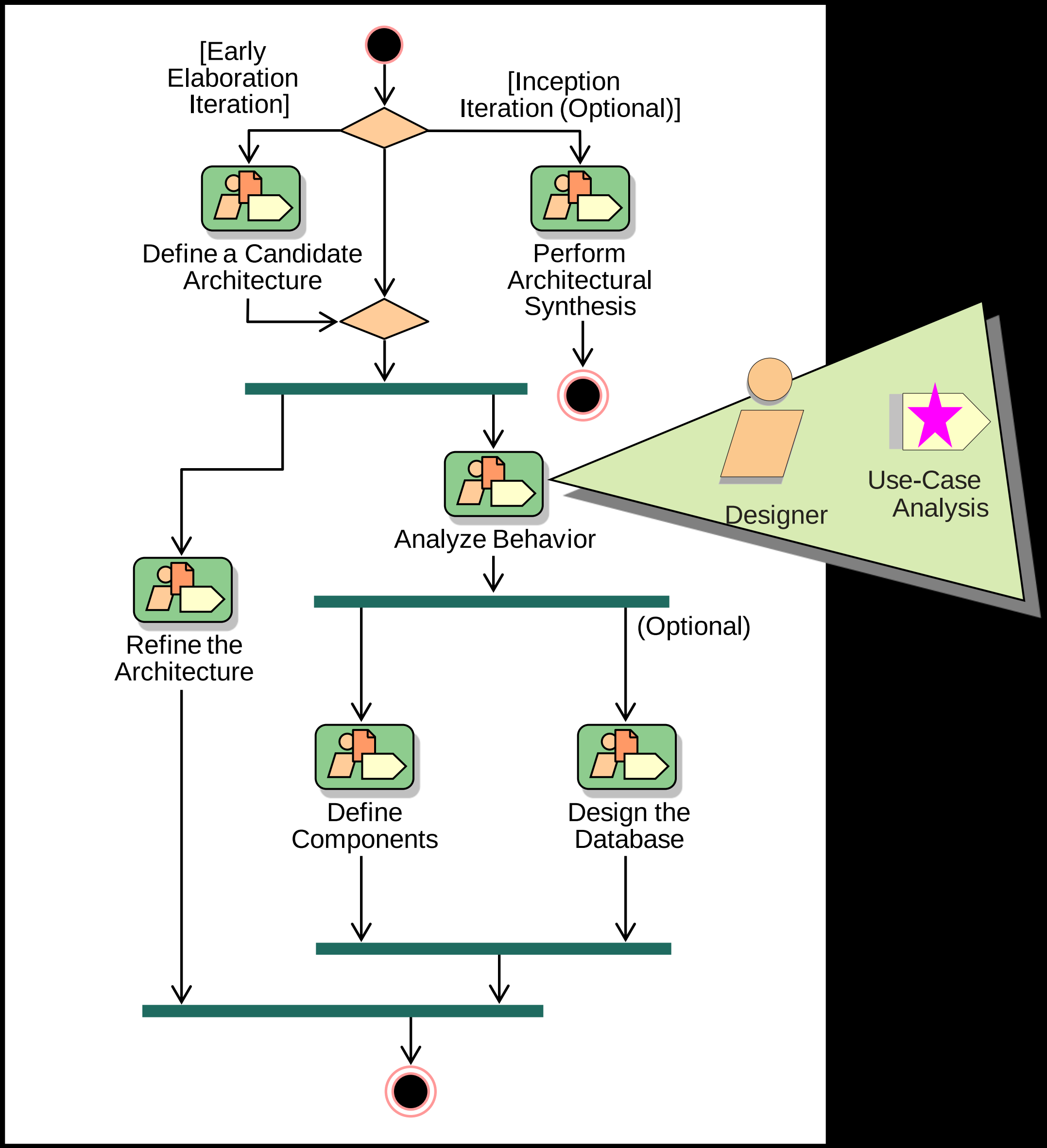
Object-Oriented Analysis and Design

Lecture 6: Use-Case Analysis

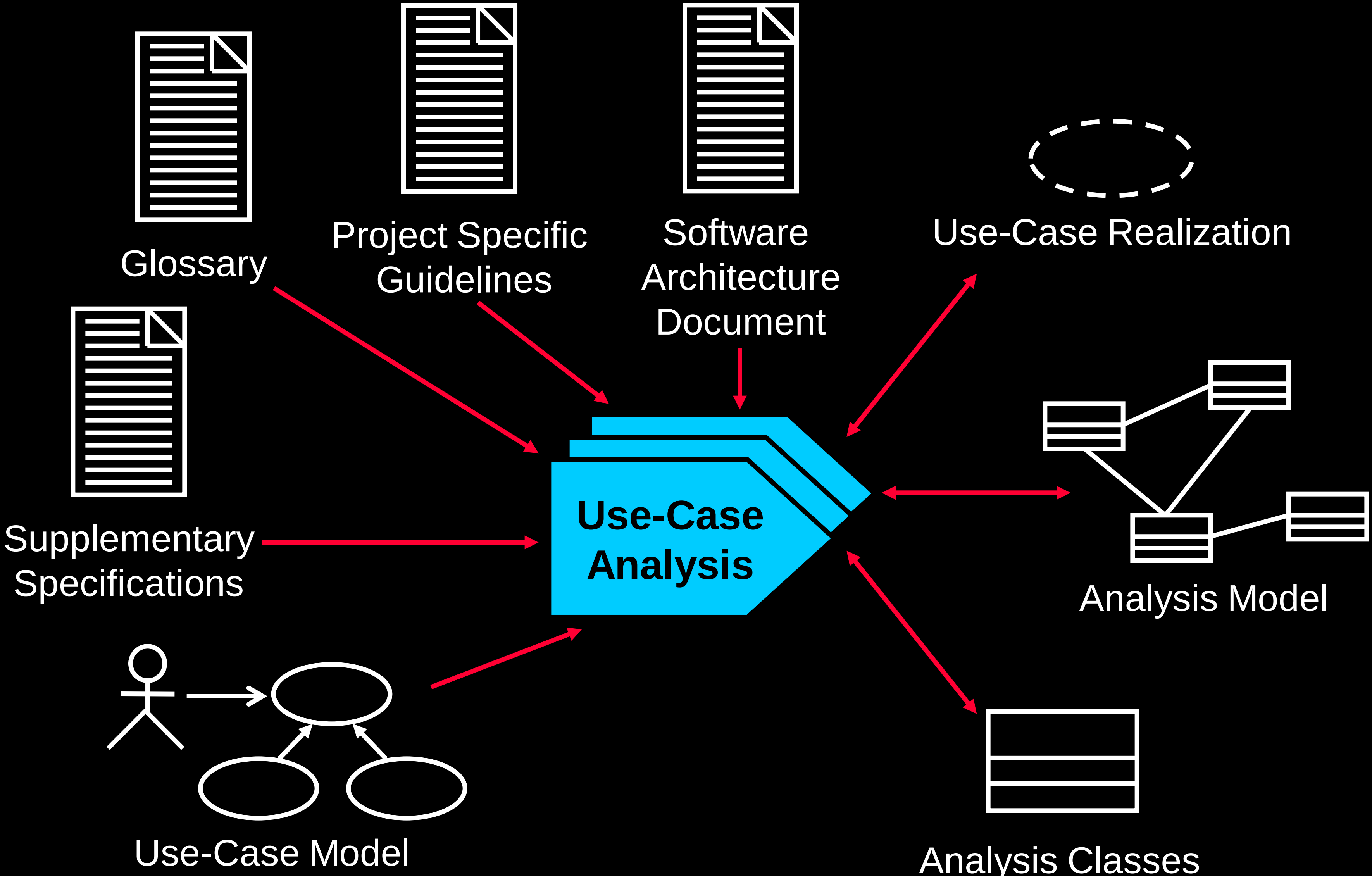
Objectives: Use-Case Analysis

- Explain the purpose of Use-Case Analysis and where in the lifecycle it is performed
- Identify the classes which perform a use-case flow of events
- Distribute the use-case behavior to those classes, identifying responsibilities of the classes
- Develop Use-Case Realizations that model the collaborations between instances of the identified classes

Use-Case Analysis in Context


[Early Elaboration Iteration]
[Inception Iteration (Optional)]
Define a Candidate Architecture
Perform Architectural Synthesis
Analyze Behavior
Designer
Use-Case Analysis
Refine the Architecture
(Optional)
Define Components
Design the Database

Use-Case Analysis Overview


Glossary

Project Specific Guidelines

Software Architecture Document

Use-Case Realization

Supplementary Specifications

Use-Case Analysis

Analysis Model

Use-Case Model

Analysis Classes

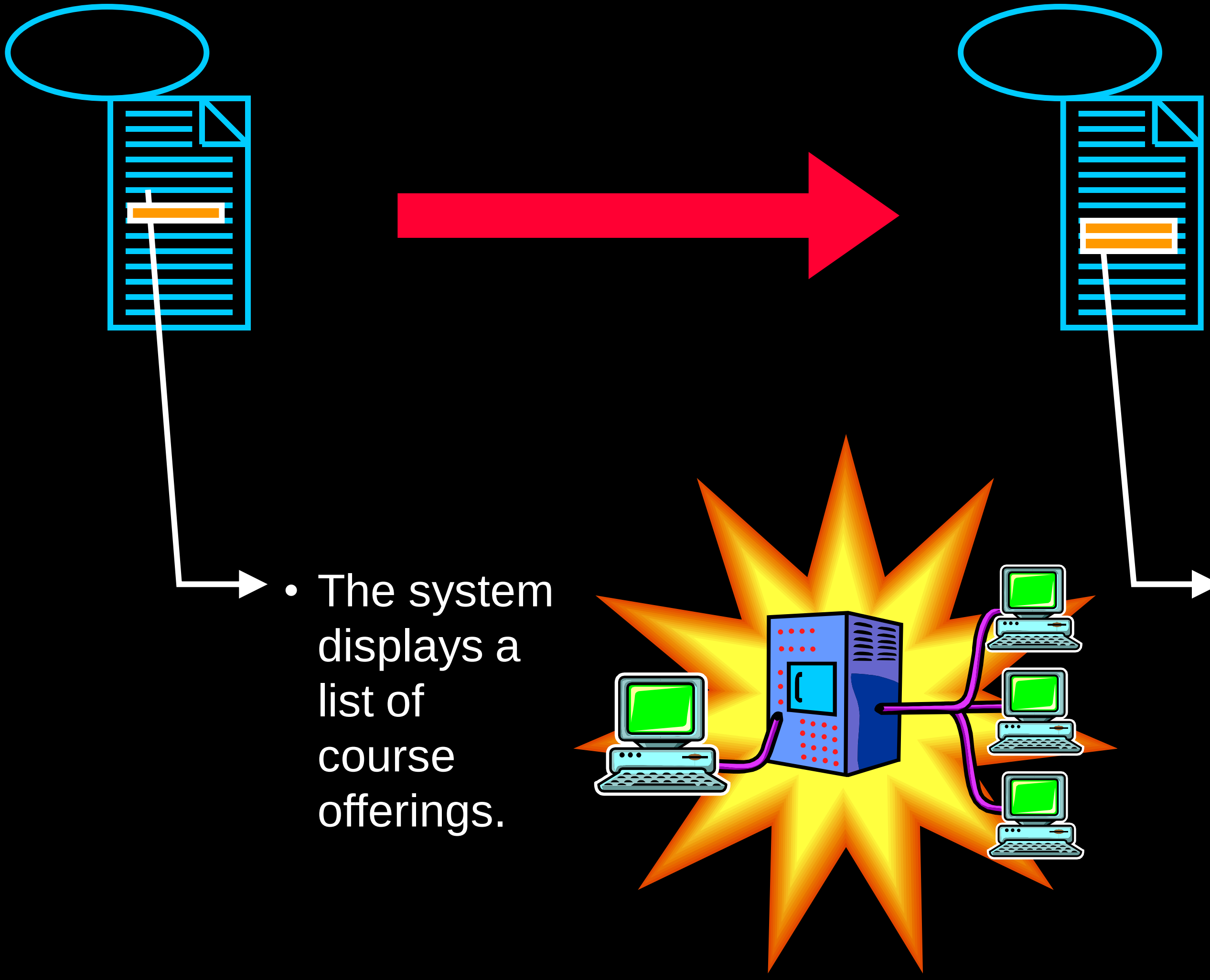
Use-Case Analysis Steps

- Supplement the Use-Case Description
- For each Use-Case Realization
  - Find Classes from Use-Case Behavior
  - Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

Use-Case Analysis Steps

- ★ Supplement the Use-Case Description
- For each Use-Case Realization
  - Find Classes from Use-Case Behavior
  - Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

Supplement the Use-Case Description



- The system displays a list of course offerings.

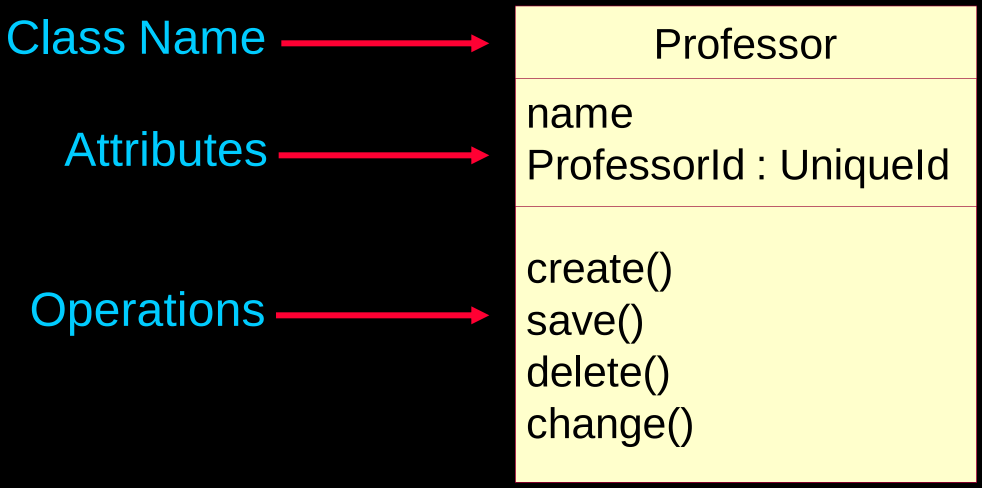
- The system retrieves and displays a list of current course offerings from the course catalog legacy database.

