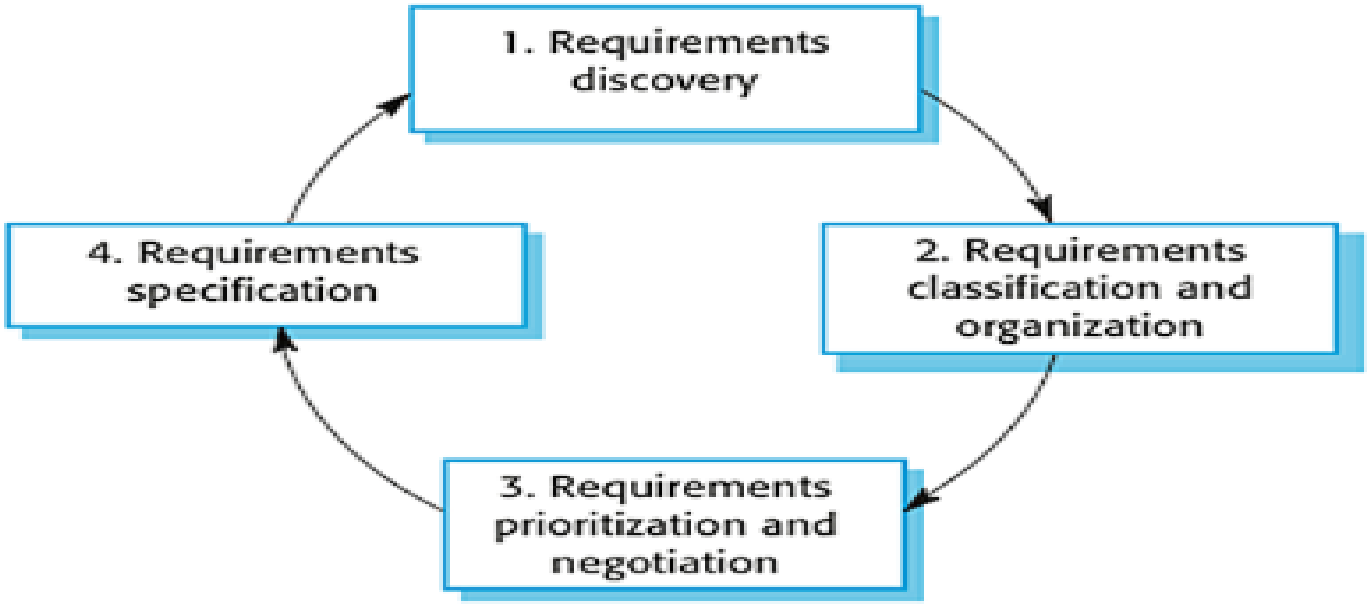
Requirement Engineering

Kiran Waghmare

Session VI



The process of requirements elicitation and analysis

Requirement Engineering

Requirements Engineering is the process of establishing the services that the customer requires from the system and the constraints under which it is to be developed and operated

Requirements may serve a dual function:

- As the basis of a bid for a contract
- As the basis for the contract itself

Steps for Requirement Engineering

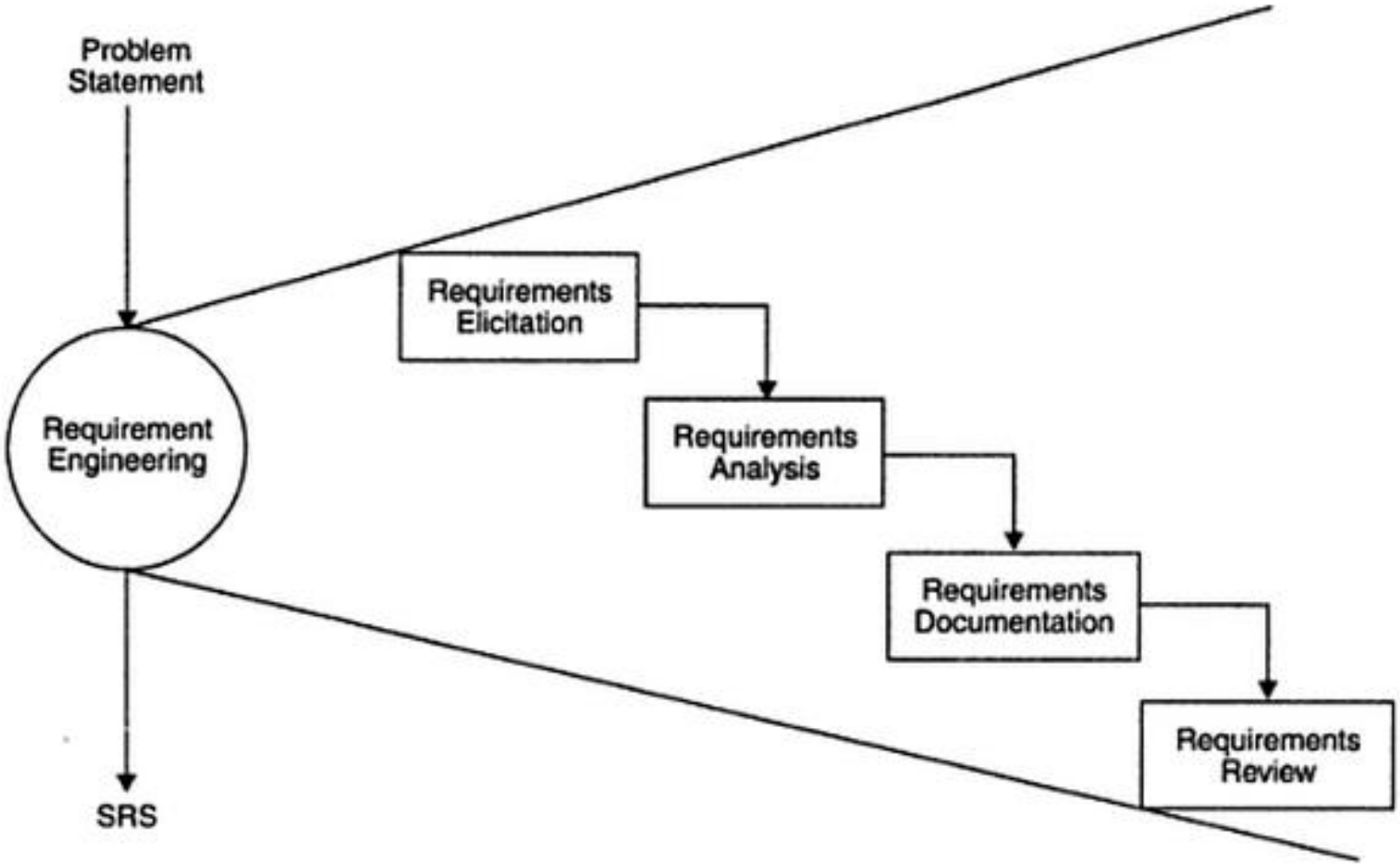
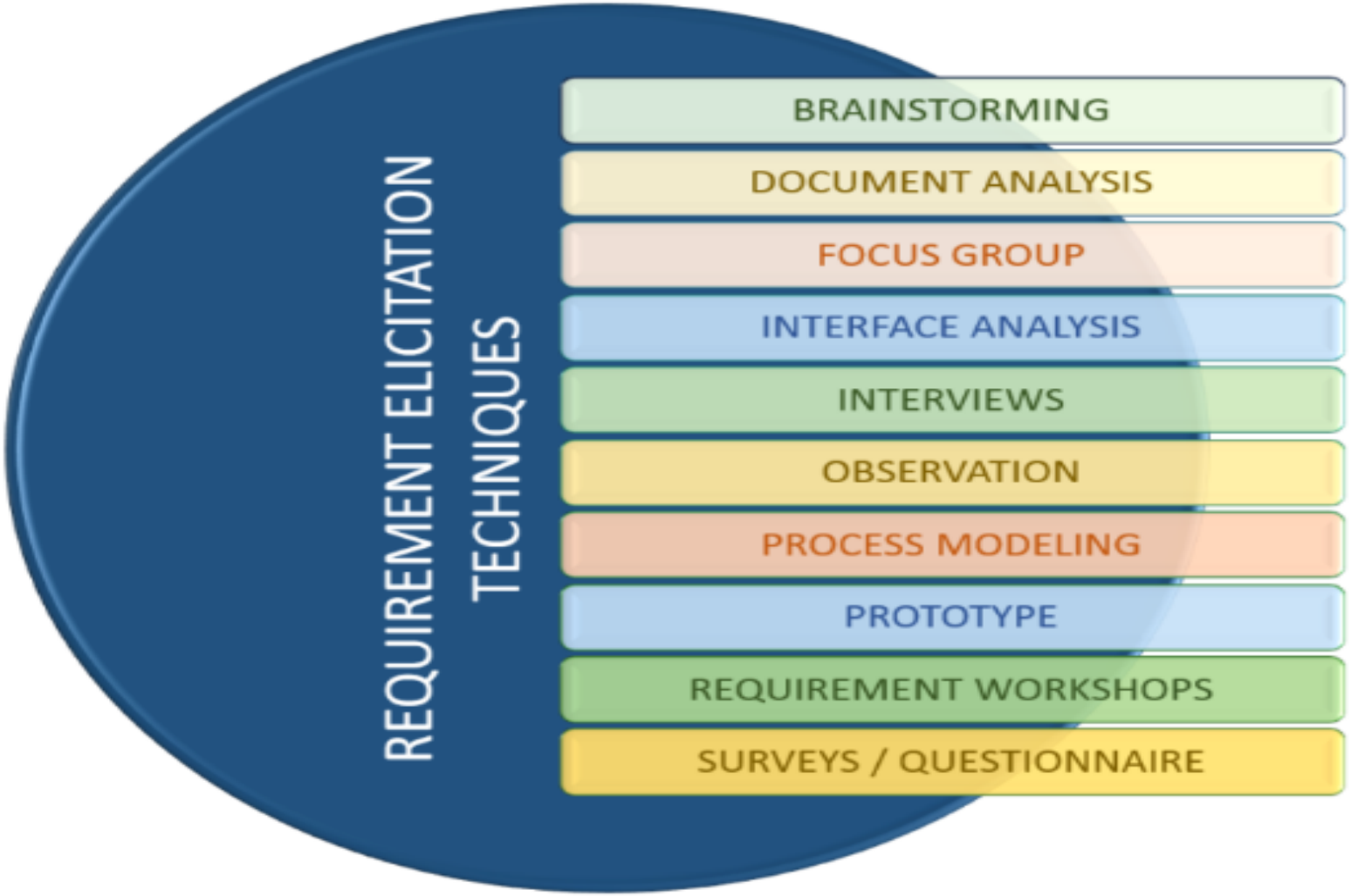


