

# CMSC 132: Object-Oriented Programming II

---



## **Problem Specification & Software Architecture**

**Department of Computer Science  
University of Maryland, College Park**

# Overview

- **Problem specification**
  - **Obstacles**
- **Software Architecture**
  - **How to divide work**
  - **Interface & conditions**

# Problem Specification

## ■ Goal

- Create complete, accurate, and unambiguous statement of problem to be solved

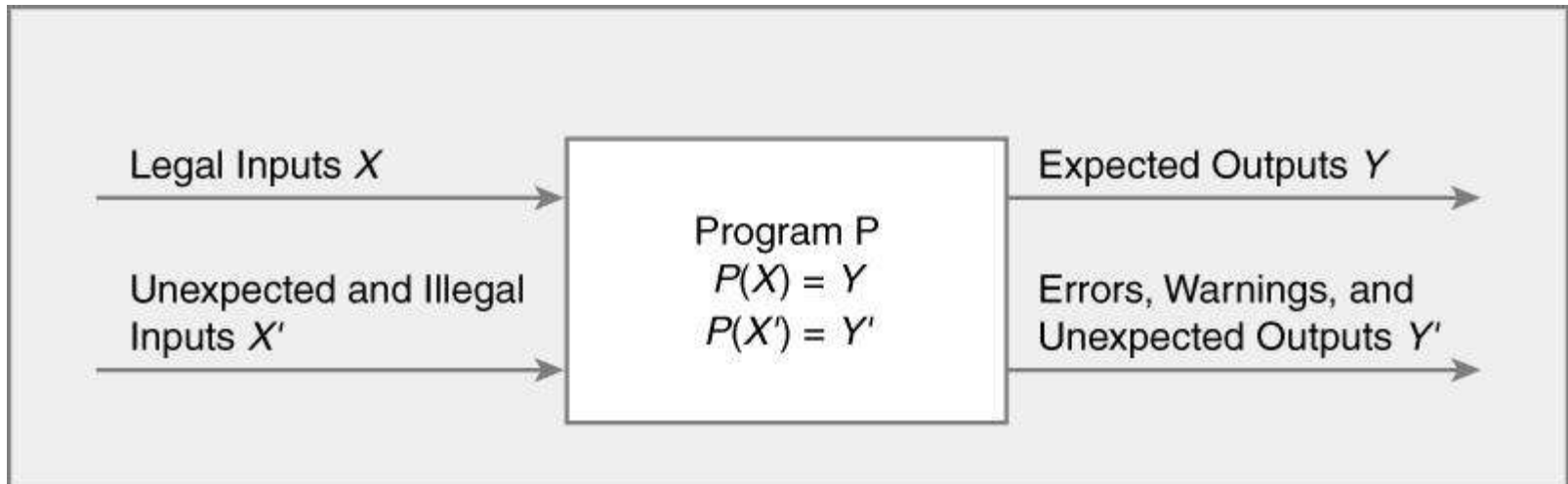
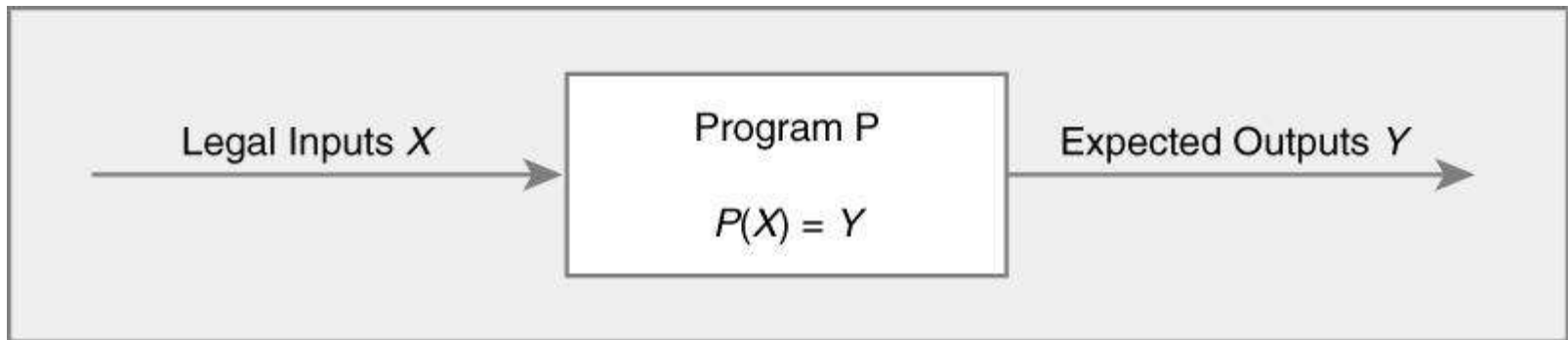
## ■ Problems

