Requirements Modeling: Behavior-Based

Prof. Alex Bardas

Behavioral Modeling

- Models the dynamics of the system
- Represents the behavior of the system as ***a function of events and time***
- Indicates how software will respond to events
  - Information being exchanged is not the essential part of the behavioral model but the fact that information has been exchanged

Identify Events

- Recap: use case
  - *A sequence of interaction* involving actors and the system
  - An event occurs when an actor and the system interact
  - Use case diagrams, activity diagrams, swimlane diagrams

    e.g., UC-1 of the secure home access case study

Use Case 1: Unlock

Use Case UC-1:	**Unlock**
Related Requirements:	REQ1, REQ2, REQ3 and REQ4
Primary Actor:	Any of: Tenant, Landlord
Actor's Goal:	To disarm the lock and get space lighted up automatically
Secondary Actors:	LockDevice, LightSwitch, Timer
Preconditions:	• The set of valid keys stored in the system database is non-empty • The system displays the menu of available functions • At the door keypad the menu choices are "Lock" and "Unlock"
Post conditions:	The auto-lock timer has started count down from autoLockInterval
Flow of Events for Main Success Scenario:	
→	1. **Tenant/Landlord** arrives at the door and selects the menu item "Unlock"
	2. include::*AuthenticateUser* (UC-7)
←	3. **System** (a) signals to the **Tenant/Landlord** the lock status, e.g., "disarmed," (b) signals to **LockDevice** to disarm the lock, and (c) signals to **LightSwitch** to turn the light on
←	4. **System** signals to the **Timer** to start the auto-lock timer countdown
→	5. **Tenant/Landlord** opens the door, enters the home [and shuts the door and locks]

Flow of Events

- UC-1
  - Tenant arrives at the door and selects the menu item “Unlock”
  - Include::AuthenticateUser (UC-7)
    - System prompts the actor for identification
    - Tenant supplies a valid key
    - System (a) verifies the key is valid; and (b) signals to actor the validity of key
  - …

Flow of Events

- UC-1
  - Tenant arrives at the door and selects the menu item “Unlock”
  - Include::AuthenticateUser (UC-7)
    - System prompts the actor for identification
    - Tenant supplies a valid key
    - System (a) verifies the key is valid; and (b) signals to actor the validity of key
  - …
  - Some events have direct impact on the flow of control
    - Key checked

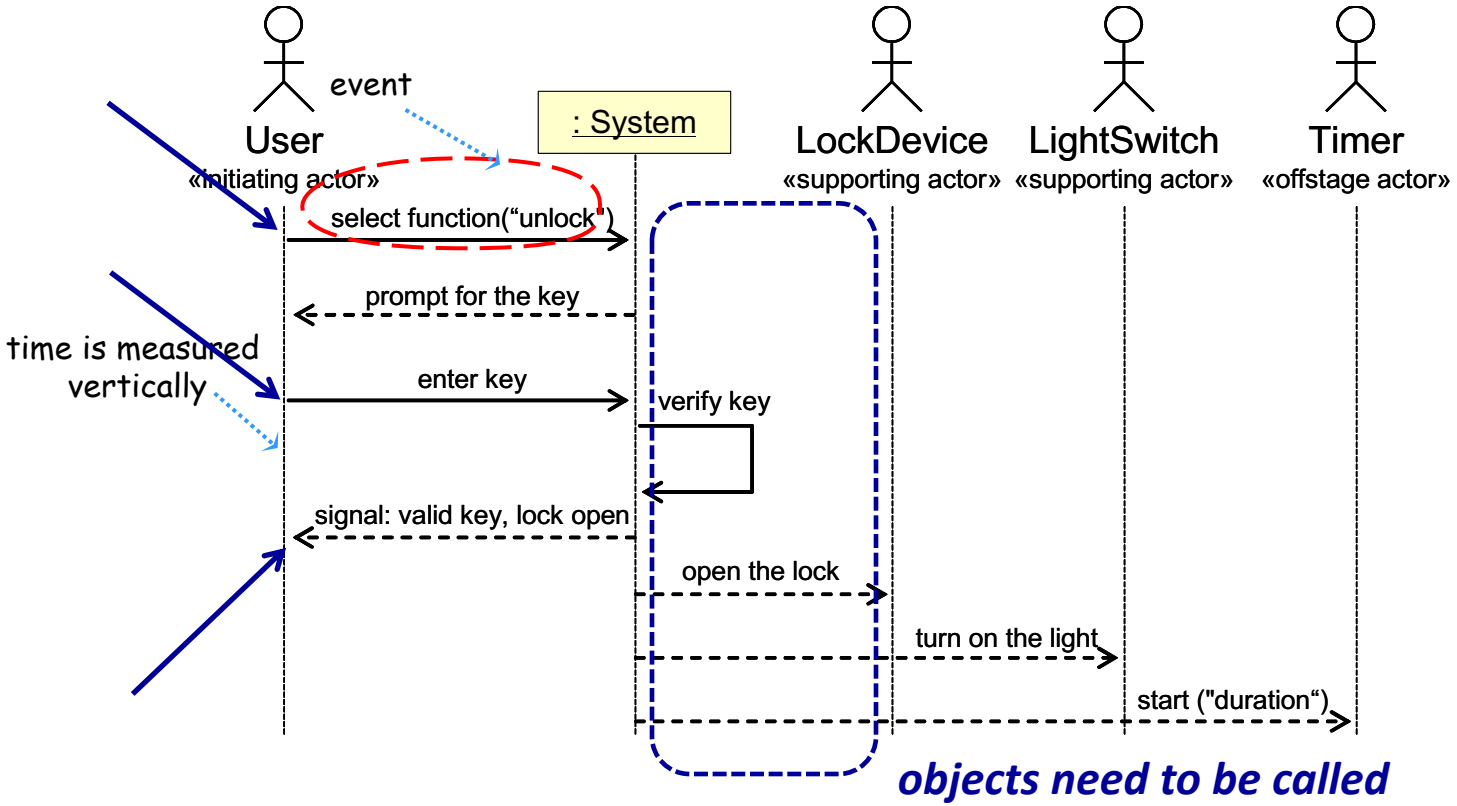
Sequence of Events

- Allocating events to objects
  - Tenant object is with key entered event
  - Objects may generate events
  - Or, objects may recognize events: key checked

