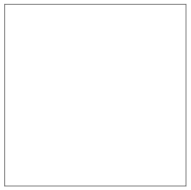
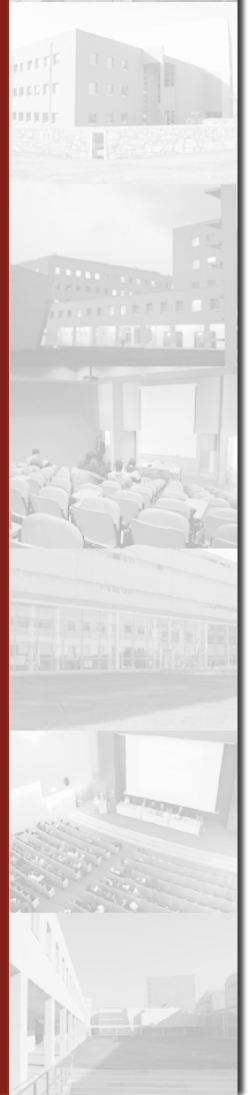
# Arquitectura de Sistemas de Software

MIEIC

Ademar Aguiar, Hugo Ferreira
ademar.aguiar@fe.up.pt



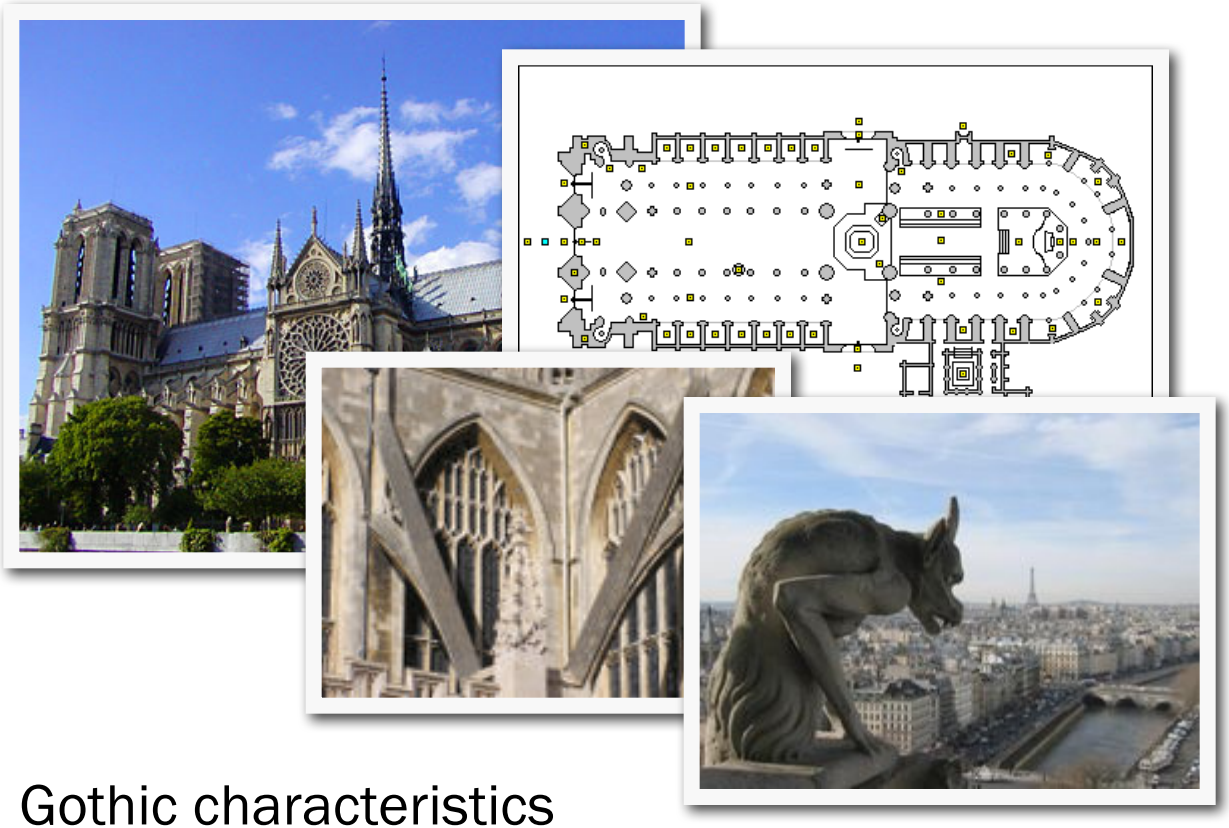



# Architectural styles

adaptado de
http://www.ugrad.cs.ubc.ca/~cs410/lectures/slides/cs410-ArchitecturalStyle1.ppt
e
http://www.ugrad.cs.ubc.ca/~cs410/lectures/slides/cs410-ArchitecturalStyle2.ppt

# Architectural styles

- Early Gothic Architecture, Notre Dame, Paris



- Gothic characteristics
  - Ogival archs, great expanses of glass, ribbed vaults, clustered columns, sharply pointed spires, flying buttresses and inventive sculptural detail such as gargoyles.
    [http://en.wikipedia.org/wiki/Gothic_architecture]

# Styles and patterns

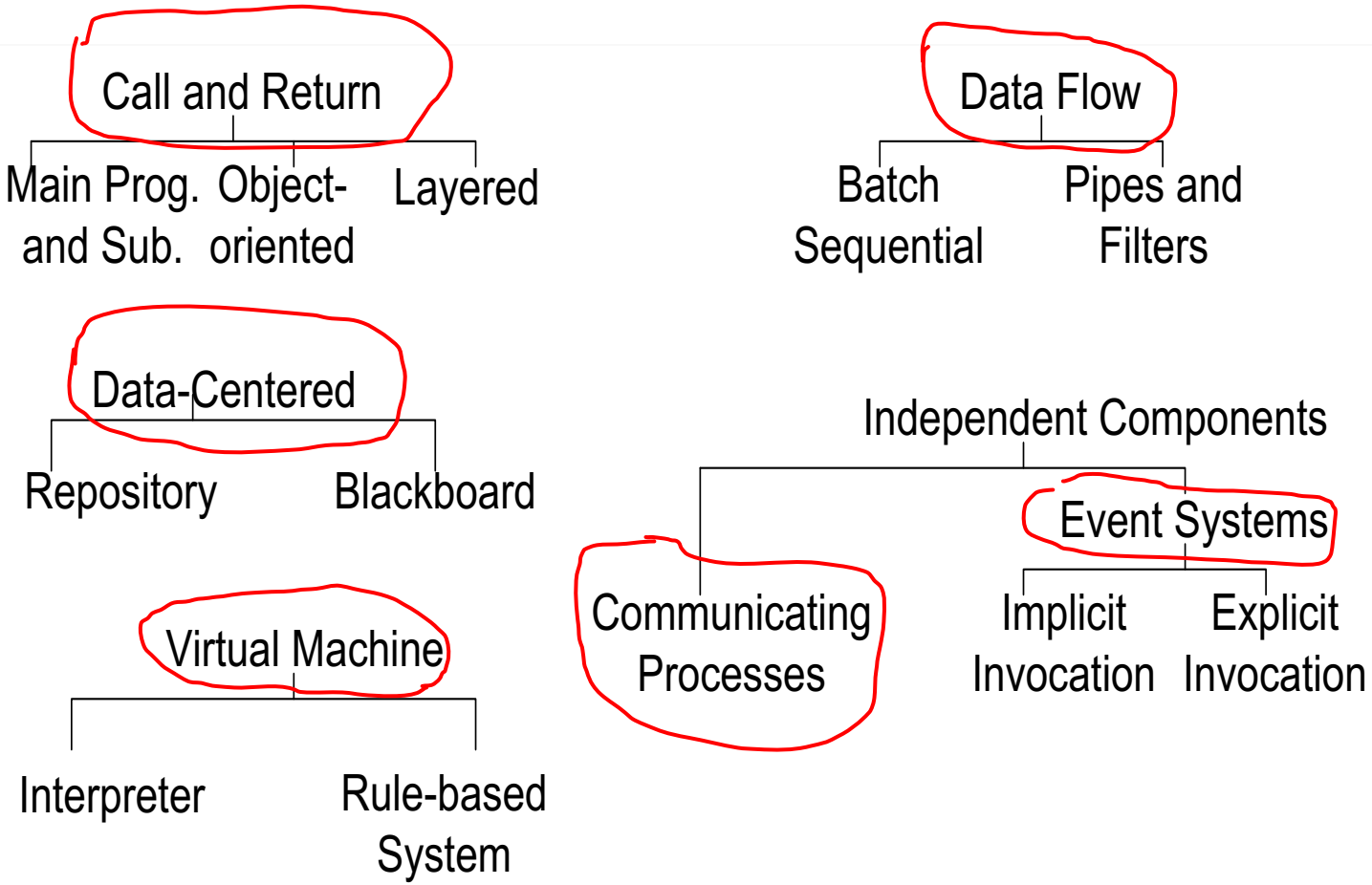
- Identifying styles and patterns can help codify and share knowledge/expertise
- Architectural Styles
  - [Shaw and Garlan, Software Architecture, Prentice Hall 96]
- Design Patterns
  - [Gamma, Helm, Johnson, Vlissides, Design Patterns, Addison Wesley 95]
  - [Buschmann et. Al, Pattern-oriented Software Architecture: A System of Patterns, John Wiley & Sons 96]
- Code Patterns
  - [Coplien, Advanced C++ Programming Styles and Idioms, Addison-Wesley 91]

