university of koblenz
Computer Science

2. Software Architecture

Engineering Web and Data-intensive Systems

Dr. Volker Riediger - Winter Term 2022/23

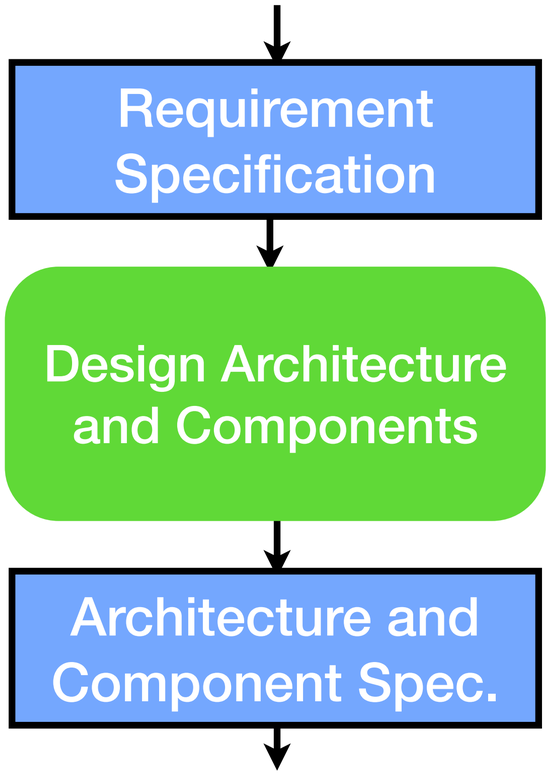
ist Institute for Software Technology

Software Architecture

- Quality Goals
- Principles
- What is Software Architecture?
- Viewpoint, View, Style, Pattern
- Some patterns in Web Applications

Image: colourbox.de

Related Activities of the SW Lifecycle



Manage Configurations

Assure Quality

Software Architecture

Some Definitions

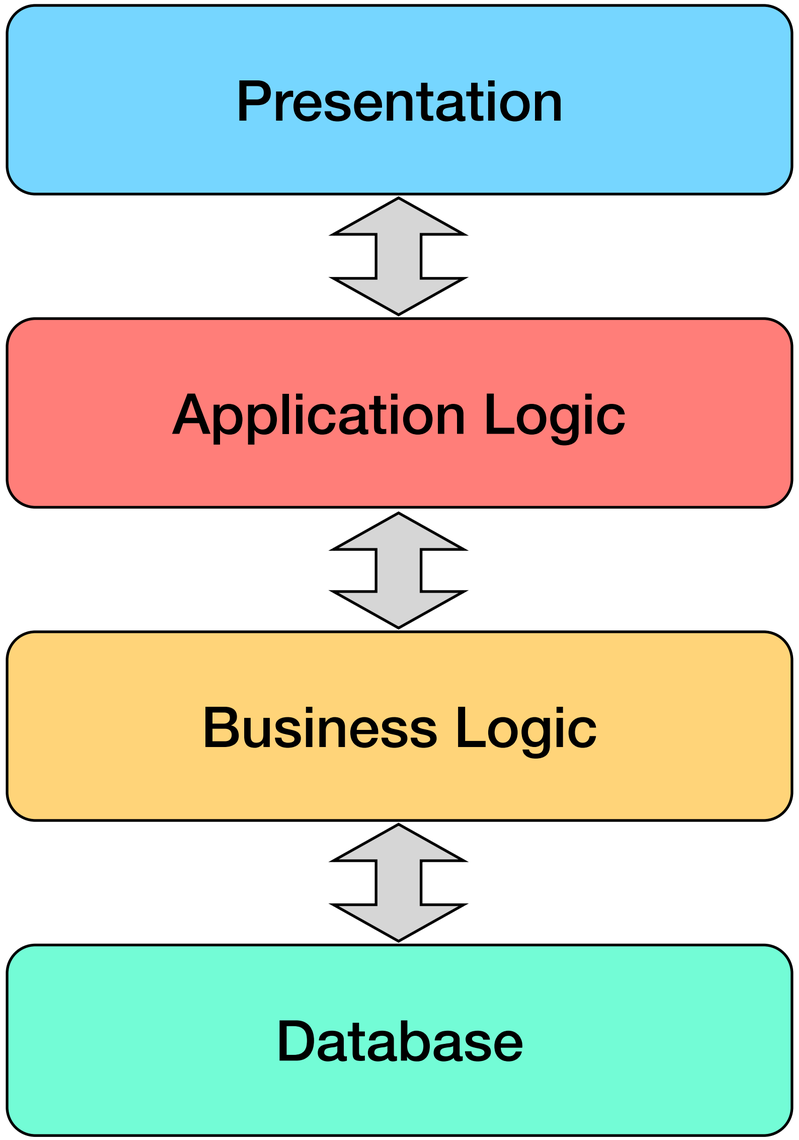
- „Software Architecture is components and connectors.“

- A **Software Architecture** is a decomposition of a system into its components and their relations, including relations to the environment of the system. The architecture defines rules and policies that determine the design and the evolution of the system.

- More definitions:
  ⇢ What is your definition of Software Architecture? [SEI 2017]

Some examples for software / system architecture diagrams

4 Layer Architecture



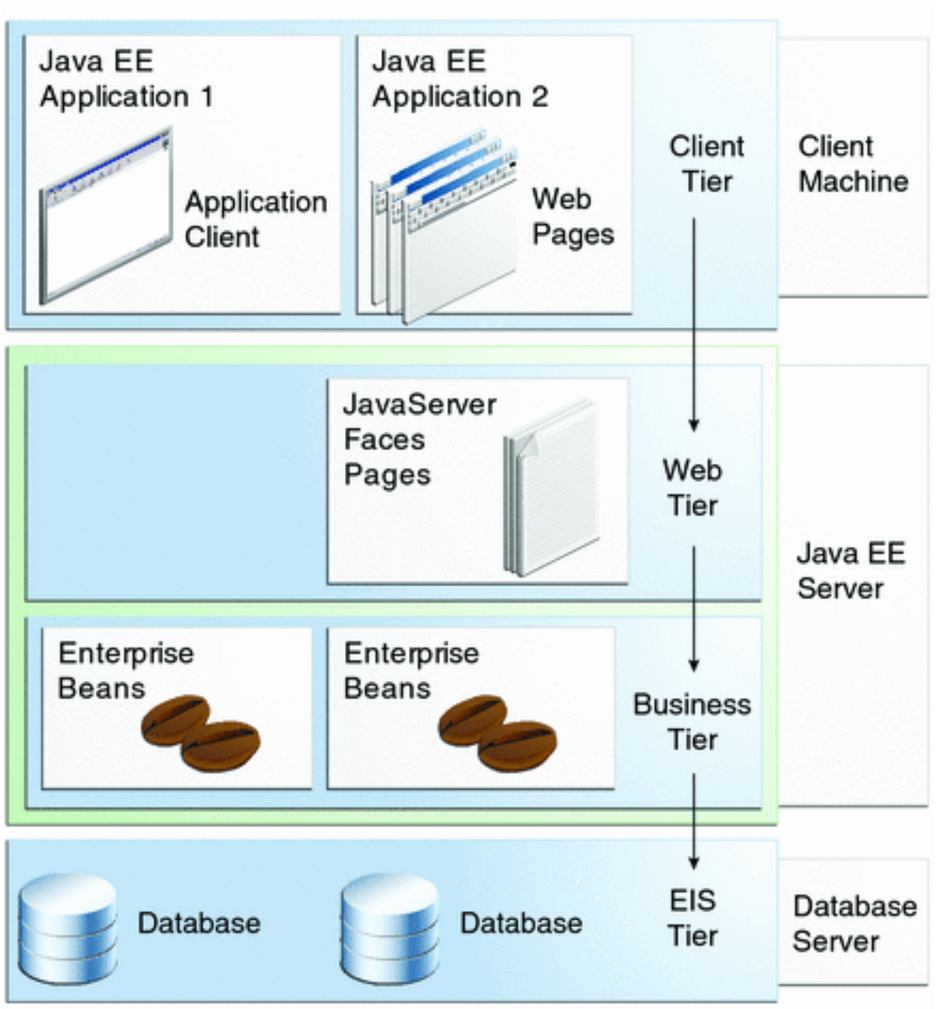
Visualization of values, Input areas, graphics, interaction with I/O devices

Control of sequence of actions, navigation, validation of input values, prevention of unauthorized access, conversion of values

Realization of basic application functions, interfaces to other systems, prevention of unauthorized access

Persistent storage of data, backup, transaction management, efficient access…

Java EE coarse grained



[JavaEE Tutorial, Orcacle Inc.]

INTERNATIONAL DATA SPACES ASSOCIATION

A TRUSTWORTHY ARCHITECTURE FOR THE DATA ECONOMY

The IDS provides self-determined control between all imaginable data endpoints

INTERNATIONAL DATA SPACES APPROACH

Endless Connectivity
Standard for data flows between all kinds of data endpoints

Trust between different security domains
Comprehensive and audit-proo security functions providing a maximum level of trust

Governance for the data economy
Usage control and enforcement for data flows and assignments of data

MISSION STATEMENT

Secure Data Exchange
It forms the basis for a variety of certifiable software solutions, smart services ...

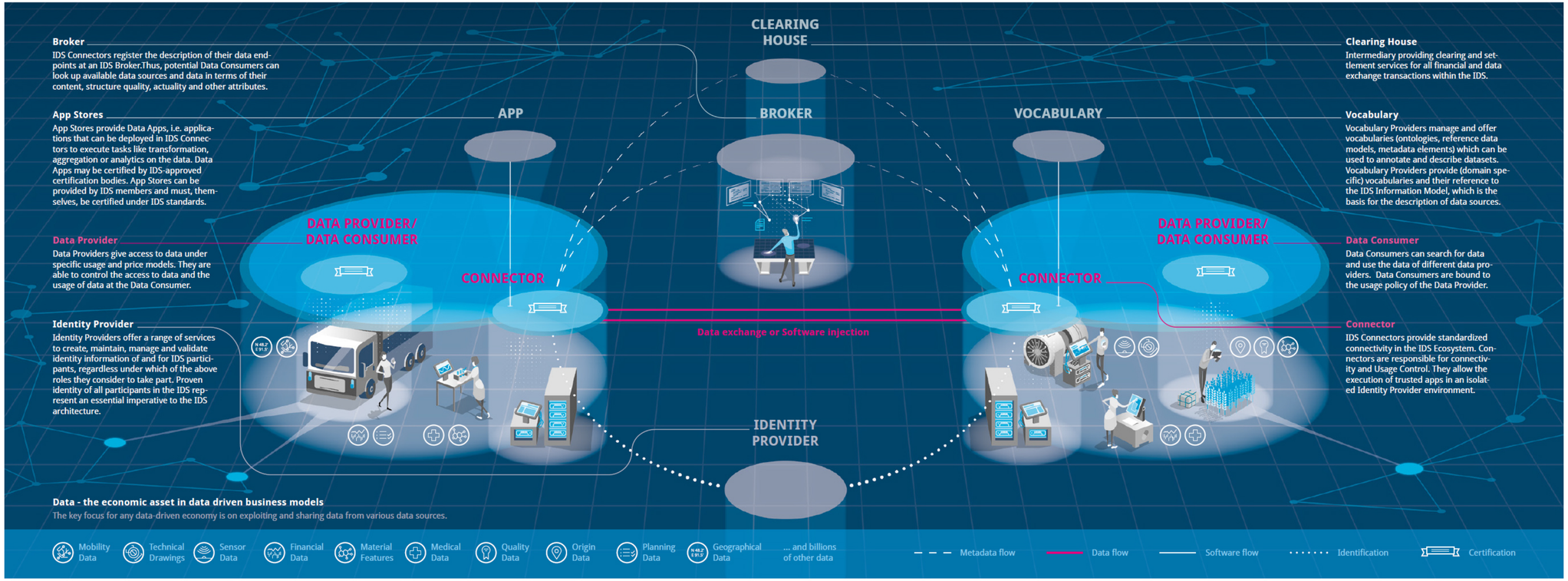
International Standards
IDSA defines the basic conditions and governance for a reference architecture and interfaces

Business Models
Data Owners remain sovereign owners of their data at any time

Use Cases
This standard is actively developed and updated on the basis of use cases

DIGITAL IDENTITIES

A network of trusted entities in the data economy requires a mechanism for digital entities, that can reliably identify a participant and can provide more information on transaction partners. Additional information must be updated regularly and be provided in a trusted manner.