System Sequence of Events

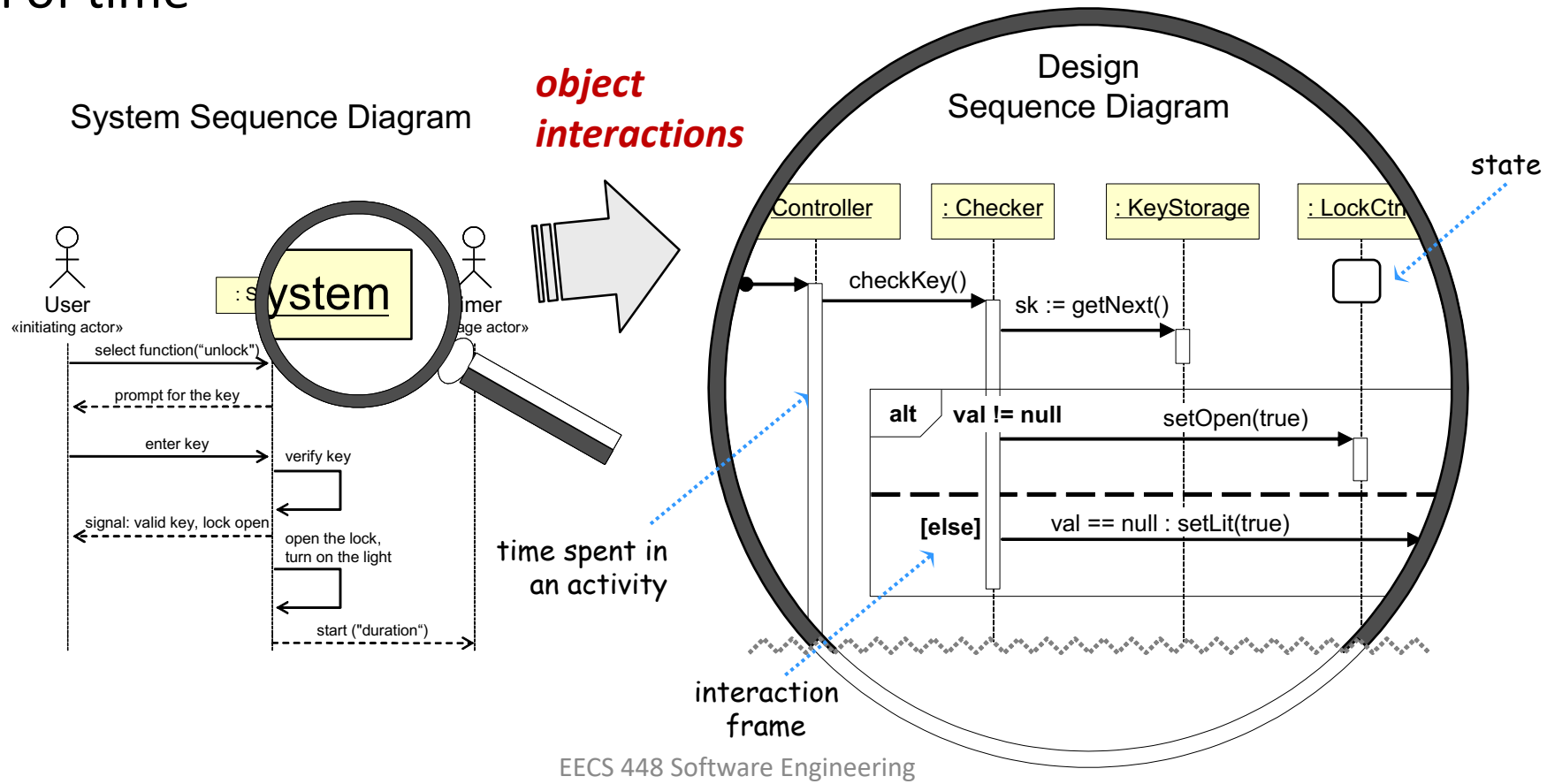
System sequence of events of UC-1

Depicts actor interactions instead of object interactions



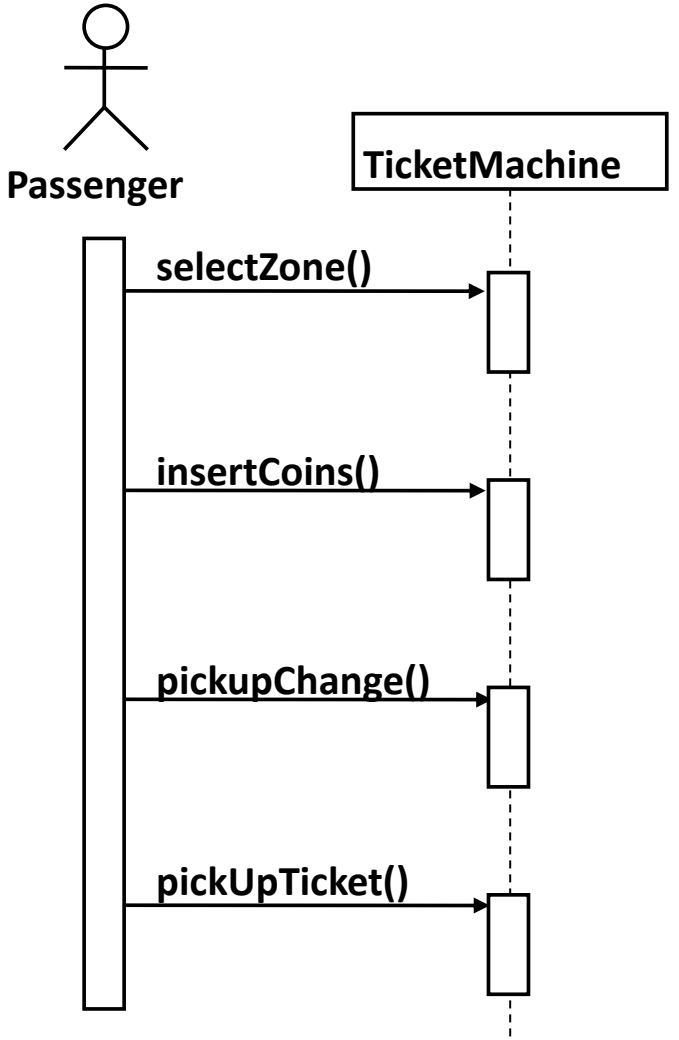
Sequence Diagram

- Captures how events cause the flow from one object to another object as a function of time



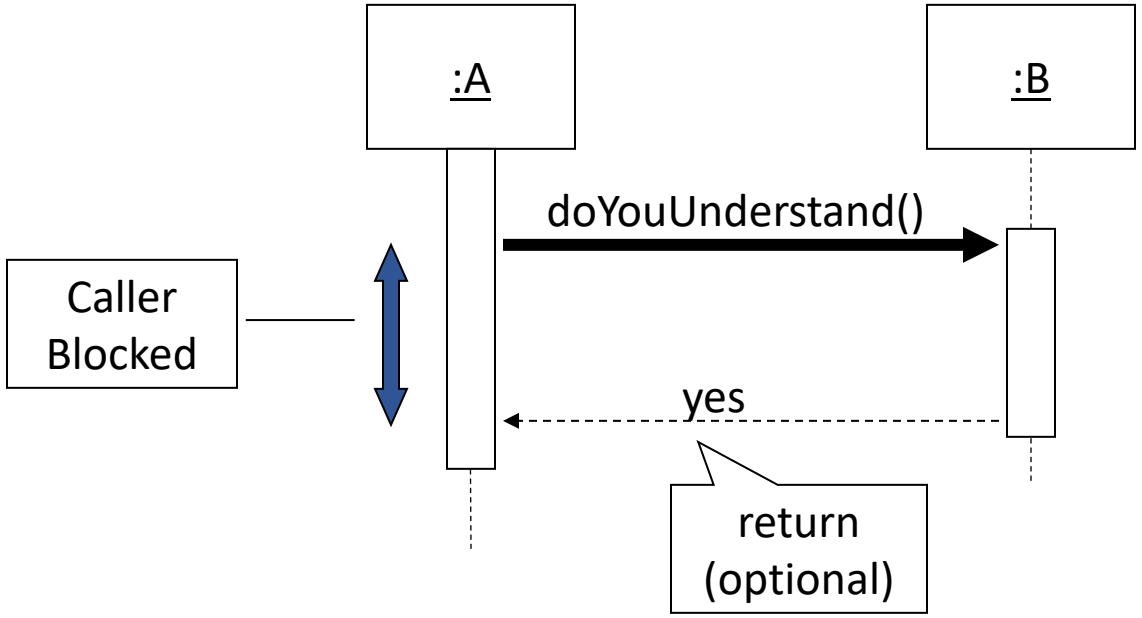
Sequence Diagram

- Objects: columns
- Messages: arrows
- Activations: narrow rectangles
- Lifelines: dashed lines



