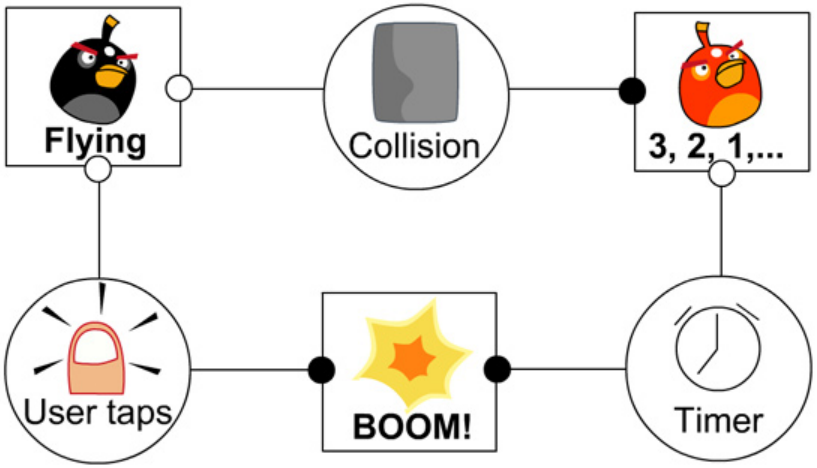
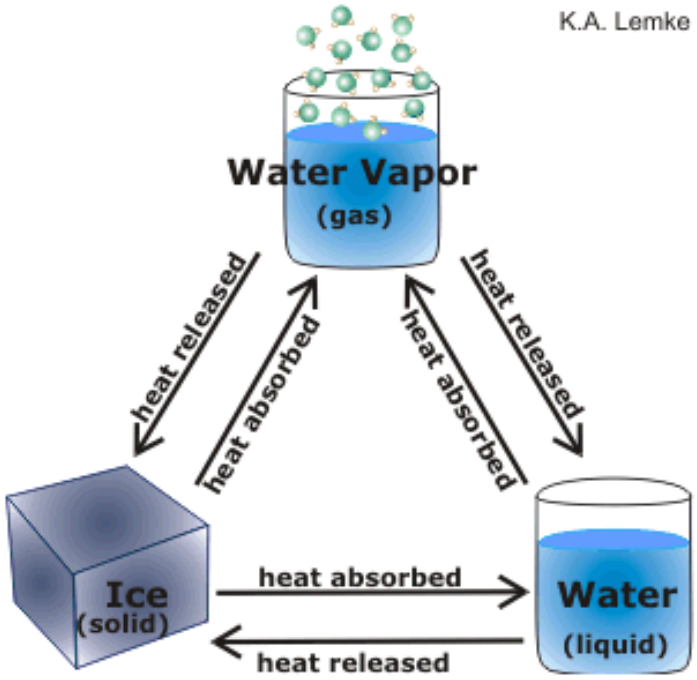
State Machine Diagrams

