LECTURE 18

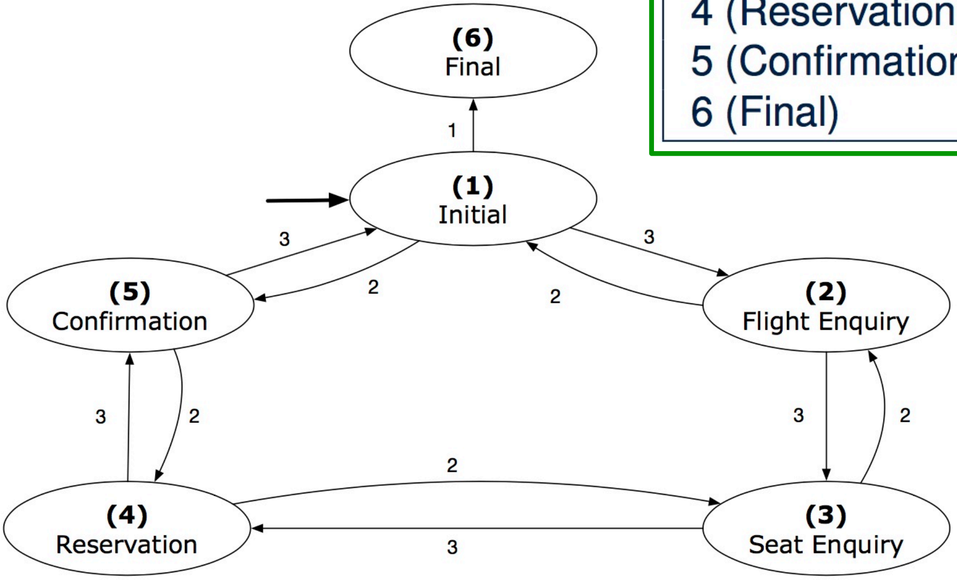
TUESDAY NOVEMBER 12

Finite State Machine (FSM)

State Transition Table

Src State \ Choice	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

State Transition Diagram



Design of a Reservation System: Second Attempt (1)

```
transition (src: INTEGER; choice: INTEGER): INTEGER
    -- Return state by taking transition 'choice' from 'src' state.
  require valid_source_state: 1 ≤ src ≤ 6
          valid_choice: 1 ≤ choice ≤ 3
  ensure valid_target_state: 1 ≤ Result ≤ 6
```

Examples:

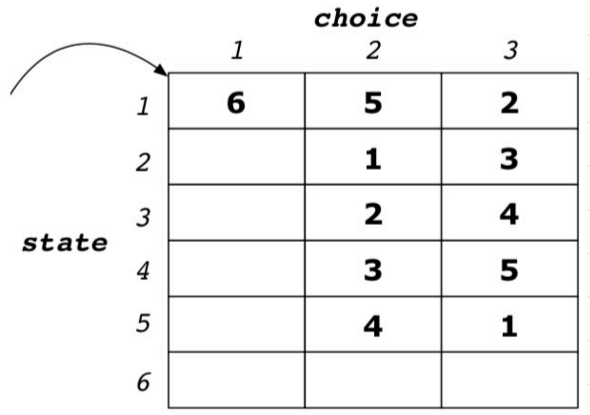
transition(3, 2)

transition(3, 3)

