Design and Software Architecture

(Chapter 4)

Outline

- What is software design/software architecture
- How can a system be decomposed into modules
- What is a module's interface
- Main relationships among modules
- Software design techniques and information hiding

The *Software Design* activity

- Decomposes system into parts, assigns responsibilities, ensures that parts fit together to achieve a global goal
- Bridge between requirements and the implementation of the software
- Produces a Software Design Document
  - describes system decomposition into modules
- Often a *software architecture* is produced prior to a software design

Software architecture

- Shows gross structure and organization of the system to be defined
- Its description includes description of
  - main components of a system
  - relationships among those components
  - rationale for decomposition into its components
  - constraints that must be respected by any design of the components
- Guides the development of the design

Two important goals

- Design for change (Parnas)
  - designers tend to concentrate on current needs
  - special effort needed to anticipate likely changes
- Product families (Parnas)
  - think of the current system under design as a member of a program family

Sample likely changes? (1)

- Algorithms
  - e.g., replace inefficient sorting algorithm with a more efficient one
  - replace internal with external sorting
- Change of data representation
  - ≈17% of maintenance costs attributed to data representation changes (Lientz and Swanson, 1980)
  - e.g., from single-linked list to double-linked list

Sample likely changes? (2)

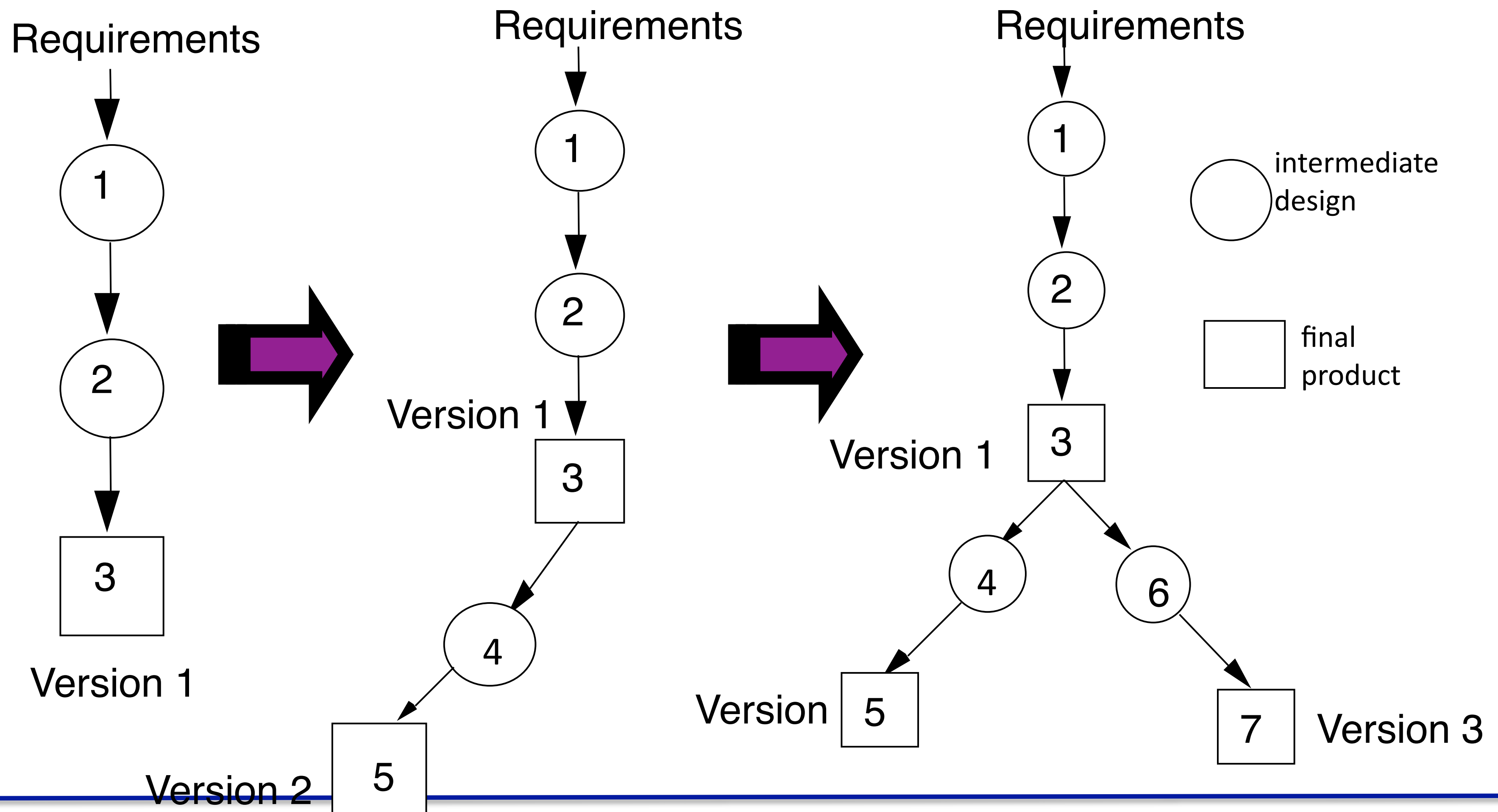
- Change of underlying abstract machine
  - new release of operating system
  - new optimizing compiler
  - new version of DBMS, etc.
- Change of peripheral devices
- Change of "social" environment
  - E.g., new tax scheme, change in health care regulations
- Change due to development process (transform prototype into product)

