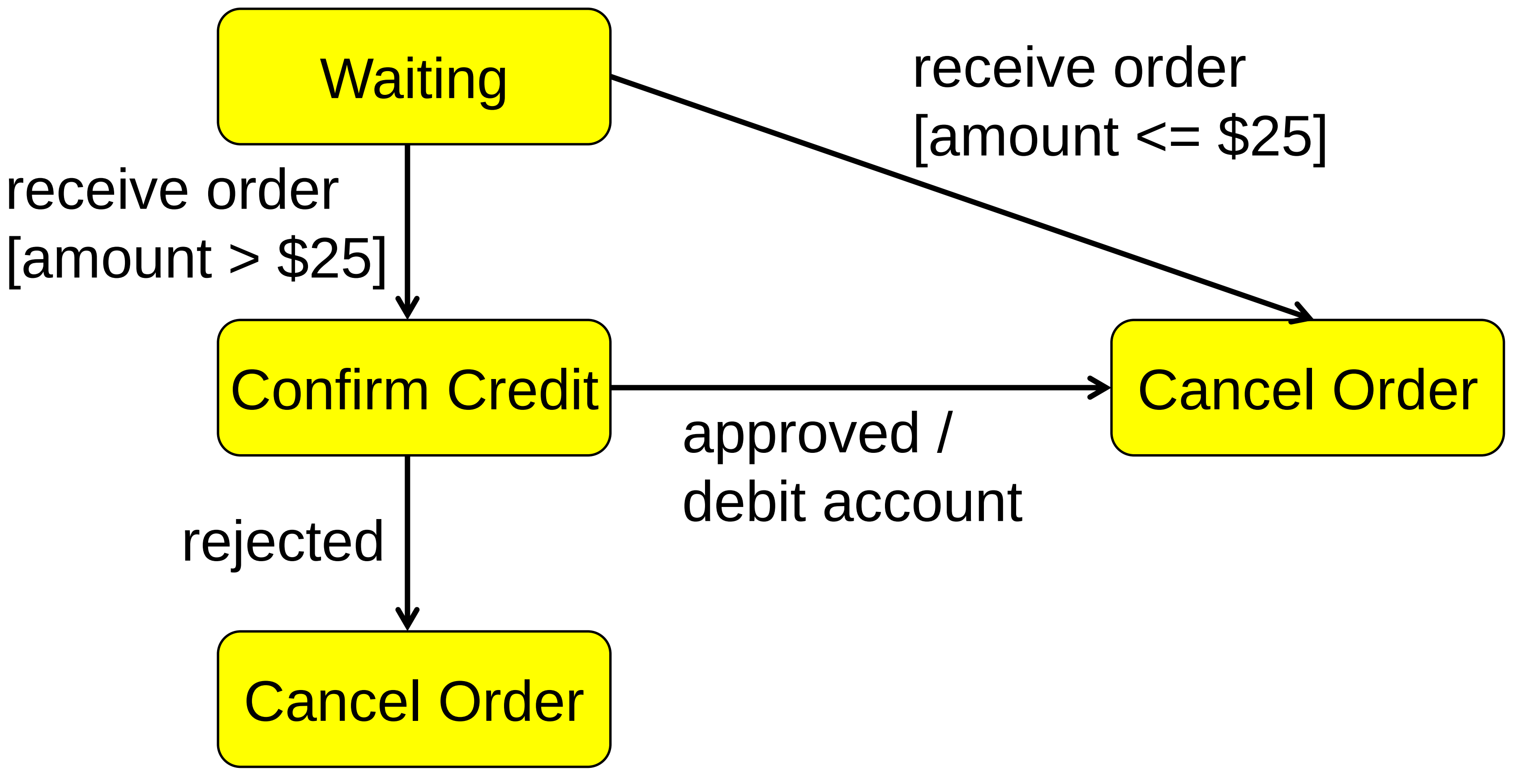
# Finite State Machine Testing

# Finite State Machines

- A finite state machine is a model to describe the dynamic behaviors of an object over time.
- A finite state machine is a localized view of an object.
- Each object is treated as an isolated entity that communicates with the rest of the world by detecting events and responding to them.

# An Example: UML State Diagrams



# EclipseUML

- EclipseUML is a UML editor and an Eclipse plugin.
- EclipseUML can draw all the diagrams in the UML 2.1.
- EclipseUML can be downloaded from http://www.eclipsedownload.com/

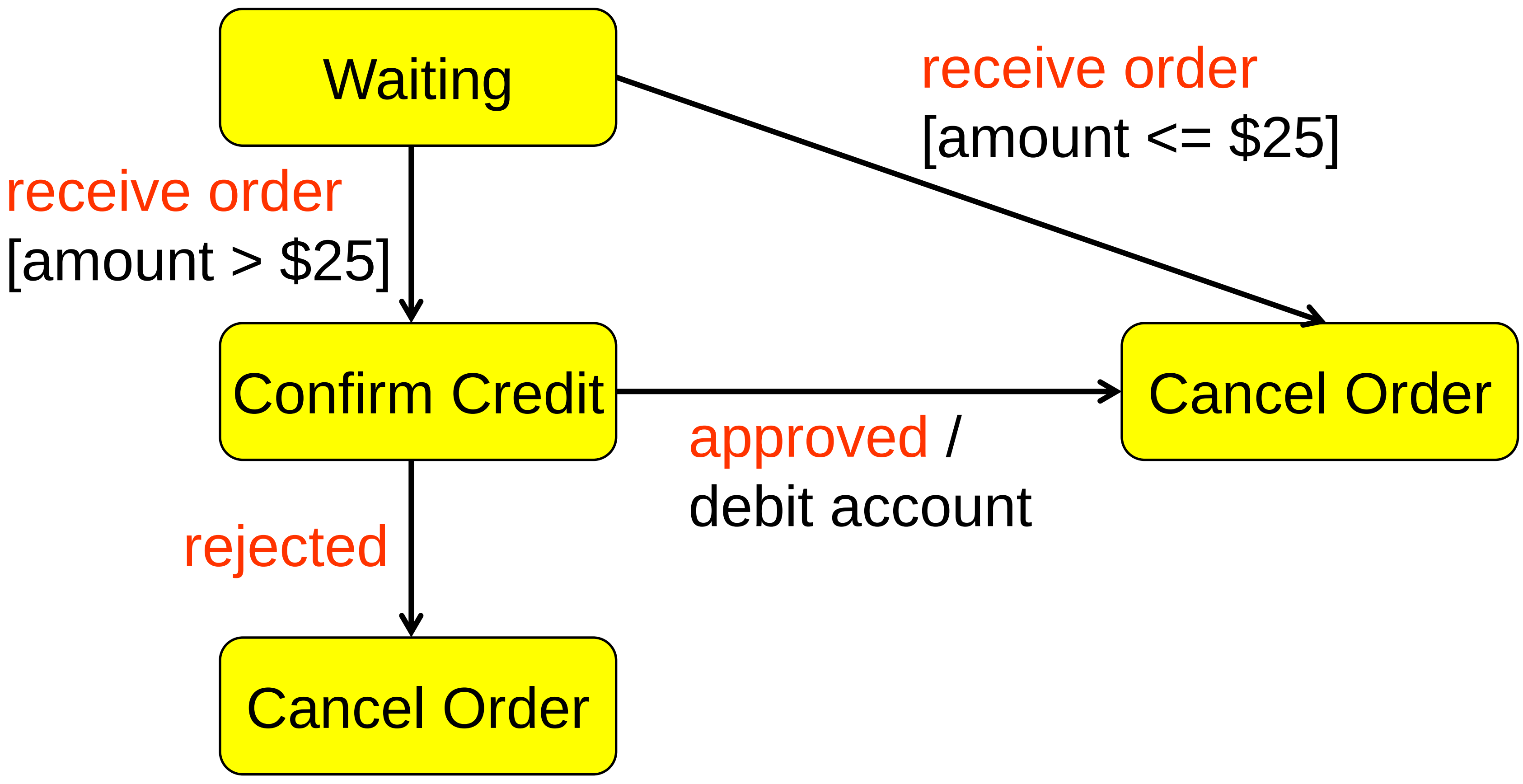
# Events

- Events represents the kinds of changes that an object can detect – the receipt of calls or explicit signals from one object to another, a change in certain values, or the passage of time.
- Anything that can affect an object can be characterized as an event.
- An event occurs at a point in time; it does not have duration.

# Event Types

- Call event: receipt of an explicit synchronous call request by an object – op(a:T).
- Change event: a change in value of a Boolean expression – when(exp).
- Signal event: receipt of an explicit, named, asynchronous communication among objects – sname(a:T).
- Time event: the arrival of an absolute time or the passage of a relative amount of time – after(time).

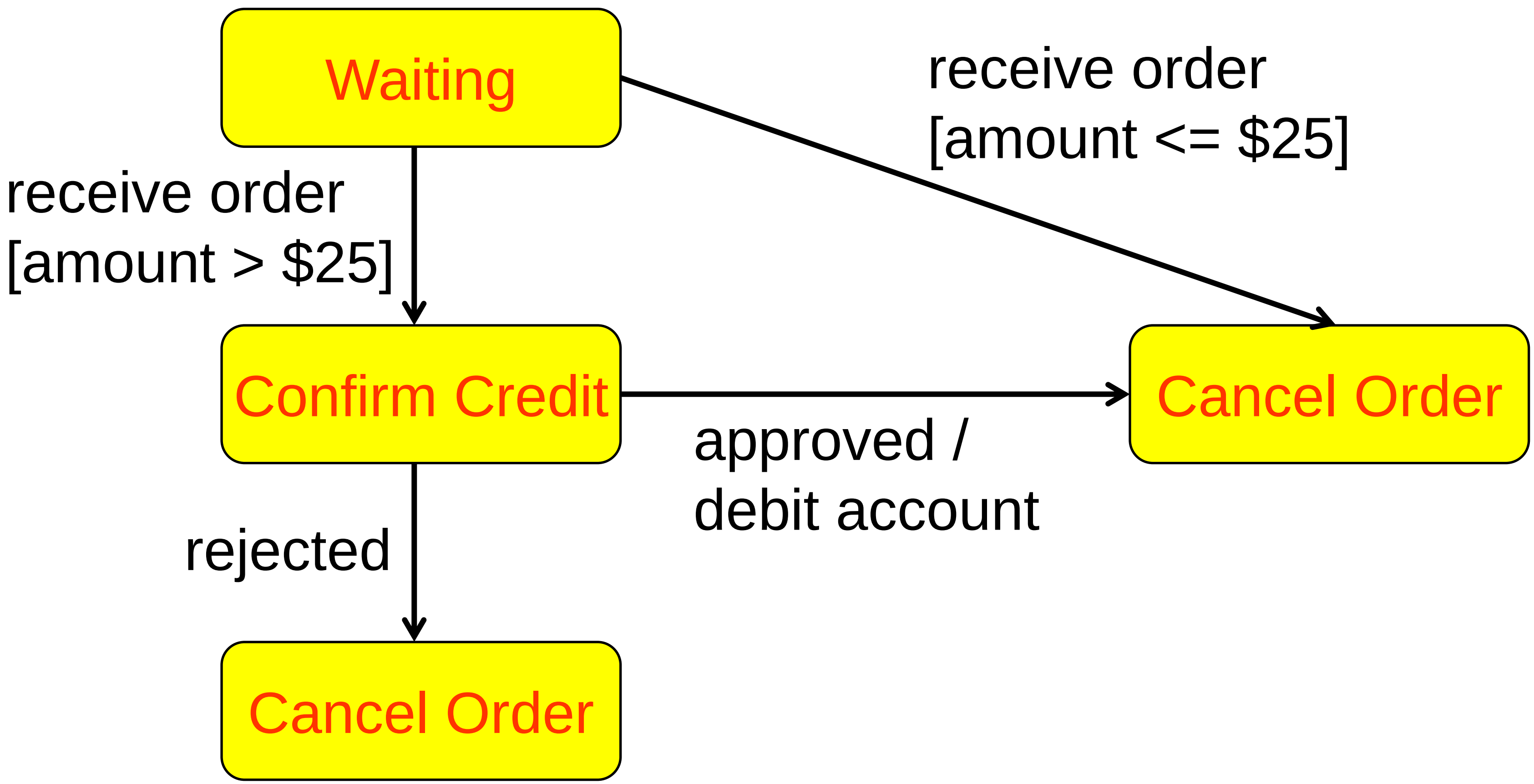
# An Example


Waiting
receive order
[amount <= $25]
receive order
[amount > $25]
Confirm Credit
Cancel Order
approved /
debit account
rejected
Cancel Order

# States

- A finite state machine defines a number of states.
- A state can be characterized in three complementary ways:
- A set of object values that are qualitatively similar in some respect;
- A period of time during which an object waits for some event or events to occur;
- A period of time during which an object performs some ongoing do activity.