State Transition Table

Choice Src State	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

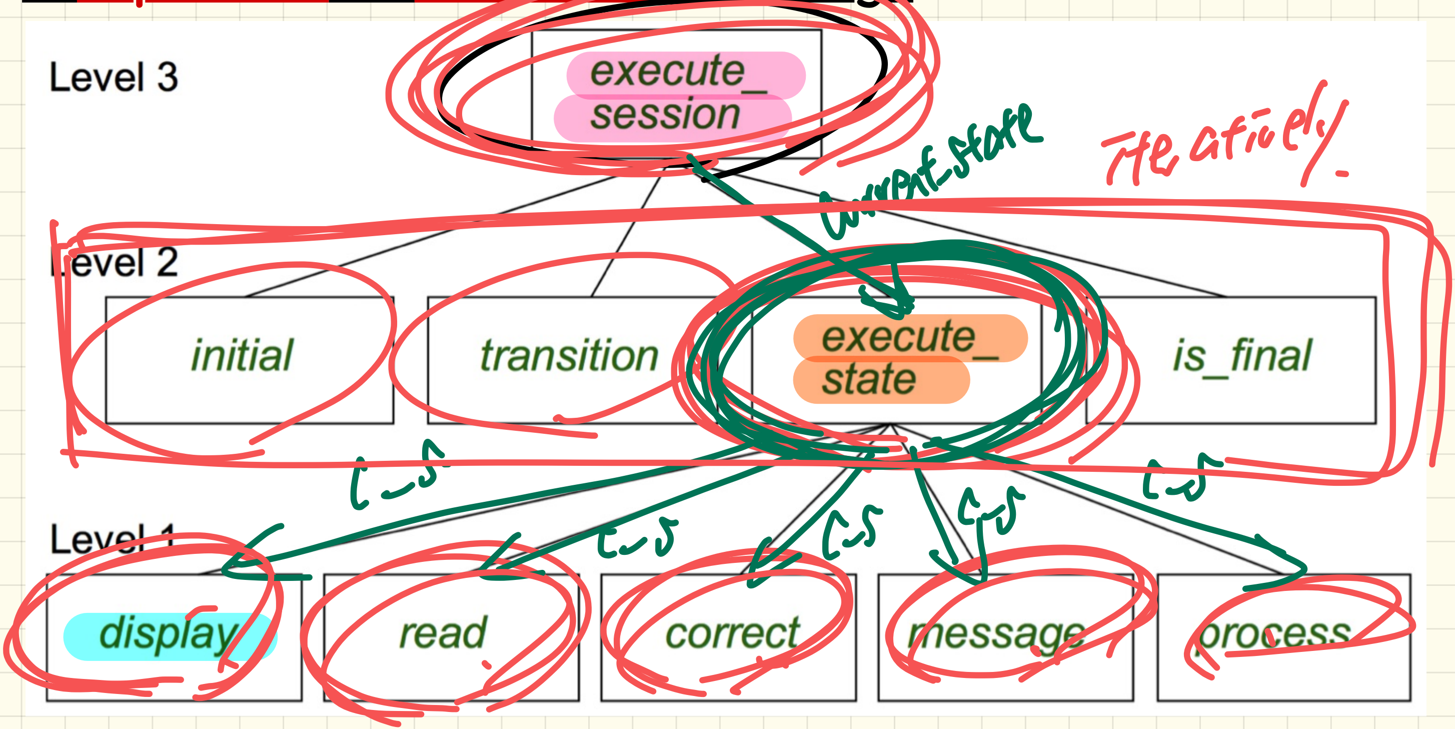
2D Array Implementation



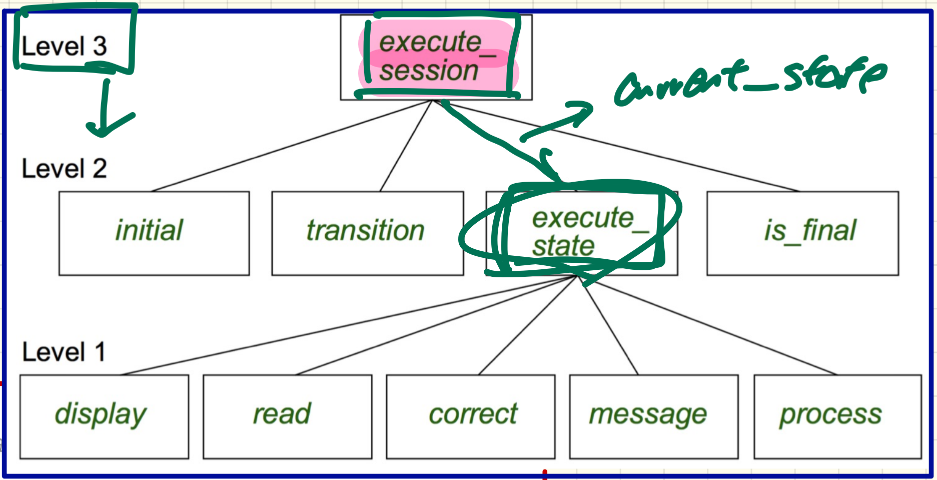
state \ choice	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

Design of a Reservation System: Second Attempt (2)

A Top-Down & Hierarchical Design



Design of a Reservation System: Second Attempt (3)

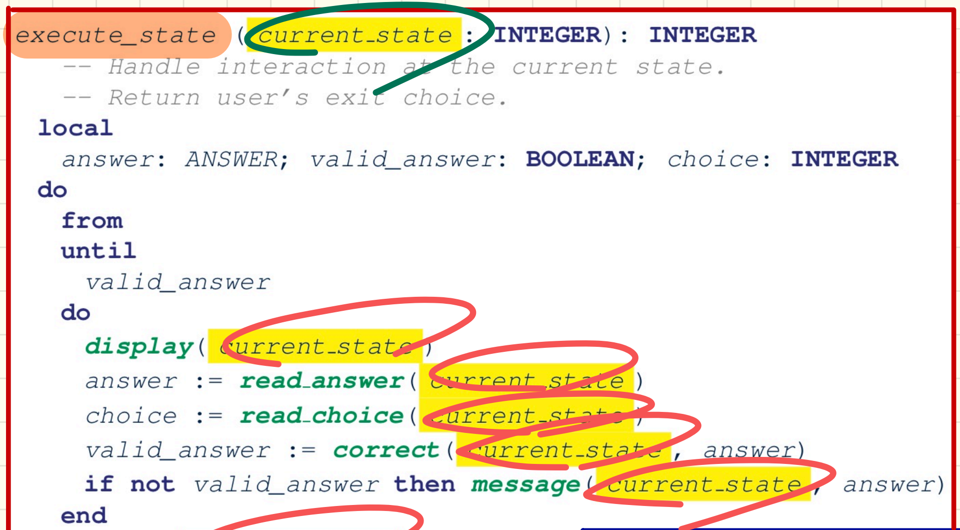


routine

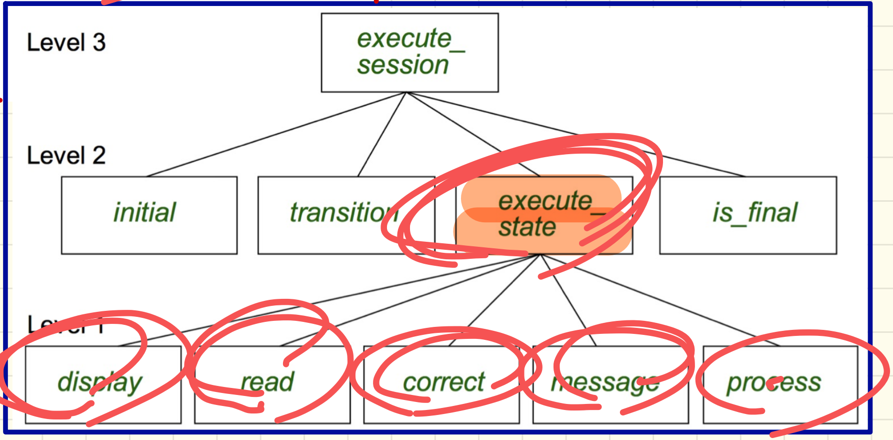
```
execute_session
  -- Execute a full intera
  local
    current_state, choice: INTEGER
  do
    from
      current_state := initial
    until
      is_final (current_state)
    do
      choice := execute_state ( current_state )
      current_state := transition (current_state, choice)
    end
  end
```

helper routine of execute_session

Design of a Reservation System: Second Attempt (4)

