Chapter 2: Software Architecture

Concepts and Styles

H.S. Sarjoughian

CSE 460: Software Analysis and Design

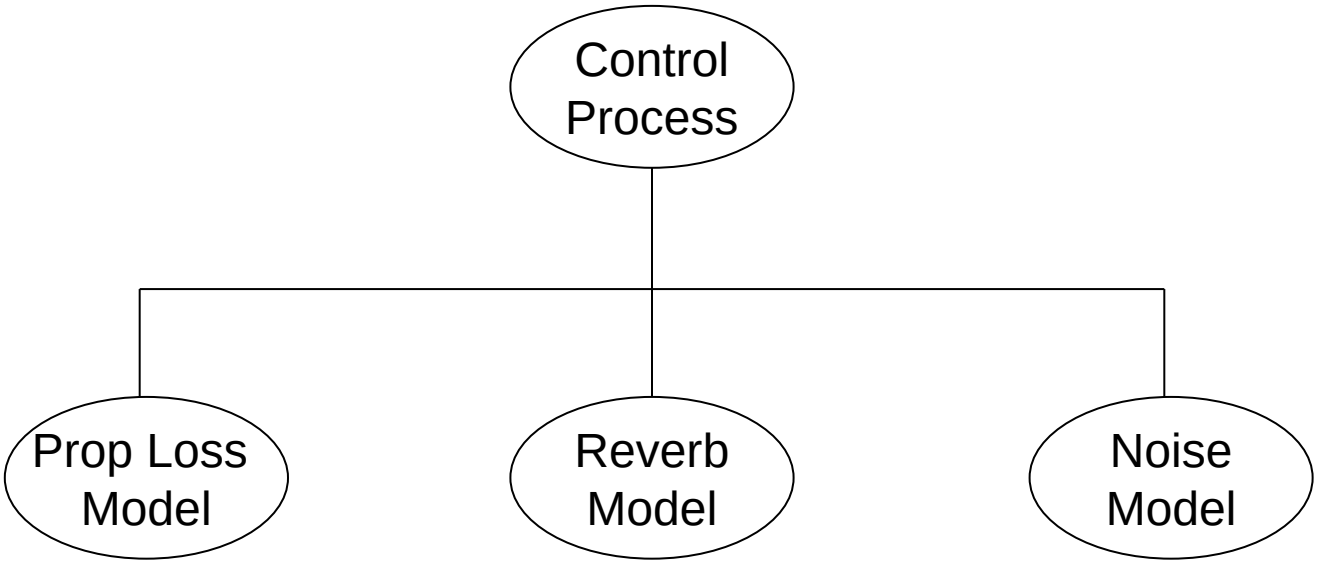
School of Computing, Informatics and Decision Systems Engineering
Fulton Schools of Engineering

Arizona State University, Tempe, AZ, USA

An "Architecture"

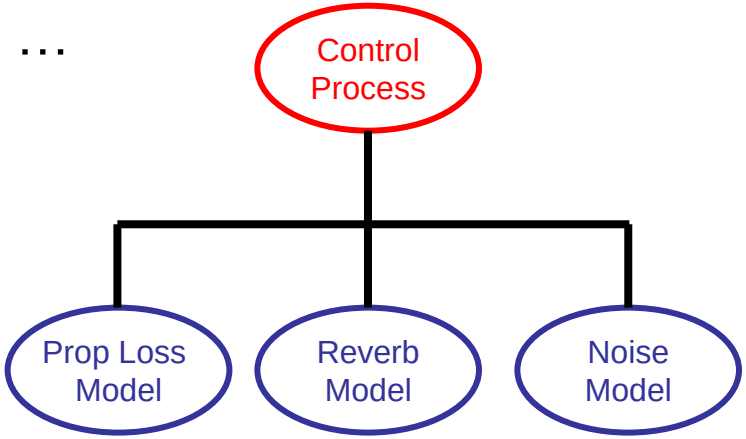
Underwater Acoustic Simulation

- Image underwater features
- Communicate information via the oceanic waveguide
- Measure oceanic properties



A Closer Look at a So-called Architecture

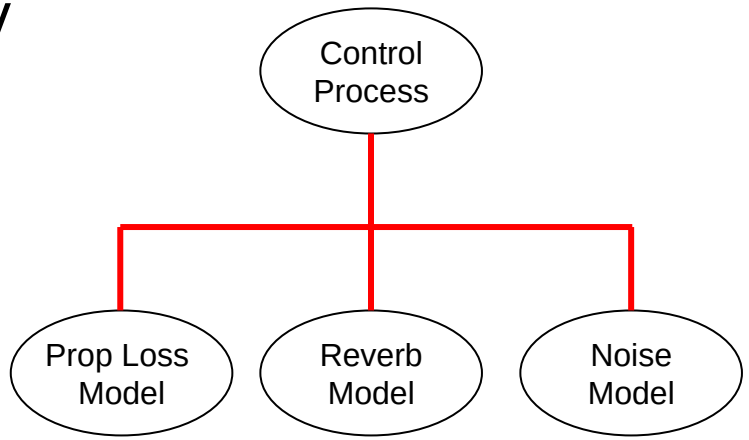
- **Component representation**
  - modules, objects, tasks, functions, …
  - assignment of labor, run-time separation, …
  - …
- **Configuration representation**
  - what does the layout signify?
    - does CP has a master/slave relationship to PLM, RM, and NM?
    - is CP placed in a separate level?
  - how do *control* and *data* flow through the system?
  - …



Underwater Acoustic Simulation

A Closer Look at a So-called Architecture (cont.)

- **Connection representation**
  - What do the “links” signify between any two components?
    - communication
    - control
    - data transmission
    - call/invoke
    - use
    - synchronization
    - …



Underwater Acoustic Simulation

Informal Software Architecture Specification

A typical software architecture can be represented as a set of **elements** and **relationships**. The architecture characterizations should help answer the following questions.

- What are the responsibilities of the elements?
- What is the significance of the relationships?
- What is the nature of the elements?
- What can we say about the architecture layout?