Broker
IDS Connectors register the description of their data endpoints at an IDS Broker.Thus, potential Data Consumers can look up available data sources and data in terms of their content, structure quality, actuality and other attributes.

App Stores
App Stores provide Data Apps, i.e. applications that can be deployed in IDS Connectors to execute tasks like transformation, aggregation or analytics on the data. Data Apps may be certified by IDS-approved certification bodies. App Stores can be provided by IDS members and must, themselves, be certified under IDS standards.

Data Provider
Data Providers give access to data under specific usage and price models. They are able to control the access to data and the usage of data at the Data Consumer.

Identity Provider
Identity Providers offer a range of services to create, maintain, manage and validate identity information of and for IDS participants, regardless under which of the above roles they consider to take part. Proven identity of all participants in the IDS represent an essential imperative to the IDS architecture.

Clearing House
Intermediary providing clearing and settlement services for all financial and data exchange transactions within the IDS.

Vocabulary
Vocabulary Providers manage and offer vocabularies (ontologies, reference data models, metadata elements) which can be used to annotate and describe datasets. Vocabulary Providers provide (domain specific) vocabularies and their reference to the IDS Information Model, which is the basis for the description of data sources.

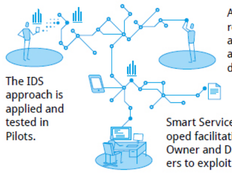
Data Consumer
Data Consumers can search for data and use the data of different data providers. Data Consumers are bound to the usage policy of the Data Provider.

Connector
IDS Connectors provide standardized connectivity in the IDS Ecosystem. Connectors are responsible for connectivity and Usage Control. They allow the execution of trusted apps in an isolated Identity Provider environment.

Data - the economic asset in data driven business models
The key focus for any data-driven economy is on exploiting and sharing data from various data sources.

USE CASES

Services and functionalities of the IDS are specified and validated in use cases.



The IDS approach is applied and tested in Pilots.

Additional requirements are fed into the architecture development.

Smart Services are developed facilitating the Data Owner and Data Consumers to exploit the IDS.

COMMUNITIES

Interest and user groups of same or similar domains with common challenges validate and proliferate the IDS approach, technology and eco-system. Based on their practical experience the IDS reference architecture and the eco-system around it are continuously developed. Thus, specific application scenarios for verticals are set up, implemented and systematically pushed forward, allowing participants to enhance existing or to launch new services.

Medical/ Health · *Banking/ Insurances* · *Smart Cities* · *Energy* · *Farm & Food* · *Materials* · *Industrial* · *Logistics*

10 THINGS TO KNOW ABOUT

Containerization, e.g. Docker · *Webservices, e.g. https, MQTT, REST, Multi Part Messages* · *Enterprise Integration Patterns* · *Message Oriented Middleware* · *Digital Identities and Digital Certificates, e.g. X509* · *Software Engineering, e.g. tools like Maven, git* · *Semantic Data Descriptions, e.g. Ressource Description Framework* · *Data Ecosystems* · *Certification, e.g. IEC 62443, ISO 27001* · *Requirements Engineering, Processes and tools, e.g. UML and BPMN*

CERTIFICATION APPROACH

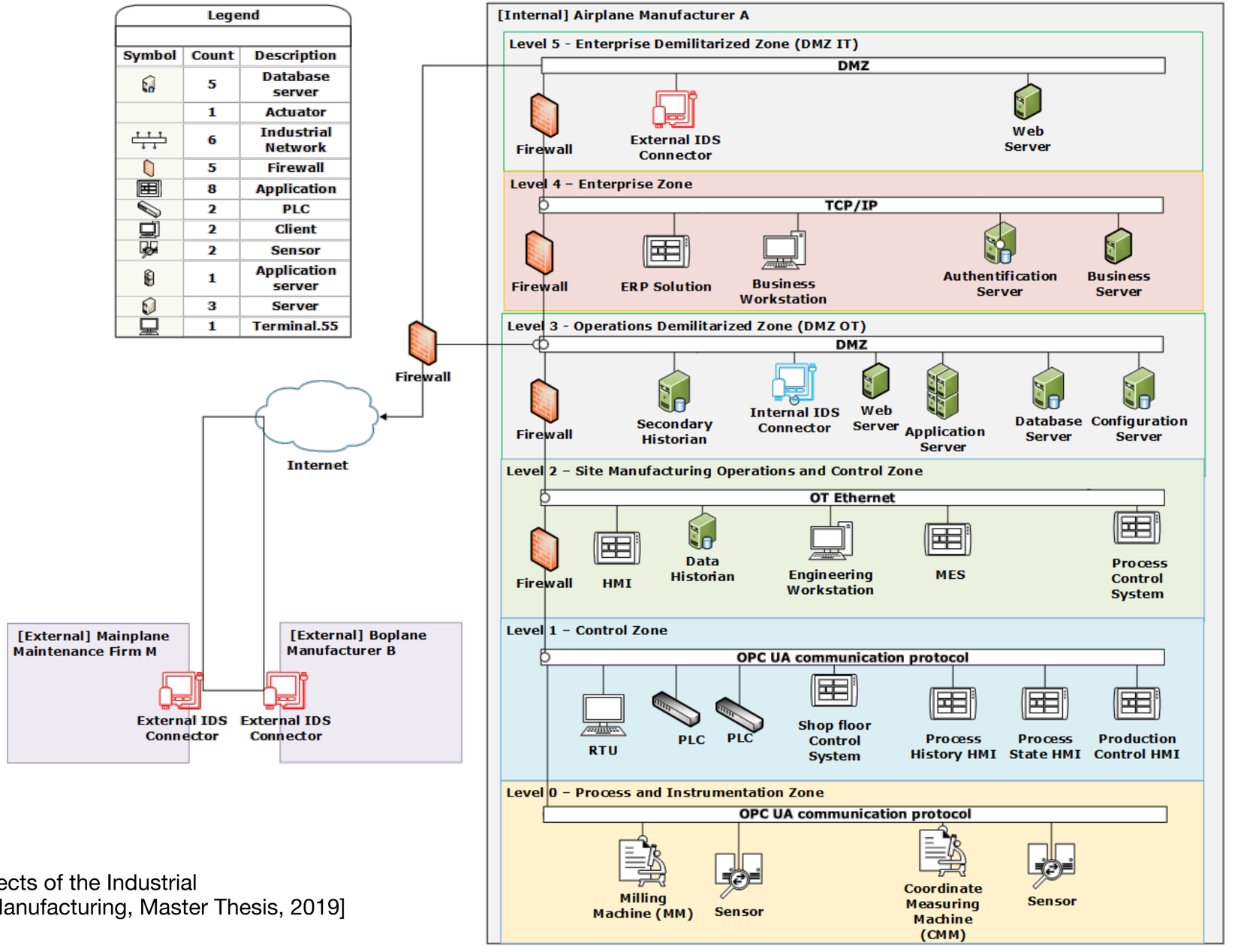
The IDS Certification Body is appointed by the IDSA and regularly aligns with the IDSA to manage the certification process, defines the standardized evaluation procedures and supervises the actions of the Evaluation Facilities. An Evaluation Facility is contracted by an Applicant and is responsible for carrying out the detailed technical and organizational evaluation work during a certification.

Participant Certification · Digital Certificate · Technical Certification

CALL TO ACTION

Become a member in the International Data Spaces Association:

[https://www.internationaldataspaces.org/wp-content/uploads/2018/09/trustworthy_architecture.jpg, Oct 31 2019]



[S. Vetter: Cyber Security Aspects of the Industrial Data Space for Zero-Defect-Manufacturing, Master Thesis, 2019]

Figure 12 Q4ZDM Architecture Vision Diagram for Pilot 1 - Airplane Turbine Manufacturing plant

(web references for “Web Application Architecture”)

