SWE 621

FALL 2018

NOTATIONS FOR DESIGN

IN CLASS EXERCISE

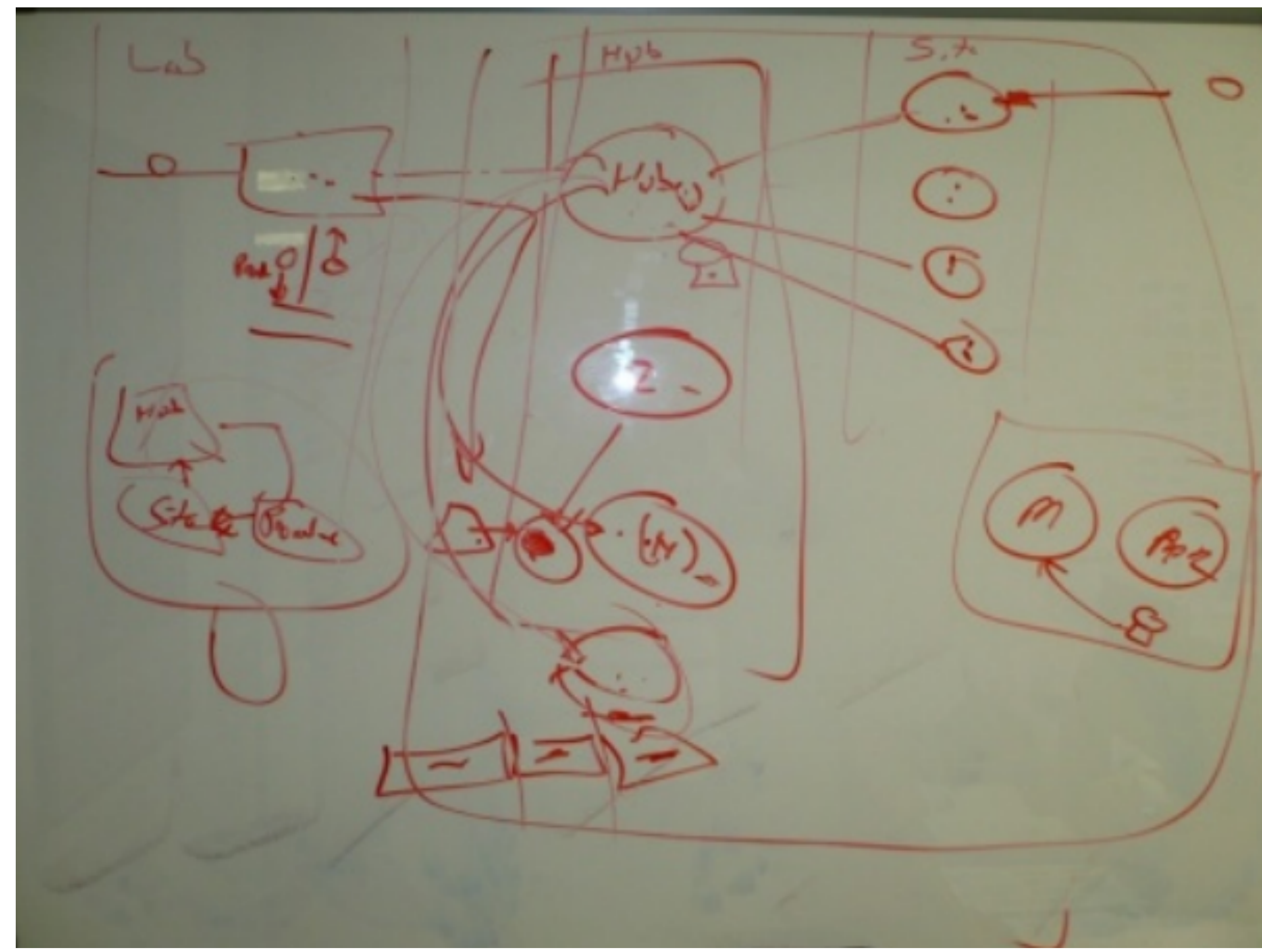
- What's your favorite software diagram? Why?

LOGISTICS

- HW2 due next week

DESIGN NOTATIONS

- Previously looked at domain modeling
    - Goal: understand the structure of **problem** domain
- This time: design modeling
    - Goal: understand the structure of **solutions**



NOTATIONS SUPPORTING DESIGN VS NOTATIONS FOR COMMUNICATION

- Design notations sometimes used a specification mechanism (e.g., model driven software engineering)
  - Goal is completeness.
  - Want notation to rigorously model system.
  - Might use model to generate code
- Also used as notation for exploring design space (this class)
  - Goal is examine alternative designs, interrogate design against specific scenarios, iterate design

HISTORY OF DESIGN NOTATIONS

- As focus changed between different design problems, notations changed with focus
- 1970s: function level design: flow charts, data flow diagrams
- 1990s: OO design: UML class diagrams, sequence diagrams
- 2000s: architecture: component and connectors

DESIGN NOTATIONS

- Offer views that show some aspects of your system and hide other
- Some important notation choices
  - Show one configuration of the system or all possible configurations
  - Show steps in a sequence of a process or snapshot
  - Show element as a black box or white box with internal visible

