Wednesday March 13

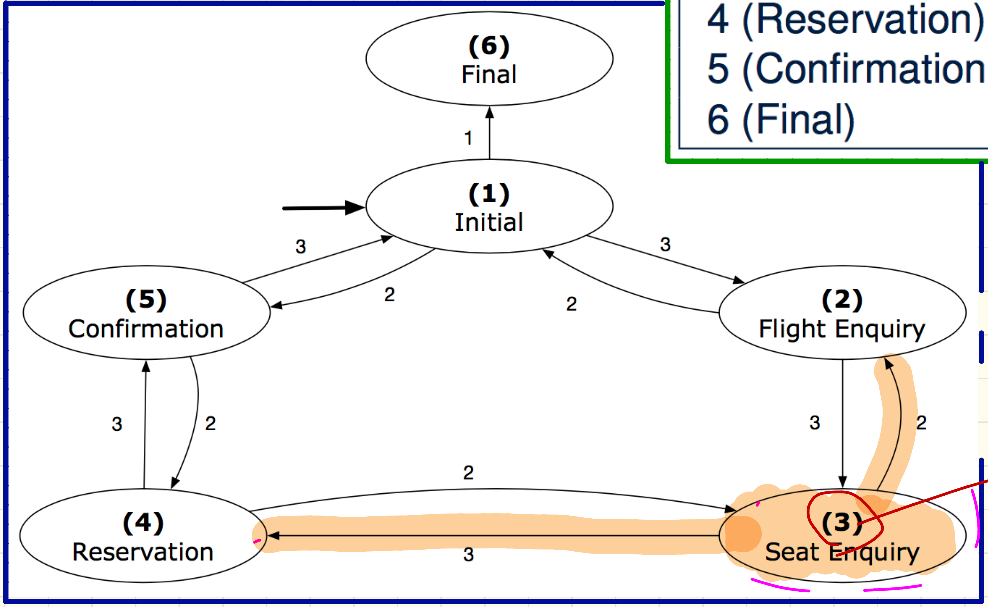
Lecture 17

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)	–	–	–

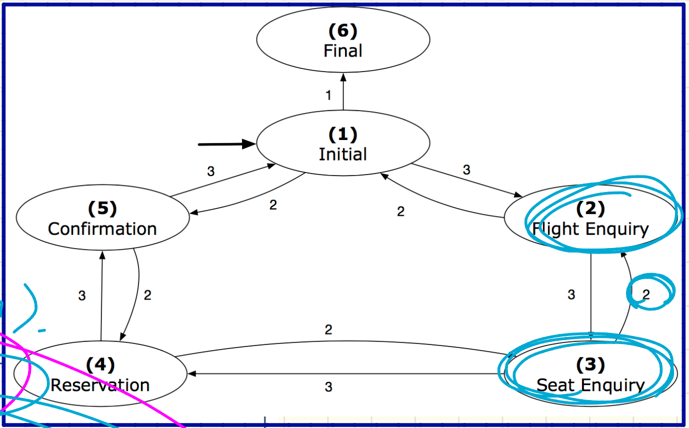
Finite State Machine



1 wrong choice

Design of a Reservation System: First Attempt

- Debugging (spa.. code).
- SCP (duplicates between labels).
- Reusability. X (1. states 2. template for interaction).



```
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
```

Replay -

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)

transition (3, 3)

states [3, 2]

states [3][2]

