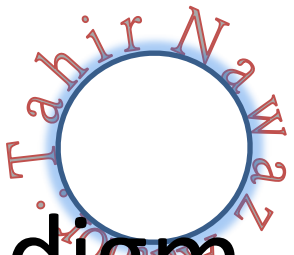
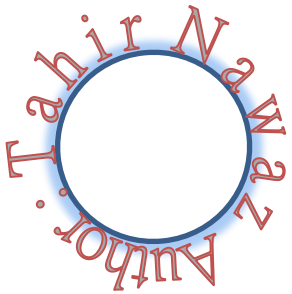
CLASS WORK



Object Oriented Paradigm



OOP’s

Problems with Structured Programming

As programs grow ever larger and more complex, even the structured programming approach begins to show signs of strain.

The project is too complex,

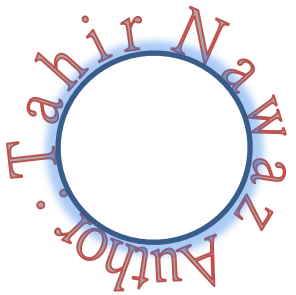
the schedule slips,

more programmers are required,

complexity increases,

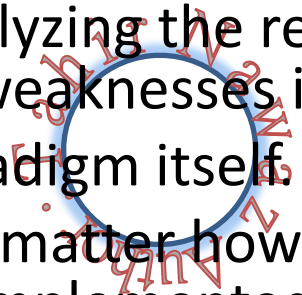
costs increases like rocket,

the schedule slips further, and disaster ensues.



Analyzing the reasons for these failures reveals that there are weaknesses in the procedural paradigm itself.

No matter how well the structured programming approach is implemented, large programs become excessively complex.



CLASS WORK

What are the reasons for these problems with procedural languages?

There are two related problems.

First, functions have unrestricted access to global data.

Second, unrelated functions and data, the basis of the procedural paradigm, provide a poor model of the real world.

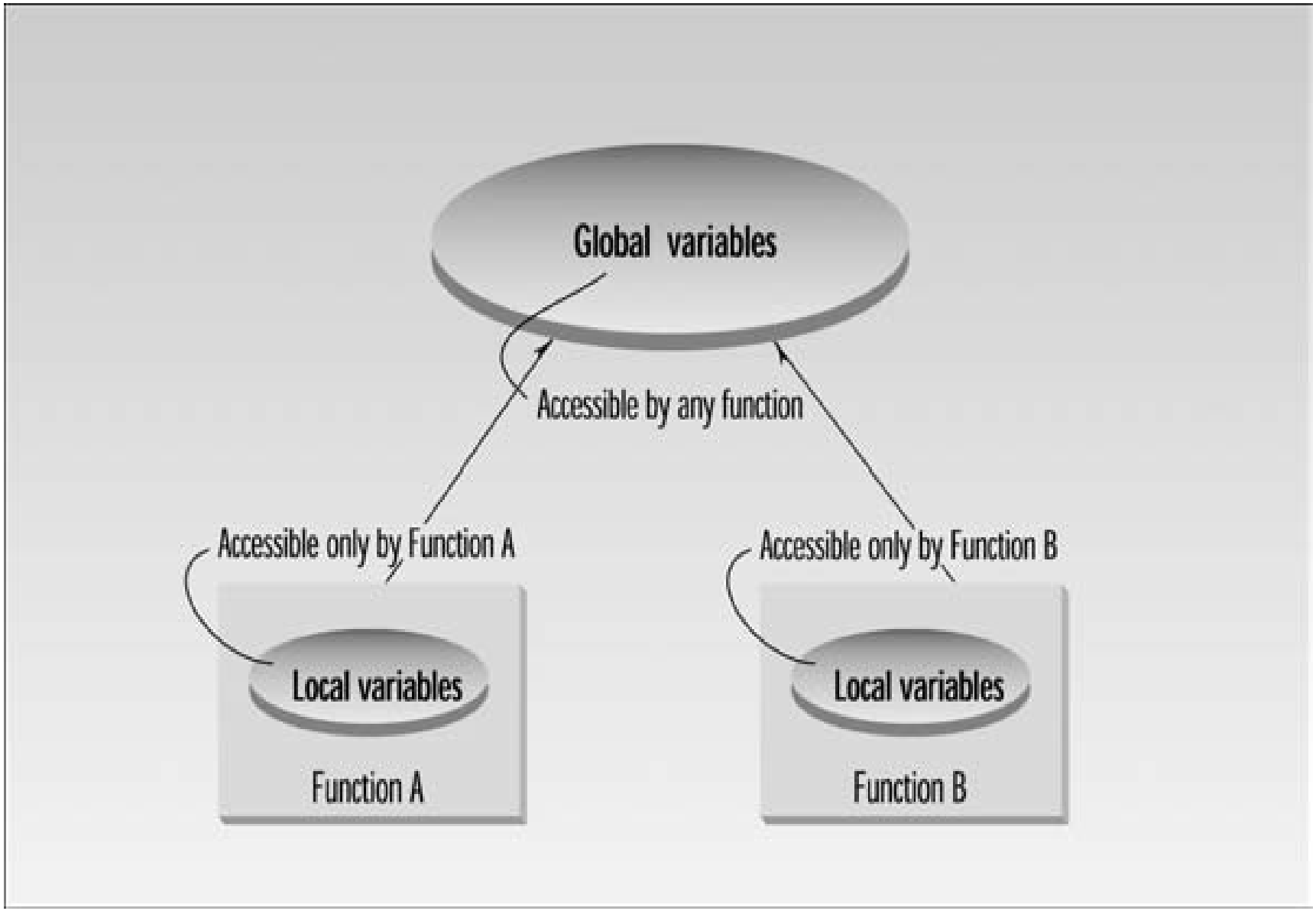
e.g. these problems in the context of an inventory program.

