Software Architecture

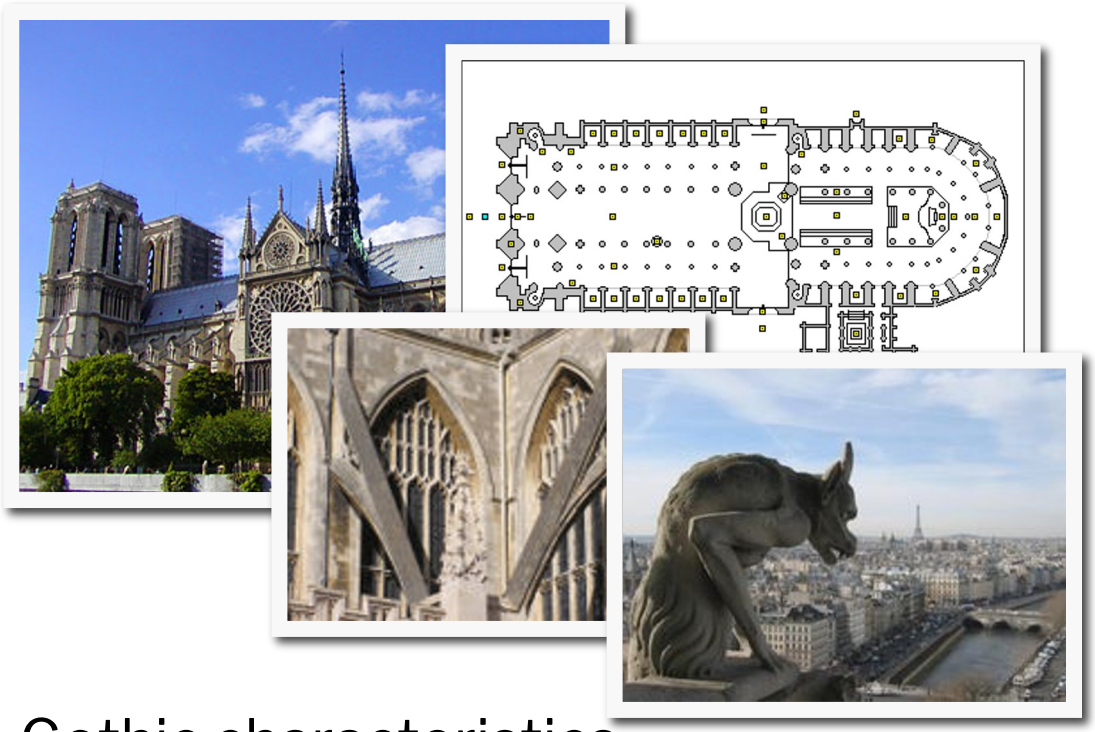
FEUP-M.EIC-ASSO-2023-24

Ademar Aguiar, Neil Harrison

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

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 et. al, 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

