Reviews

- **Project Description**
- **Requirements Overview**
- **Architectural Analysis**
- **Use Case Analysis**

Requirements Overview

- 与客户和其他涉众在系统的工作内容方面达成并保持一致
- 让开发人员对系统的需求有更好的理解
- 划分系统的边界
- 为计划迭代的技术内容提供基础
- 为估算开发系统所需成本和时间提供基础
- 定义系统的用户界面
- Use Case, Glossary, Use Case Specification, Supplement Specification.

Architectural Analysis

- **Architecture Document,**
- **Design Pattern,**
- **Key Abstraction,**
- **UseCase Realization**
- ***是否记得步骤？***

Reviews(for usecase analysis)

- **Are the classes reasonable?**
- **Does the name of each class clearly reflect the role it plays?**
- **Does the class represent a single well-defined abstraction?**
- **Are all attributes and responsibilities functionally coupled?**
- **Does the class offer the required behavior?**
- **Are all specific requirements on the class addressed?**

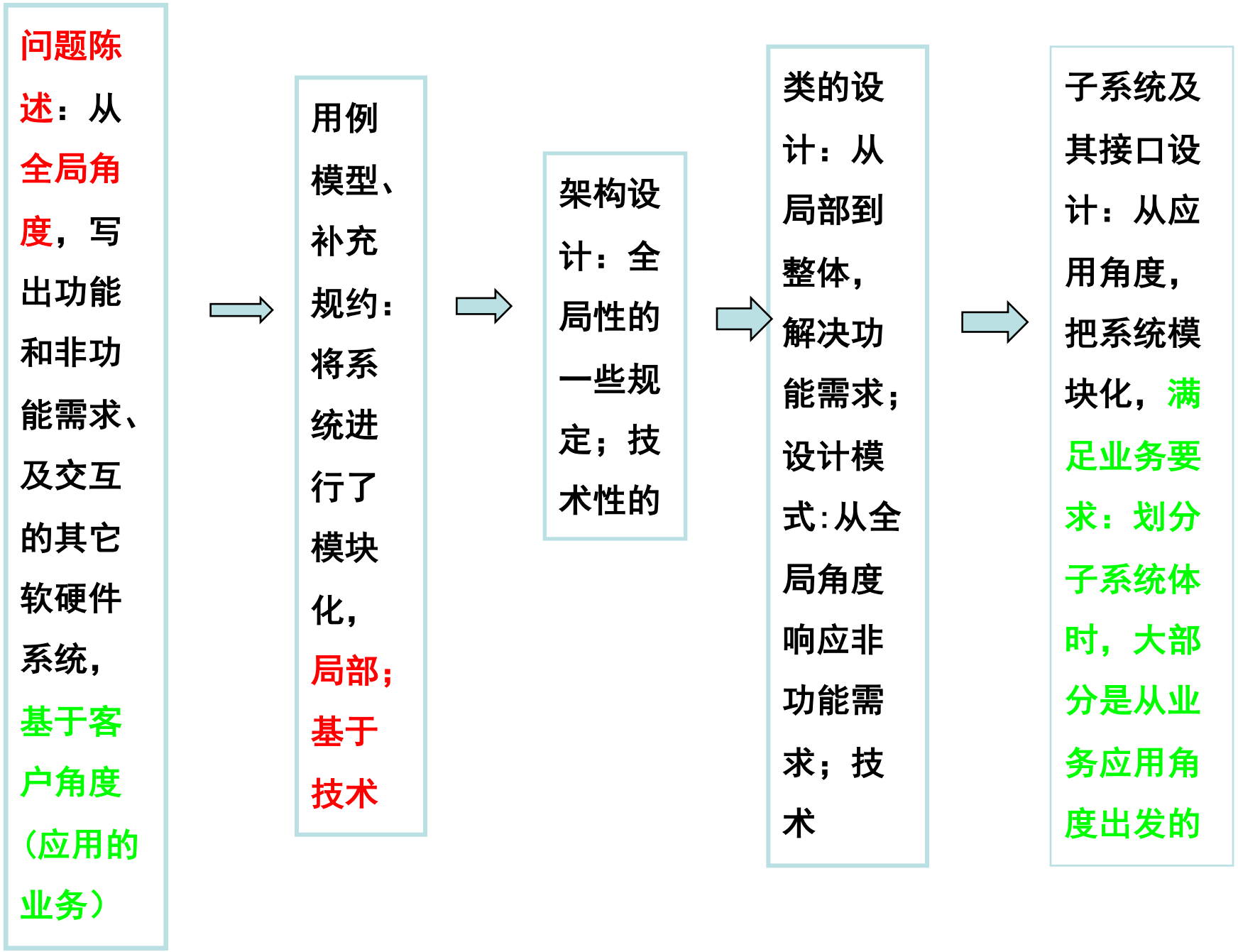
Continuous---

Reviews(for usecase analysis)

- **Have all the main and/or sub-flows been handled, including exceptional cases?**
- **Have all the required objects been found?**
- **Has all behavior been unambiguously distributed to the participating objects?**
- **Has behavior been distributed to the right objects?**
- **Where there are several Interaction diagrams, are their relationships clear and consistent?**

Use Case Analysis

- 用例分析总述
- 补充用例描述
- 查找分析类
- 将用例行为分配给分析类
- 描述分析类
- 描述分析机制
- 合并分析类


问题陈述：从全局角度，写出功能和非功能需求、及交互的其它软硬件系统，基于客户角度（应用的业务）
用例模型、补充规约：将系统进行了模块化，局部；基于技术
架构设计：全局性的一些规定；技术性的
类的设计：从局部到整体，解决功能需求；设计模式:从全局角度响应非功能需求；技术
子系统及其接口设计：从应用角度，把系统模块化，满足业务要求：划分子系统体时，大部分是从业务应用角度出发的

System Analysis & Design

chapter 9 Identify Design Elements

子系统及其接口设计

Yang YI, Ph.D

Computer Science Department, SYSU

issyy@mail.sysu.edu.cn

Tel:86-13902295111

2020/10/26

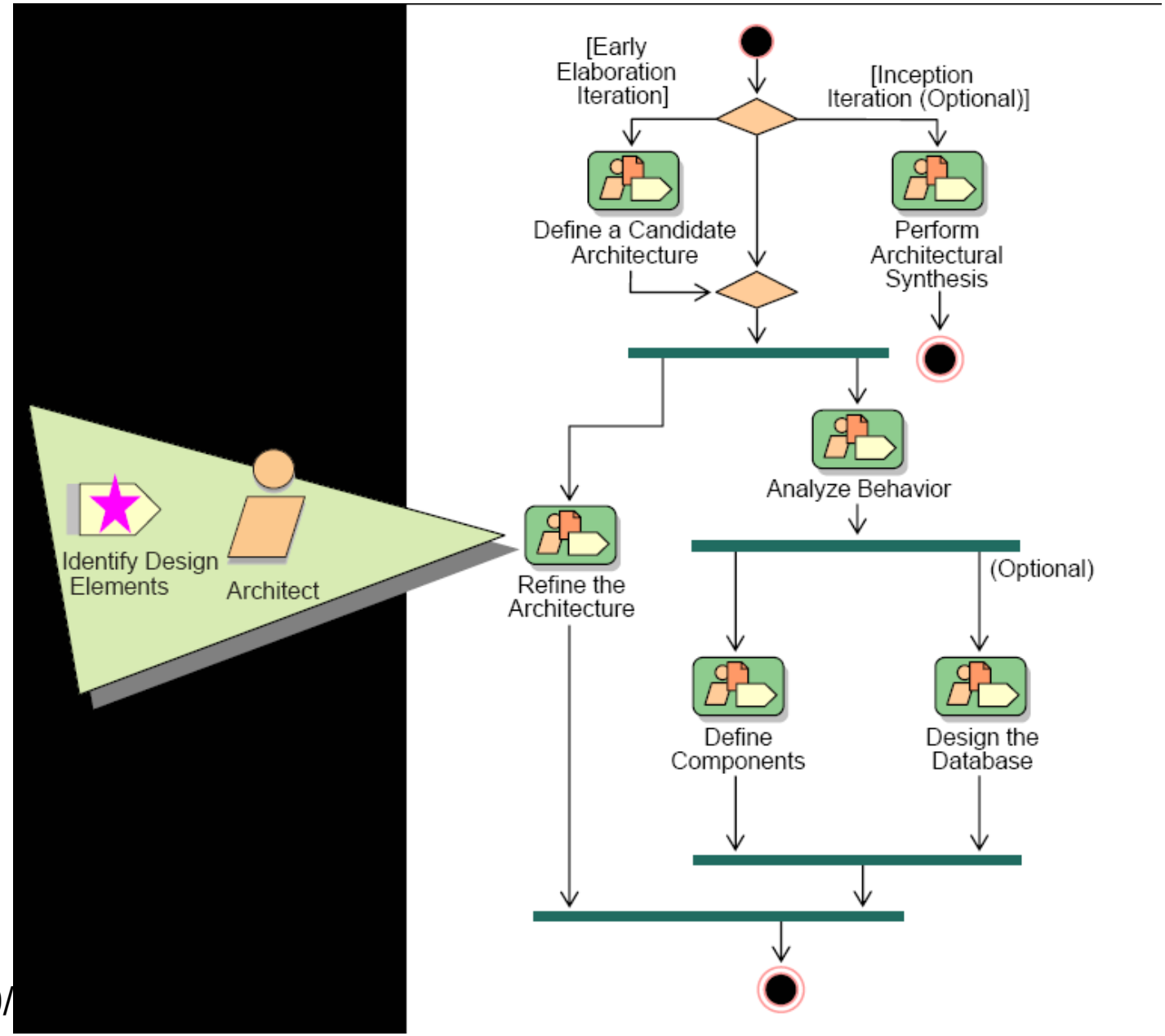
Chapter 9 Identify Design Elements

Agenda

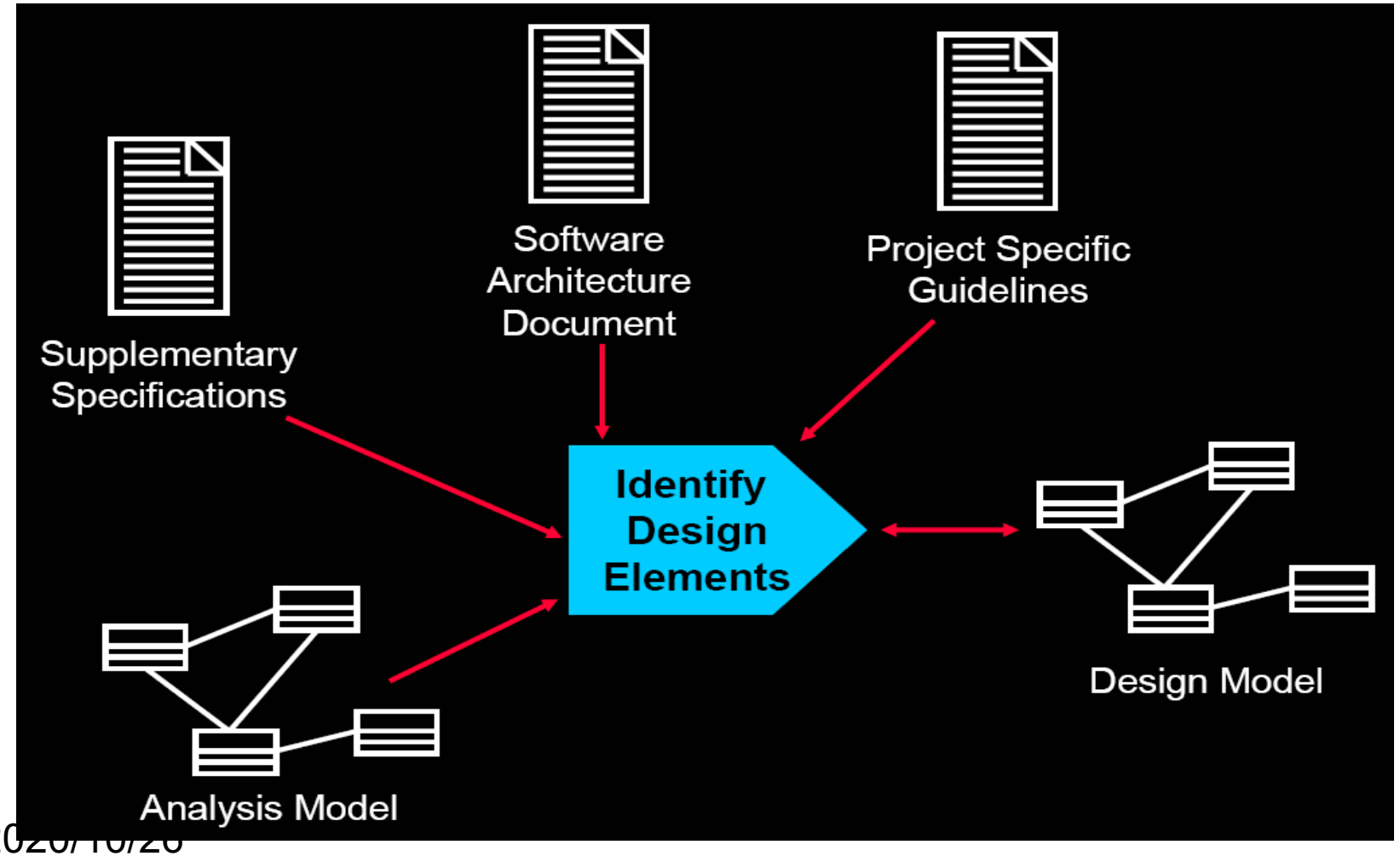
- **Objectives**
- **Context in identify design elements**
- **Identify design elements steps**
- **Exercises**