# An Example

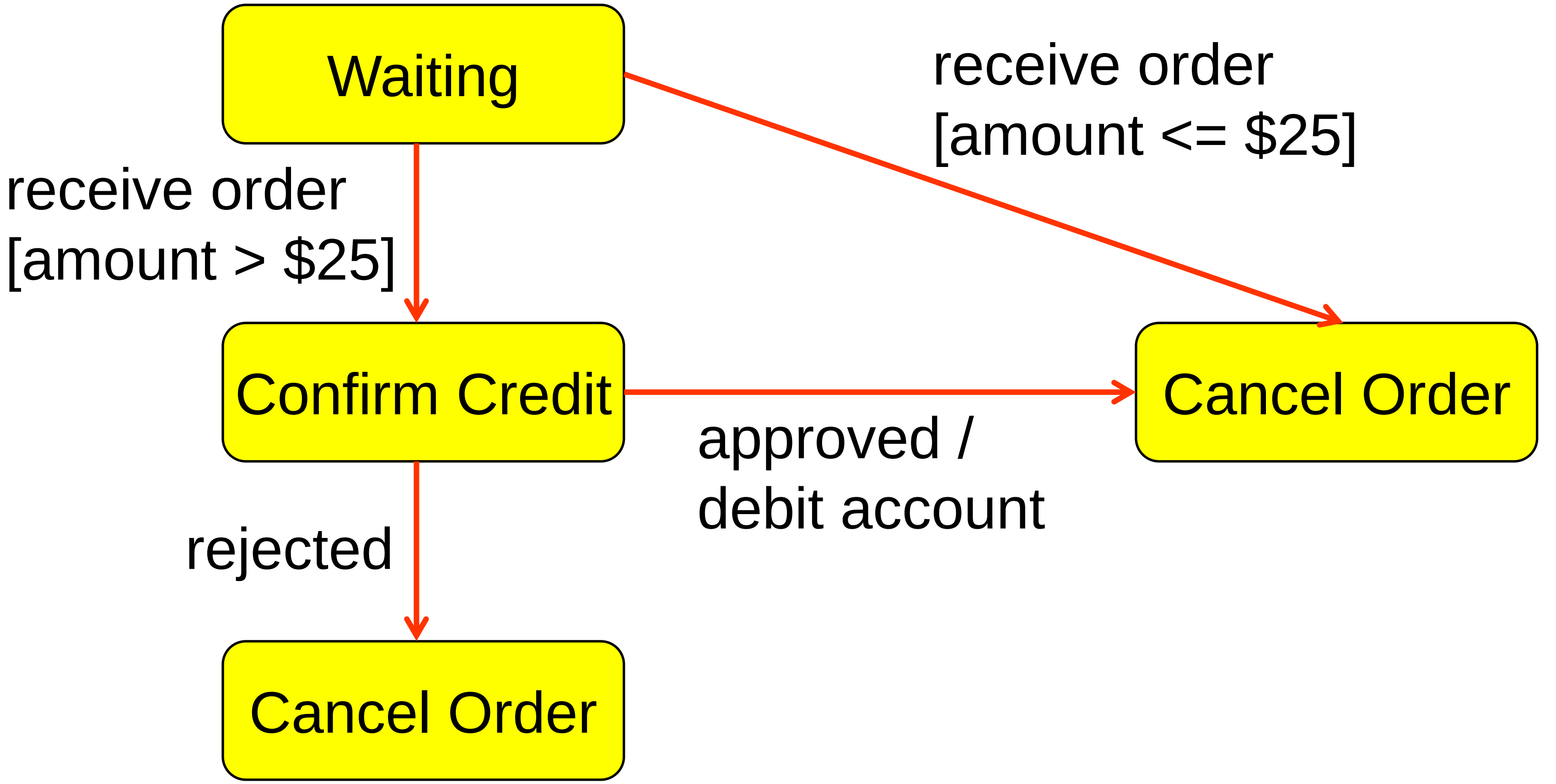


Waiting
receive order
[amount <= $25]
receive order
[amount > $25]
Confirm Credit
Cancel Order
approved /
debit account
rejected
Cancel Order

# Transitions

- A transition leaving a state defines the response of an object in the state to the occurrence of an event.
- In general, a transition has an event trigger, a guard condition, an effect, and a target state.
  – e(a:T)[guard]/activity.

# An Example


Waiting
receive order
[amount <= $25]
receive order
[amount > $25]
Confirm Credit
Cancel Order
approved /
debit account
rejected
Cancel Order

# Event Triggers

- An event trigger specifies the event that enables a transition.
- The event may have parameters, which are available to an effect specified as part of the transition.

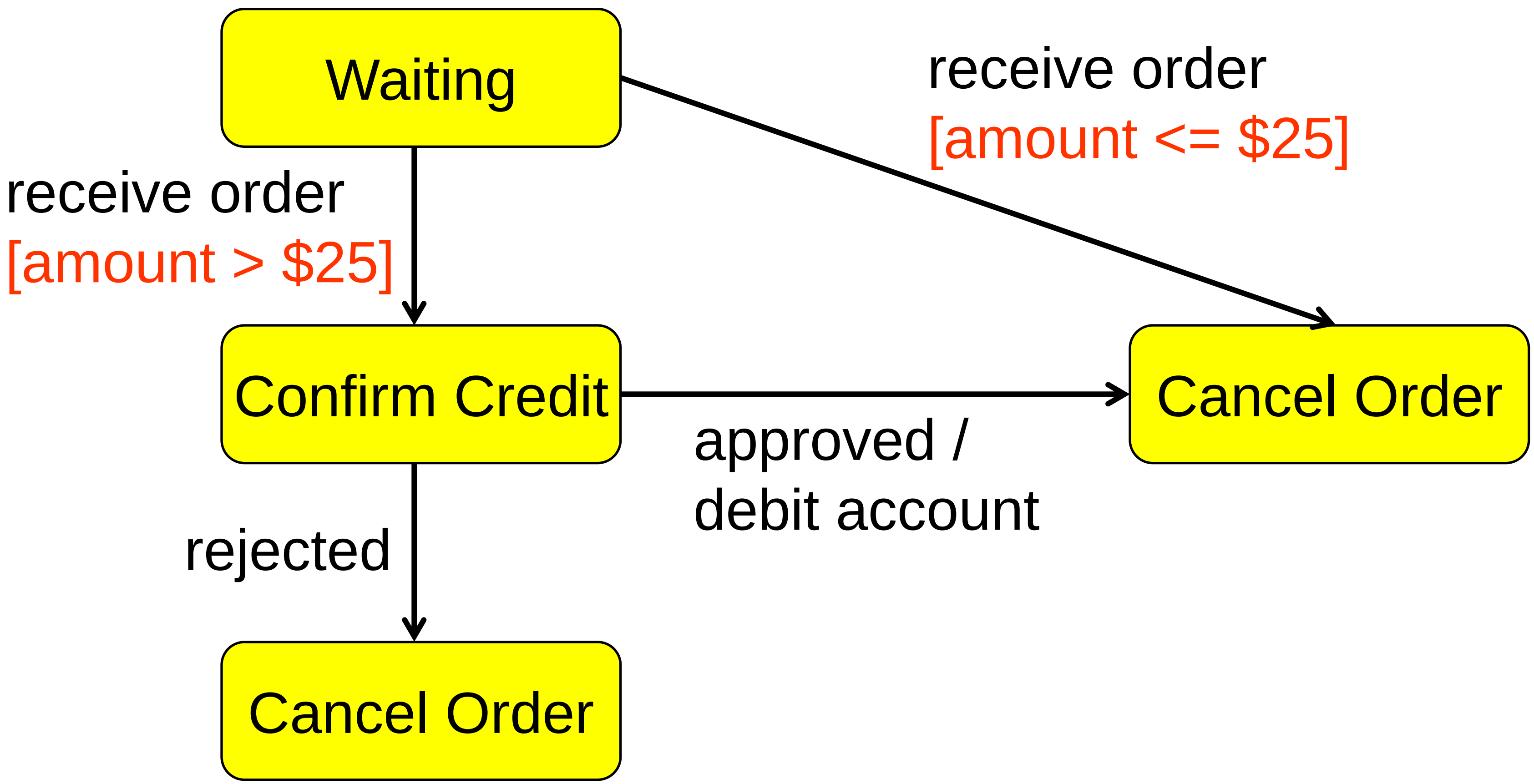
# Guard Conditions

- A transition may have a guard condition, which is a Boolean expression.
- It may reference attributes of the objects that owns the finite state machine, as well as parameters of the trigger event.
- The guard condition is evaluated when the trigger event occurs.
- If the expression evaluates as true, then the transition fires, that is, its effects occur; otherwise, the transition does not fire.

# Guard Conditions

- The same event can be a trigger for more than one transition leaving a single state.
- Each transition with the same event must have a different guard condition.
- Often, the set of guard conditions covers all possibilities so that the occurrence of the event is guaranteed to fire some transition.
- Only one transition may fire in response to one event occurrence.

