# Working with State Diagrams

- Use Cases and Scenarios provides a way to describe system behaviour.
- ***Use Case*** – Typical interaction between a user an a computer system.
- ***Scenario*** – Instance of a Use Case
- ***Interaction Diagrams*** – Capture Scenarios.

  Shows object interactions arranged in time sequence.
- Some times it is necessary to look at the behaviour inside an object.

UCSC

# Working with State Diagrams

- As the system interacts with users and other systems,
  - The objects that make up the system go through necessary changes to accommodate the interactions.
- If you are going to model systems, you must have a mechanism to model change.
- One way to characterize change is to say that its objects change their state in response to events and to time.

# Working with State Diagrams

Examples:

- When you throw a switch, a light changes its state from Off to On.
- After an appropriate amount of time, a washing machine changes its state from Washing to Rinsing.
- Hotel room changes its state to available, reserved and occupied.

•UML State diagram captures these kinds of changes.

# Working with State Diagrams

- UML *State Transition Diagrams* shows:
  - – Life history showing the different states of a given object.
  - – The events or messages that cause a transition from one state to another.
  - – The actions that results from a state change.
- *State Diagrams* are created only for classes with <u>significant</u> dynamic behaviour.

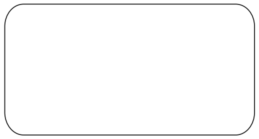
eg. ***Hotel Room*** in a Hotel Reservation System

# Modeling Dynamic Behaviour

- Interaction diagrams can be studied to determine the dynamic objects.
  - Objects receiving and sending many messages.
- If you have an attribute called *status*.
  - This can be a good indicator of various states.

# States

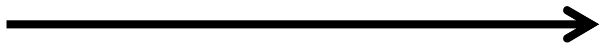
- eg. HotelRoom object can be in one of the following states.
  - Occupied, Available, Reserved
- eg. Course object (in a course registration system) can be in one of the following states.
  - Initialization,Open, Close,Cancel



***UML Notation for a State***

# State Transitions

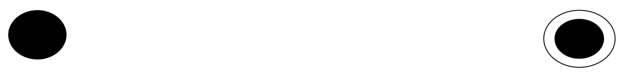
- A State Transition represents a change from an originating state to a successor state.
- An action can accompany a state transition.
- A State Transition is represented by an arrow that points from the originating state to the successor state.