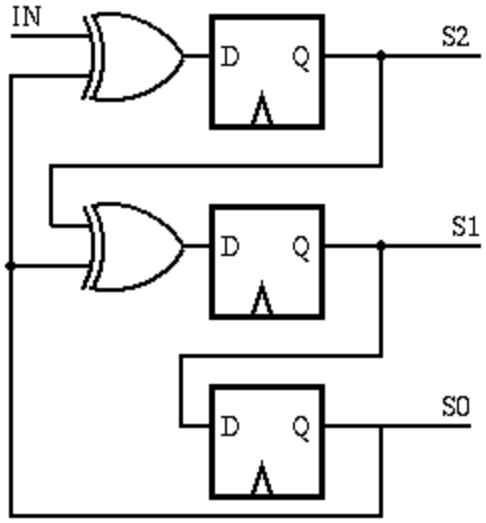
CSCI-4448 - Boese

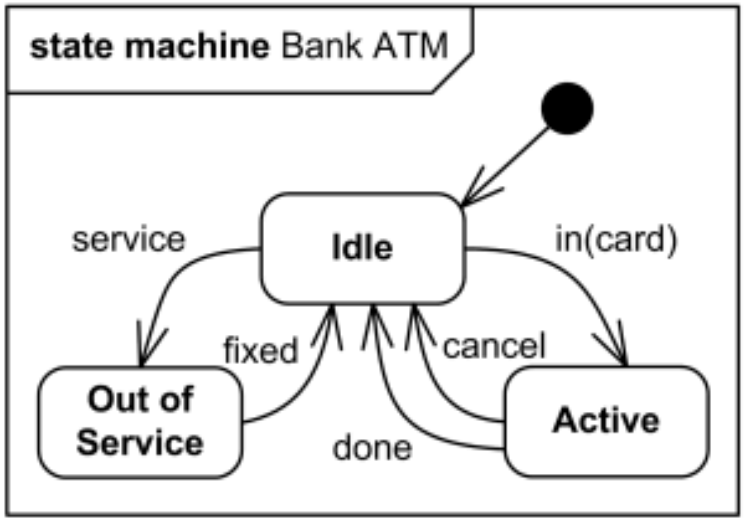
University of Colorado **Boulder**

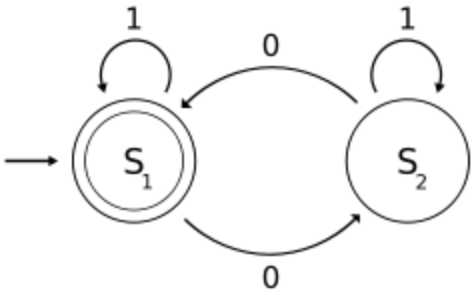
Overview





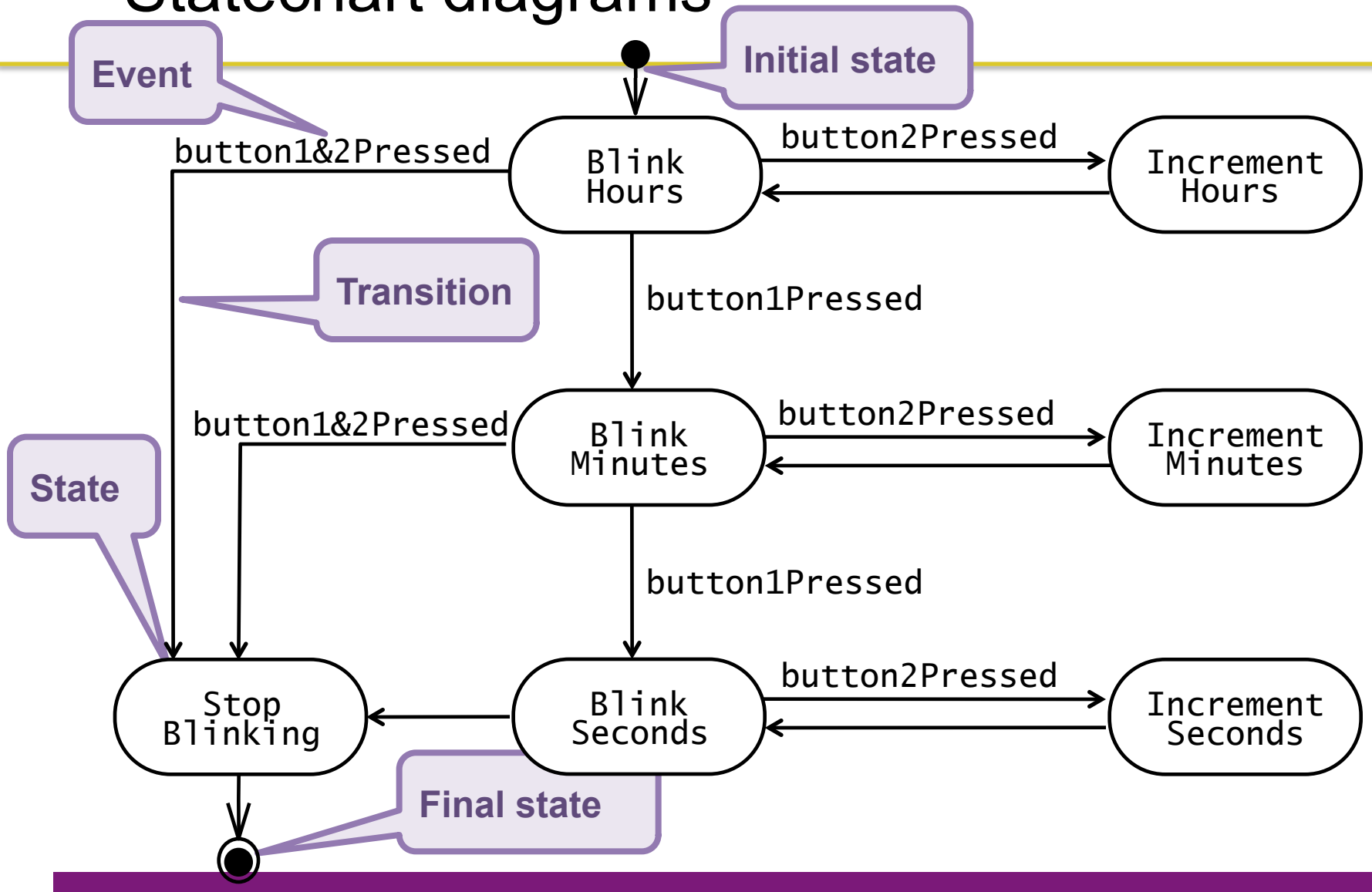






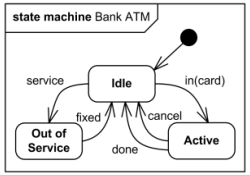
University of Colorado Boulder

Statechart diagrams



Represent behavior of *a single object* with interesting dynamic behavior.

State Machine Diagrams

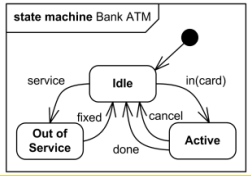


State Machine Diagrams

- Models the behavior and lifecycle of an instance of a class, across many (if not all) use cases
- **Behavioral State Machine**
  - Describe discrete behavior of an object, and how objects shift from one type of behavior to another
  - *Example: A bot in a game may be attacking, fleeing, searching, etc.*
- **Protocol State Machine**
  - Describe the different phases of some protocol, and how to transfer between phases of that protocol
  - *Example: SFTP needs to connect, validate passwords, transfer files, etc.*

Diagram Comparison

State Machine Diagram



- Good way to model objects which can be described or implemented using a state-transition paradigm
  - Objects are always in some **state**
  - Objects are subject to some **event / input**
  - Based on input and current state, objects may **transition** to another state

State Machine vs. Activity Diagram

State Machine

- Models behavior from the perspective of an object
  - *React* to various events, regardless of the source
- Component-level view of the system

Activity Diagram

- Describes interactions among the system and several actors
  - May include several objects / subsystems
- High-level / business logic view of the system

State Machine vs. Sequence Diagram

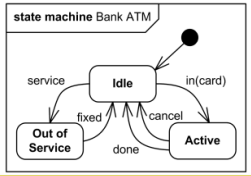
State Machine

- Demonstrates how individual objects make some decision
- Provides information on state of object
- Object may be involved in *multiple* sequence diagrams
  - Ensure non-conflicting behavior

Sequence Diagram

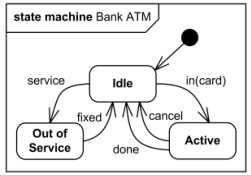
- Demonstrates how individual objects interact to achieve a goal
  - Focus on messages and communication
- Provides no information on object states
- Provides information on control
  - Loops, decisions, etc.

State Machines – When to Use



- Does a particular object or process have well-defined, high-level states?
  - Registration involves
    - *logging in*
    - *selecting classes*
    - *confirming registration*
    - *logging out*

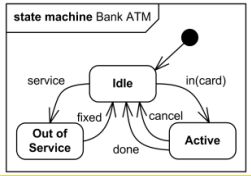
Why Use State Machines?