Objectives

- **Define the purpose of Identify Design Elements and demonstrate where in the lifecycle it is performed**
- **Analyze interactions of analysis classes and identify Design Model elements**
  - **Design classes**
  - **Subsystem**
  - **Subsystem interfaces**


[Early Elaboration Iteration]
[Inception Iteration (Optional)]
Define a Candidate Architecture
Perform Architectural Synthesis
Analyze Behavior
Identify Design Elements
Architect
Refine the Architecture
(Optional)
Define Components
Design the Database

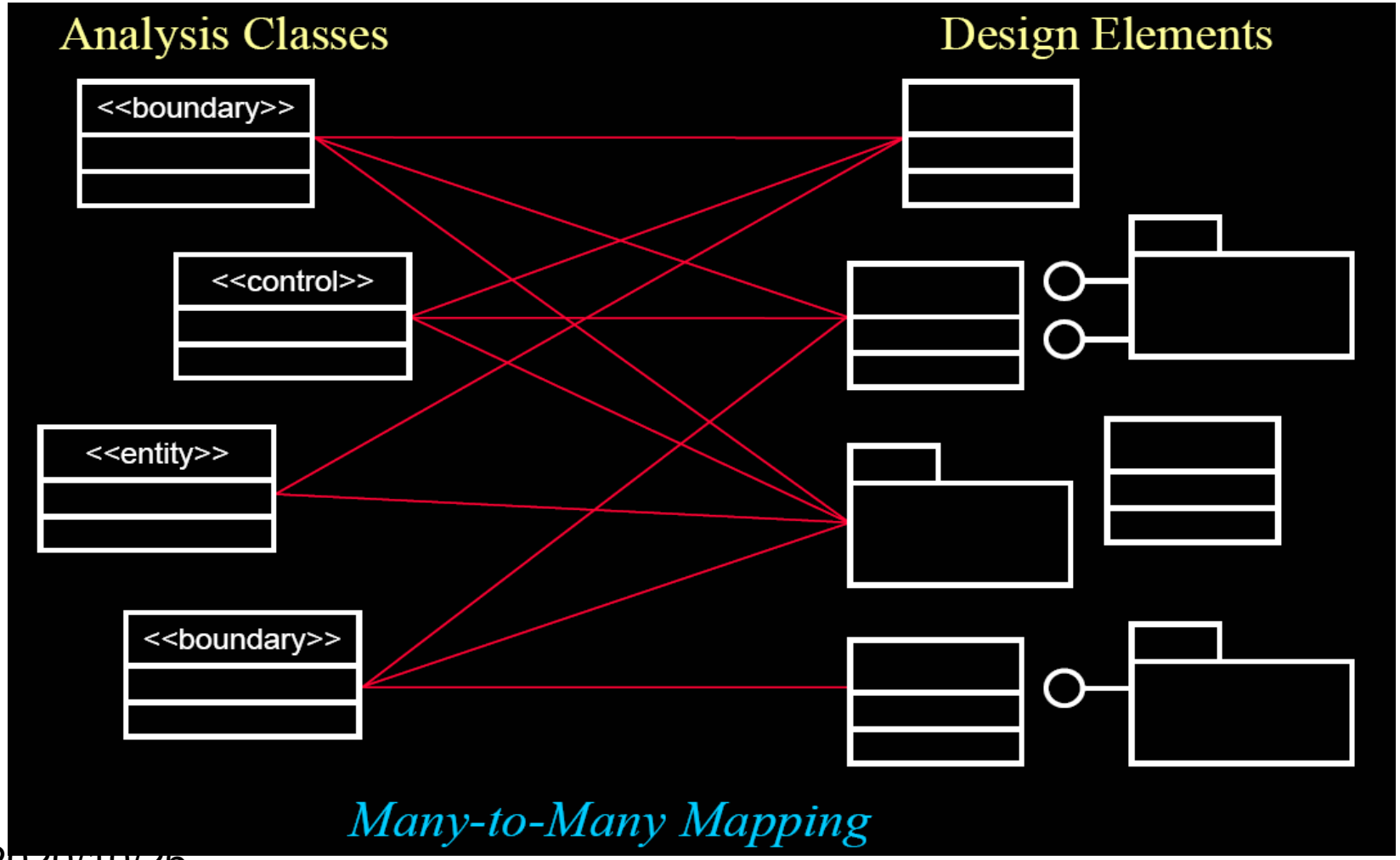
Identify Design Elements Overview



Identify Design Elements Steps

- **Identify classes and subsystems**
- **Identify subsystem interfaces**
- **Identify reuse opportunities**
- **Update the organization of the Design Model**
- **Checkpoints**

From Analysis Classes to Design Elements



Identifying Design Classes

- **An analysis class maps directly to a design class if:**
  - **–It is a simple class**
  - **– It represents a single logical abstraction**
- **More complex analysis classes may**
  - ☐ **– Split into multiple classes**
  - ☐ **– Become a package**
  - ☐ **– Become a subsystem (discussed later)**
  - ☐ **– Any combination …**

Review: Class and Package

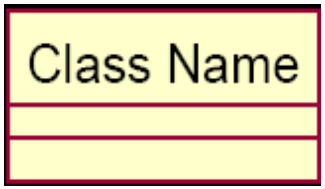
- **What is a class?**

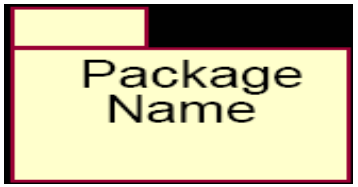
  –A description of a set of objects that share the same responsibilities, relationships, operations, attributes, and semantics

- **What is a package?**

  – A general purpose mechanism for organizing elements into groups

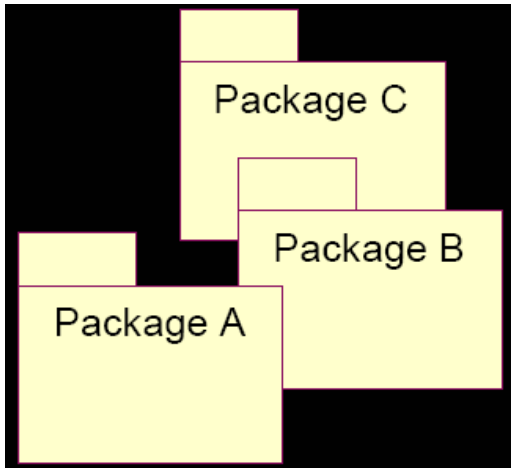
  – A model element which can contain other model elements



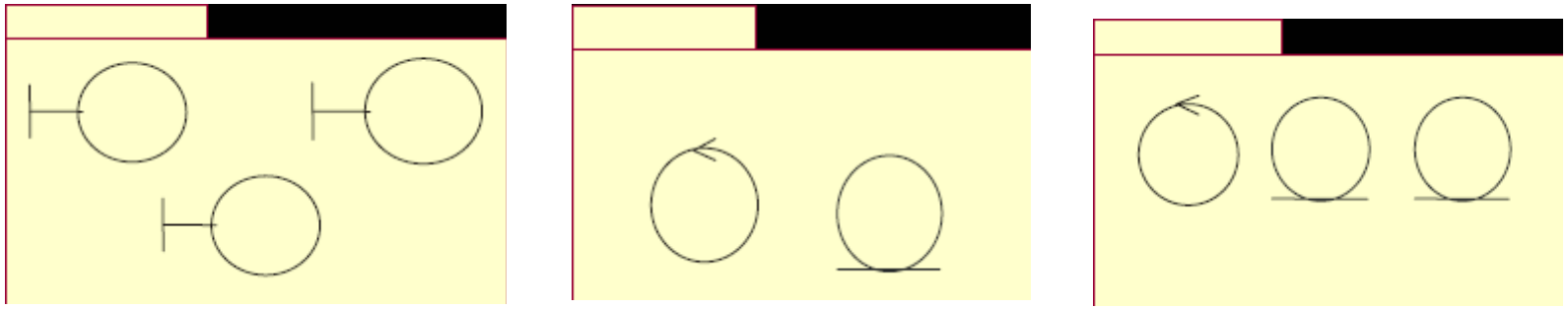


Group Design Classes in Packages

- **You can base your packaging criteria on a number of different factors, including:**
  - **Configuration units**
  - **Allocation of resources among development teams**
  - **Reflect the user types**
  - **Represent the existing products and services the system uses**



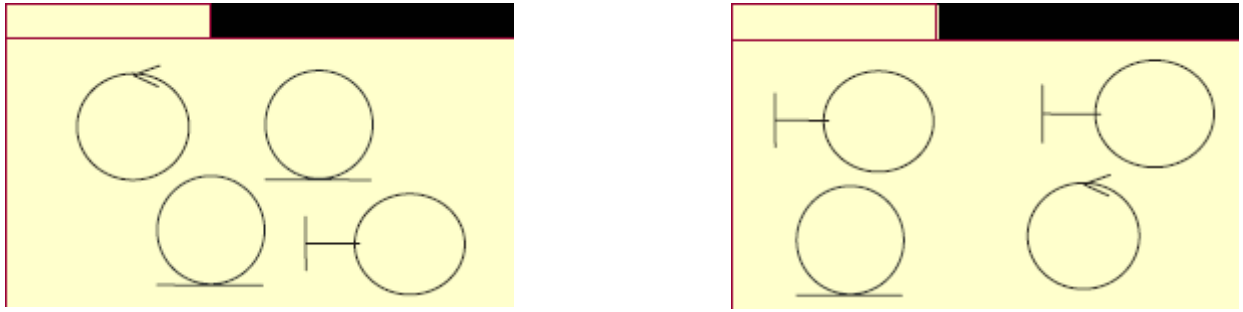
Packaging Tips: Boundary Classes



Boundary classes placed in separate packages

Continued…

Packaging Tips: Boundary Classes



Boundary classes packaged with functionally related classes

Packaging Tips: Functionally Related Classes

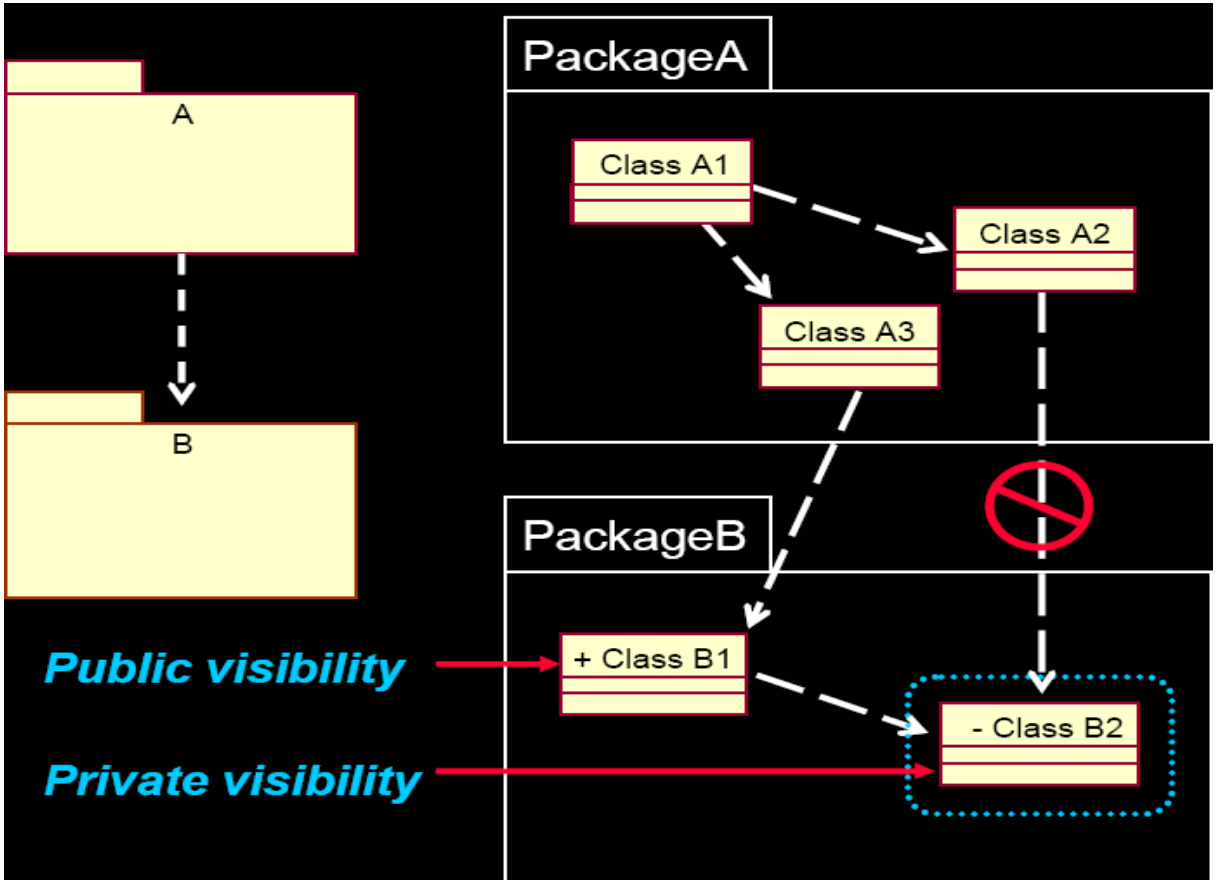
- **Criteria for determining if classes are functionally related:**
  - **Changes in one class' behavior and/or structure necessitate changes in another class**
  - **Removal of one class impacts the other class**
  - **Two objects interact with a large number of messages or have a complex intercommunication**
  - **A boundary class can be functionally related to a particular entity class if the function of the boundary class is to present the entity class**
  - **Two classes interact with, or are affected by changes in the same actor**

Continued…

Packaging Tips: Functionally Related Classes (cont.)