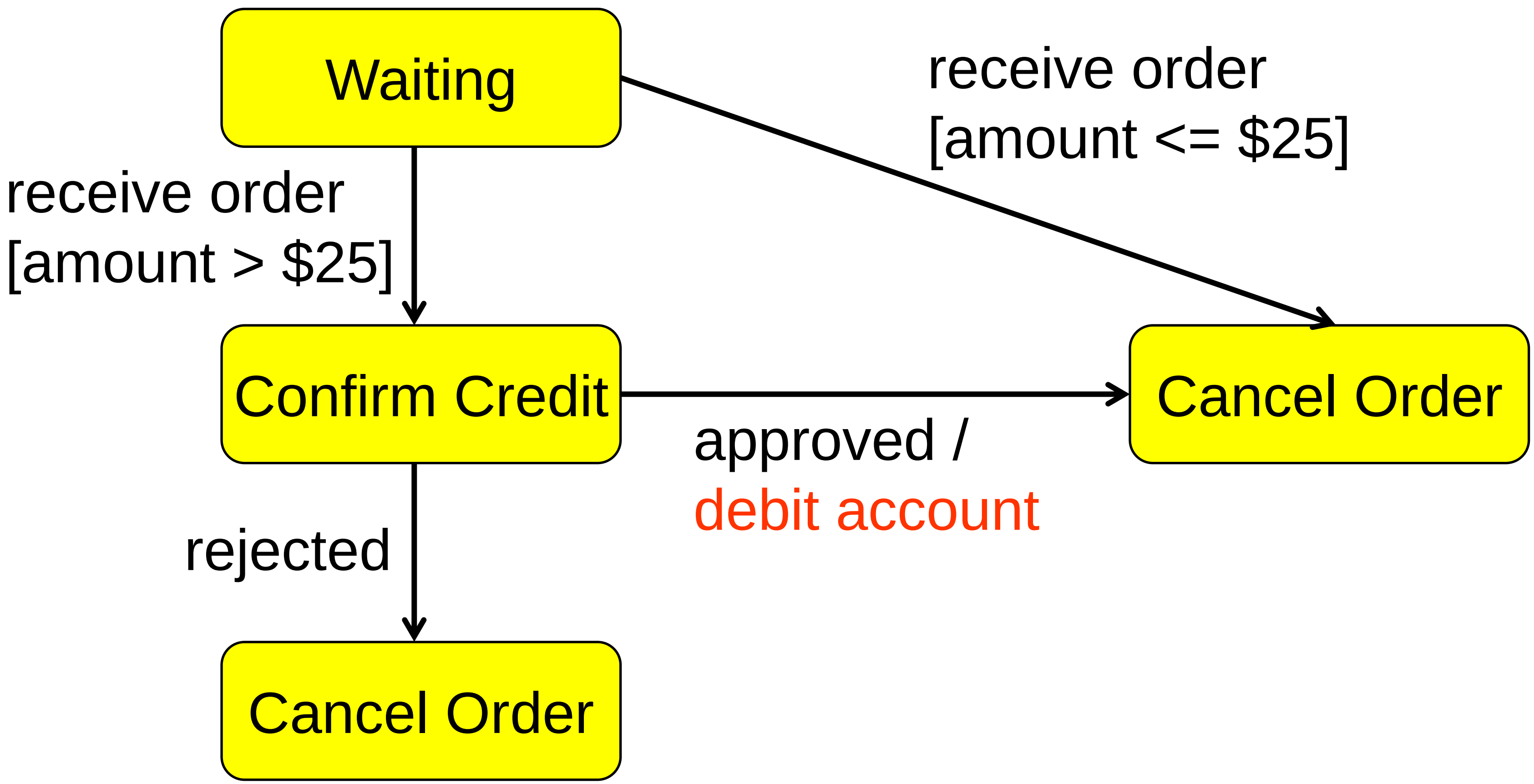
# An Example


Waiting
receive order
[amount <= $25]
receive order
[amount > $25]
Confirm Credit
Cancel Order
approved /
debit account
rejected
Cancel Order

# Effects

- When a transition fires, its effect (if any) is executed.
- An effect may be an action or an activity.
- An action is a primitive computation, such as an assignment statement, a simple arithmetic computation, sending a signal to another object, calling an operation, creating or destroying an object, and getting and setting attribute values.
- An activity is a list of actions or activities.

# An Example



Waiting
receive order
[amount <= $25]
receive order
[amount > $25]
Confirm Credit
Cancel Order
approved /
debit account
rejected
Cancel Order

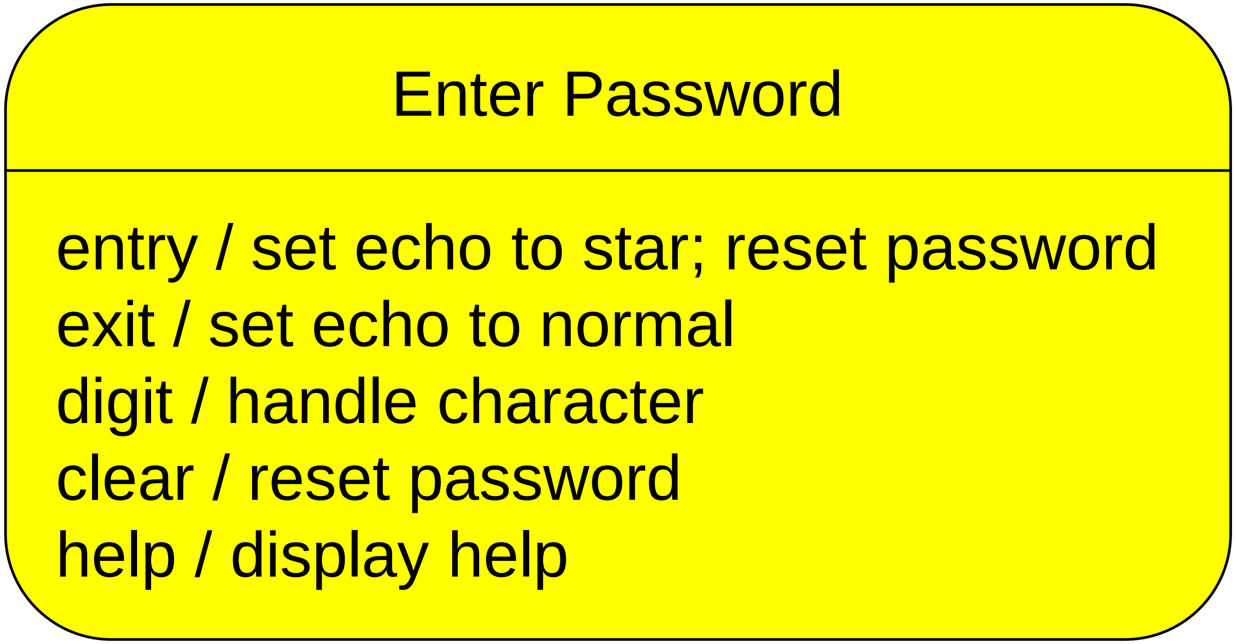
# Change of State

- When the execution of the effect is complete, the target state of the transition becomes active.

# Activities in States

- Entry activity: that is executed when a state is entered – entry/activity.
- Exit activity: that is executed when a state is exited – exit/activity.
- Internal activity: that is executed after the entry activity and before the exit activity – e(a:T)[guard]/activity.

# An Example


Enter Password
entry / set echo to star; reset password
exit / set echo to normal
digit / handle character
clear / reset password
help / display help

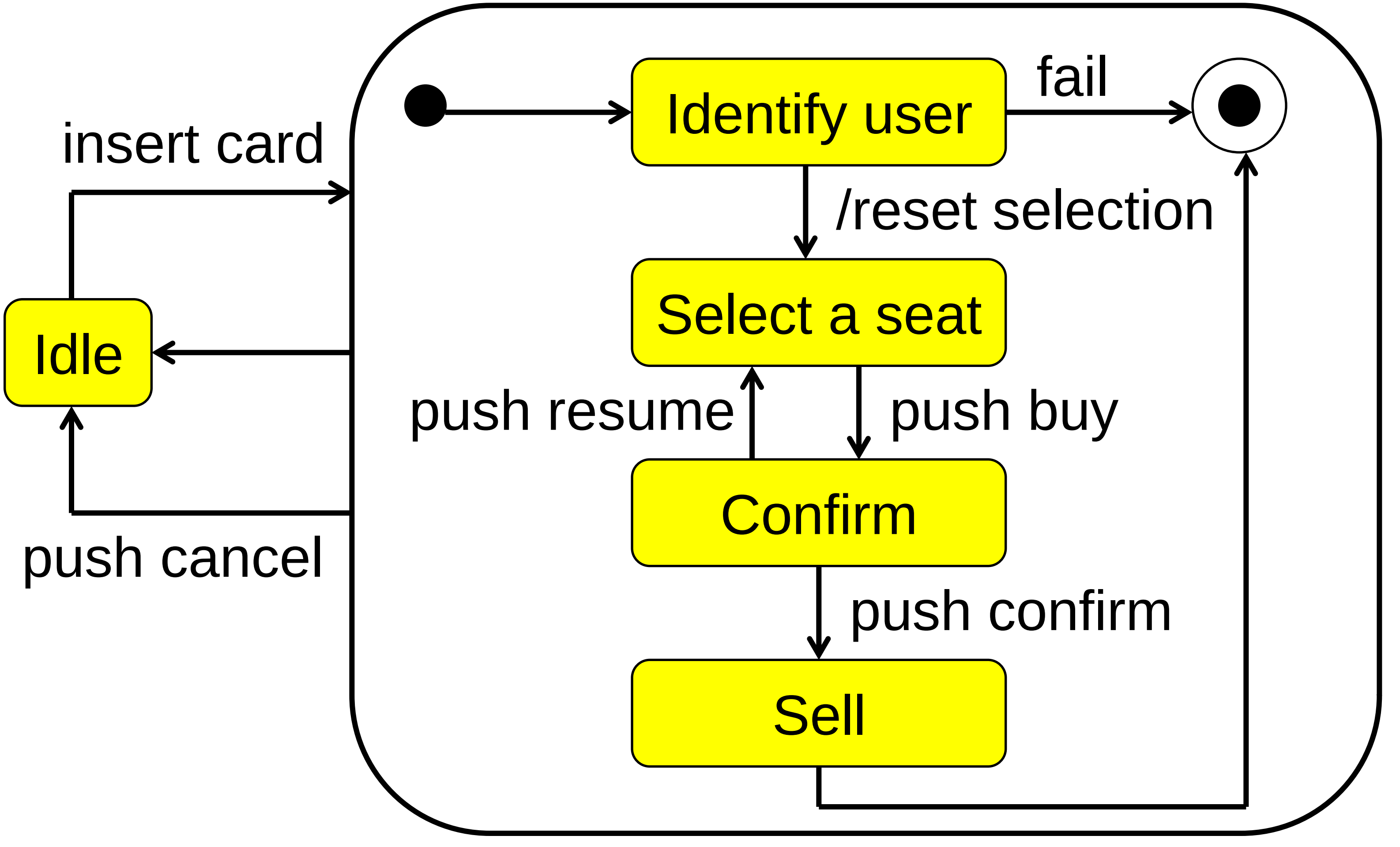
# State Types

- Initial state: a psudostate that indicates the starting state when the enclosing state is invoked.
- Final state: a special state whose activation indicates the enclosing state has completed activity.
- Terminate: a special state whose activation terminates execution of the object owning the state machine.

# State Types

- Simple state: a state with no substructure.
- Nonorthogonal state: a composite state that contains one or more direct substates, exactly one of which is active at one time when the composite state is active.

# An Example

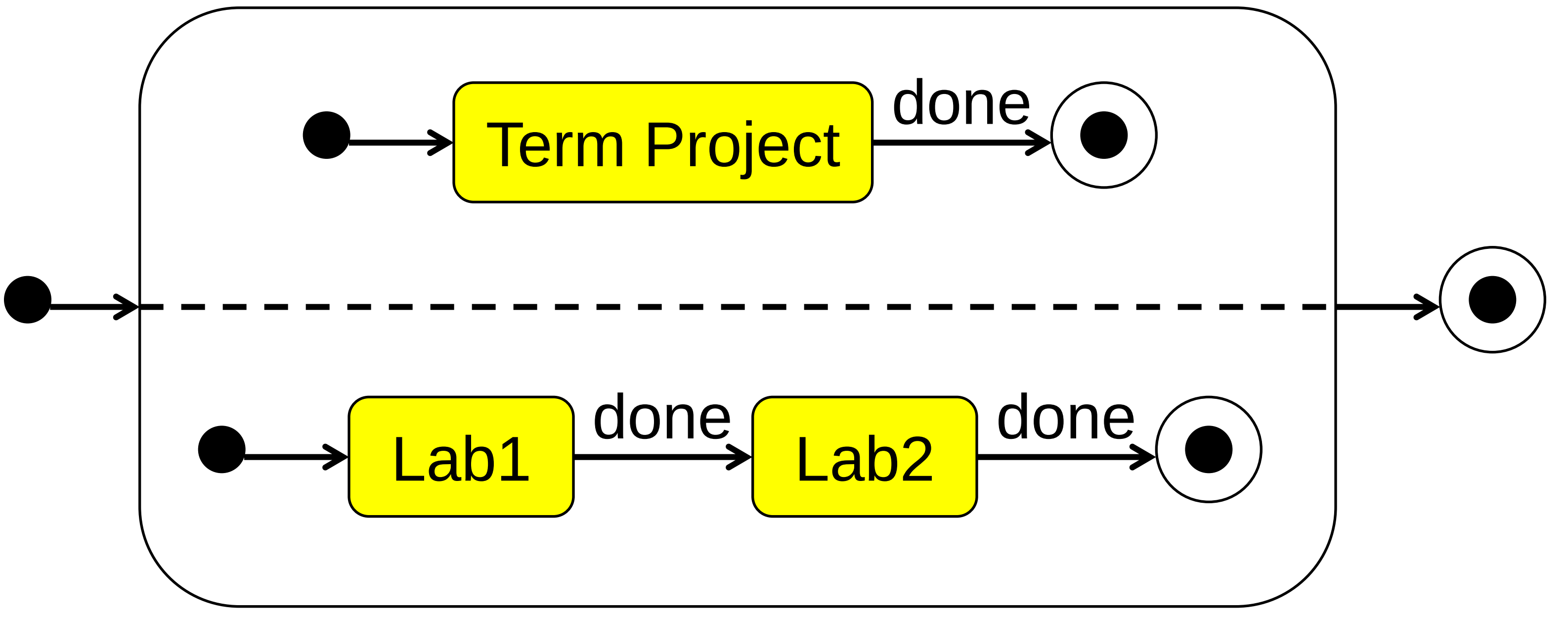


insert card
Idle
push cancel
Identify user
fail
/reset selection
Select a seat
push resume
push buy
Confirm
push confirm
Sell

# State Types

- Orthogonal state: a composite state that is divided into two or more regions. One direct substate from each region is concurrently active when the composite state is active.

