SOFTWARE ARCHITECTURE

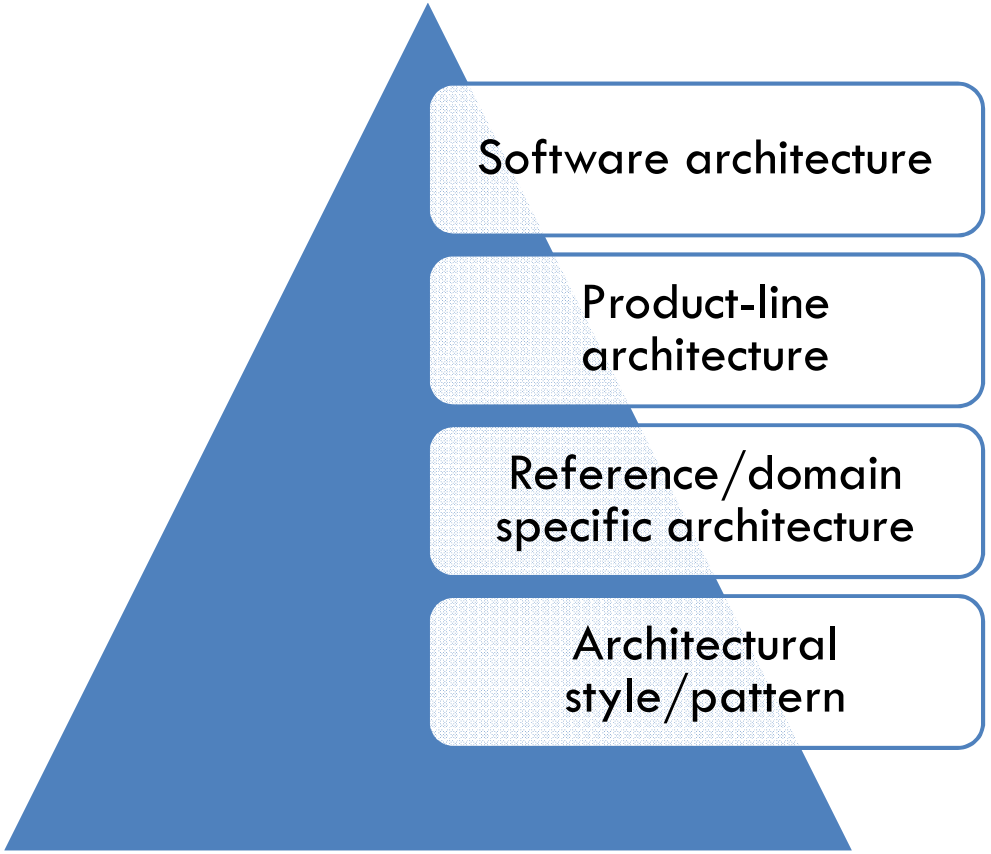
Lecture 4: Architecture Styles

Last Lecture

- How to approach quality attributes?
- Qualities achieved via design decisions
- What design decisions needed for achieving a specific quality?

- Architectural tactics:
  - availability, modifiability, performance, security, usability, testability

Lecture 1


Software architecture
Product-line architecture
Reference/domain specific architecture
Architectural style/pattern

Today Lecture

- Elements of architectural descriptions
- Software architecture patterns
  - Data Flow Systems
  - Call-and-Return Systems
  - Independent Components
  - Virtual Machines
  - Data-Centered Systems

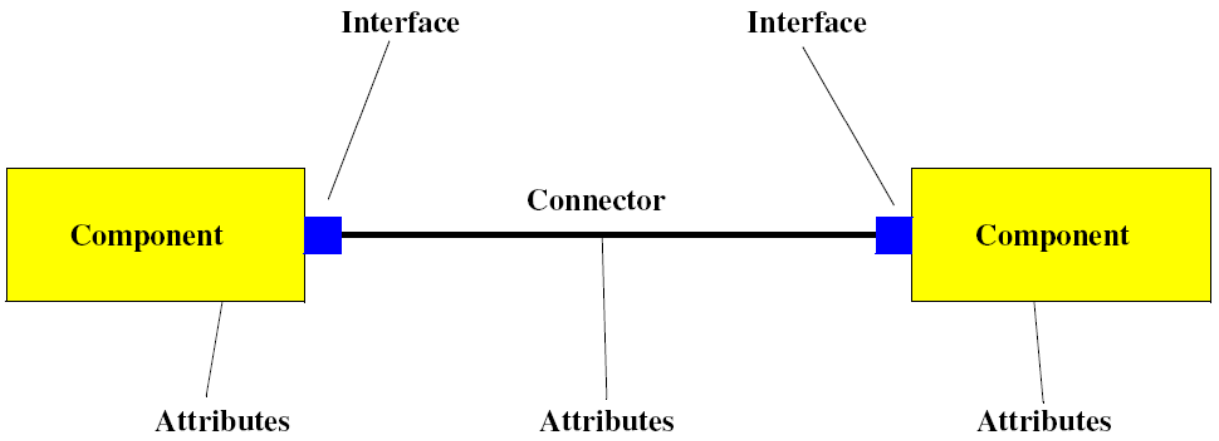
Architectural Descriptions

- An architectural description of a system includes:
  - Components
    - where the computation is performed:
    - objects, filters, databases
  - Connectors
    - performs the interaction among components:
    - procedure calls, pipes, event broadcast
  - Attributes
    - gives information for analysis and design:
    - signatures, pre/post conditions

Elements of Architectural Descriptions

Architectural Elements

Ref: Stephen H. Kaisler: Software Paradigms, Wiley, 2005.



