Simple Architecture Case Study

- Explore characteristics of different architectures in a case study
- Design considerations
  - optimum design relative to requirements
- Choice of ‘best’ architecture often driven by non-functional aspects
  - e.g. performance, changeability, maintainability, reuse

Use the original KWIC example from Garlan/Shaw Software Architecture book 1995

KWIC Case Study

Simple standard problem, *a bit artificial* but allows

- Comparison of different architectural approaches based on:
  - Change in data representation
  - Change in function
    - *change what problem is solved*
  - Change in algorithm
    - *change how the problem is solved*
  - Change in control structure
  - Performance
  - Reuse

KWIC (Key Word In Context) Case Study

- Problem description (Parnas):

"The KWIC 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."

KWIC History

The first instance of indexing using words from titles of documents is attributed to a British librarian, Andreas Crestadoro, who advocated the permutation of the words in titles in 1856 so that the subject matter index would follow the author's own definition of the contents of the book. His paper on this topic is titled *The Art of Making Catalogues of Libraries; A Method to Obtain in a Short Time a Most Perfect, Complete, and Satisfactory Catalogue of the British Museum Library, By a Reader Therein.*

In the 1950s, Hans Peter Luhn created a revolution in the history of information retrieval by applying the machine to Crestadoro's method of indexing. Luhn called his invention the Keyword-in-Context (KWIC) index. He advocated permuted title techniques and demonstrated the practical application of KWIC methods by preparing an index to papers presented at the International Conference on Scientific Information (ICSI) in November 1958 in Washington D. C. Luhn used two IBM machines, the 9900 Index Analyzer and the Universal Card Scanner, to prepare the indexes. These methods inspired and influenced others to experiment with the KWIC technique and by 1964, over forty examples of KWIC and other variations of permuted keyword indexing were in operation to aid researchers (Stevens, 1970).

KWIC Case Study

- Input: sequence of lines

Example:

Pipes and Filters
Architectures for Software Systems

- Output: Sequence of lines, circularly shifted and alphabetised

Example:

and Filters Pipes
Architecture for Software Systems
Filters Pipes and
for Software Systems Architectures
Pipes and Filters
Software Systems Architectures for
Systems Architectures for Software

KWIC Design Considerations

How sensitive design is to:

- Change in algorithm
  - shift each line as read or all lines after read
- Change in Data Representation
  - how lines are stored, how shifting represented
- Change in Function
  - ignore trivial words like ‘the’

- Performance
  - space and time limits
- Reuse
  - how parts of the design might be reused

