Lecture 21: Software Architectures

- Architectural Styles
  - Pipe and filter
  - Object oriented:
    - Client-Server; Object Broker
  - Event based
  - Layered:
    - Designing Layered Architectures
  - Repositories:
    - Blackboard, MVC
  - Process control

Analysis vs. Design

→ Analysis

- Asks "what is the problem?"
  - what happens in the current system?
  - what is required in the new system?
- Results in a detailed understanding of:
  - Requirements
  - Domain Properties
- Focuses on the way human activities are conducted

→ Design

- Investigates "how to build a solution"
  - How will the new system work?
  - How can we solve the problem that the analysis identified?
- Results in a solution to the problem
  - A working system that satisfies the requirements
  - Hardware + Software + Peopleware
- Focuses on building technical solutions

→ Separate activities, but not necessarily sequential

- …and attempting a design usually improves understanding of the problem

Software Architecture

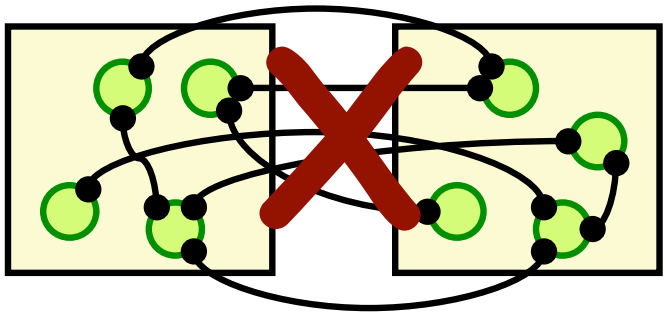
- A software architecture defines:
  - the components of the software system
  - how the components use each other's functionality and data
  - How control is managed between the components
- An example: client-server
  - Servers provide some kind of service; clients request and use services
  - applications are located with clients
    - E.g. running on PCs and workstations;
  - data storage is treated as a server
    - E.g. using a DBMS such as DB2, Ingres, Sybase or Oracle
    - Consistency checking is located with the server
  - Advantages:
    - Breaks the system into manageable components
    - Makes the control and data persistence mechanisms clearer
  - Variants:
    - Thick clients have their own services, thin ones get everything from servers
  - Note: Are we talking about logical (s/w) or physical (h/w) architecture?

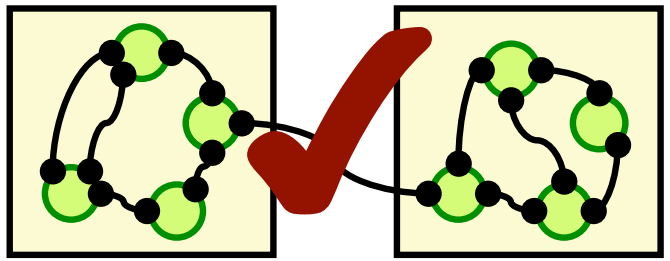
Coupling and Cohesion

→ Architectural Building blocks:

→ A good architecture:

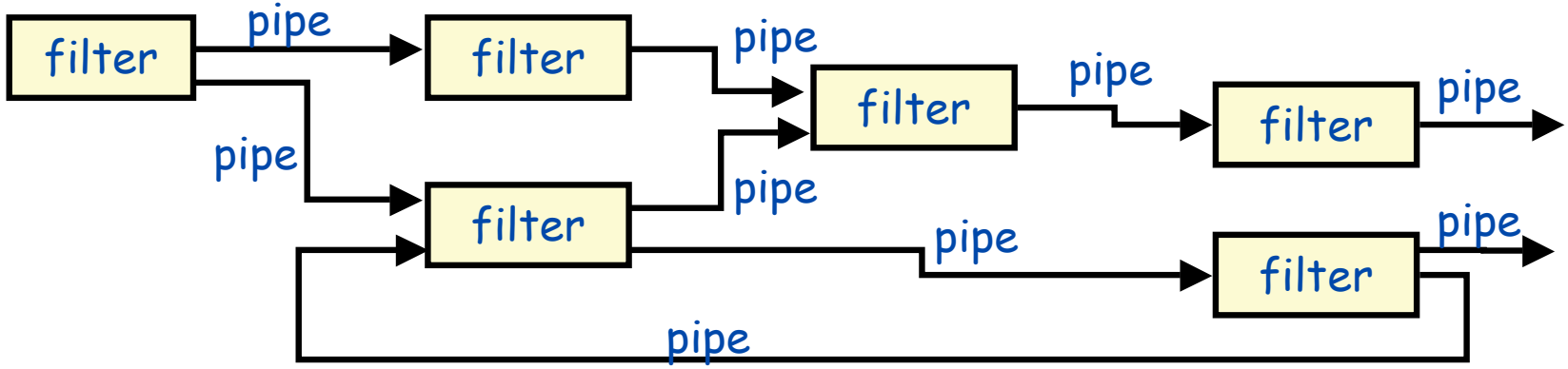
- Minimizes coupling between modules:
  - Goal: modules don't need to know much about one another to interact
  - Low coupling makes future change easier
- Maximizes the cohesion of each module
  - Goal: the contents of each module are strongly inter-related
  - High cohesion makes a module easier to understand





Pipe-and-filter

Source: Adapted from Shaw & Garlan 1996, p21-2. See also van Vliet, 1999 Pp266-7 and p279



→ Examples:

