University of Windsor

LECTURE 4 – DESIGN PATTERNS

Master of Applied Computing

COMP-8117 : Advanced Software Engineering Topics

Dr. Aznam Yacoub – aznam.yacoub@uwindsor.ca
School of Computer Science

SCHEDULE

- Introduction
- Design Patterns
- Architectural Patterns
- Conclusion

REMINDER

- During the last lecture : What is a specification ?
- Next step : What is a design ?

University of Windsor

DESIGN

- Specifications = Describes the problem, provides a conceptual model of the abstract solution
- Gives a conceptual model of the computerized solution
  - Translate the specifications and incorporates technical constraints
  - Create computations to be programmed on a computer

DESIGN

- Provides a software architecture and a hardware architecture
- Describes modules, interfaces, data and functions ← Why the boundaries between specifications and design is not always clear.
- Specifications = Point of view of the users (external view)
- Design = Point of view of the system (internal view)

DESIGN

- Another picture:
  - Specification = What the MD sees of the human body (functions)
  - Design = What the biologists sees of the human body (the internal structure and organs)

DESIGN

- Internal description of the solution (but still conceptual)
- Example :
  - Specifications: My system computes a distance between two points using the euclidian distance = sqrt((A.x-B.x)²+(A.y-B.y)²)
  - Design:
    - Class Point => x, y : float
    - Function Distance => float distance(Point A)

DESIGN

- Example :
  - Function Distance => float distance(Point A)
    - Put A.x in the registry R1
    - Put A.y in the registry R2
    - Compute $(R1-x)^2$, put the result in R3
    - Compute $(R2-y)^2$, put the result in R4
    - Compute R3 + R4, put the result in R1
    - Compute sqrt(R1), put the result in R2
    - Return R2

DESIGN

- Design is not just provide class diagrams !
- Design isn’t computer instruction but computer concepts.
- Two dimensions:
  - Structural
  - Behavioural

EXAMPLES

- Structural View of data:
  - Specification : A is a number
  - High-Level Design : A is a int
  - Low-Level Design : A is a 32 bits int
  - Programming : char A[4]; // char = 1 byte; char[4] = 32 bits

EXAMPLES

- Structural View of data:
  - Describe completely the nature of data
  - Is always independant of the real implementation (but not too far => A low-level design may use the types available in your implementation language)

DESIGN

- Design strategies depend on a design paradigm:
  - Functional : relationship between functions
  - Object-Oriented : relationsip between objects
- The design paradigm may differ from the specification paradigm, but usually they are congruent
- The design paradigm is the paradigm used in programming even if the implementation language is not designed for this paradigm

DESIGN

- Example: If your design is an OO design, the C developer will have to find a workaround to represent OO concepts in C.

- Best practice: try to make your design with the paradigm of the technology.
  - If the implementation language is functional, prefer a functional paradigm.
  - If the implementation language is OO, prefer OO paradigm.

DESIGN

- UML is not design => UML is a modelling language helping to formalize relationship. Knowing or doing UML don’t teach you how to think about objects
- Object-oriented is not the only approach to do design but it’s the well-used paradigm.

- Learn design = Learn thinking and mind-education

DESIGN

- Design = Iterative process. Difficulty is : « When should I stop ? »
- High-level design = Describe the structure by abstracting technologies. Close to the specification.
- Low-level design = Close to the implementation.