Components

- Basic building blocks
  - active computational entities in a system
  - communicates with its environment through one or more ports or interfaces
- A port may be:
  - a user interface
  - a shared variable
  - a procedure name that can be called by another component,
  - a set of events that can be emitted
  - or some other mechanism

Attributes

- Attributes of the component specify information for analysis and development of the software:
  - protocols
  - real-time constraints
  - behavioral constraints

Connectors

- Connectors define the interaction between components.
- A connector
  - links the ports of two or more components
  - has a role attached to it as an attribute that describes the behavior of the attached components, e.g. source and receiver in an unidirectional communication.
  - can have other attributes:
    - rates
    - capacities
    - latencies

Interfaces

- A component
  - defines an interface through which links with another component
  - might have multiple interfaces
- An interface
  - is associated with one and only one component
  - may connect to multiple interfaces in other components
  - may have attributes, e.g.
    - direction of communication
    - buffering capacity
    - protocols accepted

Configurations

- A configuration (topology) is a connected graph of components and connectors describing architectural structure.
- The configuration must:
  - adhere to the rules for proper connectivity
  - satisfy concurrent and distributed properties
  - adhere to constraints such as design heuristics and style rules. e.g. end-to-end connectivity may not be longer than four components.
- Configuration is analyzed for:
  - performance
  - bottlenecks
  - concurrency
  - system reliability
  - security

Architecture Styles

- An architectural style defines a family of architectures constrained by:
  - Component/connector definitions
  - Topology rules
  - Semantic constraints

Software Architecture Patterns

- An architectural pattern expresses a fundamental structural organization schema for software systems.
- It provides:
  - A set of predefined subsystems
  - Specifies their responsibilities
  - Includes rules and guidelines for organizing the relationships between them

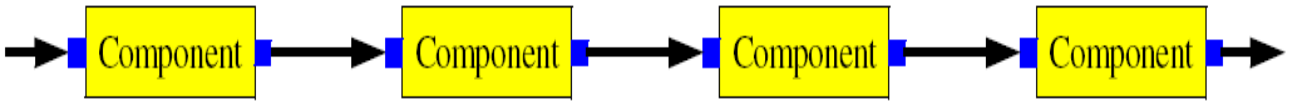
Software Architecture Patterns Classification

- Data flow systems
  - Batch sequential
  - Pipes and filters
- Call-and-return systems
  - Main program and subroutines
  - Client-server systems
  - Object-oriented systems
  - Hierarchical layers
- Independent components
  - Communicating processes
  - Event-based systems
- Virtual machine
  - Interpreters
  - Rule-based systems
- Data-centered systems
  - Database systems
  - Blackboards

A. Data Flow Systems

- The availability of data controls the computation process.
- Data centered: what to compute (not how).
- Conceptual elements:
  - process
  - file
  - data flow
- Data flows from one process to the next, through a series of processes that do operations on the data.
- Common forms:
  - Batch Sequential
  - Pipes and Filters

Data Flow Architecture



- Each component is isolated from performance of upstream component
  - Different speeds is taken care of by buffers; might lead to overflow error
- Components may be memoryless (no internal state preserved).
- Only memoryless components can be replaced at runtime

A.1. Batch Sequential Systems

- In Batch Sequential Systems
  - a number of programs are executed sequentially
  - each program runs to completion
  - data is transmitted between programs via files stored on external storage.
    - a program reads data from one or more files,
    - processes it,
    - and writes one or more files
  - after exiting the next program is started, if not the last one.
- Example: the UNIX pipe command

Batch Sequential Systems – Properties

- Low memory requirement
- High latency for I/O
- Limited concurrency
- No interactivity

A.2. Pipes and Filters Systems

- In Pipes and Filters Systems
  - data sets are processed in discrete, separable stages.
  - each stage is represented by a component.
  - each component operates on the entire data set and passes the results to next stage; the component acts as a filter.
  - connectors serve as links; the connector acts as pipes

Filters – Basics

- Basic activities:
  - enrich input data
    - add data, e.g. from a data store or computed values
  - refine input data
    - delete or sort data
  - transform input data
    - e.g. from one type to another

Filters – Types

- Two types of Filter:
  - Active Filter
    - drives the data flow
  - Passive Filter
    - driven by the dataflow
- In a Pipes and Filters Architecture at least one of the filters must be active
  - This active filter can be the environment, e.g. user input
  - If the input filter is active: A push system
  - If the output filter is active: A pull system