Product families

- Different versions of the same system
- Examples
  - family of mobile phones
    - members of the family may differ in network standards, end-user interaction languages, …
  - facility reservation system
    - for hotels: reserve rooms, restaurant, conference space, …, equipment (video beamers, overhead projectors, …)
    - for a university
      - many functionalities are similar, some are different (e.g., facilities may be free of charge or not)

The wrong way: Sequential completion

(modify existing software to get next member products)



How to do better

- Anticipate definition of all family members
- Identify what is common to all family members, delay decisions that differentiate among different members

Module

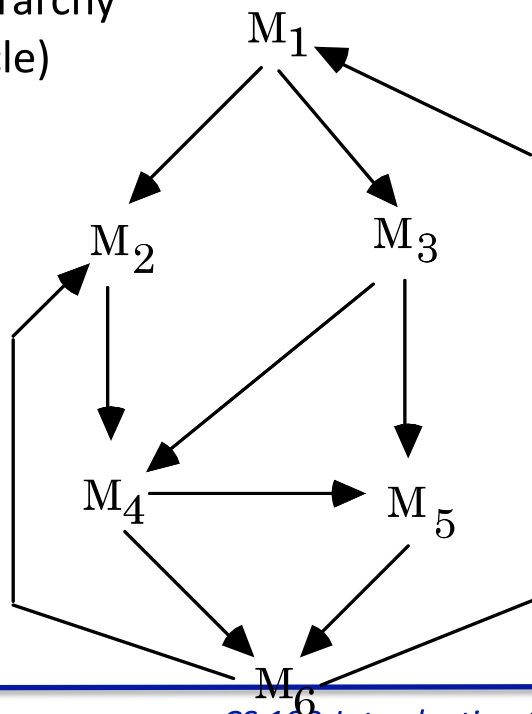
- A well-defined component of a software system
- A part of a system that provides a set of services to other modules
  - Services are computational elements that other modules may use