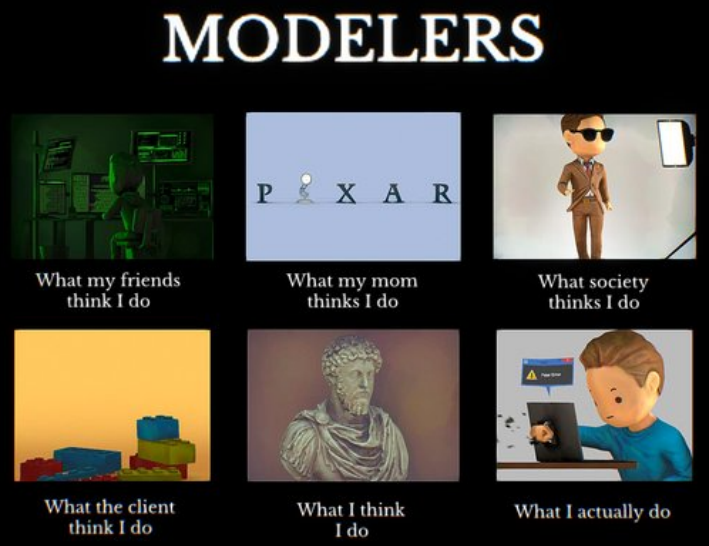
DESIGN

- Inputs = Specifications + Scenarios + Constraints + Domain Model

- Output = Models
  - Give these model to the developer and they start coding immediately software itself or test (test-driven development)
  - Produce different other specialized models

DESIGN

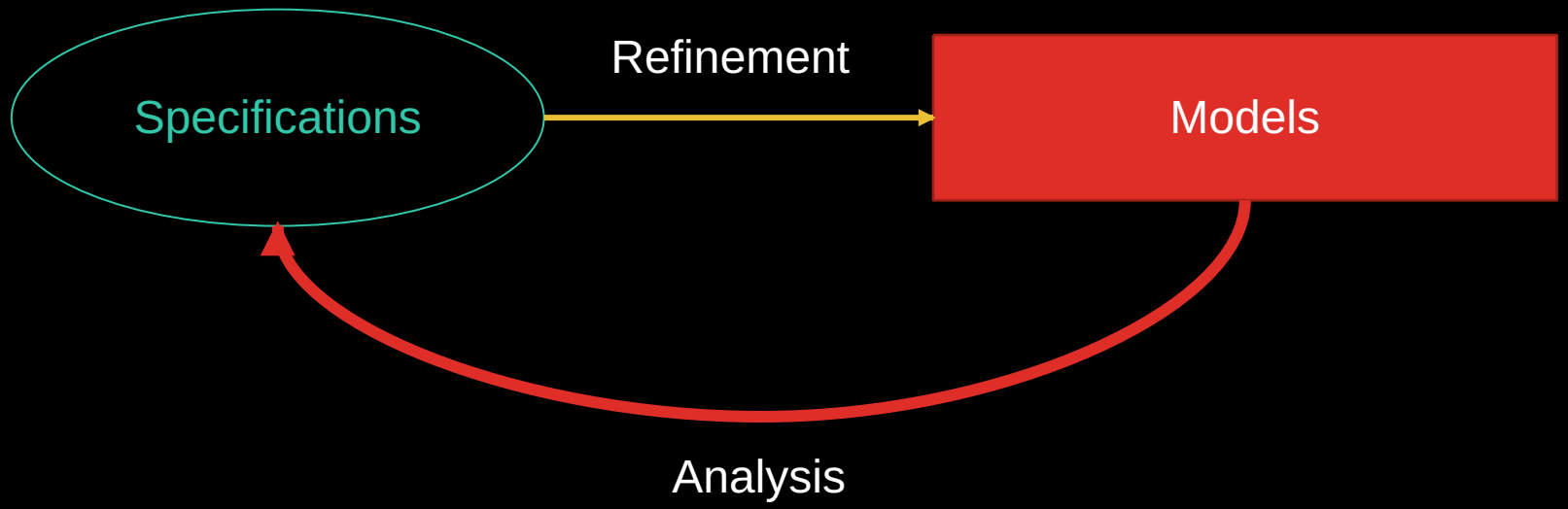
- What you have to keep in mind:
  - A program is a model. [Yacoub et al, 2016]
  - A model is always obtained from another mode [Yacoub, 2016]
  - Models describes a same system from differen points of view = paradigm = way of thinking
  - Models are specifications of other models.