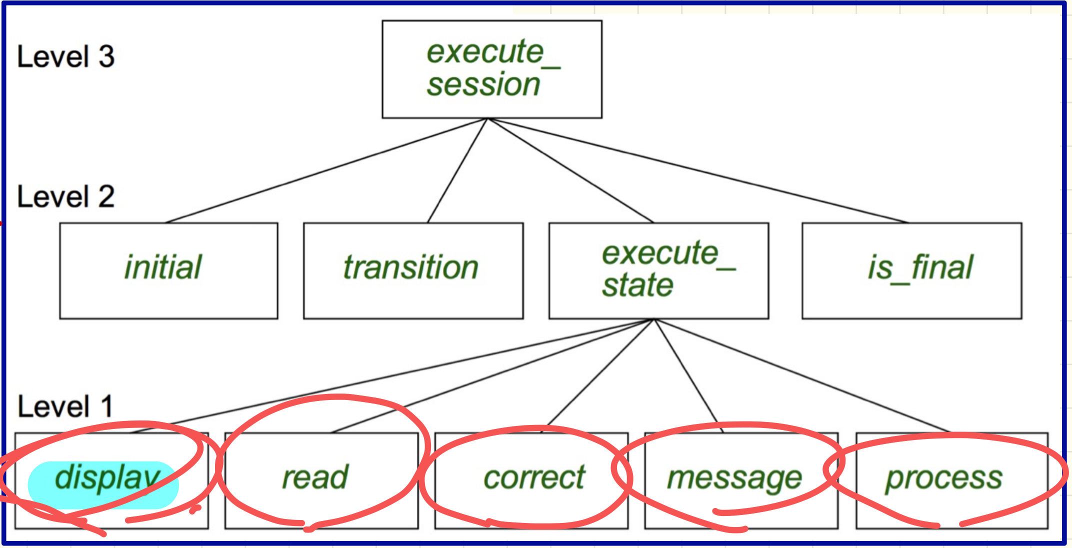


```
execute_state ( current_state : INTEGER) : INTEGER
    -- Handle interaction at the current state.
    -- Return user's exit choice.
  local
    answer: ANSWER; valid_answer: BOOLEAN; choice: INTEGER
  do
    from
    until
      valid_answer
    do
      display( current_state )
      answer := read_answer( current_state )
      choice := read_choice( current_state )
      valid_answer := correct( current_state , answer)
      if not valid_answer then message( current_state , answer)
    end
    process( current_state , answer)
    Result := choice
  end
```



Design of a Reservation System: Second Attempt (5)

```
display(current_state: INTEGER)
  require
    valid_state: 1 ≤ current_state ≤ 6
  do
    if current_state = 1 then
      -- Display Initial Panel
    elseif current_state = 2 then
      -- Display Flight Enquiry Panel
  elseif c_s = 7
    ...
    else
      -- Display
    end
  end
```