Fig. 3.1: Crucial process steps of requirement engineering.

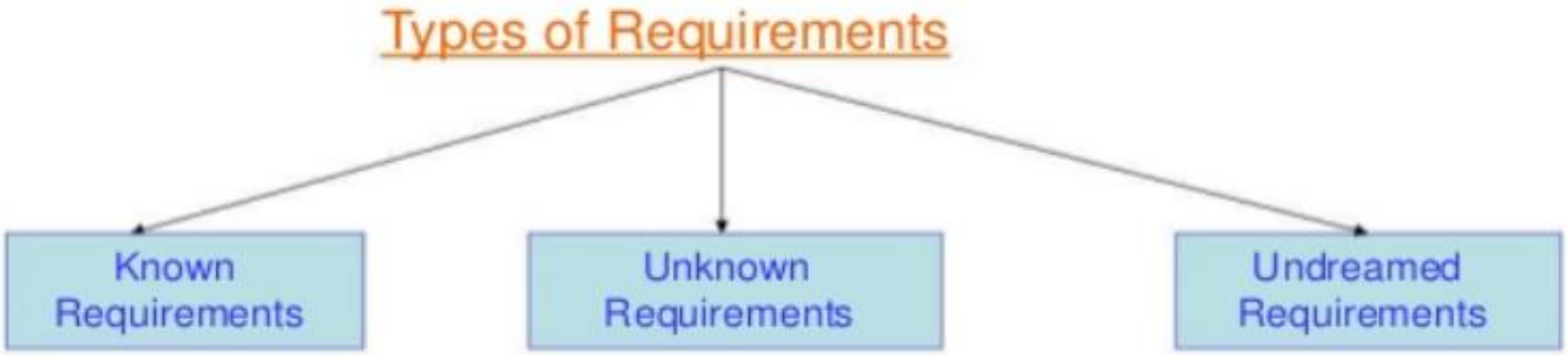
Techniques for Requirement Elicitation



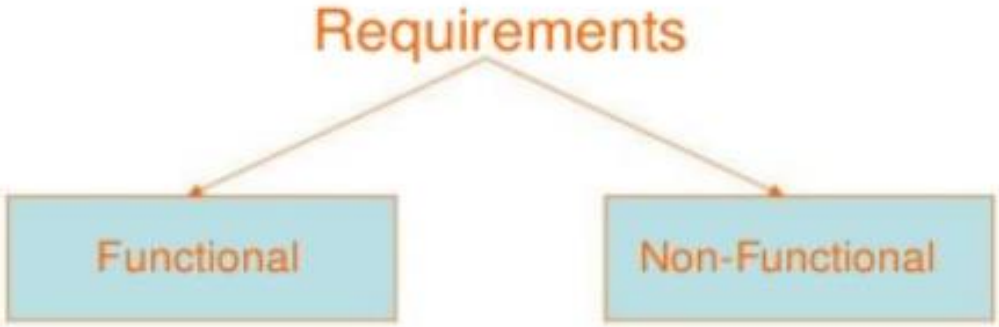
REQUIREMENTS ELICITATION

- Elicitation Techniques
  - Documentation Review
  - Brainstorming
  - Interviewing
  - Apprenticing
  - Scenario Analysis
  - Prototyping/Mock-up
  - Modeling
  - Workshops

Types of Requirements

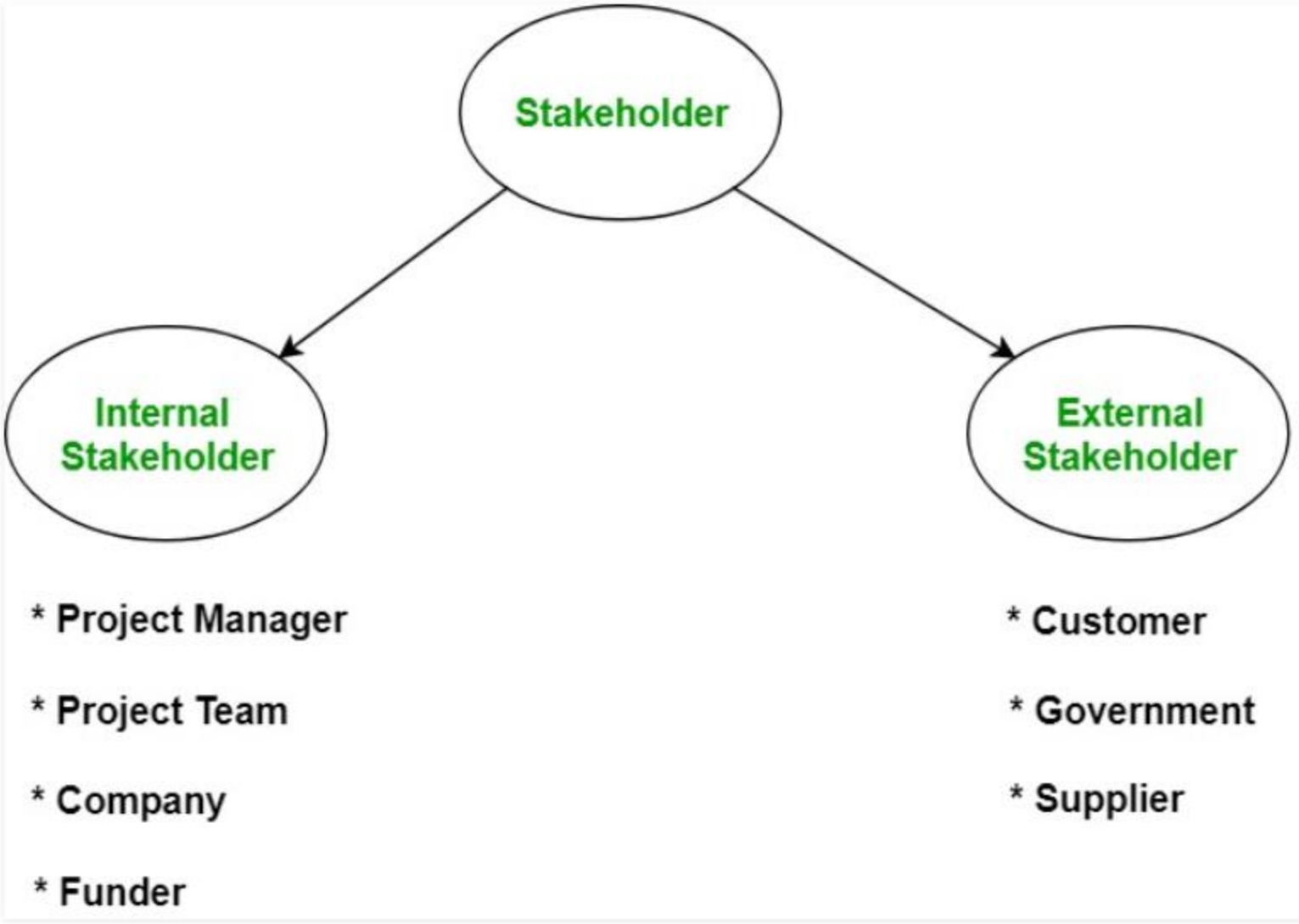


Stakeholder: Anyone who should have some direct or indirect influence on the system requirements.