- Often useful at the method or class level
- Perfection rarely applicable or available at the system level
  - E.g., complete, accurate and unambiguous

# Problem Specification

## ■ Example

### ■ Specification of input & output for program



# Problem Specification Problems

- **Description may not be accurate**
  - Problem not understood by customer
- **Description may change over time**
  - Customer changes their mind
- **Difficult to specify behavior for all inputs**
  - Usually only covers common cases
  - Hard to consider all inputs (may be impossible)
  - Most UNIX utilities used to crash with random inputs
    - An Empirical Study of the Reliability of UNIX Utilities, B.P. Miller, L. Fredriksen, and B. So, 1991

# Problem Specification Problems

## ■ Description may be ambiguous

### ■ Natural language description is imprecise

#### ■ Why lawyers use legalese for contracts

### ■ Formal specification languages are limited and may be difficult to understand

### ■ Examples

#### ■ Find sum of all values between 1 and 100 that occur in the set S

#### ■ $\text{Sum } \{ x \mid x \in S \wedge 1 \leq x \leq 100 \}$

### ■ Difficult to write specifications that are both readable and precise

# Searching for an element

■ `int search(int x, int [] a)`

■ **Precondition:**

■ `a` is an array of `n` integers, index `0..n-1`

■ `x` is an integer

■ **Postcondition**

■ if `result == -1`, there is no value `i` such that `a[i] == x`

■ else, `a[result] == x`

■ **Critique this specification**

# Implementation 1

```
■ int search(int x, int [] a) {  
    a[0] = x;  
    return 0;  
}
```

## ■ Precondition:

- a is an array of n integers, index 0..n-1
- x is an integer

## ■ Postcondition

- if result == -1, there is no value i such that  $a[i] == x$
- else,  $a[\text{result}] == x$

# Multiple occurrences

- What if there are multiple occurrences of the value  $x$  in  $a$ ?
  - Are we allowed to return the index of any of them?
  - Or should we always return the first index?

# Program Design

## ■ Goal

- Break software into integrated set of **components** that work together to solve problem specification

## ■ Problems

- Methods for decomposing problem
  - How to divide work
  - What work to divide
- How components work together

# Software Architecture

- Big picture of the software
- Components generally bigger than objects or classes

# Key Words in Context

- 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.
- On the Criteria To Be Used in Decomposing Systems into Modules, David Parnes, 1972

# Software Productivity Improvements

- This is a small system. Except under extreme circumstances (huge data base, no supporting software), such a system could be produced by a good programmer within a week or two.
  - On the Criteria To Be Used in Decomposing Systems into Modules, David Parnes, 1972
- 15 minutes
  - Bill Pugh