An architectural 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

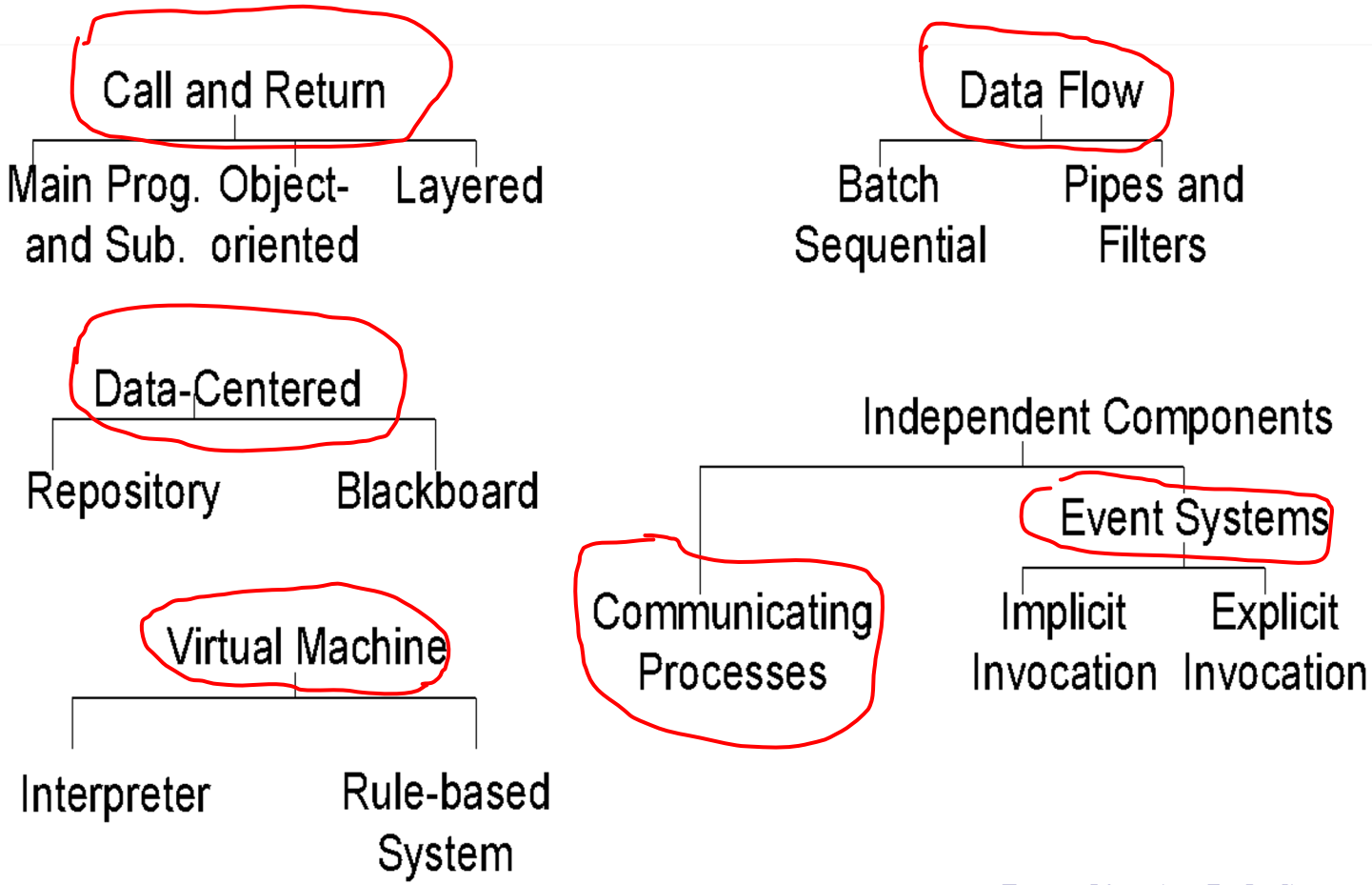
- Recurring (and proven) architectural design
- Definition of common vocabulary

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 style

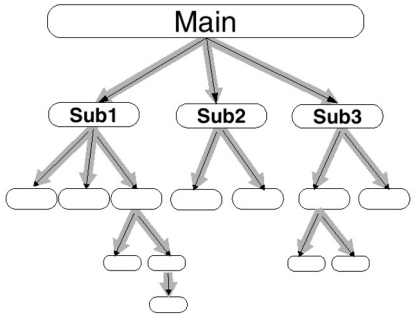
Goal:

- achieve modifiability and scalability

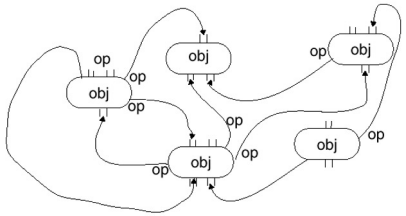
Substyles:

- **Main-program-and-subroutine**
  Decompose program hierarchically. Each component gets control and data from its parent.
- **Data Abstraction / Object-oriented**
  Achieve modifiability by encapsulating internal secrets from environment. Access to objects is only through methods. Object-oriented paradigm is distinguished from ADT by inheritance and polymorphism
- **Layered**
  Seeks modifiability and portability. Optimally, each layer communicates only with its neighbors. Sometimes, must layer bridge for performance: decreases benefits of style