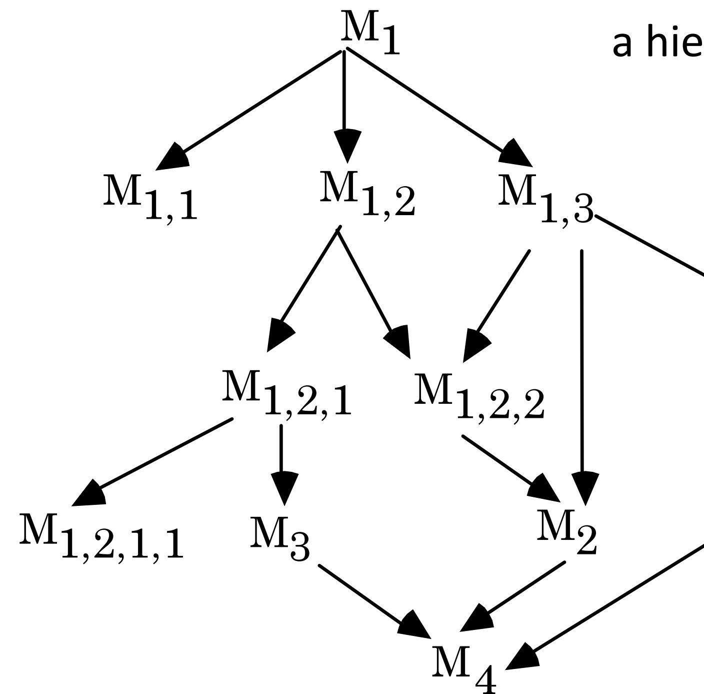
Questions

- How to define the structure of a modular system?
- What are the desirable properties of that structure?

Hierarchy: Definition

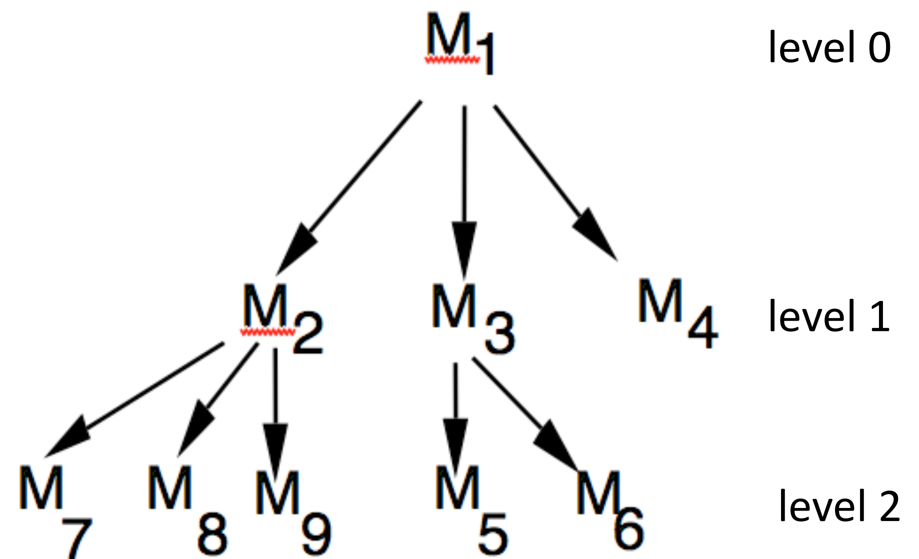
- Module relations represented as graphs
- A hierarchy is a DAG (directed acyclic graph)





Hierarchies and levels of abstraction

- A hierarchy organizes the modular structure through *levels of abstraction*
- Each level defines an *abstract (virtual) machine* for the next level



The USES relation

- A uses B
  - A requires the correct operation of B
  - A can access the services exported by B through its interface
  - it is “statically” defined
  - A depends on B to provide its services
    - example: A calls a routine exported by B
- A is a client of B; B is a server