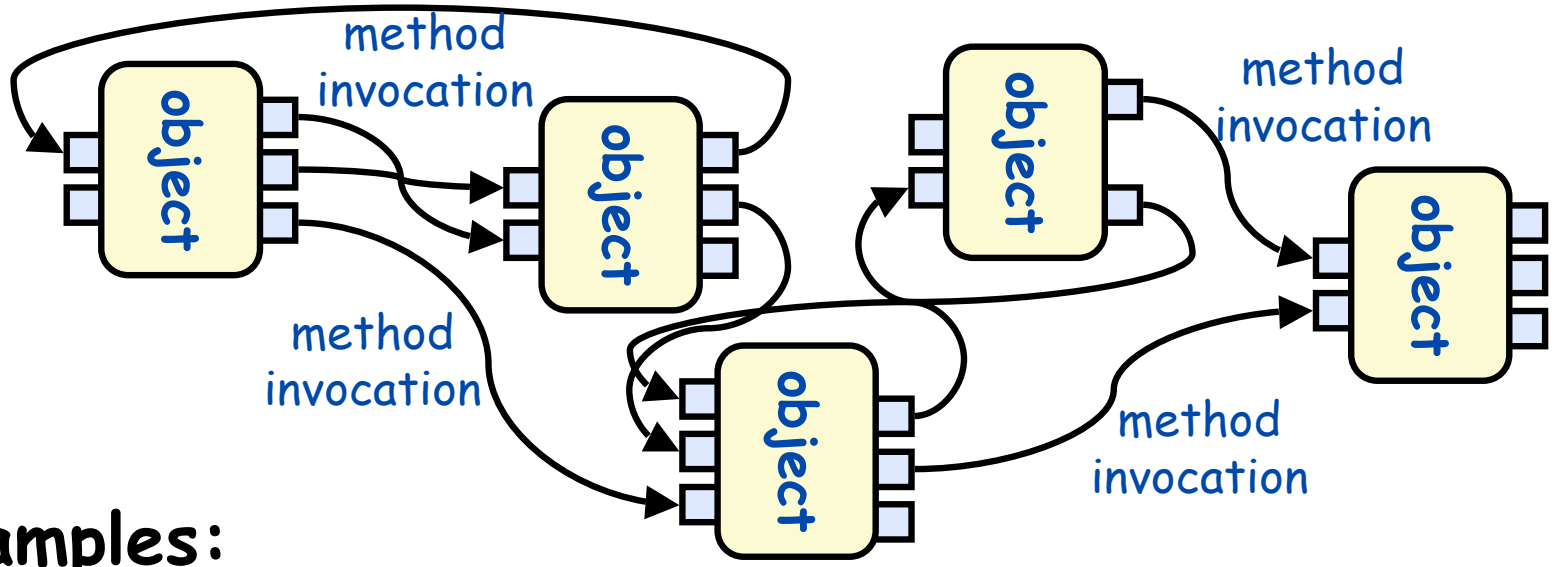
- UNIX shell commands
- Compilers:
  - Lexical Analysis -> parsing -> semantic analysis -> code generation
- Signal Processing

→ Interesting properties:

- filters don't need to know anything about what they are connected to
- filters can be implemented in parallel
- behaviour of the system is the composition of behaviour of the filters
  - specialized analysis such as throughput and deadlock analysis is possible

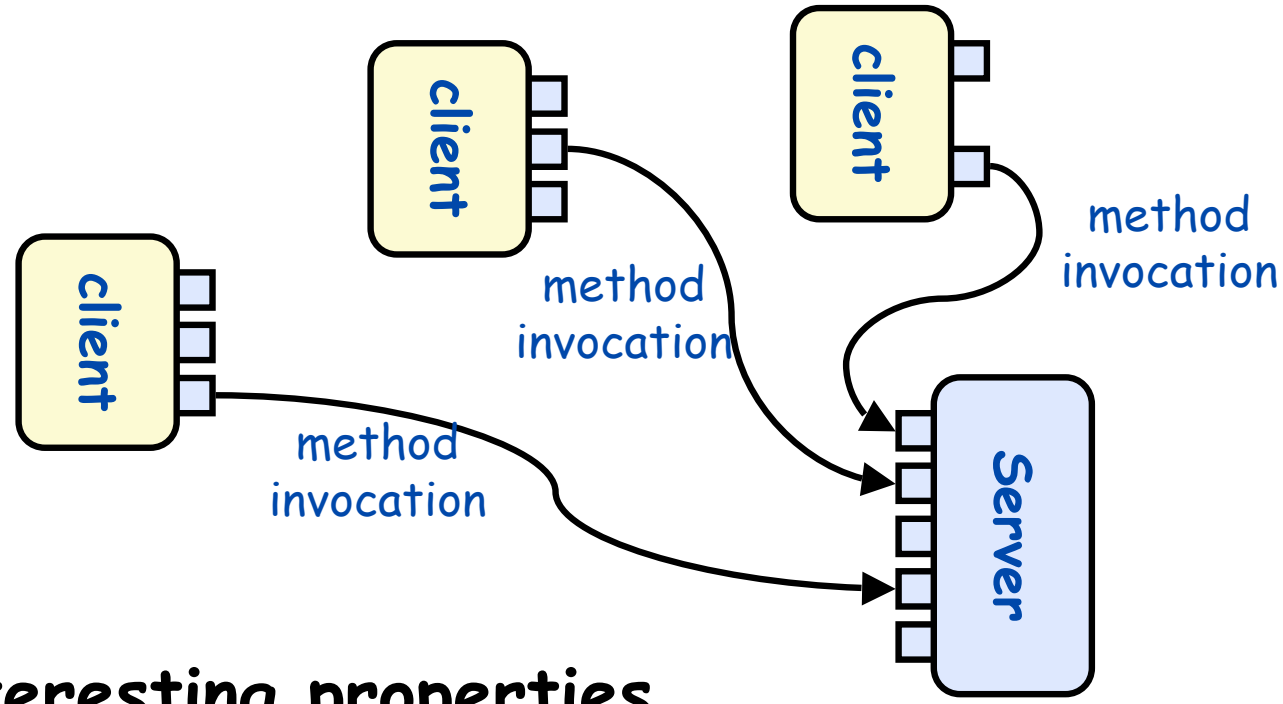
Object Oriented Architectures

Source: *Adapted from Shaw & Garlan 1996, p22-3.*



- Examples:
  - abstract data types
- Interesting properties
  - data hiding (internal data representations are not visible to clients)
  - can decompose problems into sets of interacting agents
  - can be multi-threaded or single thread
- Disadvantages
  - objects must know the identity of objects they wish to interact with

