

# Middleware and Web Services

## Lecture 1: Introduction to Architectures

**doc. Ing. Tomáš Vitvar, Ph.D.**

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



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • <http://vitvar.com/courses/mdw>



Modified: Sun Oct 15 2017, 21:54:28  
Humla v0.3

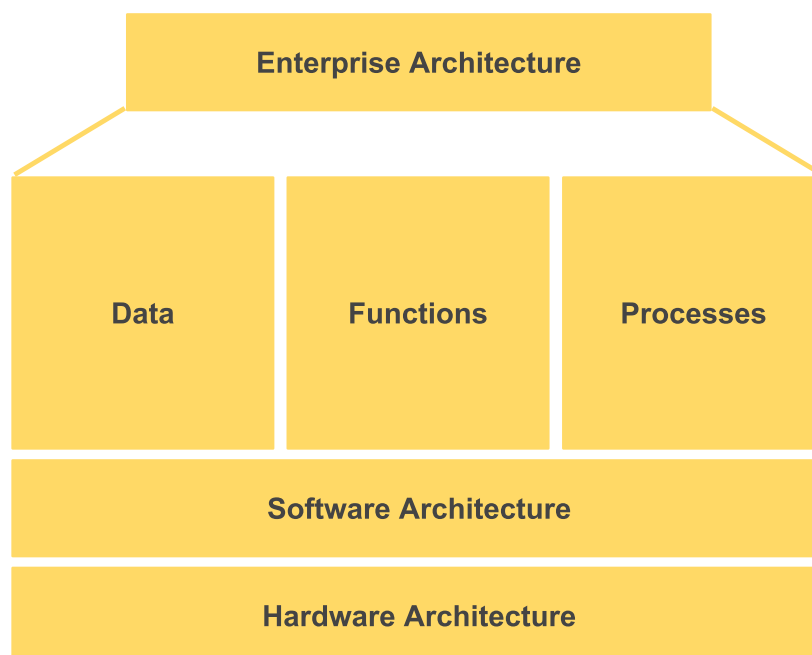
## Overview

- **Architecture Overview**
- Data, Functions and Processes
- Software Architecture

