

Use Cases

COMPSCI5059 - Software Engineering

H. Gül Calikli, Ph.D.

Overview

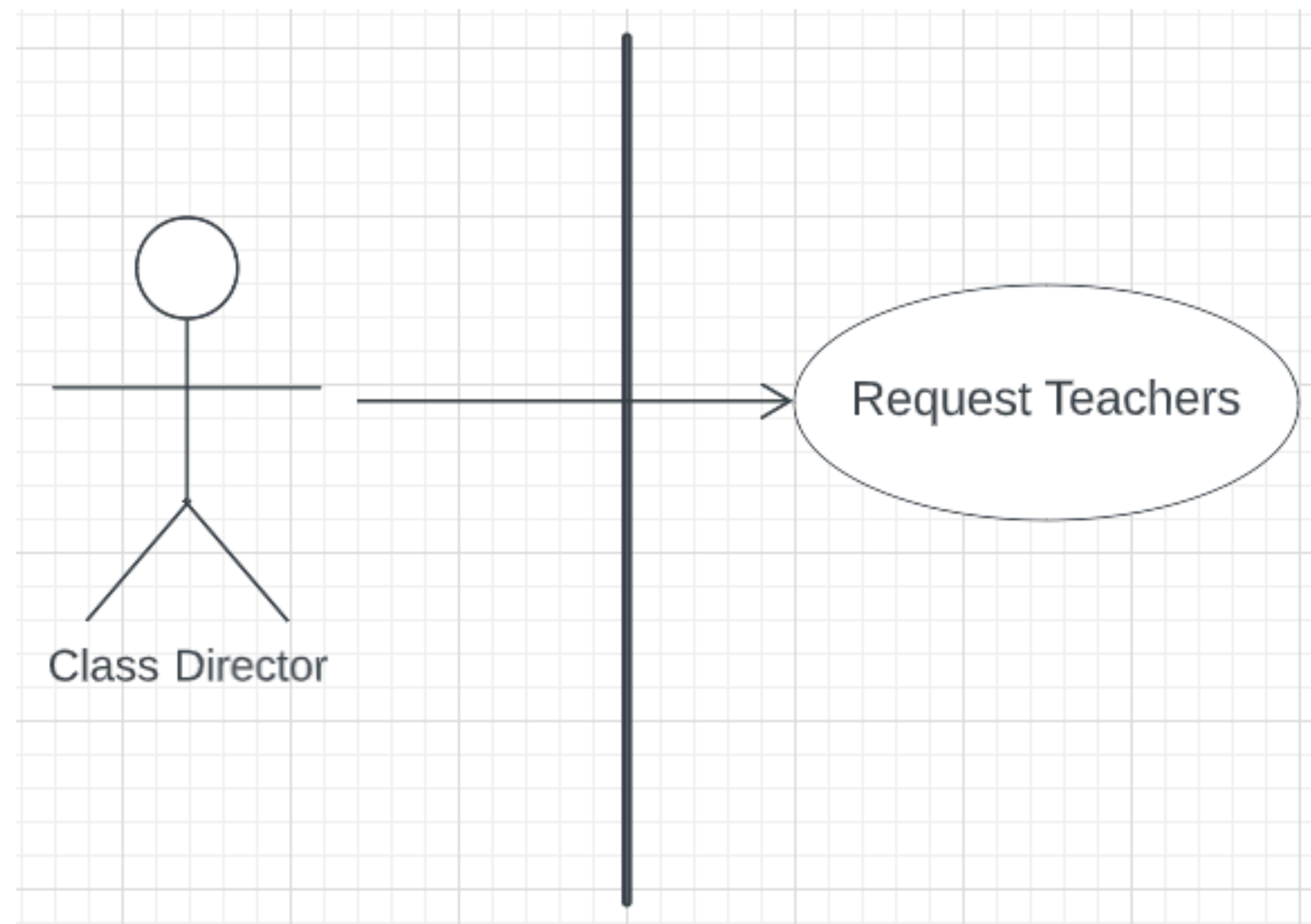
- ☐ Use Case Approach - Introduction
- ☐ Actors, Passive & Active Actors, and Actor Inheritance
- ☐ New Requirements, Non-Functional Requirements
- ☐ Prioritising Use Cases
- ☐ Scenarios, Flow of Events

Use Cases - Introduction

- The “use case” approach to developing software came before the Agile approach.
- More documentation was required.
- More time was spent to analyse and design the system before starting to code.

Use Cases - Introduction

- As an example, let's have a look at the Part Time Teachers example in Assessed Exercise 1.
- You are **not** supposed to draw any use case diagrams in your AE1 reports.

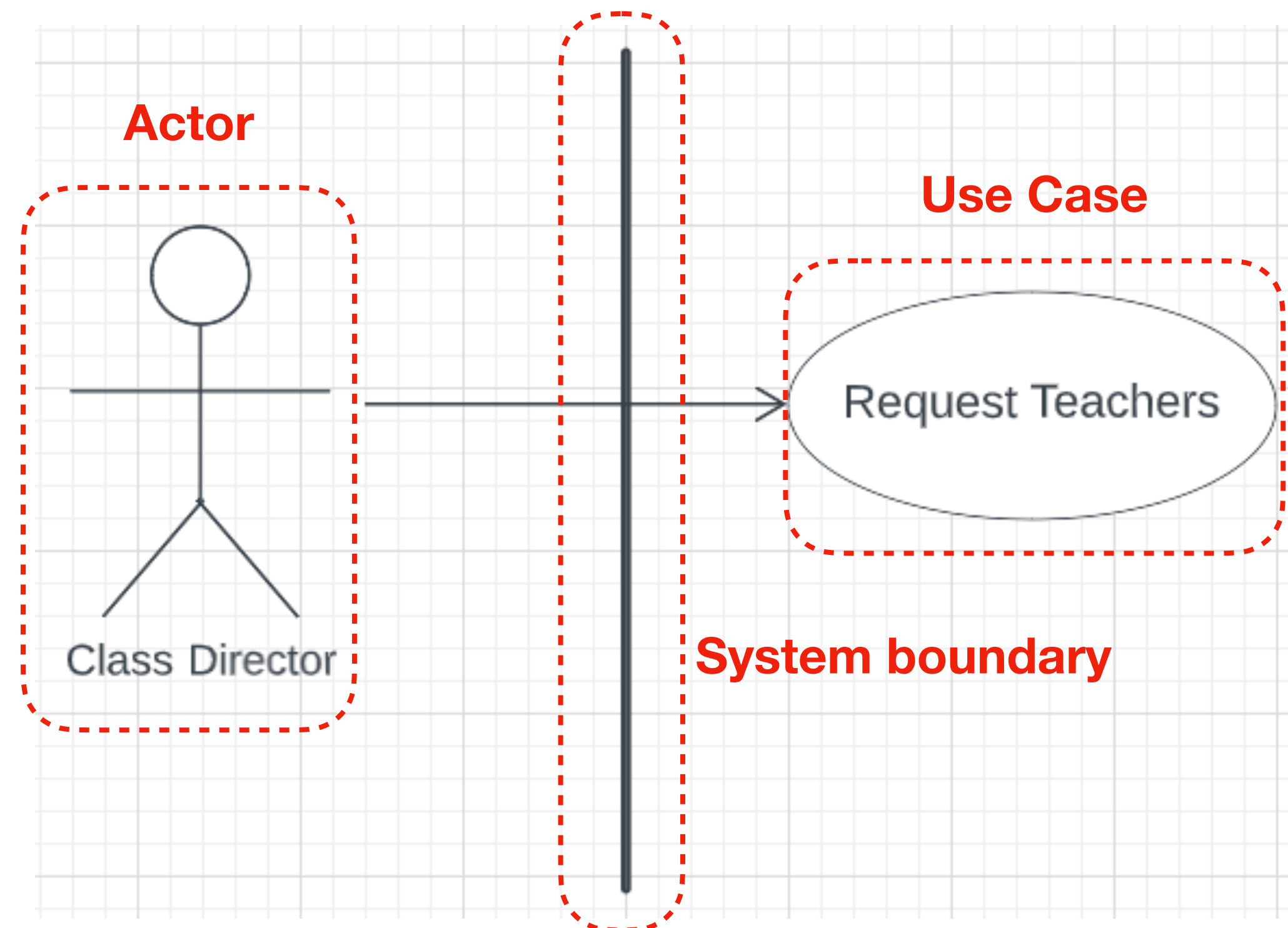


Why Learn about Use Case Approach

- The “use case” approach is old fashioned and not used any more:
 - because of excessive documentation.
- Use cases can add to an Agile approach:
 - in some case, there is too little design and documentation in Agile approaches.

Use Cases - Introduction

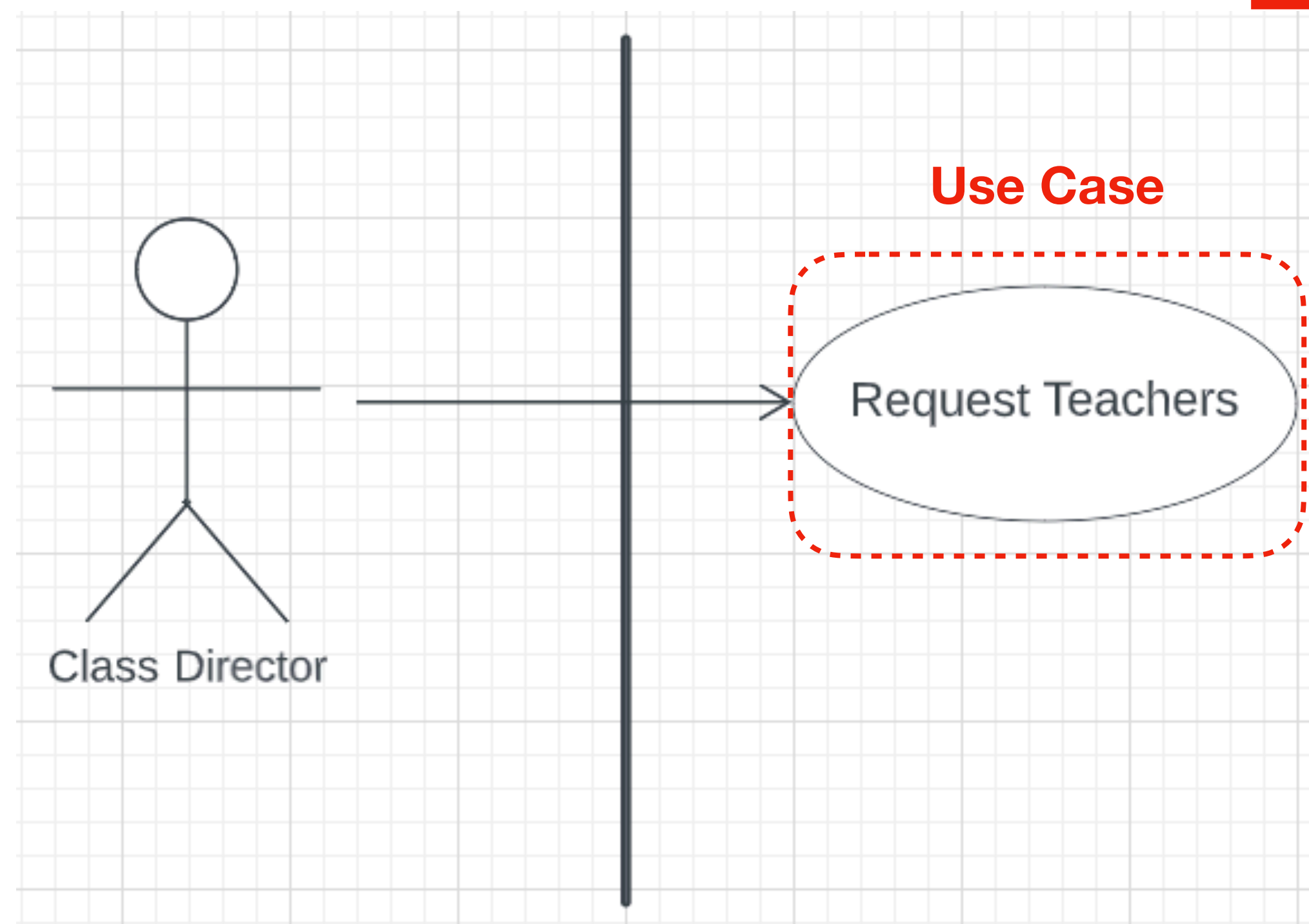
- As an example, let's have a look at the Part Time Teachers example in Assessed Exercise 1.
- You are **not** supposed to draw any use case diagrams in AE1.



Use Cases - Introduction

- A **use case** is equivalent to a **user story**.