Use-Case Analysis Steps

- Supplement the Use-Case Description
- For each Use-Case Realization
  - ★ Find Classes from Use-Case Behavior
  - Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

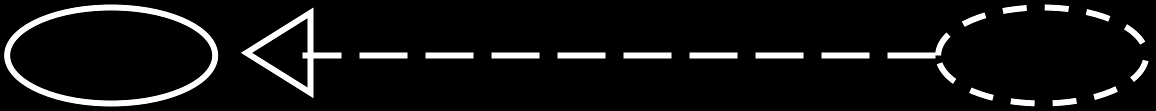
Review: Class

- An abstraction
- Describes a group of objects with common:
  - Properties (attributes)
  - Behavior (operations)
  - Relationships
  - Semantics



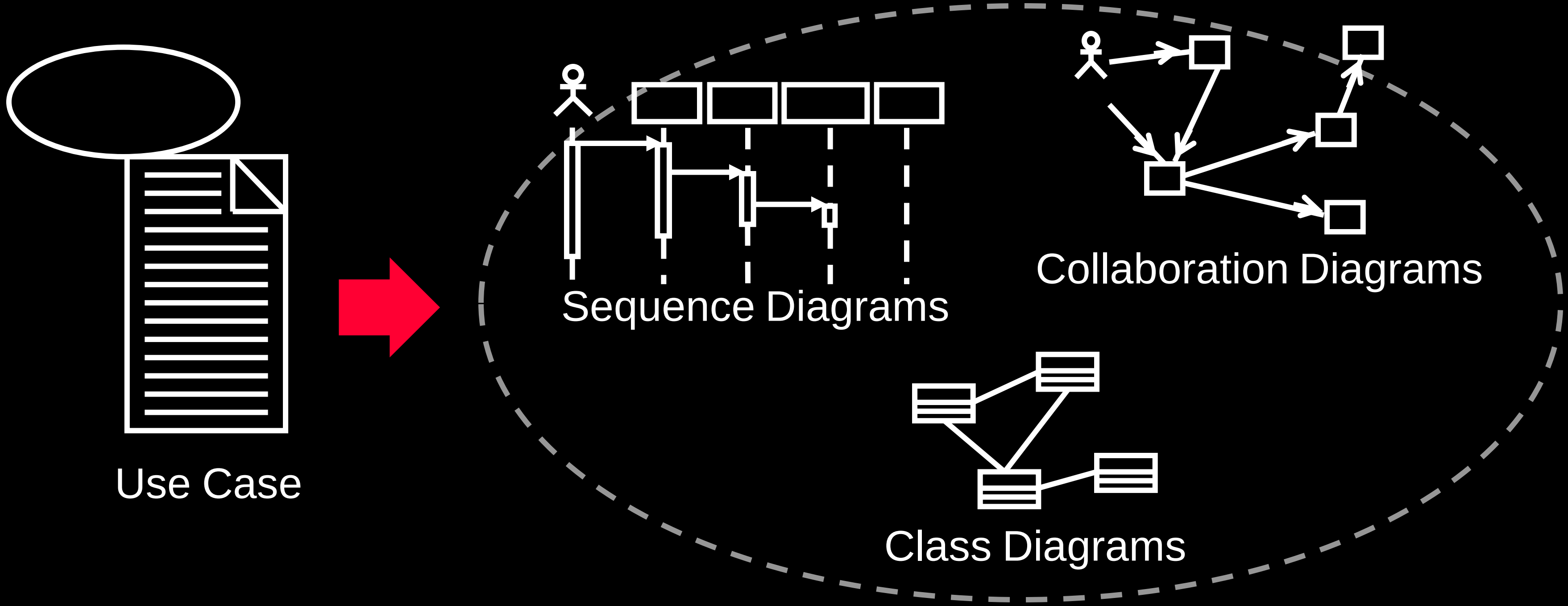
Review: Use-Case Realization

Use-Case Model *Design Model*



Use Case

Use-Case Realization



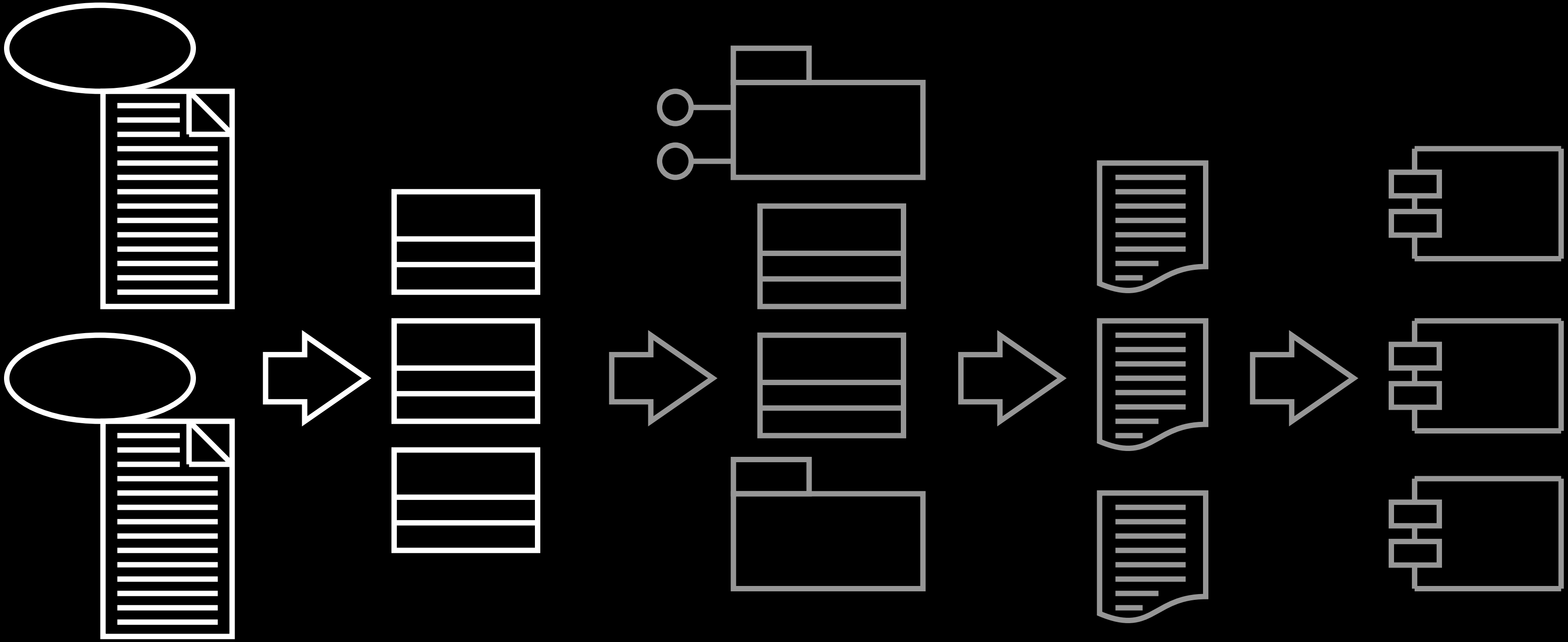
Use Case

Sequence Diagrams

Collaboration Diagrams

Class Diagrams

Analysis Classes: A First Step Toward Executables



Use Cases

Analysis Classes

Design Elements

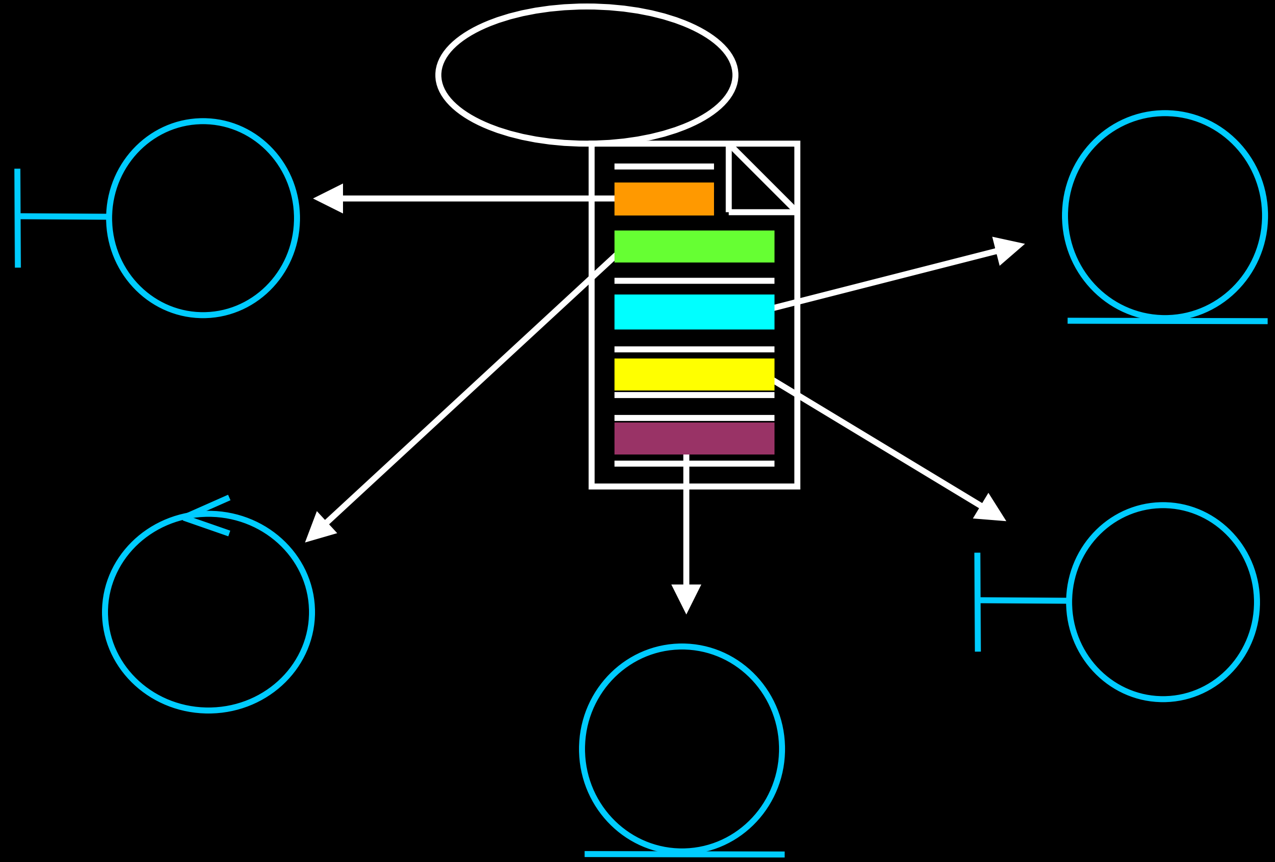
Source Code

Exec

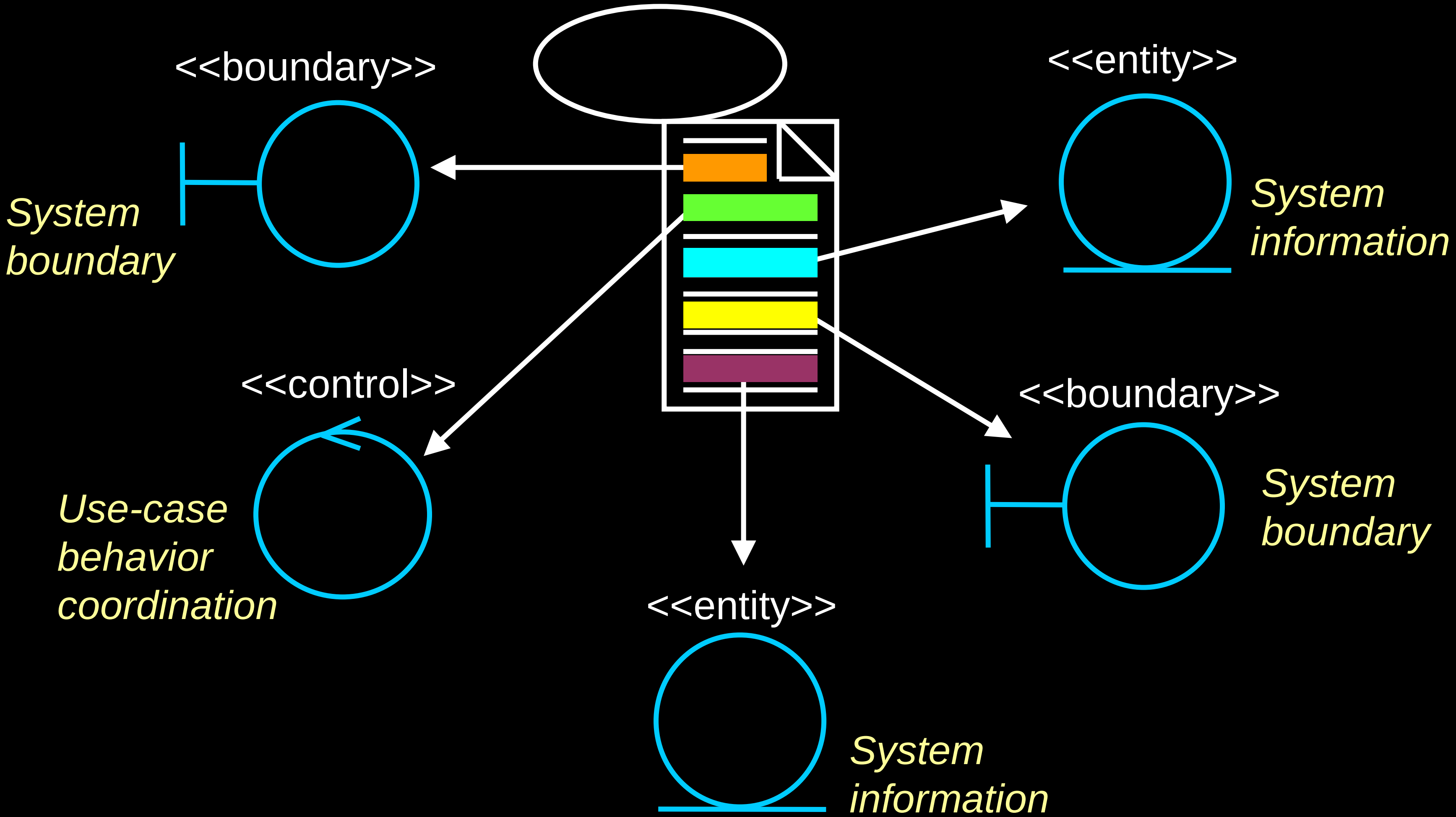
Use-Case Analysis

Find Classes from Use-Case Behavior

- The complete behavior of a use case has to be distributed to analysis classes



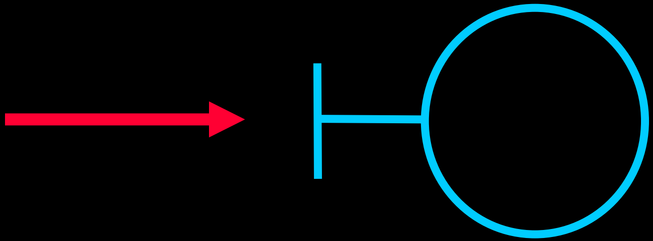
What Is an Analysis Class?



What Is a Boundary Class?

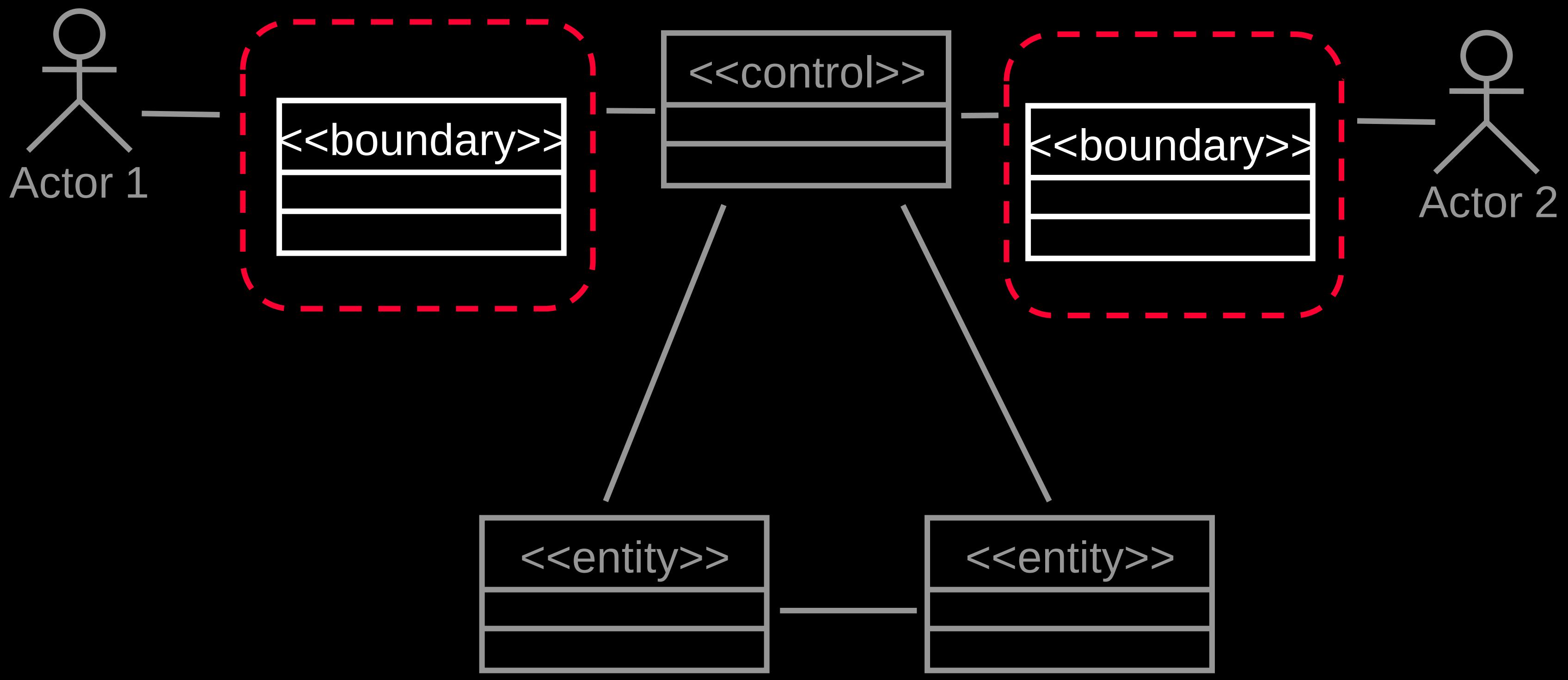
- Intermediates between the interface and something outside the system
- Several Types
  - User interface classes
  - System interface classes
  - Device interface classes
- *One boundary class per actor/use-case pair*

