Data Flow Architecture

- Batch sequential
- Pipe-and-filter

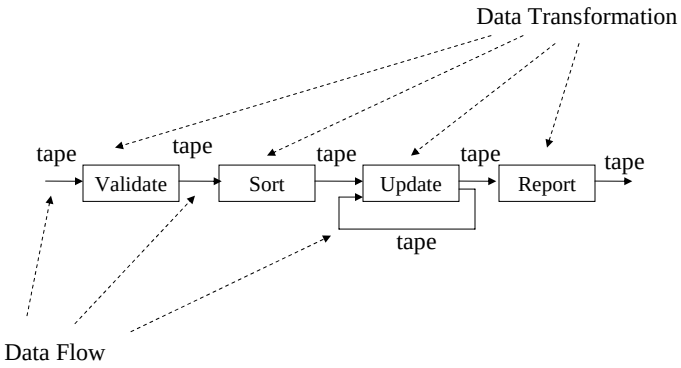
95

Typical Database Applications

- Edit program: accept transaction input and perform validation without access to DB
- Transaction sort: get the transactions into the same order as the records on the sequential master file
- Update program (one for each master file): execute the transactions by
  - Moving sequentially through master files
  - Matching each type of transaction to its corresponding account
  - Updating the account records
- Print program: produce periodic reports

96

Batch Sequential Systems



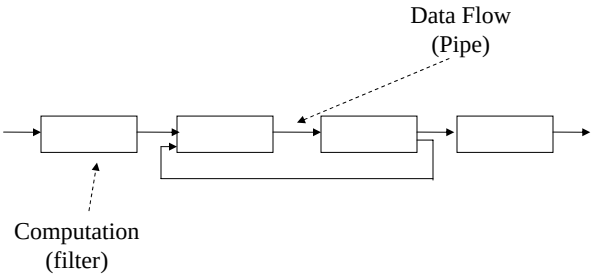
97

Batch Sequential systems

- Processing steps are independent programs
- Each step runs to completion before next step starts
- Data transmitted as a whole between steps
- Typical applications:
  - Classical data processing
  - Program development

98

Pipes and Filters



Pipes and Filters

- Pipe
  - Move data from a filter output to a filter input
    - One-way flow; may be flow control upstream but not data
  - Pipes from data transmission graphs
- Filter
  - Incrementally transform some amount of the data at inputs to data at outputs
    - Enrich data by computing and adding information
    - Refine data by concentrating or extracting information
    - Transform data by changing its representation
  - Stream-to-stream transformations
  - Use little local context in processing stream
  - Preserve no state between instantiations
- Overall computation
  - Run pipes and filters (non-deterministically) until no more computations are possible
  - Action mediated by data delivery

Pipe-and-Filter Invariants

- Filters **do not share state** with other filters.
- Filters **do not know the identity** of their upstream or downstream filters.
- The **correctness** of the output of a pipe and filter network should not depend on the order in which their filters perform their incremental processing.

101

Common Specializations

- Pipeline: the topology is restricted to linear
- Batch sequential: a special case of pipeline, when each filter processes all of its input data as a single entry
- Bounded pipe: the capacity of pipe is restricted
- Typed pipe: passing typed data

102

Pipe-and-filter in Unix

- Unix processes that transform stdin to stdout are generally called filters
  - But when they consume all the input before starting the output, they violate the filter assumption (e.g., sort)
- Unix pipes can treat files as well as filters as data sources and sinks
  - But files are passive, so you can't make arbitrary combinations
- Unix assumes that the pipes carry ASCII character streams
  - The good news: anything can connect to anything else
  - The bad news: everything must be encoded in ASCII, then shipped, then decoded
  - The cloud on the good news: filters make assumptions about the syntax of the stream

103

Data Pulling and Pushing

- What is the force that makes the data flow?
- Three choices, all with force emanating from filters:
  - Push: data source pushes data in downstream direction
  - Pull: data sink pulls data from upstream direction
  - Push/pull: a filter is actively pulling from upstream, computing, and pushing downstream
- Combinations may be complex: if more than one filter is pushing/pulling, synchronization is needed