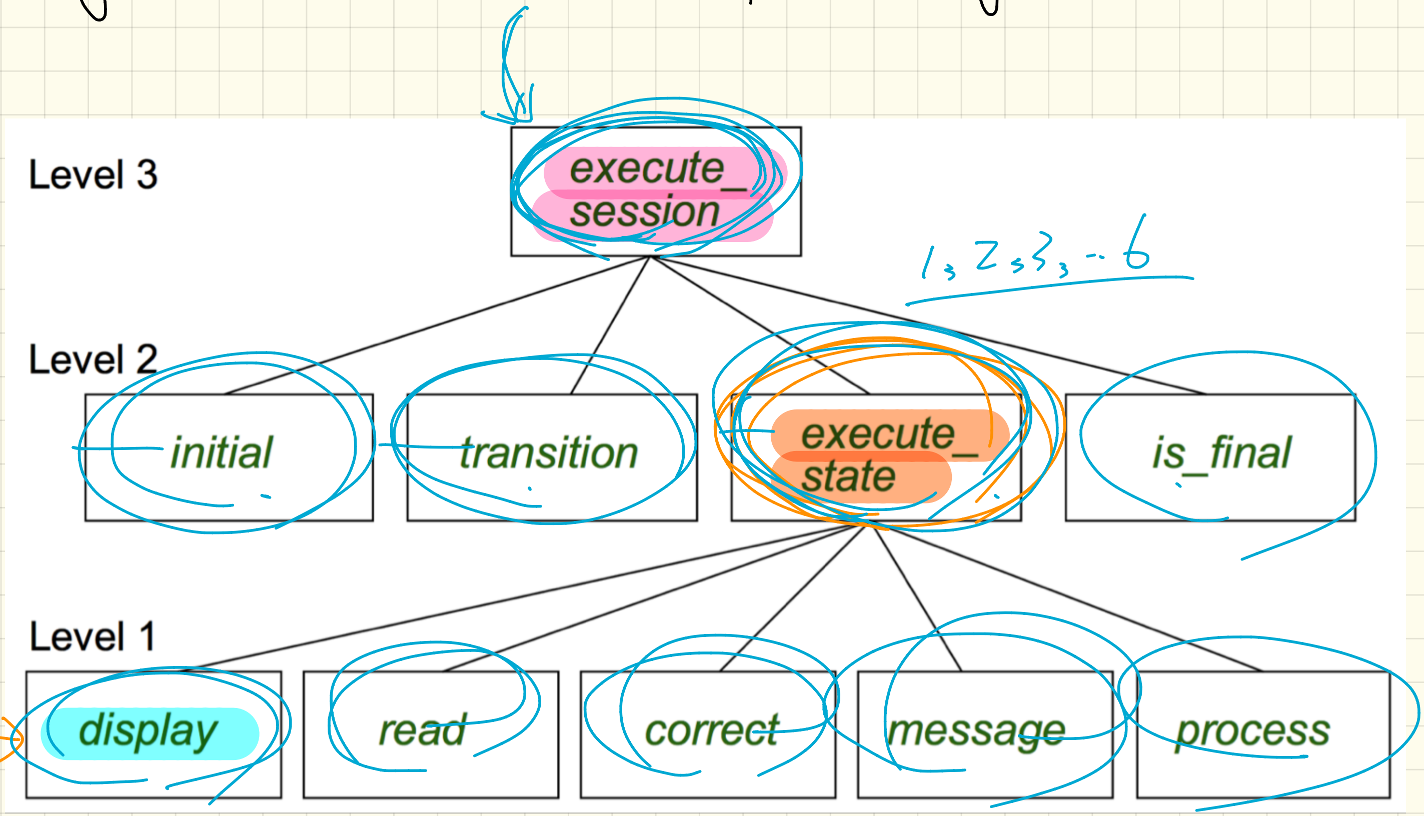
Transition Table

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)	–	–	–

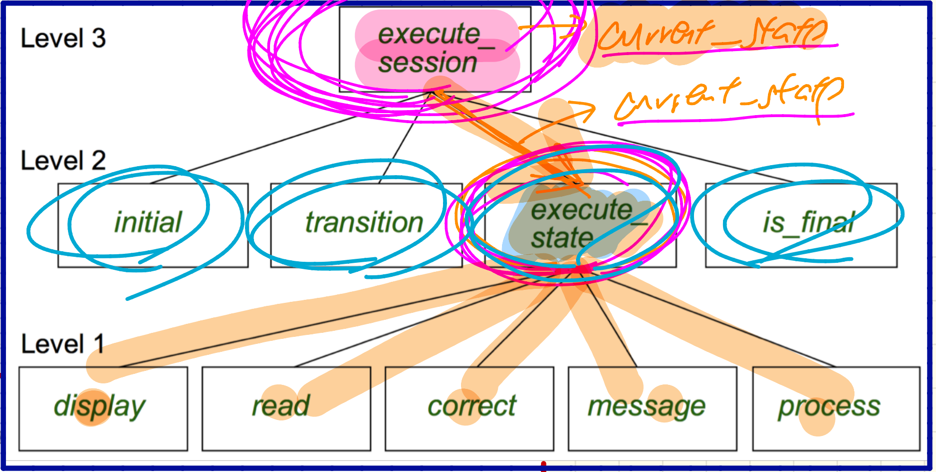
2D-Array Implementation

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: a Top-Down Design



Design of a Reservation System: Second Attempt (2)



```
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
```

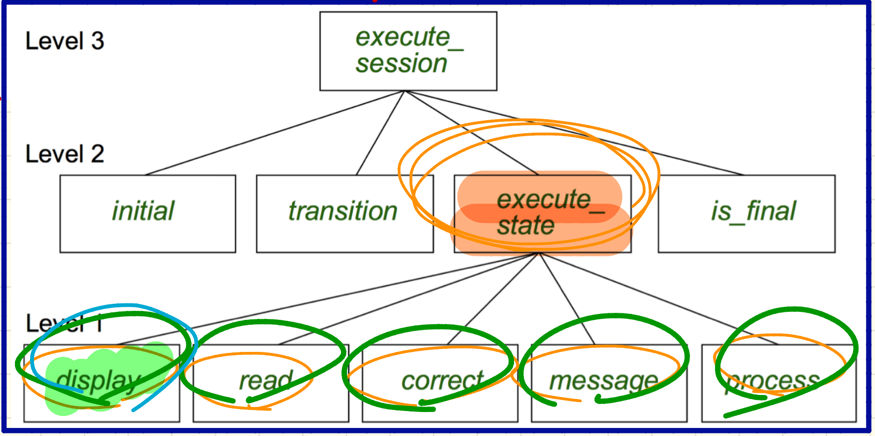
assign initial state

as soon as we reach final state, stop interacting

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                          : ARRAY[