Analysis class stereotype



Environment Dependent

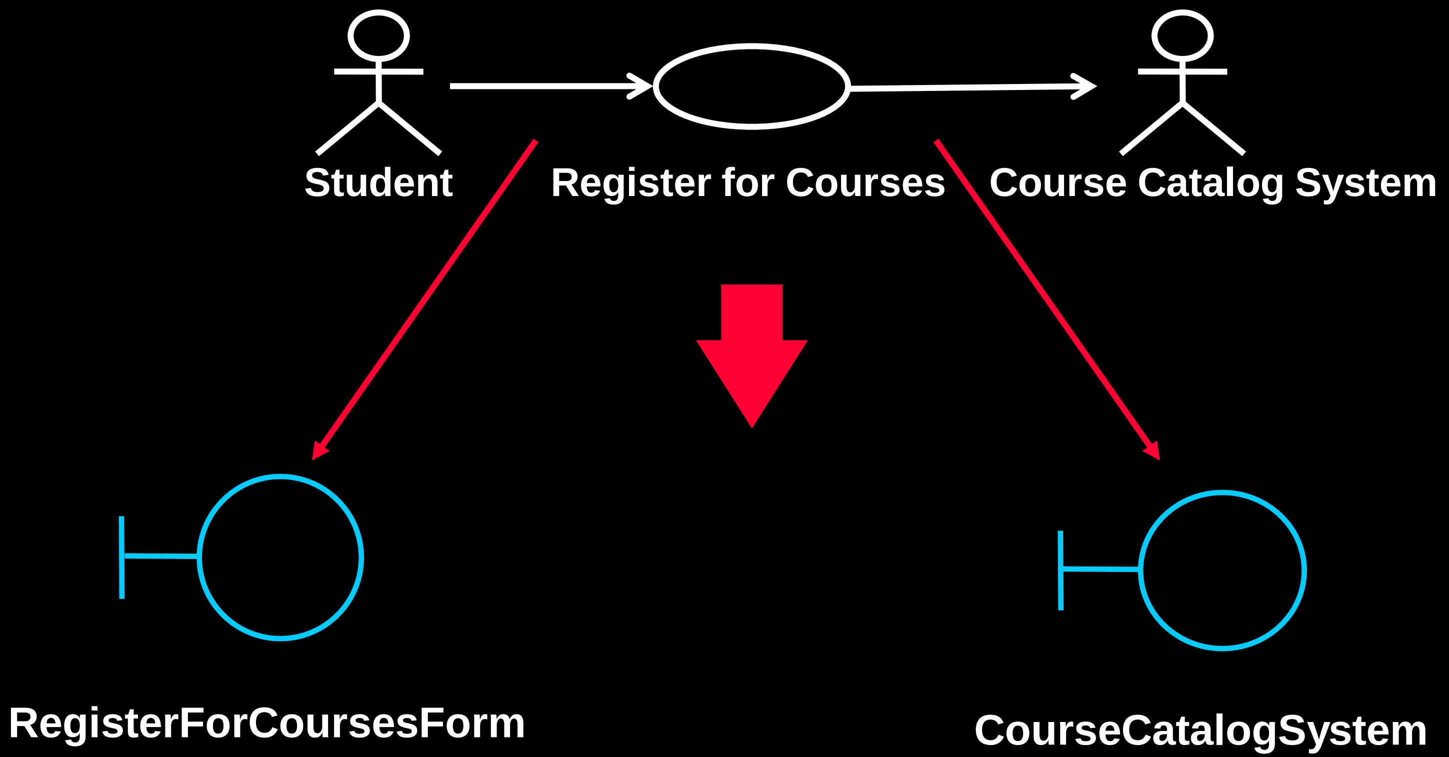
The Role of a Boundary Class



Model interaction between the system and its environment

Example: Finding Boundary Classes

- One boundary class per actor/use case pair



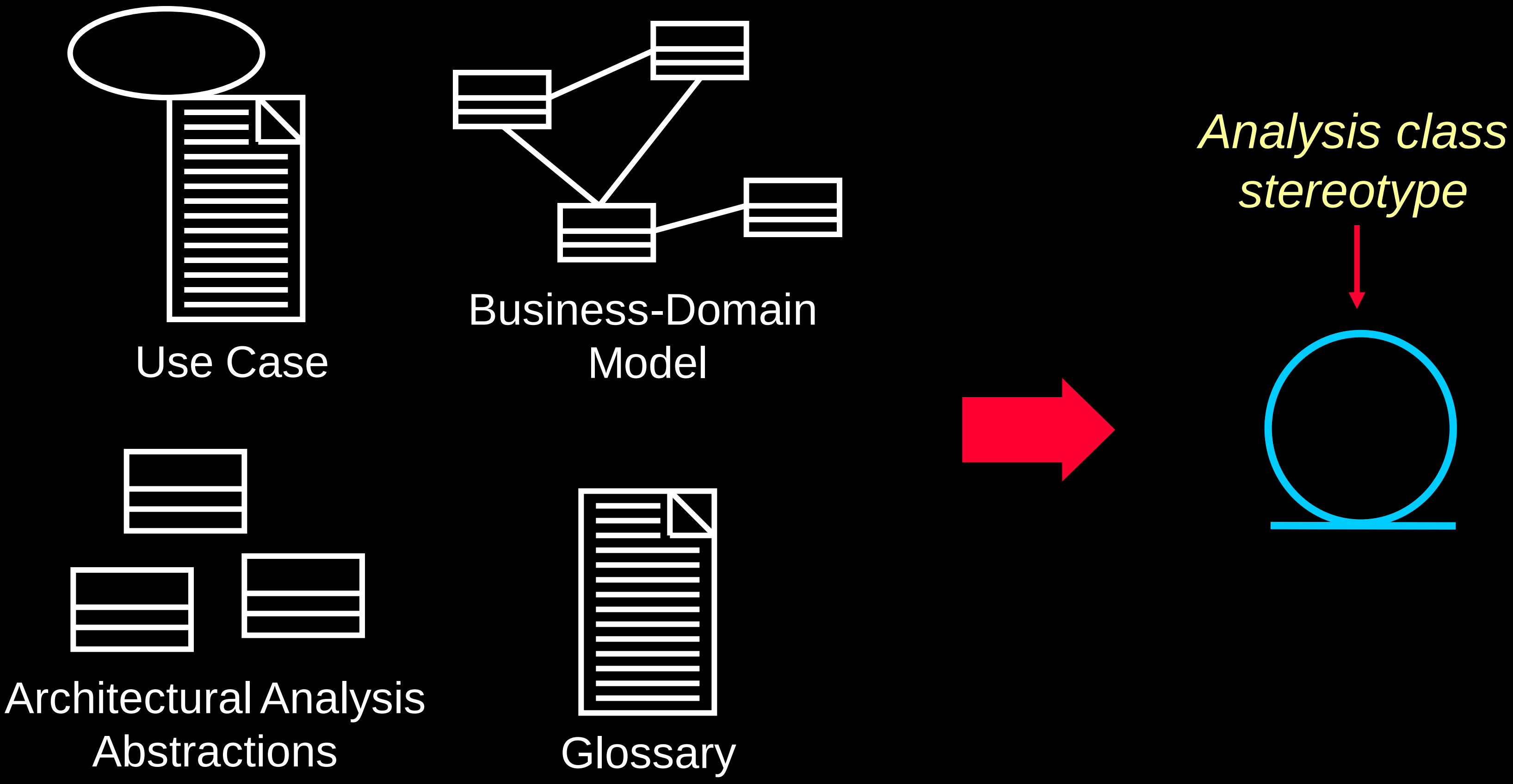
Guidelines: Boundary Class

- User Interface Classes
  - Concentrate on what information is presented to the user
  - Do NOT concentrate on the UI details
- System and Device Interface Classes
  - Concentrate on what protocols must be defined
  - Do NOT concentrate on how the protocols will be implemented

Concentrate on the responsibilities, not the details!

What Is an Entity Class?

- Key abstractions of the system



Use Case

Business-Domain Model

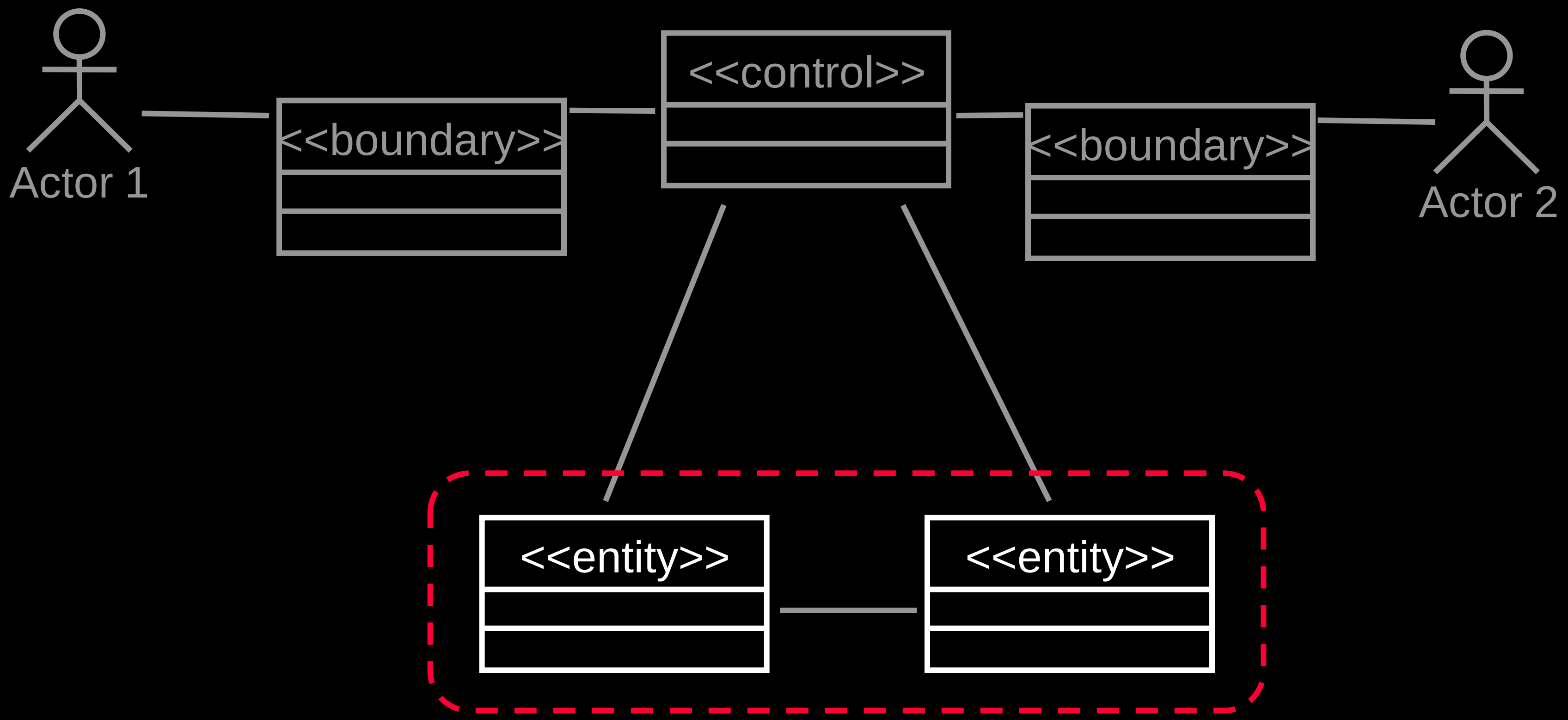
Architectural Analysis Abstractions

Glossary

Analysis class stereotype

Environment Independent

The Role of an Entity Class



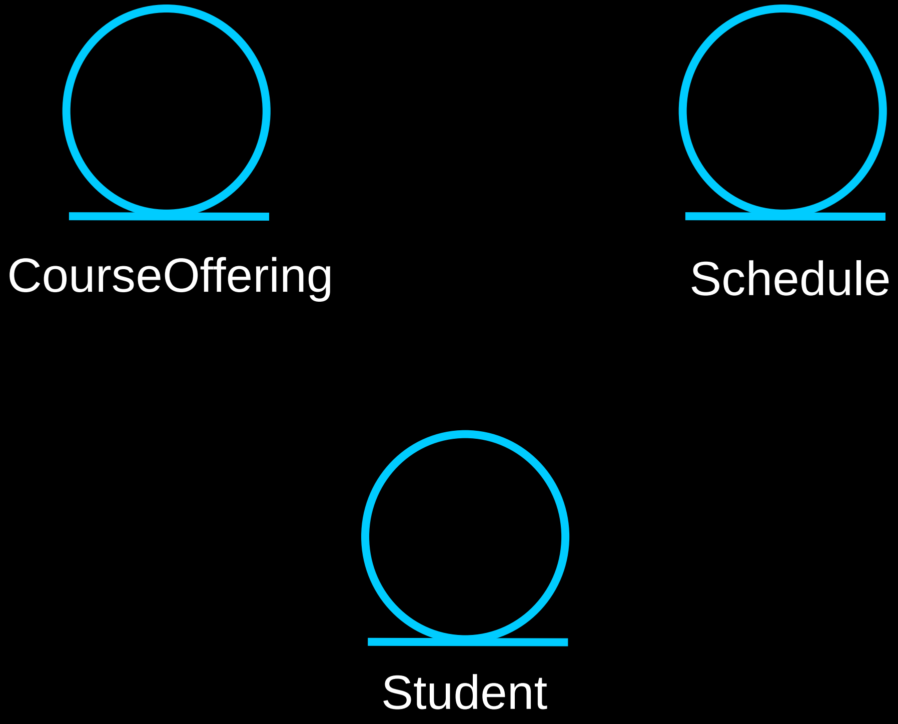
Store and manage information in the system

Example: Finding Entity Classes

- Use use-case flow of events as input
- Key abstractions of the use case
- Traditional, filtering nouns approach
  - Underline noun clauses in the use-case flow of events
  - Remove redundant candidates
  - Remove vague candidates
  - Remove actors (out of scope)
  - Remove implementation constructs
  - Remove attributes (save for later)
  - Remove operations

Example: Candidate Entity Classes

- Register for Courses (Create Schedule)



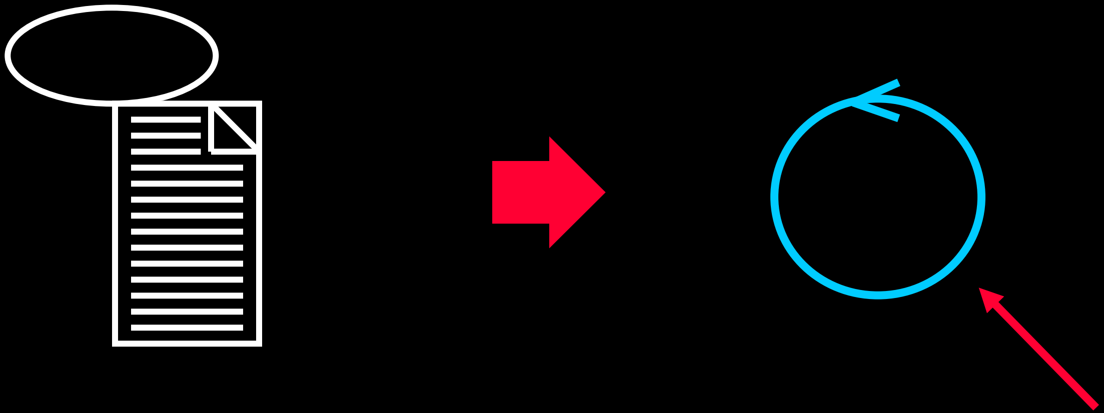
CourseOffering

Schedule

Student

What Is a Control Class?

