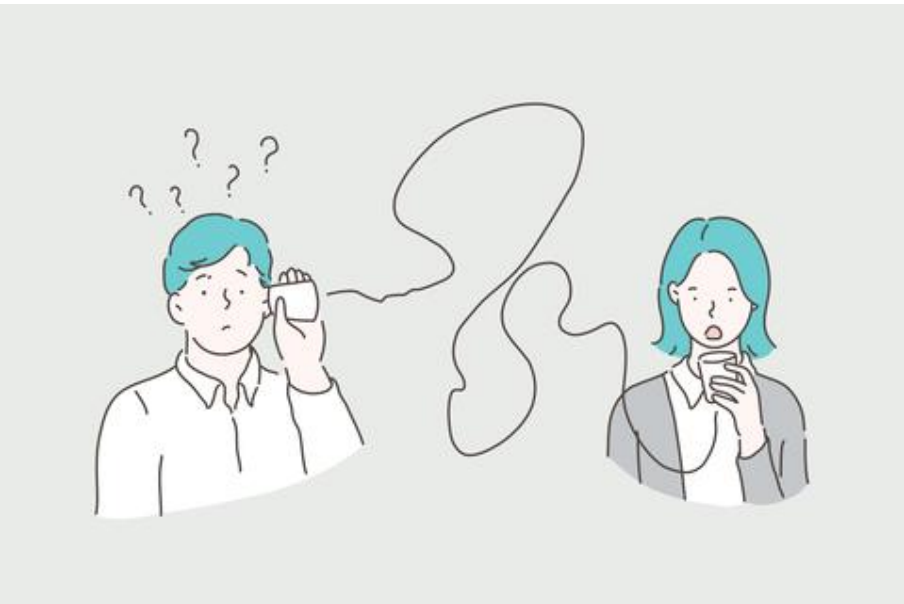
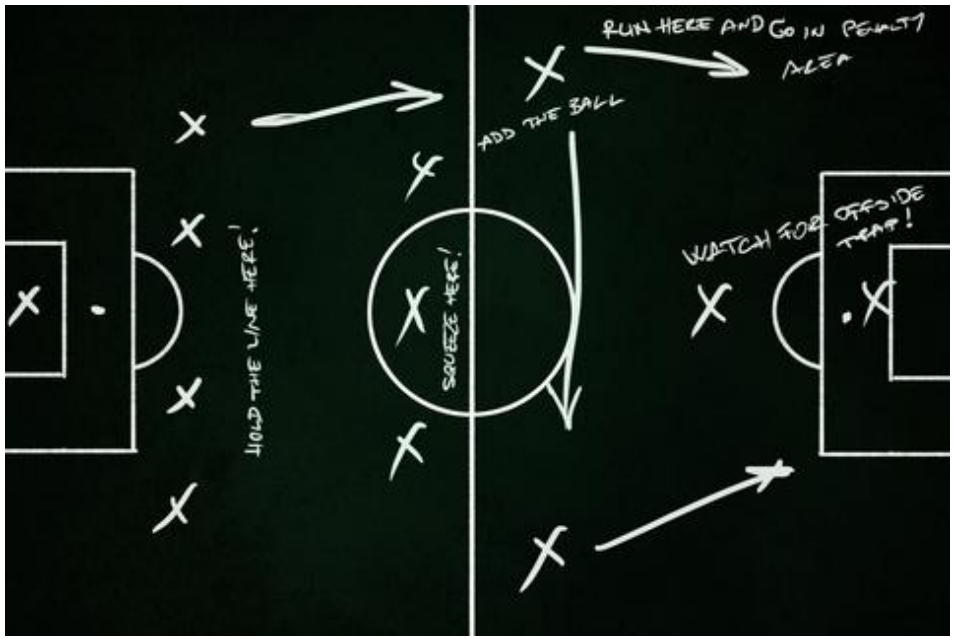
Process Design
Functional/Logical Architecture

Lecture 9
29 May 2025

Slides created by
Prof Amir Tomer
tomera@cs.technion.ac.il



ID 161250385 © Drawlab19 | Dreamstime.com

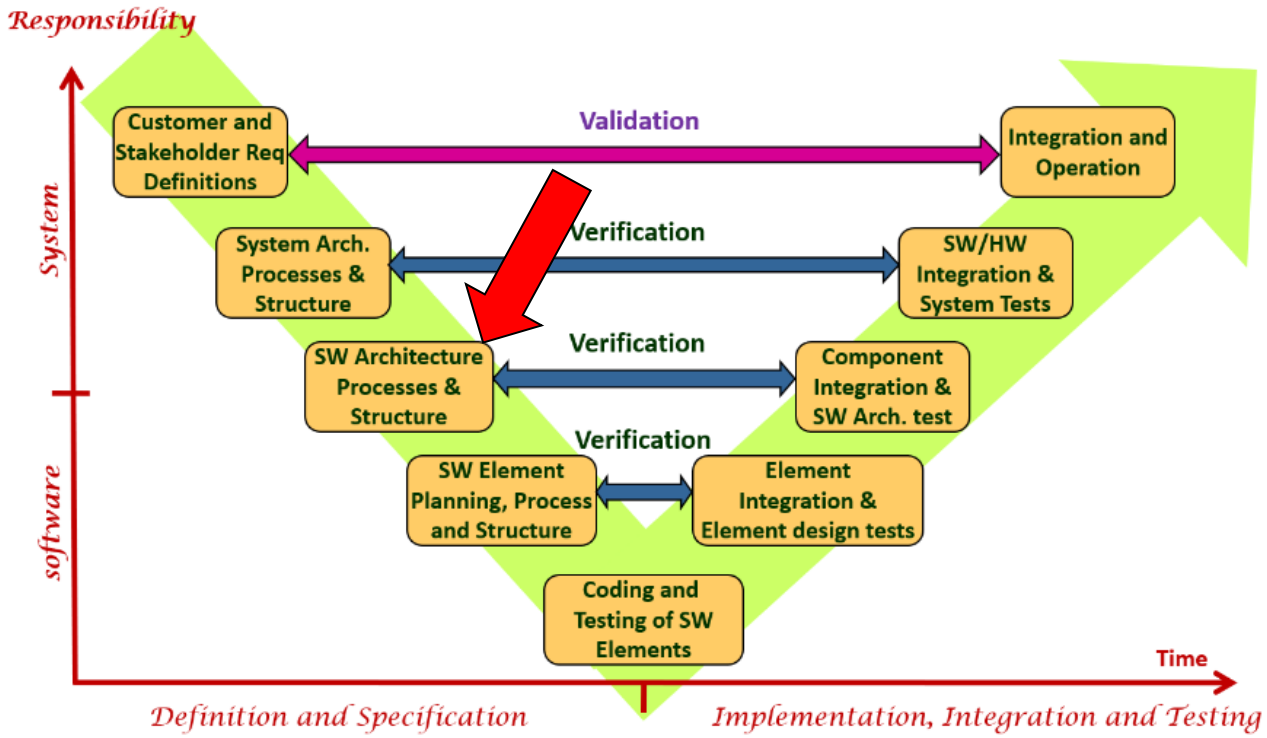


Topics for Today

- Process Design
  - Sequence Diagrams
- Software Interfaces
  - Component interfaces
  - Logical architecture

Planning Processes

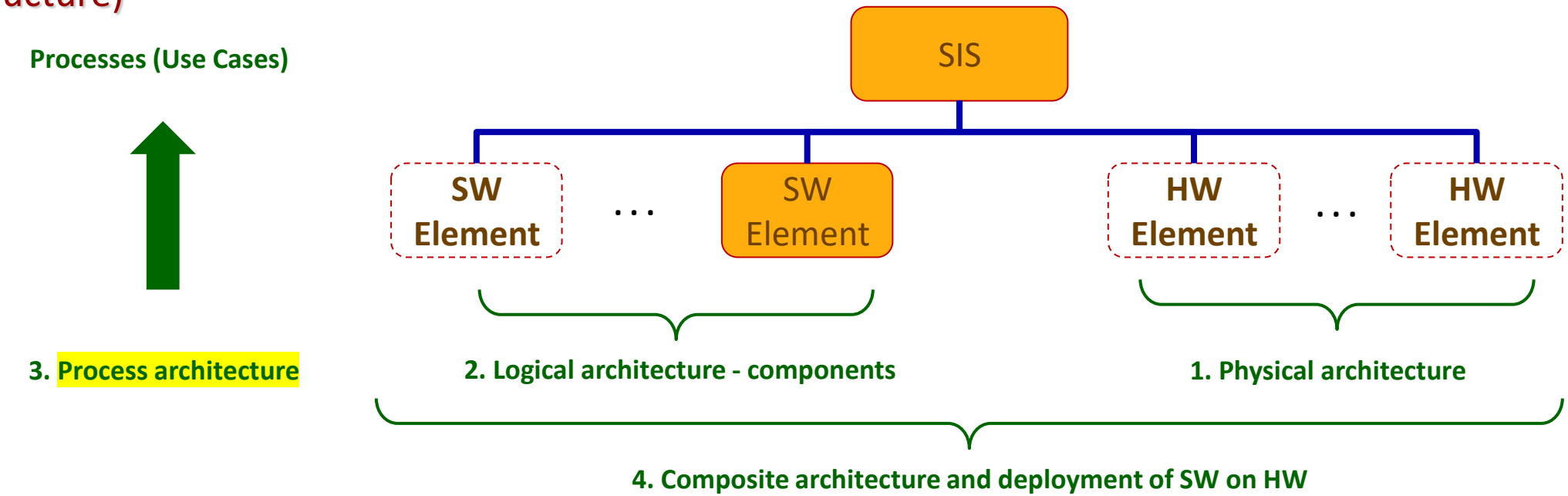
- Our goal: Define interactions between element groups to implement the system's processes
- Inputs:
  - System processes (UC)
  - Functional requirements from the requirements table
  - Activity diagrams
  - List of software elements in the system (from functional analysis)
- Outputs:
  - Sequence diagrams



System Architecture: SIS architecture includes

Reminder

1. Physical architecture: Hardware components and physical connections – static model (structure)
2. Logical architecture: Software components and logical connections – static model (structure)
3. Process architecture: Implementation of processes via interaction between components - dynamic model (behavior)
4. Composite architecture: Implementation of logical connections via physical connections – static model (structure)