Variant 1: Client Server



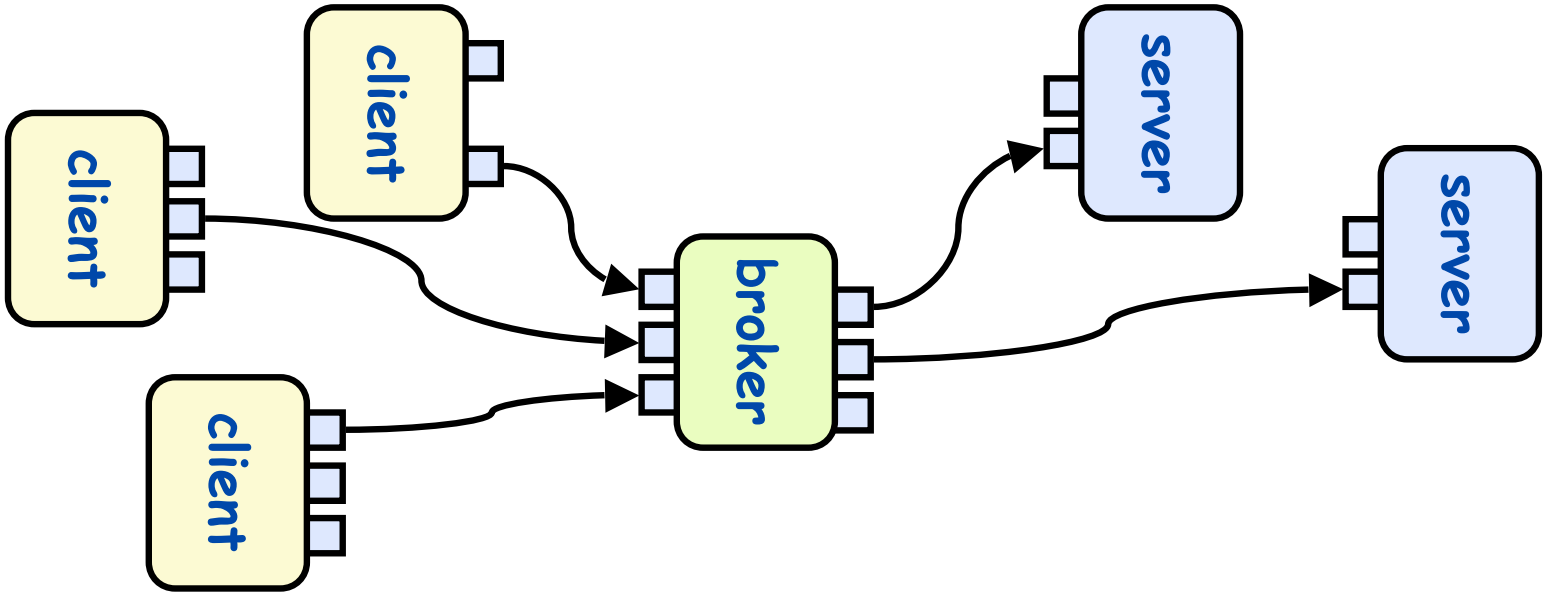
→ Interesting properties

- Is a special case of the previous pattern object oriented architecture
- Clients do not need to know about one another