Moving from Top-Down Design to OO Design

Object-Oriented

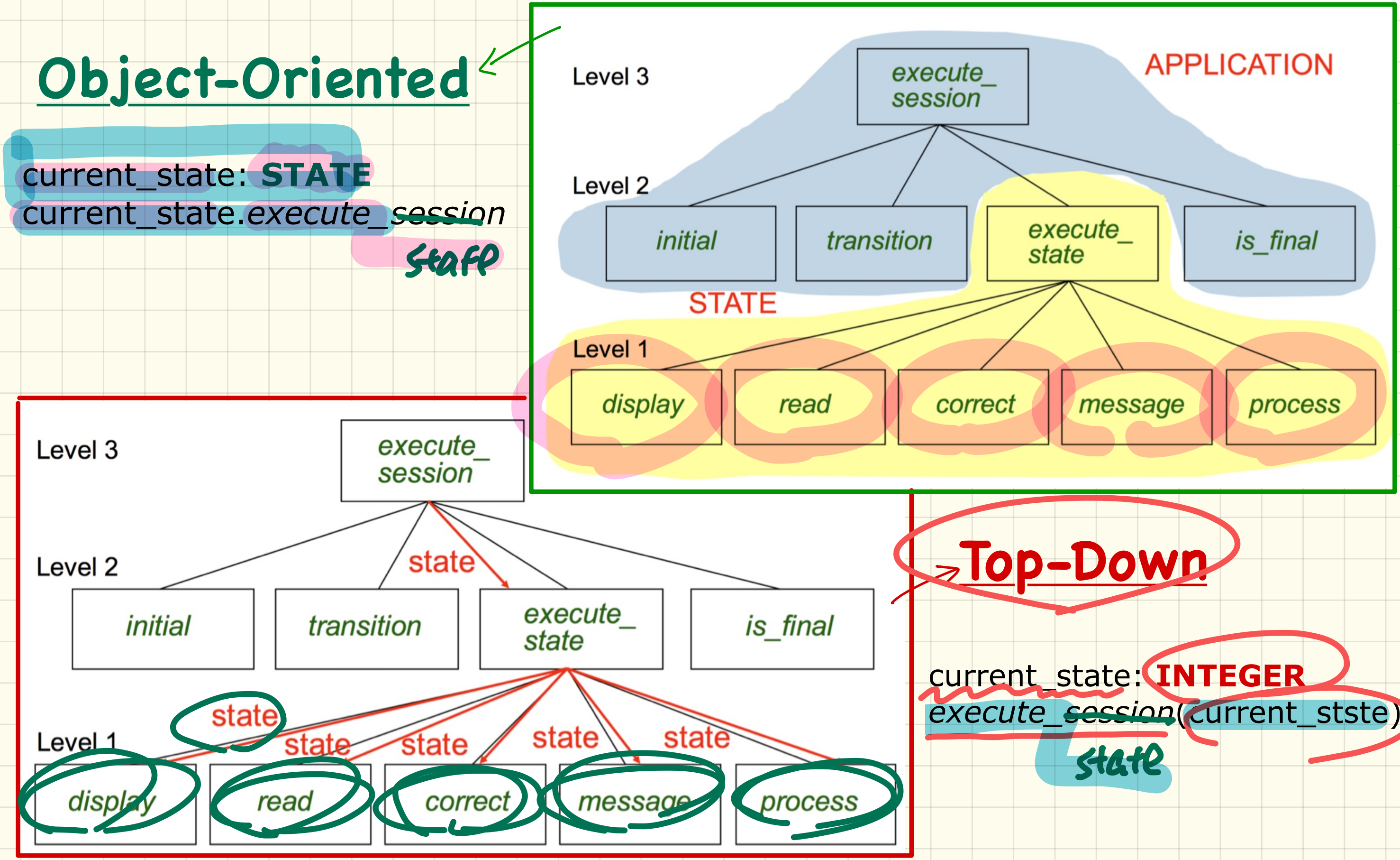
```
current_state: STATE
current_state.execute_~~session~~
```

state

Top-Down

```
current_state: INTEGER
execute_~~session~~(current_stste)
```

state



Non-OO solution

current_state : INT

1 ~ 6

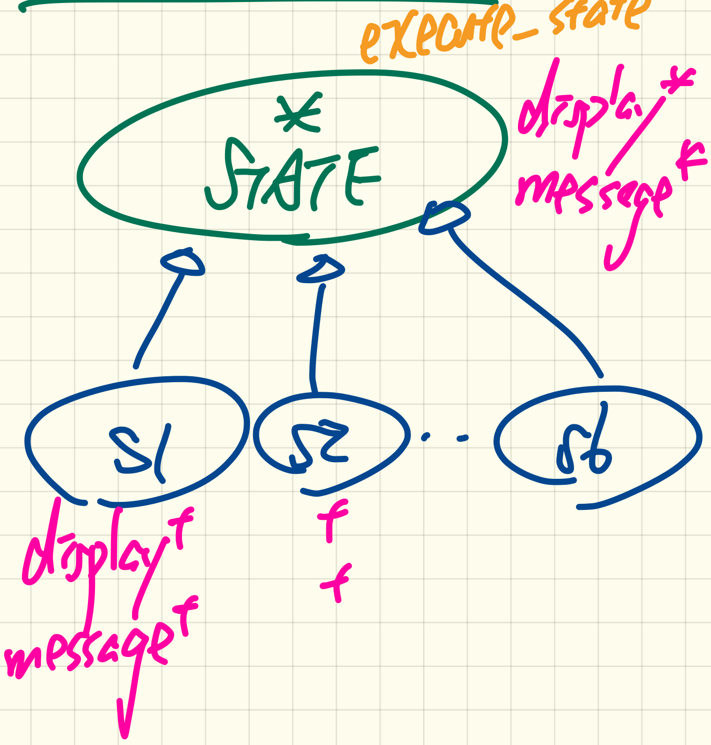
s1 ~ s6

execute_state (cs : INT)

display (cs : INT)

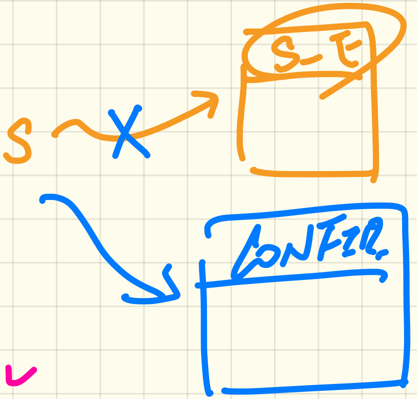
message (cs : INT)