- Use-case behavior coordinator
  - More complex use cases generally require one or more control cases

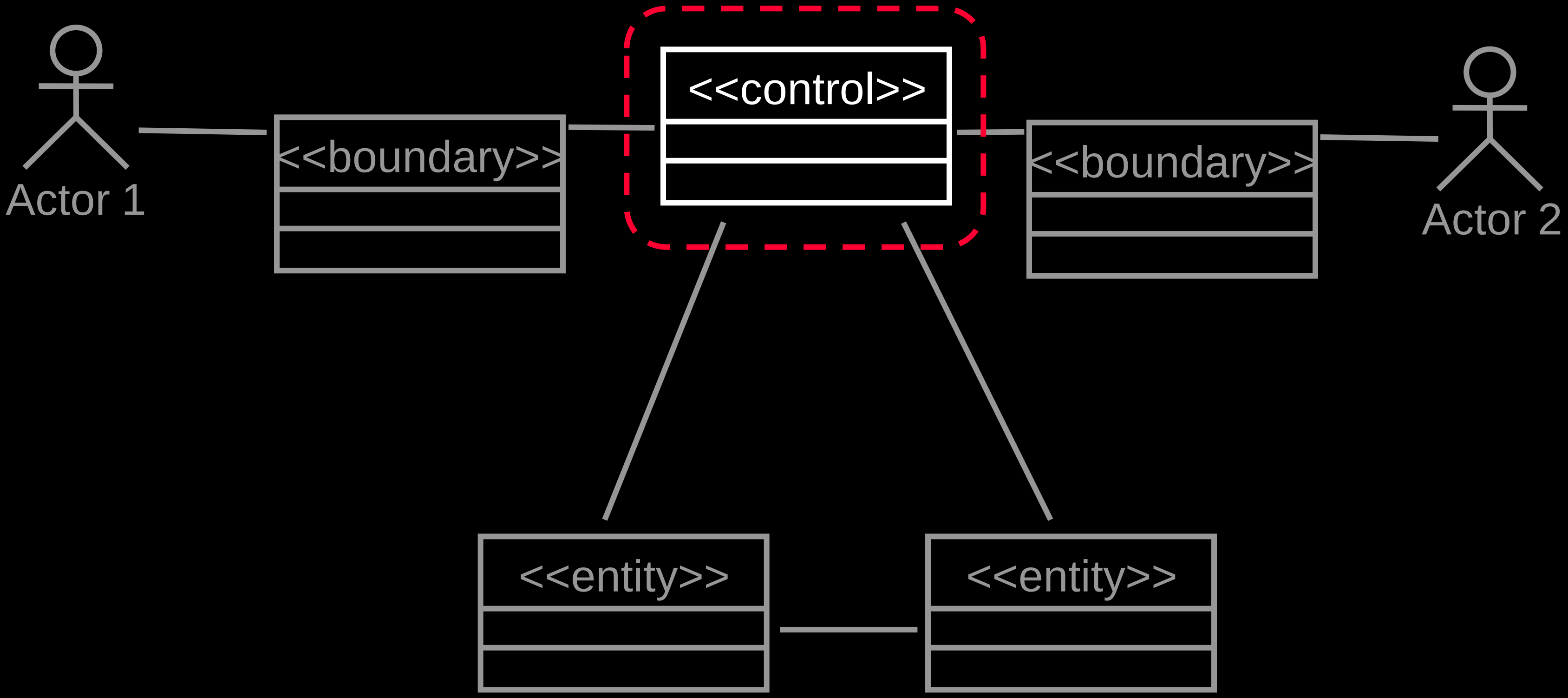


Use Case

Analysis class stereotype

Use-case dependent, Environment independent

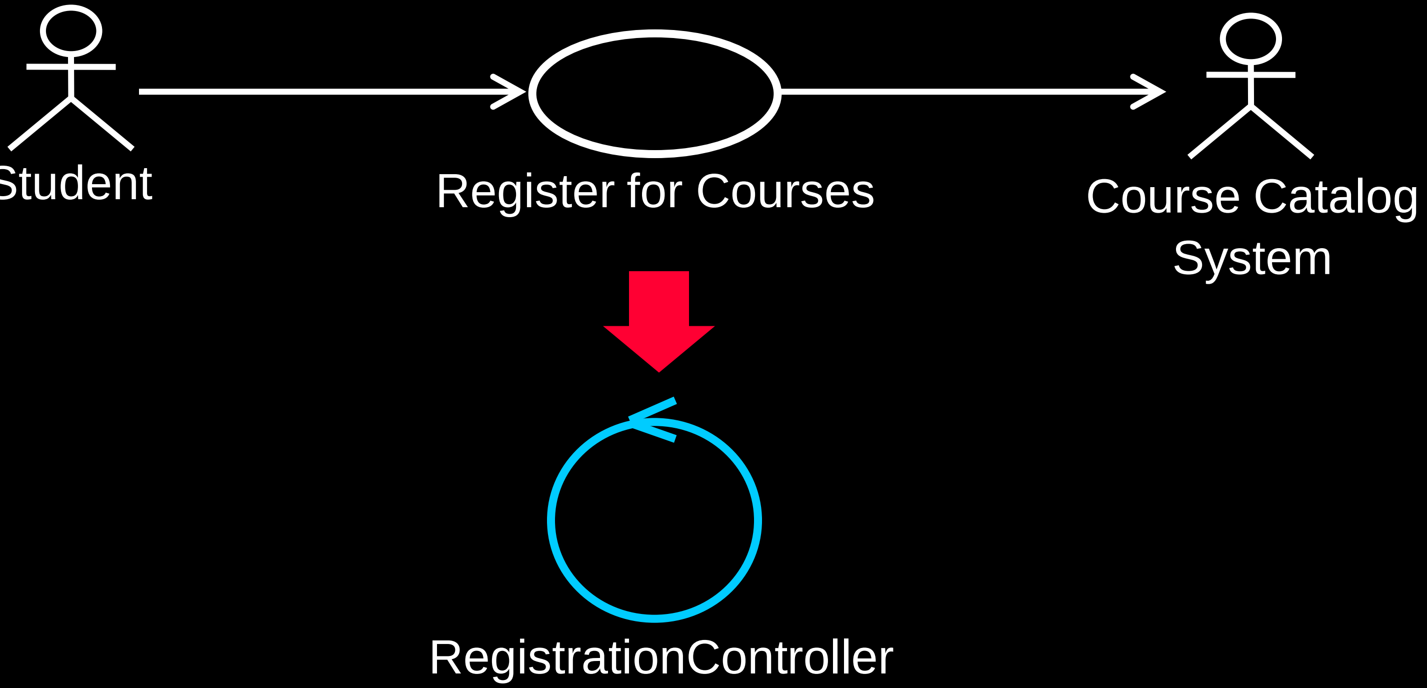
The Role of a Control Class



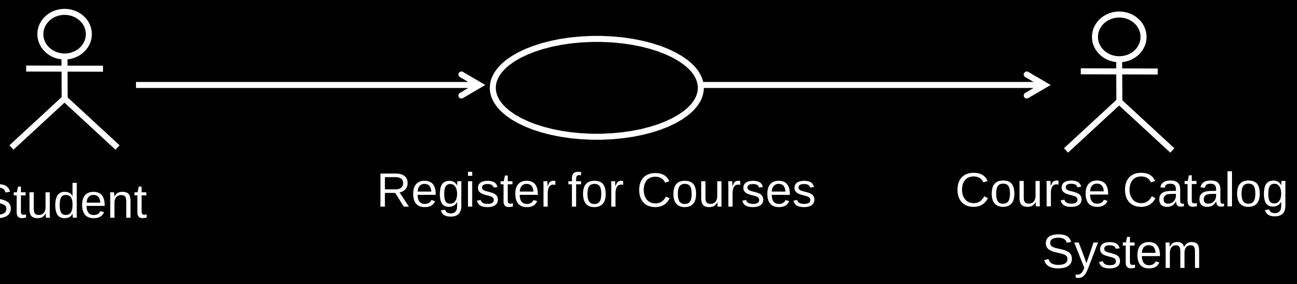
Coordinate the use-case behavior

Example: Finding Control Classes

- In general, identify one control class per use case.
  - As analysis continues, a complex use case's control class may evolve into more than one class

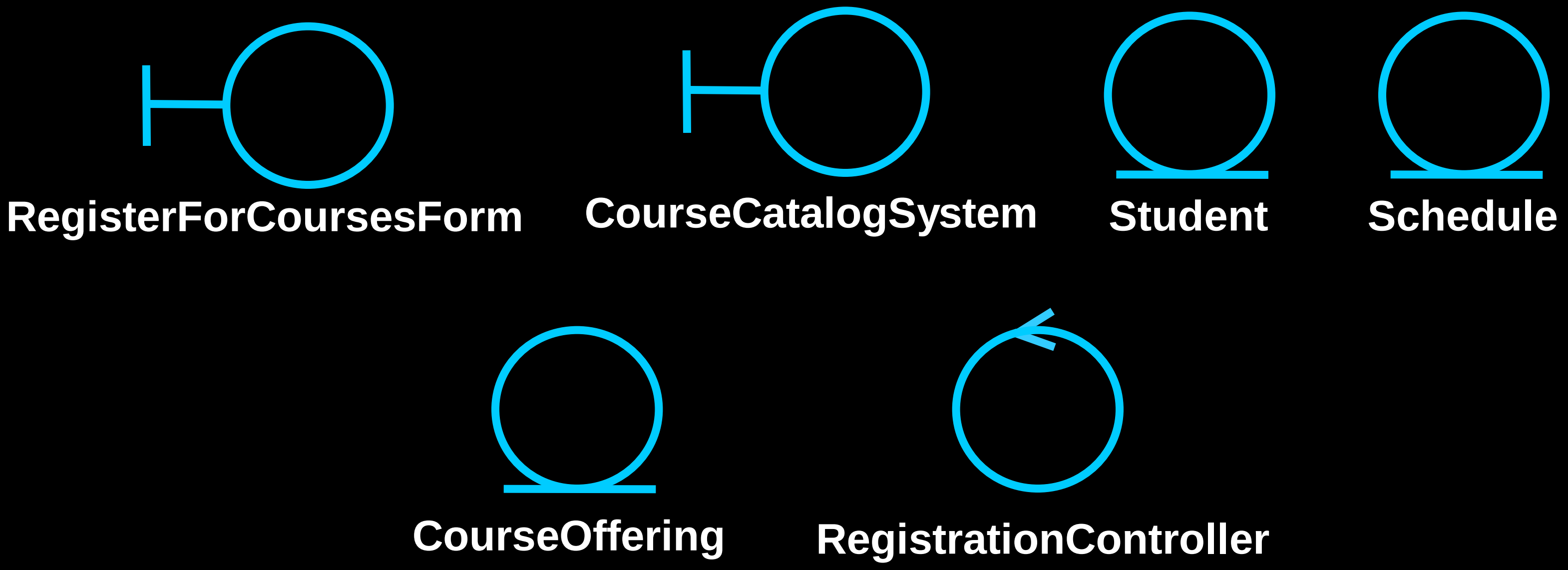


Example: Summary: Analysis Classes



Use-Case Model

Design Model

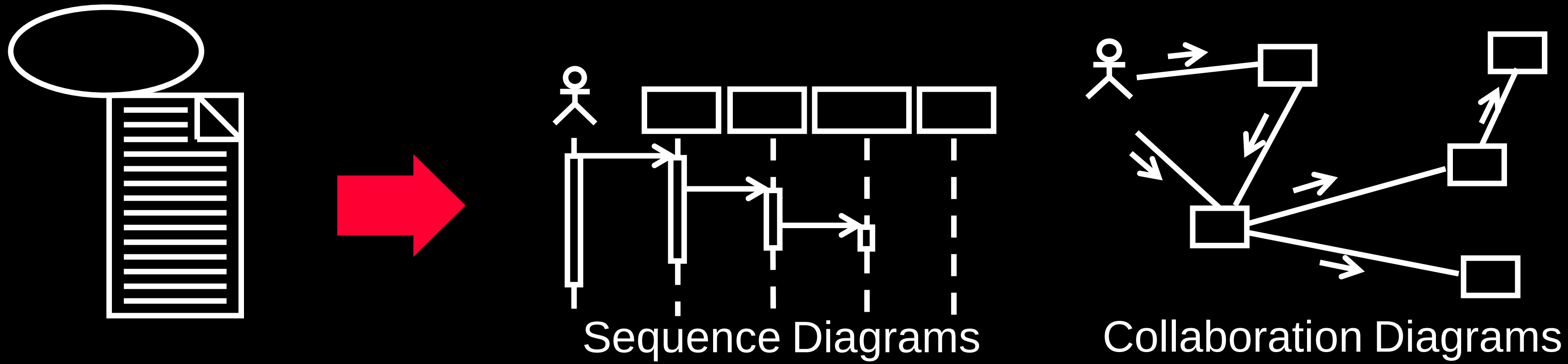


Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- For each Use-Case Realization
  - Find Classes from Use-Case Behavior
  - ★ Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

Distribute Use-Case Behavior to Classes

- For each use-case flow of events:
  - Identify analysis classes
  - Allocate use-case responsibilities to analysis classes
  - Model analysis class interactions in Interaction diagrams



Sequence Diagrams

Collaboration Diagrams

Use Case

Use-Case Realization

Guidelines: Allocating Responsibilities to Classes

- Use analysis class stereotypes as a guide
  - **Boundary Classes**
    - Behavior that involves communication with an actor
  - **Entity Classes**
    - Behavior that involves the data encapsulated within the abstraction
  - **Control Classes**
    - Behavior specific to a use case or part of a very important flow of events

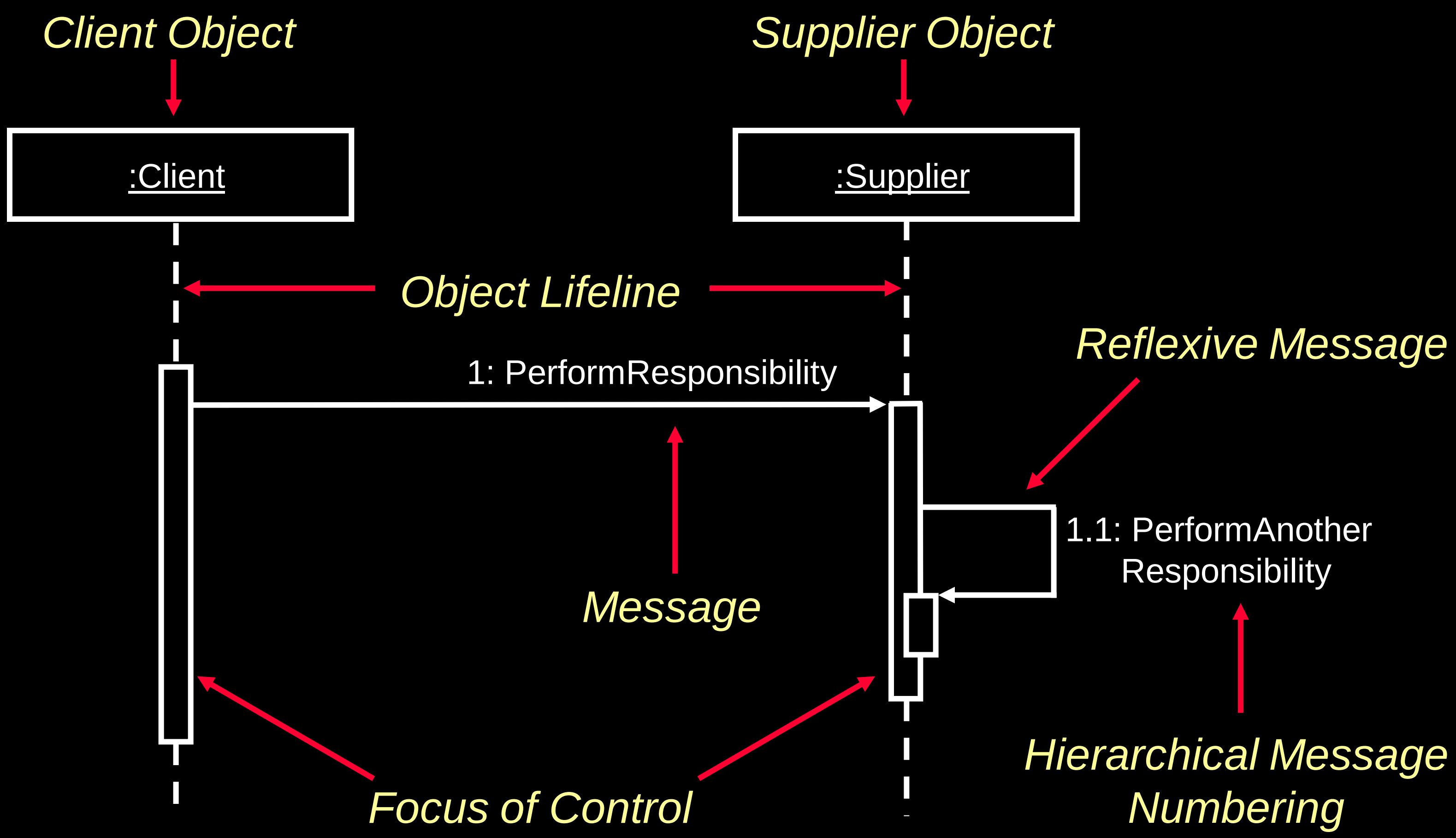
(continued)

Guidelines: Allocating Responsibilities to Classes (cont.)