University of Windsor

DESIGN

- Thinking/Design process (Cognitive Science):



DESIGN

- Be careful : Design describe the solution, not a methodology of implementation.

- Design = Art and not Science (Bertrand Meyer)
  - I can provide a scientific approach of modelling, but I can’t make you an artist
  - It’s a creative process, quality of design depends on imagination of the designer and its experience (Calvez)

ARCHITECTURES

- *Architecture* is a result of a set of design.
- Software Architecture describes the general structure of a software => gives the relationships between *internal* components of this software.
- Hardware Architecture describes the general structure of hardware => gives the relationships between external components (example : devices on a distributed network).

OBJECT-ORIENTED DESIGN

- In OOD, the most important aspect = encapsulation.

- Different popular ways of thinking (how to build your classes):
  - Responsibility-driven design (Wirfs-Brock et Wilkerson) : focus on the objects and contract => Objects are responsibles of actions and informations they share. We talk about behavioural abstractions.

  - Data-driven design : Each class is responsible of data it holds. *Keep code and data together.*

OBJECT-ORIENTED DESIGN

- In RDD, think components in term of:
  - Responsibilities = Abstraction of what they do, contract, obligation
  - Roles
  - Collaborations = Methods either act alone or together with other methods and objects.

OBJECT-ORIENTED DESIGN

- Two types of responsibilities:
  - Doing = Create something, do something, control something
  - Knowing = know about data, objects, etc.
- A responsibility is not a method, it's an abstraction. In its simplest form, it's a method, but it can be a set of objects.
- Example: You can have a method *getTotal* whose the responsibility is to compute a total value. But some other classes are responsible to send a subtotal thanks to *getSubtotal*.

OBJECT-ORIENTED DESIGN

- So for example, this is RDD.

```
function getTotal() : Real {
    foreach(i : Item) {
        total += i.getSubtotal();
    }
    return total;
}
```

- Funniest thing: RDD is obvious and natural way of thinking OOD when you don’t think about it. It becomes harder when you start asking what you should put in classes.

PATTERNS

- To help designers, we use *Patterns* or *Principles*.
- Pattern = Best practice, best reusable solutions/schemes to answer specific and repetitive design problems.
- Different kinds of patterns:
  - Architectural patterns (Decompose a system in subsystems)
  - Design patterns (Decompose a component in subcomponents)
  - Idioms (Reusable snippets of code in a specific programming language)
  - Templates (Skeleton ready to be instantiated, extended or adapted)

PATTERNS

- Patterns are idiomatic solutions build by experienced developers.
- A pattern has a name, a description of the problem, a description of the solution, presents the context of use and adaptation, forces and trade-offs
- Best practice : Use patterns each time it’s possible.
- Bad practice : Use patterns everywhere and for everything => Lead to overarchitecture or unbalanced and unefficient code.

PATTERNS

- Patterns generally lead to:
  - More genericity, More adaptability
  - Less productive code, Less maintainable architecture
- You have to analyze and always find the good tradeoff between genericity and specialization.

GRASP

- General Responsibility Assignment Software Patterns (Principles)
- Different of SOLID principles
- 9 patterns to guide RDD and assign responsibilities
- Methodical, Rational, Explainable