→ Disadvantages

- Client objects must know the identity of the server

Variant 2: Object Brokers



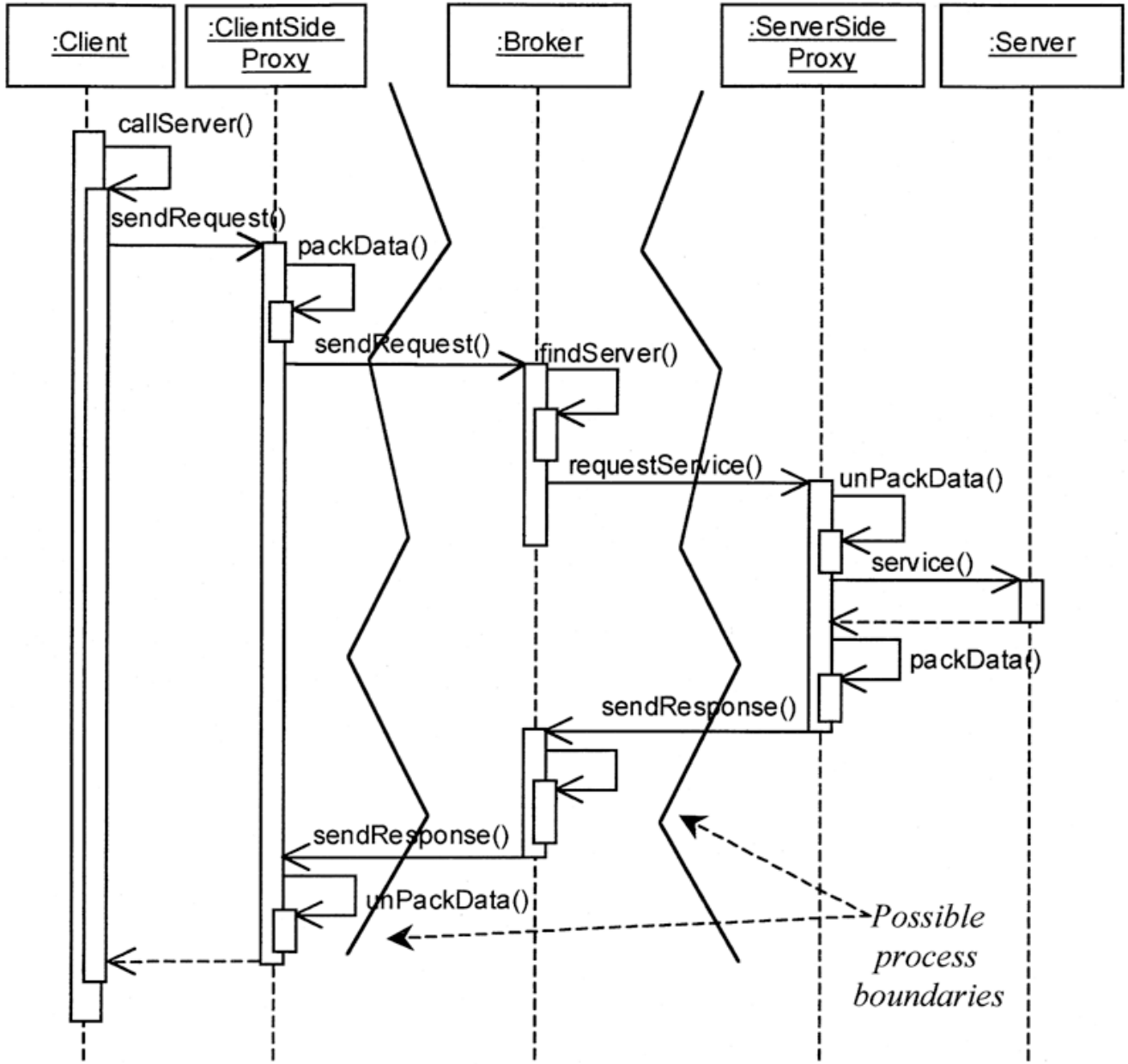
→ Interesting properties

- Adds a broker between the clients and servers
- Clients no longer need to know which server they are using
- Can have many brokers, many servers.

→ Disadvantages

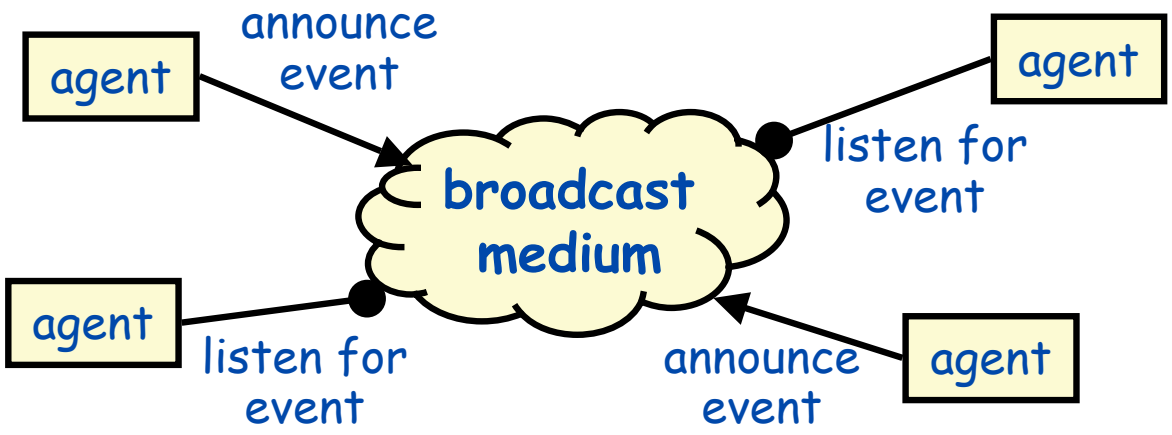
- Broker can become a bottleneck
- Degraded performance

Broker Architecture Example



Event based (implicit invocation)

Source: *Adapted from Shaw & Garlan 1996, p23-4. See also van Vliet, 1999 Pp264-5 and p278*



→ Examples

- debugging systems (listen for particular breakpoints)
- database management systems (for data integrity checking)
- graphical user interfaces

→ Interesting properties

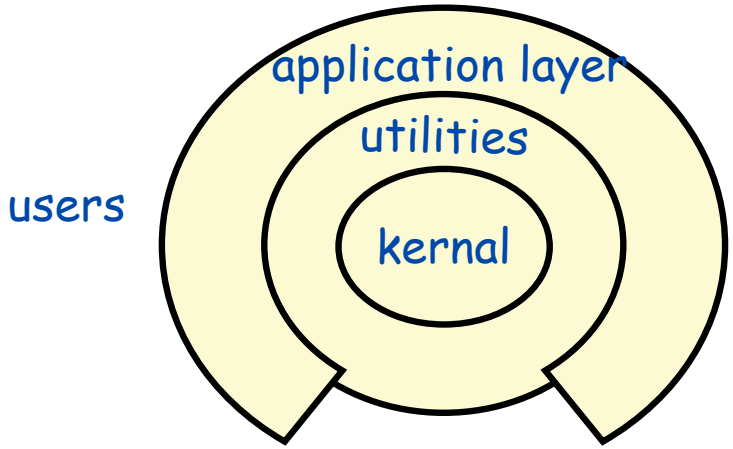
- announcers of events don't need to know who will handle the event
- Supports re-use, and evolution of systems (add new agents easily)