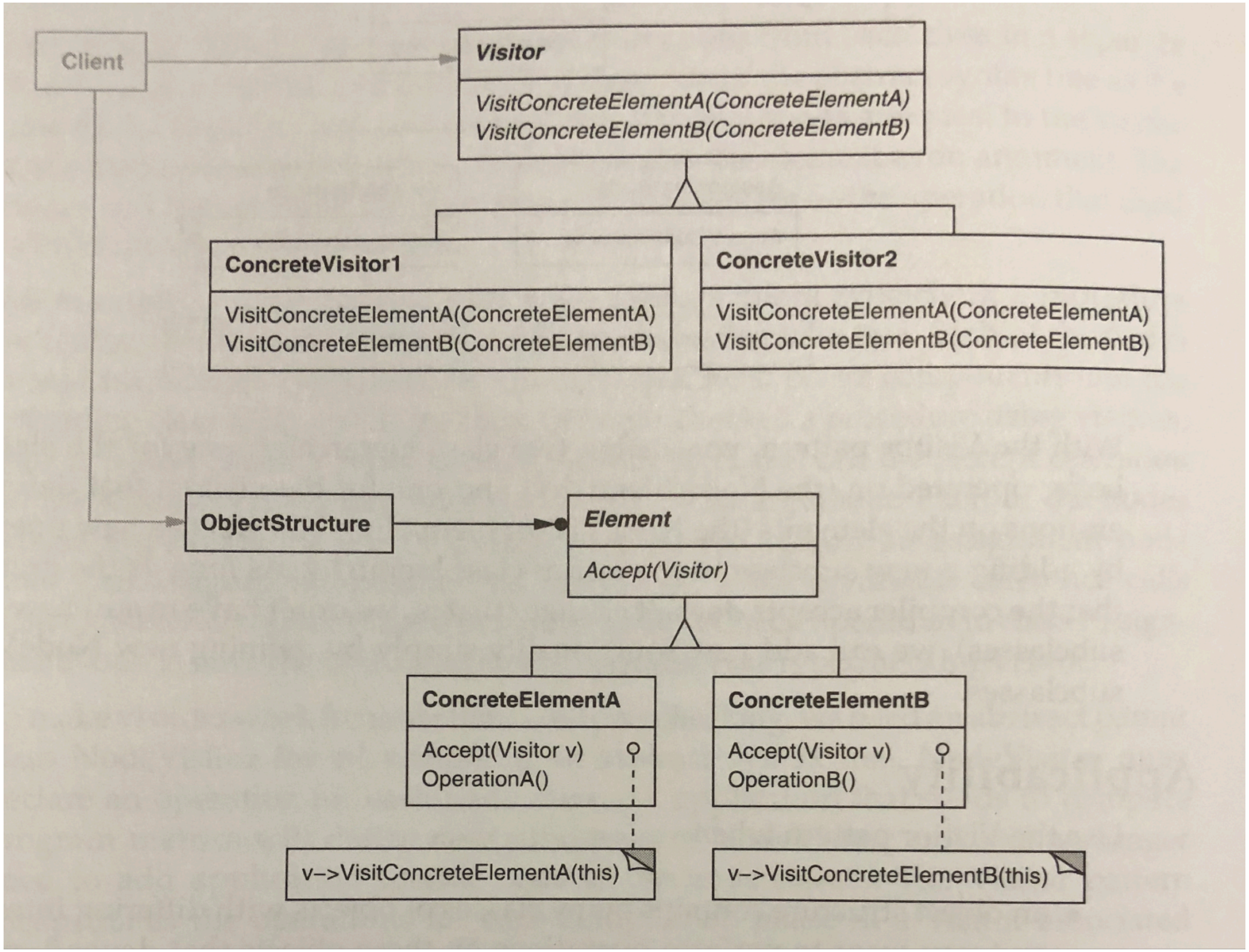
CHOOSING A DESIGN NOTATION

- Use notation that helps understand some aspect of design
  - What types of elements exist and how are they related to each other: class diagram
  - How data is passed between different elements in the system: data flow diagram
  - How objects interact to implement a scenario: sequence diagram
  - How a system transitions state as a result of interactions with environment: state chart

