Thursday Oct. 25

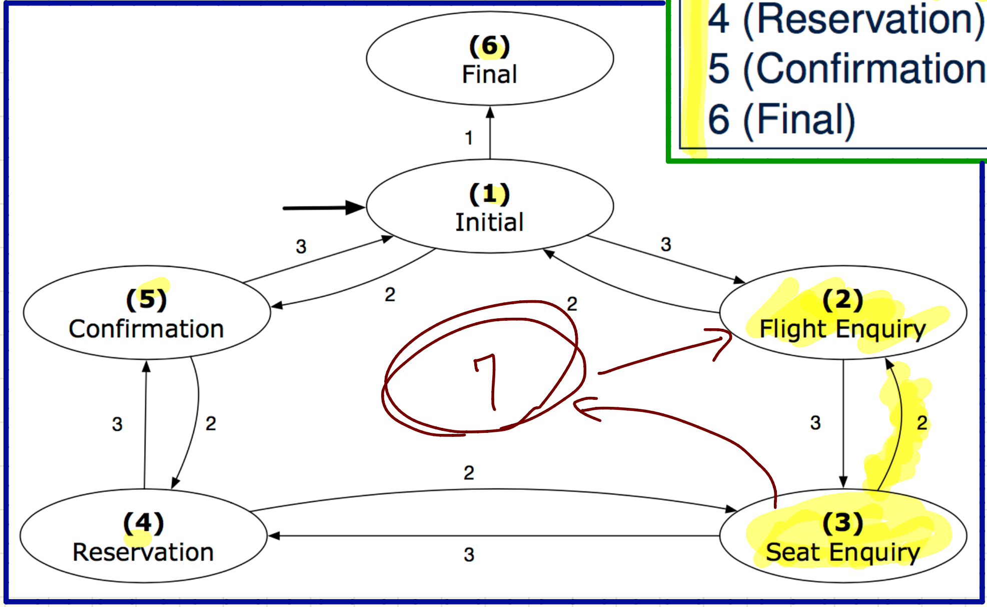
Lecture 13

State Transition Diagram (FSM)

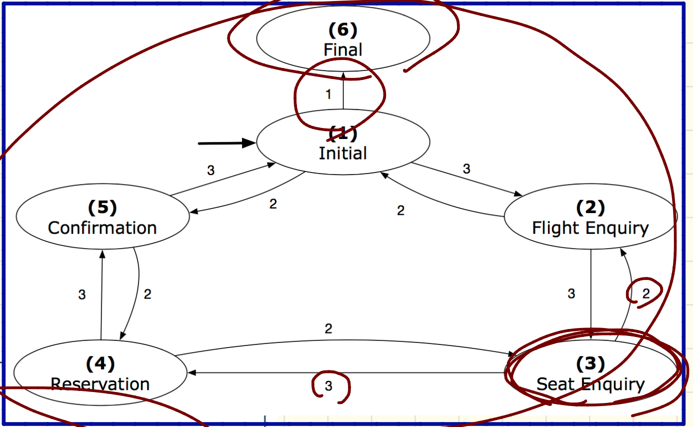
Transition Table

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)	–	–	–

Finite State Machine



Design of a Reservation System: First Attempt



```
1_Initial_panel:
   -- Actions for Label 1.
2_Flight_Enquiry_panel:
   -- Actions for Label 2.
3_Seat_Enquiry_panel:
   -- Actions for Label 3.
4_Reservation_panel:
   -- Actions for Label 4.
5_Confirmation_panel:
   -- Actions for Label 5.
6_Final_panel:
   -- Actions for Label 6.
```

```
3_Seat_Enquiry_panel:
from
  Display Seat Enquiry Panel
until
  not (wrong answer or wrong choice)
do
  Read user's answer for current panel
  Read user's choice C for next step
  if wrong answer or wrong choice then
    Output error messages
  end
end
Process user's answer
case C in
  2: goto 2_Flight_Enquiry_panel
  3: goto 4_Reservation_panel
end
```