- **Requirements**
  - Capture requirements from client by stepping through each use through the diagram
  - Resolve ambiguities in use case *before* coding
- **Validation and Verification**
  - Objects in system performed what client wanted
  - Able to find *where* faults and failures occur
    - "In this state, we expected this to happen when this message was received. Instead, this other thing happened."
  - Ensure no missing state / transition pairs
    - Know how object will respond to *all* events at *all* times
- **Documentation**
  - Easier to communicate to new developer what a particular class is for

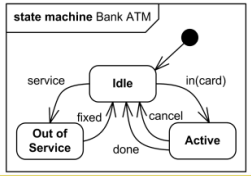
How

State Machine Diagrams



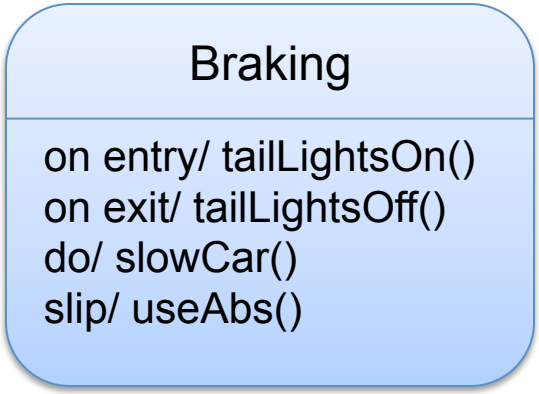
- **States**: *Rectangles*
  - Can just indicate a particular state

State Machine Diagrams

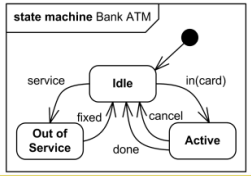


- **States**: *Rectangles*
  - Can just indicate a particular state

  - Can also provide state actions
    - *on entry / on exit* – when transitioning into or out of the state
    - *do /* - some behavior which is done while in the state
    - May also be used to provide an action associated with some internal transition

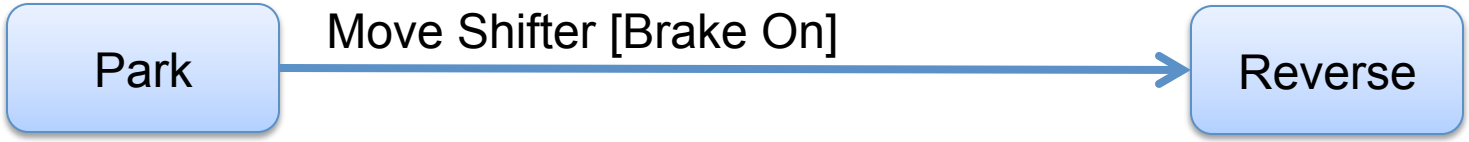


State Machine Diagrams

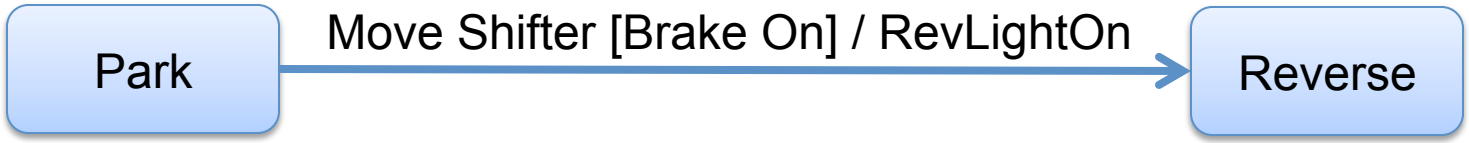


- **Transitions**: Arrows
  - Shows changes between state
    - Transitions are annotated with *triggers*