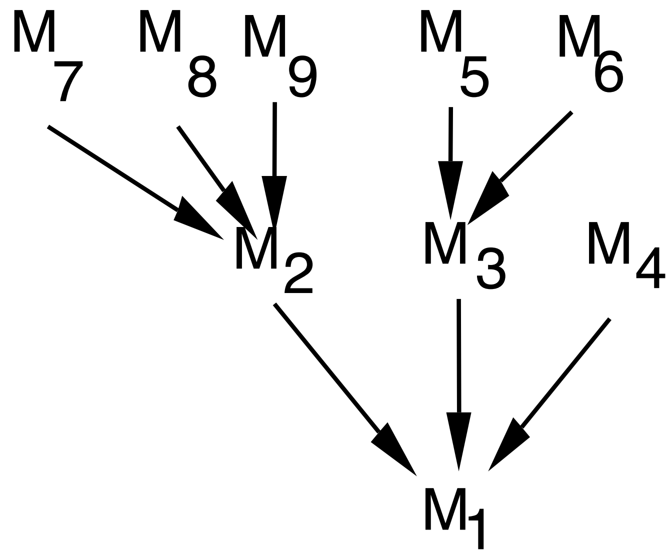
Desirable property

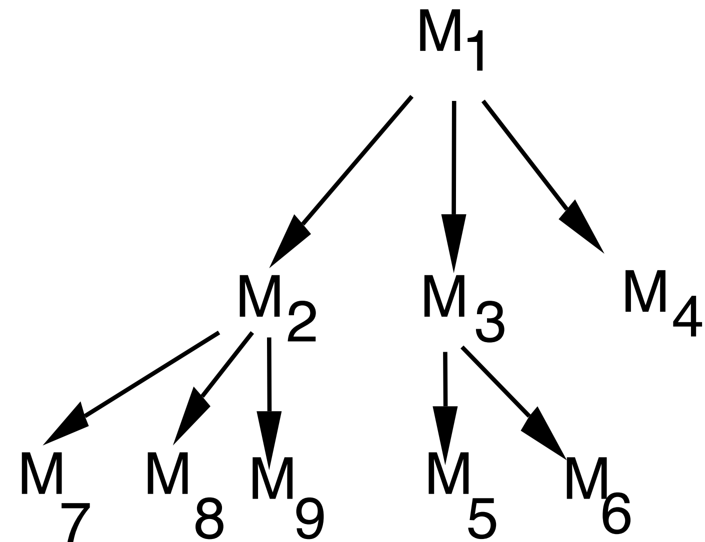
- USES should be a hierarchy
  - Otherwise “nothing works until everything works”
- Hierarchy makes software
  - easier to understand
    - we can proceed from leaf nodes (who do not use others) upwards
  - easier to build
  - easier to test

The IS_COMPONENT_OF relation

- Used to describe a higher level module as constituted by a number of lower level modules
- A IS_COMPONENT_OF B
  - B consists of several modules, of which one is A
- B COMPRISES A
- $M_{S,i}=\{M_k | M_k \in S \wedge M_k \text{ IS_COMPONENT_OF } M_i\}$
  we say that $M_{S,i}$ IMPLEMENTS M_i

A graphical view





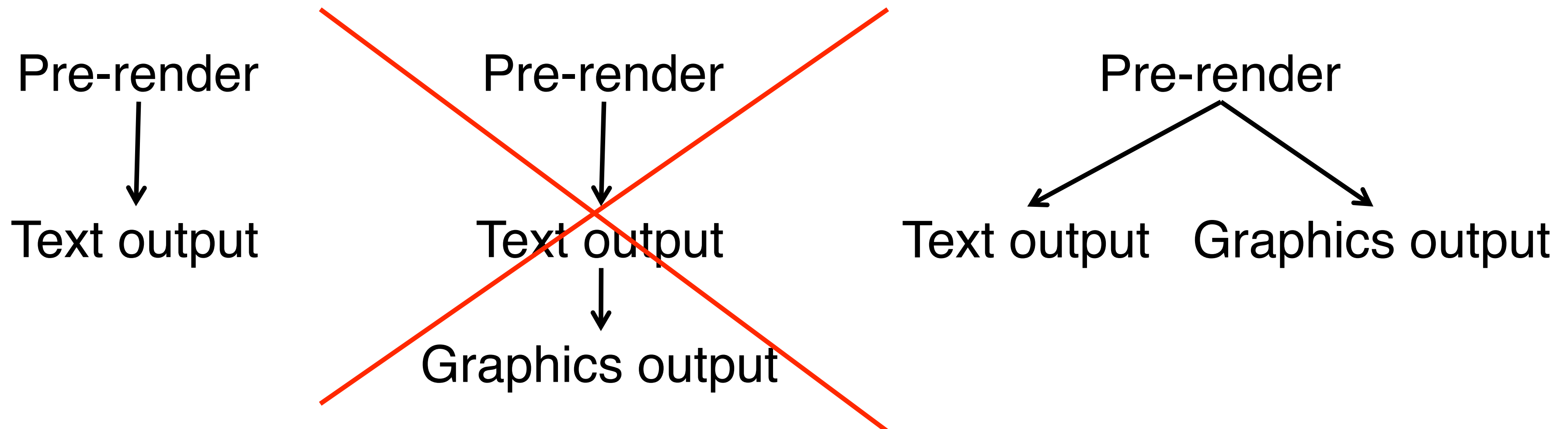
(IS_COMPONENT_OF)

