University of St.Gallen

OOP & Database Applications

Lecture 1:

Prof. Johannes Binswanger

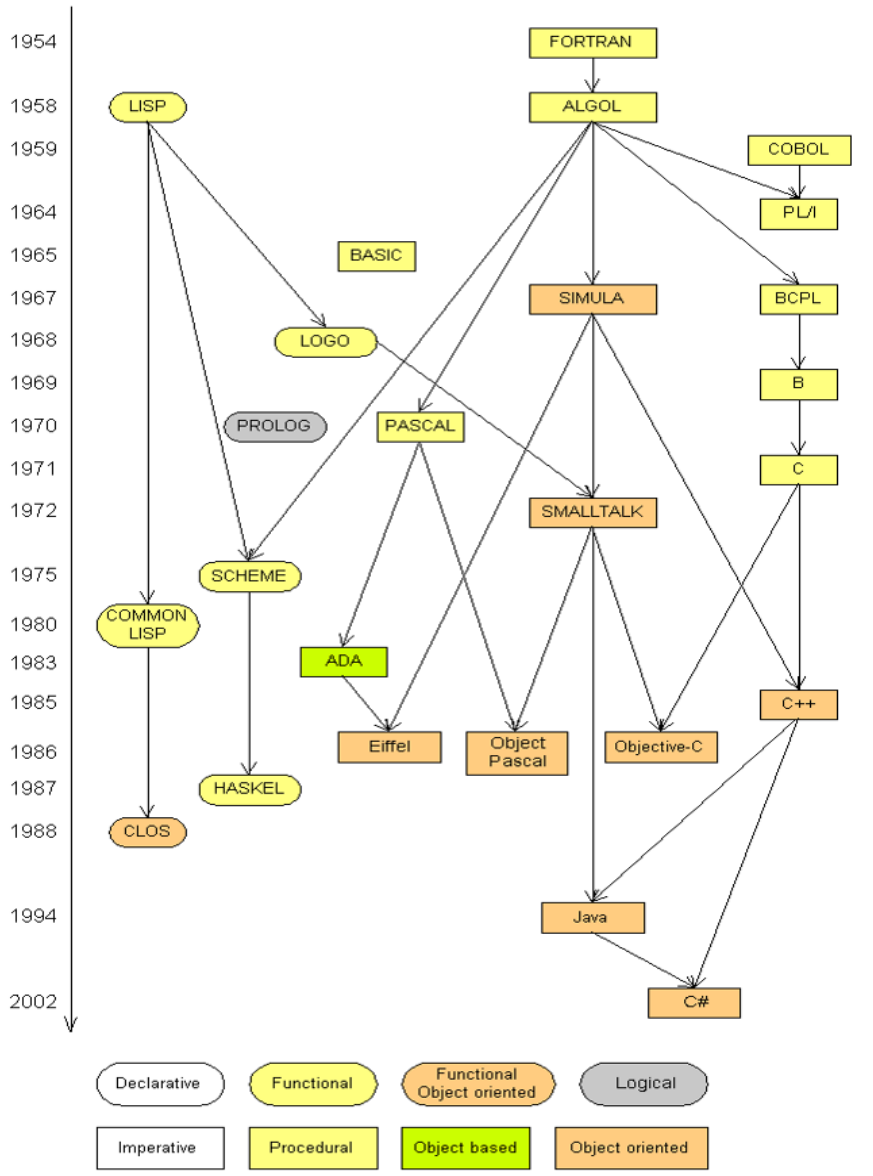
Ruben Zürcher

Programming Paradigms

Procedural	Bases on prodcedures, routines and subroutines, follows a top down approach
Functional	Avoids changing state and mutable data, similar approach to mathematical functions
Objectural	Encapsulates data and methods in objects, objects can interact.

Historical Overview

- FORTRAN:
  - Science/Engineering
- Cobol
  - Business Data (UBS backend)
- LISP
  - Logic and AI
- BASIC
  - Ease of use



Salih, Shadman. (2014)

Programming Principles

- KIS (Keep It Simple)
  - No overengineering, no spaghetti code
- DRY (Do not Repeat Yourself)
  - Code duplication = bug reuse
  - Rule of three:
    - Copy once, copy twice => make a function
- YAGNI (You ain’t gonna need it)
  - Write code when you need it, no hoarding of ideas

Python

- Easy to understand
- Very flexible
- Has amazing libraries
- Good start for Big Data and AI

- We will use Python 3 !!!

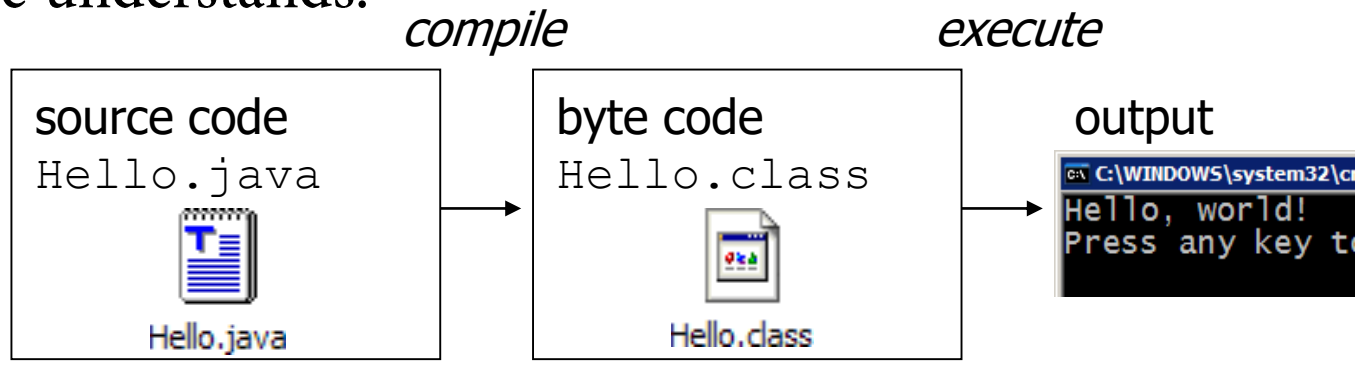
Programming basics

- **code** or **source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console**: The text box onto which output is printed.
  - Some source code editors pop up the console as an external window, and others contain their own console window.

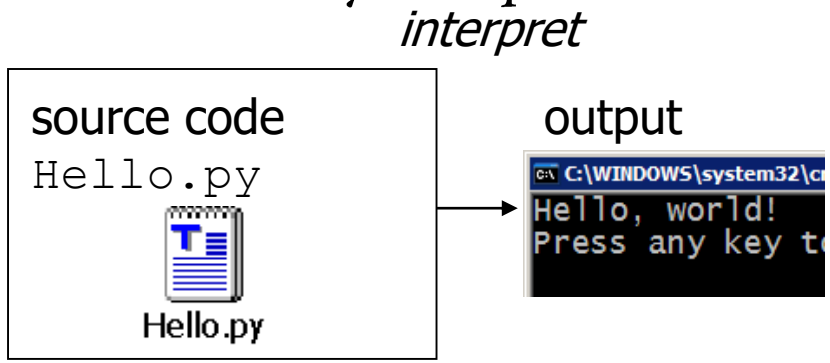
Stepp, Marty. (2009)

Compiling and Interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



- Python is instead directly *interpreted* into machine instructions.



Stepp, Marty. (2009)

Set Up

- Win:
  - Download and Install VMware Workstation Player
  - https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/14_0

- Mac:
  - Download Virtual Box
  - https://www.virtualbox.org/wiki/Downloads

OOP & Database Applications

Lecture 2: Datatypes, Operations & Statements

Prof. Johannes Binswanger

Ruben Zürcher

Variables

- variable:
  - A piece of memory that can store a value
  - Assign value with = (e.g. x = 4)
  - Rules for naming a variable:
    1. Names cannot start with a number
    2. There can be no spaces in the name (use _ instead)
    3. Cannot incorporate any of these symbols :'",<>/?|\()!@#%&^$*~-+
    4. It is considered best practice that name are lowercase (exception: cubeVolume)

Numbers

```
# Integers
age = 24
age = int("24")

# Floating Point
e = 2.71828182845
e = float("2.71828182845")
```

Strings

```
# String
place = "University of St. Gallen"
place = 'University of St. Gallen'


# Multi-Line String
info = """this course takes place at the University
of St. Gallen on Wednesday at 16:00."""
```

Lists

```
# Lists
things = []

# Appending & Extending
things.append("chill")
things.extend([False, 42])

=> things = ["chill", False, 42]
```

Booleans

```
# a boolean which is True
is_True = True


# variables which return True
things_True = True and 1 and "text" and {'a':'b'}
and ['c':'d']


# variables which return False
things_False = False or 0 or"" or {} or [] or None
```

Arithmetic Operations

```
a = 10          # 10
a += 10         # 20
a  -= 10        # 0

b = a + 10      # 20
c = a  - 10     # 0
```

```
d = a * 2       # 20
e = a / 2       # 5
f = a % 3       # 1
g = a ** 2      # 100

h = a + a       # 20
```

String Operations

```
uni = "HSG " + "ETH"
uni +=" UZH"
# HSG ETH UZH


Sports = ', '.join(['Hockey', 'Soccer'])
# Hockey, Soccer
```

String Operations

```
# String with '
myName = "I'm Fabio"
# I'm Fabio


# String with «
Quote = '"To be or not to be"'
# "To be or not to be"
```

String Operations

```
# using escape characters in print("")

print("This is the first line\n this is the second line")
# This is the first line
# this is the second line

print("This is the first part\t this is the second part")
# This is the first part      this is the second part
```

String Operations

```
# length of string
myName = "I'm Fabio"
len(myName)
# 9

myName[0]
# I
```

```
myName[4:]
# Fabio


myName[:4]
# I'm

myName[-1]
# o
```

String Operations

```
myName[::1]
# I'm Fabio


myName[::2]
# ImFbo


myName[::-1]
# oibaF m'I
```

List Operations

```
numbers = [1, 2, 3, 4, 5]

len(numbers)
# 5
numbers[0]
# 1
numbers[0:2]
# [1, 2]
```

```
numbers.pop()
# 5
numbers
# 1, 2, 3, 4

Numbers.reverse()
# 4, 3, 2, 1
```

List Operations

```
mix = ['c', 'a', 'e', 'b']

mix.sort()
# ['a', 'b', 'c', 'e']
```

```
l1 = [1,2,3]
l2 = [4,5,6]
l3 = [7,8,9]
matrix = [l1,l2,l3]
# [[1, 2, 3,], [4, 5, 6], [7, 8, 9]]

matrix[1][2]
# 6
```

Logical Comparison

```
# logical and
a and b
# logical or
a or b
# logical negation
not a
# compound
(a and not (b or c))
```

Arithmetic Comparision

```
# ordering
a > b
a >= b
a < b
a <= b
# equality/difference
a == b
a != b
```

Identity Comparison

```
# identity
1 is 1 == True
# Non Identity
1 is not '1' == True

# example
1 and True == True
1 is True == False
```

If Else

```
if True:
        print('that is correct')
# that is correct

flag = True
if flag:
        print('ready steady go')
else:
        print('please wait')
# ready steady go
```

If Elif Else

```
course = 'OOP‘
if course == 'SGMM':
      print('That will be though')
elif course == 'OOP':
      print('Best course ever')
else:
      print('we do not offer that course')
# Best course ever
```

For Loops

```
nums = [1, 2, 3, 4, 5]
for num in nums:
        print(num)
# 1
# 2
# 3
# 4
# 5
```

```
nums_sum = 0
for num in nums:
        nums_sum += num

print(nums_sum)
# 15
```

While Loop

```
counter = 0
while counter < 4:
        print('iteration is at: ' + counter)
        counter +=1

# iteration is at: 0
# iteration is at: 1
# iteration is at: 2
# iteration is at: 3
```

While Loop

```
counter = 0
while counter < 4:
      print('iteration is at: ' + counter)
      if counter == 1:
         pass
      elif counter % 3 == 0:
         break
      else:
         print(counter)
      counter += 1
```

University of St.Gallen

OOP & Database Applications

Lecture 3: Methods & Functions

Ruben Zürcher

Functions

- Groups multiple lines of code into one block
- Allows to specify input parameters which are used in the function
- Reusable code
- Common tasks are best written into a function
- `def` is a built-in keyword which tells python that you are about to *create* a function
- built-in functions of Python 3

Functions

```
def nameOfFunction ():
     pass                          #never forget to indent

def add_function(arg1,arg2):
     return arg1 + arg 2

print(add_function(1,2))
# 3
```

Functions

```
def squareNumber(num):
        """
        INPUT: a number
        OUTPUT: the square of the number
        """
        return num**2
```

Methods

- Methods are functions built into objects e.g a List
- Perform specific routines on an object and can take argument
  - Form: object.method(arg1,arg2,self…) e.g. List.append(9)

- We will go more indepth during the Object-Oriented-Programming part of this course

Method

```
a = [1, 2, 3, 4, 5]
a.append(6)

a.count(3)

help(a.count())
# help will print out the docstring and give you
more information about a method/function
```

Lambda

- Is a «anonymous» function
- Has the same properties as a normal function
- Can be created ad-hoch without the need of `def`
- **Only a single expression, not a block of statements**
- Can be used as a parameter in a function/method (advanced)
- Further reading

Lambda

```
def squareNumber(num):
     return num**2


def squareNumber(num): return num**2
#bad coding style!!!


lambda num: num**2
squareNumber2 = lambda num: num**2
```

Scopes

- When you create a variable, the variable name is stored in a namespace.
- The variables names have a so called scope. This scope defines the visibility of the variable in the code.