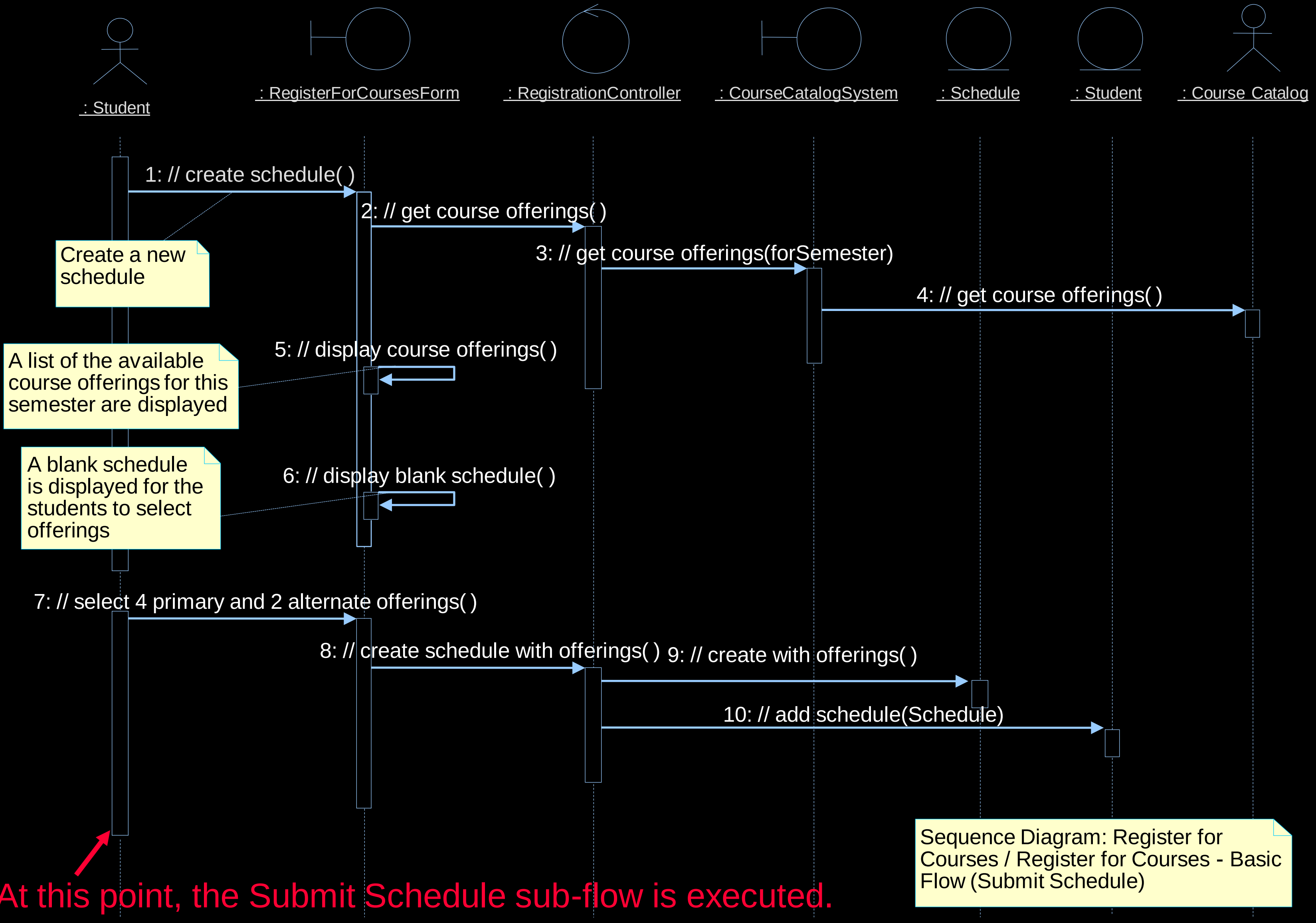
- Who has the data needed to perform the responsibility?
  - If one class has the data, put the responsibility with the data
  - If multiple classes have the data:
    - Put the responsibility with one class and add a relationship to the other
    - Create a new class, put the responsibility in the new class, and add relationships to classes needed to perform the responsibility
    - Put the responsibility in the control class, and add relationships to classes needed to perform the responsibility

The Anatomy of Sequence Diagrams

This is a sample script.



Example: Sequence Diagram



: Student
: RegisterForCoursesForm
: RegistrationController
: CourseCatalogSystem
: Schedule
: Student
: Course Catalog
1: // create schedule( )
2: // get course offerings( )
3: // get course offerings(forSemester)
4: // get course offerings( )
5: // display course offerings( )
6: // display blank schedule( )
7: // select 4 primary and 2 alternate offerings( )
8: // create schedule with offerings( )
9: // create with offerings( )
10: // add schedule(Schedule)

Create a new schedule

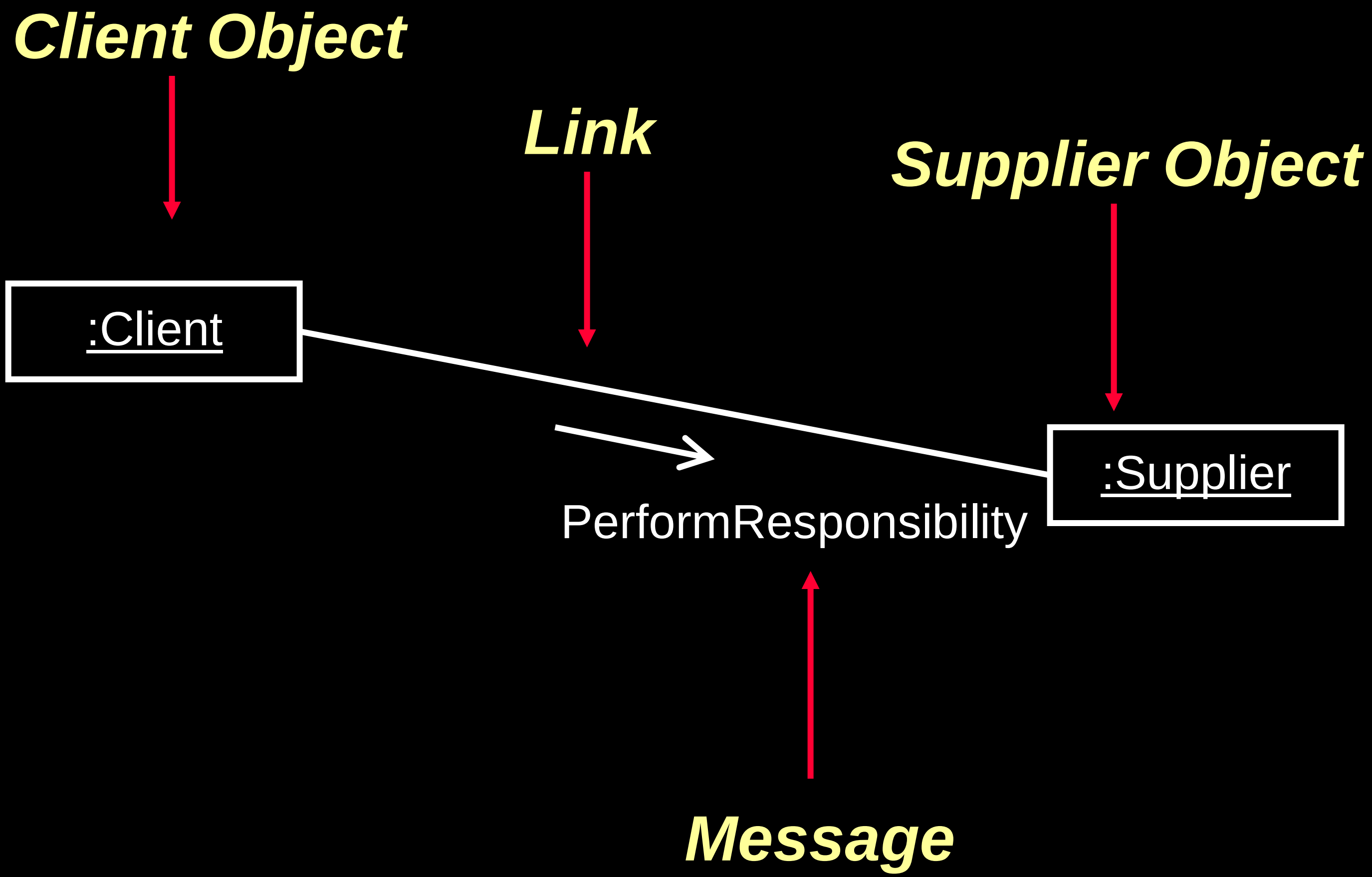
A list of the available course offerings for this semester are displayed

A blank schedule is displayed for the students to select offerings

Sequence Diagram: Register for Courses / Register for Courses - Basic Flow (Submit Schedule)

At this point, the Submit Schedule sub-flow is executed.

The Anatomy of Collaboration Diagrams



Client Object

Link

Supplier Object

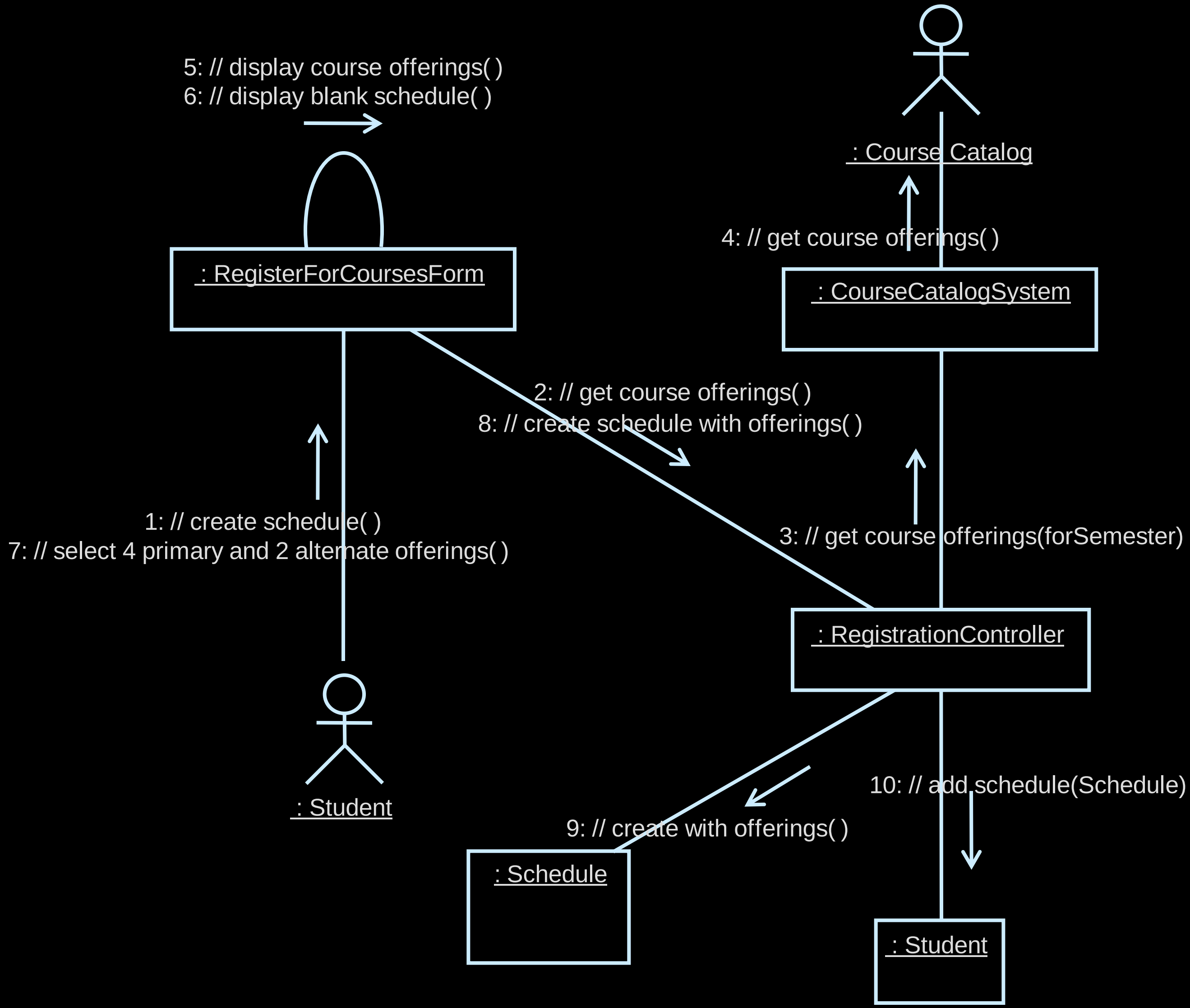
:Client

:Supplier

PerformResponsibility

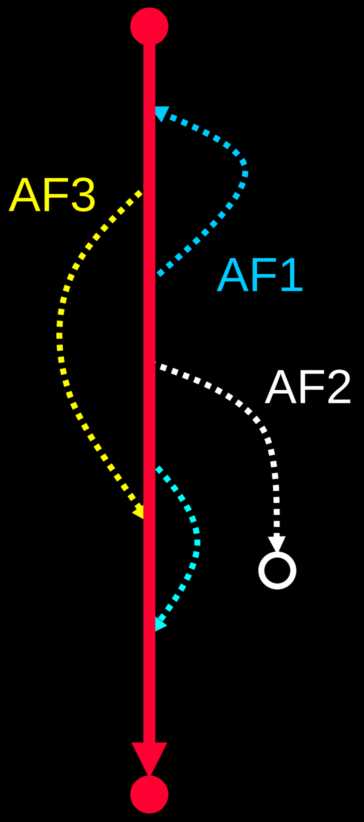
Message

Example: Collaboration Diagram

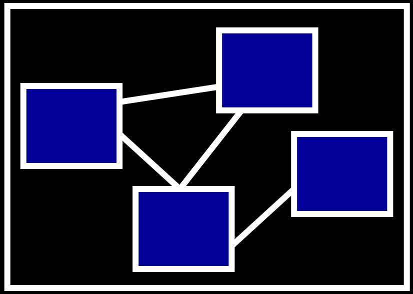

5: // display course offerings( )
6: // display blank schedule( )
: Course Catalog
4: // get course offerings( )
: RegisterForCoursesForm
: CourseCatalogSystem
2: // get course offerings( )
8: // create schedule with offerings( )
1: // create schedule( )
7: // select 4 primary and 2 alternate offerings( )
3: // get course offerings(forSemester)
: RegistrationController
: Student
10: // add schedule(Schedule)
9: // create with offerings( )
: Schedule
: Student