NOTATIONS FOR DESIGN

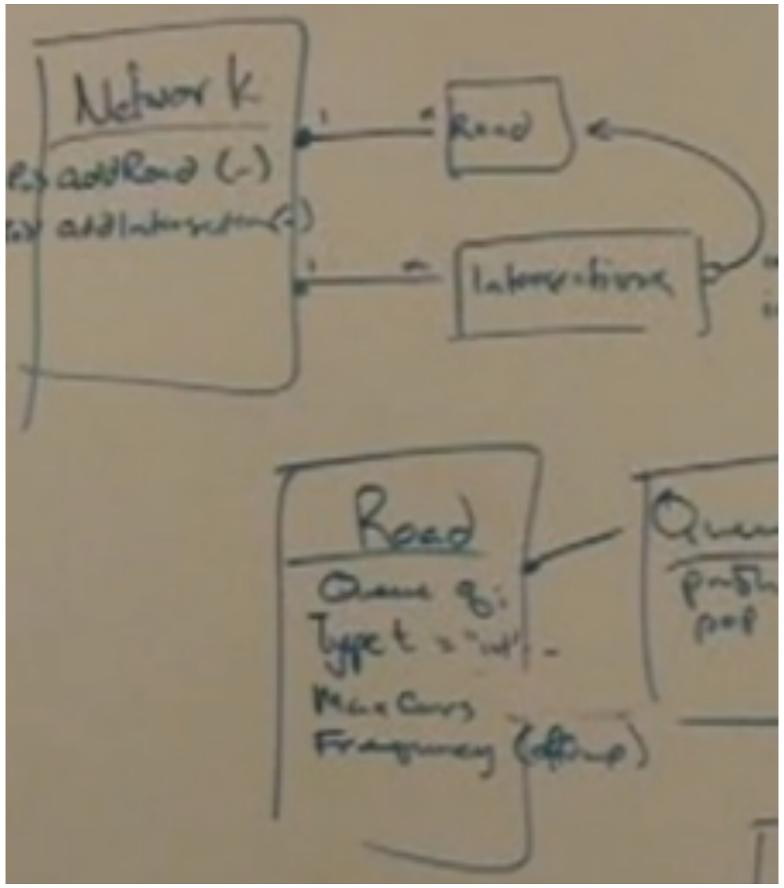
- Class diagrams
- Data flow diagram
- Sequence diagram
- Statecharts
- Component and connectors

CLASS DIAGRAMS

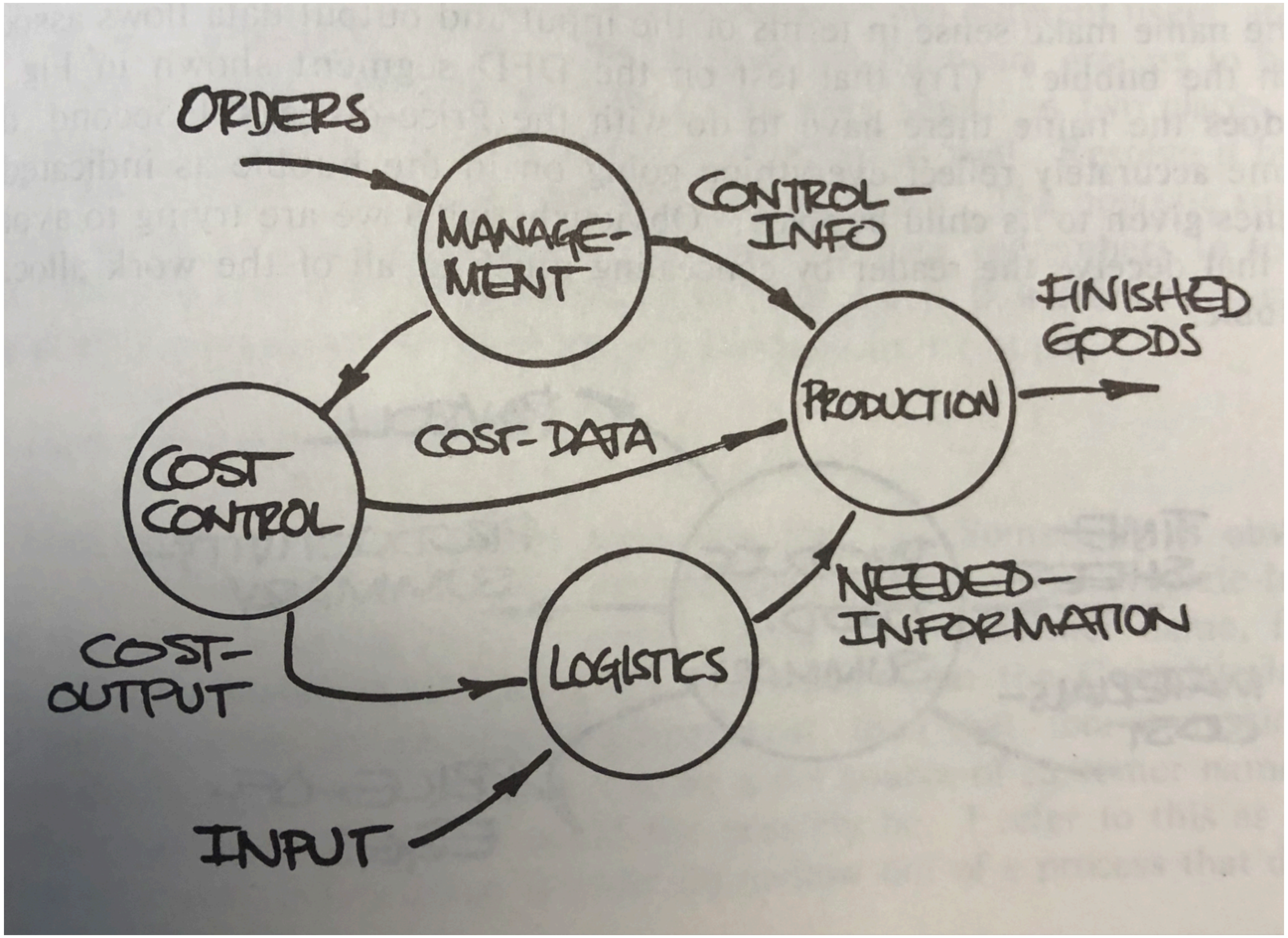


CLASS DIAGRAMS

- Class: a class is n the system
- Inheritance (lines between): class A inherits from class B
- Containment: class A has a collection of instances of class B