Viewpoint, View, Style

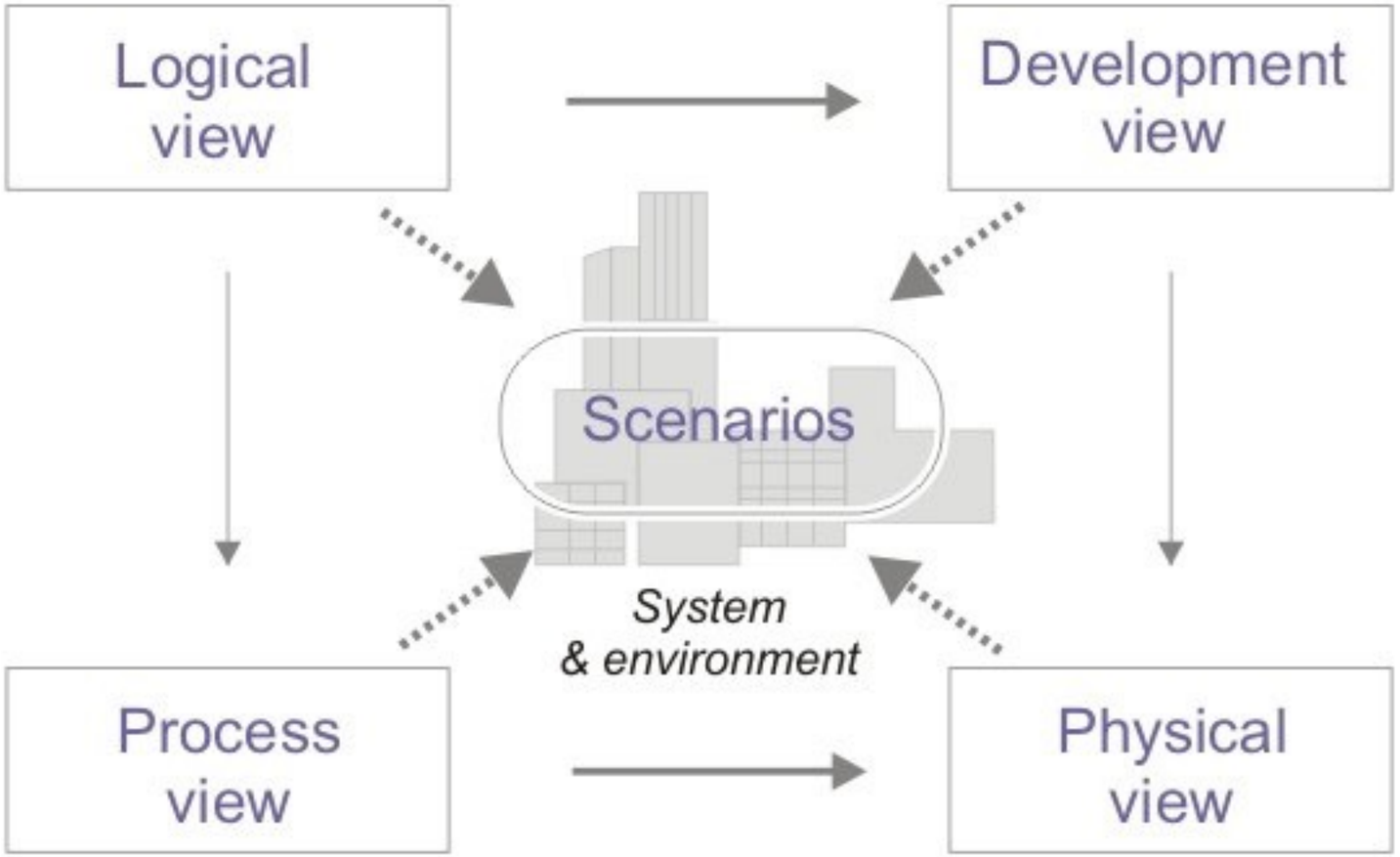
- Architecture **Viewpoints**
  - Which aspects shall be described?
  - What type of components and relations will be shown?
  - What are the rules that restrict the architecture?
- Architecture **Views**
  - What is the architectural description of a concrete system?
  - Each view corresponds to a viewpoint!
- Architecture **Styles**
  - Which (reusable) patterns exist?
  - What are the responsibilities of the different components?
  - How do the components communicate?

Examples for Components and Relations

- **Components**
  - (complete) System, Subsystem
  - Module, Package, Class, Component
  - Layer, Tier
  - Procedure, Function, Method
  - Segment, Paragraph, Refinement,
  - File, Relation, (Database-)Table, Directory
  - Global Data Structure, Object, Variable
  - Partner System
  - Physical Machine
  - Network Device
  - …
- **Relations**
  - A contains B
  - A realizes B
  - A uses B
  - A calls B
  - A follows B
  - A instructs B
  - A sends message to B
  - A delivers data to B
  - ...

4+1 View Model

Architectural Views defined by Philippe Kruchten (1994)



4+1 View Model

Architectural Views defined by Philippe Kruchten (1994)

- The logical view describes the design's object model when an object-oriented design method is used. To design an application that is very data-driven, you can use an alternative approach to develop some other form of logical view, such as entity-relationship diagram.
- The process view describes the design's concurrency and synchronization aspects.
- The physical view describes the mapping of the software onto the hardware and reflects its distributed aspect.
- The development view describes the software's static organization in its development environment.
- Software designers can organize the description of their architectural decisions around these four views, and then illustrate them with a few selected use cases, or scenarios, which constitute a fifth view. The architecture is partially evolved from these scenarios.

⇢ http://www.ics.uci.edu/~andre/ics223w2006/kruchten3.pdf

Example Viewpoints

- Structural Perspective

Components	Relations
subsystem, package, class, method	contains

- Usage Perspective

Components	Relations
component, class, method	uses, calls

- Data Flow Perspective

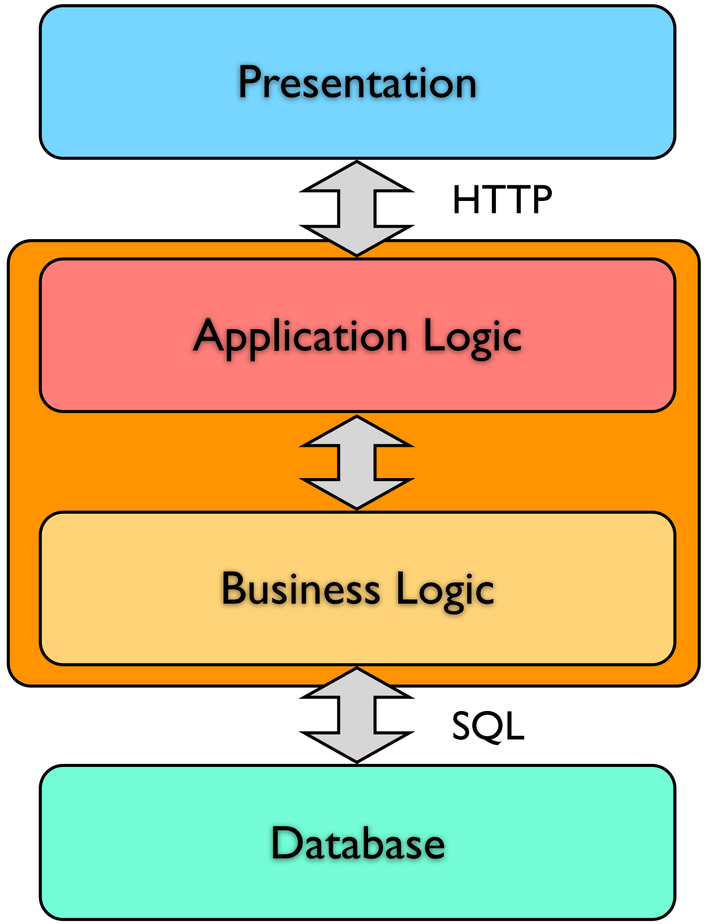
Components	Relations
activity, process, data object	reads, writes

- ...

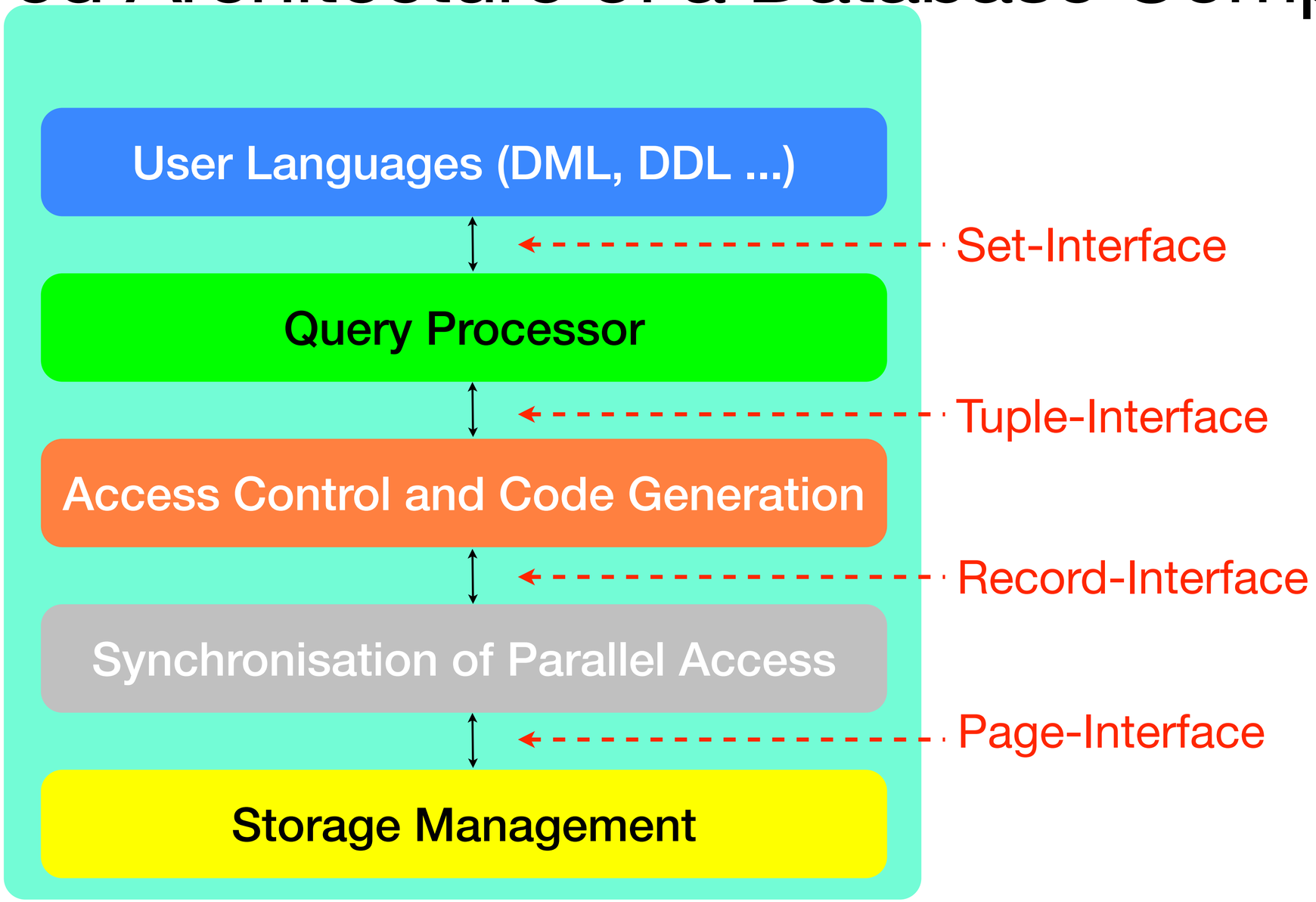
Granularity / Level of Detail

- Different Levels of Detail / Levels of Abstraction
  - from coarse design
  - to detailed design
- Each component of a software architecture can be recursively decomposed.
- **Example**
  Architecture of the database layer of an information system can be decomposed into further layers, which, in turn contain more detailed parts, e.h. pipe-filter views, which can be decomposed…