One Interaction Diagram Is Not Good Enough

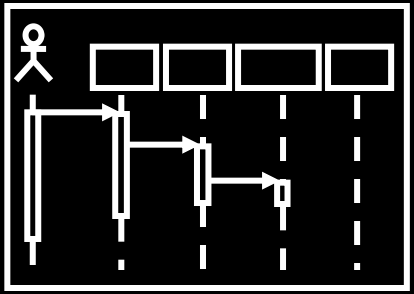
Basic Flow



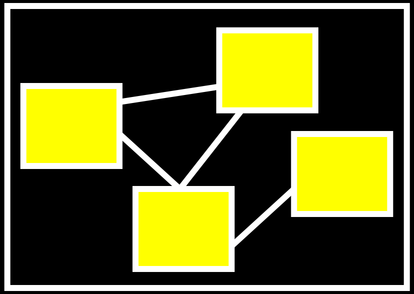
Alternate Flow 1



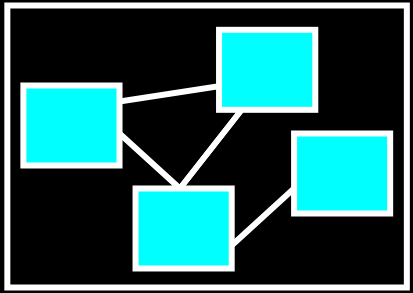
Alternate Flow 2



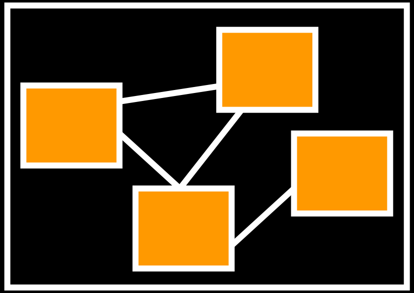
Alternate Flow 3



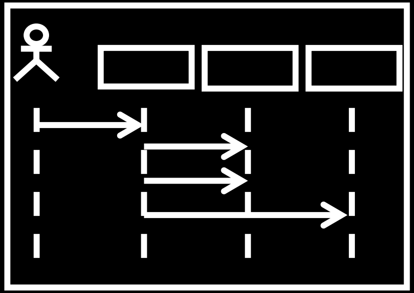
Alternate Flow 4



Alternate Flow 5



Alternate Flow n



Collaboration Diagrams vs. Sequence Diagrams

- Collaboration Diagrams
  - Show relationships in addition to interactions
  - Better for visualizing patterns of collaboration
  - Better for visualizing all of the effects on a given object
  - Easier to use for brainstorming sessions

- Sequence Diagrams
  - Show the explicit sequence of messages
  - Better for visualizing overall flow
  - Better for real-time specifications and for complex scenarios

Use-Case Analysis Steps

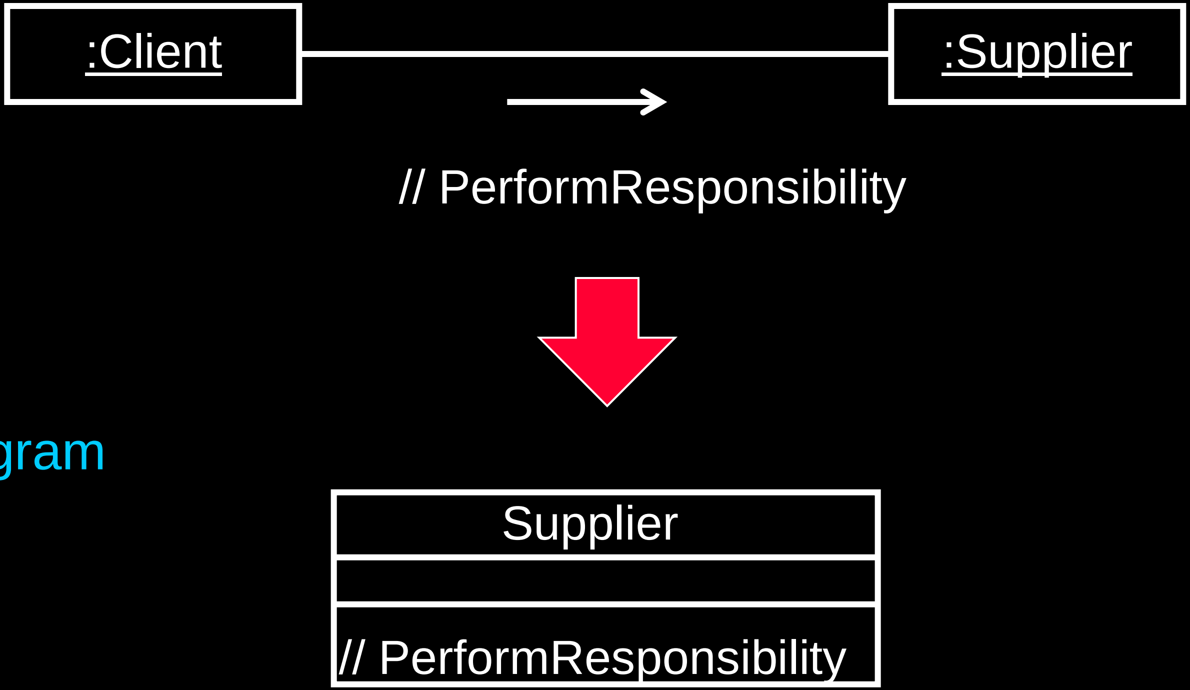
- Supplement the Use-Case Descriptions
- For each Use-Case Realization
  - Find Classes from Use-Case Behavior
  - Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - ★ Describe Responsibilities
  - Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

Describe Responsibilities

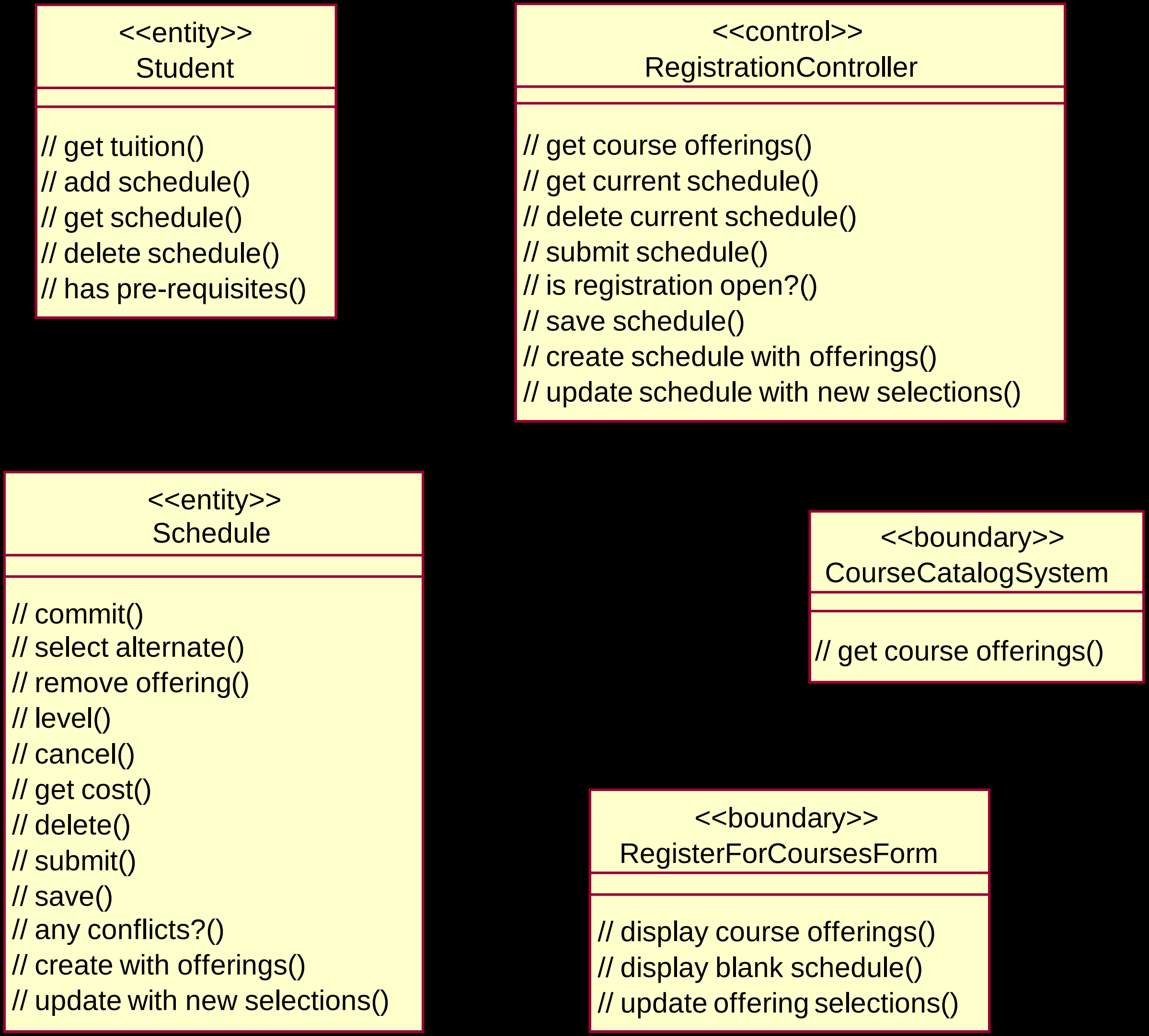
- What are responsibilities?
- How do I find them?

Interaction Diagram

Class Diagram



Example: View of Participating Classes (VOPC) Class Diagram



<<entity>>
Student

// get tuition()
// add schedule()
// get schedule()
// delete schedule()
// has pre-requisites()

<<control>>
RegistrationController

// get course offerings()
// get current schedule()
// delete current schedule()
// submit schedule()
// is registration open?()
// save schedule()
// create schedule with offerings()
// update schedule with new selections()

<<entity>>
Schedule

// commit()
// select alternate()
// remove offering()
// level()
// cancel()
// get cost()
// delete()
// submit()
// save()
// any conflicts?()
// create with offerings()
// update with new selections()

<<boundary>>
CourseCatalogSystem

// get course offerings()

<<boundary>>
RegisterForCoursesForm

// display course offerings()
// display blank schedule()
// update offering selections()

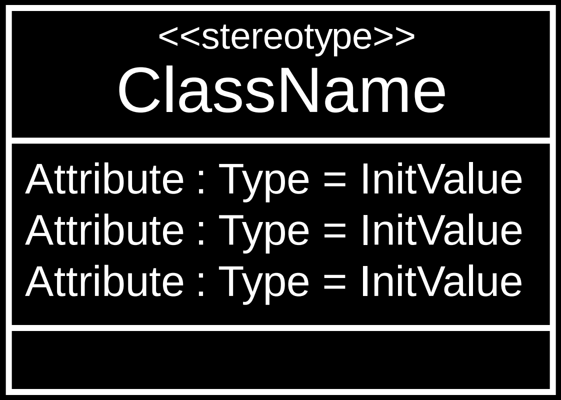
Maintaining Consistency: What to Look For

- In order of criticality
  - Redundant responsibilities across classes
  - Disjoint responsibilities within classes
  - Class with one responsibility
  - Class with no responsibilities
  - Better distribution of behavior
  - Class that interacts with many other classes

Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- For each Use-Case Realization
  - Find Classes from Use-Case Behavior
  - Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - ★ Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

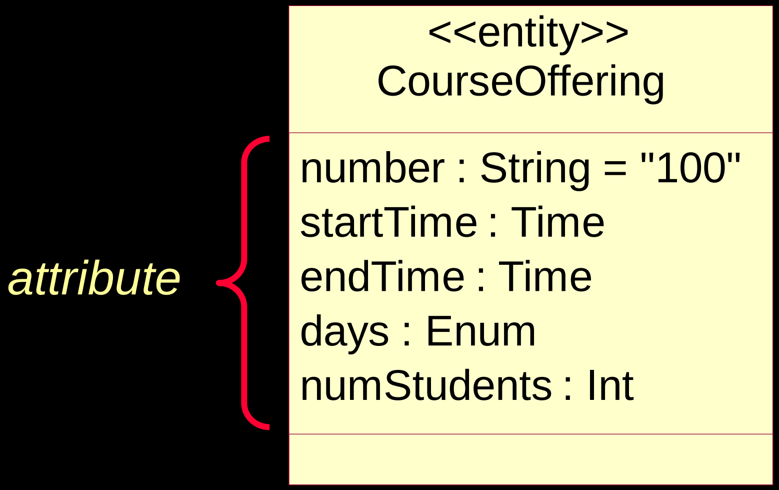
Review: What Is an Attribute?



<<stereotype>>
ClassName

Attribute : Type = InitValue
Attribute : Type = InitValue
Attribute : Type = InitValue

In analysis, do not spend time on attribute signatures



<<entity>>
CourseOffering

number : String = "100"
startTime : Time
endTime : Time
days : Enum
numStudents : Int

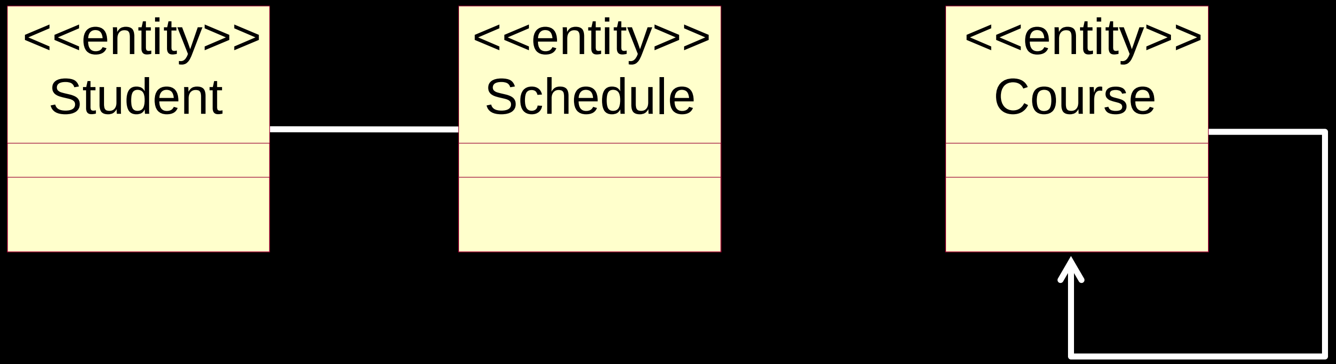
attribute

Finding Attributes

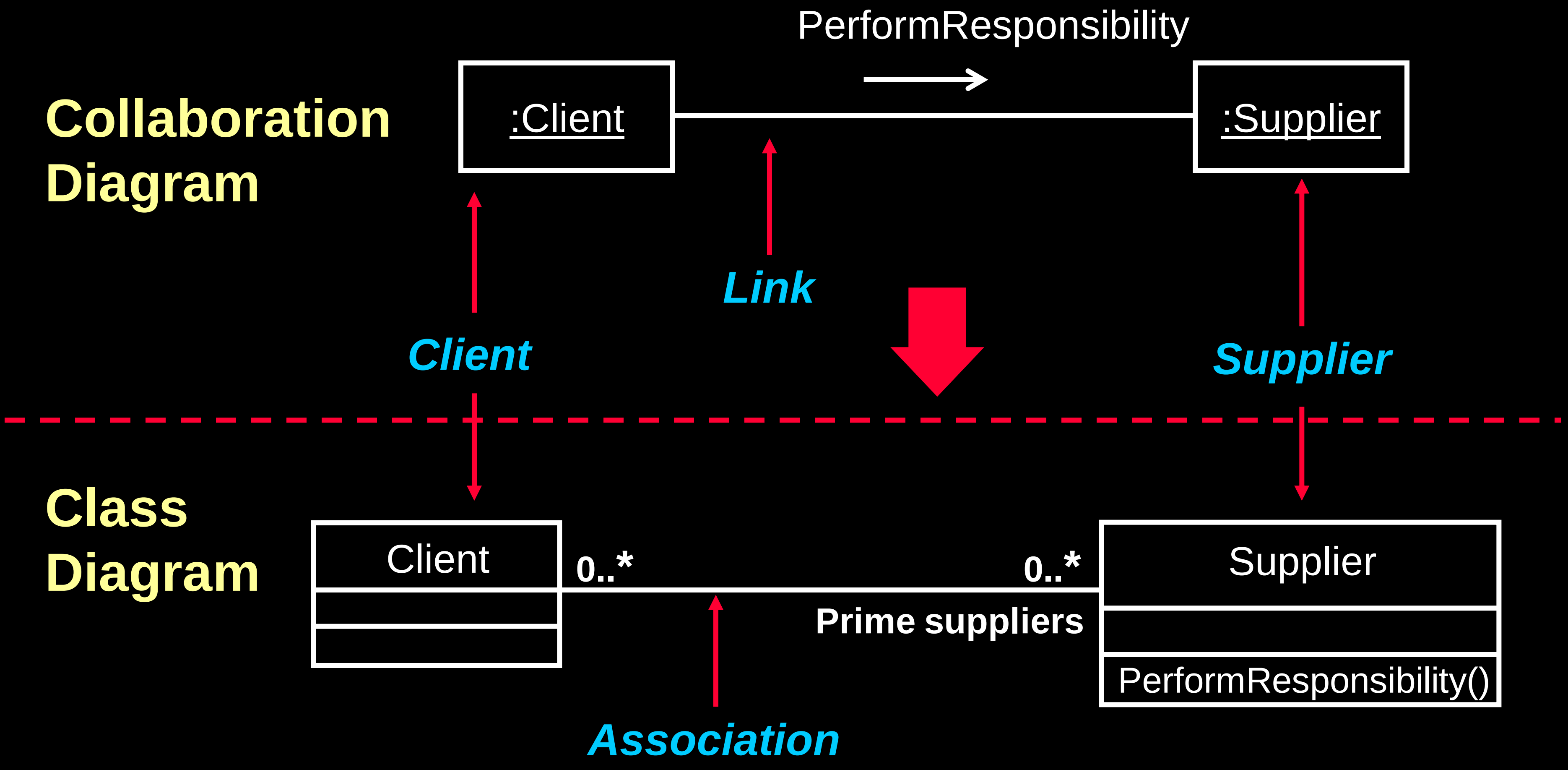
- Properties/characteristics of identified classes
- Information retained by identified classes
- “Nouns” that did not become classes
  - Information whose value is the important thing
  - Information that is uniquely "owned” by an object
  - Information that has no behavior

Review: What Is an Association?