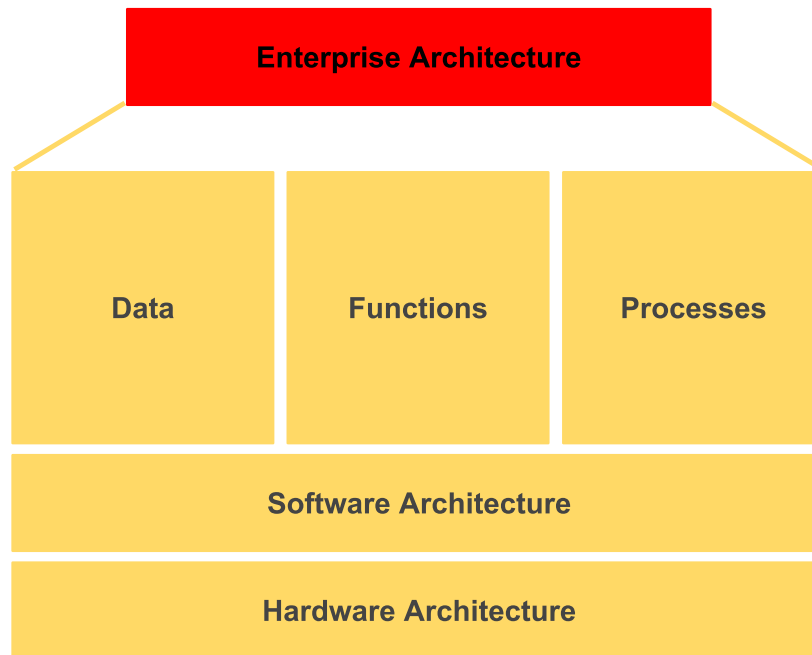
# Global Architecture

- 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



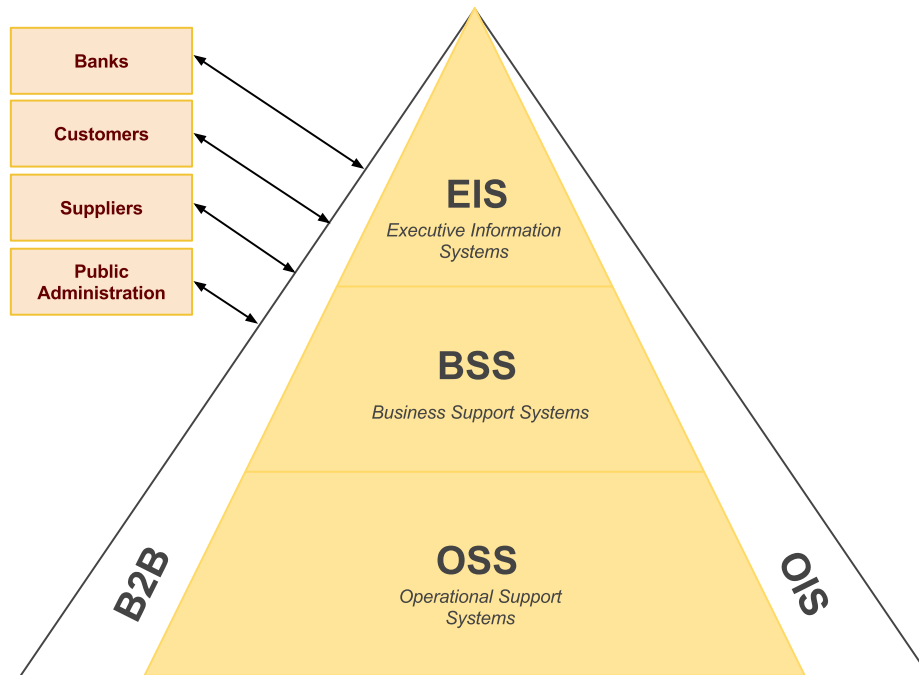
# Enterprise Architecture



## 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

- Customer
  - *user needs: support for business processes*
  - *defines business requirements*
  - *roles: enterprise architect, developers, admins, users*
- Supplier (enterprise system/application provider)
  - *solutions and customization according to requirements*
  - *roles: technical and solution architects, developers, admins*
- Vendor (technology provider)
  - *product development according to market needs*
  - *roles: product managers, developers, reference users*