One important global data item in such a program is the collection of items in the inventory. Various functions access this data to input a new item, display an item, modify an item, and so on.

Unrestricted Access

- In a procedural program, for e.g. one written in C Language
  - There are two kinds of data.
- *Local data is hidden inside a function, and is used exclusively by the function.*
  - *In the* inventory program a display function might use local data to remember which item it was displaying. Local data is closely related to its function and is safe from modification by other functions.
- However, when two or more functions must access the same data—and this is true of the most important data in a program—then the data must be made *global, as our collection of inventory* items is.
- Global data can be accessed by *any function in the program*. The arrangement of local and global variables in a procedural program is shown.

CLASS WORK

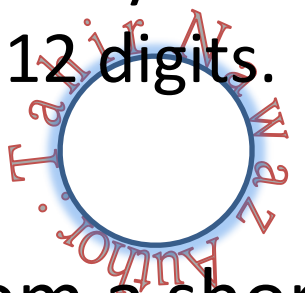

Global variables
Accessible by any function
Accessible only by Function A
Accessible only by Function B
Local variables
Function A
Local variables
Function B

- In a large program, there are many functions and many global data items.
- The problem with the procedural paradigm is that this leads to an even larger number of potential connections between functions and data, as shown

- This large number of connections causes problems in several ways.
- First, it makes a program's structure difficult to conceptualize. Second, it makes the program difficult to modify.

- A change made in a global data item may necessitate rewriting all the functions that access that item.