# Architectural styles

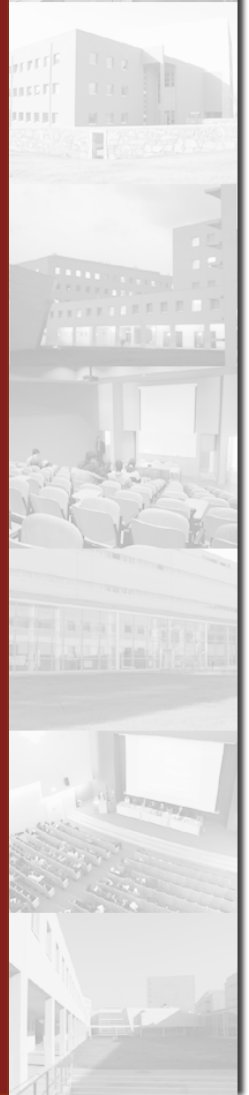
- A style consists of:
  - **a set of component types** (e.g., process, procedure) that perform some function at runtime
  - **a topological layout of the components** showing their runtime relationships
  - **a set of semantic constraints**
  - **a set of connectors** (e.g., data streams, sockets) that mediate communication among components
- Styles provide guidance for architectural design based on the problem domain and the context of use
  - Recurring (and proven) architectural design
  - Definition of common vocabulary
- Each style has:
  - components, connectors, key characteristics, strengths and weaknesses, variants and specializations.

*From Chapter 5, Software Architecture in Practice, p. 94*

# A catalog of architectural styles




*From Chapter 5, Software Architecture in Practice, p. 95*

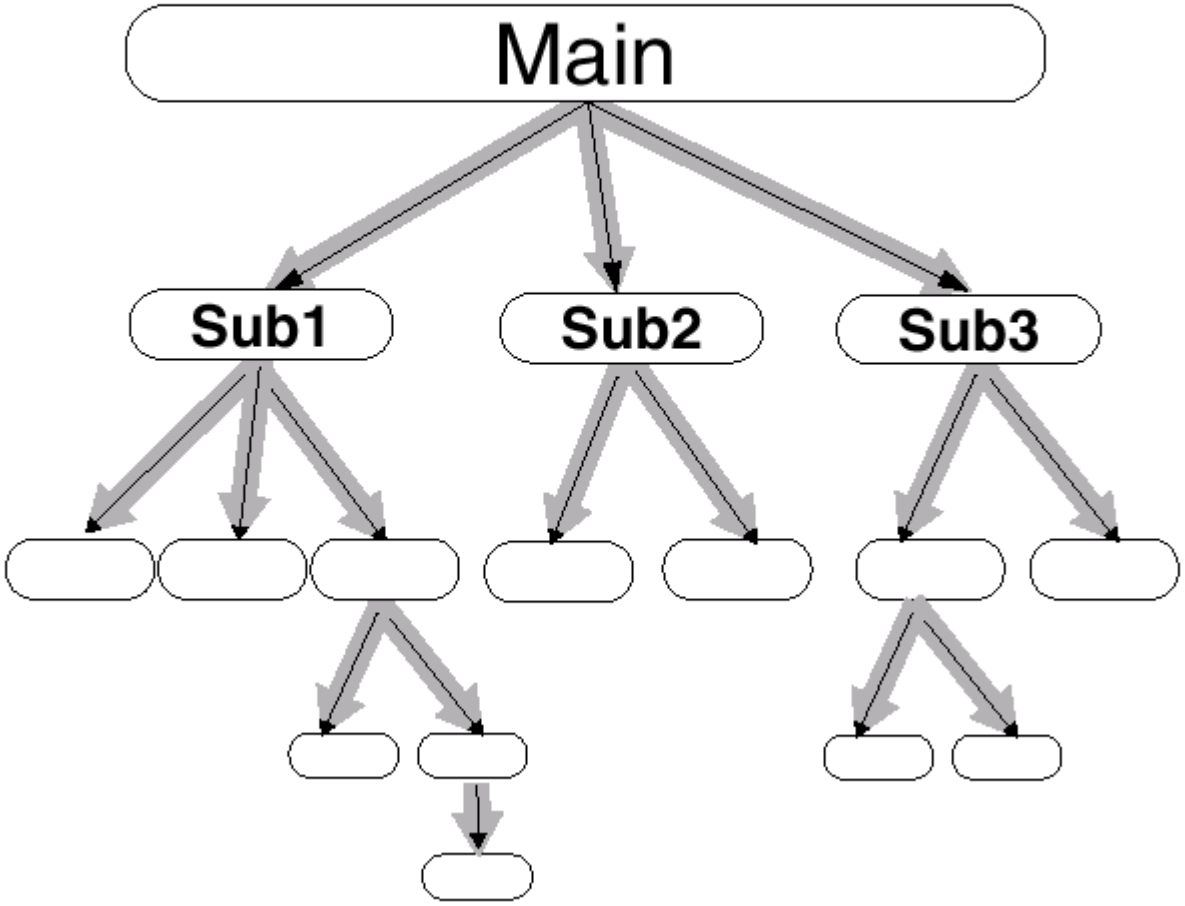



# Call and Return

# Call-and-Return style

- Goal: achieve modifiability and scalability
- Substyles:
  - Main-program-and-subroutine
    - Decompose program hierarchically. Each component gets control and data from its parent.
  - Remote procedure call
    - Components are distributed across a network. Remote call is between two computers, but otherwise looks like a procedure call.
  - Object-oriented or abstract data type
  - Layered