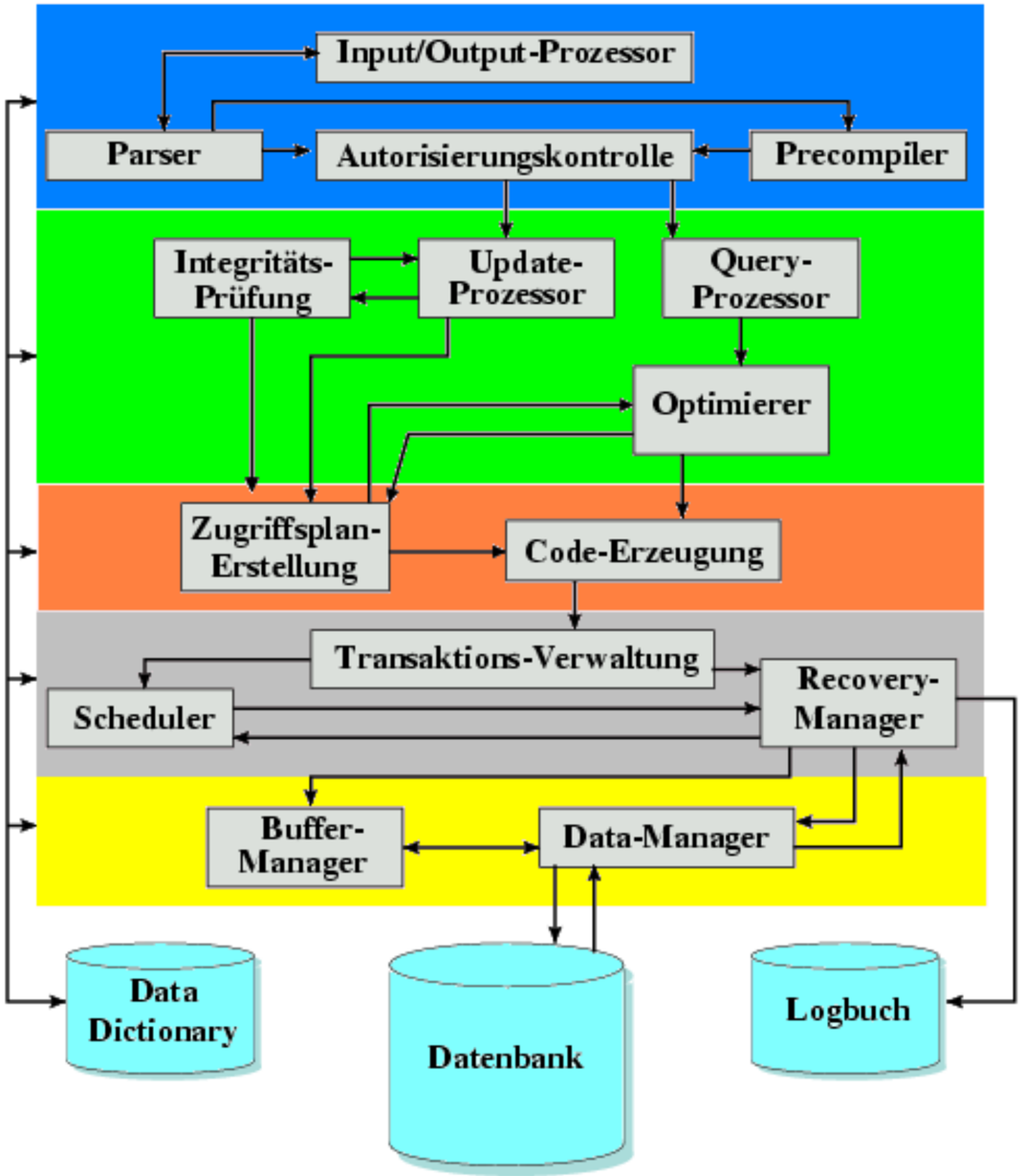
Coarse


Presentation
HTTP
Application Logic
Business Logic
SQL
Database

Layered Architecture of a Database Component



Pipe-Filter/Layer/Repository Structure of Query Processing



Software Quality

- Usability (external quality)
  - Reliability
    = Correctness (including Precision)
    + Robustness
    + Resilience
  - Ease of use
  - Efficiency
- Controllability (internal quality)
  - Maintainability
    = Verifiability
    + Flexibility
    + Understandability
  - Portability
  - Scalability

Software Architecture and Quality

- A good (suitable, adequate, flexible, reusable, …) software architecture doesn't come for free.
- Instead, it results from experience, established standards, technologies and frameworks, conventions, …
- Decisions about software architecture are mostly driven by non-functional requirements and expectations.
- Over time, suitable solutions for various requirements emerge and evolve.

  This results in architectural styles and architectural patterns that can be used as blueprints for new systems.

Principles

- Following the fundamental and structural principles leads to higher quality.
- Fundamental principles
  - Simplicity
  - Analogy
  - Completeness
  - Verifiability
  - Least Surprise
- Structural principles
  - Decomposition
  - Encapsulation
  - Locality
  - Separation of Concerns
  - Abstraction
  -

Principle of Simplicity

- (also known as KISS = keep it small and simple)

  From a range of possible solutions, pick the most simple one.

- Supports maintainability and understandability

Principle of Analogy

- (also known as principle of uniformity)

  Similar things are to be handled similarly.

- Supports understandability and leads to definition of conventions

Principle of Completeness

- **All possible cases (alternatives) have to be considered, even if only a few of them are explicitly required.**
- Supports robustness
- This doesn’t mean that all cases have to be elaborated

Principle of Verifiability

- **Means and methods to verify whether the goals have been achieved has to be provided.**

- Supports reliability and correctness

Principle of Least Surprise

- **The behavior of the system shall be most obvious (match expectations).**
- Supports robustness and usability
- Standards contribute to application of this principle

Principle of Decomposition

- **On each layer (level of detail), the product shall be rationally decomposed into its components and their relations.**

- Supports understandability and flexibility

- Decomposition can be performed top-down or bottom-up
- The number of components on each level should be low (5…7 is the “magical range”)

Principle of Encapsulation

- (also known as information hiding)
  Details that are only of local importance have to be made inaccessible from the outside.

- Supports flexibility
- Increases understandability by minimizing interfaces

Principle of Locality

- **Things that belong together have to be physically kept together.**

- Supports maintainability

Principle of Separation of Concerns

- **Things that are not related to each other have to be kept separately.**

- Supports structuring, flexibility, and maintainability

Principle of Abstraction

- **Things that have common properties have to be identified, even if they are used in different places.**
- Supports maintainability
- Leads to thorough review and comprehension of the system

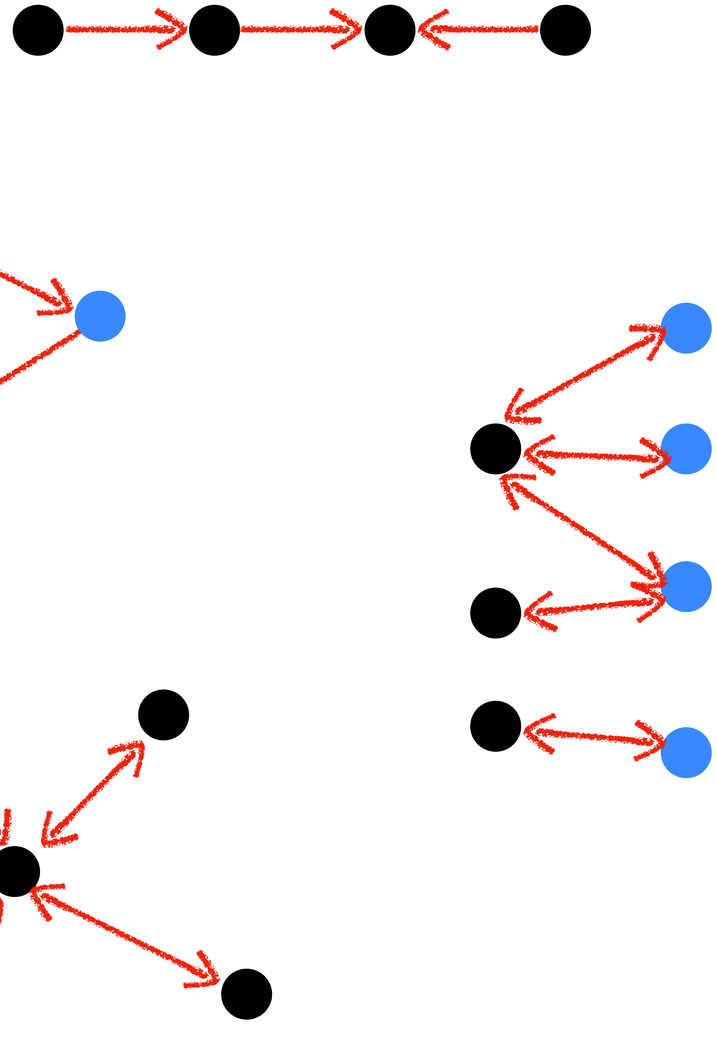
Architectural Styles

- Architecture models are composed of components and their relations.

- Architectural styles describe the basic structure of the architecture diagrams.
- Such diagrams are graphs that consist of nodes (representing components) and edges (representing relations).
- Hence, architectural styles determine the structure of the component graphs.

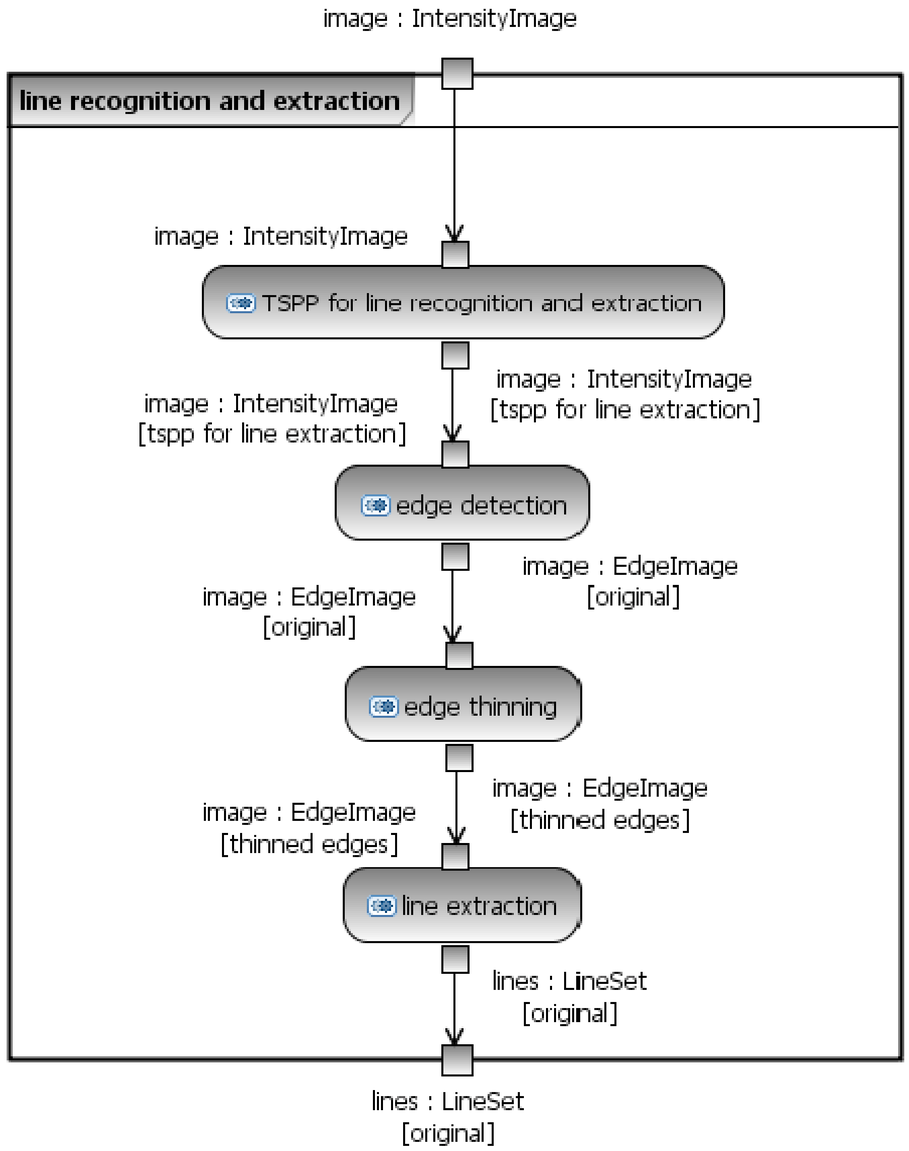
Graph Structures

- Examples
  - linear structure
  - tree-shaped
  - acyclic
  - hierarchical, layered
  - bi-partite
  - star-shaped
  - …



