DCC / ICEx / UFMG

# Architectural Patterns: From Mud to Structure

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

## From Mud to Structure

- Layered Architecture
  - It helps to structure applications that can be decomposed into group of tasks
- Pipes and Filters
  - It provides a structure for systems that process a stream of data
- Blackboard
  - It is useful for problems for which no deterministic solution is known

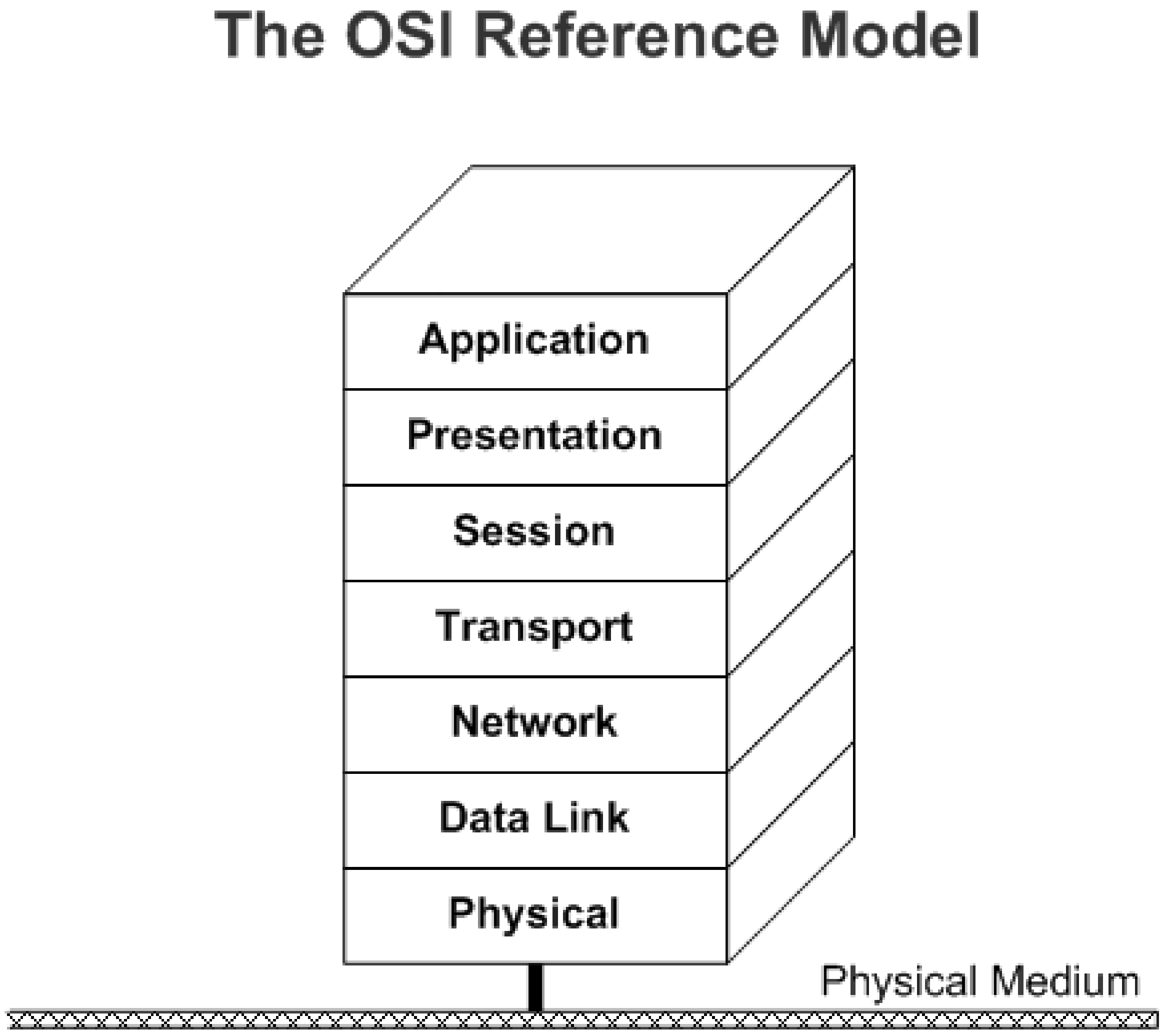
## Layers

## Layers Architectural Pattern

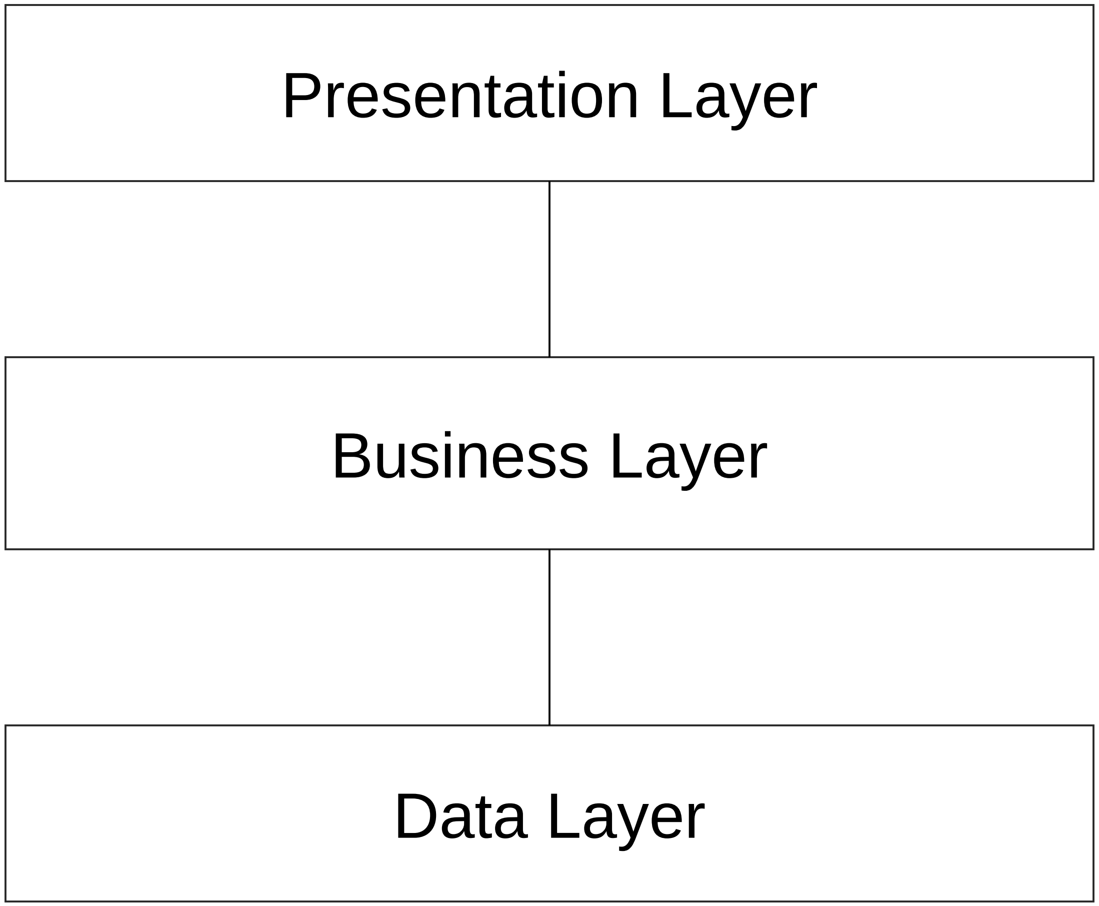
- It organizes the system as groups of tasks
  - Each group of task is at a particular level of abstraction (layer)

- Each layer…
  - … is client of the lower layer
  - … provides services to the upper layer

## Example of 7 Layers



## Example of 3 Layers



## Benefits

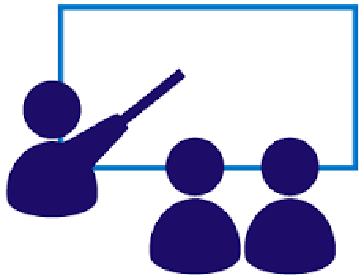
- Reuse of layers
  - Each layer embodies a well-defined abstraction with a documented interface
- Incremental Development
  - A lower layer does not depend on upper layers
- Exchangeability
  - A layer can be replaced by a semantically-equivalent one

## Liabilities

- Lower efficiency
  - Data have to be transferred through several layers (communication overhead)
- Difficulty of establishing the correct granularity of layers
  - Which services should go to each layer?
- Cascades of changing behavior
  - Changes in one layer may require updates to the others