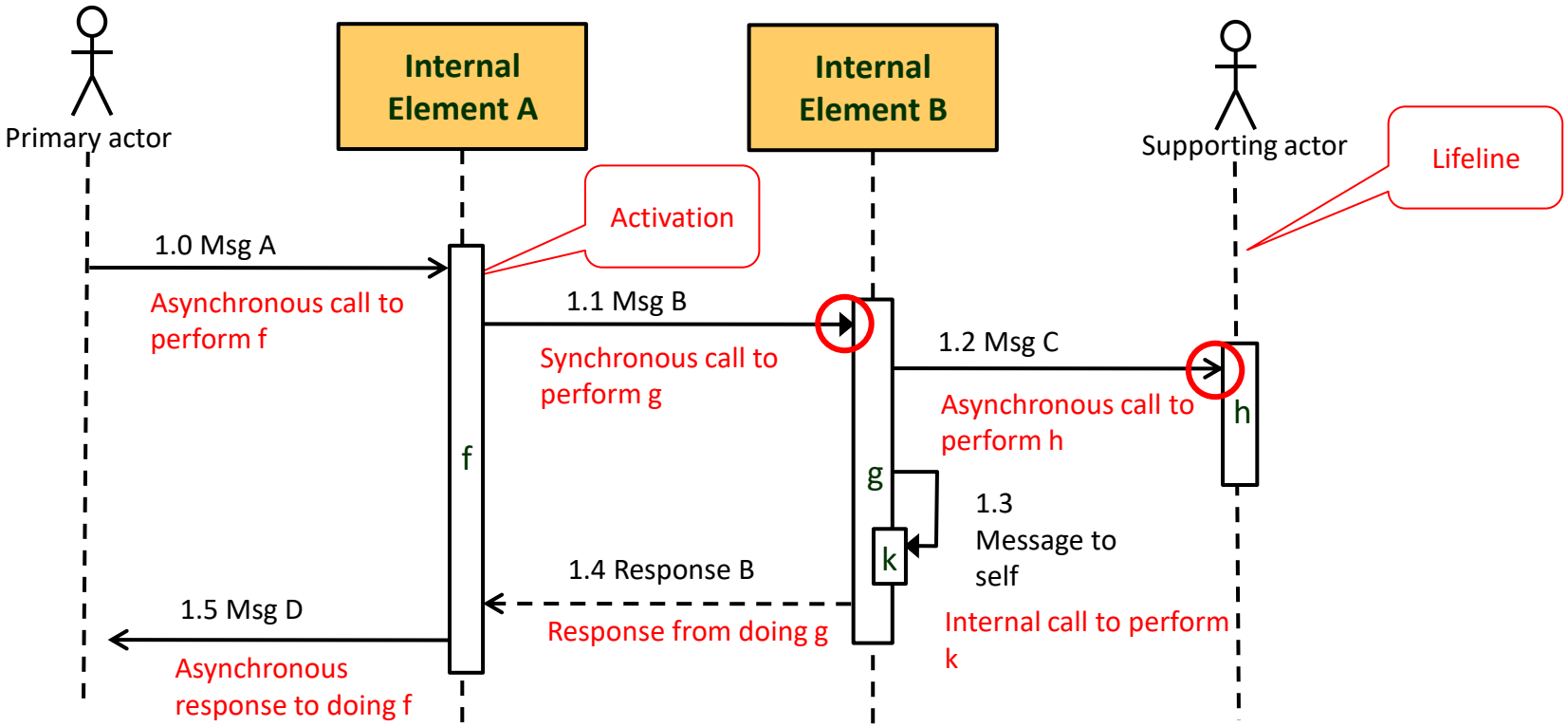


Implementing system processes via functional components

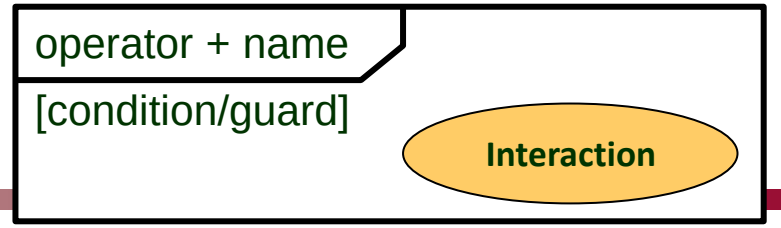
- Functional components are the task force that will perform the system processes
  - System processes = Use Cases
  - Every component is assigned a functional task
  - Components interact to perform the system processes
    1. Interactions between components (internal)
    2. Interactions between components and the environment (external)

Sequence diagram

- Describes the process as an interaction between components
  - **Internal elements** in the system (subsystems, computers, components, objects)
  - **External entities** (primary/supporting actors, external elements)



Interaction Frames



- Fragments of the sequence diagram that show some nesting or control blocks within the sequence. Operators:

seq:

- Frame that names a set of steps that must be completed before proceeding
- Like <<include>>-ing a use case within the sequence diagram, set can also be performed on its own

ref

- References another sequence diagram

par

- Run more than one block in parallel

loop

- A loop
- Remember to write the loop condition

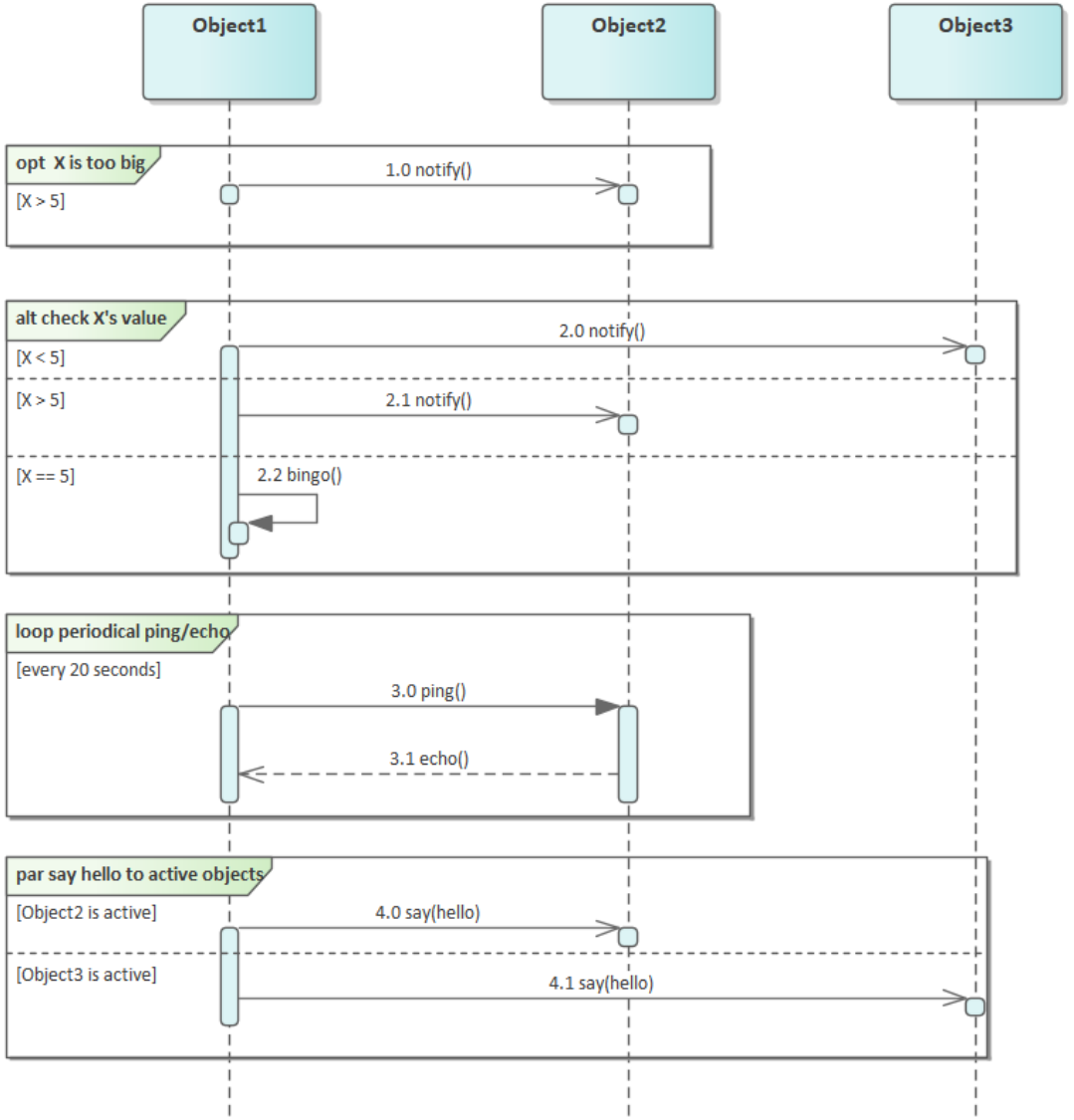
alt

- Like an if-then-else or switch statement
- Remember to write the condition

opt

- Like an if-then statement
- Remember to write the condition

Sample sequence diagram with control blocks

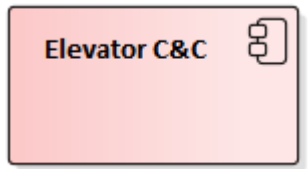


Refining the elevator example

Reminder

Single car operations

1. Receive user requests from in-car floor buttons
2. Receive requests from central operations
3. Managing travel plan
4. Engine control (travel, stop)
5. Door control (open, close)



Command and Control for Single Car

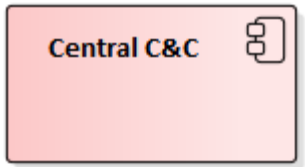
29 May 2025

Refining the elevator example

Reminder

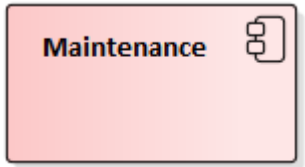
Whole system operations

System Command and Control



1. Receive user requests from per-floor up/down buttons
2. Assigning calls to elevator cars
3. Sending calls to elevator cars
4. Identify emergency events and take care of rescue operations

System Maintenance



5. Built in Test (BIT)
6. Support technician tests
7. Support technician repair tasks

Configuration Management

8. Start up
9. Shut down
10. Sabbath mode

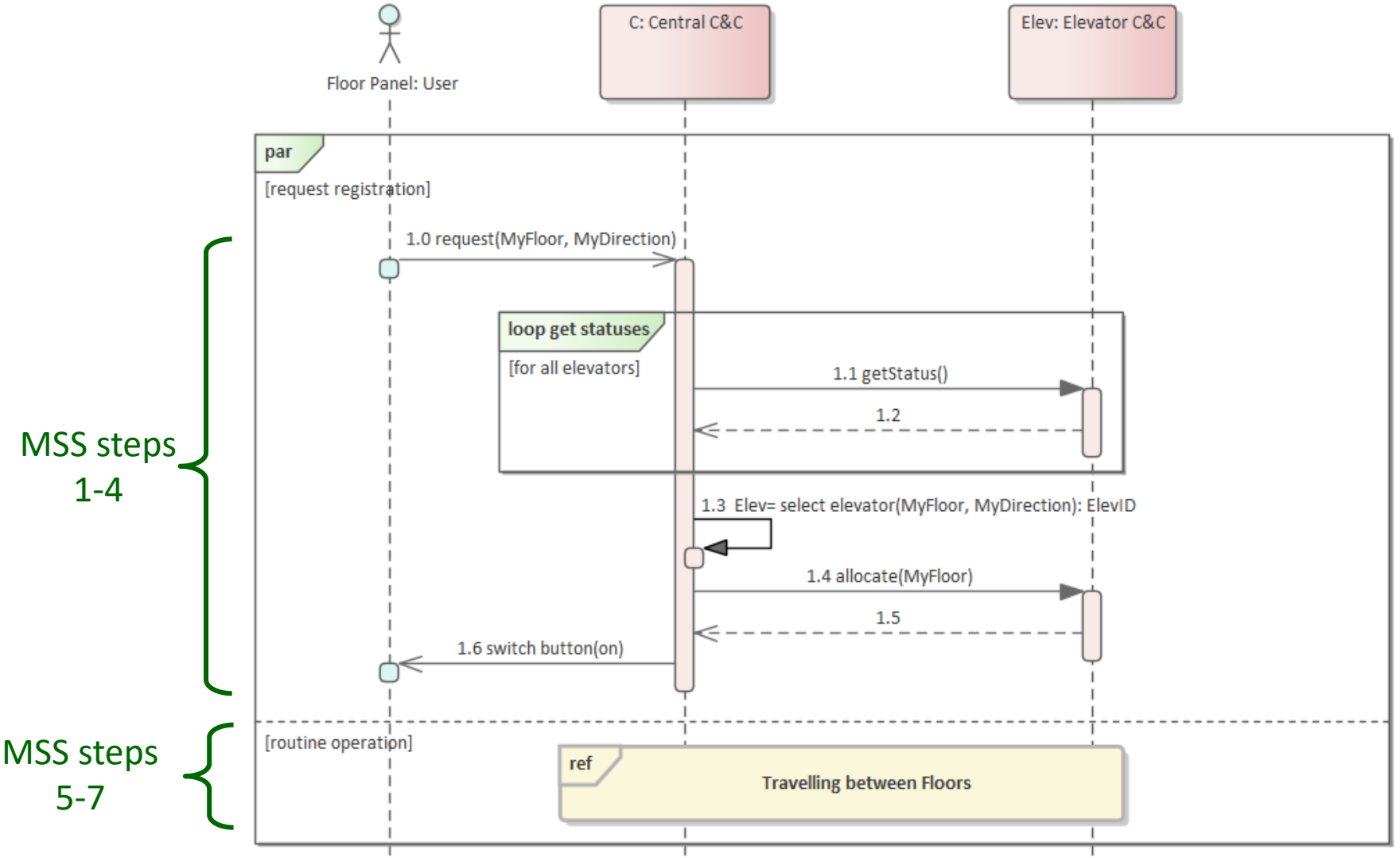
SUC-1 Call Elevator

Reminder

SUC-1	Call Elevator
Actors and Goals	Passenger: To receive an elevator car available for travel
Stakeholders and Interests	None
Pre-conditions	• User is on a floor in the building with an elevator door • System is operational (post-condition of UC: Start-up)
Post-conditions	An elevator car is at the user's floor with the door open (destination floor)
Trigger	**Passenger** pushes the up or down button on the floor
Main Success Sequence (MSS)	1. The system records the button press 2. The button lights up 3. The system finds a car traveling in the desired direction 4. The system assigns a stop for the car 5. The elevator arrives at the floor 6. The door opens 7. The floor button turns off These steps take place in parallel
Branch A	Alternative from step 2 of MSS: The button is already lit 2A1. Return to step 5 of the MSS.
Requirements trackback	…