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
```

e.g. transition (3, 2) 2

→ transition (3, 3)

$$\begin{tikzcd} 3 \arrow[r, "2"] & 2 \end{tikzcd}$$

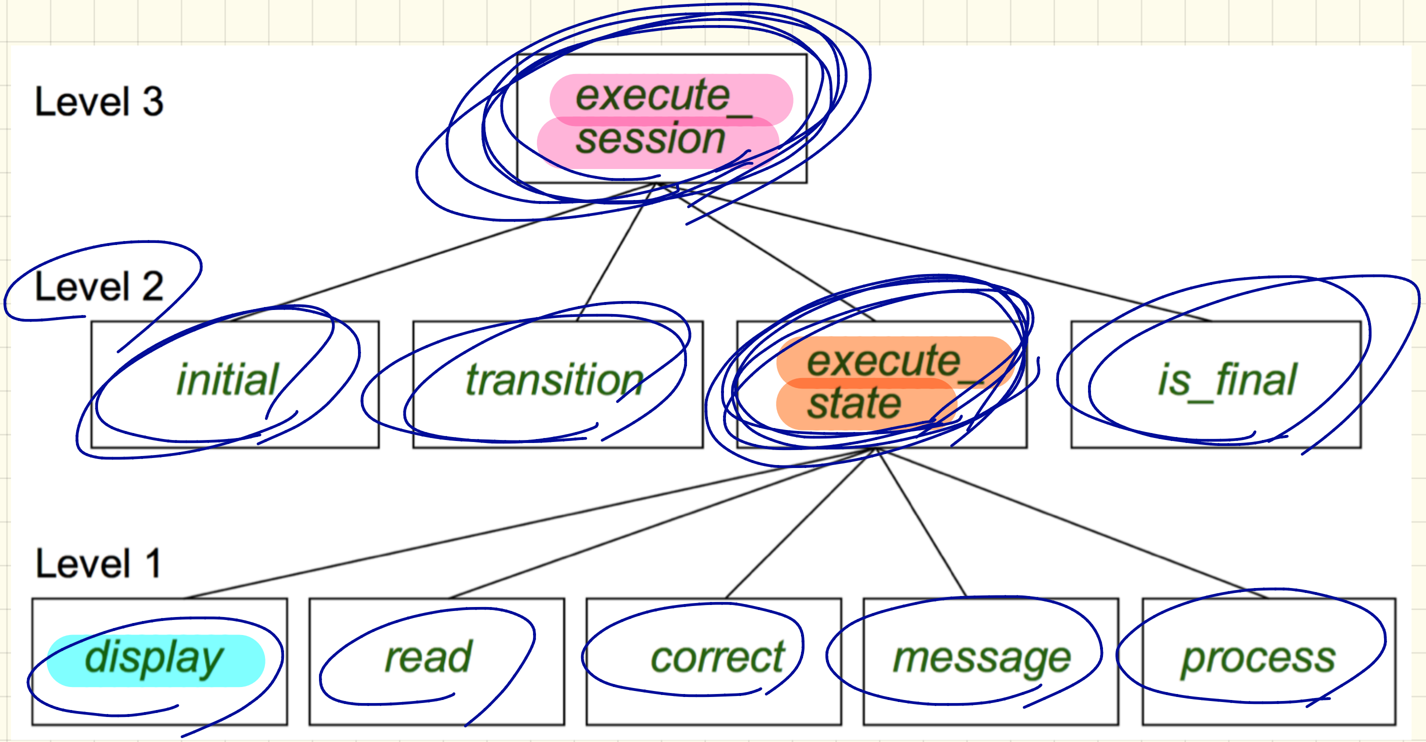
Transition Table

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)	–	–	–

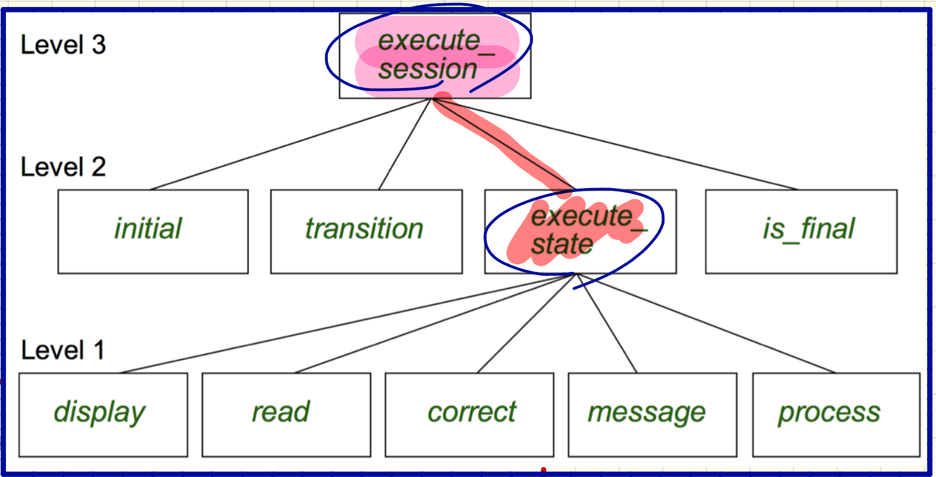
2D-Array Implementation

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: a Top-Down Design



Design of a Reservation System: Second Attempt (2)