DATA FLOW DIAGRAMS

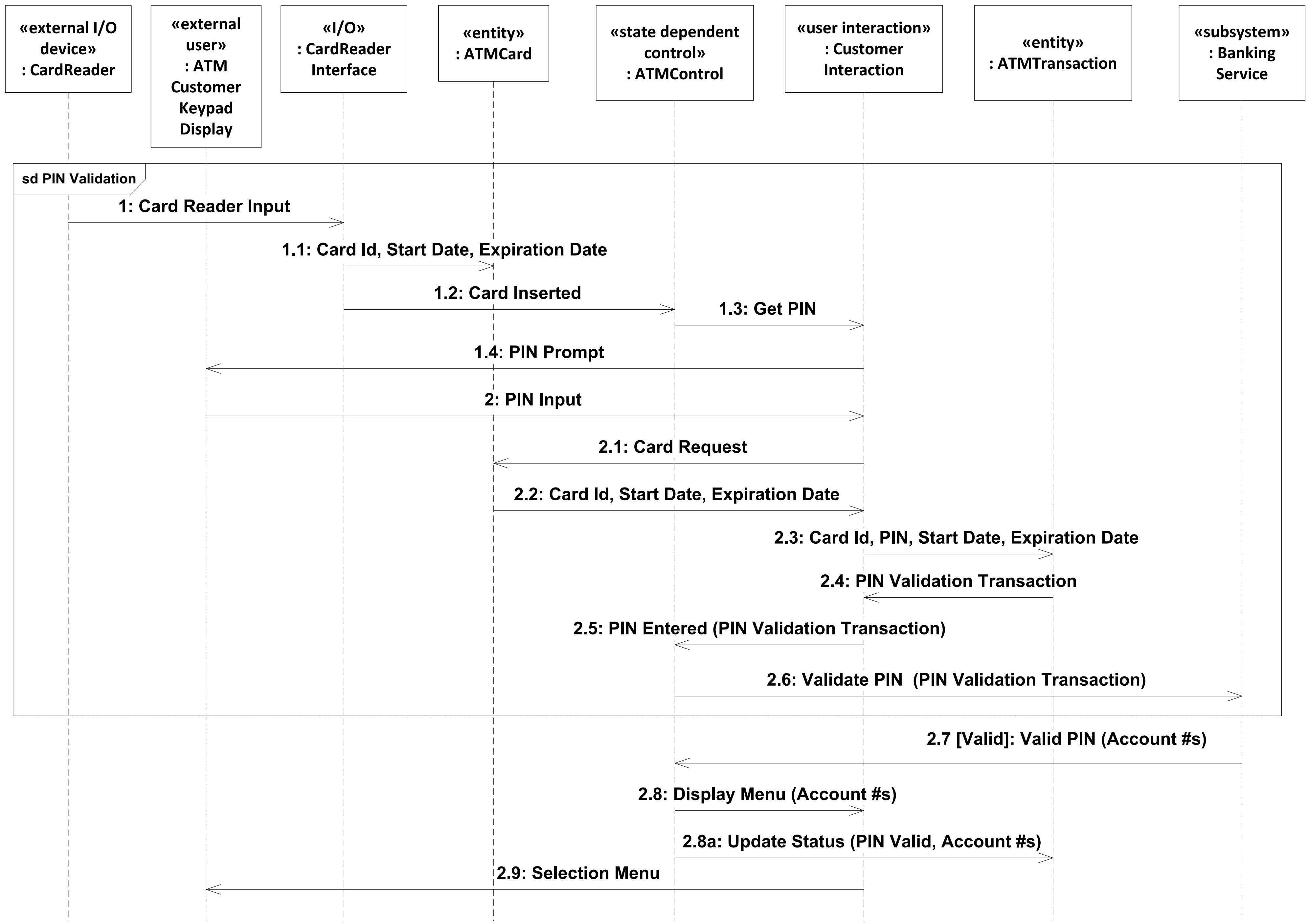


DATA FLOW DIAGRAMS

- Store or processing element (nodes): some system element that performs some computation
- Flow (edge): data that is sent from one element to another element