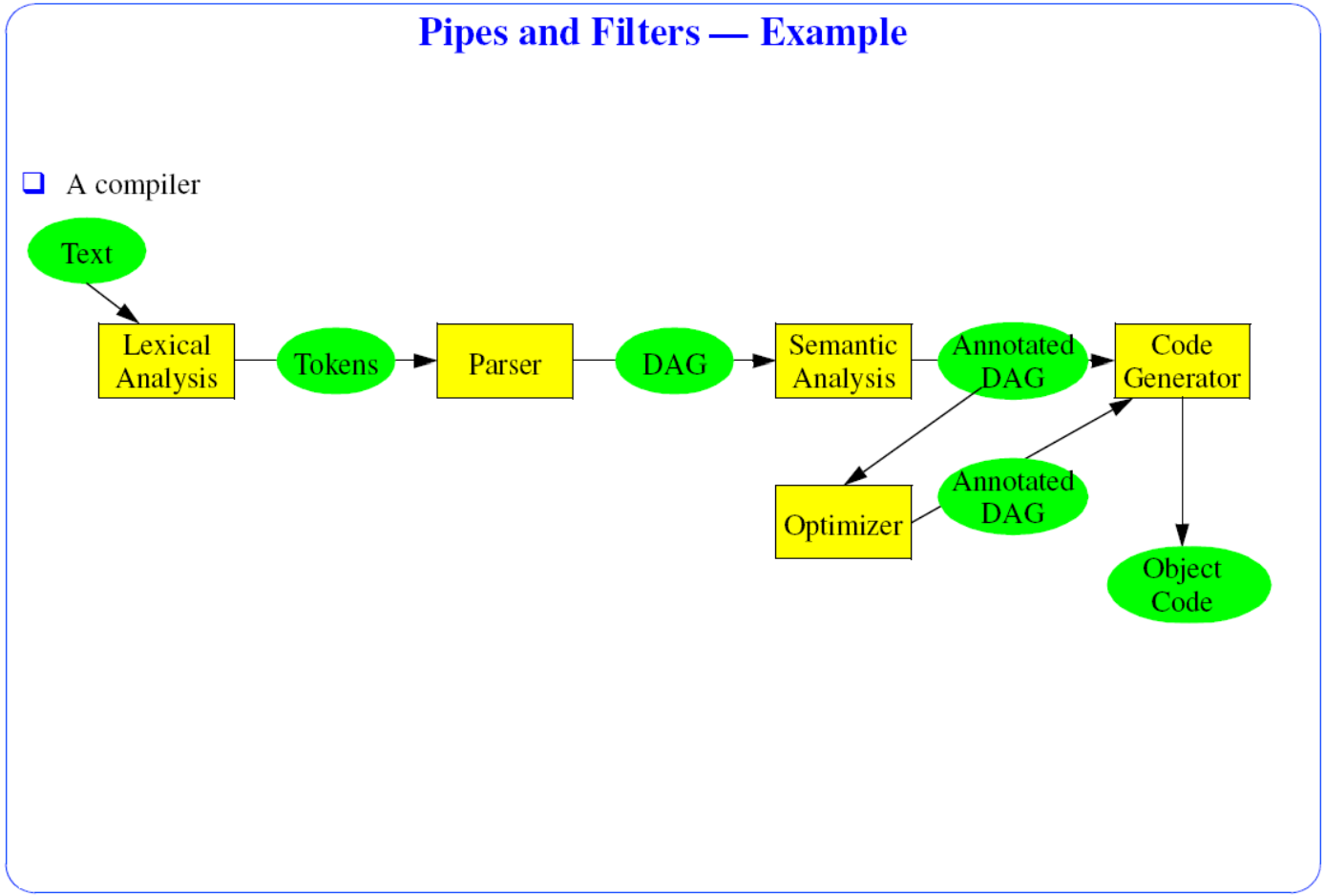
Pipes

- A Pipe
  - transfers data from one data source to one data sink
  - may implement a (bounded/unbounded) buffer
  - may connect
    - two threads of a single process
      - may contain references to shared language objects
    - two processes on a single host computer
      - may contain references to shared operating systems objects (e.g. files)
    - two processes on any host computer
      - distributed system

Pipes and Filters – Properties

- Pure data-driven interaction.
- Each component has a set of inputs and a set of outputs.
- Data transmitted as a whole between Filters.
- Filters are independent programs that can be recombined freely to build family of systems.
- Each transformation step is completed before the next step starts.
- Filters ignore identity of other filters.

Pipes and Filters – Example



Pipes and Filters – Other Examples

- Radar Tracking System
- Excel spreadsheets
- Shell scripts
- In Java: The Decorator Design Pattern for files

Pipes and Filters – Advantages

- Easy to understand system input/output.
- Support for reuse
  - Any two filters can be hooked together provided they agree on data transmission
- Easy maintenance:
  - new filters can be added
  - old filters can be replaced
- Throughput and deadlock analysis possible
- Natural support for concurrency
  - Different programs
  - No shared memory

Pipes and Filters – Disadvantages

- Filter ordering crucial.
- Multiple I/O:
  - difficult to maintain correspondence between two separate but related streams.
  - difference in arrival data rate might impose a performance bottleneck.

What are not Data Flow Systems

- File-sharing systems
- CVS system
- Word processing

B. Call and Return Systems

- Synchronous execution:
  - A component
    - calls another component
    - ceases execution and waits until the other component is ready
    - may get a return value
    - continues execution
  - is used when the order of computation is fixed
- Common forms:
  - Main program and subroutines
  - Client-server systems
  - Object-oriented systems
  - Hierarchical layers

B.1. Main Program and Subroutines

- A main program acts as the controller
- One or more subroutines perform specific functions when called by the main program
- Hierarchical decomposition
- Single thread of control
- Subroutines typically aggregated into subsystems
- Hierarchical reasoning
- The correctness of execution is dependant on the correctness of the main program and all subroutines that are called
- Old-fashioned, popular in the 70's to 90's

Advantages and Disadvantages

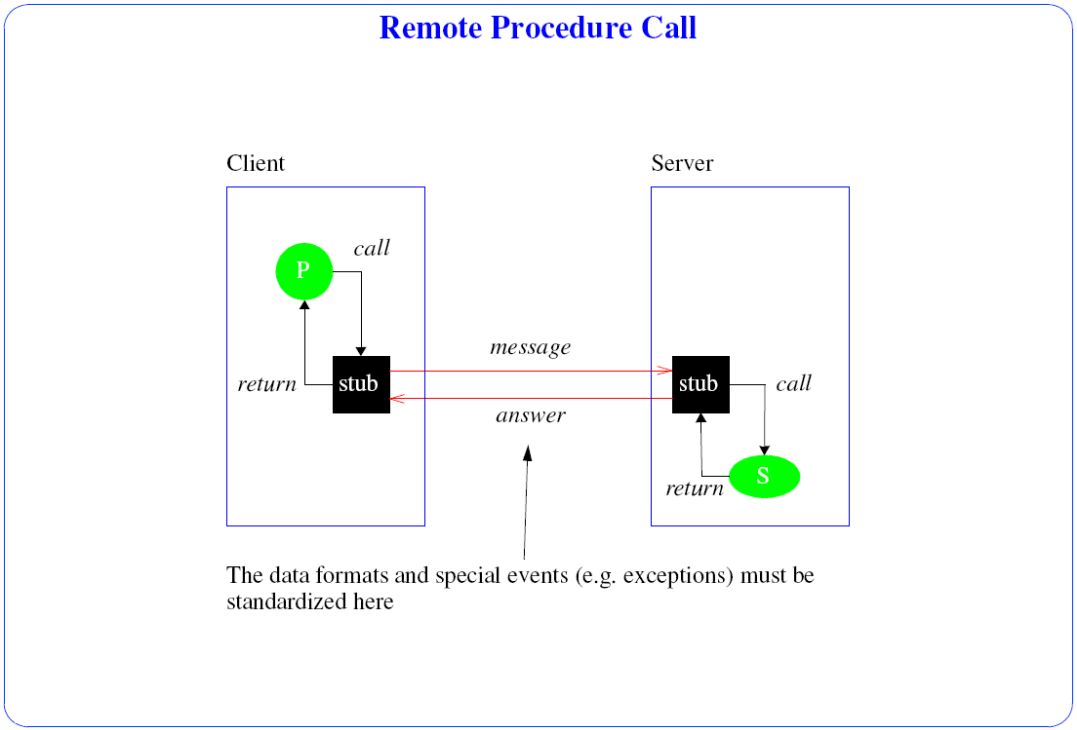
- Advantages:
  - Simple to learn and understand
- Disadvantages:
  - Correctness depends on the subroutines that are called
  - Tends to have different subroutines doing the same thing, "spaghetti code"
  - Does not scale easy to bigger systems

B.2. Client-Server Systems

- A Server provides different services.
- A Client uses the services as (remote) subroutines from one ore more Servers.
- The Server might be a Client to another Server.
- A variant of the Main Program and Subroutines, but the clients and servers
  - are implemented as separate entities (processes),
  - might be situated at different processors, and
  - can be implemented in another style (most often some variant of object oriented style)

Remote Procedure Call

- The mechanism for calling a server is called Remote Procedure Call, RPC.
  - Requires a protocol.



Client-Server Systems – Variants

- Fat Client
  - more functionality placed in the client.
- Fat Server
  - more functionality placed in the server.
- There can be many versions of this:
  - all business services in the client
  - part of business services in the client
  - only GUI in client.