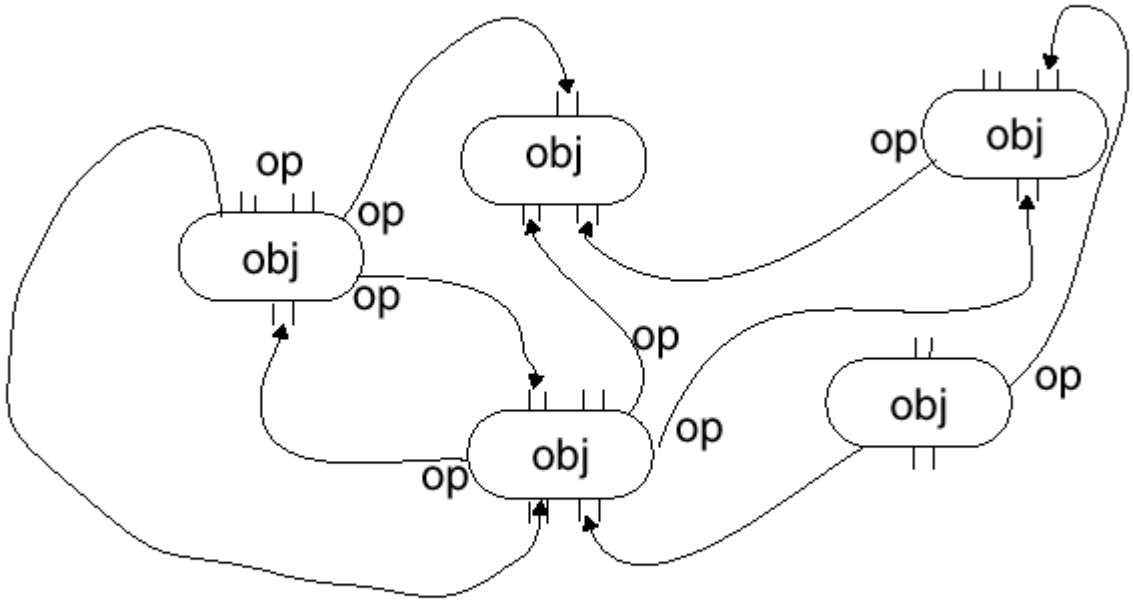
# Main Program/Subroutines



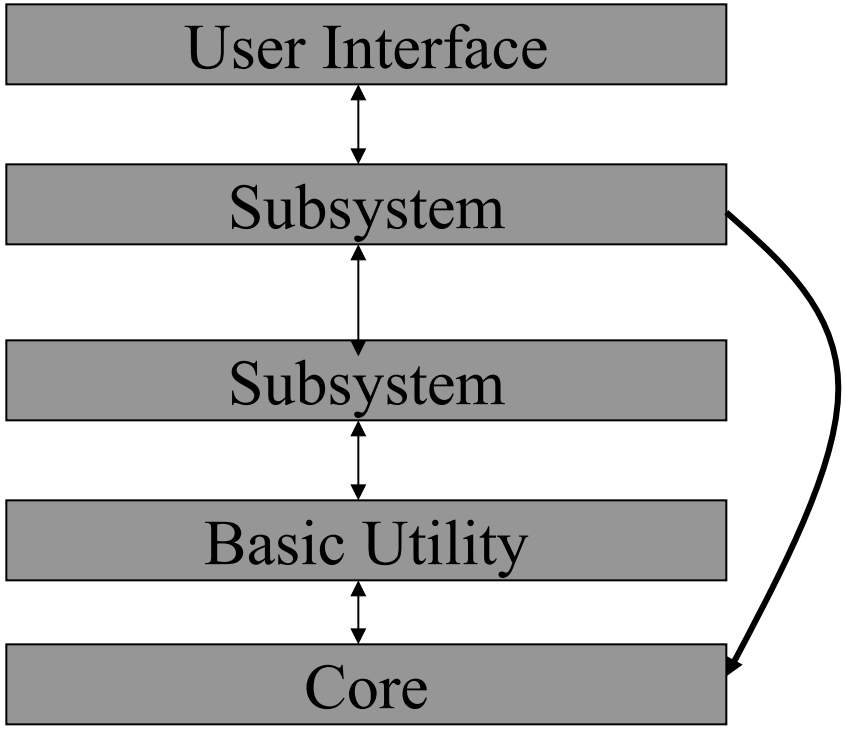
# Object-oriented (ADT) substyle

- Objects help achieve modifiability by encapsulating internal secrets from environment
- Access to objects is only through methods (constrained forms of procedure calls)
- Object-oriented paradigm is distinguished from ADT by inheritance and polymorphism