# An Example



# Test Coverage Criteria

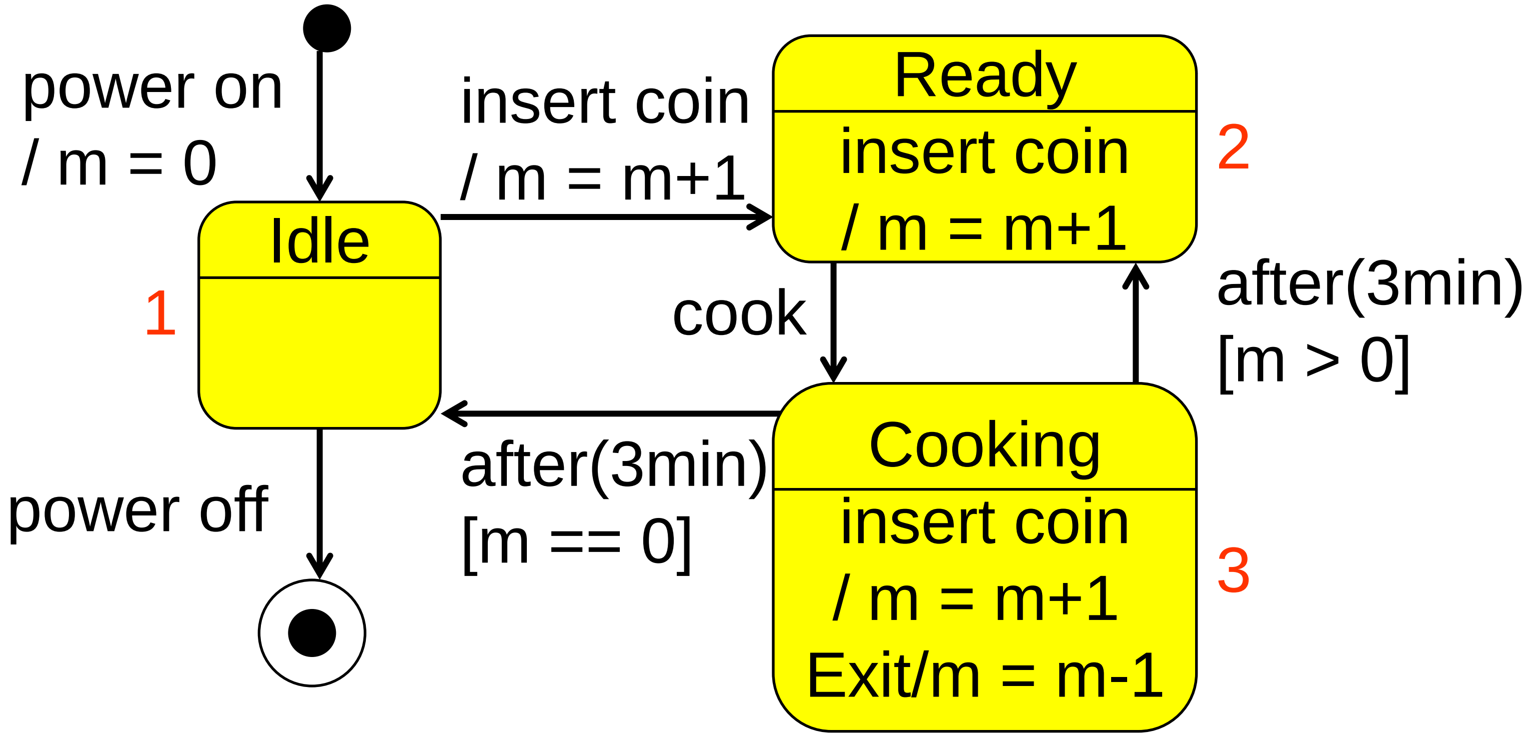
- All-state coverage
- All-transition coverage
- All-definition coverage
- All-use coverage
- All-definition-use coverage
- All-path coverage

All-state coverage and All-transition coverage: Control flow

All-definition coverage, All-use coverage and All-definition-use coverage: Data flow

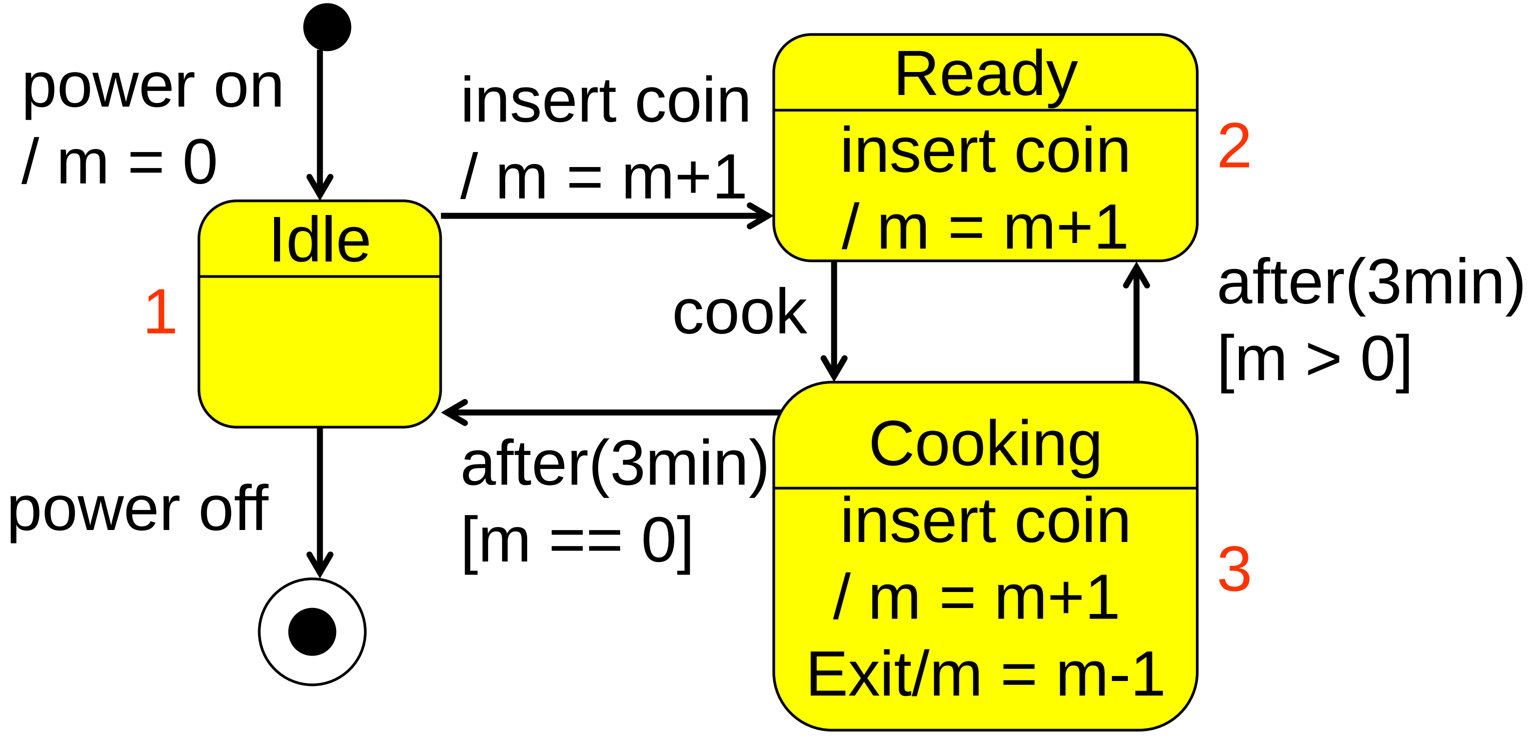
All-path coverage: Both

# An Example: Coffee Cooking Machine



# All-State Coverage

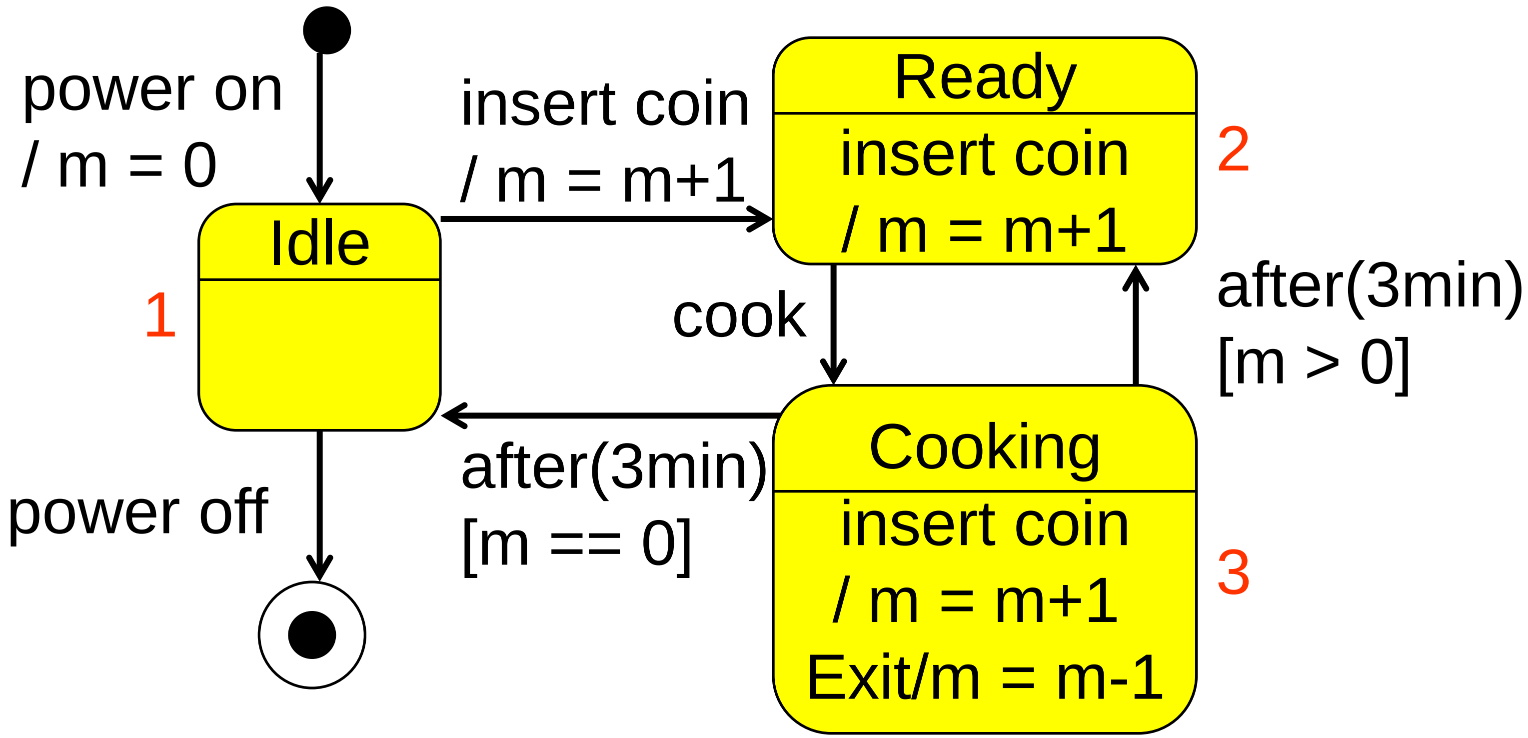
The set of test cases covers all the states in the diagram