  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
```

```
case current_state of
  1 : ____
  2 : ____
  3 : ____
  4 : ____
  5 : ____
  6 : ____
```



delete state 2 add state 7

display(s: INT)

if s = 1 then

⋮

~~elseif s = 2 then~~

~~⋮~~

elseif s = 3 then

⋮

elseif s = 7 then

⋮

read_answer(s: INT)

if s = 1 then

~~elseif s = 2 then~~

elseif s = 3 then

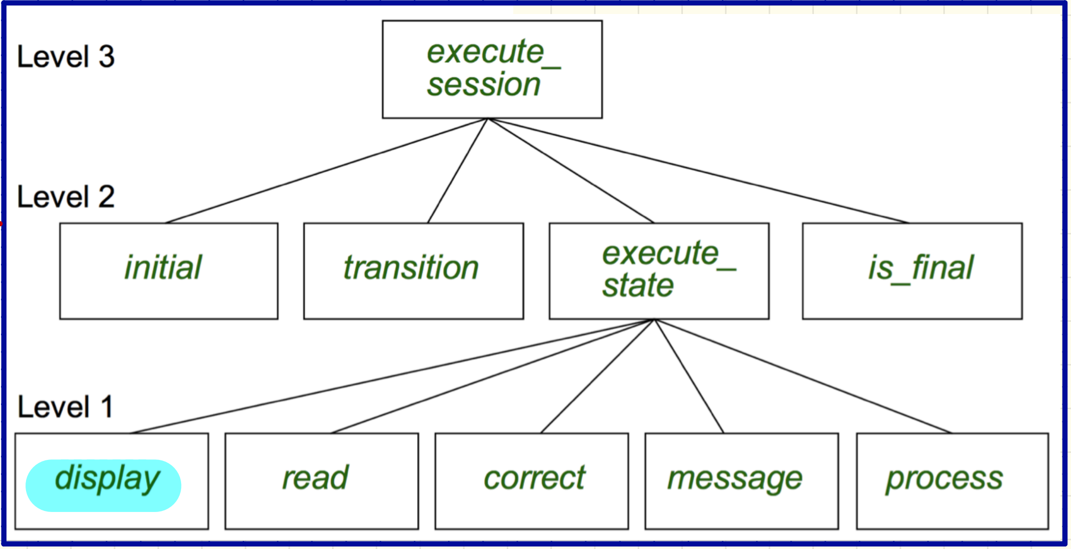
⋮

elseif s = 7 then

⋮

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
```