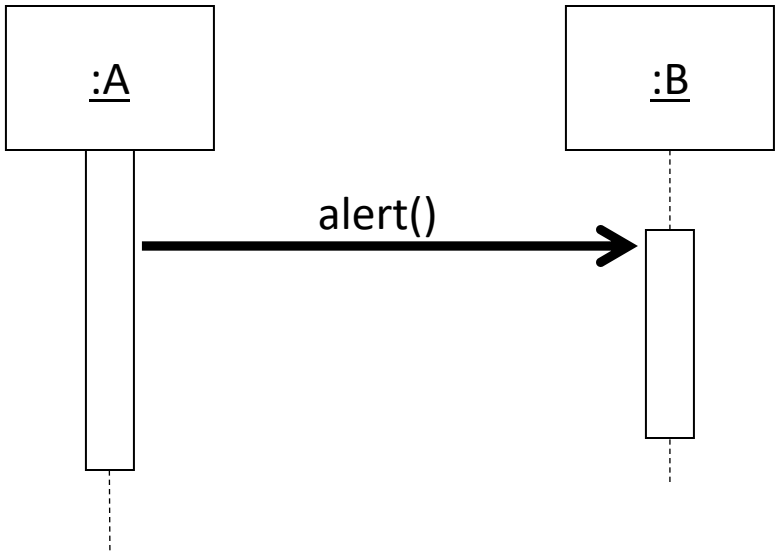
Sequence Diagram

- Synchronous message
  - The routine that handles the message is completed before the caller resumes execution



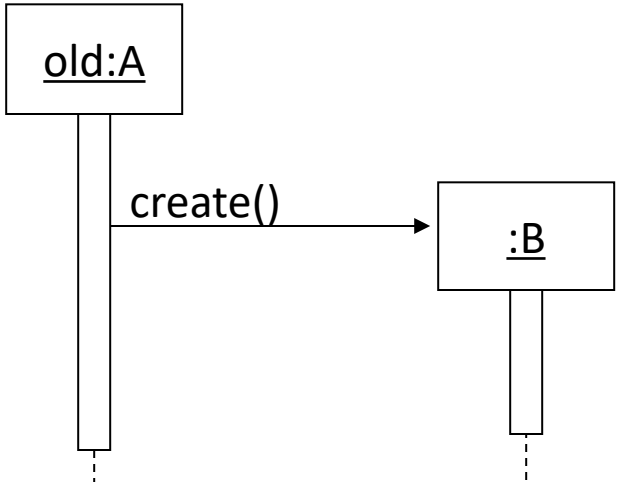
Sequence Diagram

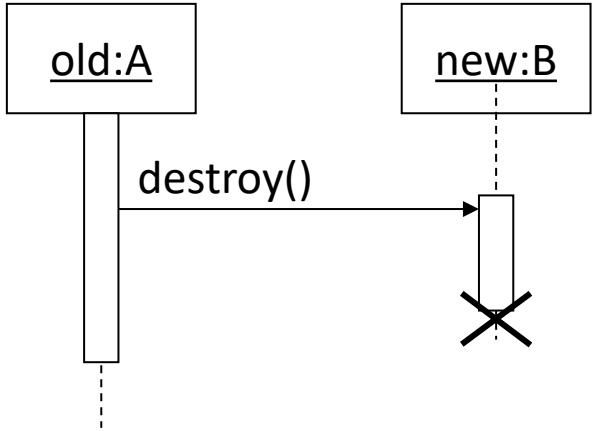
- Asynchronous message
  - Sender does not wait for the receiver to finish processing the message
  - Continues immediately



Sequence Diagram

- Message creation
  - Denoted by a message arrow pointing to the object
- Message destruction
  - Denoted by an X mark at the end of the destruction activation

