Computer Science

5. Persistence (Part I)

Engineering Web and Data-intensive Systems

Dr. Volker Riediger - Winter Term 202/23

Institute for Software Technology

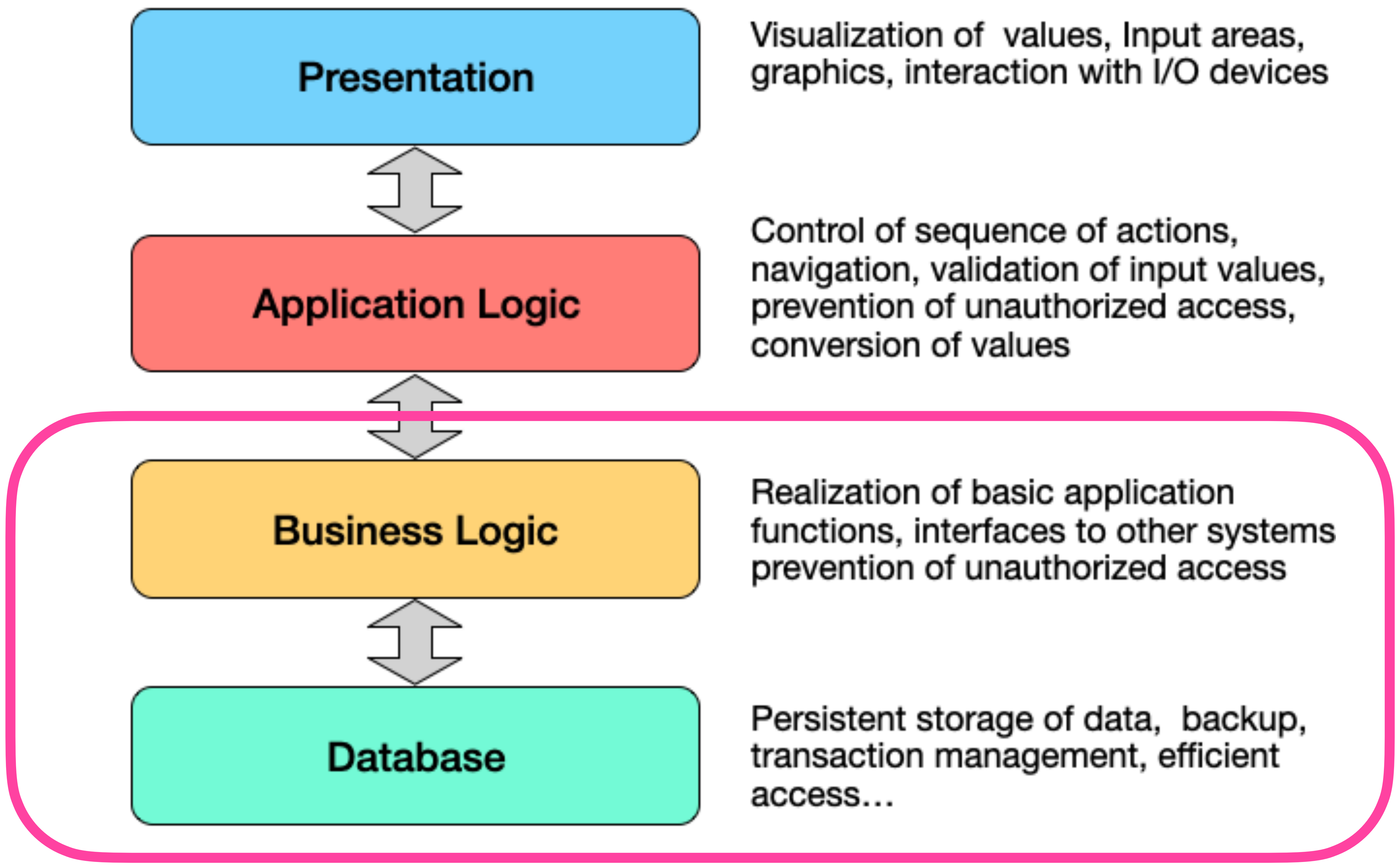
Persistence (Part I)

- Overview
- Data Properties
- Persistence Tasks
- O/R Mapping

Image: colourbox.de

Persistence: Overview

- Data Properties (I)
- Persistence Tasks (I)
- Persistence vs. Scaling (III)
    - Data Intensive Systems
    - Distributed Storage
    - CAP, ACID, BASE
- Data Mappings
    - Relational (I)
    - Graph (II)
    - Document (III)



5.1 Data Properties

What is “persistence”?

Quotes from live discussion with students

- the storage that is independent of the on- and offline status of the system
- if data that was stored lasts even in case of failures
- characteristics of a state that outlives the process that crates it

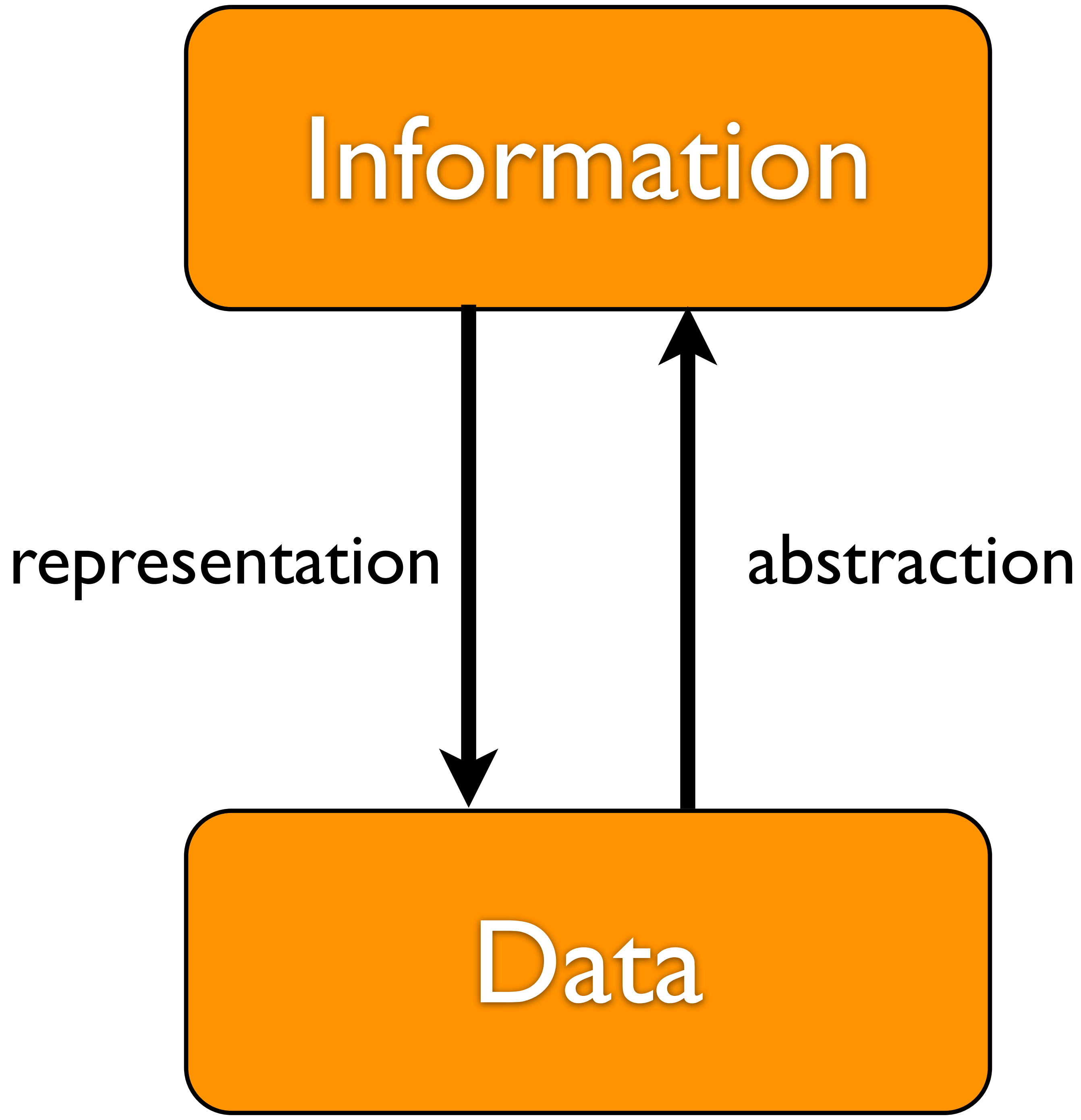
What is “data”?