```
execute_session
    -- Execute a full intera
  local    1≤  ≤6
    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
```

Design of a Reservation System: Second Attempt (2)

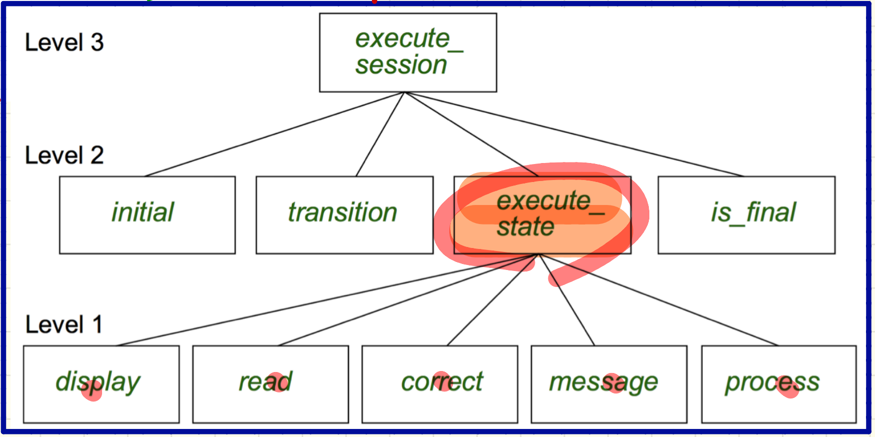
```
execute_state ( current_state : INTEGER) : INTEGER
  -- Handle interaction at the current state.
  -- Return user's exit choice.
local
  answer: ANSWER; valid_answer: BOOLEAN; choice: INTEGER
do
  from
  until
    valid_answer
  do
    display(current_state)
    answer := read_answer(current_state)
    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
```

display (current_state)

INT

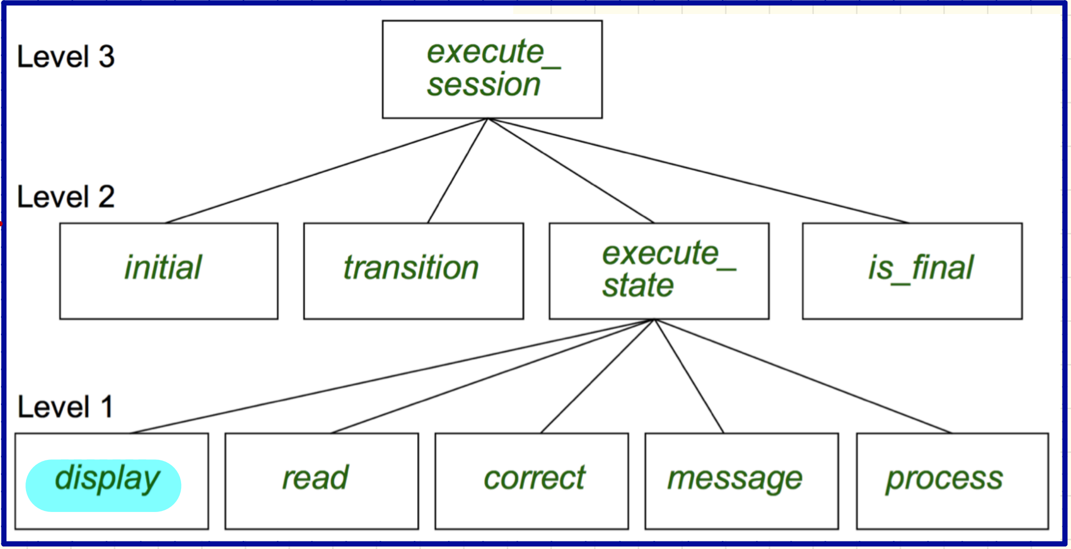
current_state display

STATE



Design of a Reservation System: Second Attempt (3)