    - Transitions may have *guards* associated with them

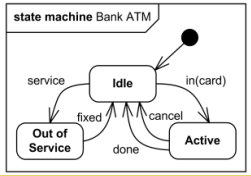


    - Transitions may cause some *effect* or *action*

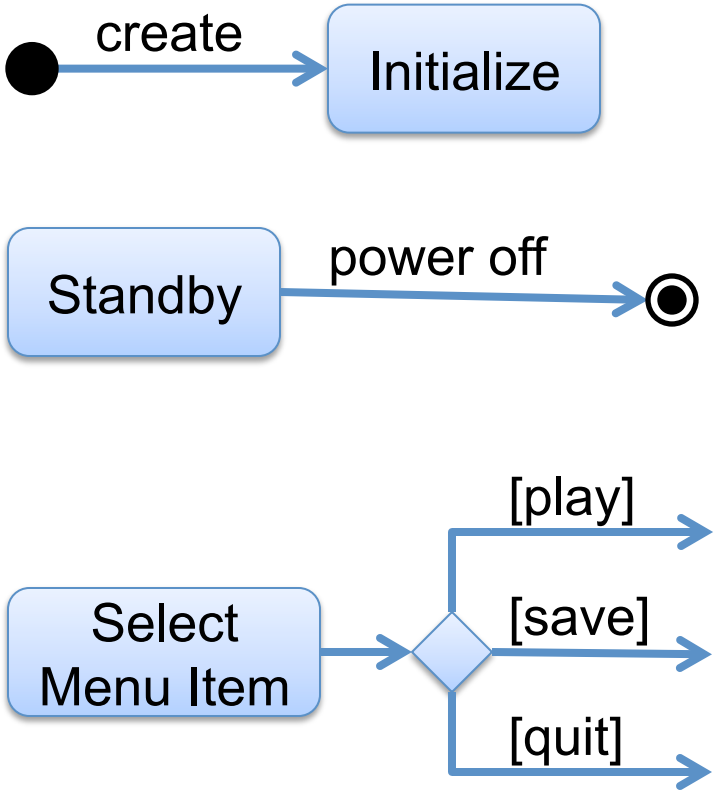


Trigger [Guard] / Effect

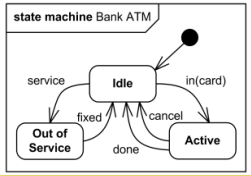
State Machine Diagrams



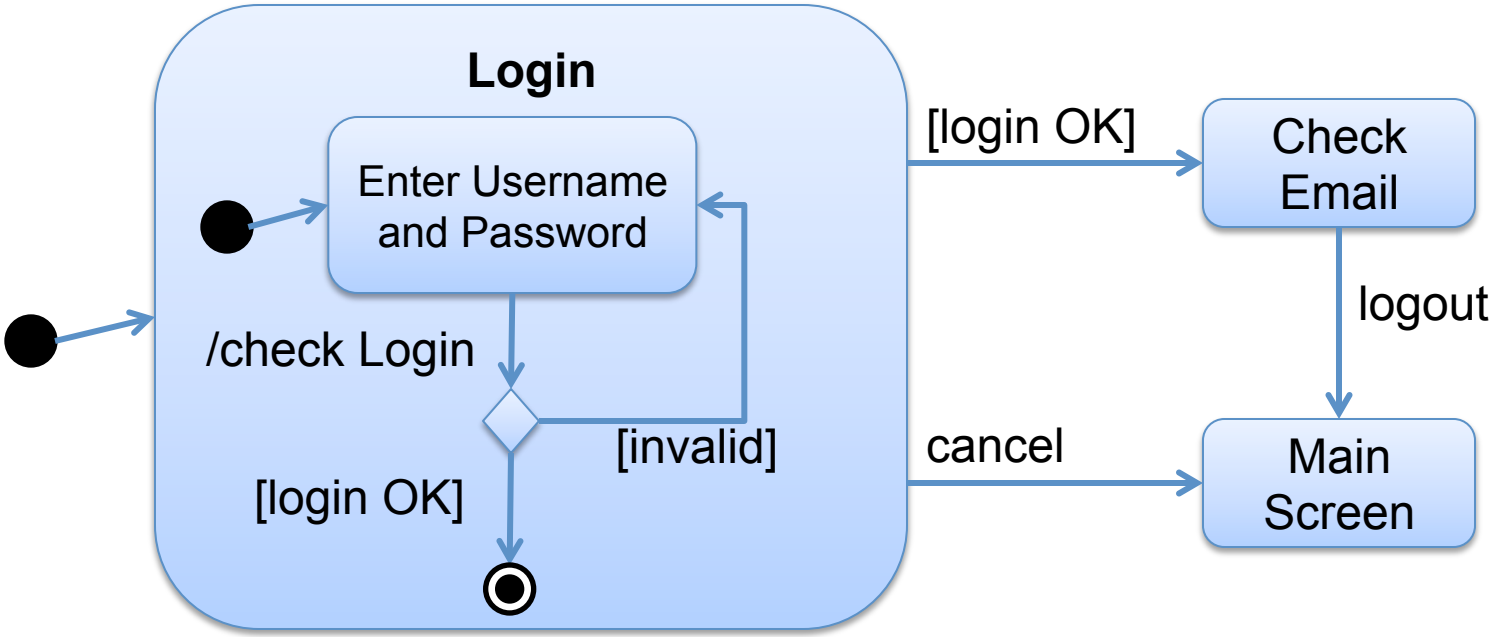
- **Pseudo-States**
  - **Initial State**
    - *Filled circle*
  - **Final States**
    - *Dotted circle*
  - **Choice pseudo-state**
    - *Diamonds*



State Machine Diagrams

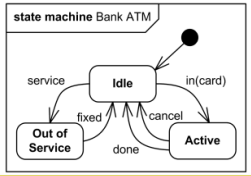


- **Compound States**
  - A state contains some sub-state machine
  - Sub-state machine can be treated independently
  - Host state may transition based on current sub-state



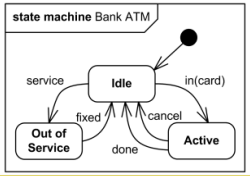
Designing State Machines

Creating State Machines



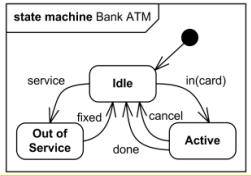
- Identify important objects to analyze in the system
  - Not everything needs to be or is well modeled as a state machine
  - Button Widget: Two states – **clicked** or **not clicked**
    - *Very simple – probably better to consider this an event*
  - TCP Client: TCP has several phases to its protocol, and well defined messages within the protocol
    - ***States**: SYN_SENT, SYN_RCVD, ESTABLISHED, CLOSED*
    - ***Events**: Receive SYN, Receive ACK, Receive SYN-ACK*
    - *Very complicated behavior to model! (Three- and four-way handshakes)*

Creating State Machines



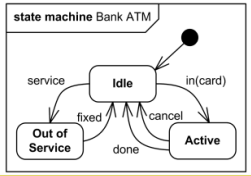
- Generate or look at a description of the object being implemented
  - In a platformer game: "The heroine of the game is controlled by the player, who can make the heroine jump by pressing A, move left or right, run by holding down A, and duck by pressing down. If the heroine stands too long without input from the user, she get's bored and start's playing with a yo-yo."
  - What are possible states?
    - *What situations cause the heroine to behave differently?*
  - What are possible events?
    - *What can cause the heroine to change her behavior?*

Creating State Machines



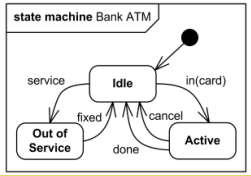
- Generate or look at a description of the object being implemented
  - In a platformer game: "The heroine of the game is controlled by the player, who can make the heroine jump by pressing A, move left or right, run by holding down A, and duck by pressing down. If the heroine stands too long without input from the user, she get's bored and start's playing with a yo-yo."
  - What are possible states?
    - *Jumping, Moving, Running, Ducking, Standing, Bored*
  - What are possible events?