Scope

```
x = 25

def printer():
    x = 10
    return x

print(x)
print(printer())
```

Scopes

- Local : variables created within a function
- Enclosing functions: variables of all enclosing functions, from inner to outer.
- Global: variables created at the top-level of a file, or decleared as `global` within `def`
- Built-in: precreated variable by python (similar to existing functions)

Enclosing Functions Scope

```
name = "this is a global name"

def say():
    # Enclosing function
    name = "Fabio"
    def hey():
        print("hey " + name)
    hey()
say()
```

Global Scope

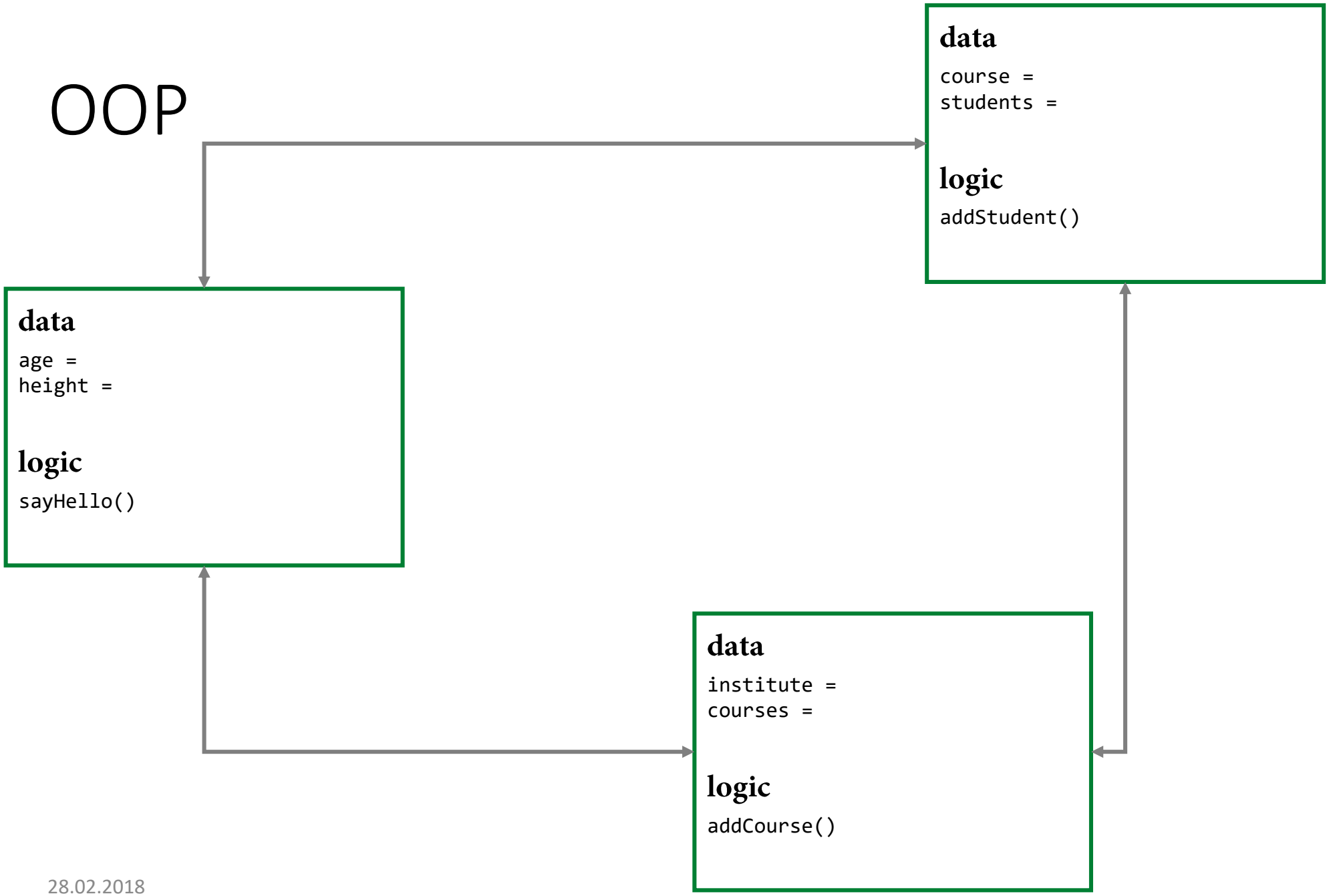
```
x = 25

def printer():
    global x
    x = 10
    return x

print(x)
print(printer())
```

OOP & Database Applications

Lecture 4: Object Oriented Programming

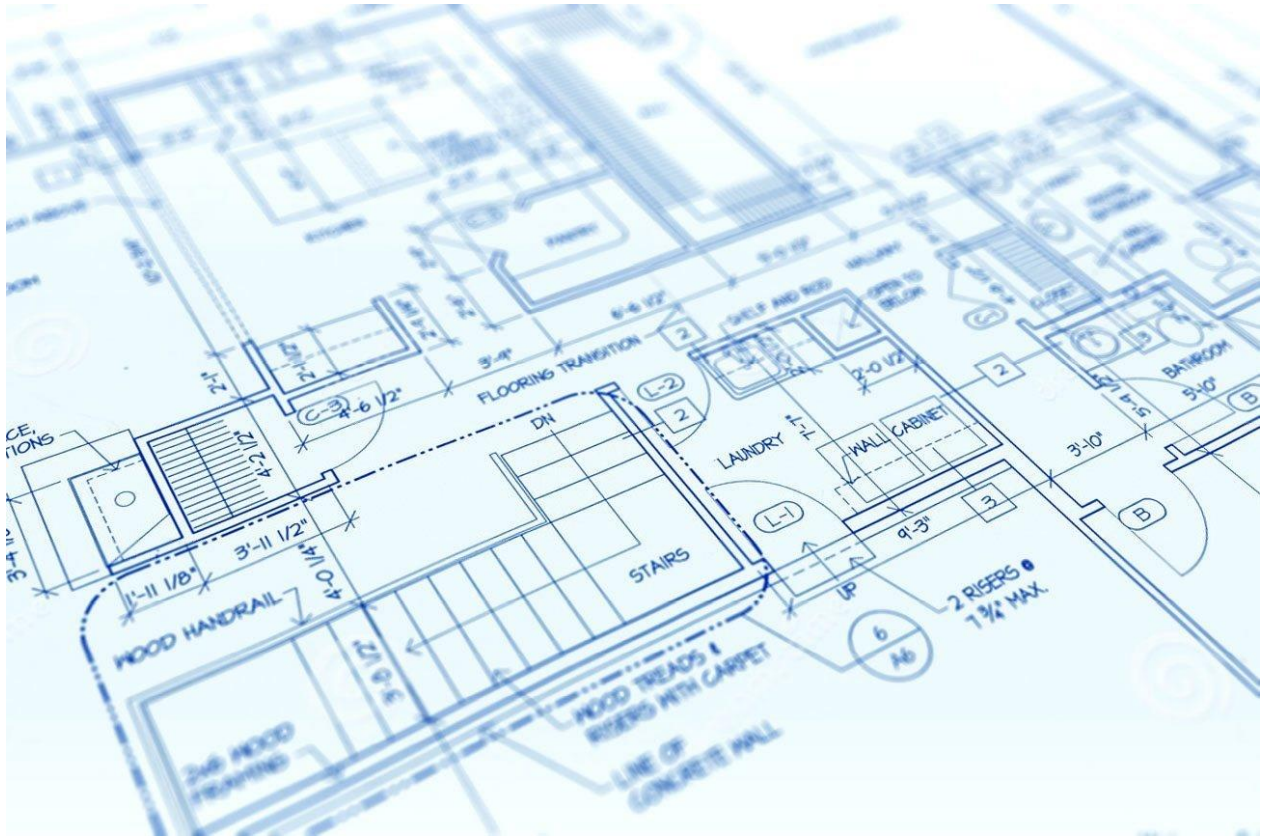
Ruben Zürcher

OOP


data
course =
students =
logic
addStudent()
data
age =
height =
logic
sayHello()
data
institute =
courses =
logic
addCourse()

Class

- It is a blueprint, definition and description of a case



Class Examples

Car

E-Mail

Stock

User

Date

Window

Class

Properties

name
birthdate
height
weight
gender

Methods

walk
run
jump
speak
eat

Object

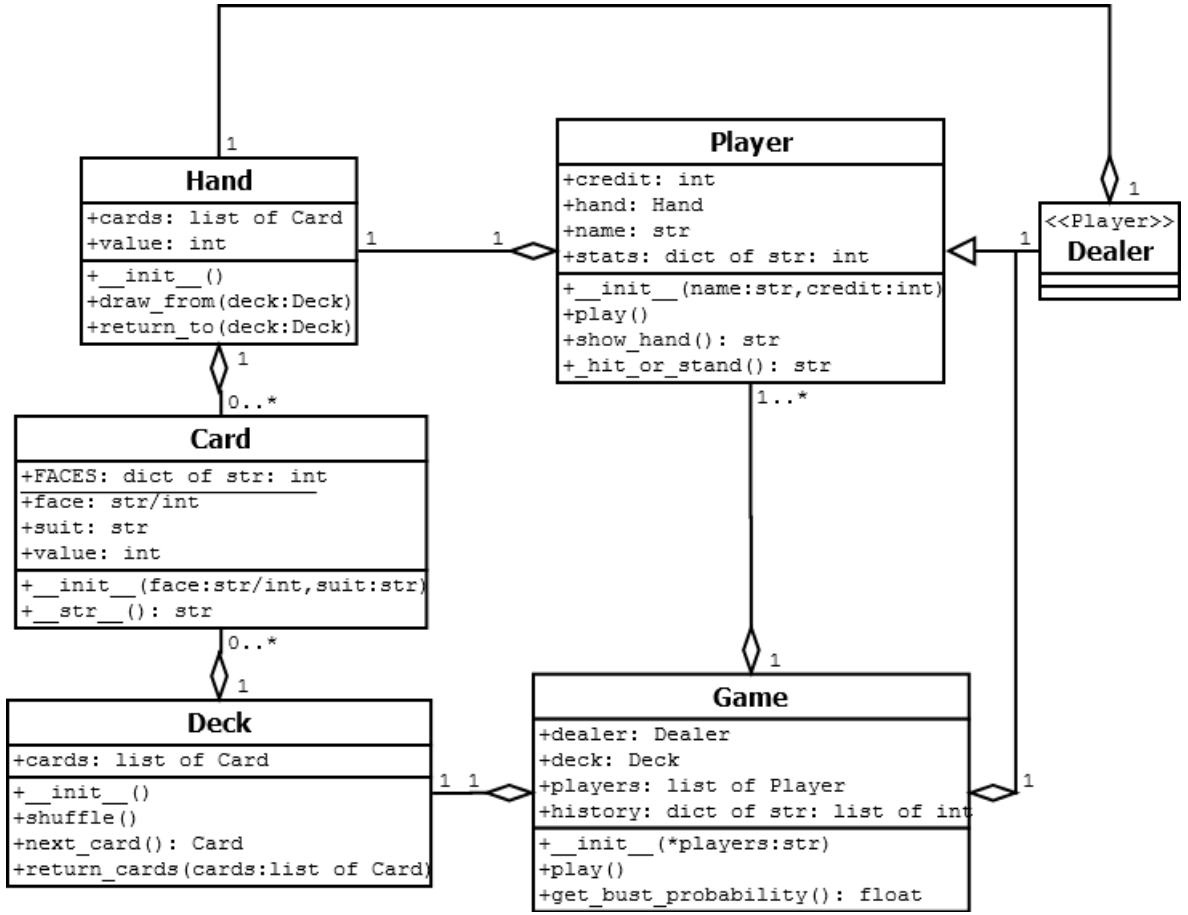
- An object is created from a class, it is an instance of a class

