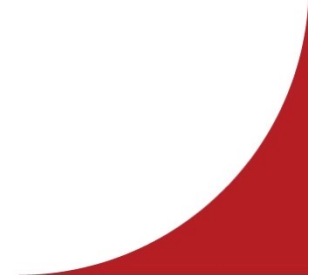
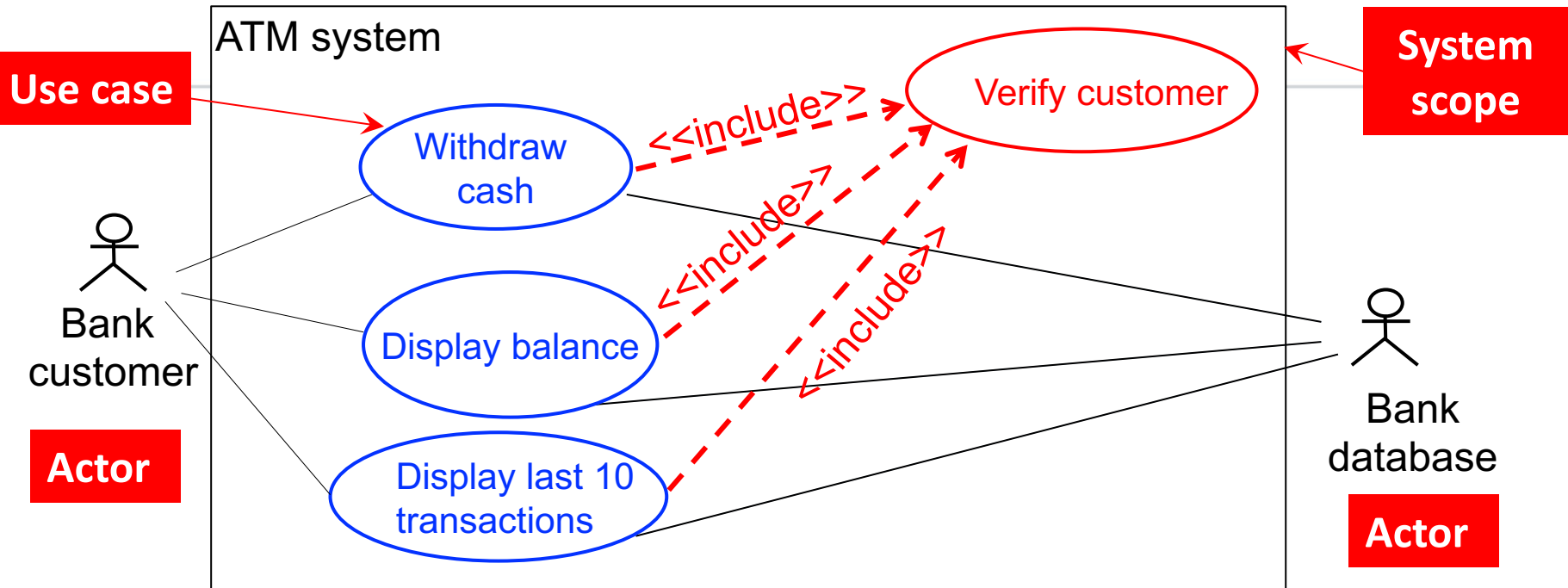

SCC.211 Software Design

Workshop 2: Use Cases

Gerald Kotonya – g.kotonya@lancaster.ac.uk
School of Computing and Communications
InfoLab21