SEQUENCE DIAGRAMS: VALIDATE PIN

Figure 11.2 Sequence Diagram for Validate PIN use case – Valid Pin



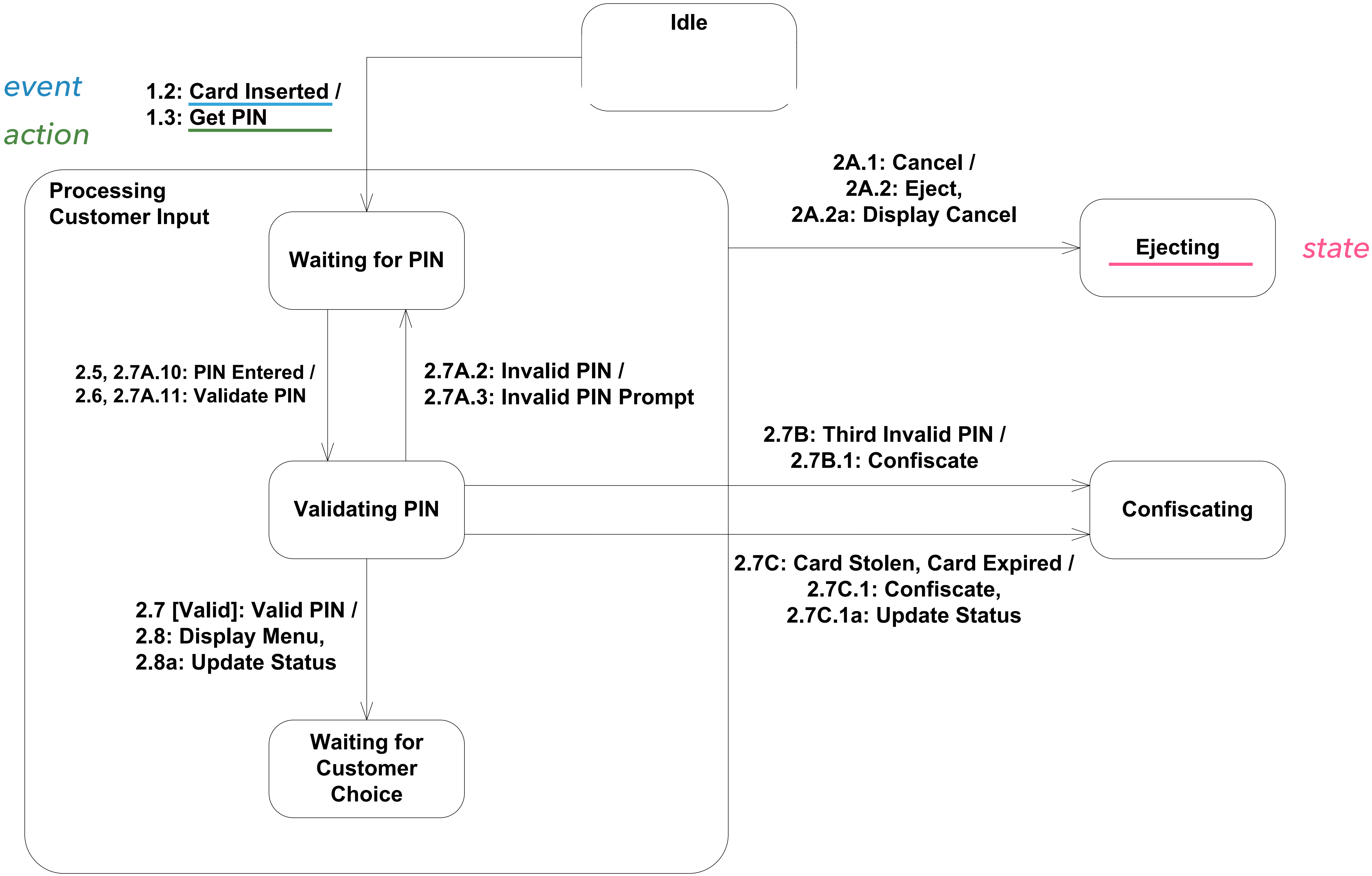
SEQUENCE DIAGRAMS

- Each object has a lifeline (vertical dashed line)
- Time flows from top to bottom
- Objects send messages to each other
- Can describe alternative sequences with labels on messages and boxes to group related behavior

IN CLASS ACTIVITY: COMMAND PATTERNS

- Encapsulate a request as an object
  - Enables queue or logging requests, and supports undoable operations
- Build a UML class diagram and sequence diagram describing command pattern