Also, your user stories in your AE1 reports can be different than the user stories (and use cases) presented in this lecture. There are many different ways of requirements elicitation for AE1.

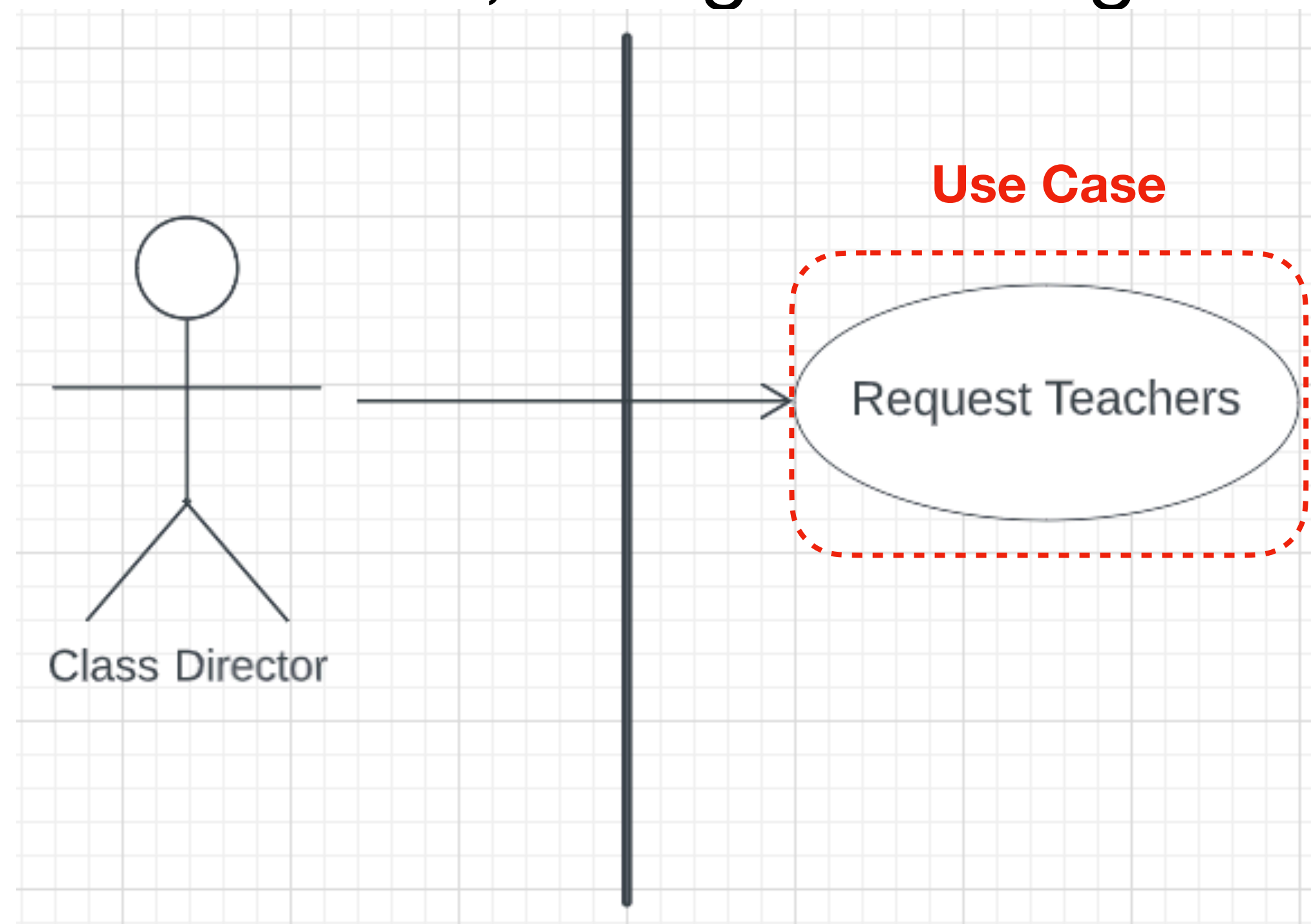


User Story

As a class director, I want to request teachers, so that ...

Use Cases - Introduction

- A **use case** is equivalent to a **user story**.
- We find the use cases by looking at the actor and seeing what activity they initiate.
- Use cases should be short, doing one thing well to prevent ambiguity.

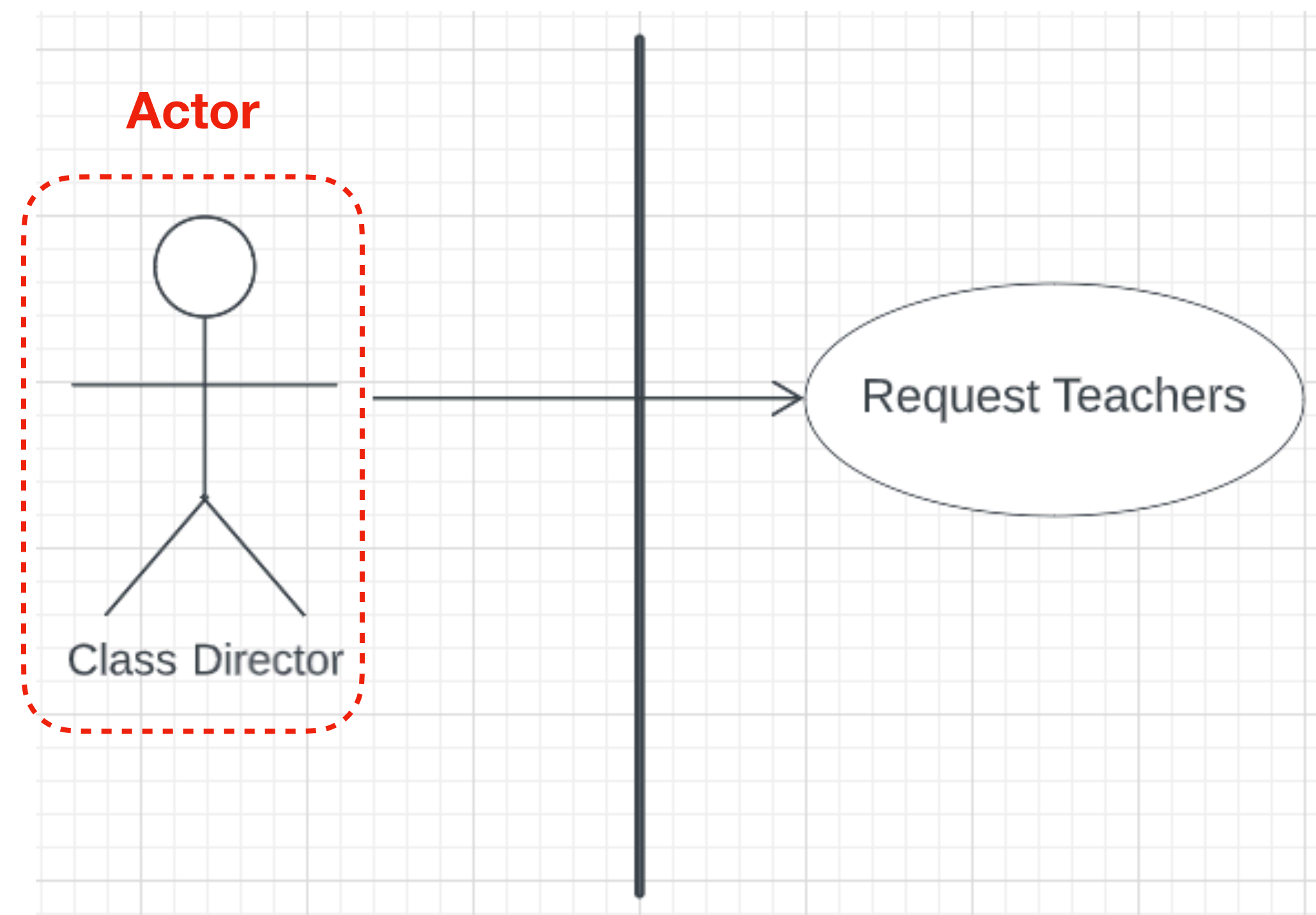


User Story

As a class director, I want to request teachers, so that ...

Use Cases - Introduction

- An **actor** is equivalent to a **user role**.

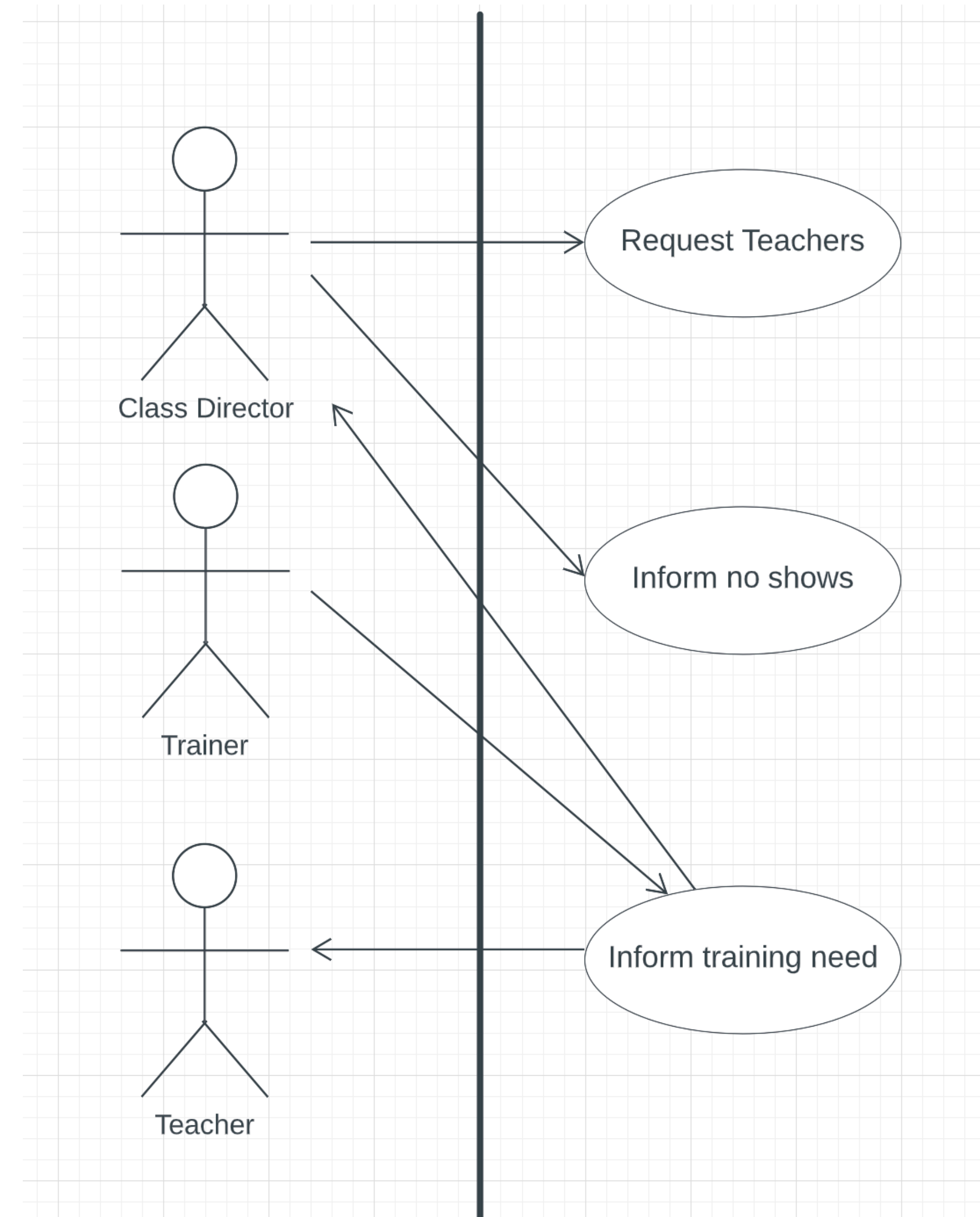


User Story

As a **class director**, I want to request teachers, so that ...