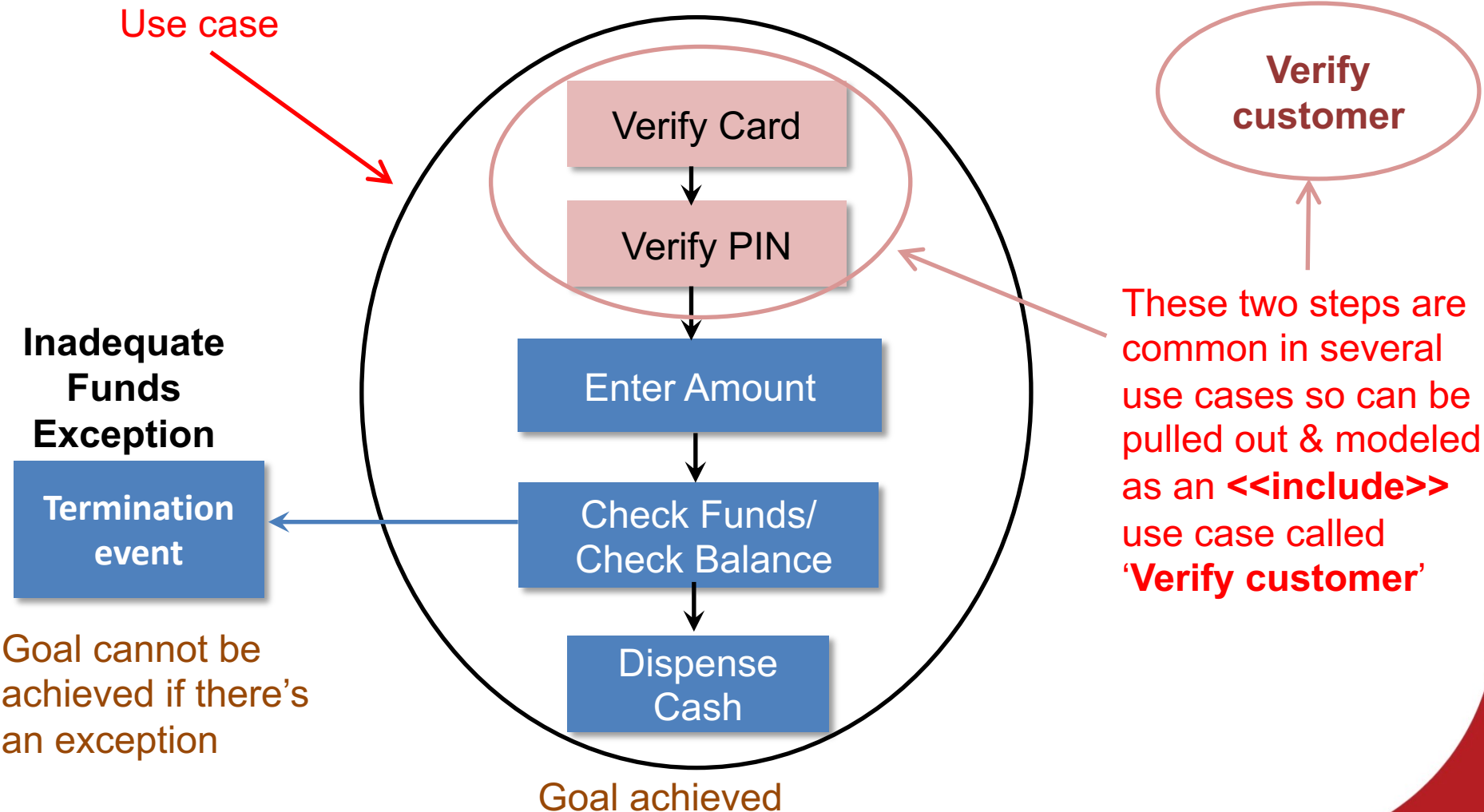
Recap: Use case diagram



- The primary actor is the bank customer whose intention is to achieve one or more of 3 the goals described represented by use cases (shown in blue)
- The customer account database is a secondary actor helping the customer to achieve their goal
- All the 3 use cases require the customer to be verified. This implies that customer verification is common behaviour in the three use cases. We can extract and represent it using the `<<include>>` keyword (see “red” use case)

Withdraw Cash use case

Normal Scenario/Happy Day Scenario (showing sequence of events)



<<include>> and <<extend>> relationship

- There may be use cases that share common or mandatory behavior.
 - Common or shared behavior is represented using the **<<include>>** relationship, represented with a broken arrow that points from the use case that requires the behavior to the shared use case.
- There are cases where a use case may conditionally add steps to another use case. For example, optional behavior
 - This behaviour can be modeled using the **<<extend>>** relationship
 - Optional or alternative behaviour is depicted using a broken arrow, drawn from the alternate/optional **use case** to the **use case** that it extends

Guidelines for writing use cases

- To identify the right use cases, always keep in mind the **GOAL** that the actor wants to achieve by using the system.
 - What **TASK(S)** do you want the system to help you achieve?
- Avoid excessive use structuring.
 - Remember use cases are about communicating high-level system behaviour not low-level implementation detail
 - Avoid functional decomposition. High-level use cases cannot be *decomposed* into lower-level use cases
 - A use case can ONLY be linked to another use via an **<<include>>** or **<<extend>>** relationship. Never directly.
- Please note the difference between the **<<include>>** and **<<extend>>** relationship

Conference paper review system

Scientific conferences are held in order to announce and discuss new results and ideas. The core activity of organizing a conference centres on selecting interesting research to be presented. This is done by via an open invitation calling for papers to be submitted and setting up a (geographically distributed) panel of reviewers. Submitted papers are then circulated to the reviewers who select the best papers to appear on the programme.

An automated system is needed to support the paper reviewing process as follows:

1. Authors submit papers to the system. A paper submission **must** be accompanied by authors' affiliation details. The program chair (PC) of the conference checks the validity of submitted papers and assigns each valid paper a unique number. Invalid submission are rejected at this point and the authors informed. After the paper submission deadline, the PC uses the system to distribute valid submitted papers among the reviewers. Each paper is sent to three distinct reviewers, none of whom is an author of the paper.
2. The reviewers submit their reviews to the system before an agreed deadline.
3. The PC notifies the authors about the review results.
4. Authors of accepted papers submit revised final version papers. The PC checks that the re-submitted papers have taken into account reviewer comments.