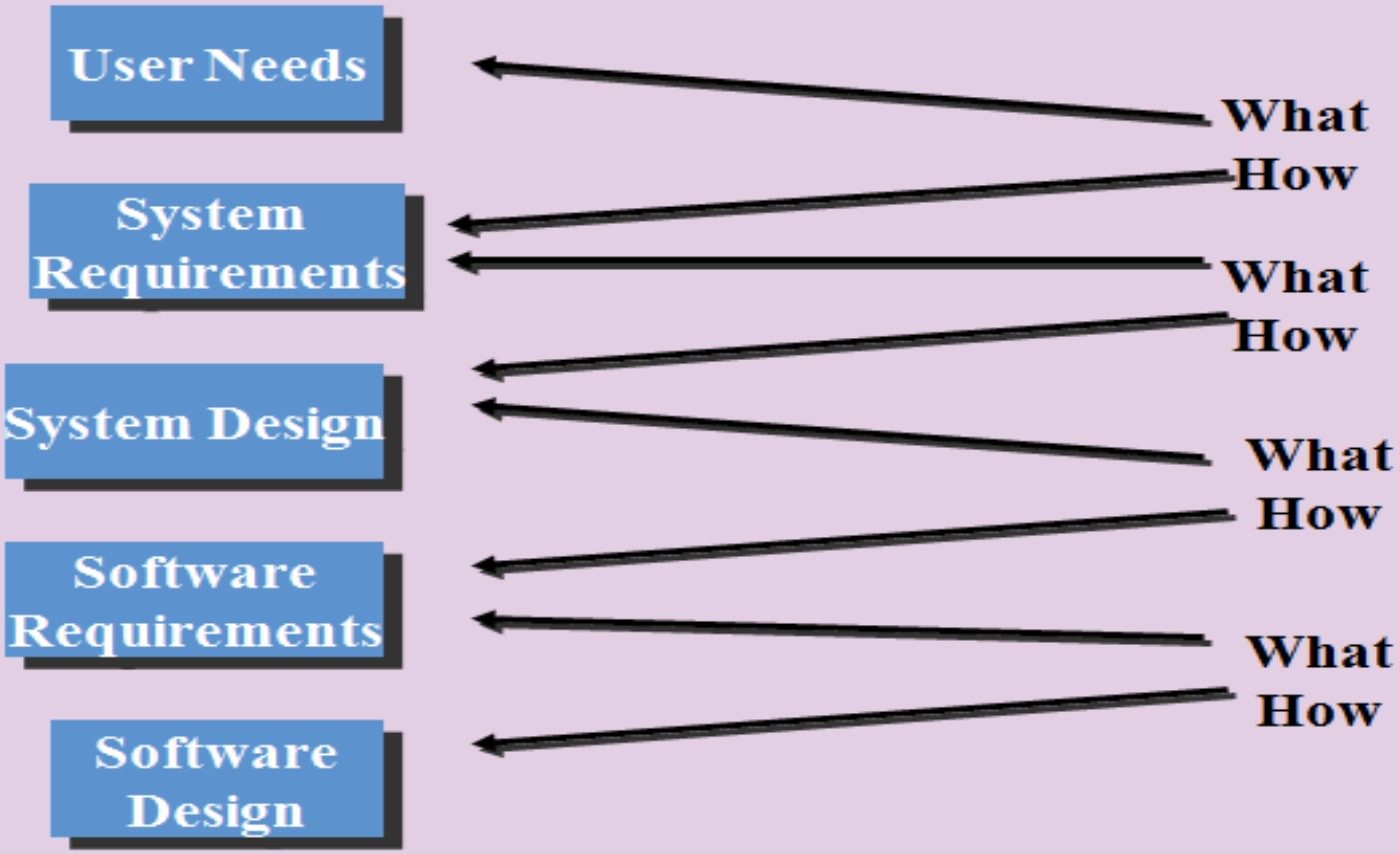
Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.
  - While developing software, ‘Must have’ must be implemented, ‘Should have’ is a matter of debate with stakeholders and negation, whereas ‘could have’ and ‘wish list’ can be kept for software updates.


Stakeholder
Internal Stakeholder
External Stakeholder
* Project Manager
* Project Team
* Company
* Funder
* Customer
* Government
* Supplier

Specification Phase

What vs. How Dilemma[3]



Requirements vs. Design

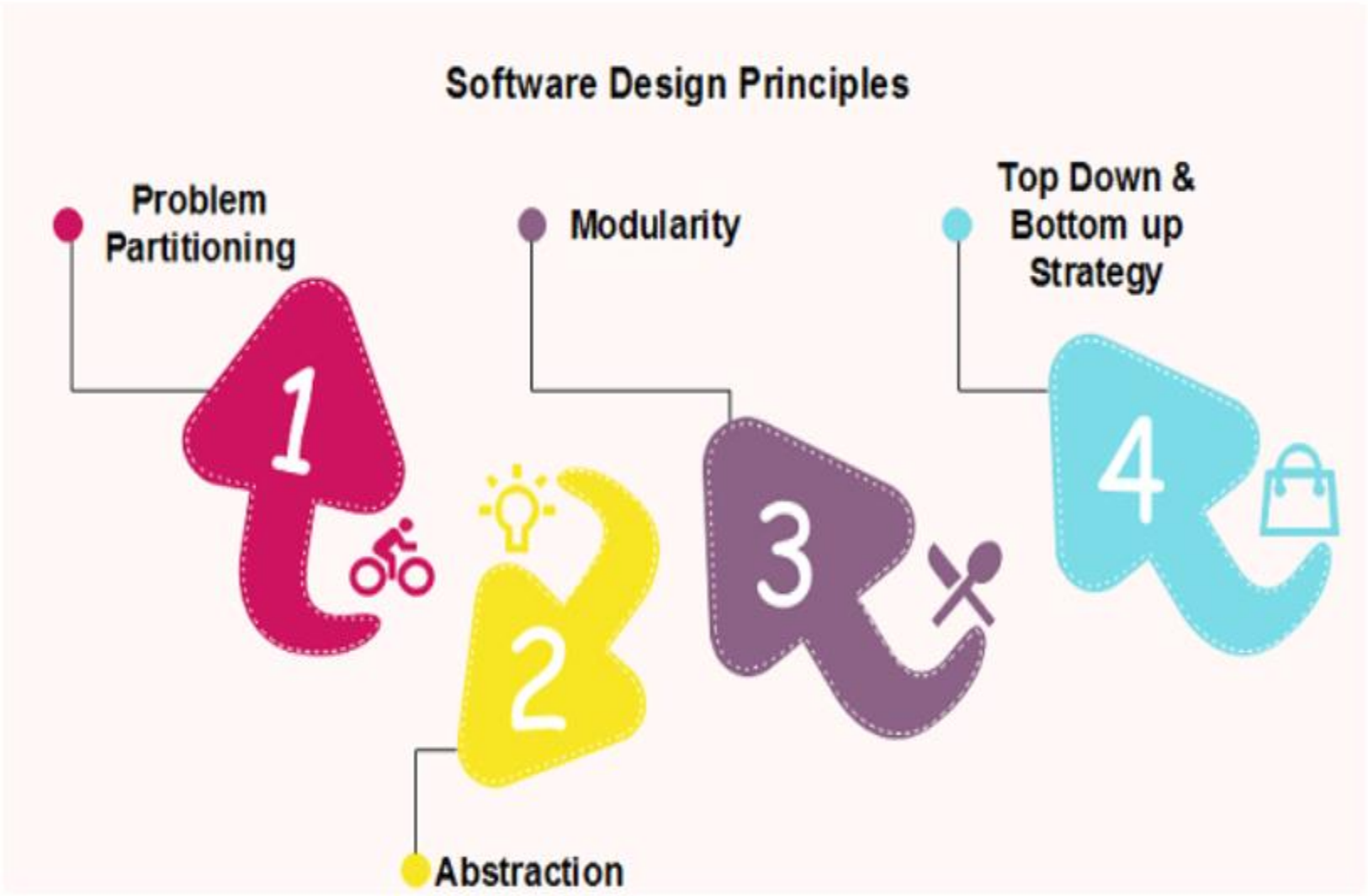
Requirements	Design
Describe what will be delivered[3]	Describe how it will be done[3]
Primary goal of analysis: **UNDERSTANDING**[3]	Primary goal of design: **OPTIMIZATION**[3]
There is more than one solution	There is only one (final) solution
Customer interested	Customer not interested (Most of the time) except for external

Key Difference between Structured and OOA and OOD

Phase	Structured paradigm	Object –Oriented paradigm
Analysis	**Structured Requirements** • DFD • Structure Language • Decision Table/Tree • ER Analysis	**Requirement Engineering** • Use - Case Model • Object Model
Design	**DB Engine** • DB Normalization • GUI design	**Physical DB Design** • Design Elements • Design System Architecture • Design Classes • GUI Design