```
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
    ...
    else
      -- Display
    end
  end
```



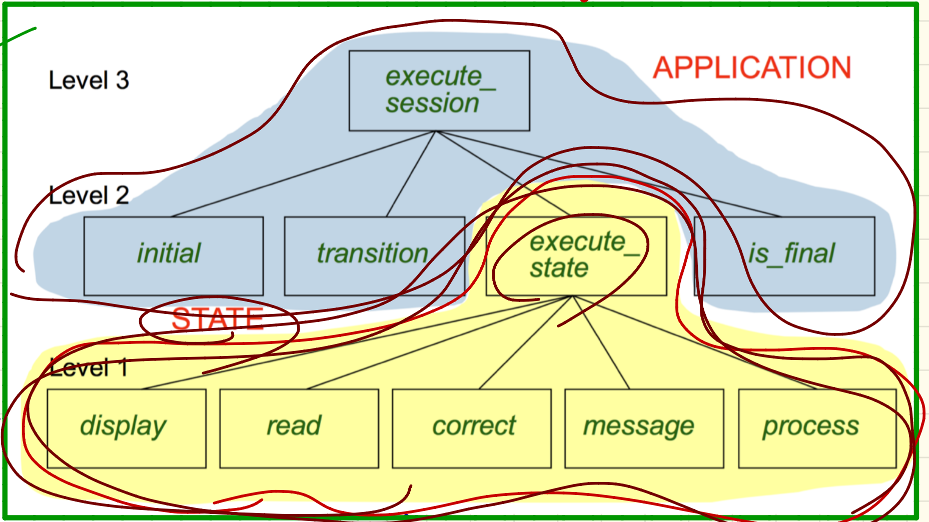
Moving from Hierarchical Design to OO Design

good

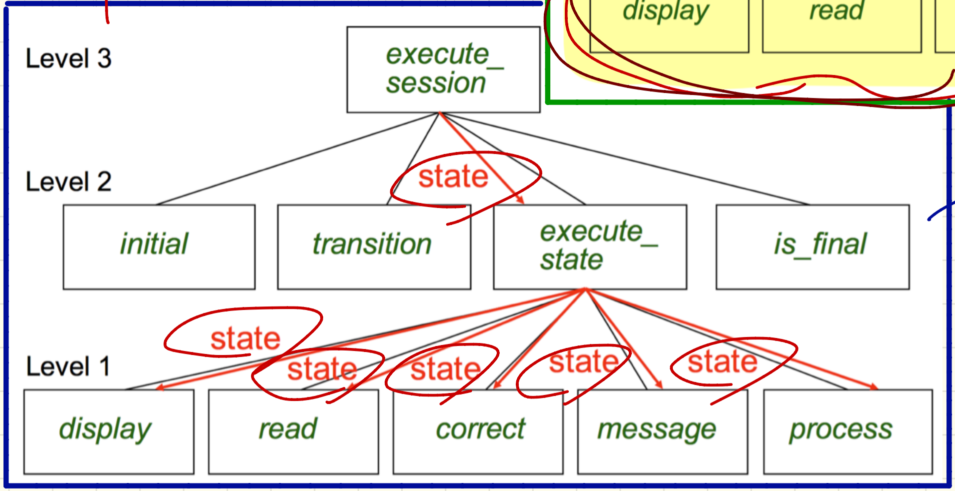
OO

current_state : STATE

current_state . execute_session



poor



HIERARCHICAL

current_state : $\boxed{\text{INTEGER}}$

execute_session (current_state)

STATE PATTERN Architecture

BON

deferred class STATE

