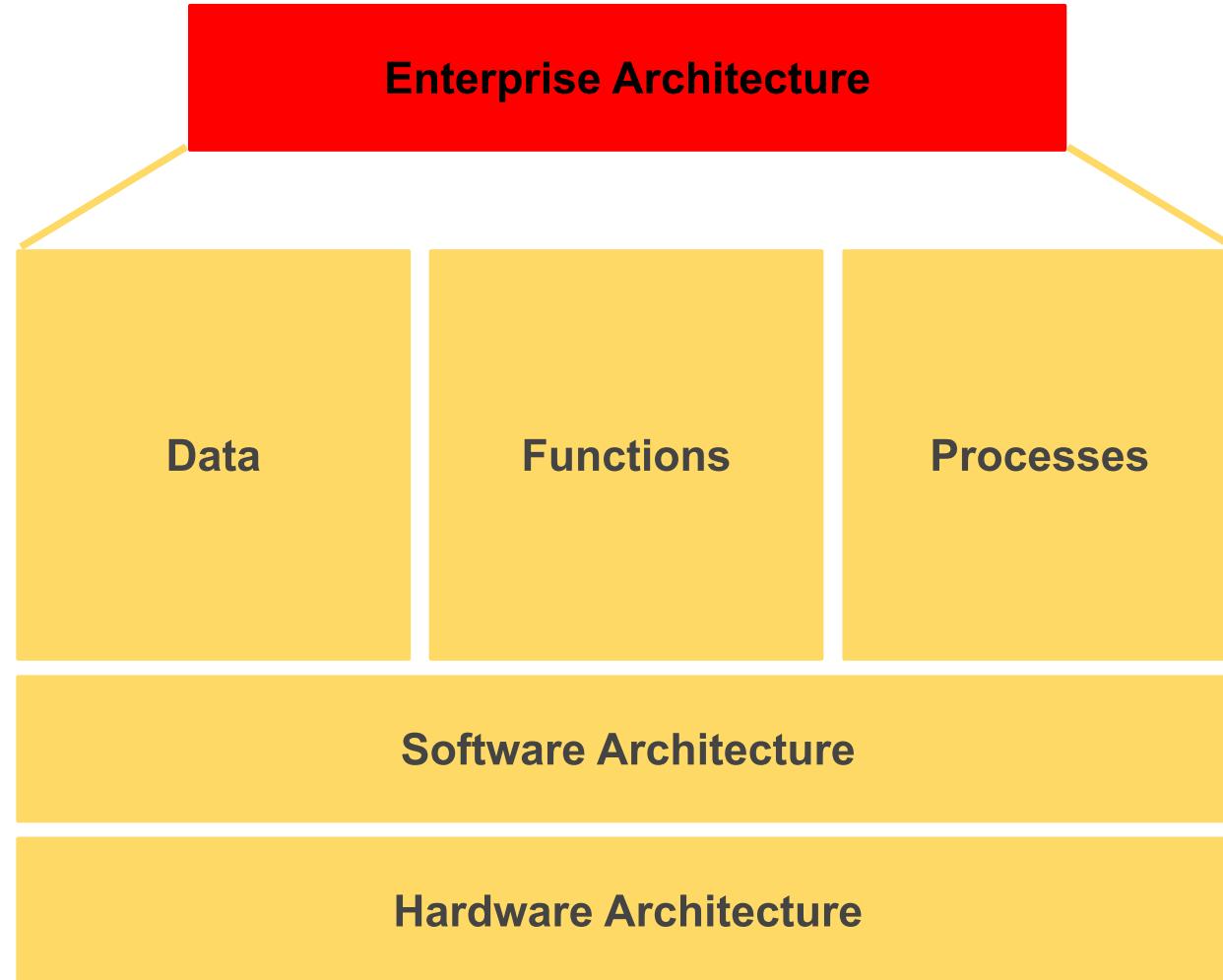
# Global Architecture

- A **structure** and a **behavior** of system's parts
- Complexity – views on the global architecture
  - *basic architectural views (also called dimensions): enterprise, data, functional, process, software, hardware.*
- Development
  - *basic methodology and actors:*
    - ~ analysis, design, implementation, testing, maintenance
    - ~ end-user, architect, developer, administrator
  - *basic architectural development levels:*
    - ~ conceptual, logical, physical.
- Global architecture and cloud computing
  - *data, functions, processes are application (domain) specific*
  - *software architecture defines a software platform*
  - *hardware architecture defines an infrastructure*