29 May 2025

Sequence diagram: SUC-1 Call Elevator



SUC-2 Ride Elevator

Reminder

SUC-2	Ride Elevator
Actors and Goals	Passenger: To arrive at the desired floor
Stakeholders and Interests	Safety standards: Ensure passengers are not stuck in elevators
Pre-conditions	• User is in an elevator car • System is operational (post-condition of UC: Start-up)
Post-conditions	• Car arrives at the desired floor (destination) • Passengers can leave the elevator (interest)
Trigger	**Passenger** pushes the button for the desired floor
Main Success Sequence (MSS)	1. The car records the button pressed and adds a scheduled stop at the floor 2. The button lights up 3. The car door closes (if it was open) 4. The car continues traveling to the next floor in its stop list 5. The car stops at the floor 6. The door opens 7. The floor's light turns off 8. Return to step 3.

These steps occur in parallel to the rest of the UC

29 May 2025

SUC-2 Ride Elevator

Reminder

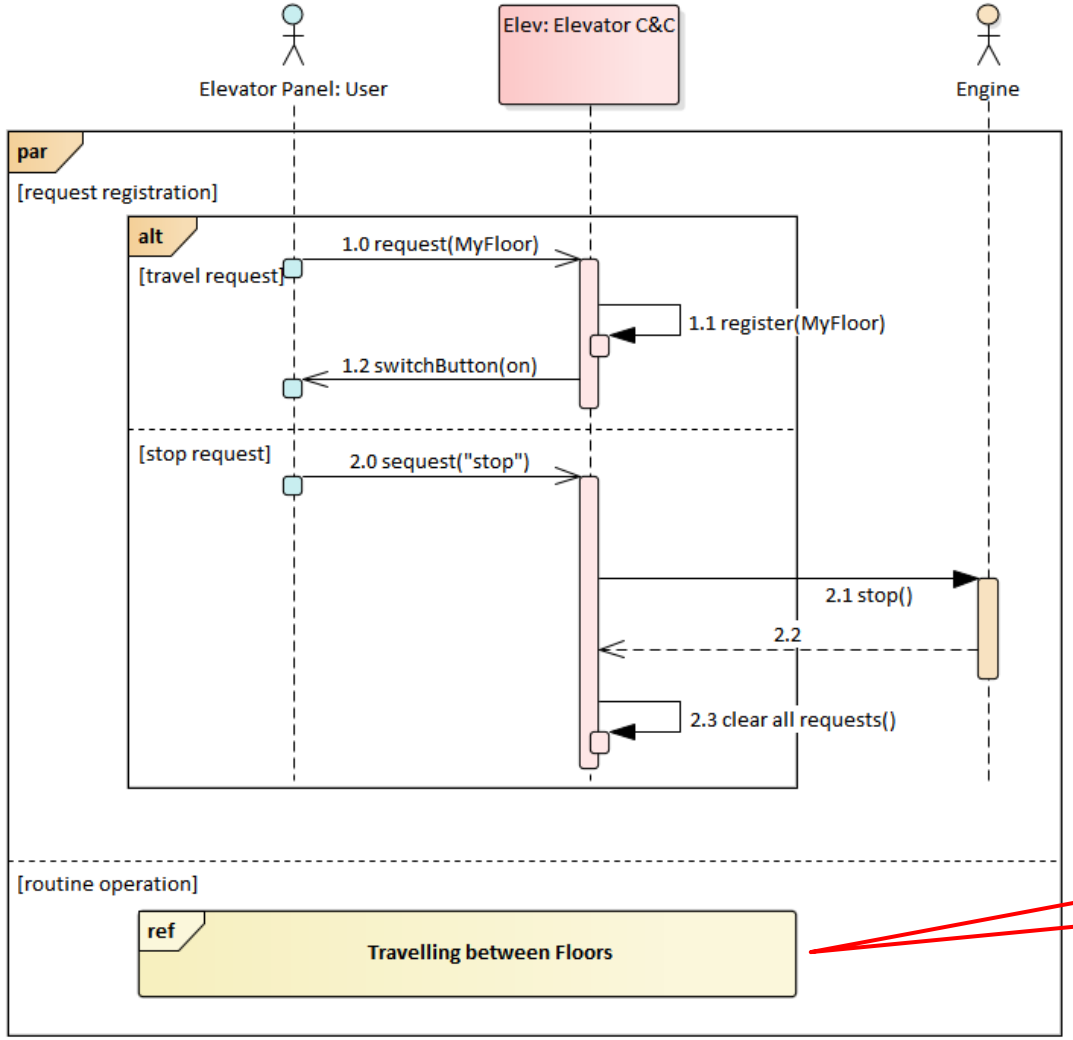
SUC-2	Ride Elevator
Branch A	Exception from step 4 of the MSS: Passenger presses the emergency stop button 4A1. The car immediately stops 4A2. The car cancels all upcoming planned stops 4A3. End of use case
Branch B	Exception from step 4 of the MSS: Car gets stuck 4B1. Call for rescue is issued (transfer to SUC-3 that extends)
Requirements trackback	…