Creating State Machines



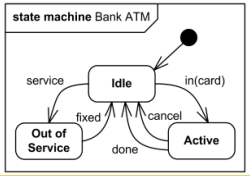
- Generate or look at a description of the object being implemented
  - In a platformer game: "The heroine of the game is controlled by the player, who can make the heroine jump by pressing A, move left or right, run by holding down A, and duck by pressing down. If the heroine stands too long without input from the user, she get's bored and start's playing with a yo-yo."
  - What are possible states?
    - *Jumping, Moving, Running, Ducking, Standing, Bored*
  - What are possible events?
    - *Press Left or Right, Release Left or Right, Press Down, Press A, Release A, Idle Too Long*

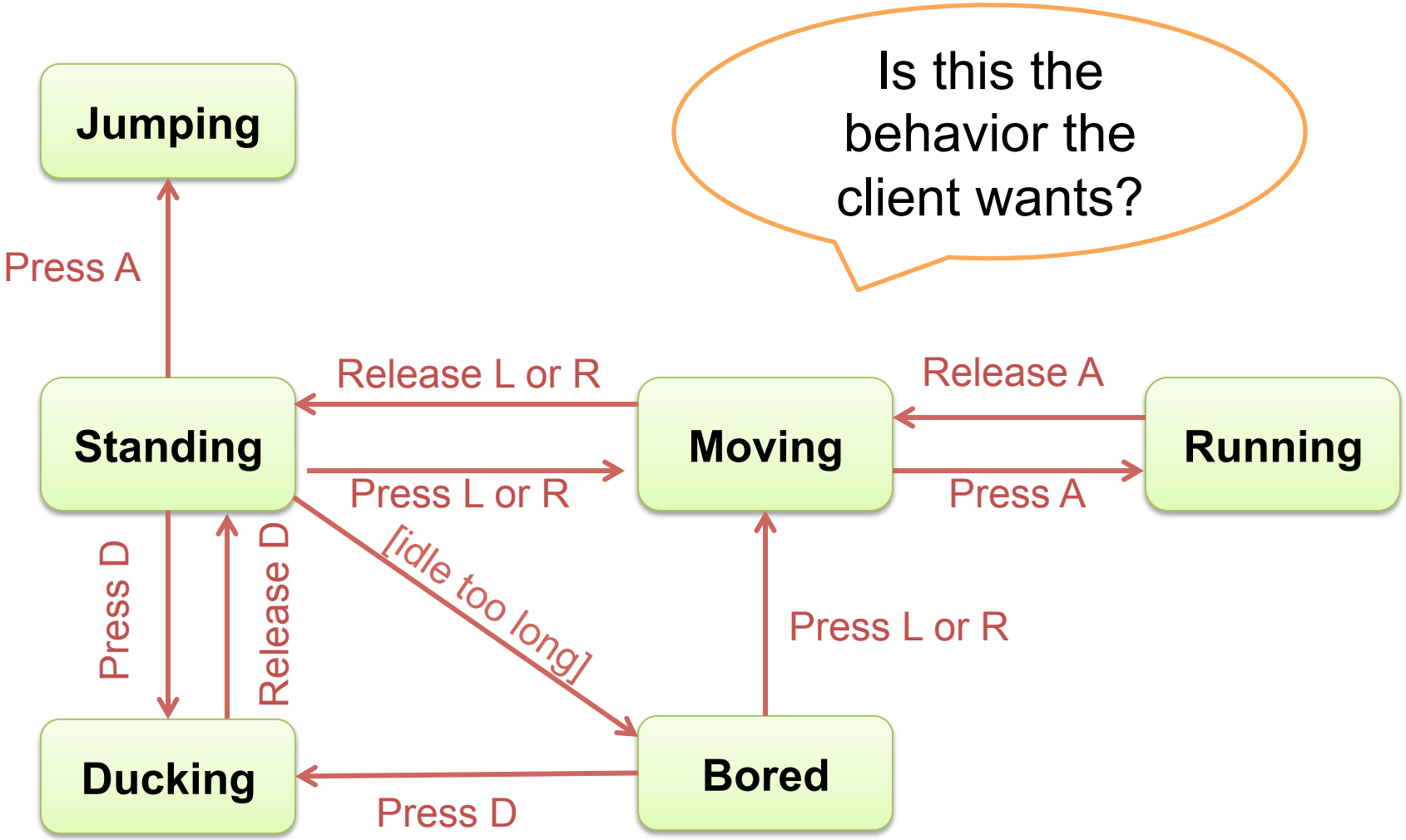
Creating State Machines



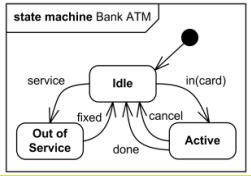
- Determine the transitions
  - Which state / event pairs cause the object's behavior to change, and to what?
  - **Press A, Standing → Enter Jumping State**
  - **Press Down, Standing → Enter Ducking State**
  - **Press A, Moving → Enter Running State**
  - **Release A, Moving → Enter Moving State**
  - **etc.**
- Use the state / event pairs to draw the transitions

Creating State Machines





Creating State Machines



Wait, how does the heroine land after jumping?

What if the heroine is running, and the player releases L / R?