- The semantic relationship between two or more classifiers that specifies connections among their instances
  - A structural relationship, specifying that objects of one thing are connected to objects of another



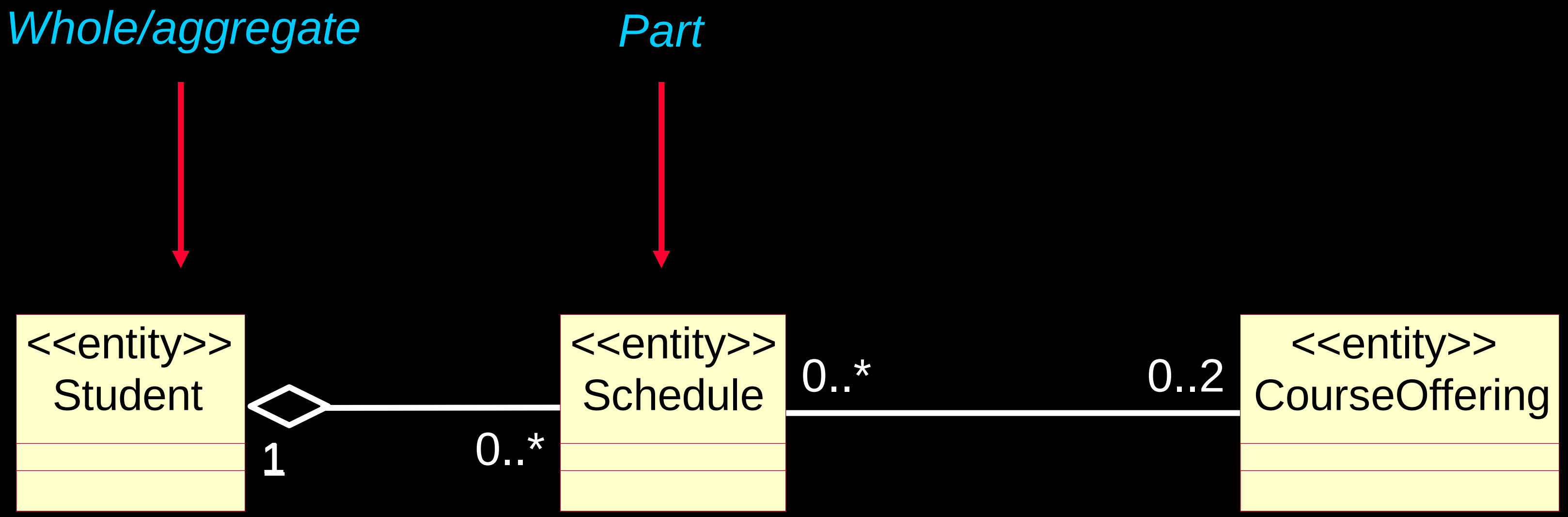
Finding Relationships



Relationship for every link!

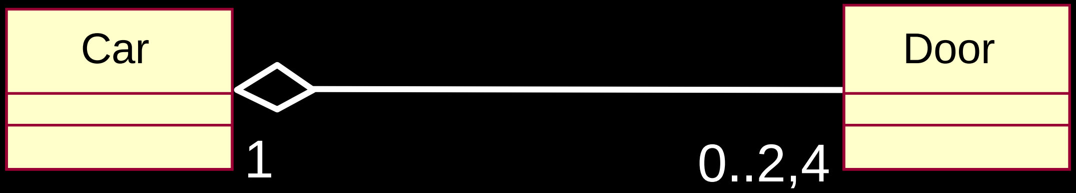
Review: What Is Aggregation?

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts

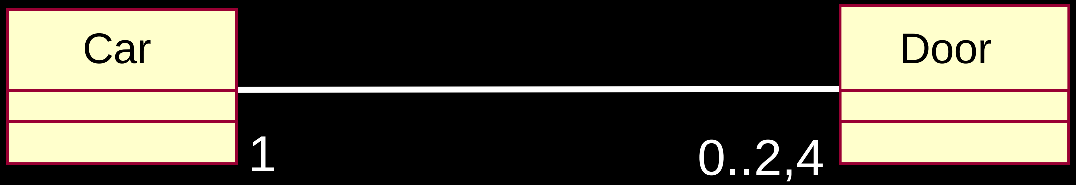


Association or Aggregation?

- If two objects are tightly bound by a whole-part relationship
  - The relationship is an aggregation.



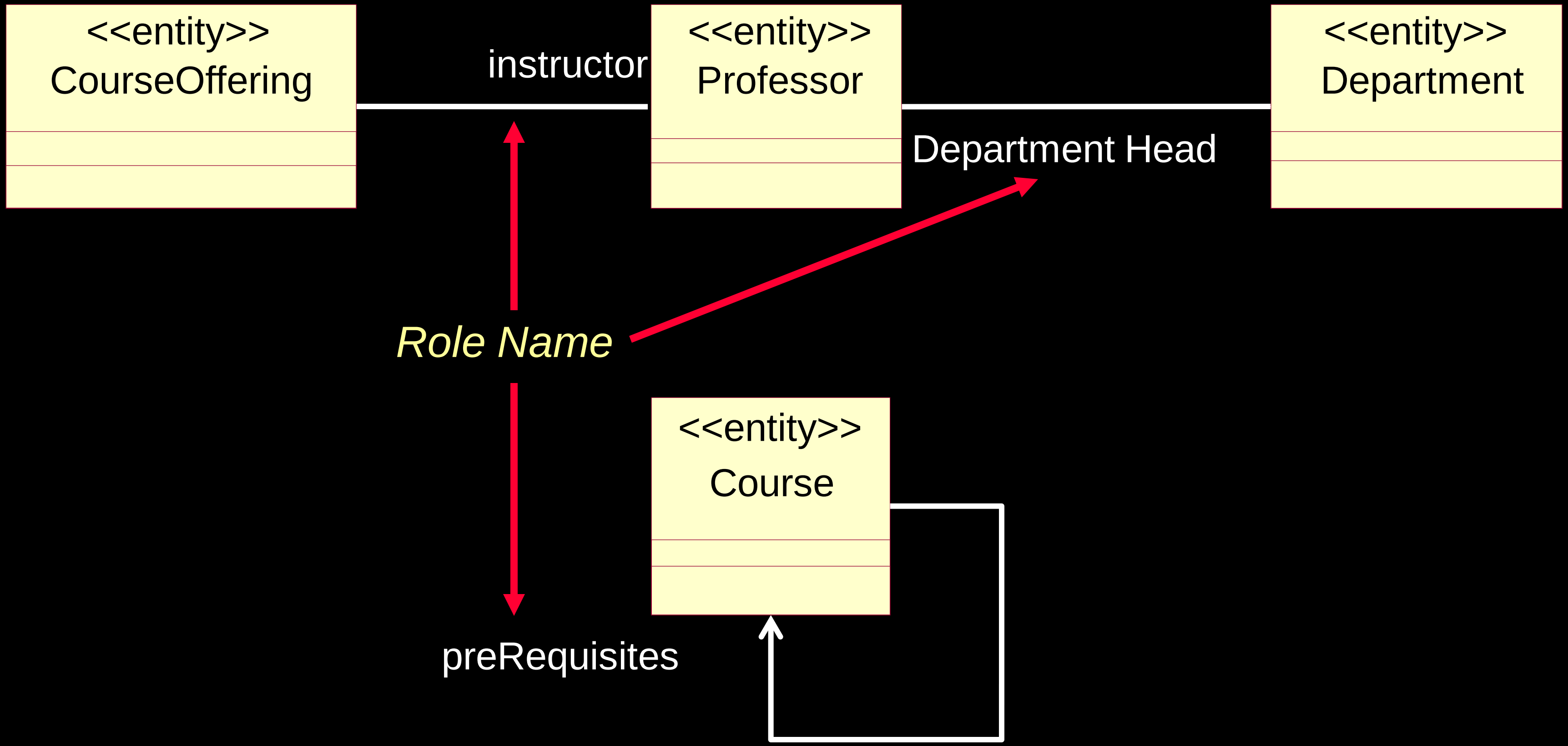
- If two objects are usually considered as independent, although they are often linked
  - The relationship is an association.



When in doubt use association

What Are Roles?

- The “face” that a class plays in the association

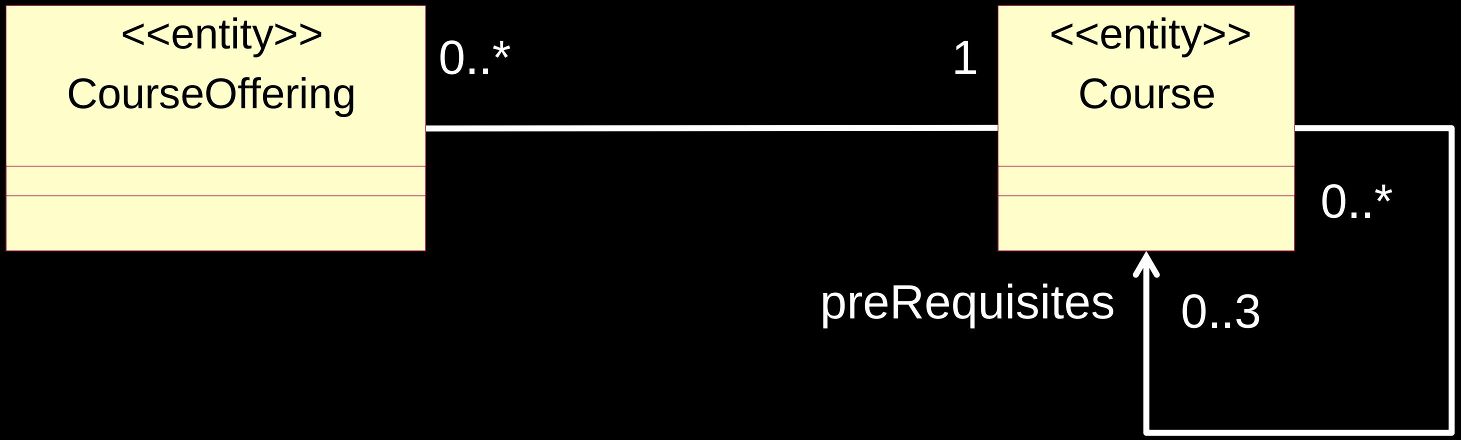


Review: Multiplicity

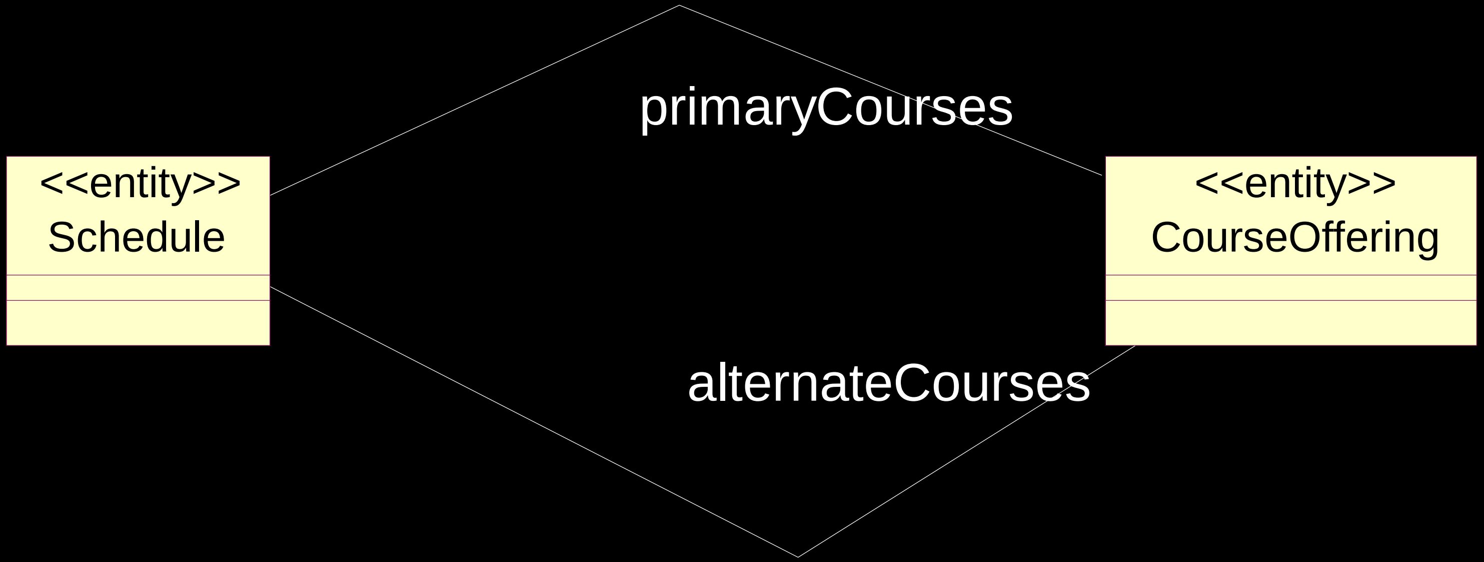
Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional scalar role)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

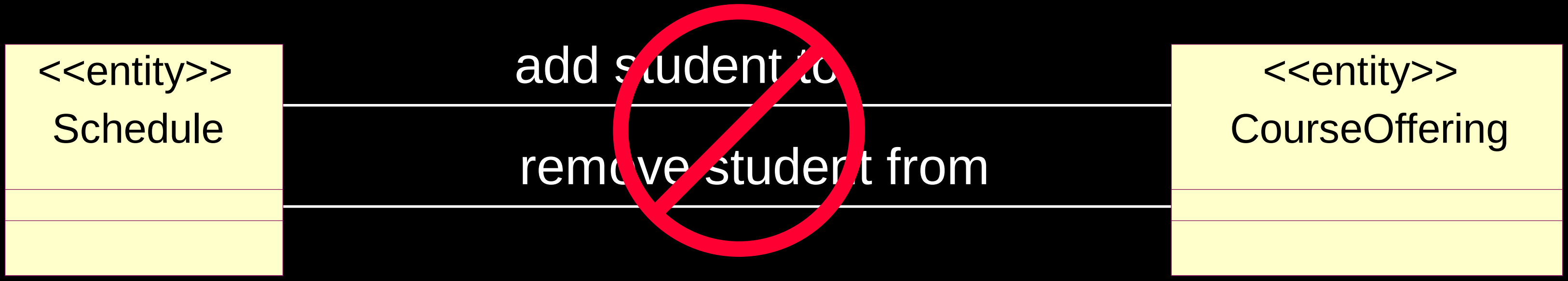
What Does Multiplicity Mean?

- Multiplicity answers two questions:
  - Is the association mandatory or optional?
  - What is the minimum and maximum number of instances that can be linked to one instance?



