Lecture 5

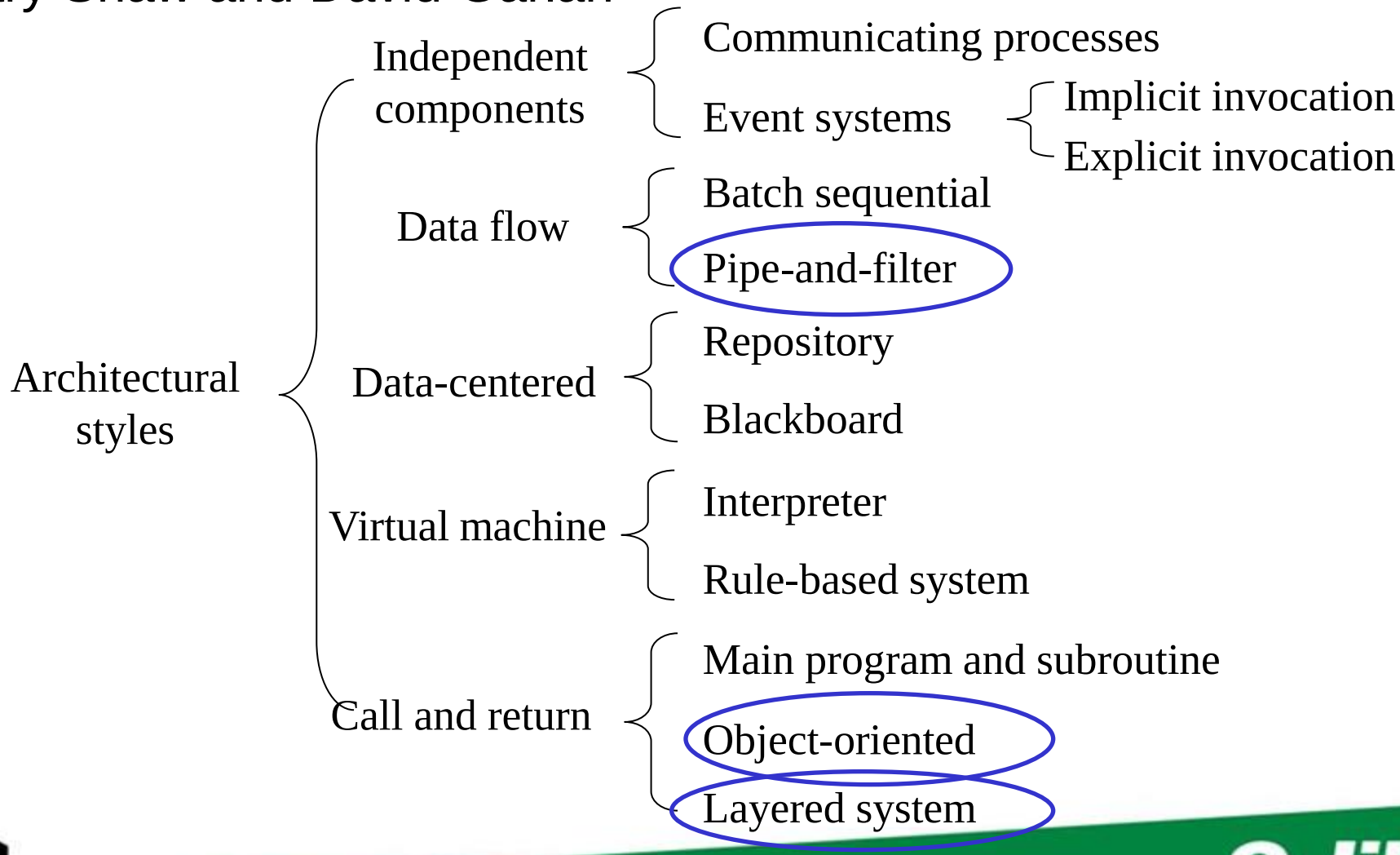
Software Architecture Design

Today's lecture

- A catalogue of software architectural styles
- Use of appropriate styles for a design problem
- Combination of styles in one design
- Case studies
  - Case study 1: keyword in context
  - Case study 2: keyword frequency vector

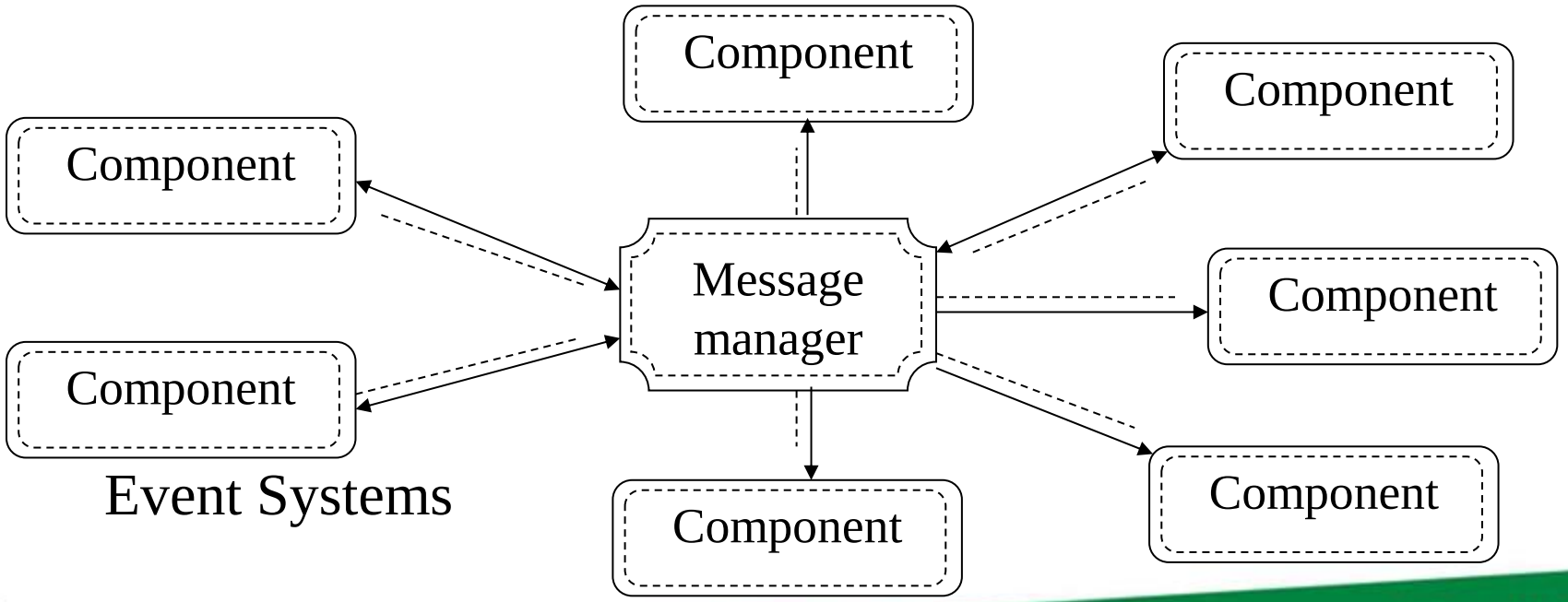
A Catalogue of Software Architectural Styles

Mary Shaw and David Garlan



Independent Component Architectures

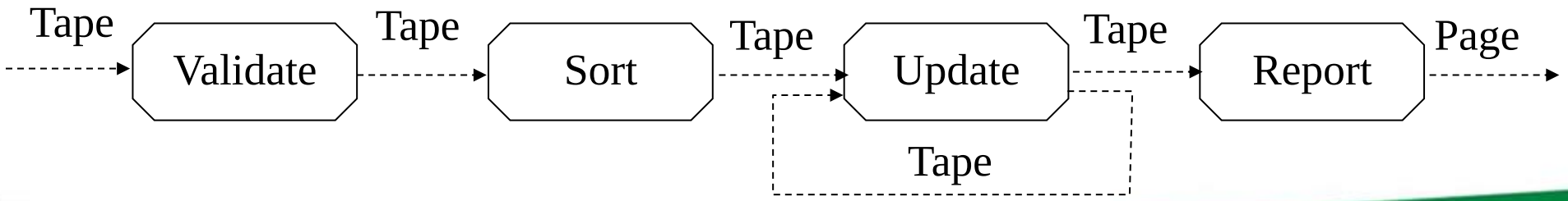
- Components:
  - a number of independent processes or objects
- Connectors:
  - communicate through messages