Jumping

Press A

Standing

Release L or R

Moving

Release A

Running

Press L or R

Press A

Press D

Release D

[idle too long]

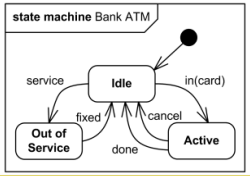
Press L or R

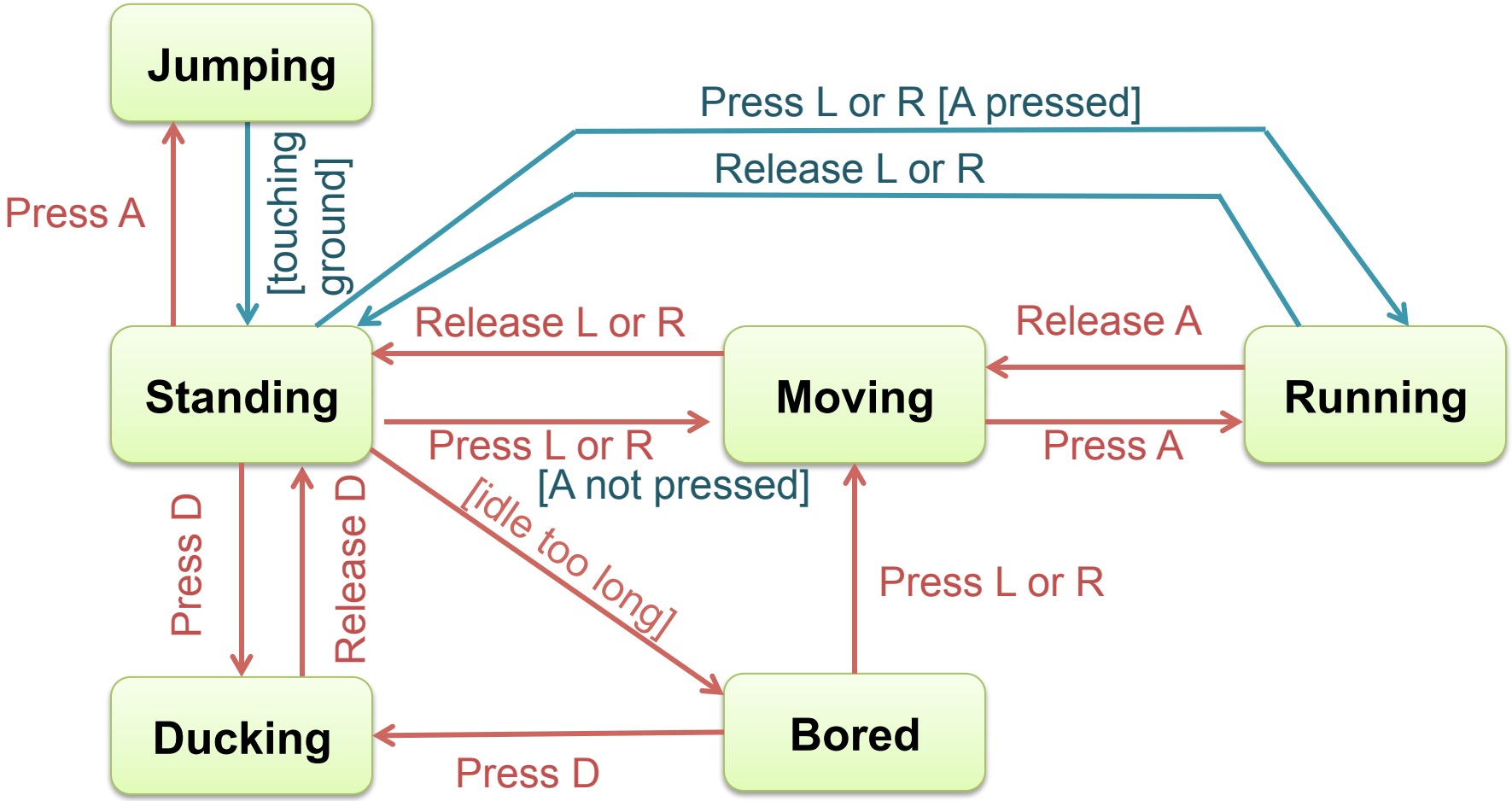
Ducking

Press D

Bored

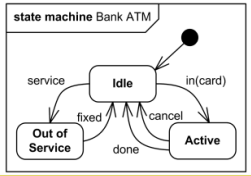
Creating State Machines





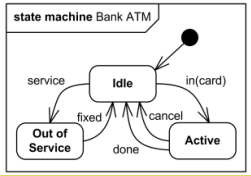
More Examples

TCP Client (RFC793, pg. 21)



- TCP connections undergo many state changes
  - *LISTEN – Waiting for a connection request*
  - *SYN-SENT – Waiting for response from a connection request*
  - *SYN-RECEIVED – Acknowledged connection request*
  - *ESTABLISHED – Open connection*
  - *FIN-WAIT-1 – Waiting for / acknowledge termination request*
  - *FIN-WAIT-2 – Waiting for termination request*
  - *CLOSE-WAIT – Wait for connection termination from user*
  - *CLOSED – No connection exists*
  - *LAST-ACK – Wait for acknowledgement of termination*
  - *TIME-WAIT – Wait for time to pass*
  - *CLOSING – Waiting for termination request acknowledge*