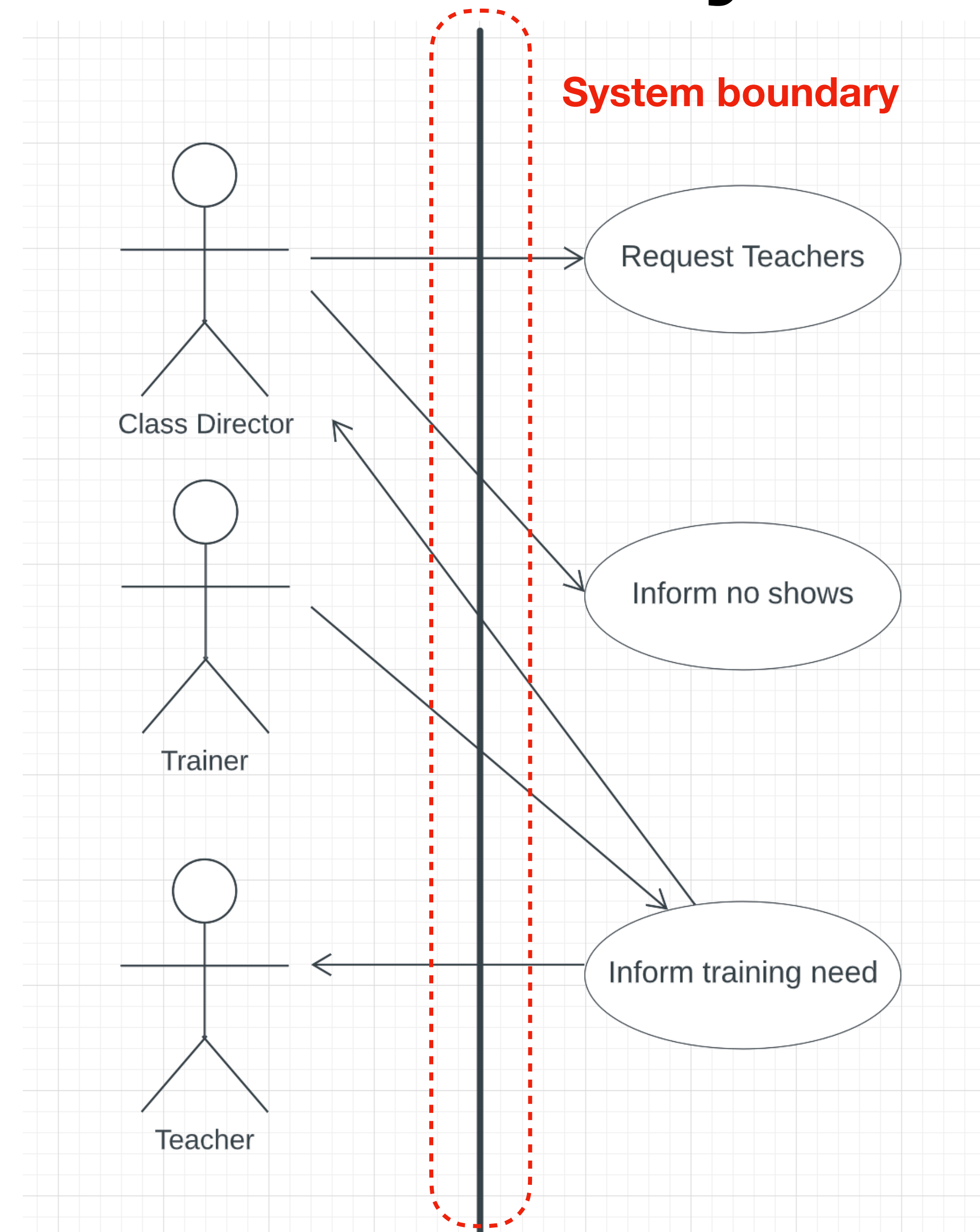
Use Case Diagram

- All actors and use cases are combined in a **use case diagram** to give the big picture.



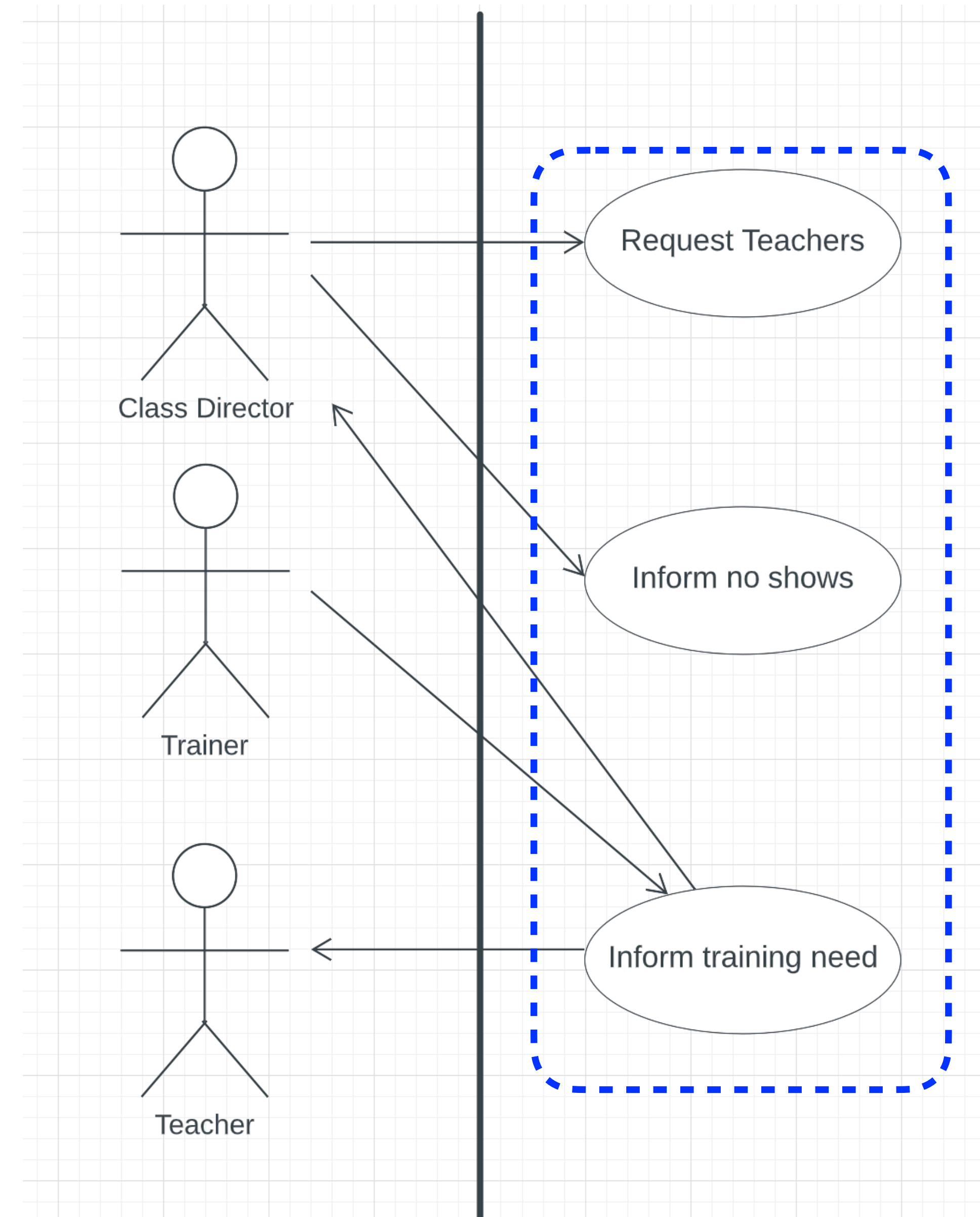
Defining the System Boundary

- Once we have an idea about what the system will do, we need to identify the **system boundary**.
- We discover the system boundary by describing the actors and use cases.



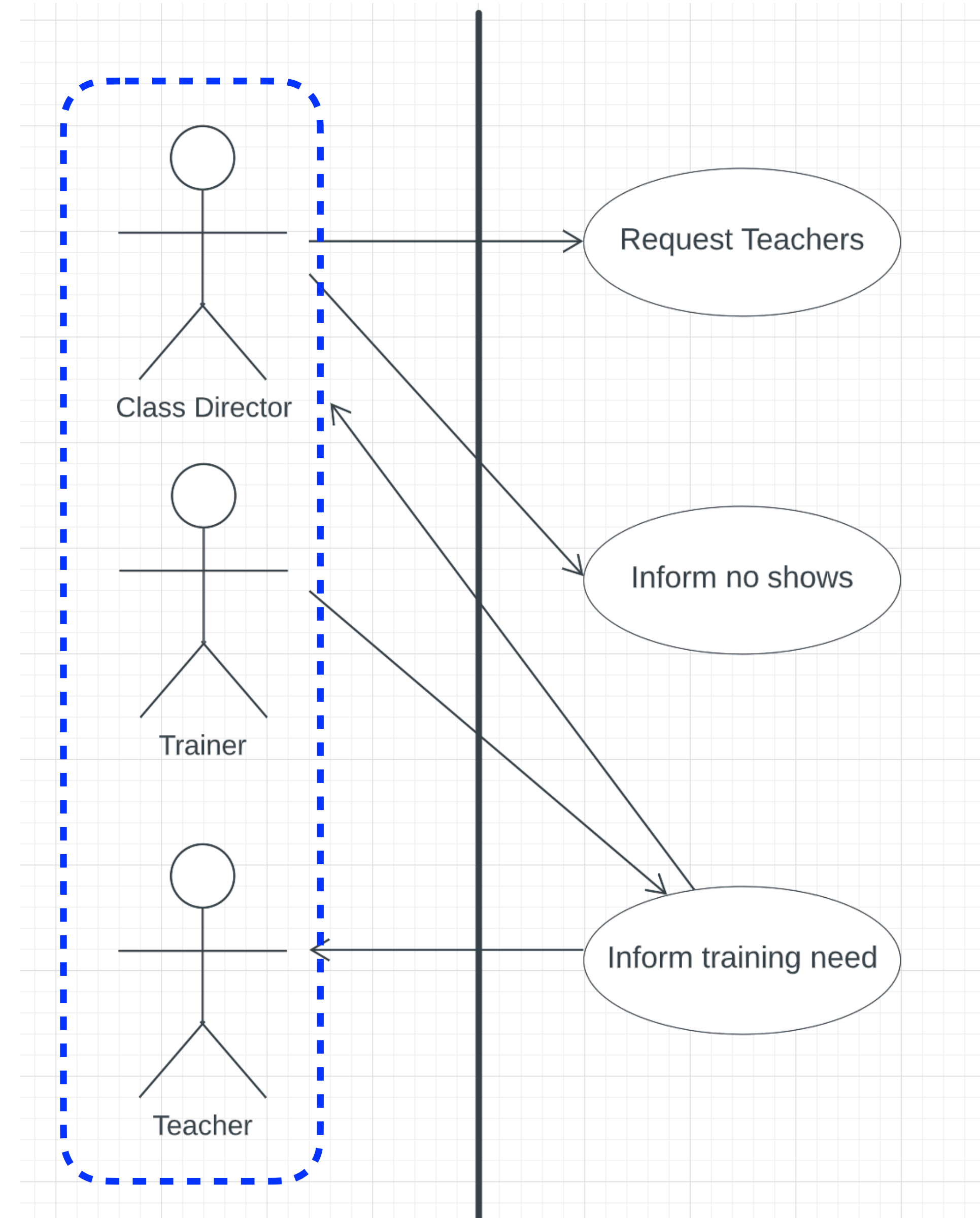
Defining the System Boundary

- If something is **inside** the system boundary then we have to create it (e.g., develop that functionality).



Defining the System Boundary

- If something is **outside** the system boundary then it is already there, then what we create will just have to interact with it.
- Objects outside the system are called **actors**.



Overview

☒ Use Cases - Introduction

☐ Actors, Passive & Active Actors, Actor Inheritance, 

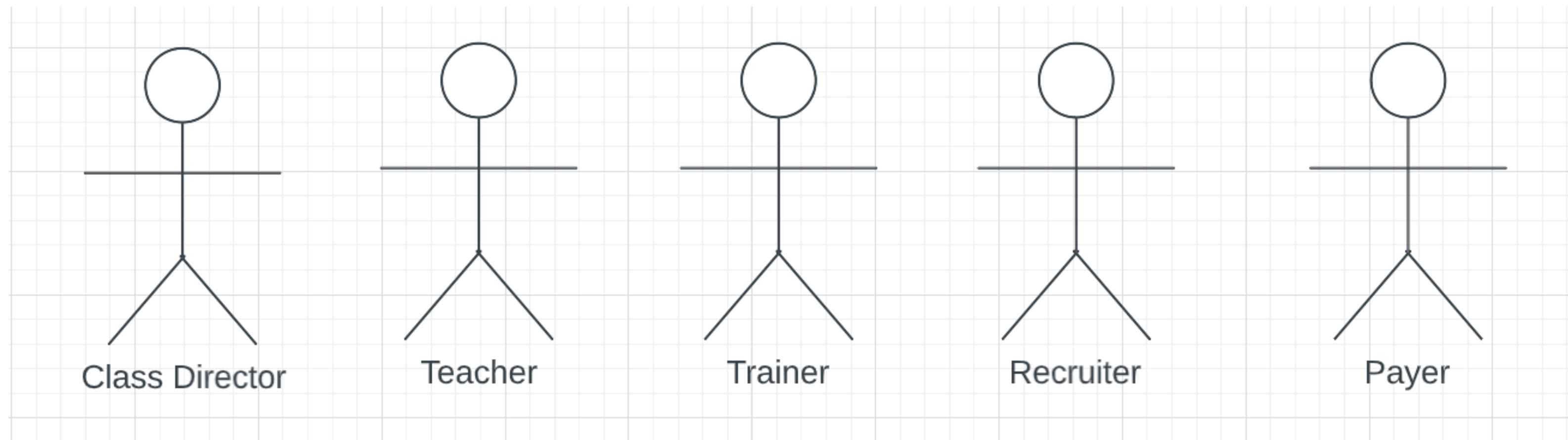
☐ New Requirements, Non-Functional Requirements

☐ Prioritising Use Cases

☐ Scenarios, Flow of Events

Actors

- Each actor needs a short description defining their role in the system.