104

Pipe-and-Filter Advantages

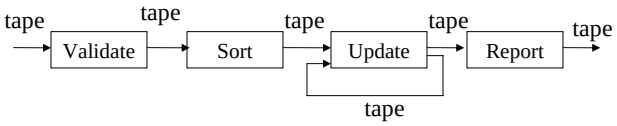
- Easy to understand the overall input/output behavior of a system as a simple composition of the behaviors of the individual filters.
- They support reuse, since any two filters can be hooked together, provided they agree on the data that is being transmitted between them.
- Systems can be easily maintained and enhanced, since new filters can be added to existing systems and old filters can be replaced by improved ones.
- They permit certain kinds of specialized analysis, such as throughput and deadlock analysis.
- They naturally support concurrent execution.

Pipe-and-Filter Disadvantages

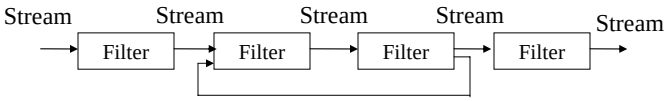
- Not good for handling interactive systems, because of their transformational character.
- Excessive parsing and unparsing leads to loss of performance and increased complexity in writing the filters themselves.

Batch Sequential vs Pipe-and-Filter (Unix)

Batch Sequential



Pipe-and-Filter (Unix)



107

Batch Sequential vs Pipe-and-Filter (Unix)

- Coarse-grained
- Total
- High latency
- Random access to input ok
- No concurrency

- Fine-grained
- Incremental
- Results start immediately
- Processing localized in input
- Feedback loops possible

•Decompose task into fixed sequence of computations
•Interact only through data passed from one to another

108

Key Word In Context (KWIC)

- Problem description:
  - The KWIC index system accepts an ordered set of lines,
    - each line is an ordered set of words
    - each word is an ordered set of characters
  - Any line may be “circularly shifted” by repeatedly
    - removing the first word
    - 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

Pipe and Filter
Software Architecture in Practice

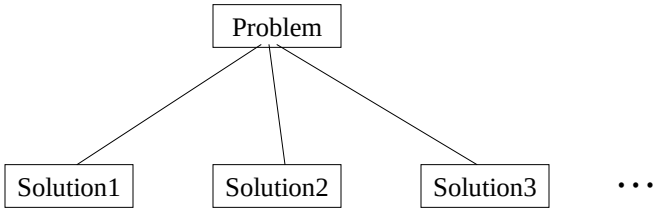
Pipe and Filter
and Filter Pipe
Filter Pipe and

Software Architecture in Practice
Architecture in Practice Software
in Practice Software Architecture
Practice Software Architecture in

and Filter Pipe
Architecture in Practice Software
Filter Pipe and
in Practice Software Architecture
Pipe and Filter
Practice Software Architecture in
Software Architecture in Practice

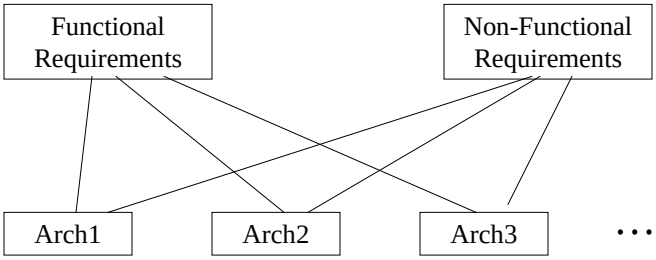
Solutions (Design Alternatives)

- There can be many solutions to this problem



111

How to Choose



Basis for tradeoff analysis & design rationale

112

Non-Functional Requirements

- Modifiability
  - Changes in processing algorithms, e.g.
    - Line shifting
      - One at a time as it is read
      - All after they are read
      - On demand when the alphabetization requires a new set of shifted lines
    - batch alphabetizer vs incremental alphabetizer
  - Changes in data representation, e.g.
    - Storing characters, words and lines
      - in 1-D/2-D array/linked list, compressed/uncompressed
      - Explicitly/implicitly (as pairs of index and offset)
      - Core storage/secondary storage
- Enhanceability
  - Enhancement to system function, e.g.,
    - eliminate noise words (a, an, the, and, etc.)
    - The user deletes lines from the original or shifted lines
- Performance
  - Space and time
- Reusability
  - To what extent can the components serve as reusable entities