→ Disadvantages

- Components have no control over ordering of computations

Layered Systems

Source: *Adapted from Shaw & Garlan 1996, p25. See also van Vliet, 1999, p281.*



→ Examples

- Operating Systems
- communication protocols

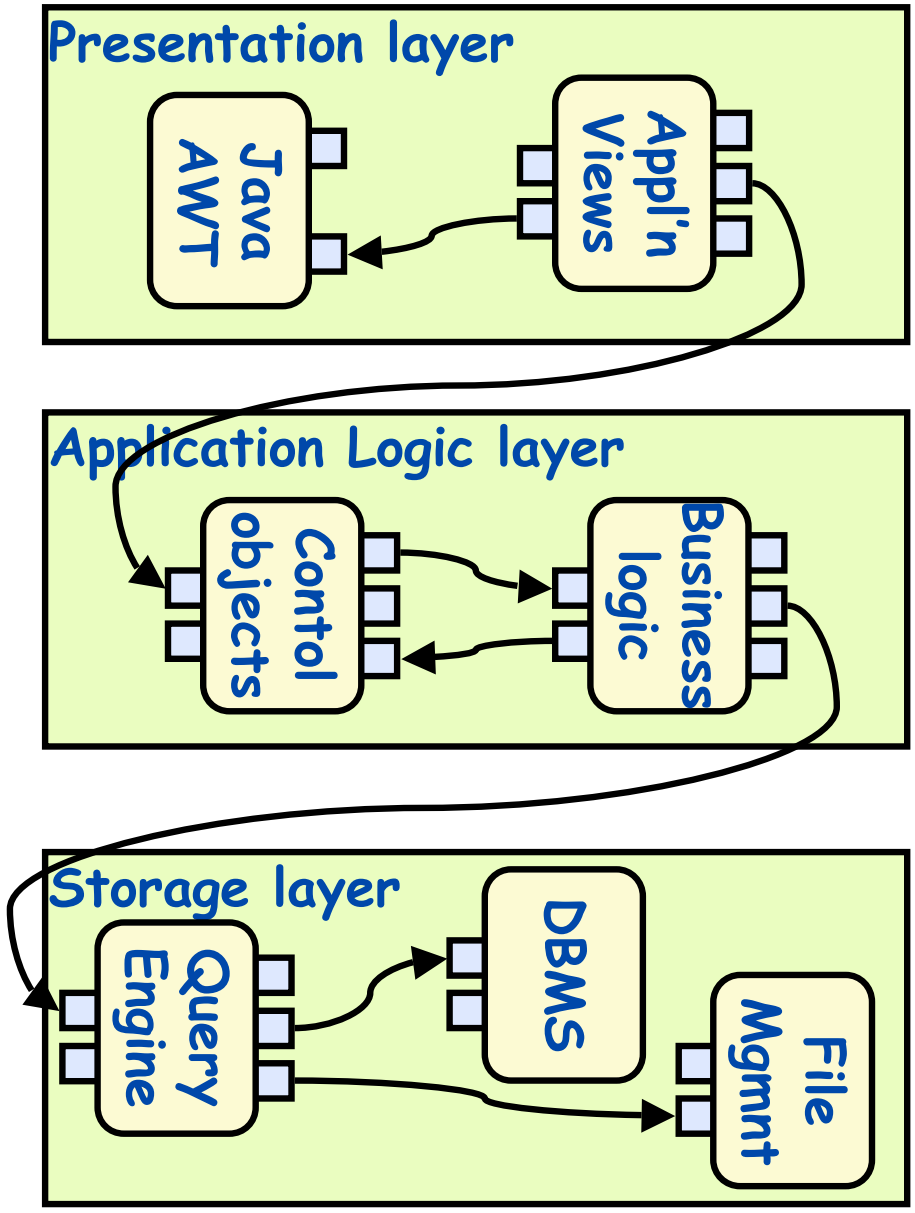
→ Interesting properties

- Support increasing levels of abstraction during design
- Support enhancement (add functionality) and re-use
- can define standard layer interfaces

→ Disadvantages

- May not be able to identify (clean) layers

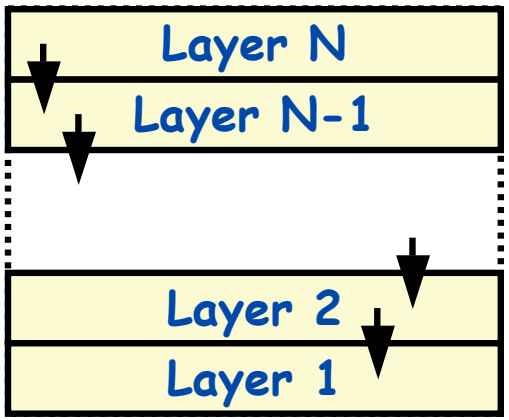
Variant: 3-layer data access



Open vs. Closed Layered Architecture

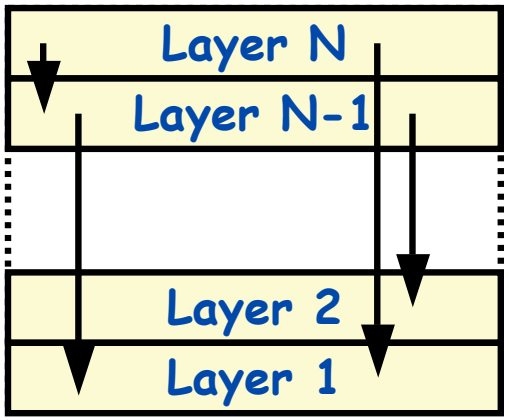
→ closed architecture

- each layer only uses services of the layer immediately below;
- Minimizes dependencies between layers and reduces the impact of a change.



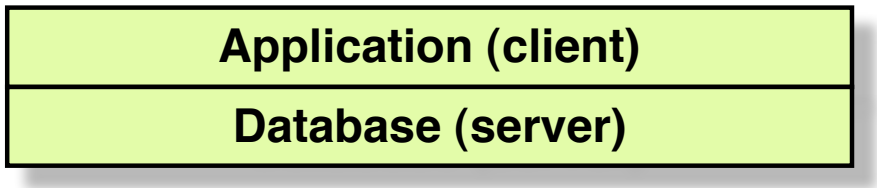
→ open architecture

- a layer can use services from any lower layer.
- More compact code, as the services of lower layers can be accessed directly
- Breaks the encapsulation of layers, so increase dependencies between layers

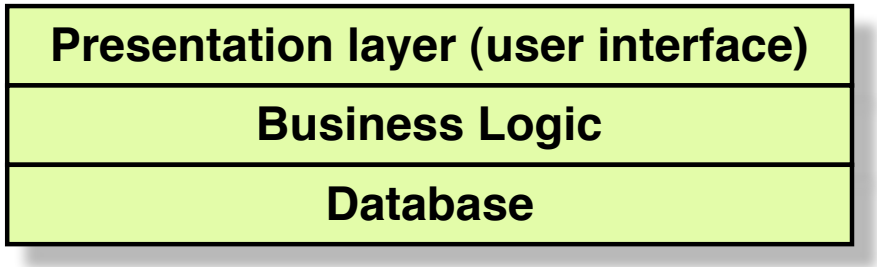


How many layers?