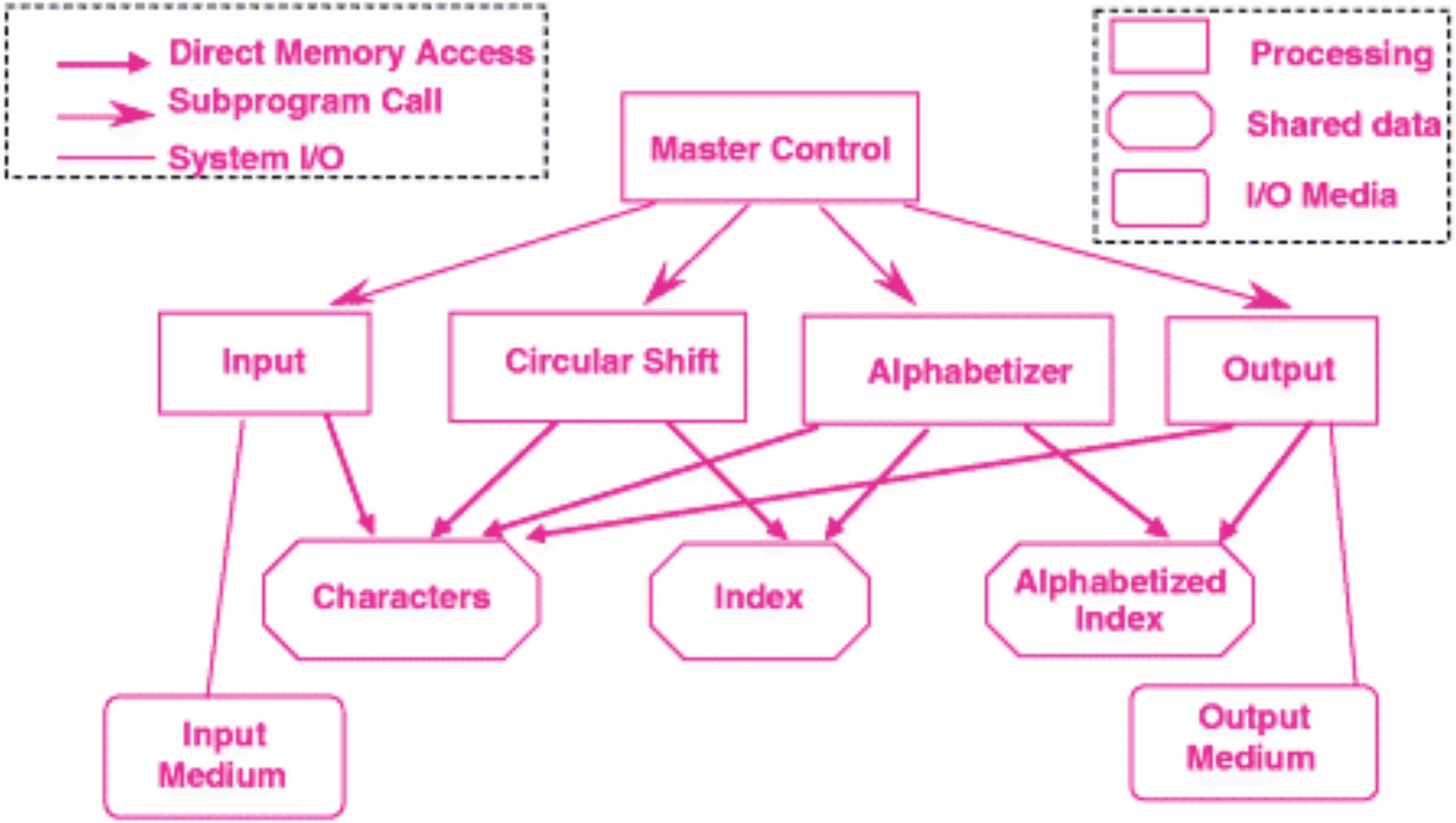
Solution 1

- Decompose the overall processing into a sequence of processing steps
  - Read Lines, Make Circ Shifts, Alphabetize, Print
- Each step transforms the data completely
- Intermediate data stored in shared memory
  - arrays of characters with indexes
- Relies on sequential processing

Solution 1: Modularization

- Module 1: Input
  - Reads in lines and stores them
  - Format: array of words, array of pointers to start of each line
- Module 2: Circular Shift
  - Reads data, and forms alphabetized index, with a new array of pairs, index of start of each circular shifts, index of start of original
- Module 3: Alphabetize
  - Creates new index in alphabetical order
- Module 4: Output
- Module 5: Main control
  - Controls sequencing of module invocations
  - Handles errors

Architecture of Solution 1



Properties of Solution 1

- Old fashioned batch sequential processing
- Uses shared data to get good performance
- Processing steps controlled by Main module
  - Single thread
- Shared data structures cause inter-module dependency