1 → 2 → 3 → 1

# All-Transition Coverage

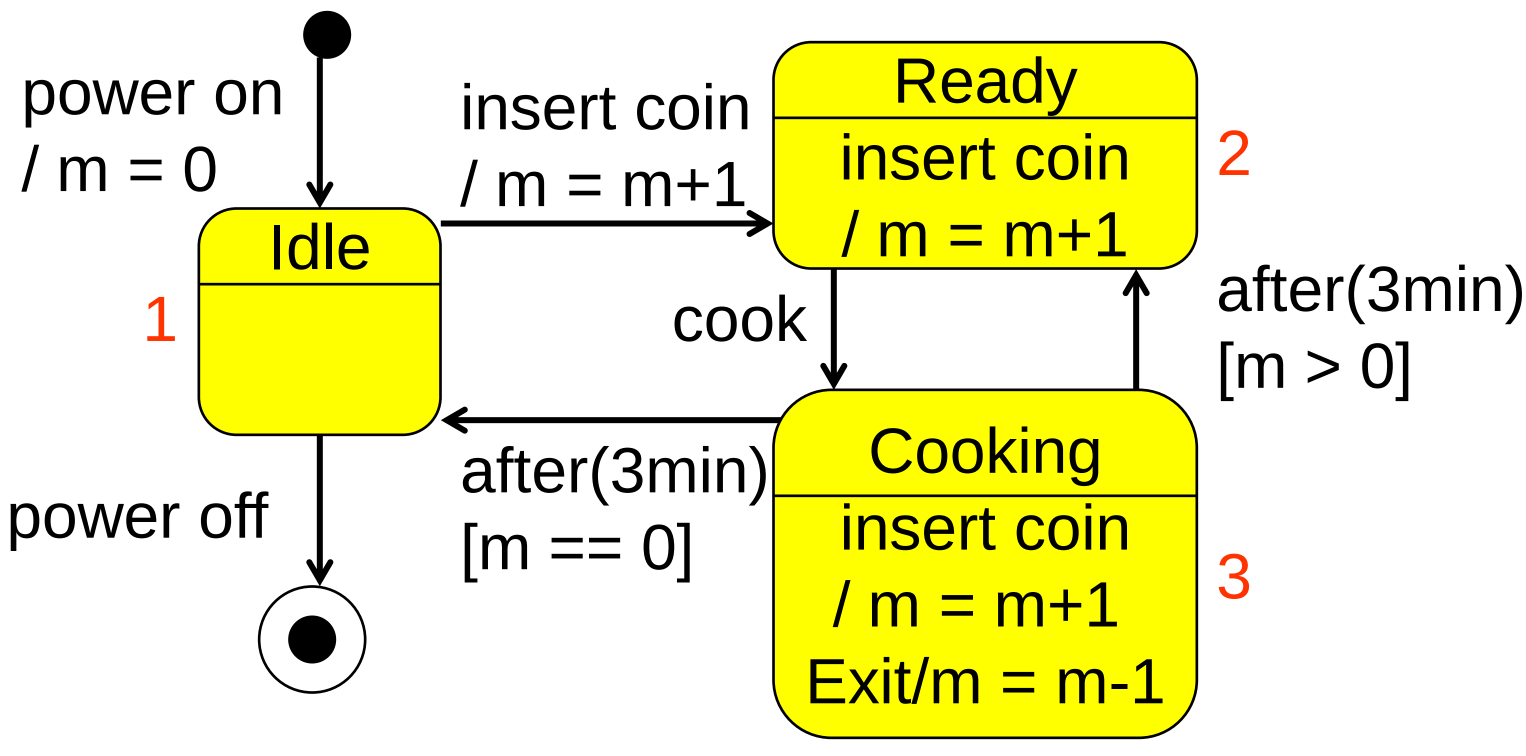
The set of test cases covers all the transitions in the diagram




1 → 2 →* 2 → 3 → 2 → 3 → 1

# All-Path Coverage

The set of test cases covers all the paths in the diagram

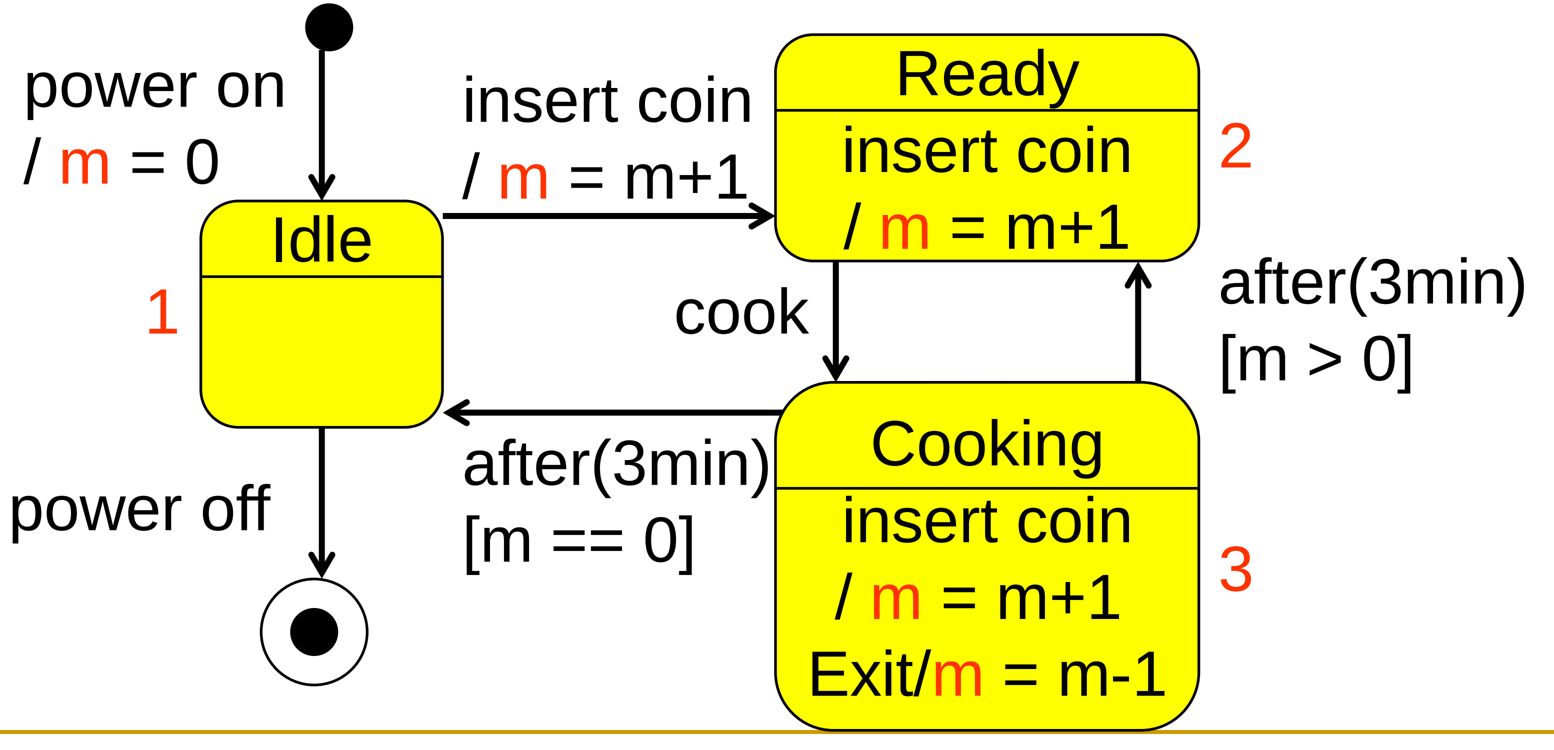



$1 \rightarrow 2 \rightarrow 3 \rightarrow 1$

$1 \rightarrow 2 \rightarrow^* 2 \rightarrow 3 \rightarrow 2 \rightarrow 3 \rightarrow 1$ … (infinite)

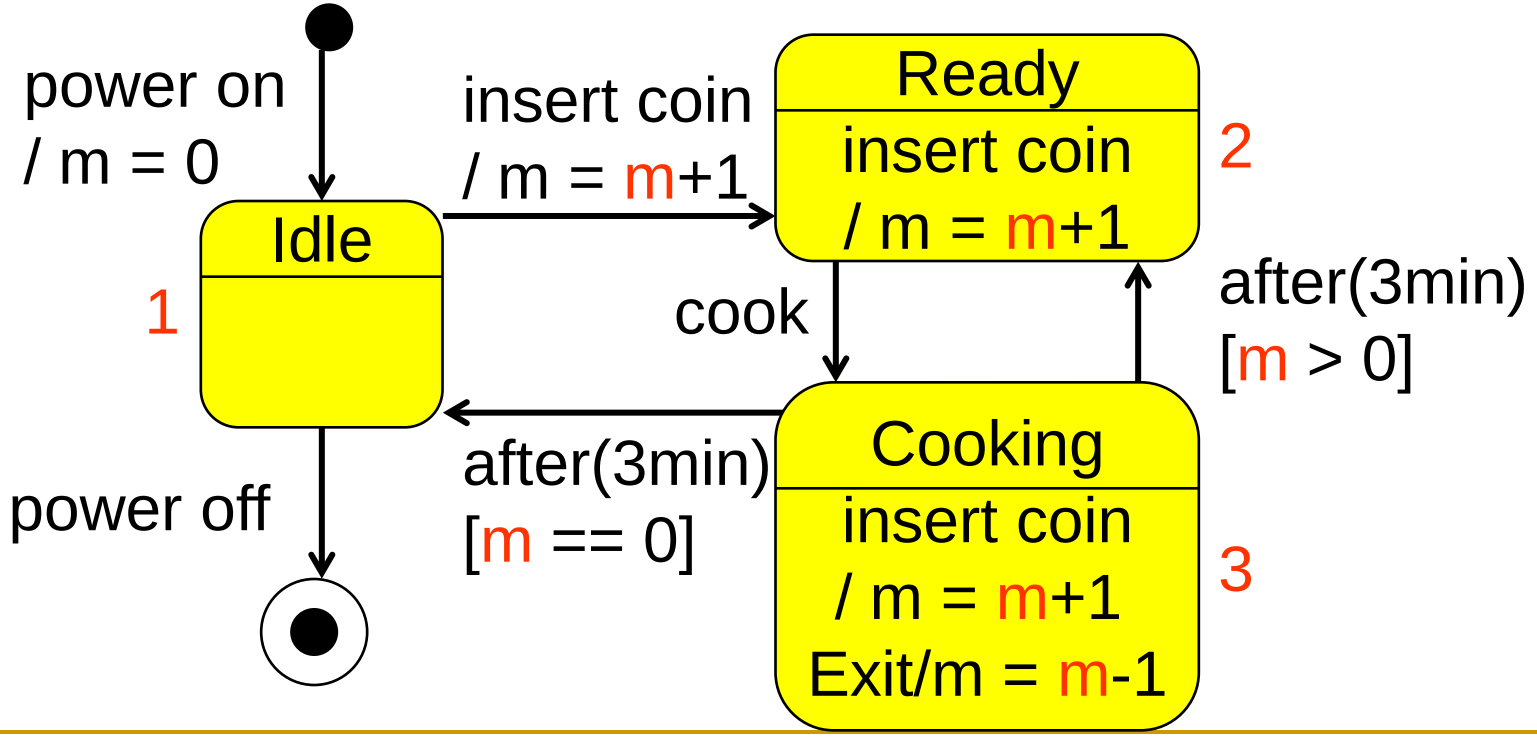
# Definitions of Variables

- An occurrence of a variable is a definition of the variable if a value is bound to the variable at that occurrence.