STATECHART



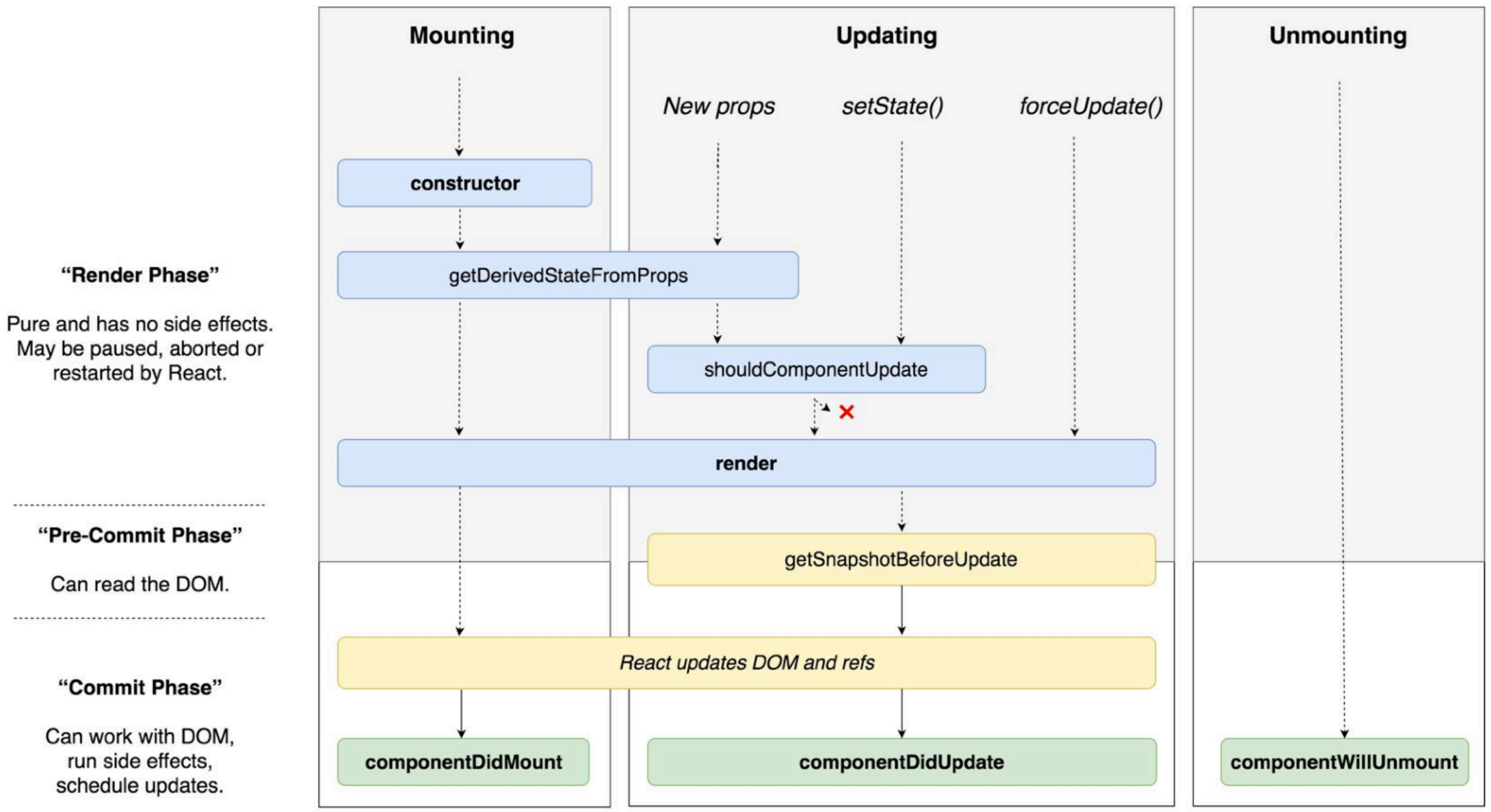
event
action
1.2: Card Inserted /
1.3: Get PIN
Idle
Processing
Customer Input
Waiting for PIN
2.5, 2.7A.10: PIN Entered /
2.6, 2.7A.11: Validate PIN
2.7A.2: Invalid PIN /
2.7A.3: Invalid PIN Prompt
Validating PIN
2.7 [Valid]: Valid PIN /
2.8: Display Menu,
2.8a: Update Status
Waiting for
Customer
Choice
2A.1: Cancel /
2A.2: Eject,
2A.2a: Display Cancel
Ejecting
state
2.7B: Third Invalid PIN /
2.7B.1: Confiscate
Confiscating
2.7C: Card Stolen, Card Expired /
2.7C.1: Confiscate,
2.7C.1a: Update Status

STATECHARTS

- State: a recognizable situation that exists over an interval of time
- Event: input to state machine that causes transition
- Action (optional): output generated by system when state transition occurs