Software Design

Kiran waghmare

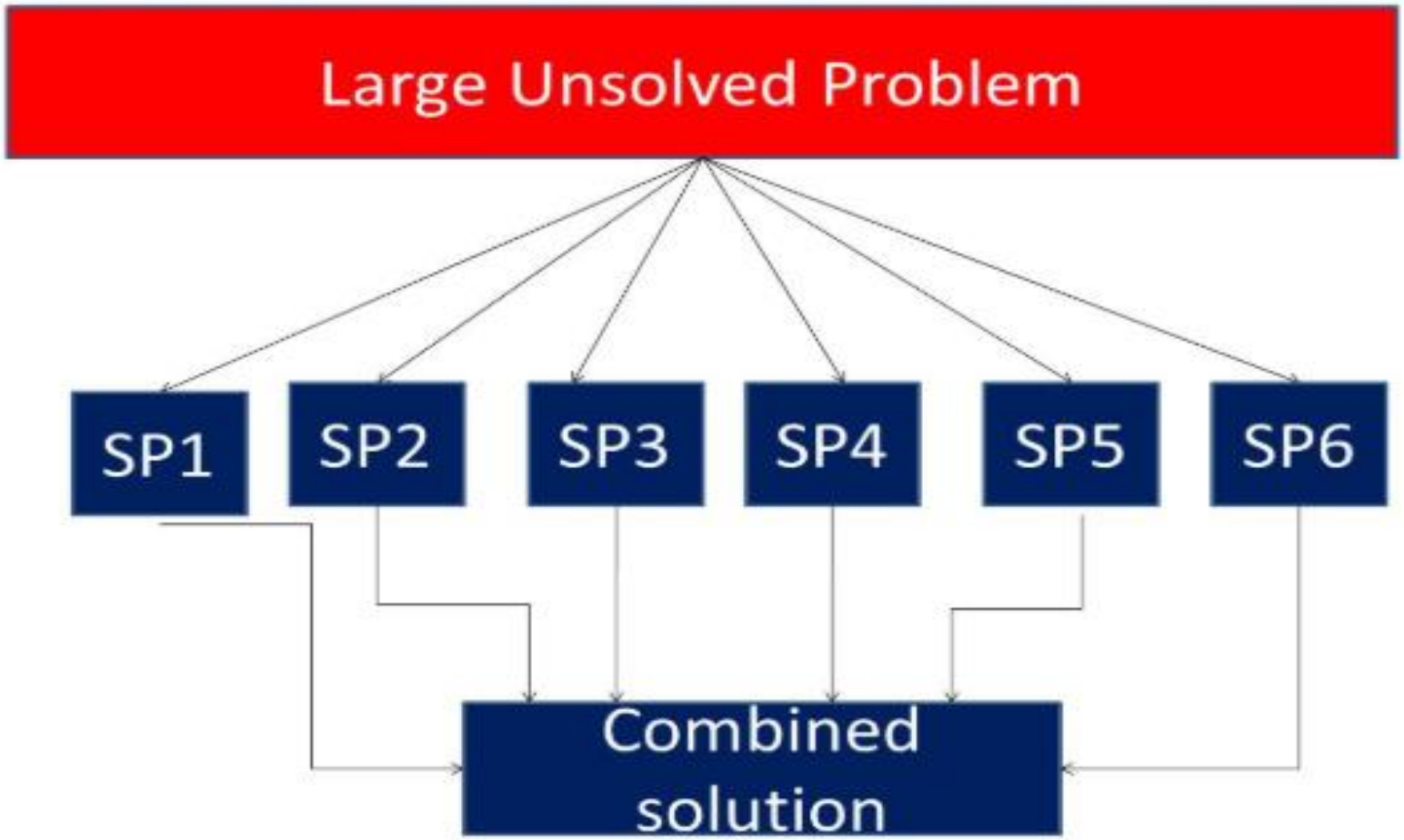

Software Design Principles
Problem Partitioning
Modularity
Top Down & Bottom up Strategy
1
2
3
4
Abstraction

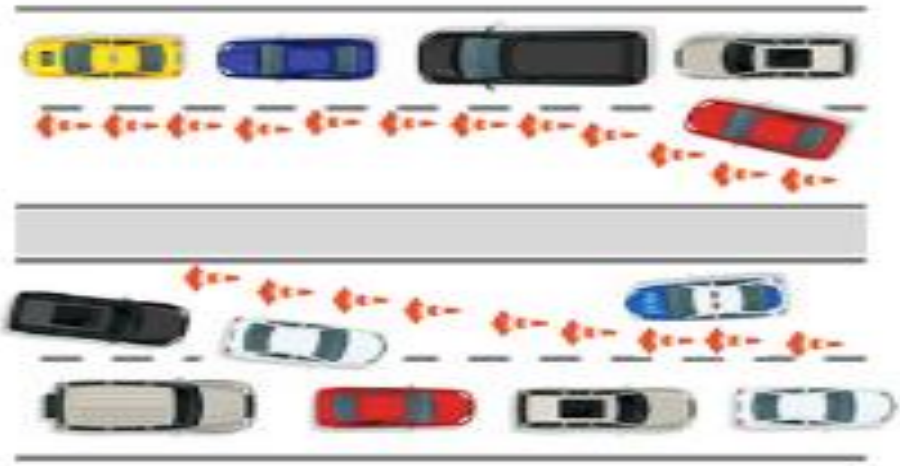
Benefits of Problem Partitioning

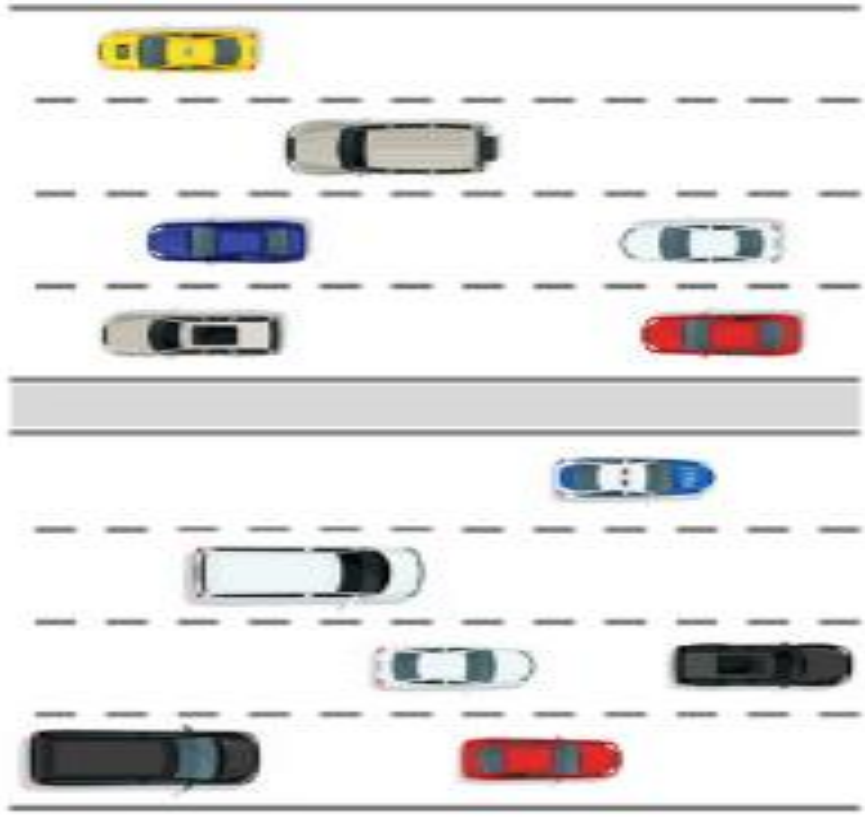
- Software is easy to **understand**
- Software becomes **simple**
- Software is easy to **test**
- Software is easy to **modify**
- Software is easy to **maintain**
- Software is easy to **expand**