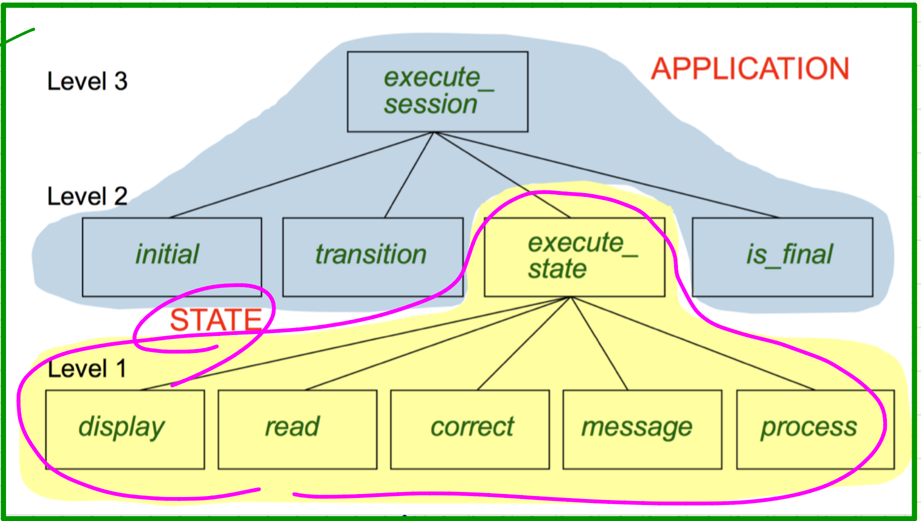


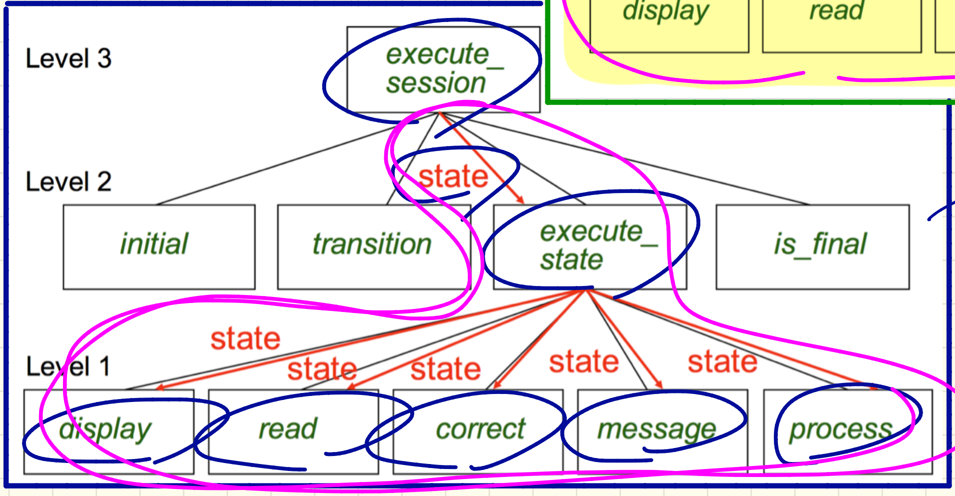
Moving from Hierarchical Design to OO Design

OO

current_state : STATE

current_state . execute_session





HIERARCHICAL

current_state : INTEGER

execute_session (current_state)

↳ read (current_state)

non-OO.

