CS 490 - Architecture Patterns and Styles
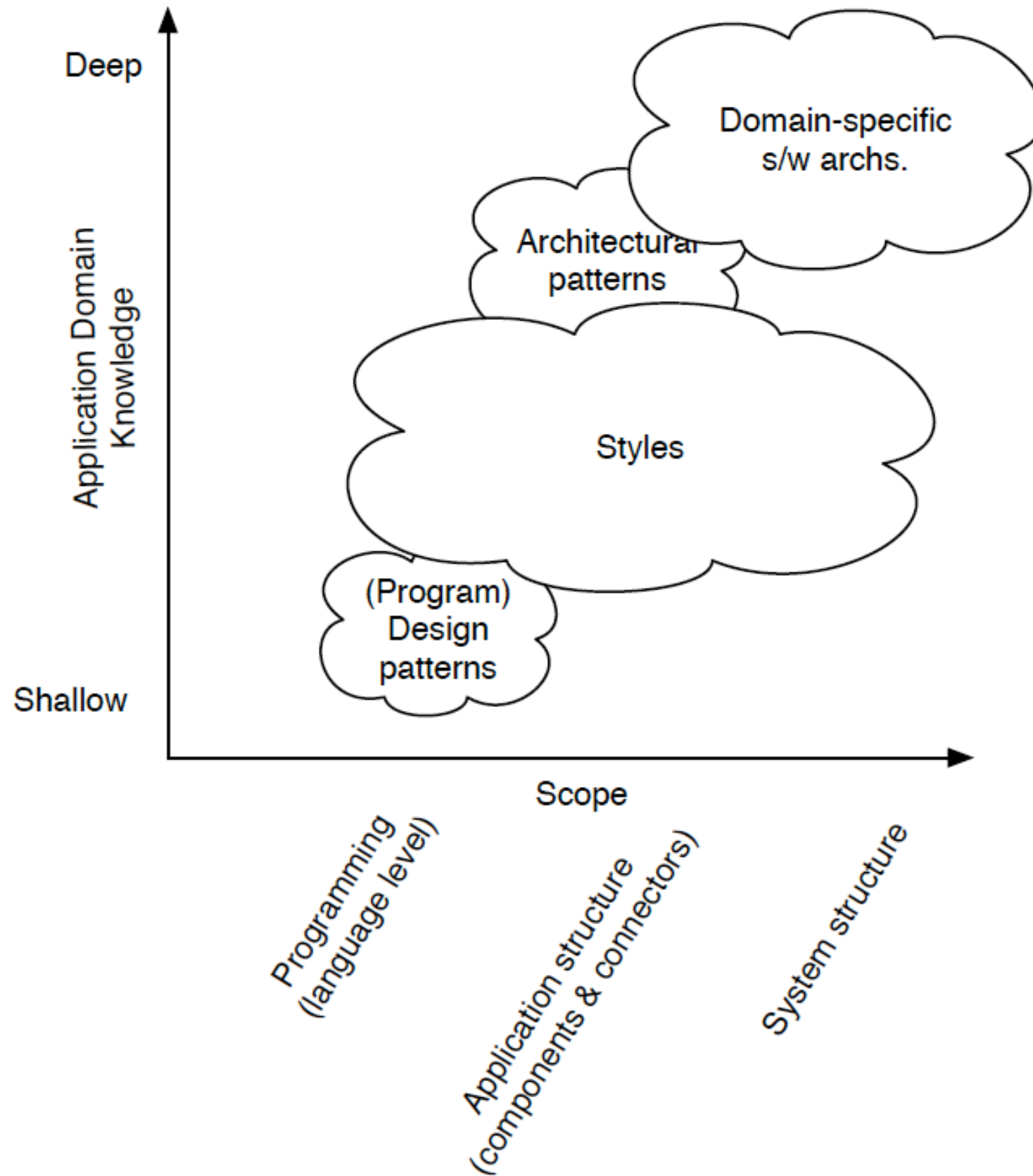
Instructor: Yongjie Zheng

How do we design architecture?

- Creativity

  - This requires extensive experience, broad training, ...

- Principles, process, and methods

  - Goals, activities, and principles

  - Process

  - Design methods: object-oriented design, functional design, and quality-driven design

- Reuse

  - Horizontal reuse: **architecture patterns and styles**

  - Vertical reuse: product-line architectures

Architecture Patterns and Styles

- Architecture pattern: a named collection of architecture design decisions that are applicable to a **recurring design problem**, parameterized to account for different software development contexts in which that problem appears.