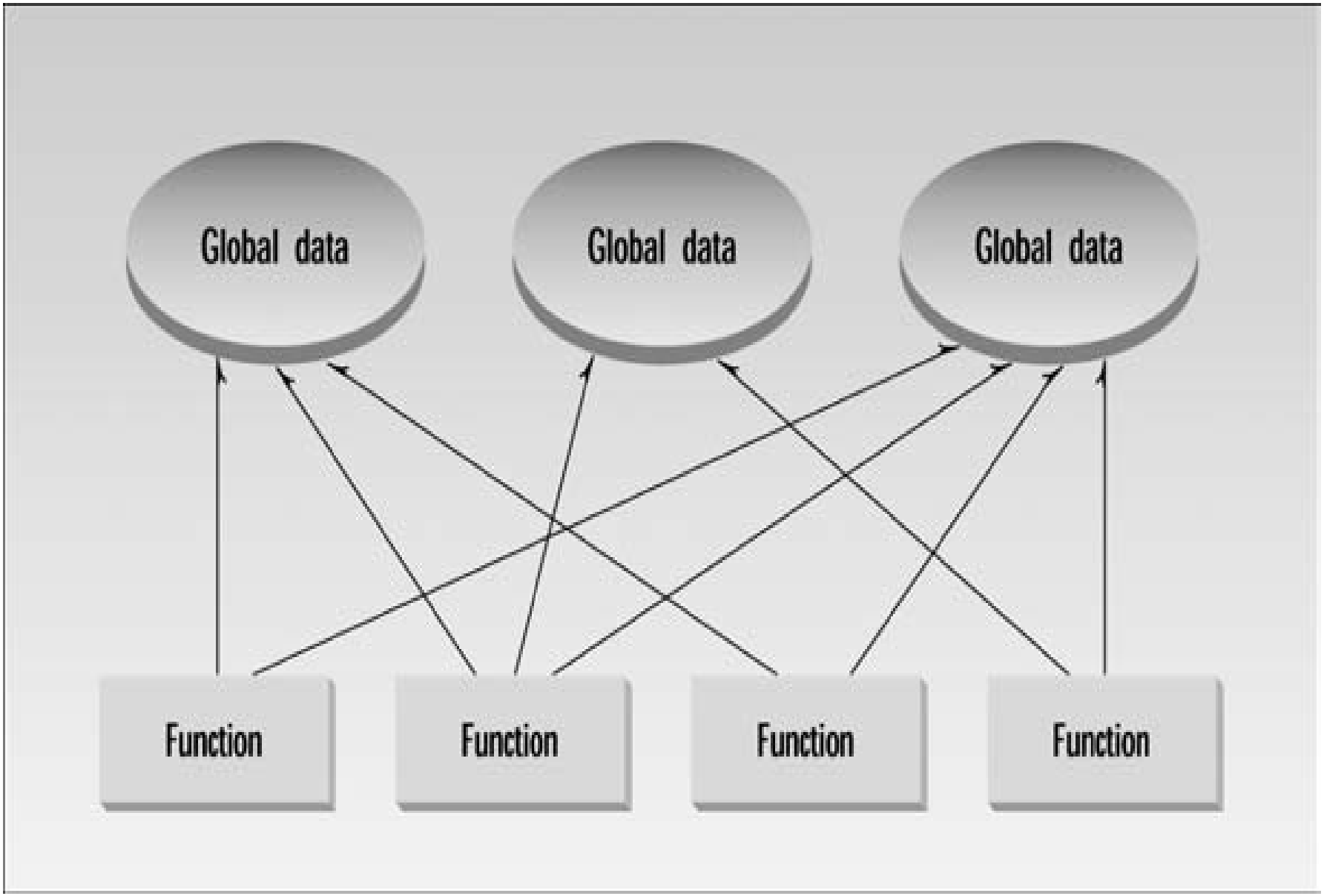
- e.g.
  - in our inventory program, someone may decide that the product codes for the inventory items should be changed from 5 digits to 12 digits.



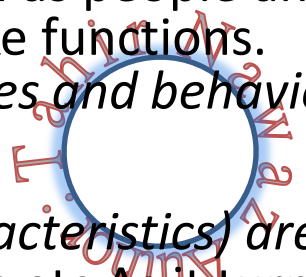
- This cause the need to change from a short to a long data type.
  - Now all the functions that operate on the data must be modified to deal with a long instead of a short.

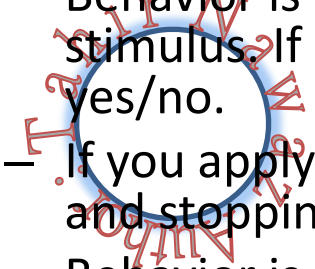
CLASS WORK

CLASS WORK


Global data
Global data
Global data
Function
Function
Function
Function

- **Real-World Modeling**
  - The second—and more important—problem with the procedural paradigm is that its arrangement of separate data and functions does a poor job of modeling things in the real world.
  - In the physical world we deal with objects such as people and cars. Such objects aren't like data and they aren't like functions. Complex real-world objects have both *attributes and behavior.*
- **Attributes**
  - Examples of attributes (sometimes called *characteristics) are, for people, eye color* and job title; and, for vehicles etc As it turns out, attributes in the real world are equivalent to data in a program: they have a certain specific values, such as blue or four.
- **Behavior**
  - Behavior is something a real-world object does in response to some stimulus. If anyone ask his boss for a raise, He/she will generally say yes/no.
  - If you apply the brakes in a car, it will generally stop. Saying something and stopping are examples of behavior.
  - Behavior is like a function: you call a function to do something and it does it.