# Uses of Variables

- An occurrence of a variable is a use of the variable if the value of the variable is referred at that occurrence.

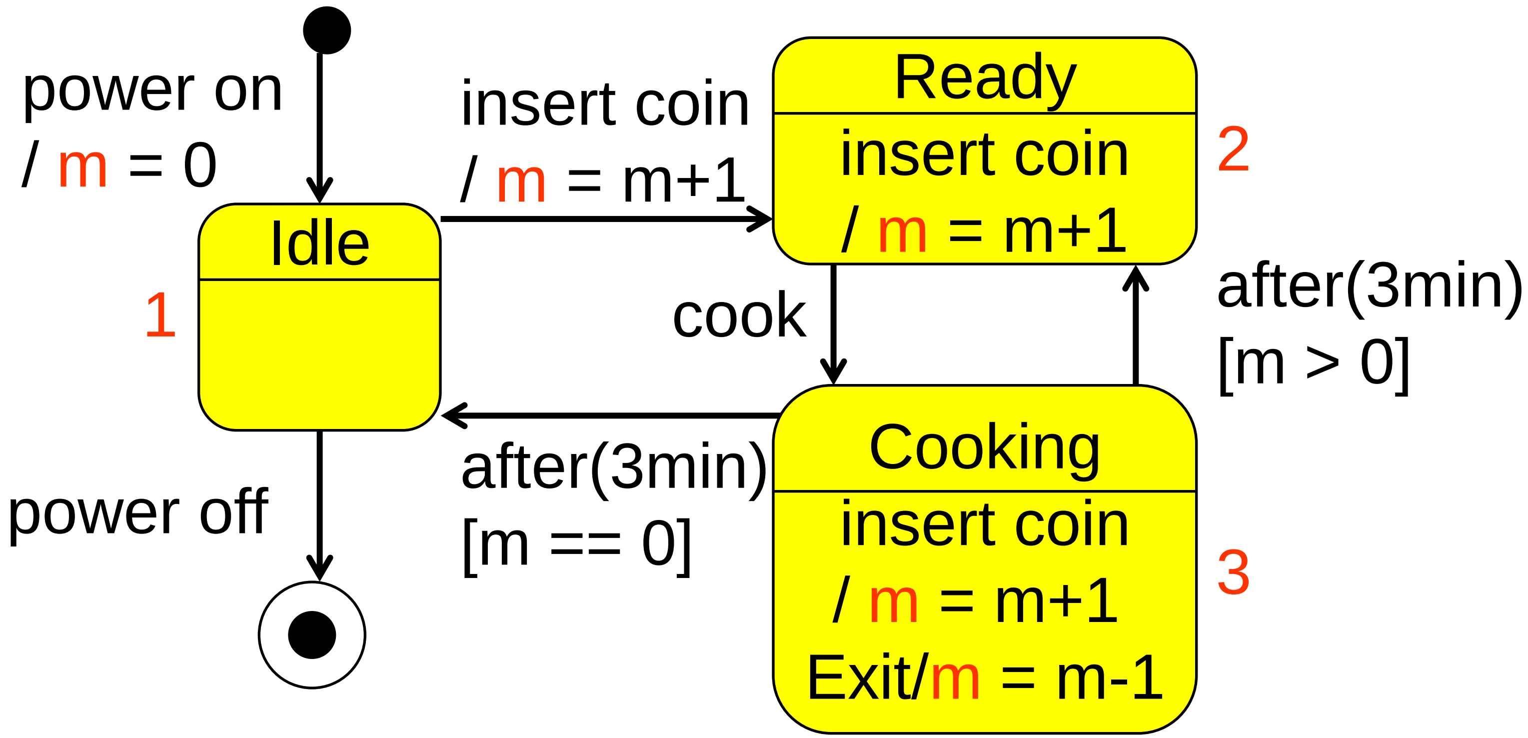


# Definition-Use Pairs

- The value of a definition of a variable may be used by several different uses of the variable.
- A use of a variable may use the value defined by several different definitions of the variable.
- Each definition and each of its uses compose a definition-use pair.
- The set of definition-use pairs includes all the data flow relations.

# All-Definition Coverage

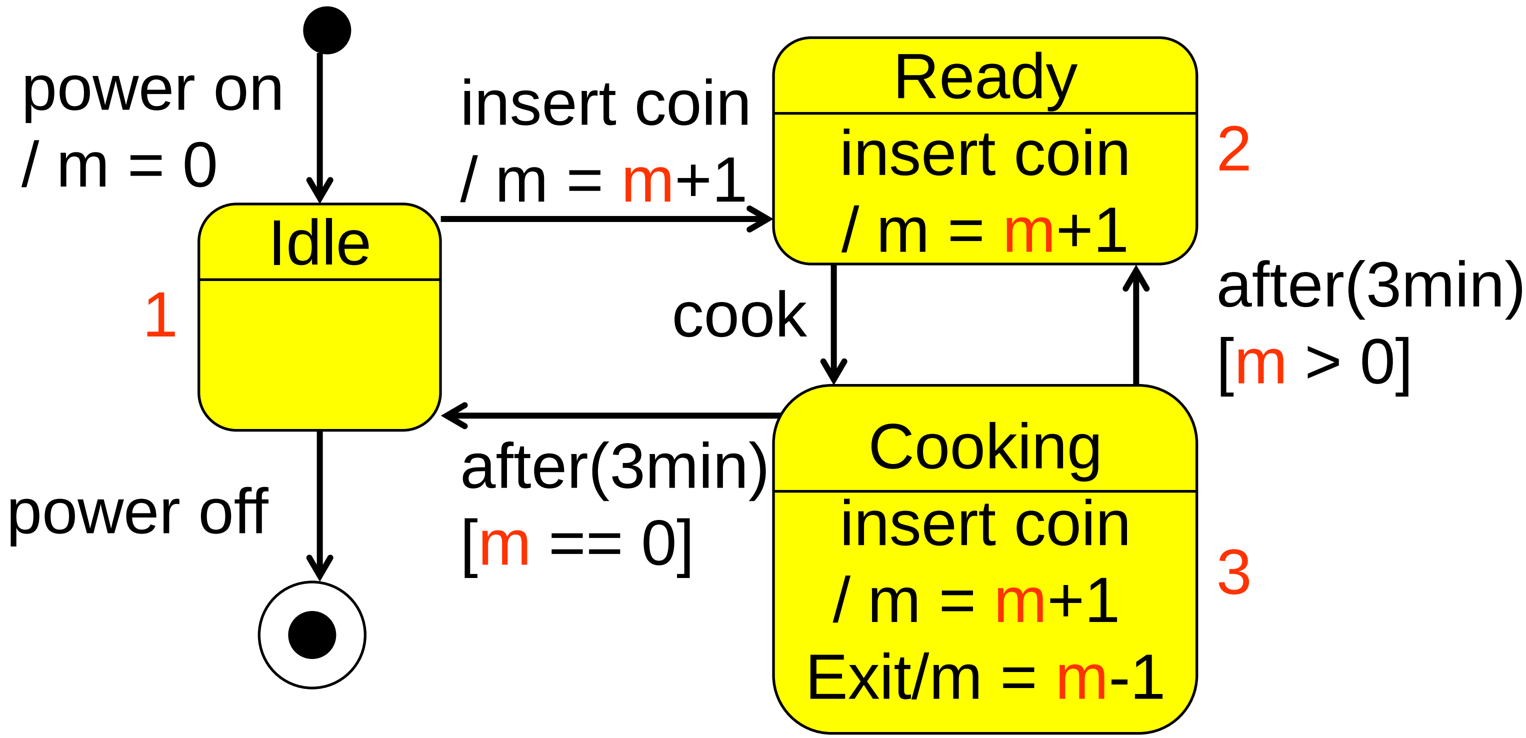
The set of test cases covers all the definitions in the diagram