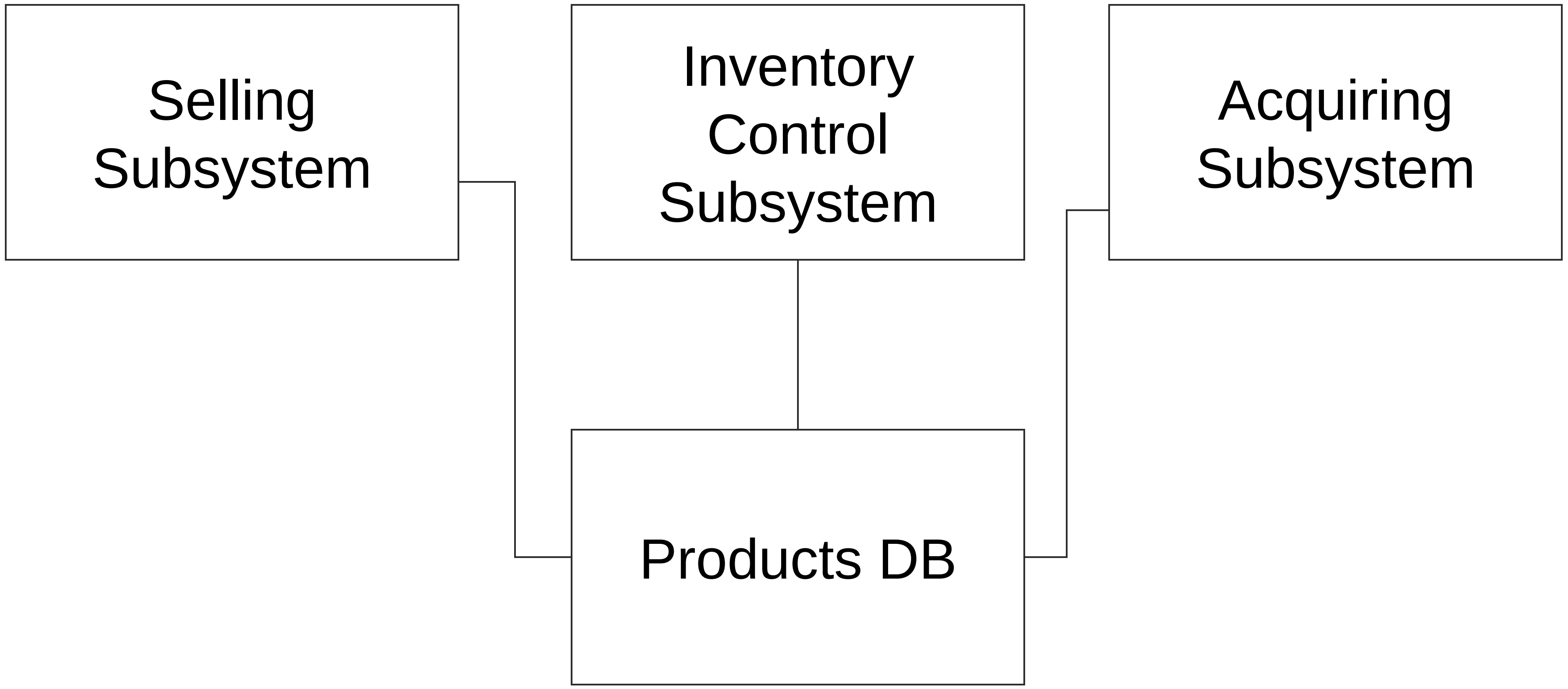
## Blackboard

## Blackboard



- Blackboard is useful for problems with no deterministic solution
  - Several specialized components assemble their knowledge to build a partial solution
- Components collaborate
  - Some components generate / write data
  - Some components use / read data
- This pattern is often used to share data among different subsystems

# Example of Blackboard

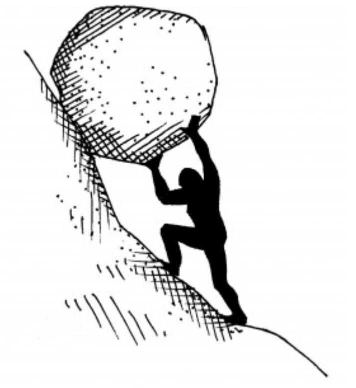


## Benefits

- Easy way to share data
  - Centralized backup and data protection
- Support for changeability and maintainability
  - A subsystem does not need to know the other subsystems
  - It is easy to aggregate additional subsystems
- Support for fault tolerance and robustness

## Liabilities

- Difficulty of testing
  - The solution may follow a non-deterministic algorithm
- All subsystems must understand the same format of data
  - They can have different requirements
- It may be hard to maintain a large dataset