Students Management System

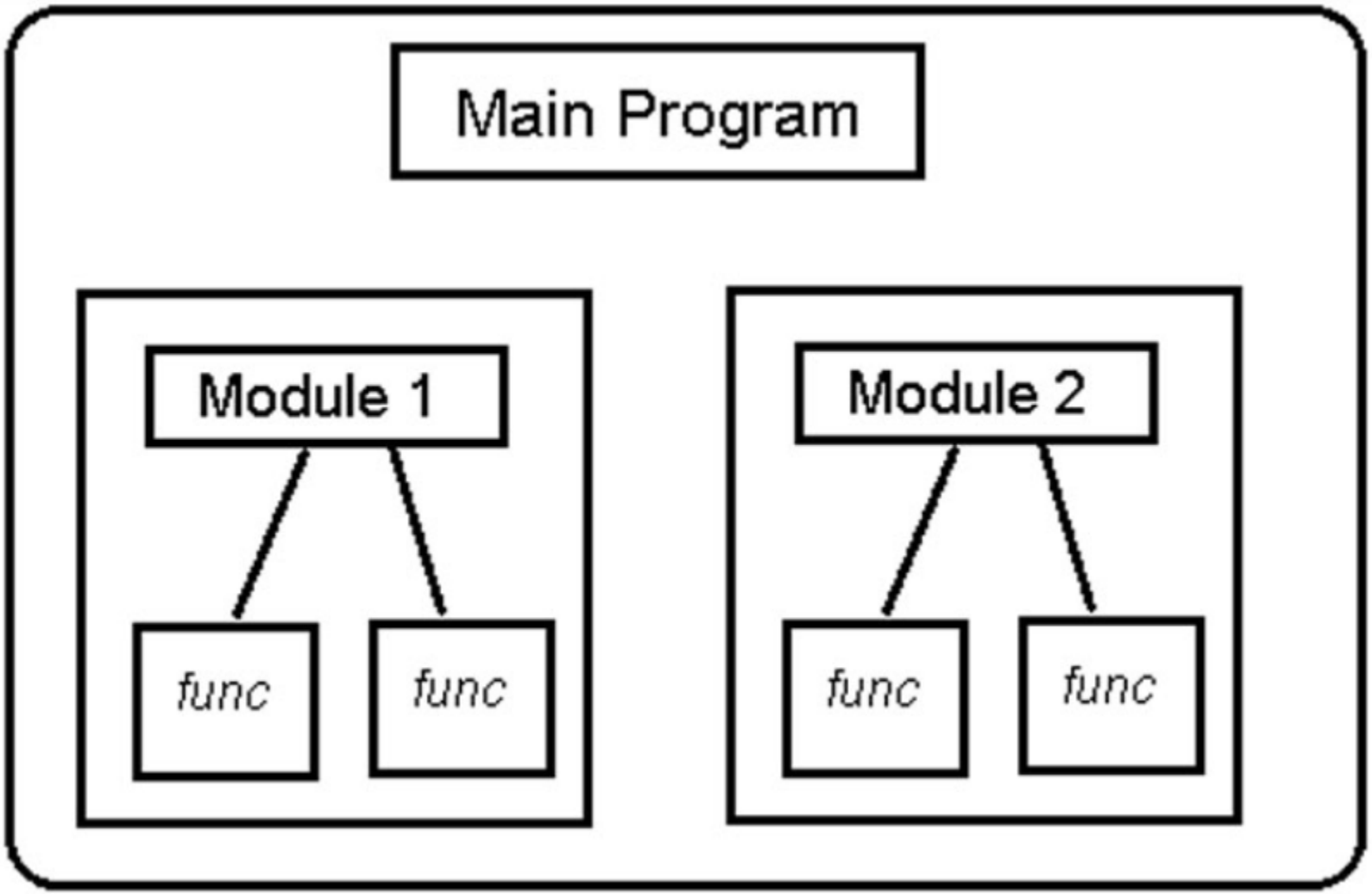
Decomposition Example



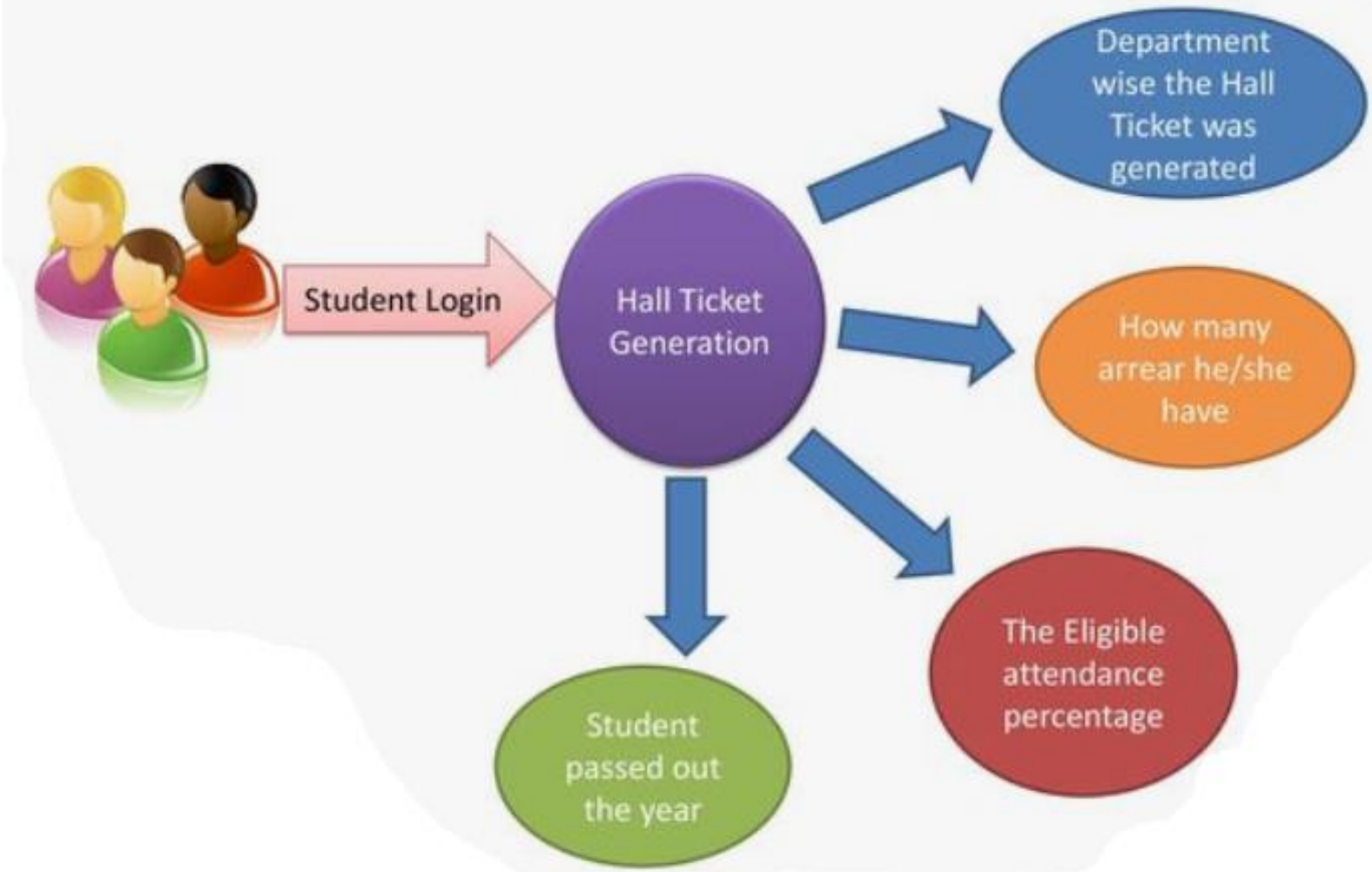




Modular Design



Examination Management software

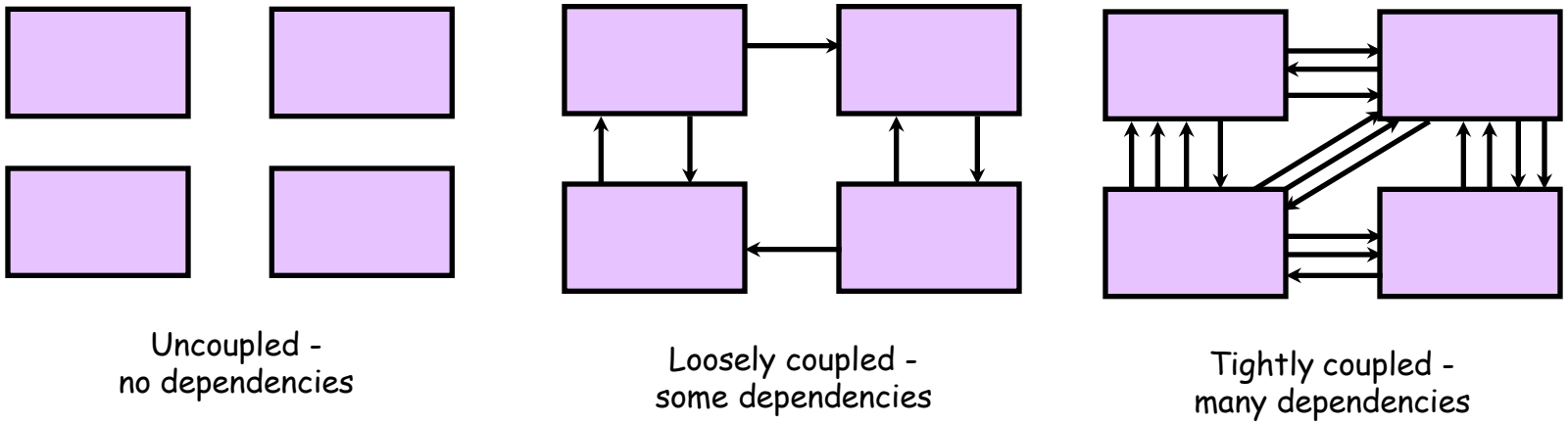


Coupling

- Coupling is **measure of the independence of components.**
- Coupling is **related to** cohesion
- It is an indication the **strength of interconnections between the components** in a design

