Two orthogonal hierarchies of the system

- ✓ "part-of" hierarchy
- ✓ A red rose is a rose
- ✓ A yellow rose is a rose
- ✓ "is a" hierarchy
- ✓ We call these hierarchies the *class structure and the object structure of* the system, respectively

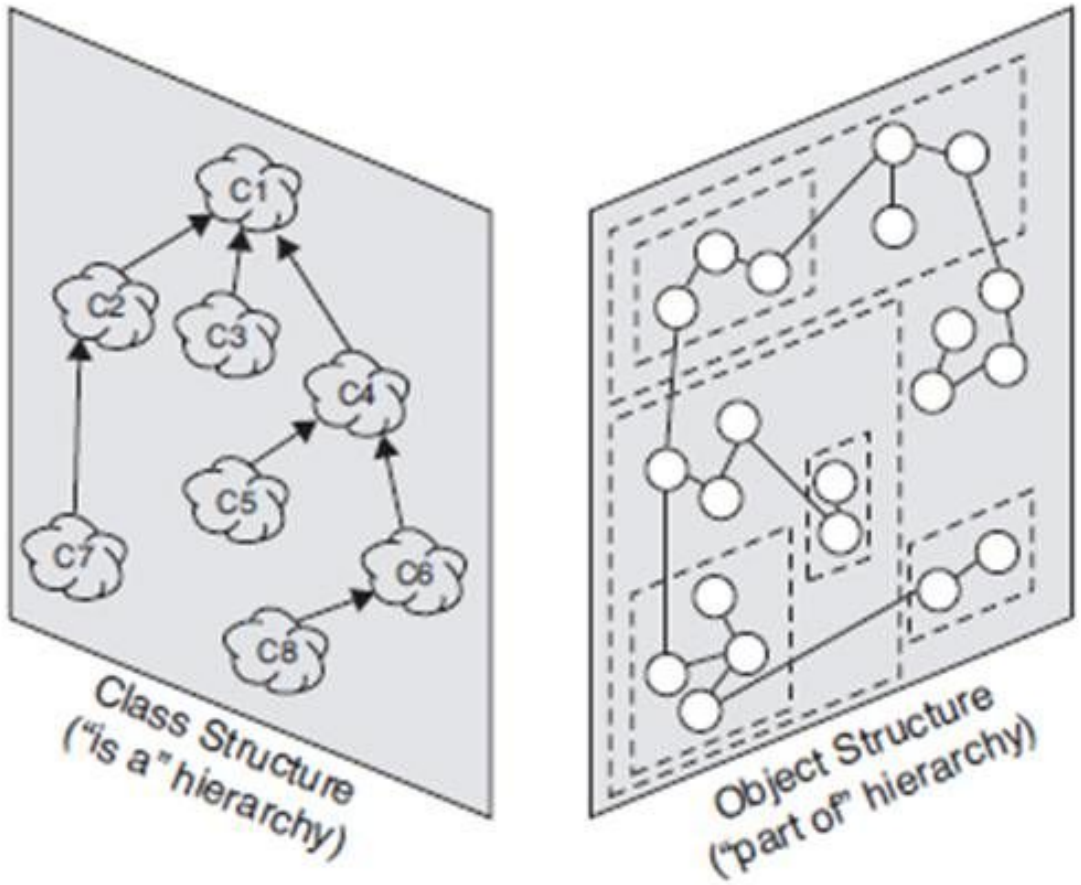


Figure 1–1 The Key Hierarchies of Complex Systems

- There are usually many more objects than classes
- From the same class structure, there are many different ways that these objects can be instantiated and organized

How to handle complexity

- **Divide and rule**

  Decompose the complex system into smaller and smaller parts, each of which may then be designed independently

- ***Algorithmic Decomposition***

- ***Object-Oriented Decomposition***

- Which is the right way to decompose a complex system

- **Algorithmic Decomposition**: view software as a process, break down the software into modules that represent steps of the process. Data structures required to implement the program are a secondary concern.
- **Object-Oriented Decomposition**: view software as a set of well-defined objects that model entities in the application domain.

'Develop a software for a bank that can be used to open accounts of customers. Customers can deposit and withdraw money from the bank. A customer can also close the account."