# Data Abstraction/Object Oriented

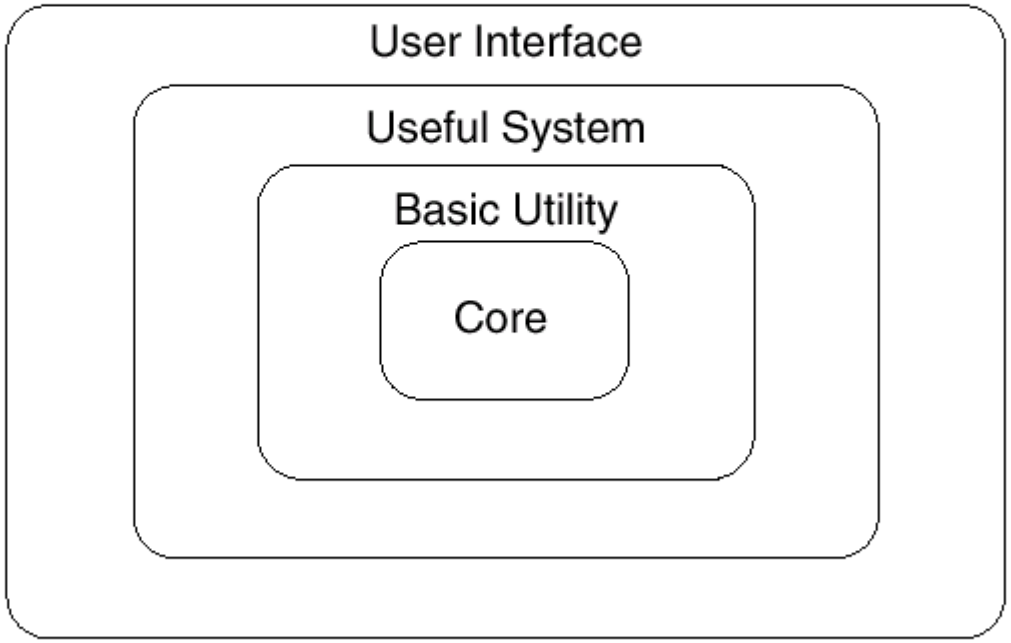


# Layered substyle

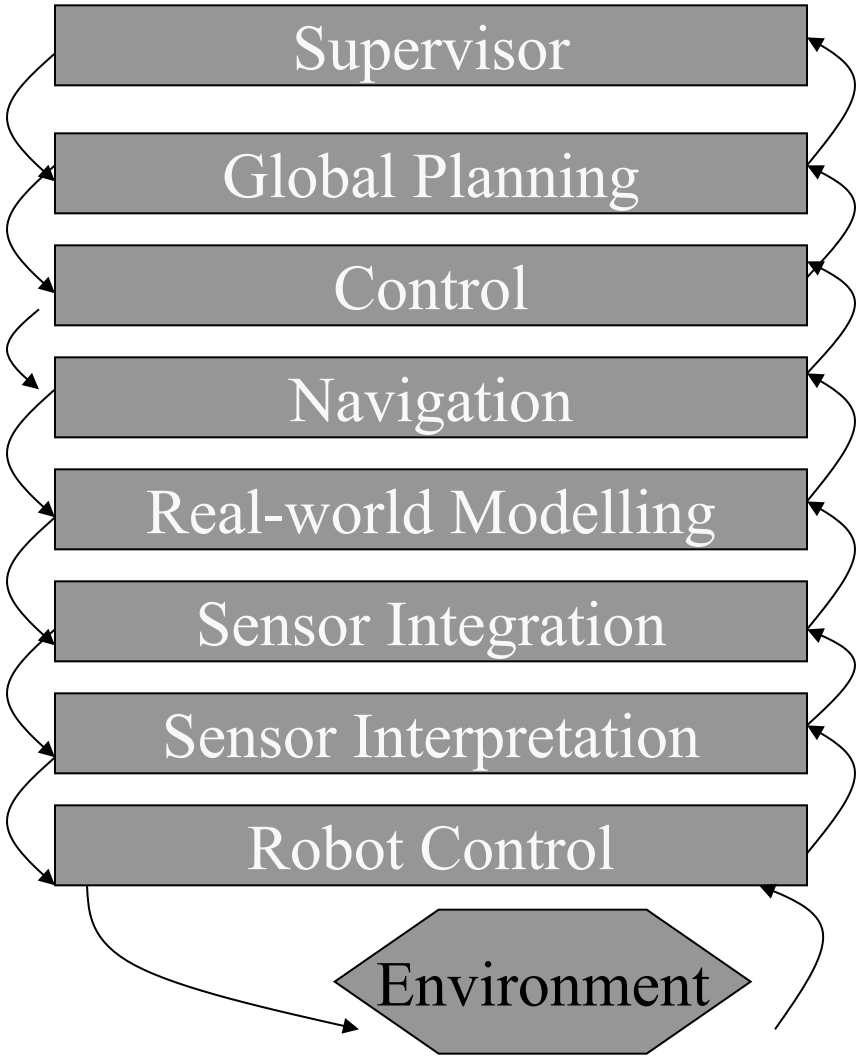



- Goals: modifiability and portability
- Optimally, each layer communicates only with its neighbors
- Sometimes, must layer bridge for performance: decreases benefits of style

# Layers



# Layered substyle: Example



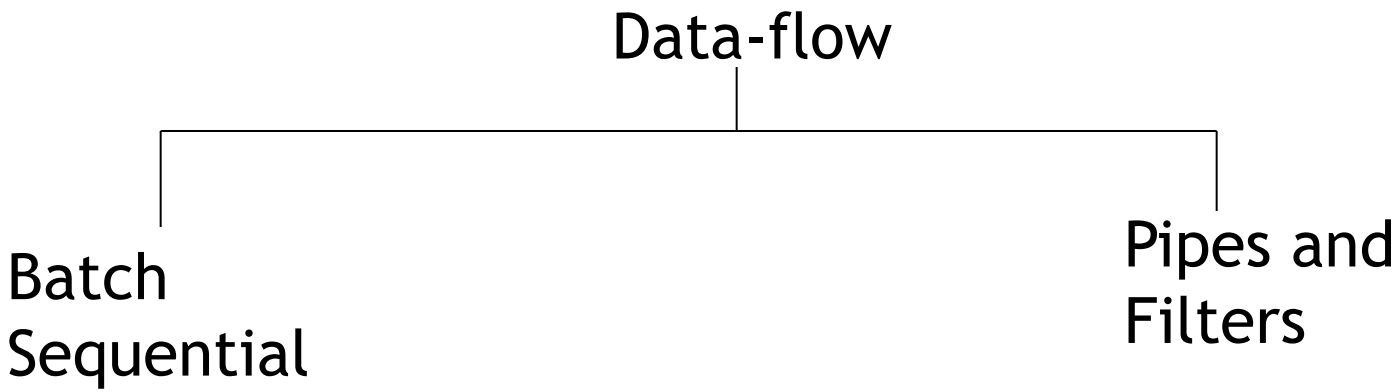

- Outline of a design used on a mobile robot
- Level 1 is robot control
- Levels 2-3 deal with input from real world
- Level 4 manages robot's view of world
- Level 5 manages robot navigation
- Levels 6-7 schedule and plan robot's activities
- Top-level is UI and supervisory

Universidade do Porto
Faculdade de Engenharia
FEUP

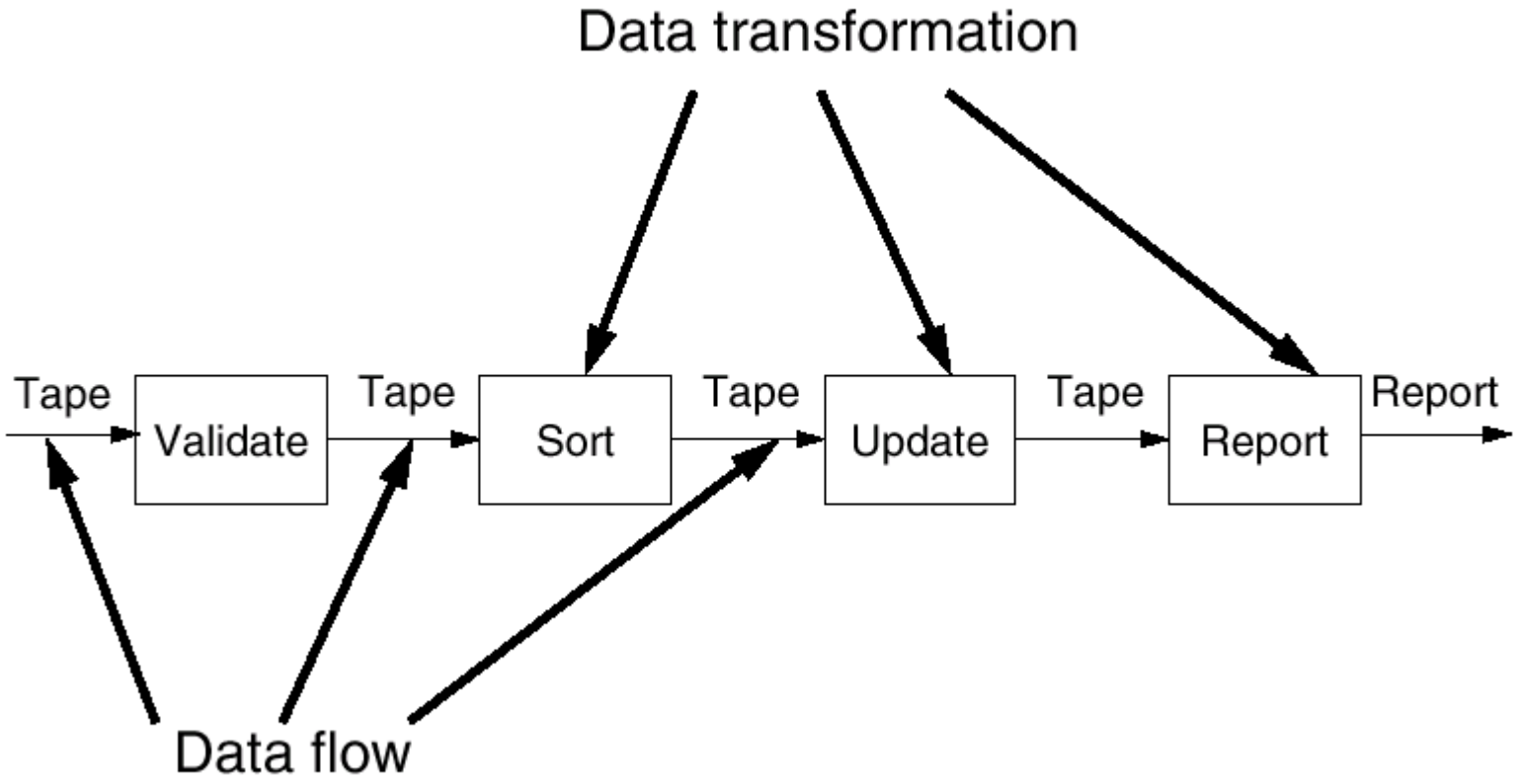
# Data-Flow

# Data-Flow style

- Goals: achieve reuse and modifiability
- Components are independent programs
  - each step runs to completion before the next starts
  - each batch of data is transmitted as a whole between steps
- Classic data-processing approach