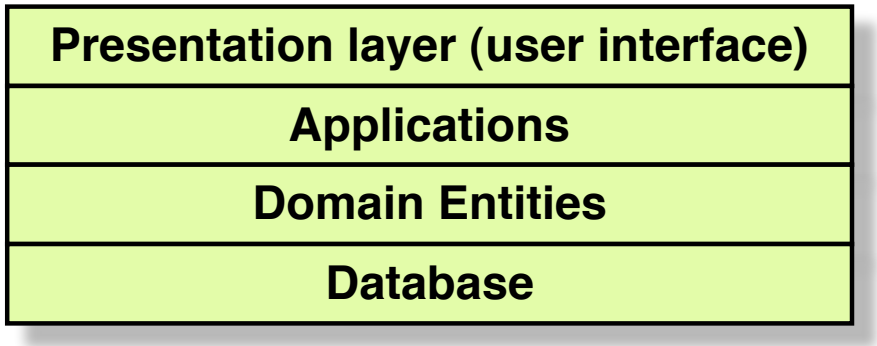
- 2-layers:
  - application layer
  - database layer
  - e.g. simple client-server model



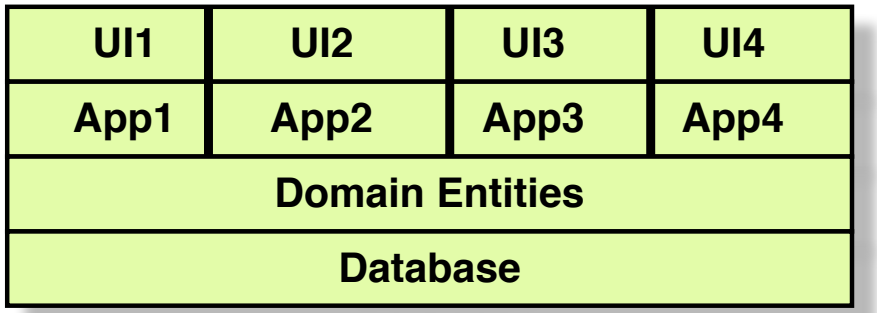
- 3-layers:
  - separate out the business logic
    - helps to make both user interface and database layers modifiable



- 4-layers:
  - Separates applications from the domain entities that they use:
    - boundary classes in presentation layer
    - control classes in application layer
    - entity classes in domain layer