Exercise – Please use PlantUML

1. Draw a use case diagram for the conference paper reviewing system
 2. Suggest two possible non-functional requirements the conference reviewing system
 3. A PlantUML example is provided in the next slide
-

If you get the following error,

“....Dot executable does not exist. Cannot find Graphviz...”

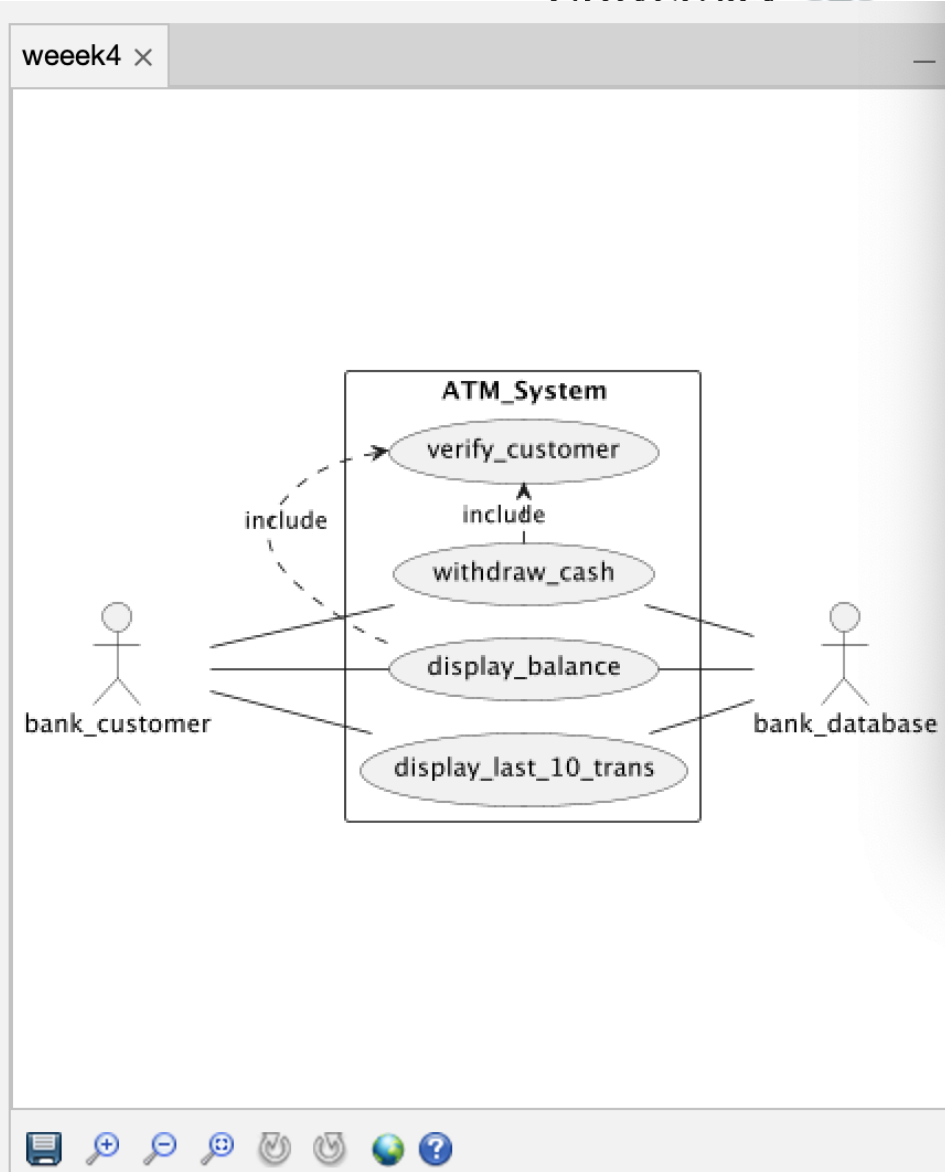
when running **PlantUML** on your laptop, add the statement:

!pragma layout smetana immediately after **@startuml** and try again.

```

1  @startuml
2
3
4  left to right direction
5  actor bank_customer
6  actor bank_database
7
8  rectangle ATM_System{
9  usecase withdraw_cash
10 usecase display_balance
11 usecase display_last_10_trans
12 usecase verify_customer
13
14 bank_customer--withdraw_cash
15 bank_customer--display_balance
16 bank_customer--display_last_10_trans
17
18 withdraw_cash--bank_database
19 display_balance--bank_database
20 display_last_10_trans--bank_database
21
22 withdraw_cash->verify_customer:include
23 display_balance->verify_customer:include
24 }
25 @enduml
26

```



Thank You!

Questions?