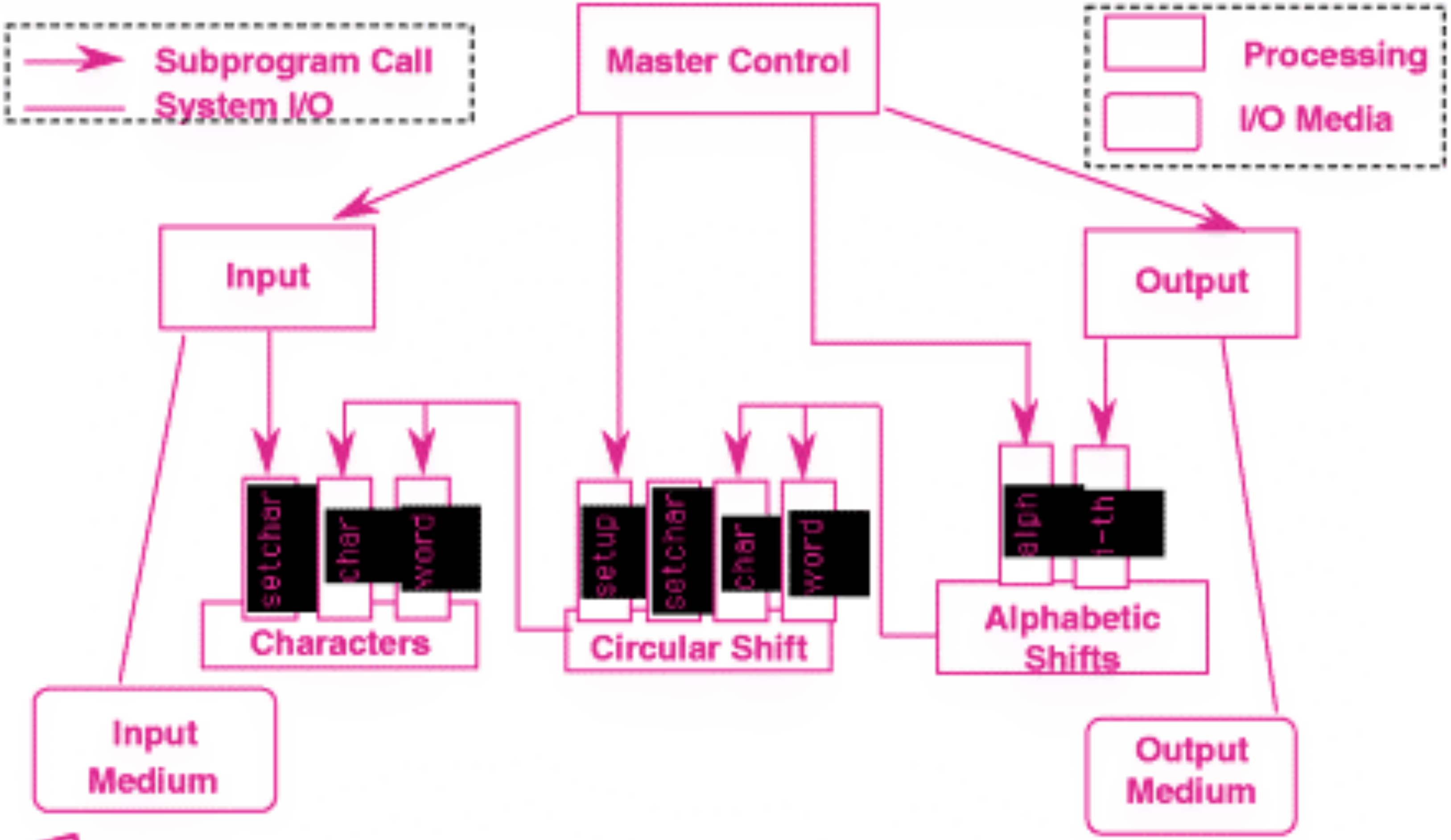
Solution 2

- Use similar main module control
- Encapsulate data, using objects for:
  - Initial lines
  - Shifted lines
  - Alphabetized lines
- Objects
  - Handle the representation of data
  - Access is via interfaces

Solution 2: Encapsulated Modules

- Module 1: Line storage
  - Manage lines and characters
- Module 2: Input
- Module 3: Circular Shift
- Module 4: Alphabetize
- Module 5: Output
- Module 6: Master Control

Architecture of Solution 2



Properties of Solution 2

- Module interfaces are abstract
  - Hide data representation
  - Hide internal algorithm to process data
  - Must be used correctly by client/user
- Allows independent development of modules
- (When Parnas used this example, claimed that the executable code could end up similar to 1)

Comparison: Soln 1 vs Soln 2

- Change in Algorithm
  - Soln 1: batch algorithm wired in
  - Soln 2: provides several alternatives
- Change in Data Representation
  - Soln 1: shared common data format
  - Soln 2: data representation hidden
- Change in Function
  - Soln 1: ok - need to add new phase of processing
  - Soln 2: similar to soln 1
- Performance
  - Soln 1: good
  - Soln 2: probably not as good
- Reuse
  - Soln 1: poor as tied to particular data formats
  - Soln 2: better