Quotes from live discussion with students

- a raw version of information
- a set of information represented in machine readable form
- is a factual material which can be processed and used for a specific purpose
- a raw of characters or numerics that is fed to a system to gain information from it
- a raw available input that is not processed
- an entity that holds information in any form
- some set of items from where we can get information

Information vs. Data

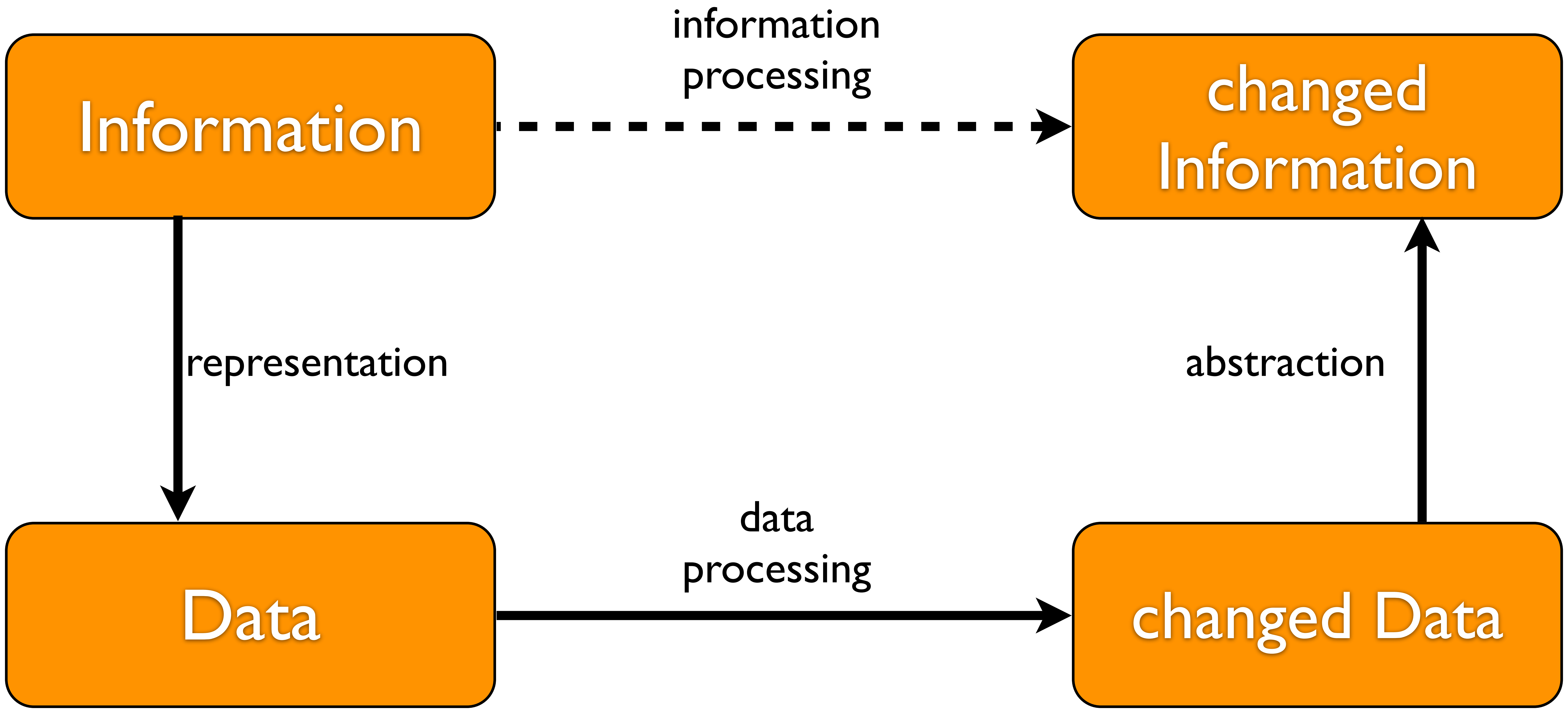
- Many possible meanings „in the real world“, e.g. the number 65 could be a number of participants in a lecture, an age, a billing amount, …
- Representation of values, e.g, numbers, in different systems:
  65 (decimal)
  41 (hexadecimal)
  LXV (roman)
  ...
- Different data representations for the same information
- Different abstractions for the same data



➜ [Gumm2011] Informationen und Daten

Information Processing

- Computers process data, not information
- Data processing creates/updates/removes data
- Changes in data represent changes information



Example: Representation of Numbers

- Representation of numbers (information) in binary code as a sequence of bits (Data)

- e.g. the natural number 65

$$65 \xrightarrow{\text{representation}} 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1$$

$$0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$
$$= 0 + 64 + 0 + 0 + 0 + 0 + 0 + 1 = 65$$

- Bits with different weights (powers of 2)

Example: Abstractions of Data

- Different abstractions (interpretations) of the same data, e.g. the bit sequence:

`01000001` → an many more…

Abstraktionen

Number 65
(⇢Binary Code,)

Number 41
(⇢BCD Code)

Letter A
(⇢ASCII-Code)

8 answers to yes/no questions
N Y N N N N N Y

Red value of a pixel in an image

What does “data intensive” mean?

Quotes from live discussion with students

- processes that operate on a vast amount of data
- high throughput of data which is stored, processed, or presented
- a system that handles large amounts of data and its representation

- Wikipedia entry on Data Intensive Computing

What are data properties?

Quotes from live discussion with students & my aspects

- format (e.g. machine readable)
- size
- encoding
- complexity
- accuracy
- type
- meta data
- availability
- relevance
- …

- Size
- Type
- Lifetime
- Structure
- Cardinality
- Frequency of C/R/U/D operations
- Prevalent access mode
- Processing type
- Query type
- Consistency/Correctness requirements
- Availability requirements
- …

How do data differ in life time?

IP Packet

life insurance
contract

Session data

System Logs

Bank Transactions

Passwords

Parameters for
Function calls
or Services

Results of Surveys
Scientific documents

very short → very long

How do data differ in size?

video

geographical data

connection data
(e.g. phone calls)

KLIPS

social networks

open data

image

accounting data
in banking systems

wikipedia

cache
content

tiny

huge

Storage alternatives

Quotes from live discussion with students

- individual files (aka documents)
- relational database
- object database
- nosql (non-relational) db
- paper (e.g. blockchain keys)
- …

Storage properties

- Structures
  - Relations (tables)
  - Documents
  - Key-Value based
  - Object-oriented
  - Graphs
  - …
- Location
  - Client-side
  - Server-side
  - “On the Cloud”
  - Distributed
  - …
- Physical storage
  - In-Memory
  - Disk
  - Tape
  - Optical
  - Paper
  - …

5.2 Persistence Tasks

Persistence Tasks

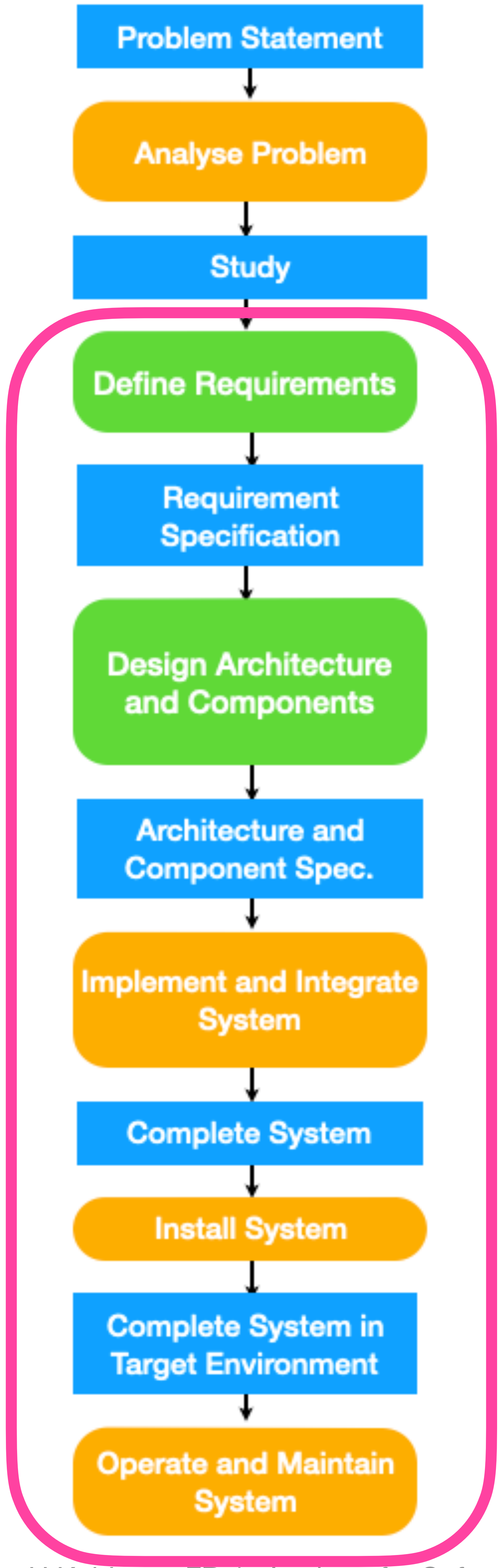
Essential tasks for maintainng data in a web application

- Information Engineering
    - Define conceptual schema
    - Specify constraints and invariants
- Derive logical schema
    - Select persistence technology
    - Map conceptual schema to selected technology
    - Specify logical schema
    - Implement physical schema
- Manage database content
    - Connect persistence layer to business logic
    - Query the database
    - Update data
    - Maintain distributed data

Information Engineering

Information Engineering
(not a focus in this lecture…)

- Relevant in many phases of the software lifecycle
  - Requirements Elicitation
  - Architectural design
  - Implementation, Deployment, Operation
- Conceptual, logical, and physical schemas
- In this lecture:
  - Domain Modeling
  - Mapping to logical schemas
  - Distributed data (foundational aspects)