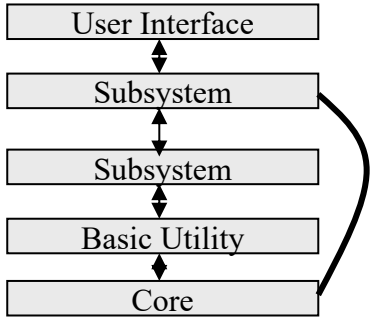
Main Program/Subroutines



Data Abstraction/Object Oriented



Layered



Data-Flow style

Goals: achieve reuse and modifiability

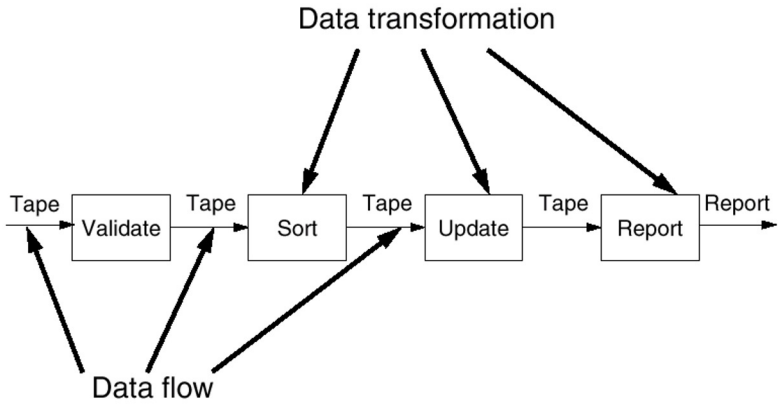
Components are independent programs

- each step runs to completion before the next starts
- each batch of data is transmitted as a whole between steps

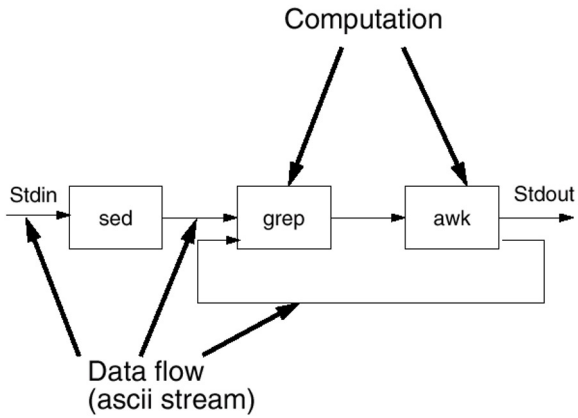
Classic data-processing approach

Substyles: **Batch sequential**; **Pipes and filters**.

Batch sequential



Pipes and Filters



Data-Centered style

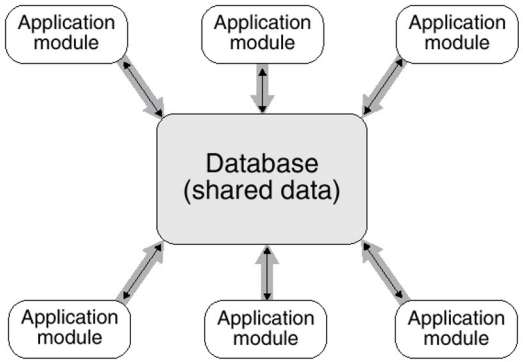
Two-substyles: Repository and Blackboard

When a system can be described as a centralized data store that communicates with a number of clients

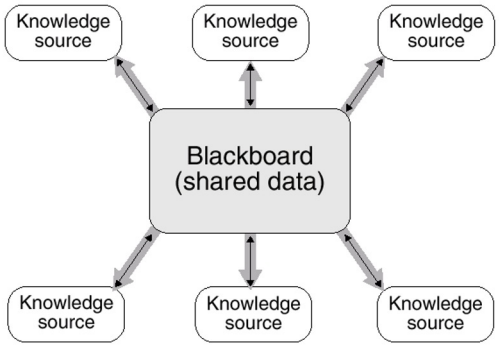
In a (passive) repository, such as shown on left, data might be stored in a file.