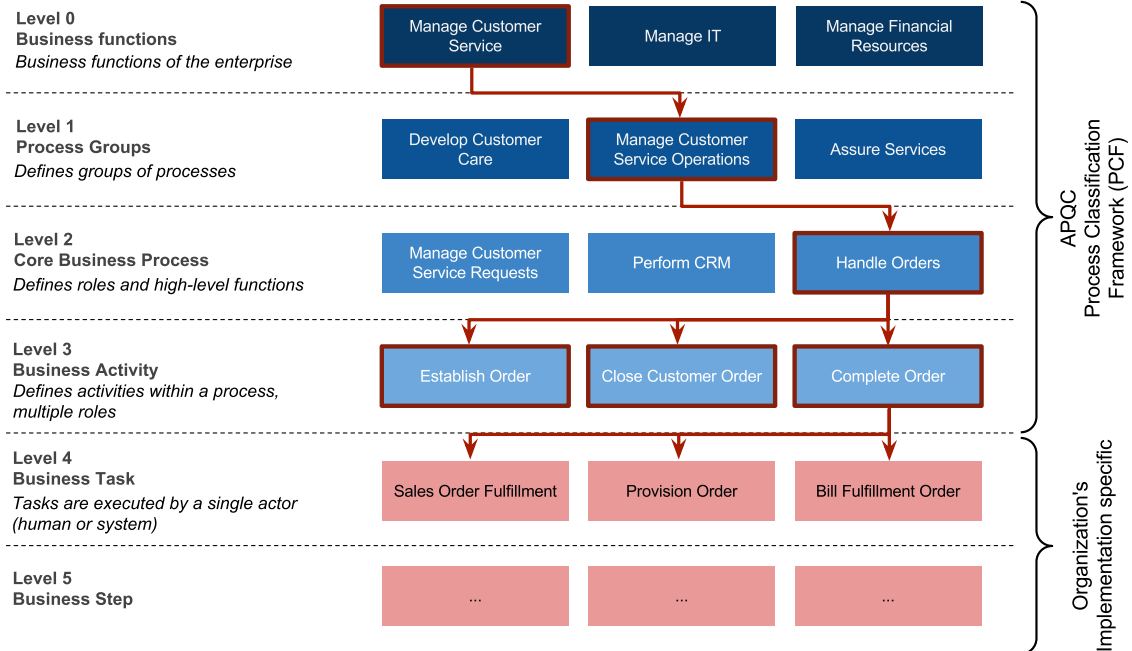
## Architect Roles

- Technical Architect
  - *Technical architecture design*
  - *technology configurations, performance*
- Solution Architect
  - *Requirements gathering, analysis*
  - *Solution design (data, functions, process)*
- Enterprise Architect
  - *High-level enterprise architecture design*
    - *Applications, processes, data models*
  - *Should be aligned with industry standards*
    - *APQC – American Productivity & Quality Center (Process Classification Framework)*
    - *TM Forum – eTOM – Enhanced Telekom Operations Map (Business Process Framework)*

## Overview

- Architecture Overview
- Data, Functions and Processes
  - *Integration*
- Software Architecture

# Process Classification Framework

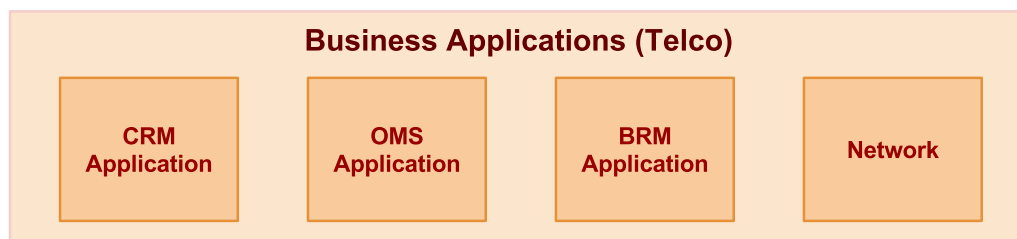


## Order Process Example in Telco

- Order to Cash Process (O2C)
  - end-to-end (E2E) order process



- Involved applications
  - integrated applications



## Syntax and Domain Semantics

- Syntax
  - Data format, representation, serialization
  - Various languages at various architectural levels:  
*XML, JSON, Class/object models in a specific programming language, SQL, DB native structures, ...*
  - They have formal grammars, can be checked for the correct syntax
- Domain semantics
  - Meaning of terms in a domain they are being used
  - We understand meaning of terms:
    - Through syntax by using the natural language
    - Through some **agreement** among users of the terms
  - Every applications can use different semantics
    - Need to mediate data from one application to another

## Simplified Order Type Example



## Overview of Representations

- Languages to realize architectures
  - *development levels* × *architectural views*