# Views



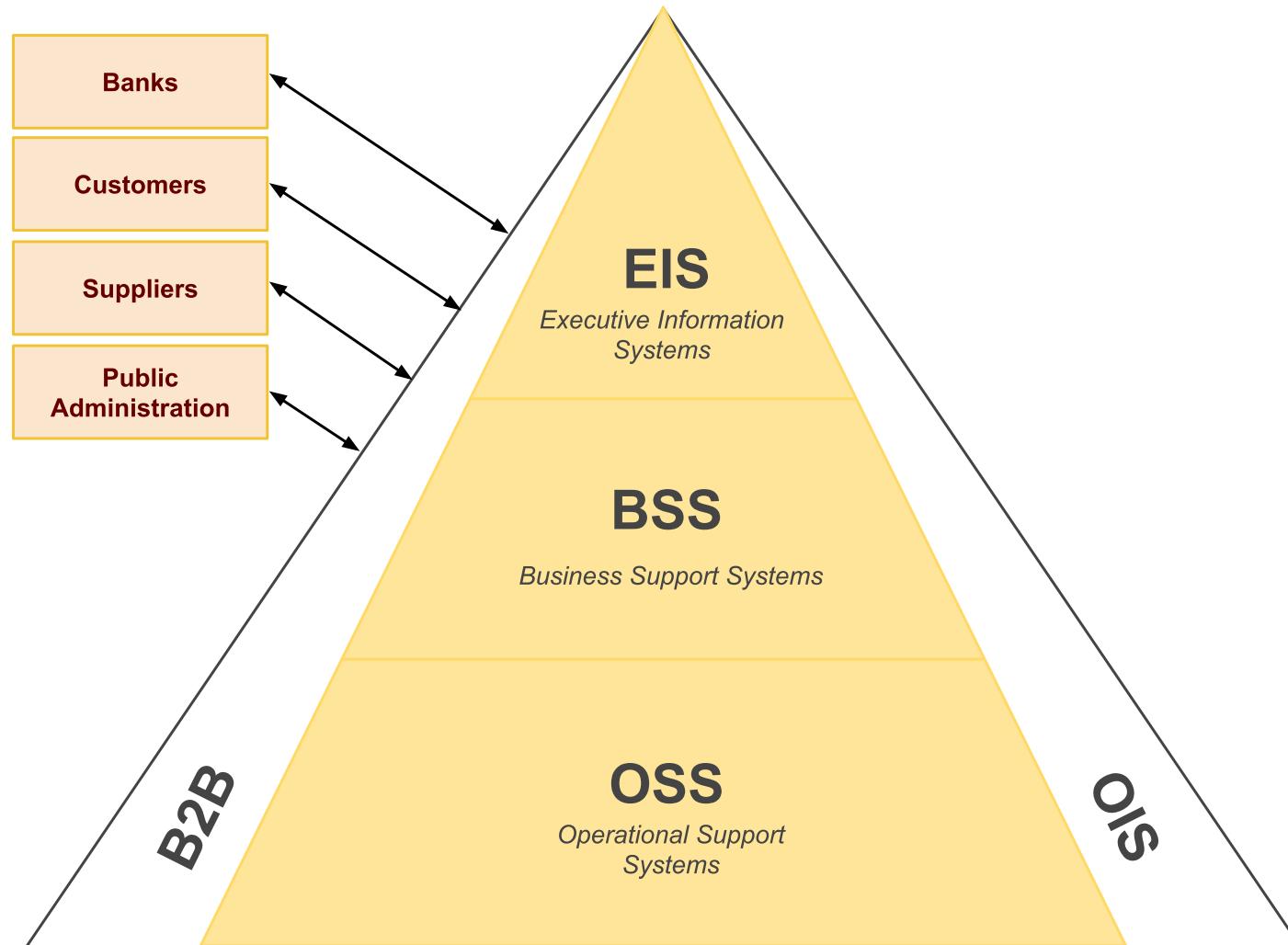
# Enterprise Architecture



# Enterprise Architecture Levels

- Defines a structure of an enterprise system
  - *Abstracts from data, functions, processes, software, hardware*
  - *divides enterprise system into functional blocks – applications*
    - *Order Management System (OMS)*
    - *Customer Relationship System (CRM)*
    - *Billing and Revenue Management (BRM)*
  - *applications correspond to domains such as sales, finance, procurement, production, etc.*
- Enterprise architecture levels
  - *Operational Support Systems (OSS)*
  - *Business Support System (BSS)*
  - *Executive Information Systems (EIS)*
  - *Office Information Systems (OIS)*
  - *Integration*
    - *Business-to-Business (B2B)*
    - *Enterprise Application Integration (EAI)*