Encapsulation

Encapsulation

Class Diagramm

Class Diagramm



Flipboard Challange

- 3 Groups
  - Cinema
  - ATM
  - Library (with borrowing)

- Airline Challange

OOP & Database Applications

Lecture 5: Object Oriented Programming in Python

Ruben Zürcher

Professor Binswanger

Class

```
Class Employee(object):
   age = 0
   name = «»
```

Constructor

```
class Employee:

    # the constructor defines the basic properties
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
```

Method

```
class Employee(object):

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary

   def displayEmployee(self):
      print("Name: « +self.name +", Salary: «+
            self.salary)
```

Initialize an Object

```
emp1 = Employee(«Fabio», 85000)

emp1.displayEmployee()

# Name: Fabio, Salary: 85000
```

Getter and Setter (generic)

- Getter returns a value of a variable
- Setter sets a new value for a variable

Why?:

- for completeness of encapsulation
- to implement validation
- to maintain a consistent interface in case internal details change

Getter and Setter method

```
class Employee(object):
    name = «???»

    def getName(self):
        return self.name

    def setName(self,name):
        self.name = name
```

Importing a Class

To import a class we use (at the top of the file):

from <filename> import <classname>

Our Employee Class is stored in employee.py

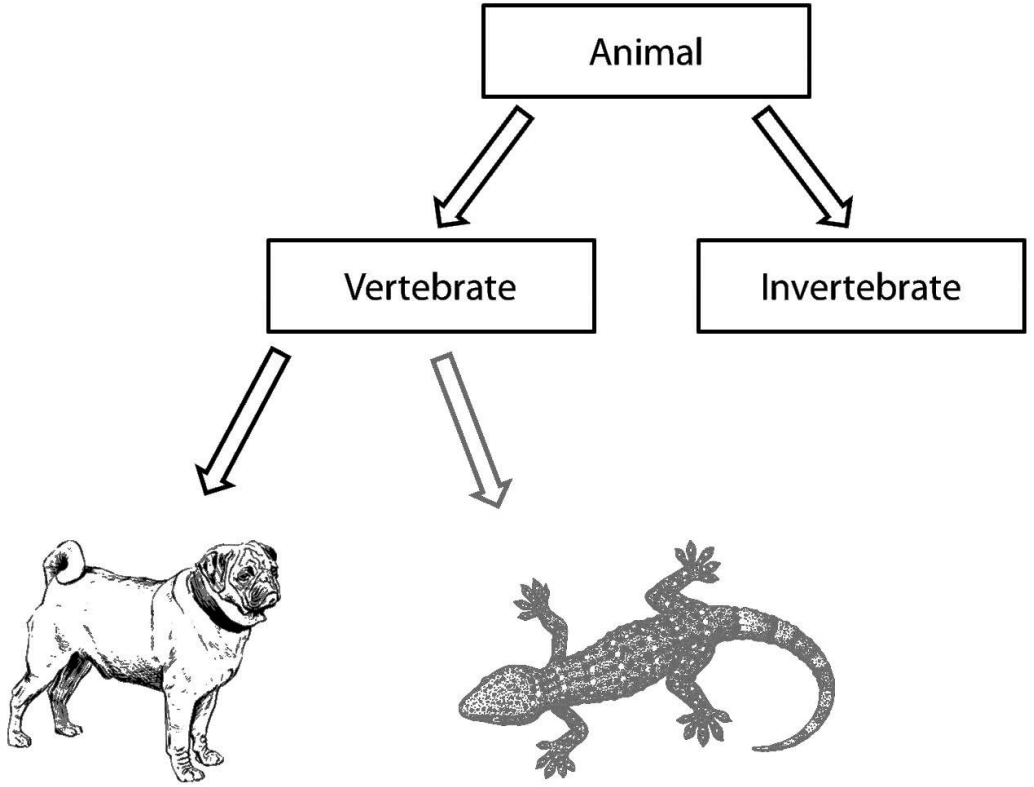
from employee import Employee

Inheritance

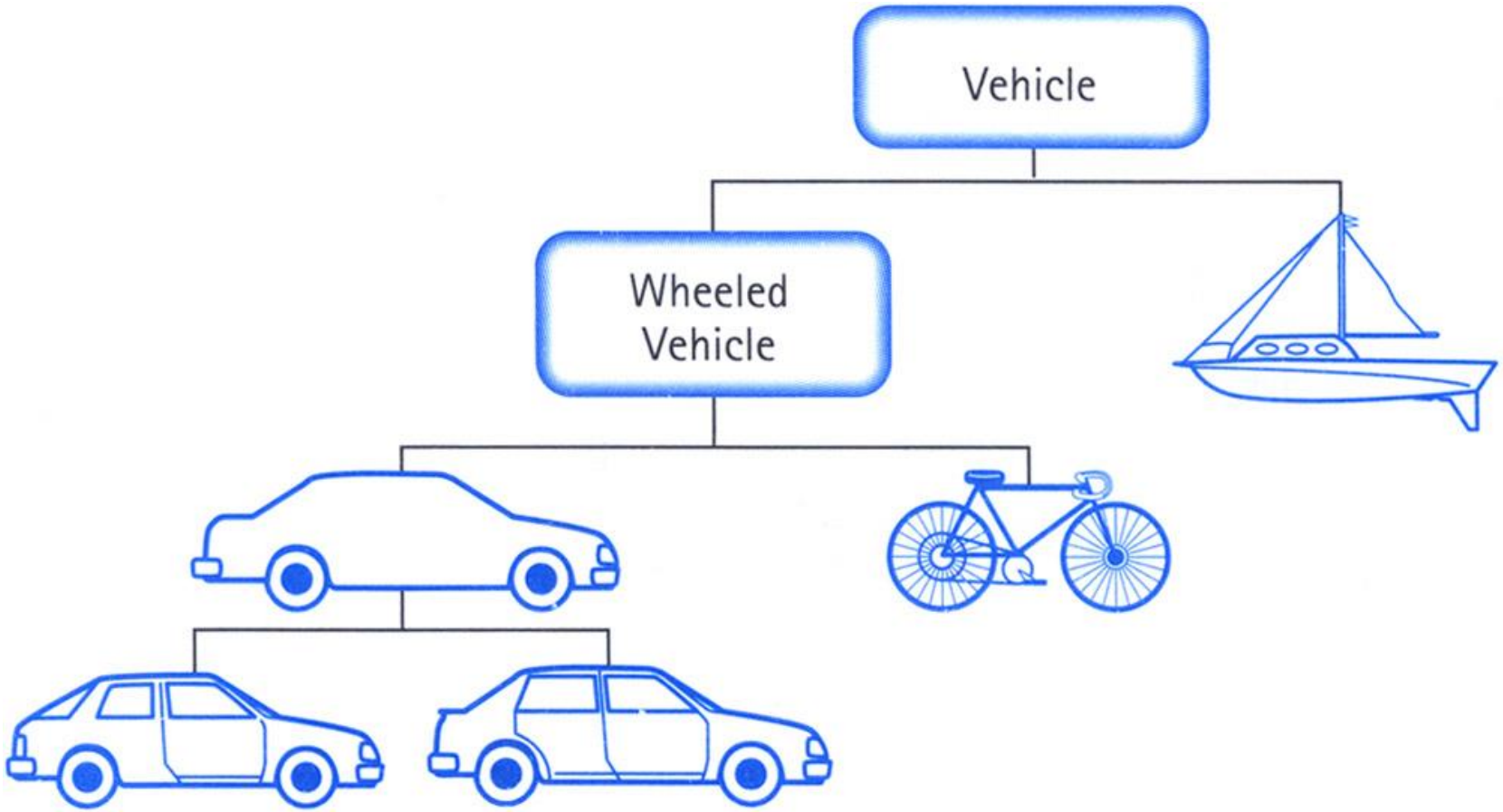
- Inheritance is the concept of a child class (sub class) automatically inheriting the variables and methods of ist parent class (super class)

- The benefit of inheritance is the concept of reusability of code:
  - A subclass only needs to define the diffrences between itself and the super class.

Inheritance



Inheritance



Inheritance

```
class childClass(parentClass):
    name = «???»

    def getChildAttribute(self):
        return self.childAttribute

    def setChildAttribute (self, childAttribute):
        self.childAttribute = childAttribute
```

Task

- Construct the class diagramm out of the Library code files & enhance the code so that it runs
- Build a text based game like Zork (seperate the «game engine» from the content of the game). Classes as data objects (rooms, characters, weapons).
- Try to model an airline's operation (airline challange). You will need among others a fleet, routes & airports.

- [Try to model the international trade]

University of St.Gallen

OOP & Database Applications

Lecture 6: Application Programming Interface - API

Ruben Zürcher

Professor Binswanger

API Definition

- An application program interface (API) is code that allows two software programs to communicate with each other.
- An API provides the guidance and assistance that a developer should need to interface two systems.

- An API is the interface through which you access someone elses code or through which someone else's code accesses yours. In effect the public methods and properties.

API Usage

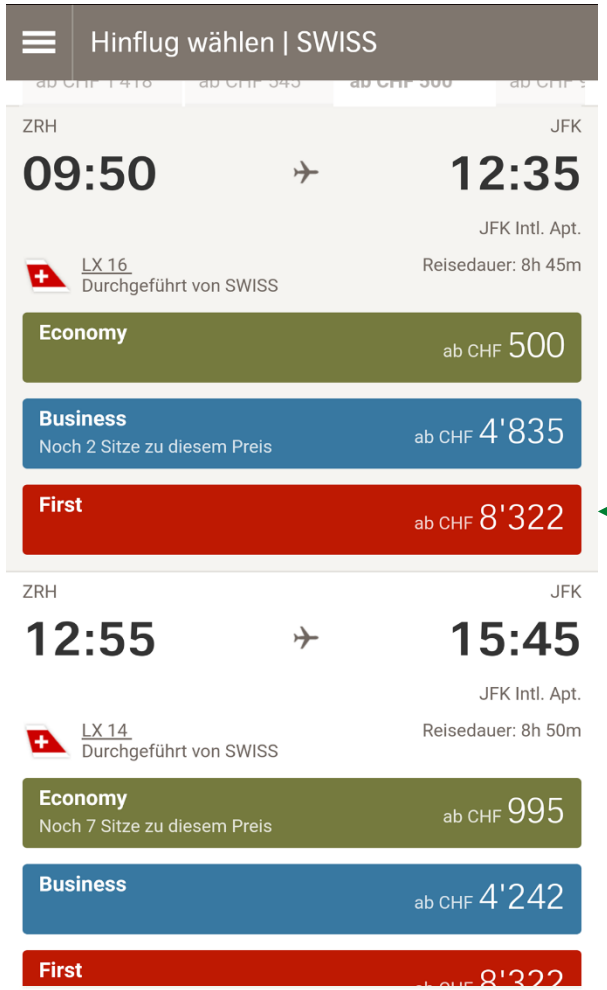

ASSETS
Data, Software,
Services, Products
APIs
DEVELOPERS
Use APIs
to create apps
SOFTWARE &
APPLICATIONS
powered by APIs
END USERS

API Example


Flug buchen
Flug buchen
Hin- und Rückflug
Von
Basel (BSL)
Nach
Reisedaten
Hinflug
27.03.2018
Rückflug
03.04.2018