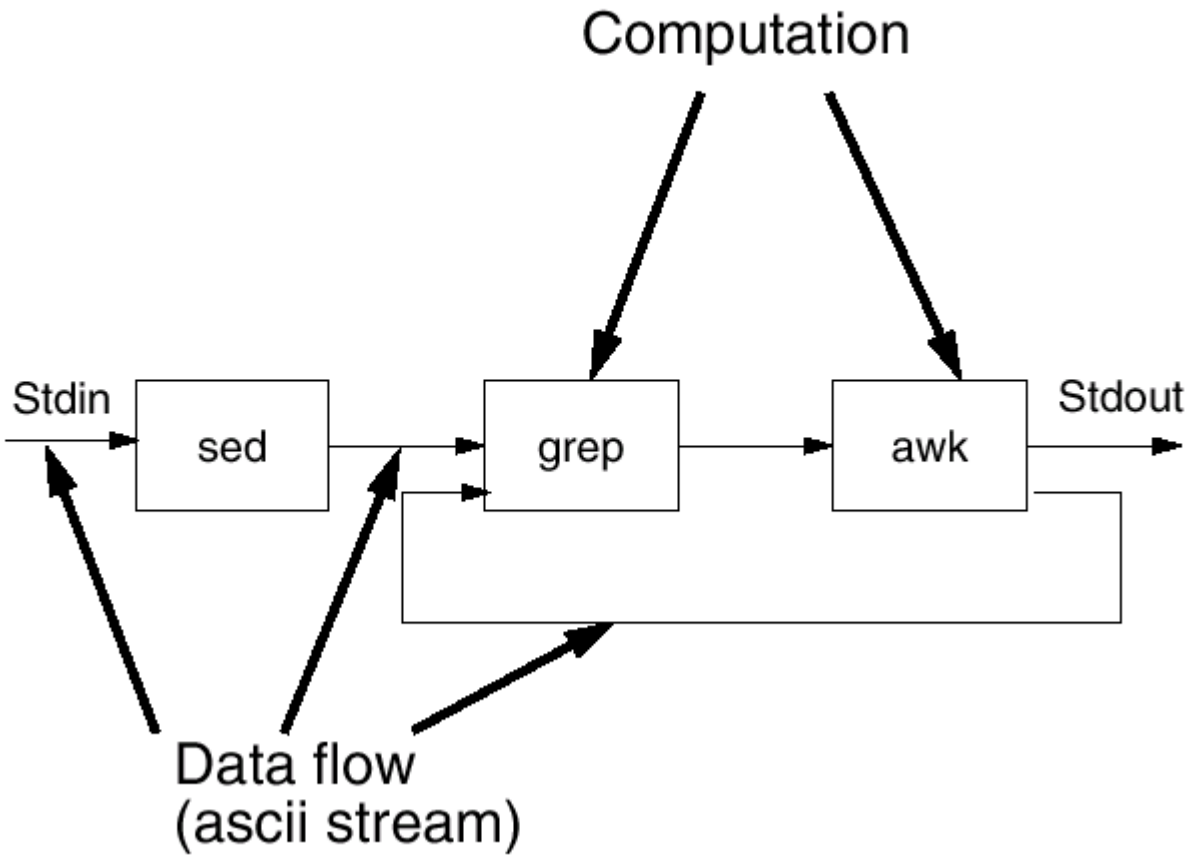
# Batch Sequential



# Pipe-and-Filter substyle

- Incremental transformation of data by successive components
  - Filters transform data using little contextual information and retain no state between instantiations
  - Pipes are stateless and are used to move streams of data between filters
- A pipe's source end connects to a filter's output port; sink end connects to filter's input port
- Pipes and filters run non-deterministically until no more computations or transmissions are possible
  + Simple to reason about, eases maintenance, enhances reuse, can easily enhance performance by parallelizing or distributing
  - Interactive apps are difficult, filters can't cooperate to solve a problem, performance is often bad