### Abbreviations

BPMN=Business Process Management Notation

BPEL=Business Process Execution Language

PIP=Partner Interface Process

UML=Unified Modeling Language

### Legend

Architectural view

Example UML diagrams

Example implementation technologies

Development Level

Example Business Standards

Physical Operational Environment

## Overview

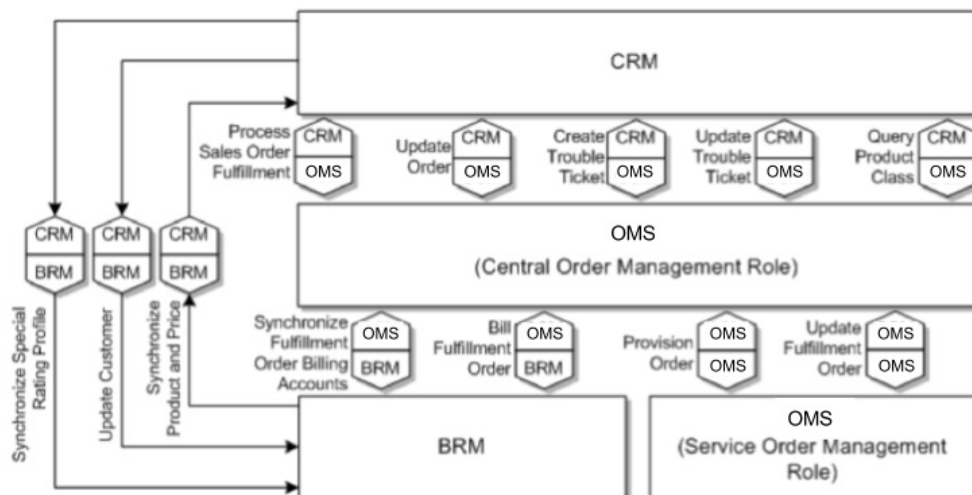
- Architecture Overview
- Data, Functions and Processes
  - *Integration*
- Software Architecture



## Integrating Applications

- Intra-Enterprise Integration
  - Applications exist in a specific area
  - Functions and data often overlap across areas
  - There is a need to integrate applications within enterprise:
    - Applications need to share the same data that are often in different formats.
    - Applications need to communicate – a result of one process may trigger another one.
- Inter-Enterprise Integration
  - Also called Business-to-Business Integration (B2B)
  - Automation support for communication and collaboration among enterprises
  - For example, B2B automates customers' orders processing, tracking orders, etc.

## Integration Example – O2C



## Integration Issues

- Key to integration = **interface**
  - *standards – data, functions, processes, technical aspects*
    - *enterprise standards, committee standards*
    - *unified environment from a single vendor*
  - *mediators*
    - *where standard do not work out*
- Data
  - *Message exchange formats, data representation*
    - *often standardized*
  - *Semantics of data*
    - *also standardized, more difficult*
- Functions and processes
  - *how apps' functionalities should be consumed and orchestrated, protocols, naming issues*
  - *A service concept*

## Overview

- Architecture Overview
- Data, Functions and Processes
- Software Architecture
  - *Types, Separation of Concerns, Interface*
  - *Client/Server Architectures*