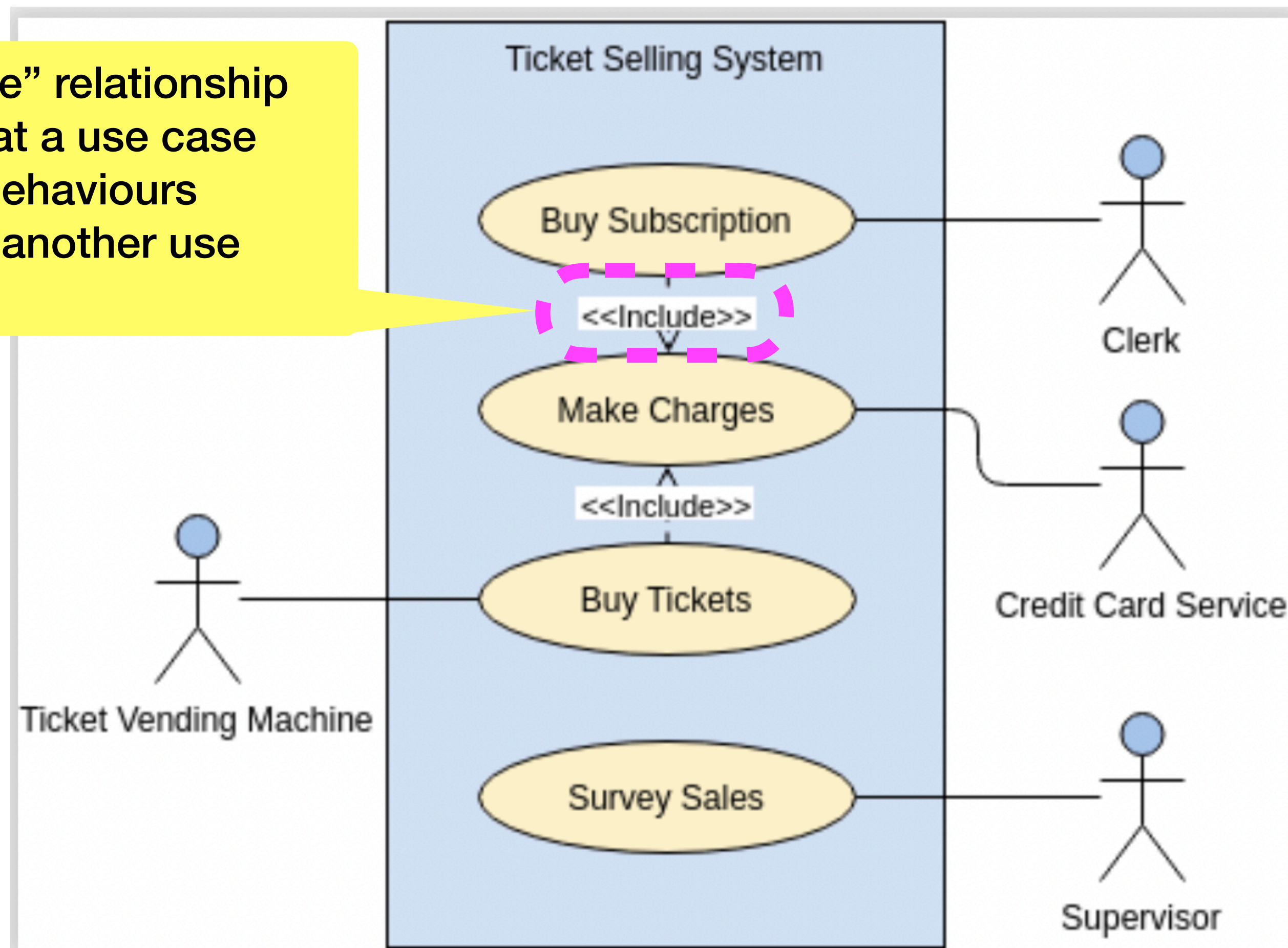
Some Examples:

- **Class director:** A person who requires part-time teachers for his class.
- **Recruiter:** Finds people to do teaching
- **Payer:** Organises payment for teachers
- **Trainer:** Trains part-time teaching staff.

Actors

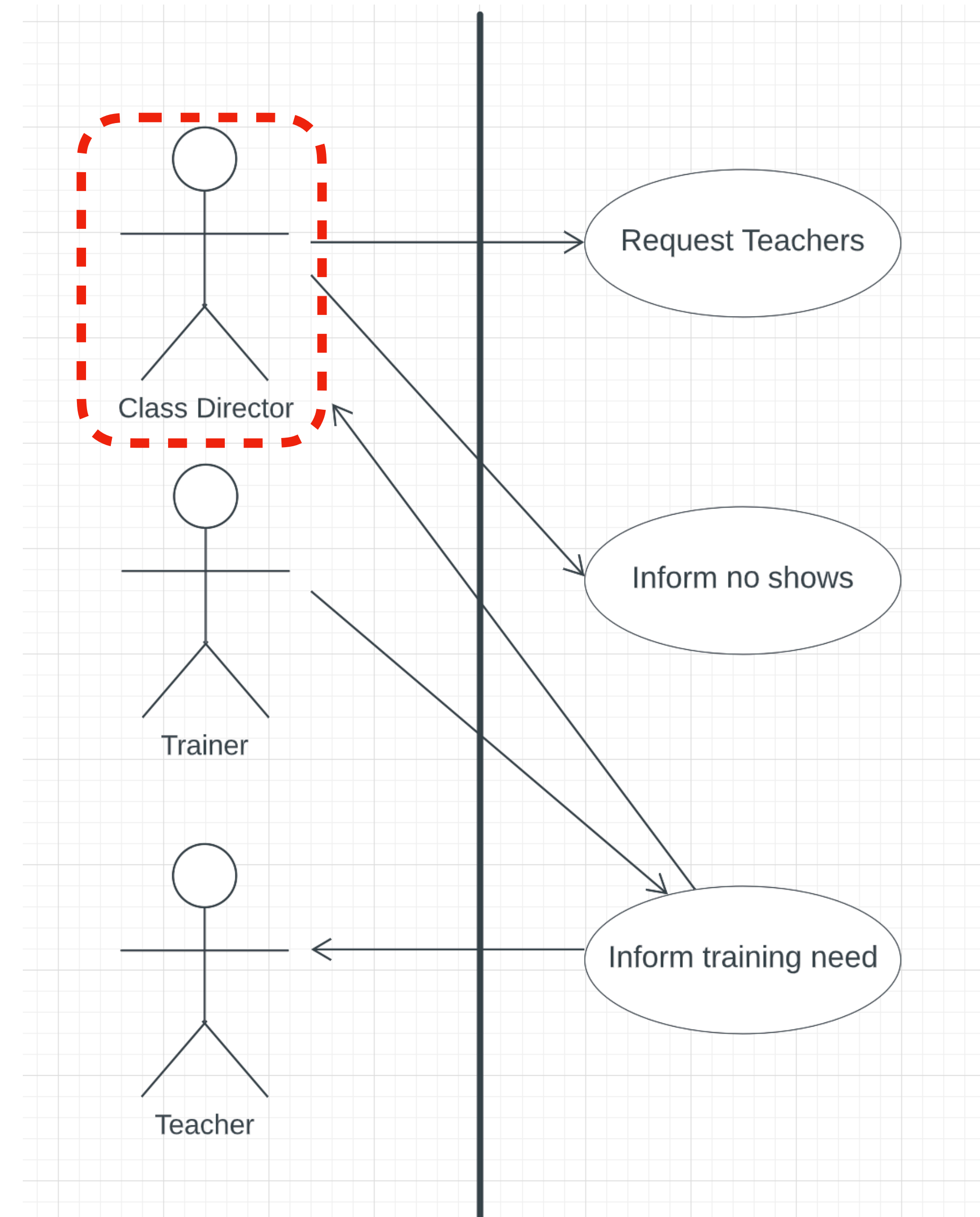
- Besides people, actors can be other pieces of software, hardware or databases.

An “include” relationship defines that a use case contains behaviours defined in another use case.



Actors

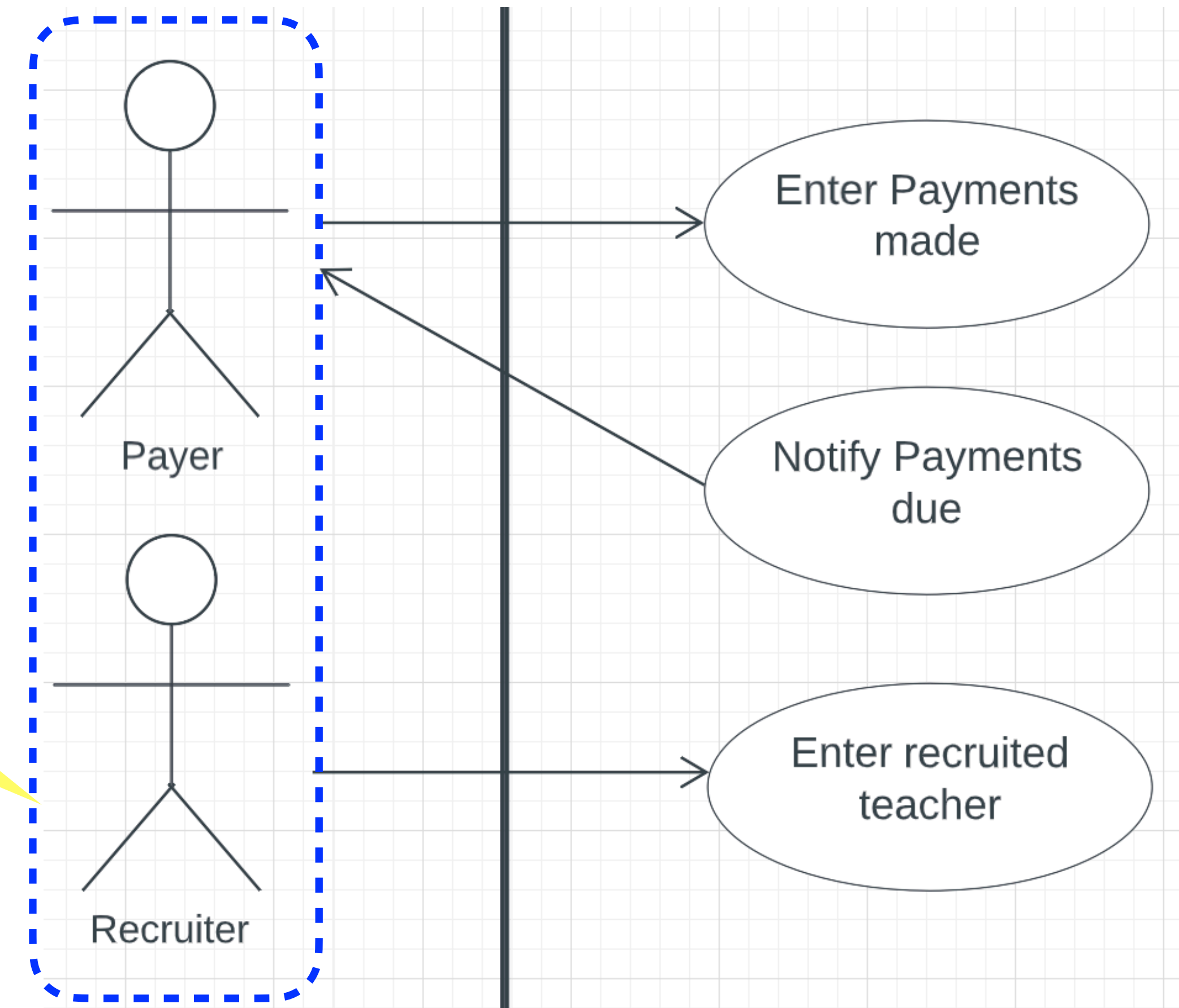
- Several different people may all perform the same role, hence be represented by a single actor.
- There will usually be several **class directors**, one for each class.
- There will be a **single** “Class Director” actor in the use case diagram since all class directors interface with our system in the same way.