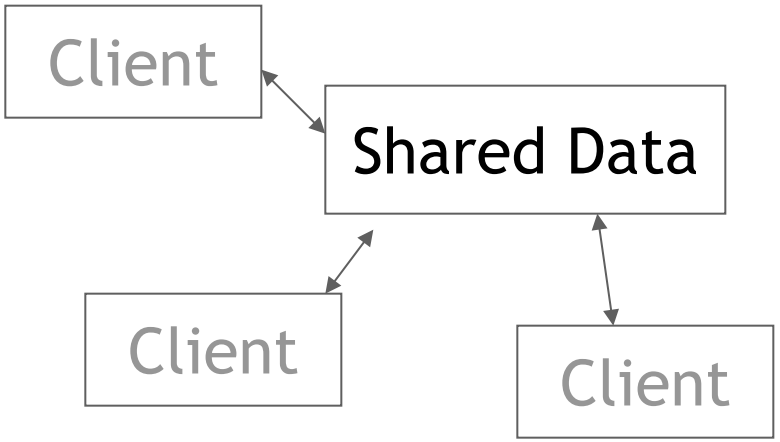
# Pipes and Filters



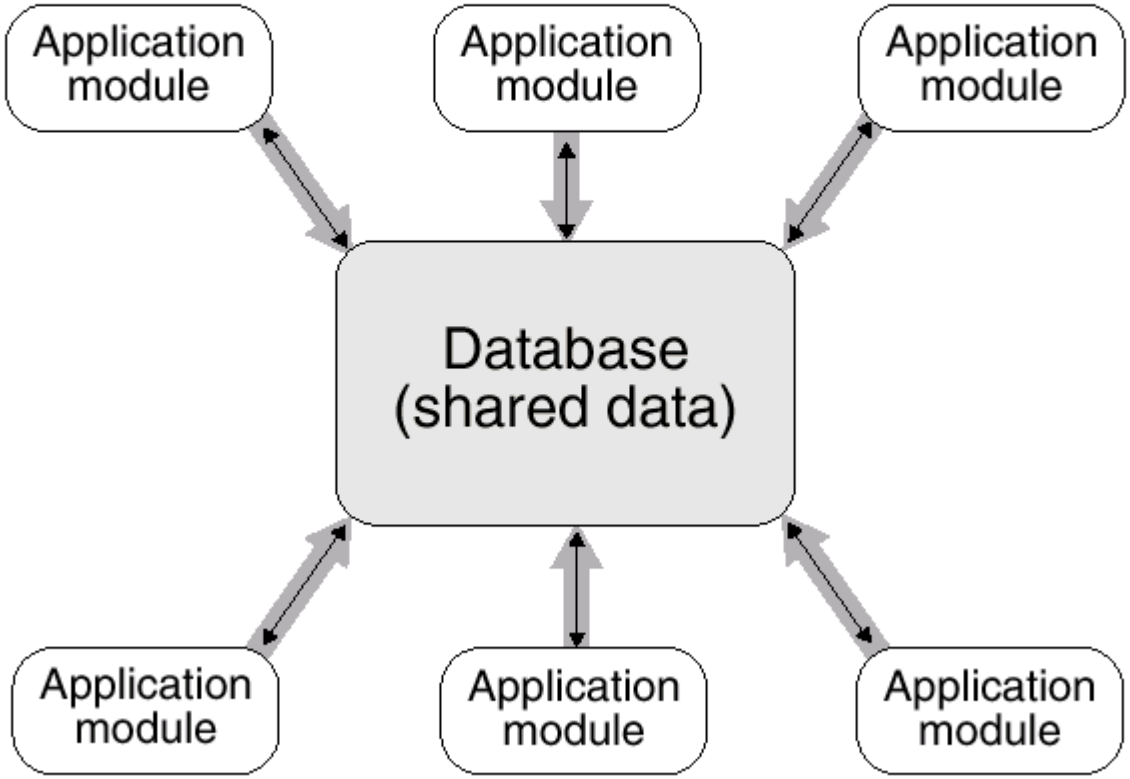

# Data-Centered

# Data-Centered styles

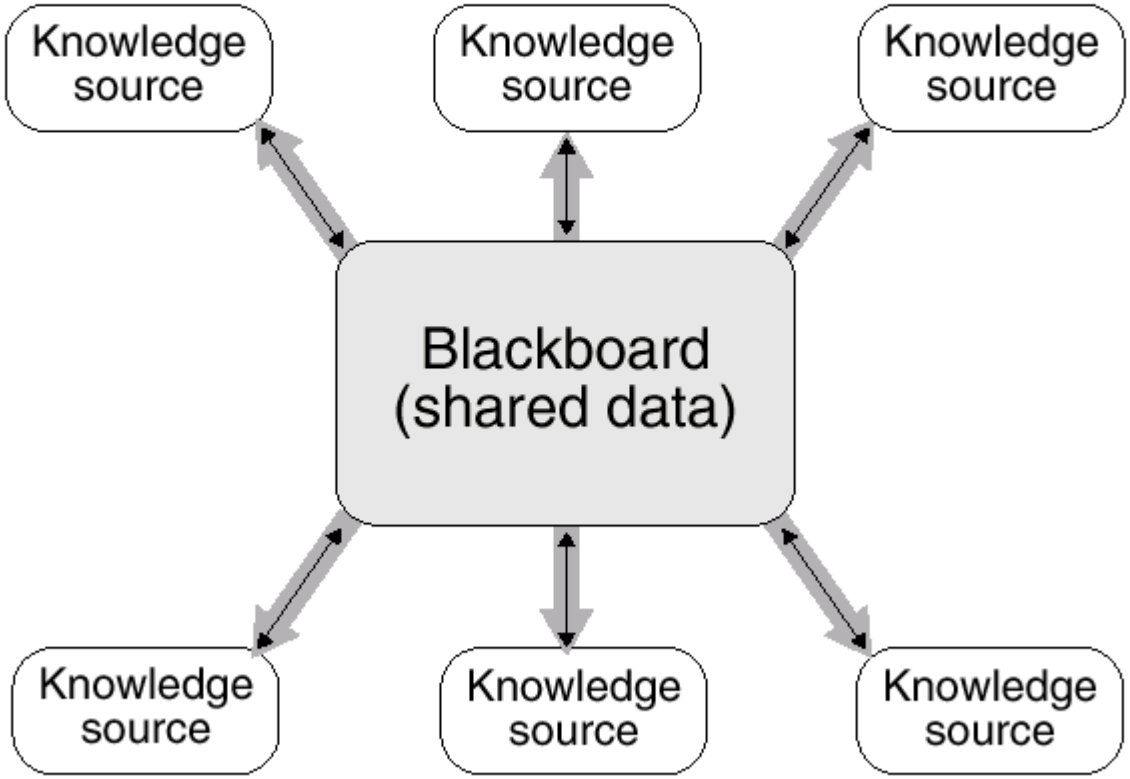
- Two-substyles: Repository and Blackboard
- When a system can be described as a centralized data store that communicates with a number of clients
- In a (passive) repository, such as shown on left, data might be stored in a file.
- In an active repository, such as a blackboard, the blackboard notifies clients when data of interest changes (so there would be control from data to clients)

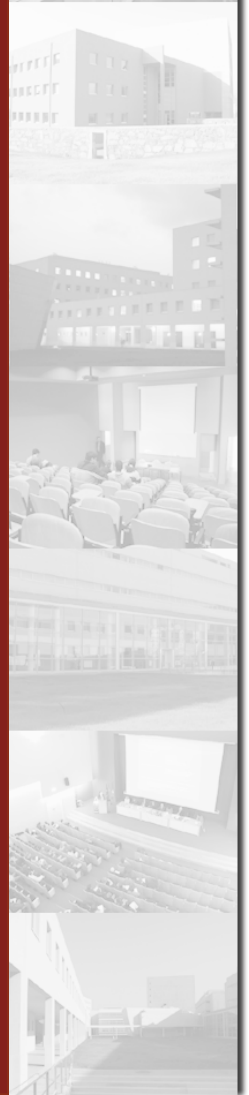


# Database


Application module
Application module
Application module
Database (shared data)
Application module
Application module
Application module

# Blackboard


Knowledge source
Knowledge source
Knowledge source
Blackboard (shared data)
Knowledge source
Knowledge source
Knowledge source

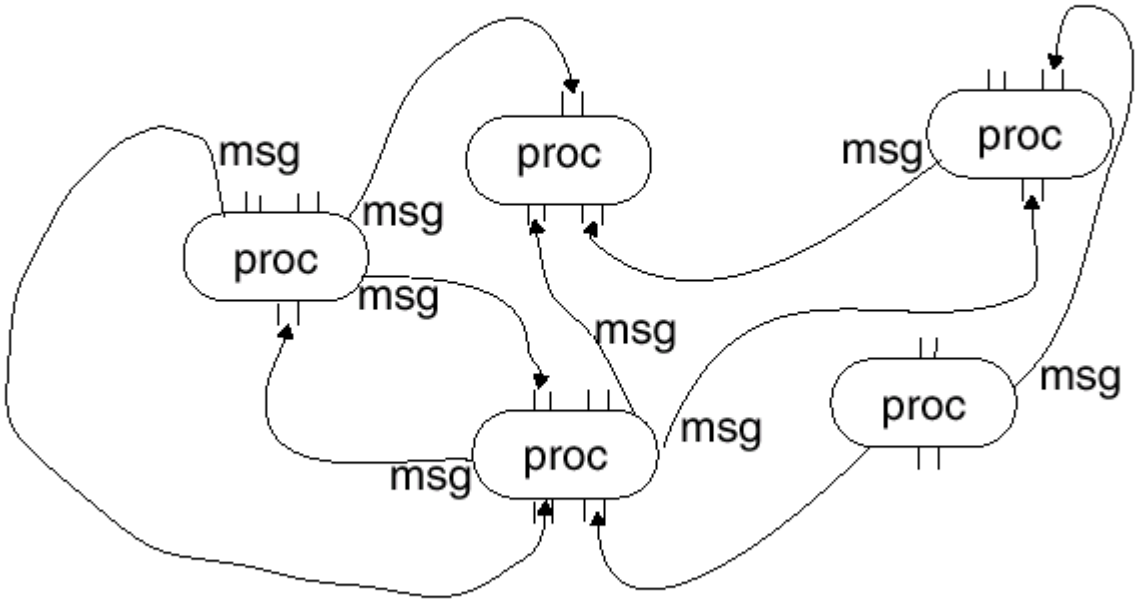



# Independent components

# Independent components style

- We'll look at the implicit invocation (event) substyle
- Goals: achieve modifiability by decoupling various parts of the computation
- Approach: have independent processes or objects communicate through messages

# Communicating Processes



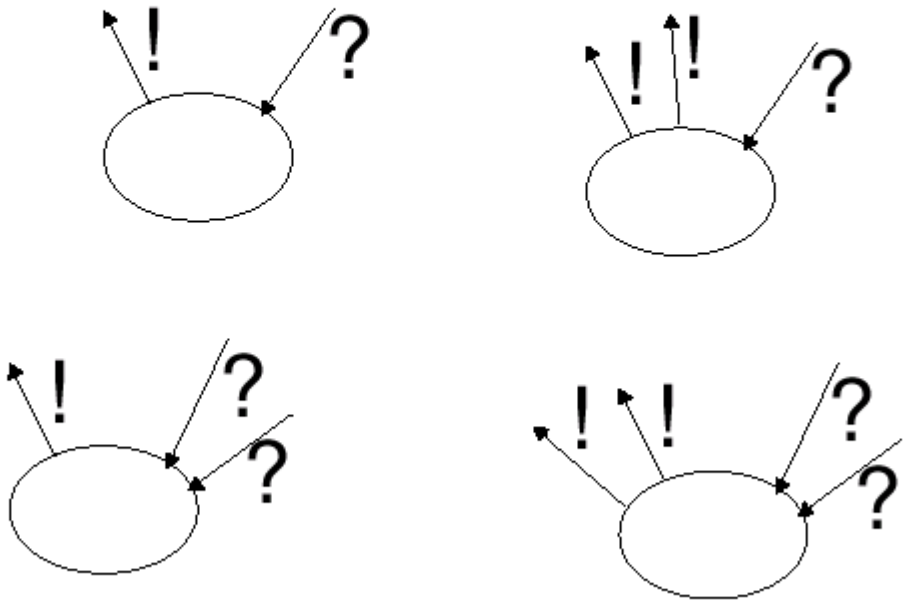
# Event systems

- Individual components announce data they wish to share (publish) with others
- Other components may register an interest in the kind of data (subscribe). They are invoked when the data appears
- How does this achieve modifiability?