Data Flow Architectures

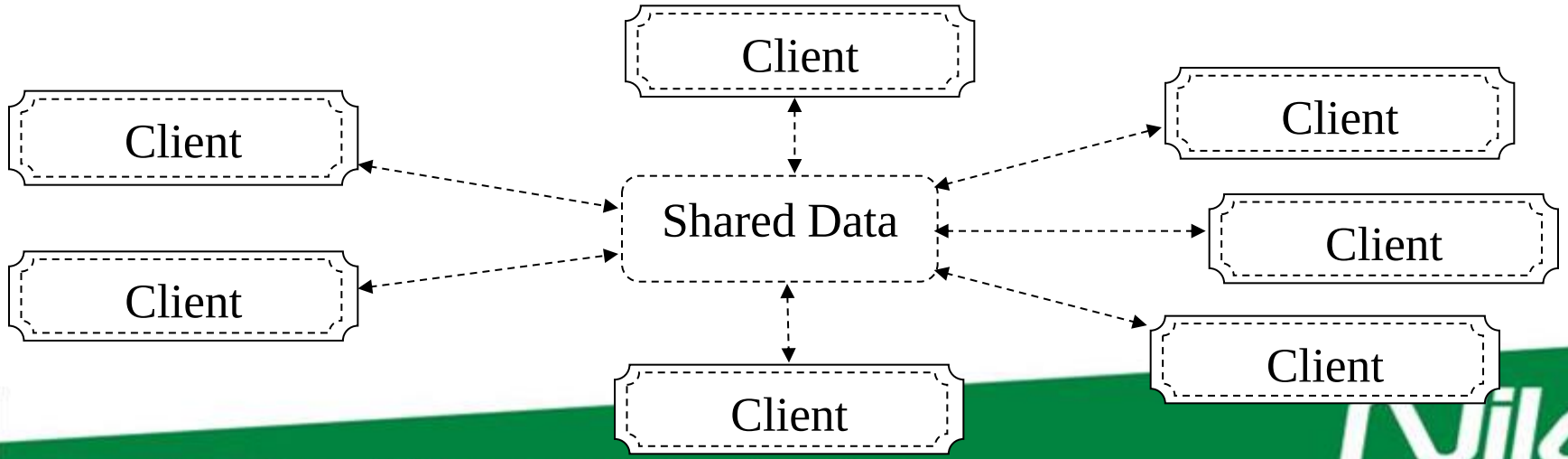
Batch Sequential Processing Style

- Components:
  - Called processing steps
  - Independent programs
- Connectors:
  - Data and control passing from step to step



Data Centred Architectures

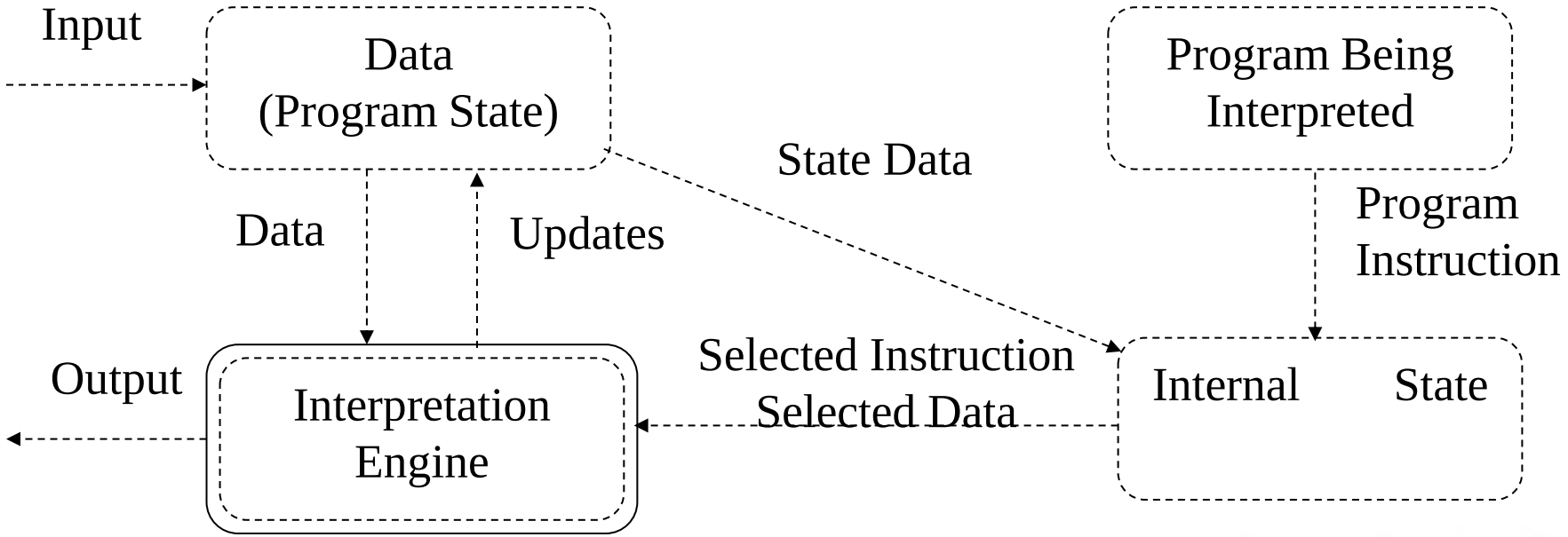
- Components:
  - Widely accessible data stores
    - Repository: passive
    - Blackboard: active
  - Clients - computation components
- Connectors:
  - Repository: independent thread of control
  - Blackboard: activated by the blackboard



Virtual Machine Architecture

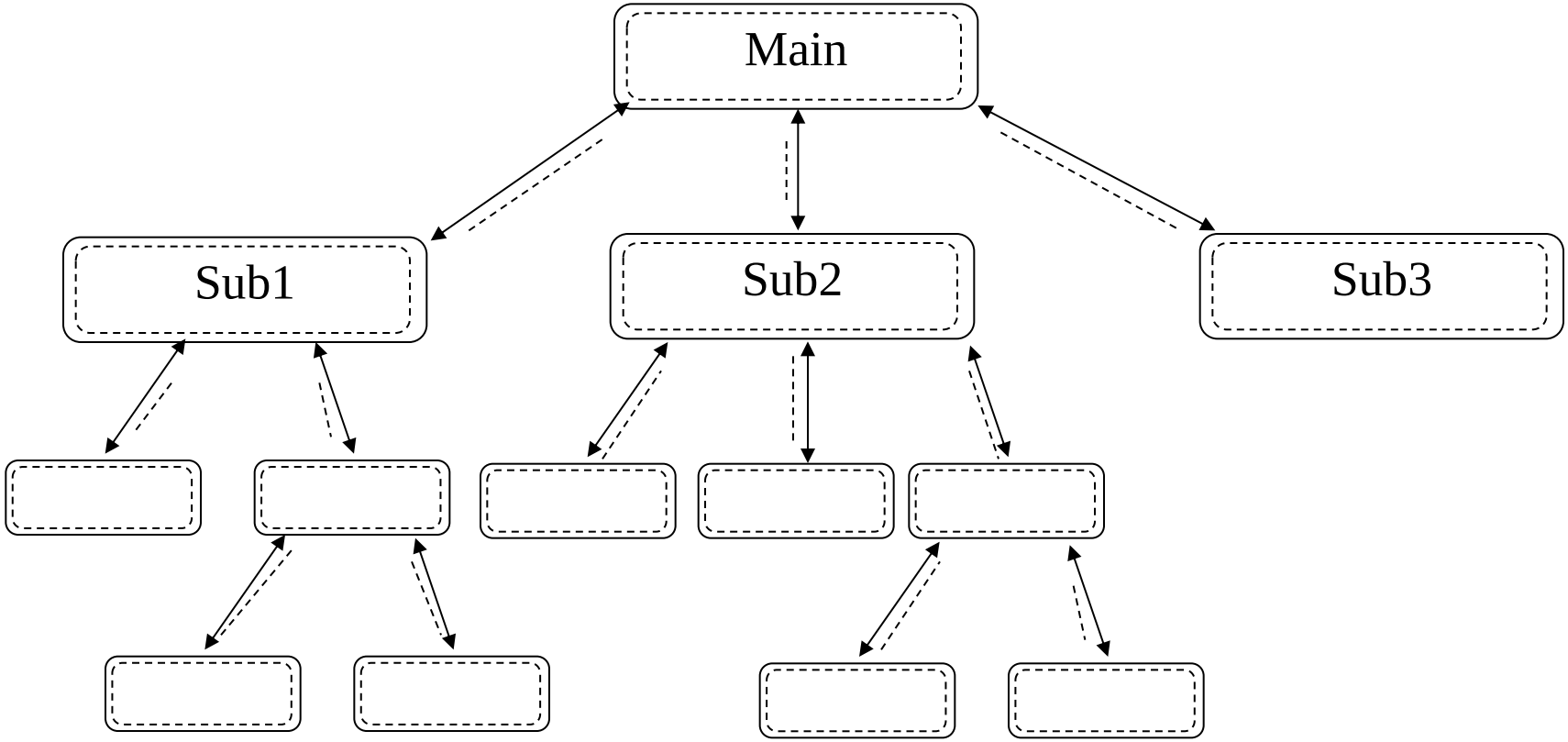
Portability is the main advantage.