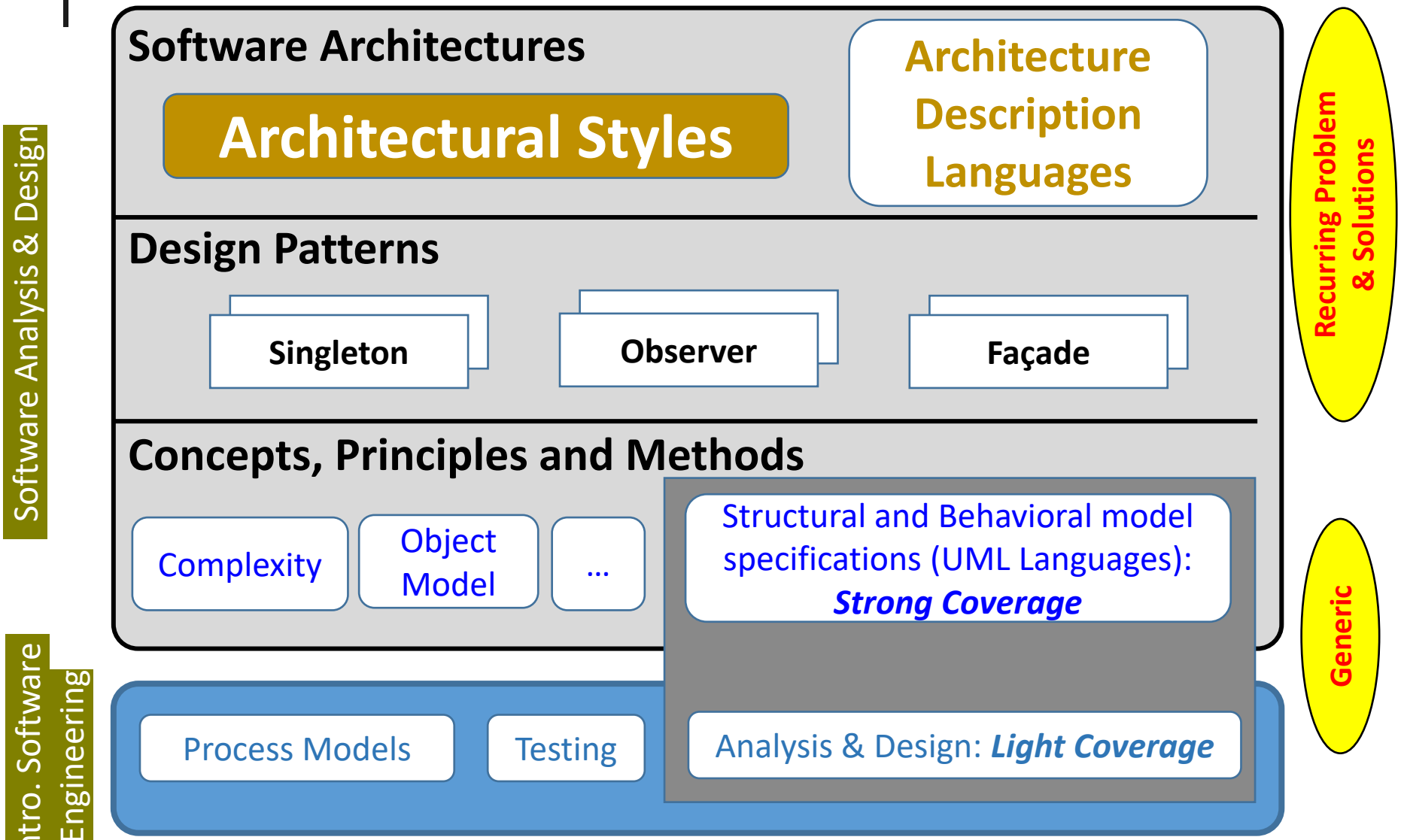
Abstract Definition of Software Architecture (cont.)

The **software architecture** of a computing system is the structure *or* structures of the system composed of *software components*, the *externally visible properties* of the components, and the *relationships* among them [SAP, 2003]

- software architecture emphasizes certain things and hides others
- software architecture can have many structures

A software architecture can be evaluated in terms of its goodness – how well it supports or enables a system's desired external behavior among other requirements

Software Architectures



Concepts for Defining Software Architecture

- **Software architecture style** can be defined to be a structure composed of nodes with arcs connecting them to one another
  - **Components (nodes)**
    - Each component should be specified
      - Type: software, hardware, or hybrid hardware/software
      - Behavior – static and dynamic aspects
      - Roles – processing, control, mediating, …
  - **Connections (arcs)**
    - The type and nature (e.g., asynchronous) of each connection should be specified
      - Communication
      - Control
      - Send/receive
  - **Structure**
    - The layout of the architecture should be specified
      - Kinds of interactions – hierarchical, flat, layered, ad-hoc, …

Architecture Style

- Architecture style: an abstraction of a set of architectures – few key features and few rules for combining the features
- Each architecture style signifies the level of effort necessary for developing the system given its scale, complexity, …
- Architecture style is ***not*** an architecture
  - number of components are not specified
  - does not specify all kinds of interactions that may take place among components
  - functionality of each component is rather vague – detailed/complete behavior is not specified on purpose

architecture style is an abstraction of a set of "similar" architectures

Lists of architecture styles are available in the same place as this class notes (see MovingFromQualitiesToArchitecture.pdf)

Architecture Style Solution Template

Architecture style (also referred to as system pattern) specifies a family of architectures. An architecture style as a solution consists of:

- **Overview –** provides a description for the pattern
- **Elements –** components (e.g., databases, processes, …) performing some form of computation (functions)
- **Relations** – interaction mechanisms such as procedure call, events, message bus, …
- **Semantic constraints** – specifies important and necessary constraints covering topology, element behavior, and interaction mechanisms (e.g., use of a particular communication protocol or data storage medium)
- **Weaknesses –** highlights structural and behavioral limitations

Some Common Architecture Styles (cont.)