Actors

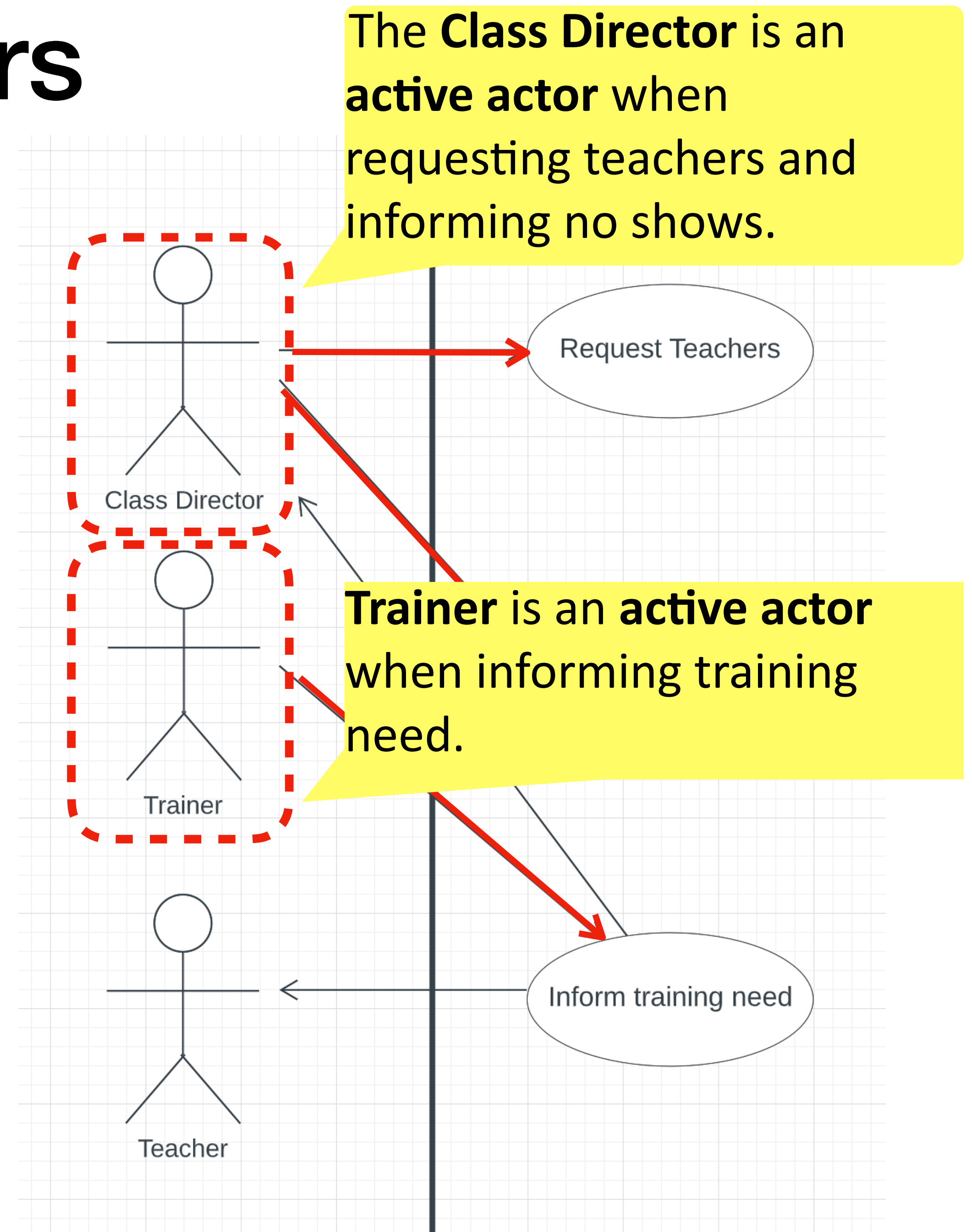
- One person might take on different roles and hence be represented by several actors.

One person may recruit people and also pay them.



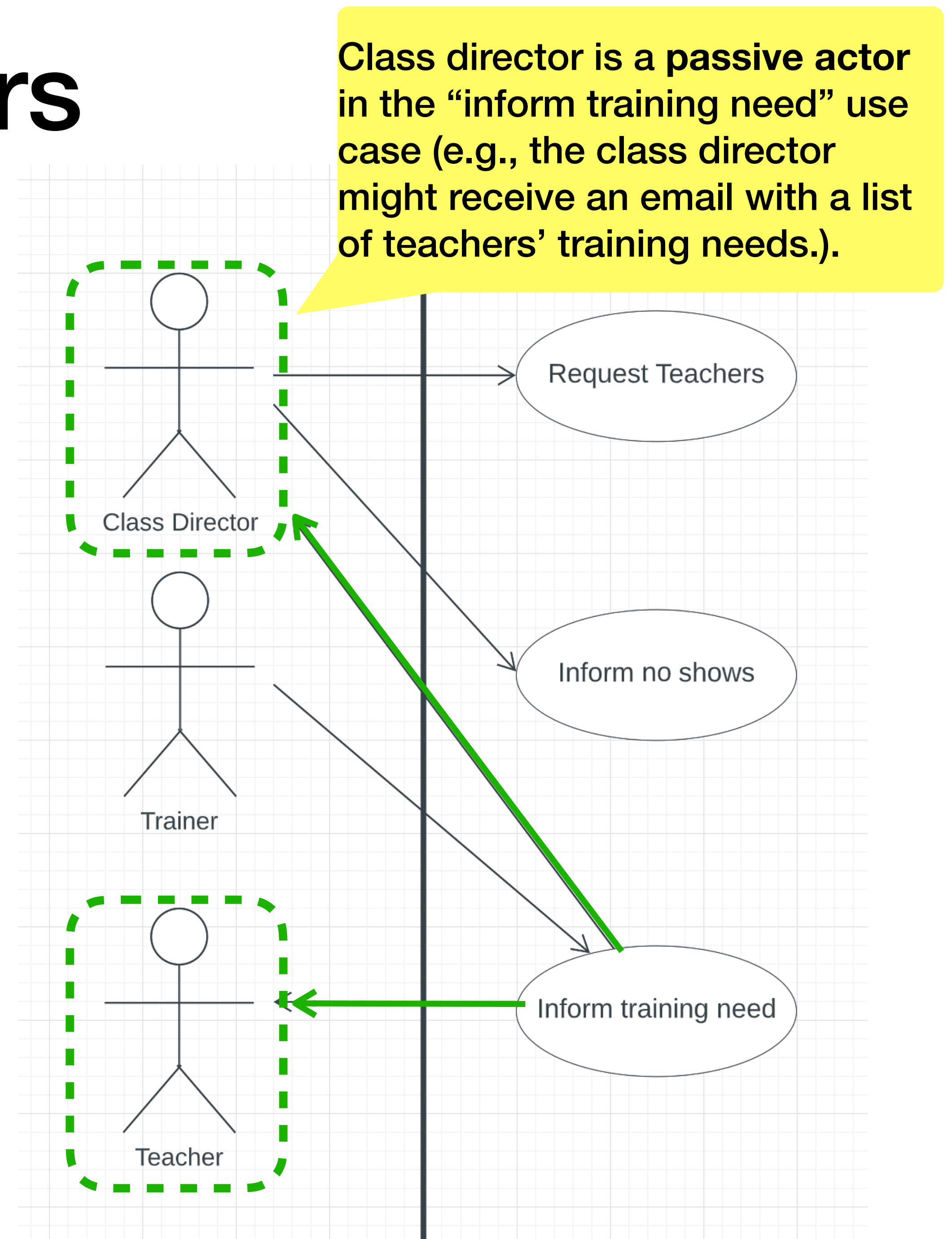
Actors

- **Active actors** will generate the use cases.
- If the actor is a person, think of him or her sitting down at a computer and using the software.



Actors

- **Passive actors** are actors that are actors that respond to activities.
- Passive actor is contacted by the use case and does not initiate any activity.

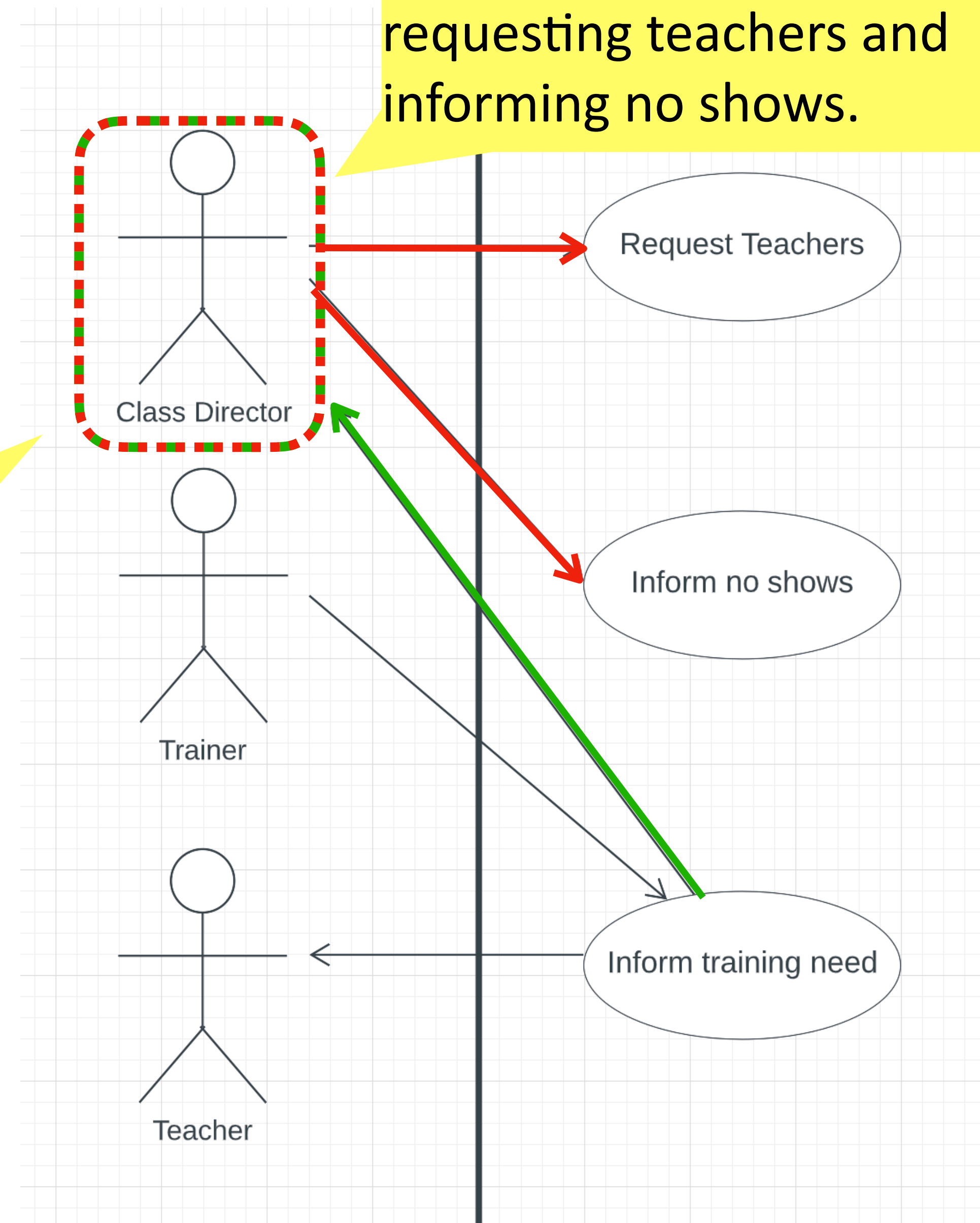


Actors

- An actor can be a **passive actor** for some use cases and an **active actor** other use cases.