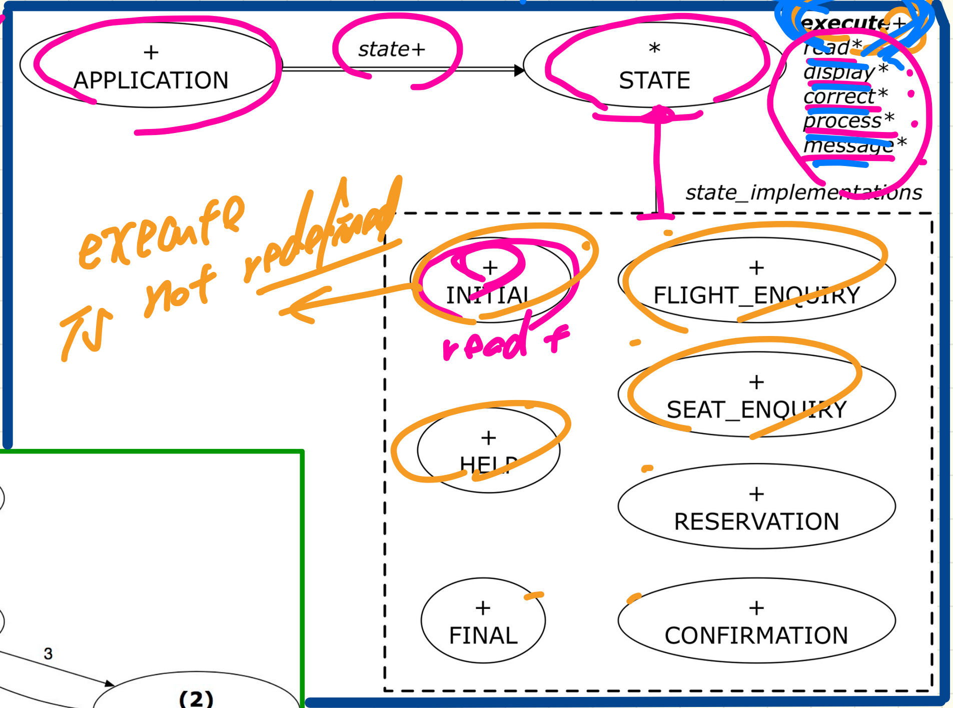
OO solution



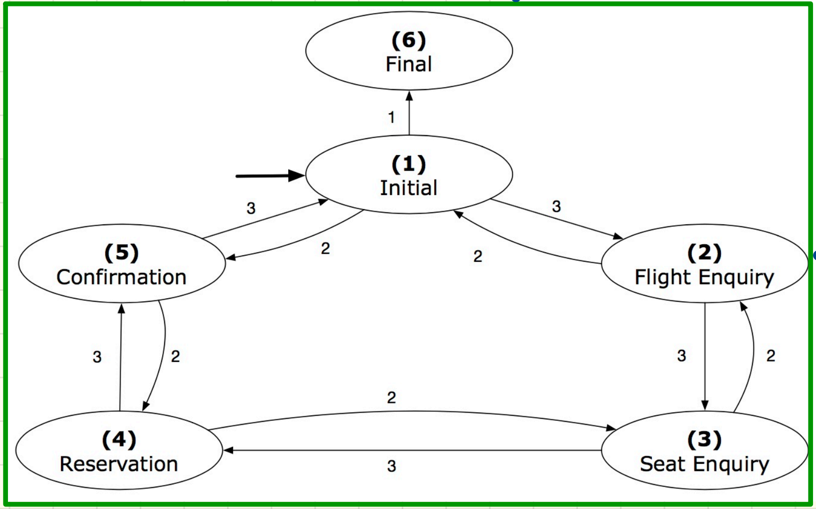
State Pattern: Architecture

TEMPLATE ← execute_state





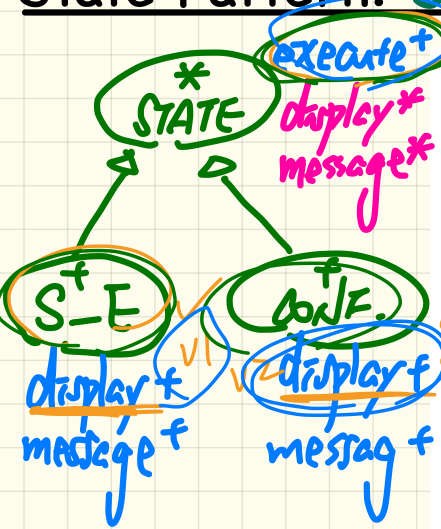
execute is not redefined

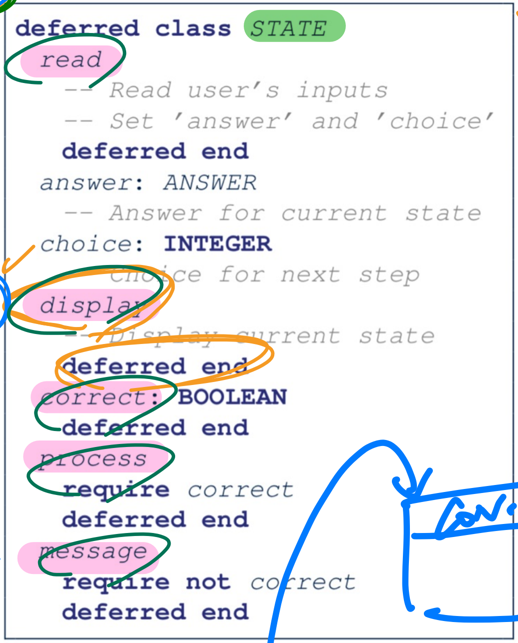


```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

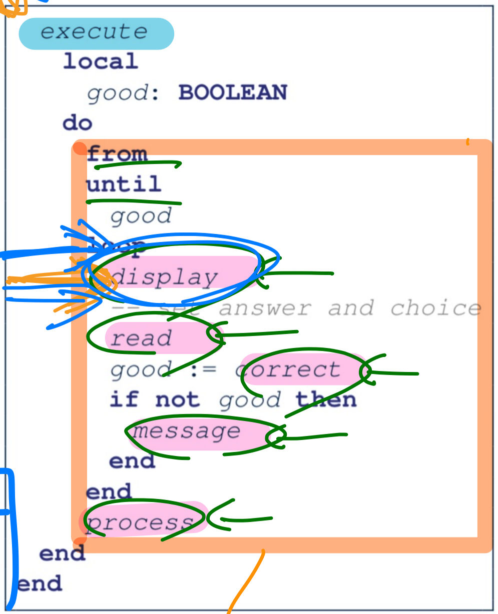
→ version from STATE.

State Pattern: State Module