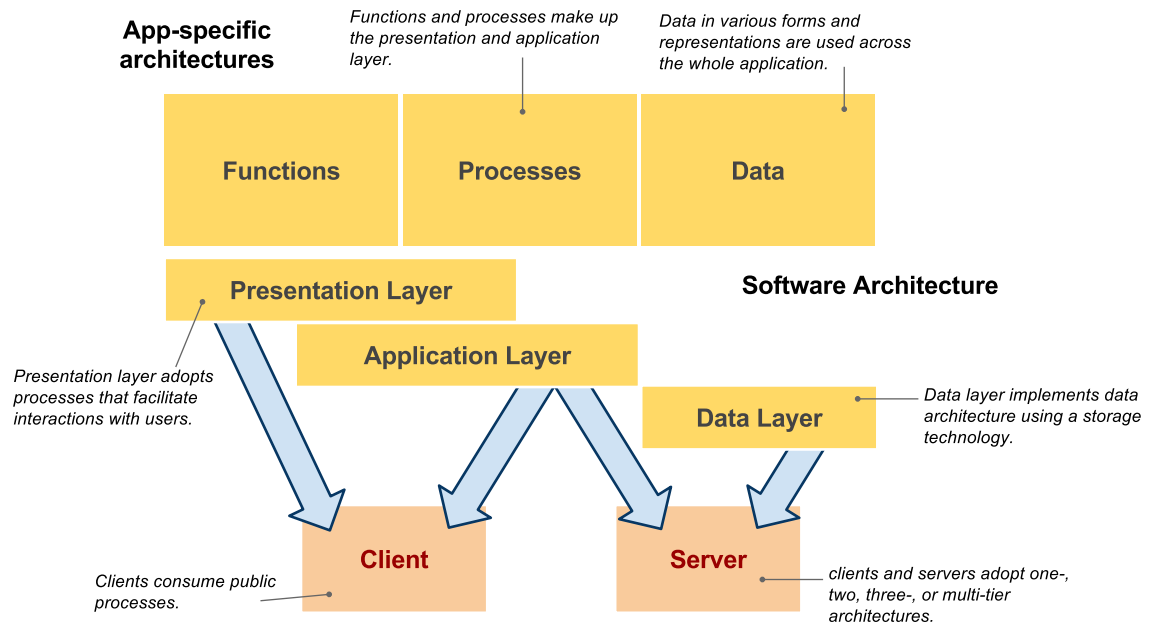
## Software Architecture Types

- **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*

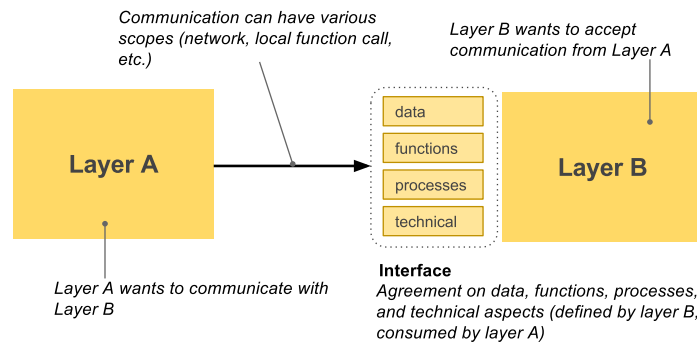
# Software Architecture Layers



## Interface

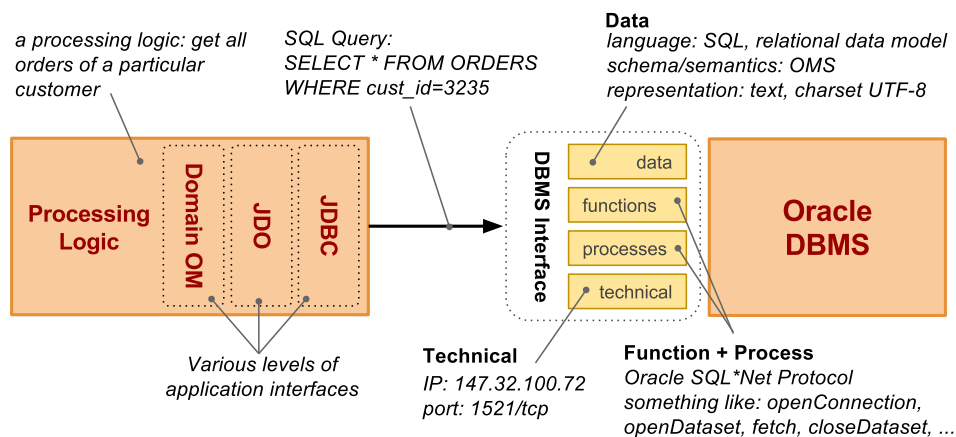
- **Definition**
  - *Agreement (contract) between two or more layers during communication*
- **May be achieved by**
  - *Through standards (accepted or enforced),*
  - *Through a social agreement during design*
  - *A dominant position of a technology on the market*
- **Interface includes subsets of 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 Scope



- Scope
  - local: layer A and B on one machine
  - network: layer A and B on separated machines (client/server)
- Application Programming Interface (API) ~ local scope
  - OS/complex apps' interface for extension development
- Web Services and Web API (network scope)
  - Web API is a new term – API of Web Applications