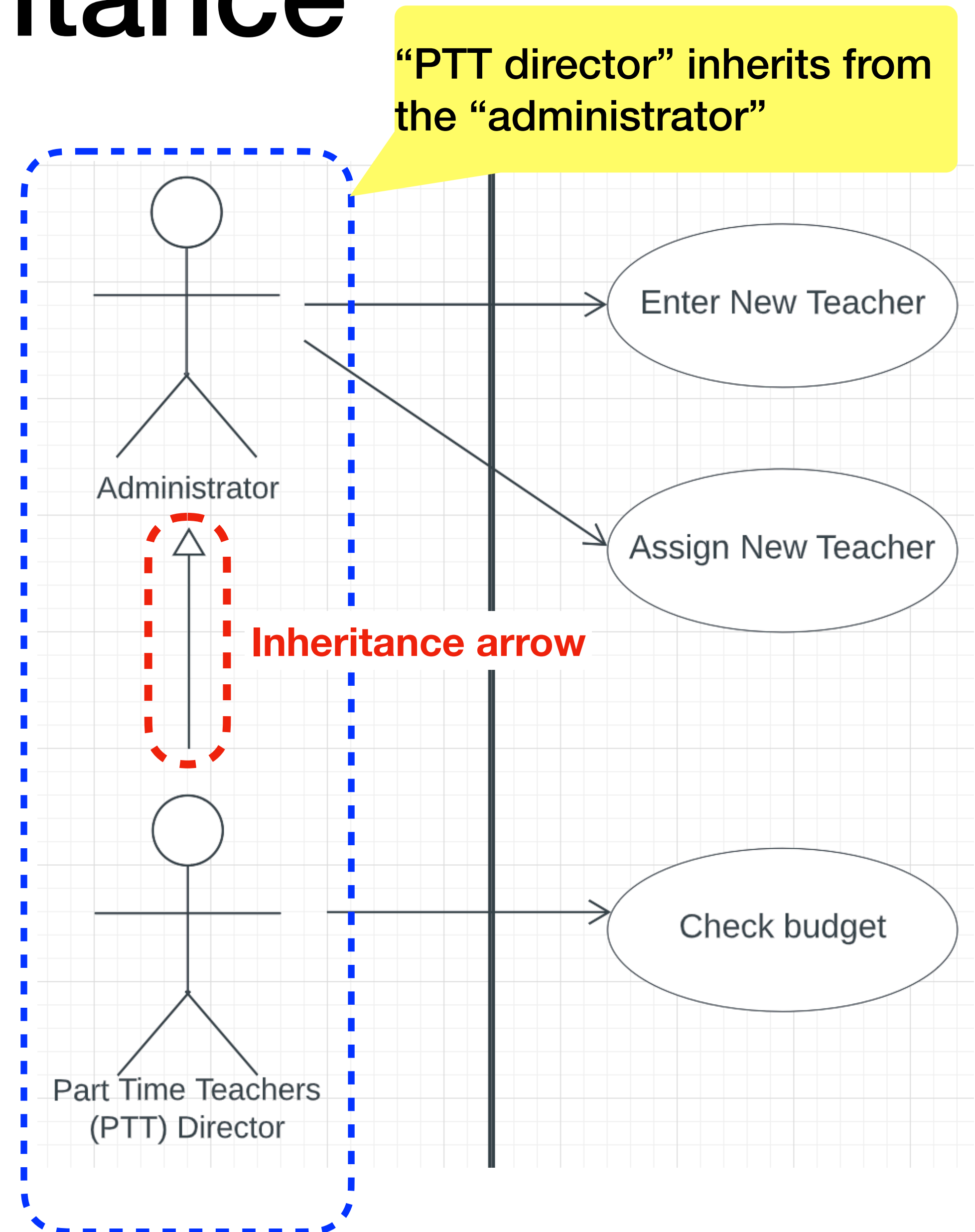
Class director is a **passive actor** in the “inform training

The **Class Director** is an **active actor** when requesting teachers and informing no shows.



Actor Inheritance

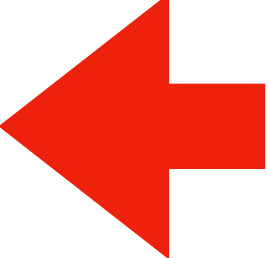
- Two actors can be similar but also have differences as well.
- In such a case, we can simplify the use case diagram by saying that one actor inherits from another actor.



Overview

☒ Use Cases - Introduction

☒ Actors, Passive & Active Actors, Actor Inheritance,

☐ New Requirements, Non-Functional Requirements 

☐ Prioritising Use Cases

☐ Scenarios, Flow of Events

New Requirements

- If we discover new requirements, we should ask the following questions:
 - Are these requirements necessary for the system?
 - Are they something our system would naturally do?
 - Can they be handled by one of the existing actors?
 - How do they affect our current risk analysis?

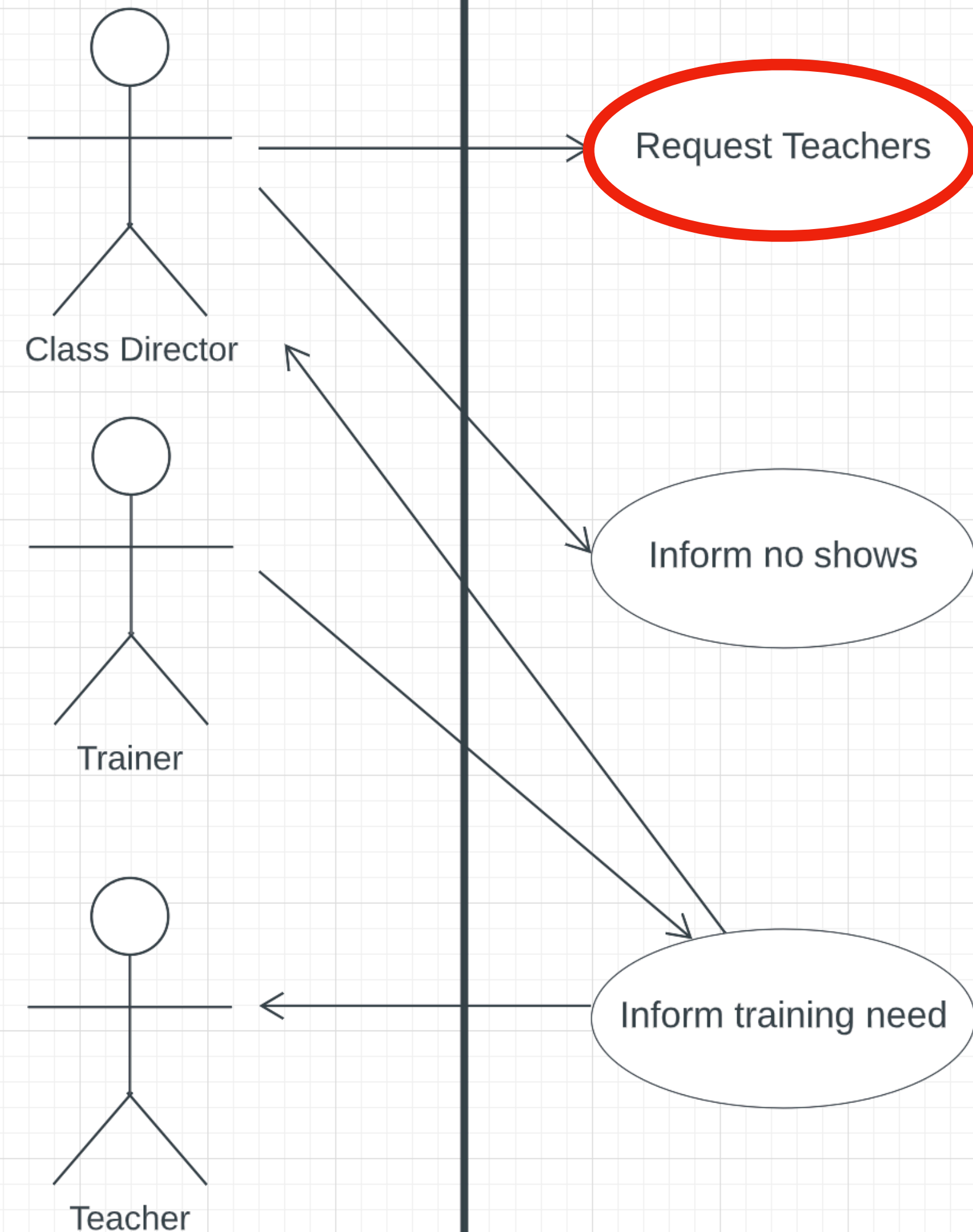
Non-Functional Requirements

- **Use cases** are **functional requirements** and define system functionality.
- Use cases are **not** a good way of defining non-functional requirements (e.g., system security, performance).
- Non-functional requirements can apply to a single use case or the whole system.

Non-Functional Requirements - Example

- *“The class director must give at least one week’s notice of teaching requirements, to allow for recruitment.”*

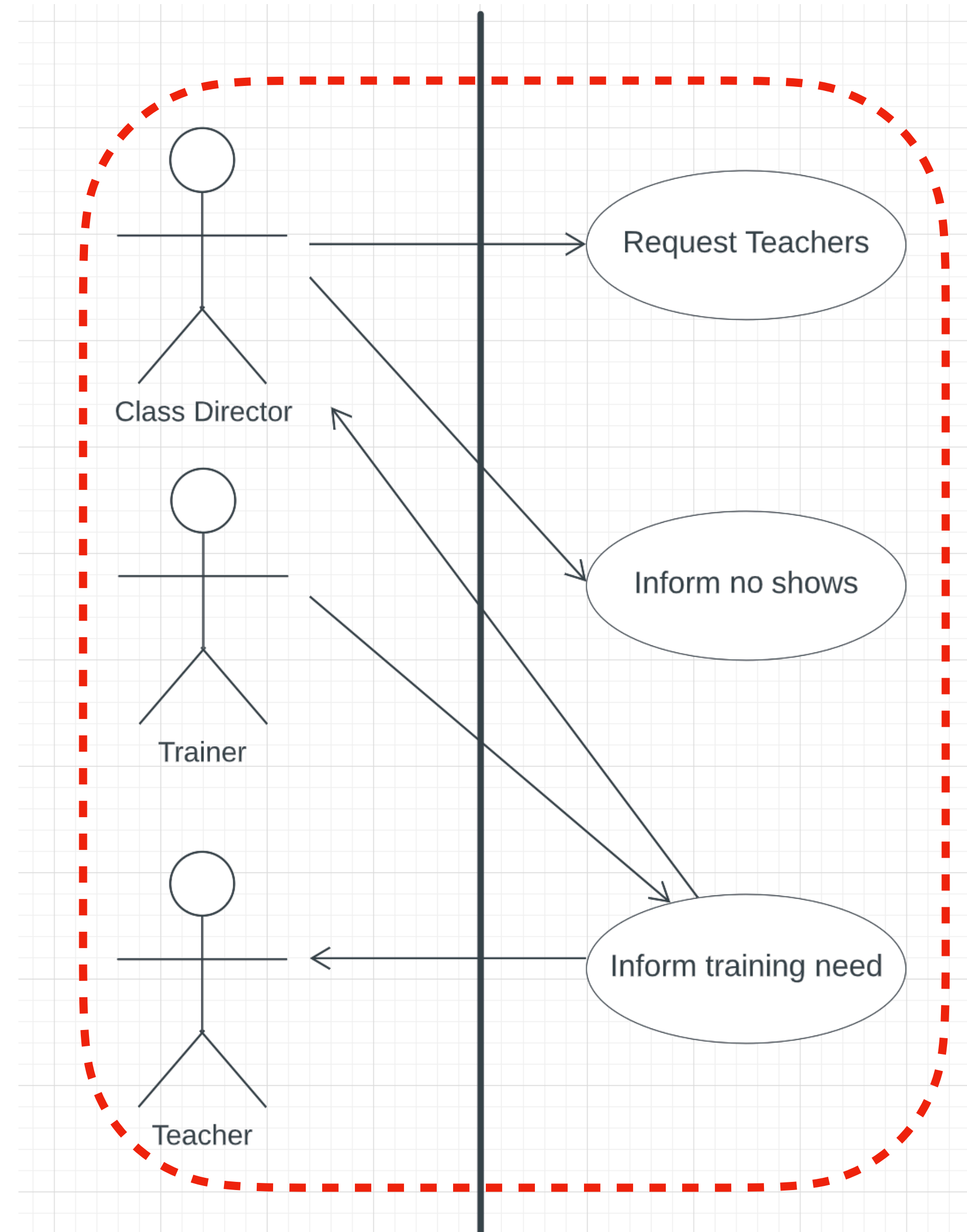
This is a non-functional requirement for the **“Request Teachers”** use case.



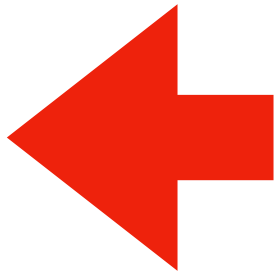
Non-Functional Requirements - Example

- *“Information stored must always be kept up-to-date and consistent at the end of each day.”*

This is a **system-wide** non-functional requirement.