Sequence Diagram for SUC-2 Ride Elevator

Steps 1-2 of the MSS

Branch A

Steps 3-8 of the MSS

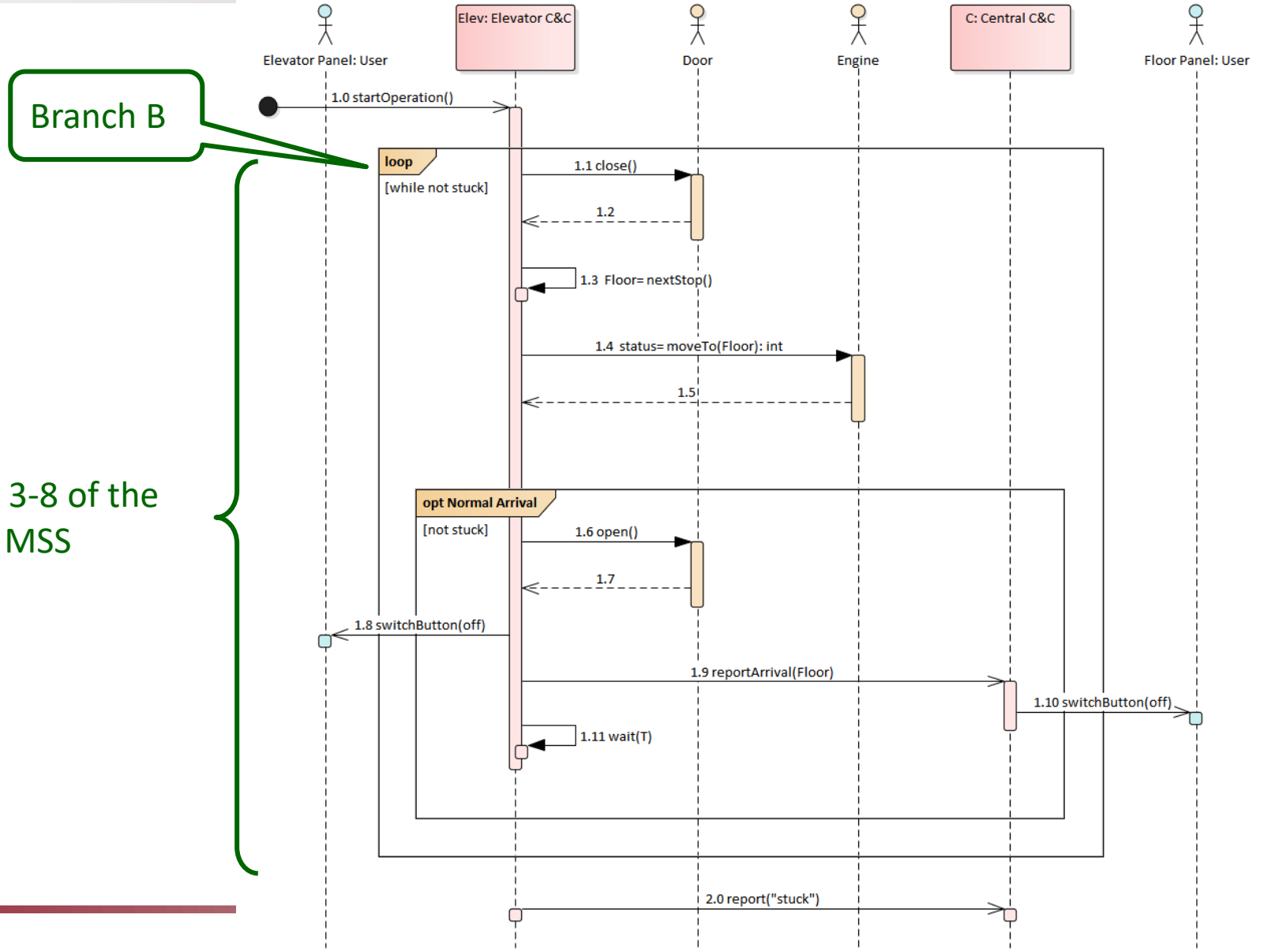


References sequence diagram on another slide

29 May 2025

Sequence Diagram: Elevator regular operation

ref
Travelling between Floors



Branch B

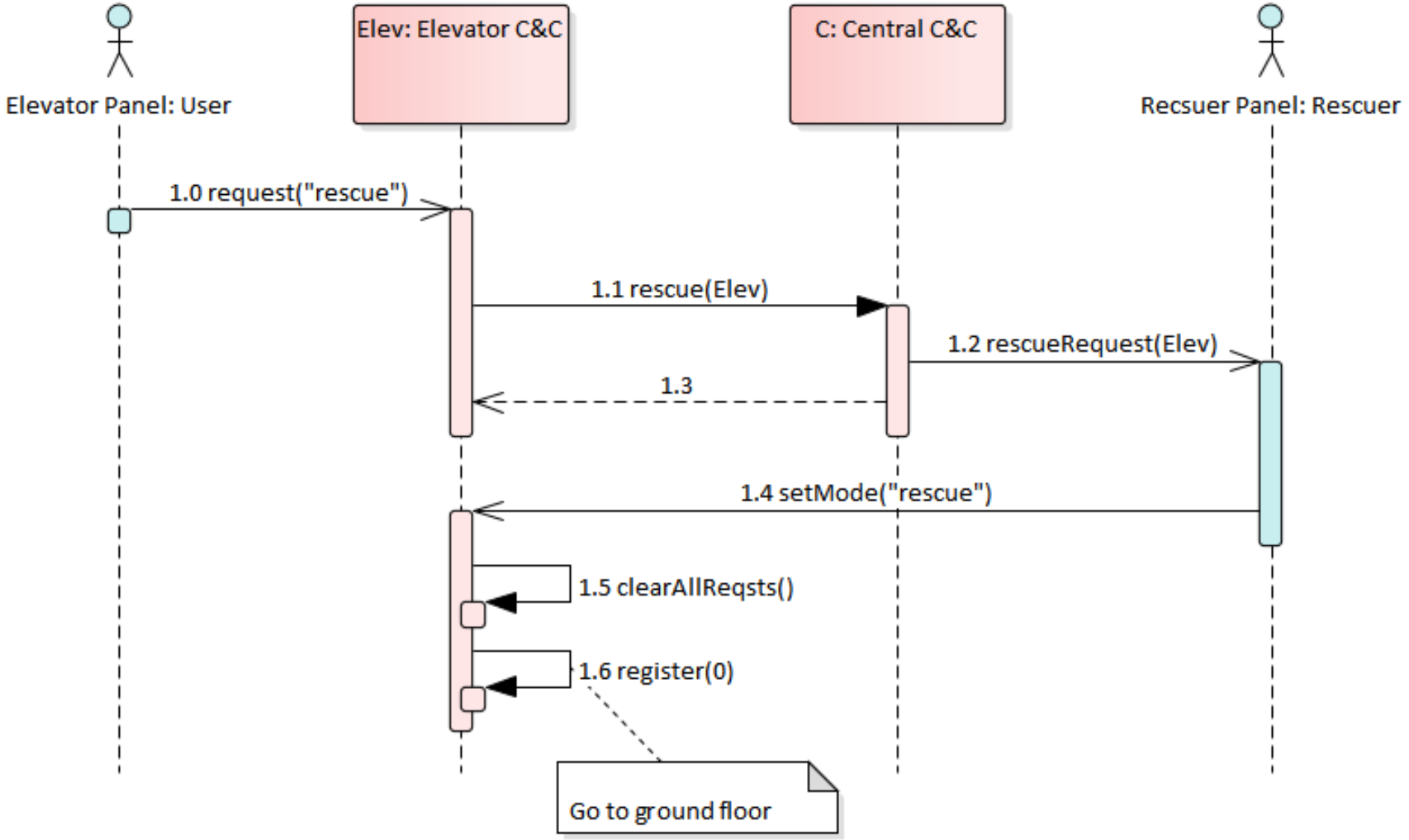
Steps 3-8 of the MSS

SUC-3 Rescue

Reminder

SUC-3	Rescue
Actors and Goals	Passenger: To be rescued from a stuck elevator Rescuer: Supporting actor (helps rescue)
Stakeholders and Interests	Safety standards: Ensure passengers are not stuck in elevators
Pre-conditions	• Car is stuck (got stuck while traveling, extending SUC-2)
Post-conditions	• Passenger can exit the car (goal + interest)
Trigger	**Passenger** calls for rescue
Main Success Sequence (MSS)	1. The system calls the rescuer 2. The rescuer goes to the machine room and starts “rescue” mode 3. The elevator arrives at the ground floor 4. The door opens
Branch	None
Requirements Trackback	…

Sequence diagram form SUC-3: Rescue



In class task: Implement a UC via sequence diagrams

- Create sequence diagrams to implement 3 of the ePark use cases
- Use the components created in the previous in class assignment

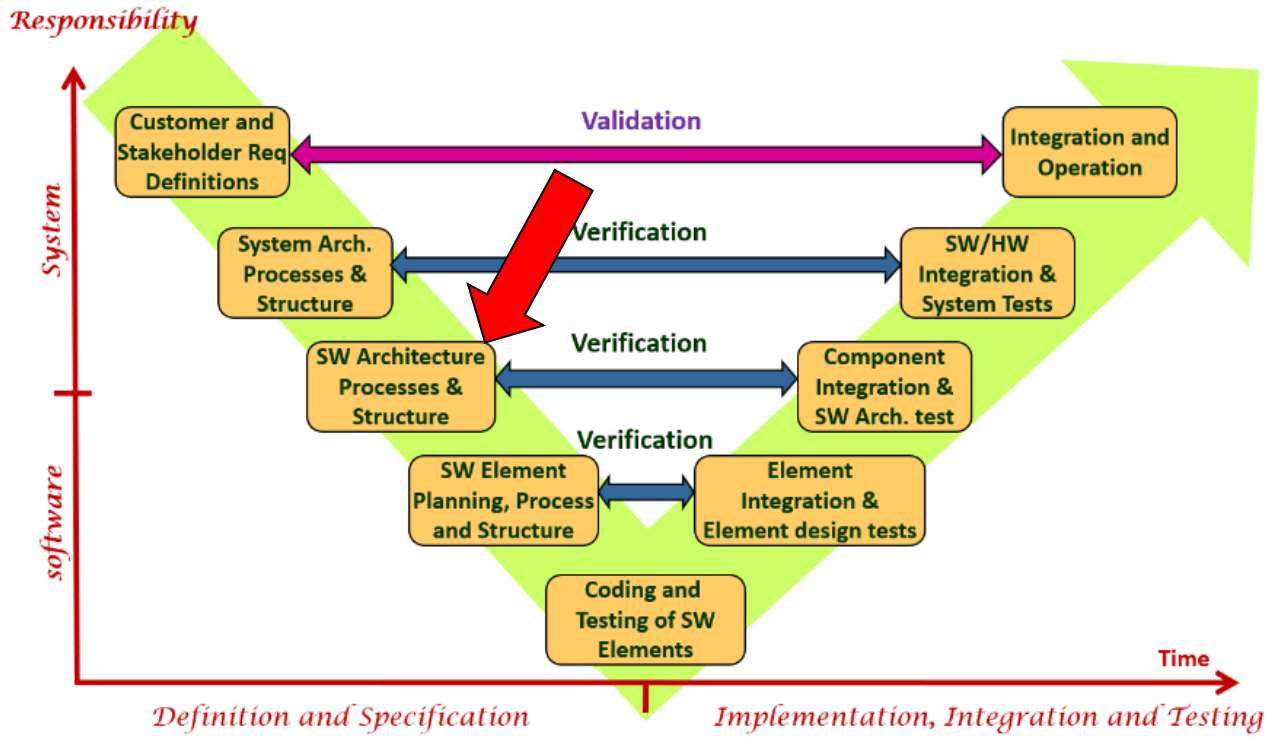
So Far

- Process Design
  - Sequence Diagrams
- Software Interfaces
  - Component interfaces
  - Logical architecture

Planning Software Architecture Activities

[Software architecture: Structure]

- Our goals:
  - Define the structure (organization and interfaces) of software components
  - Show how logical interfaces (software) work with physical interfaces (hardware)
- Inputs:
  - Software components and processes (Sequence diagrams)
  - Deployment diagram
- Outputs:
  - Component diagram (logical and functional architecture)



Functional components

Reminder

Functional components (logical, software)

- Can perform one or more operation (provide services)
- Has 1+ interfaces through which other components or external entities can start the functionality (others receive services)
  - With or without sending information
- Can run operations on other entities or components
- Doesn't constrain implementation choices (many ways to implement)

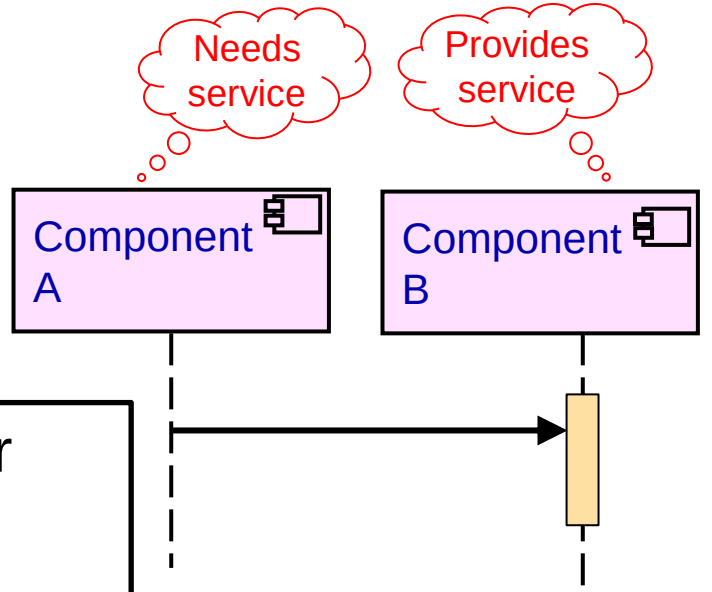
Component is a functional black box to others

- Outsiders can probe the component's functional readiness via input/output
- Can measure its performance externally only
- Can be replaced with other components with identical input/output specifications and behavior

Functional Interfaces (Software Interfaces)

- Components interact with environment (other components or external entities) via two types of interfaces

1. Provided interfaces (provided by the component)
   - Examples:

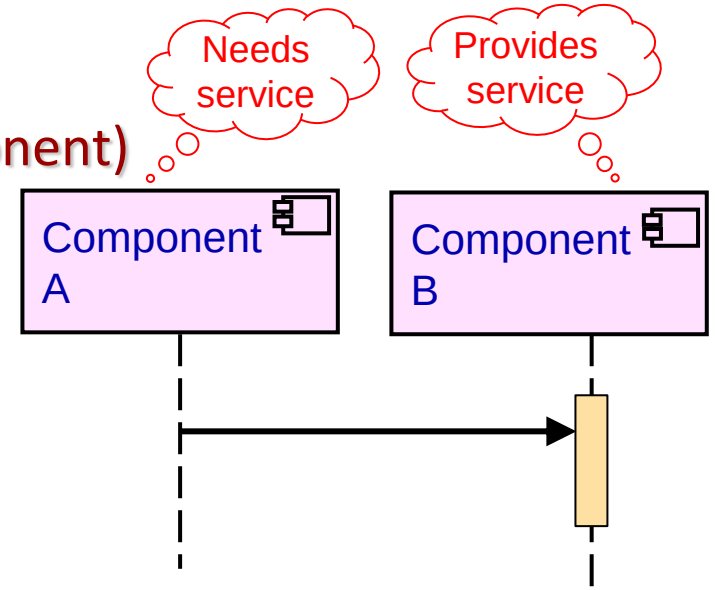


Public methods • name, parameters, return type • Application Program Interface – API	SQL query interface	Infra-red (IR) bar code scanning • 1d or 2d
Accept uploaded files	Dialog box	HTML form

Functional Interfaces (Software Interfaces)

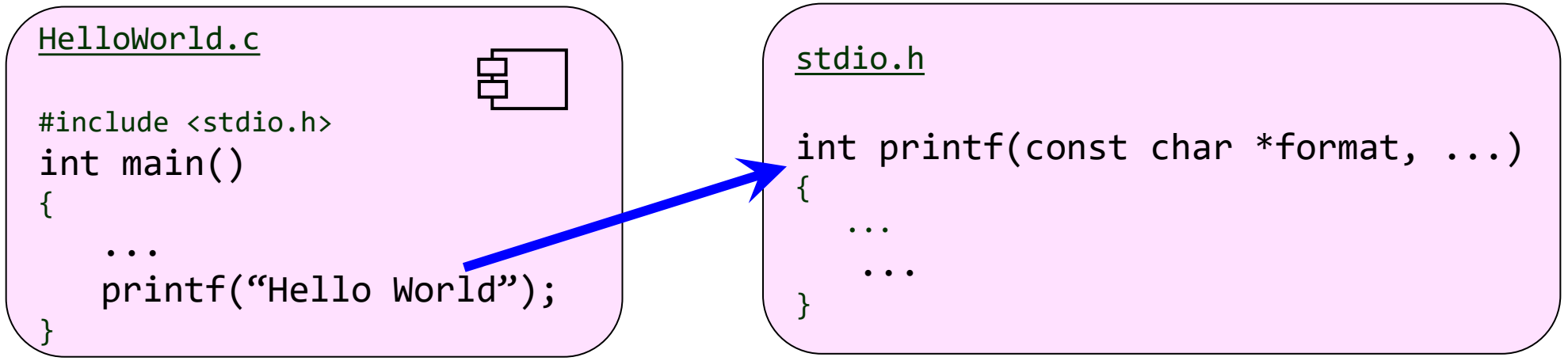
- Components interact with their environment (other components or external entities) via two types of interfaces