- **Data-flow:** dominated by motion of data through the system, with no “upstream” content control by recipient
- **Data-centered:** dominated by a complex central data store, manipulated by independent computations
- **Call-and-return:** dominated by order of computation, usually with a single thread of control
- **Virtual machine:** characterized by translation of one instruction set into another
- **Independent components:** dominated by communication patterns among independent, usually concurrent, processes

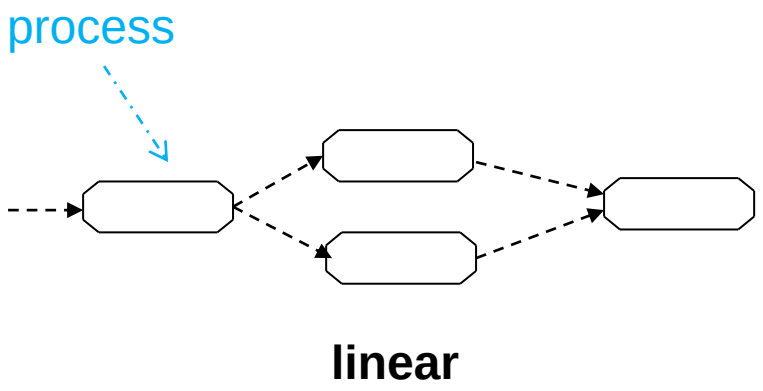
Data-Flow Architecture Style

- **Data-flow properties**
  - orderly movement of data from one process to next is its key feature – data transformation and latency are essential considerations
  - there exists explicit pattern for data flow
  - flow of computation *(i.e., control)* is governed by availability of data
    - in a **pure data-flow architecture**, all interactions among components are based on data alone
    - control is a "side effect" of data flow

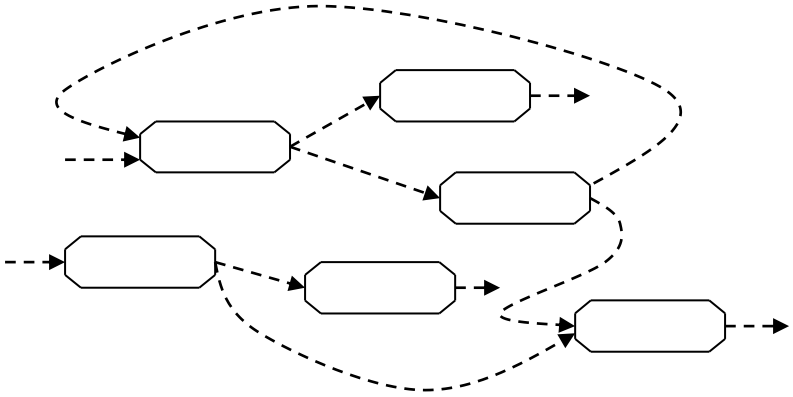
- **Quality attribute goals**
  - reuse
  - modifiability

Data-Flow Architecture Style (cont.)

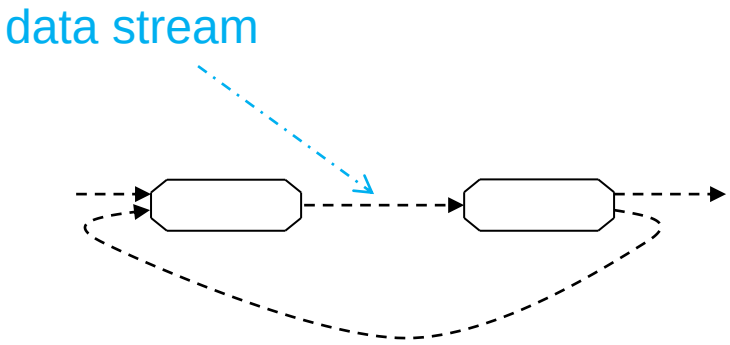
- **Patterns of data flow**
  - linear
  - cyclic
  - arbitrary



linear

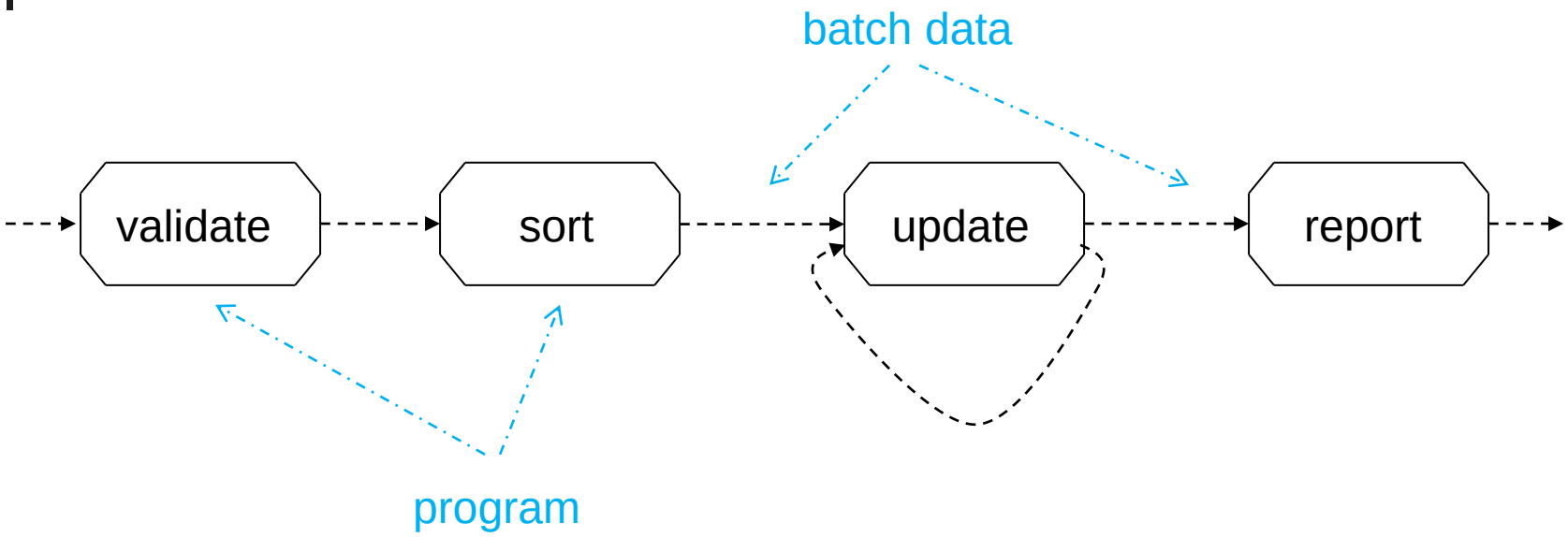


arbitrary



cyclic

Data-Flow: Batch Sequential Style



Characteristics:

- processing steps are independent
- each step must complete before the next step can begin
- batch data is transmitted in whole from one filter to the next

Example

- classical data processing (e.g., banking)

Data-Flow: Pipe-and-Filter

- **Filter – a data stream transducer**
  - each filter is a standalone component
  - incremental transformation of data stream by successive filters – operates on input (source data stream) data to produce output (target data stream)
    - each filter incrementally incorporates additional information
    - filter processing of data may include a new representation
    - does **not preserve state** information between instantiations
- **Pipe – a data stream transmitter**
  - moves data stream from one filter's output to another filter's input
    - uni-directional flow
  - each pipe has a source end that can only be connected to a filter's output port and a sink end that can only be connected to a filter's input port

Data-Flow: Pipe-and-Filter (cont.)

- **Pipe-and-filter operations**
  - processing continues (usually non-deterministically) until no more computations or data stream transmissions are possible
  - some filters may require queues to handle data streams
- **Pipe-and-filter can be hierarchical** – a set of pipes and filters can be packaged as a filter
- **Pipe-and-filter advantages**
  - allows parallel and/or distributed computing
  - enables reuse and maintenance since each filter can be treated as a black-box
- **Pipe-and-filter disadvantages**
  - interactive applications is hard to develop in this style
  - ordering of filters can be complex

TABLE 13.4 Pipe-and-Filter Pattern Solution

Overview	Data is transformed from a system's external inputs to its external outputs through a series of transformations performed by its filters connected by pipes.
Elements	*Filter,* which is a component that transforms data read on its input port(s) to data written on its output port(s). Filters can execute concurrently with each other. Filters can incrementally transform data; that is, they can start producing output as soon as they start processing input. Important characteristics include processing rates, input/output data formats, and the transformation executed by the filter. *Pipe,* which is a connector that conveys data from a filter's output port(s) to another filter's input port(s). A pipe has a single source for its input and a single target for its output. A pipe preserves the sequence of data items, and it does not alter the data passing through. Important characteristics include buffer size, protocol of interaction, transmission speed, and format of the data that passes through a pipe.
Relations	The *attachment* relation associates the output of filters with the input of pipes and vice versa.
Constraints	Pipes connect filter output ports to filter input ports. Connected filters must agree on the type of data being passed along the connecting pipe. Specializations of the pattern may restrict the association of components to an acyclic graph or a linear sequence, sometimes called a pipeline. Other specializations may prescribe that components have certain named ports, such as the *stdin*, *stdout*, and *stderr* ports of UNIX filters.
Weaknesses	The pipe-and-filter pattern is typically not a good choice for an interactive system. Having large numbers of independent filters can add substantial amounts of computational overhead. Pipe-and-filter systems may not be appropriate for long-running computations.

Source: Software Architecture in Practice, 3rd Ed.

Data-Flow: Pipe-and-Filter (cont.)

Data-Flow: Pipe-and-Filter (cont.)

Source: Software Architecture in Practice, 3rd Ed.