Advantages of OO Decomposition

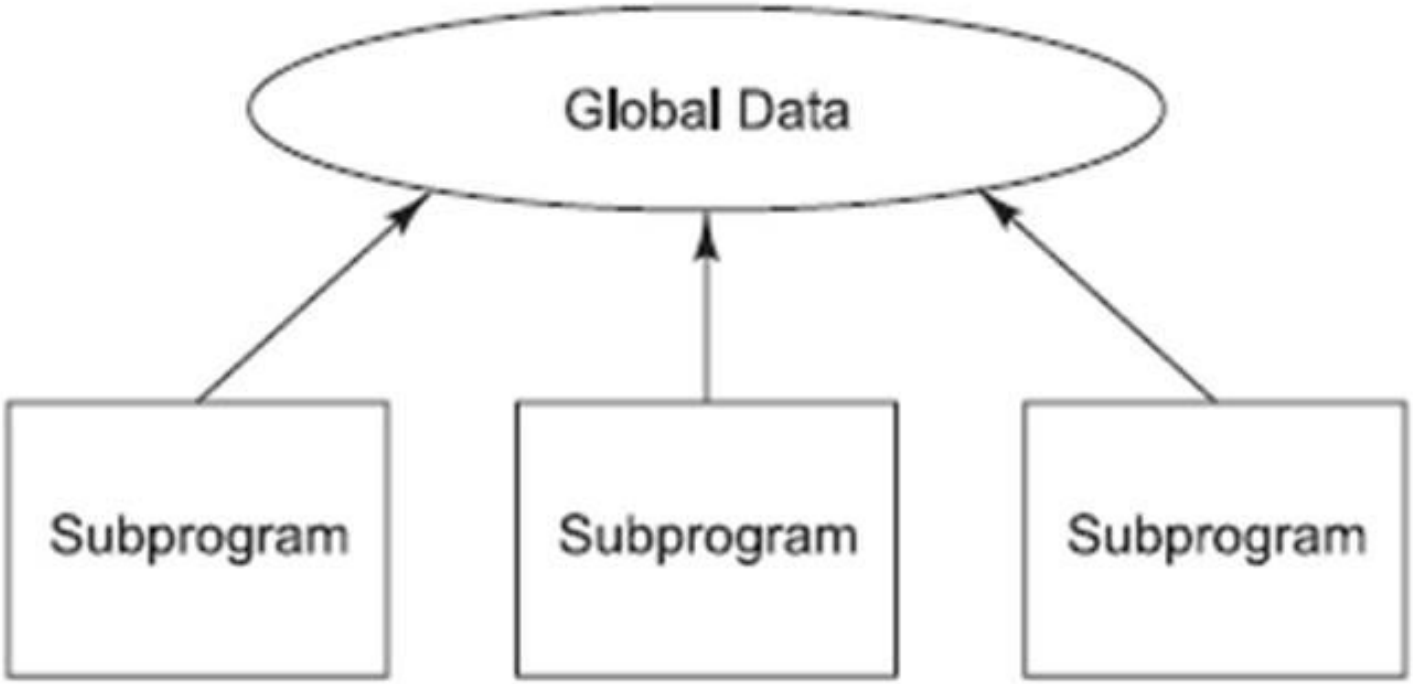
- encourages reuse of software
- Allow software evolve as system requirements change
- More intuitive

- **Programming** was initially done by toggling in the binary machine instructions
- Worked for few hundred instructions long programs
- Assembly language was invented to deal with larger increasingly complex programs
- As programs continued to grow, high-level languages were introduced that gave the programmer more tools with which to handle complexity
- The first widespread language was FORTRAN.