```
deferred class STATE
  read
    -- Read user's inputs
    -- Set 'answer' and 'choice'
    deferred end
  answer: ANSWER
    -- Answer for current state
  choice: INTEGER
    -- Choice for next step
  display
    -- Display current state
    deferred end
  correct: BOOLEAN
    deferred end
  process
    require correct
    deferred end
  message
    require not correct
    deferred end
```



```
execute
  local
    good: BOOLEAN
  do
    from
    until
      good
    loop
      display
      -- Set answer and choice
      read
      good := correct
      if not good then
        message
      end
    end
    process
  end
end
```

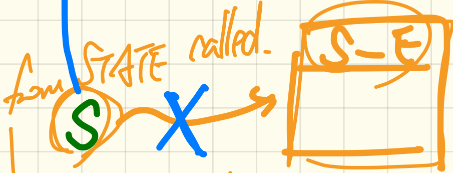
→ from STATE is called. execute display from Con.

Con.

```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

→ DT: S_E. execute

→ DT: Con. execute



from STATE called.

TEMPLATE

execute display from S_E.

s : STATE

Create {STATE} s.make ✗

deferred.

n_S : INTEGER

current_state : STATE

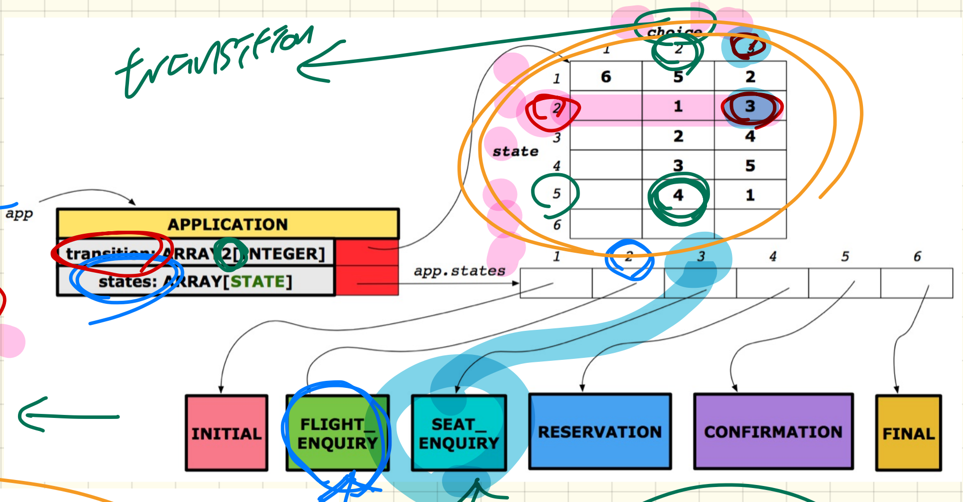
c_S := states[2]

n_S := transition(2, 3)

3

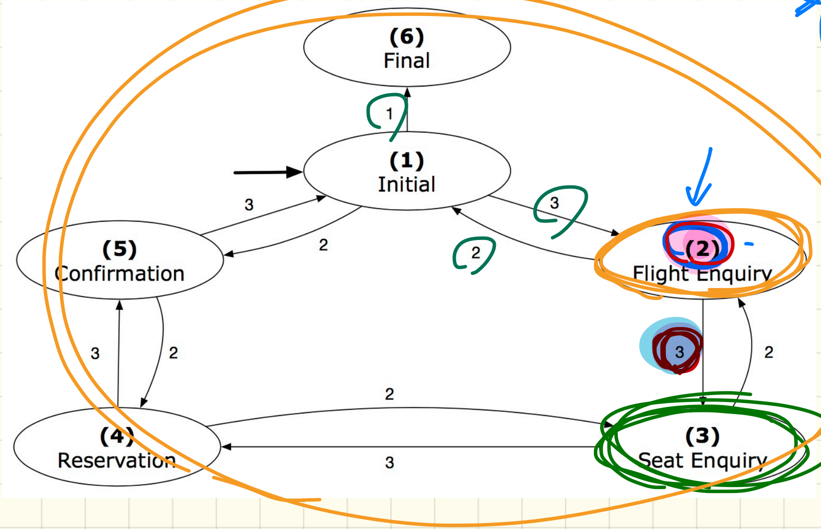
c_S := states[n_S]

transition

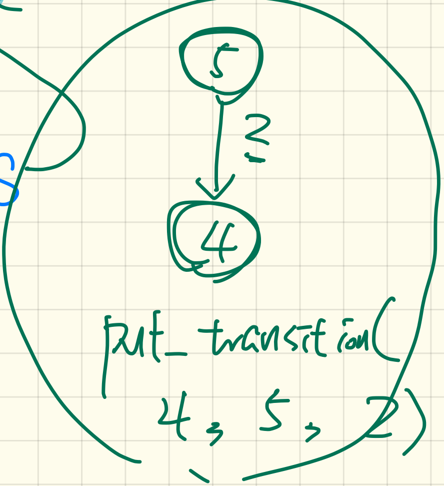


state \ choice	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

1	2	3	4	5	6



c_S



put_transition(4, 5, 2)