*UML Notation for State Transition*

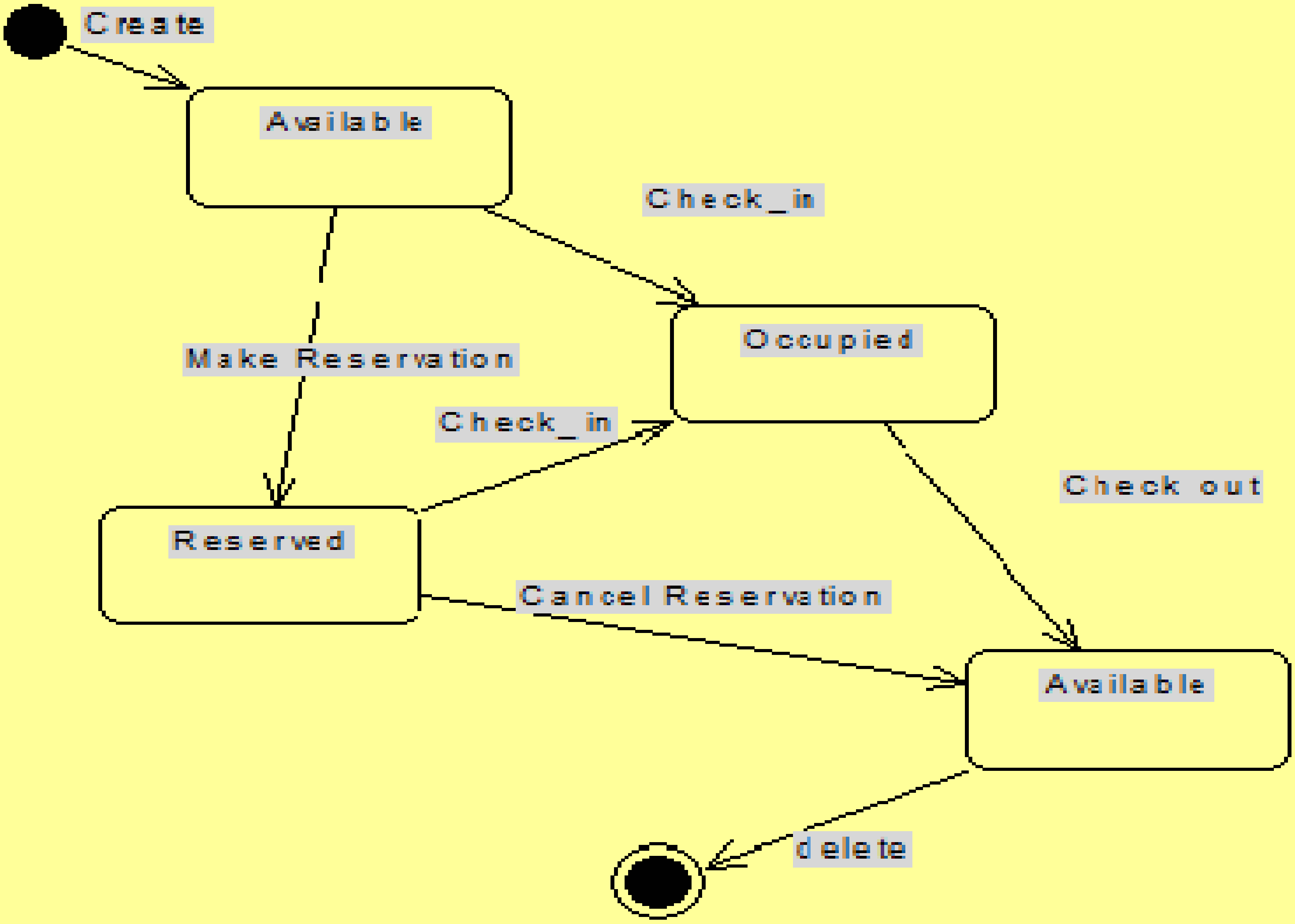
# Special States

- There are two special states that are added to the state transition diagram.
- **Start** state – Each diagram must have one and only one start state.
- **Stop** state – An object can have multiple stop states.