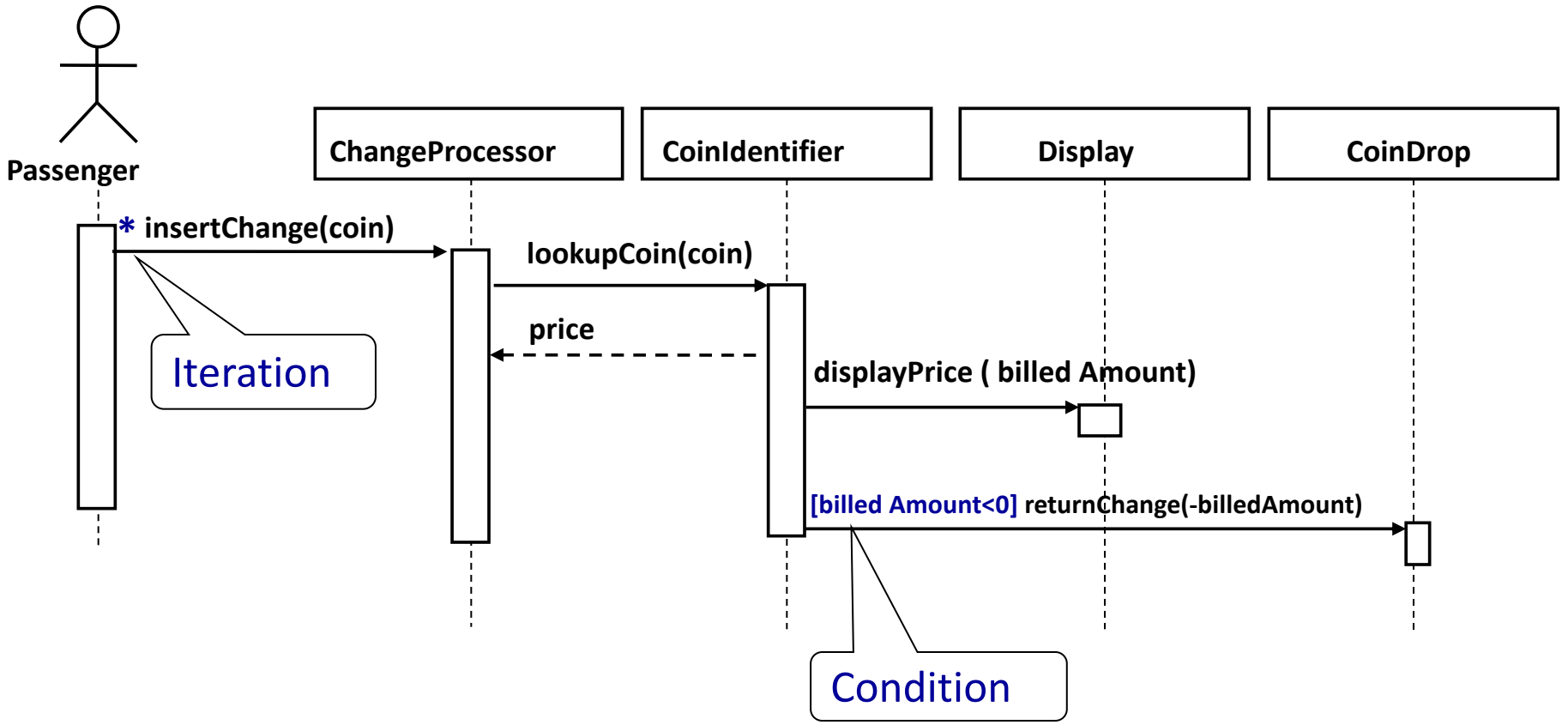




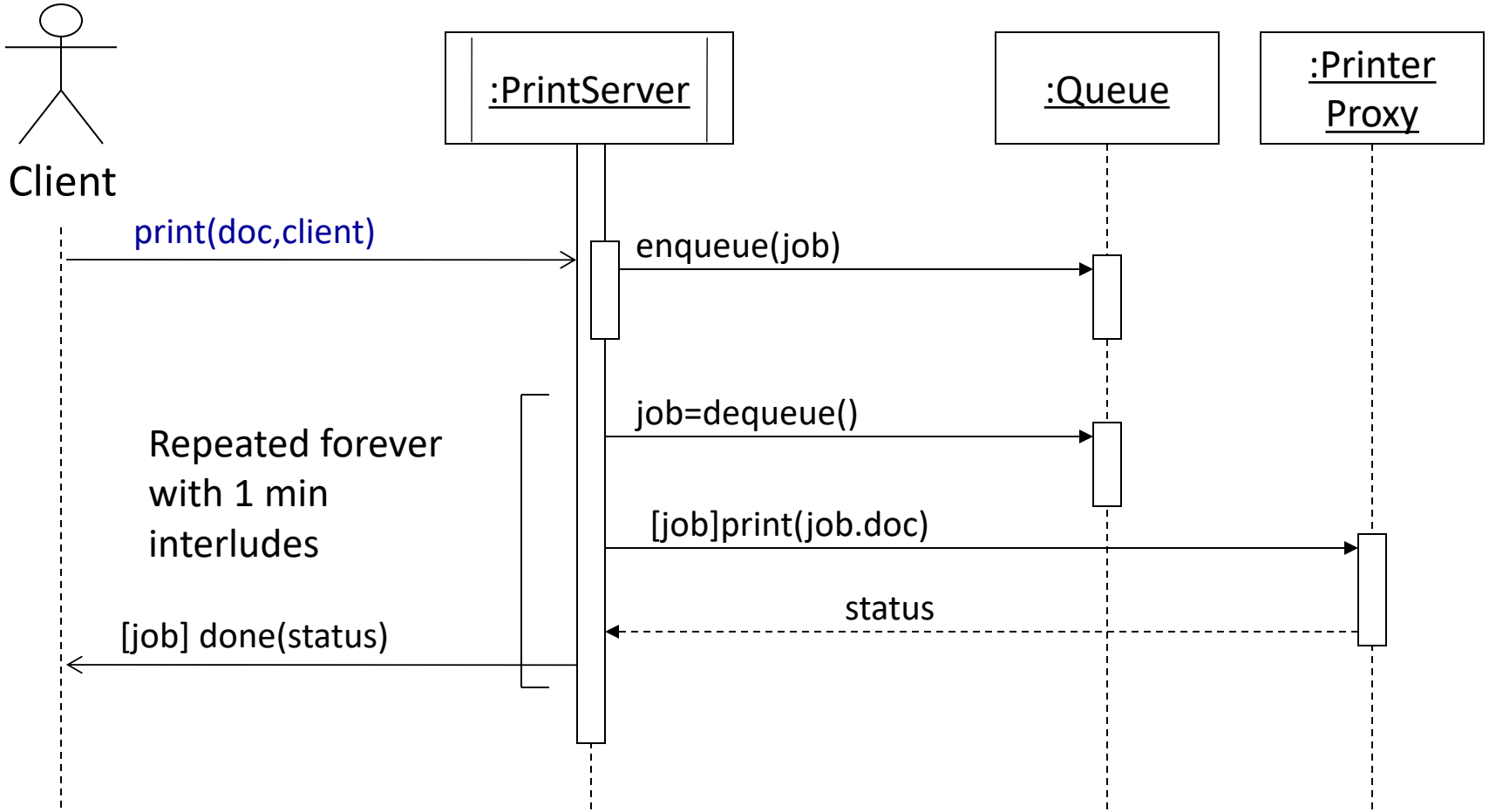
Sequence Diagram

- Iteration
  - Denoted by a * preceding the message name
- Condition
  - Denoted by Boolean expression in [ ] before the message name