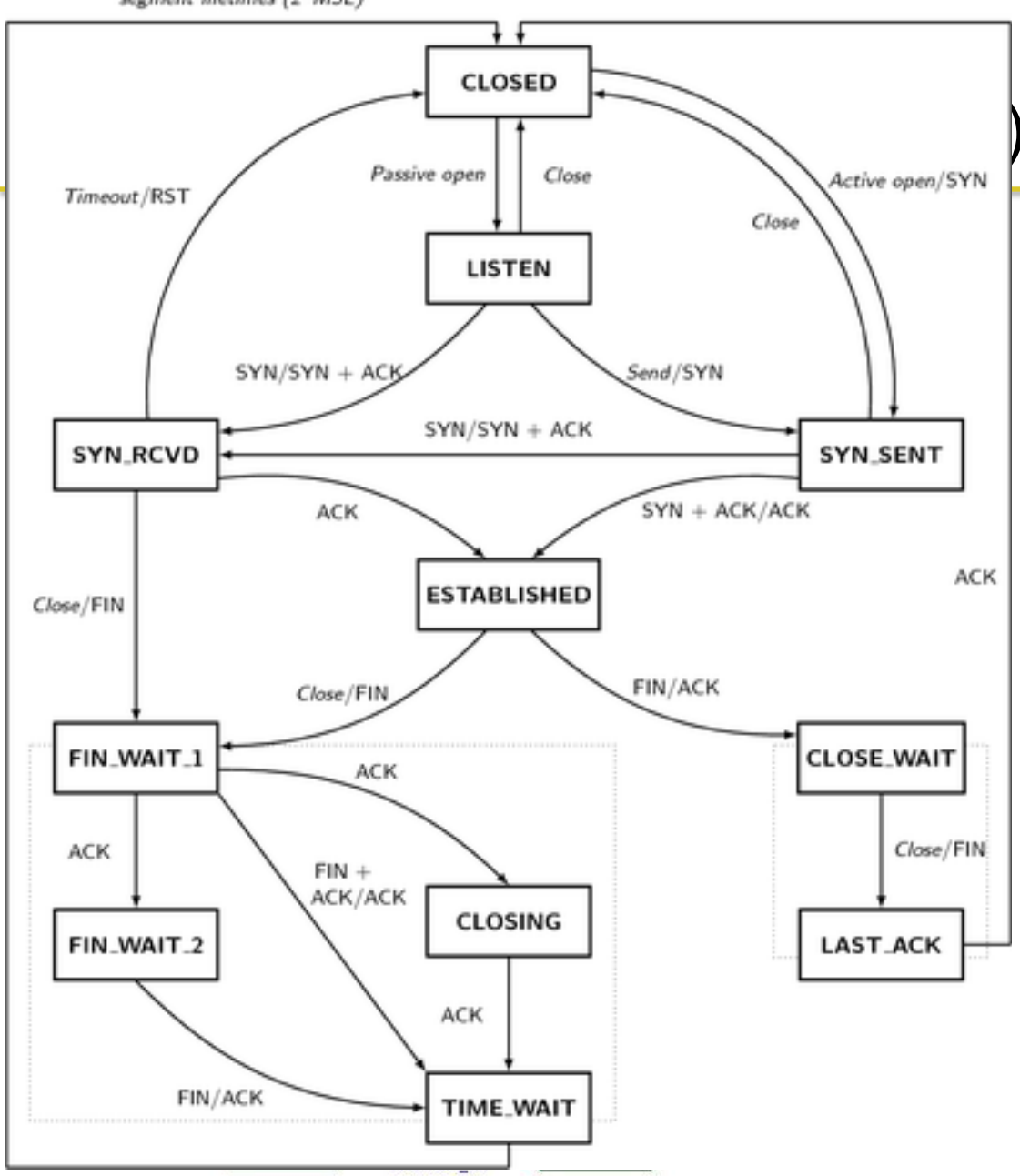
TCP Client (RFC793, pg. 22)

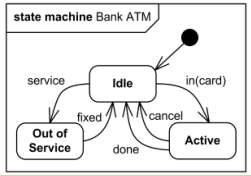


- TCP connections exchange several messages between client and server
  - *OPEN*
  - *SEND*
  - *RECEIVE*
  - *CLOSE*
  - *ABORT*
  - *STATUS*

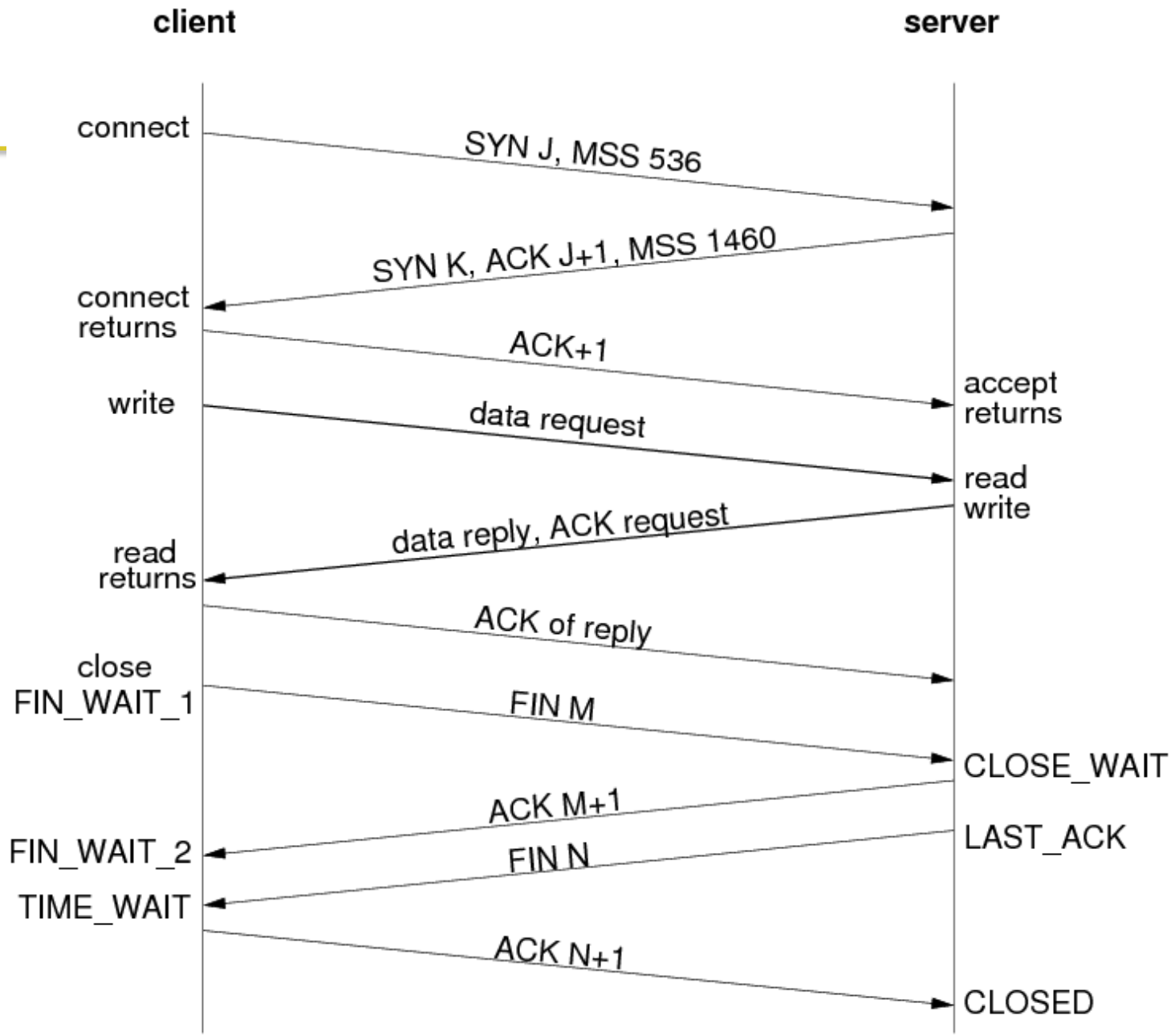
  Events from User
  - *SYN*
  - *ACK*
  - *RST*
  - *FIN*