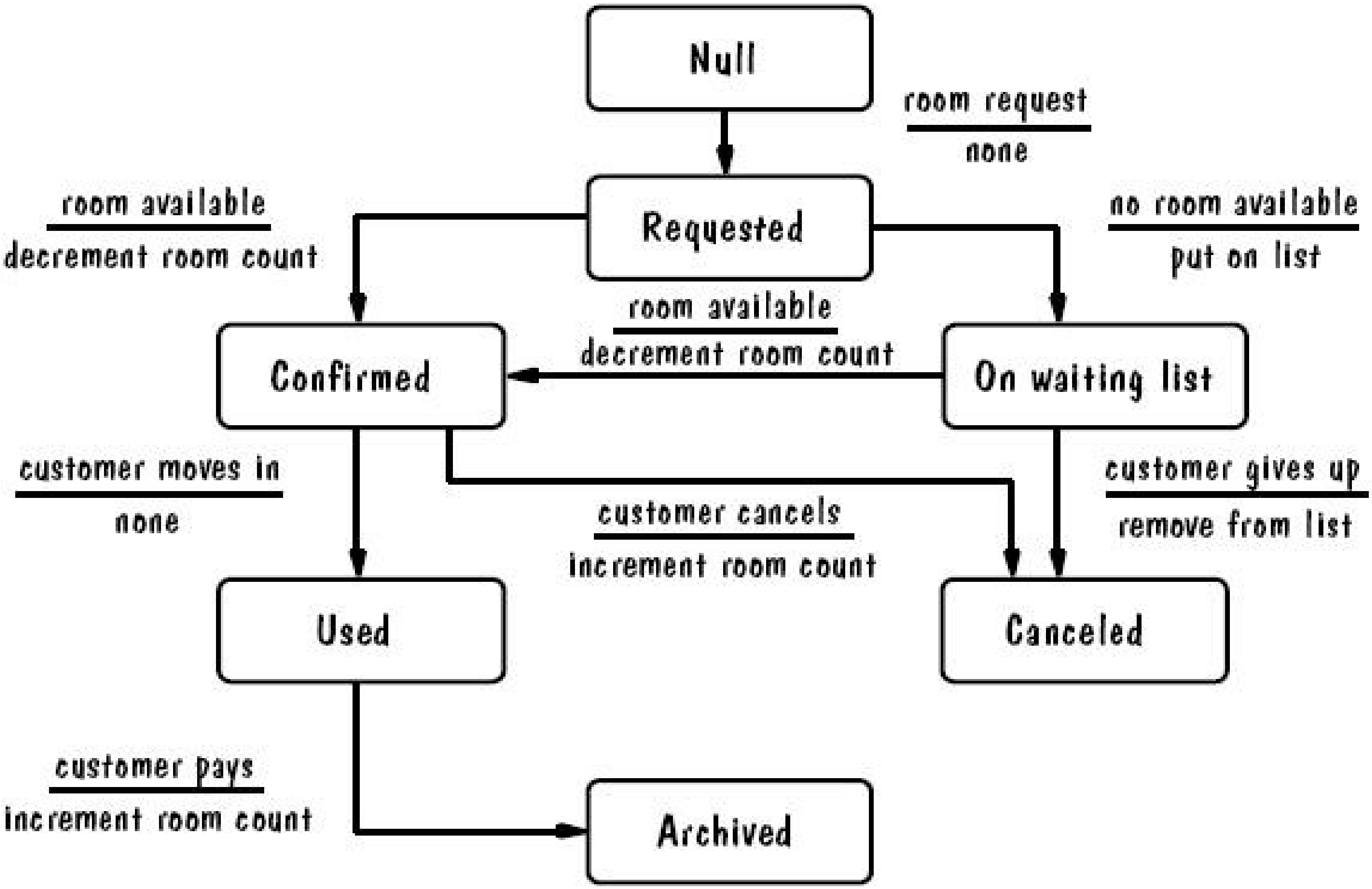


Start State Stop State

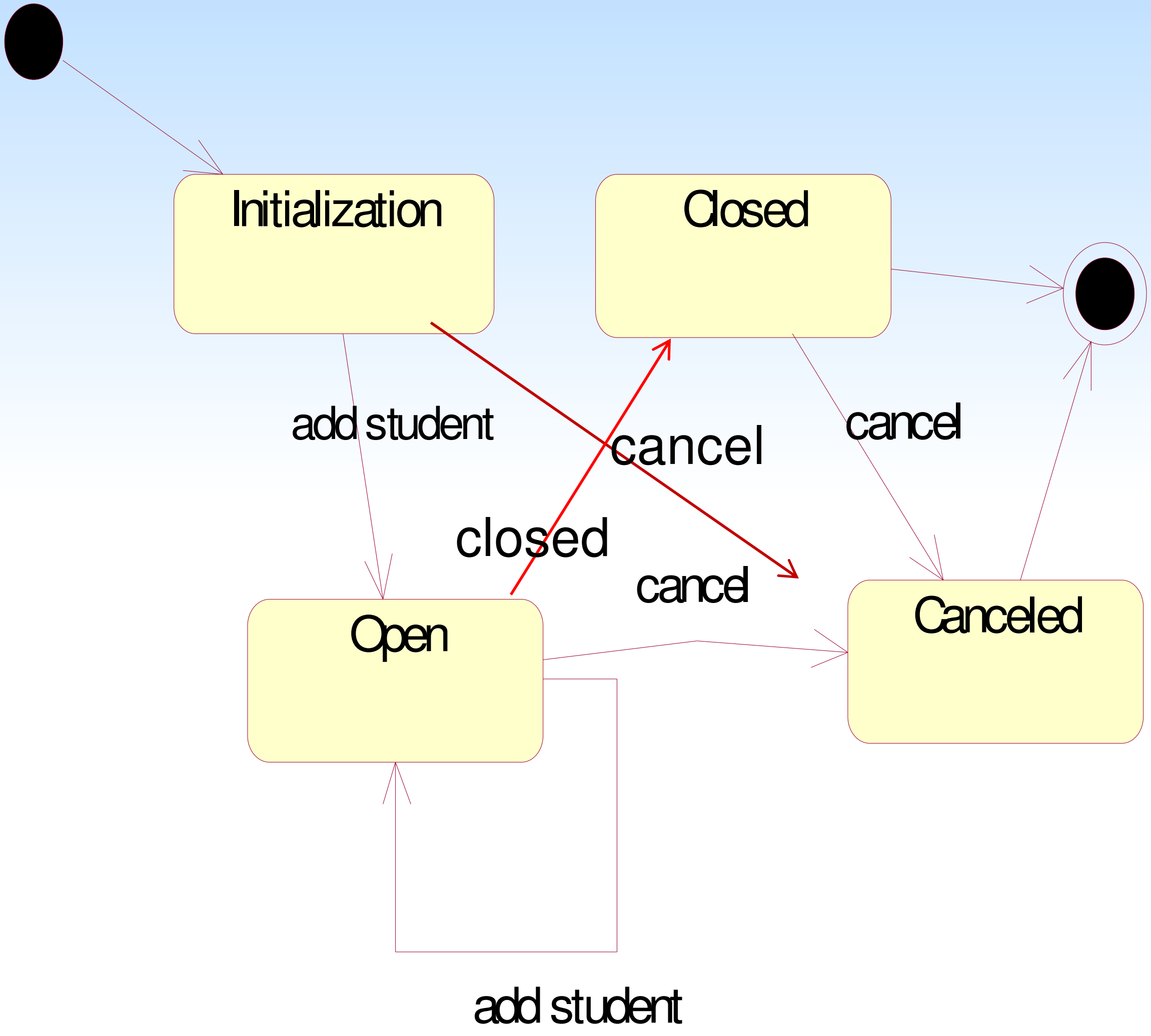
# State Transition Diagram –Hotel Room Class