State Pattern: Application Module

```
class APPLICATION create make
feature {NONE} -- Implementation of Transition Graph
  transition: ARRAY2[INTEGER]
    -- State transitions: transition[state, choice]
  states: ARRAY[STATE]
    -- State for each index, constrained by size of 'transition'
feature
  initial: INTEGER
  number_of_states: INTEGER
  number_of_choices: INTEGER
  make(n, m: INTEGER)
    do number_of_states := n
       number_of_choices := m
       create transition.make_filled(0, n, m)
       create states.make_empty
    end
feature
  put_state(s: STATE; index: INTEGER)
    require 1 ≤ index ≤ number_of_states
    do states.force(s, index) end
  choose_initial(index: INTEGER)
    require 1 ≤ index ≤ number_of_states
    do initial := index end
  put_transition(tar, src, choice: INTEGER)
    require
      1 ≤ src ≤ number_of_states
      1 ≤ tar ≤ number_of_states
      1 ≤ choice ≤ number_of_choices
    do
      transition.put(tar, src, choice)
    end
invariant
  transition.height = number_of_states
  transition.width = number_of_choices
end
```

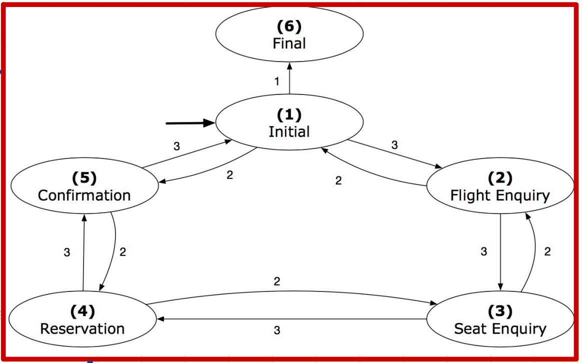
State Pattern: Test

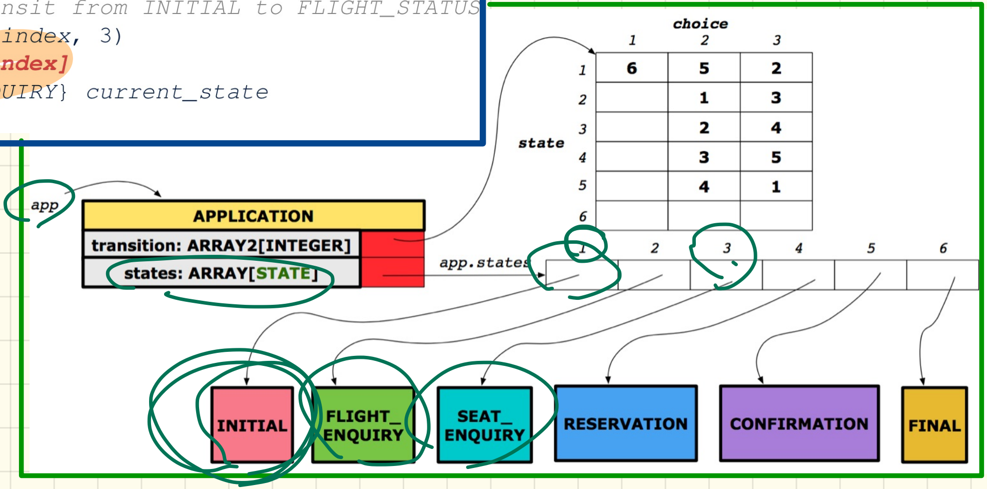
```
test_application: BOOLEAN
  local
    app: APPLICATION ; current_state: STATE ; index: INTEGER
  do
    create app.make (6, 3)
    app.put_state (create {INITIAL}.make, 1)
    -- Similarly for other 5 states.
    app.choose_initial (1)
    -- Transit to FINAL given current state INITIAL and choice
    app.put_transition (6, 1, 1)
    -- Similarly for other 10 transitions.

    index := app.initial
    current_state := app.states [index]
    Result := attached {INITIAL} current_state
    check Result end
    -- Say user's choice is 3: transit from INITIAL to FLIGHT_STATUS
    index := app.transition.item (index, 3)
    current_state := app.states [index]
    Result := attached {FLIGHT_ENQUIRY} current_state
end
```