# Enterprise Architecture Representation



# Organization Types in Enterprise Systems

- **Customer Organization**
  - **Primary need:** *Business process automation and optimization*
  - **Responsibilities:** *Define business requirements, acceptance criteria, change management*
  - **Key roles:** *Enterprise architect, business analysts, end users, IT administrators*
  - **Example:** *Bank implementing new CRM system for customer management*
- **Supplier Organization (System Integrator)**
  - **Primary need:** *Deliver tailored solutions meeting customer requirements*
  - **Responsibilities:** *Solution design, customization, implementation, support*
  - **Key roles:** *Solution architects, technical architects, developers, project managers*
  - **Example:** *Accenture implementing SAP for manufacturing company*
- **Vendor Organization (Technology Provider)**
  - **Primary need:** *Develop market-driven products and platforms*
  - **Responsibilities:** *Product roadmap, R&D, platform maintenance, market analysis*
  - **Key roles:** *Product managers, platform architects, developers, DevOps engineers*
  - **Example:** *Microsoft developing Azure cloud services*

# Architect Roles and Responsibilities

- **Enterprise Architect**
  - **Scope:** Organization-wide architecture strategy and governance
  - **Focus:** Business-IT alignment, application portfolio, data architecture
  - **Deliverables:** Enterprise architecture blueprints, technology roadmaps
  - **Standards:** TOGAF, industry-specific (eTOM for telecom)
- **Solution Architect**
  - **Scope:** End-to-end solution design for specific business problems
  - **Focus:** Requirements analysis, system integration, functional design
  - **Deliverables:** Solution design documents, integration patterns, data flows
  - **Skills:** Business analysis, system design, stakeholder management
- **Technical Architect**
  - **Scope:** Technology implementation and infrastructure design
  - **Focus:** Performance, scalability, security, technology selection
  - **Deliverables:** Technical specifications, deployment guides, performance benchmarks
  - **Skills:** Deep technical expertise, cloud platforms, DevOps practices

# Modern Technical Architect Roles

- **Cloud Architect**
  - *Cloud-native solutions, migration strategies, multi-cloud designs*
  - *AWS, Azure, GCP certifications and expertise*
- **Security Architect**
  - *Zero-trust architecture, compliance frameworks (GDPR, SOX)*
  - *Identity management, encryption, threat modeling*
- **Data Architect**
  - *Data lakes, data warehouses, real-time analytics*
  - *Data governance, privacy, master data management*
- **API Architect**
  - *API strategy, microservices design, API governance*
  - *REST, GraphQL, event-driven architectures*
- **DevOps Architect**
  - *CI/CD pipelines, infrastructure as code, monitoring*
  - *Kubernetes containerization, observability platforms*

# Overview

- Architecture Overview
- Software Architecture
  - *Types, Separation of Concerns, Interface*
  - *Client/Server Architectures*

# Software Architecture Types

- Centralized – Client/Server (C/S)
  - *Central server, a bunch of clients*
  - *monolithic, two-, three-, multi-tier architectures*
  - *Single point of failure!*
    - *when a server fails the whole system fails*
    - *need for a scalable and **highly reliable** server-side solutions*
  - *Enterprise systems (mostly) use centralized solutions*
    - *But, enhanced with peer-to-peer principles*
- Decentralized – Peer-to-Peer (P2P)
  - *Reliability*
    - *when a node fails, other nodes take up its function*
  - *Scalability*
    - *multiple nodes can share the load*
    - *such as messaging systems in enterprise systems*

# Separation of Concerns

- Separation of Concerns
  - *also called Separation of Layers*
  - *Concern – any piece of interest (part) in the application*  
→ *concerns should overlap in functionality as little as possible*
  - *Basic application concerns: data manipulation, data integrity, application logic, user-interactions*
  - *Software architecture separates concerns into layers*  
→ *presentation, application, data*
- Interface
  - ~ *agreement on "how layers should communicate"*
  - *most important artifact in Separation of Concerns*
  - *If an interface is in place, application development and innovation can happen **independently** at each layer*

# Interface

- Definition
  - *Agreement (contract) between two or more layers during communication*
- May be achieved by
  - *Through standards (accepted or enforced),*
  - *Through a social agreement during design*
  - *A dominant position of a technology on the market*
- Interface includes subsets of domain architectures
  - *Subsets that are subject to communication between layers*
  - ***data*** – defines communication language (*syntax, semantics*),
  - ***functions*** – defines entry points (*operations*),
  - ***processes*** – defines valid states and transitions between them
  - ***technical details*** – protocols, ports, IP addresses, etc.