- Architecture style: a named collection of architectural design decisions that (1) are applicable in a given development **context**, (2) **constrain** architectural design decisions that are specific to a particular system within that context, and (3) elicit **beneficial** qualities in each resulting system.

Deep

Application Domain Knowledge

Shallow

Domain-specific s/w archs.

Architectural patterns

Styles

(Program) Design patterns

Scope

Programming (language level)

Application structure (components & connectors)

System structure

4

Architecture Patterns and Styles

- Architecture patterns

    - Model-View-Controller

    - Sense-Compute-Control

- Architecture styles

    - Pipe-and-filter

    - Implicit invocation

    - Blackboard

    - Layered

- Some other patterns and styles

    - State-Logic-Display (Three-Tier), Client-Server, Interpreter, REST, etc.
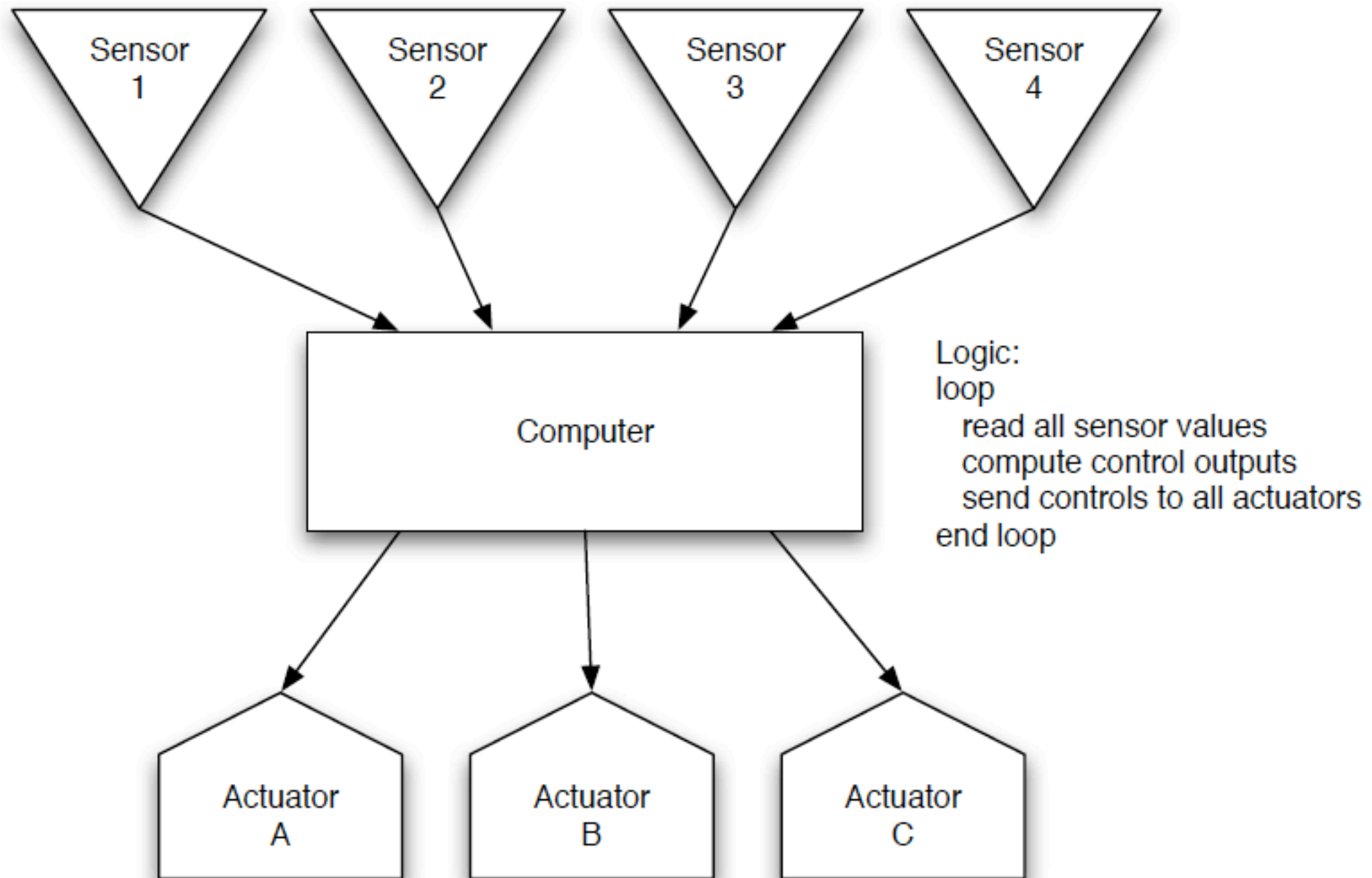
Model-View-Controller (MVC)
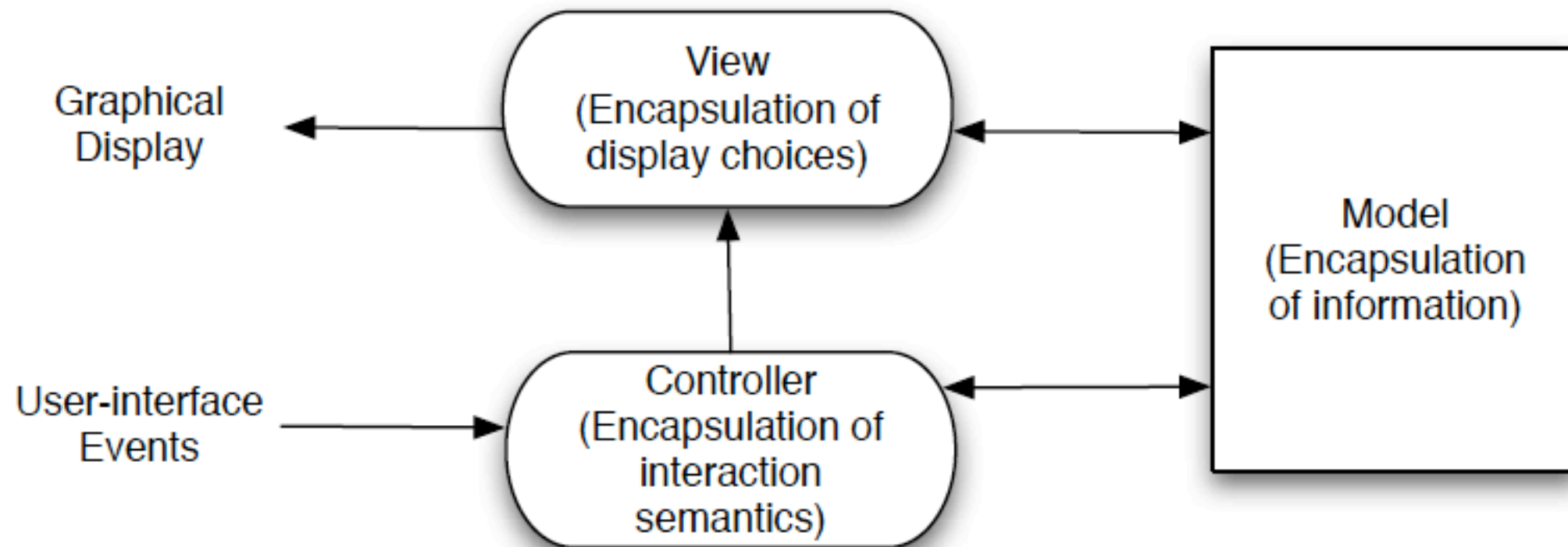
- Model: the information used by the application.
- View: screen presentation of the information.
- Controller: defines the way the user interface reacts to user input and maintains view-model consistency.
- Model-View: the subscribe/notify relationship.
- View and Controller may be combined in some cases.