- **Criteria for determining if classes are functionally related (continued):**
  - **Two classes have relationships between each other**
  - **One class creates instances of another class**
- **Criteria for determining when two classes should *not* be placed in the same package:**
  - **Two classes that are related to different actors should not be placed in the same package**
  - **An optional and a mandatory class should not be placed in the same package**

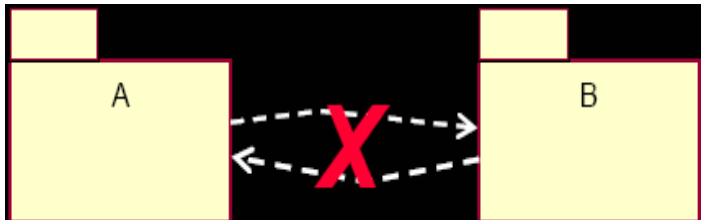
Package Dependencies: Package Element Visibility



- ***Only public classes can be referenced outside of the owning package***

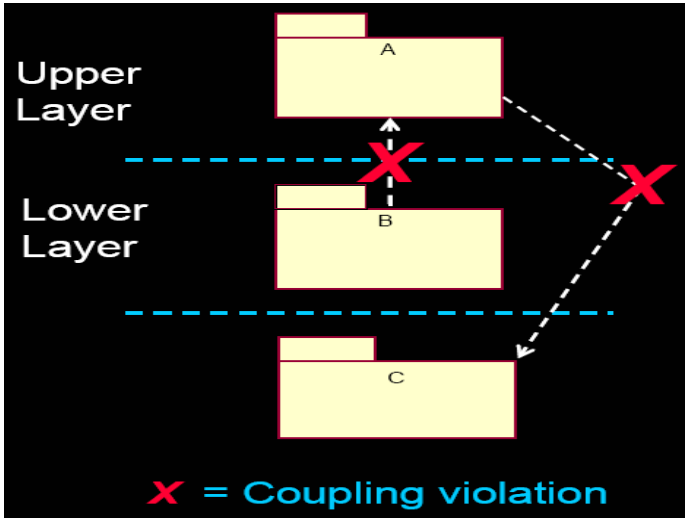
Package Coupling: Tips

- **Packages should not be cross-coupled**



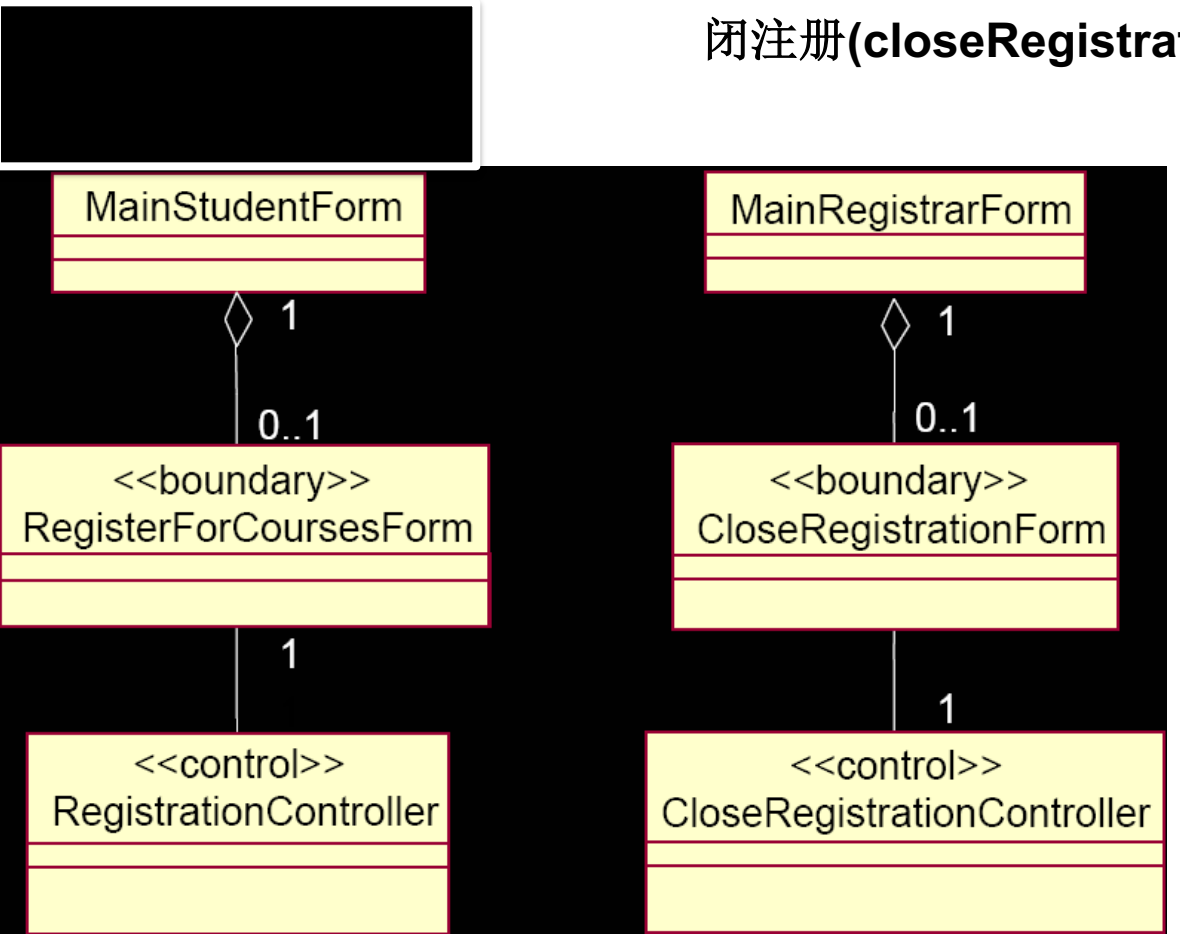
- Packages in lower layers should not be dependent upon packages In upper layers

- In general, dependencies should not skip layers



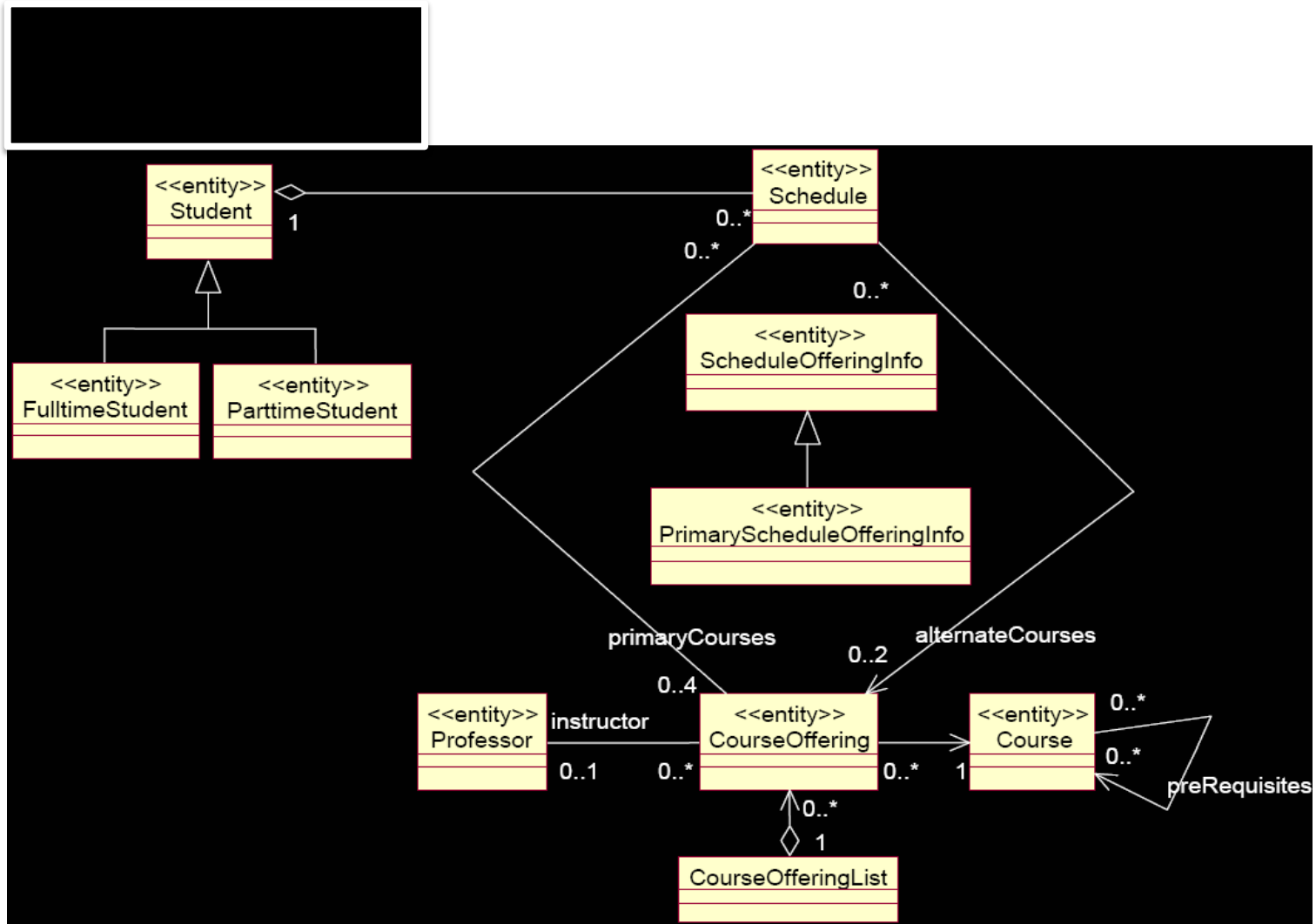
Example: Registration Package

- 此图基于本课程一直讲的案例:注册课程
- 这是一个**package**,用来后面形成子**Subsystem**
- 这个子系统把注册课程（**registerForCouses**)和关闭注册**(closeRegistration)**放到一个**package**里

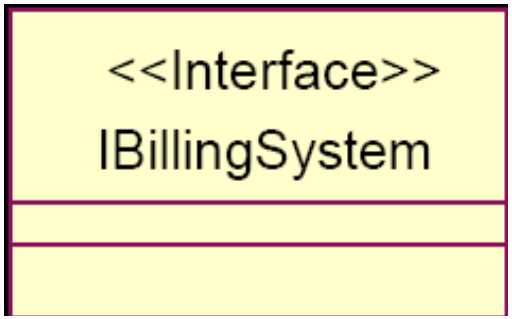


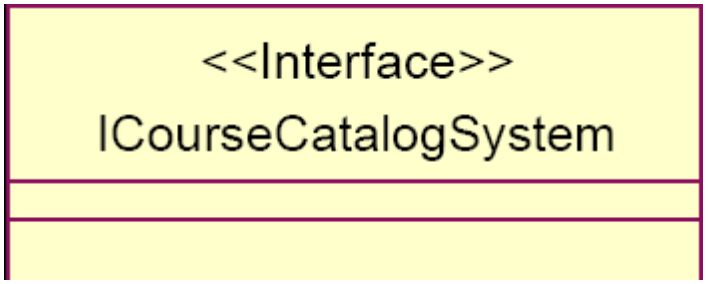
- 此包内含界面层和控制层的若干，没有**Model(**数据）
- 具体方案完全由架构设计师来定
- 在界面设计方案中，采用了**Builder Pattern**