# Hotel Room –


Null
room request
none
Requested
room available
decrement room count
no room available
put on list
Confirmed
room available
decrement room count
On waiting list
customer moves in
none
customer cancels
increment room count
customer gives up
remove from list
Used
Canceled
customer pays
increment room count
Archived

# State Transition Diagram– Course Class



# State Transition Details

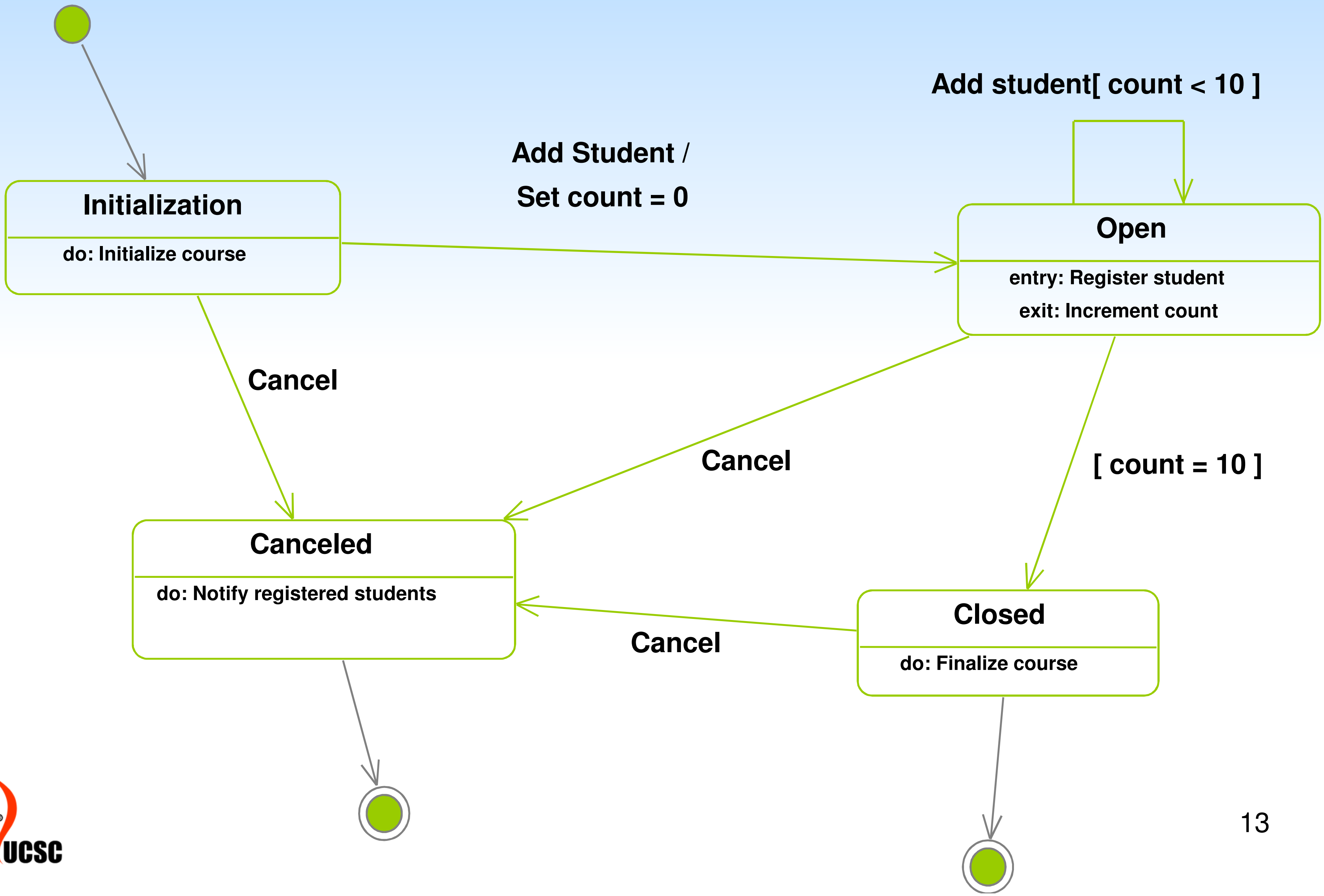
- A State Transition may have the following associated with:
  - an action and/or