STYLES OF PROGRAMMING

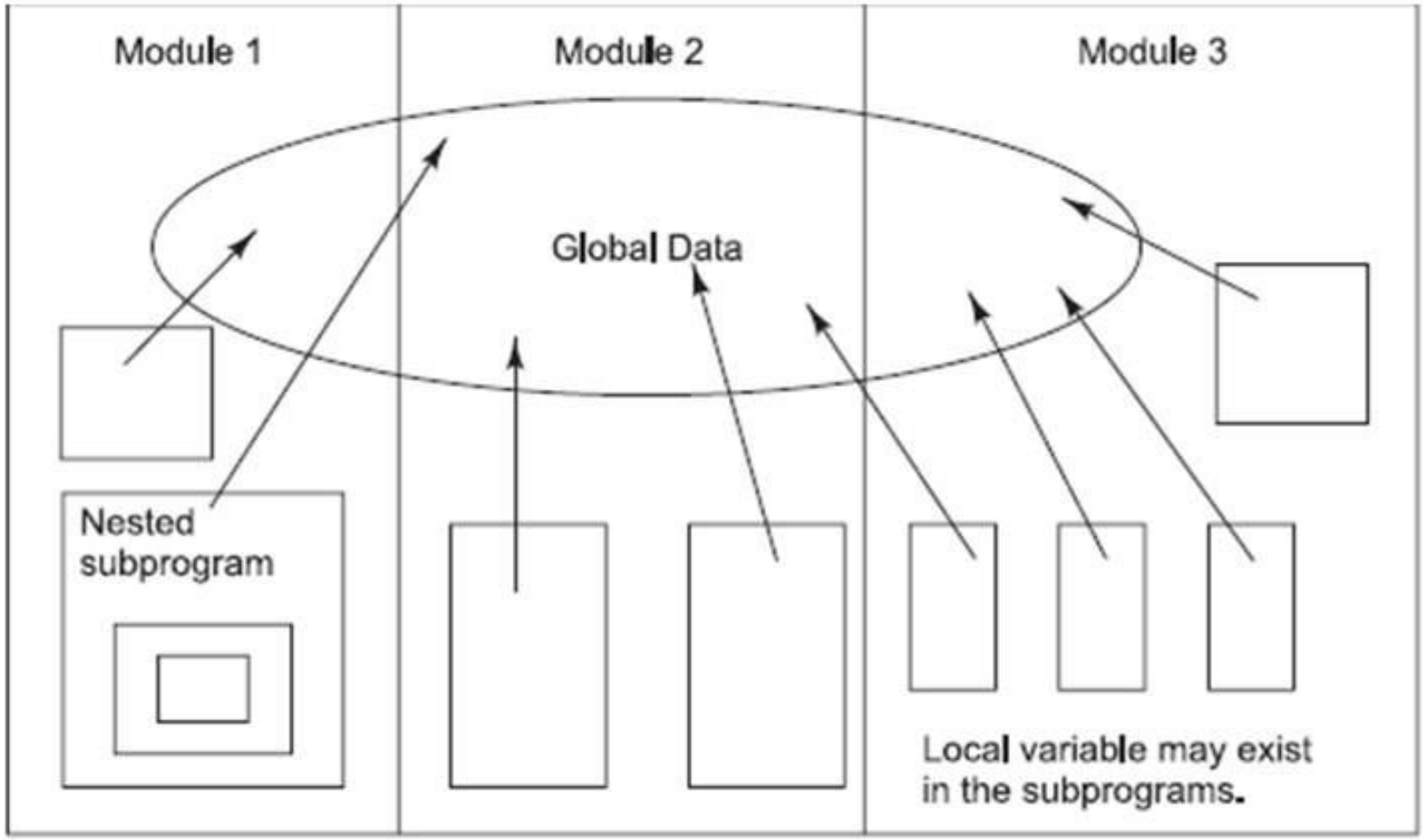
First-Generation Languages (1954-1958) - Wegner

FORTRAN I	Mathematical expressions
ALGOL 58	Mathematical expressions
Flowmatic	Mathematical expressions
IPL V	Mathematical expressions



Second-generation languages (1959–1961)

FORTRAN II	Subroutine, separate compilation
ALGOL 60	Block structure, data types
COBOL	Data description, file handling
Lisp	List processing, pointers, garbage collection



Third-generation languages (1962–1970)