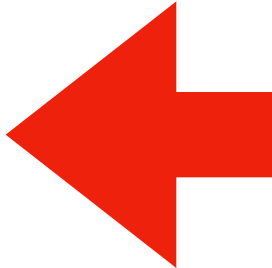
Overview

- ☒ Use Cases - Introduction
- ☒ Actors, Passive & Active Actors, Actor Inheritance,
- ☒ New Requirements, Non-Functional Requirements
- ☐ Prioritising Use Cases 
- ☐ Scenarios, Flow of Events

Prioritising Use Cases

- Use the MoSCoW approach that has the following prioritisation categories:
 - Must have
 - Should have
 - Could have
 - Would like to have

Overview

- ☒ Use Cases - Introduction
- ☒ Actors, Passive & Active Actors, Actor Inheritance,
- ☒ New Requirements, Non-Functional Requirements
- ☒ Prioritising Use Cases
- ☐ Scenarios, Flow of Events 

Flow of Events

- The most common methods for writing the flow of events:
 - Informal text
 - Numbered steps
 - Activity diagrams

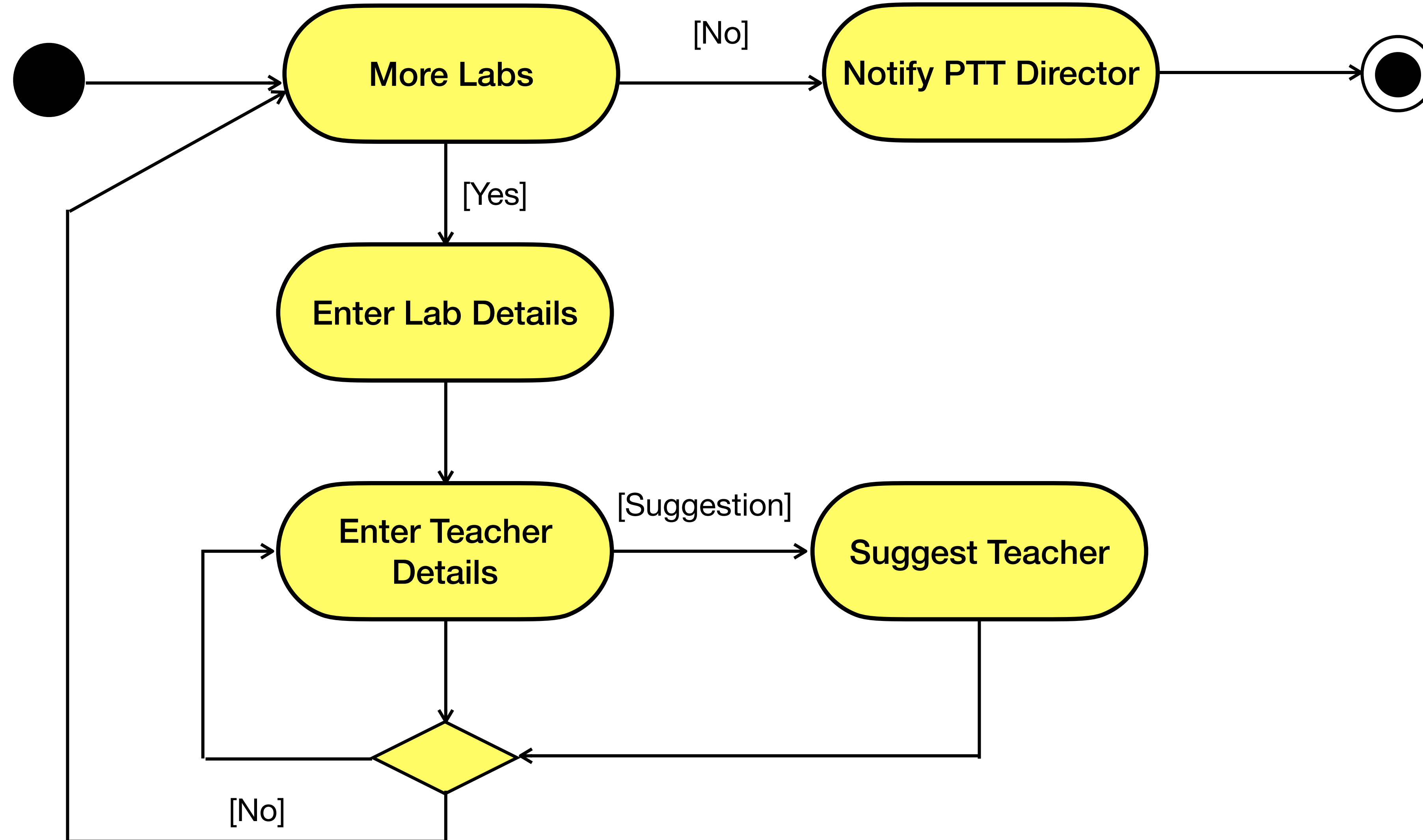
“Request Teacher” Flow of Events: Informal Text

- Rob accesses the system and brings up class 1.
- He enters the lab for Tuesday 10-12 in semester 1.
- He enters the need for a tutor and an undergraduate demonstrator.
- He enters the Lab for Wednesday 3-5 in semester 1.
- He enters the need for a tutor and an undergraduate demonstrator.
- He suggests Jeremy as the tutor for this lab.

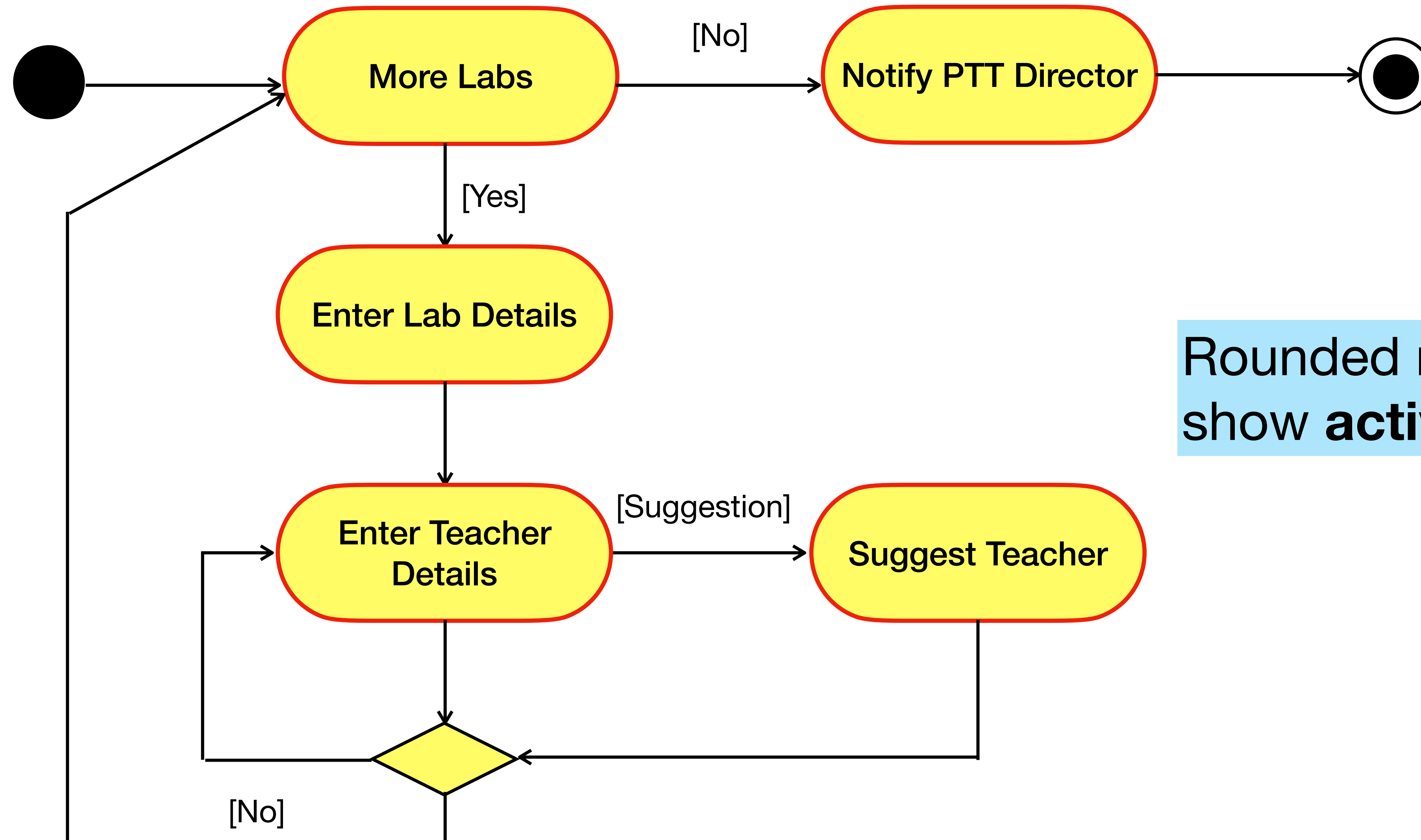
“Request Teacher” Flow of Events: Numbered Steps

- 1.** For each lab associated with the class, enter class and lab name, day and time and weeks during which it will run.
- 2.** Enter skills level required for each person. A lab may be staffed by more than one person. Skills levels are tutor, graduate demonstrator, undergraduate demonstrator.
- 3.** Enter suggested teachers, to help the recruiter.
- 4.** Notify PTT Director.