GRASP

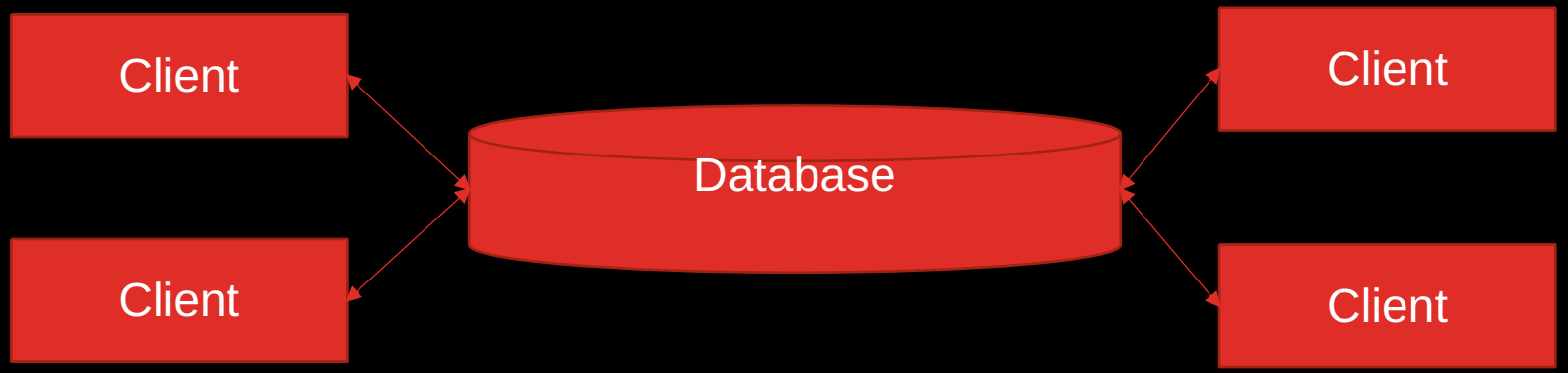
- 9 patterns to guide RDD and assign responsibilities
  - Creator
  - Controller
  - Pure Fabrication
  - Information Expert
  - High Cohesion
  - Indirection
  - Low Coupling
  - Polymorphism
  - Protected Variations
- See the Danya Rao's Lecture for more details about GRASP.

GOF DESIGN PATTERNS

- Gang of Four = Gamma, Helm, Johnson, Vlissides
- 23 Design Patterns for OOD
- See the other lecture for more details about GoF