# Modularization

## ■ Module 1: Input

- reads input and stores the lines

## ■ Module 2: Circular shift

- Prepares index with one entry per shifted line

## ■ Module 3: Alphabetizing

- Produces a sorted index

## ■ Module 4: Output

- Produces a formatted output

## ■ Module 5: Master Control

# Kwic architecture

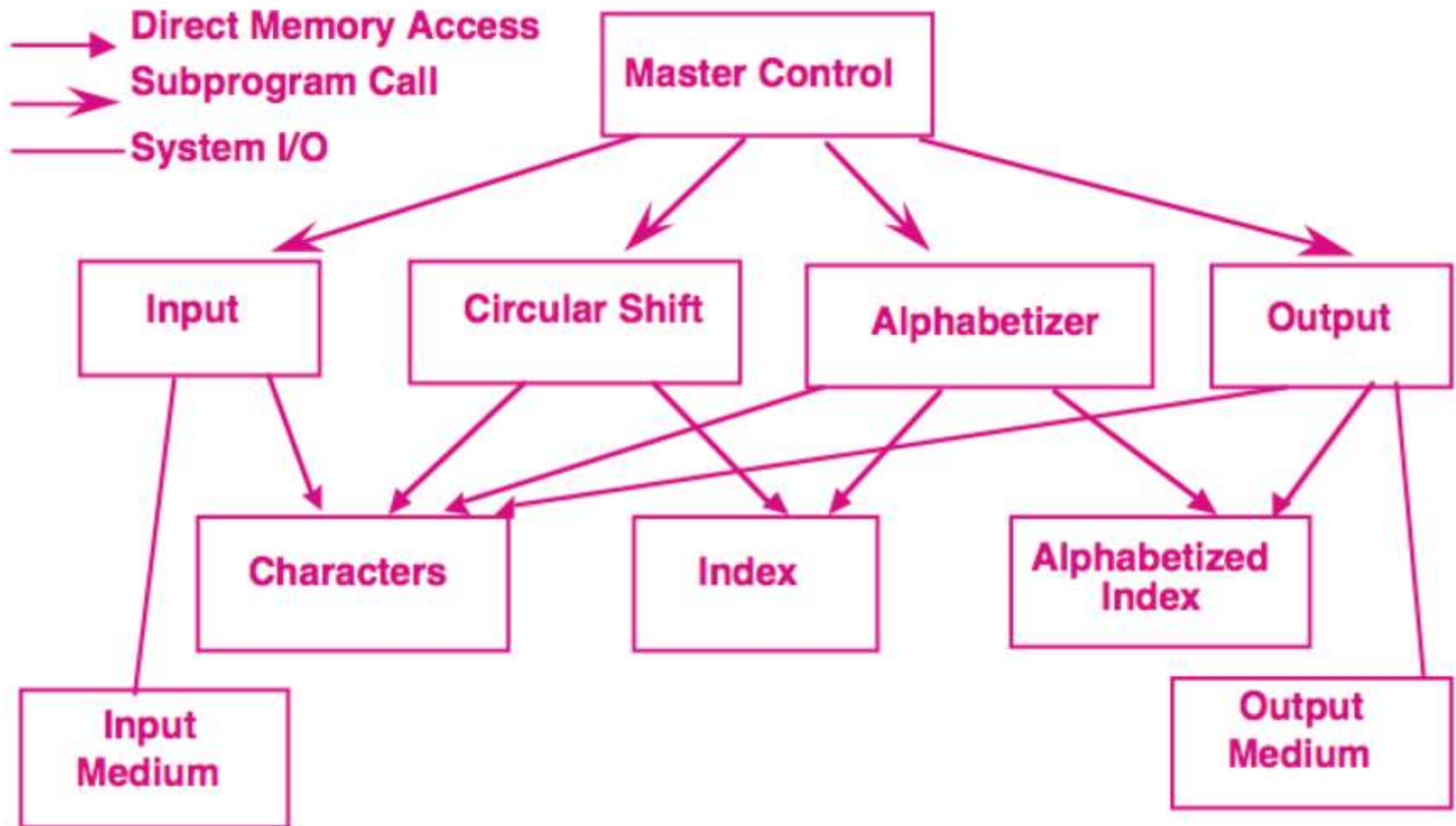


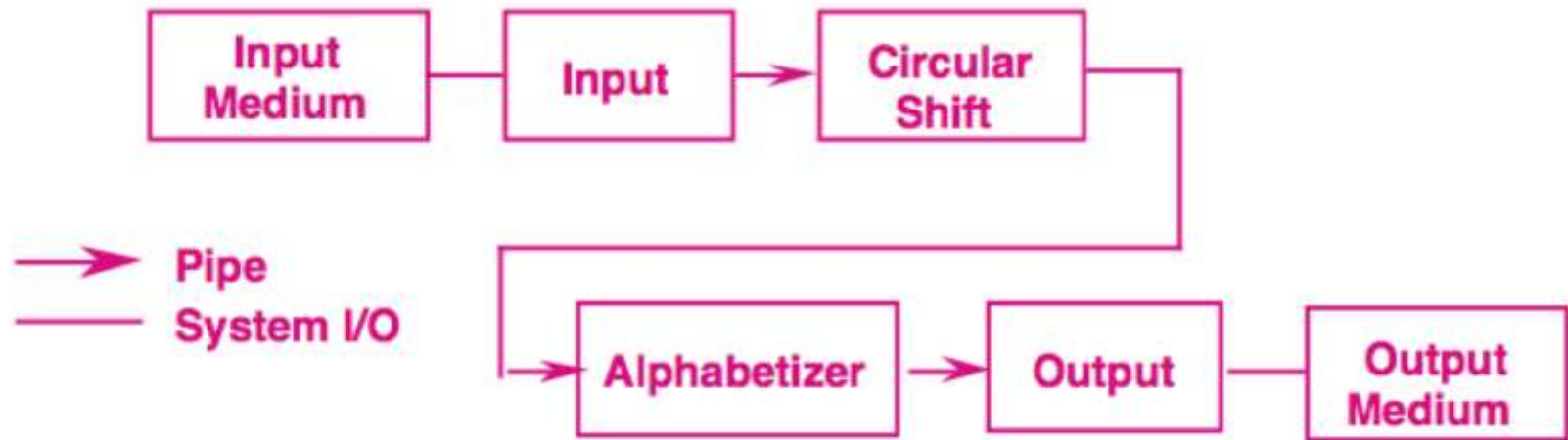
Figure 6: KWIC – Shared Data Solution

# Commentary

- Using this solution data can be represented efficiently, since computations can share the same storage. The solution also has a certain intuitive appeal, since distinct computational aspects are isolated in different modules. However, as Parnas argues, it has a number of serious drawbacks in terms of its ability to handle changes. In particular, a change in data storage format will affect almost all of the modules. Similarly changes in the overall processing algorithm and enhancements to system function are not easily accommodated. Finally, this decomposition is not particularly supportive of reuse.

■ **An Introduction to Software Architecture, David Garlan and Mary Shaw**