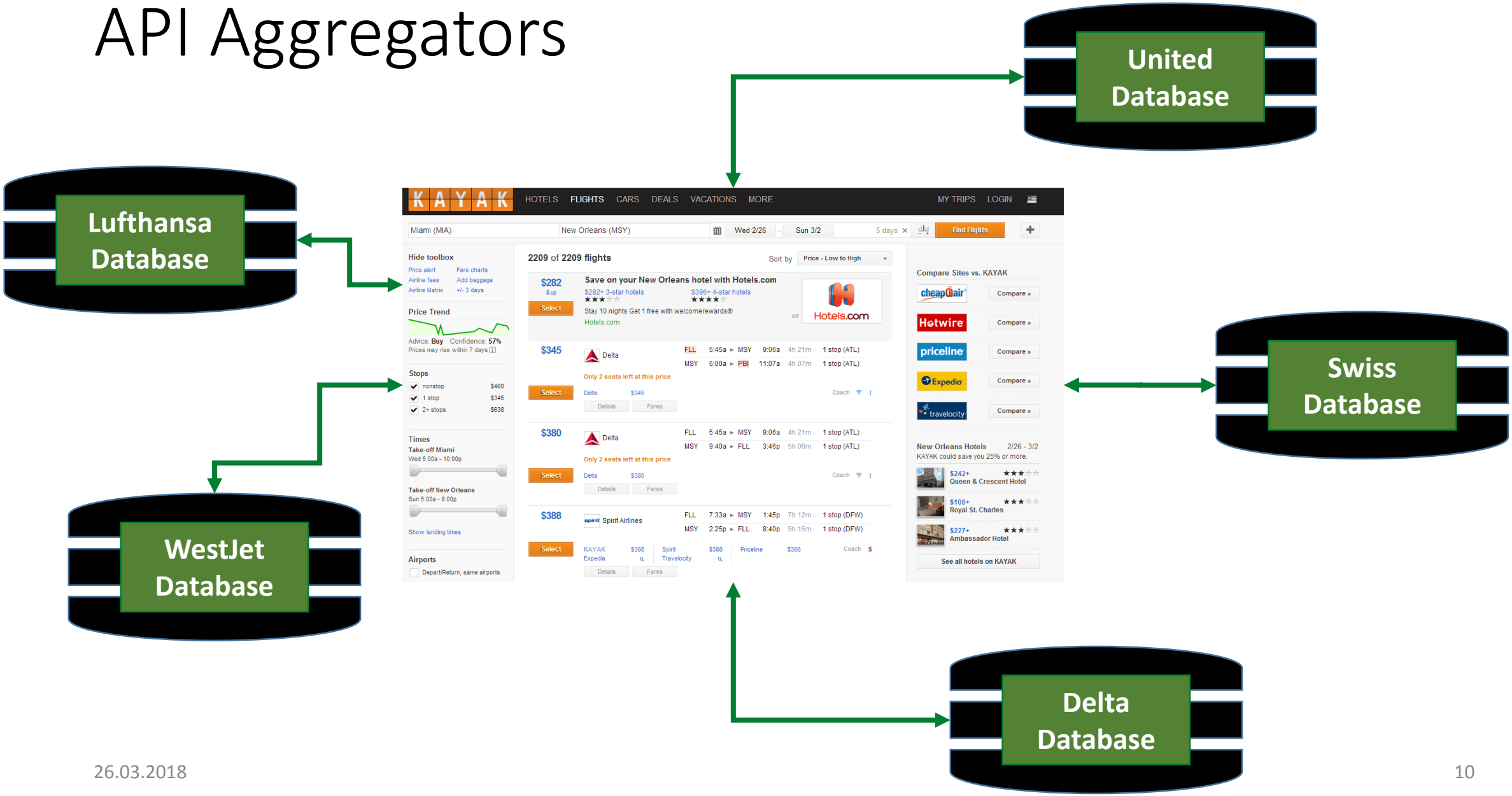

Swiss
Database

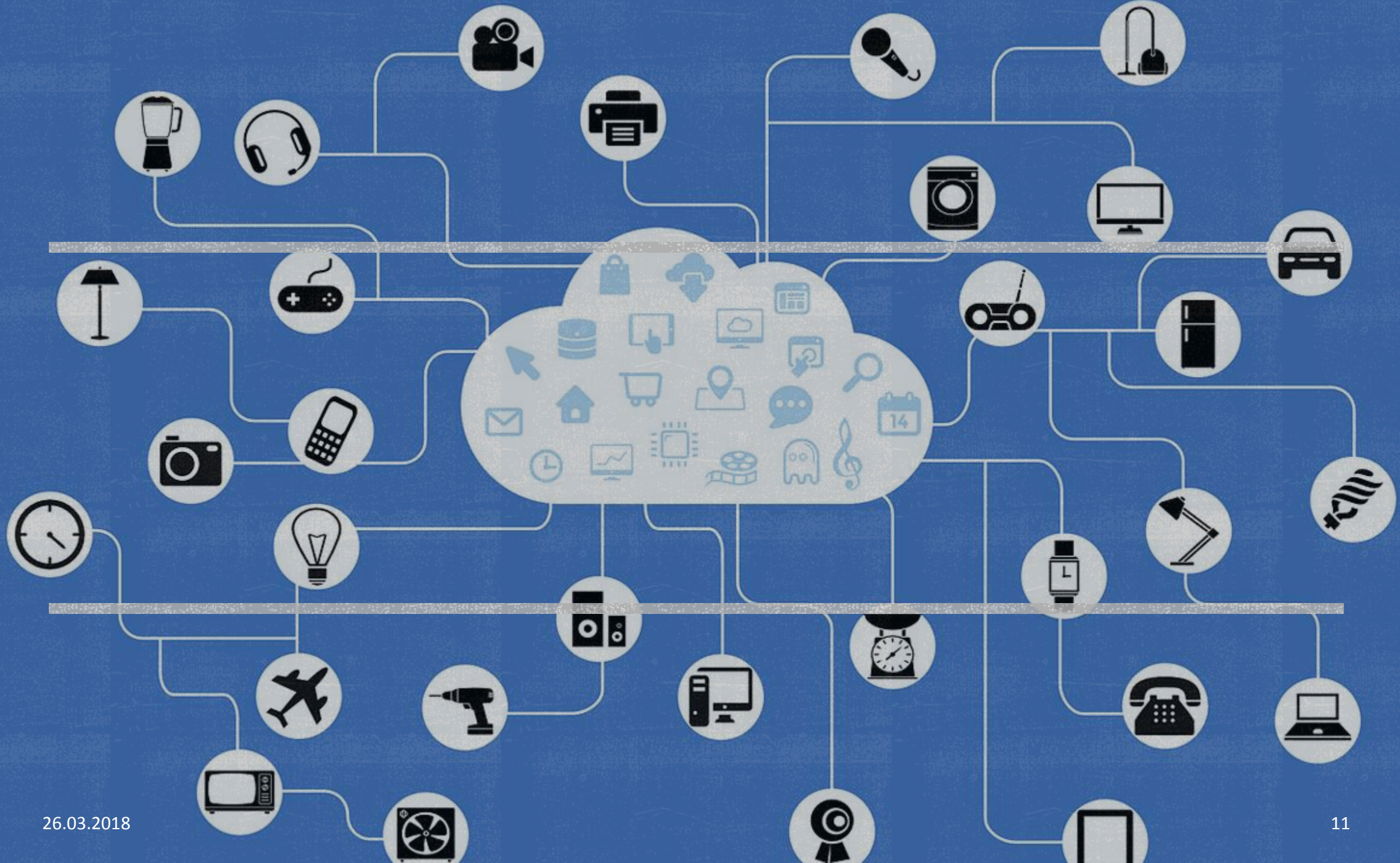
API Example


Hinflug wählen | SWISS
ZRH
JFK
09:50
12:35
JFK Intl. Apt.
LX 16
Durchgeführt von SWISS
Reisedauer: 8h 45m
Economy
ab CHF 500
Business
Noch 2 Sitze zu diesem Preis
ab CHF 4'835
First
ab CHF 8'322
ZRH
JFK
12:55
15:45
JFK Intl. Apt.
LX 14
Durchgeführt von SWISS
Reisedauer: 8h 50m
Economy
Noch 7 Sitze zu diesem Preis
ab CHF 995
Business
ab CHF 4'242
First


Swiss
Database

API Aggregators





API Call

- Asking for information is a API Call.
- Recieving information from an API Call is done through a HTTP request

- https://maps.googleapis.com/maps/api/geocode/json?address=NYC
- (To use it frequently you need to sign up as a developer with your google account)
- (use https when possible | secure transmisson of data)

OOP & Database Applications

Lecture 7: Databases

Ruben Zürcher

Professor Binswanger

Why do we use Databases

- Size
- Ease of Updating
- Accuracy
- Security
- Redundancy
- Importance

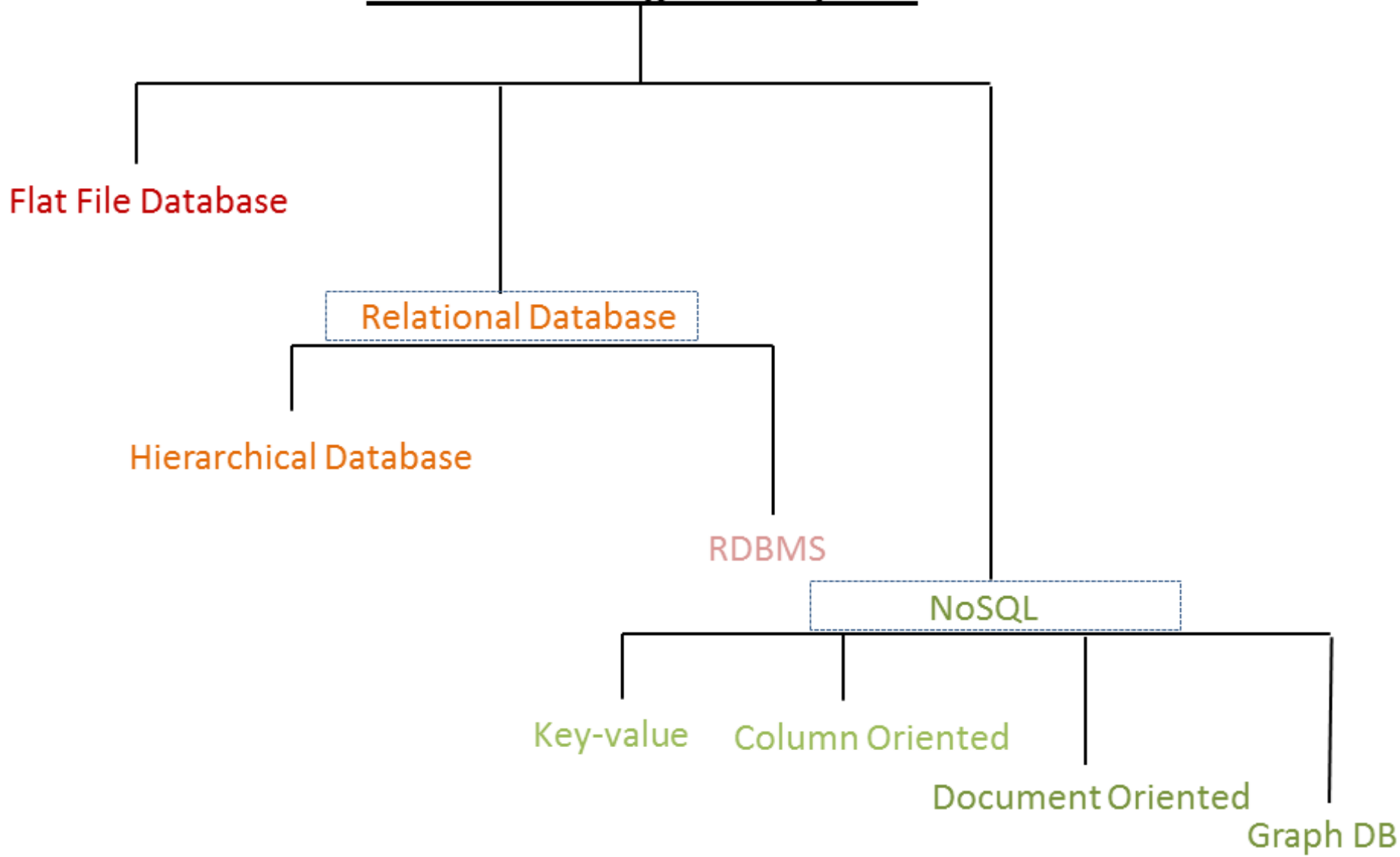
Definition of a Database

- A database is **a data structure** that stores **organized** information.

- A database is a collection of information that is **organized** so that it can be easily accessed, managed and updated.