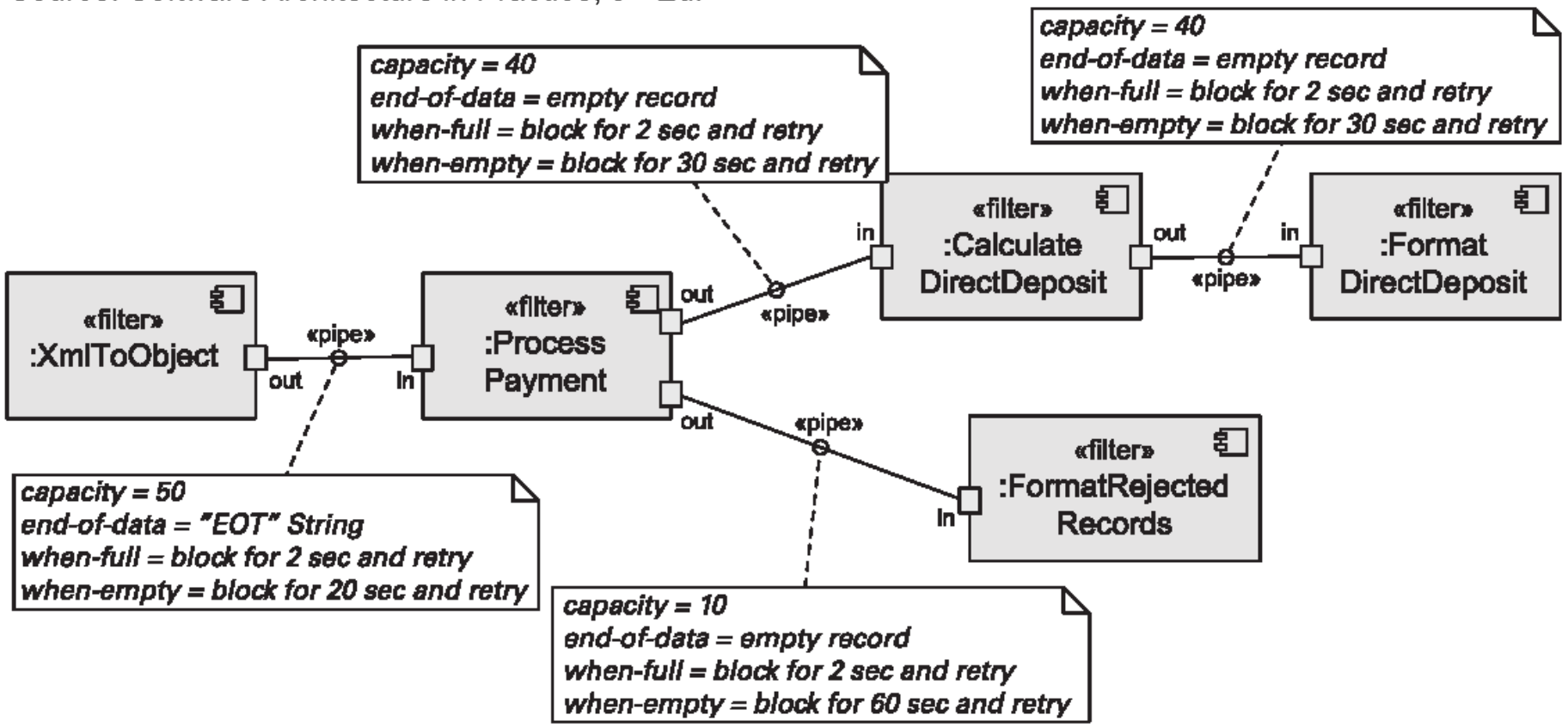


FIGURE 13.8 A UML diagram of a pipe-and-filter-based system

Key Word in Context (KWIC) Example

The **Key Word In Context** index system accepts an ordered set of lines; each line is an ordered set of words; and each word is an ordered set of characters. Any line may be circularly shifted by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order

Example

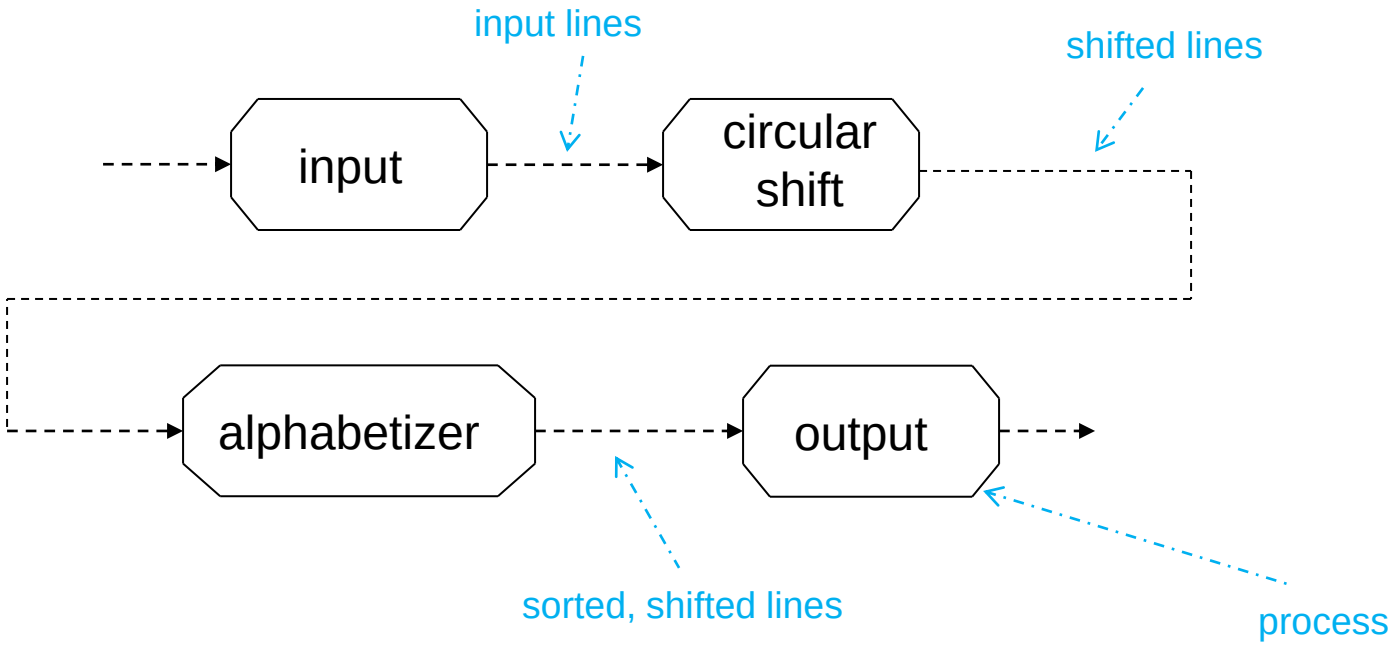
Inputs: sequence of lines

- pipes and filters
- introduction to software architecture

Outputs: sequence of lines, circularly shifted and alphabetized

- and filter pipes
- architecture introduction to software
- filter pipes and
- introduction to software architecture
- pipes and filter
- software architecture introduction to
- to software architecture introduction

Pipe-and-Filter: KWIC Example



Advantages

- intuitive flow of processing
- each filter can function independently – reusability quality attribute
- new functions can be added – modifiability quality attribute

Disadvantages

- difficult to modify the design (making it interactive requires use of some persistence shared data storage)