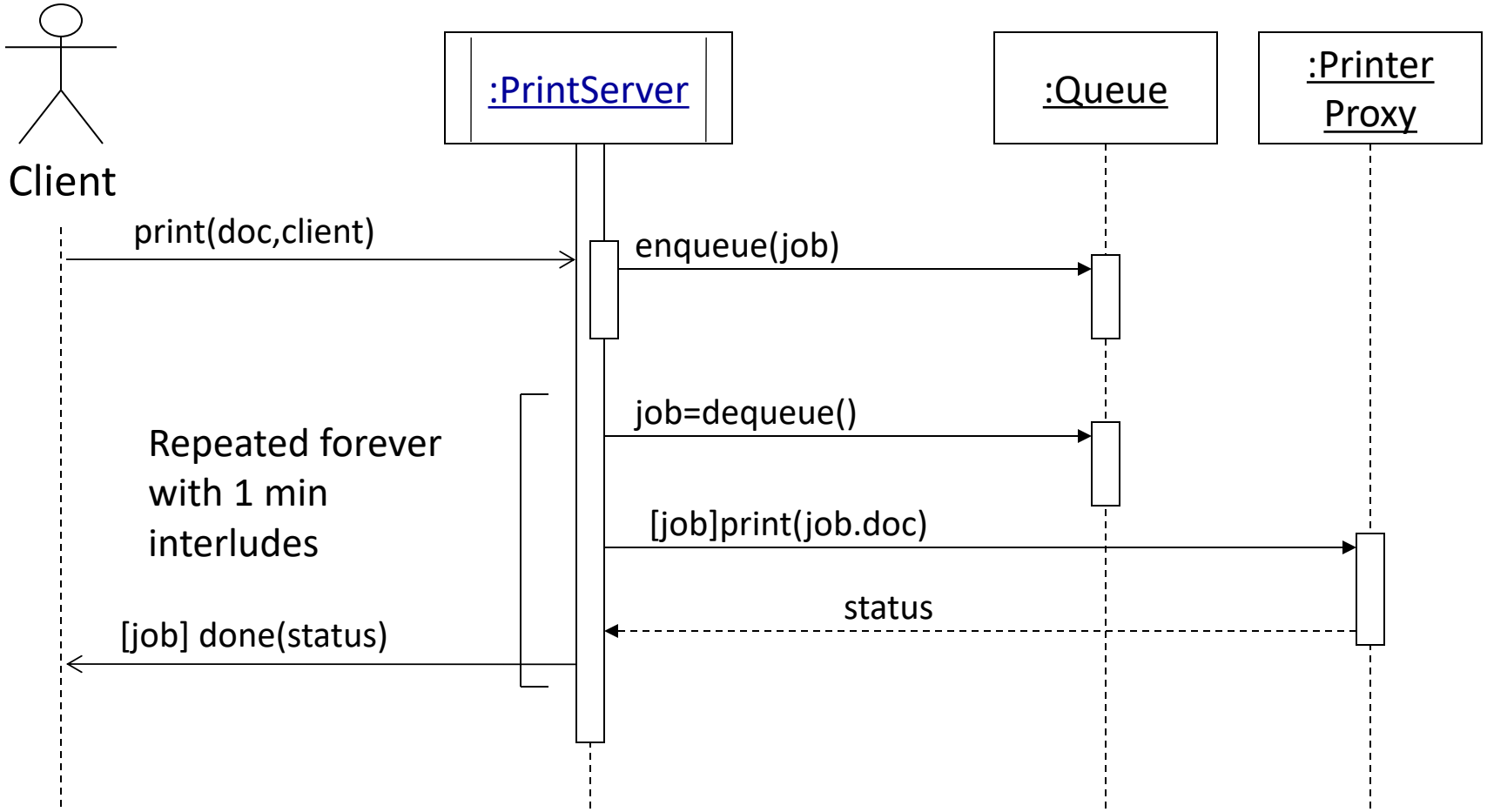
Sequence Diagram



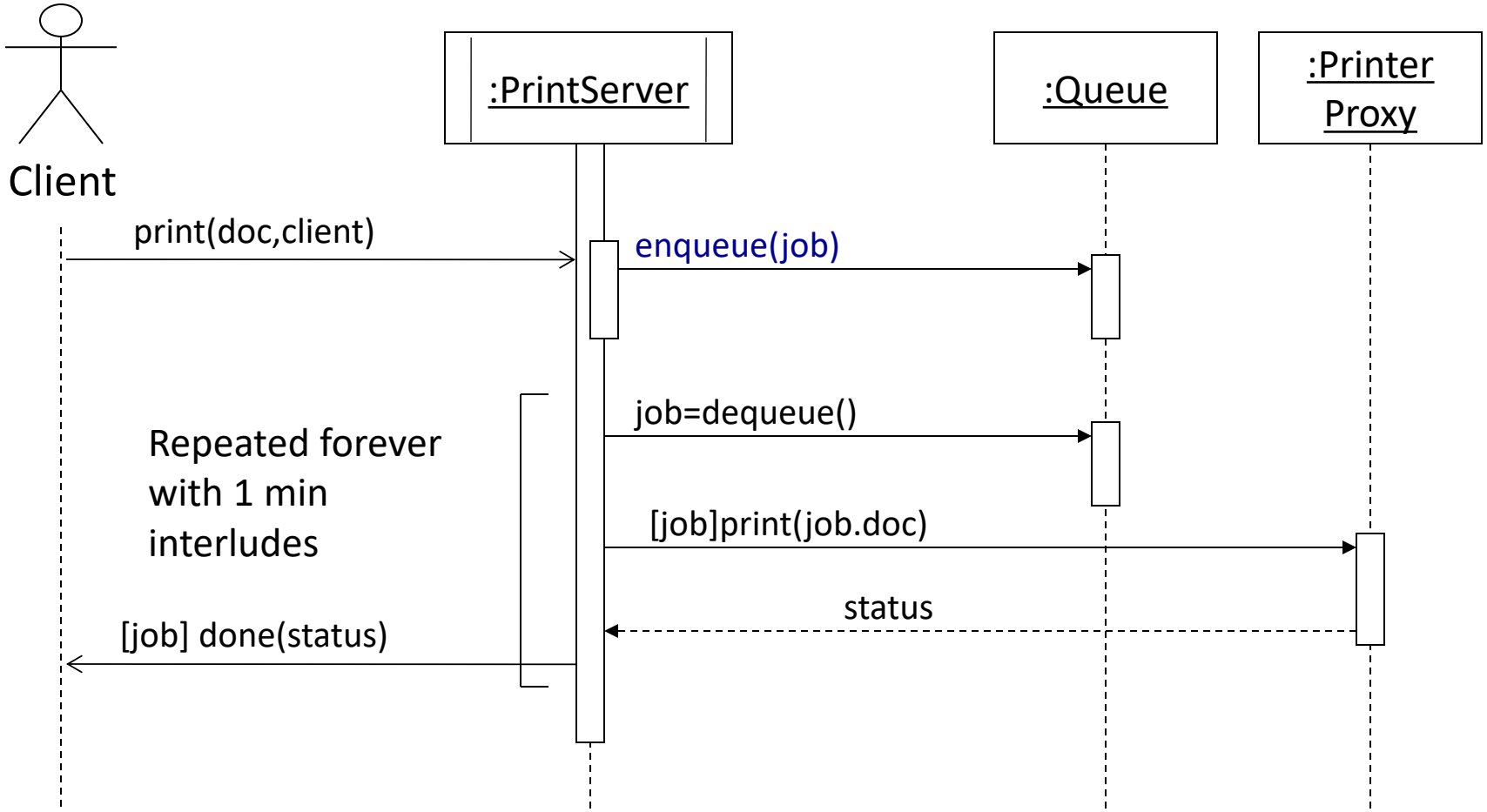
Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)

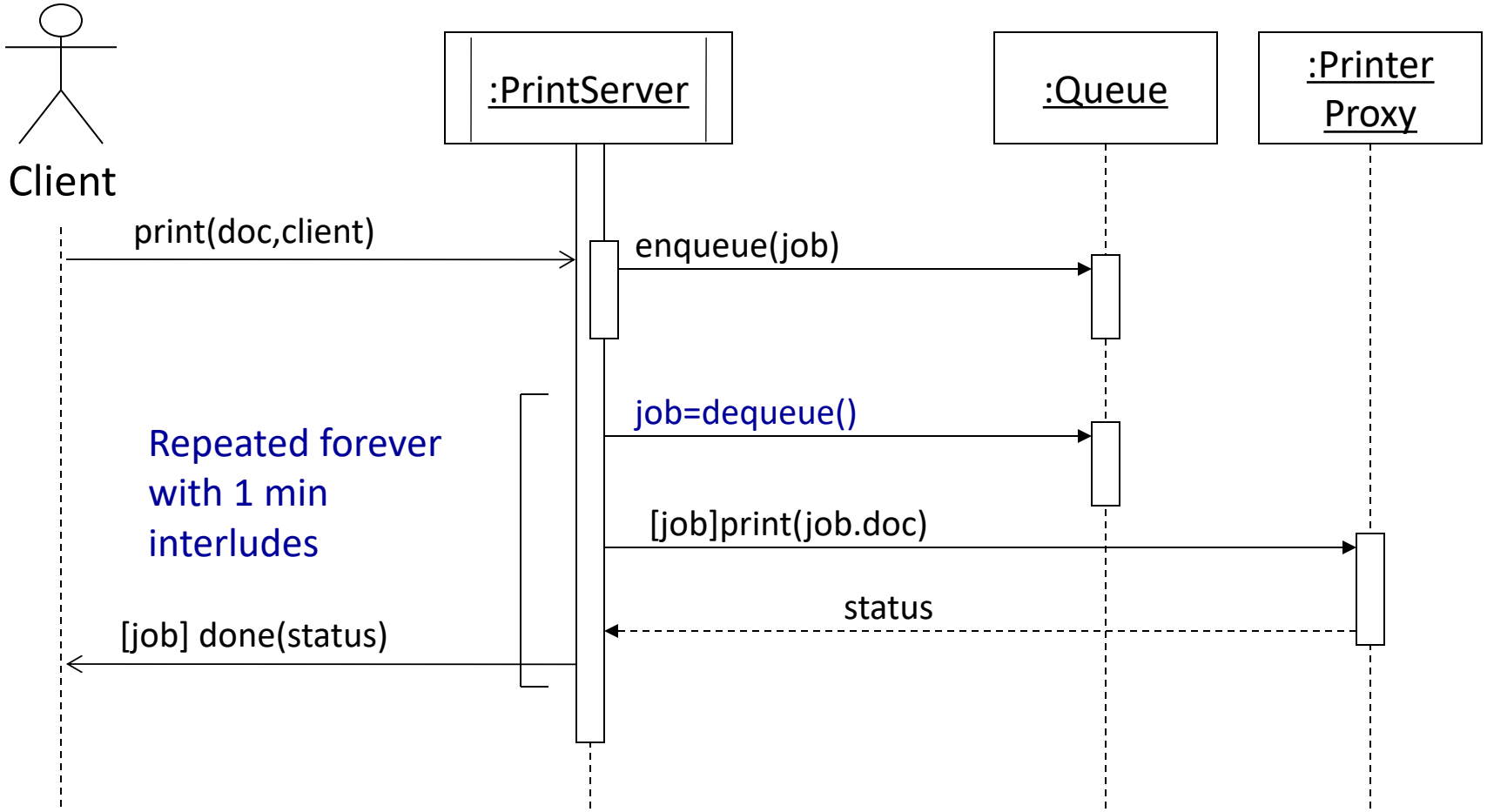
Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)