(COMPRISES)

They are a hierarchy

Product families

- Careful recording of (hierarchical) USES relation and IS_COMPONENT_OF supports design of program families



Interface vs. implementation (1)

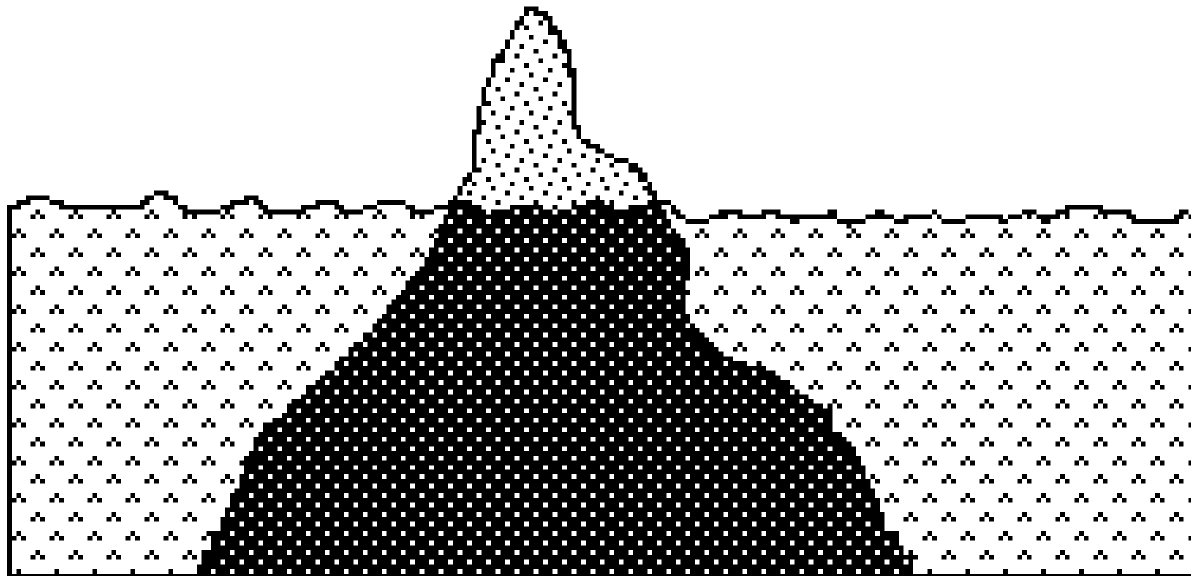
- To understand the nature of USES, we need to know what a used module *exports* through its *interface*
- The client *imports* the resources that are exported by its servers
- Modules *implement* the exported resources
- Implementation is *hidden* to clients

Interface vs. implementation (2)

- Clear distinction between interface and implementation is a key design principle
- Supports separation of concerns
  - clients care about resources exported from servers
  - servers care about implementation
- Interface acts as a contract between a module and its clients

Interface vs. implementation (3)

interface is like the tip of the iceberg



Information hiding

- Basis for design (i.e. module decomposition)
- Implementation secrets are hidden to clients
  - C++, Java ?
- They can be changed freely if the change does not affect the interface
- Golden design principle
  - *INFORMATION HIDING*
    - *Try to encapsulate changeable design decisions as implementation secrets within module implementations*

How to design module interfaces?