For example, the Java language is built to run on top of the Java Virtual Machine, which allows the language to be platform independent.



Call-and-Return Architectures

Main-Program-And-Subroutine Architectures



Which architectural style should be used?

- The selection of software architectural style should take into consideration of two factors:
  - The nature of computation
    - The nature of the computation problem to be solved, e.g. the input/output data structure and features, the required functions of the system, etc.
  - The quality concerns
    - The quality concerns that the design of the system must take into consideration, such as performance, reliability, security, maintenance, reuse, modification, interoperability, etc.

Heuristic rules for style selection

Style	Nature of Computation	Quality Concerns
Data flow	The input and output data of the problem to be solved are both well defined and easily identified. The computation is to produce output from the input as the direct result of sequentially transforming an input in a time-independent fashion.	Integratability from relatively simple interface between components. Reusability by interchanges of components to change functionality.
Batch sequen-tial	There is a single output operation that is the result of reading and processing a single collection of input. The processing of the input consists of a number of sequentially connected intermediate transformations.	Reusability and modifiability, especially when performance is not a major concern and synchronisation related complexity can be reduced.
Pipe-and-filter	The computation involves transformations on continuous streams of data. The transformations are incremental, i.e. applied on the elements of the stream of data. One transformation can begin before the previous step has completed.	Scalability, especially to process the input data of unbounded length of data stream. Performance, to respond to the input as soon as an element of the input data is fed into the system before the whole stream of data is obtained.

Style	Nature of Computation	Quality Concerns
Call and Return	The order of computation is fixed. Components cannot make useful progress while waiting for the results of requests to other components.	Modifiability, integratability, reusability.
Layered systems	The computational 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. Reuse of an already developed computing infrastructure layer, such as operating system, network packages, etc.
Main-program-and-subroutine with shared data	The problem to be solved at a higher level of abstraction can be decomposed into a number of smaller problems at a lower level of abstraction that can be solved and their results can be put together to solve the higher level problem.	Performance, especially the space of memory used to store intermediate results can be significantly reduced without duplication when shared with components, and time to access the intermediate results can be reduced through direct access rather than other means of communication. Reusability, especially when a subroutine solves a problem that reoccurs in the application domain frequently.

Style	Nature of Computation	Quality Concerns
Data abstraction	Computation is based on a relatively fixed variety of entities and each of these entities has a fixed number of operations.	Reusability, especially the reuse of the components that represent those computational entities that reoccur frequently in the application domain. Modifiability, especially when the representations of the computational entities are likely to change.
Abstract data type		Integratability, especially through careful attentions to the design of the interfaces between the components.
Object-oriented	In addition to the common features of the computation in all data abstraction sub-styles, the entities in the system have commonalities and fall into a nature hierarchy of inheritance.	Reusability and modifiability as in all data abstraction sub-styles. Moreover, it also aims at reusability and modifiability when the types of entities are relatively stable in the application domain while the system's functionality is likely to be changed, especially enhanced, in the future.

Style	Nature of Computation	Quality Concerns
Independent components	One component can continue to make progress some what independent of the states of other components. The system is to be run on a network of computer systems.	Modifiability, especially when performance tuning via reallocating work among processes and reallocating processes to computers is necessary. Performance, especially to achieve maximal utilisation of the computational resource is important.
Communicating processes	Message passing is sufficient as an interaction mechanism.	
Event-based implicit invocation	Computations are triggered by a collection of events. Event originators and event processors can be decoupled.	Flexibility for the modification system's functionality especially enhancement of functionality. Scalability in the sense of adding processes that are triggered by the events already detected/ signaled in the system. Modifiability for change the way that events are processed.

Style	Nature of Computation	Quality Concerns
Client-server	Computation can be reviewed as providing services to a number of users who share the access to the information and/or computation resources. Computation tasks can be divided between instigators of service requests and executors of those requests or between producers and consumers of data.	Scalability in terms of the number of users who can have access to the system simultaneously as well as from a wide geographic area. Modifiability, especially when users' interface may change and/or new functionality is likely to be added into the system as new services.
Data-centred	The central issue in the computation is the storage, representation, management and retrieval of a large amount of interrelated long-lived data. The data are usually highly structured. The order that components are executed in is determined by a stream of incoming requests to access/update the data.	Scalability in terms of the amount of data to be stored and processed by the system. Modifiability to change the producing and processing the data while the structure of the data remains relatively stable.

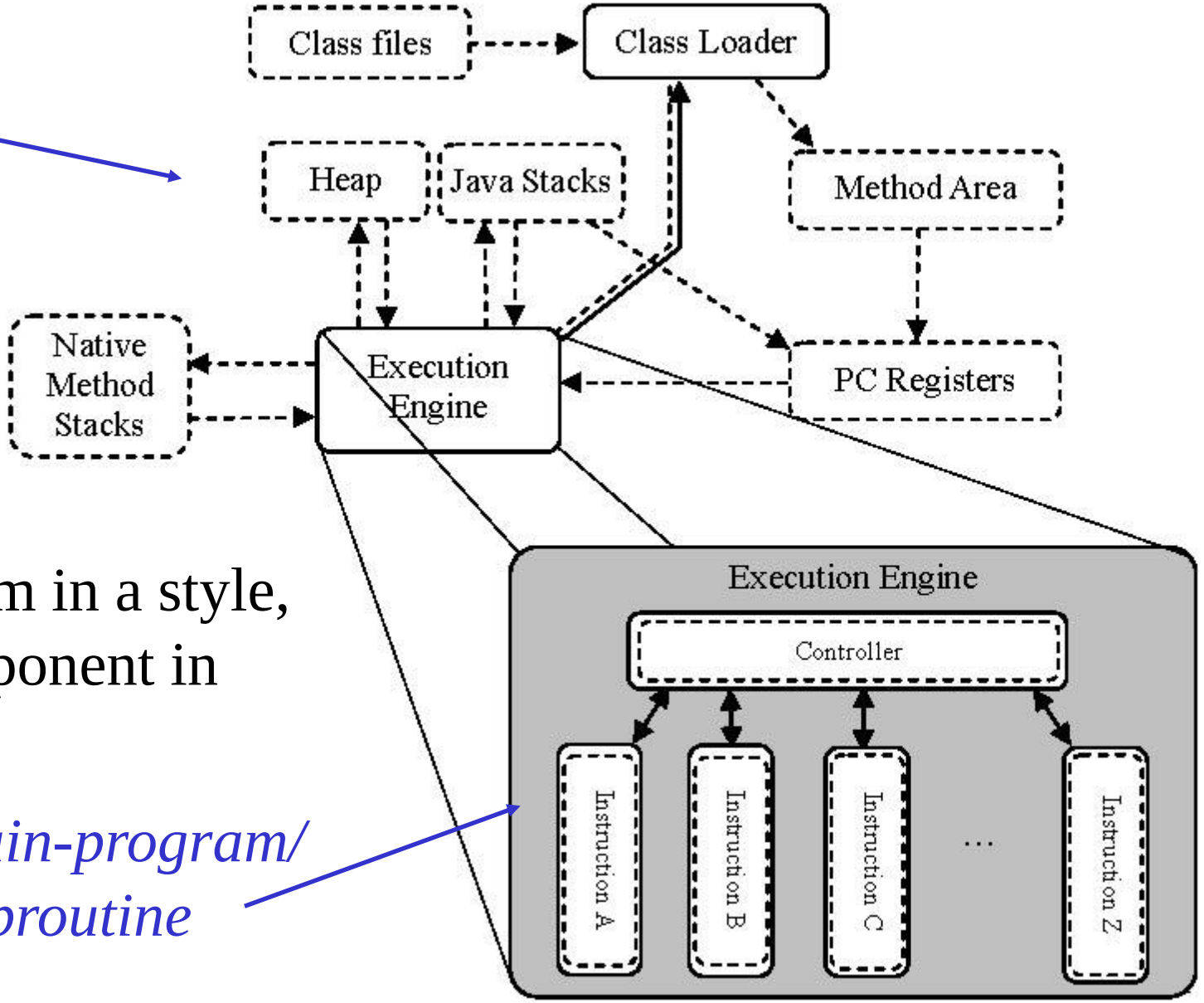
Style	Nature of Computation	Quality Concerns
Virtual machine	Data consists of two parts: one is the information to be processed and the other controls how the first part is to be processed.	Portability of the processing of the data in different hardware and software platforms.