- A database is a set of data that has a regular **structure** and that is **organized** in such a way that a computer can easily find the desired information.
- Data is a collection of distinct pieces of information, particularly information that has been formatted (i.e., organized) in some specific way for use in analysis or making decisions.

Database Management system



https://www.analyticsvidhya.com/wp-content/uploads/2014/11/DBs-table1.png

Relational Databases (RDBMS)

- Relational databases are made up of a set of tables with data that fits into a predefined category.
- It is "relational" because the values within each table are related to each other. Tables may also be related to other tables. The relational structure makes it possible to run queries across multiple tables at once.
- Data is stored in a structured format in a table, using rows and columns.

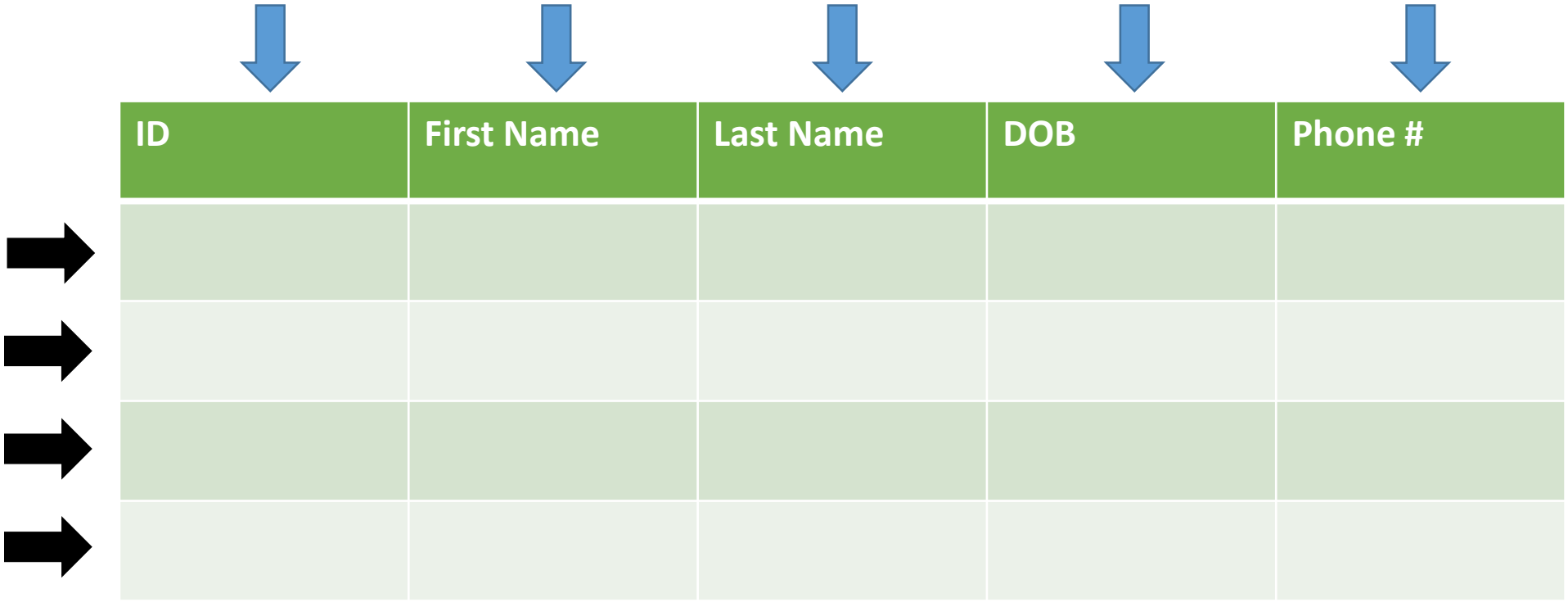
Design of a Table

Fields

maintain information

Records/ Tuples

row entry



ID	First Name	Last Name	DOB	Phone #

Example

Student
Student ID
Student Name
OS

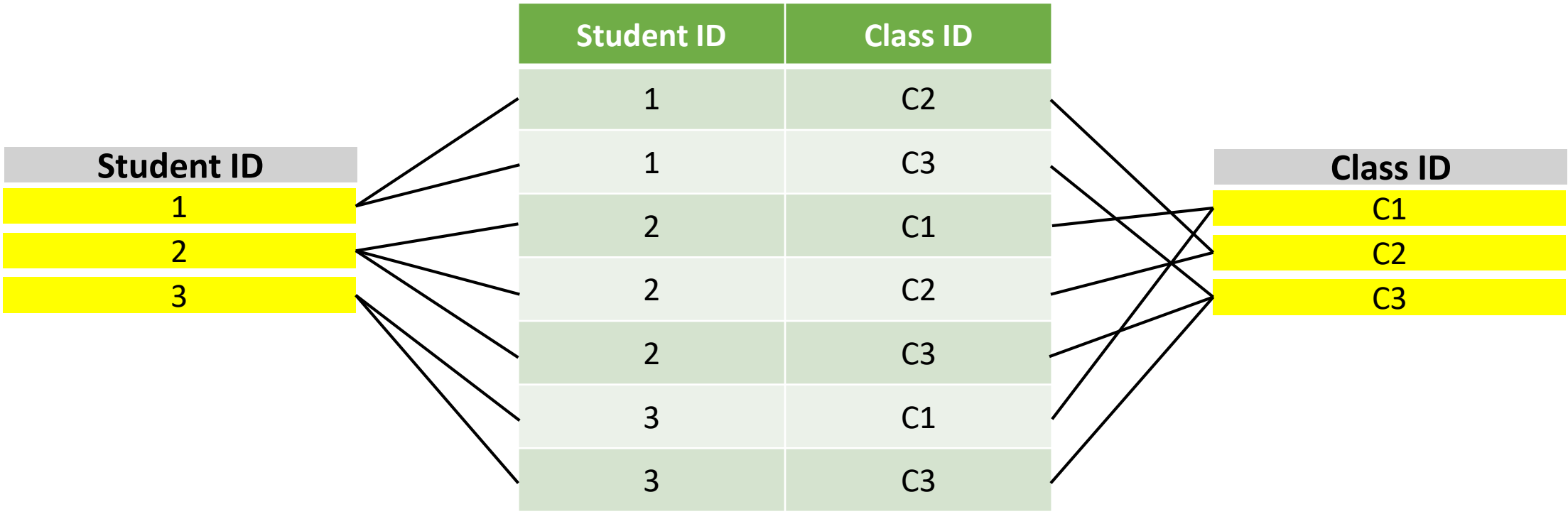
Class
Class ID
Class Name
Instructor

Student ID	Student Name	OS
1	Laura	Windows
2	Laura	OS X
3	Ben	OS X

Class ID	Class Name	Instructor
C1	Rise of Microsoft	Bill Gates
C2	Computer 101	Pacman
C3	Opsec	Sabu

Student ID	Student Name	OS
1	Laura	Windows
2	Laura	OS X
3	Ben	OS X

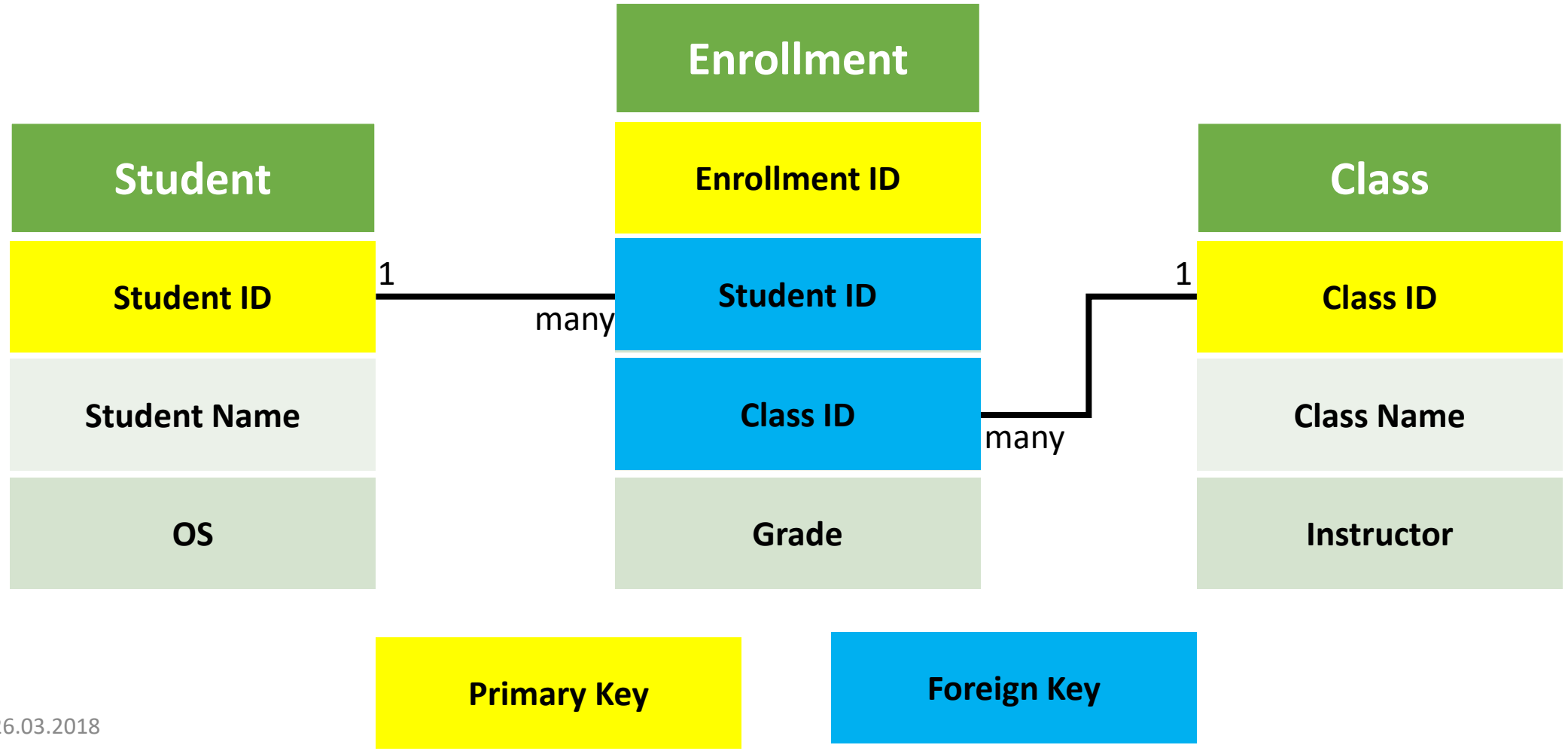
Class ID	Class Name	Instructor
C1	Rise of Microsoft	Bill Gates
C2	Computer 101	Pacman
C3	Opsec	Sabu


Student ID
1
2
3
Student ID
Class ID
1
C2
1
C3
2
C1
2
C2
2
C3
3
C1
3
C3
Class ID
C1
C2
C3

Student
Student ID
Student Name
OS

Enrollment
Student ID
Class ID
Grade

Class
Class ID
Class Name
Instructor


Enrollment
Student
Enrollment ID
Class
Student ID
1
Student ID
1
Class ID
Student Name
Class ID
many
Class Name
many
OS
Grade
Instructor
Primary Key
Foreign Key

Keys

- They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.

- Primary Key:
  - Uniquely identifies each record in a database table.
  - Primary keys must contain UNIQUE values, and cannot contain NULL values.
  - A table can have only one primary key, which may consist of single or multiple fields.

- Foreign Key:
  - A key used to link two tables together.
  - A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

SQL

- Structured Query Language (SQL) is a standard computer language for relational database management and data manipulation. SQL is used to query, insert, update and modify data. Most relational databases support SQL.

- SQL can be split up into two the two categories:
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)

Data Definition Language

- DDL statements are used to define the database structure or schema.