Comparison of Architecture Styles via Some Common Features

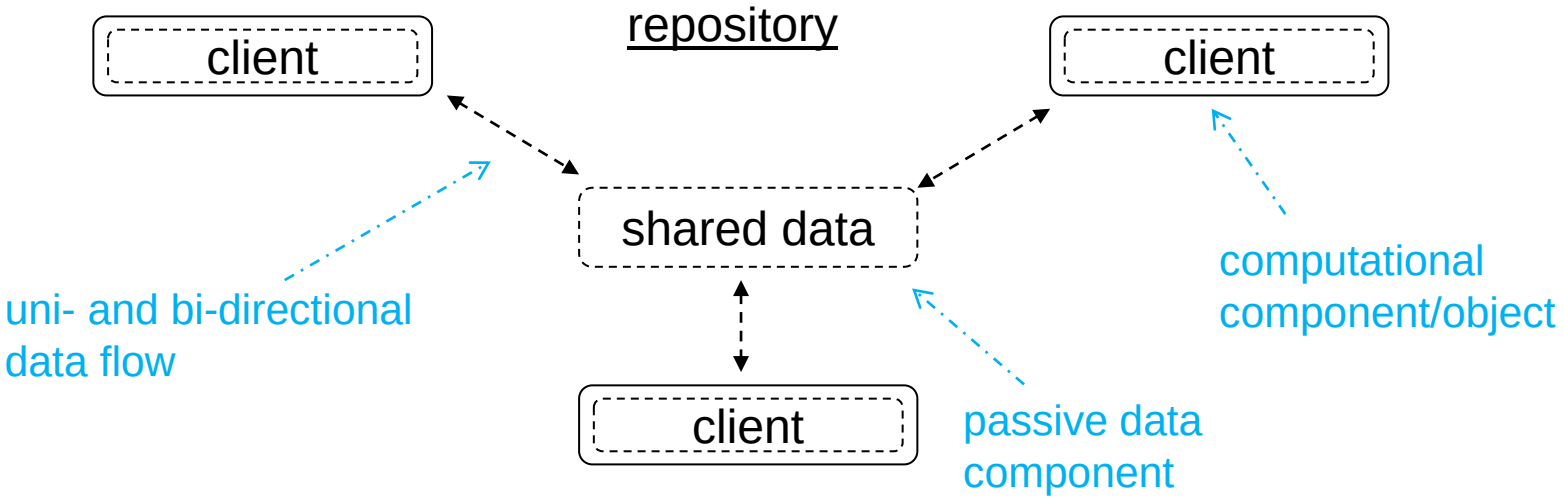
- Architecture styles can be studied (described, compared, and analyzed) in terms of
  - **constituent parts** (components and connectors)
  - **control issues**
    - topology – linear(uni-directional), arbitrary, hierarchical, fixed, star
    - synchronicity – sequential, synchronous, asynchronous, opportunistic
    - binding time – write-time, compile time, invocation time, runtime
  - **data issues**
    - topology
    - continuity – continuous, sporadic, high- and low-volume
    - mode – passed, shared, broadcast, multicast
    - binding time
  - **control/data interaction issues**
    - shape – isomorphic control and data topologies
    - directionality – same or opposite

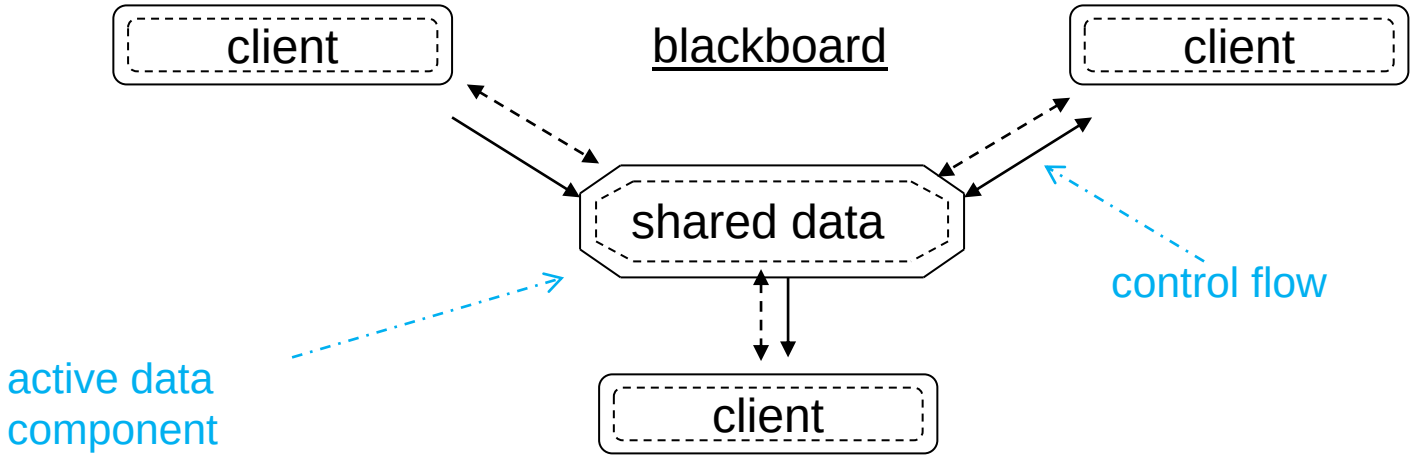
see Table 5.1, 5.2, and 5.3 (Ch. 5, SAP)

Data-Centered Style

- Database and clients are components
  - each client interacts with a centralized database – data may be shared among clients, clients are largely independent from one another
- Quality attribute goals
  - scalability – additional clients or data can be added
  - modifiability – some clients may have additional functionality
- Purpose
  - access/update large amounts of data
- Styles/operations
  - repository (passive)
  - blackboard (active) – notifies clients when some data of interest changes