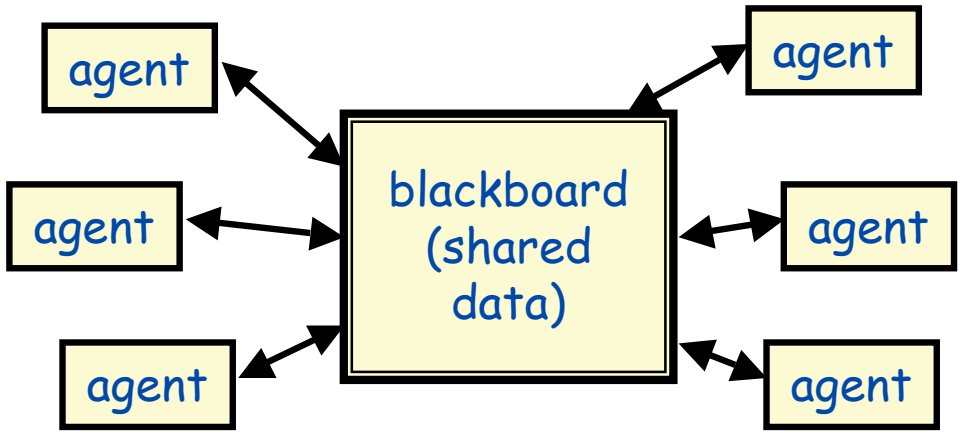


- Partitioned 4-layers
  - identify separate applications



Repositories

Source: *Adapted from Shaw & Garlan 1996, p26-7. See also van Vliet, 1999, p280*



→ Examples

- databases
- blackboard expert systems
- programming environments

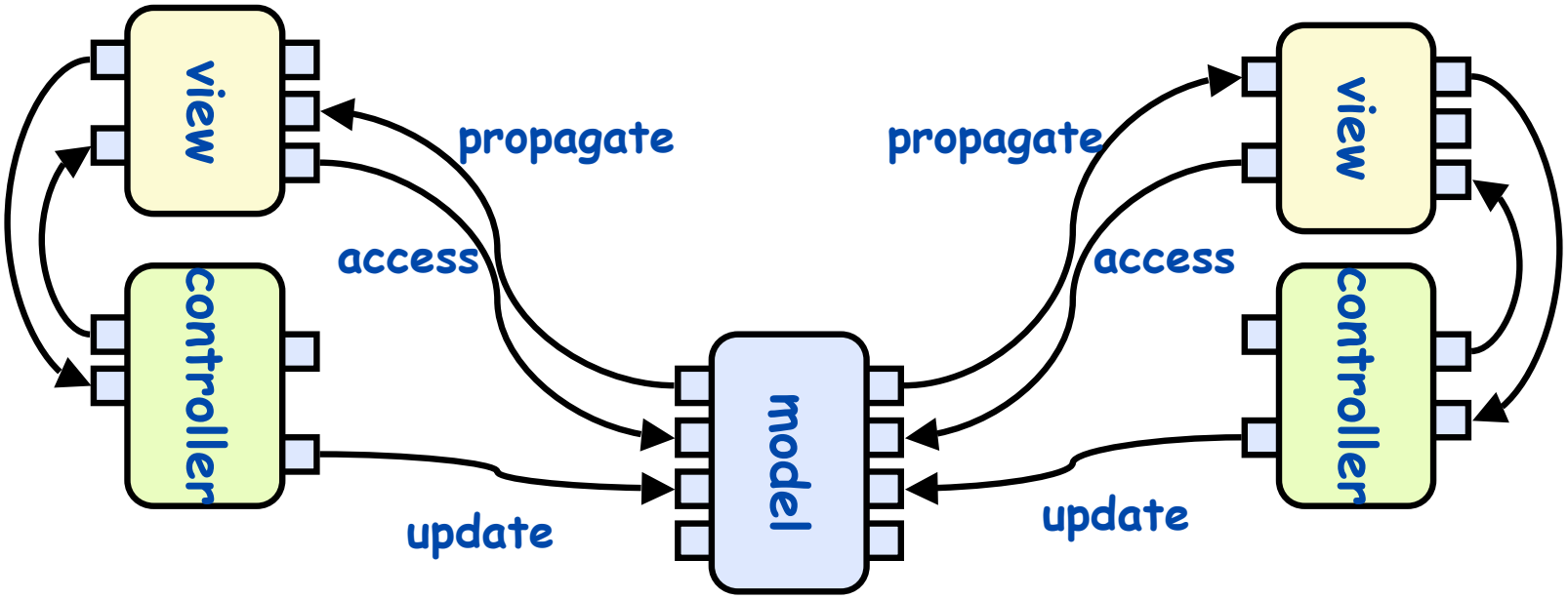
→ Interesting properties

- can choose where the locus of control is (agents, blackboard, both)
- reduce the need to duplicate complex data