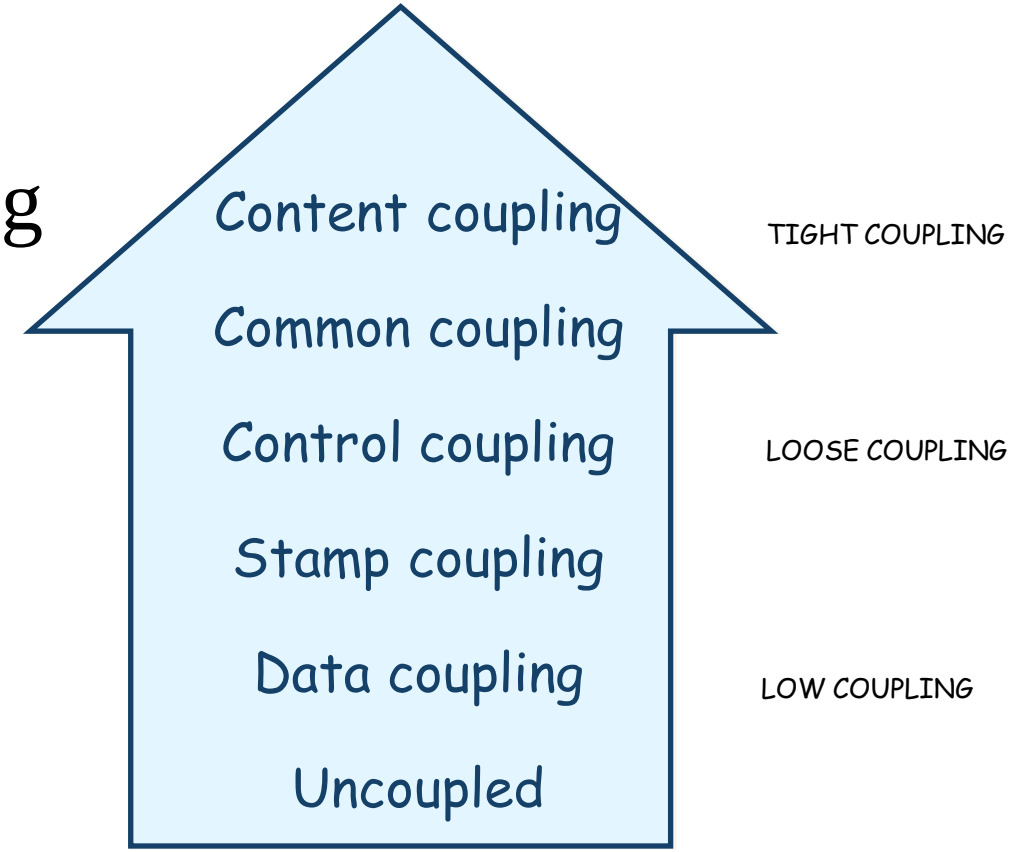
Types of Coupling

- Two modules are **tightly coupled** when they depend a great deal on each other
- **Loosely coupled** modules have some dependence, but their interconnections are weak
- **Uncoupled** modules have no interconnections at all; they are completely unrelated



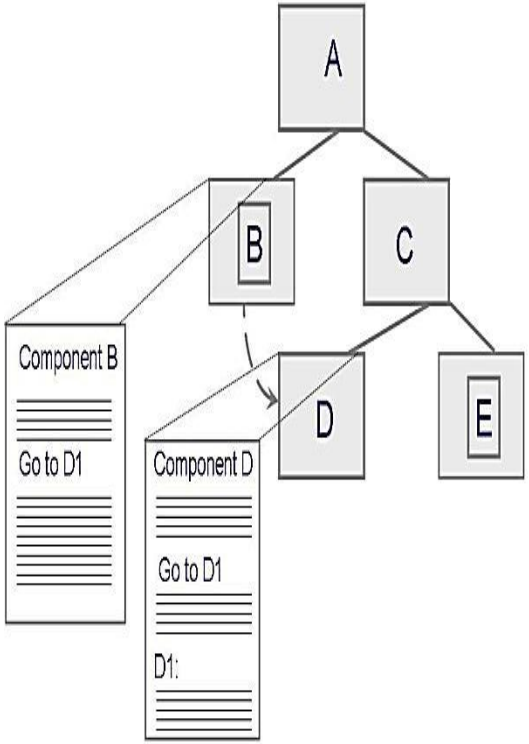
Coupling: Types of Coupling

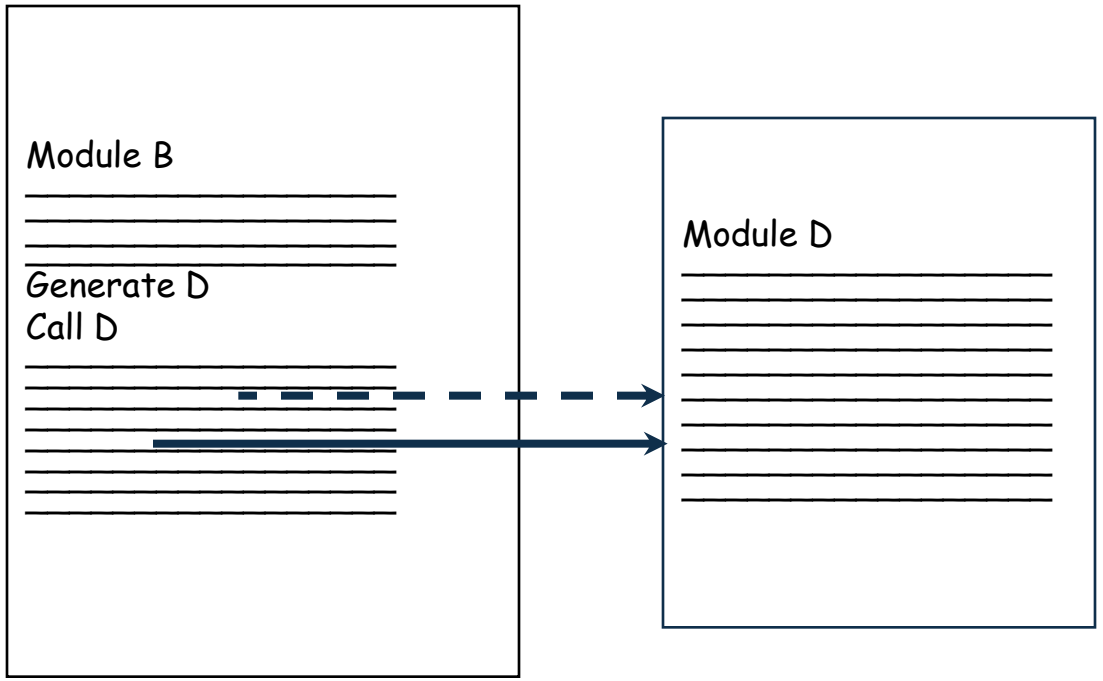
1. Content coupling
2. Common coupling
3. Control coupling
4. Stamp coupling
5. Data coupling



Content Coupling

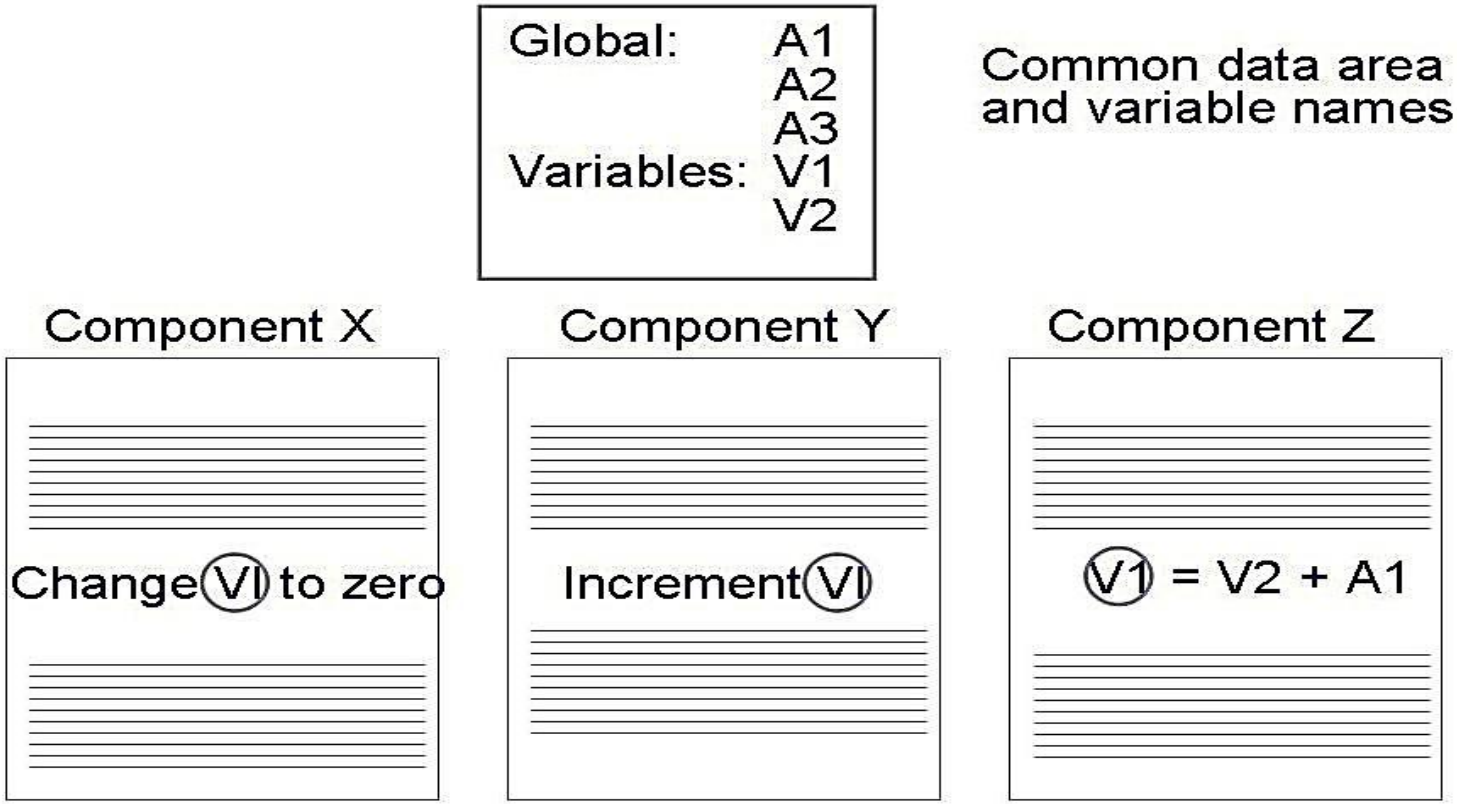
- Occurs when one component modifies an internal data item in another component, or when one component branches into the middle of another component