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



Blackboard



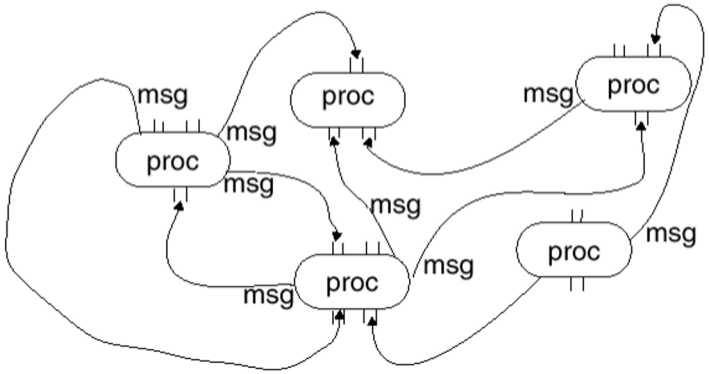
Independent components style

Goals: achieve modifiability by decoupling various parts of the computation

Approach: have independent processes or objects communicate through messages

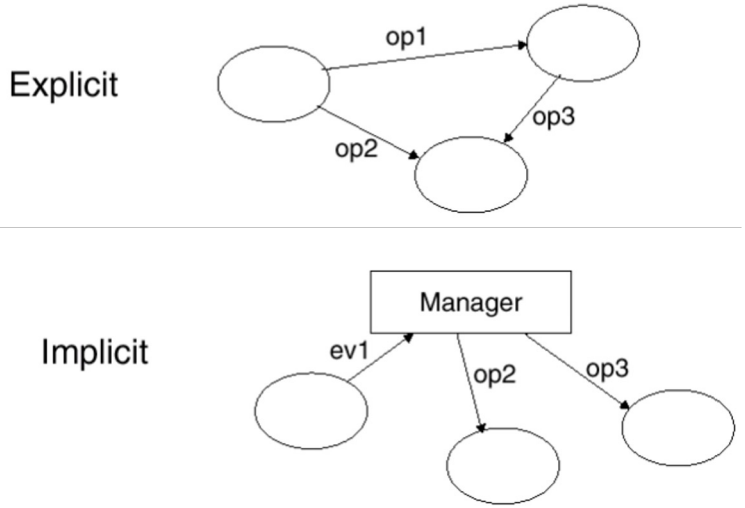
Substyles: **Communicating processes**; **Event systems**.

Communicating Processes



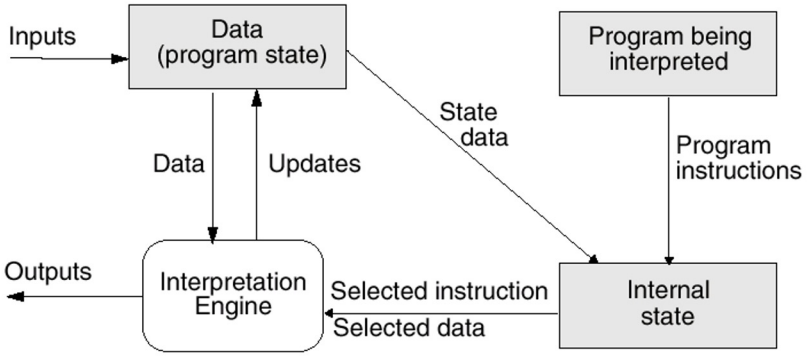
Event systems with

Implicit vs. Explicit Invocation

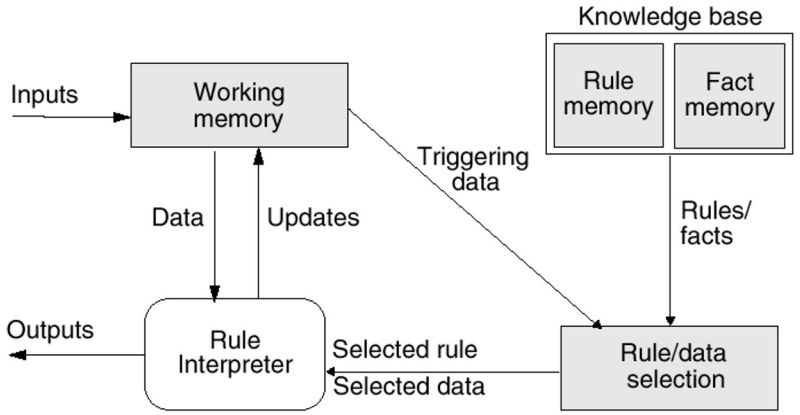


Virtual machine style

Interpreter



Rule-Based System



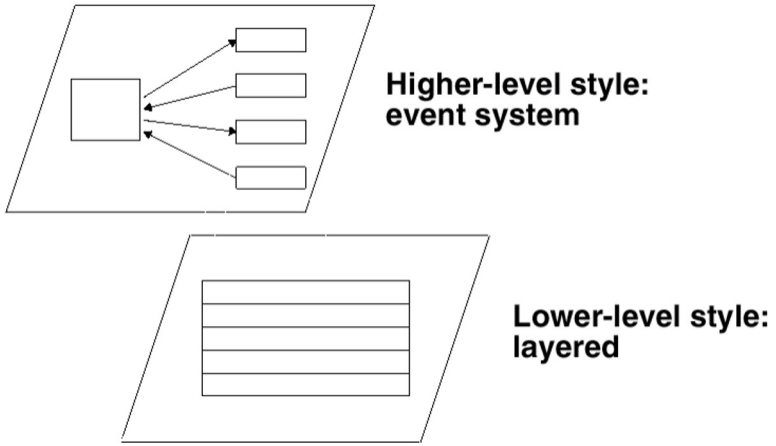
Mixing styles

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

Example



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?

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

Binding Time: When is the partner in a communication established?

- Program-write time?
- Compile-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

Group work

Split into groups

Identify the main architectural styles of a well-known system:

- 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 system 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

10 seconds presentation of the results

system name, style name 1, style name 2(, style name 3)

Thank you!