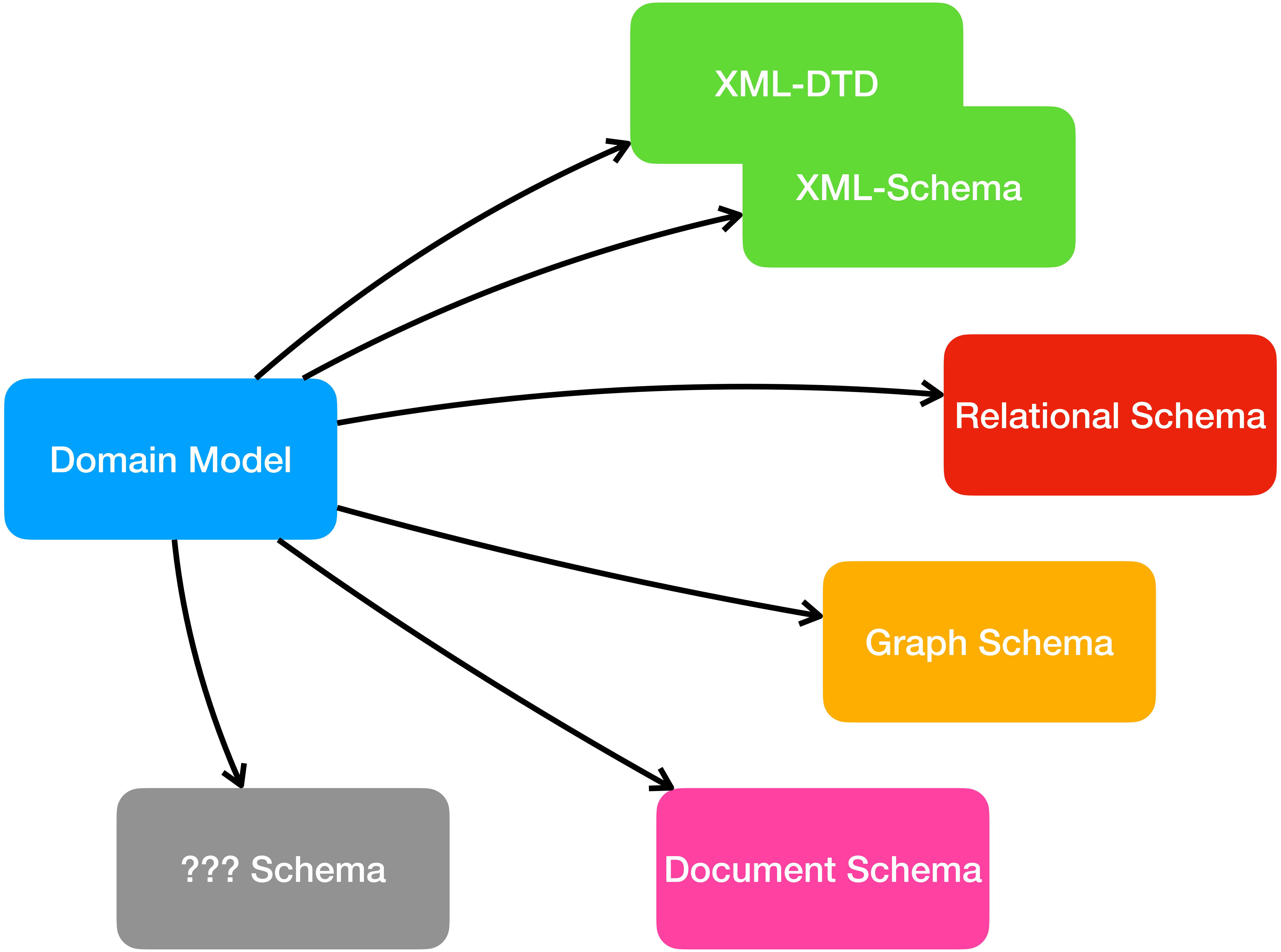
Derive Logical Schema

Which Persistence Technology?

Many things have to be taken into account…

- Fixed vs. flexible data model
- Size of data
- Requirements for consistency, response times, availability
- Data protection and security requirements
- Data governance
- Type and frequency of updates
- Centralized vs. distributed storage
- Types of questions to be answered
- Types of applications to be supported
- Number of concurrent users
- Cost
- …

Mapping schemas is a transformation



- Model-to-Model (M2M) transformation
- (Semi-)automatic or manual
- Dependent on capabilities/ expressiveness of source and target models
- “simulate” or “emulate” or “map” missing concepts

Relational Schemas

- Schema consists of
  - Tables, attributes, domains
  - Keys, foreign keys, indexes, etc.
  - Views
  - Triggers, stored procedures
- Schema must exist before data can be stored

- Database engine ensures basic consistency constraints
  e.g. referential integrity or range of attribute values
- Schema changes are complex and expensive
  - Applications have to co-evolve
  - Alternatively, changes must be “hidden”, e.g. by application-specific views

NoSQL Databases

- NoSQL - Not only SQL
- For most NoSQL databases, no fixed schema is required to store data
- Schema structure emerges from data content
- Constraints have to be enforced by applications
- New data and new relations can be added easily

- However…
  - Schema information is essential to formulate meaningful queries
  - Applications need a schema (“domain model”)
  - Applications have to be aware of/resilient against “unexpected” objects and relations

Graph DB Schemas

- Data stored as nodes and edges
- Depending on the capabilities of the graph model
  - directed or undirected edges
  - attributes (properties) on nodes and/or edges
  - node/edge types and generalization between types
  - ordered relations, i.e. all edges of a node have a deterministic order
  - hypergraphs (relations with more than 2 ends)
  - subgraphs (contained in nodes and/or edges)
  - distributed graphs
  - …

Document DB Schemas

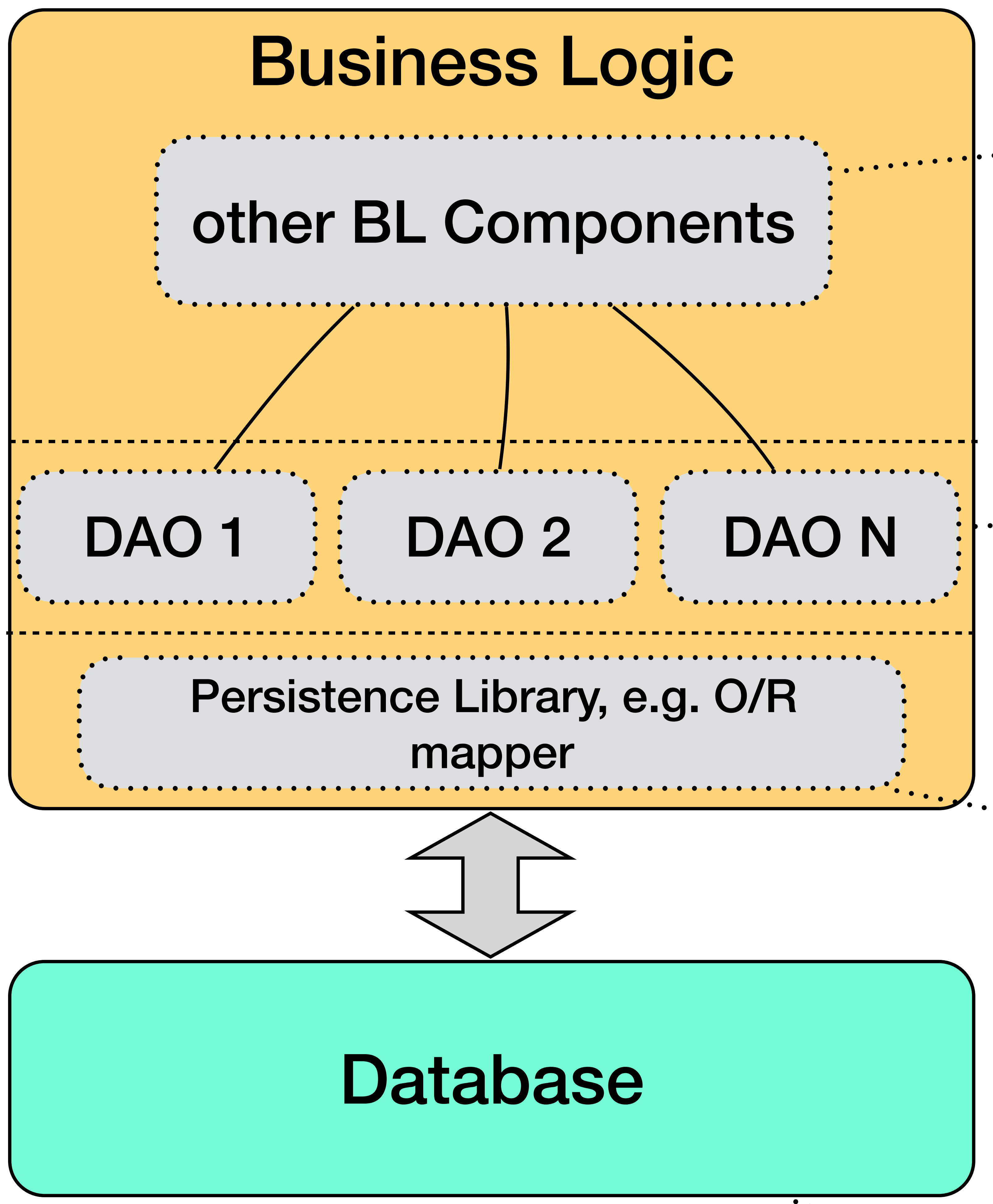
- Data stored in documents
- Usually structured content
  - e.g. XML documents
  - JSON documents
- Each document can have different schema
- Identification by document ID
- Access via REST API
- Depending on the document DB implementation
  - Attachments of various types
  - Links between documents to avoid redundancy
- Usually hard to define integrity constraints (i.e. not checked automatically like in relational DB)

Database Schemas

- In many cases, a database structures already exist
- New applications have to use that database
- Application data mappings have to be defined in a way that the mapping framework uses the existing structures

Manage Database Content

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, …

Tasks of a Data Mapper

- Store state of business objects in the database
- Recreate business objects and their state from database content
- Map in-memory to external IDs (bi-directional)
- Monitor state change of business objects
  - creation of objects
  - change of attribute values
  - link/unlink other business objects
  - removal of objects
  - Synchronize in-memory and external representations
- Convert data types between programming language and database
- Generate DB queries to execute updates
- Manage concurrent access and transactions

5.3 Data Mapping I - Relational

Foundations

- Object-Oriented (OO) design and OO programming
  - „natural“ programming paradigm
  - Model a part of the reality as objects (entities) and their interrelationships
  - Description by models (e.g. UML class diagrams)
  - Implementation with OO programming languages like Java

- Relational databases
  - Implementation of relational algebra
    (E. F. Codd 1972)
  - Widely adopted, very mature
  - Scalable
  - Query- and manipulation languages like SQL

Impedance Mismatch

(a term from electrical engineering) summarizes the problems of mapping object oriented concepts to relational databases, and vice-versa.