```
execute
  do
    display
  end
```

① S_E ② Confirm.

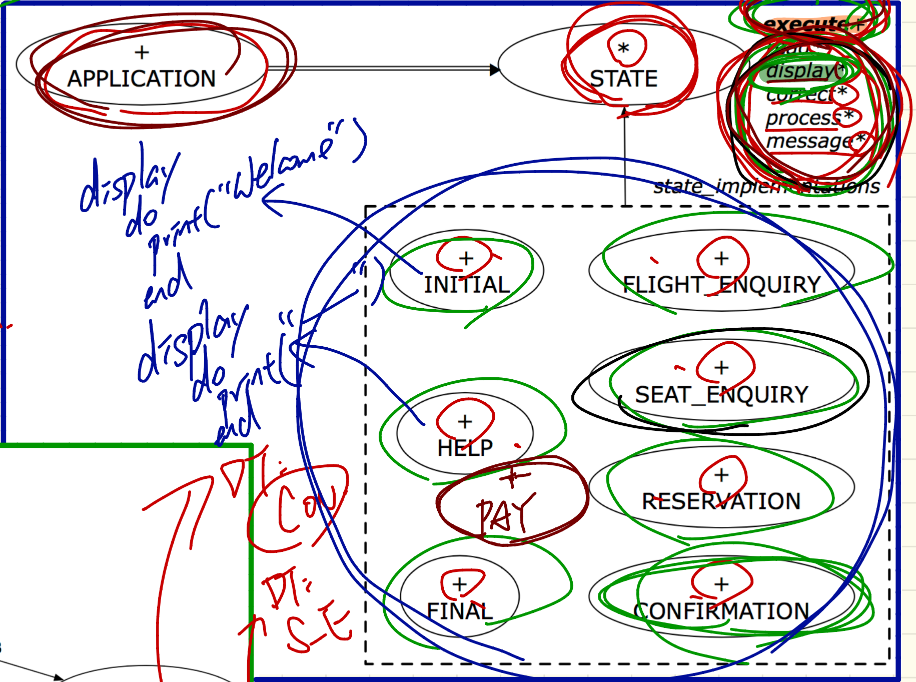
```
display
  do
    print("Welcome")
  end
```

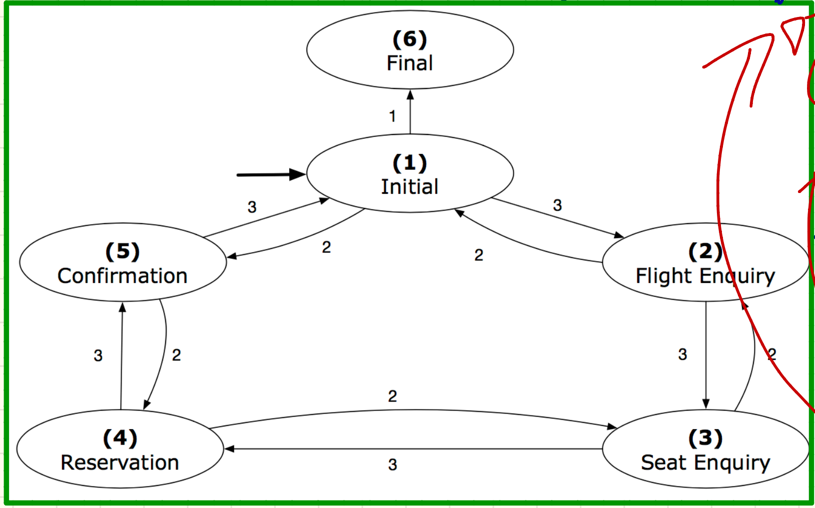
```
display
  do
    print("
  end
```

DT: CON

DT: S_E

PAY

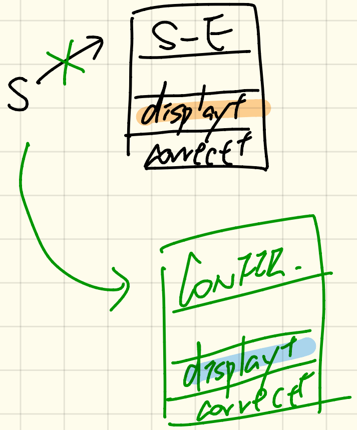




```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

① ②

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
```

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

Dynamic Binding,

TEMPLATE

due to the DT of current object determines which version of display will be called.

$\{0,1\}$

$$\begin{tikzcd}
{} \arrow[r] & A \arrow[r, bend left, "1"] & B \arrow[l, bend left, "0"]
\end{tikzcd}$$

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
```