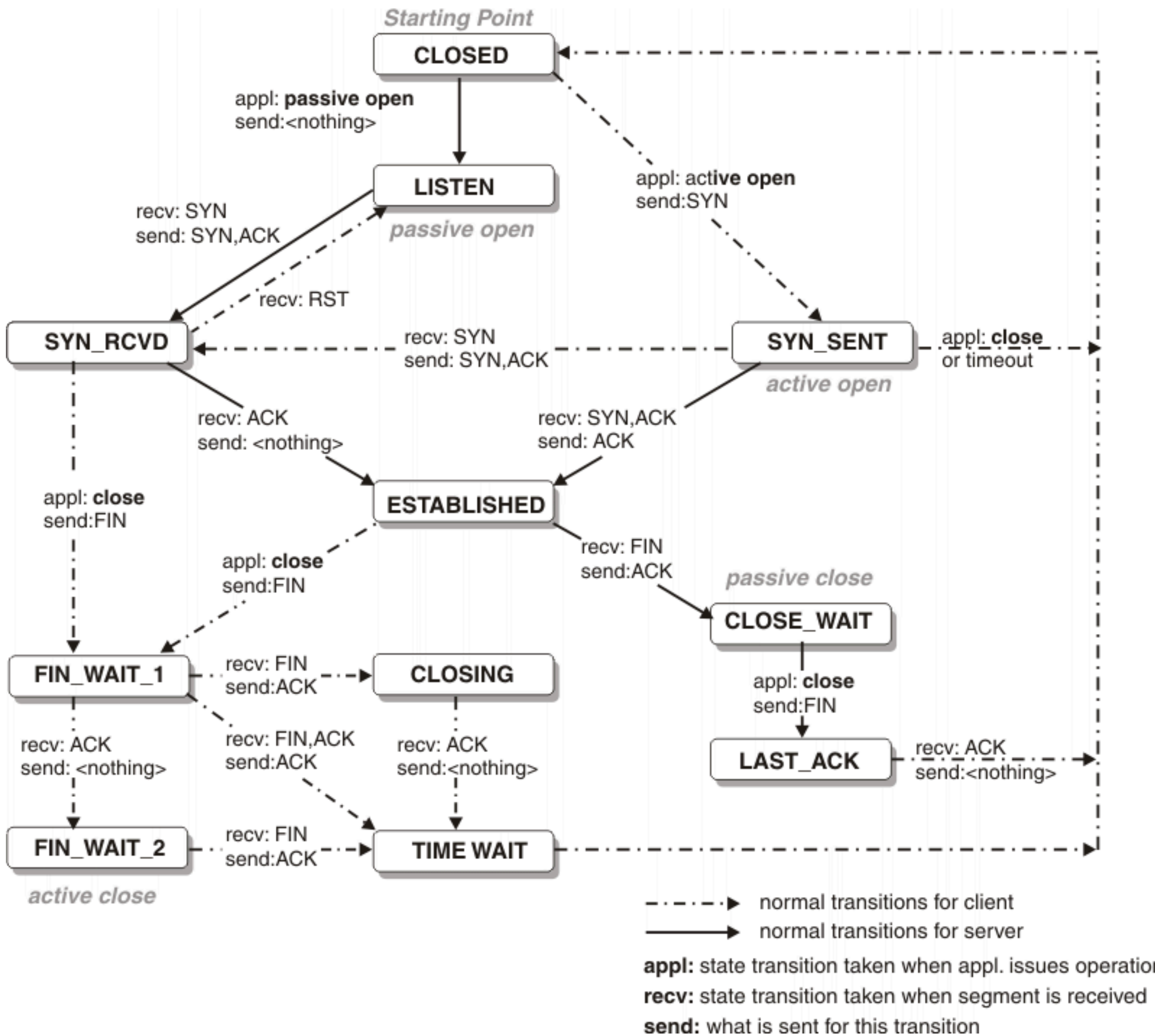
LIFE CYCLES

- Elements may change over time, with different operations and state available depending on which step an element is in

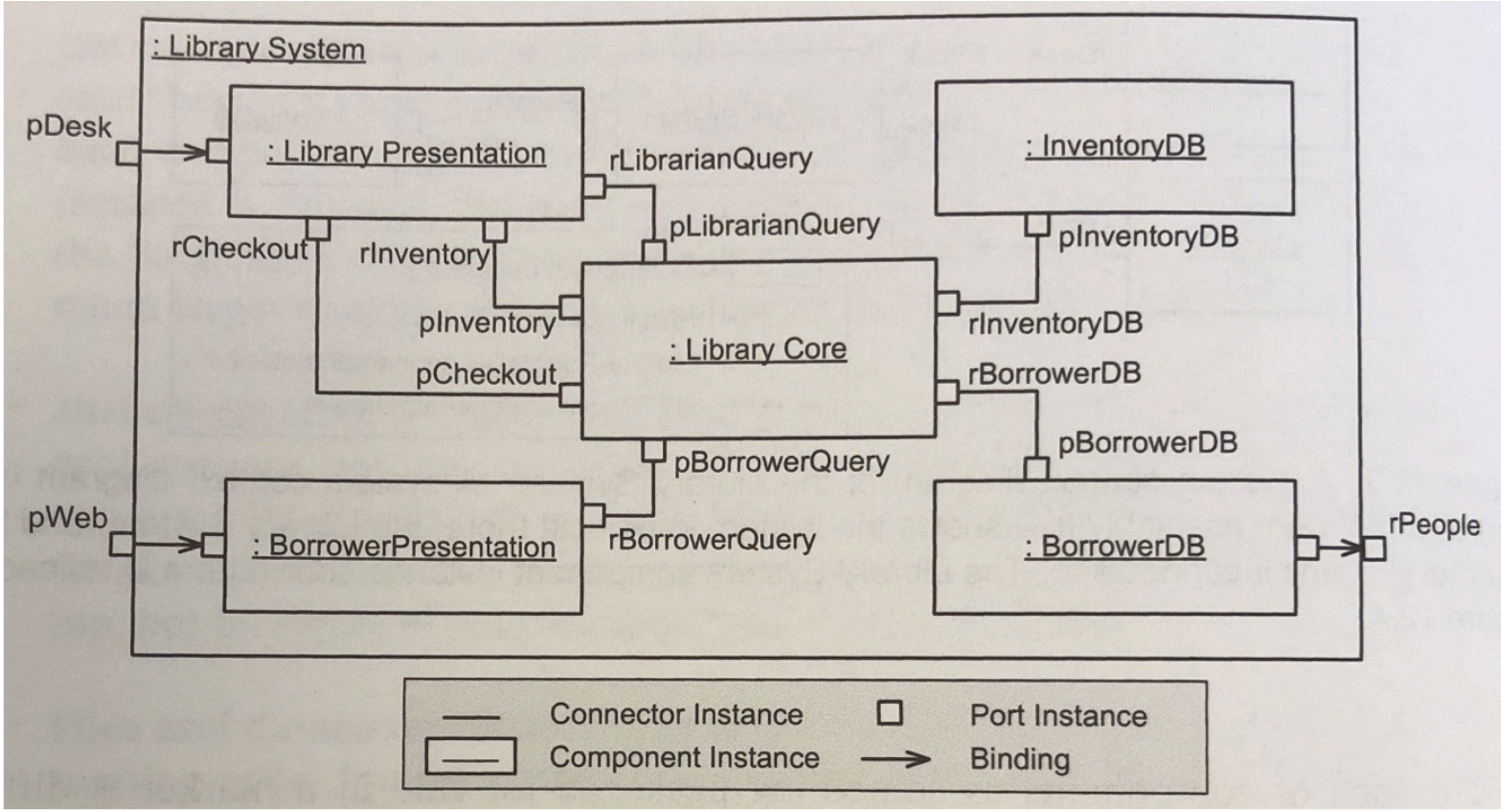


LIFECYCLES

- State charts can be used to depict the lifecycle of an element



COMPONENTS AND CONNECTORS



COMPONENTS AND CONNECTORS

- Components: the principle computation elements and data stores that execute in a system; instance, not type
- Connector: a runtime **pathway of interaction** between two or more components
- Port: communication that occurs into or out of a component