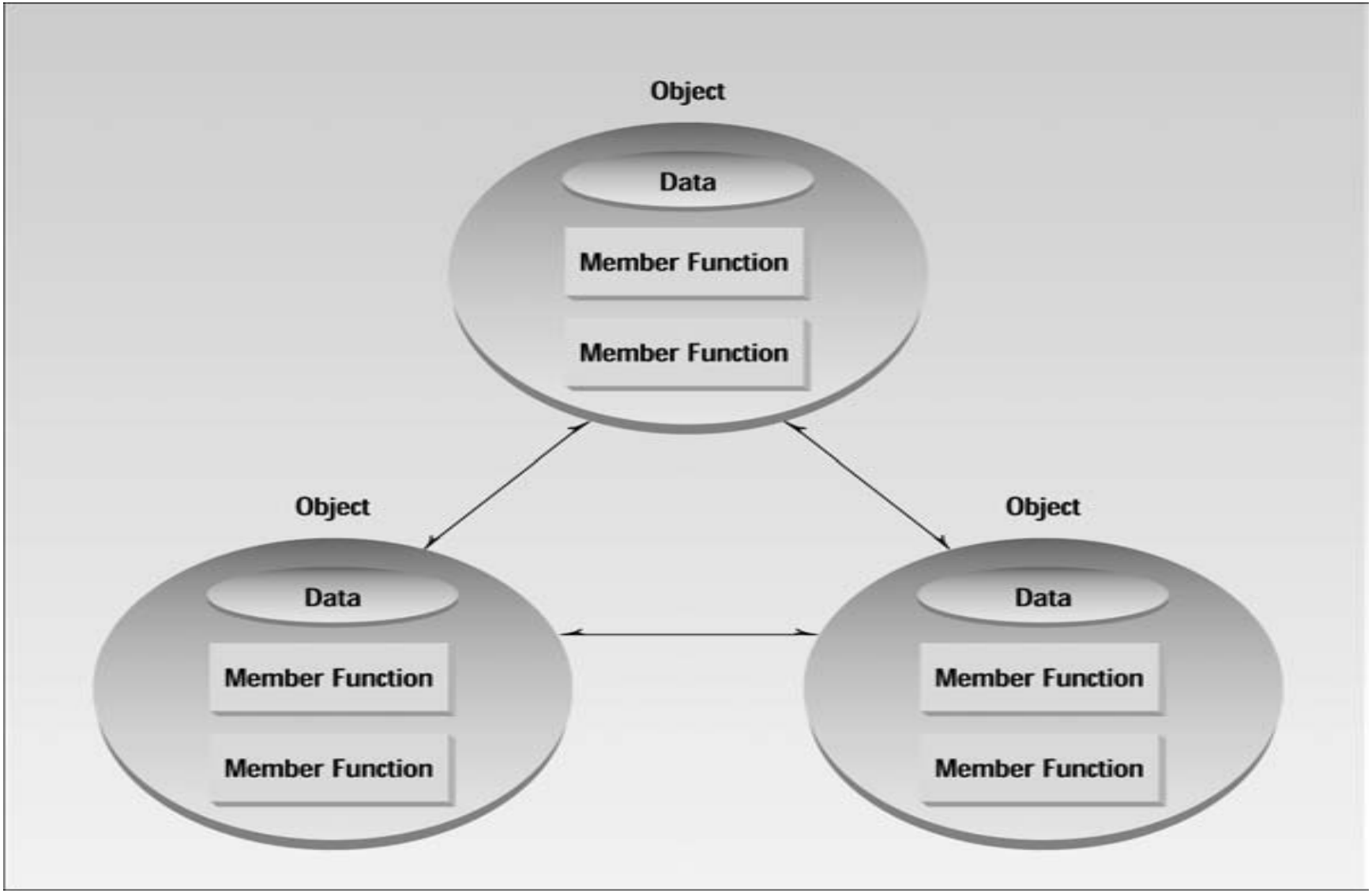


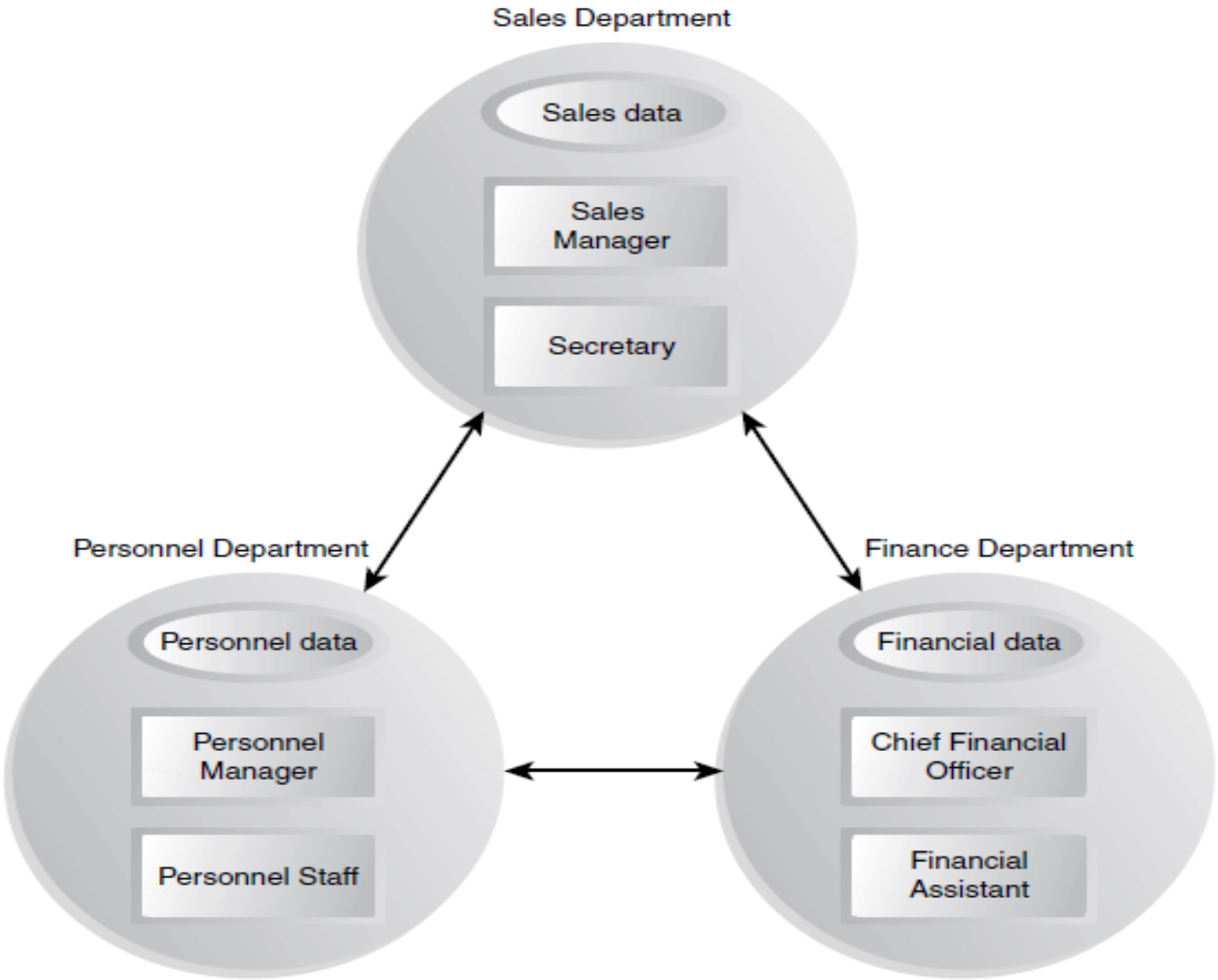


CLASS WORK

- **The Object-Oriented Approach**
  - The fundamental idea behind object-oriented languages is to combine into a single unit both *data and the functions that operate on that data. Such a unit is called an Object.*
- An object's functions, called *member functions in C++, typically provide the only way to* access its data. If you want to read a data item in an object, you call a member function in the object. It will access the data and return the value to you. You can't access the data directly.
- The data is *hidden, so it is safe from accidental alteration. Data and its functions are said to be encapsulated into a single entity. Data encapsulation and data hiding are key terms in the* description of object-oriented languages.

CLASS WORK


Object
Data
Member Function
Member Function
Object
Data
Member Function
Member Function
Object
Data
Member Function
Member Function



The corporate paradigm.

Reusability

- Once a class has been written, created, and debugged, it can be distributed to other programmers for use in their own programs.
- This is called *reusability. It is similar to* the way a library of functions in a procedural language can be incorporated into different programs.