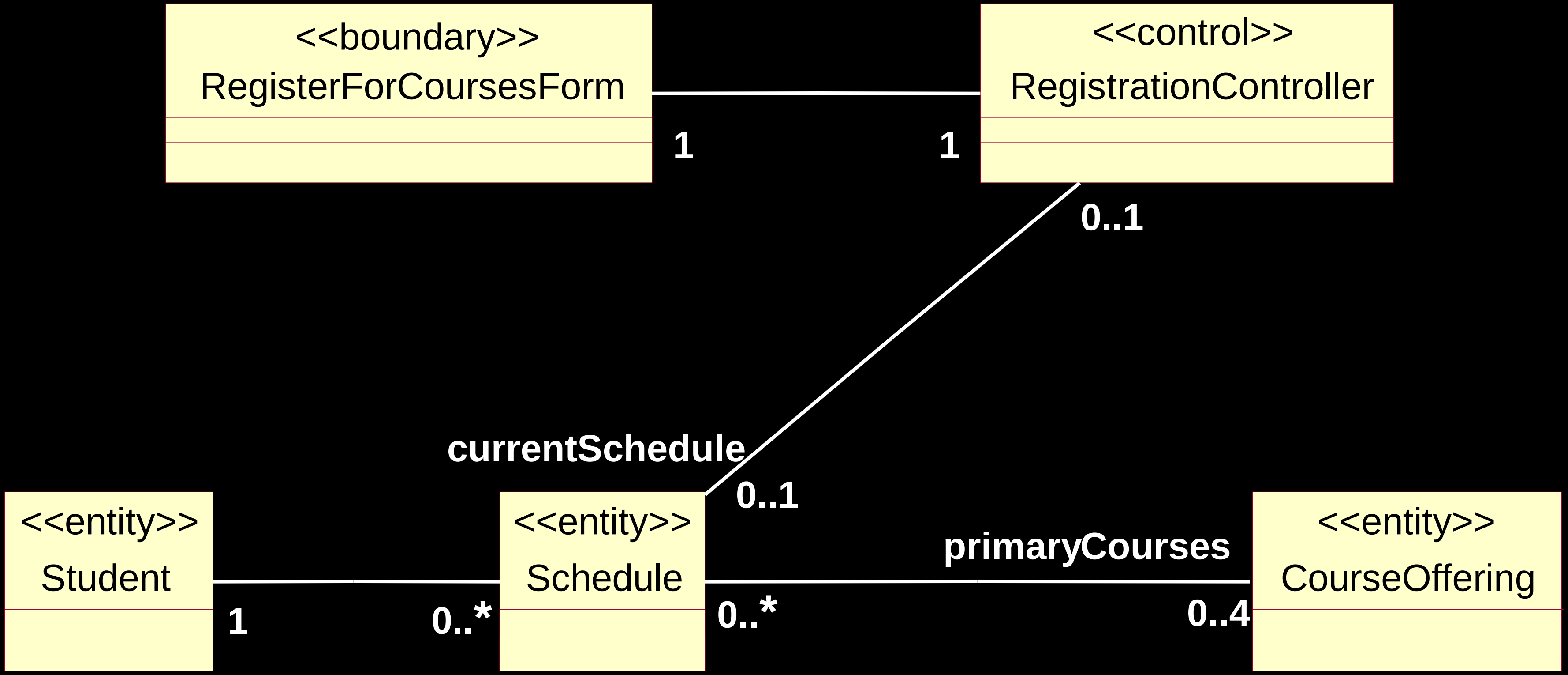
Example: Multiple Associations





Multiple associations must reflect multiple roles.

Example: VOPC: Finding Relationships



Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- For each Use-Case Realization
  - Find Classes from Use-Case Behavior
  - Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - ★ Qualify Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

Review: Why Use Analysis Mechanisms?

Oh no! I found a group of classes that has persistent data. How am I supposed to design these things if I don’t even know what database we are going to be using?

That is why we have a persistence analysis mechanism. We don’t know enough yet, so we can bookmark it and come back to it later.

Analysis mechanisms are used during analysis to reduce the complexity of analysis, and to improve its consistency by providing designers with a shorthand representation for complex behavior.

Describing Analysis Mechanisms

- Collect all analysis mechanisms in a list
- Draw a map of the client classes to the analysis mechanisms
- Identify characteristics of the analysis mechanisms

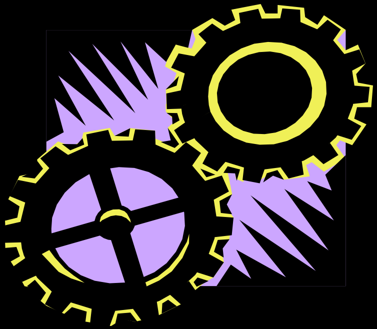
Example: Describing Analysis Mechanisms

- Analysis class to analysis mechanism map

Analysis Class	Analysis Mechanism(s)
Student	Persistency, Security
Schedule	Persistency, Security
CourseOffering	Persistency, Legacy Interface
Course	Persistency, Legacy Interface
RegistrationController	Distribution

Example: Describing Analysis Mechanisms (cont.)

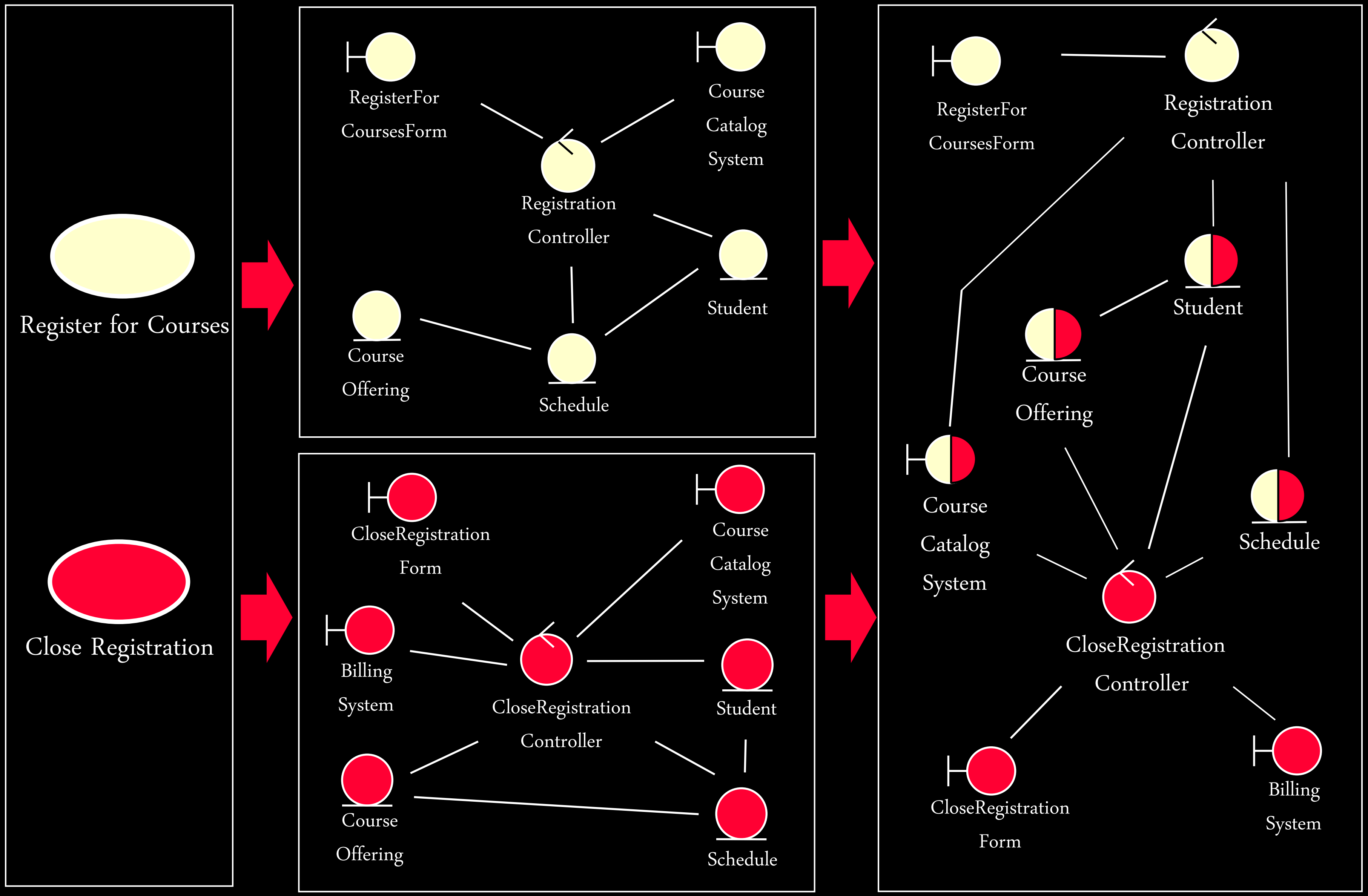
- Analysis mechanism characteristics
- Persistency for Schedule class:
  - Granularity: 1 to 10 Kbytes per product
  - Volume: up to 2,000 schedules
  - Access frequency
    - Create: 500 per day
    - Read: 2,000 access per hour
    - Update: 1,000 per day
    - Delete: 50 per day
  - Other characteristics



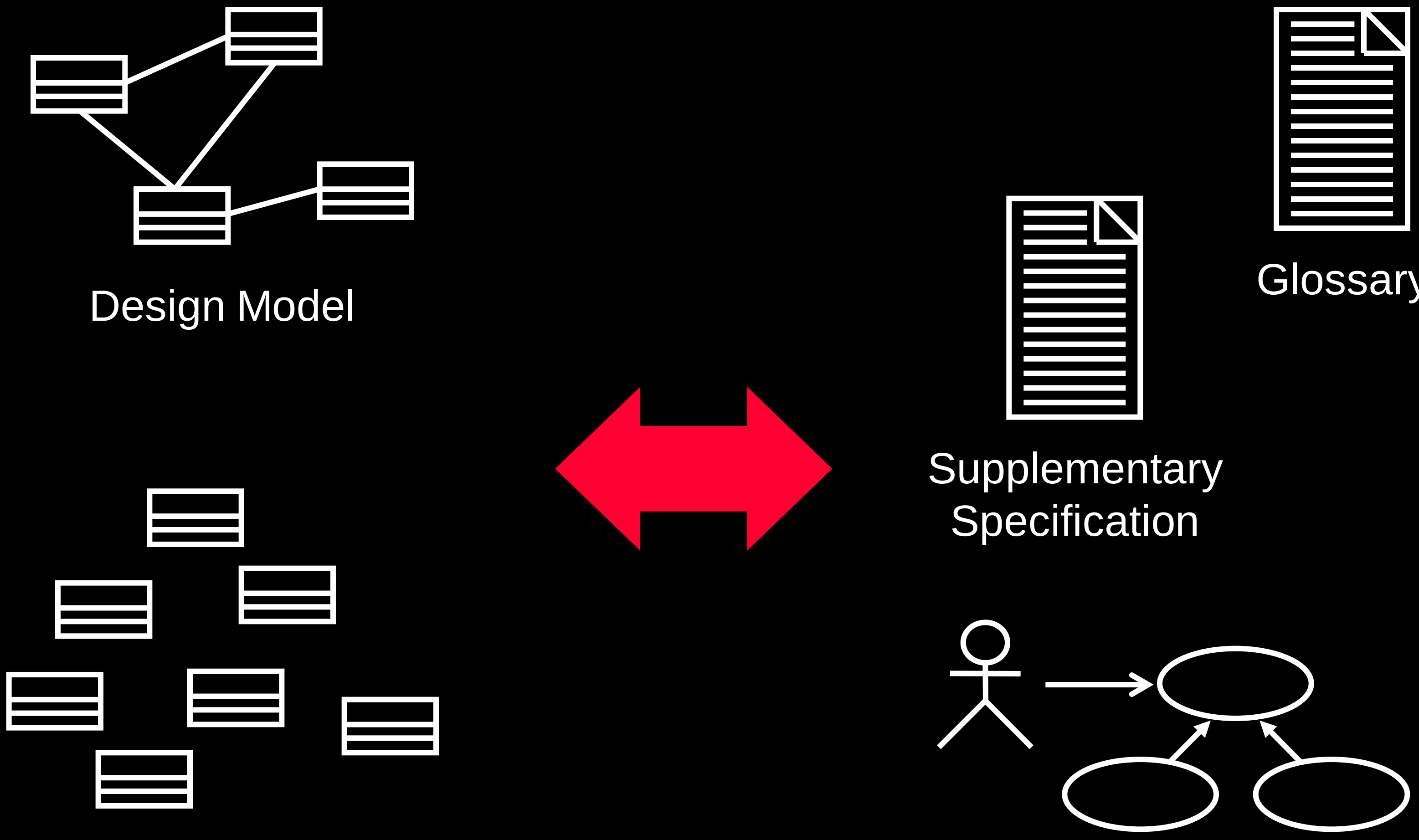
Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- For each Use-Case Realization
  - Find Classes from Use-Case Behavior
  - Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- ★ Unify Analysis Classes
- Checkpoints

Unify Analysis Classes



Evaluate Your Results



Design Model

Glossary

Supplementary Specification

Analysis Classes

Use-Case Model

Use-Case Analysis Steps

- Supplement the Use-Case Descriptions
- For each Use-Case Realization
  - Find Classes from Use-Case Behavior
  - Distribute Use-Case Behavior to Classes
- For each resulting analysis class
  - Describe Responsibilities
  - Describe Attributes and Associations
  - Qualify Analysis Mechanisms
- Unify Analysis Classes
- ★ Checkpoints

Checkpoints: Analysis Classes

- 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?

(continued)

Checkpoints: Use-Case Realizations

- 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?

Review: Use-Case Analysis

- What is the purpose of Use-Case Analysis?
- What is an analysis class? Name and describe the three analysis stereotypes.
- What is a Use-Case Realization?
- Describe some considerations when allocating responsibilities to analysis classes.
- How many Interaction diagrams should be produced during Use-Case Analysis?

Exercise: Use-Case Analysis

- Given the following:
  - Use-Case Model, especially the use-case flows of events
  - Key abstractions/classes
  - The Supplementary Specification
  - The possible analysis mechanisms

(continued)

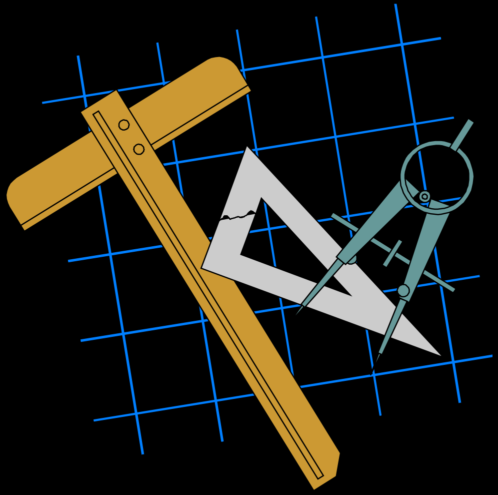
Exercise: Use-Case Analysis (cont.)

- Identify the following for a particular use case:
  - The analysis classes, along with their:
    - Brief descriptions
    - Stereotypes
    - Responsibilities
  - The collaborations needed to implement the use case
  - Analysis class attributes and relationships
  - Analysis class analysis mechanisms

(continued)

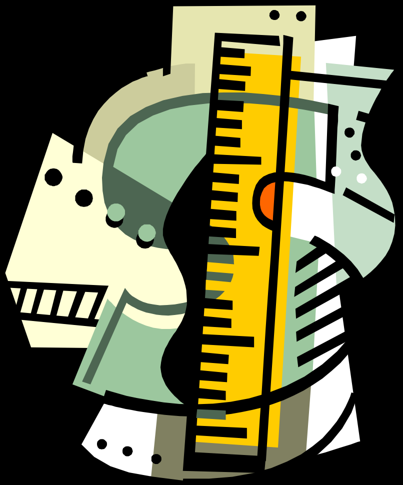
Exercise: Use-Case Analysis (cont.)

- Produce the following for a particular use case:
  - Use-Case Realization Interaction diagram for at least one of the use-case flows of events
  - VOPC class diagram, containing the analysis classes, their stereotypes, responsibilities, attributes, and relationships
  - Analysis class to analysis mechanism map



Exercise: Review

- Compare your Use-Case Realization with the rest of the class
  - Do the Interaction diagrams carry out the use-case flow of events?
  - Are the stereotypes behaving properly?
  - Is each association supported by a link?
  - Does each association have multiplicity assigned?
  - Have role names been assigned? Do they accurately represent the face the class plays in the relationship?



Payroll System