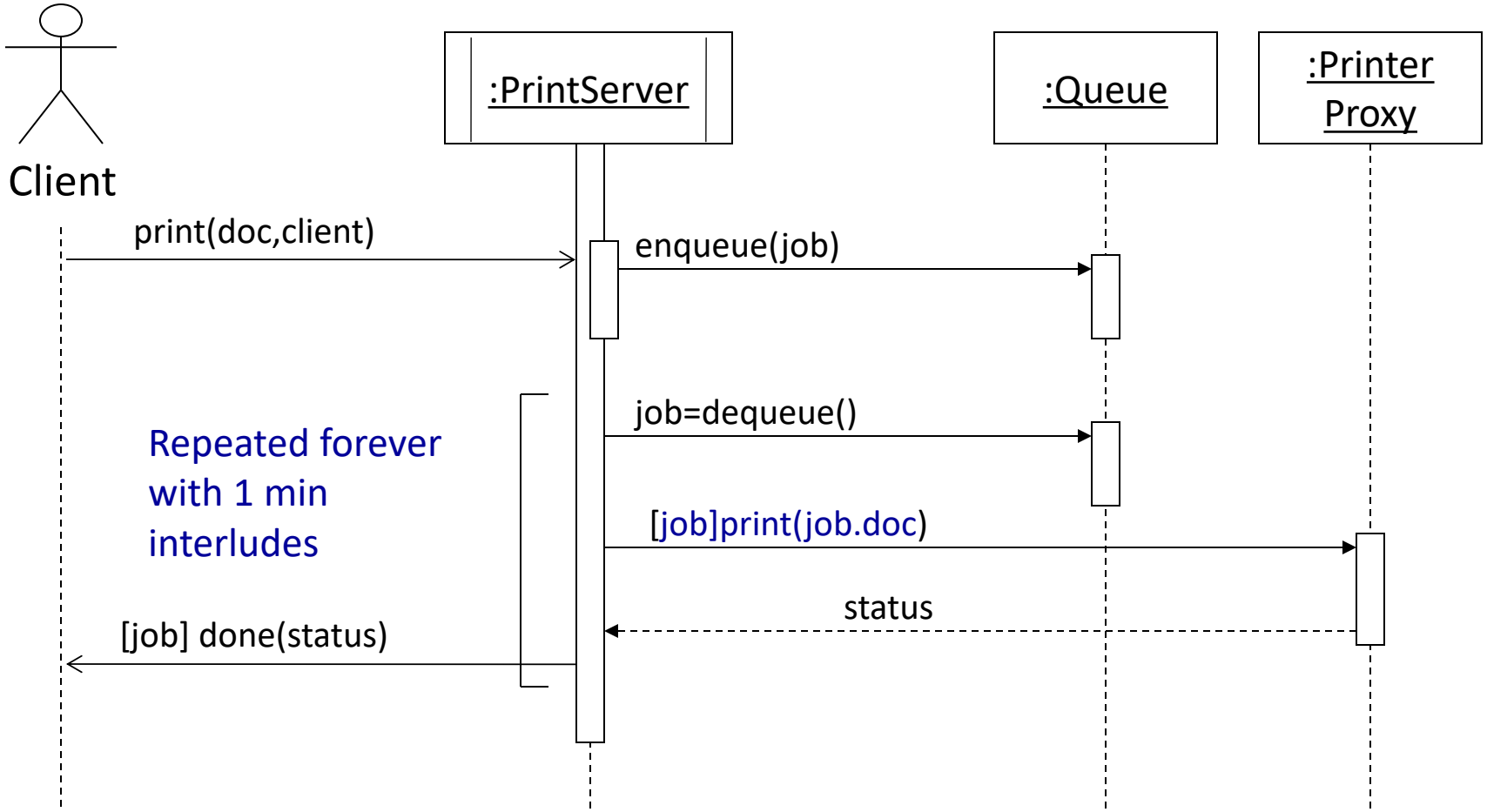
Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)