    (behaviour that occurs when the state transition occurs.)
  - a guard condition

    (allows state transition only if it is true.)
- A State Transition may also trigger an event

  A message that is sent to another object in the system.

# State Transition Diagram
## Course Offering with State Details



**Initialization**

do: Initialize course

Add Student /

Set count = 0

Add student[ count < 10 ]

**Open**

entry: Register student

exit: Increment count

Cancel

Cancel

[ count = 10 ]

**Canceled**

do: Notify registered students

Cancel

**Closed**

do: Finalize course

# State Details

- ***Activity*** : behaviour that an object carries out while it is in a particular state.
  - An activity is shown inside the state itself, preceded by the word *do* and a colon.
- **Entry Action** :
  - Behaviour that occurs while the object is transitioning into the state.
  - Shown inside the state, preceded by the word *entry* and colon.

# State Details cont…

- **Exit Action :** occurs as part of the transition out of a state.
  - Shown inside the state, preceded by the word *exit* and colon.
- The behaviour in an activity, entry action, or exit action can include sending an event to some other object.

# State Details con…

- In this case, the activity, entry action, or exit action is preceded by a **^**

**Do:^Target.Event(Arguments)**

*Target* - object receiving the event

*Event* - message being sent

*Arguments* – parameters of the message being sent

Eg.

Do:^CourseRoster.Create

# Sub States

- The GUI that we interact in a system, can be in one of three states.
  - Initializing
  - Working
  - Shutting Down

- As a result of activities in the initializing state, the GUI transitions into working state.

- When one chooses to shut down the PC, trigger event is generated that causes the transition to shutdown state, and eventually PC turns off.

# Sub States

- When GUI is in the working state, a lot is happening behind the scenes.

  Eg. Type a keystroke, move the mouse, press a mouse button etc.

- It then must register those inputs and change the display to visualize those actions for you onscreen.

# Sub States

- Sub states come in two varieties
  - Sequential , Concurrent
- Sequential sub states occur one after the other.
  - e.g. Sub states of Working state
  - Awaiting user input, registering user input, visualizing user input

# Sub States

- User input triggers the transition from awaiting to registering
- Activities within registering transition the GUI into visualizing.

# Sub States

- Thus the GUI goes through changes while its within the working state.
- Those changes are changes of State.
- They are called **Sub states** because they reside within a state.