Solution 3: Event-listener

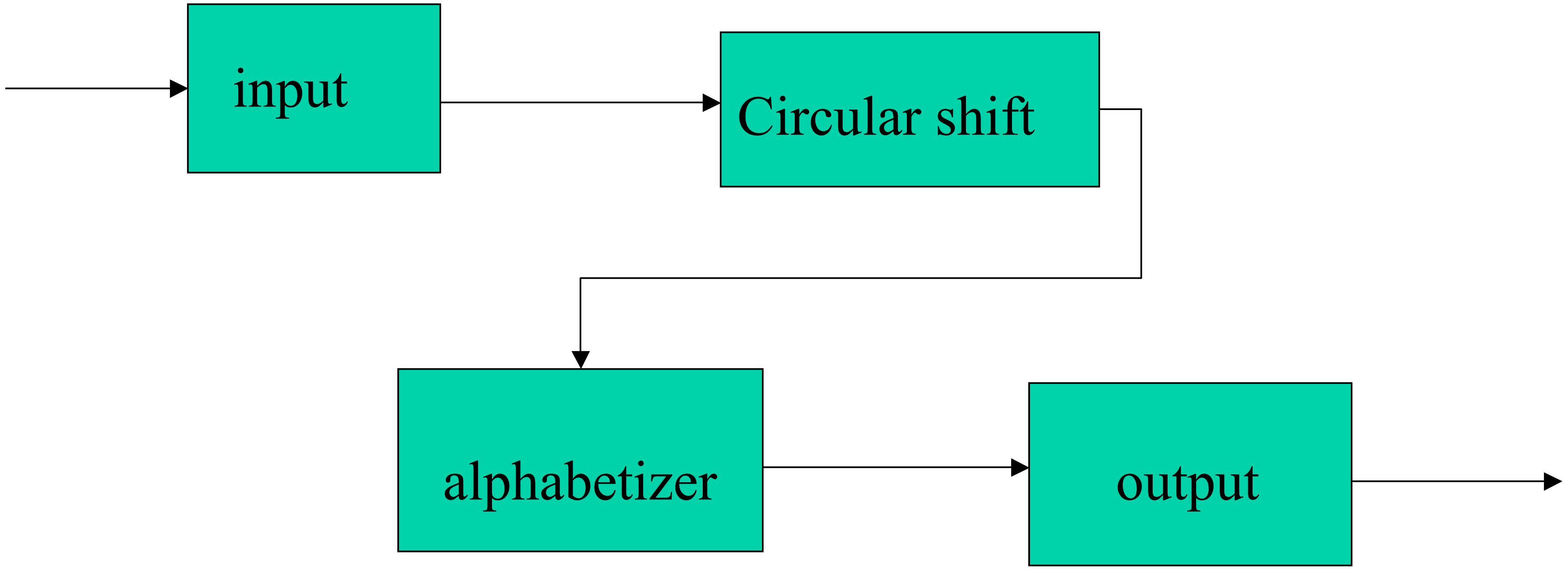
- Solution 2, but with implicit invocation
- Avoid the explicit master control by having the modules generate events when data available, and modules which use the data have listeners.

Advantage: allows easy integration of new modules which can be added as listeners

Disadvantage: flow of control less predictable

Solution 4: pipes and filters

- 4 filters
  - input; shift; alphabetise; output
- filters process data independently
- filters run whenever data is available
- data sharing is only data sent via pipes



Pipes and filters solution

Pipes and Filter Solution:pros

- follows processing flow
- supports reuse
- algorithm can be changed by using different combination of filters
- new functions can be added via new filters
- easily modified since filters independent

Pipes and Filter Solution:cons

- inefficient since all data copied by filter
- change in data representation affects adjacent filter(s)
- difficult to make system interactive
  - e.g. to allow user to override data

Summary pipe and filter solution

- change in algorithm: +
- change in data representation: -
- change in function: +
- performance: -
- reuse: +

Architecture case studies

- Compare via different properties
- For particular application, certain concerns may dominate
  - e.g. changes in function need not be considered, efficiency is of most importance