- The four commands we will mostly us are:
  - CREATE - to create objects in the database
  - ALTER - alters the structure of the database
  - DROP - delete objects from the database
  - TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed

Data Manipulation Language

- DML statements are used for managing data within schema objects

- The four commands we will mostly us are:
  - SELECT - retrieve data from the a database
  - INSERT - insert data into a table
  - UPDATE - updates existing data within a table
  - DELETE - deletes all records from a table, the space for the records remain

Task

- Expand upon the Student Database
- Built the World Database from a diagramm and import data
- Reverse Engineer the Sakila Database and import the data and following that export the Database including structure and data

OOP & Database Applications

Lecture 8: Databases

Ruben Zürcher

Professor Binswanger

Normalization

- Normalization is the process of reorganizing data in a database so that it meets two basic requirements:
  1. There is no redundancy of data (all data is stored in only one place).
  2. Data dependencies are logical (all related data items are stored together).
     => Data integrity

- There are three main Types of Normalization:
  1. First normal form (1NF)
  2. Second normal form (2NF)
  3. Third normal form (3NF)

First Normal Form

- Eliminate redundant records in individual tables
- Each cell is to be Single valued
- Entries within a field are of the same type
- Rows are uniquely indentified => add a Unique ID or group colums together into a primary key

Second and Third Normal Form

- 2NF:
  - Table needs to be in the 1NF
  - All attributes (Non-Key-Collums) dependant on the key

- 3NF:
  - Table needs to be in the 2NF
  - All fields can be determined only by the key in the table and no other collums

Normalization Example

Name	Product	Address	Newsletter	Supplier	Producer #	Price
Jordan	iPhone	Langgasse 8	Apple Newsletter	Apple	0900 900 900	900
Michelle	Samsung Galaxy	Marktplatz 4	Samsung Newsletter	Samsung	0100 100 100	850
Tom	iPhone, Samsung A4	Bahnhofstrasse 23	Apple & Samsung Newsletter	Wholesale	0800 800 800	1500
Jordan	Samsung Galaxy	Marktplatz 4	Samsung Newsletter	Samsung	0100 100 100	850

Normalization Example NF 1

ID	Name	Product	Address	Newsletter	Supplier	Producer #	Price
C1	Jordan	iPhone	Langgasse 8	Apple Newsletter	Apple	0900 900 900	900
C2	Michelle	Samsung Galaxy	Marktplatz 4	Samsung Newsletter	Samsung	0100 100 100	850
C3	Tom	iPhone,	Bahnhofstrasse 23	Apple Newsletter	Apple	0900 900 900	900
C3	Tom	Samsung A4	Bahnhofstrasse 23	Samsung Newsletter	Samsung	0100 100 100	600
C4	Jordan	Samsung Galaxy	Marktplatz 4	Samsung Newsletter	Samsung	0100 100 100	850

Normalization Example NF 2

ID	Name	Address	Newsletter
C1	Jordan	Langgasse 8	Apple Newsletter
C2	Michelle	Marktplatz 4	Samsung Newsletter
C3	Tom	Bahnhofstrasse 23	Apple Newsletter
C3	Tom	Bahnhofstrasse 23	Samsung Newsletter
C4	Jordan	Marktplatz 4	Samsung Newsletter

ID	Item
C1	iPhone
C2	Galaxy
C3	iPhone
C3	A4
C4	Galaxy

Item	Supplier	Producer #	Price
iPhone	Apple	0900 900 900	900
Galaxy	Samsung	0100 100 100	850
A4	Samsung	0100 100 100	600

Normalization Example NF 3

ID	Name	Address	Newsletter
C1	Jordan	Langgasse 8	Apple Newsletter
C2	Michelle	Marktplatz 4	Samsung Newsletter
C3	Tom	Bahnhofstrasse 23	Apple Newsletter
C3	Tom	Bahnhofstrasse 23	Samsung Newsletter
C4	Jordan	Marktplatz 4	Samsung Newsletter

ID	Item
C1	iPhone
C2	Galaxy
C3	iPhone
C3	A4
C4	Galaxy

Item	Supplier	Price
iPhone	Apple	900
Galaxy	Samsung	850
A4	Samsung	600

Supplier	Producer #
Apple	0900 900 900
Samsung	0100 100 100

Normalization Example NF 3

ID	Name	Address
C1	Jordan	Langgasse 8
C2	Michelle	Marktplatz 4
C3	Tom	Bahnhofstrasse 23
C4	Jordan	Marktplatz 4

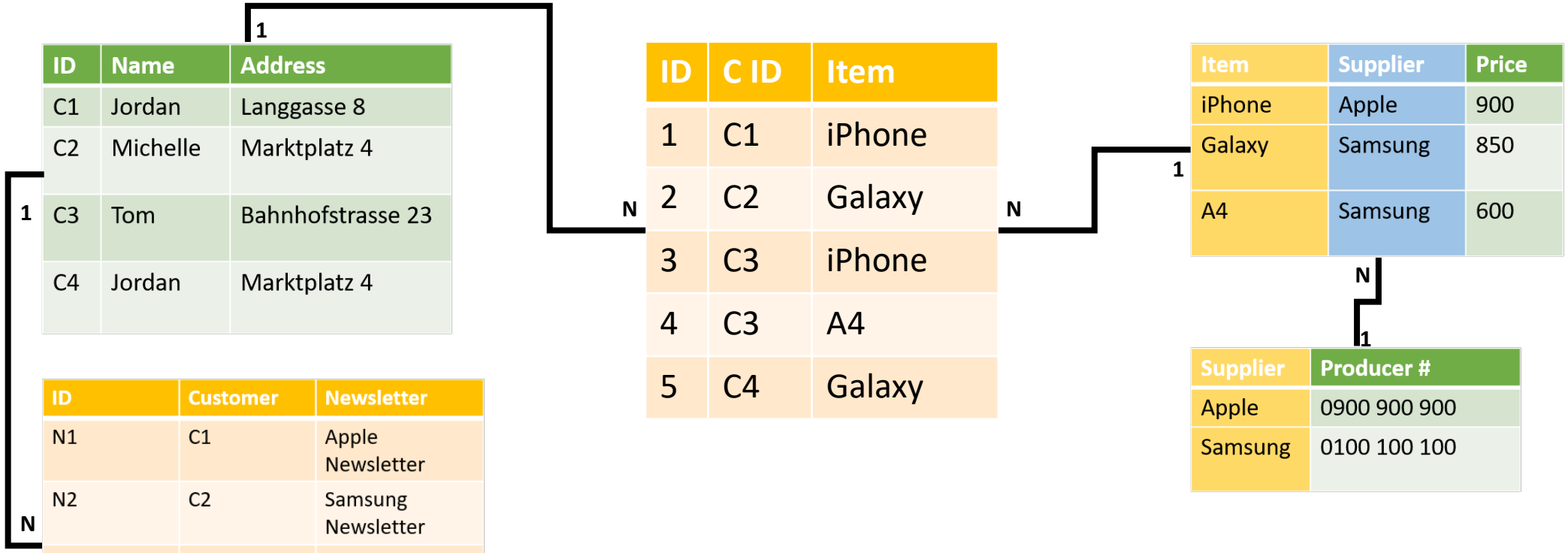
Customer	Newsletter
C1	Apple Newsletter
C2	Samsung Newsletter
C3	Apple Newsletter
C3	Samsung Newsletter
C4	Samsung Newsletter

ID	Item
C1	iPhone
C2	Galaxy
C3	iPhone
C3	A4
C4	Galaxy

Item	Supplier	Price
iPhone	Apple	900
Galaxy	Samsung	850
A4	Samsung	600

Supplier	Producer #
Apple	0900 900 900
Samsung	0100 100 100

Normalization Example NF 3



ID	Name	Address
C1	Jordan	Langgasse 8
C2	Michelle	Marktplatz 4
C3	Tom	Bahnhofstrasse 23
C4	Jordan	Marktplatz 4

ID	C ID	Item
1	C1	iPhone
2	C2	Galaxy
3	C3	iPhone
4	C3	A4
5	C4	Galaxy

Item	Supplier	Price
iPhone	Apple	900
Galaxy	Samsung	850
A4	Samsung	600

Supplier	Producer #
Apple	0900 900 900
Samsung	0100 100 100

ID	Customer	Newsletter
N1	C1	Apple Newsletter
N2	C2	Samsung Newsletter
N3	C3	Apple Newsletter
N4	C3	Samsung Newsletter
N5	C4	Samsung Newsletter

Data Manipulation Language

- SELECT - retrieve data from the a database

- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain

Data Manipulation Language: INSERT

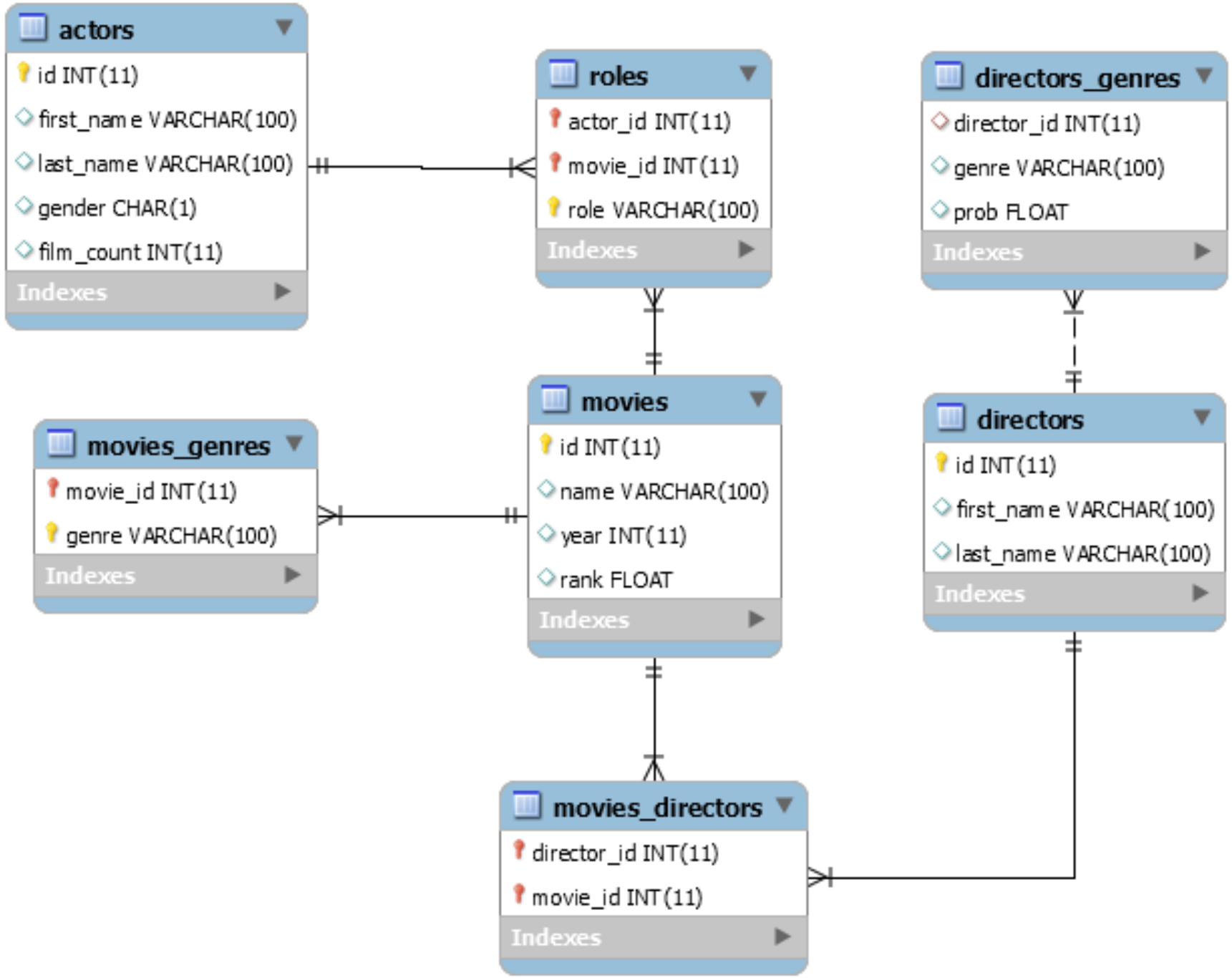
The INSERT command in SQL is used to add records into an (or multiple) existing table.