MVC, cont.

- Typically, a MVC application works as follows:
  - The user interacts with the application.
  - The controller handles the input event from the user interface.
  - The controller may ask the model to update its information in response to the user input, or ask the view to re-draw without updating the model.
  - If the model is updated, the view is notified (indirectly).
 - The application waits for additional user inputs.

Sense-Compute-Control

Sensor 1 · Sensor 2 · Sensor 3 · Sensor 4

Computer

Logic:
loop
  read all sensor values
  compute control outputs
  send controls to all actuators
end loop

Actuator A · Actuator B · Actuator C

Sense-Compute-Control

- Typically used in structuring embedded real-time control applications (e.g. robotic control, automotive applications).

- Typically, clock-driven.

- Timely response is essential.

- Note that there is implicit feedback in such applications via the external environment.

Pipe-and-Filter
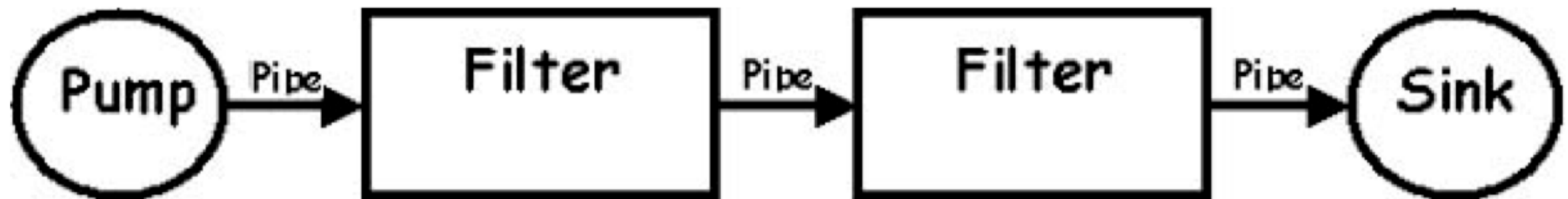
Also known as the data flow style.

Pipe-and-Filter

- Separate programs are executed, potentially **concurrently**; data is passed as a **stream** from one program to the next.

- Filters transform input data streams into output data streams.

- Pipes transmit outputs of one filter to inputs of another.

- Constraints

  - Filters are mutually independent and do not share state.

  - A standard input and output stream

- Benefits

  - Filters can be easily composed for a large variety of tasks.

- Example: the Unix shell

 - E.g. ls | grep "5555" | more

Implicit Invocation

- Instead of invoking a procedure directly, a component can announce (or broadcast) one or more events. Other components in the system can register an interest in an event by associating a procedure with the event. When the event is announced the system itself invokes all of the procedures that have been registered for the event. <u>Thus an event announcement ``implicitly'' causes the invocation of procedures in other modules</u>.

- Variations: Publish-Subscribe, Event-Based.

Implicit Invocation

- Usually requires the external support (e.g. operating systems, middleware, programming language features) to handle generation/notification of events.

- Constraints

  - Announcers of events do not know which components will be affected by those events.

- Benefits

  - The system is relatively easy to evolve (e.g. addition of new observers).

- Example

  - User interface development

The Blackboard Style

Blackboard (shared data)
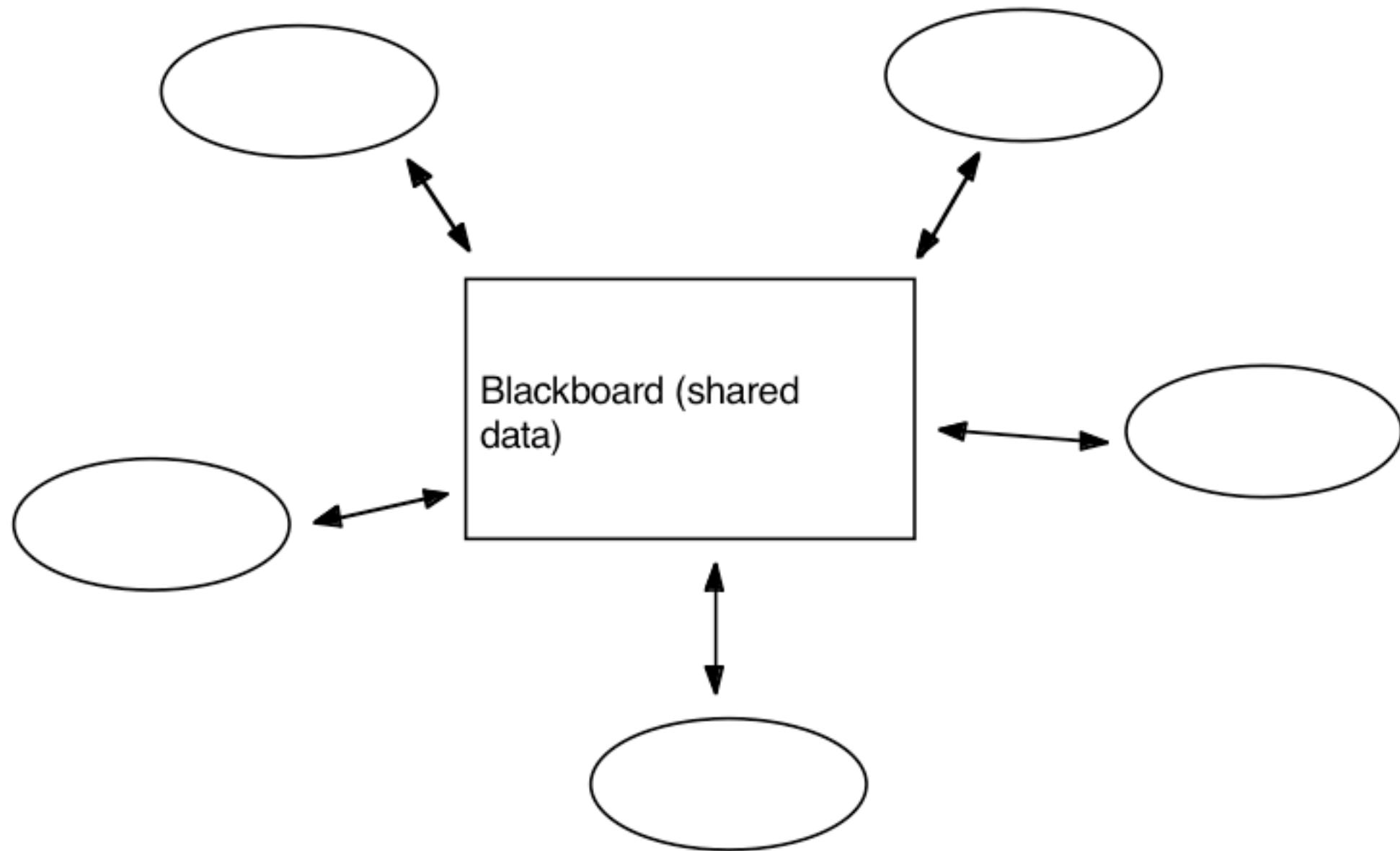
The Blackboard Style