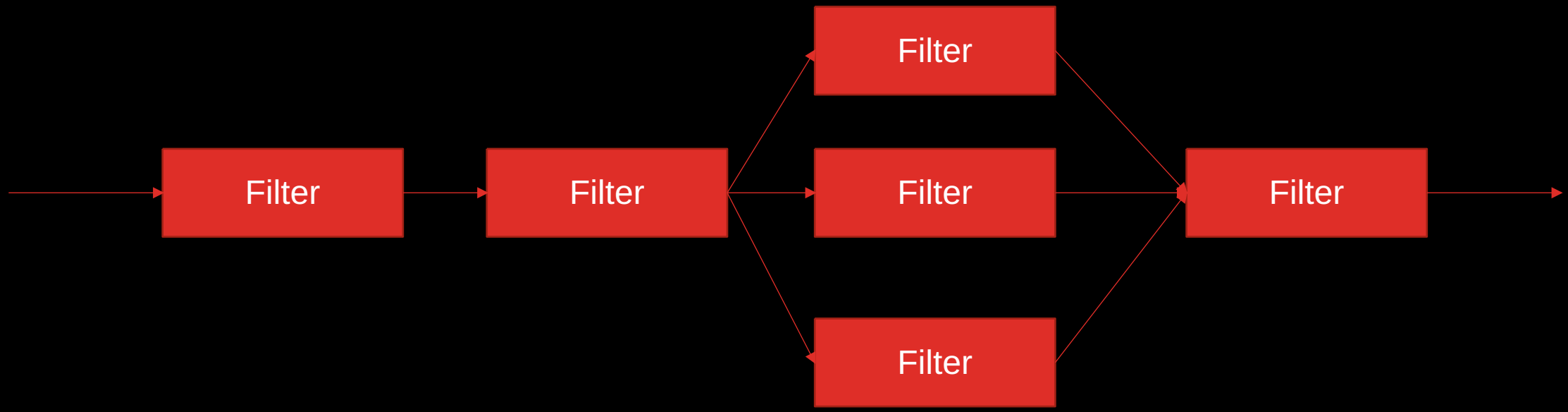
ARCHITECTURE

- Different kind of architecture patterns:
  - Data-driven / Date-centered



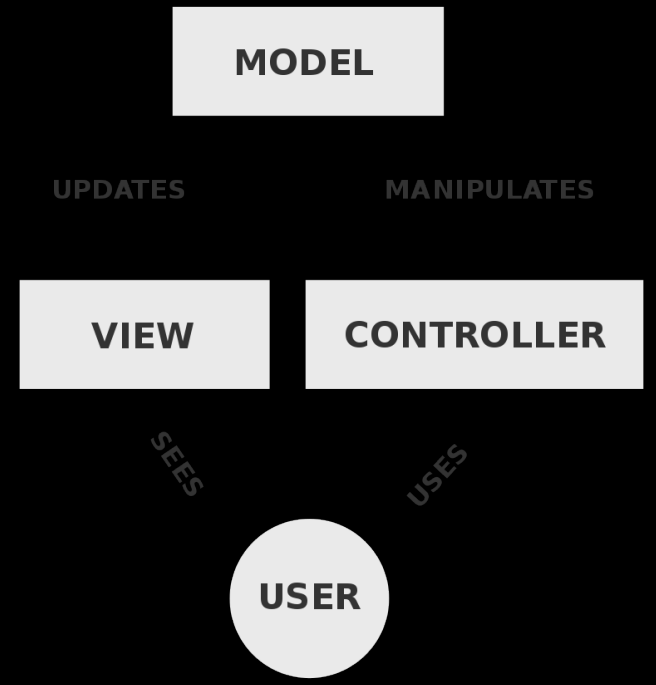
ARCHITECTURE

- Different kind of architecture patterns:
  - Dataflow:



ARCHITECTURE

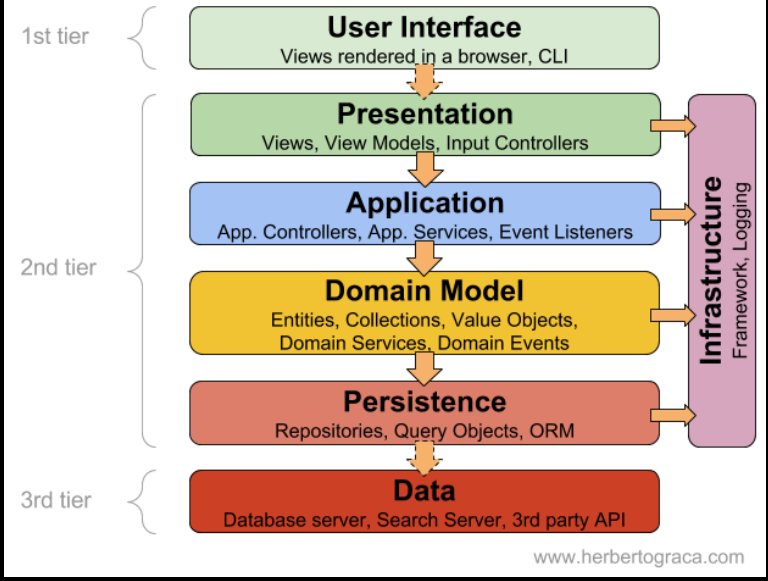
- Different kind of architecture patterns:
  - Call and return (Functional)
  - Event-Driven
  - Model-View-Controller
  - Model-View-Viewmodel
  - Model-View Présenter
  - …