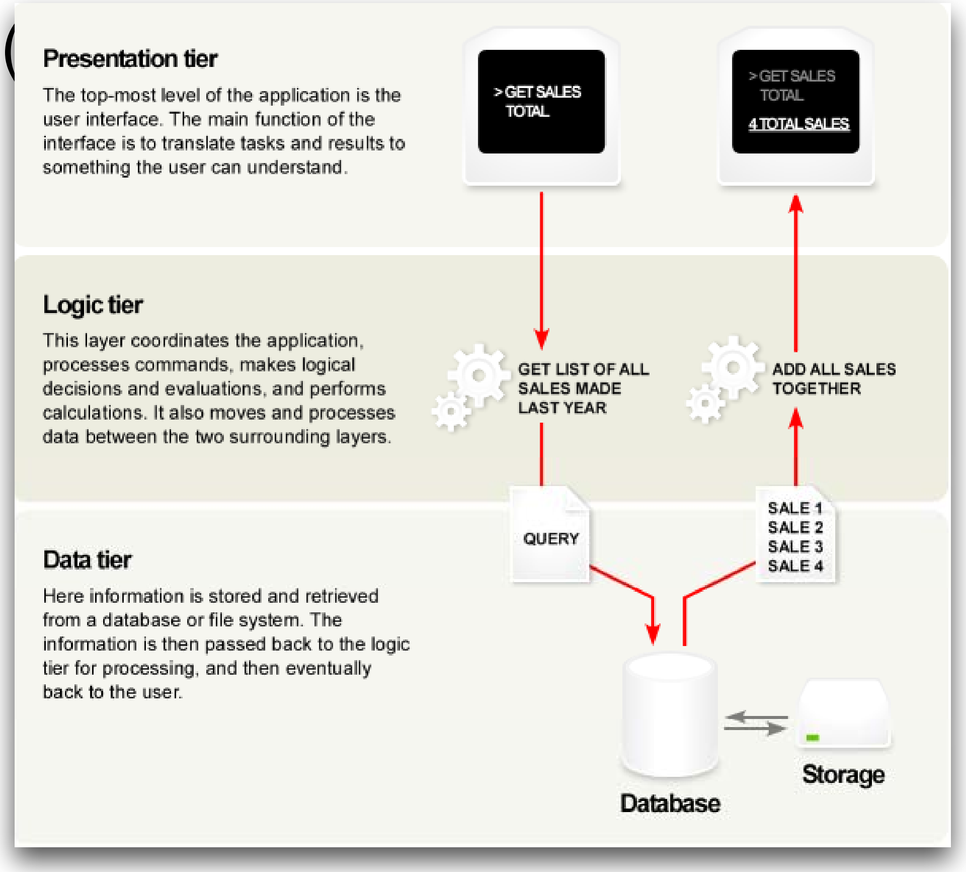
Pipe-Filter Style

- Active components (filters) are conne
- Data flow perspective
- Linear structure



Layered Style

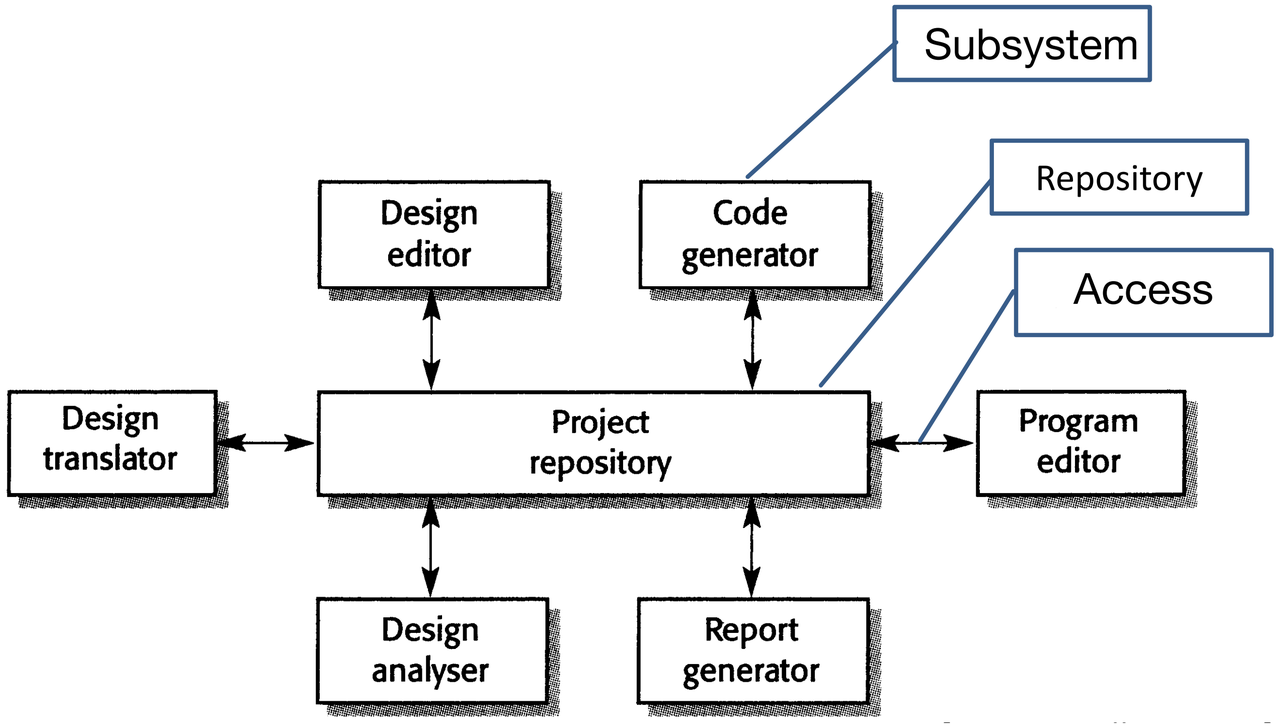
- Each layer provides services to the layer (
- **Usage** perspective
- **Hierarchical** (tree-shaped) structure



Repository Style

- Many independent components work on a common data set.

- Usage perspective
- Star-shaped structure



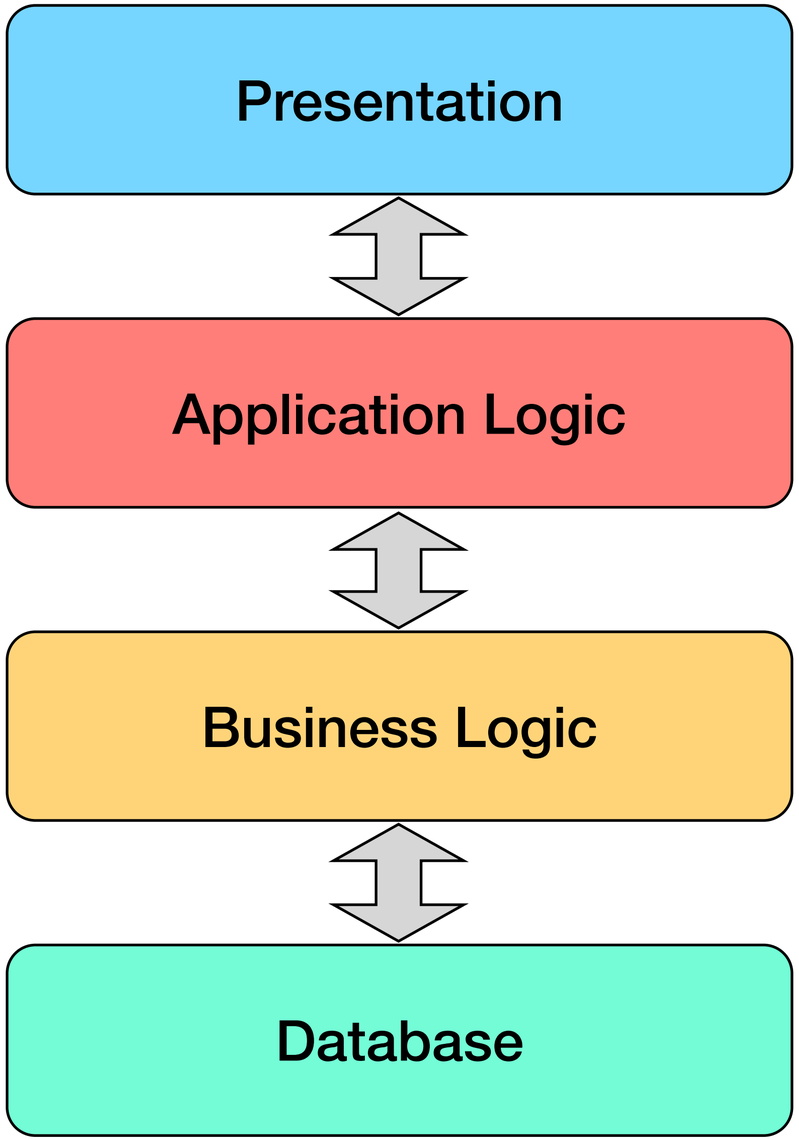
Architectural Patterns in Web Applications

- Architectural patterns provide an adaptable blue print for system-specific solutions.

- We deal with patterns for Web Applications of the types
  (cf. the classification in the motivation chapter)
  - transactional
  - portal-oriented
  - collaborative / social

- The patterns serve a specific purpose, that is, they contribute to realization of functionality and to achieving quality goals expressed in non-functional requirements.