- OO programming
  - classes, associations, generalization, association classes
  - object identity, objects, attribute values, polymorphism
  - relations between objects
  - access via OO programming languages

- Relational databases
  - tables (mathematical relations)
  - relation schema (attributes, domains)
  - records with attribute values (tuples, rows)
  - data dependencies (keys, foreign keys, …)
  - access via query language (SQL)

Impedance Mismatch

- OO programming
  - Constructor to create **Objects**
  - Relations via **references** (pointers)
  - Navigation between objects by **paths of references**
  - Navigability can be **restricted**
  - Execution of **behavior** (methods)

- Relational databases
  - Insert, update, and removal of **records**
  - Relations via **foreign keys** (inclusion dependencies)
  - Navigation by **computation of sets**
  - **No behavior** for records

Object Relational Mapping

- O/R mapping defines transformations that partly solve the „Impedance Mismatch“.
- Transformations required for the schema as well as for the data (instance) levels:
  - Schema: Class model ↔ relational database schema
  - Data: Object network ↔ table contents
- Problems to be solved are mappings of…
  - … classes, attributes, names, domains
  - … objects and identities
  - … relations
  - … generalization hierarchies

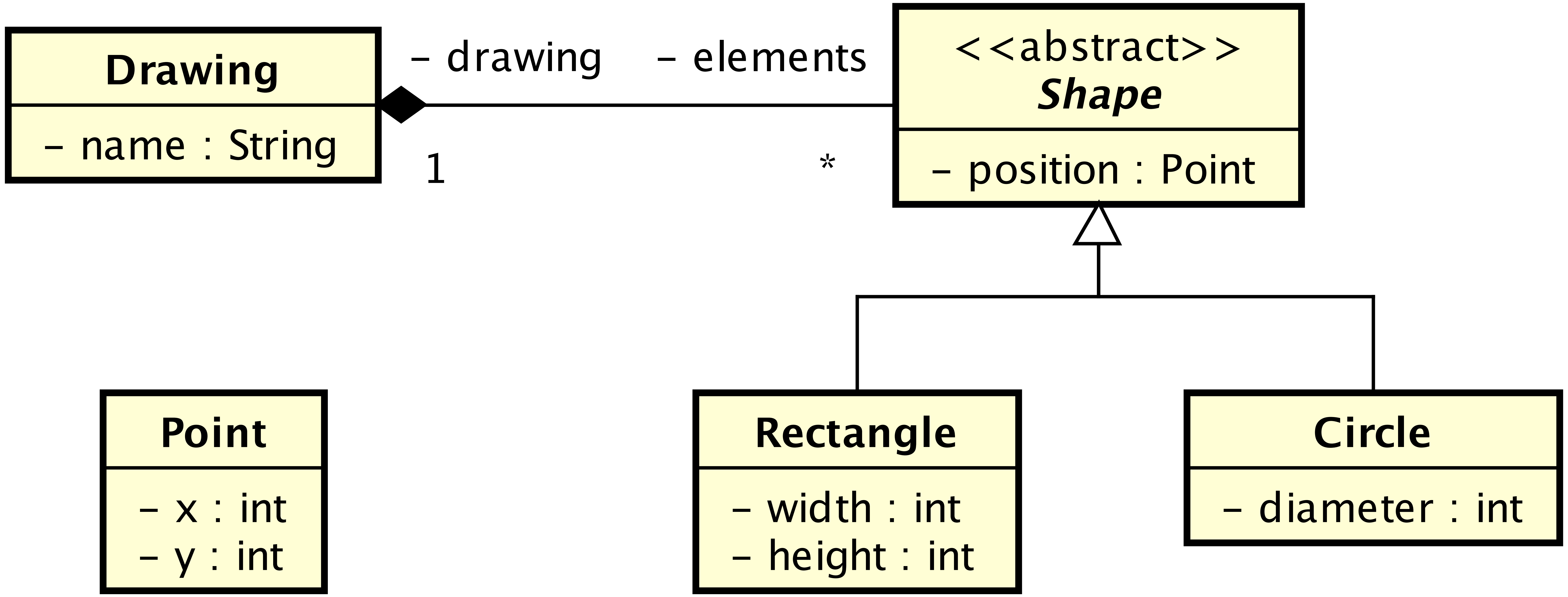
Classes and Objects

- *Classes* denote a set of similar objects, their properties (attributes with types) and their behavior (methods).
- *Generalization* between classes denote commonalities between classes. As a result, properties, relationships, and behavior is inherited by the subclasses.
- An *object* is an instance of a class with concrete property values and a set of relations to other objects

- Example:
  - Drawing, shape, rectangle, circle...
  - Class vs. instance
  - Equality vs. identity
  - Navigation in object network via paths