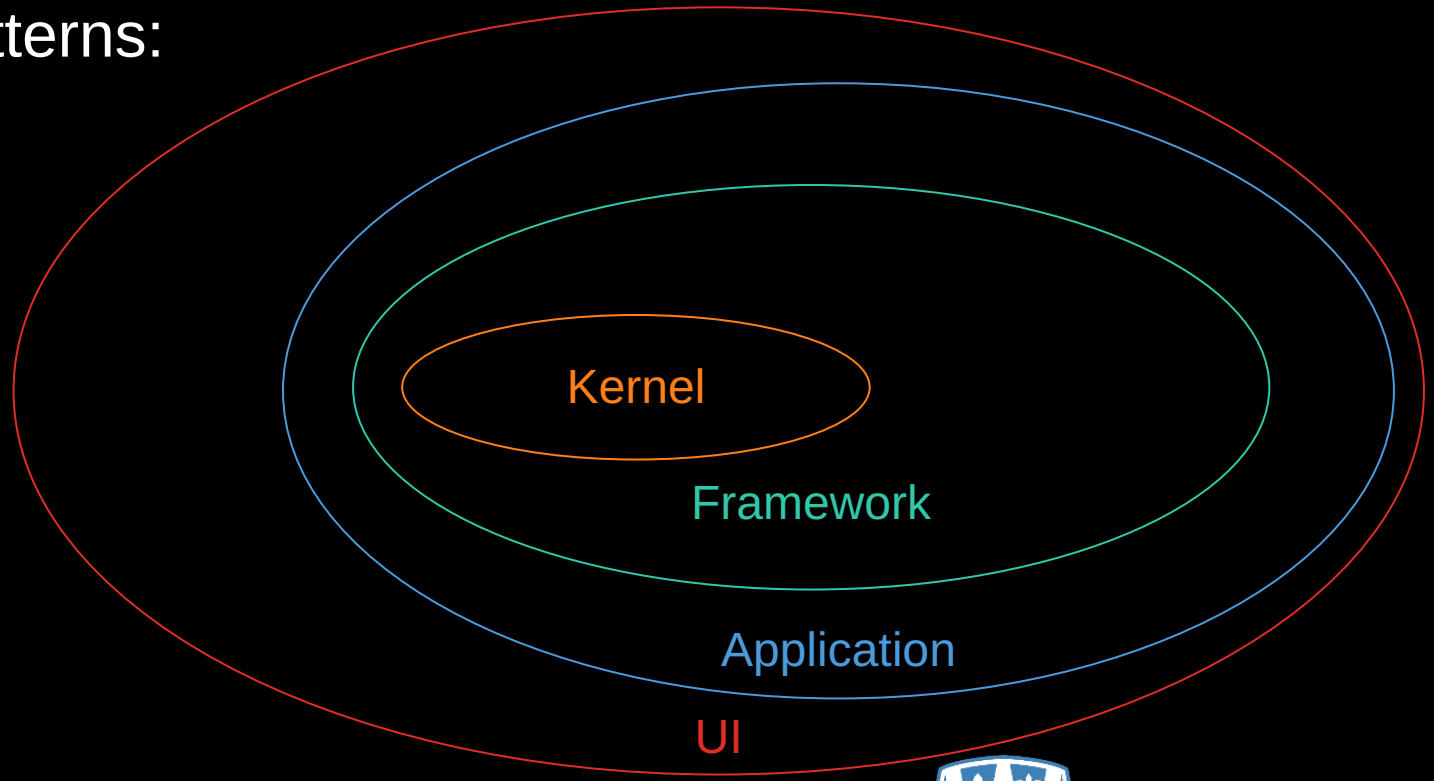


University of Windsor

ARCHITECTURE

- Different kind of architecture patterns:
  - Client / Server
  - Layered Architecture