Client-Server Systems – States

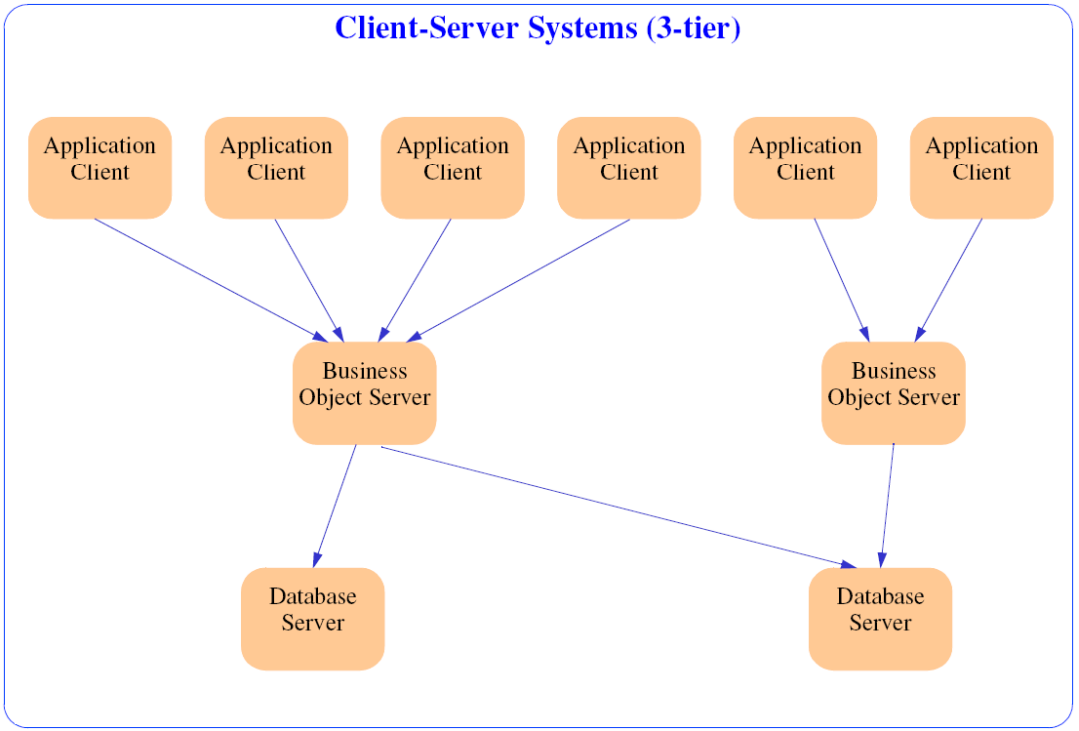
- Stateful:
  - Information about the Client state is maintained by the Server
    - Might be useful if there is multiple messages sent to the server.
    - The information in messages can be reduced.
    - Security might be easier to implement.
    - But can lead to complex protocols and error handling.
- Stateless:
  - Information about the Client state is not maintained by the Server
    - The normal solution if there is only service requests that are independent of each other and only use a single message each
    - Simple protocols and error handling must be performed by the client
    - If it is used for multiple dependant requests the server must rely on the client
    - Security problem. e.g. NFS protocol.

Client-Server Systems – Issues

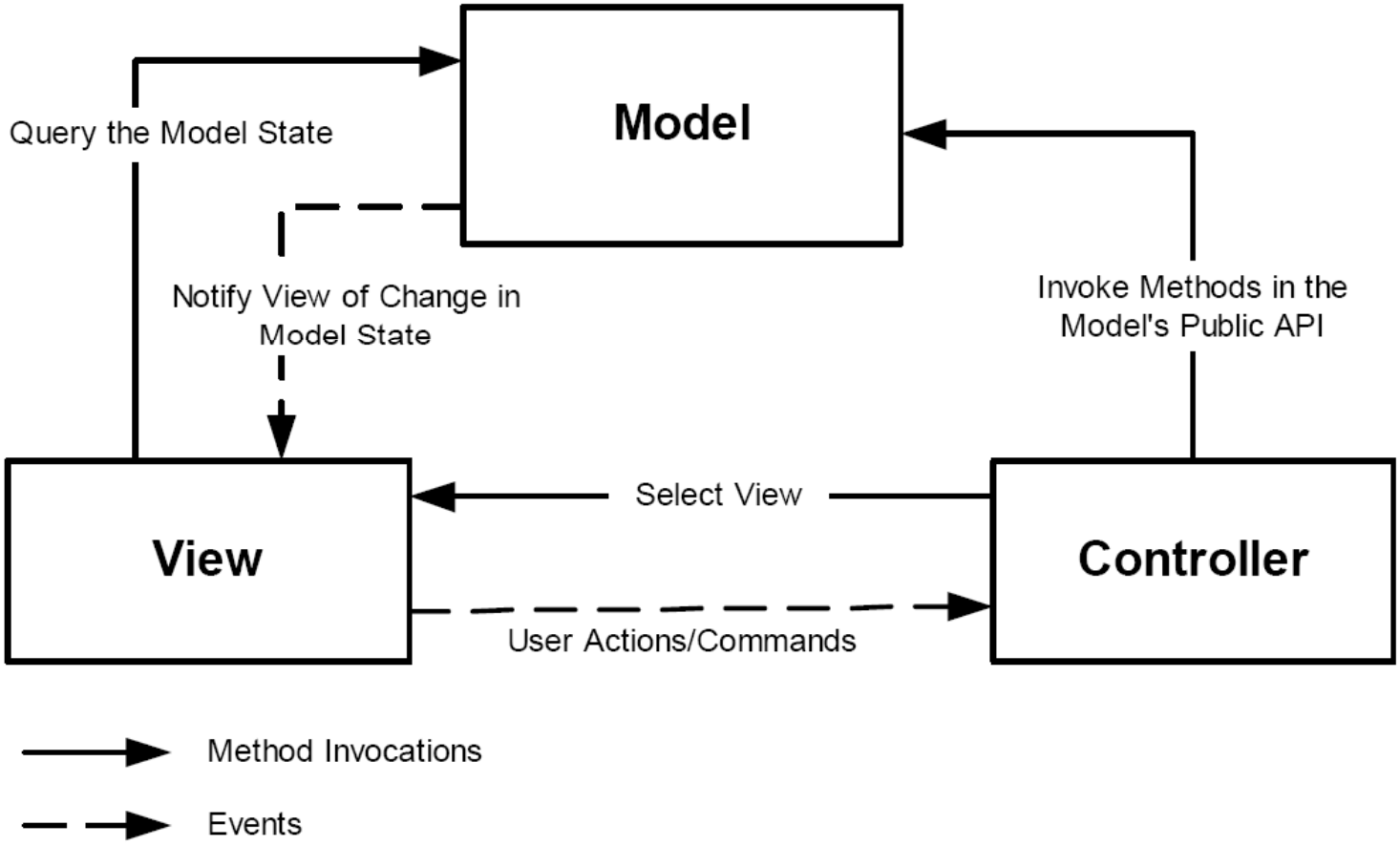
- When designing client-server systems the following must be considered:
  - Authentication:
    - Verifying the identity of the client and the server
  - Authorization:
    - Verifying the rights/privileges of the client
  - Data security:
    - Protect the data stored on the server
  - Protection:
    - Protect the server from malfunctioning clients
  - Middleware:
    - How to connect the clients to the server