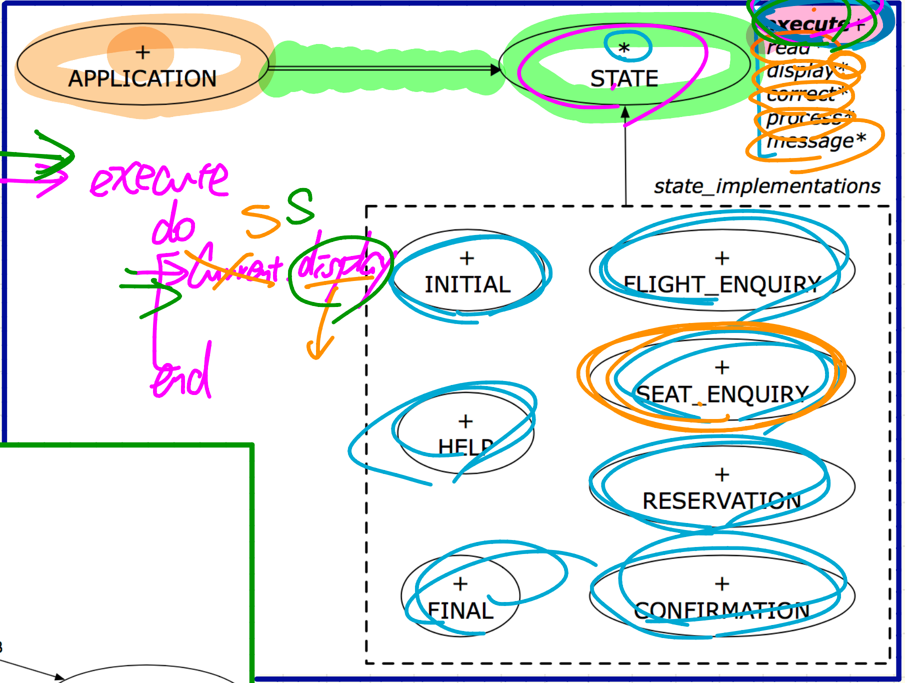
```
current_state := 2
→ execute_state (current_state)
current_state := 4
→ execute_state (current_state)
```

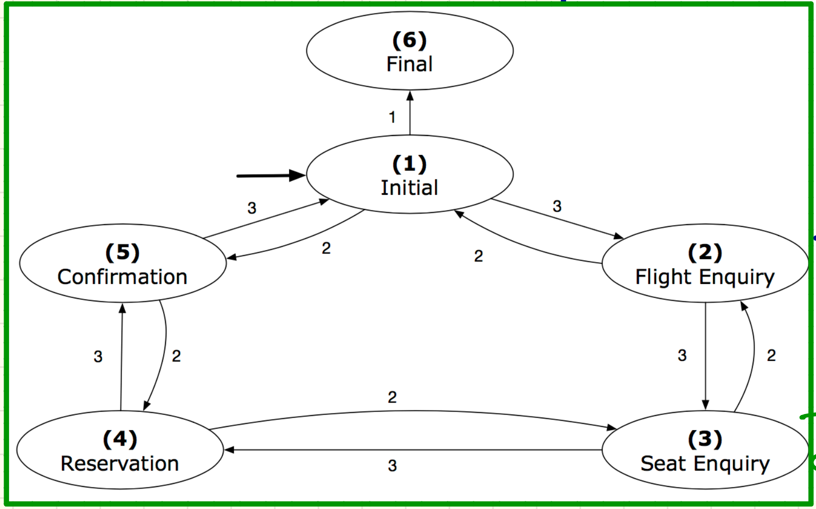
change input into context object

OO.

```
current_state : STATE
create {FLIGHT_INQ} current_state.make
→ current_state.execute
create {RESERVATION} current_state.make
→ current_state.execute
```

STATE PATTERN : Architecture





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

→ call the S_E version of display

→ call the CON. version of display