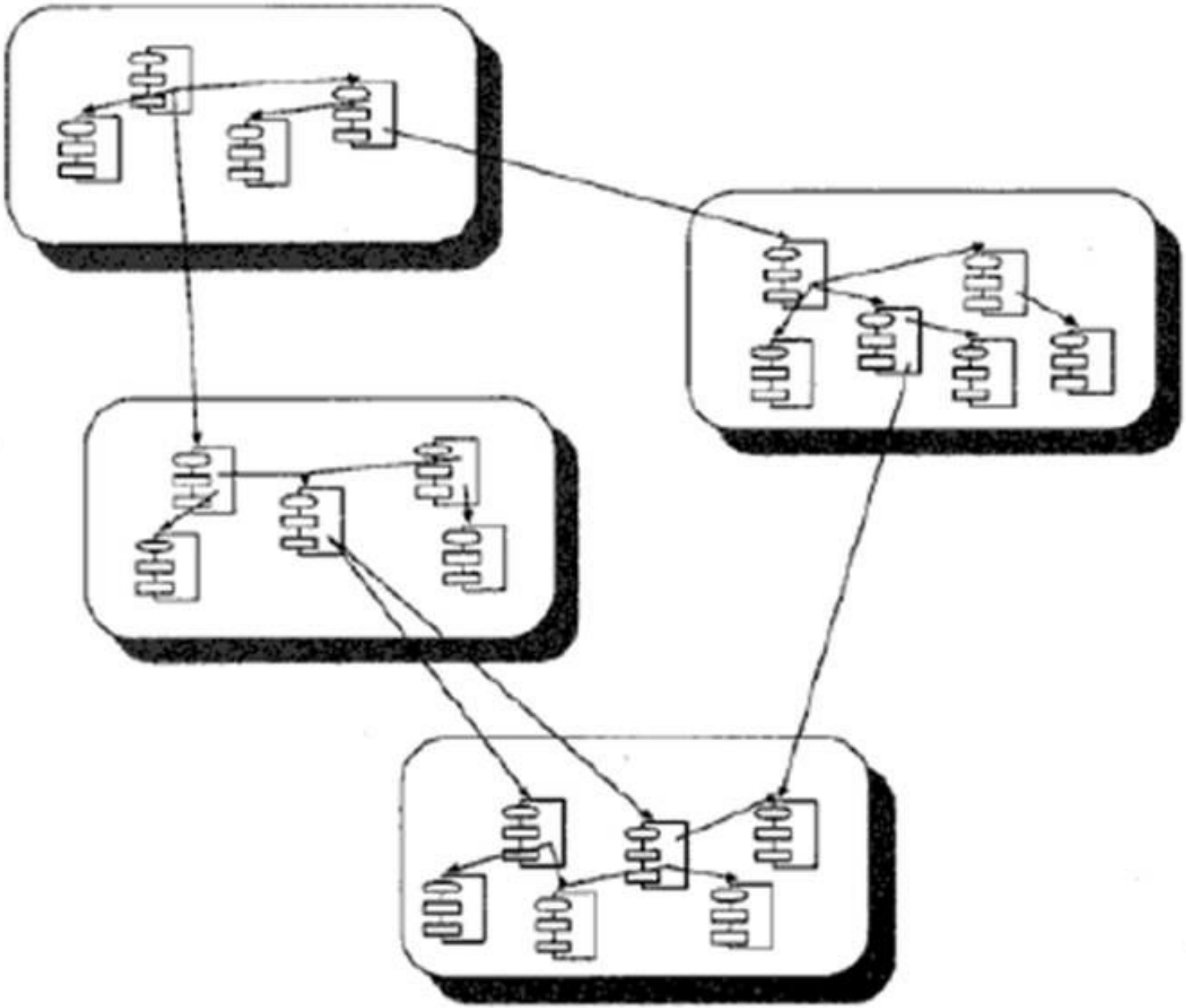
PL/1	FORTRAN + ALGOL + COBOL
ALGOL 68	Rigorous successor to ALGOL 60
Pascal	Simple successor to ALGOL 60
Simula	Classes, data abstraction

- Object-orientation boom (1980–1990)

Smalltalk 80	Pure OO language
C++	Derived from C and Simula
Ada83	heavy Pascal influence
Eiffel	Derived from Ada and Simula

- Emergence of frameworks (1990–today)

Visual Basic	Eased development of the GUI
Java	Successor to Oak
Python	OO scripting language
J2EE	Java-based framework for enterprise computing
.NET	Microsoft's O-based framework



OO Programming Languages

Program two similar but separate forms for a website, to processes information about cars and trucks.

The information to be recorded for cars:

- Color
- Engine Size
- Transmission Type
- Number of doors

The information to be recorded for trucks:

- Color
- Engine Size
- Transmission Type
- Cab Size
- Towing Capacity

Two Programming Paradigms

- All computer programs consist of two elements
  - code and data
- A program can be organized
  - around its code or (what is happening)
  - around its data (who is being affected)
- These are the two paradigms that govern how a program is constructed
  - *process-oriented model*
  - *object-oriented programming*

File1.c:

```
int x;
p1(){}
```

File2.c:

```
int x;
p2(){}
```

```
gcc file1.c file2.c
```