Common Coupling

- Making a change to the common data means tracing back to all components that access those data to evaluate the effect of the change



Control Coupling

- When one module **passes parameters or a return code** to control the behavior of another module

- It is impossible for the controlled module to function **without some direction** from the controlling module

Stamp Coupling

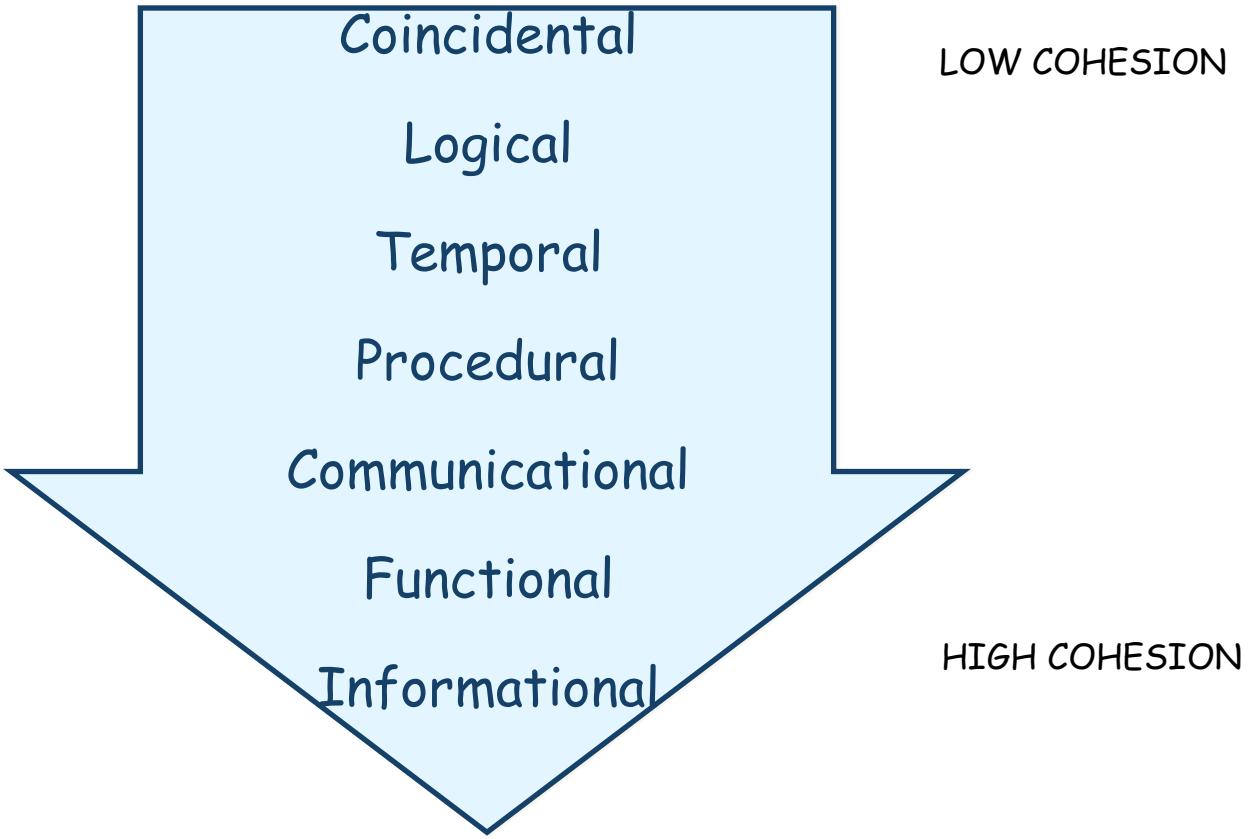
- Stamp coupling occurs **when complex data structures are passed** between modules
  - Stamp coupling represents a more complex interface between modules, because the modules have to agree on the data's format and organization

Data Coupling

- If **only data values, and not structured data, are passed,** then the modules are connected by **data coupling**
  - Data coupling is simpler and less likely to be affected by changes in data representation

Cohesion

- Cohesion refers to the **dependence within and among a module's internal elements**
- (e.g., data, functions, internal modules)



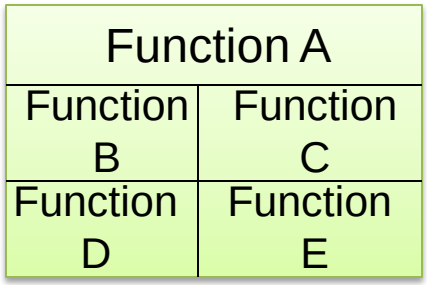
Cohesion (continued)

- Coincidental (worst degree)
  - **Parts are unrelated** to one another
- Logical
  - Parts are **related only by the logic** structure of code
- Temporal
  - Module's **data and functions related** because they are used at the same time in an execution
- Procedural
  - Similar to **temporal**, and functions pertain to some related action or purpose

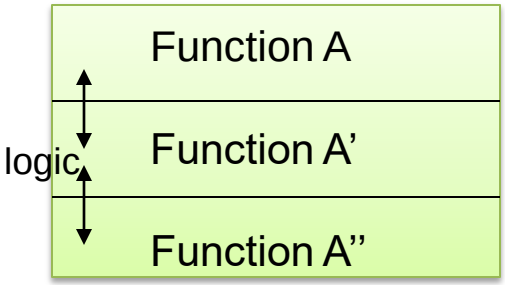
Cohesion

- Communication
  - Operates on the **same data set**
- Functional (ideal degree)
  - **All elements essential** to a single function are contained in one module, and all of the elements are essential to the performance of the function
- Informational
  - **Adaption of functional cohesion** to data abstraction and object-based design