# Implementing event systems

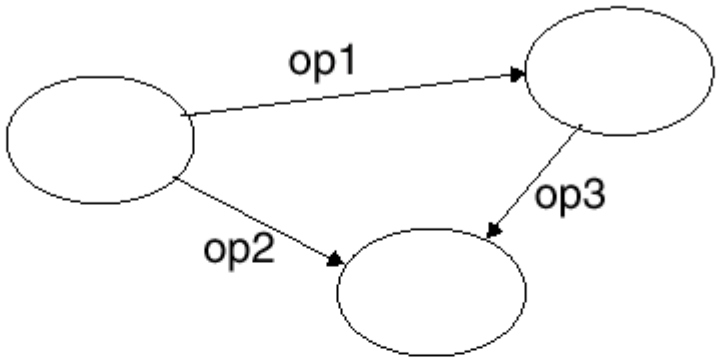
- Several different ways to achieve an event system:
  - Callbacks
  - Event Handling in Java Swing
  - Observer design pattern
  - Event notification pattern
  - Message Manager
  - (this is not a complete list)

# Event Systems

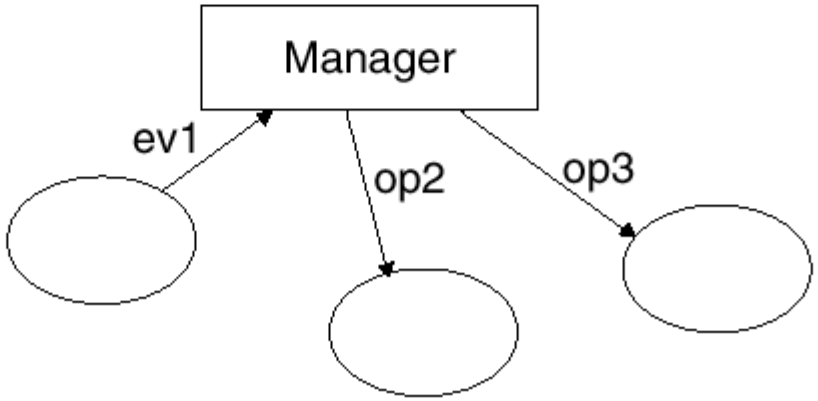


# Implicit vs. Explicit Invocation

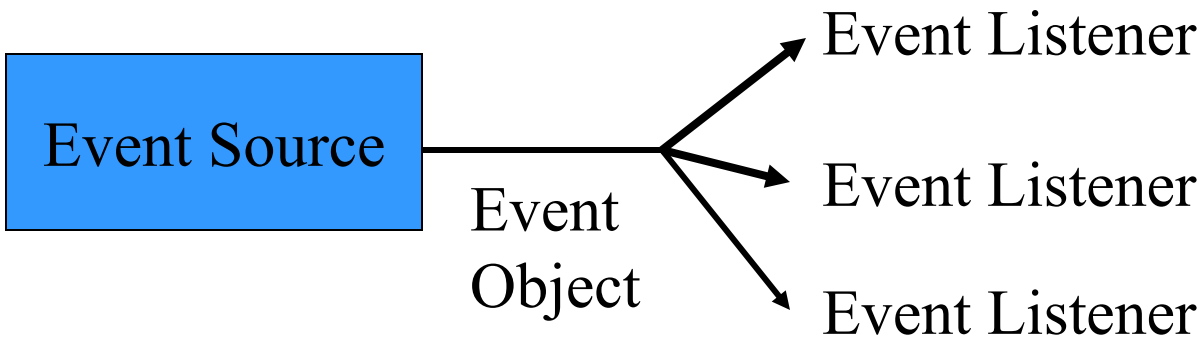
Explicit




Implicit



# Event handling in Java Swing




- Some examples:

| | |
|---|---|
| User closes a window | WindowListener |
| Table selection changes | ListSelectionListener |
| User clicks a button | ActionListener |

# Java Swing…

- ```
  public class MyClass implements ActionListener {
     public void actionPerformed( ActionEvent e )
     { … }
  }
  ```

- ```
  someComponent.addActionListener( instanceOfMyclass );
  ```

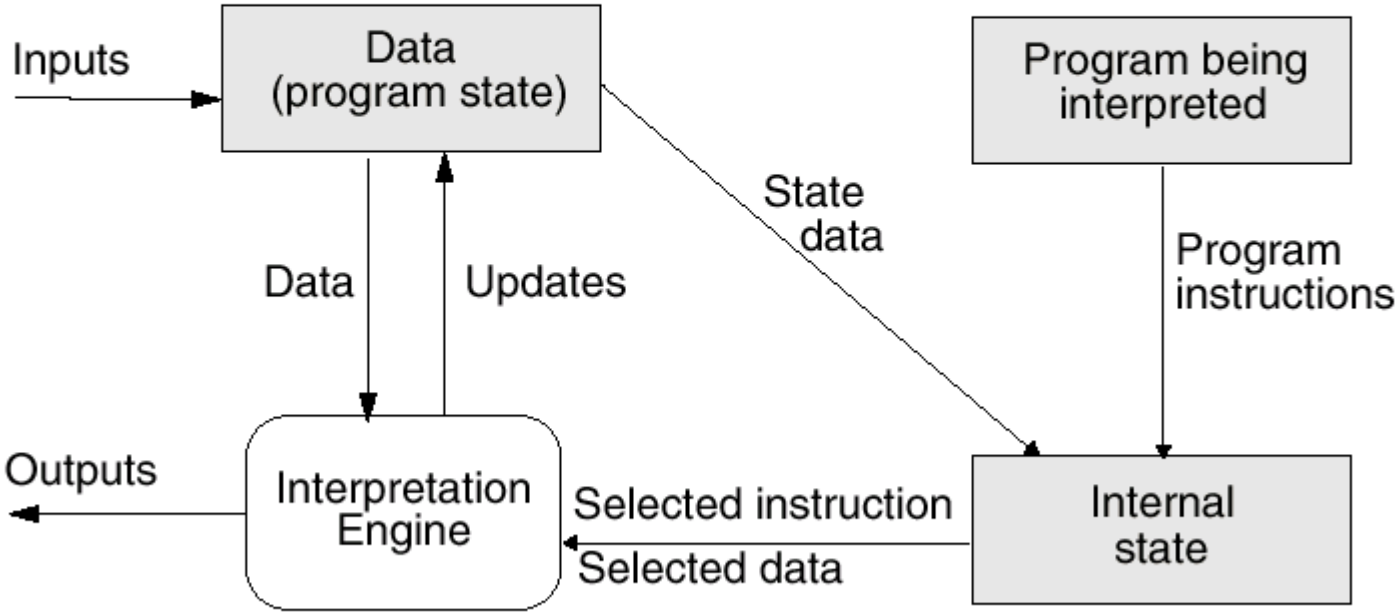
# Message manager

- Sometimes you want even more flexibility
- For instance, if building a programming environment, perhaps you want to be able to add tools on the fly and have them respond to events
- A message manager can help
  - Receive messages and route them to registered parties