# Interface Example: REST API

- **Data** interface: JSON format
- **Function** interface: HTTP methods
  - *GET /customers/{id}* - retrieve customer
  - *POST /customers* - create customer
  - *PUT /customers/{id}* - update customer
- **Process** interface: state transitions
- **Technical** interface: HTTPS, port 443, authentication

# Overview

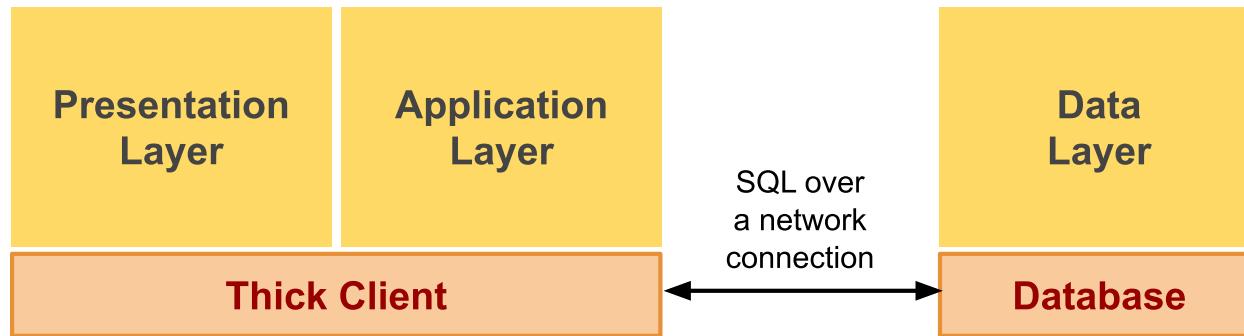
- Architecture Overview
- Software Architecture
  - *Types, Separation of Concerns, Interface*
  - *Client/Server Architectures*

# Monolithic Architecture



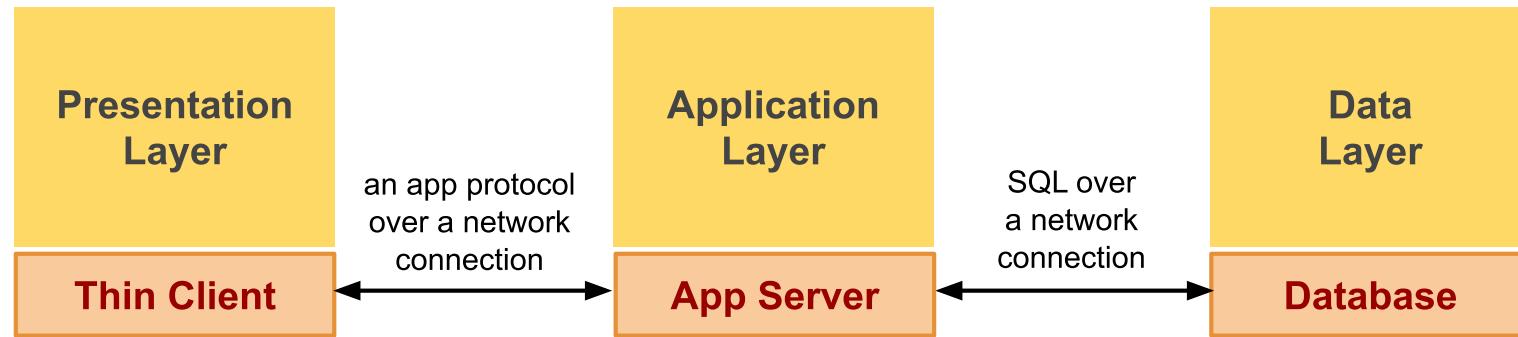
- All layers on a single machine
  - *usually non-portable apps; specific OS*
  - *first types of computer systems, typical for 90-ties*
  - *single-user only; standalone apps, minimal integration*
  - *technologies: third-gen programming languages, local storage systems*
- Drawbacks
  - *hard to maintain (updates, distribution of new versions)*
  - *data security issues*
  - *performance and scalability issues*