# of states

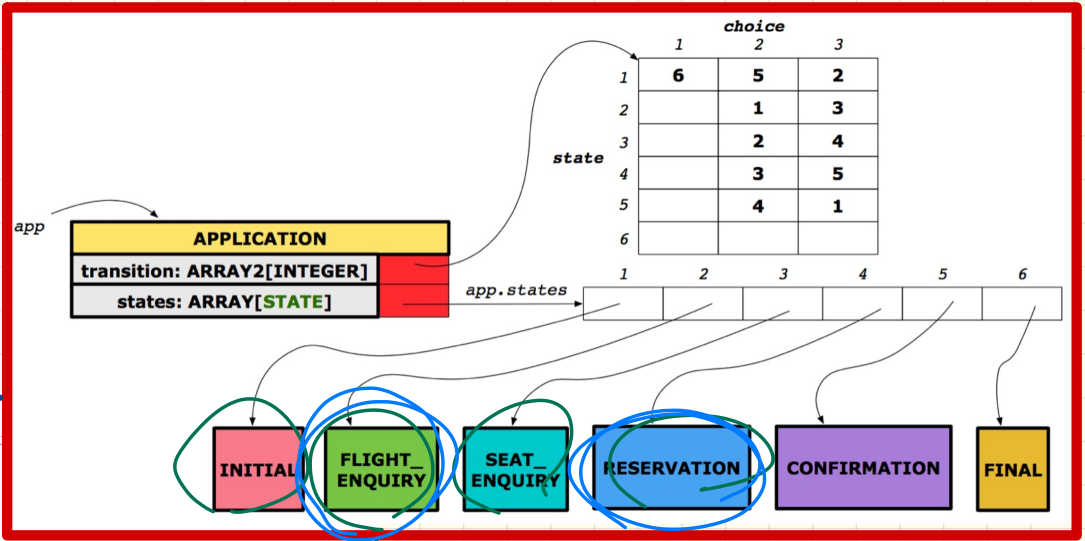
# of transitions





		choice	
state	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

State Pattern: Interactive Session



		choice		
		1	2	3
state	1	6	5	2
	2		1	3
	3		2	4
	4		3	5
	5		4	1
	6			

```
class APPLICATION
feature {NONE} -- Implementati
  transition: ARRAY2[INTEGER]
  states: ARRAY[STATE]
feature
  execute_session
    local
      current_state: STATE
      index: INTEGER
    do
      from
        index := initial
      until
        is_final (index)
      loop
        current_state := states[index] -- polymorphism
        current_state.execute -- dynamic binding
        index := transition.item (index, current_state.choice)
      end
    end
end
```

indexing into the polymorphic states array.

ST: STATE

STATE.

current_state :=

ST: STATE states[index]

TEMPLATE

display

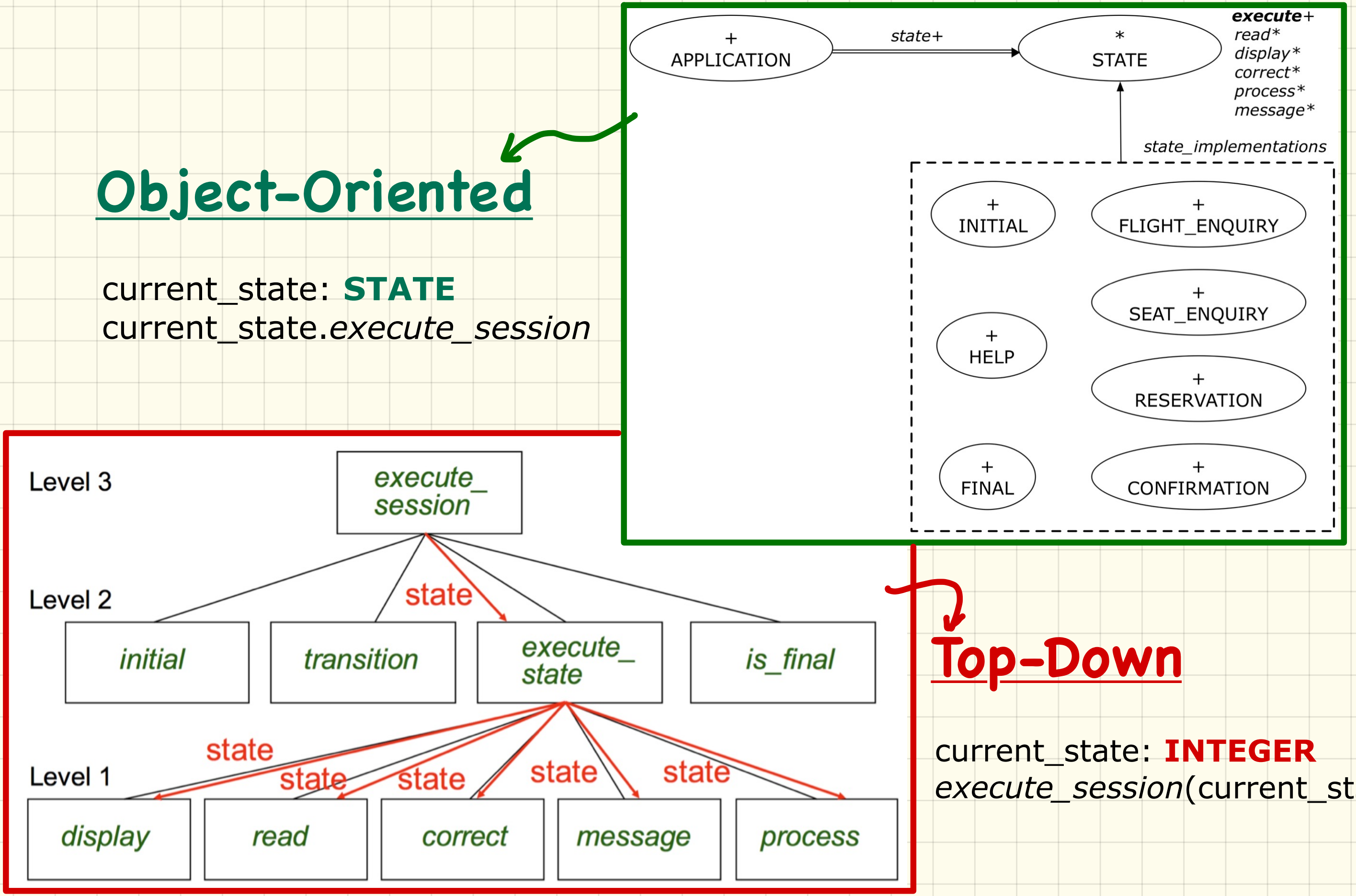
message

ST: STATE

Interactive System: Top-Down Design vs. OO Design

Object-Oriented

current_state: **STATE**
current_state.*execute_session*



Top-Down

current_state: **INTEGER**
execute_session(current_stste)