→ Disadvantages

- blackboard becomes a bottleneck

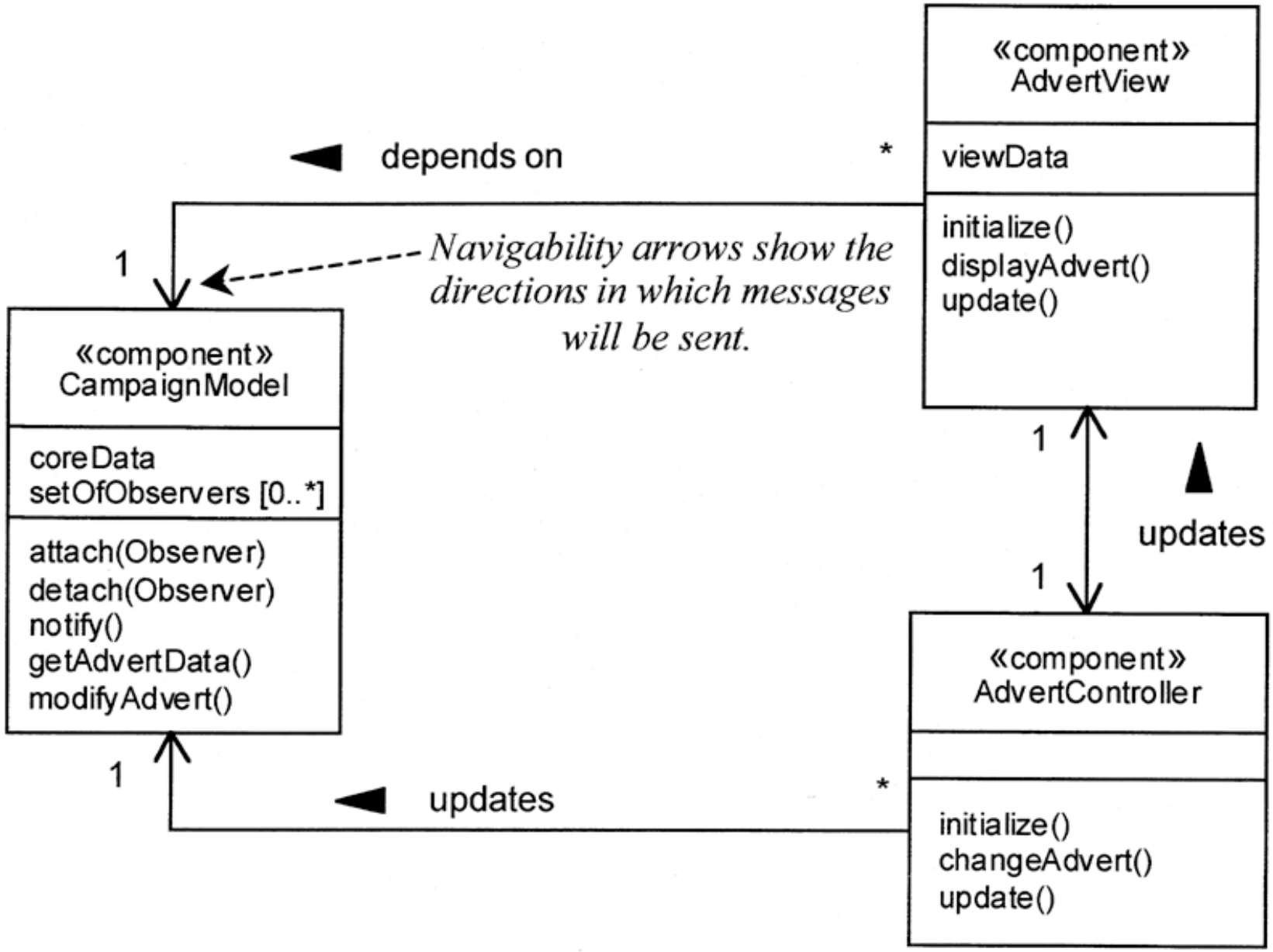
Variant: Model-View-Controller



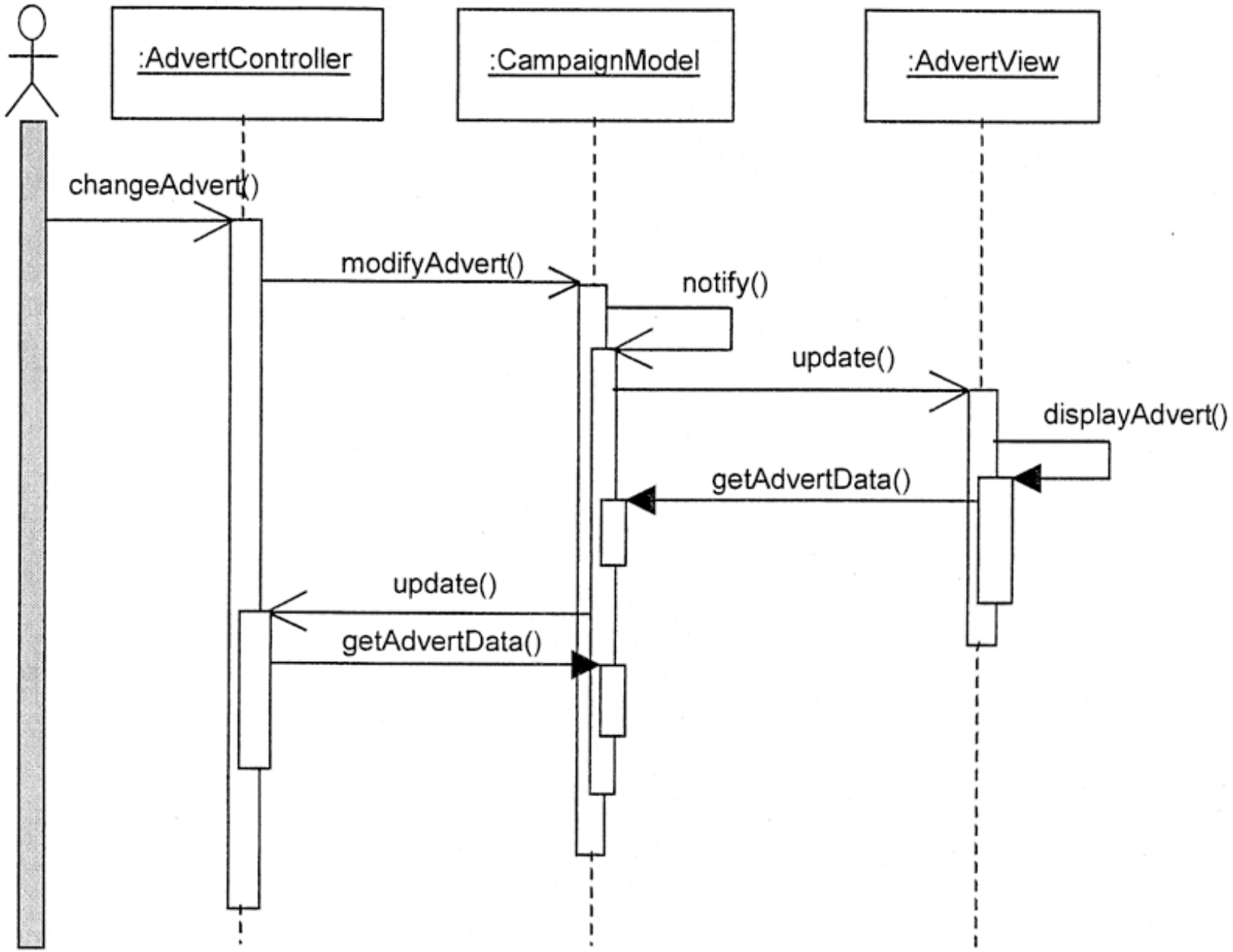
→ Properties

- One central model, many views (viewers)
- Each view has an associated controller
- The controller handles updates from the user of the view
- Changes to the model are propagated to all the views

Model View Controller Example

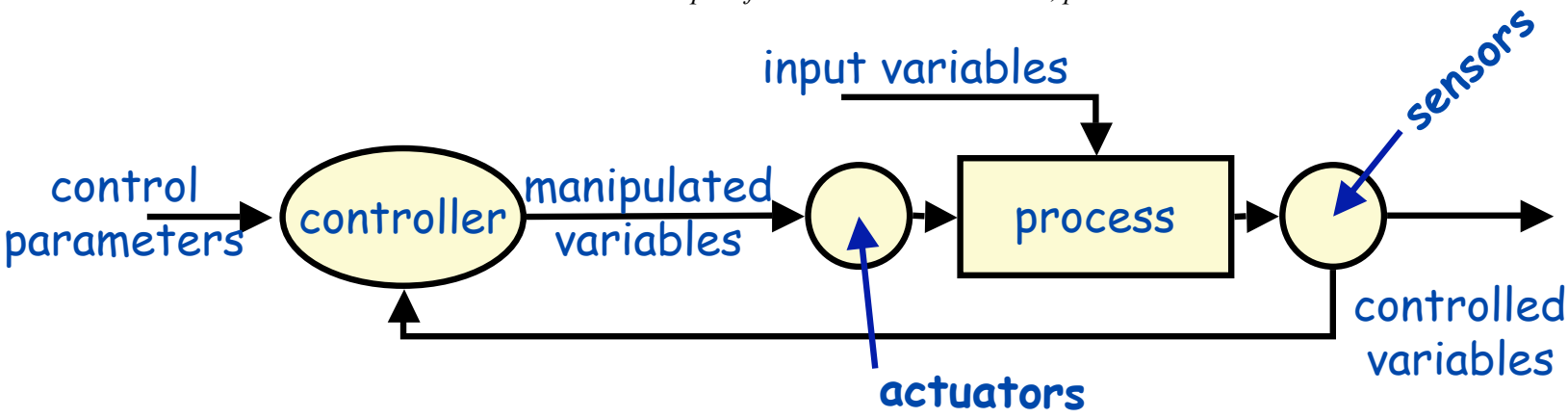


MVC Component Interaction



Process Control

Source: *Adapted from Shaw & Garlan 1996, p27-31.*



→ Examples

- aircraft/spacecraft flight control systems
- controllers for industrial production lines, power stations, etc.
- chemical engineering

→ Interesting properties

- separates control policy from the controlled process
- handles real-time, reactive computations

→ Disadvantages

- Difficult to specify the timing characteristics and response to disturbances