1 → 2 →* 2 → 3 →* 3 → 2 → 3 → 2 → 3 → 1

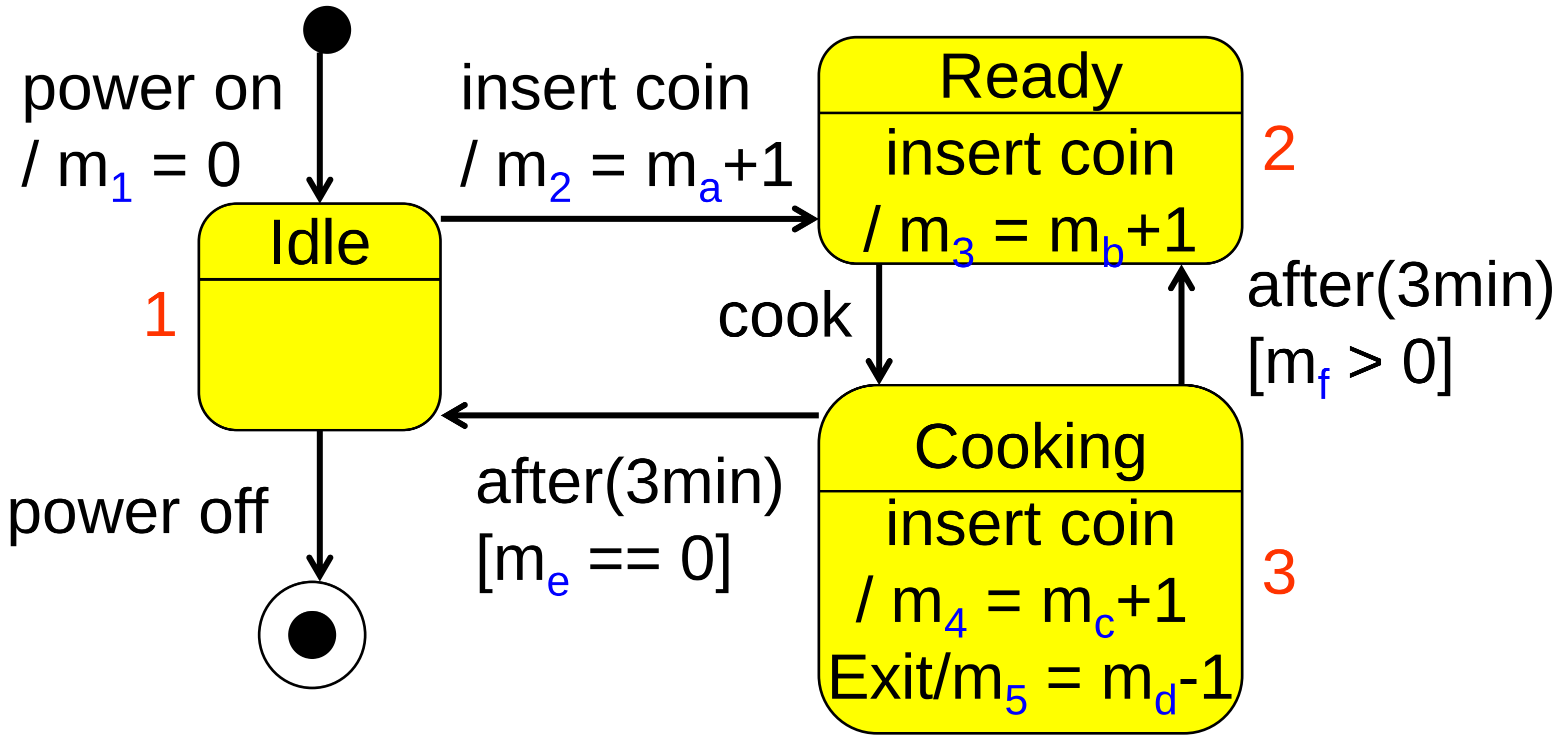
# All-Use Coverage

The set of test cases covers all the uses in the diagram




1 → 2 →* 2 → 3 →* 3 → 2 → 3 → 2 → 3 → 1

# All-Definition-Use Coverage



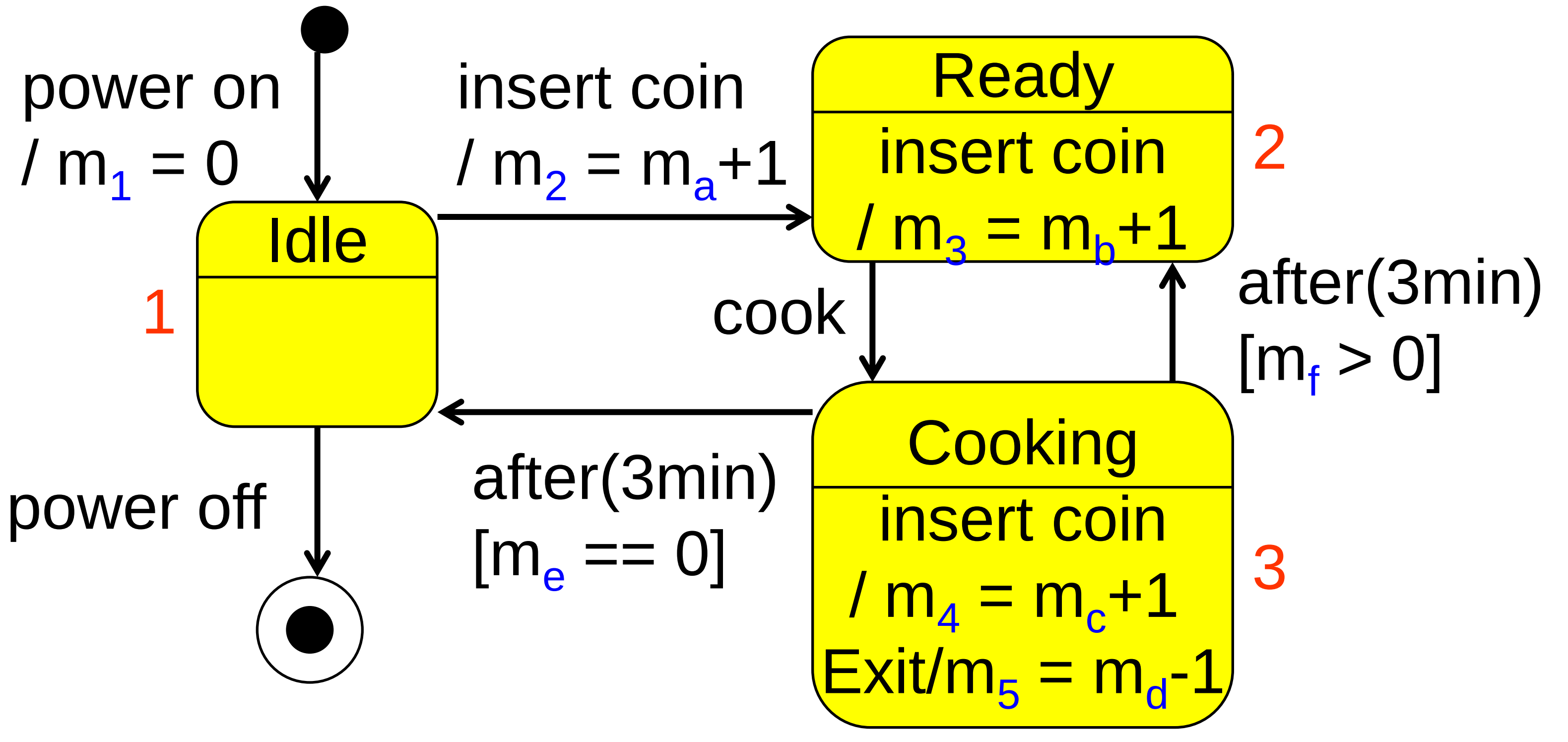

$(m_1, m_a)$, $(m_2, m_b)$, $(m_2, m_c)$, $(m_2, m_d)$,

$(m_3, m_b)$, $(m_3, m_c)$, $(m_3, m_d)$, $(m_4, m_c)$, $(m_4, m_d)$,

---

$(m_5, m_a)$, $(m_5, m_b)$, $(m_5, m_c)$, $(m_5, m_d)$, $(m_5, m_e)$, $(m_5, m_f)$.

# All-Definition-Use Coverage



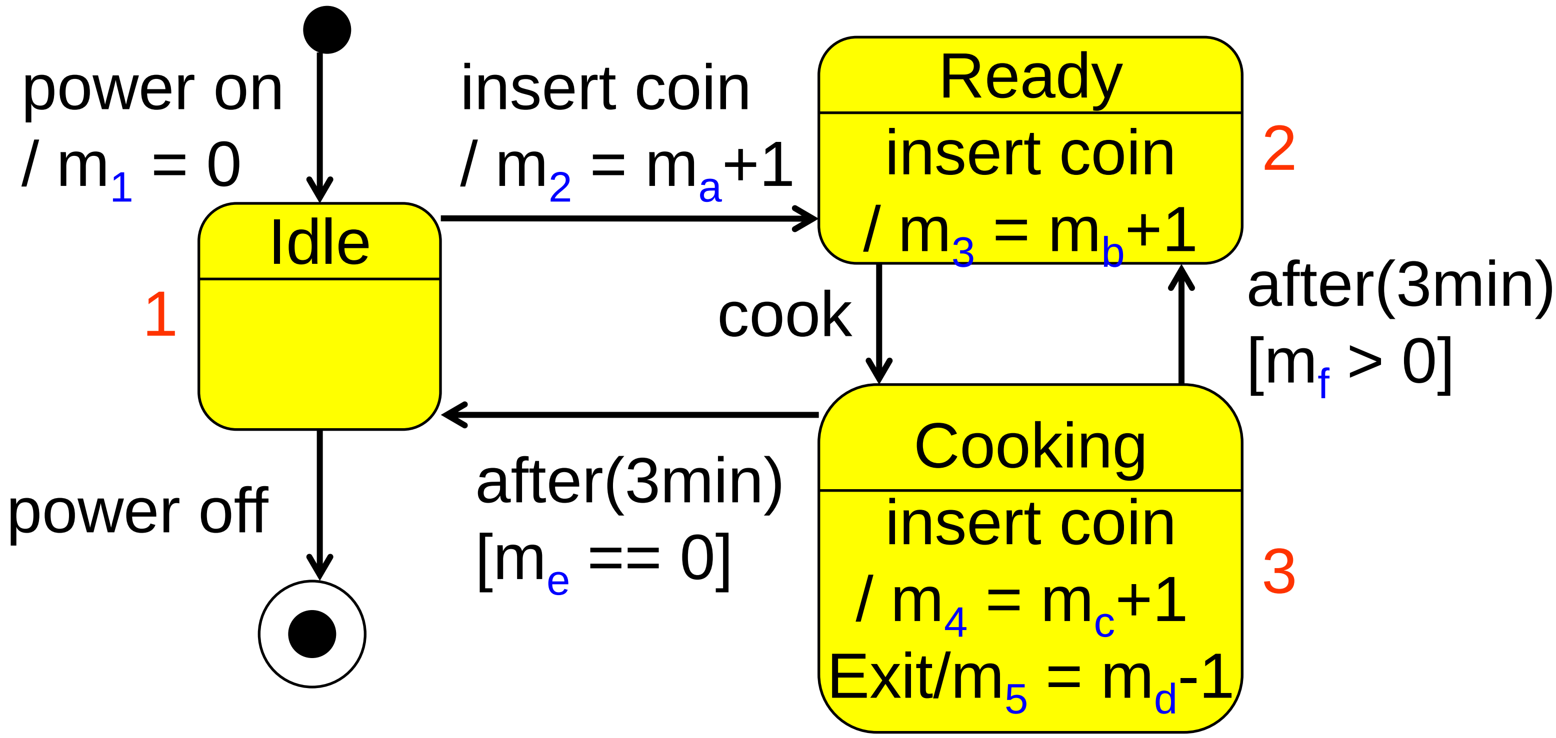

1 → 2 → 3 → 1 → 2 → 3 → 1

(1, a), (2, d), (5, e), (5, a)

1 → 2 →* 2 →* 2 → 3 → 2 → 3 → 2 → 3 → 1

(2, b), (3, b), (3, d), (5, f), (5, d)

# All-Definition-Use Coverage




1 → 2 → 3 →* 3 →* 3 → 2 → 3 → 1

(2, c), (4, c), (4, d)

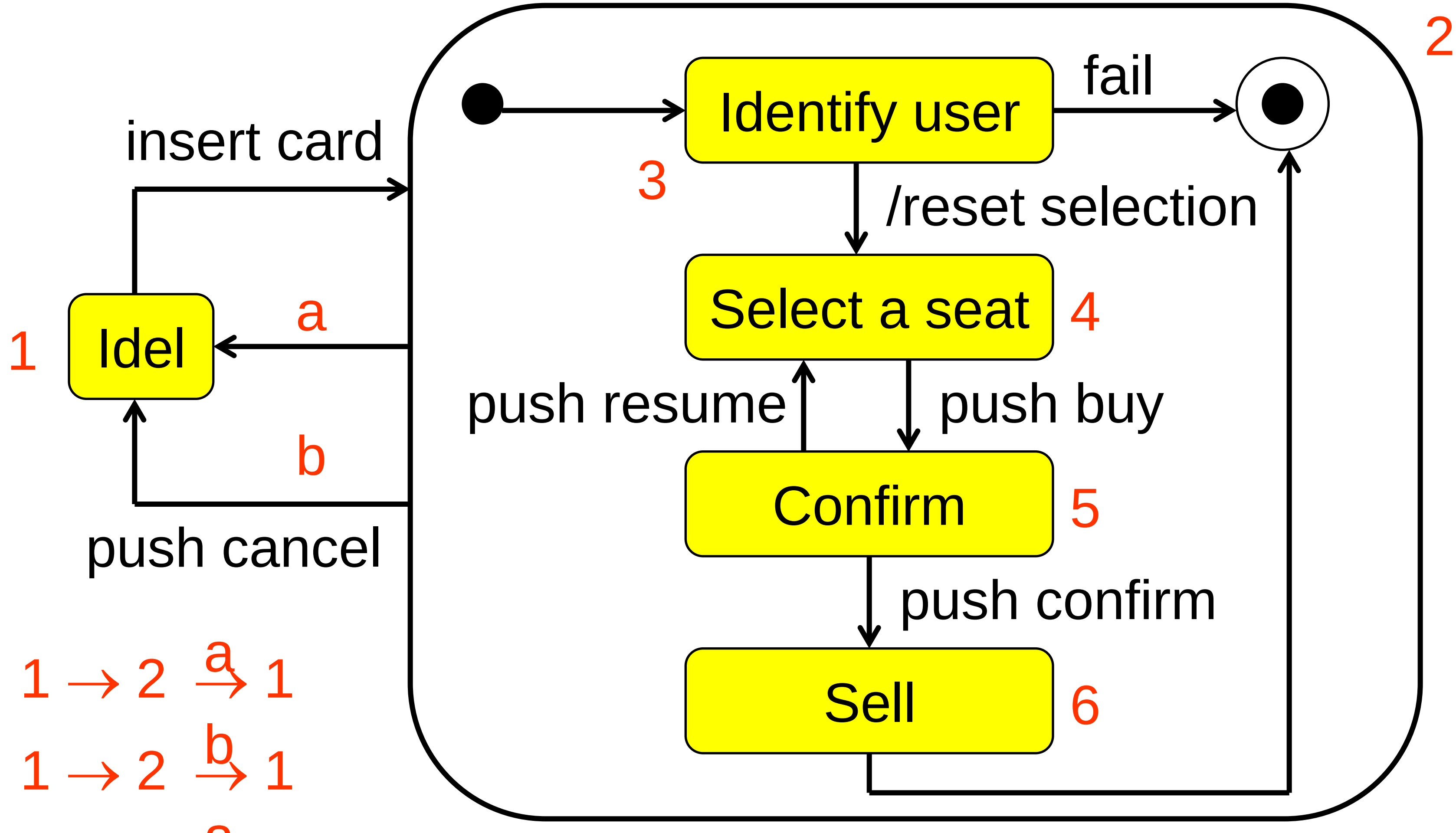
1 → 2 →* 2 → 3 →* 3 → 2 →* 2 → 3 → 2 → 3 → 2 → 3 → 1