4 Layer Architecture



Visualization of values, Input areas, graphics, interaction with I/O devices

Control of sequence of actions, navigation, validation of input values, prevention of unauthorized access, conversion of values

Realization of basic application functions, interfaces to other systems, prevention of unauthorized access

Persistent storage of data, backup, transaction management, efficient access…

Layered Architecture

Layering contributes to the following principles: Decomposition, Encapsulation, Locality, and Separation of Concerns

- **Decomposition**
- By assigning specific obligations to the various layers, the system becomes more structured and more easy to understand.
- This contributes to understandability and maintainability.

- Efficiency can also benefit from those layers: the client-server relation between layers allows to distribute a system over several physical machines.
- It’s possible to scale each individual layer by adding/removing machines depending on the system load.

Layered Architecture

Layering contributes to the following principles: Decomposition, Encapsulation, Locality, and Separation of Concerns

- **Encapsulation**
- Each layer provides a functional interface (service, set of operations) to its clients.
- To use such a service, it's sufficient to know about the specification (contract) of the service (that is the "WHAT"), but not the specifics of the realization (that is the "HOW").
- Adds flexibility to the service implementation.
- Encapsulation is a key principle for Service Oriented Architectures (SOA).

Layered Architecture

Layering contributes to the following principles: Decomposition, Encapsulation, Locality, and Separation of Concerns

- **Encapsulation**
- Supports reuse, e.g. common so-called “infrastructure” services can be accessed by many applications.
- Verifiability is supported by well-defined interfaces that can be tested independently.
- In large applications, encapsulation enables parallel development (by scaffolding).

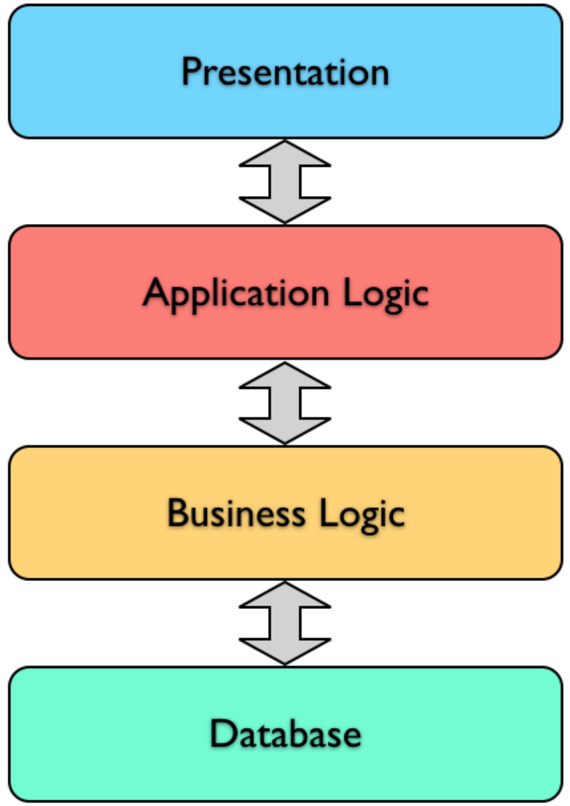
Layered Architecture

Layering contributes to the following principles: Decomposition, Encapsulation, Locality, and Separation of Concerns

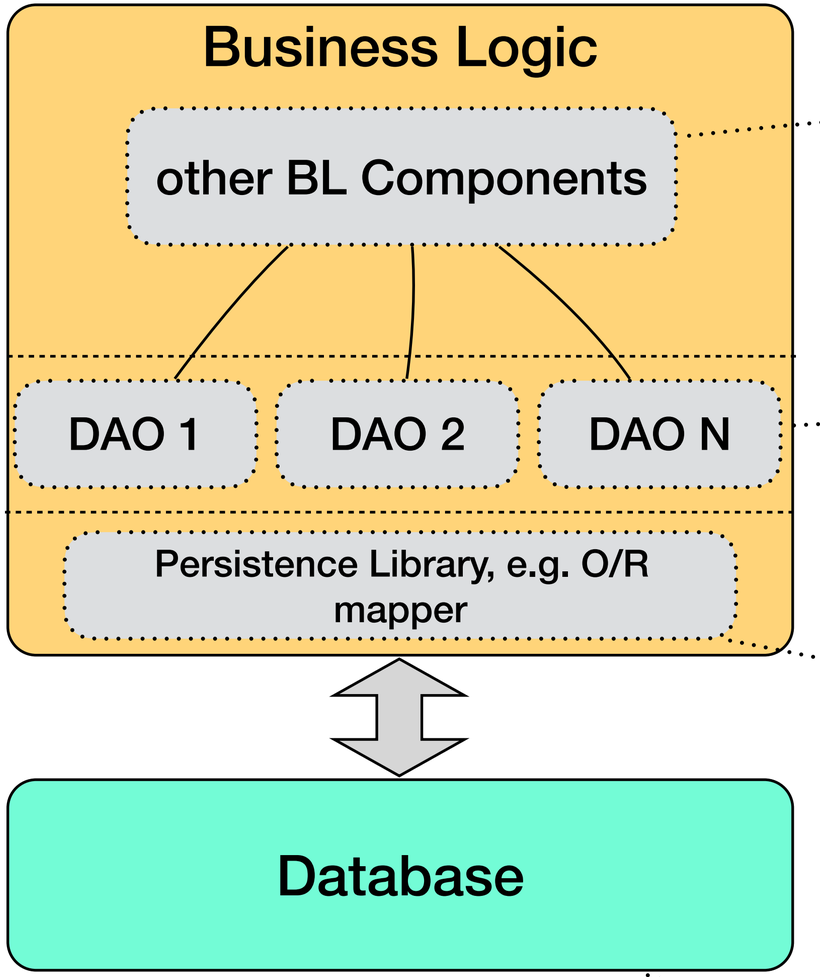
- **Locality, and Separation of Concerns**
- Different obligations of the layers result in meaningful grouping of functionality.
- This allows to assign development and maintenance tasks to experts that must not know all details of the other components.
- In each layer, the most appropriate technologies can be used to solve the problem without influencing other parts.

Architectural Patterns at Layer Boundaries

- In a **layered architecture**, the services provided by a layer are provided to an upper layer and realized using the services of subordinate layers.
- Interface definitions at the boundaries between layers are especially important.
- The patterns on the next slides detail on some properties of those boundaries.
- Highlighted comments illustrate the purpose of the patterns.



Data Access Object (DAO), Data Mapper



Handle business objects of the application domain (→ domain model)

Encapsulates specifics of the persistence library and/or database engine.

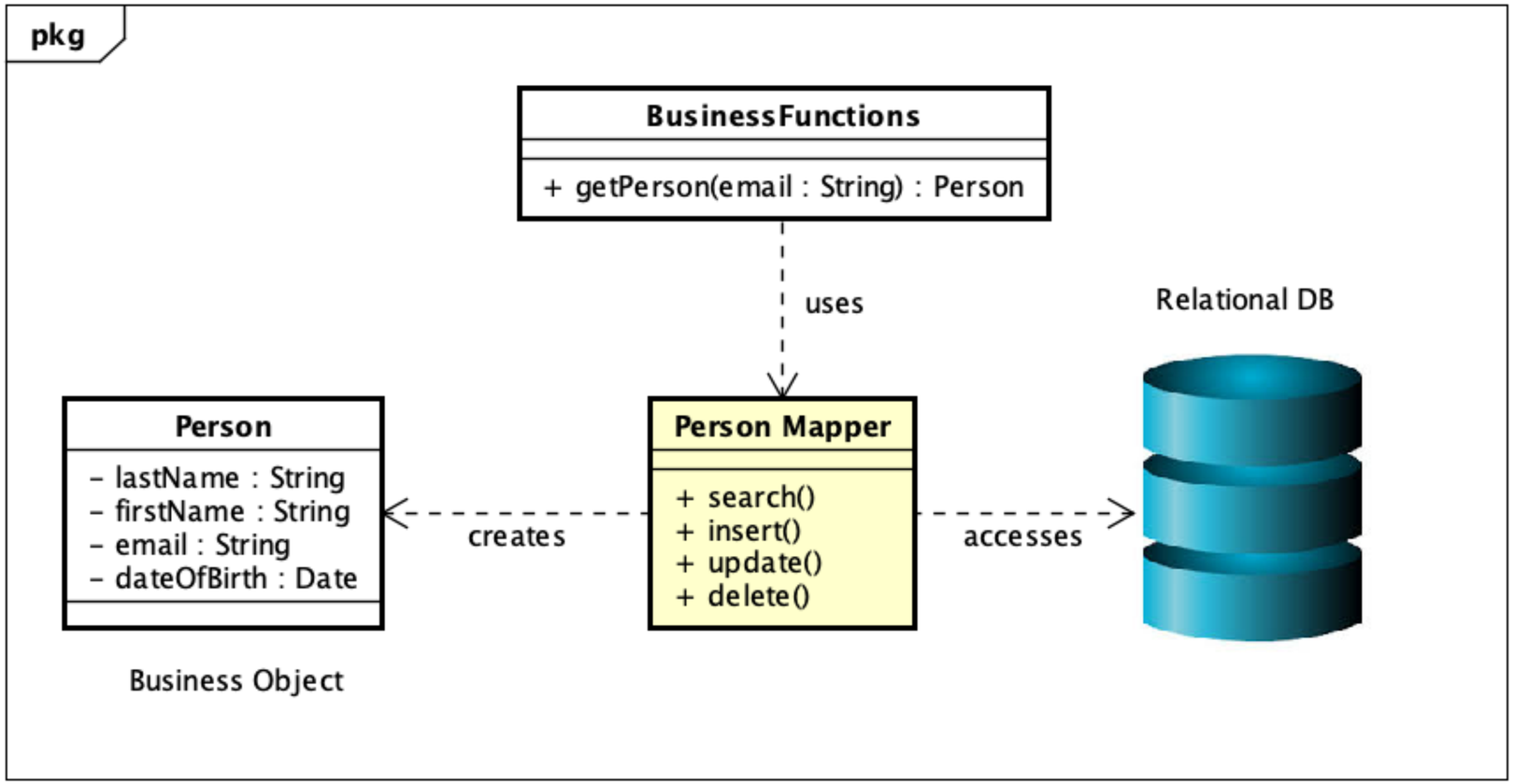
Converts DB objects into business objects and vice-versa

Supports mapping of business objects to database-specific structures.
Uses DB-specific drivers, (query-)languages, structures.