Syntax 1:

```
INSERT INTO table_name
VALUES (value1, value2, value3,...);
```

Syntax 2:

```
INSERT INTO table_name(column1, column2, column3,...)
VALUES (value1, value2, value3,...);
```


actors
id INT(11)
first_name VARCHAR(100)
last_name VARCHAR(100)
gender CHAR(1)
film_count INT(11)
Indexes
roles
actor_id INT(11)
movie_id INT(11)
role VARCHAR(100)
Indexes
directors_genres
director_id INT(11)
genre VARCHAR(100)
prob FLOAT
Indexes
movies_genres
movie_id INT(11)
genre VARCHAR(100)
Indexes
movies
id INT(11)
name VARCHAR(100)
year INT(11)
rank FLOAT
Indexes
directors
id INT(11)
first_name VARCHAR(100)
last_name VARCHAR(100)
Indexes
movies_directors
director_id INT(11)
movie_id INT(11)
Indexes

INSERT Exampels

- INSERT INTO actors

  VALUES (1,"hans","peter","m",5); <= sequence!!!

- INSERT INTO actors(id,first_name)

  VALUES (2,"erich");

- INSERT INTO actors(id,last_name,first_name)

  VALUES (3,"meier","tom");

Data Manipulation Language: DELETE

The DELETE command is used to remove records from an existing table. It can only delete row(s) and not column(s).

Syntax:

```
DELETE FROM table_name
WHERE some_column=some_value;
```

Note : The WHERE clause specifies which record or records that should be deleted. **If you omit the WHERE clause, all records will be deleted!**

DELETE Examples

- DELETE FROM actors

  WHERE id = 1;

- DELETE FROM actors

  WHERE last_name = "Meier";

- DELETE FROM actors

  WHERE film_count <= 5;

Data Manipulation Language: UPDATE

The UPDATE command can be used to modify the records in the existing table, either in bulk or individually.

Syntax:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value;
```

NOTE : As I mention above that update can modify either a single record or a bulk record. So the where clause in the update statement helps us to do that. If we use a where clause then whenever the condition in true the record will get updates and **if don't use the where clause then all the records are appended**.

UPDATE Examples

- UPDATE actors
  set film_count = 5;

- UPDATE actors
  set film_count = 4 where id=2;

- UPDATE actors
  set film_count = film_count * 3
  where id = 4;

Data Manipulation Language: SELECT

The Select command can be used to retrieve data from a table, either the complete table data, or partial by specifying conditions using the WHERE clause.

Syntax:

```
Select column1, column2
FROM table_name
[WHERE some_column=some_value];
```

NOTE : [] mean optional information to query more specifically, with the keyword AS we can give a field a new name, DISTINCT eliminates duplicates (when selecting multiple fields only if it is duplicate in both fields in the same row)

SELECT Examples

- Select id, first_name, last_name
  from actors;
- SELECT *
  FROM imdb_small.actors;
- SELECT distinct last_name, first_name
  from actors;
- SELECT *
  from actors
  WHERE film_count < 5 or film_count > 10;

Sorting the SELECT Command

- It is possible to sort the result from a SELECT command with ORDER BY

Syntax:

```
Select column1, column2
FROM table_name
[WHERE some_column=some_value]
ORDER BY column_name ASC (or DESC)
```

Aggregating the SELECT Command

- We can use specific aggregation function on fields to get specific results.

Syntax:

```
Select aggregation(column1), aggregation(column2)
FROM table_name
[WHERE some_column=some_value];
```

The aggragation function which are most used are: SUM, COUNT, MAX, MIN, AVG

Advanced SELECT Examples

- SELECT *
  FROM actors
  order by id DESC;

- SELECT COUNT(distinct last_name)
  from actors;
- SELECT COUNT(gender)
  from actors
  WHERE gender = “f";

JOINS

- A join is command to combine data from two sets of data (i.e. two tables). It creates a set that can be saved as a table or used as it is.

- There are 5 typical Joins:
  - Inner
  - Left Outer
  - Right Outer
  - Full Outer
  - Cross

CROSS JOIN

- CROSS JOIN returns the Cartesian product of rows from tables in the join. Thus it combines every row of table a with every row of table b.

Syntax 1:

```
Select *
From table1 cross join table 2;
```

Syntax 2:

```
Select *
From table1,table2;
```

CROSS JOIN Examples

- Select *

  from directors,directors_genres;

- Select *

  from directors

  cross join directors_genres;

INNER JOIN

- An inner join selects records from two tables that hold matching values. Records that do not hold matching or common values are excluded from the output.

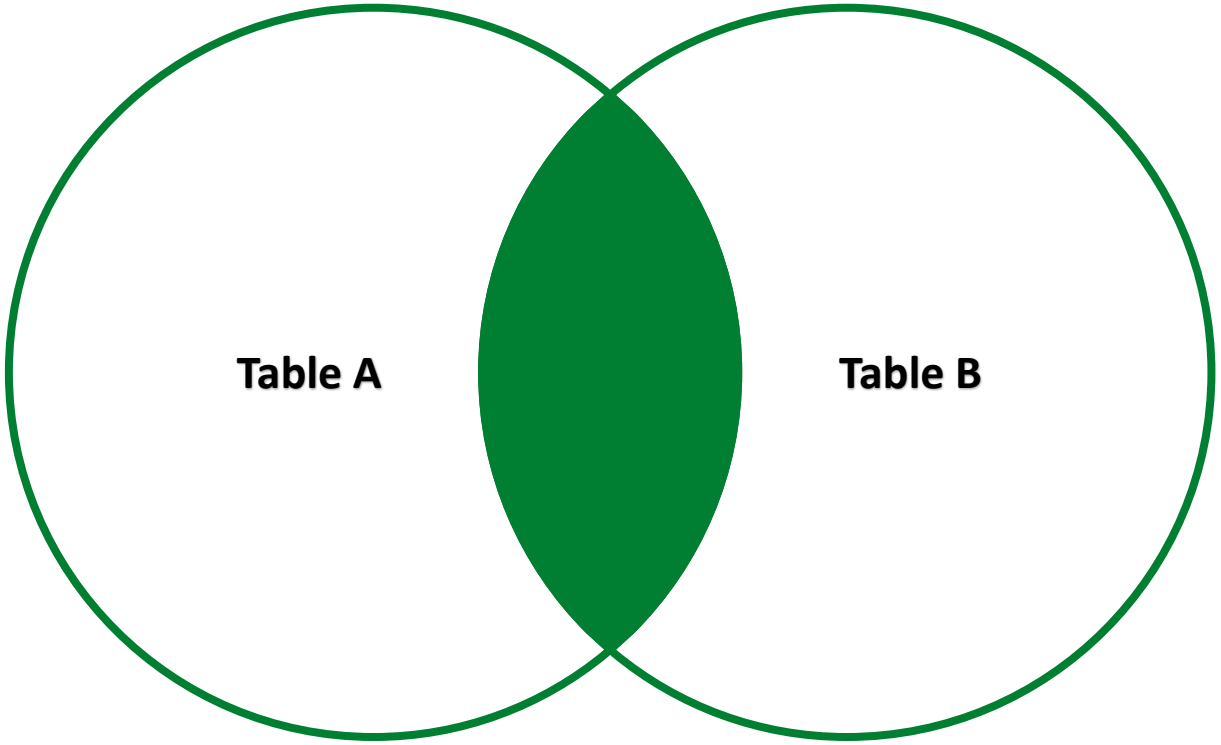
Syntax 1:

```
Select*
from table1 join table2
on table1.column_name =table2.column_name;
```

Syntax 2:

```
Select *
from table1,table2
where table1.column_name = table2.column_name;
```

INNER JOIN Illustration



INNER JOIN Examples

- Select*

  from directors join directors_genres

  on directors.id =directors_genres.director_id;

- Select *

  from directors,directors_genres

  where directors.id =directors_genres.director_id;

Tasks

- Database Exercises 1 and 2 on Dropbox (1.16 and from ~2.10 the exercises get more difficult => you may need to google the GROUP BY command and also what queries within queris are)

- For those who want to learn more i can recommend the following:
  - https://www.w3resource.com/sql-exercises/sql-retrieve-from-table.php
  - https://www.w3resource.com/sql-exercises/sql-aggregate-functions.php
  - https://www.w3resource.com/sql-exercises/sql-exercises-quering-on-multiple-table.php
  - https://www.w3resource.com/sql-exercises/sql-joins-exercises.php

OOP & Database Applications

Lecture 9: Databases in Python

Ruben Zürcher

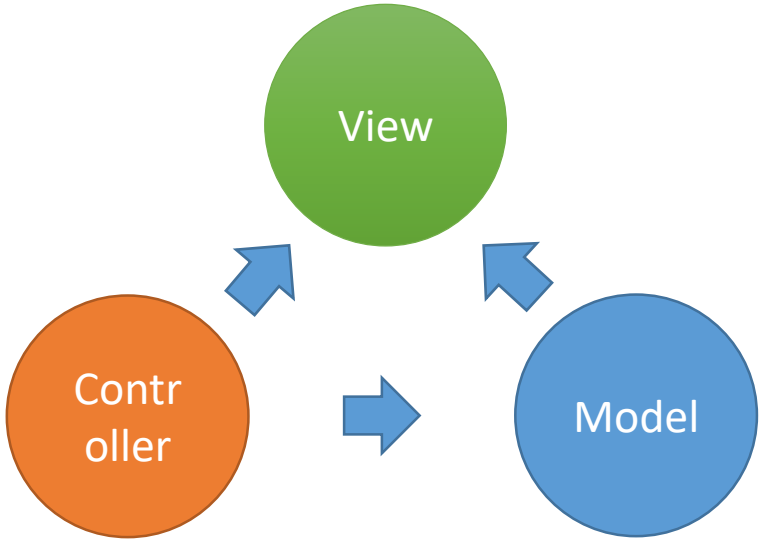
Professor Binswanger

Agenda

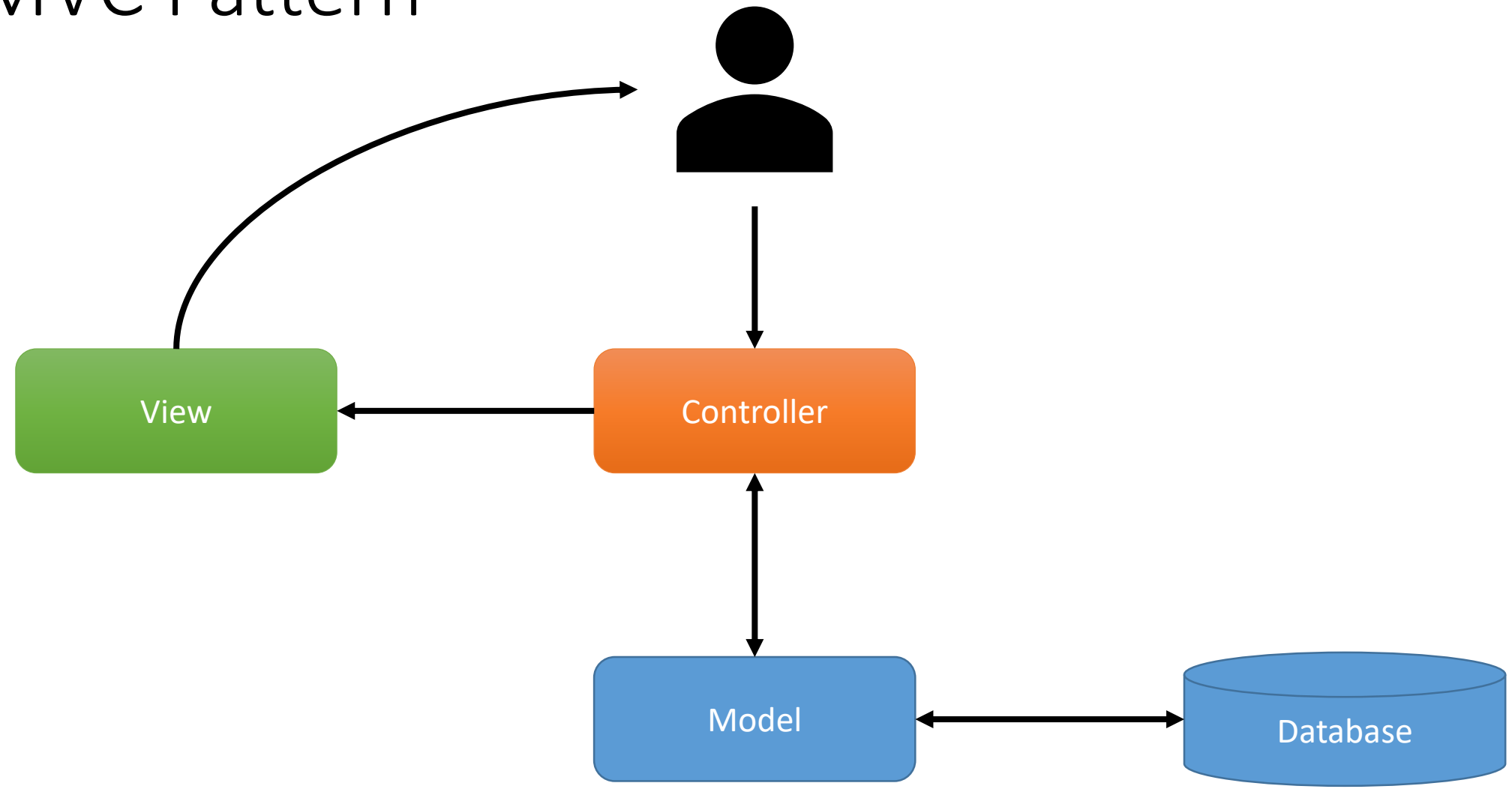
1) MVC
2) DAO
3) Database Connection

MVC – Model View Controller

- Software Architectural Design Pattern
- One of the most frequently used patterns
- Seperates appliaction functionality
- Promotes organized and structured programming



MVC Pattern



Model

- Interactions with the database(Insert, Select, Update, Delete)
- Processing data from or to the database
- Communicates (only) with the Controller

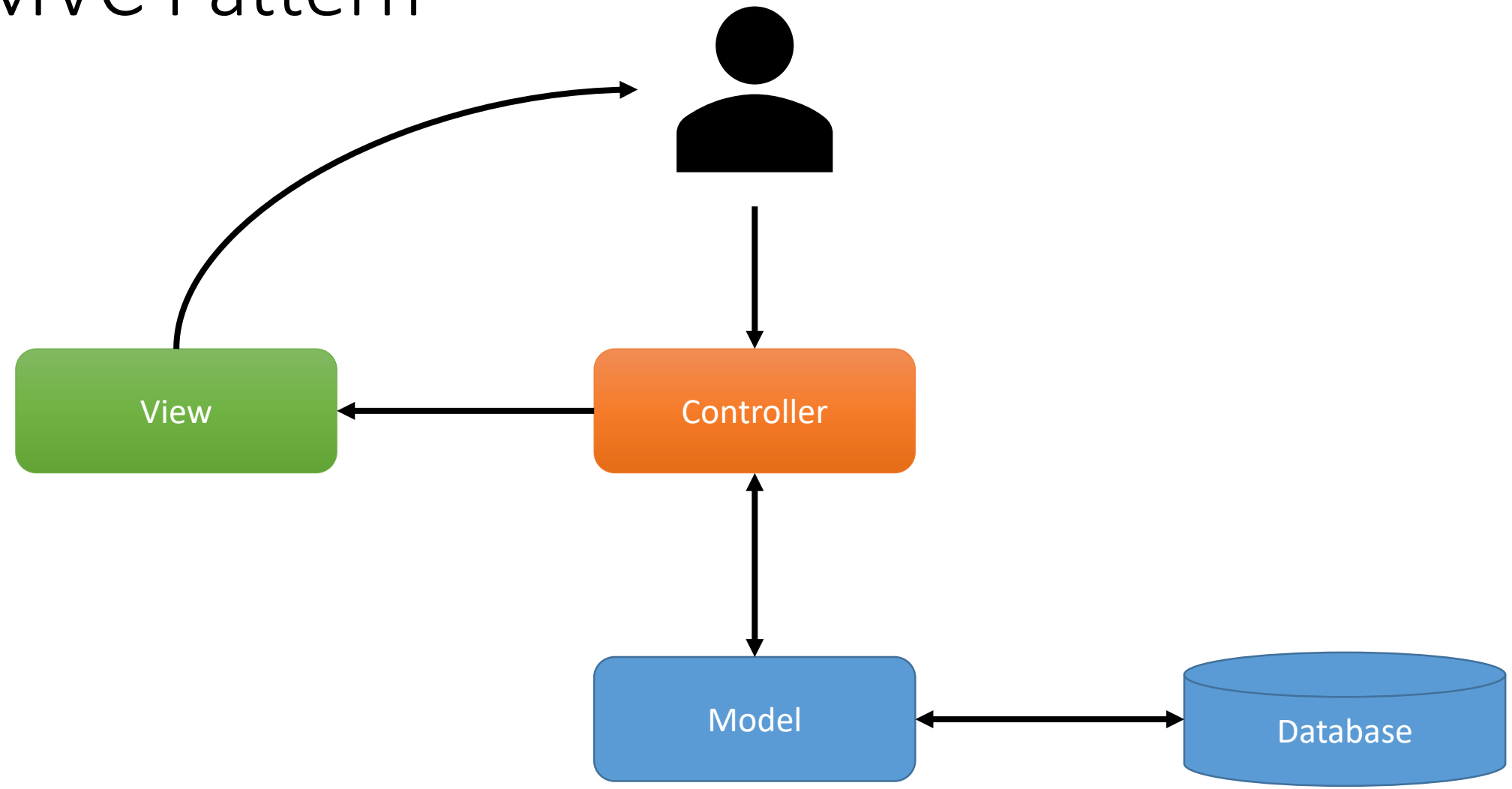
View

- The only thing the End-Users sees
- UI
- Communicates only with the Controller
- Can except dynamic values from the Controller
- No business logic

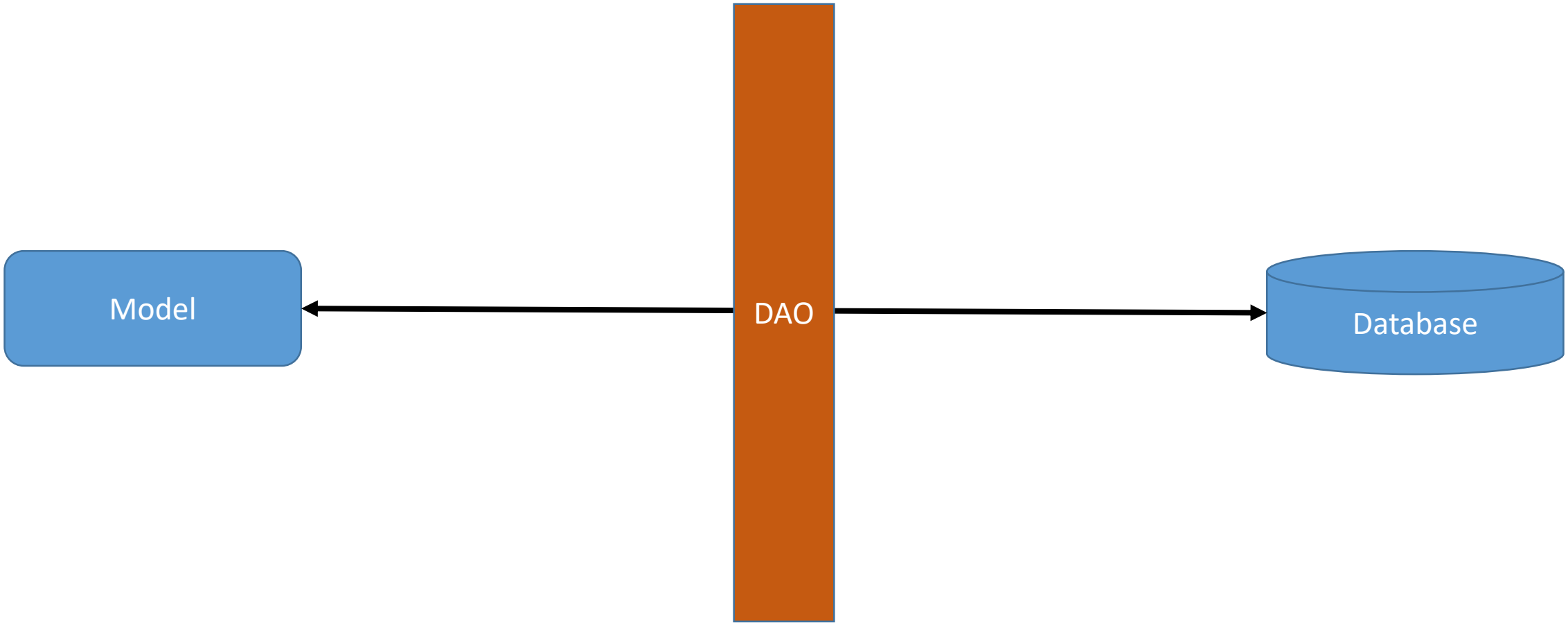
Controller

- The Middle-Man (brain)
- Recieves input from the user (through the view)
- Gets data from the model
- Passes data to the View
- Part of the business logic
- (Can recieve info from the database)

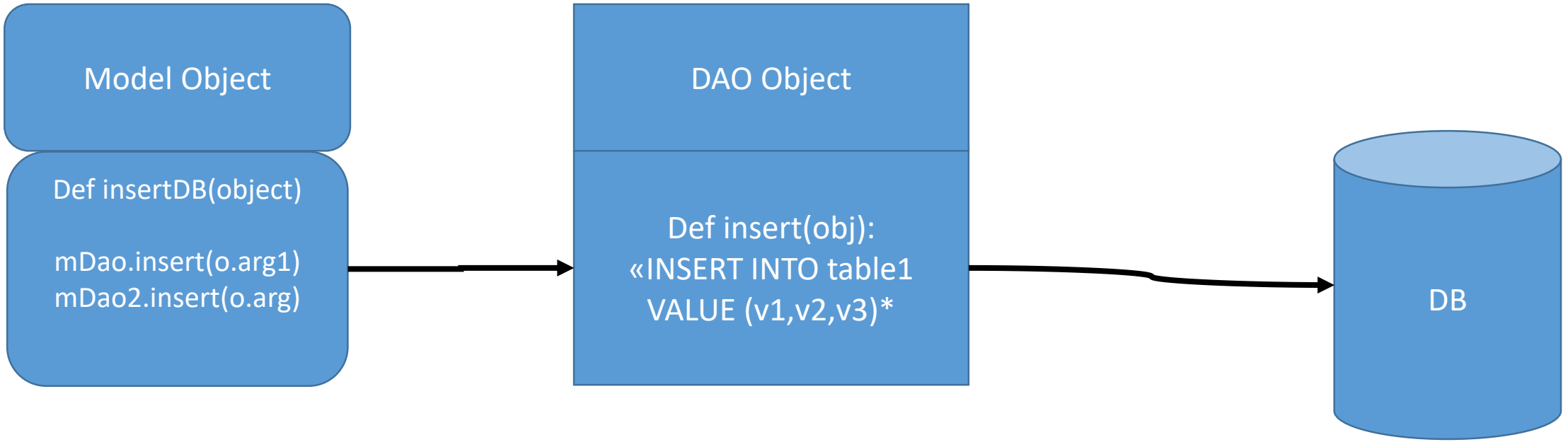
MVC Pattern



DAO/Persistence Layer



DAO/ Persistence Layer



DAO/ Persistence Layer

- Data Access Object
- Separates a data resource's client from its data access mechanism
- Implements the commands to the Database
- Reusability

- (if the data source changes, from MySQL to Microsoft SQL, you will not have to rewrite all the code but only the DAO Layer)

Add a Database User for OOP

- `CREATE USER 'oop-fs18'@'localhost' IDENTIFIED BY 'OOP-fs18';`

- `GRANT ALL PRIVILEGES ON * . * TO 'oop-fs18'@'localhost';`

Connect to Database

```
Import pymysql

connection = pymysql.connect(host='localhost',
                             user='oop-fs18',
                             password='OOP-fs18',
                             db='World’,
                             charset='utf8mb4’,
 cursorclass=pymysql.cursors.DictCursor)
```

Prepared Statement

- To protect against SQL Injection
- Using parameters to dynamically enter and change values

1. Define your query
2. Define the variables for your query
3. Execute the query

Prepared Statement in Python

```
try:
    with connection.cursor() as cursor:
        sql = "INSERT INTO `table_name`
            (`field1`, `field2`,`field3`)
            VALUES (%s, %s, %s)"
        cursor.execute(sql, [obj.field1, obj.field2,
                             obj.field3])

finally:
        connection.commit()
```

Commit

- The COMMIT command confirms the executed statements to the database. Without it, the data does not actually get transfered into the database.

- There is no clear time when to commit transactions. Best principle is, to commit when all the data which belongs to one object as been temporarily been executed into the database.

Questions from last week

Tasks

- Complete the World Database Persistence Layer
- Your Projects