Client-Server Systems – 3-tiers

- 3-tier: Functionality divided into three logical partitions:
  - Presentation services
  - Business services
  - Data services

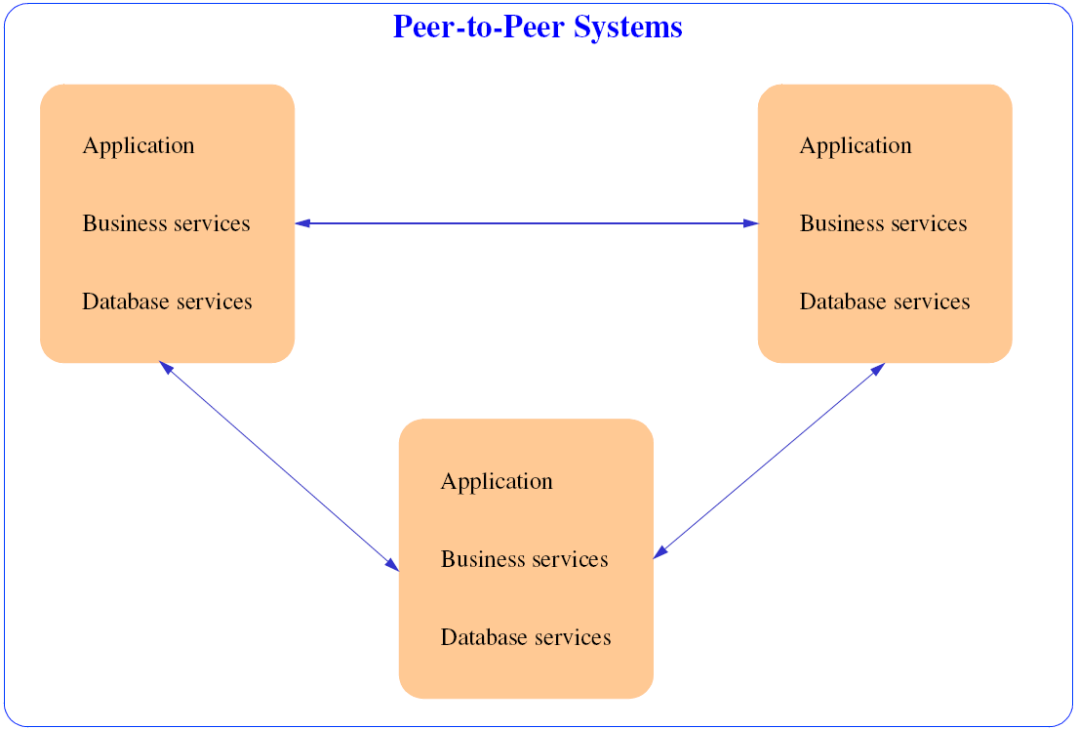


Model-View-Controller (MVC) Systems



Peer-to-Peer Systems

- Peer-to-Peer Systems, P2P are
  - like Client-Server but all processes can be both clients and servers
  - more flexible but also more complicated
  - there might occur:
    - deadlock
    - starvation