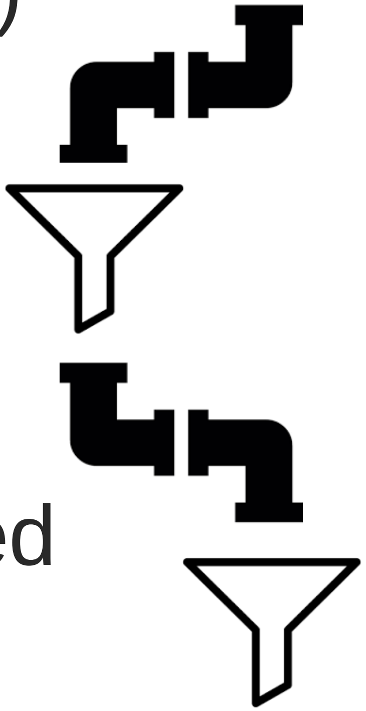
## Pipes and Filters

## Pipes and Filters

- This pattern provides a structure for systems that process a stream of data
- It defines two roles
  - **Pipes**: data is passed through these components
  - **Filters**: processing steps are encapsulated in filter components
- Recombining filters allows you to build families of related systems

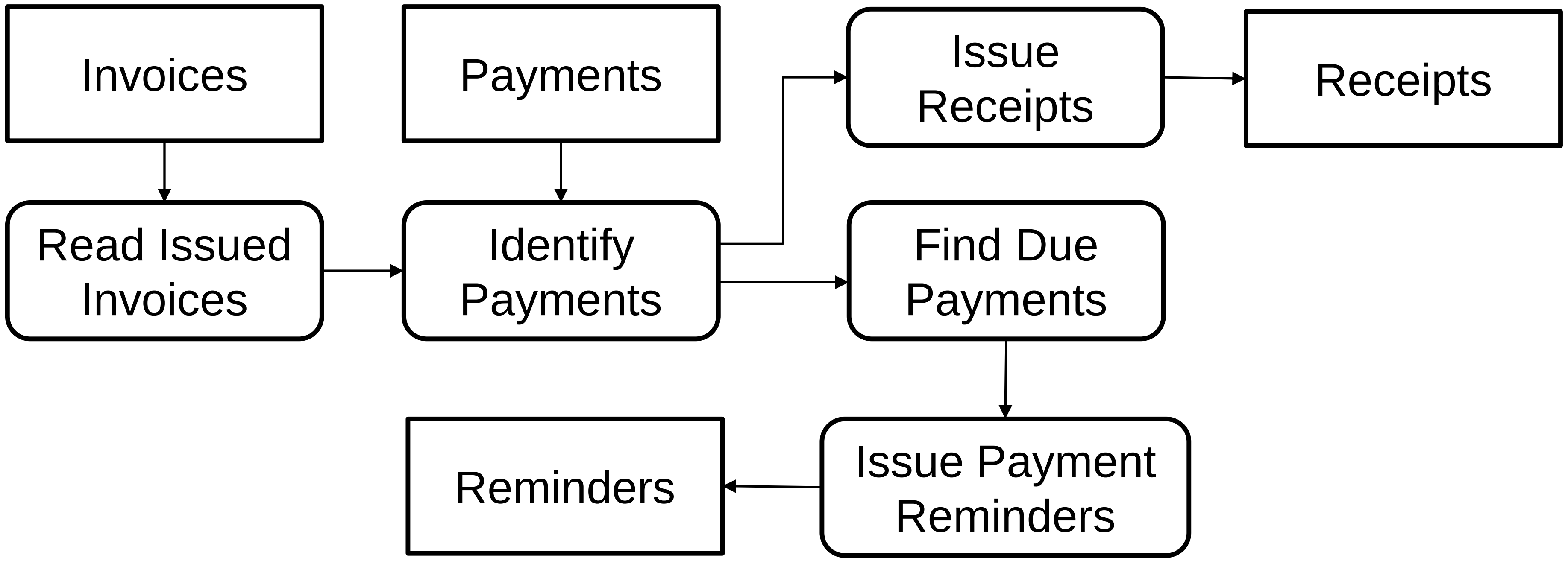
## Architectural Pattern Solution

- It divides the tasks of a system into several processing steps (filters)
- These steps are connected by the data flow (pipes)
    - Output data of one step is the input data of another step
- The sequence of filters combined by pipes is called pipeline



# Example of Pipes and Filters

- Input: Invoices and Payments
- Output: Receipts and Reminders



## Benefits

- Flexibility by filter exchange
  - Filters have a simple interface and a well-defined responsibility
- Flexibility to recombine and reuse
- Efficiency by parallel processing
  - Not only for sequential pipelines
- The workflow style is similar to several business models

## Liabilities

- Filters require a common format of data
  - In case of extensive data transformation, performance becomes a major concern
- Error handling is complex
  - Pipeline components do not share any global state
  - It is hard to give a general strategy for error handling

## Bibliography

- F. Buschmann et al. **Pattern-Oriented Software Architecture: A System of Patterns**. John Wiley & Sons, 1996.
  - Chap. 2  Architectural Patterns