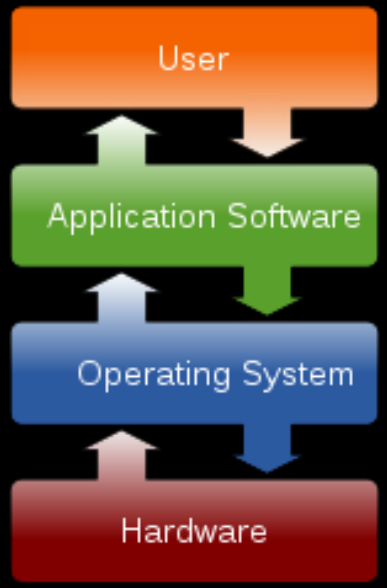
University of Windsor

ARCHITECTURE

- Different kind of architecture patterns:
  - Front-end / Back-end : Generally architecture of WebApp and Website
  - In software engineering, this separation is vague…
    - Front-end = presentation layer
    - Back-end = data access layer

ARCHITECTURE

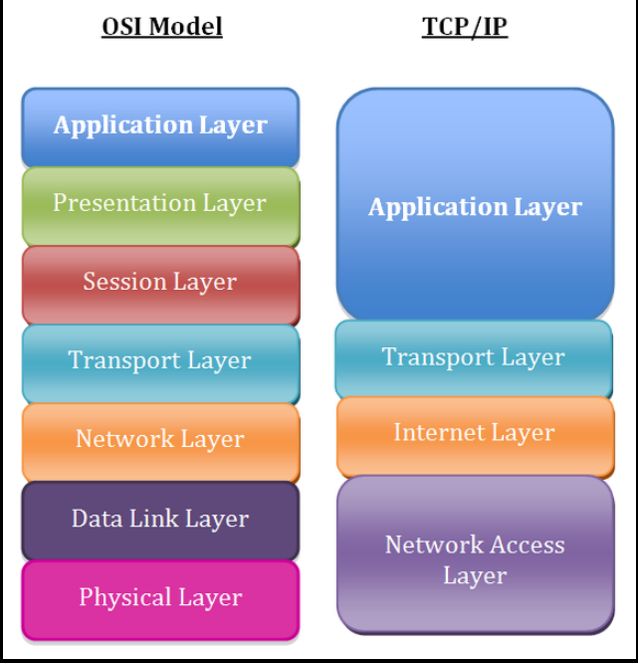
- Different kind of architecture patterns:
  - Front-end / Back-end / Full-stack ?



Where is the front-end, where is the back-end ?

ARCHITECTURE

- Different kind of architecture patterns:
  - Front-end / Back-end / Full-stack ?