Example: University Artifacts Package



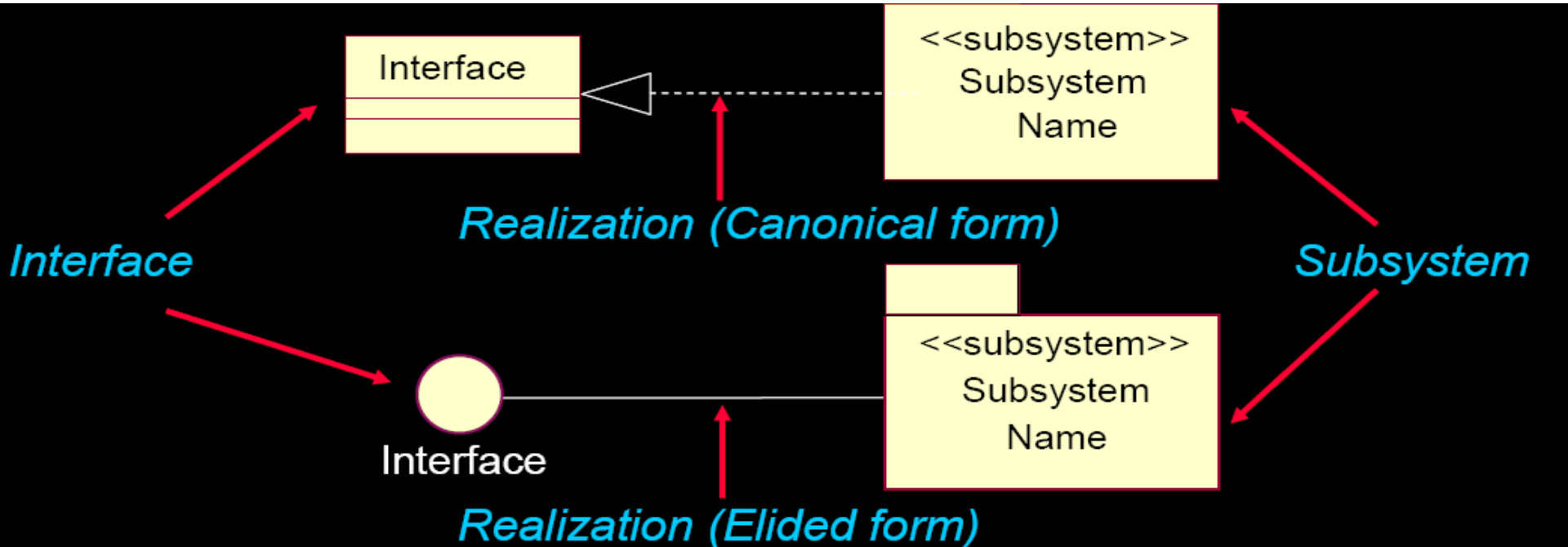
Example: External System Interfaces Package





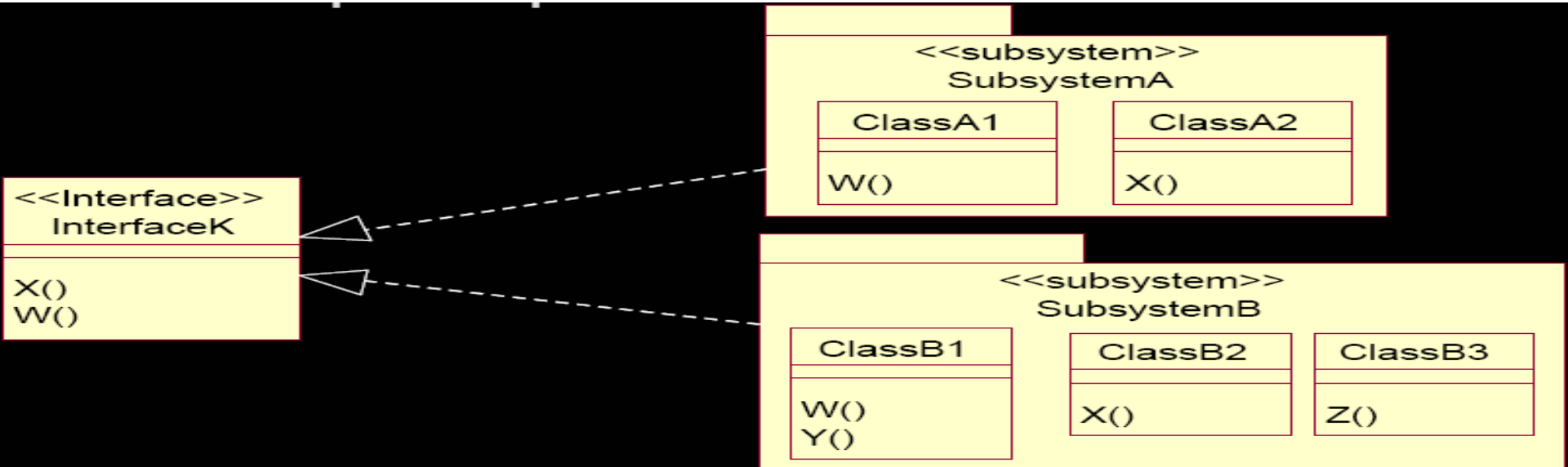
Review: Subsystems and Interfaces

- Are a "cross between" a package (can contain other model elements) and a class (has behavior)
- Realizes one or more interfaces that define its behavior



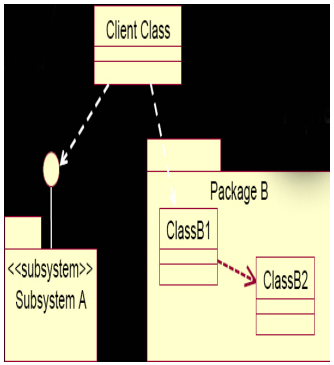
Subsystems and Interfaces (cont.)

- **Subsystems :**
  - **Completely encapsulate behavior**
  - **Represent an independent capability with clear interfaces (potential for reuse)**
  - **Model multiple implementation variants**



Packages versus Subsystems

- **Subsystems**
  - **Provide behavior**
  - **Completely encapsulate their contents**
  - **Are easily replaced**
- **Packages**
  - **Don’t provide behavior**
  - **Don’t completely encapsulate their contents**
  - **May not be easily replaced**



Subsystem Usage

- **Subsystems can be used to partition the system into parts that can be independently:**
  - **ordered, configured, or delivered**
  - **developed, as long as the interfaces remain unchanged**
  - **deployed across a set of distributed computational nodes**
  - **changed without breaking other parts of the systems**
- **Subsystems can also be used to:**
  - **partition the system into units which can provide restricted security over key resources**
  - **represent existing products or external systems in the design (e.g. components)**

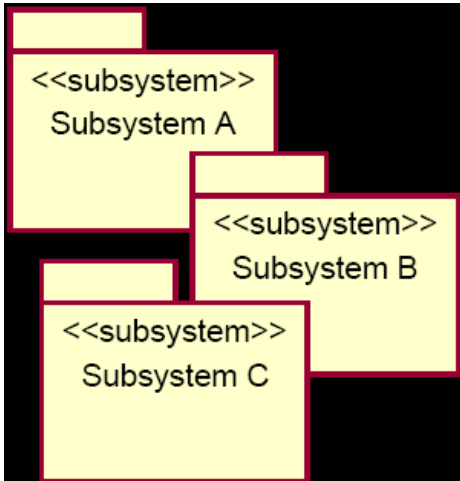
Subsystems raise the level of abstraction

Identifying Subsystems Hints

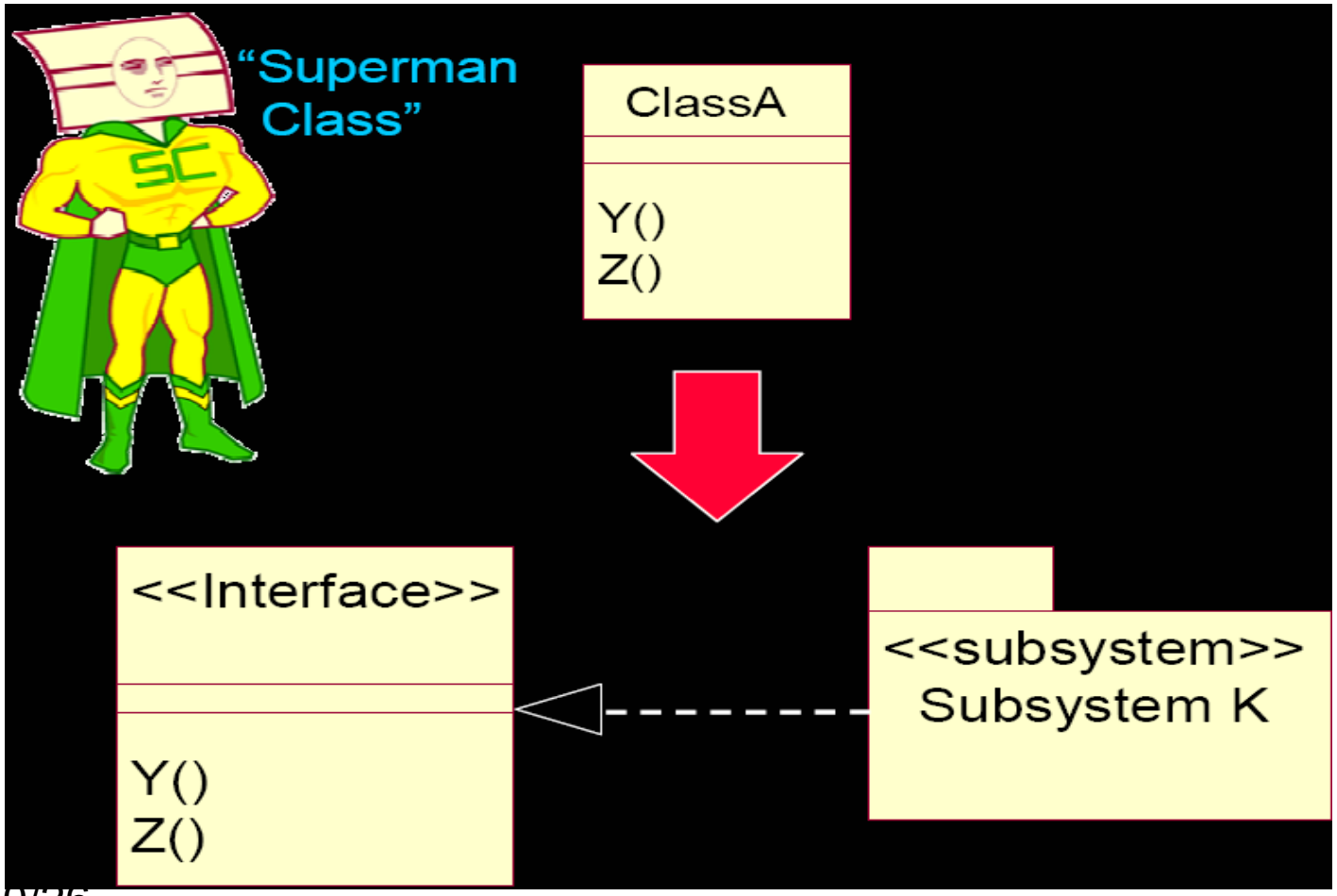
- **Look at object collaborations.**
- **Look for option.**
- **Look to the user interface of the system.**
- **Look to the actors.**
- **Look for coupling and cohesion between classes.**
- **Look at substitution.**
- **Look at distribution.**

Candidate Subsystems

- **Analysis classes which may evolve into subsystems:**
  - **Classes providing complex services and/or utilities**
  - **Boundary classes (user interfaces and external system interfaces)**
- **Existing products or external systems in the design (e.g .,components):**
  - **Communication software**
  - **Database access support**
  - **Types and data structures**
  - **Common utilities**
  - **Application-specific products**



Identifying Subsystems



Identify Design Elements Steps

- **Identify classes and subsystems**
- **Identify subsystem interfaces**
- **Identify reuse opportunities**
- **Update the organization of the Design Model**
- **Checkpoints**