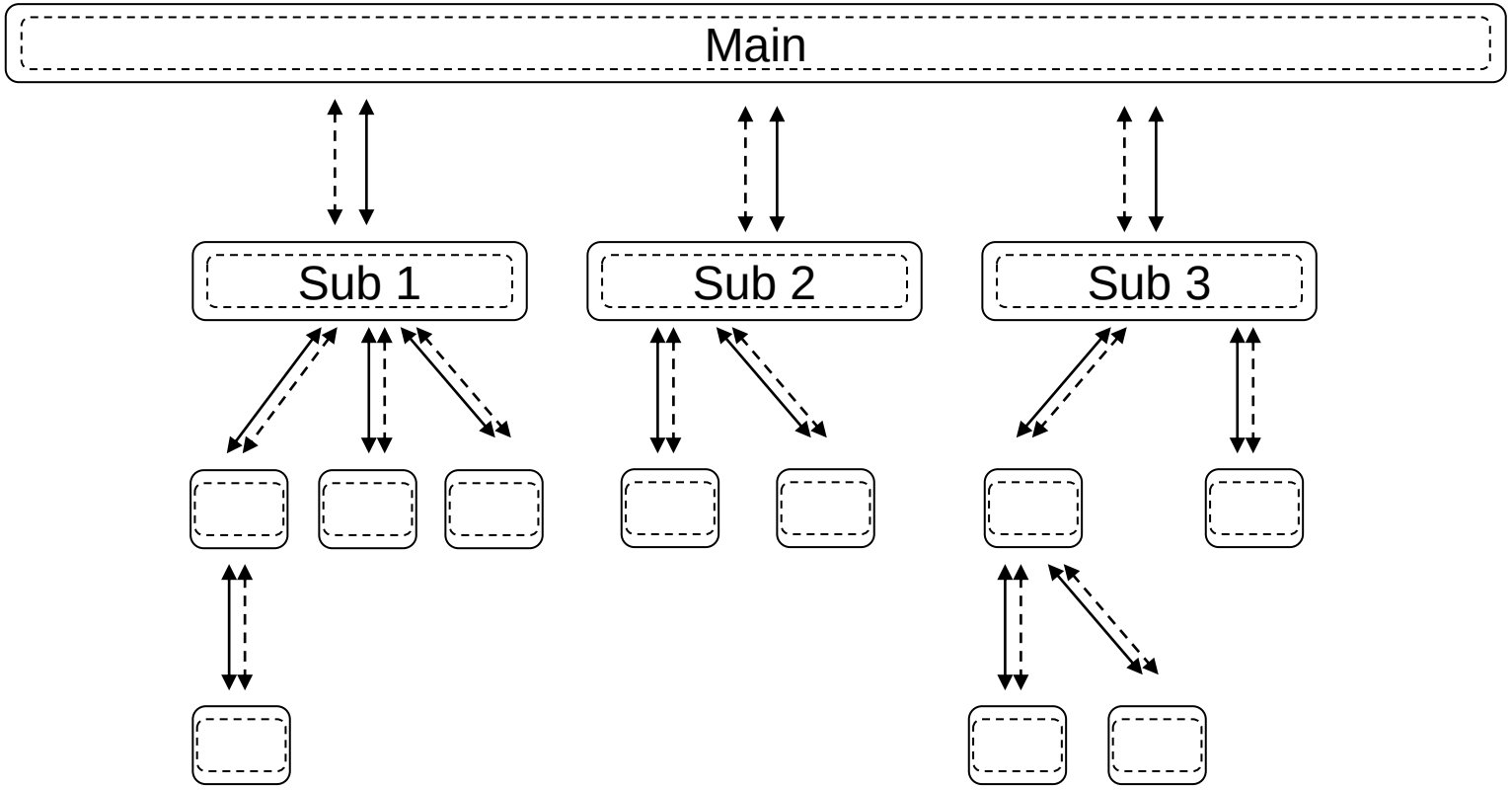
Repository and Blackboard Styles



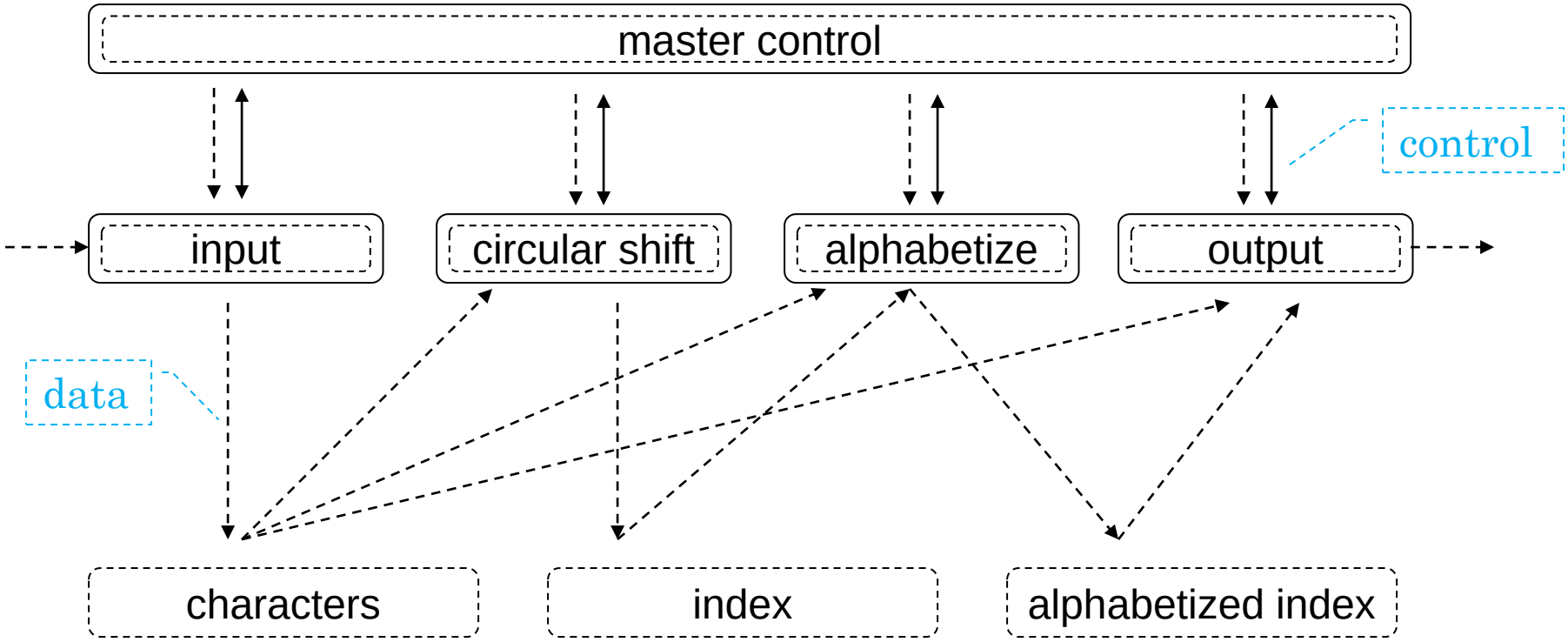


Main-Program-and-Subroutine Style

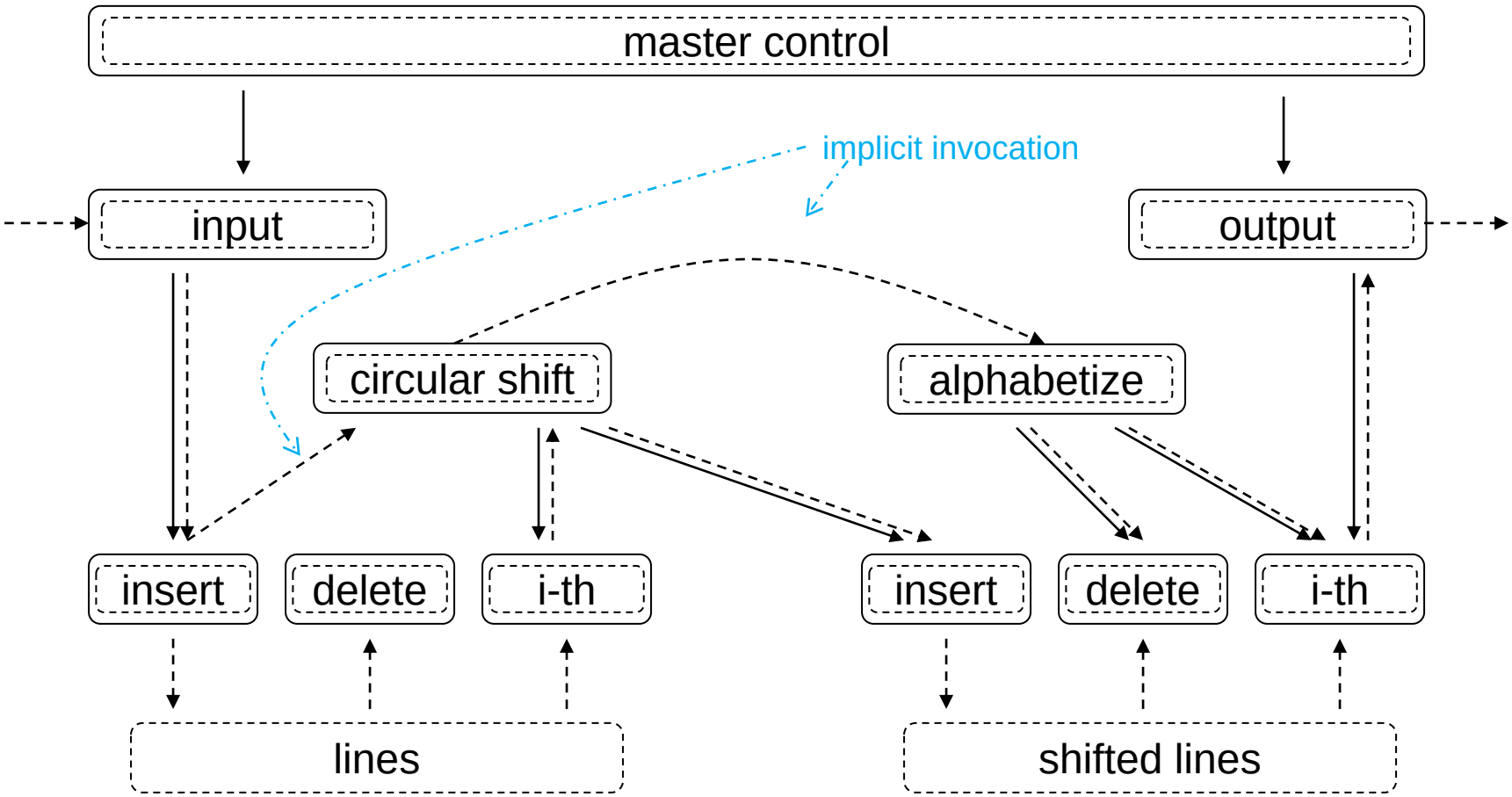
- Functionality is decomposed hierarchically – each component interacts (control/data) directly with its parent and child
- Usually there exists one thread of control



Main-Program-and-Subroutine: KWIC Example



Implicit Invocation: KWIC Example



Evaluation of KWIC Alternative Architectures w.r.t. Quality Attributes

	Pipe-and-Filter	Main-Program-and-Subroutine with Shared Data	Implicit Invocation
change in algorithm	+	–	+
change in data representation	–	–	–
change in function	+	–	+
performance	–	+	–
reuse	+	–	–

Analysis of alternative architectures (e.g., KWIC) depends on extensive (broad and deep) knowledge of architecture styles and a given problem (domain-specific knowledge). I.e., use of an approach is necessary for evaluating "goodness" of an architecture

- Software Architecture Analysis Method can be used to formulate architecture goals and determine how well an architecture can satisfy them (Ch. 9 & 10, SAP)

Choosing an Architecture Style

Some useful rules of thumb can be used to determine candidate architecture styles. Goals such as “predictable” flow of control and data point to select architecture styles

Call-and-Return	▪ order of computation is fixed; generally one component is active at any given time – other components must sit idle
Object-oriented	modifiability and integrability are important
– Objects	– lots of opportunities for inheritance (reuse) and information hiding
Layered	▪ tasks can be divided between those specific to the application and those generic to many applications but specific to the underlying computing platform ▪ portability across computing platforms ▪ use of existing layers