- Two kinds of components

  - Central data structure.

  - A collection of independent components that operate on the central data.

- Constraints

  - The current state of the central data structure is the main trigger of selecting processes to execute.

- Benefits

  - Ease of adaptation, enhanced scalability

- Examples

  - AI systems

  - Compiler

Layered Styles

- An architecture is separated into ordered layers, and each layer exposes an interface to be used by above layers.

- Advantages

  - Changes in a layer affect at most the adjacent two layers.

  - Different implementations of layer are allowed as long as interface is preserved.

- Disadvantages

  - Performance

- Instances: virtual machine.
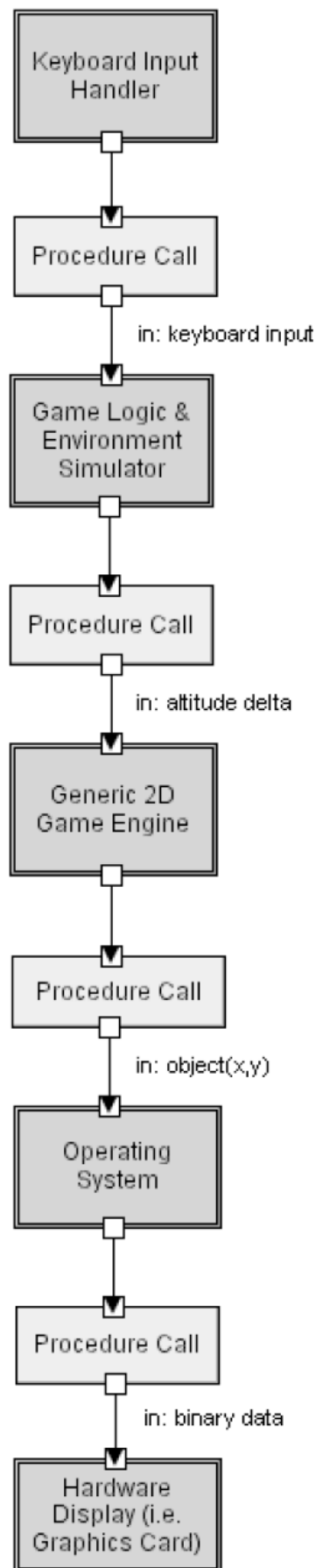
- A layer offers a set of services ("a machine with a bunch of buttons and knobs") that may be accessed by programs residing within the layer above it.

- In a strictly virtual machines style, programs at a given level may only access the services provided by the layer immediately below it.

- Benefits: clear dependence structure.

- Typical uses: network protocol stacks, database management systems.

Reference

- Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy. Software Architecture: Foundations, Theory, and Practice. John Wiley and Sons. ISBN-10: 0470167742; ISBN-13: 978-0470167748. 2010.