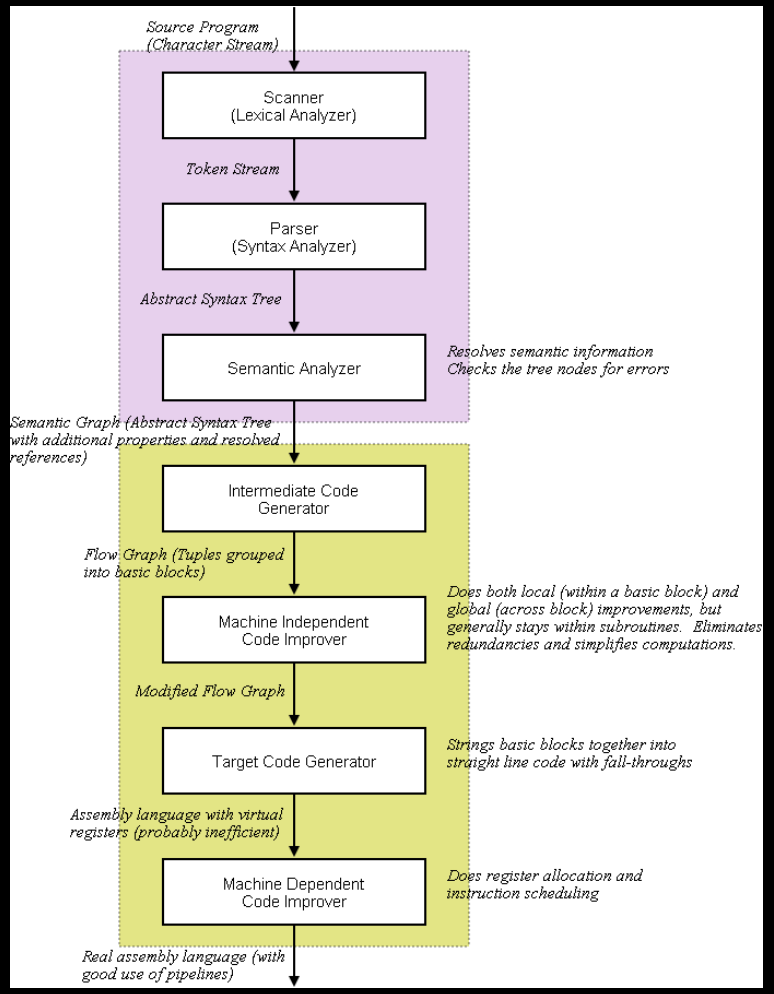


Where is the front-end, where is the back-end ?

ARCHITECTURE

- Different kind of architecture patterns:
  - Front-end / Back-end / Full-stack ?

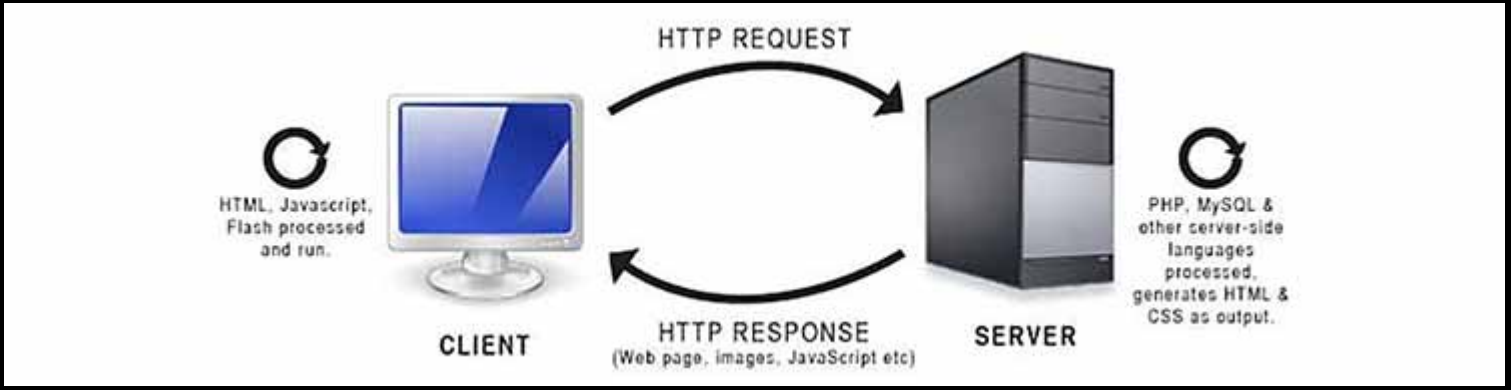
Where is the front-end, where is the back-end ?



University of Windsor

ARCHITECTURE

- Different kind of architecture patterns:
  - Front-end / Back-end / Full-stack ?



Where is the front-end, where is the back-end ?

ARCHITECTURE

- Hardware Architecture :
  - 1-Tier = Centralized
  - 2-Tier = Client/Server with Database on the server
  - 3-Tier = Client presents, Logical thread handled by an application server, database on another server
  - N-Tier = Layered Architecture

CONCLUSION

- Design is more complex than specifications
- Depending on the level of abstraction, you need more or less technical knowledge
- Iterative Process
- Design is a way of thinking
- Use Patterns to understand how to solve problems

REFERENCES

This lecture is based on:

- COMP-8117 (Winter 2020) – Dr. Ziad Kobti
- Software Engineering (Fall 2020) – Dr. Amine Hamri, Dr. Aznam Yacoub
- Software Engineering – Ian Sommerville