# Kwic architecture, pipes and filters



*Figure 9: KWIC – Pipe and Filter Solution*

# Commentary

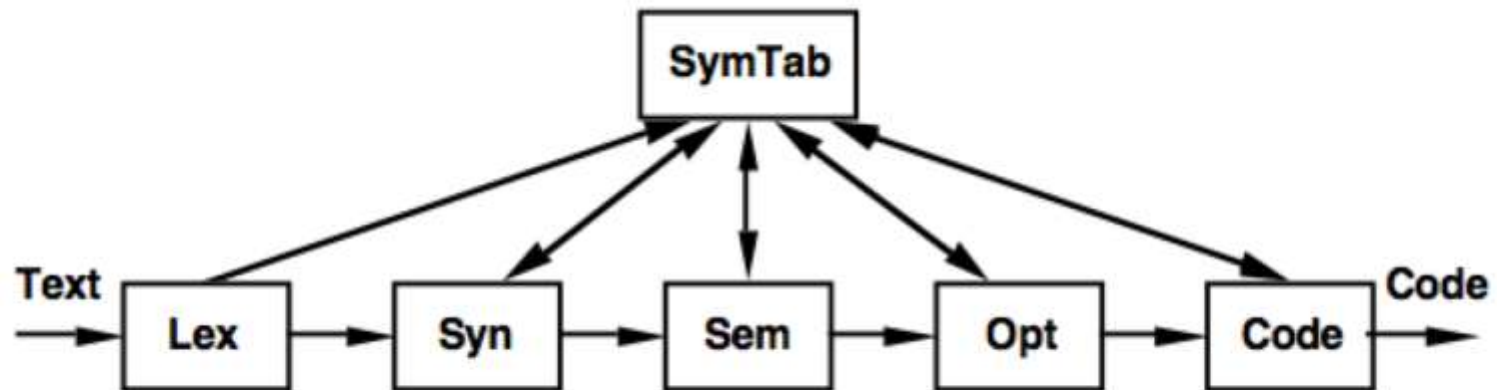
- This solution has several nice properties. First, it maintains the intuitive flow of processing. Second, it supports reuse, since each filter can function in isolation (provided upstream filters produce data in the form it expects). New functions are easily added to the system by inserting filters at the appropriate point in the processing sequence. Third, it supports ease of modification, since filters are logically independent of other filters.
- On the other hand it has a number of drawbacks. First, it is virtually impossible to modify the design to support an interactive system. For example, in order to delete a line, there would have to be some persistent shared storage, violating a basic tenet of this approach. Second, the solution is inefficient in terms of its use of space, since each filter must copy all of the data to its output ports.
  - **An Introduction to Software Architecture, David Garlan and Mary Shaw**