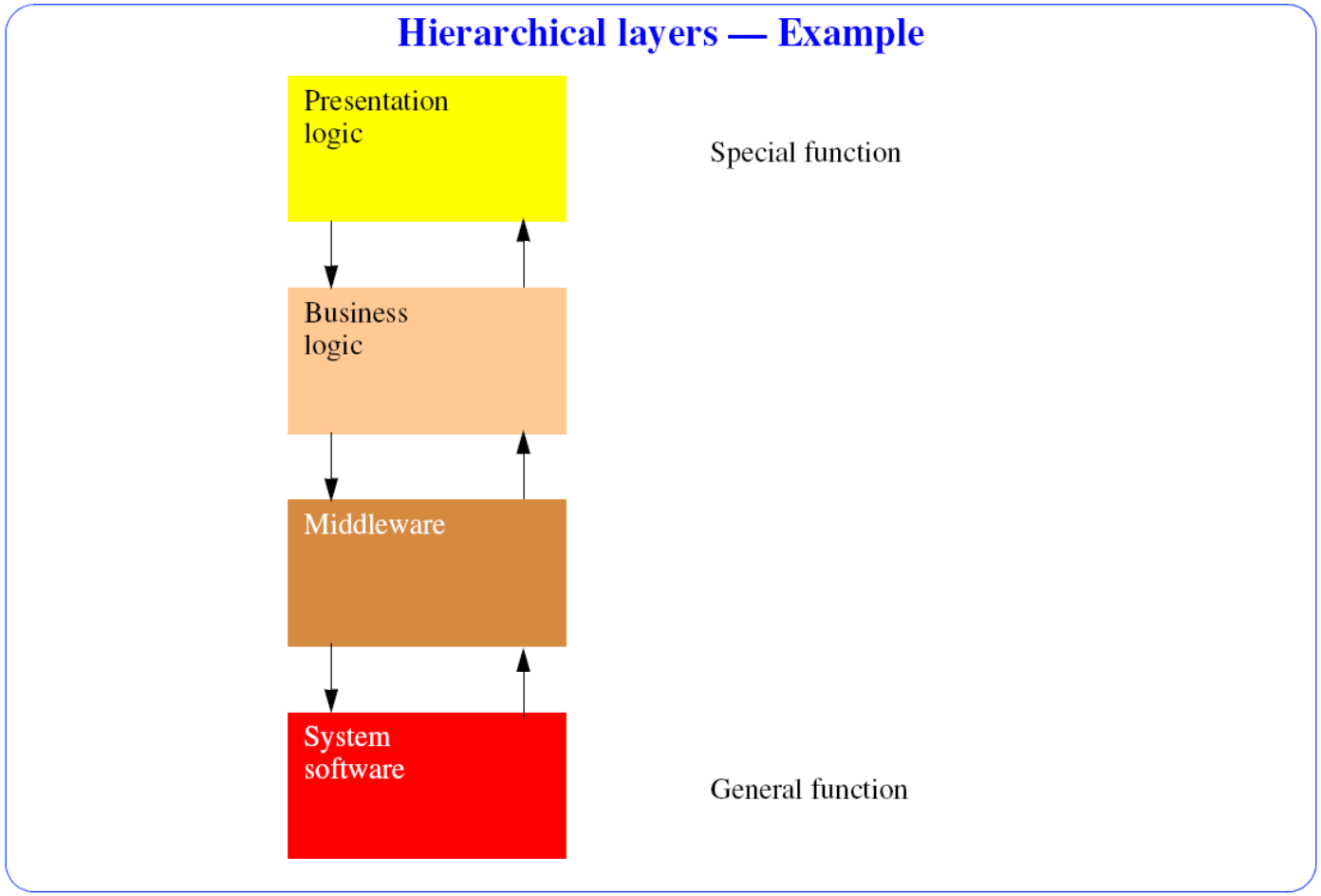
B.3. Object-Oriented Systems

- Special case of Main Program and Subroutines
- Encapsulation
  - Restricted access to information
  - This is the most important quality of O-O!
- Structure that follows the Problem Domain
  - Natural understanding of the system
- Dynamic binding
  - Actual operation to call is invoked at runtime
- Inheritance
  - Share definition of functionality

B.4. Hierarchical Layers

- A hierarchically layered system is partitioned into layers that are put above each other.
- Each layer act as
  - a virtual machine for the layers above it
  - a client to the layers below it
- Communication between the layers occurs through well-defined APIs.
- Special useful for Large Systems that can have several layers

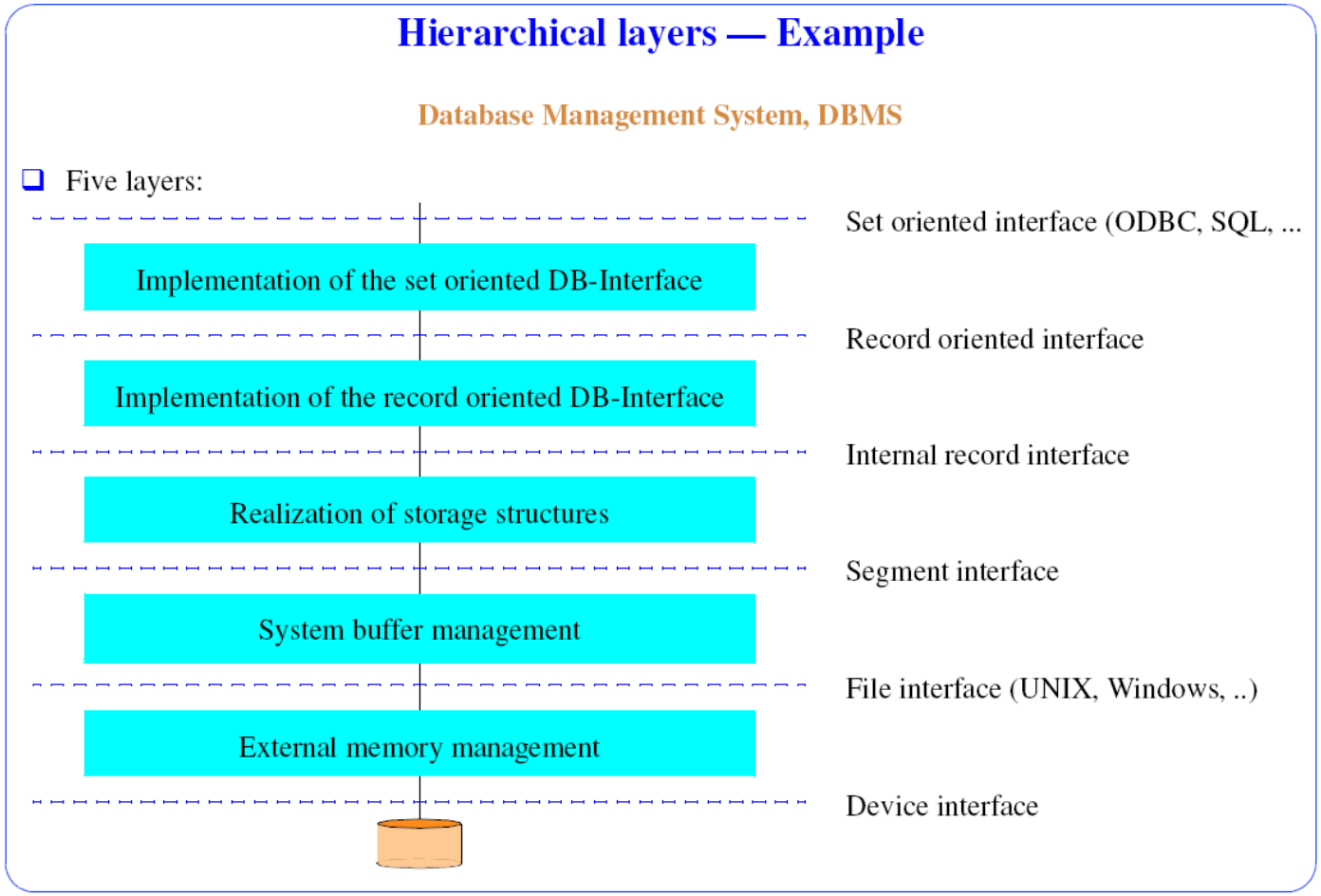
Hierarchical Layers – Variant



Hierarchical Layers – Considerations

- Visibility
  - Dependencies only within current layer and the layer below.
- Volatility
  - Upper layer affected by requirement changes.
  - Lower layers affected by environment changes.
- Generality
  - More abstract model elements in lower layers
- Number of layers
  - Small system: 3-4 layers
  - Complex system: 5-7 layers

Hierarchical Layers – Example



Advantages and Disadvantages

- Advantages
  - Supports designs based on increasing levels of abstraction
    - Suitable design method
  - Supports maintenance:
    - Only interaction with closest layers
  - Supports reuse:
    - Possibility to standard layer APIs
- Disadvantages
  - Not all systems are appropriate to structure in levels
  - Even, if so, it might be good for performance to have closer coupling with the basic functions
    - e.g. the ISO's OSI had too many layers to work well in real-time applications

C. Independent Components

- Event Based Systems
  - Processing is triggered by events.
  - An event might be:
    - A user clicking an icon or graphical object.
    - A signal from a sensor triggered by some physical event.
    - A signal from another system.
    - A signal from a timer:
      - time-out
      - scheduled events
    - An alarm going off.
  - There is no flow control.