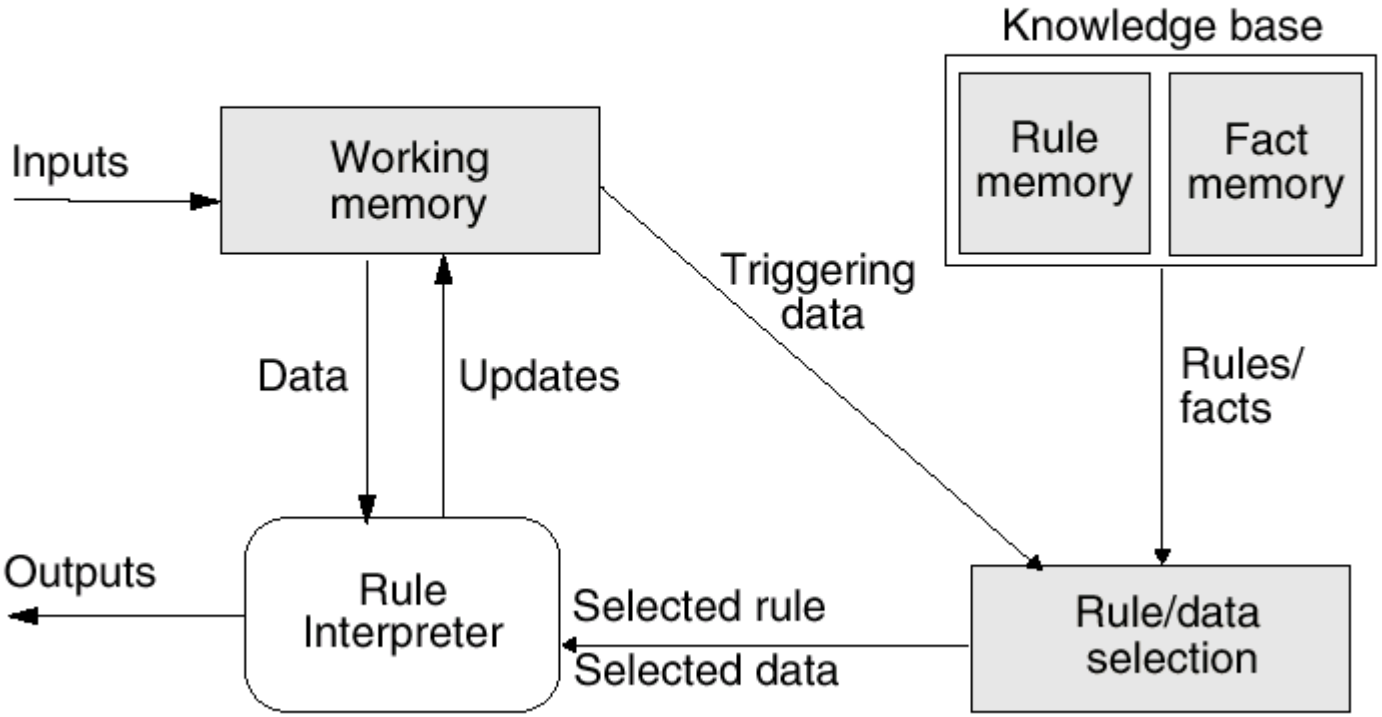
Universidade do Porto
Faculdade de Engenharia
FEUP

# Virtual Machines

# Interpreter



# Rule-Based System



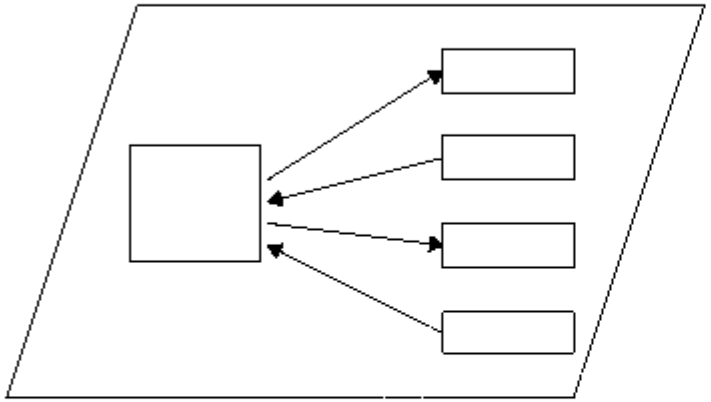

# Mixing styles

# Heterogeneous use

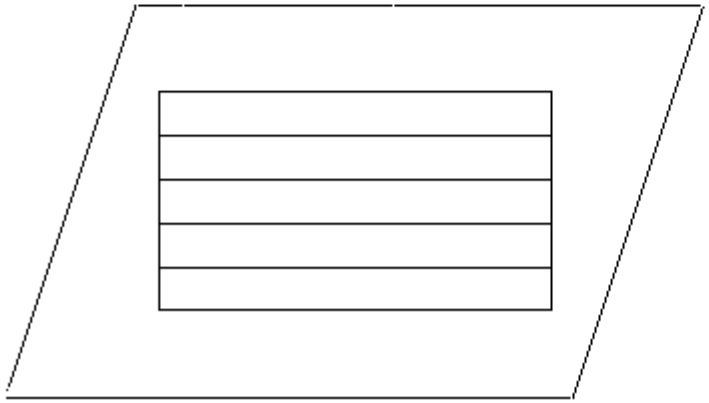
- Systems are generally built from multiple styles
- Three kinds of heterogeneity:
  - *Locationally heterogeneous*: run-time structures reveal different styles in different areas
  - *Hierarchically heterogeneous*: a component of one style, when decomposed is structured according to another style
  - *Simultaneously heterogeneous*: different styles depending on point of view

# Heterogeneous Styles

**A system need not be comprised of only one style, e.g.**



**Higher-level style: event system**



**Lower-level style: layered**

# Characterizing and comparing styles

- The following categories are useful in comparing and characterizing styles:
  - What kinds of components and connectors are used in the style?
  - What are the control structures?
  - How is data communicated?
  - How do data and control interact?
  - What kind of reasoning does the style support?

# Components & Connectors

- “A component is a unit of software that performs some function at runtime.” [p.105]
- “A connector is a mechanism that mediates communication, coordination, or cooperation among components” [p. 105]

# Control Issues

- How does control pass among components?
- *Topology*:
  - What is the topology of a batch-sequential data-flow style?
  - What is the topology of a main-program-and-subroutine style?

Other topologies include star, arbitrary… Topology may change at runtime.

# Control Issues…

- *Synchronicity*: How interdependent are the component's actions upon each other's control states?
  - E.g., lockstep - state of one component implies state of all others
  - E.g., synchronous - components synchronize regularly; other state relationships are unpredictable

# Control Issues…

- *Binding Time*: When is the partner in a communication established?
  - Program-write time?
  - Compile-time?
  - Invocation-time?

# Issues…

- Data Issues:
  - *Topology*
  - *Continuity*: Continuous vs. sporadic; volume
  - *Mode*: Passed, shared, copy-out-copy-in, etc.
  - *Binding Time*: Same as control issue
- Control/Data Interaction Issues:
  - *Shape* of control and data topologies
  - *Directionality*: does data flow in direction of control?
- Type of Reasoning

# Assignment

- Identify the main architectural styles of your subsystem
  - Identify the components and connectors of those styles;
    - What kinds of components and connectors are used in the style?
    - What are the control structures?
    - How is data communicated?
    - How do data and control interact?
    - What kind of reasoning does the style support?
  - Systems can be heterogeneous in terms of styles, mainly three kinds:
    - Locationally heterogeneous: run-time structures reveal different styles in different areas
    - Hierarchically heterogeneous: a component of one style, when decomposed is structured according to another style
    - Simultaneously heterogeneous: different styles depending on point of view
  - Your subsystem may not be ”unifiable” into a single responsibility. In this case, divide it into orthogonal subsystems, and treat each other separately:
    - But how will they integrate/communicate?
- Identify architectural / design problems
- 90 seconds presentation of the results