- Example: design of an interpreter for language MINI
  - We introduce a SYMBOL_TABLE module
    - provides operations to
      - CREATE an entry for a new variable
      - GET the value associated with a variable
      - PUT a new value for a given variable
  - the module hides the internal data structure of the symbol table
  - the data structure may freely change without affecting clients

Interface design

- Interface should not reveal what we expect may change later
- It should not reveal unnecessary details
- Interface acts as a firewall preventing access to hidden parts

Prototyping

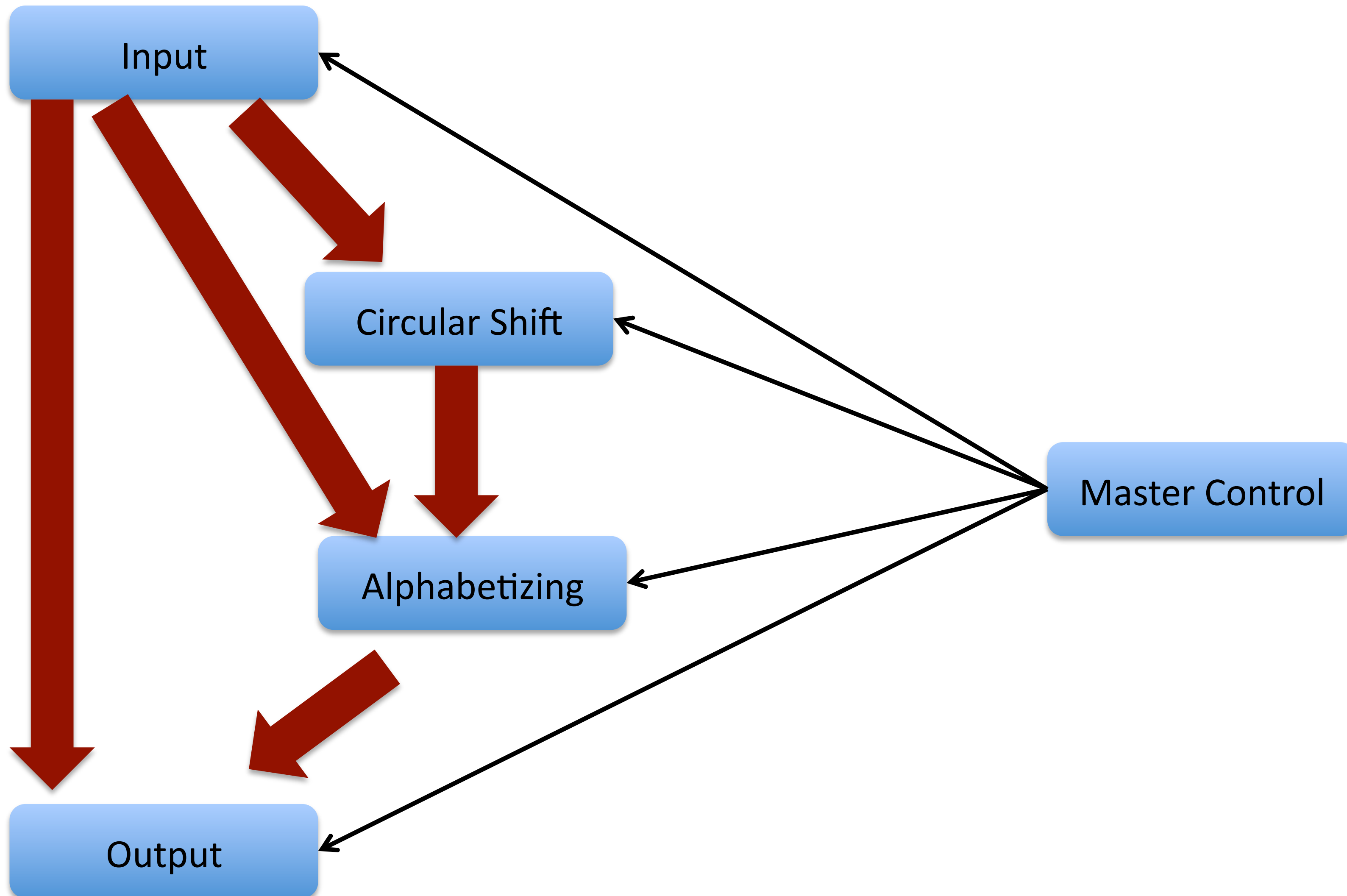
- Once an interface is defined, implementation can be done
  - first quickly but inefficiently
  - then progressively turned into the final version
- Initial version acts as a prototype that evolves into the final product