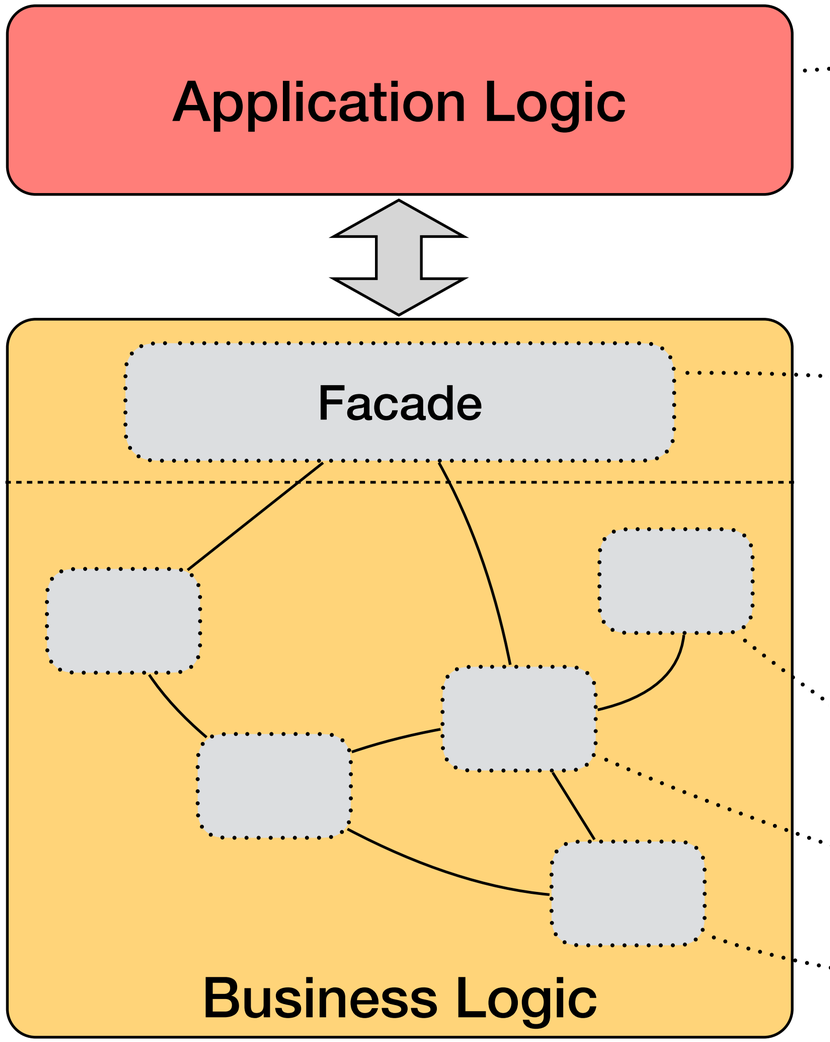
Provides basic services for data storage, transactions, replication, backup, …

Data Access Object (DAO), Data Mapper

- DAO encapsulates DB specifics and thereby adds flexibility, e.g. to switch to a different DB vendor or DB type.



(Remote) Facade



Uses BL functionality to realize application functions

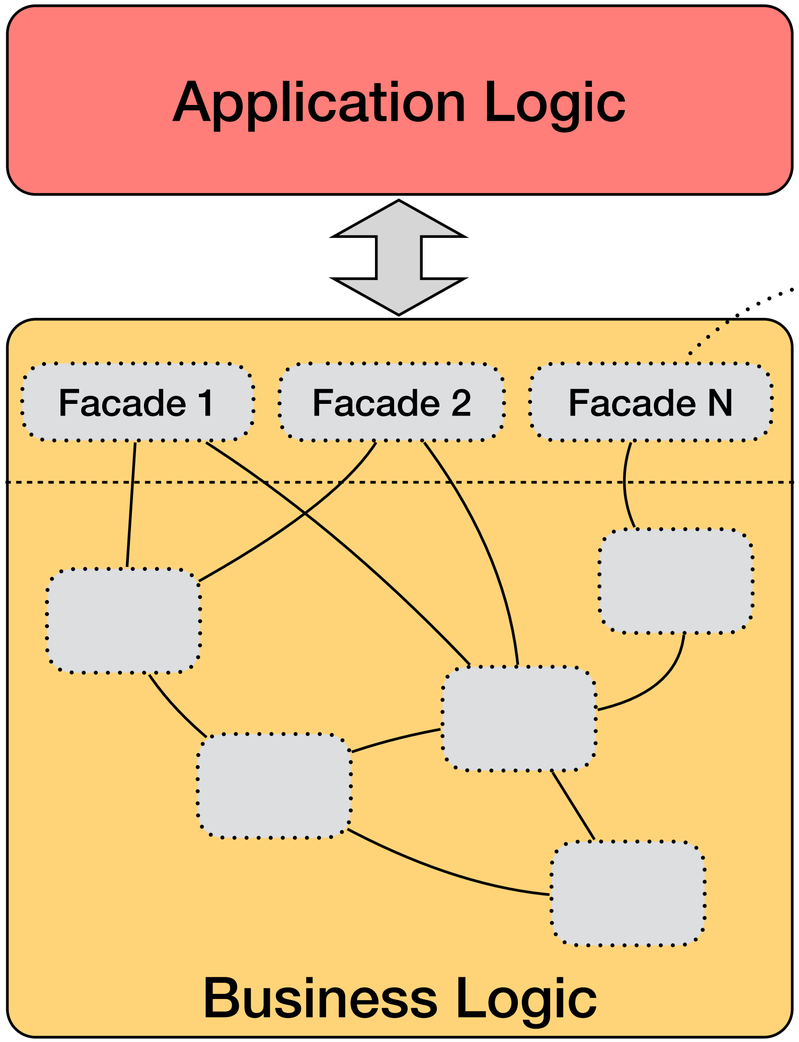
Defines exported services (interface) and delegates to specific BL components.
Groups data and calls to minimize communication overhead.

Hides internal complexity.

Often uses stateless interface.

BL components implement basic business functionality.
May have complex dependencies.

(Remote) Facade

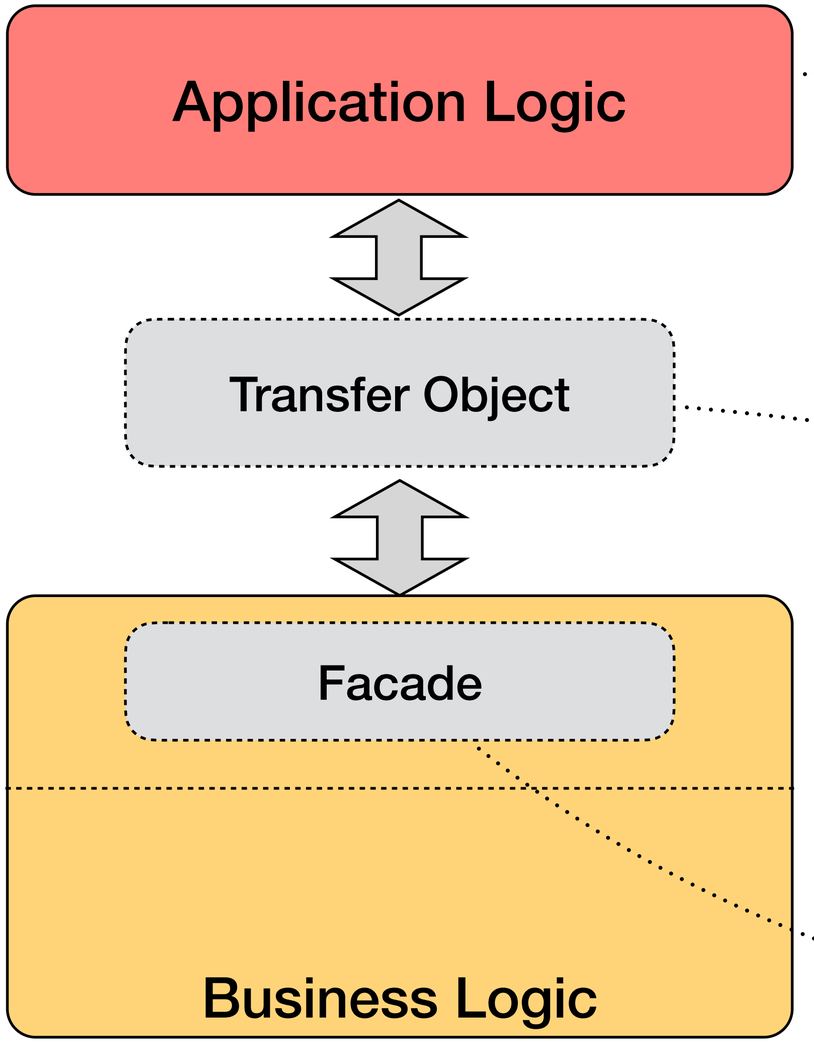


BL can have multiple facades.
This allows…

… to provide different protocols,
e.g. RPC or REST

… and/or to group functionality,
e.g. split into user functions,
admin functions

(Data) Transfer Object (DTO or TO)



Uses BL functionality to realize application functions
Serve as pure data objects without specific behavior.

Combine data from various business objects.

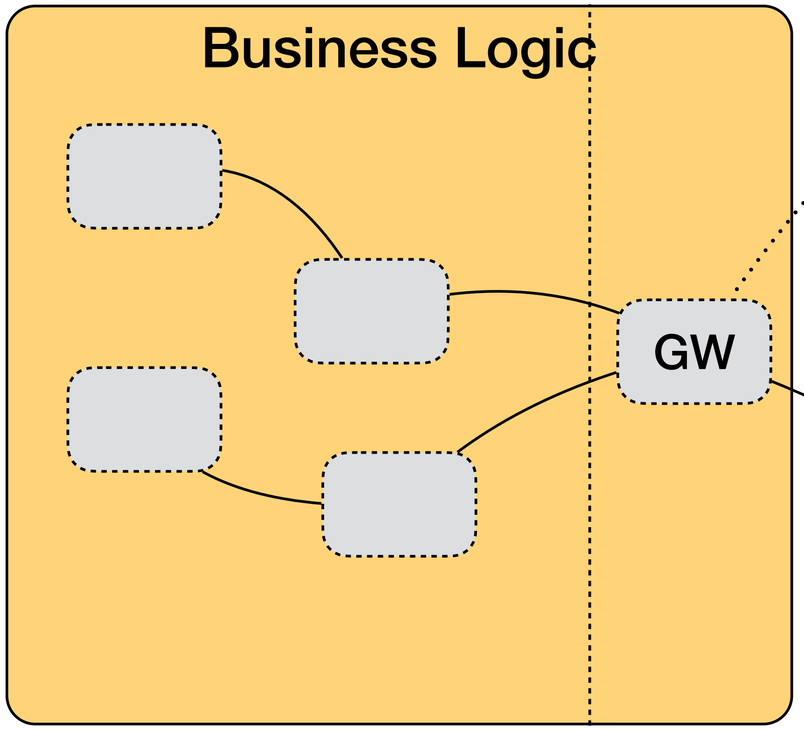
Different DTOs allow for arbitrary level of detail.

Convert between internal and external data formats, e.g. binary, XML, JSON…

May contain AL specific data, e.g. change markers.

Service interface relies on operation-specific transfer objects.