Identifying Interfaces

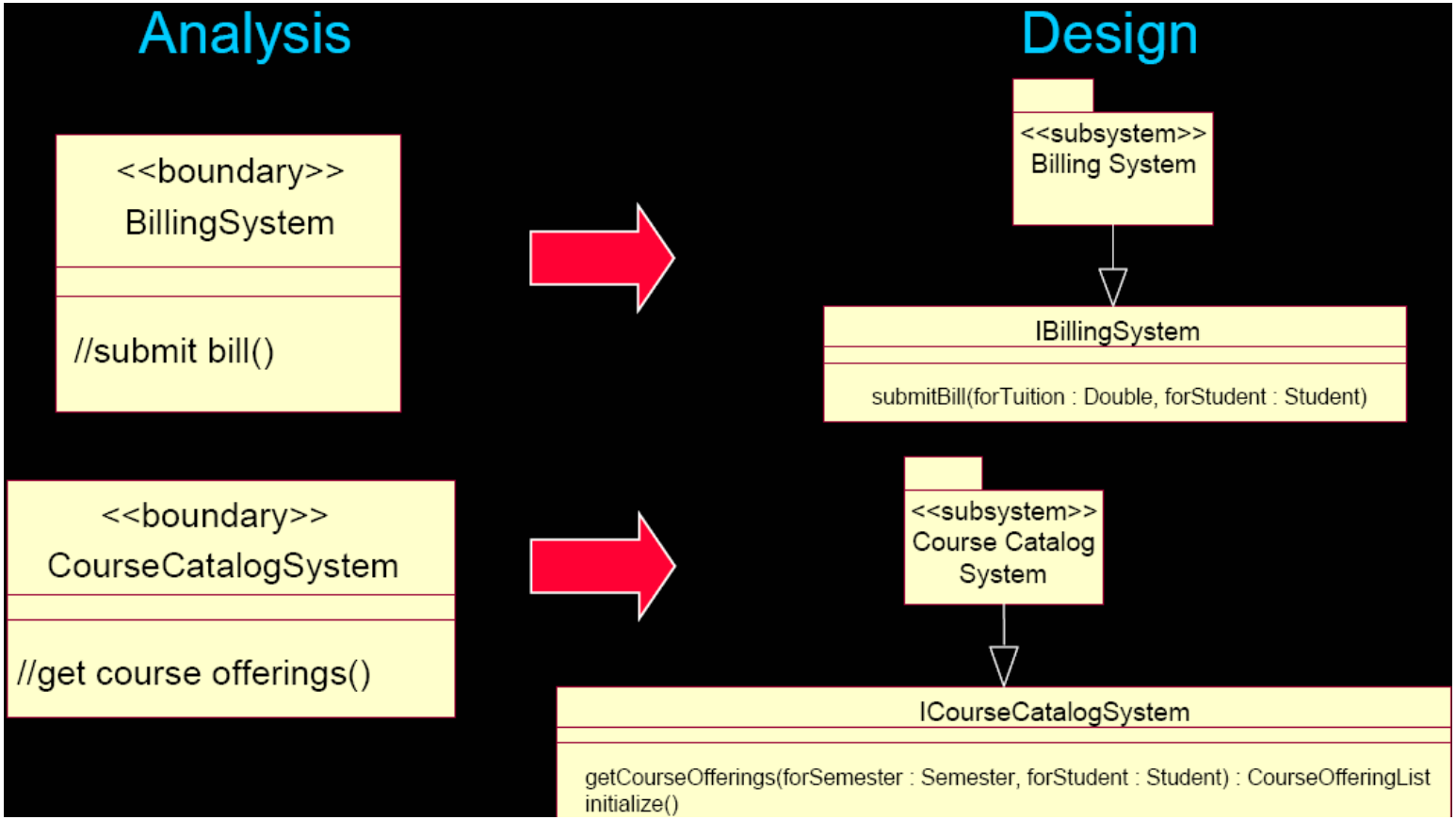
- **Purpose**
  - **To identify the interfaces of the subsystems based on their responsibilities**
- **Steps**
  1. **Identify a set of candidate interfaces for all subsystems, and define the behavior specified by the interfaces**
  2. **Look for similarities between interfaces，so as to combine those similar interfaces constructed**
  3. **Determine if the interface could represent the subsystems, if not, modify the interfaces**
  4. **Map the interfaces to subsystems**

Stable, well-defined interfaces are key to a stable, resilient architecture

Interface Guidelines

- **Interface name**
  - **Reflects role in system**
- **Interface description**
  - **Conveys responsibilities**
- **Operation definition**
  - **Name should reflect operation result**
  - **Describes what operation does, all parameters and result**
- **Interface documentation**
  - **Package supporting info: sequence and state diagrams, test plans, etc.**

Example: Design Subsystems and Interfaces

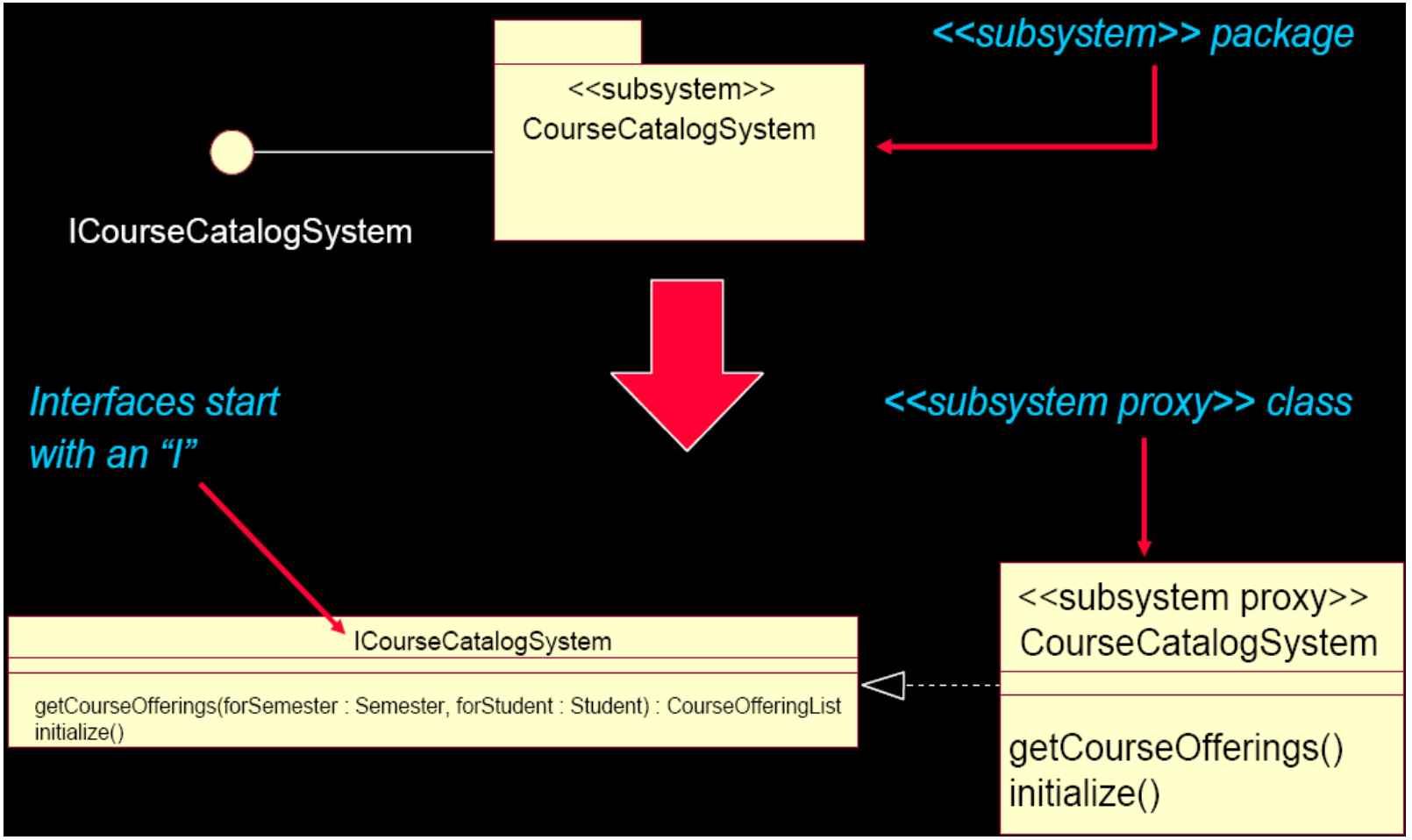


All other analysis classes map directly to design classes

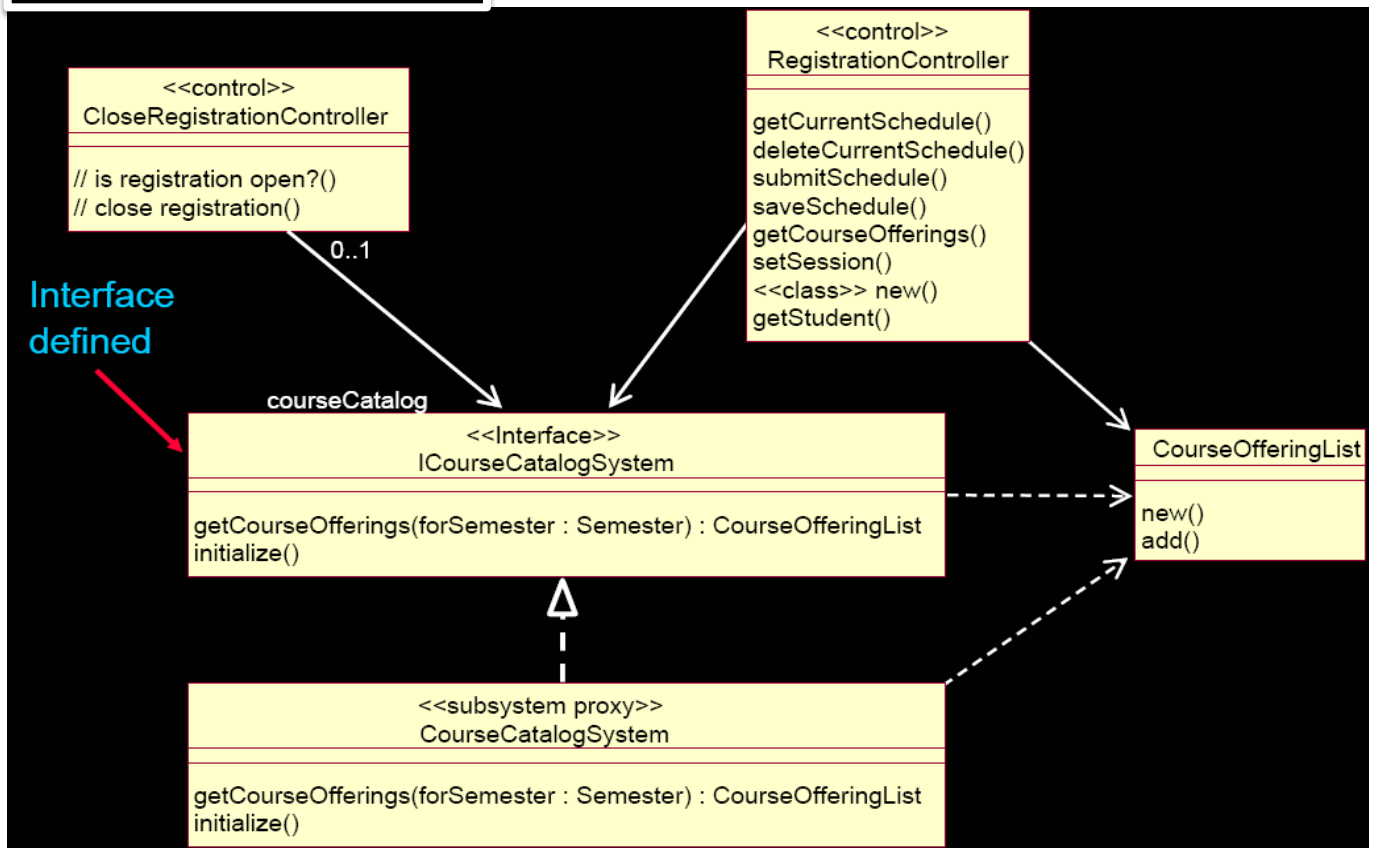
Example: Analysis-Class-To-Design-Element Map

Analysis Class	Design Element
CourseCatalogSystem	CourseCatalogSystem Subsystem
BillingSystem	BillingSystem Subsystem
All other analysis classes map directly to design classes	

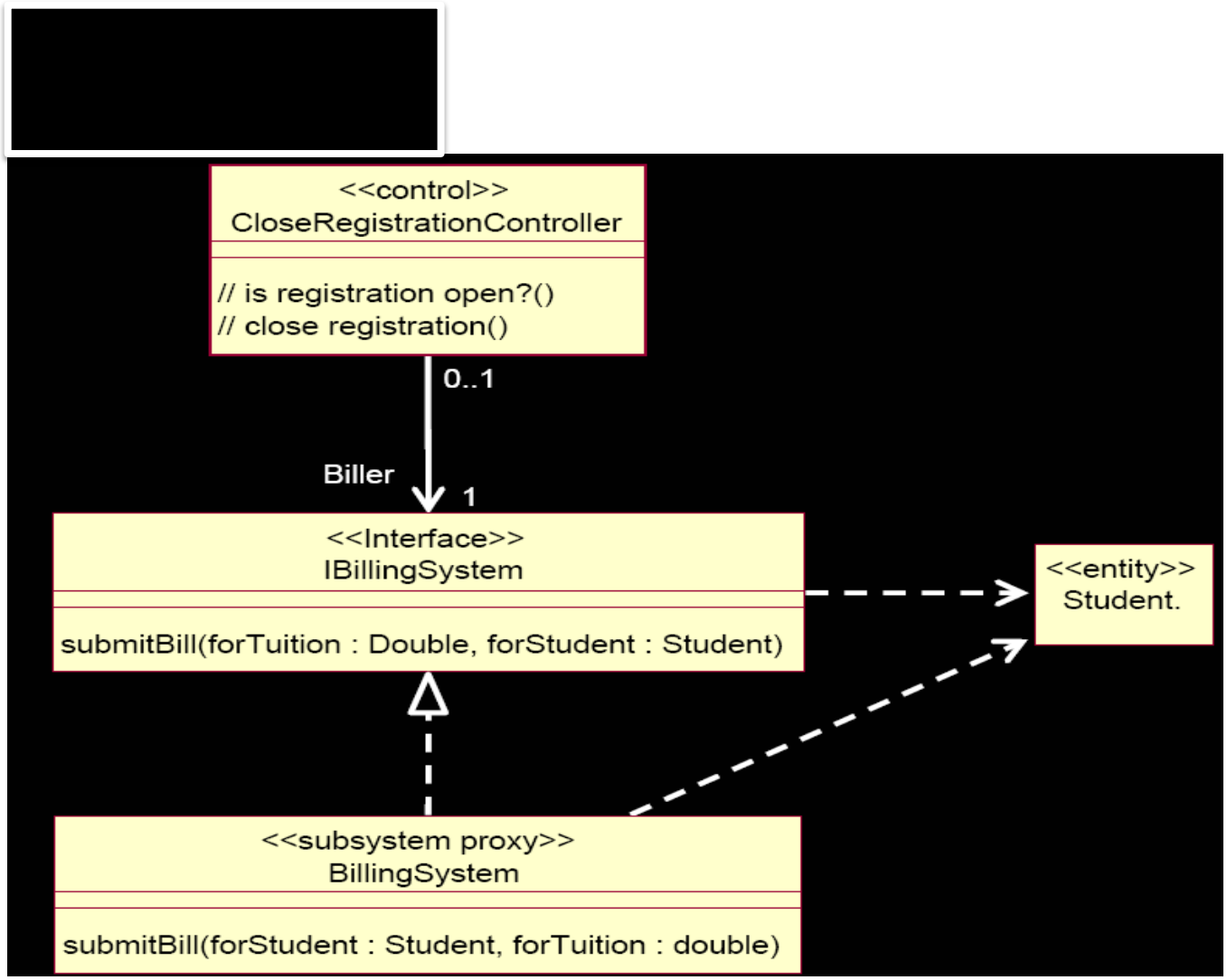
Modeling Convention: Subsystems and Interfaces