CONNECTORS

- Simplest example: method call
- But also **any** other way components interact

Connector type	Notes
Local procedure call	Most common connector when components are all in the same memory space.
Remote procedure call	Concrete examples include SOAP and HTTP requests. Both local and remote procedure call connectors are kinds of request-reply connectors.
SQL or other datastore	Declarative language used to load/store data.
Pipe	Simple producer-consumer relationships between components.
Shared memory	Fast but complex communication.
Event broadcast	Consumers depend only on events, not on producers.
Enterprise bus	Standardizes intra-application communication for assembly of large systems.
Data drop	Distribution mechanism for shared data from single source.
Incremental replication	Handles state synchronization.

PORTS

- Could be a group of related public methods
  - e.g., a Java Element implements IRunnable interface, which becomes a port
- Could be a communication modality
  - e.g., interaction that HTTP requests, database, event system

IN CLASS ACTIVITY: WORD PROCESSOR

- Imagine a word processor with the following functionality
  - Users can enter and format text
  - Users can insert images, which can also be manipulated through formatting commands
  - Formatting commands are invoked through menus, toolbars, dialogs and update text
  - Users can undo and redo commands
- Describe a design for this system using a component and connector diagram

INTERROGATING A DESIGN MODEL

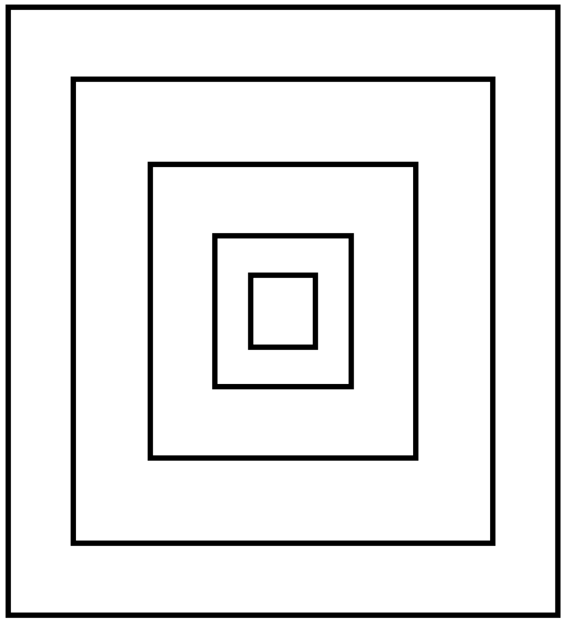
- Just like domain model, want to understand if a design supports a scenario
- As you simulate each step in the scenario, does your design still work?
  - Are there additional elements or relationships you should add?
  - Is there a way to your design can support the scenario more simply?

USING DESIGN NOTATIONS

- Notation offers a view with which to see design
  - Key choice is what part of design do you want to focus on
- Modeling activities can be driven by risk
  - What are you worried about **not** working
  - What do you need to model to reduce this risk
  - Might be possible to reduce by building a model; or by building an implementation

CHOICE OF ABSTRACTION LEVEL

- Systems are hierarchic, where elements contains elements which contain elements
- How deep you choose to go should depend on what you are trying to model and understand it
  - If you don't need some detail, don't include it!
  - May end up with very different models of the same thing depending on what you are trying to understand about it



NOTATIONS AS STARTING POINTS

- If you need to express something that's not in your modeling notations, it's ok to create new notation
- Can change visual variables of marks to communicate information
  - e.g., color, shape, dashing
  - Black edges are method call connectors and green edges are HTTP request connectors
- Can add annotations (i.e., text) to elements or relations to explain constraints or decisions
  - e.g., this port is only available after system initialization

SUMMARY

- Design notations help to think through a design
- Many choices about what to show
  - One config of the system or all
  - Steps in a sequence of a process or snapshot
  - Element as a black box or with internals visible
- Often start with a scenario or risk, want to understand how to support the scenario or reduce the risk through a design

IN CLASS ACTIVITY

DESIGN ACTIVITY: PLUGIN ARCHITECTURE

- Your goal: design a plugin architecture for a drawing application
- In a plugin architecture, plugins are
  - written by third parties (i.e., not you)
  - dynamically loaded at runtime into your application
  - invoked through an interface, without knowing anything about implementation
- Deliverables:
  - component and connector model showing elements in your system and where plugins can connect
  - state chart describing the lifecycle of a plugin

DESIGN ACTIVITY: STEP 2

- Join together with another group

- Compare your models.
- What assumptions contributed to the differences, if any? How are you solutions similar or different?