Combinations of Styles

- The design of a software system may need to combine different styles to solve the design problem.
- Heterogeneous style
  - Systems that are not in a single style
- How styles can be combined together:
  - Hierarchically heterogeneous styles
  - Simultaneously heterogeneous styles
  - Locationally heterogeneous styles

Hierarchical heterogeneous styles

Virtual machine

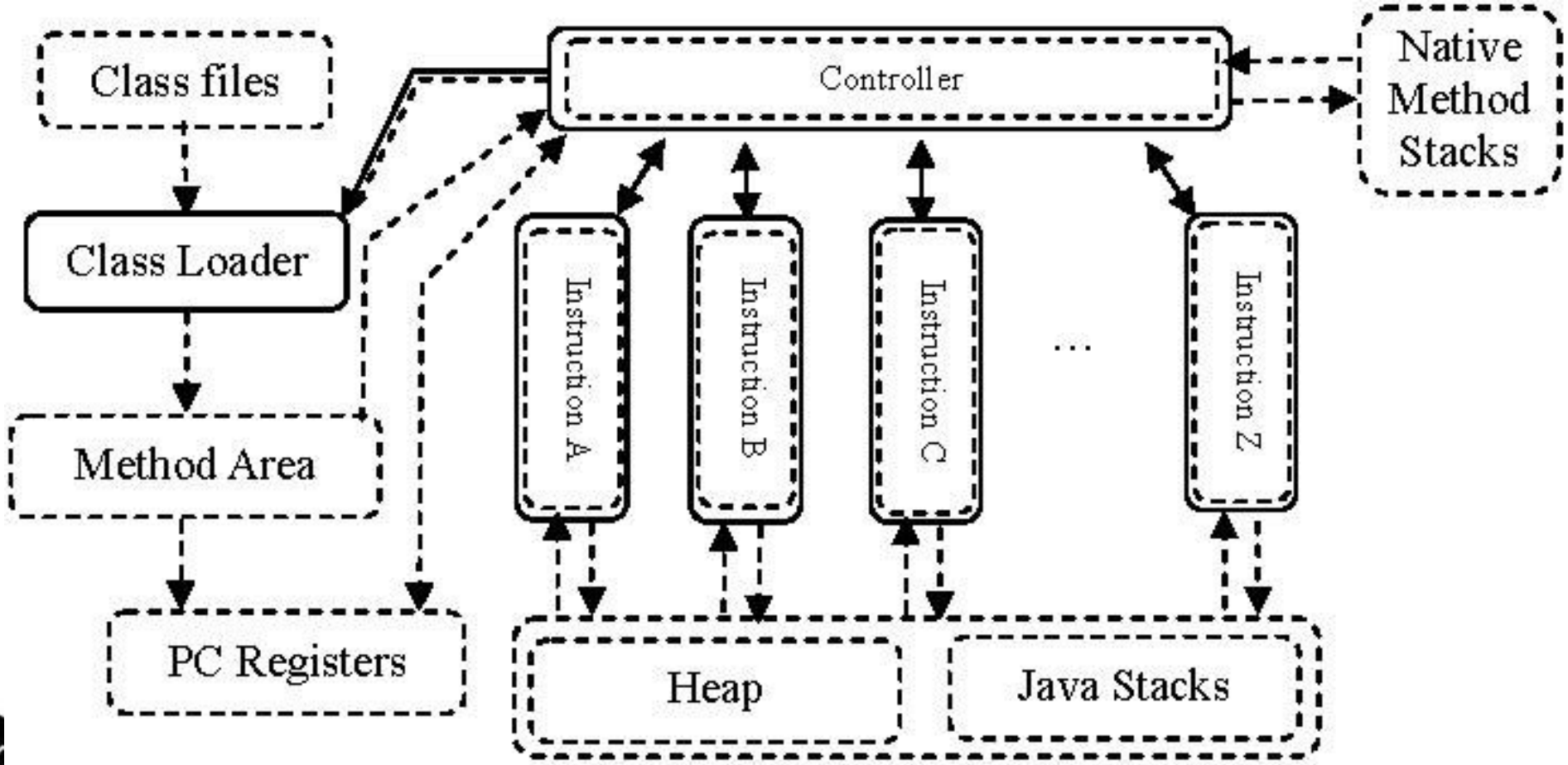


Whole system in a style, while a component in another style

Main-program/ Subroutine

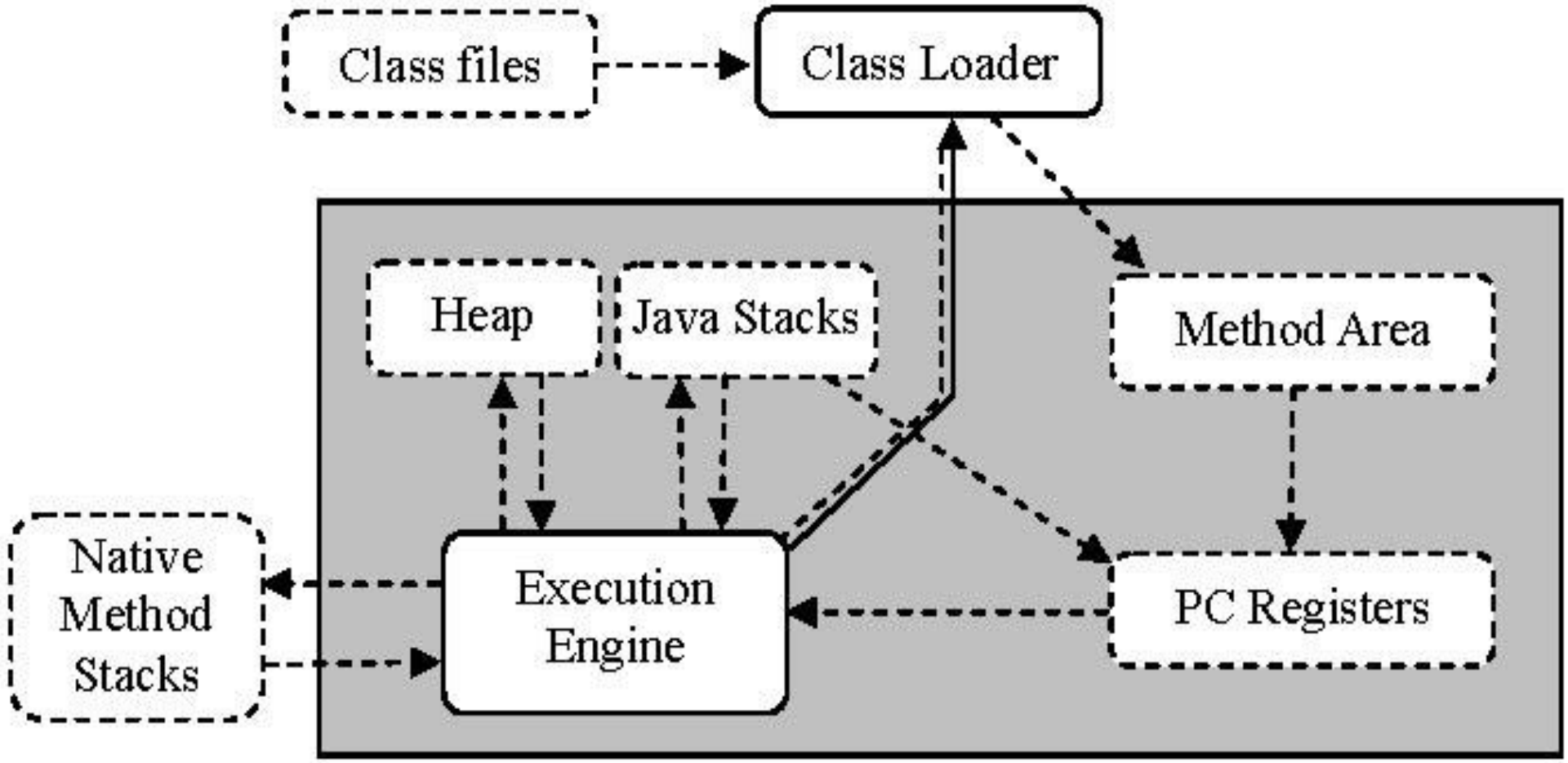
Simultaneously heterogeneous styles

The architecture of a system can be described as a number of different styles



Locationally heterogeneous styles

Different subsets of the components and connectors fall into different architectural styles



Case Study 1: KFV

The Problem: Keyword Frequency Vector

The keyword frequency vector (KFV) of a text file is a sequence of pairs of keywords and their frequency of appearance in the text

- A good representation of the contents of a text
- Widely used in information retrieval
- Can be extracted from texts automatically
- Small words (such as ‘a’, ‘the’, ‘is’, ‘it’) are removed from the vector
- The same word of different forms should be treated as one

Example of KFV

Input

The keyword frequency vector of a text file is a sequence of pairs of keywords and their frequency of appearance in the text. It is a good representation of the contents of the text. Keyword frequency vectors are widely used in information retrieval. For example the following is the keyword frequency vector of this paragraph.