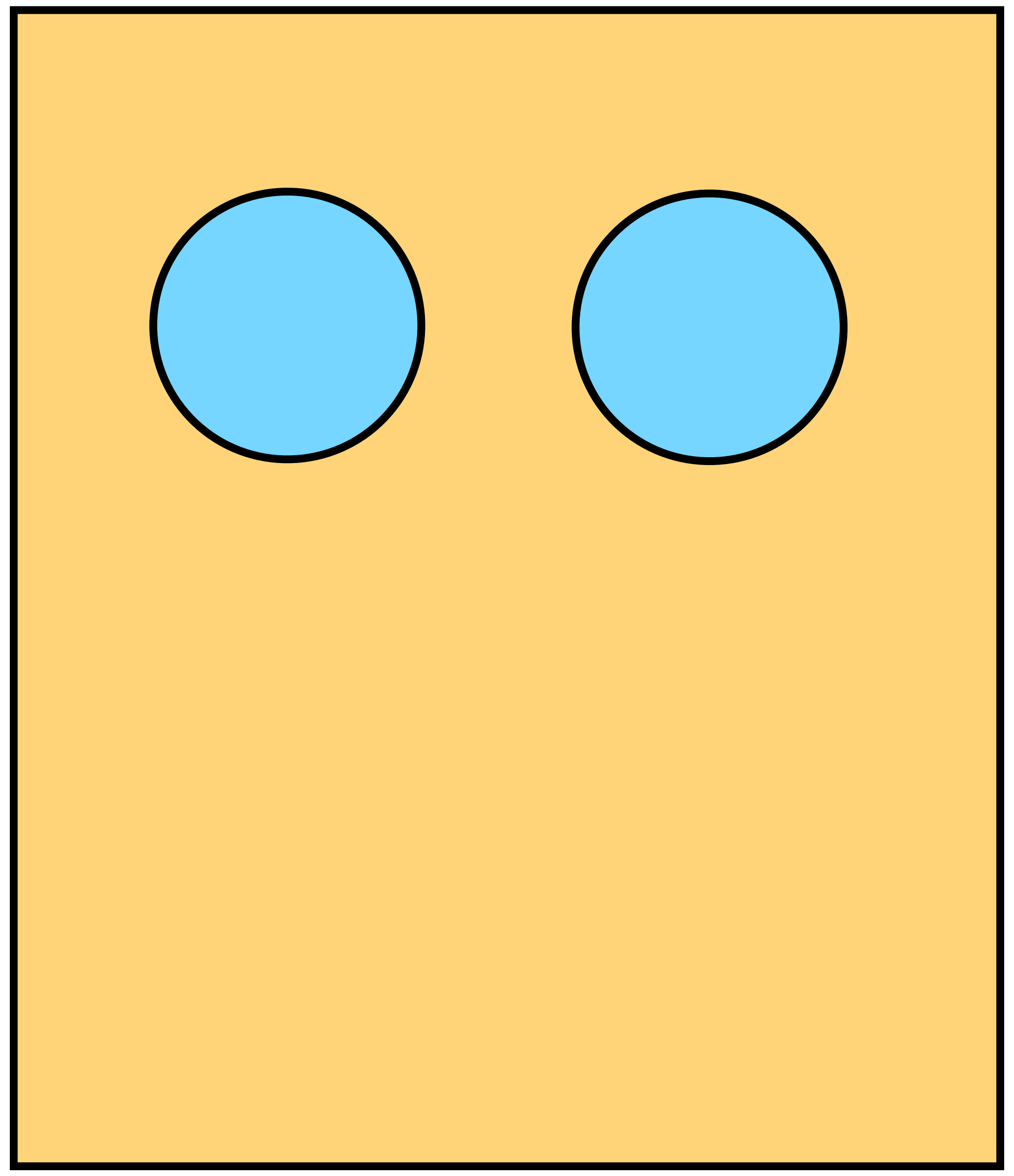
OO model for simple Drawings

Classes with Associations

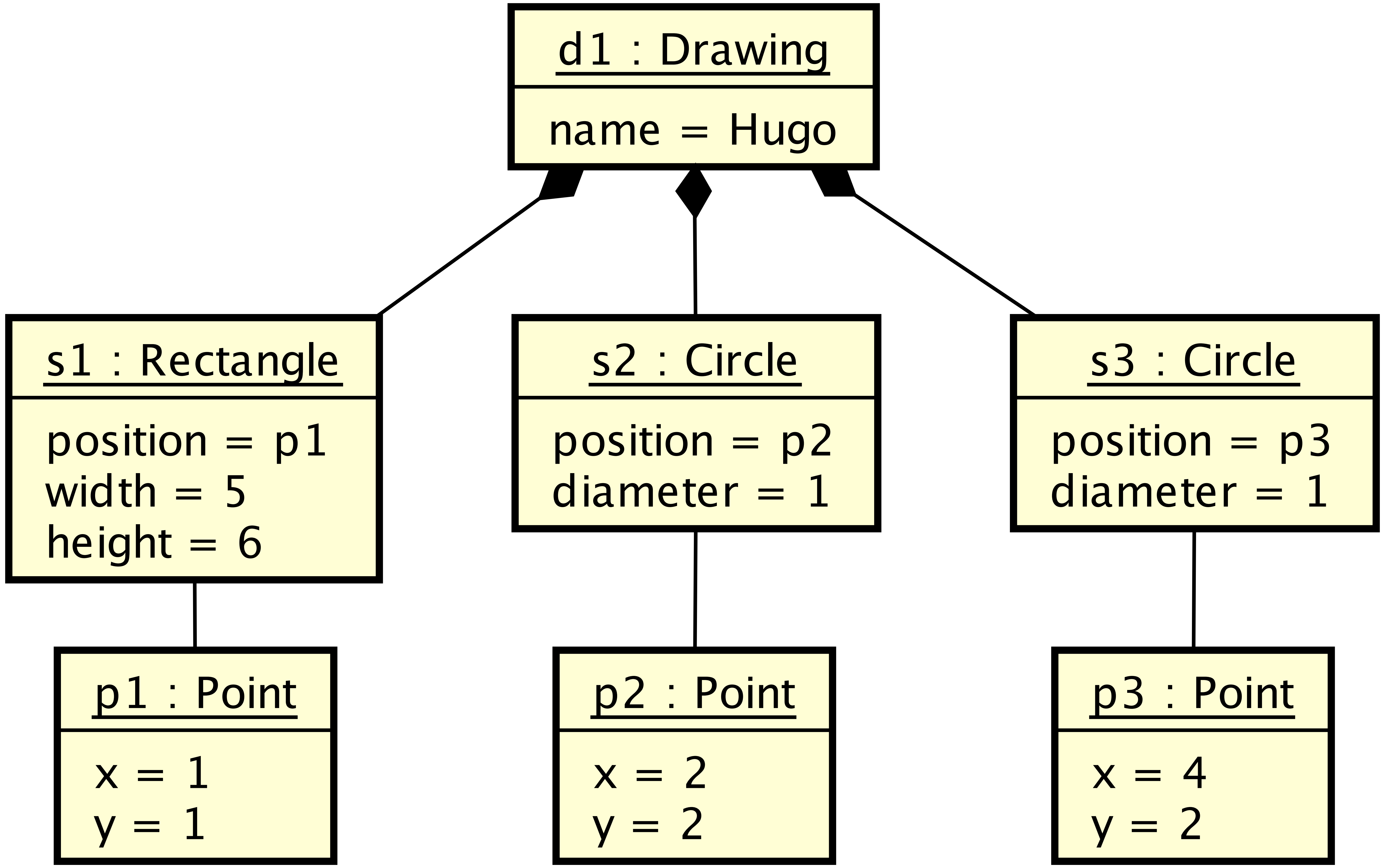


OO instance - a concrete Drawing

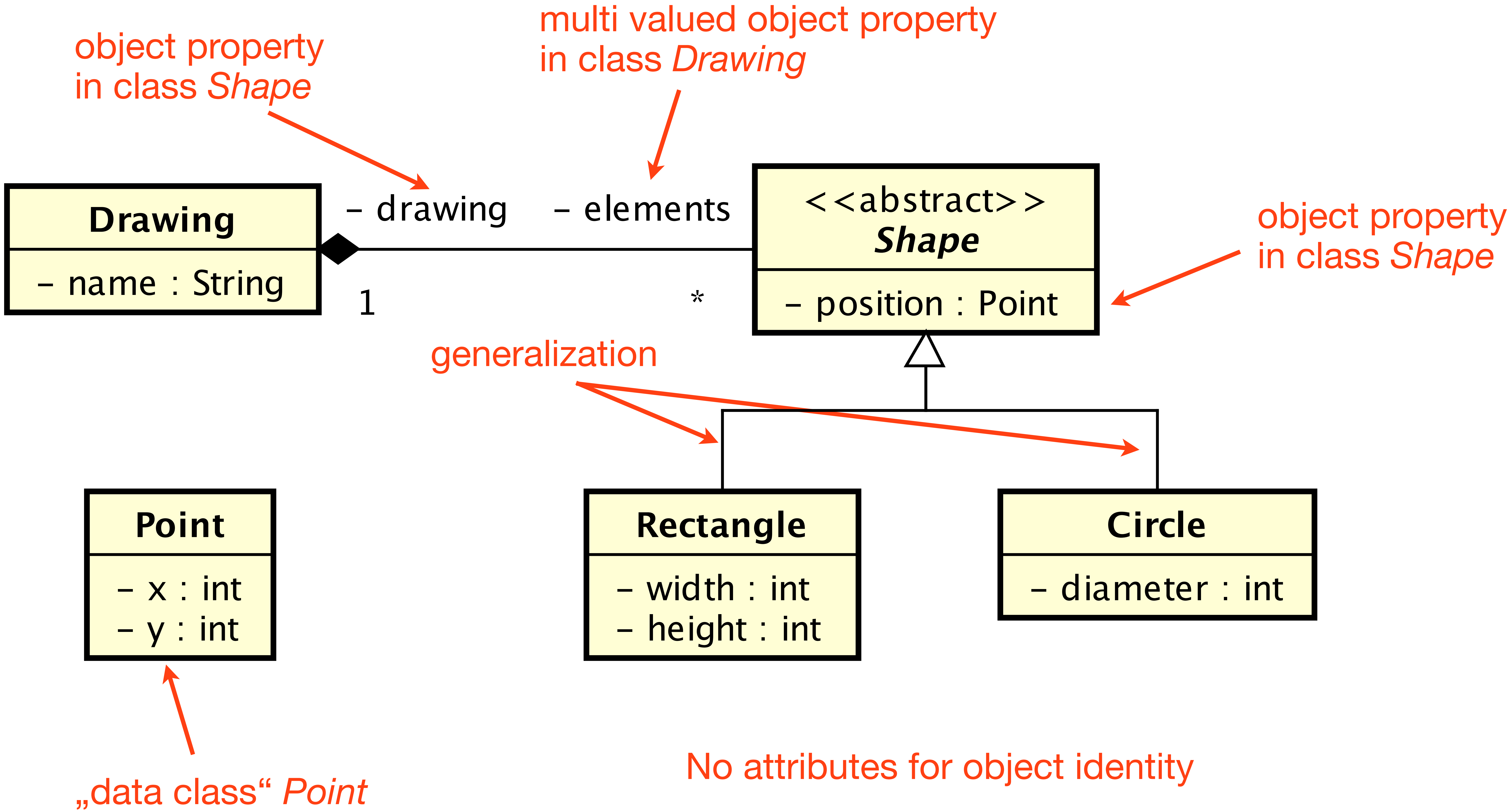
Objects with links



Hugo



Example OO Model



Identity vs. Equality

- Equality („equals“)
  - All property values of two different objects are the same, hence these objects are considered equal
  - („duplicate rows“, „doublet“)
- Identity („==“)
  - One and the same object (same address in memory)
  - Identity implies equality
- In Java programs, the object identifier is handled implicitly („this“-reference)
  - Mapping of object identifiers in O/R Mapping is mostly realized in databases by an additional numeric column „ID“ as primary key
  - Object ID is called a surrogate key and belongs to the *shadow information*
  - Shadow information are data that have to be stored in addition to the actual attribute values.

Relational database

- Description of structure by relational schema:
  Tables, attributes (columns), types, keys, foreign keys, indexes, …
- Data is kept in relations:
  Table contents, tuples (rows), attribute values
- Computation and change of data via SQL
  `SELECT, INSERT, UPDATE, DELETE`
- Navigation via **sets**:
  - Projection (restricts a table to a subset of columns)
  - Selection (restricts a table to a subset of rows)
  - Join (combines two or more tables)
- Results of computations are new relations (tables)

Problem areas in O/R Mapping

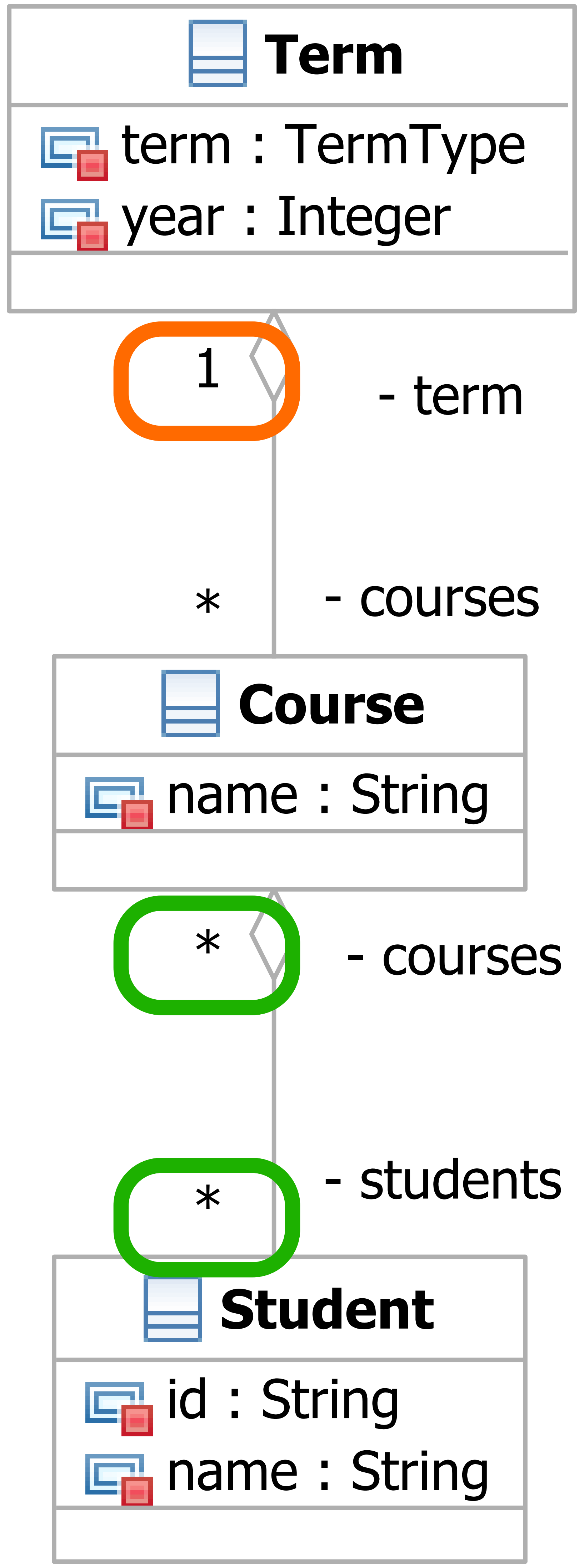
- Matching of attribute domains
- Mapping of set and list valued attributes
- Mapping of generalization hierarchies
- Compensation of different navigation mechanisms:
  path oriented in programs, set oriented in relational databases
- Preservation of data consistency between internal storage in a (Java-) program and external storage in a relational database
- Minimization of additional effort (in programming as well as at runtime)

- Relational databases generally don’t provide means to represent and execute behavior of objects.