2. Required Interface (component needs service from another component)
   - Calls other components via their provided interfaces
   - Examples:
     - Same as before, just the other way around (except for user interfaces)



Interfaces between connected components

- When components are modules in the same software (compiled or linked together)
  - Call the API via direct method call (e.g. using a static library)



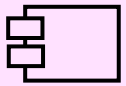
Requires service of printing to the screen

Provides service of printing to the screen

Interfaces between components in the same environment

- Components are in the same run time environment (e.g. OS)
  - Call via an agreed upon interface (e.g. using a dynamic link library DLL)

MyProg.java

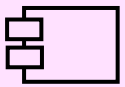


```
// Get DLL handle
     int DllHan = DllLib.DLLGetHandle("DynamicLibrary");
//Get the function pointer
   int DLLMethod = DllLib.DLLGetMethod(DllHan, "DllMethod");
//Call the Method
   int ans = DllLib.call(DllMethod);
```

OS Service

Source component: Requires the service

DynamicLibrary.dll



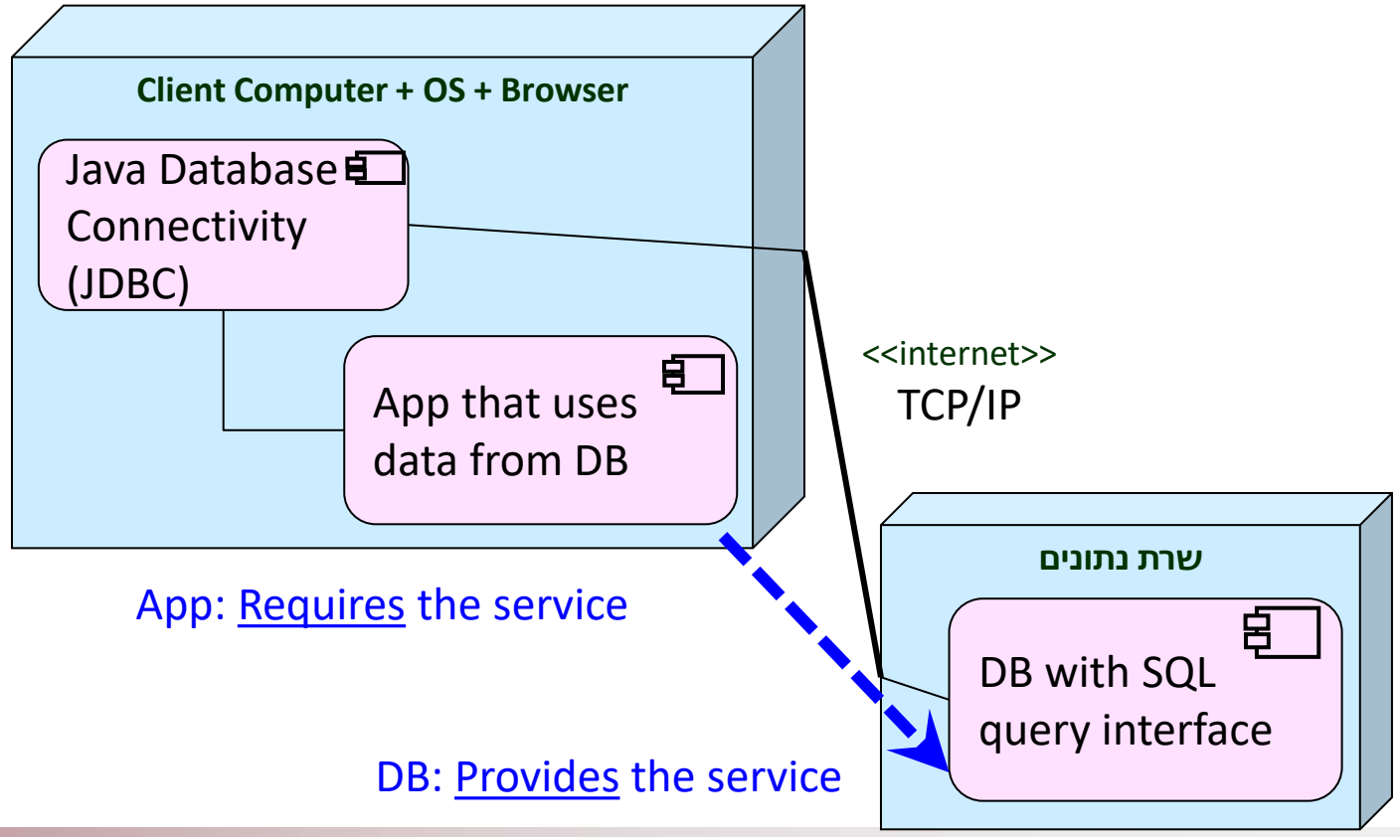
```
...

void DllMethod(...)
{
   ...
   ...
}
```

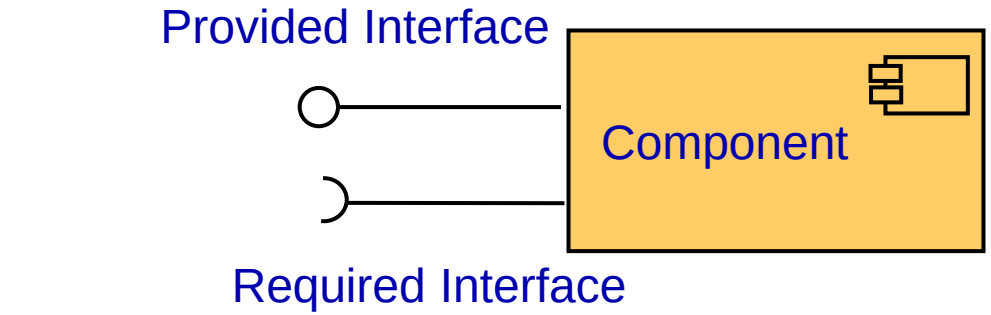
Dynamic library component: Provides the service

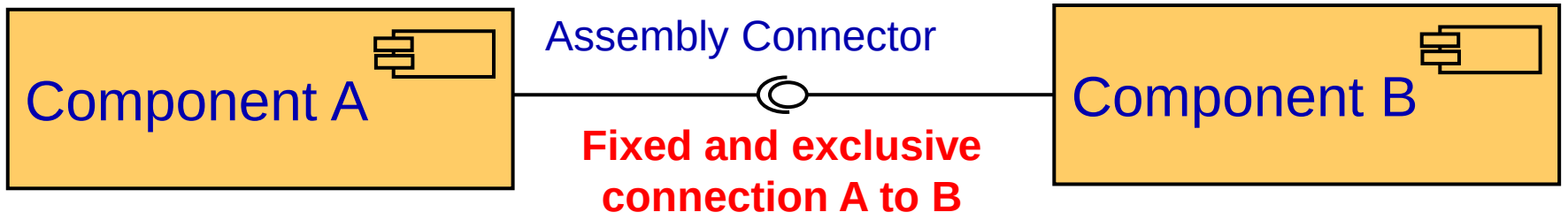
Interfaces between components on different computers

- Components are in different runtime environments and communicate via message passing
  - Pass messages via a hardware interface
- Example: Accessing a database server from a Java app). Steps include:
  1. Build the query (app)
  2. Convert query to internet message (JDBC)
  3. Send message via internet (OS)
  4. Server receives message (OS)
  5. Extract query from message (OS)
  6. Perform query (DB)



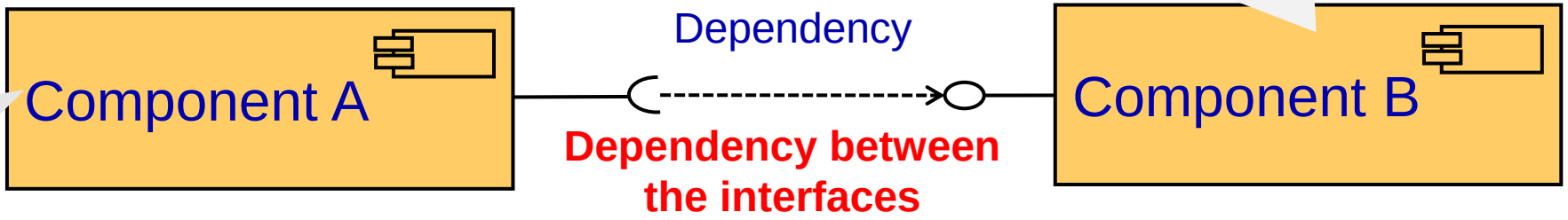
Components and Interfaces in UML





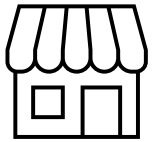
A is dependent on B:
If B changes, A may have to as well

Is the required interface always dependent on the provided one?



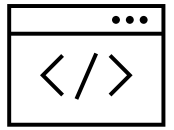
What's the difference between buying a product in a store or online?

Buying in a store (Synchronous)



- Process:
  1. Arrive at store, choose the product
  2. Pay
  3. Buyer leaves the store with the product
- Knowledge required
  1. Buyer knows the store's address
  2. Store doesn't know the buyer's address

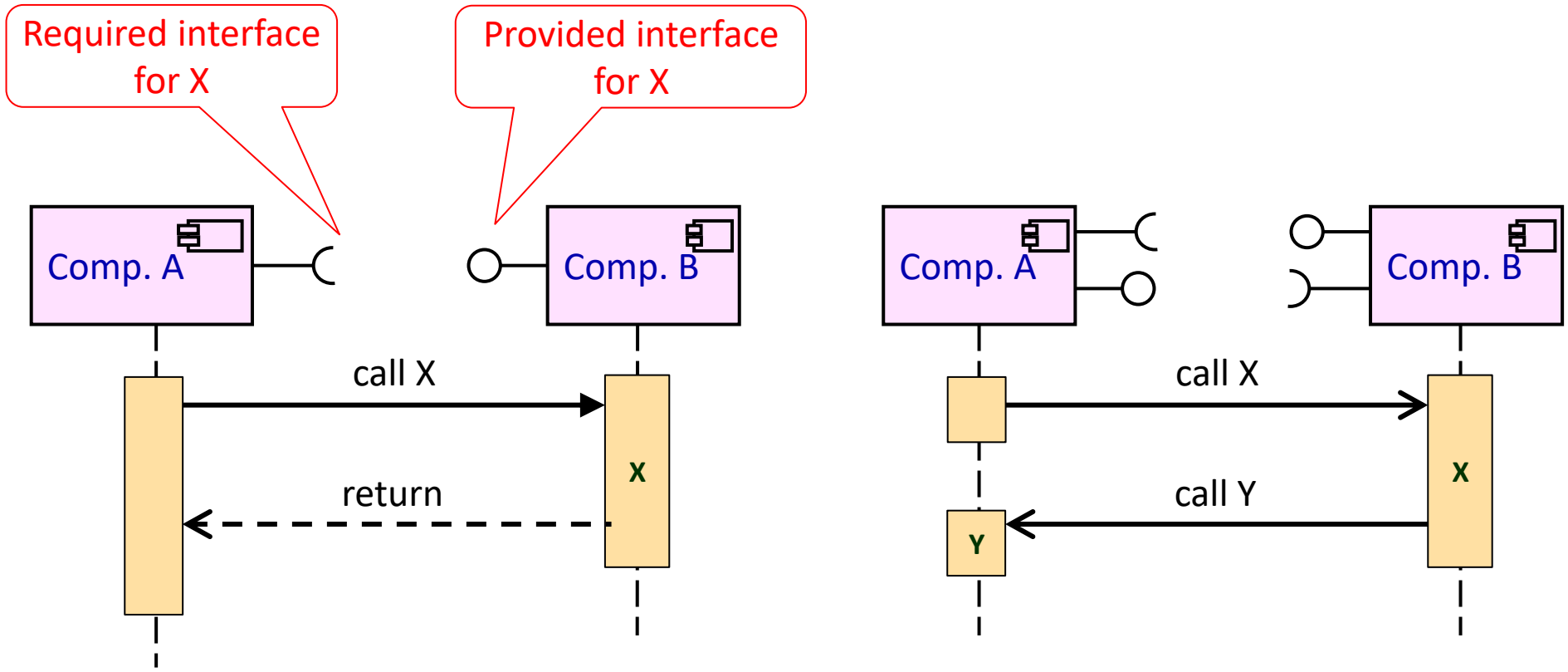
Buying online (Asynchronous)