Event-Based Systems

- Implicit Invocation.
  - Instead of invoking a procedure directly:
    - A component can announce one or more events
    - Other components can register for listening to events
    - When an event occurs all the registered procedures for it will be invoked
- Components:
  - Interface defines a set of incoming calls.
  - Interface defines a set of outgoing events.
- Order of invocations is non-deterministic.
- Model of computation:
  - Reactive systems, the computer reacts to interaction with the environment.
  - Algorithmic systems, algorithms decide the interaction with the environment.

Event-Based Systems – Advantages

- Suited for
  - user input
  - network communication
  - inputs from other physical events
- Strong support for evolution and reuse:
  - Any component can be introduced by registering to events.
  - Components may be replaced without affecting the rest of the system.

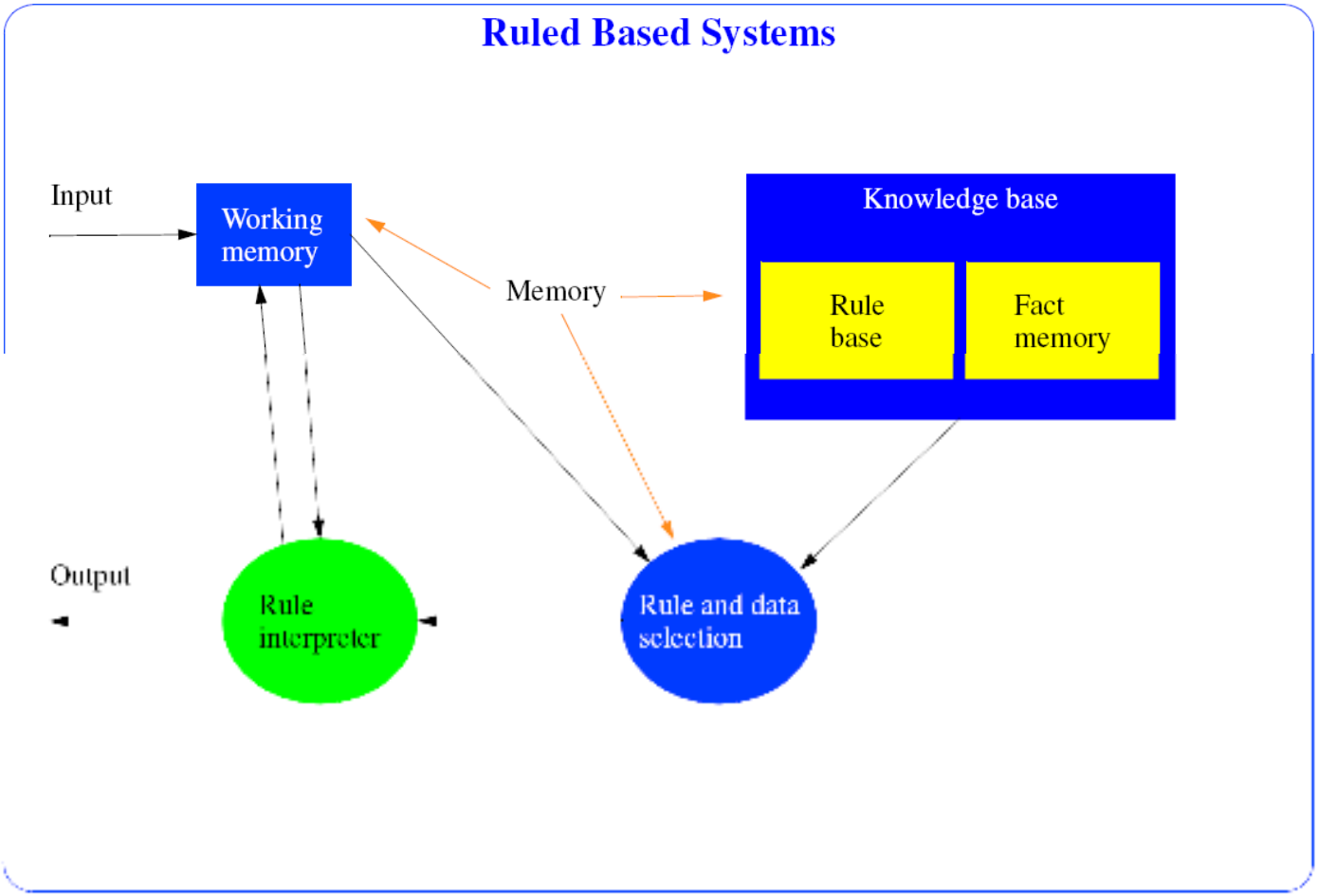
Event-Based Systems – Disadvantages

- There is no central knowledge of what is performed by the system.
  - non-deterministic invocation of components.
  - This is in a way a good thing but:
    - Components don't know the order of invocations.
    - There might have to be introduced mutual exclusion.
      - Complicated programming
      - Possible deadlocks
      - Performance bottlenecks

D. Virtual Machines

- A virtual machine is an emulator that executes on top of a hardware or software platform
  - Gives the same services as a given hardware but using software rather than hardware
- Examples:
  - Java VM
  - PC emulator on (old) MAC
  - Ruled Based Systems
    - Letting machines solve problems using human expert rules
    - Problem-solving as sets of situation-action rules
    - Execution sequenced in response to the conditions of the computation
    - Needs an interpreting system to execute

Rule-Based Systems



Advantages and Disadvantages

- Advantages
  - Provides complete protection of system resources
  - Still permits sharing of resources
- Disadvantages
  - A virtual machine is difficult to implement
  - Must follow a complicated specification
    - e.g. emulating Windows on a Linux machine

E. Data-Centered Systems

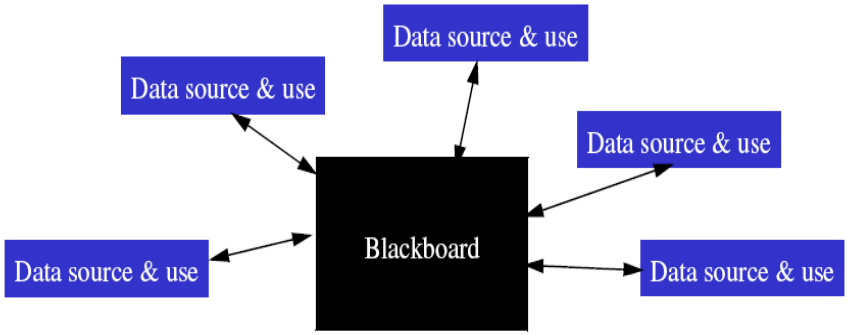
- Data-centered systems use a central data store to store all problem-related information.
  - Database
  - Blackboard
  - Repository