Example: Subsystem Context: Course Catalog System



Example: Subsystem Context: Billing System



Identify Design Elements Steps

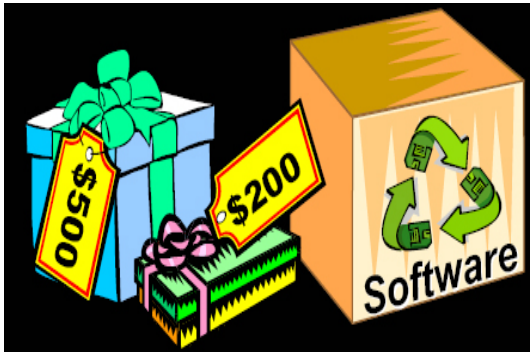
- **Identify classes and subsystems**
- **Identify subsystem interfaces**
- **Identify reuse opportunities**
- **Update the organization of the Design Model**
- **Checkpoints**

Identification of Reuse Opportunities

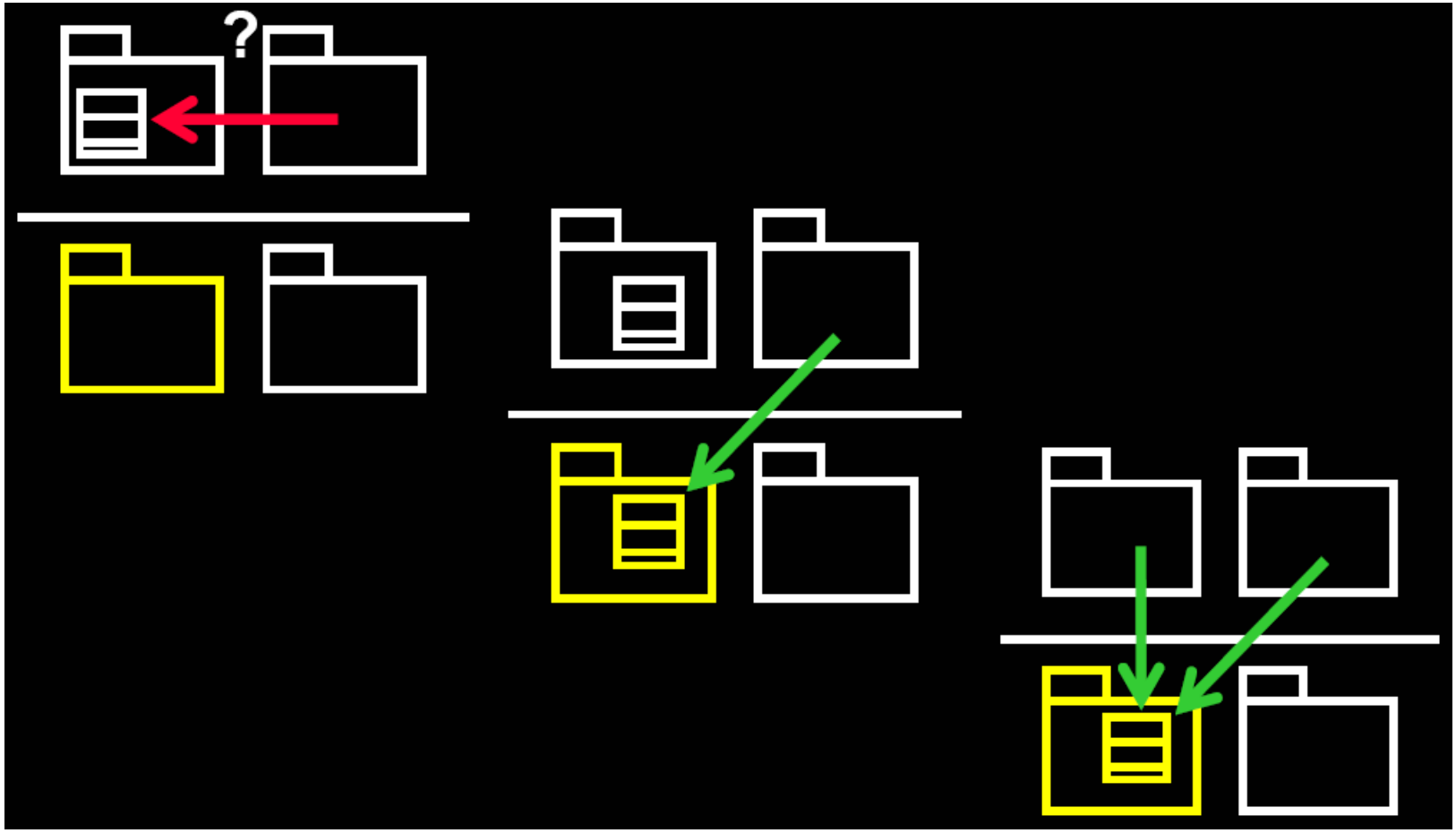
- **Purpose**
  - ✓ **To identify where existing subsystems and/or components can be reused based on their interfaces.**
- **Steps**
  - ✓ **Look for similar interfaces**
  - ✓ **Modify new interfaces to improve the fit**
  - ✓ **Replace candidate interfaces with existing interfaces**
  - ✓ **Map the candidate subsystem to existing components**

Possible Reuse Opportunities

- **Internal to the system being developed**
  - ✓ **Recognized commonality across packages and subsystems**
- **External to the system being developed**
  - ✓ **Commercially available components**
  - ✓ **Components from a previously developed application**
  - ✓ **Reverse engineered components**



Reuse Opportunities Internal to System

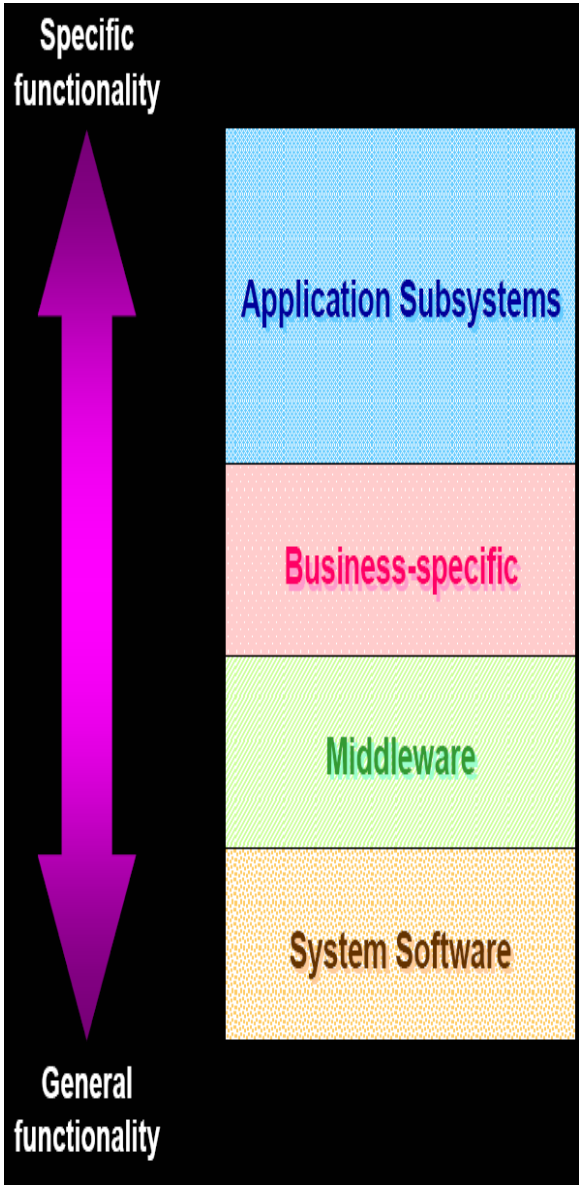


Identify Design Elements Steps

- **Identify classes and subsystems**
- **Identify subsystem interfaces**
- **Identify reuse opportunities**
- **Update the organization of the Design Model**
- **Checkpoints**

Review: Typical Layering Approach

- **Distinct application subsystems that make up an application – contains the value adding software developed by the organization.**
- **Business specific – contains a number of usable subsystems specific to the type of business.**
- **Middleware – offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.**
- **System software – contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers and so on.**