Parnas – Decomposing Systems into Modules

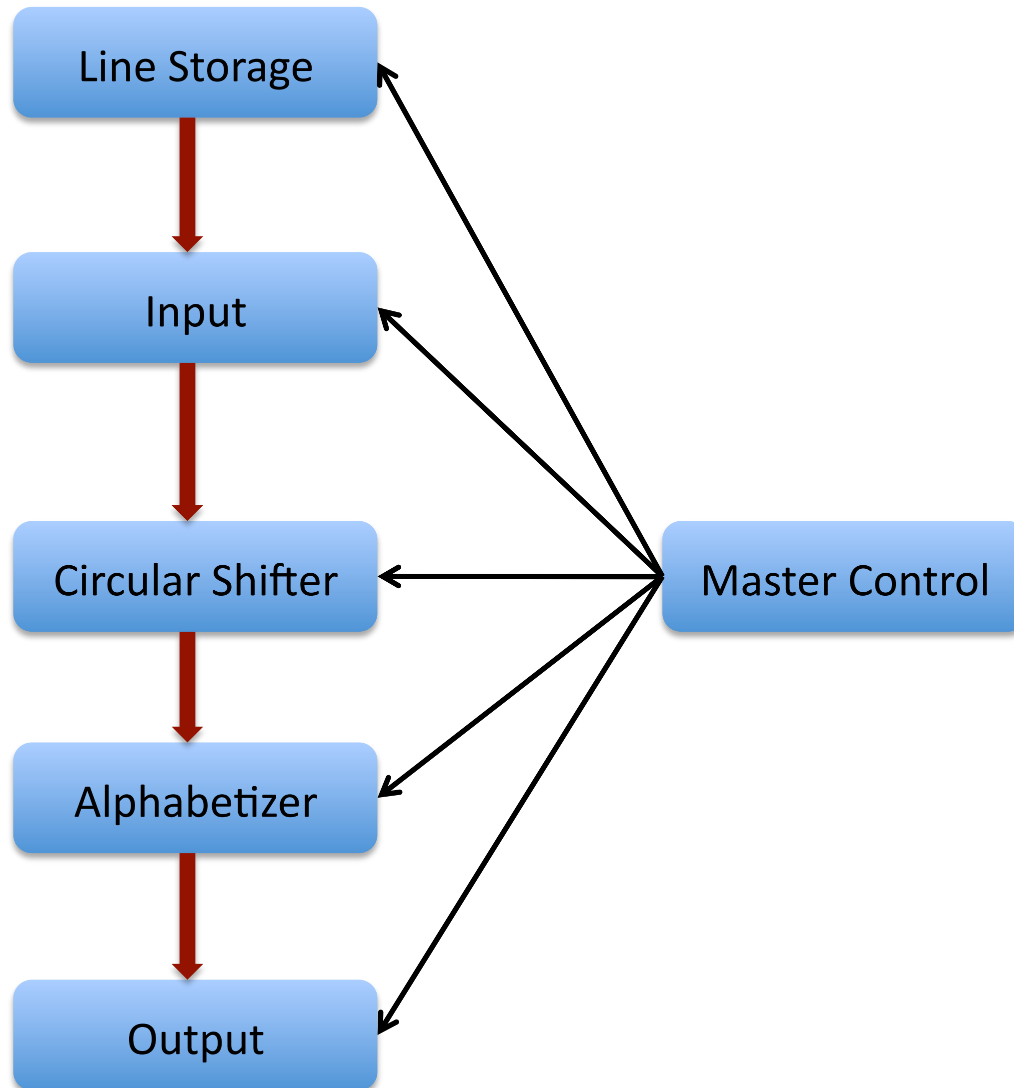
- Allow one module to be written with little knowledge of the code in another module

- Allow modules to be reassembled and replaced without reassembly of the whole system

Decomposition 1



Decomposition 2



Decomposition criteria

1. Abstract data types (data + procedures in the same module)
2. If a calling sequence is necessary, the sequence is invisible to outside modules
3. Format of control blocks hidden
4. Hide data representations
5. Sequence in which items are processed should be confined to one module

Reading for next class

Chapter 4 (sections 4.2.3—4.2.6)