SOFTWARE ENGINEERING
(Week-5)

USAMA MUSHARAF

MS-CS (Software Engineering)

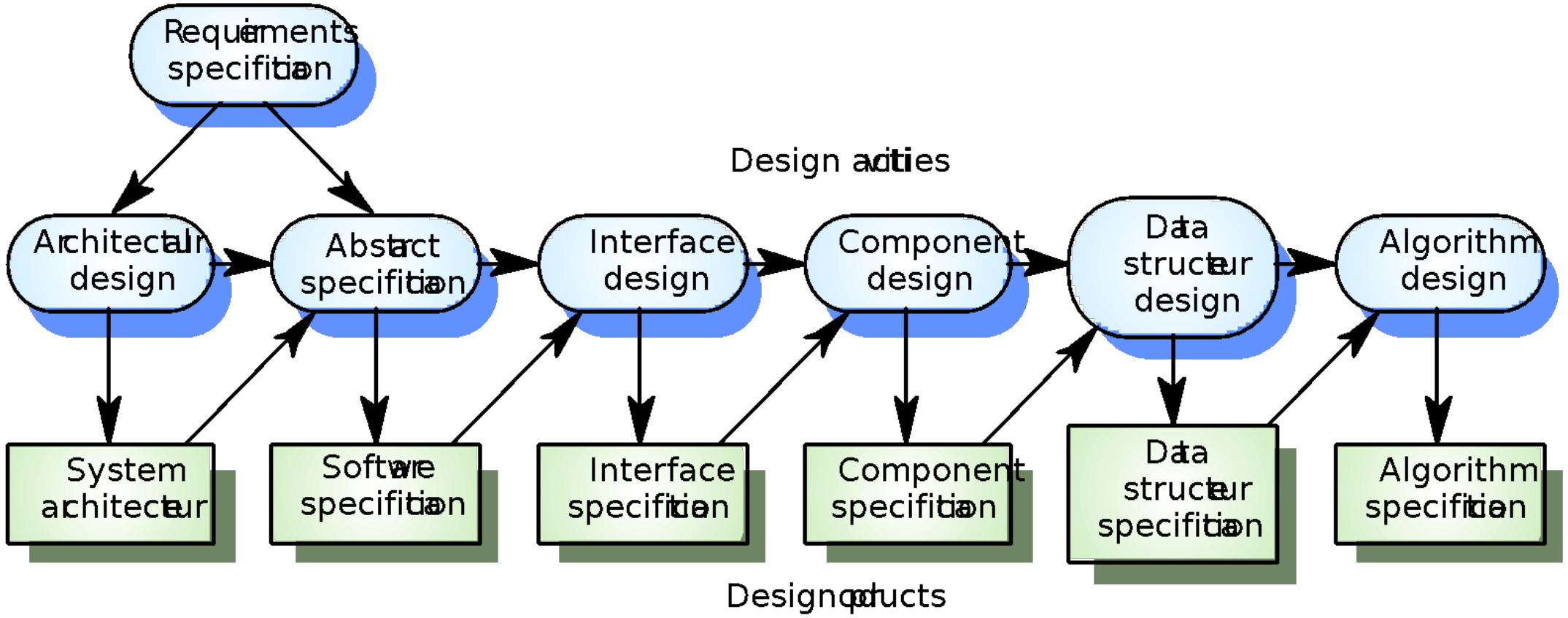
LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

AGENDA OF WEEK # 5

Software Design & Architecture
Design Principles
Design Concepts
Effective Modular Design
Types of Coupling & Cohesion

THE SOFTWARE DESIGN PROCESS



DESIGN PROCESS ACTIVITIES

- Architectural design
  - Modules, inter-relationships etc
- Abstract specification
  - Services of each sub-system, constraints etc
- Interface design
  - Interface to other sub-system or outside environment
- Component design
  - Services allocated to components and their interfaces designed
- Data structure design
- Algorithm design

LEVELS OF SOFTWARE DESIGN

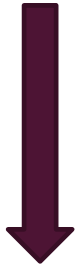
Architectural design (high-level design)

- architecture - the overall structure, main modules and their connections
- addresses the main non-functional requirements (e.g., reliability, performance)
- hard to change

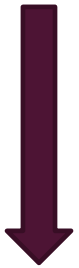
Detailed design (low-level design)

- the inner structure of the main modules
- detailed enough to be implemented in the programming language

Architecture



Design



Implementation

DESIGN VS. ARCHITECTURE

Architecture is concerned with the selection of architectural elements, their interaction, and the constraints on those elements and their interactions

Design is concerned with the modularization and detailed interfaces of the design elements, their algorithms and procedures, and the data types needed to support the architecture and to satisfy the requirements.

Architecture…is specifically not about…details of implementations (e.g., algorithms and data structures.)

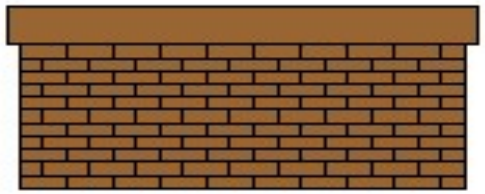
Software Development

- Lists
- Arrays
- Class
- Object
- Procedures
- Functions
- Algorithms
- Etc.

```
//********************************************************************
import inventoryItem;
import java.util.StringTokenizer;
import java.io.*;

public class inventory
{
  //-----------------------------------------------------------------
  // Reads data about a store inventory from an input file,
  // creating an array of inventoryItem objects, then prints them.
  //-----------------------------------------------------------------
  public static void main (String[] args)
  {
    final int MAX = 100;
    inventoryItem[] items = new inventoryItem[MAX];
    StringTokenizer tokenizer;
    String line, name, file="inventory.dat";
    int units, count = 0;
    float price;

    try
for (int scan = 0; scan < count; scan++)
        System.out.println (items[scan]);
    }
    catch (FileNotFoundException exception)
    {
      System.out.println ("The file " + file + " was not found.");
    }
    catch (IOException exception)
    {
      System.out.println (exception);
    }
  }
}
```

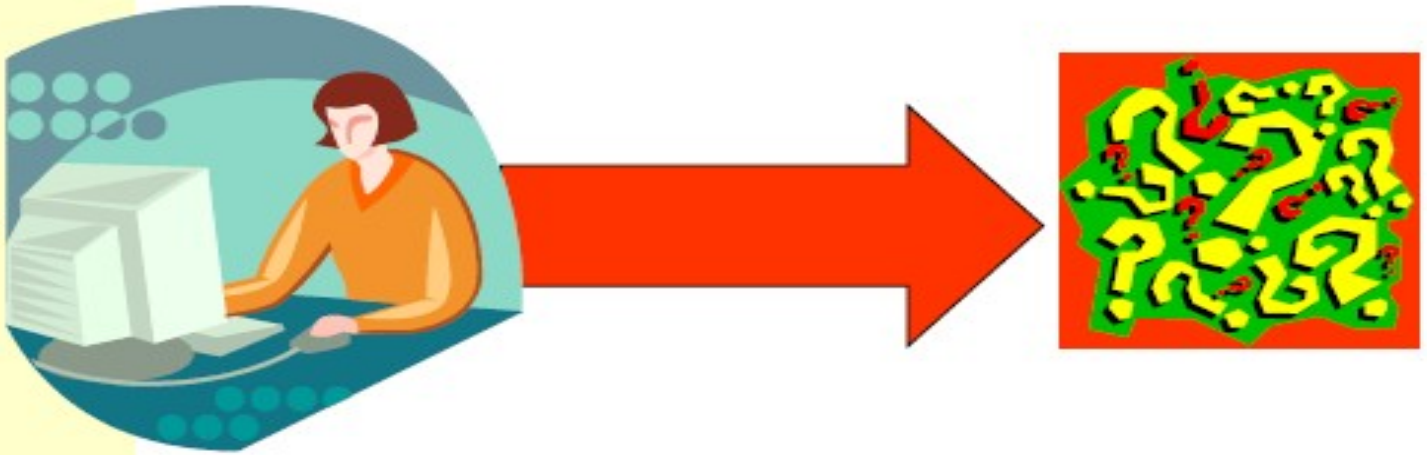


Large-scale,complex software systems...

- Large (Distributed) System
- Many people working on the same problem
- Overly Complex
- Millions of Code...
- Should be deliverd on time and within budget!

Coding only will not do…

- Lists
- Arrays
- Class
- Object
- Procedures
- Functions
- Algorithms
- Etc.



More programmers...?

Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships between them.

DESIGN PRINCIPLES

DESIGN PRINCIPLES

1- The design process should not suffer from "tunnel vision."

A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job, and the design concepts.

2- The design should be traceable to the analysis model.

DESIGN PRINCIPLES

3- The design should not reinvent the wheel.

Systems are constructed using a set of design patterns.

These patterns should always be chosen as an alternative to reinvention.

Time is short and resources are limited!

DESIGN PRINCIPLES

4- The design should “minimize the intellectual distance” between the software and the problem as it exists in the real world.

5- The design should exhibit uniformity and integration

DESIGN PRINCIPLES

6- The design should be structured to accommodate change

7- The design should be assessed for quality as it is being created

DESIGN CONCEPTS

FUNDAMENTAL CONCEPTS OF DESIGN

abstraction—data, procedure, control

refinement—elaboration of detail for all abstractions

modularity—compartmentalization of data and function

architecture—overall structure of the software

- Styles and patterns

procedure—the algorithms that achieve function

hiding—controlled interfaces

ABSTRACTION

"Capture only those details about an object that are relevant to current perspective"

Suppose we want to implement abstraction for the following statement,

"Ali is a PhD student and teaches BS students"

*Here object Ali has two **perspectives** one is his **student perspective** and second is his **teacher perspective.***

ABSTRACTION

A cat can be viewed with different perspectives.

Ordinary Perspective	Surgeon's Perspective
A pet animal with	A being with
Four Legs	A Skeleton
A Tail	Heart
Two Ears	Kidney
Sharp Teeth	Stomach

ABSTRACTION

Driver's View

Engineer's View

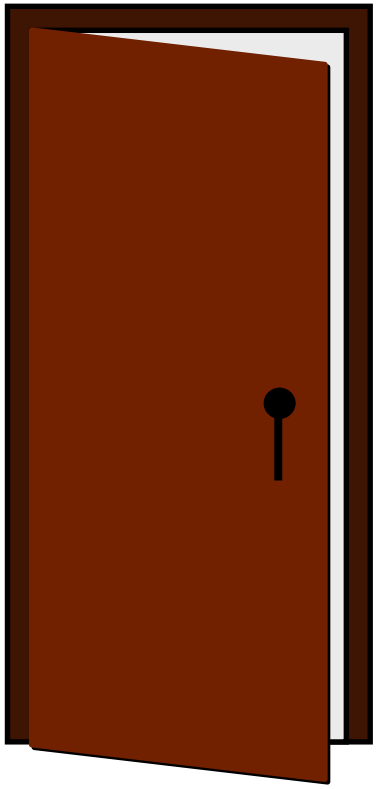
ABSTRACTION ADVANTAGES

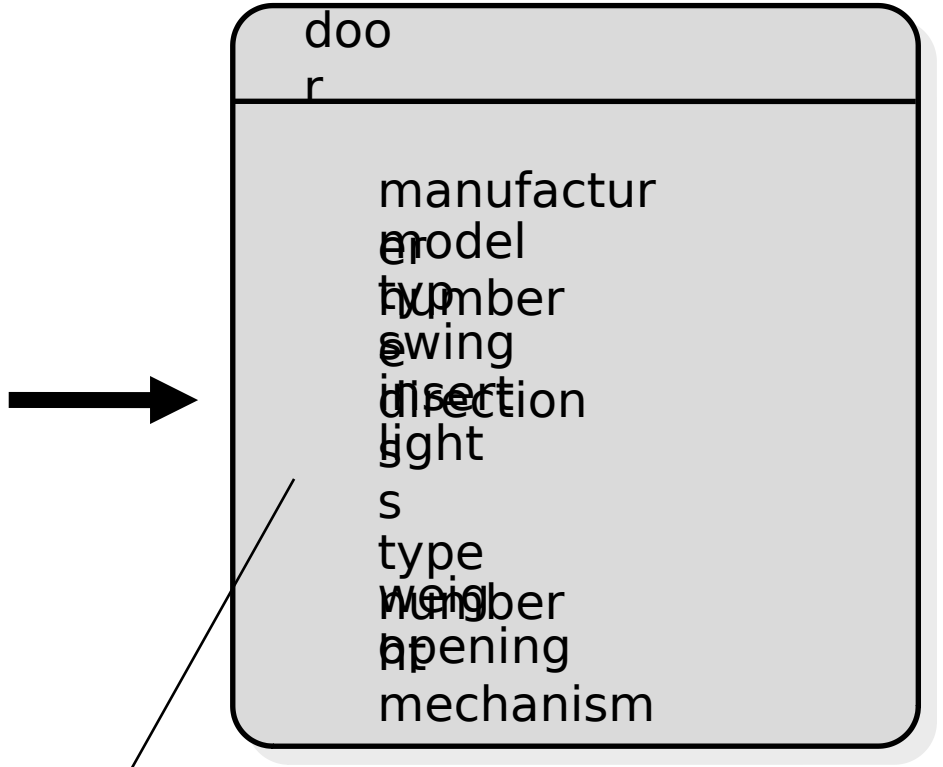
It helps us understanding and solving a problem using object oriented approach as it hides extra irrelevant details of objects.

Focusing on single perspective of an object provides us freedom to change implementation for other aspects of for an object later.

Abstraction is used for achieving information hiding as we show only relevant details to related objects, and hide other details.

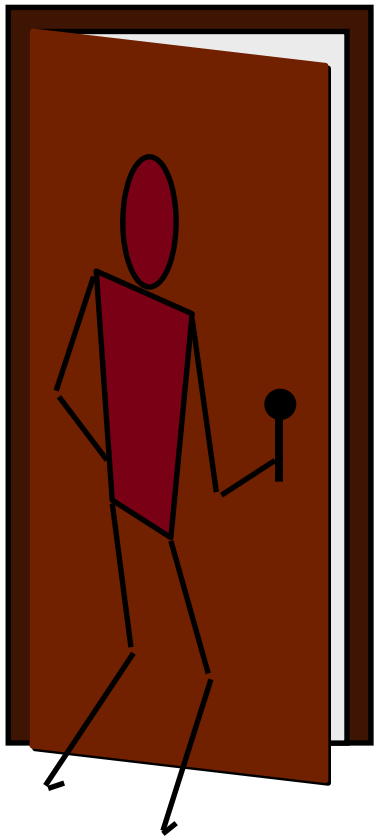
DATA ABSTRACTION

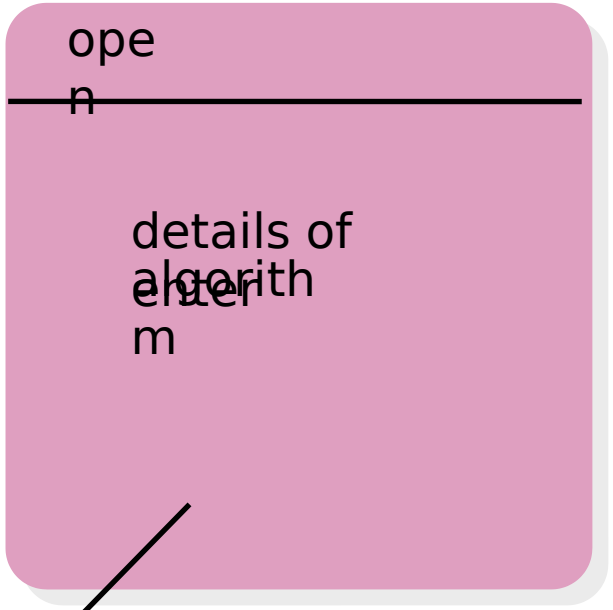




implemented as a data structure

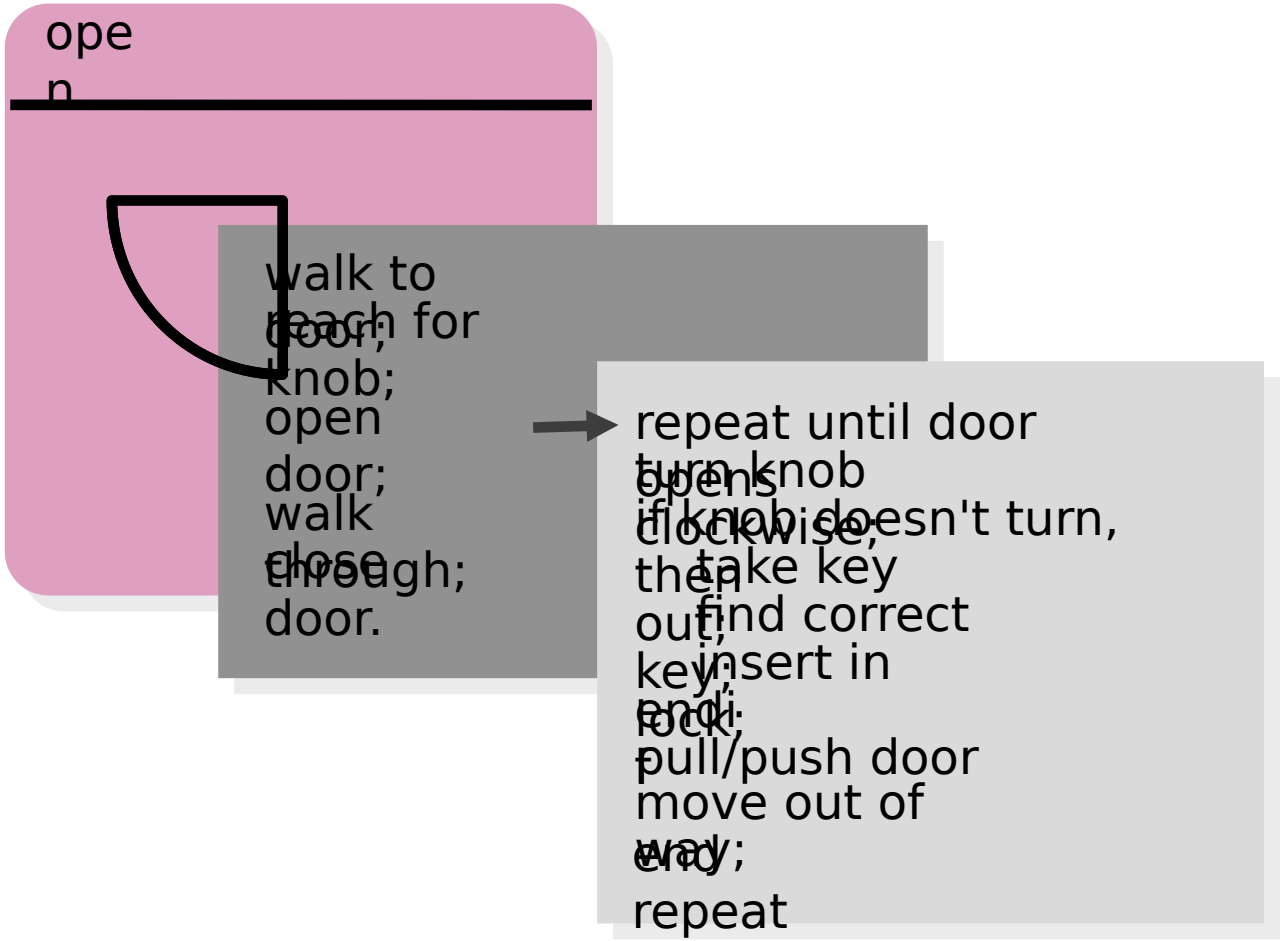
PROCEDURAL ABSTRACTION





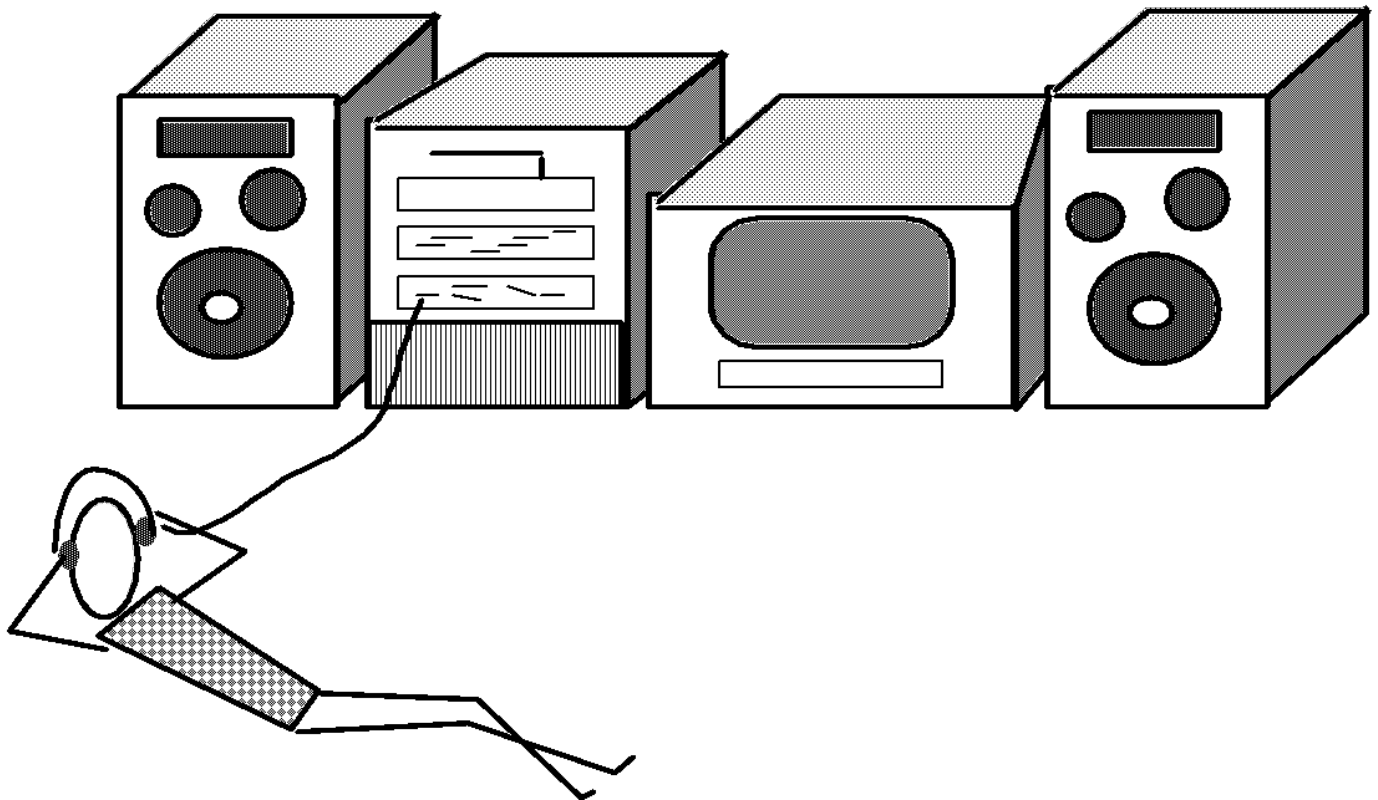
implemented with a "knowledge" of the object that is associated with enter

STEPWISE REFINEMENT



MODULAR DESIGN

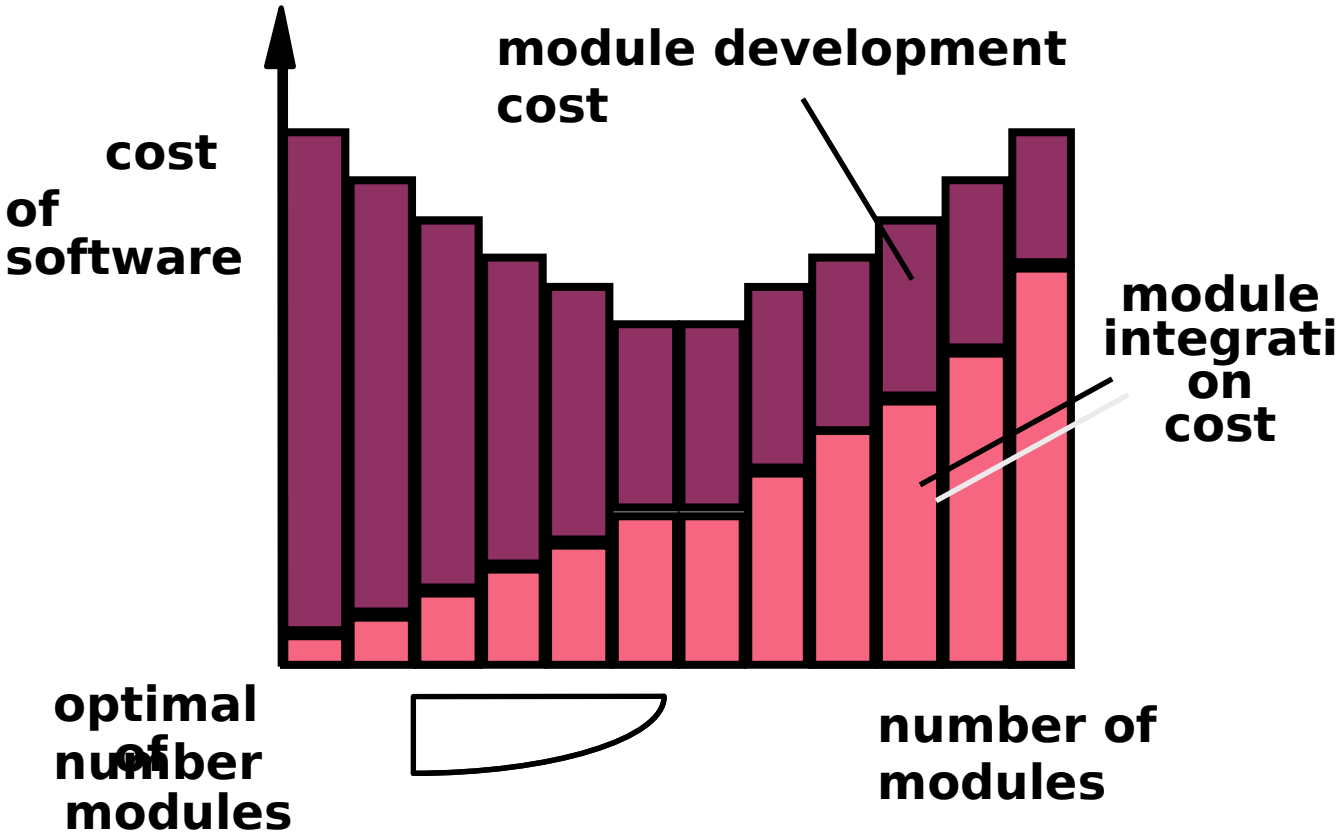
easier to build, easier to change, easier to fix …



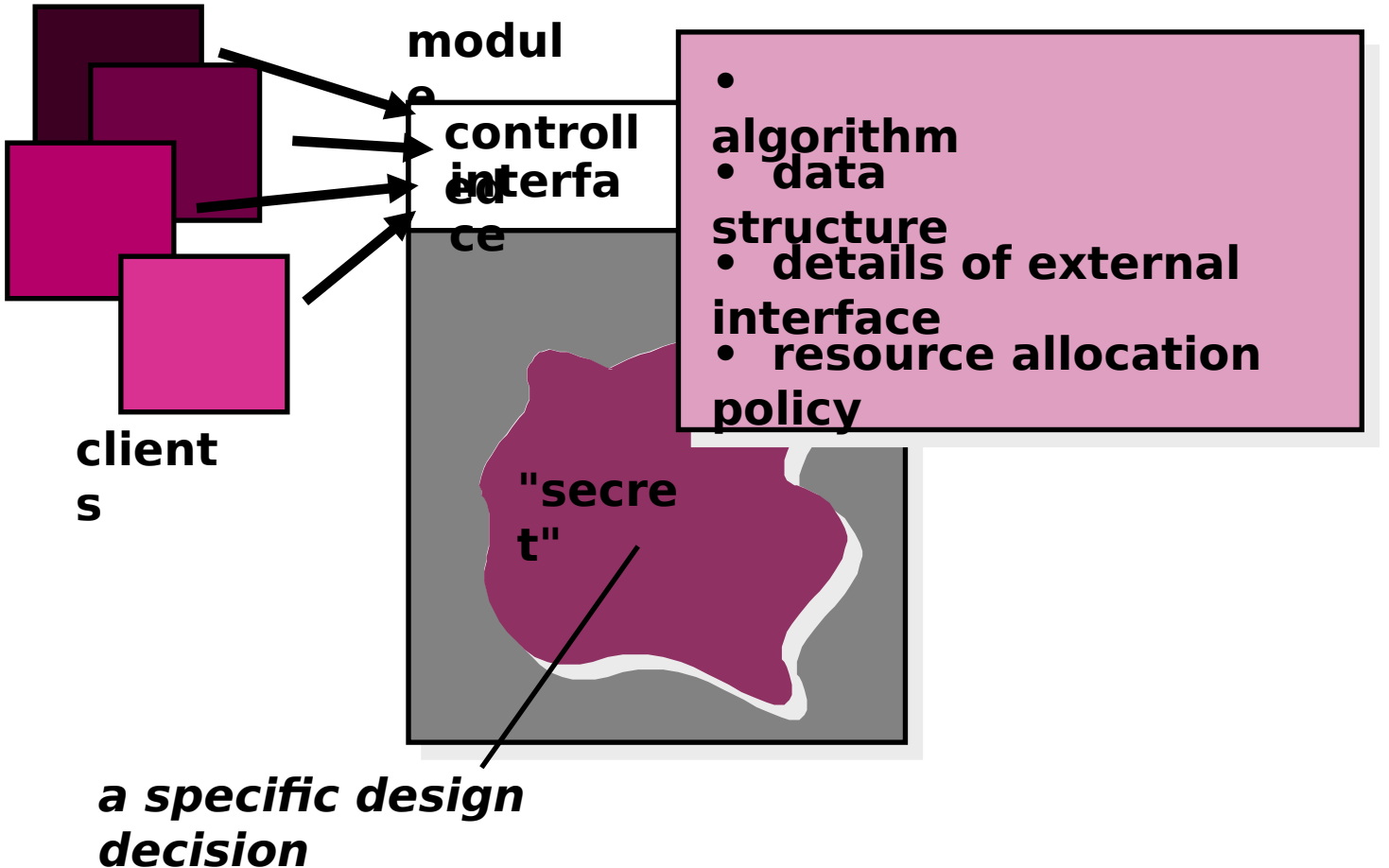
- **Easier to manage**
- **Easier to understand**
- **Reduces complexity**
- **Delegation / division of work**
- **Fault isolation**
- **Independent development**
- **Separation of concerns**
- **Reuse**

MODULARITY: TRADE-OFFS

What is the "right" number of modules for a specific software design?



INFORMATION HIDING



INFORMATION HIDING

Design the modules in such a way that information (data & procedures) contained in one module is inaccessible to other modules that have no need for such information.

Independent modules.

Benefits:

when modifications are required, it reduces the chances of propagating to other modules.

EFFECTIVE MODULAR DESIGN

FUNCTIONAL INDEPENDENCE

COHESION - the degree to which a module performs one and only one function.

COUPLING - the degree to which a module is "connected" to other modules in the system.

COUPLING

Coupling is a measure of independence of a module or component.

Loose coupling means that different system components have loose or less reliance upon each other.

Hence, changes in one component would have a limited affect on other components.

COUPLING

High coupling causes problems

- Change propagation- ripple effect
- Difficulty in understanding
- Difficult reuse

COHESION

Cohesion is a measure of the degree to which the elements of the module are functionally related.

It is the degree to which all elements directed towards performing a single task are contained in the component.

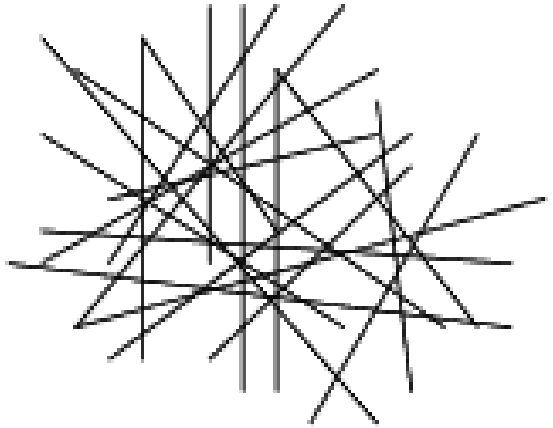
Basically, cohesion is the internal glue that keeps the module together.

A good software design will have high cohesion

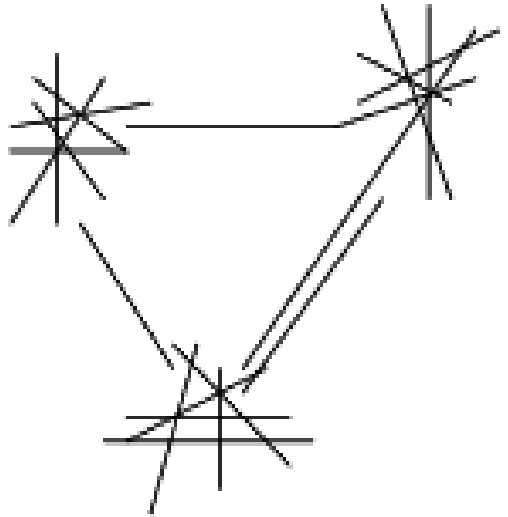
COUPLING & COHESION

A Software should be Lesly coupled and highly cohesive.

COUPLING

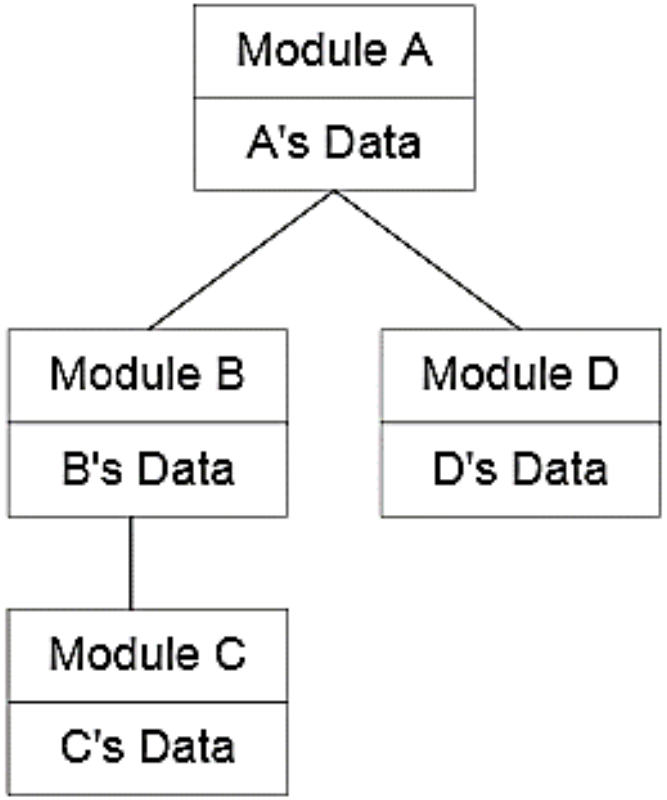


High Coupling

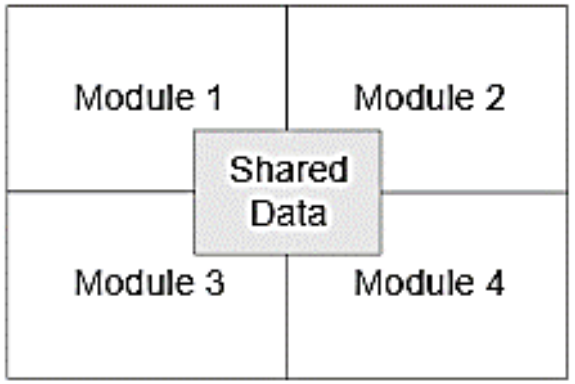


Low Coupling

COUPLING

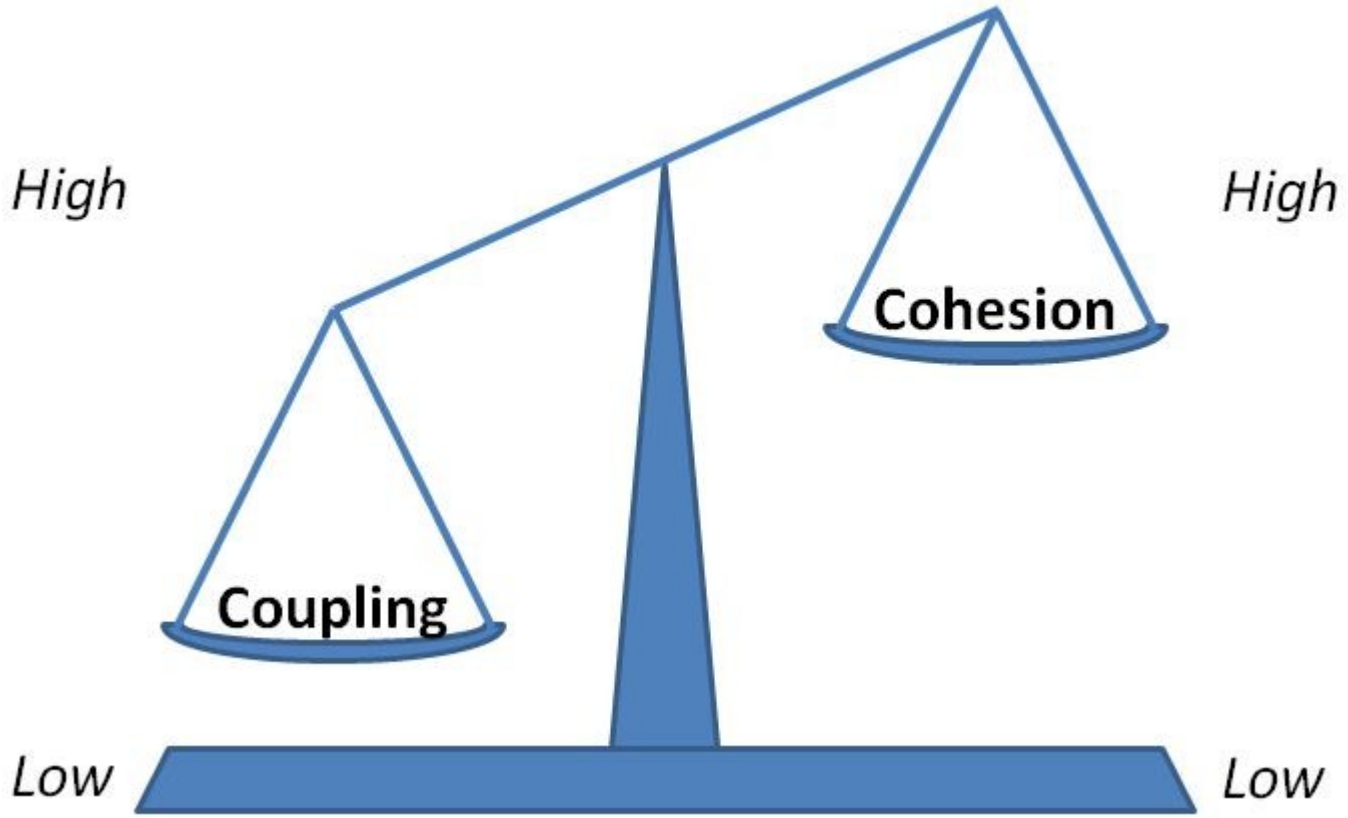


Low Coupling

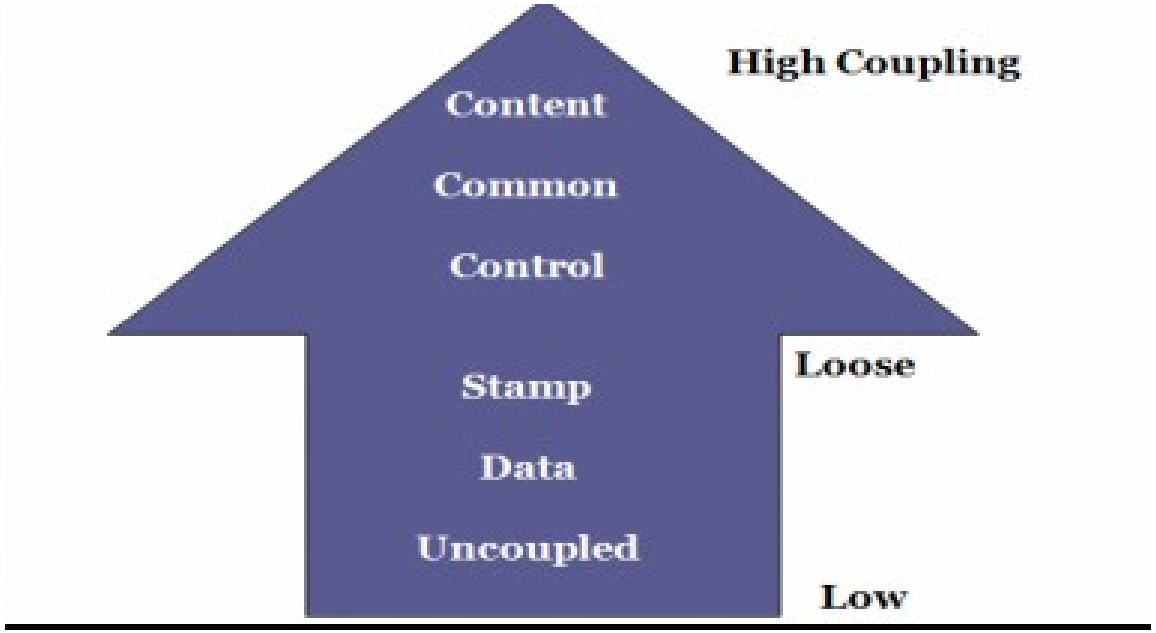


High Coupling

RELATIONSHIP BETWEEN COUPLING AND COHESION



TYPES OF COUPLING

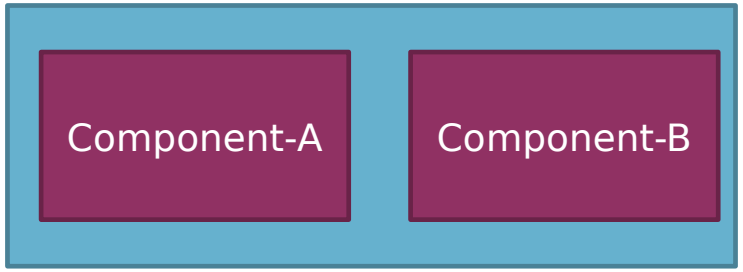


CONTENT COUPLING

This is highest level of coupling and occurs if there are two (or more) modules and if one refers to the ``inside'' - the ``internal'' or ``private'' part - of the other in some way.

Component A handles lookup data.

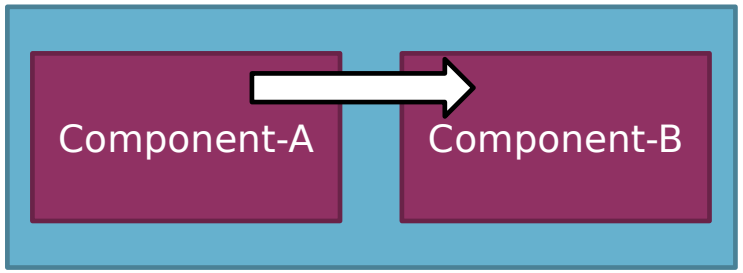
Component is handling the property of adding new customer when needed.



CONTENT COUPLING EXAMPLE

When we are searching the customer data and customer is not found, lookup component adds customer by directly modifying the contents of the data structure containing customer data (new customer).

This is a very high level of coupling in which one component is directly modifying the content of another component and this is not wanted in any software design.



CONTENT COUPLING EXAMPLE

Solution:

When customer not found, component calls the AddCustomer () method that is responsible for maintaining customer data rather than directly modifying data structure.