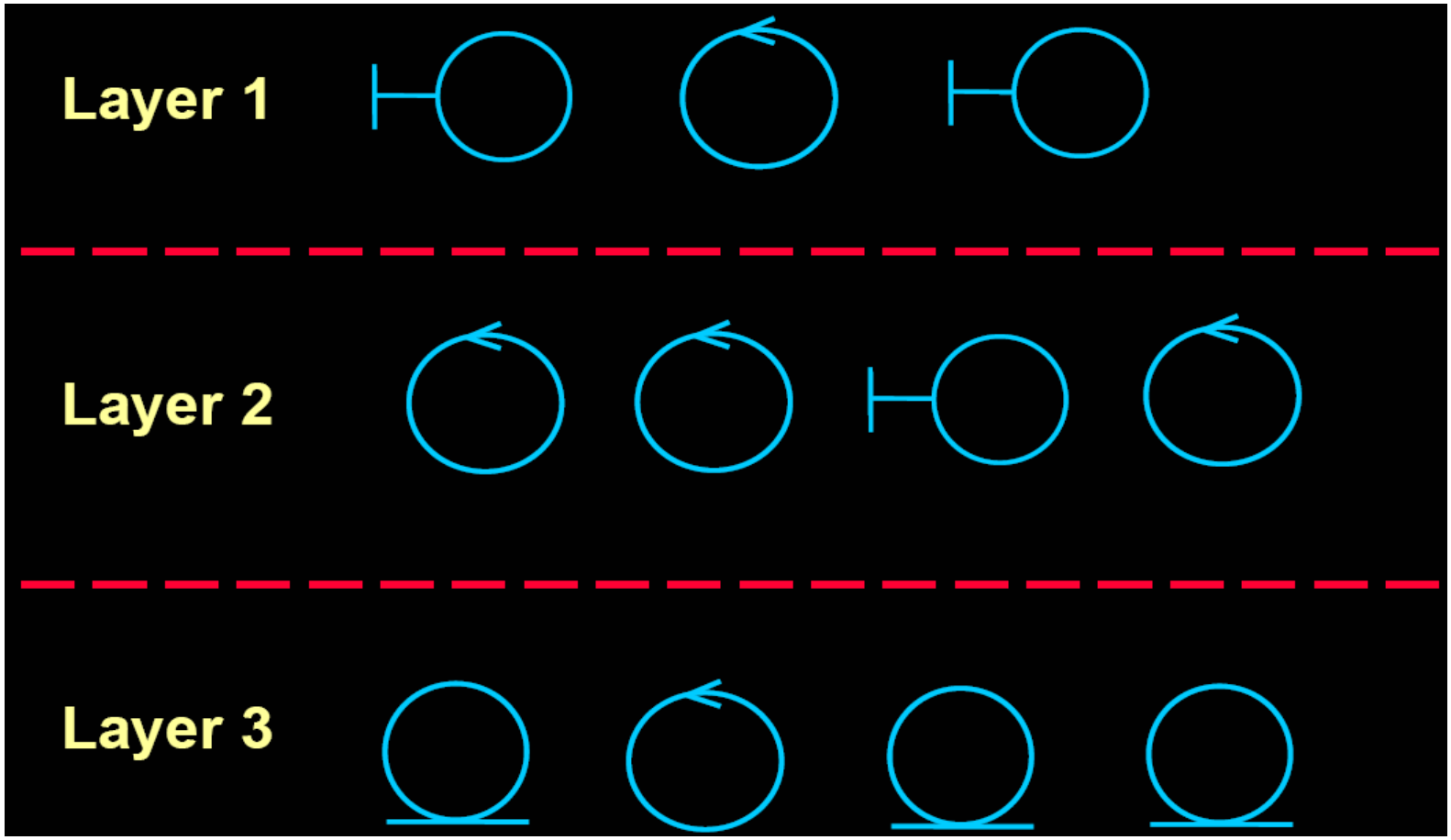


Layering Considerations

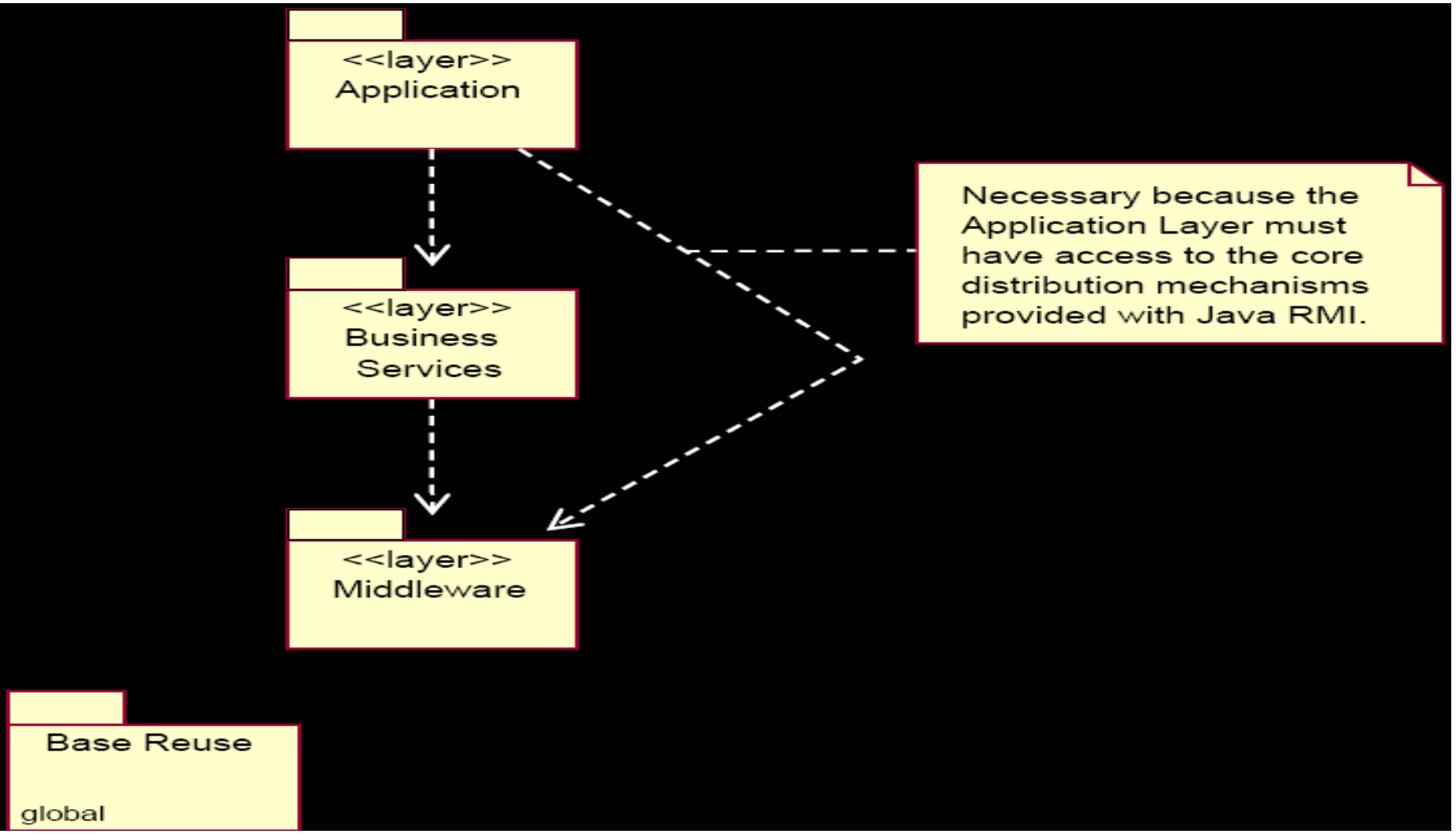
- **Visibility (*the state of being able to see or be seen*)**
  - ✓ **Dependencies only within current layer and below**
- **Volatility (*unpredictability*)**
  - ✓ **Upper layers affected by requirements changes**
  - ✓ **Lower layers affected by environment changes**
- **Generality**
  - ✓ **More abstract model elements in lower layers Number of layers**
  - ✓ **Small system: 3-4 layers**
  - ✓ **Complex system: 5-7 layers**

Goal is to reduce coupling and to ease maintenance effort.

Design Elements and the Architecture



Example: Architectural Layers

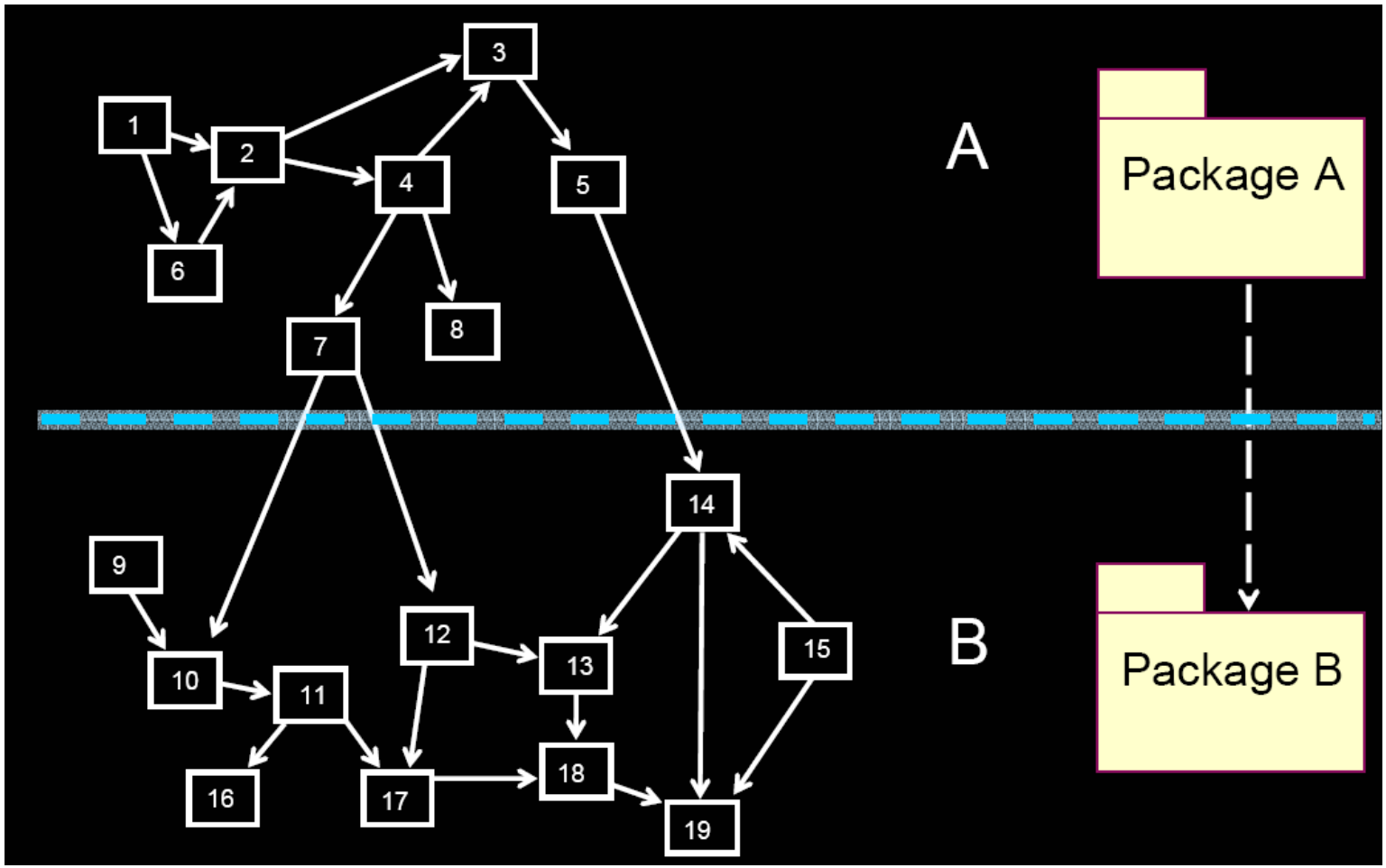


Partitioning Considerations

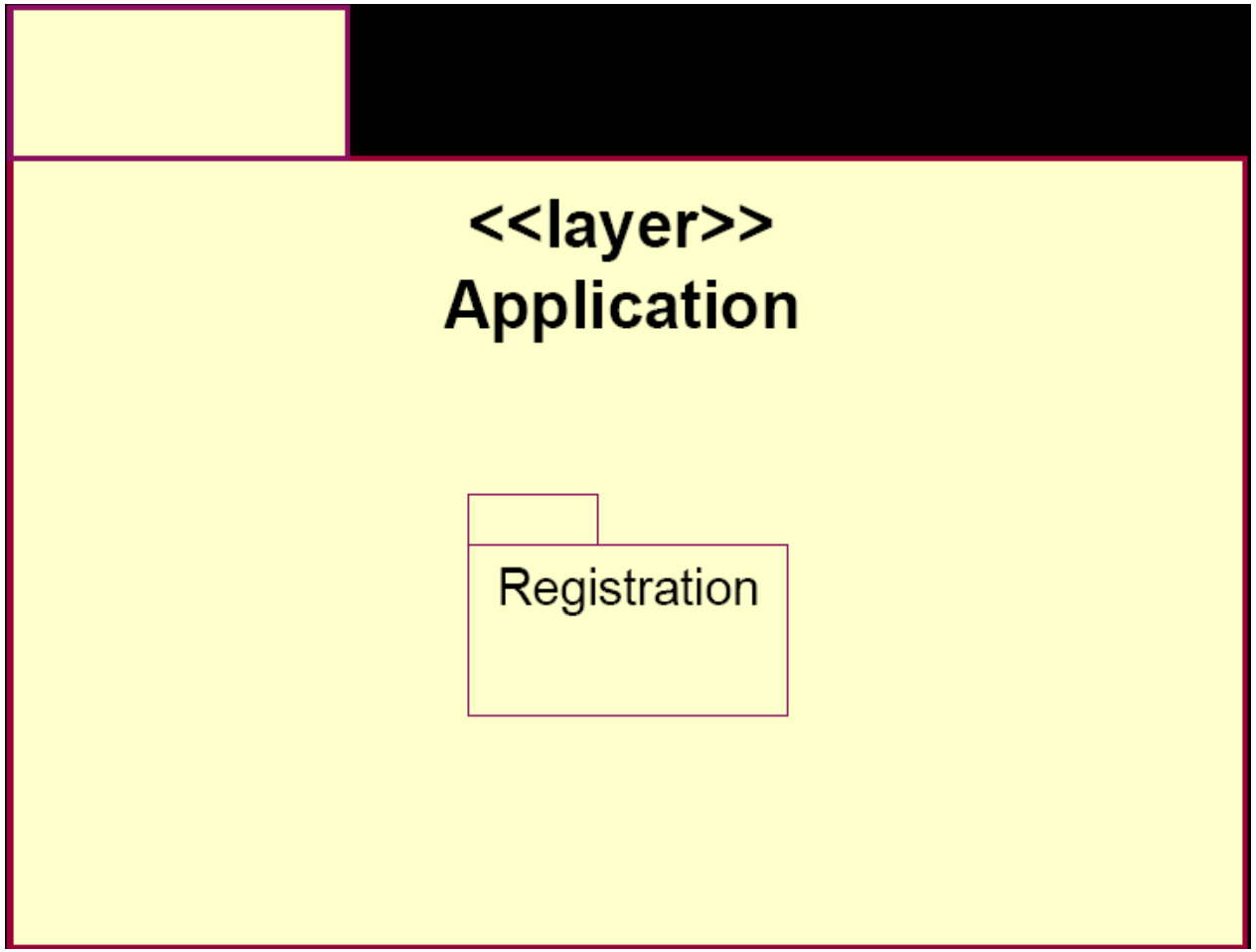
- **Coupling and cohesion**
- **User organization**
- **Competency and/or skill areas (*能力和/或技能*)**
- **System distribution**
- **Secrecy**
- **Variability (*变化性*)**

Try to avoid cyclic dependencies.