Complex attributes

- Set and list valued attributes are forbidden in relational databases (first normal form), only atomic values allowed.
- Depending on the upper bound of multiplicity on the „far end“ association:
  - max = 1: realization by attribute (=column) possible
  - max > 1: realization by join table required
- Alternatives for structured attributes (value objects, data classes):
  - Mapping into separate table
  - Embedding of attributes (sometimes requires renaming)

Complex Attributes and Join Tables



TERM

ID	TERM	YEAR
20191	SUMMER	2019
20192	WINTER	2019

Join table required

COURSE_STUDENT

COURSE_ID	STUDENT_ID
4001	5
4001	7
4003	5
4003	7
4003	13

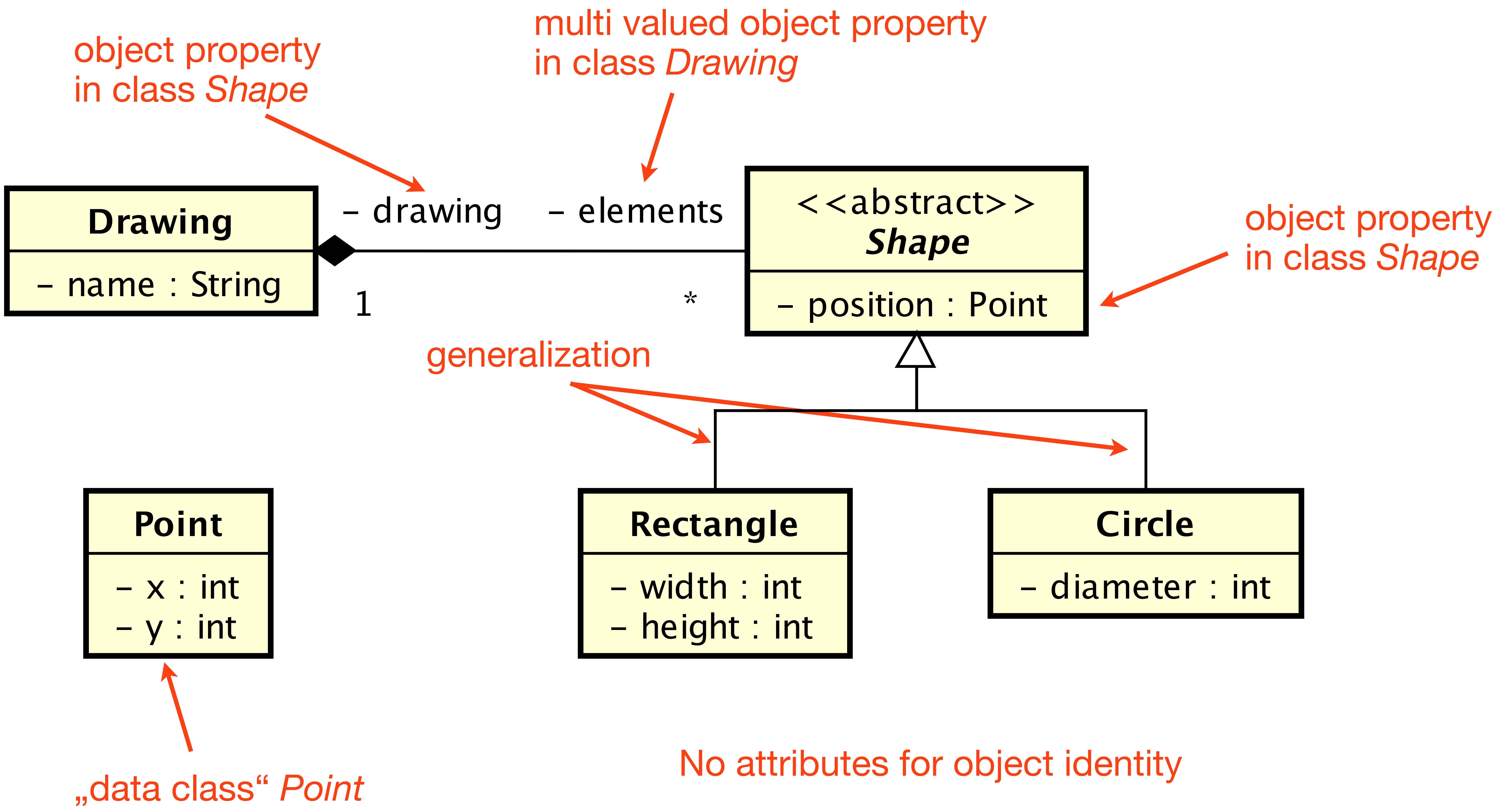
COURSE

ID	NAME
4001	Java EE
4003	EWADIS

STUDENT

ID	STUD_ID	NAME
5	208123456	Alice
7	205234567	Bob
13	206234567	Charlie

Example OO Model



No attributes for object identity

Mapping of generalization hierarchies

- Tasks to complete:
  - Representation of the subset relation between sub and super classes
  - Consideration of differences of the subclasses
  - Accounting for commonalities
  - Realization of common access (e.g. in selection and iteration) via superclasses

  - Additionally
    - Disjoint / non disjoint generalization (multiple superclasses)
    - Incomplete / complete generalization (superclass can have instances or not)

Mapping of generalization hierarchies

- Strategie for mapping generalizations

  1. One (**single**) table for **all** classes of the hierarchy
     (JPA: `SINGLE_TABLE`)

  2. **Separate** tables for **each** class of the hierarchy
     (JPA: `JOINED`)

  3. **Separate** tables for **each non-abstract** class of the hierarchy
     (JPA: `TABLE_PER_CLASS`)

Mapping of generalization hierarchies

Strategy „`SINGLE_TABLE`“

- One (**single**) table for **all** classes of the hierarchy
  - add ID column (shadow information)
  - add DTYPE as so-called discriminator column (shadow information)
  - rename attributes to prevent duplicate column names
  - unite attribute sets

- **Consequences**
  - All attributes of an object are stored in a single row of only one table.
  - Superfluous attributes contain NULL values.

Mapping - Variant #1

- multi valued attribute elements realized as column in the SHAPE table
- strategy `SINGLE_TABLE`
- class Point is **embedded**

RED column headers: primary keys

DRAWING

ID	NAME
10	Hugo
11	Cecil

FOREIGN KEY (DRAWING)
REFERENCES
DRAWING(ID)

SHAPE

ID	DTYPE	X	Y	WIDTH	HEIGHT	DIAMETER	DRAWING
20	Rectangle	1	1	5	6	NULL	10
21	Circle	2	2	NULL	NULL	1	10
22	Circle	2	4	NULL	NULL	1	10
23	Rectangle	0	0	16	9	NULL	11

Mapping - Variant #2

- multi valued attribute elements realized as column in the SHAPE table
- strategy `SINGLE_TABLE`
- class Point kept **separately**

POINT

ID	X	Y
30	1	1
31	2	2
32	2	4
33	0	0

DRAWING

ID	NAME
10	Hugo
11	Cecil

FOREIGN KEY (POSITION)
REFERENCES POINT(ID)

FOREIGN KEY (DRAWING)
REFERENCES DRAWING(ID)

SHAPE

ID	DTYPE	POSITION	WIDTH	HEIGHT	DIAMETER	DRAWING
20	Rectangle	30	5	6	NULL	10
21	Circle	31	NULL	NULL	1	10
22	Circle	32	NULL	NULL	1	10
23	Rectangle	33	16	9	NULL	11

Mapping of generalization hierarchies

Strategy “`JOINED`”

- **Separate** tables for **each** class of the hierarchy
  - add ID column (shadow information)
  - add DTYPE as so-called discriminator column (shadow information)
  - define foreign key constraints for subclass tables

- **Consequences**
  - Attribute values of a single object get distributed over multiple tables. To reconstruct (resurrect, de-serialize) an object from the database into memory, all tables have to be joined.
  - The ID sets of tables for subclasses have to be disjoint.
    Otherwise, the same object would belong to more than one class.