- Process
  1. Browse to store website, choose the product
  2. Pay
  3. Buyer leaves website, product remains at the store
  4. Store ships the buyer the product
- Knowledge required
  1. Buyer knows the store's address
  2. Store knows the buyer's address

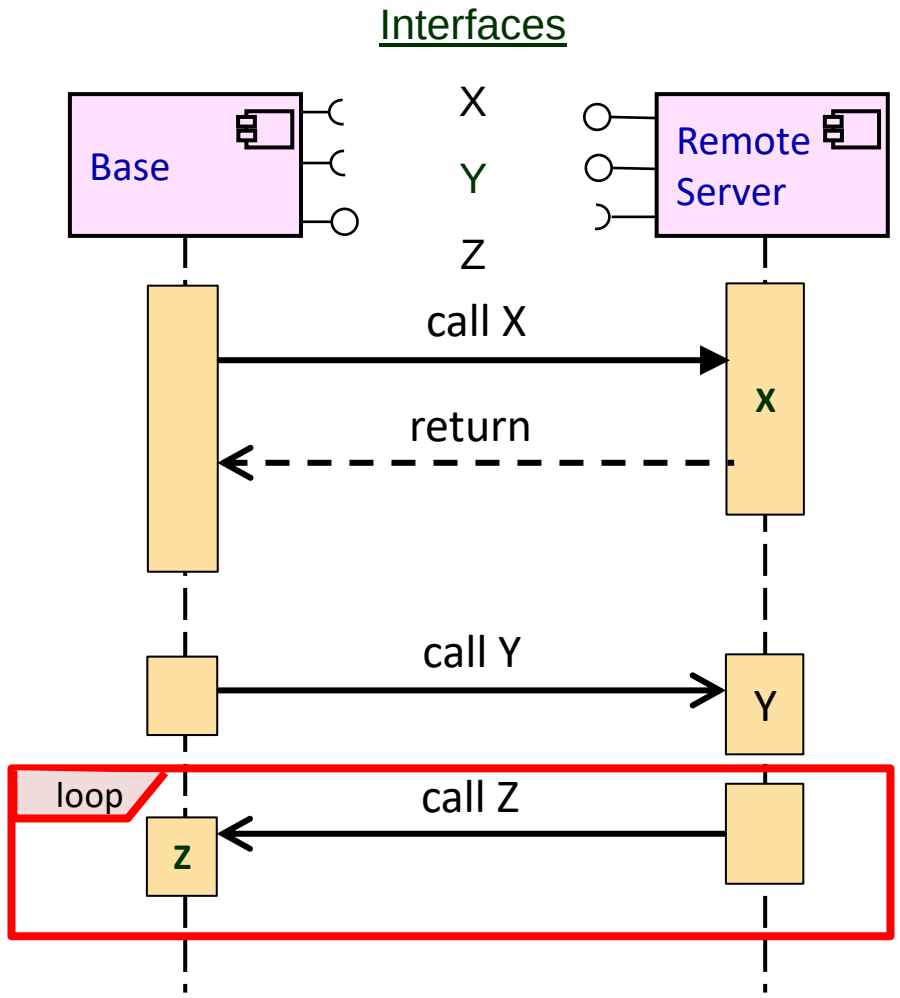
Identifying component interfaces based on environment interactions

- Synchronous calls: Response comes back as part of the call (and on its interface)
- Asynchronous calls: No immediate response, so caller must provide another interface for the response



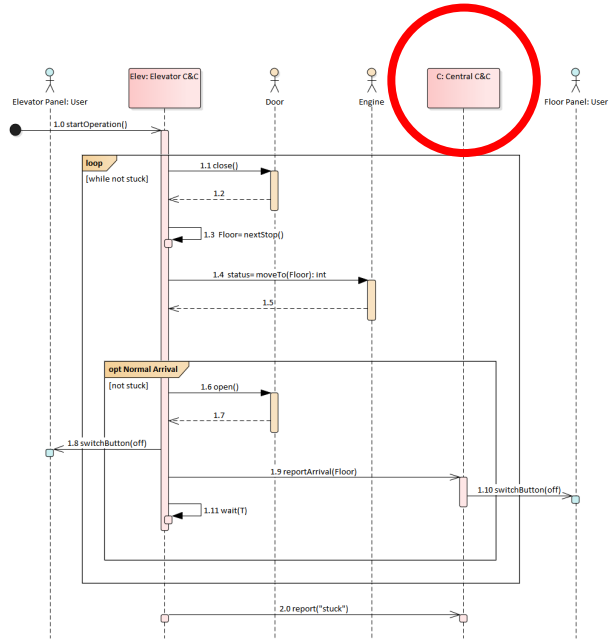
Example: Difference between one-time data request and repeat data transfer

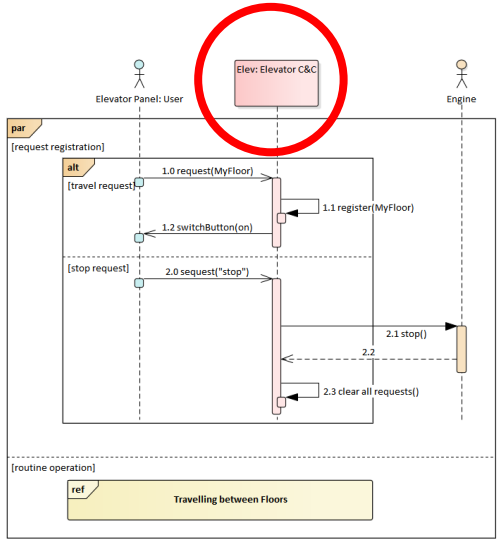
- X = Synchronous request of data
- Y = Choose data for repeated data transfer
- Z = Repeated data transfer

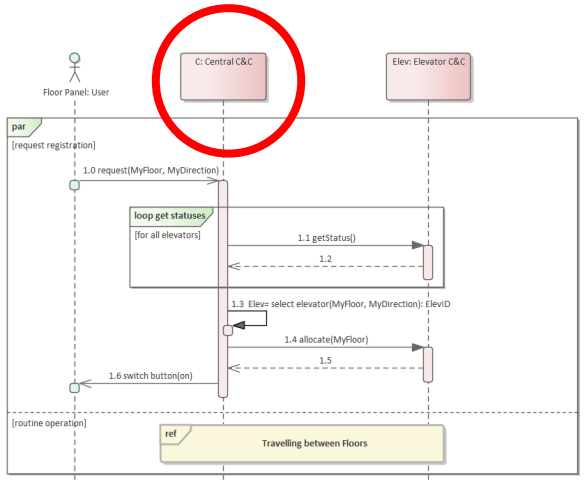


Defining interfaces for a single component

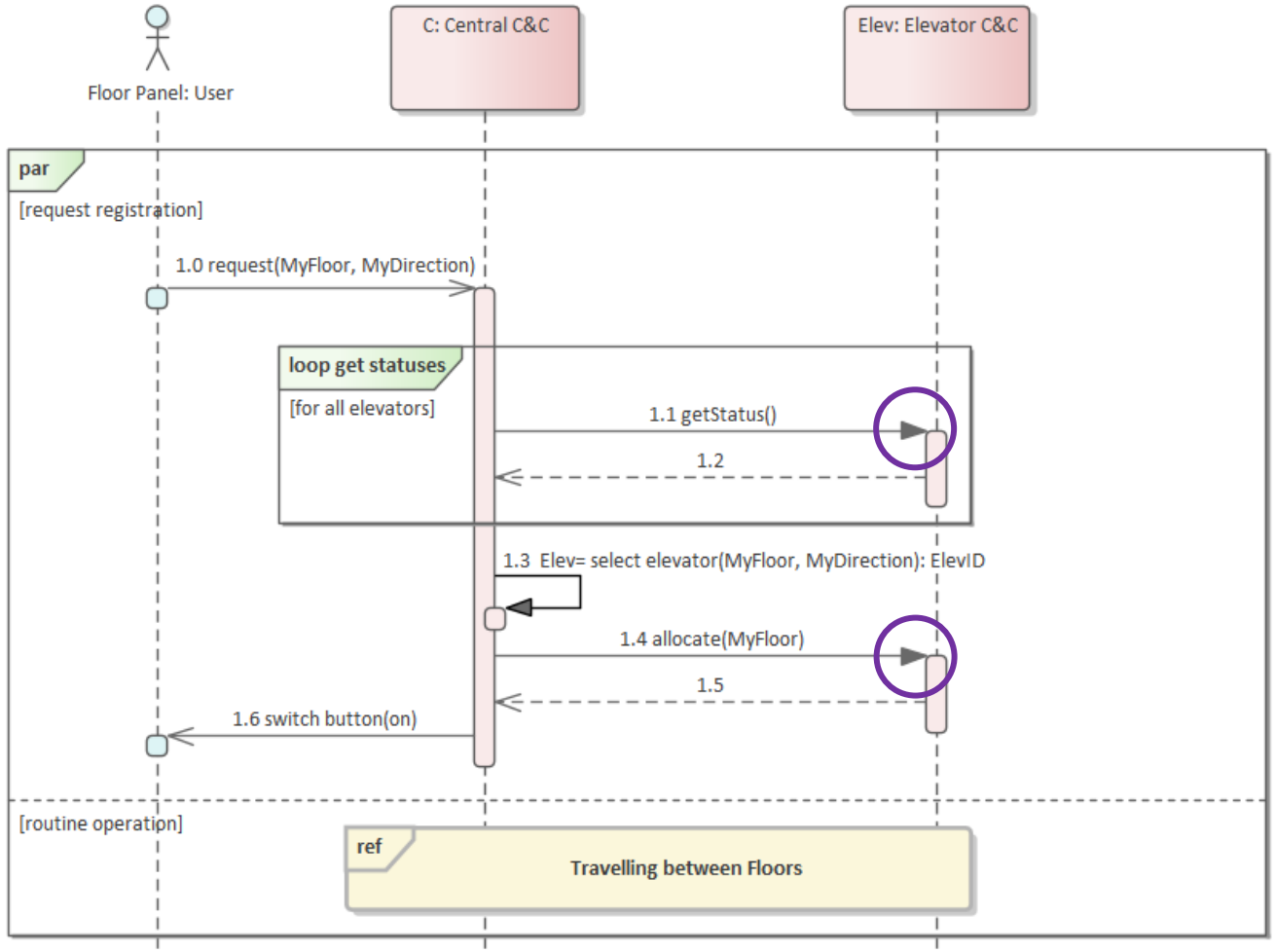
- Examine all the calls to and from the component in the sequence diagrams
  - Example: the Elevator C&C component in the elevator example