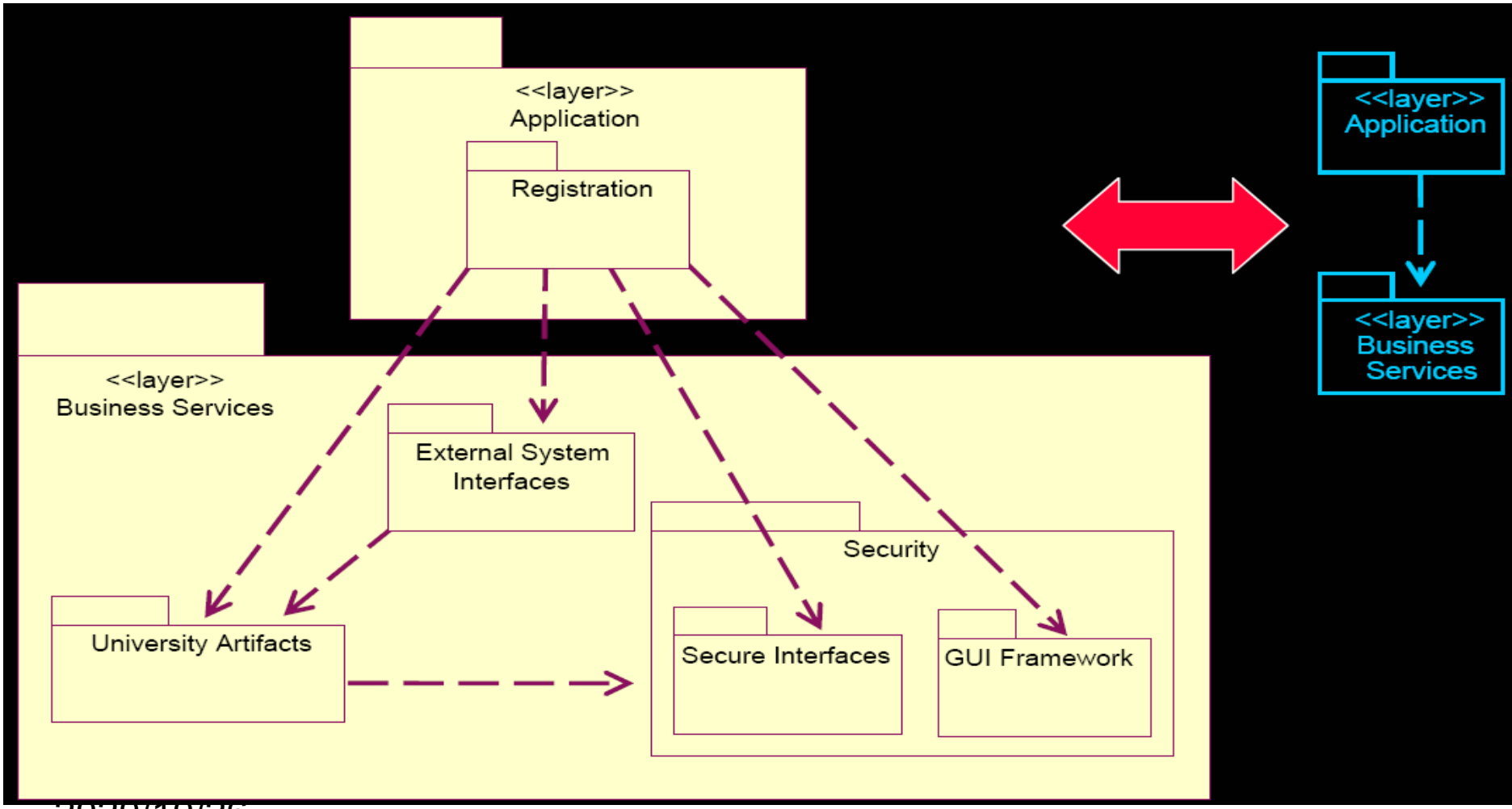
Example: Partitioning



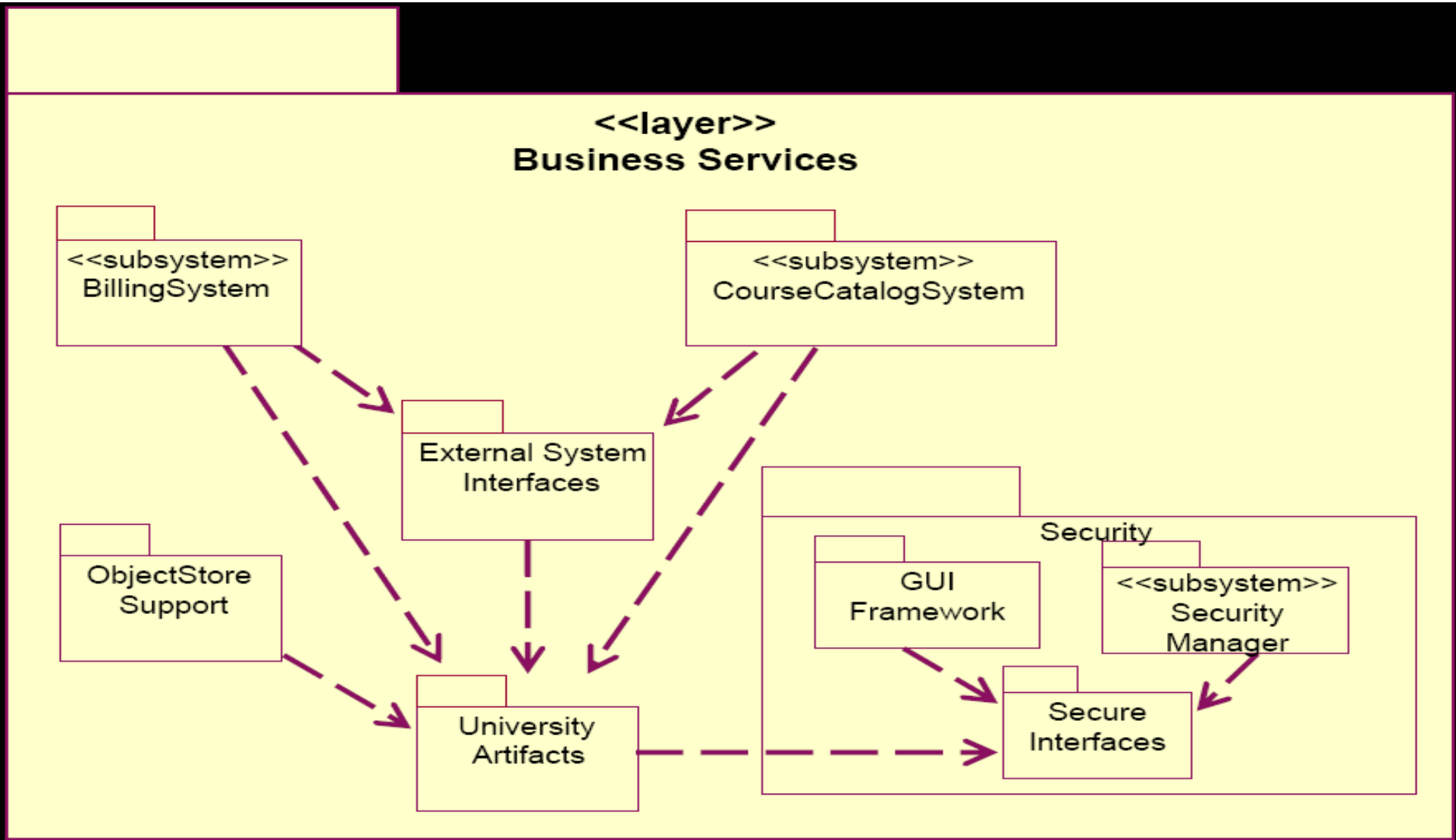
Example: Application Layer



Example: Application Layer Context

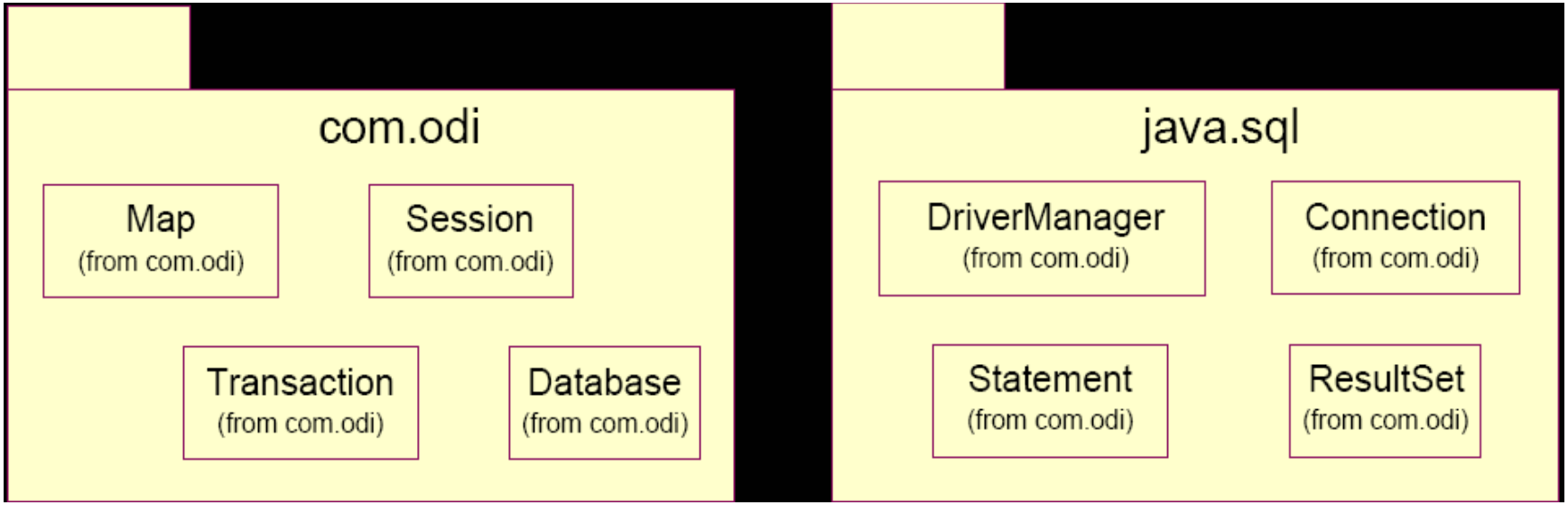


Example: Business Services Layer



Example: Business Services Layer Context

Example: Middleware Layer



Identify Design Elements Steps

- **Identify classes and subsystems**
- **Identify subsystem interfaces**
- **Identify reuse opportunities**
- **Update the organization of the Design Model**
- **Checkpoints**

Checkpoints

- **General**
  - **Does it provide a comprehensive picture of the services of different packages?**
  - **Can you find similar structural solutions that can be used more widely in the problem domain?**
- **Layers**
  - **Are there more than seven layers?**
- **Subsystems**
  - **Is subsystem partitioning done in a logically consistent way across the entire model?**

continued

Checkpoints (cont.)

- **Packages**
  - ✓ **Are the names of the packages descriptive?**
  - ✓ **Does the package description match with the responsibilities of contained classes?**
  - ✓ **Do the package dependencies correspond to the relationships between the contained classes?**

continued

Checkpoints (cont.)

- **Packages**
  - ✓ **Do the classes contained in a package belong there according to the criteria for the package division?**
  - ✓ **Are there classes or collaborations of classes within a package that can be separated into an independent package?**
  - ✓ **Is the ratio between the number of packages and the number of classes appropriate?**

continued

Checkpoints (cont.)

- **Classes**
  - **Does the name of each class clearly reflect the role it plays?**
  - **Is the class cohesive (i.e., are all parts functionally coupled)?**
  - **Are all class elements needed by the use-case realizations?**
  - **Do the role names of the aggregations and associations accurately describe the relationship?**
  - **Are the multiplicities of the relationships correct?**

作业 9th

1. 子系统和包有什么关系?
2. 根据你的理解，子系统大致起到什么作用?
3. 怎样划分子系统?
4. 怎样定义接口？
5. 子系统及其接口设计的步骤和制品是什么？
6. 把软件的结构层次化划分时，应遵循的规律有哪些?

Chapter 9 Identify Design Elements

Agenda

- **Objectives**
- **Context in identify design elements**
- **Identify design elements steps**
- **Exercises**

Lab: Identify Design Elements

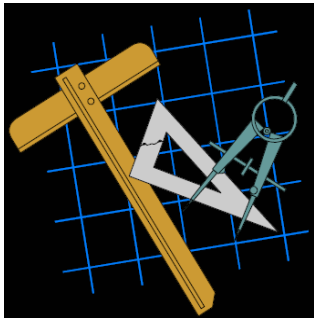
- **Given the following:**
  - **The analysis classes and their relationships**
  - **The layers, packages, and their dependencies**

Lab: Identify Design Elements (cont.)

- **Identify the following:**
  - **Design classes, subsystems , their interfaces and their relationships with other design elements**
  - **Mapping from the analysis classes to the design elements**
  - **The location of the design elements (e.g. subsystems and their design classes) in the architecture (i.e., the package/layer that contains the design element)**

Lab: 识别设计元素（子系统及其接口设计）

- 做出如下制品:
  - Table1 mapping analysis classes to design elements
  - Table2 listing design elements and their “owning” package
  - Class diagram ， for each subsystem developing an interface realization
  - Class diagram, containing all subsystem and its interface, and other design elements



Lab: Review（每组同学自己逐题思考）

- **Compare your results with the rest of the class**
  - **What subsystem did you find? Is it partitioned logically? Does it realize an interface (s)?**
  - **What analysis classes does it map to?**
  - **Do the package dependencies correspond to the relationships between the contained classes ?**
  - **Are the classes grouped logically?**
  - **Are there classes or collaborations of classes within a package that can be separated into an independent package?**