STATE PATTERN : TEST

index into states array for re-assigning current_state

```
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 1.
    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
```

states

transitions

tar

src

choice

S1: STATE

DT of current_state

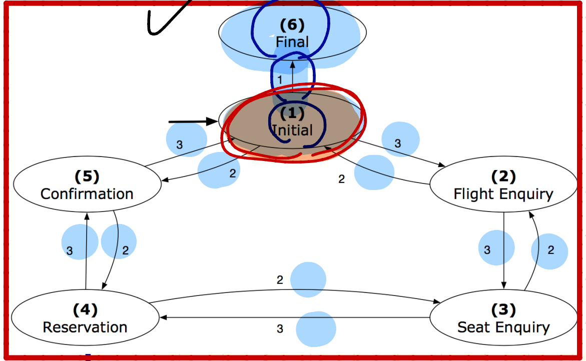
INITIAL

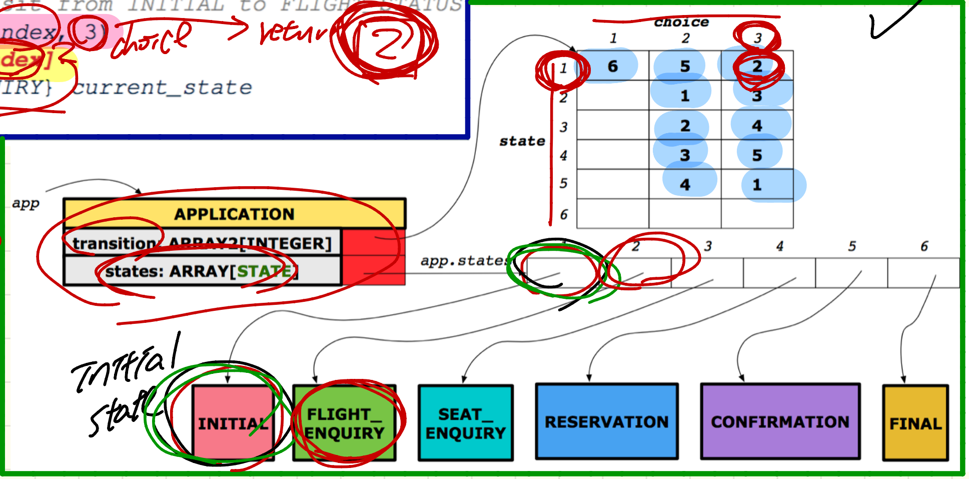
C.S.

choice → return 2

D.T. of current_state?

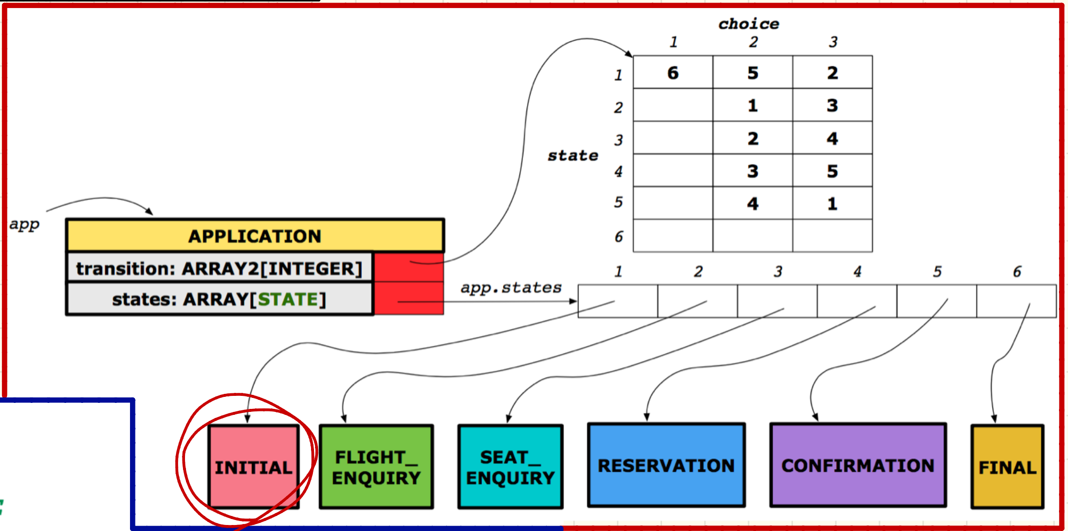
F_T





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

STATE PATTERN: Interactive Session



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

```
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
```

C.S.