# Two-tier Client/Server Architecture



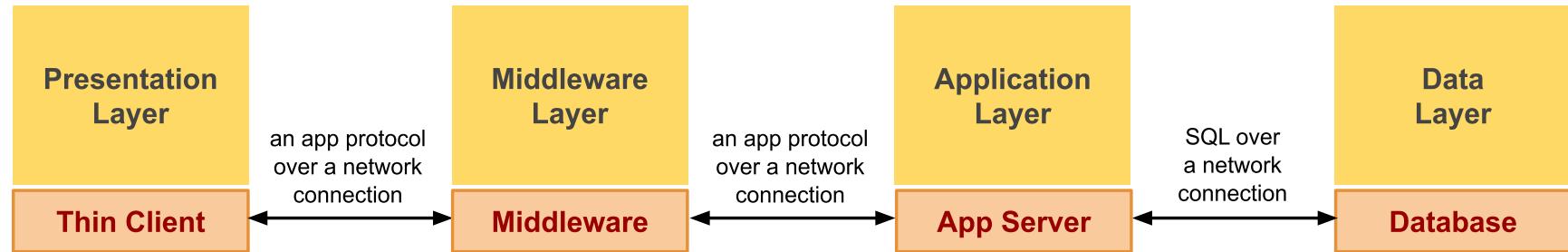
- Presentation and app layers separated with data
  - *Thick client – desktop application, OS-dependent*
  - *Data on a separate server (DBMS)*
  - *Multi-user system, all sharing a database*
  - *Storage system of high performance, transactions support*
  - *SQL technology; native OS desktop application*
- Drawbacks
  - *Thick client hard to maintain (reinstallation with every update)*
  - *No app logic sharing (only through copies)*
  - *Data-oriented integration (integrity in the app logic!)*

# Three-tier Client/Server Architecture



- All layers on separated machines
  - *Thin client – desktop application or interpreted code*
  - *Multi-user system, all sharing app logic and a database*
  - *App server of high performance, scalability*
- Drawbacks
  - *Spaghetti integration*
  - *Limited, single app server scalability*

# Multi-tier Client/Server Architecture



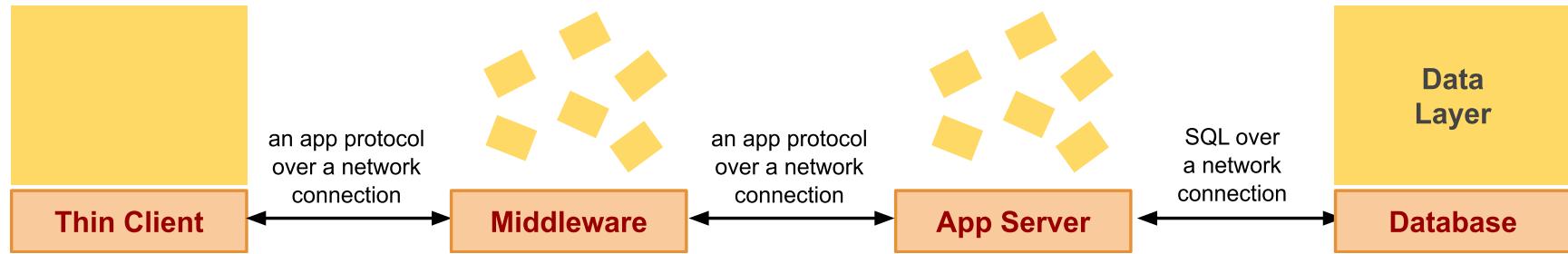
- Additional middleware layer
  - *provides value-added services for communications*
  - *individual servers or a compact solution (e.g., Enterprise Service Bus)*
- Drawbacks
  - *Monolithic apps are difficult to scale as a whole*
  - *Deployment overhead*
  - *A single technological environment for all app functions in the monolith*

# Client/Server Architecture (microservices)



- Microservice architecture
  - *Middleware, app and DB monoliths are microservice architecture*
  - *Improved scalability and technology neutrality of app components*
- Service orchestration layer
  - *Kubernetes (K8s)*
  - *Large K8s cluster for all, middleware, app, DB*
  - *Separate K8s cluster*

# Client/Server Architecture (microservices)



- Not-a-microservice Architecture
  - *Monoliths deployed to Kubernetes cluster*
  - *Improved Deployments (via container images)*
  - *Improved fail-over*
  - *Not cheaper (Kubernetes costs come into play)*

# Types of Middleware

- Scalability
  - *They help to achieve high performance through better scalability*
  - *Messaging Servers (message queues, publish/subscribe)*
  - *Load Balancers*
  - *Proxy servers, reverse proxy*
- Functional
  - *They help to achieve more flexible integration*
  - *Process servers*
  - *Repositories, registries of services/components*
  - *Mediators – data interoperability, process interoperability, technical interoperability (SOAP server)*
  - *Monitors for analytics of apps usages*
- Security
  - *Firewalls, Gateways, ...*