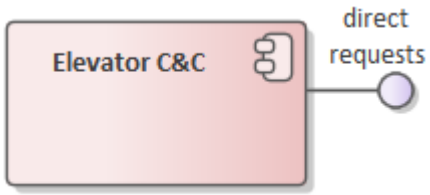






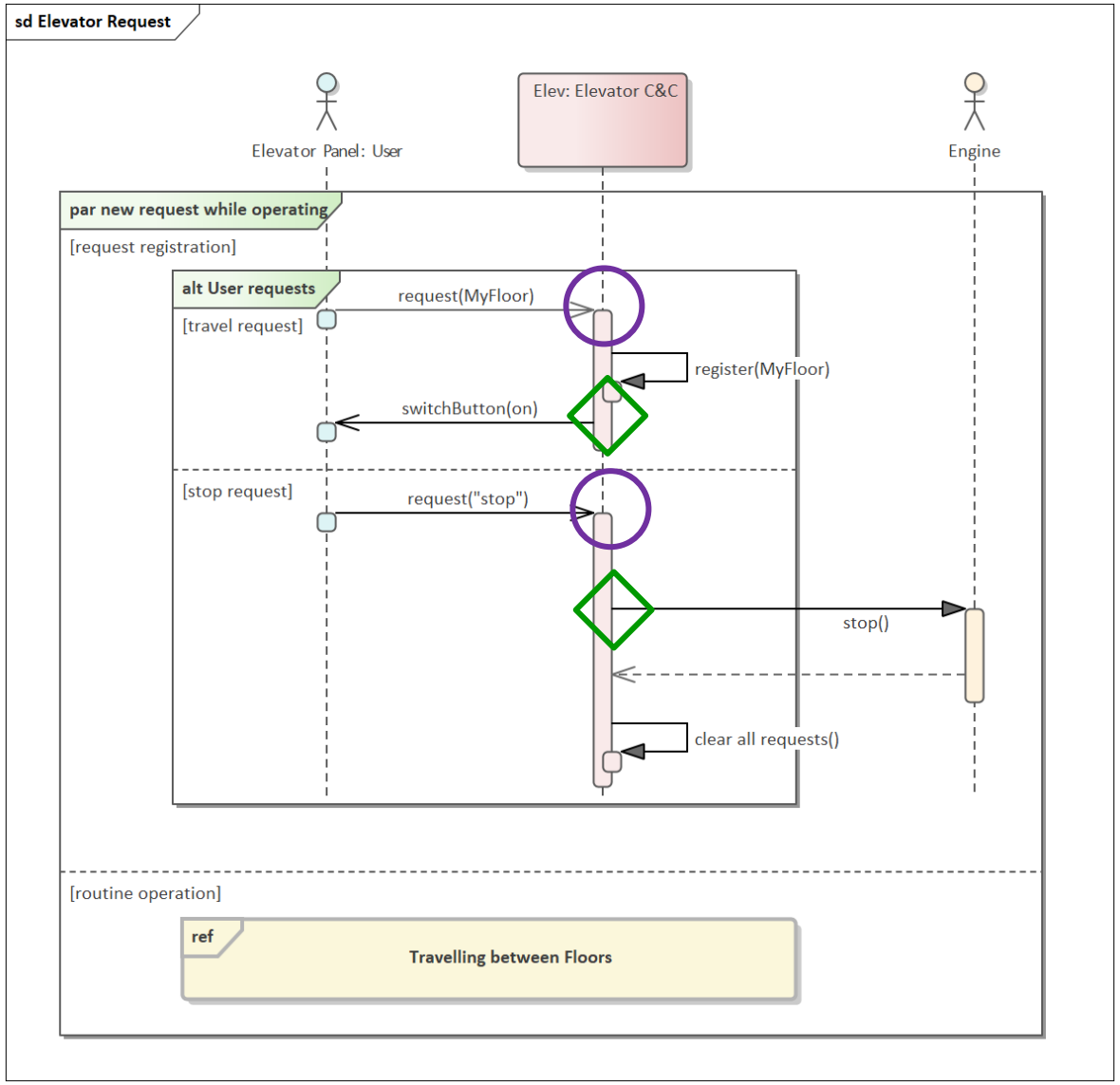
Elevator C&C Component – Sequence Diagram “Call Elevator”

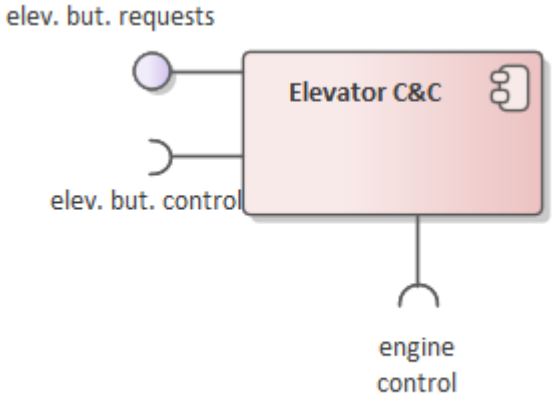




○ **Calls to the component**

Elevator C&C Component – Sequence Diagram “Ride Elevator”

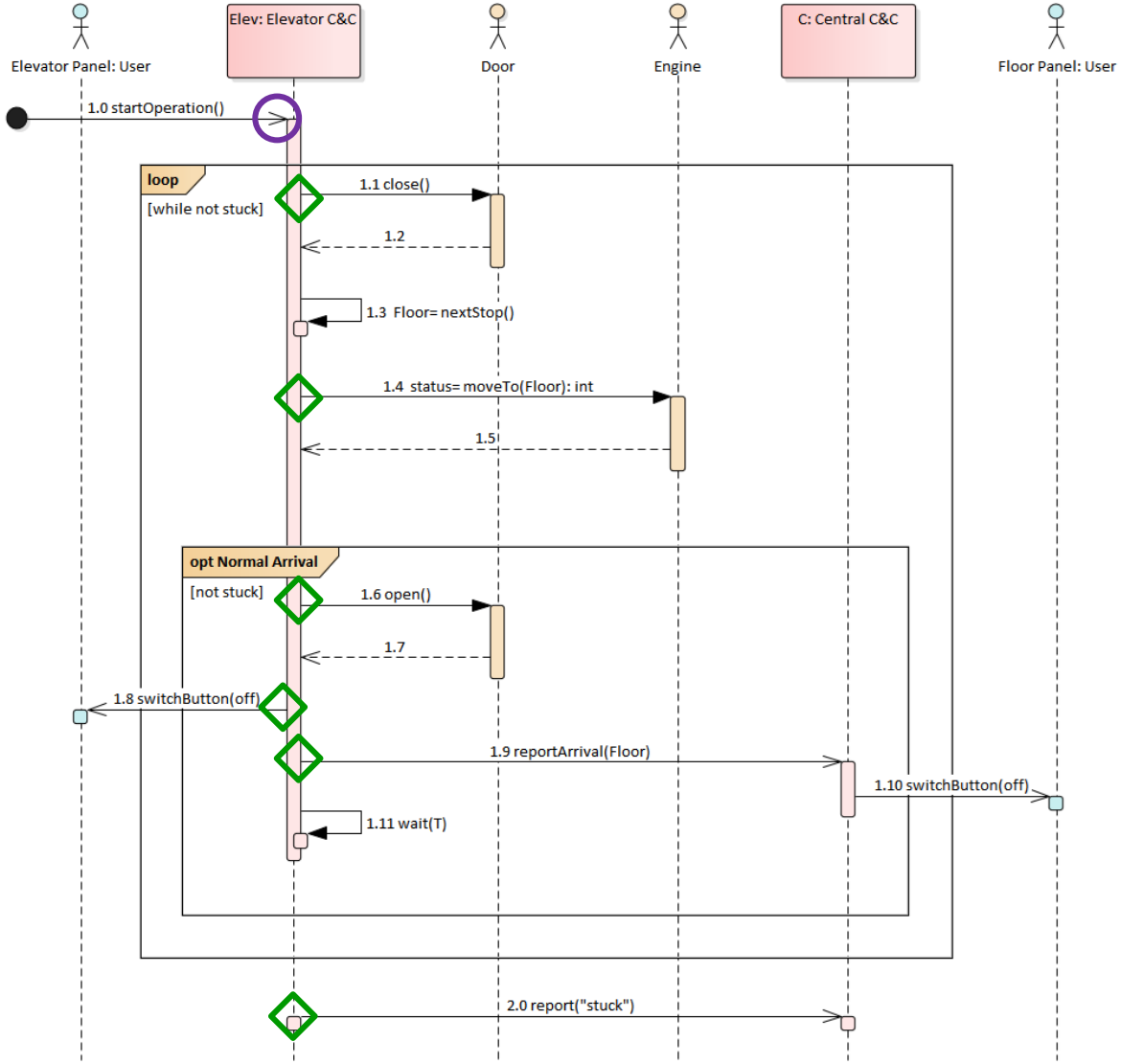


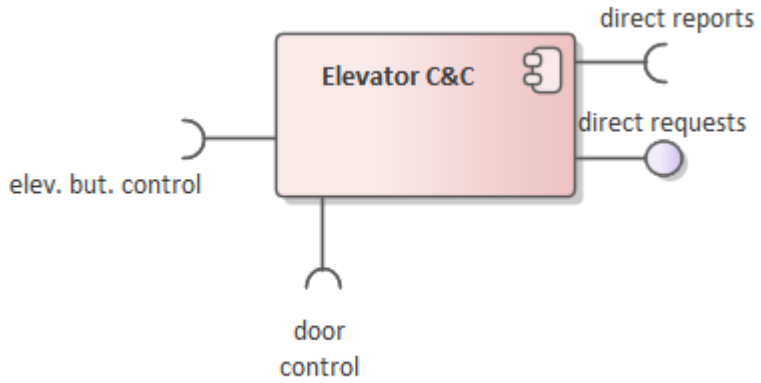


○ **Calls to the component**

◇ **Calls from the component**

Elevator C&C Component – Sequence Diagram “Travelling between floors”