(3, c), (5, b)

# Nonorthogonal States

- If a complete path contains a nonorthogonal state *s*, we can substitute each complete subpath within the state *s* for the state *s* in the complete path to generate a set of expanded complete paths.

# An Example




$1 \rightarrow 2 \xrightarrow{a} 1$

$1 \rightarrow 2 \xrightarrow{b} 1$

$1 \rightarrow 3 \xrightarrow{a} 1$

$1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 4 \rightarrow 5 \rightarrow 6 \xrightarrow{a} 1$

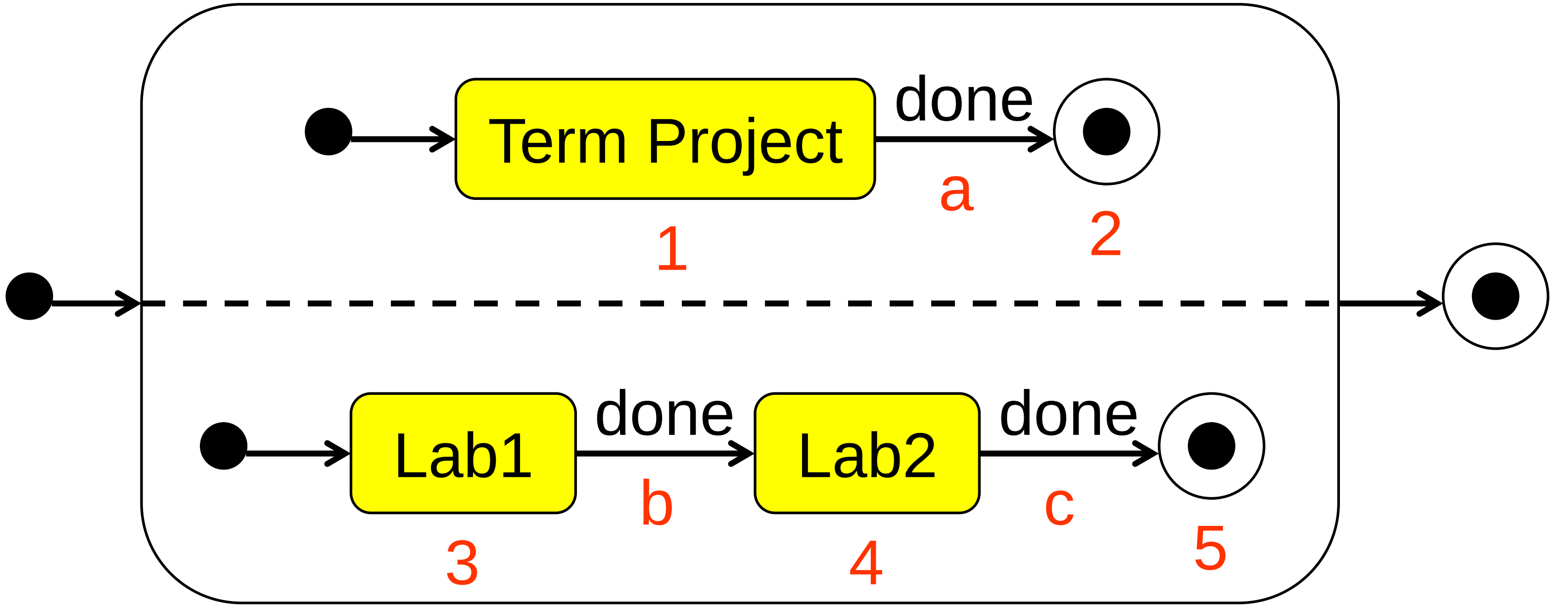
# Orthogonal States

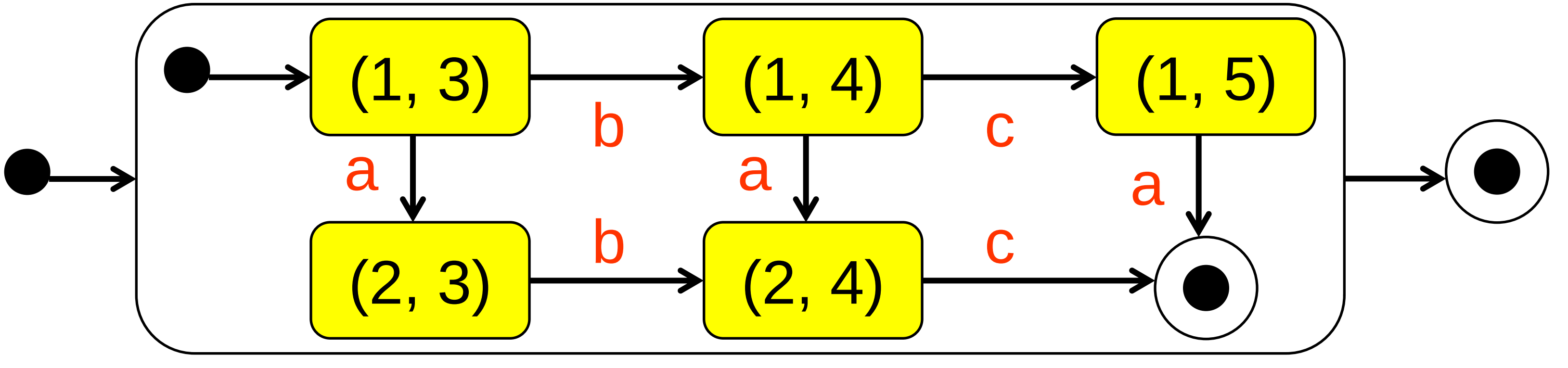
- If a complete path contains an orthogonal state *s*, we can also substitute each complete subpath within the state *s* for the state *s* in the complete path to generate a set of expanded complete paths.
- The concurrency in the orthogonal node makes the determination of complete subpaths complex.
- The orthogonal state is transformed into a nonthogonal state.

# Orthogonal States to Nonorthogonal States

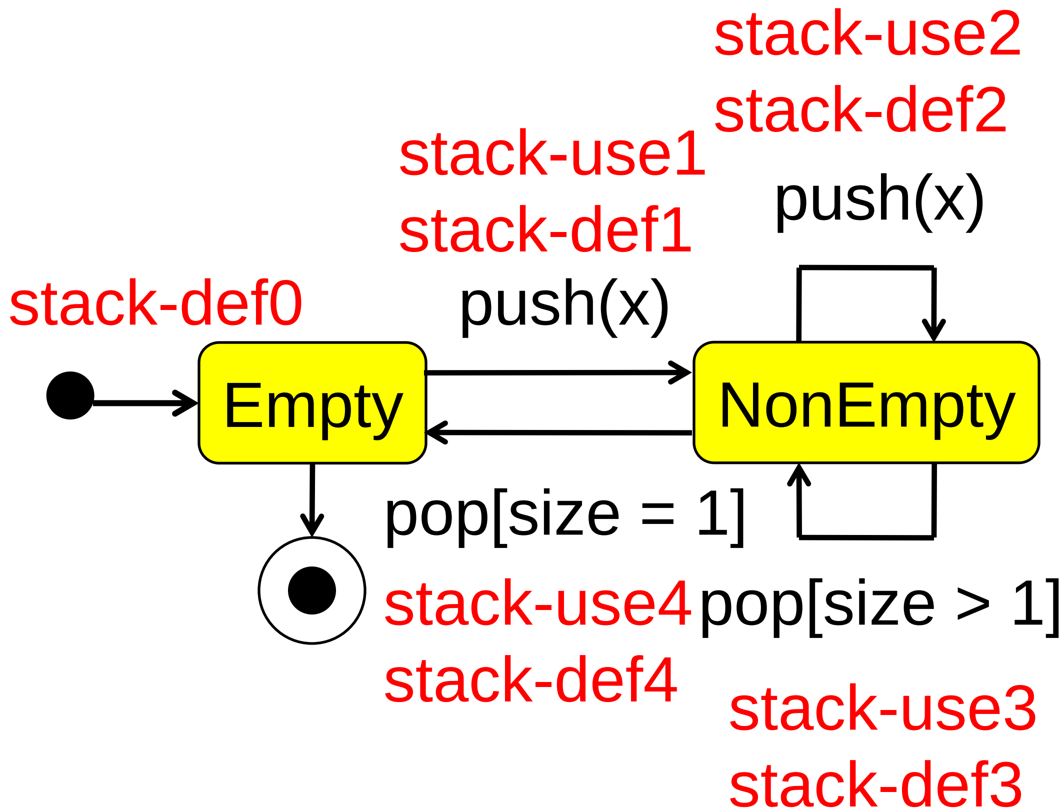
- Let the orthogonal state has *n* regions and the *i*th region has $m_i$ states.
- Each new state is an *n*-tuple $(x_1, \ldots, x_i, \ldots, x_n)$, where $x_i$ is an old state in the *i*th region.
- There is a transition from $(x_1, \ldots, x_{i1}, \ldots, x_n)$ to $(x_1, \ldots, x_{i2}, \ldots, x_n)$ if there is a transition from $x_{i1}$ to $x_{i2}$ in the *i*th region.

# An Example


Term Project
done
a
1
2
Lab1
done
b
Lab2
done
c
3
4
5


(1, 3)
b
(1, 4)
c
(1, 5)
a
a
a
(2, 3)
b
(2, 4)
c

# An Example




- ~~(def0, use1)~~
- ~~(def1, use2)~~
- ~~(def1, use4)~~
- ~~(def2, use2)~~
- ~~(def2, use3)~~
- ~~(def3, use2)~~
- ~~(def3, use3)~~
- ~~(def3, use4)~~
- ~~(def4, use1)~~

path1: $E \rightarrow N \rightarrow E \rightarrow N \rightarrow E$

path2: $E \rightarrow N \rightarrow_h N \rightarrow_h N \rightarrow_p N \rightarrow_h N \rightarrow_p N \rightarrow_p N \rightarrow E$