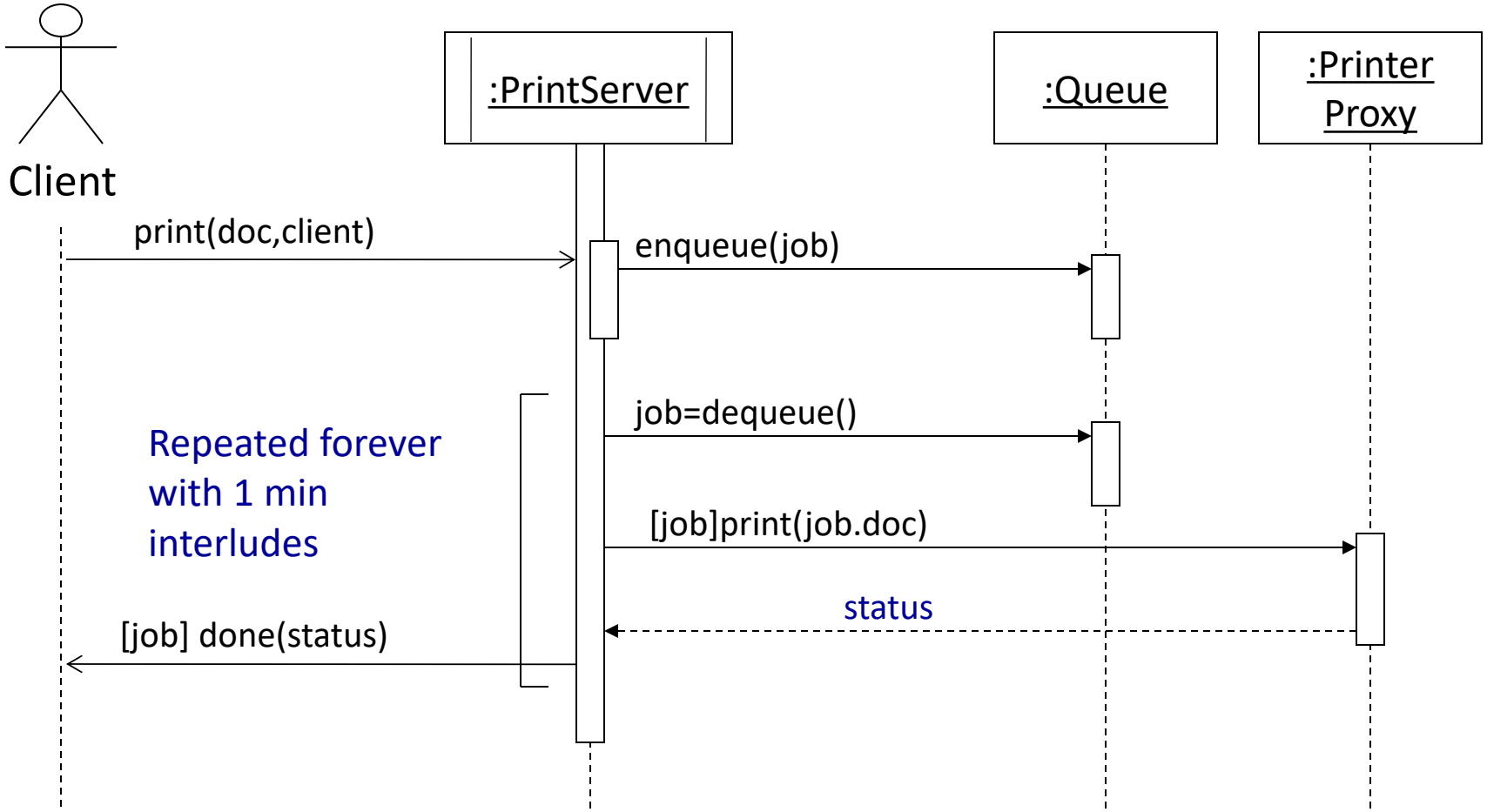
Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)

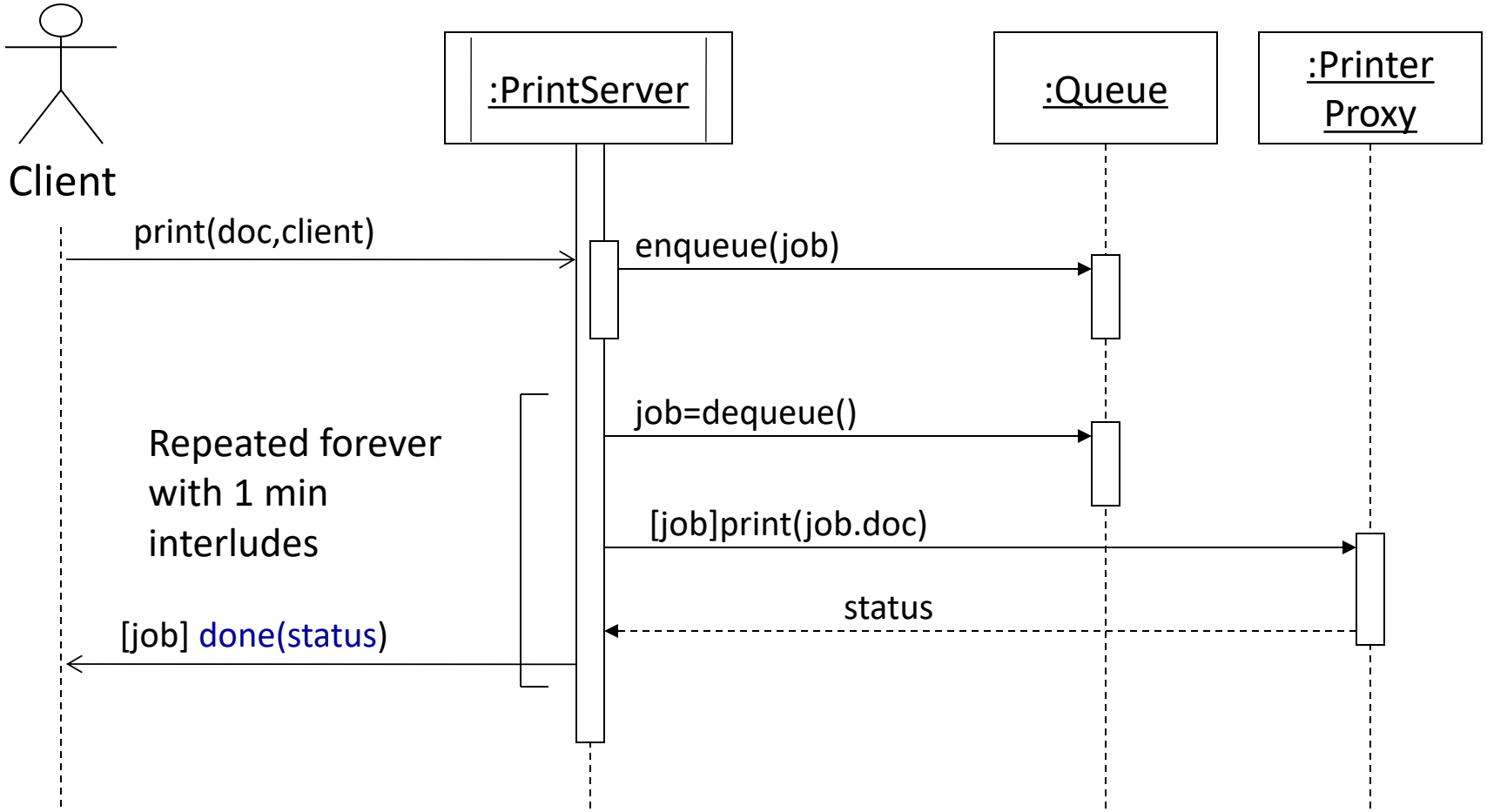
Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)

Examples

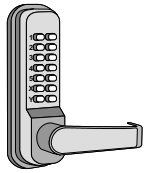

Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)

Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)

State Representations

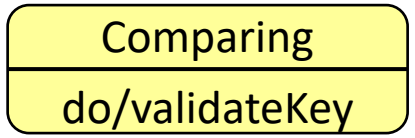
- Represent the state of a class as the system performs its function
  - A passive state: the current status of all attributes of an object
    - "timestamp" of "Key" is updated by the "key entered" event



  - An active state: the current status of the object as it undergoes an ongoing transformation or processing
    - "devStatus" of "LightDevice" is "on" or "off"
    - States of "Player" is "moving", "at rest", "injured", etc.