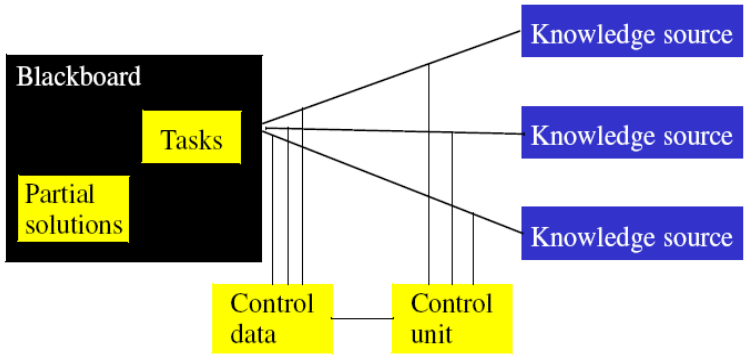
Blackboard Systems

- Allows multiple processes to communicate by reading and writing information and requests to a global data store.
- Each participating process has expert knowledge in its own field and some knowledge of how to solve the problem. A problem can not be solved by one process alone.
- Processes communicate strictly through the common blackboard whose content is visible to all processes.
- A control unit is responsible for selecting an appropriate process to solve it.

Blackboard Systems – Design

A Blackboard is a database into which a process can insert, retrieve, or remove data.

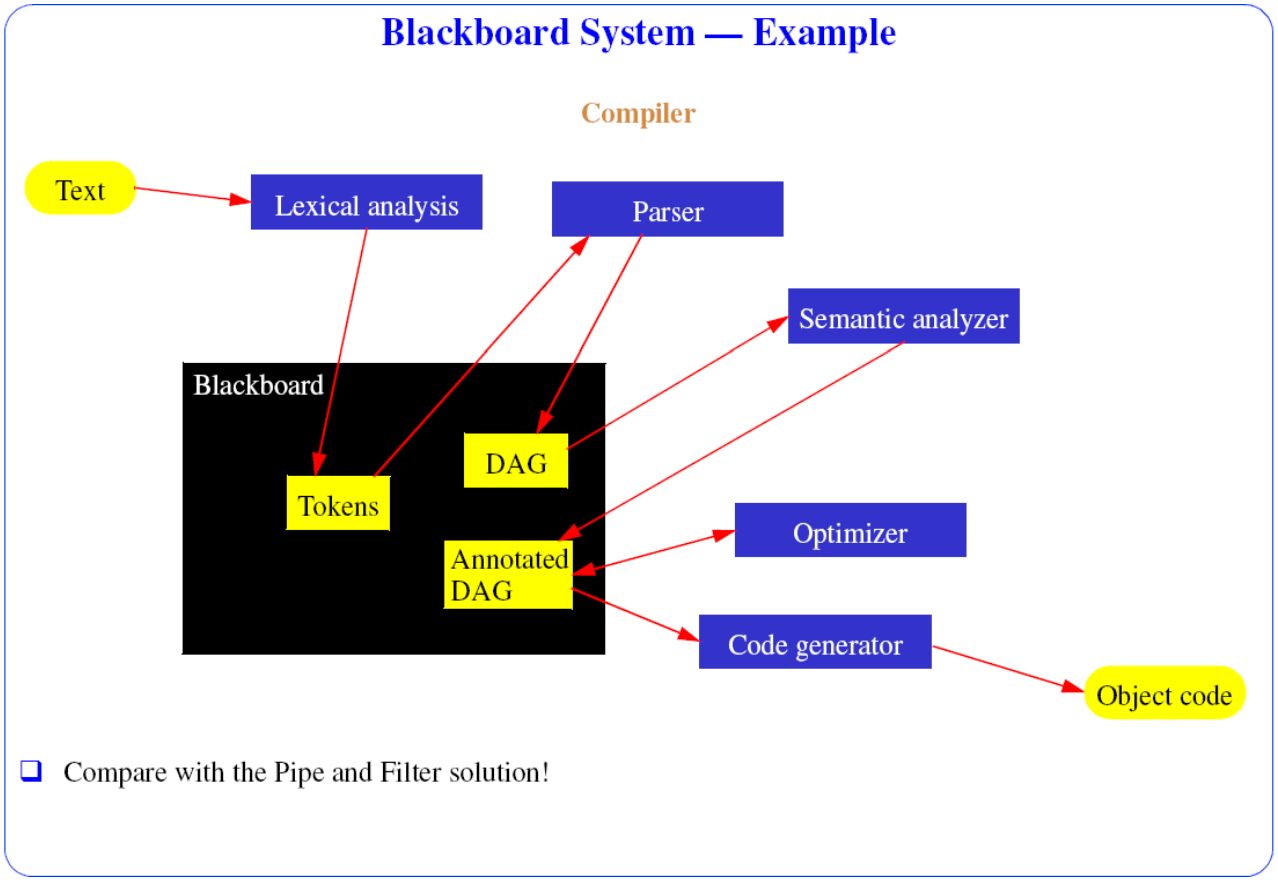




Advantages and Disadvantages

- Advantages
  - Suitable when there are diverse sources of data
  - Suitable for physically distributed environments
  - Suitable for scheduling and postponement of tasks and decisions
  - Suitable for team problem-solving approaches
  - Posting of problem subcomponents and partial results
- Disadvantages
  - Expensiveness
  - Difficult to determine partition of knowledge
  - Control unit can be very complex

Blackboard Systems – Example



Data-Centered Style: Advantages

- Data Integrity: Data is entered at one place. No risk for duplicates.
- Design Reuse: Accurate, reliable data are available when needed.
- View-Generation: Alternate views of the data are facilitated by a single source of data.
- Process Flexibility: The data management process is not constrained to application usage or sequence.
- Data Interaction Independent of Application: Data can be accessed by the user through multiple applications.
- Scalability: The database can grow when needed.

Heterogeneous Styles

- Systems are seldom built from a single style
- Kinds of heterogeneity:
  - Location heterogeneity; runtime structures reveal different styles in different areas
  - Hierarchy heterogeneity; different styles in decomposed areas
  - Simultaneous heterogeneity; any of the several styles may well be apt descriptions of the system