Gateway



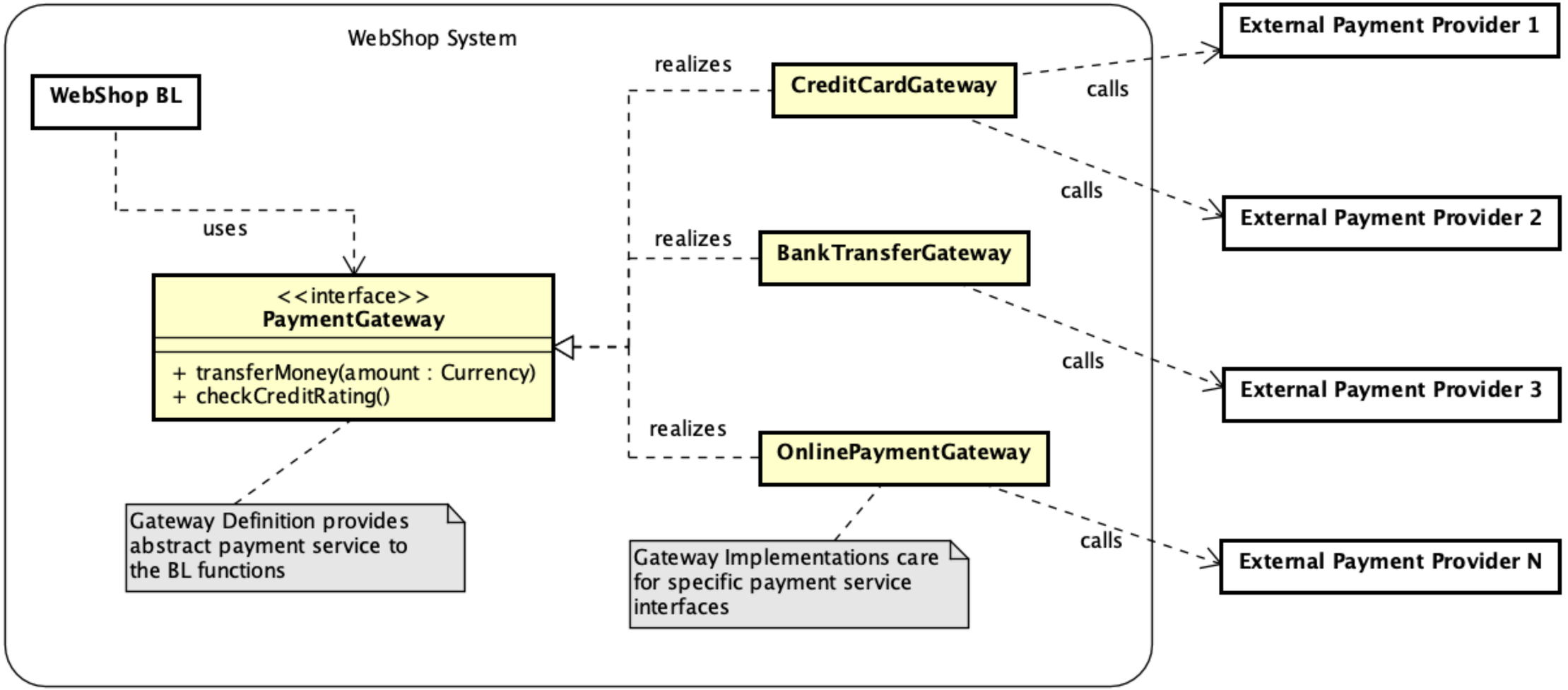
Encapsulates functionality of external systems. Provides uniform access to those systems from the rest of the business logic.

Allows for testing by providing mockup services.

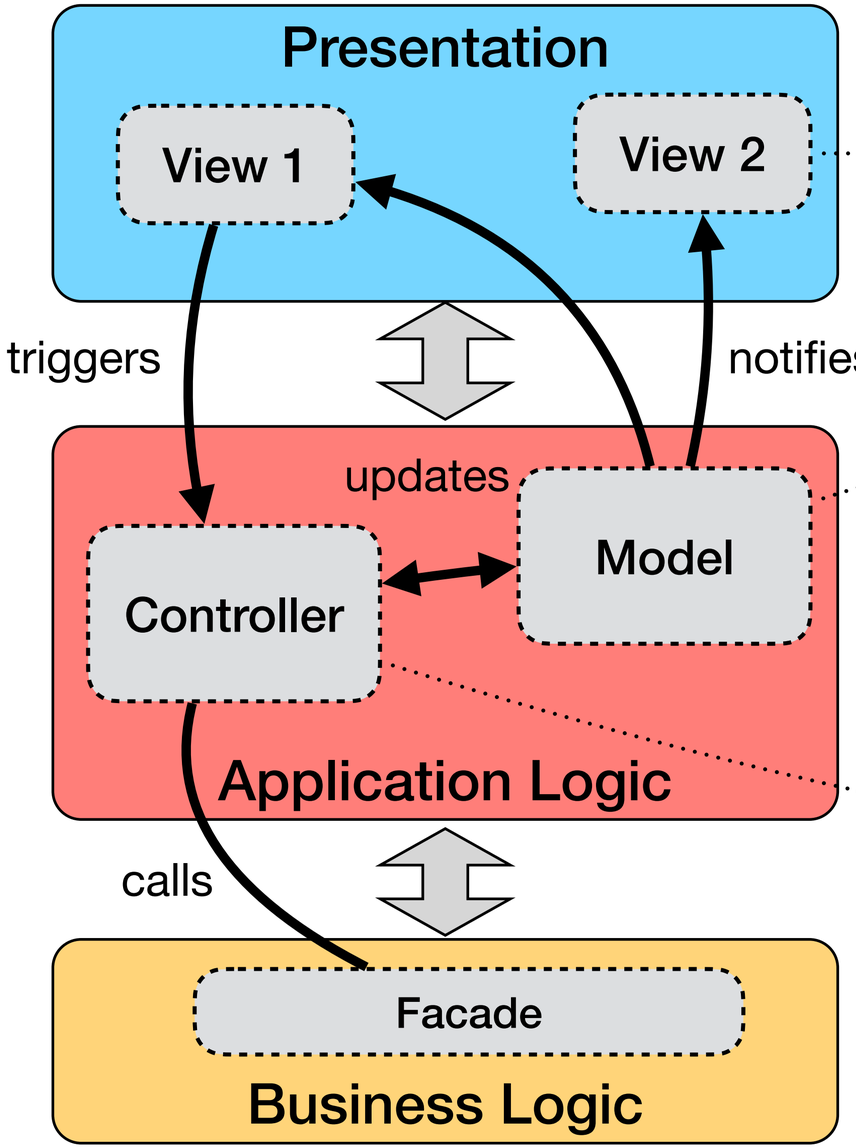
calls

External Partner System

Gateway


WebShop System
WebShop BL
uses
<<interface>>
PaymentGateway
+ transferMoney(amount : Currency)
+ checkCreditRating()
Gateway Definition provides abstract payment service to the BL functions
realizes
CreditCardGateway
realizes
BankTransferGateway
realizes
OnlinePaymentGateway
Gateway Implementations care for specific payment service interfaces
calls
calls
calls
calls
External Payment Provider 1
External Payment Provider 2
External Payment Provider 3
External Payment Provider N

Model-View-Controller (MVC)



User Interface Component
visualizes data, triggers interactions.

Contains data objects.
Provides functionality to retrieve data.
Issues update notifications to views.

Implements update and change operations.
Determines navigation decisions.
Delegates to business logic operations.

What we have learned…

Architecture

- ✓ Quality Goals
- ✓ Principles
- ✓ What is Software Architecture?
- ✓ Viewpoint, View, Style, Pattern
- ✓ Some patterns in Web Applications

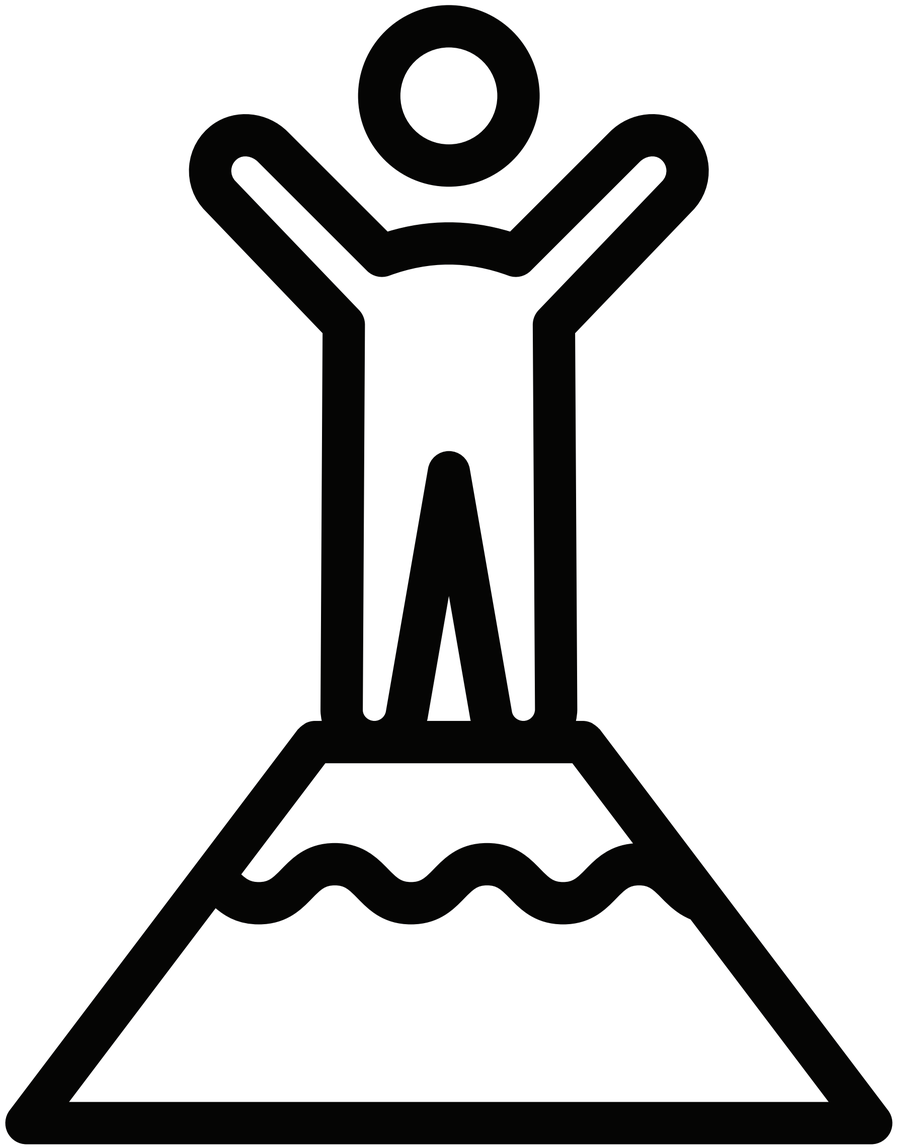


Image: colourbox.de