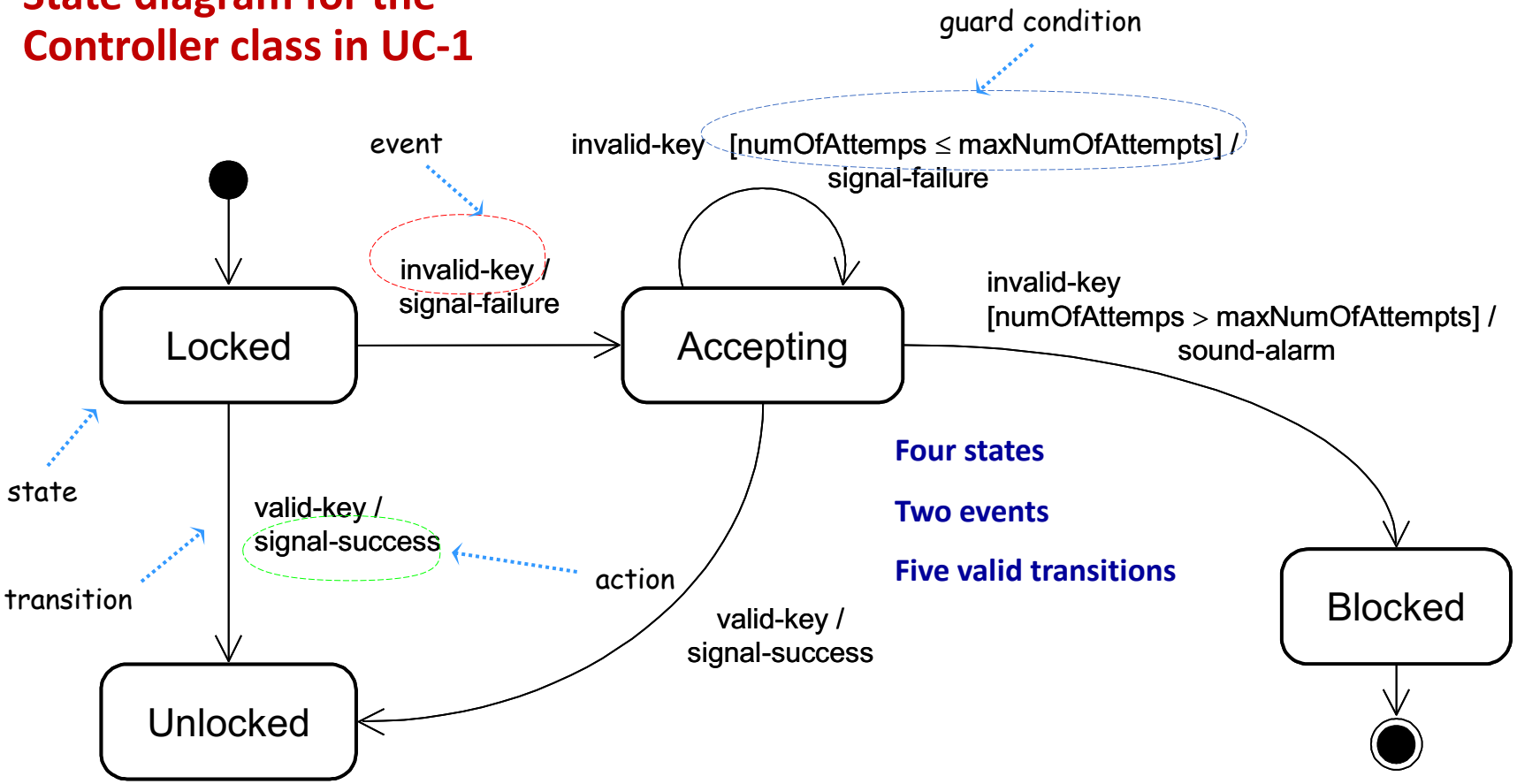
State Diagrams

- **State Diagram** indicates *how an individual class changes state based on external events*.
- It models an object's states, actions performed on the states, and transitions between the states.
  - State transition denoted by arrow, labeled by the event;
  - A "guard" is specified as a Boolean condition that should be satisfied to cause a transition;
  - An "action" occurs concurrently with state transition (consequence)
  - A special "do activity" state (denoted as "*do/<activity>*")



A State Diagram Example

State diagram for the Controller class in UC-1



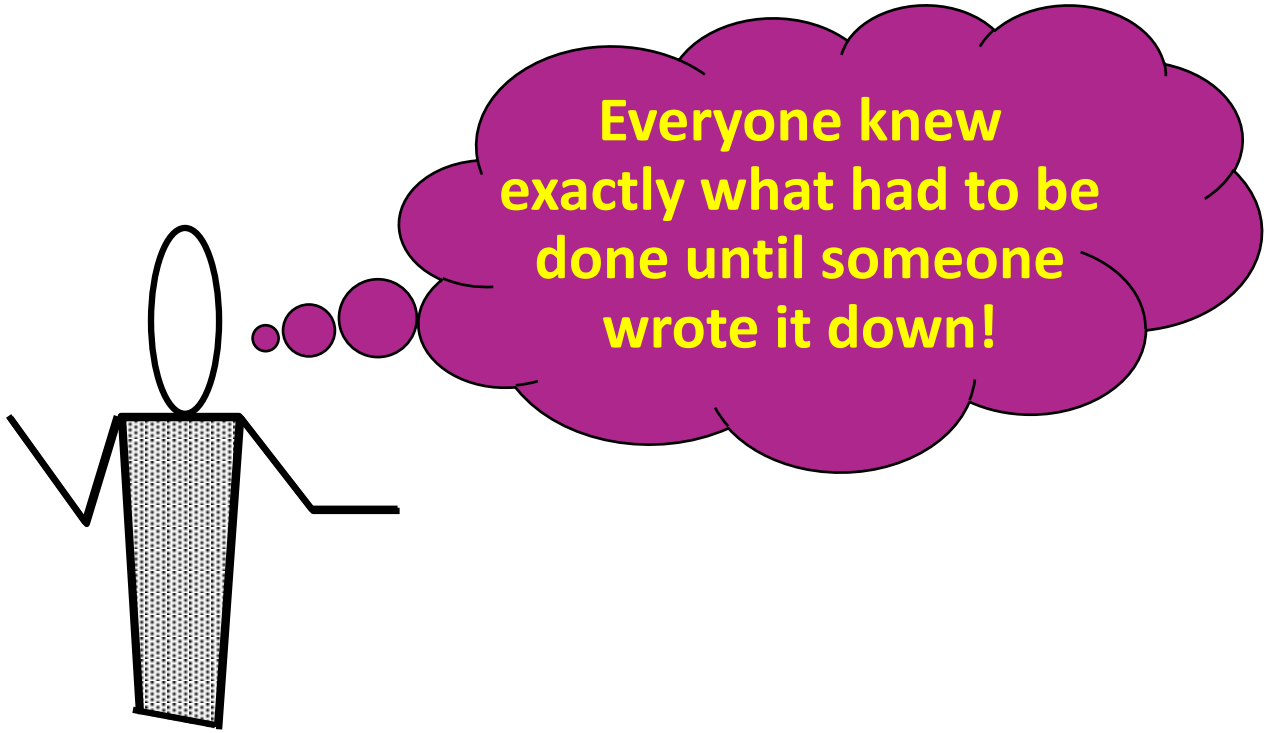
Wrap-Up

- **How does the system behave?**
  - Make a list of the different states of a system
- **How does the system change state?**
  - Indicate how the system makes a transition from one state to another
    - Indicate event
    - Indicate action
- Draw a state diagram or a sequence diagram.

Wrap-Up

- To create the model
  - Evaluate all use-cases to fully *understand the sequence of interaction* within the system
  - *Identify events* that drive the interaction sequence and understand how these events relate to specific objects
  - *Create a sequence* for each use-case
  - Build a *state diagram* for the system
  - Review the behavioral model to verify accuracy and consistency

Keep in Mind! In Software Specification …



References

- Prof. Fengjun Li's EECS 448 Fall 2015 slides

- This slide set has been extracted and updated from the slides designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill 2014) by Roger Pressman