Mapping - Variant #3

DRAWING

ID	NAME
10	Hugo
11	Cecil

FOREIGN KEY (DRAWING)
REFERENCES DRAWING(ID)

SHAPE

ID	DTYPE	X	Y	DRAWING
20	Rectangle	1	1	10
21	Circle	2	2	10
22	Circle	2	4	10
23	Rectangle	0	0	11

FOREIGN KEY (ID)
REFERENCES
SHAPE(ID)

RECTANGLE

ID	WIDTH	HEIGHT
20	5	6
23	16	9

FOREIGN KEY (ID)
REFERENCES
SHAPE(ID)

CIRCLE

ID	DIAMETER
21	1
22	1

- multi valued attribute elements realized as column in the SHAPE table
- strategy `JOINED`
- class Point **embedded**

Mapping - variant #4

POINT

ID	X	Y
30	1	1
31	2	2
32	2	4
33	0	0

DRAWING

ID	NAME
10	Hugo
11	Cecil

FOREIGN KEY (DRAWING)
REFERENCES DRAWING(ID)

FOREIGN KEY (POSITION)
REFERENCES POINT(ID)

SHAPE

ID	DTYPE	POSITION	DRAWING
20	Rectangle	30	10
21	Circle	31	10
22	Circle	32	10
23	Rectangle	33	11

FOREIGN KEY (ID)
REFERENCES
SHAPE(ID)

RECTANGLE

ID	WIDTH	HEIGHT
20	5	6
23	16	9

FOREIGN KEY (ID)
REFERENCES
SHAPE(ID)

CIRCLE

ID	DIAMETER
21	1
22	1

- multi valued attribute elements realized as column in the SHAPE table
- strategy `JOINED`
- class Point kept **separately**

Mapping of generalization hierarchies

Strategy “`TABLE_PER_CLASS`”

- **Separate tables** for each **non-abstract class** of the hierarchy
    - add ID column (shadow information)
    - DTYPE discriminator column is optional (each type is in a different table)
    - add inherited attributes from superclasses to column definitions for subclass tables
      i.e., „flatten“ the hierarchy

- **Consequences**
    - ID sets for all tables have to be disjoint
    - All attributes of an object are stored in a single row of only one table. No joins required.
    - Iteration via superclasses is difficult.
      In the next example, there is no explicit representation for the Shape class.

Mapping - Variant #5

DRAWING

ID	NAME
10	Hugo
11	Cecil

FOREIGN KEY (DRAWING)
REFERENCES DRAWING(ID)

FOREIGN KEY (DRAWING)
REFERENCES DRAWING(ID)

RECTANGLE

ID	X	Y	WIDTH	HEIGHT	DRAWING
20	1	1	5	6	10
23	0	0	16	9	11

CIRCLE

ID	X	Y	DIAMETER	DRAWING
21	2	2	1	10
22	2	4	1	10

- multi valued attribute elements realized as column in the RECTANGLE and CIRCLE tables
- strategy `TABLE_PER_CLASS`
- class Point **embedded**

Advantages/Drawbacks of the Strategies

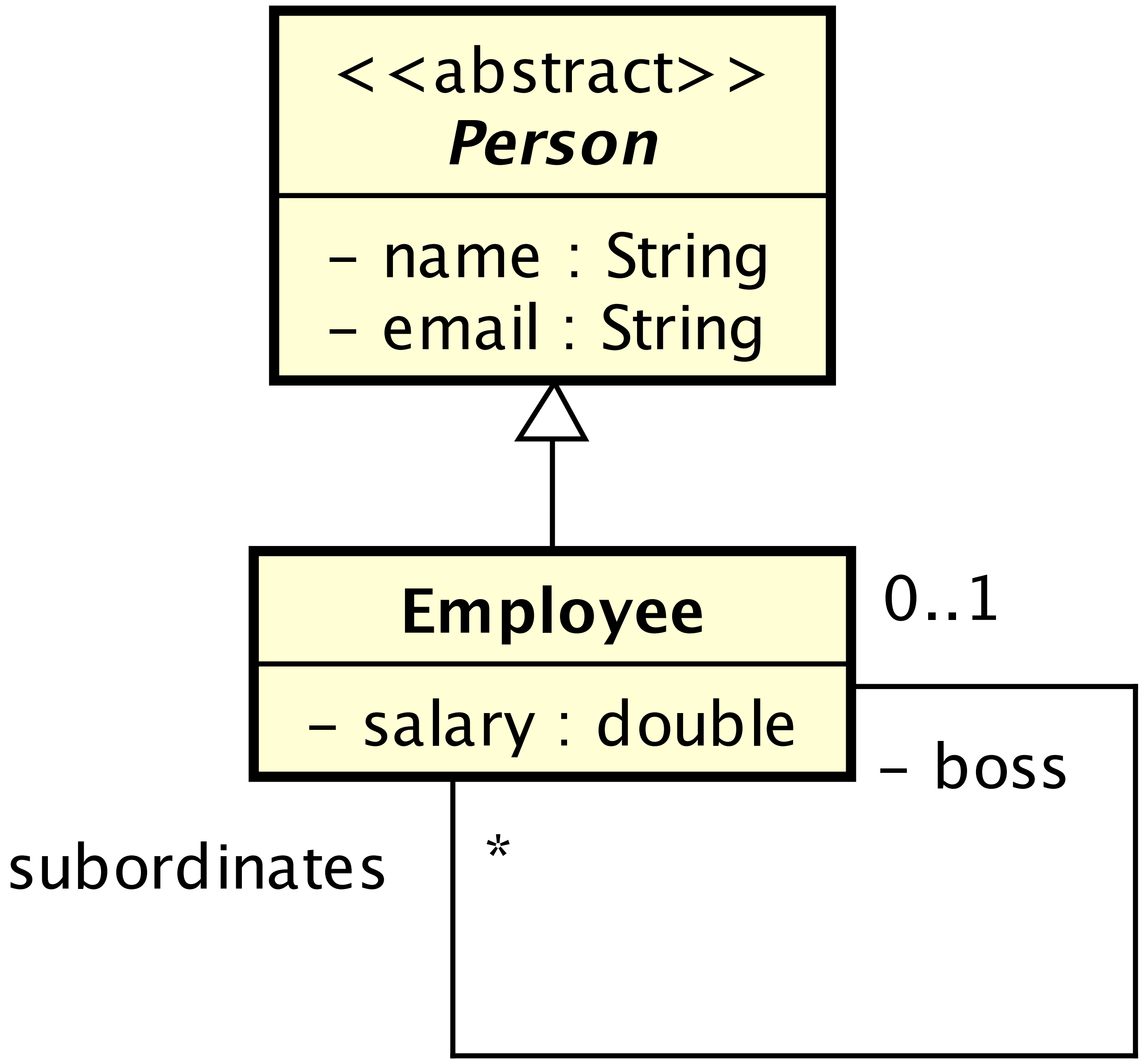
Beware! This is a simplifying summary!

- Mapping in real applications needs thorough investigation of requirements, quantity structure, memory footprint, access modes, access frequency…

	`SINGLE_TABLE`	`JOINED`	`TABLE_PER_CLASS`
Joins	none	many	few(er)
Memory and Runtime overhead	NULL values	indexes	indexes
Retrieve attributes of an object	single row	via joins	single row
SQL Access via superclass	use DTYPE	use DTYPE and joins	no simple solution
Multiple inheritance	ok	ok	ok

Recursive Associations (1)

- Association from a class to itself
- Mapping rules for complex attributes apply
- Object property `boss`
- Integrity constraint for corresponding table column
- Self-referencing table
- New records depend on existing parents
- Insertion order for records: parents before children



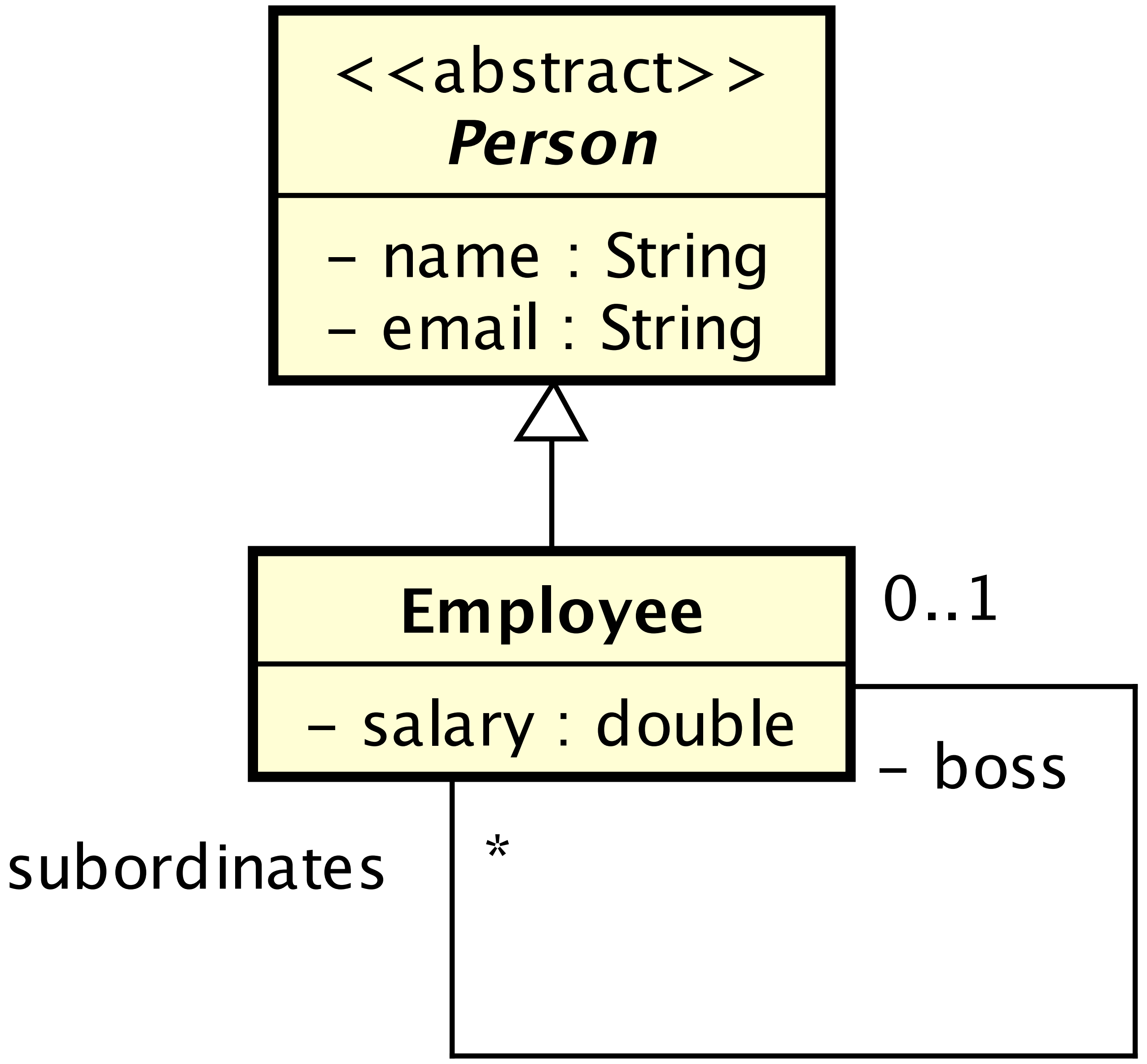
PERSON

ID	DTYPE	NAME	EMAIL	SALARY	BOSS
10	Employee	Alice	…	…	13
11	Employee	Bob	…	…	13
12	Employee	Charlie	…	…	13
13	Employee	Debbie	…	…	NULL
14	Employee	Eleanor	…	…	10