Output

Word	Frequency
keyword	5
frequency	4
text	4
vector	4
appearance	1
content	1
example	1
file	1
follow	1
good	1
information	1
pair	1
paragraph	1
representation	1
retrieval	1
sequence	1
use	1
widely	1

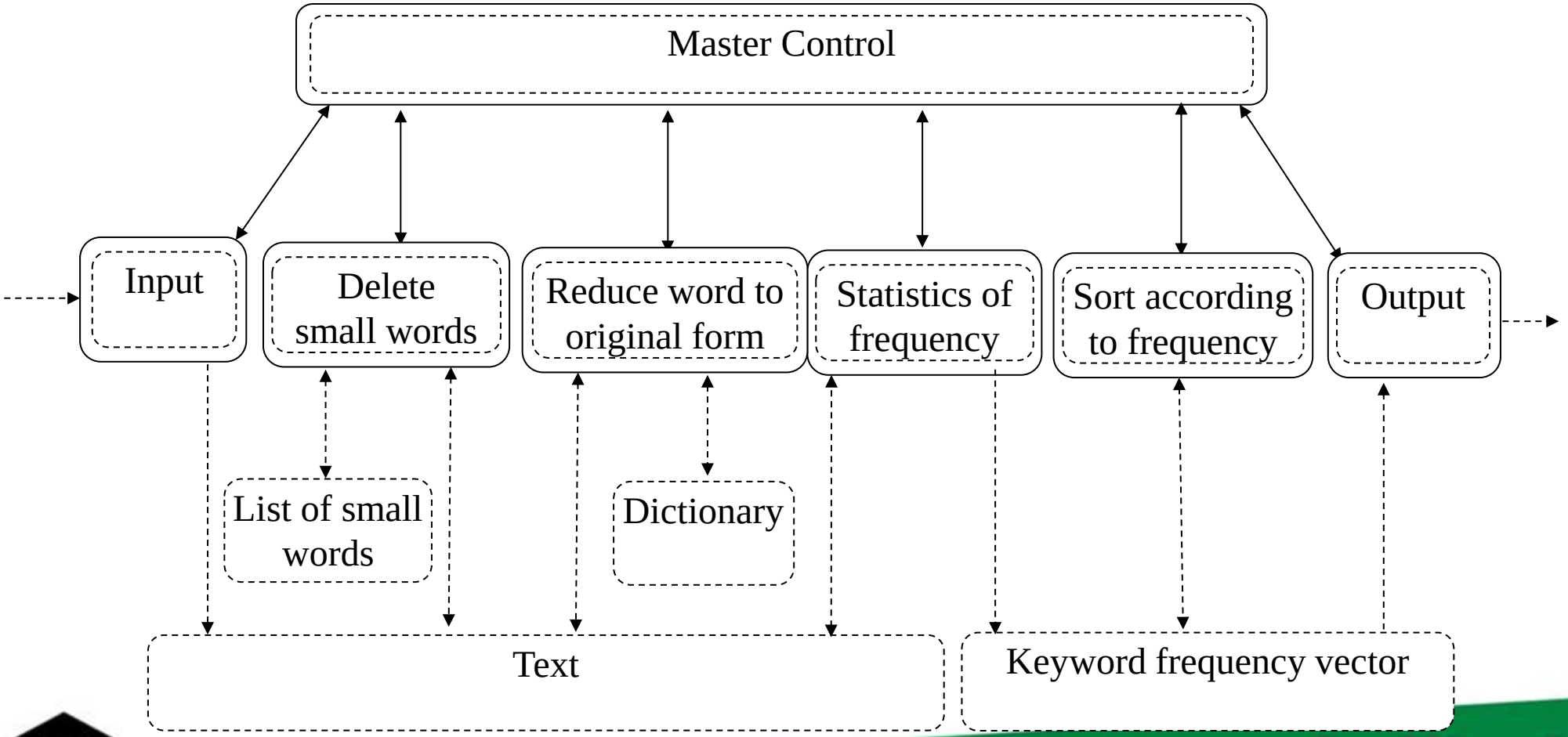
Analysis of the Design Problem (1)

Usage Scenarios	Natural of Computation
Processing web pages as a part of web search engine	• Process the whole html file • Treat synonyms as the same word • Store the KFV explicitly in a database • The texts are in ASC II, Unicode-8, or Unicode-16 format. • The texts contain html tags, URLs, etc.
Processing archived documents in an online library	• Process incrementally paragraph by paragraph as it is read from the input device • The texts are coded in ASC II format
Processing texts as a part of office automation tool	• Process on demand when required • Input is in word processor internal coding • Store the KFV implicitly as links to the texts • Process implicitly and to be interactive

Analysis of the Design Problem (2)

	Web	Library	Office
Scalability	web page	book	text file
Performance	Batch processing	Batch processing	Interactive
Interoperability	none	none	with office tools
Modifiability	• *wrt changes in input data representation* • *wrt processing modes* • *wrt internal data representation*		
Reusability	Reuse components to develop software in different usages, i.e. • As web page processing, • As library database, and • As office automation tool		

Design 1:
Main Program/Subroutines with Shared Data



Functions of the Components

(1) input:

- To get the input text from input device or any other source of information
- To store the text into internal memory in an appropriate format
- *The design of internal format will be determined by detailed design*

(2) delete small words:

- The small words contained in the text are deleted from the text as it is stored in the internal memory
- It will use a list of small words

(3) reduce word to its original form:

- Each word left in the text are then reduced to its original form
  - 'Architectures' ⇒ 'architecture'
  - 'Calculi' ⇒'calculus'
  - 'Followed' ⇒'follow'

A dictionary will be used

(4) statistics of frequency:

- To count the occurrences of a word in the text to generate a sequence of pairs comprising the word and its frequency
- *This sequence of pairs is not necessarily ordered according to the frequency, but may be in the alphabetic order of keywords*
- *The result will be stored in another memory storage*

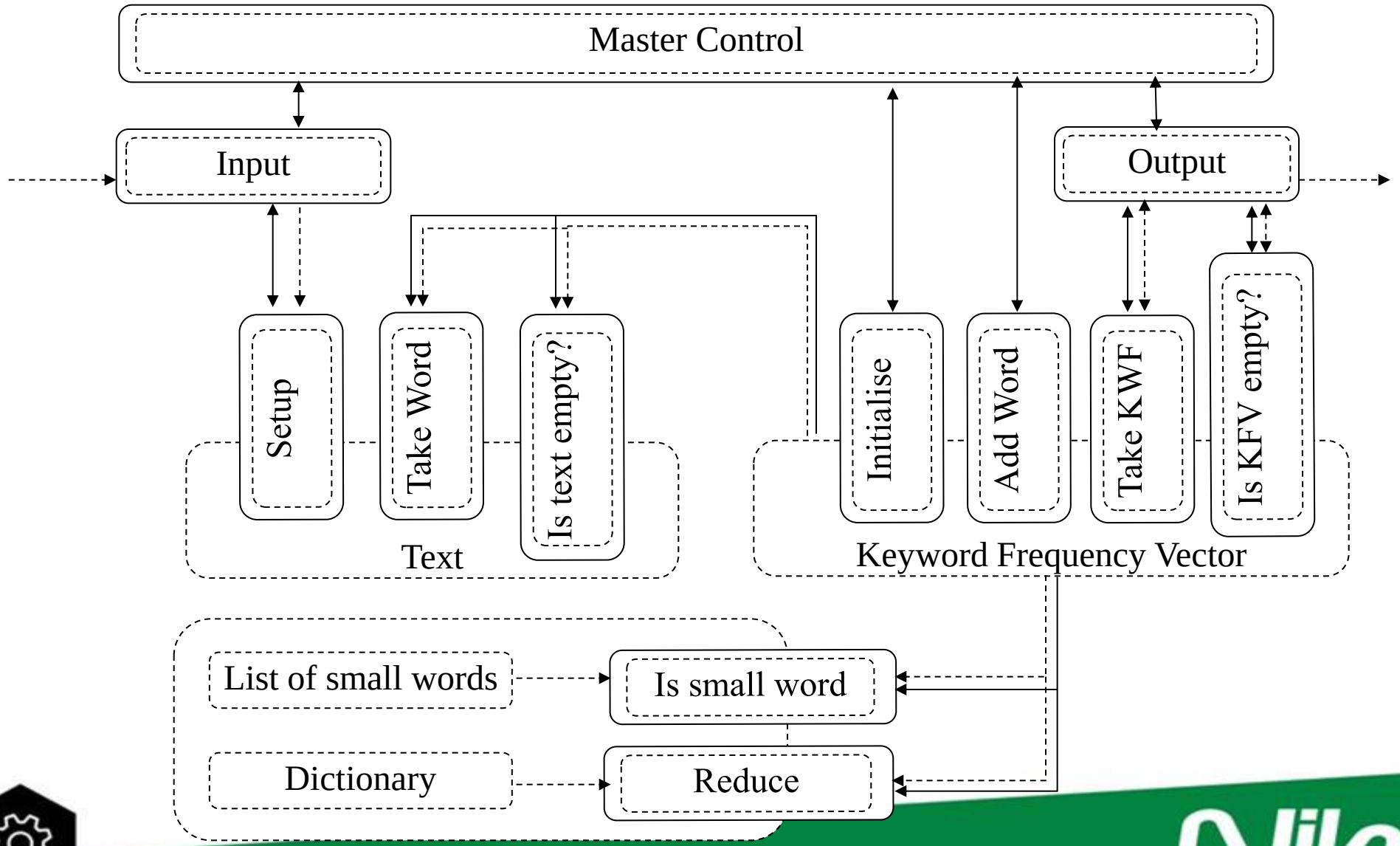
(5) sort according to the frequency:

- Sort the sequence of pairs of keywords and their frequencies into an order according to the frequency

(6) output:

- Translate the keyword frequency vector into required output format
- Output to the device

Design 2: Abstract Data Type



(1) Word ADT:

(a) Is-small-word:

- A Boolean function that checks if the parameter is a small word.
- It returns TRUE if the word is listed in a list of small words, otherwise it returns FALSE.

(b) Reduce:

- A function on words
- It changes a word to its original form according to a dictionary of words and returns back.

(2) Text ADT:

(a) Setup:

- get the text from the input component
- translate the text into an internal format
- stores the text in its internal data storage

(b) Take-word:

- a function that returns one word in the text and deletes it from its internal data storage.

(c) Is-text-empty:

- A Boolean function that returns TRUE if the internal storage is empty, otherwise returns FALSE when it contains at least one word.

(3) Keyword Frequency Vector ADT:

(a) Add-word: adds a word to the keyword frequency vector

- Calls is-small-word function of the word ADT
- If the function returns TRUE, then do nothing
- ELSE calls the reduce function of the word ADT and searches the keyword frequency vector
- If the vector already contains the word, then its frequency is added by 1, else the keyword is added into the vector with frequency 1

(b) Is-KFV-empty:

- A Boolean function that returns TRUE if the vector is empty, otherwise, it returns FALSE

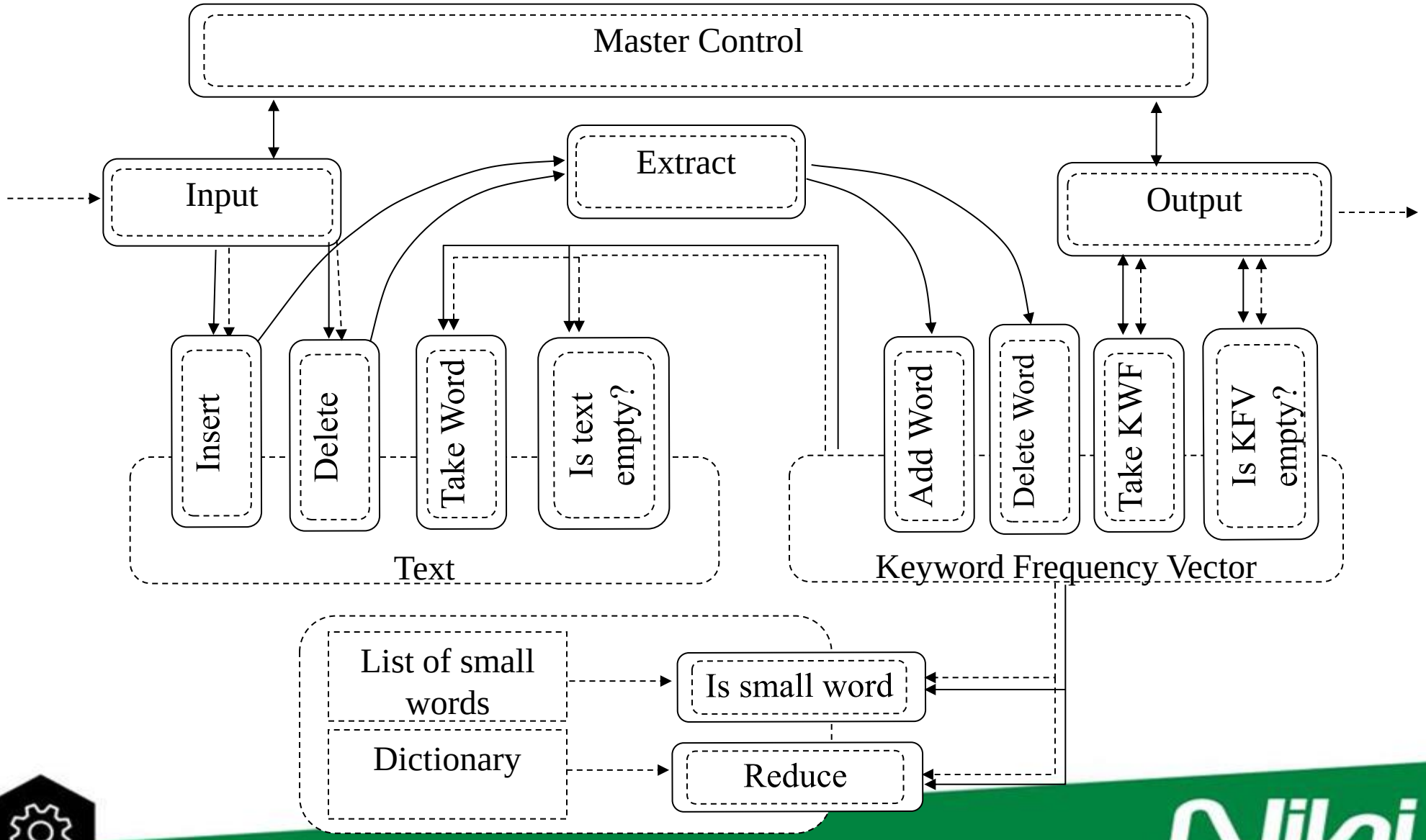
(c) Take-KWF:

- Return the frequency and keyword of highest frequency
- Delete the keyword from the vector

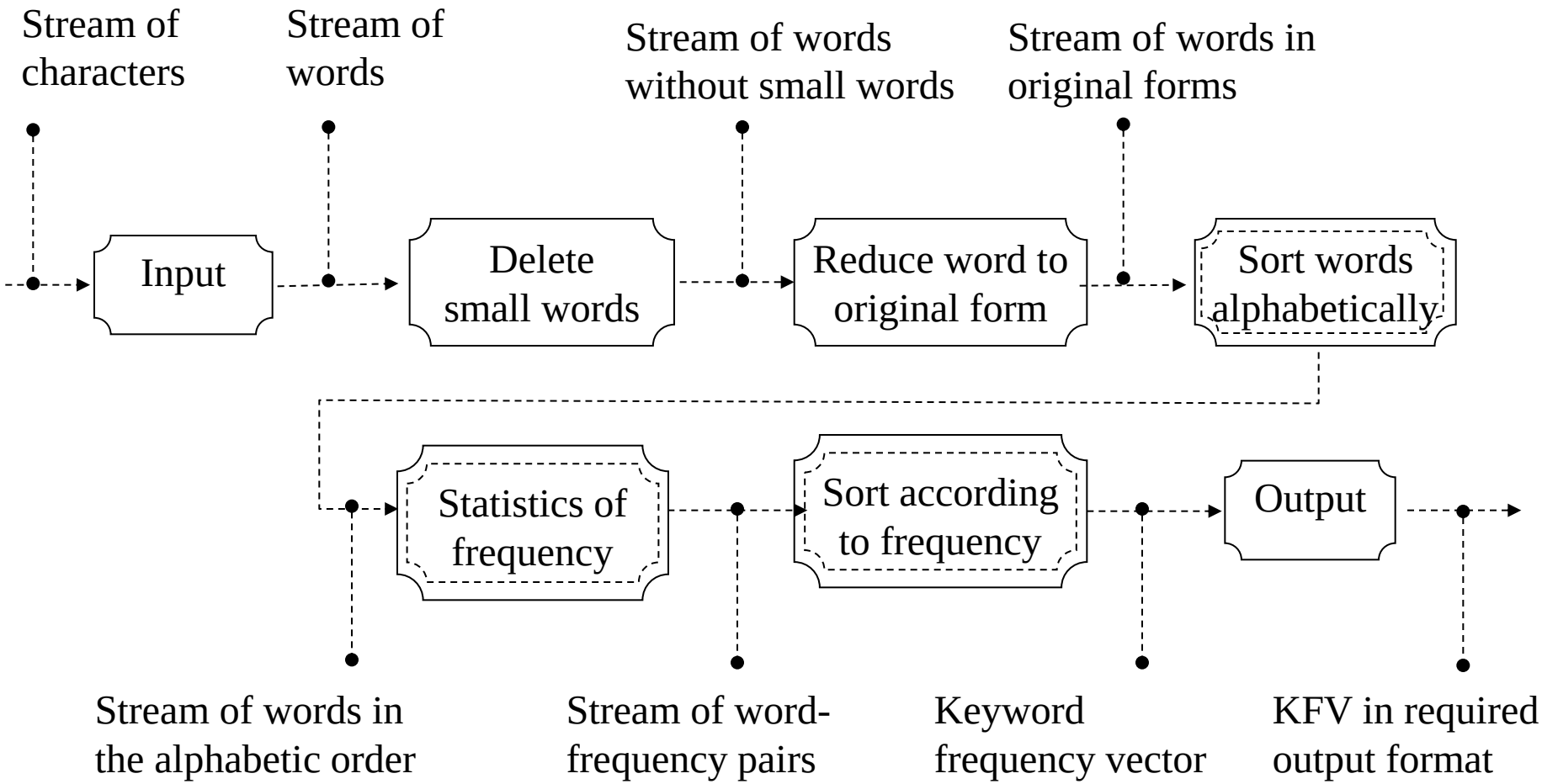
(d) Initialise:

- It initialises the internal representation of the vector

Design 3: Implicit invocation



Design 4: Pipe-and-filter



Filters

(1) *Input*: Takes the stream of characters and breaks it down to a stream of words.

(2) *Delete small words*: Removes the small words in the input stream of words.

(3) *Reduce words to original forms*: Changes each word in the stream of words into their original forms.

(4) *Sort words alphabetically*: Takes the stream of words and sort it into alphabetical order.

(5) *Count the frequency*: Count the occurrences of each word in the stream and generates a stream of keyword-frequency pairs.

(6) *Sort vector according to frequency*: Sort the stream of keyword-frequency pairs according to frequency.

(7) *Output*: Takes a stream of keyword-frequency pairs that is sorted according to the frequency and generates a keyword frequency vector in the required output format.

Case Study 2: KWIT

The Problem: Keyword in Context

- ***Input:***
  - An ordered sequence of lines of text.
  - Each line is an ordered sequence of words
  - each word is an ordered sequence of characters.
- ***Output:***
  - lines are ‘circularly shifted’ by repeatedly removing the first word and appending it at the end of the line.
  - outputs a listing of all circular shifts of all lines in alphabetical order.

Example of Keyword in Context

- ***Input***: sequence of lines

  An introduction to software architecture

  Key word in context

- ***Output***: circularly shifted, alphabetically ordered lines

  An introduction to software architecture

  Architecture an introduction to software

  Context key word in

  In context key word

  Introduction to software architecture an

  Key word in context

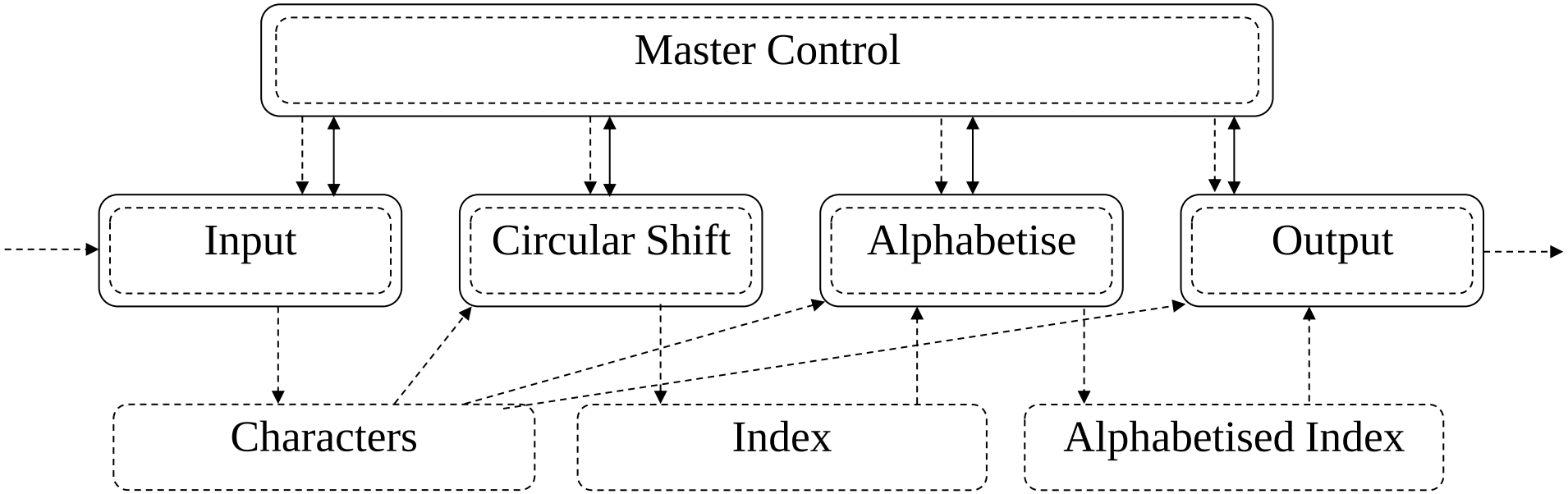
  Software architecture an introduction to

  To software architecture an introduction

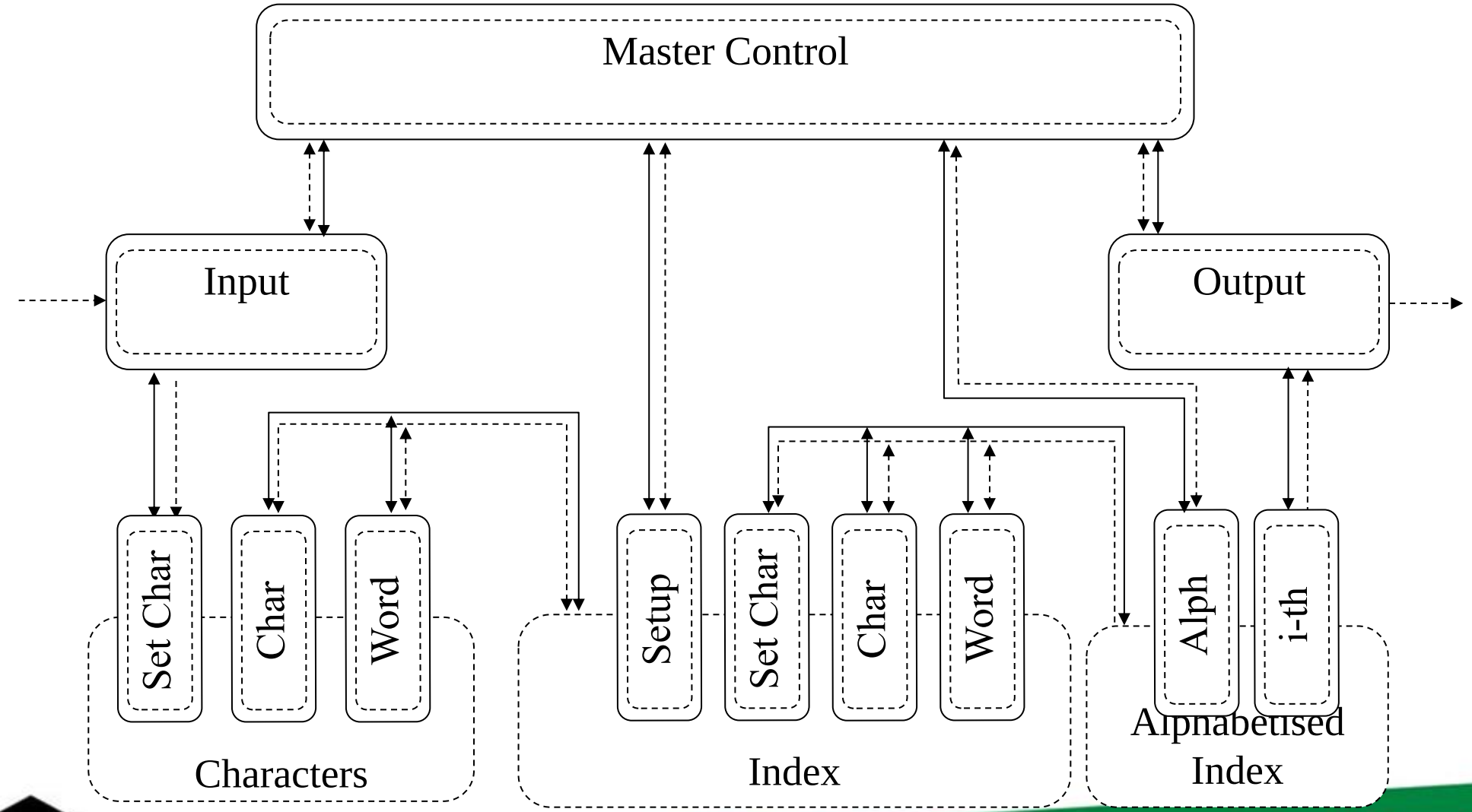
  Word in context key

Design 1:

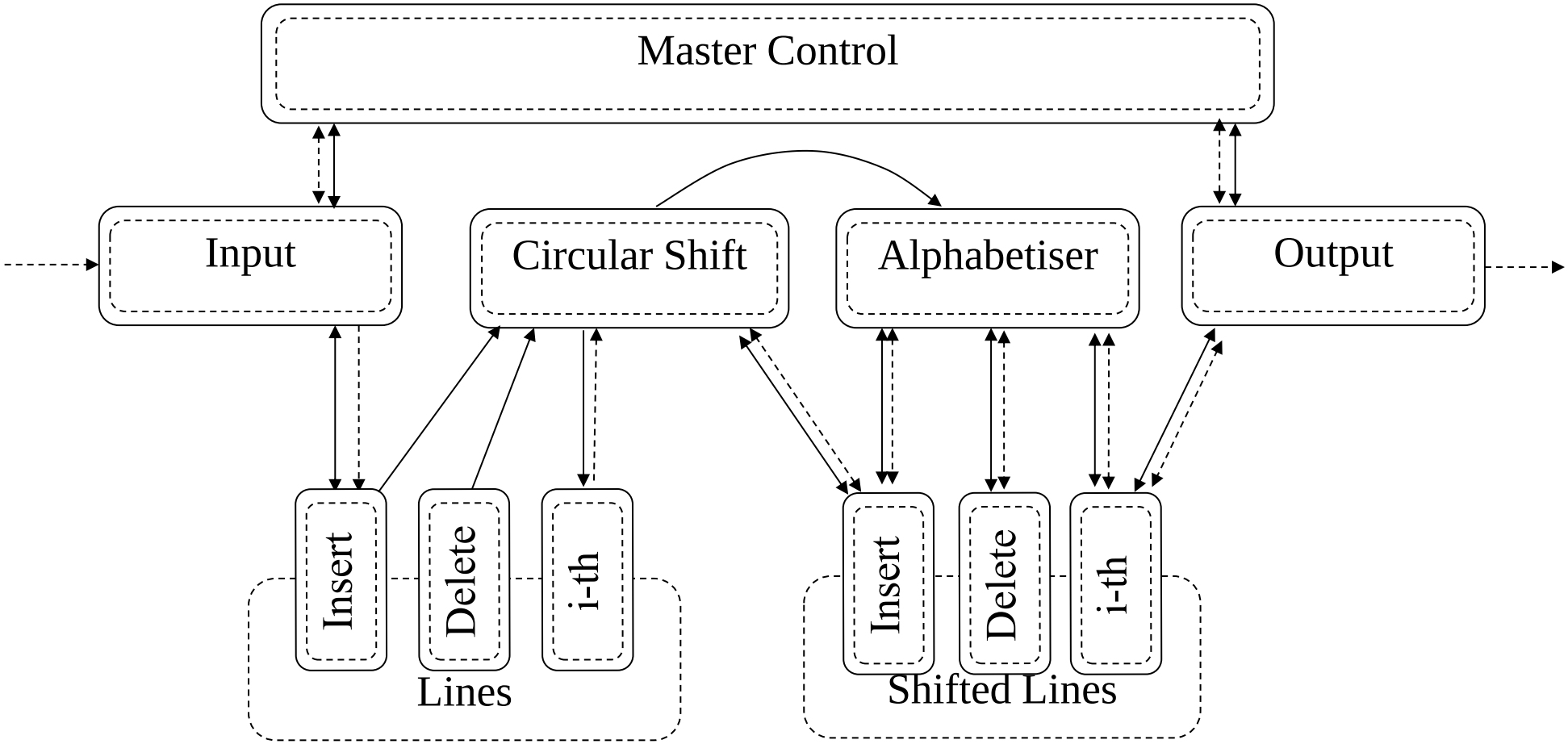
Main Program/Subroutines with Shared Data



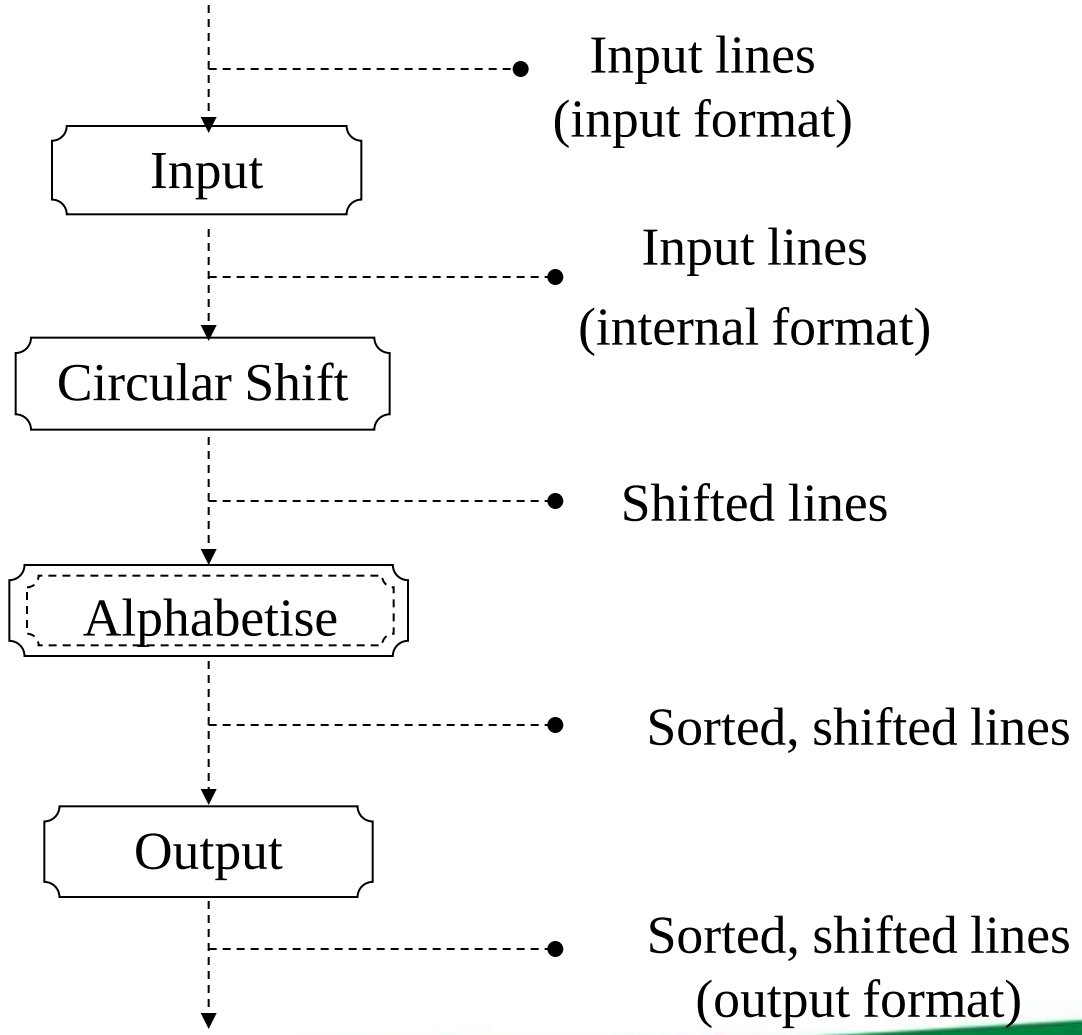
Design 2: Abstract data types



Design 3: Implicit Invocation



Design 4: Pipe-and-Filter



Further Readings

[1] Zhu, H., Software Design methodology. Chapter 7, pp172-198.

[2] Shaw, M and Garlan, D., Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996. *Chapter 3*: Case studies, pp33~68.

[3] Bass, L., Clements, P. and Kazman, R., Software Architecture in Practice, Addison Wesley, 1998.

(a) *Chapter 5*: Moving From Qualities to Architecture: Architectural Styles, pp93~122

(b) *Chapter 7*; The World Wide Web, pp145~163

(c) *Chapter 8*: CORBA, pp165~187