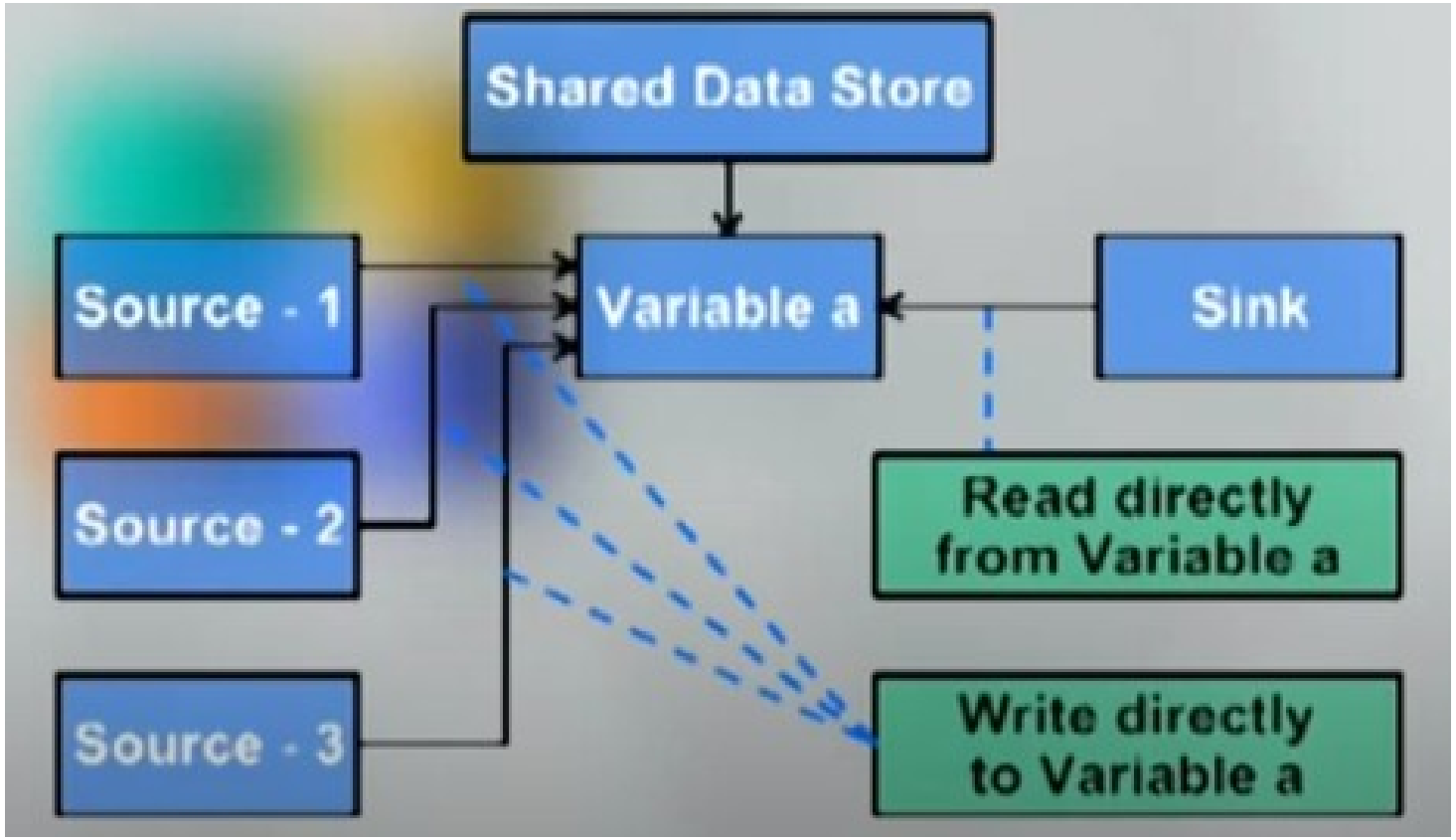
COMMON COUPLING

Common coupling occurs when modules communicate using global data structures.

For example, programming allows the developer to declare a data element as external, enabling it to be accessed by all modules.

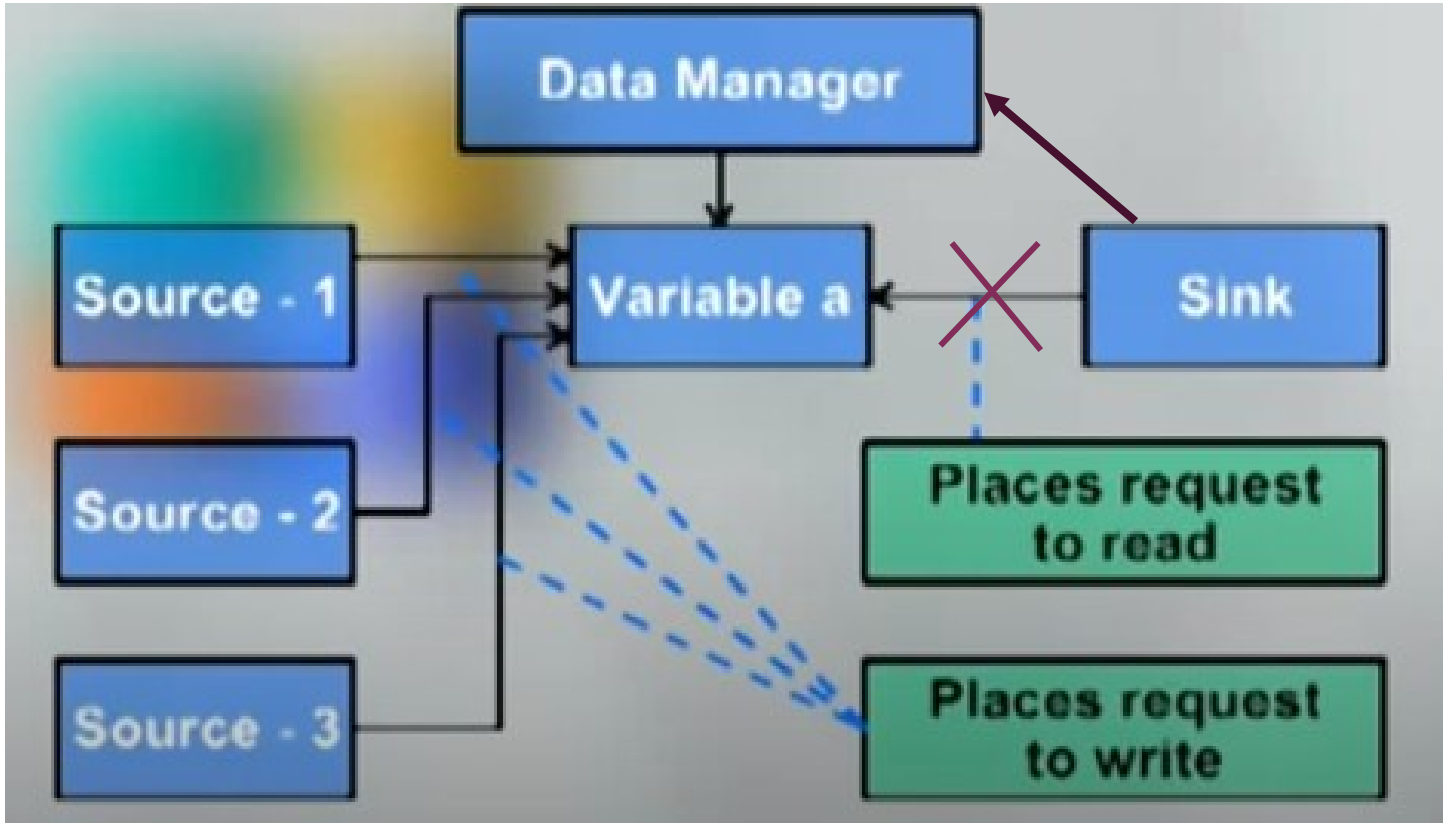
Common coupling is also known as “Global coupling”. We can say that two components share data using Global data structures.

COMMON COUPLING EXAMPLE



COMMON COUPLING

Solution:



HAVE A GOO DAY!