FOREIGN KEY (BOSS)
REFERENCES PERSON(ID)

Recursive Associations (2)

- Alternate mapping with join table
- Inserts into `PERSON` table in arbitrary order
- Join table depends on existing person records



PERSON

ID	DTYPE	NAME	EMAIL	SALARY
10	Employee	Alice	…	…
11	Employee	Bob	…	…
12	Employee	Charlie	…	…
13	Employee	Debbie	…	…
14	Employee	Eleanor	…	…

FOREIGN KEY (PERSON)
REFERENCES PERSON(ID)

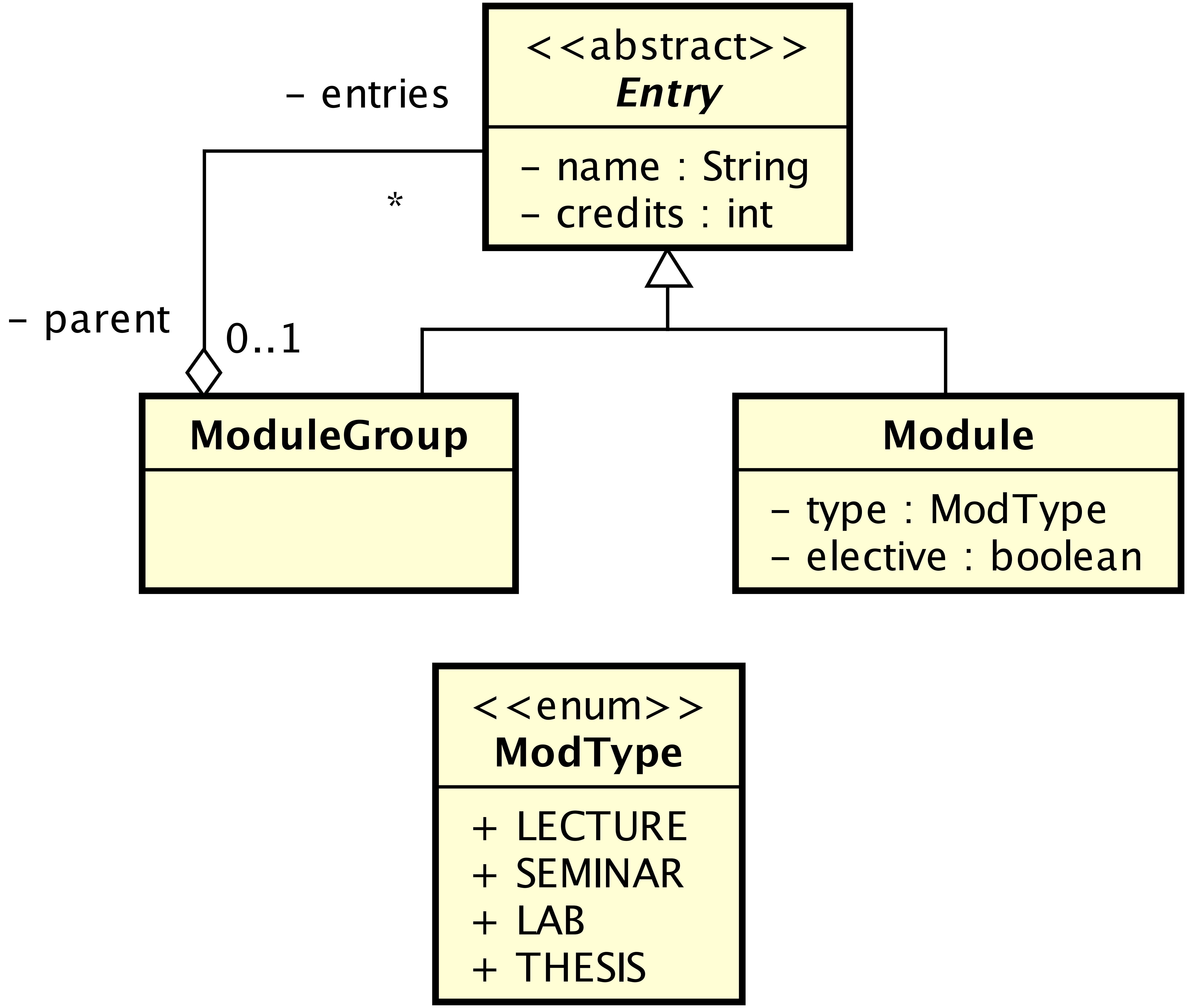
PERSON_BOSS

SUBORD_ID	BOSS_ID
10	13
11	13
12	13
14	10

FOREIGN KEY (BOSS)
REFERENCES PERSON(ID)

Recursive Associations (3)

- Arbitrary hierarchies via composite pattern lead to indirect dependencies
- `Module` and `ModuleGroup` both refer to `parent`
- Depending on mapping strategy for the generalization similar solutions (see previous slides):
  - self-referencing tables
  - join tables
- Creation of tables and constraints requires specific order due to cyclic dependencies



What we have learned…

Persistence (Part I)

- ✓ Overview
- ✓ Data Properties
- ✓ Persistence Tasks
- ✓ O/R Mapping

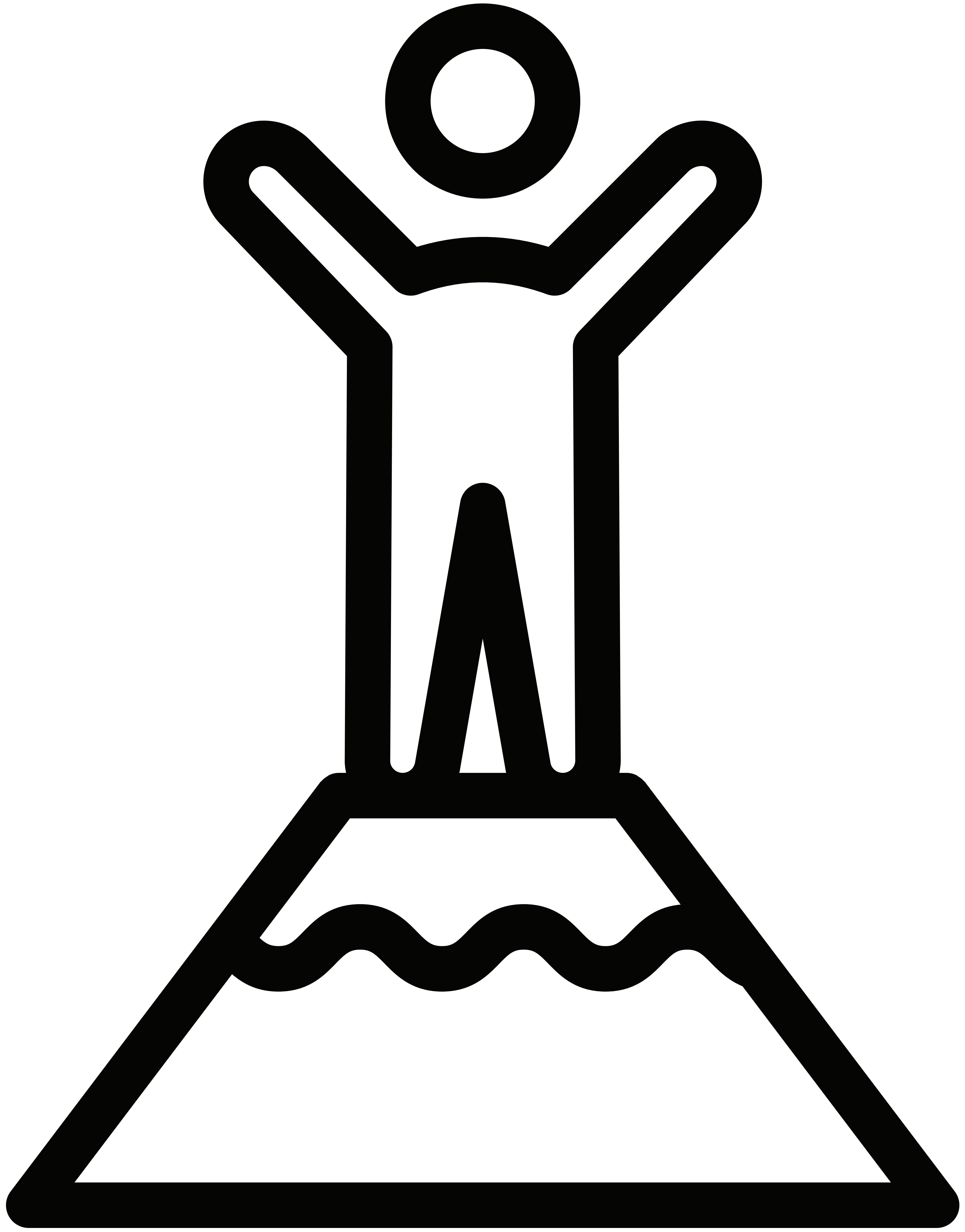


Image: colourbox.de