113

Solution 1: Pipe and Filter

- Decompose the overall processing into a sequence of processing steps:
  - Read input
  - Make shifts
  - Alphabetize
  - Print results
- Each step transforms the data completely

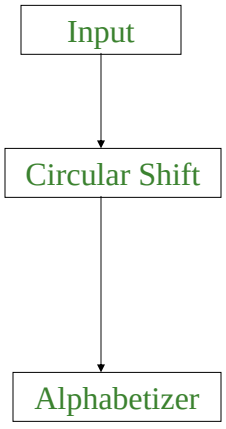
114

Solution 1: Modularization

- Module 1: Input
  - Reads data lines
  - Passes it to Circular Shift
- Module 2: Circular Shift
  - Called after Input is done
  - Reads lines
  - Makes shifts
  - Passes the results to Alphabetize
- Module 3: Alphabetize
  - Called after Circular Shift
  - Reads lines
  - Arrays the lines in alphabetical order
- Module 4: Output
  - Called after alphabetization
  - Prints nicely formatted output of shifts

115

Solution 1: Pipe and Filter



Pipe and Filter
Software Architecture in Practice

Pipe and Filter
and Filter Pipe
Filter Pipe and

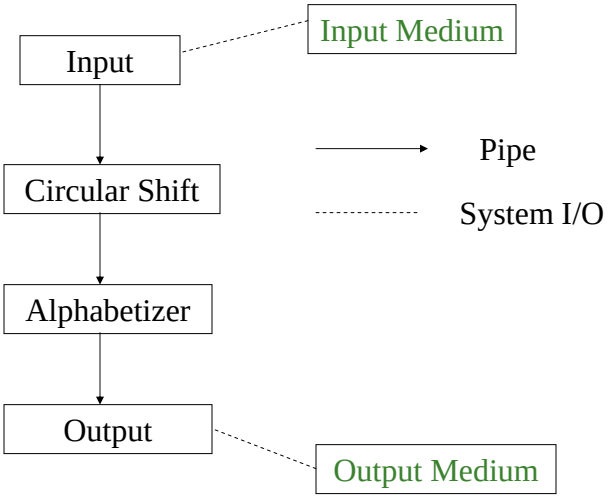
Software Architecture in Practice
Architecture in Practice Software
in Practice Software Architecture
Practice Software Architecture in

and Filter Pipe
Architecture in Practice Software
Filter Pipe and
in Practice Software Architecture
Pipe and Filter
Practice Software Architecture in
Software Architecture in Practice

116

KWIC: Pipe and Filter

Filter:
Input
Circular Shift
Alphabetizer
Output



117

Advantages

- Maintain the intuitive flow of processing
- Support reuse:
  - Each filter is independent
  - New filters can be added for new functions
- Easy to modify:
  - The change of the function of one filter will not affect other filter since they are independent

118

Drawbacks

- Hard for interactive system, e.g. deleting a line requires some persistent shared storage
- Space inefficiency:
  - Each filter must copy all data

Non-Functional Requirements

- Modifiability
  - Changes in processing algorithms, e.g.
    - Line shifting
      - One at a time as it is read
      - All after they are read
      - On demand when the alphabetization requires a new set of shifted lines
    - batch alphabetizer vs incremental alphabetizer
  - Changes in data representation, e.g.
    - Storing characters, words and lines
      - in 1-D/2-D array/linked list, compressed/uncompressed
      - Explicitly/implicitly (as pairs of index and offset)
      - Core storage/secondary storage

Filter Independency

- Enhanceability
  - Enhancement to system function, e.g.,
    - eliminate noise words (a, an, the, and, etc.), by inserting filters at the appropriate points
    - The user deletes lines from the original or shifted lines – hard for interactive system
- Performance
  - Space and time – inefficient due to data replication
- Reusability
  - To what extent can the components serve as reusable entities – high due to filter independency

Pipe and Filter

- Components: Filters
- Connectors: Pipes
- Styles: Pipe-and-Filter
- Rationale: Non-Functional Requirements (NFRs)

121