“Request Teacher” Flow of Events: Activity Diagram

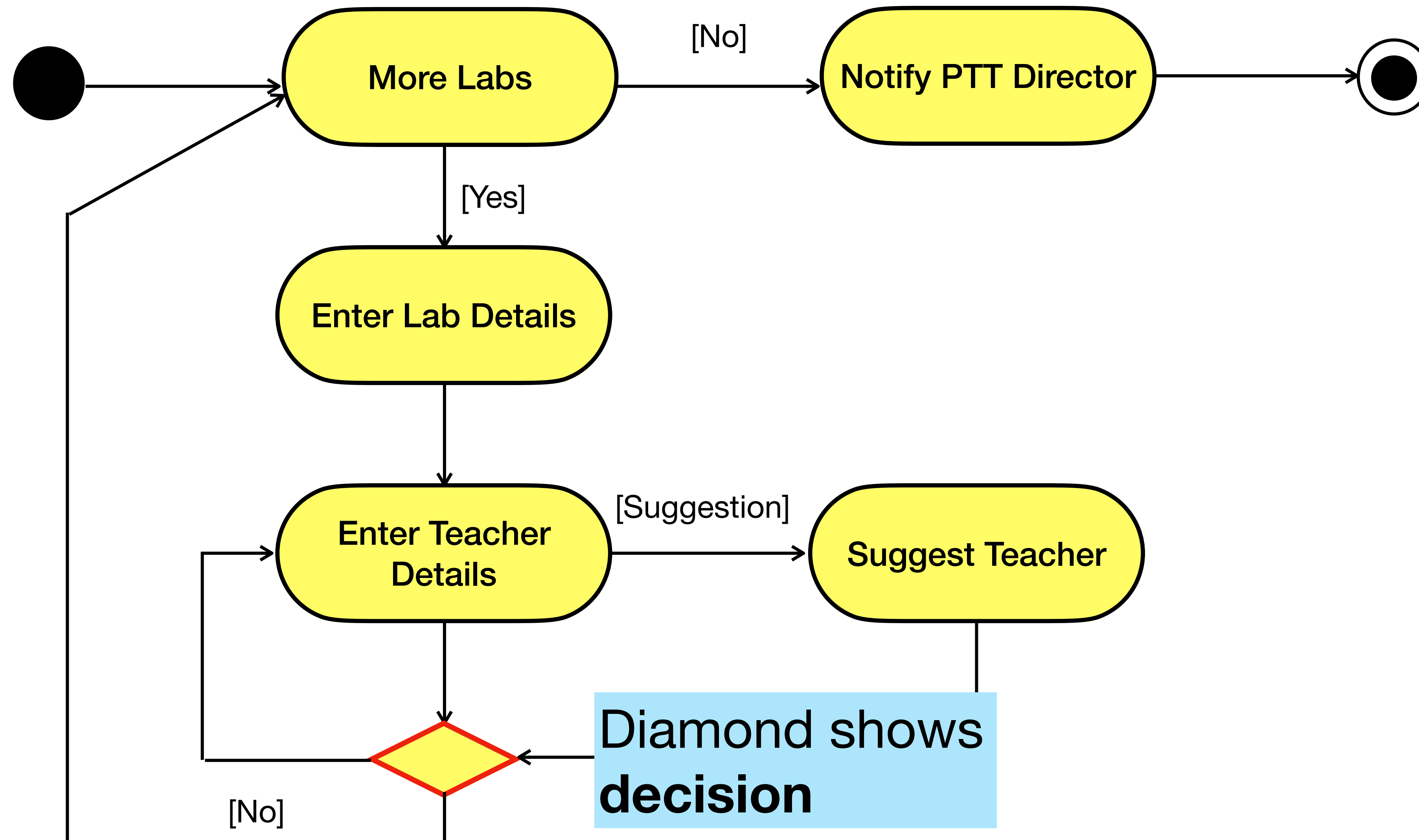


“Request Teacher” Flow of Events: Activity Diagram

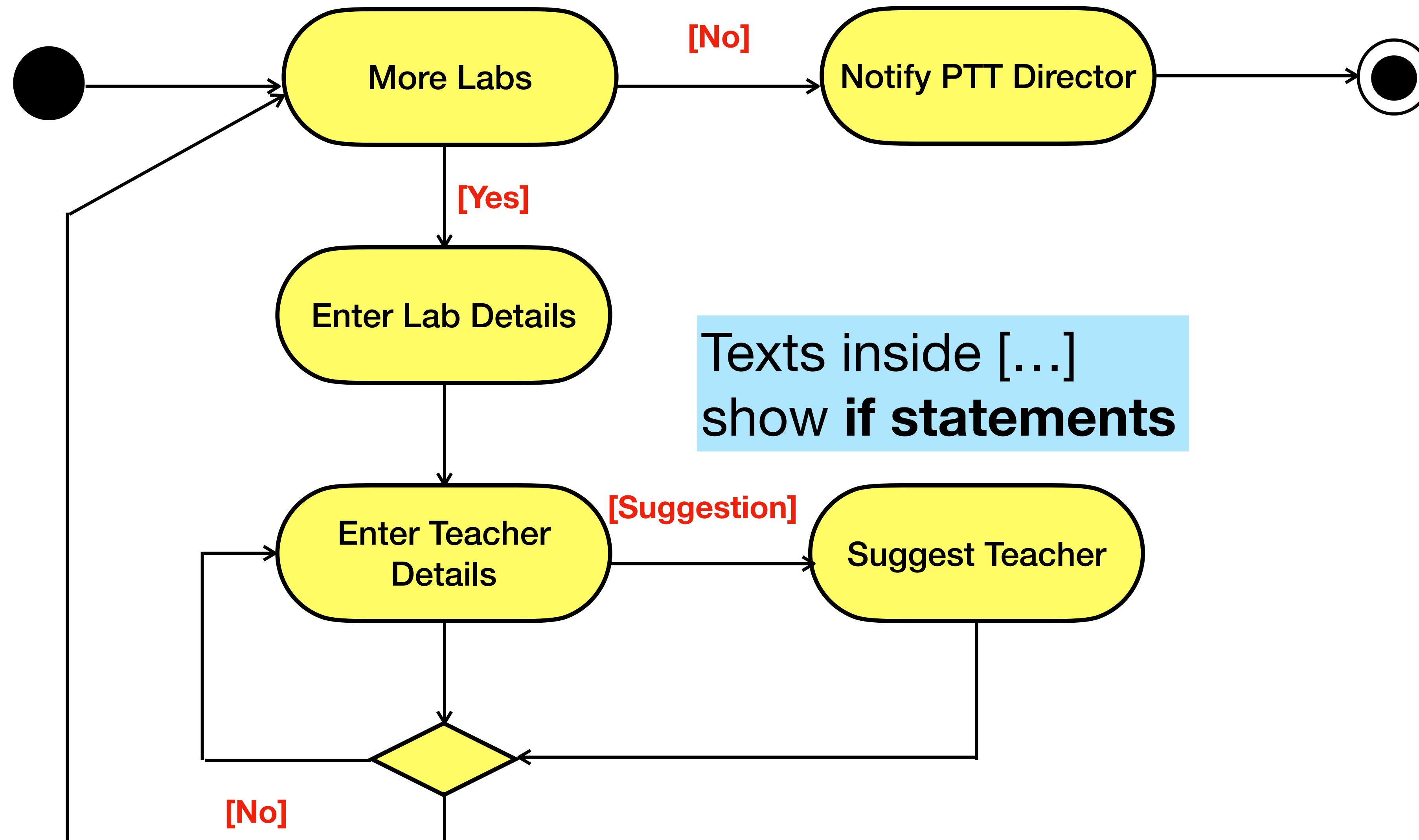


Rounded rectangles
show **activities**

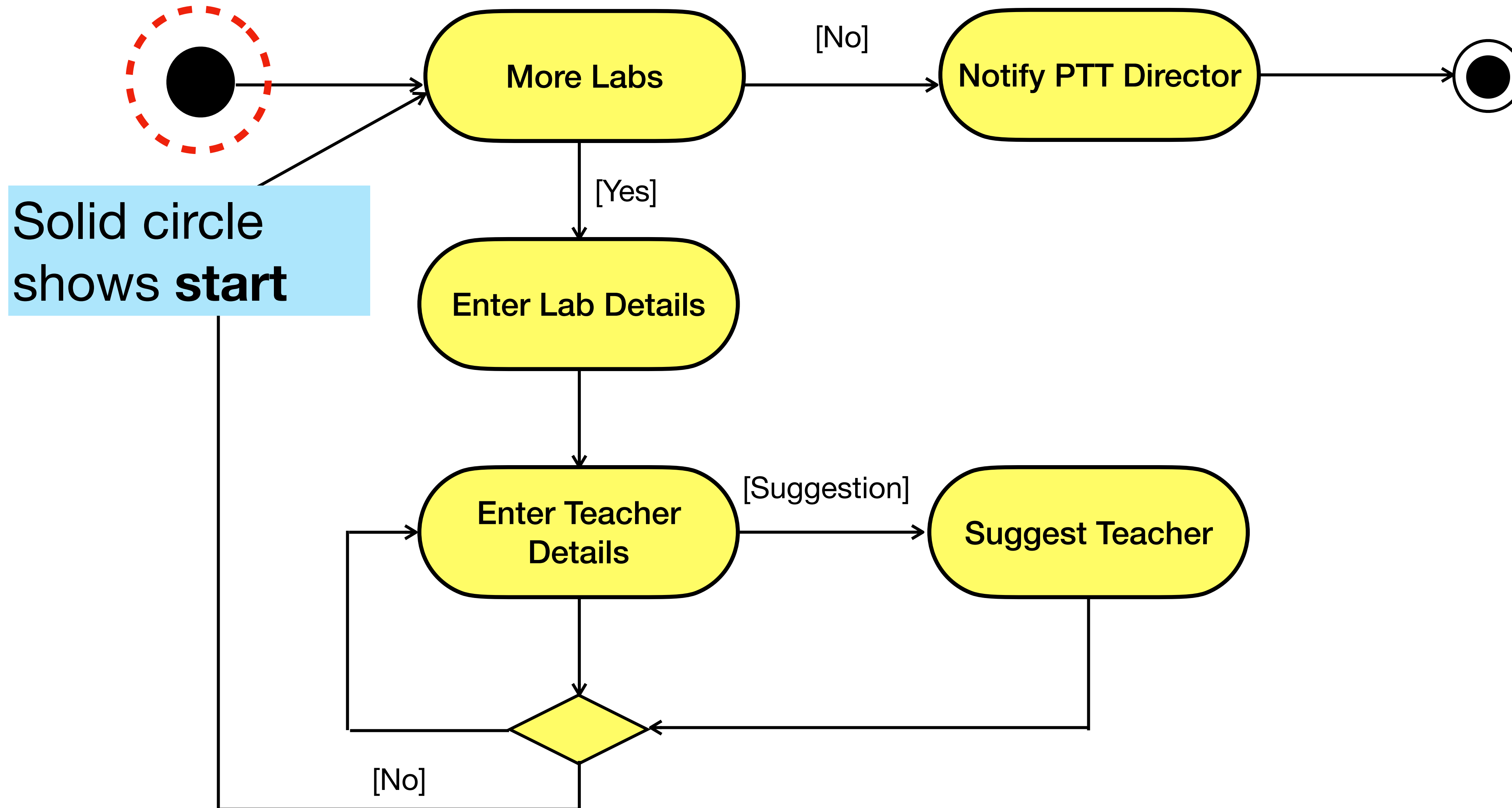
“Request Teacher” Flow of Events: Activity Diagram



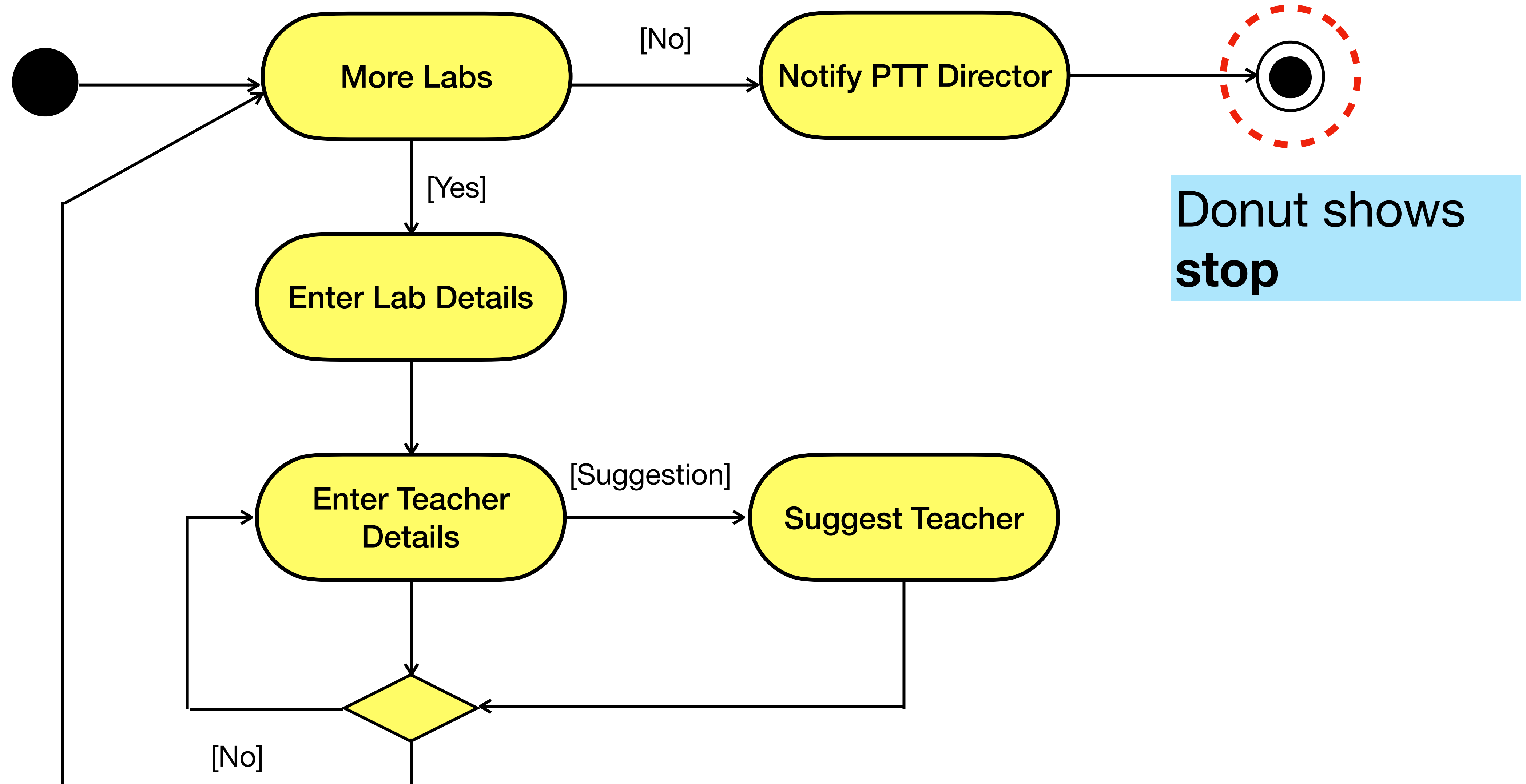
“Request Teacher” Flow of Events: Activity Diagram



“Request Teacher” Flow of Events: Activity Diagram



“Request Teacher” Flow of Events: Activity Diagram



Interactive Exercise

- Use your computer or smartphone to access the exercise
- If you use your **computer** or **smartphone**: Open a web browser, enter www.menti.com and enter code **4663 1858** with your smartphone you can also use the following QR code