# Sub States

- Sub states come in two varieties.
  - Sequential , Concurrent
- Sequential Sub state
  - Occur one after the other.

  Eg. Sub states within the GUI’s Working state

# Sub States

- Concurrent Sub state
  - Within the working state, the GUI is not just waiting for you.
  - It is also watching the system clock and updating an applications display.
  - e. g. Application might include an onscreen clock that the GUI has to update.

# Sub States

- Concurrent Sub state cont…
  - The sequences are concurrent with one another.
  - Concurrent sub states proceed at the same time.
  - A dotted line separate concurrent sub states.

# UML 2.0 State Diagrams

- UML 2.0 has added some new state relevant symbols called **connection points.**
- They represent points of entry into a state or exists out of a state.
- Lets look at the different state of a book in a library.

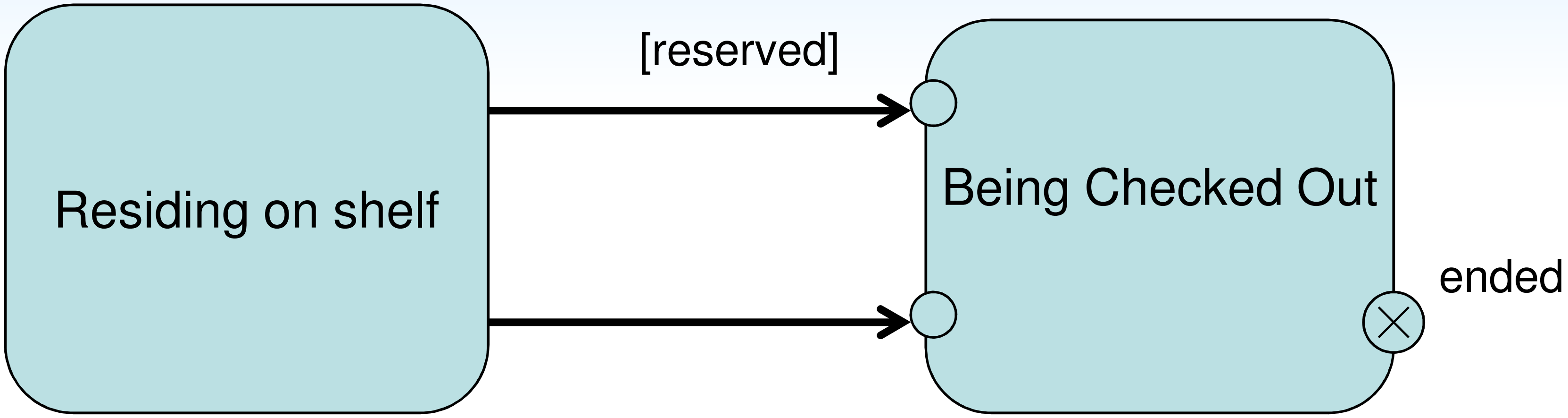
# UML 2.0 State Diagrams

- At first the book is residing on a shelf.
- If a borrower has called in to reserve the book, a librarian retrieves the book and brings it into the state of *"Being checked out".*
- If a borrower comes to the library, browses through the shelves, selects the book, and decides to borrow it.
- Again it enters into the state of *"Being checked out"*, but in a different way.

# UML 2.0 State Diagrams

- You can think of each way of getting to the Being-checked-out state as going through a separate **entry point**
- Suppose the borrower is trying to borrow more than the allotted limit or has number of unpaid fines.
- If that is the case the book abruptly exits via an **exit point,** from “*Being-checked-out” state*

# Entry points and exit point in a UML state diagram



# Why are State diagrams important?

- They model the changes that just one object goes through.
- They help analysts, designers, and developers understand the behavior of the objects in a system.
- A Class diagram and an object diagram show  only static aspects of a system.  They do not show the dynamic details of the behaviors.

# Why are State diagrams important?

- Developers, in particular, have to know
  - how objects are supposed to behave because they have to implement these behaviors in software.
  - It is not enough to implement only objects.
  - Developers have to make that object do something.
- State diagrams ensure that they won’t have to guess about what the object is supposed to do.