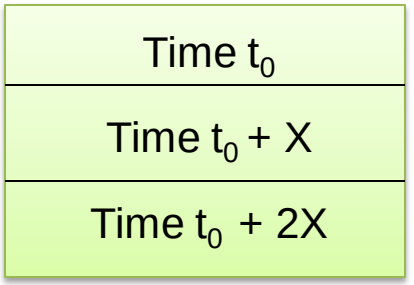
Examples of Cohesion-1



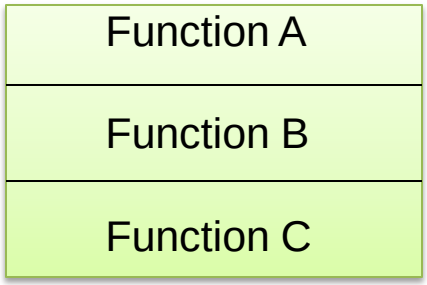
Coincidental
Parts unrelated



Logical
Similar functions

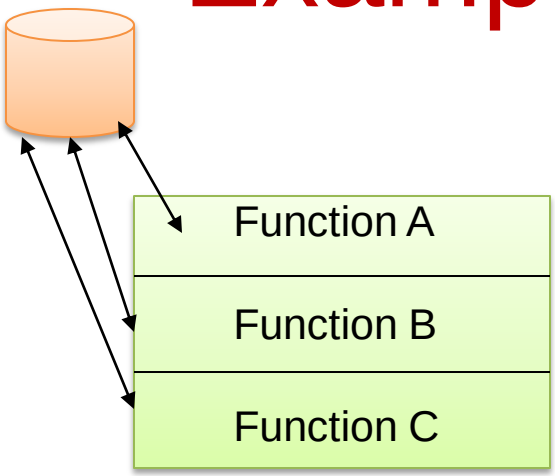


Temporal
Related by time

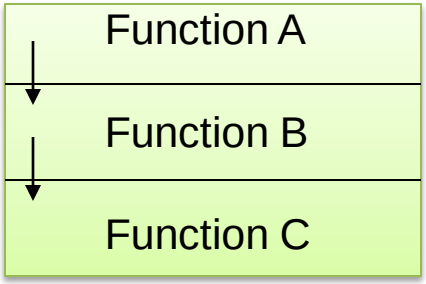


Procedural
Related by order of functions

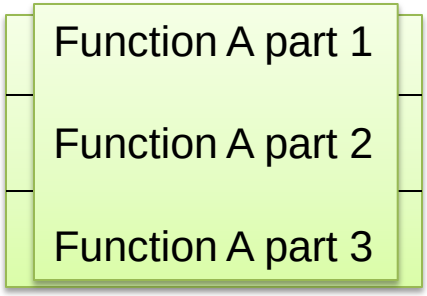
Examples of Cohesion-2



Communicational
Access same data

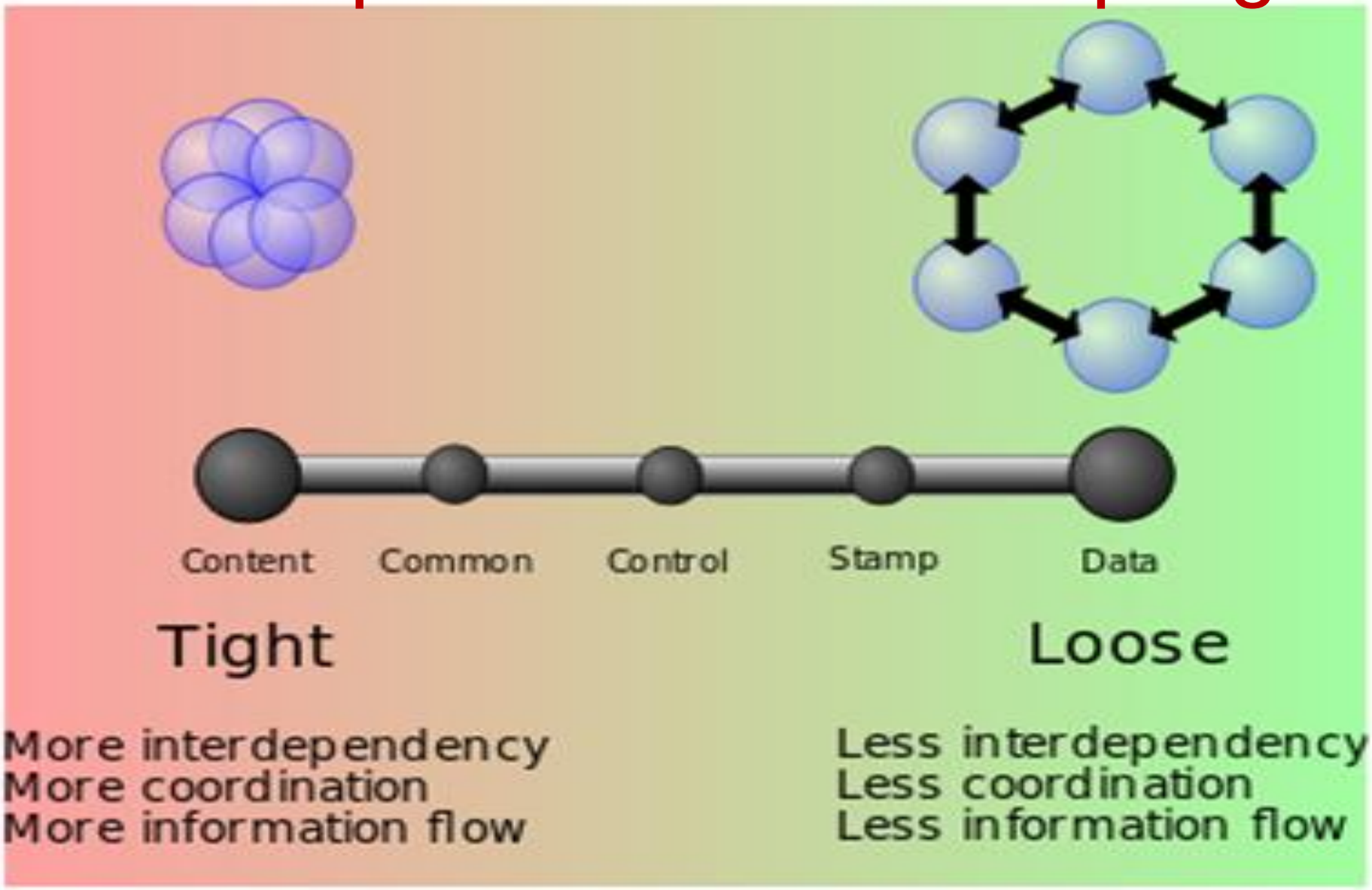


Sequential
Output of one is input to another



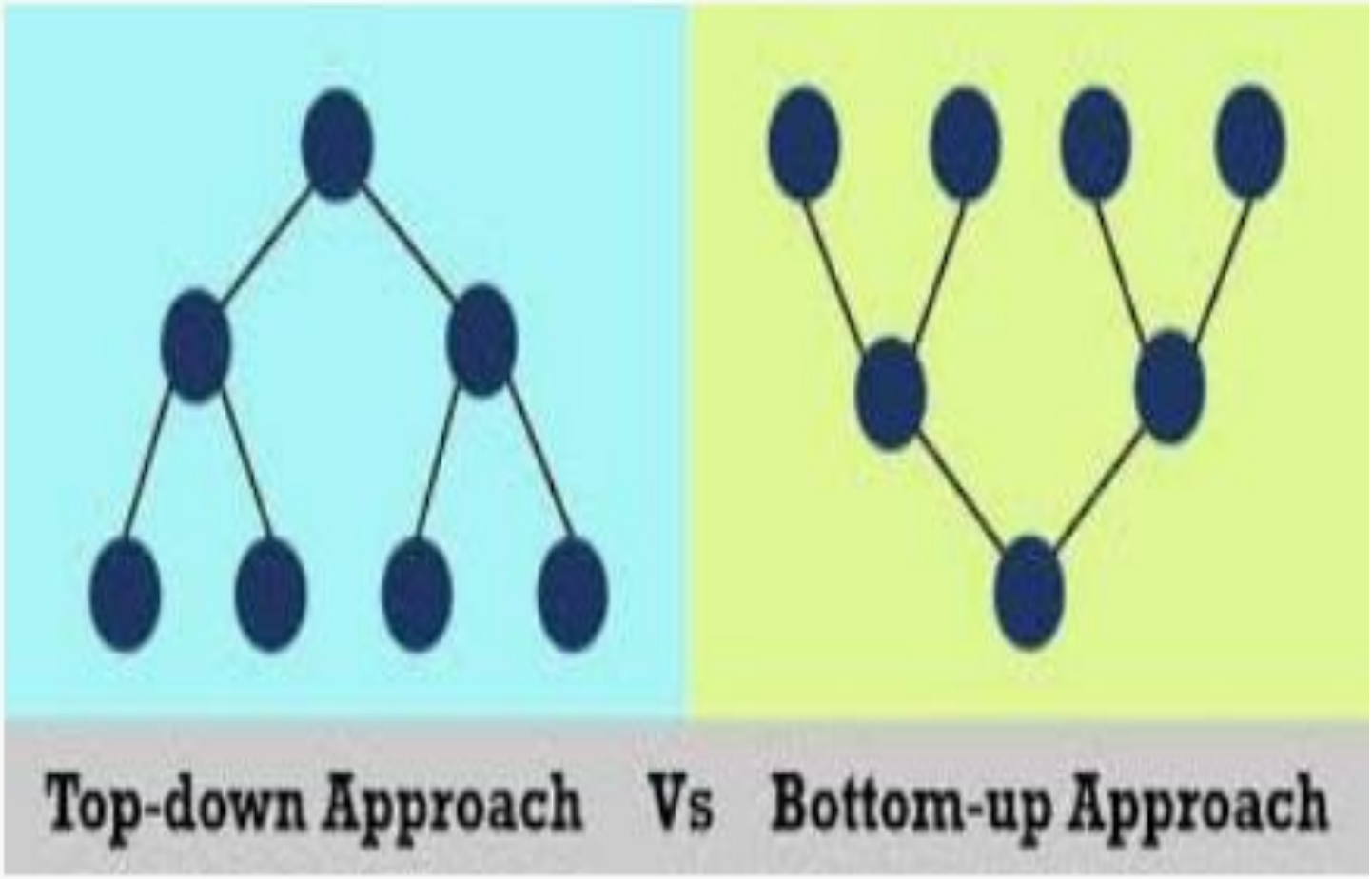
Functional
Sequential with complete, related functions

Conceptual Model of Coupling



Consequences of Coupling

- High coupling
  - Components are difficult to understand in isolation
  - Changes in component ripple to others
  - Components are difficult to reuse
    - Need to include all coupled components
    - Difficult to understand
- Low coupling
  - May incur performance cost
  - Generally faster to build systems with low coupling


Top-down Approach Vs Bottom-up Approach