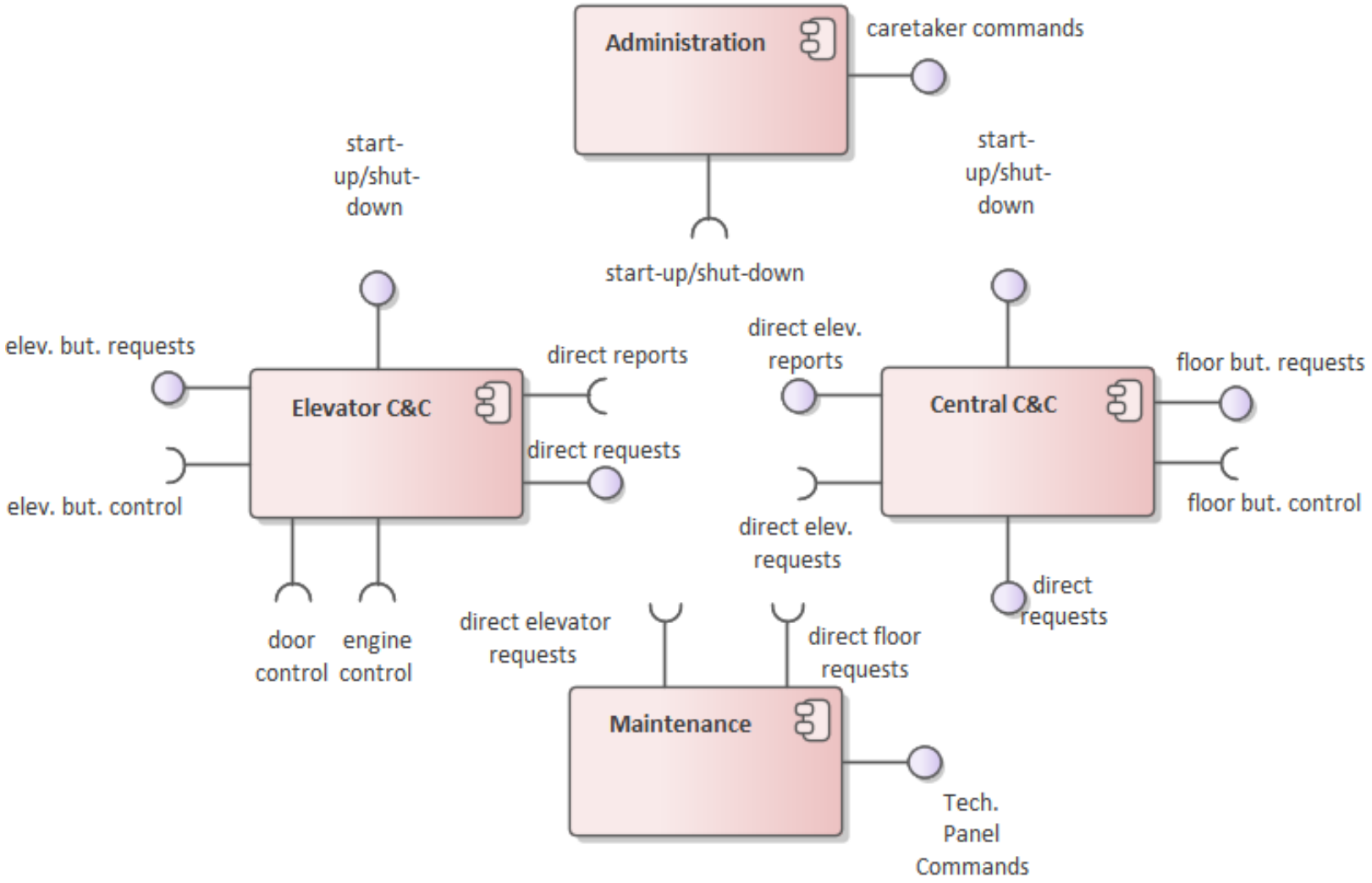




○ **Calls to the component**

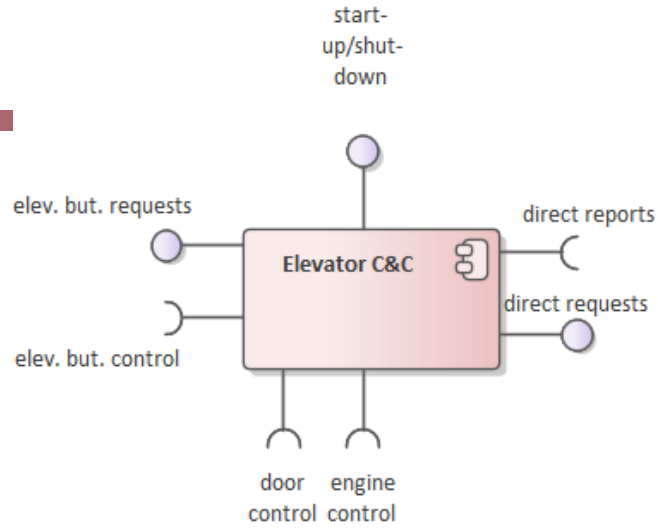
◇ **Calls from the component**

Elevator System: Functional components and interfaces



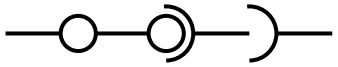
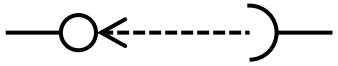
Component Documentation

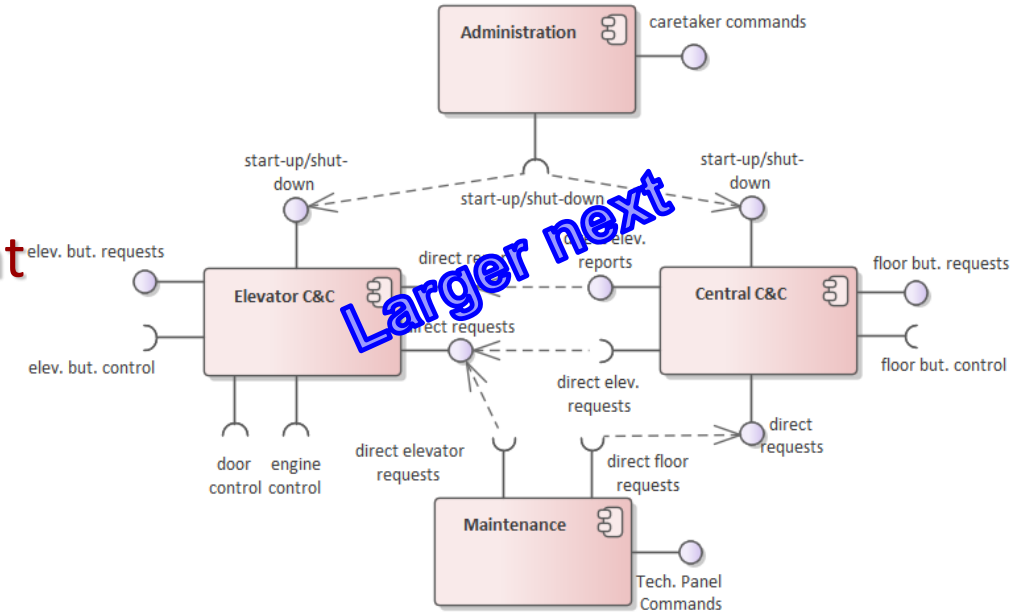
- Document every functional component



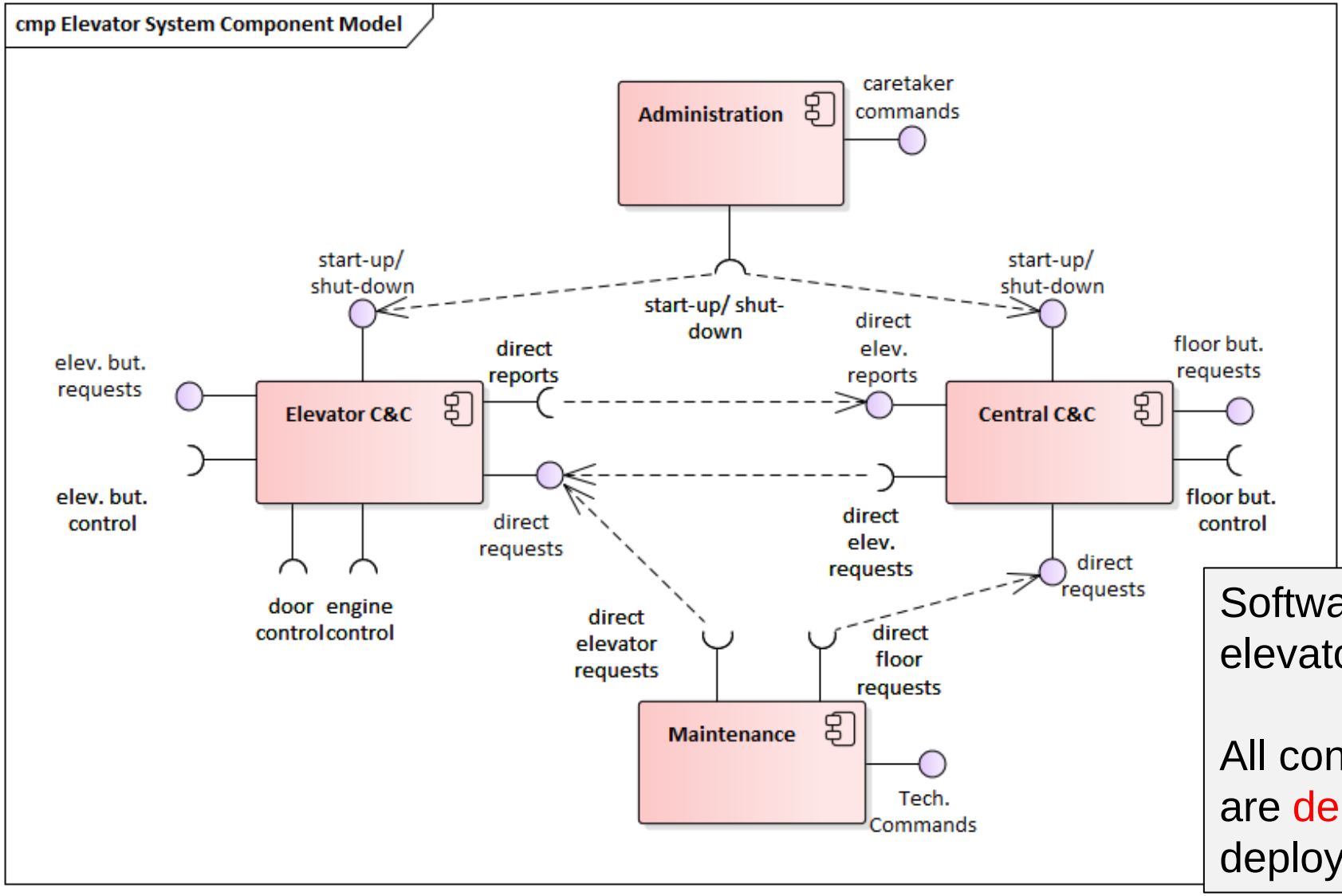
Component		Interfaces		
Name	Role	Name	Type	Service
Elevator C&C	Controls a single elevator car	elev. but. Requests	Provided	Receives user requests from buttons in the car
		Direct requests	Provided	Receives requests from the central command
		Start up/shut down	Provided	Starts up or shuts down the car
		Elev. But. Control	Required	Turns buttons on or off
		Door control	Required	Controls the doors
		Engine control	Required	Controls the engine
		Direct reports	Required	Reports elevator state

Software Architecture (Logical or Functional architecture)

- Functional components and the connections between them
  - Required connection: Between components that must be in the same hardware
    - Shown via an assembly connector 
  - Dependency: Between components that can be installed on separate hardware
    - Shown via a dependency 
- Unconnected interfaces are external environment facing interfaces



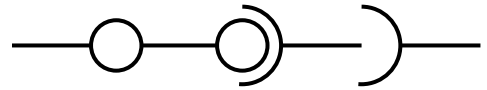
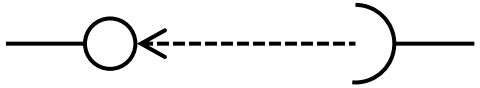
Software Architecture (Logical or Functional architecture)



Software architecture for the elevator system

All connections between interfaces are dependencies, giving maximal deployment flexibility.

Documenting the logical architecture

- In addition to the diagram, document each component in the architecture with its known details
  - Functional components and their roles
  - Functional interfaces for each component, their types, and what services they offer
- Document using text or within a modelling tool
- Connections between interfaces are shown in the figure
  - Assembly connector shows a connection that must be on the same hardware 
  - Dependencies show connections that are deployment-specific 
  - Unconnected interfaces interact with the external environment
    - More on this later

In Class Assignment: Software Architecture

- Build the software architecture for ePark
  1. Open the component diagram you built
  2. Open the sequence diagrams that you built based on the use cases
  3. For each component
     1. Find all **incoming** calls in the sequence diagrams and define **provided** interfaces for them
     2. Find all **outgoing** calls in the sequence diagrams and define **required** interfaces for them
  4. Connect all interfaces between components as appropriate

Conclusion

- Process Design
  - Sequence Diagrams
- Software Interfaces
  - Component interfaces
  - Logical architecture