# Complex Interfaces

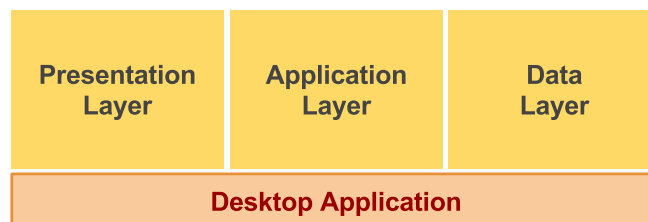


- More levels of interfaces
  1. DBMS native interface
  2. JDBC – universal connectors for various DBMS systems
  3. JDO – mapping of Java classes to data objects
  4. Domain Object Model (OM) – app-specific (~API, SDK)
    - try to be as universal as possible; cover many technologies

## Overview

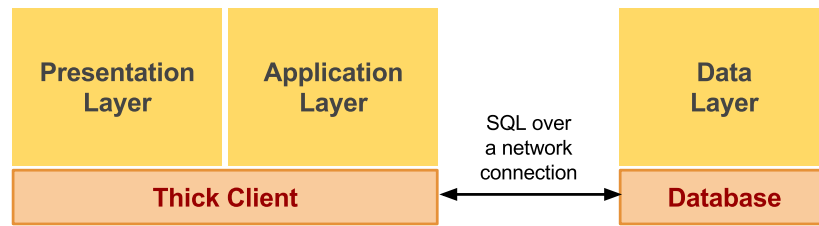
- Architecture Overview
- Data, Functions and Processes
- Software Architecture
  - *Types, Separation of Concerns, Interface*
  - *Client/Server Architectures*

## Monolithic Architecture



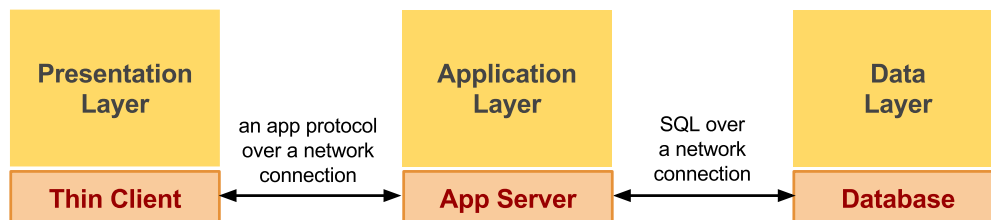
- All layers on a single machine
  - *usually non-portable apps; specific OS*
  - *first types of computer systems, typical for 90-ties*
  - *single-user only; standalone apps, minimal integration*
  - *technologies: third-gen programming languages, local storage systems*
- Drawbacks
  - *hard to maintain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

## Two-tier Client/Server Architecture



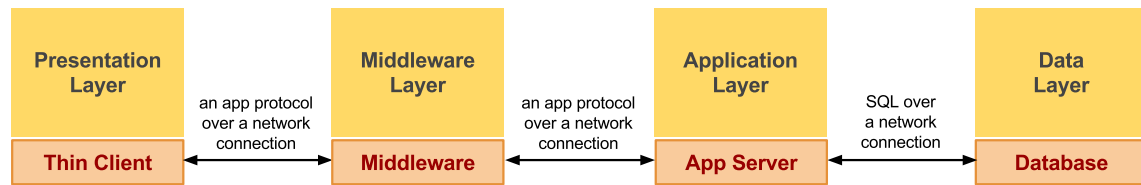
- Presentation and app layers separated with data
  - *Thick client – desktop application, OS-dependent*
  - *Data on a separate server (DBMS)*
  - *Multi-user system, all sharing a database*
  - *Storage system of high performance, transactions support*
  - *SQL technology; native OS desktop application*
- 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 (see [Lecture 0](#))*
  - *Limited, single app server scalability*

## Multi-tier Client/Server Architecture



- Additional middleware layer
  - *loose coupling, shift towards peer-to-peer architectures*
  - *provides value-added services for communications*
  - *individual servers or a compact solution (e.g., Enterprise Service Bus)*

## 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, ...*