  TCP Events

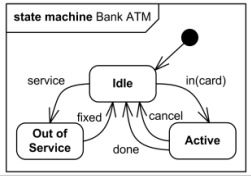




- States correspond to RFC793
- Transitions have a guard
  - Which message received?
- Transitions have an action
  - What message to send?

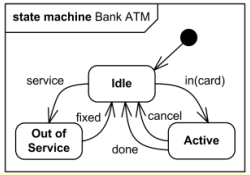

client
server
connect
SYN J, MSS 536
SYN K, ACK J+1, MSS 1460
connect returns
ACK+1
accept returns
write
data request
read write
data reply, ACK request
read returns
ACK of reply
close FIN_WAIT_1
FIN M
CLOSE_WAIT
ACK M+1
LAST_ACK
FIN_WAIT_2
FIN N
TIME_WAIT
ACK N+1
CLOSED

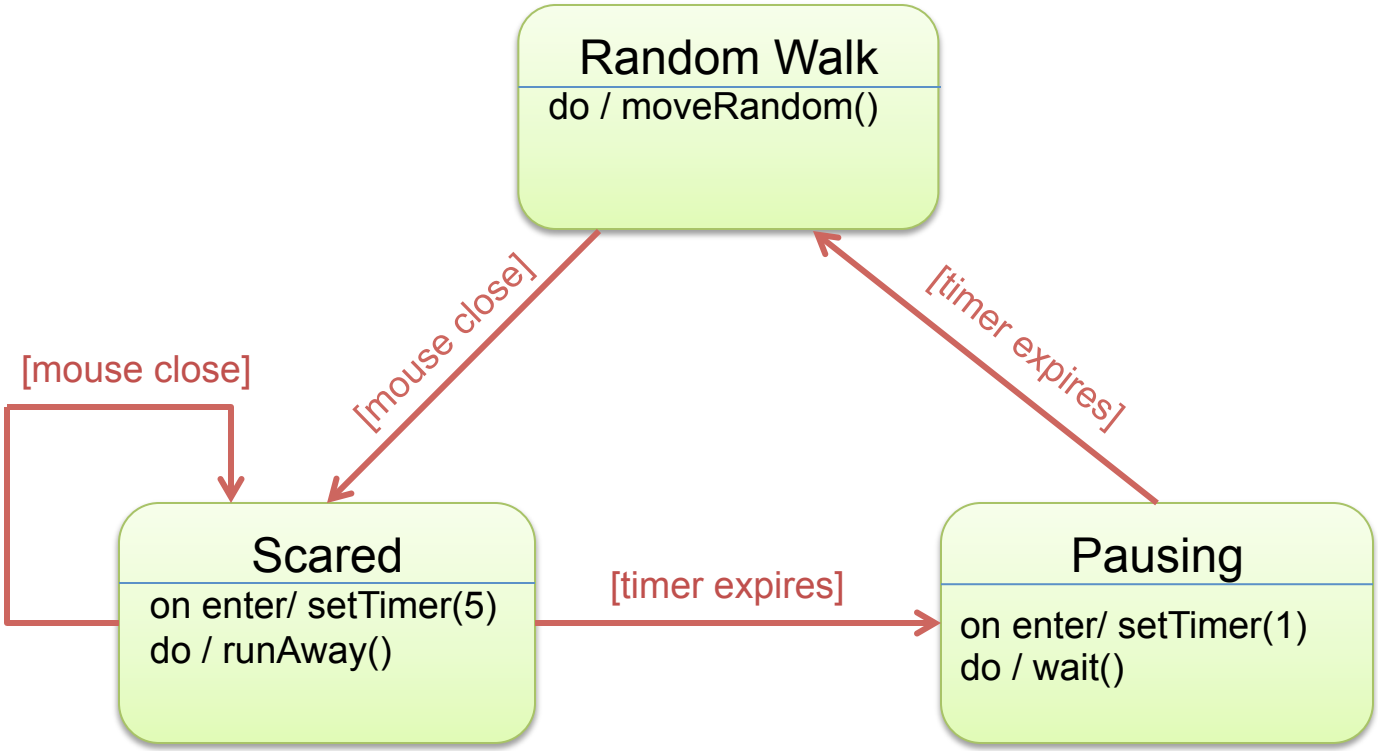
AI Behavior



- Consider a small toy AI, which responds to the user's mouse position:
  - When it's not near the mouse, it walks around at random
  - When it's near the mouse, it gets scared and runs away
  - It remains scared until it's been away from the mouse for at least 5 seconds
  - After 5 seconds, it pauses for a second, and then returns to walking randomly

AI Behavior


state machine Bank ATM
service
Idle
in(card)
fixed
cancel
Out of Service
done
Active


Random Walk
do / moveRandom()
[mouse close]
[mouse close]
[timer expires]
Scared
on enter/ setTimer(5)
do / runAway()
[timer expires]
Pausing
on enter/ setTimer(1)
do / wait()