Independent Components	multiprocessor system; loosely coupled interactions; performance tuning (SW/HW reallocation)
Event systems	▪ Decouple consumers of events from their producers ▪ Scalability

see Table 5.4 (Ch. 5, SAP 1997)

Summary

- Each architecture style represents common ways components and connectors can work together (static topology and dynamic control) to solve a generic class of problems within some given constraints
- Architecture styles can be evaluated in terms of their strengths and weaknesses in support of known quality attributes
- Architecture styles are not “definitive”
  - each style accommodates a subset of quality attributes
  - an architecture is often constructed from multiple architecture styles
  - each style can be examined in terms of its constituent parts, control and data issues, and interactions between data and control
- A substantial repertoire of architecture styles exist
  - rules of thumb exist to guide the architect in selecting architecture styles

References

- *Software Architecture in Practice, 4rd Edition, SAP2, Bass, L. Clements, P., and Kazman, R., Addison Wesley, SEI Series in Software Engineering, 2021*
- *Software Architecture in Practice, 2nd Edition, SAP2, Bass, L. Clements, P., and Kazman, R., Addison Wesley, SEI Series in Software Engineering, 2003*
- *Software Architecture in Practice, SAP, Bass, L. Clements, P., and Kazman, R., Addison Wesley, SEI Series in Software Engineering, 1998*
- *Shaw, M. and P. Clements, "A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems", Proc. COMPSAC97, 21st Int'l Computer Software and Applications Conference, August 1997, pp. 6-13*
- *Underwater Acoustic Modeling and Simulation, https://www.aticourses.com/sampler/Underwater%20Acoustic%20Modeling%20&%20Simulation.pdf*