# Different architecture styles

- The same system can be described using several different architecture styles
  - Pipes and filters
    - What is the data, and what components do they move through
  - Blackboard
    - Components communicate through a shared, updatable blackboard

# Compiler Architecture

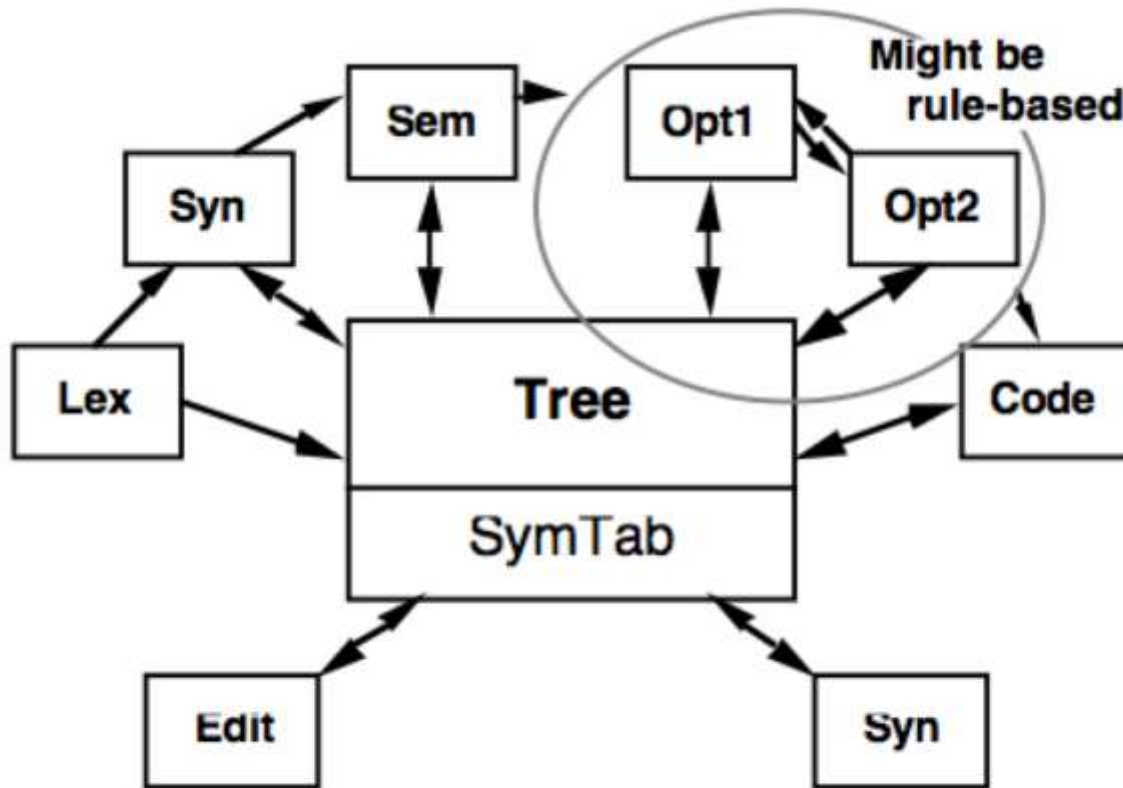
## ■ Pipes and Filters



*Figure 16: Traditional Compiler Model with Shared Symbol Table*

# Compiler Architecture, revisited

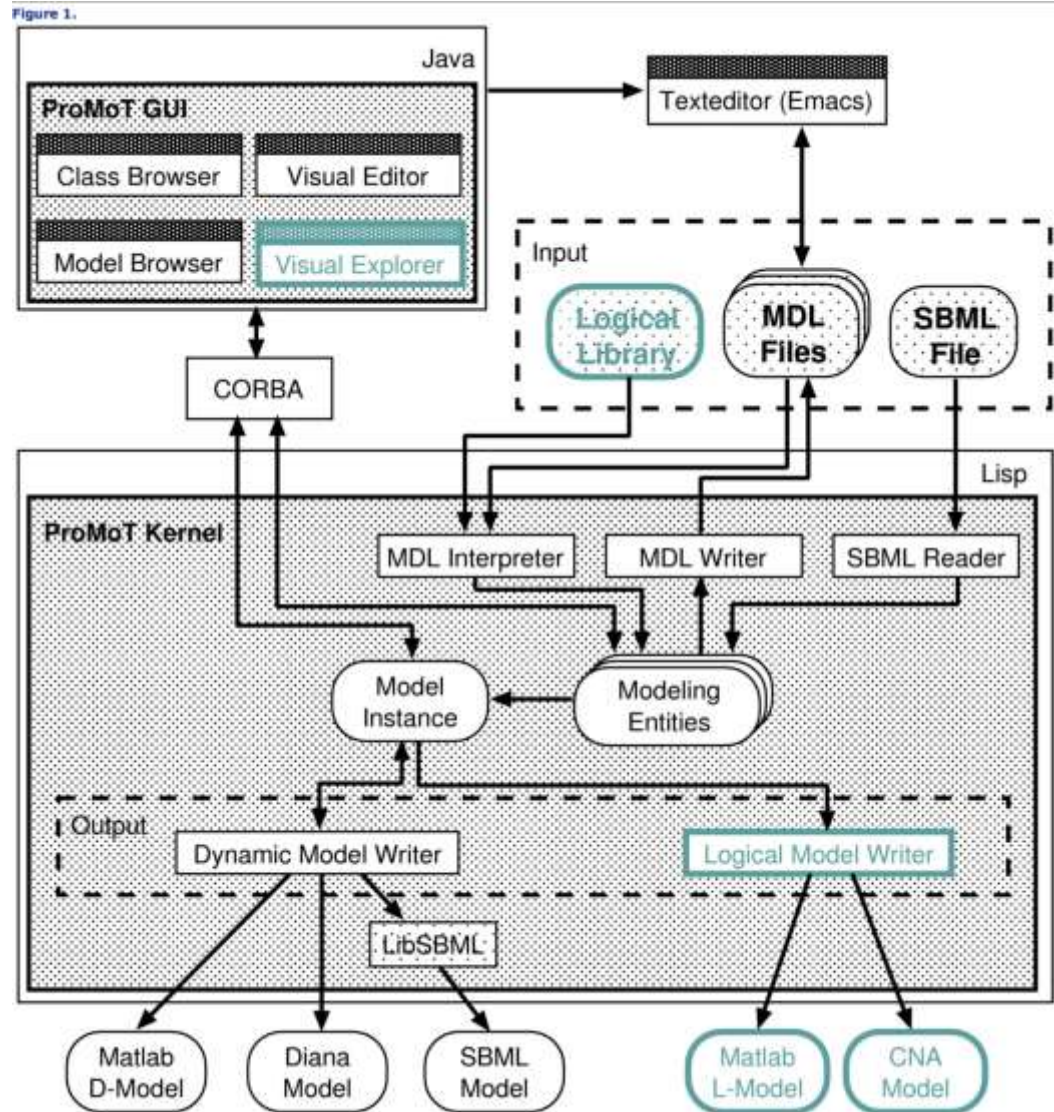
## ■ Blackboard



*Figure 18: Canonical Compiler, Revisited*

# Architecture of ProMoT

Just an arbitrary  
example of a real-  
world software  
architecture



# UML – Unified Modeling Language

- A standard format for pictorial description of software systems
- The standard describes 13 different kinds of diagrams
  - Class/object diagrams
  - Sequence diagrams
- Diagrams can be much richer than boxes and arrows
  - Various picture details used to describe properties

# CS History Moment: Donald Ervin Knuth

- Renowned computer scientist and Professor Emeritus at Stanford
- Author of the seminal multi-volume work “*The Art of Computer Programming*”
  - A comprehensive series of books (several volumes) that cover many kinds of programming algorithms and their analysis.
- Creator of the TeX computer typesetting system



# References

- On the criteria to be used in decomposing systems into modules, David Parnes, 1972
  - [http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro\\_softarch/intro\\_softarch.pdf](http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf)
- An Introduction to Software Architecture, David Garlan and Mary Shaw, January 1994
  - [http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro\\_softarch/intro\\_softarch.pdf](http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf)