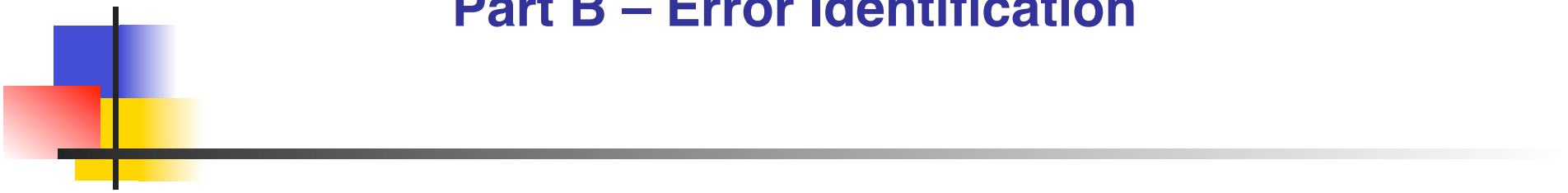
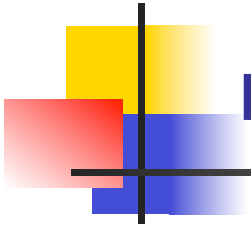


# State-Based Testing
# Part B – Error Identification

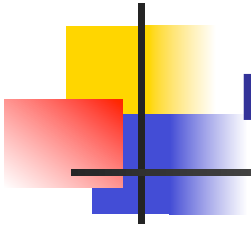
Generating test cases  for complex behaviour

Reference: Robert V. Binder
*Testing Object-Oriented Systems: Models, Patterns, and* Tools
Addison-Wesley, 2000, Chapter 7



# Flattening the statechart

- Statecharts are great for communication, reducing clutter etc.
- They might hide subtle bugs
  - **e.g. entering a sub-state rather than a super-state**



# Flattening the statechart – 2

- For testing we need to expand them to full transition diagrams
  - **Expansion makes implicit transitions explicit, so they are not lost**
  - **Expansion is a flat view**
    - **Includes everything from inheritance in OO and sub-states in statecharts**
- An automatable process

# Concurrent statechart

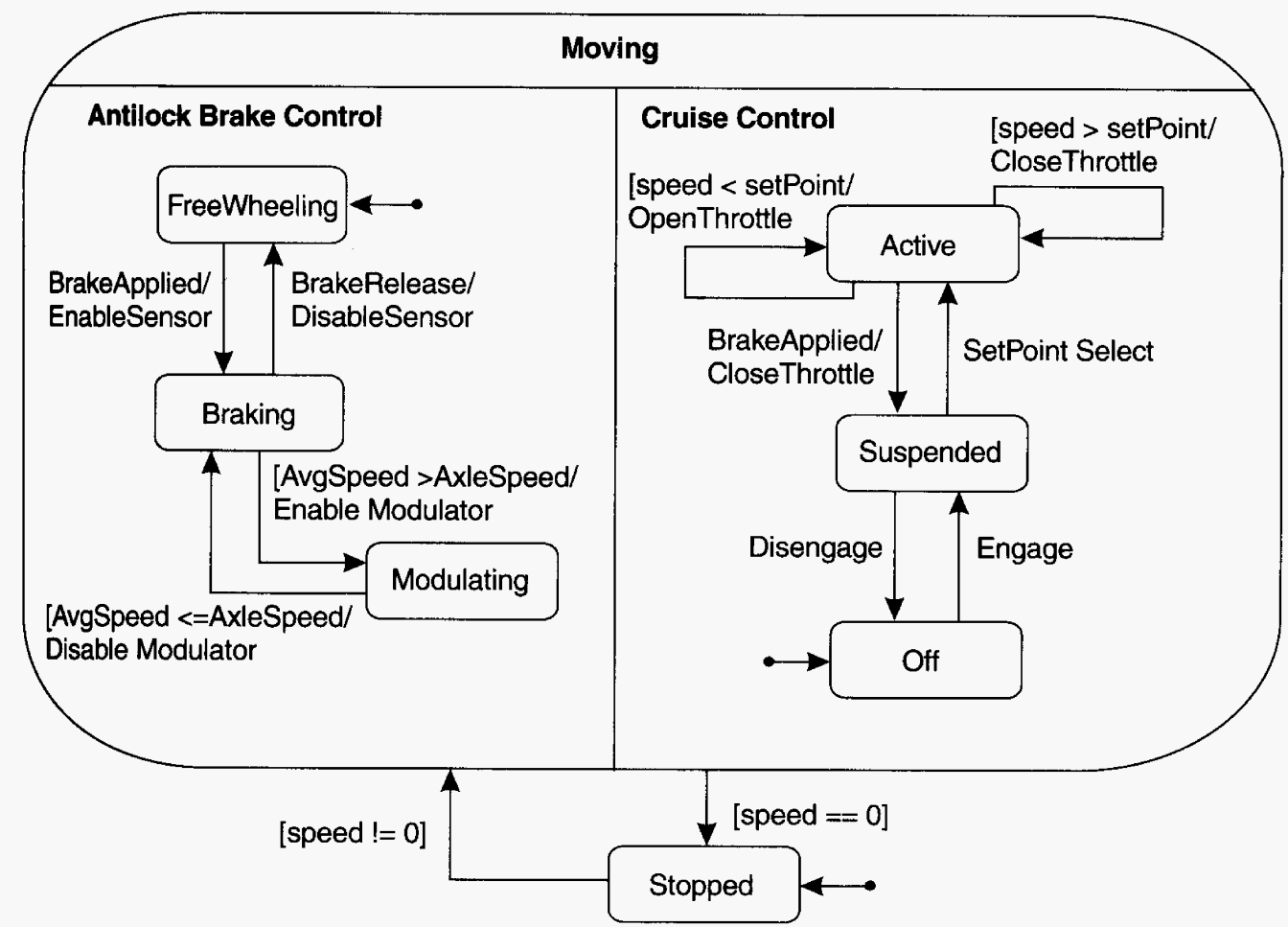
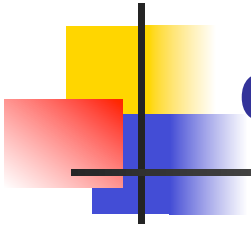



FIGURE 7.14 Automotive control statechart with orthogonal states.

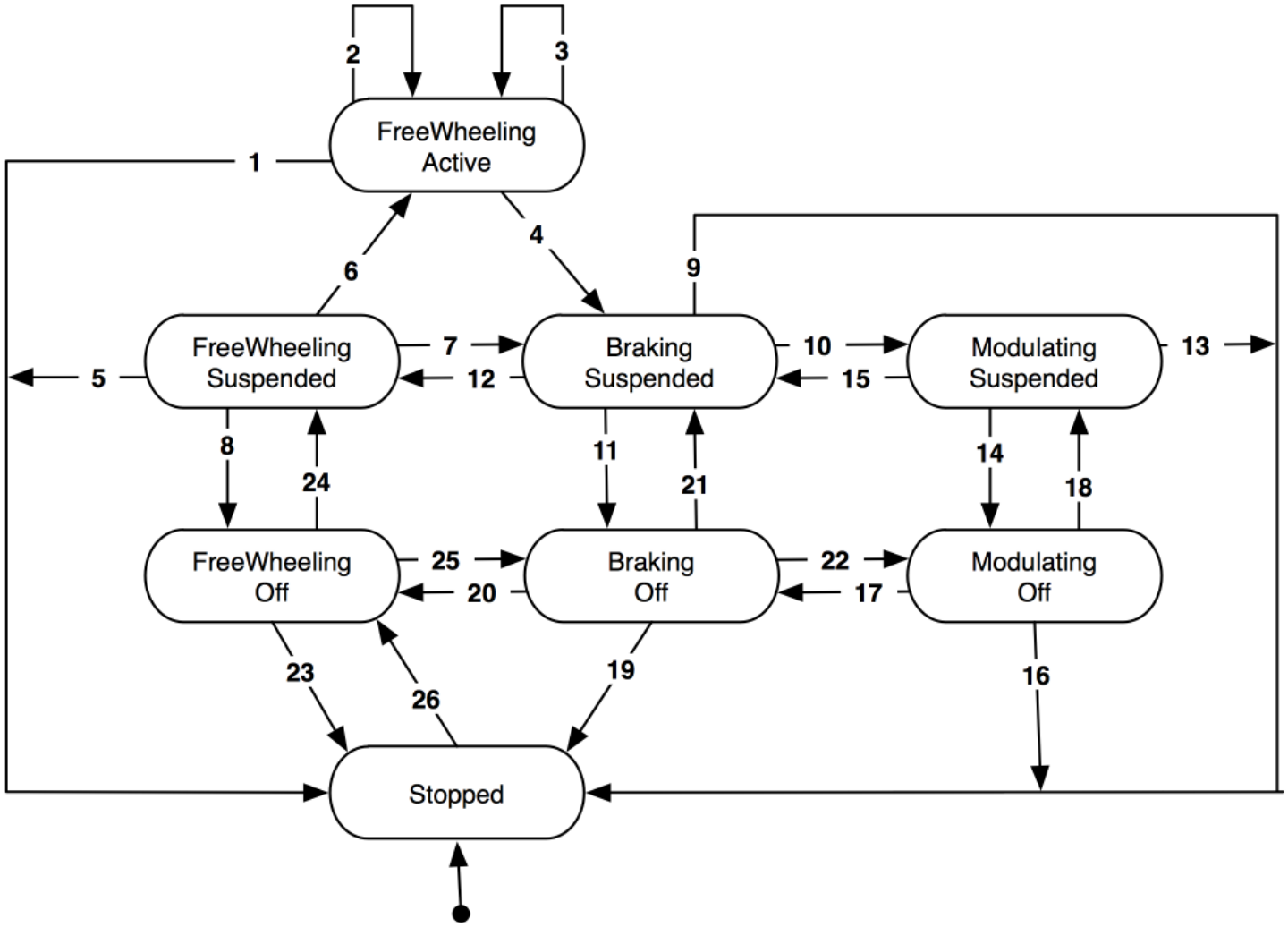


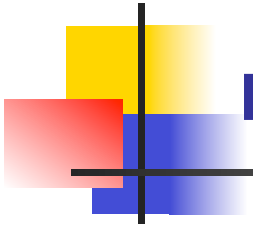
# Concurrency Hides Problems

- Concurrency hides implicit state combinations
  - **Hides potential serious defects**
  - **Arise from implicit state combinations**

- Explicit violations of implicit prohibitions should be tested

# Expanding the Example





# Expanding the Example – 2

## Events, guards and output for the automotive control FSM

1 speed = 0
2 [speed ≥ setPoint] / CloseThrottle
3 [speed < setPoint] / OpenThrottle
4 BrakeApplied / CloseThrottle / EnableSensor

5 speed = 0
6 SetPoint Select
7 BrakeApplied / EnableSensor
8 Disengage

9 speed = 0
10 [AvgSpeed > AxleSpeed] / EnableModulator
11 Disengage
12 BrakeRelese / DisableSensor

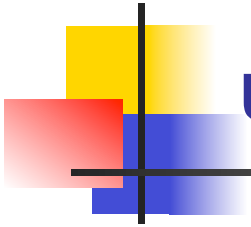
13 speed = 0
14 Disengage
15 [AvgSpeed ≤ AxleSpeed] / DisableModulator

16 speed = 0
17 [AvgSpeed ≤ AxleSpeed] / DisableModulator
18 Engage

19 speed = 0
20 BrakeRelease / DisableSensor
21 Engage
22 [AvgSpeed > AxleSpeed] / EnableModulator

23 speed = 0
24 Engage
25 BrakeApplied / EnableSensor

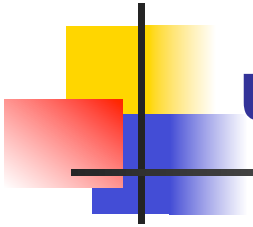
26 speed ≠ 0



# Unspecified Event/State Pairs

- State machine models will not include all events for all states

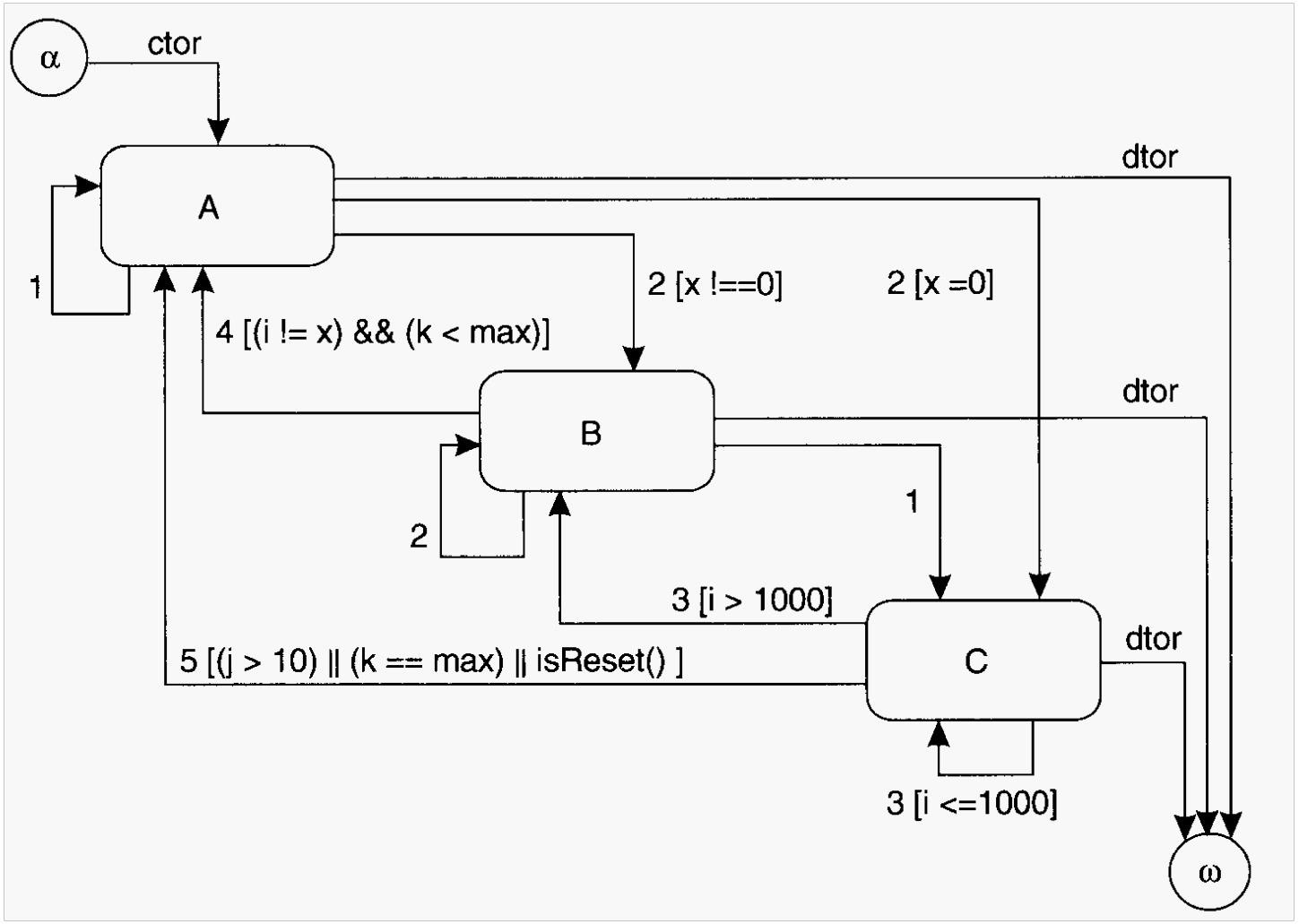
- Implicit transitions may be
  - **Illegal**
  - **Ignored**
  - **Or a specification omission**

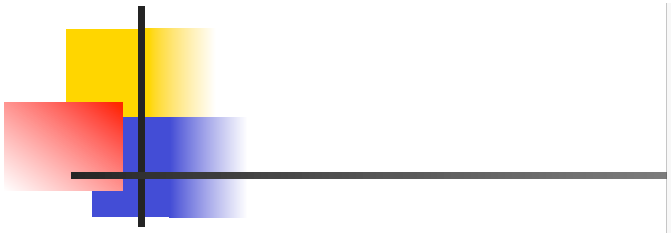


# Unspecified Event/State Pairs – 2

- Accepted illegal events lead to bugs called **sneak paths**
- For testing purposes, we cannot ignore implicit behaviour
  - **Develop a Response Matrix**

# Example statechart



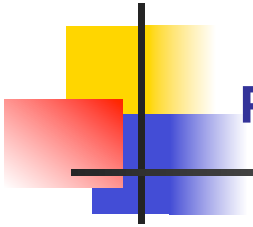


# Response matrix

| Events and Guards | | | | Accepting State/Expected Response | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | α | A | B | . . . | C | |
| ctor | | | | ✓ | 6 | 6 | 6 | 6 | 6 |
| Event 1 | | | | ☒ | ✓ | ✓ | . . . | 2 | 6 |
| Event 2 | x == 0 | | | | | | | | |
| | DC | | | ☒ | ☒ | ✓ | . . . | 2 | 6 |
| | F | | | ☒ | ✓ | ☒ | . . . | ☒ | ☒ |
| | T | | | ☒ | ✓ | ☒ | . . . | ☒ | ☒ |
| Event 3 | i <= 1000 | | | | | | | | |
| | DC | | | ☒ | 2 | 2 | . . . | ☒ | 6 |
| | Off | | | ☒ | 2 | 2 | . . . | ✓ | ☒ |
| | On | | | ☒ | 2 | 2 | . . . | ✓ | ☒ |
| Event 4 | i != x | k < max | | | | | | | |
| | DC | DC | | ☒ | 2 | ☒ | . . . | 2 | 6 |
| | F | F | | ☒ | ☒ | 1 | . . . | ☒ | ☒ |
| | F | T | | ☒ | ☒ | 2 | . . . | ☒ | ☒ |
| | T | F | | ☒ | ☒ | 2 | . . . | ☒ | ☒ |
| | T | T | | ☒ | ☒ | ✓ | . . . | ☒ | ☒ |
| Event 5 | i > 10 | k == max | isReset() | | | | | | |
| | DC | DC | DC | ☒ | 2 | 5 | . . . | ☒ | 6 |
| | F | F | F | ☒ | ☒ | ☒ | . . . | 5 | ☒ |
| | F | F | T | ☒ | ☒ | ☒ | . . . | ✓ | ☒ |
| | F | T | F | ☒ | ☒ | ☒ | . . . | ✓ | ☒ |
| | F | T | T | ☒ | ☒ | ☒ | . . . | ✓ | ☒ |
| | T | F | F | ☒ | ☒ | ☒ | . . . | ✓ | ☒ |
| | T | F | T | ☒ | ☒ | ☒ | . . . | ✓ | ☒ |
| | T | T | F | ☒ | ☒ | ☒ | . . . | ✓ | ☒ |
| | T | T | T | ☒ | ☒ | ☒ | . . . | ✓ | ☒ |
| dtor | | | | ☒ | ✓ | ✓ | . . . | ✓ | 2 |

*Key:* T = true, F = false, DC = don't care; for Action codes, see Table 7.3

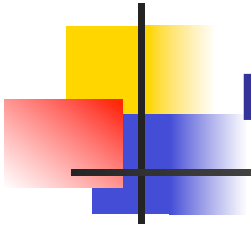
☐ Not applicable ☒ Excluded ☑ Explicitly specified transition



# Possible responses to illegal events

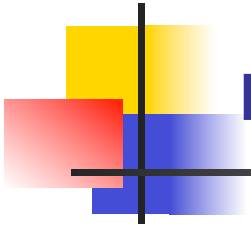
**TABLE 7.3** Response Codes for Illegal Events

| Response Code | Name | Response |
|---|---|---|
| 0 | Accept | Perform the explicitly specified transition |
| 1 | Queue | Place the illegal event in a queue for subsequent evaluation and ignore |
| 2 | Ignore | No action or state change is to be produced, no error is returned, no exception raised |
| 3 | Flag | Return a nonzero error code |
| 4 | Reject | Raise an `IllegalEventException` |
| 5 | Mute | Disable the source of the event and ignore |
| 6 | Abend | Invoke abnormal termination services (e.g., core dump) and halt the process |



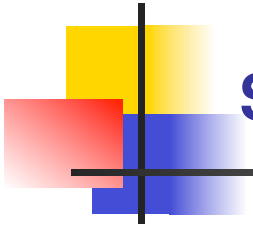
# Designing responses to illegal events

- Abstract state should not change
  - **Concrete state may change due to exception handling**
- Illegal event design question
  - **Handle with defensive programming**
    - **Uncooperative (defensive) systems**
  - **Handle with precondition contracts**
    - **Cooperative systems**



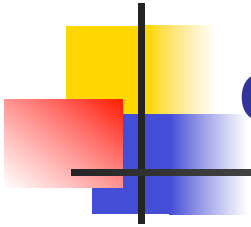
# Designing responses to illegal events – 2

- Possible responses
  - **Raise exception**
  - **Treat message as a noop**
  - **Attempt error recovery**
  - **Invoke abnormal termination**
- Tester needs to decide expected responses so actual responses can be evaluated



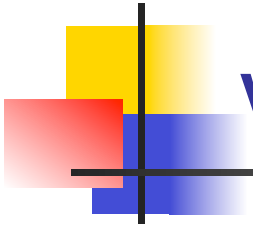
# State model validation

- Before it is used to generate test cases a state model must be
  - **Complete**
  - **Consistent**
  - **Correct**
    - **Passes checklists**



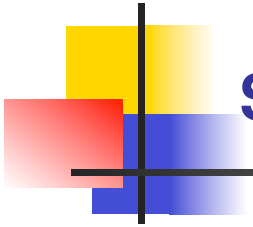
# Checklist questions

- What is a checklist?
- Why do we use checklists?
- What checklists would be useful for statecharts?



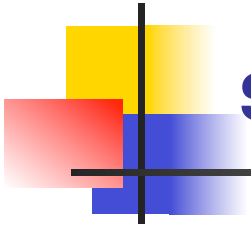
# Validation checklists

- We will look at five validation checklists
  - **Structure checklist**
  - **State name checklist**
  - **Guarded transition checklist**
  - **Robustness checklist**
  - **Well-formed subclass behaviour checklist**



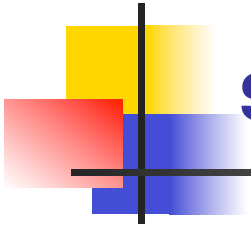
# Structure checklist question

- **What would you look for to verify the structure of a statechart is correct?**



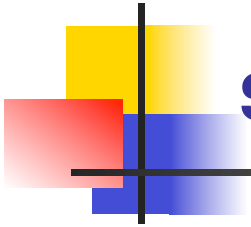
# Structure checklist

- There is an initial state with only outbound transitions
- There is a final state with only inbound transitions
  - **If not, explicit reason is needed**
- Except for the initial and final states, every state has at least one incoming and one outgoing transition



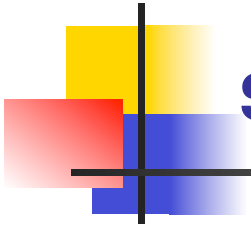
# Structure checklist – 2

- Every state is reachable from the initial state
- The final state is reachable from all states
- No equivalent states



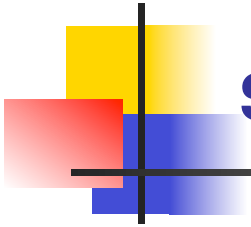
# Structure checklist – 3

- Every defined event and every defined action appears in at least one transition
- The events accepted in a particular state are
  - **Unique**
  - **Or differentiated by mutually exclusive guards**
- Complete specification
  - **For every state, every event is accepted or rejected**
    - **Either explicitly or implicitly**



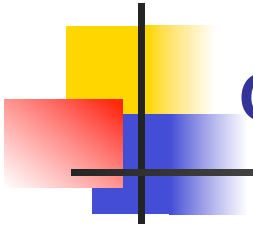
# State name checklist

- Poor names are indications of
  - **Incomplete design**
  - **Or incorrect design**
- Names must be meaningful in the context of the application
- Adjectives are best
  - **Past participles are OK**



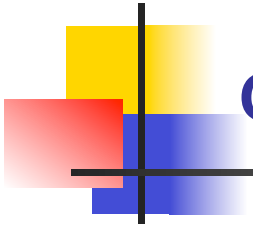
# State name checklist – 2

- State names should be passive
- If a state is not necessary, leave it out
  - **“Wait states” are often superfluous**



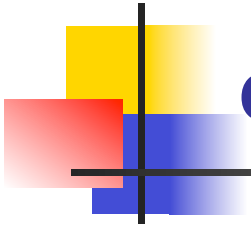
# Guarded transition checklist question

- **What would you look for to ensure the guards on transitions are correct in a statechart?**



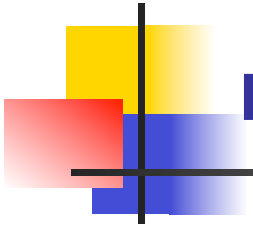
# Guarded transition checklist

- Guard variables are visible
- The entire range of truth values for a particular event is covered
- Each guard is mutually exclusive of all other guards



# Guarded transition checklist – 2

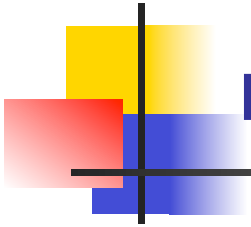
- Guards with three or more variables are modeled with a decision table
- The evaluation of a guard does not cause side effects



# Robustness checklist

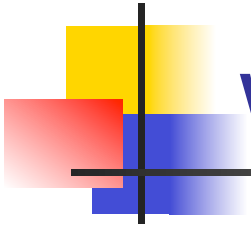
- There is an explicit spec for an error-handling or exception-handling mechanism for implicitly rejected events

- Illegal events do not corrupt the machine
  - **Preserve the last good state**
  - **Reset to a valid state**
  - **Or self-destruct safely**



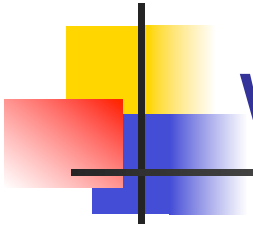
# Robustness checklist – 2

- Actions have no side effects on the resultant state
- For contract violations specify mechanism for
  - **Explicit exception**
  - **Error logging**
  - **Recovery**



# Well-Formed Subclass Behaviour Checklist

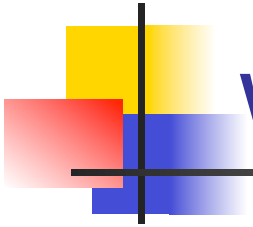
- Does not remove any superclass states
  - **All transitions accepted in the superclass are accepted in the subclass**
- All guards on superclass transitions are
  - **The same for subclass transitions**
  - **Or weaker for subclass transitions**



# Well-Formed Subclass Behaviour Checklist – 2

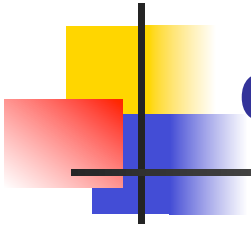
- Subclass does not weaken the state invariant of the superclass

- Subclass may add an orthogonal state defined with respect to its locally introduced instance variables



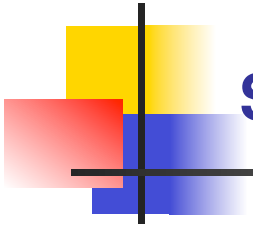
# Well-Formed Subclass Behaviour Checklist – 3

- All inherited actions are consistent with the subclass's responsibilities
  - **Verify name-scope sensitive or dynamic binding of intraclass messages is correct**
- All inherited accessor events are appropriate in the context of the subclass
- Messages sent to objects that are variables in a guard expression do not have side effects on the class under test



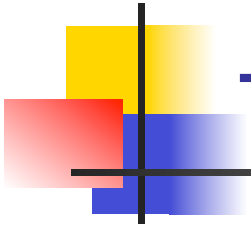
# Control faults for state machines

- An incorrect sequence of events is accepted
- An incorrect sequence of outputs is produced



# State control faults question

- **What types of state control faults can occur in a statechart?**



# Types of state control faults

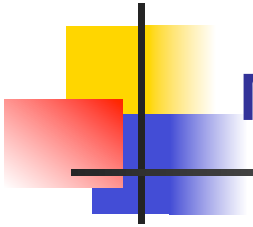
| | |
|---|---|
| Missing transition | Incorrect transition |
| Missing action | Incorrect action |
| Trap door | Sneak path |
| Corrupt state | Illegal message failure |

**Occur individually or any nightmare combination**

**Are all combinations possible?**

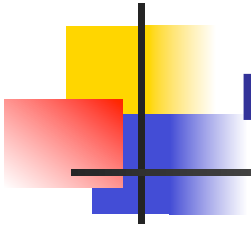
# State control faults

| Event | Action | Resultant State | Error Description |
|---|---|---|---|
| OK | OK | OK | Normal transition |
| | | Wrong | Incorrect state |
| | | Corrupt | Corrupted state |
| | Wrong/ undefined | OK | Incorrect action |
| | | Wrong | Incorrect action, wrong state |
| | | Corrupt | Incorrect action, corrupt state |
| | Missing | OK | Missing action |
| | | Wrong | Missing action, incorrect state |
| | | Corrupt | Missing action, corrupt state |
| Reject Legal | DC | Same | Missing transition, no side effect |
| | | Defined | Missing transition, side effect |
| | | Corrupt | Missing transition, corrupt state |
| Accept Illegal | Defined for this machine | Same | Sneak path, no side effect |
| | | Defined | Sneak path with side effect |
| | | Corrupt | Sneak path to corrupt state |
| | Undefined | Same | Sneak path, no side effect, incorrect output |
| | | Defined | Sneak path with side effect, incorrect output |
| | | Corrupt | Sneak path to corrupt state, incorrect output |
| Accept Undefined | Defined for this machine | Same | Trap door to action |
| | | Defined | Trap door to action, side effect |
| | | Corrupt | Trap door to action, corrupt state |
| | Undefined | Same | Trap door with incorrect output |
| | | Defined | Trap door with incorrect output and side effect |
| | | Corrupt | Trap door with incorrect output to corrupt state |



# Missing transition question

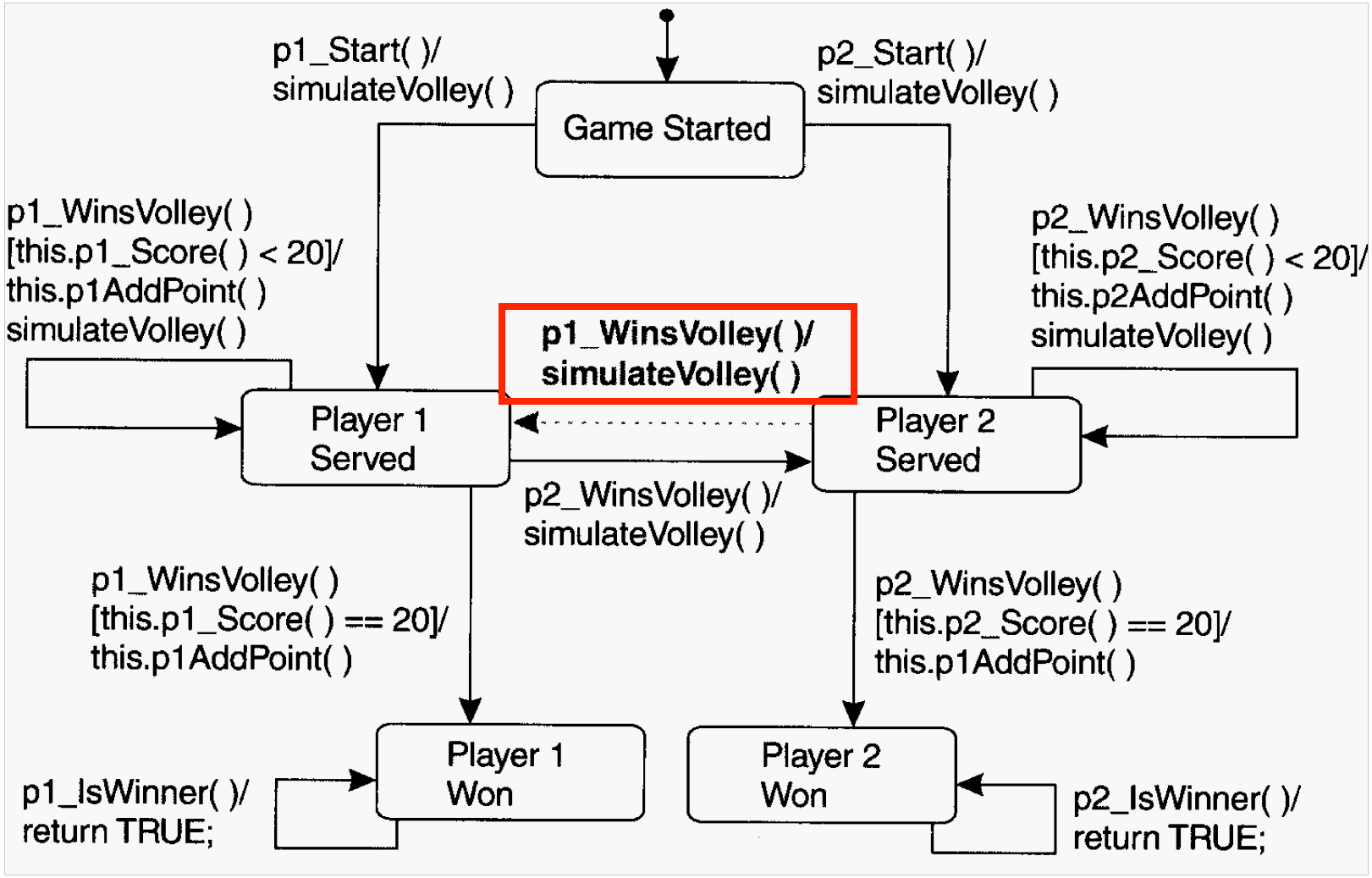
- **How can you tell from the behaviour of a statechart that there is a missing transition?**

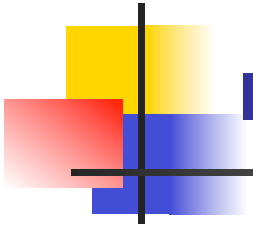


# Missing transition

- Implementation does not respond to a valid event-state pair
- Resultant state is incorrect but not corrupt

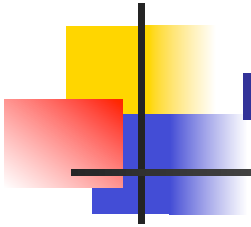
# Missing transition – 2





# Incorrect transition question

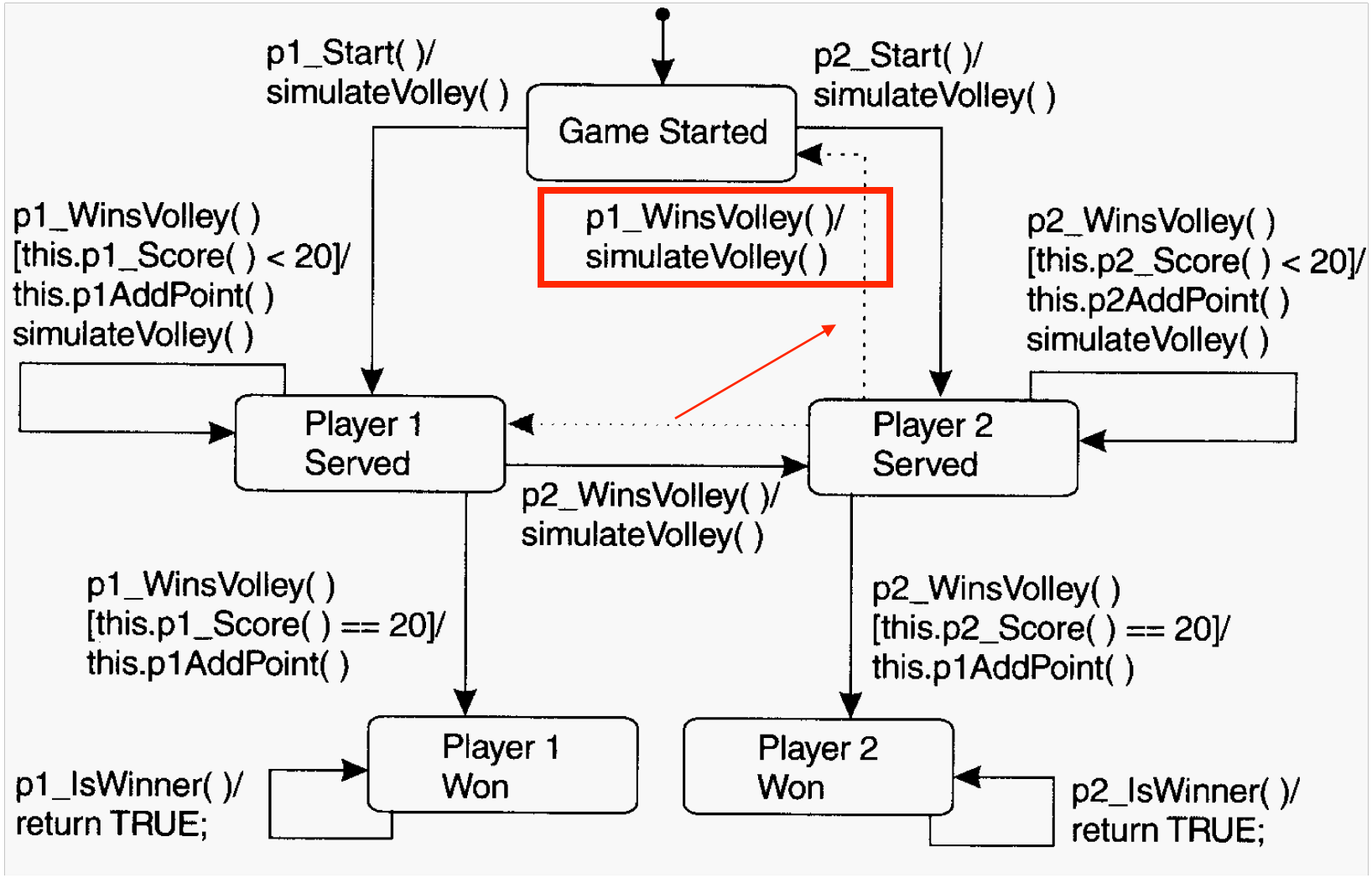
- **How can you tell from the behaviour of a statechart that there is an incorrect transition?**

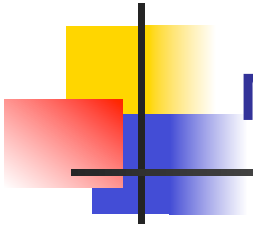


# Incorrect transition

- Implementation behaves as if an incorrect resultant state has been reached
- Resultant state is incorrect but not corrupt

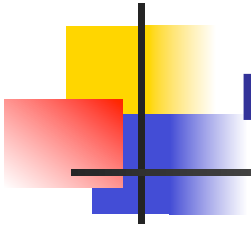
# Incorrect transition – 2





# Missing action question

- **How can you tell from the behaviour of a statechart that there is a missing action?**

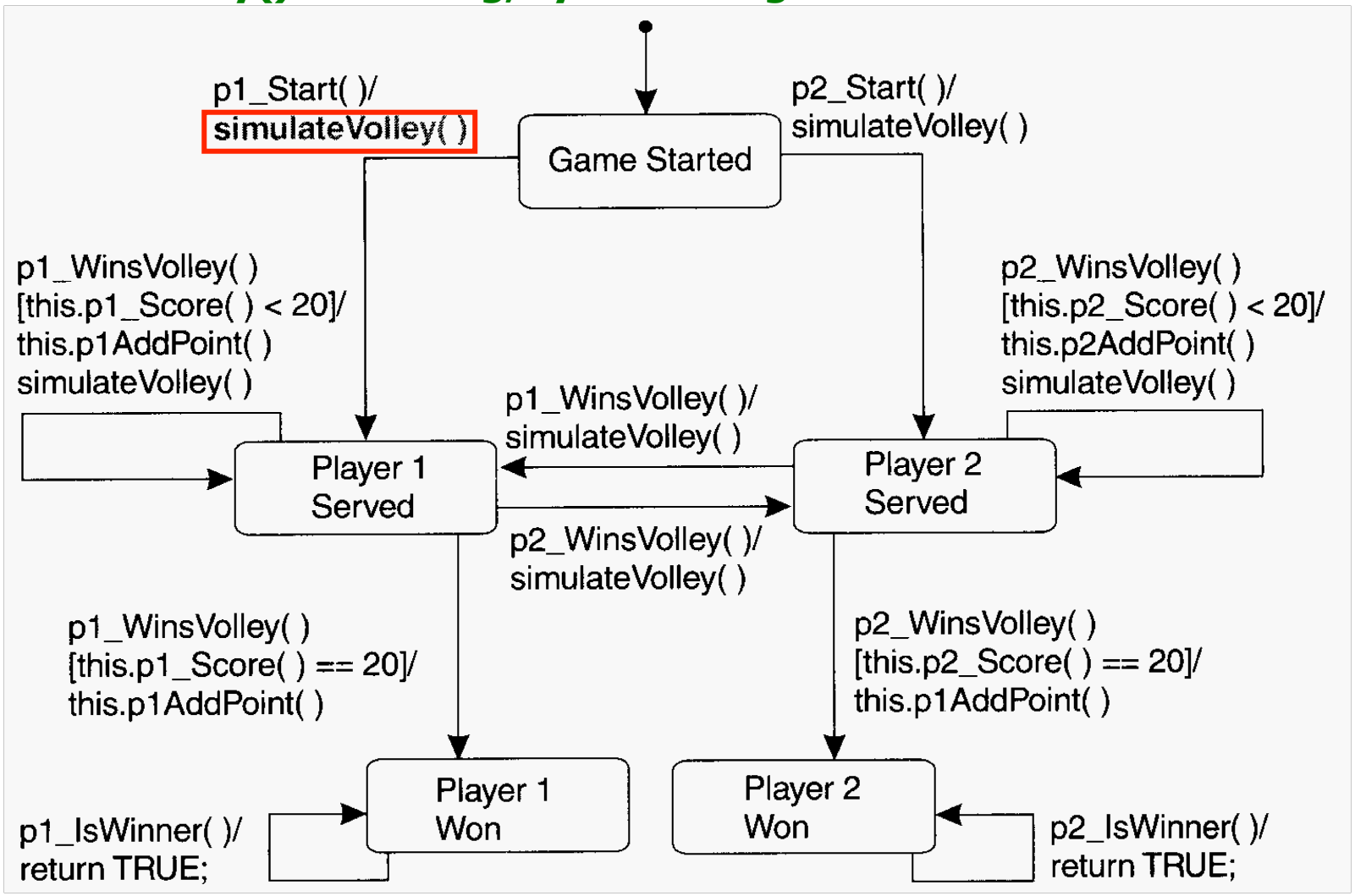


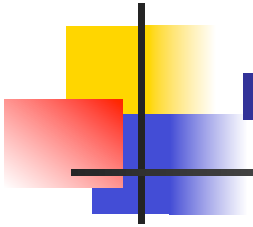
# Missing action

- Implementation does not have an action for a transition
  - **Can have incorrect output**
  - **Later be in an incorrect state**
  - **Wait forever for the missing action to occur**

# Missing action – 2

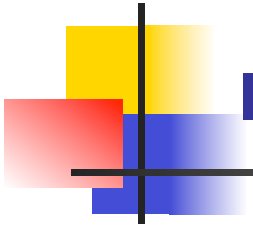
**If simulateVolley() is missing, system hangs**





# Incorrect action question

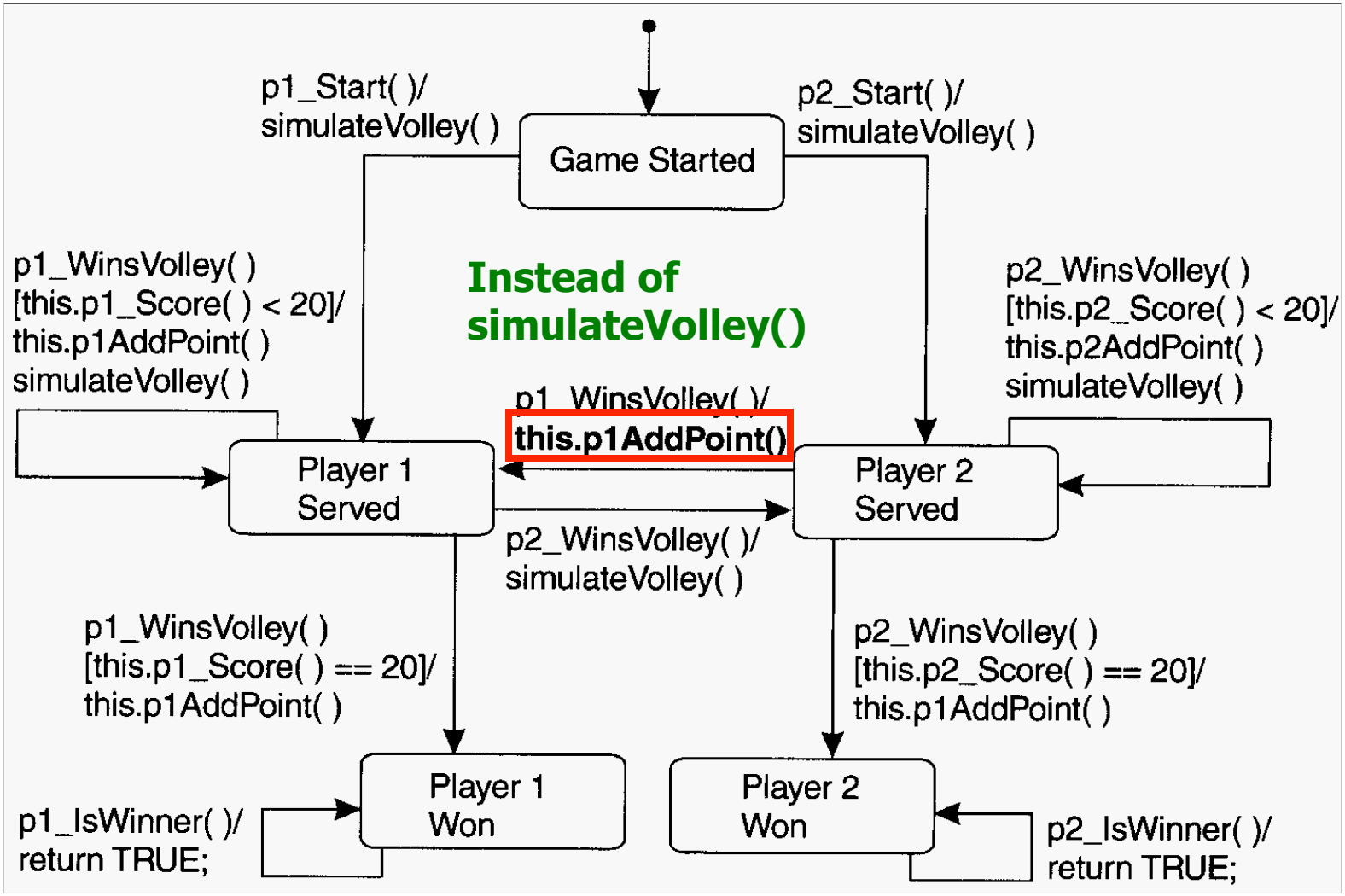
- **How can you tell from the behaviour of a statechart that there is an incorrect action?**

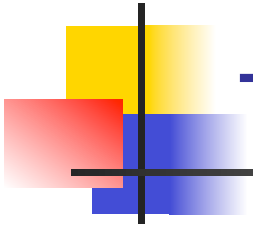


# Incorrect action

- Implementation produces the wrong action for a transition
  - **Different case from incorrect output for an action**
  - **Have incorrect output**
  - **Later be in an incorrect state**

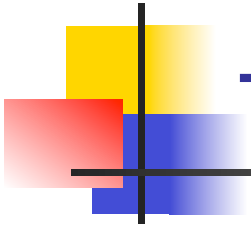
# Incorrect action – 2





# Trap door question

- **How can you tell from the behaviour of a statechart that there is a trap door?**



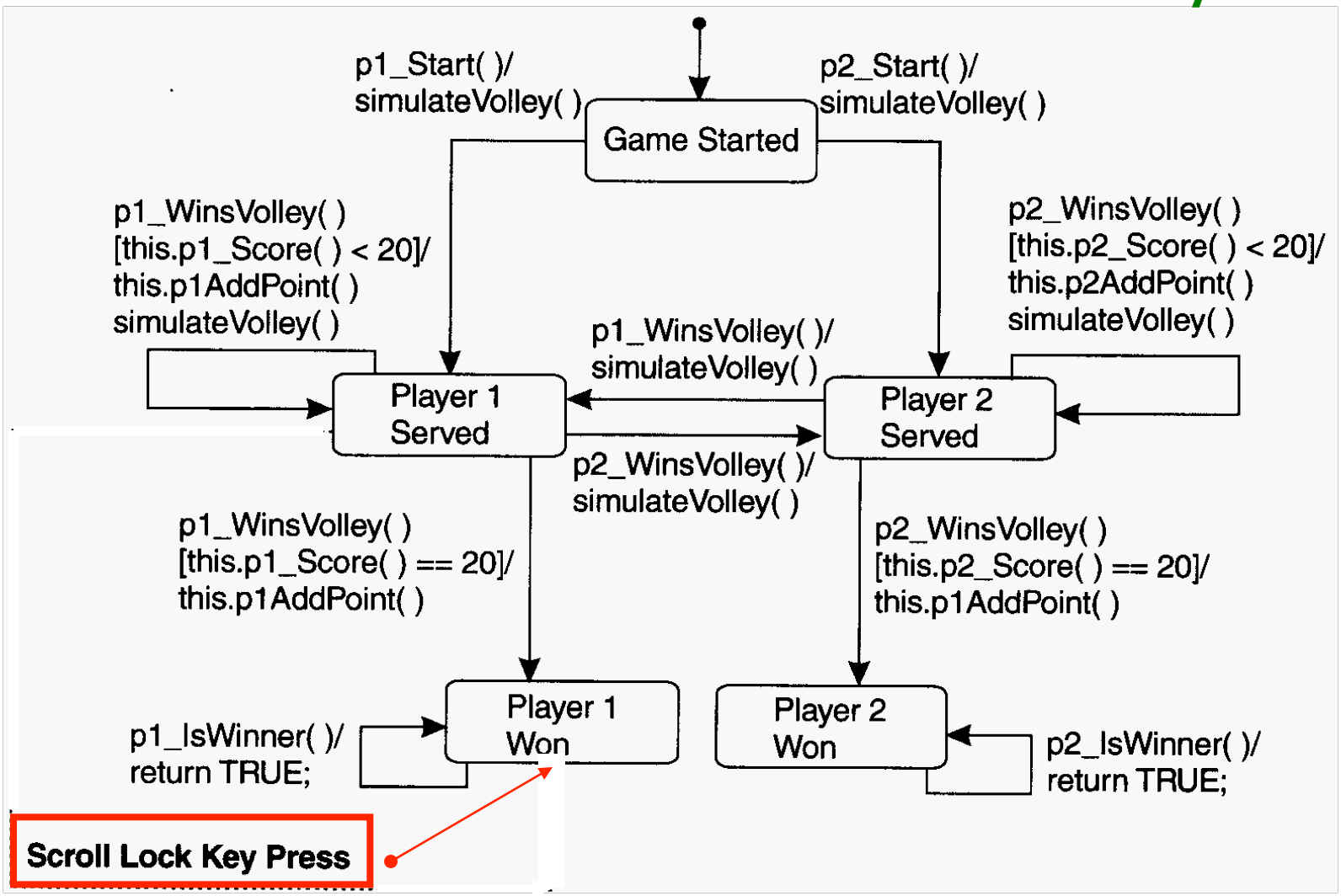
# Trap door

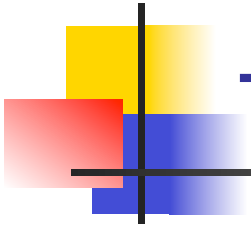
- Implementation accepts an **entry event** that is not defined in the specification

- Can result in
  - **Incorrect output**
  - **Corrupt state**
  - **Enter wrong state**
  - **Or combination**

# Trap door – 2

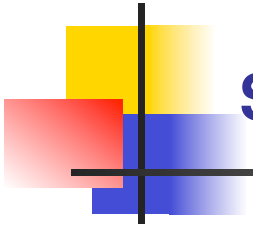
**Transition enters the system.**





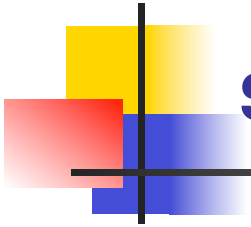
# Trap door – 3

- Can result from
  - **Obsolete features that were not removed**
  - **Inherited features that are inconsistent with the requirements of the subclass**
  - **"*Undocumented*" features added by the developer for debugging purposes**
  - **Sabotage for criminal or malicious purposes**



# Sneak path question

- **How can you tell from the behaviour of a statechart that there is a sneak path?**

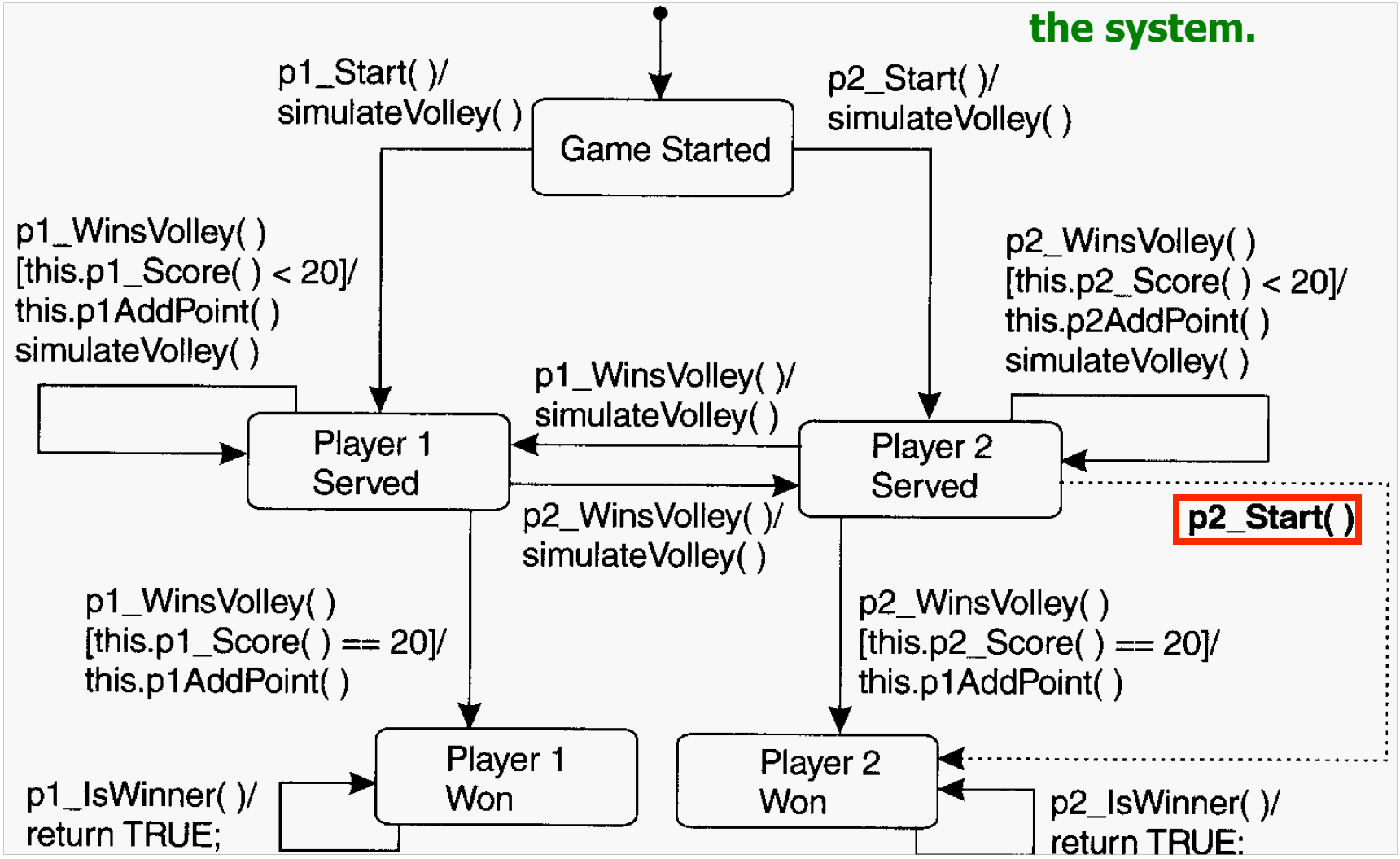


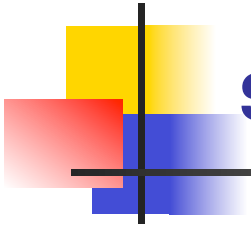
# Sneak path

- Implementation for a state accepts an event that is
  - **Illegal**
  - **Or unspecified**
- Can result in
  - **Incorrect output**
  - **Corrupt state**
  - **Enter wrong state**
  - **Or combination**

# Sneak path

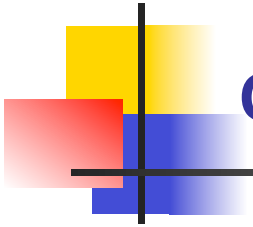
**Transition is within the system.**





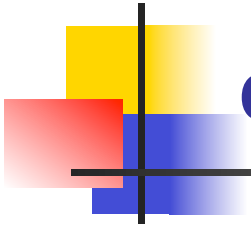
# Sneak path – 3

- Can result from
  - **Obsolete features that were not removed**
  - **Inherited features that are inconsistent with the requirements of the subclass**
  - **"*Undocumented*" features added by the developer for debugging purposes**
  - **Sabotage for criminal or malicious purposes**



# Corrupt state question

- **How can you tell from the behaviour of a statechart that there is a corrupt state?**

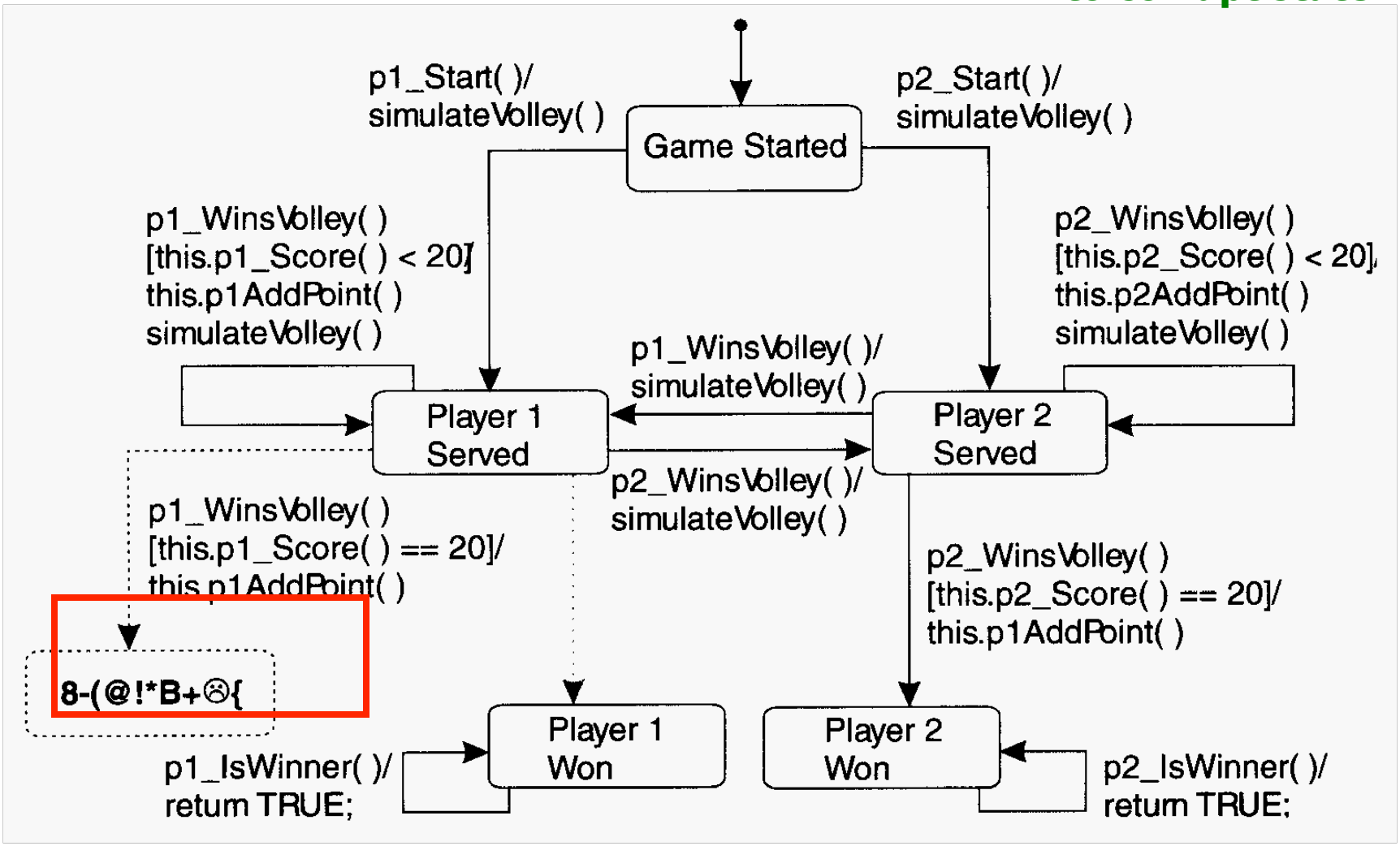


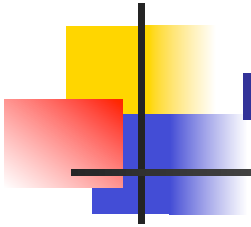
# Corrupt state

- Implementation computes a state that is not valid
  - **Either the class invariant or state invariant is violated**
    - **Due to coding and / or design errors**

# Corrupt state – 2

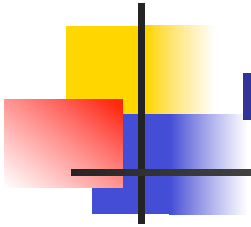
**Incorrect transition to corrupt state**





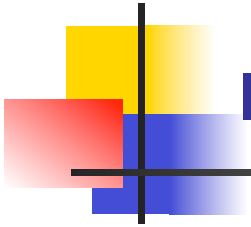
# Incorrect Composite Behaviour

- Misuse of inheritance with modal classes can lead to state control bugs
  - **Subclasses can conflict with sequential requirements of a superclass**
  - **Need to test beyond the scope of one class**



# Incorrect Composite Behaviour – 2

- Bugs occur for the following reasons
  - **Missing or incorrect redefinition of a method**
  - **Subclass extension of the local state conflicts with a superclass state**
  - **Subclass fails to retarget a superclass transition**
    - **Switches to an incorrect or undefined superclass state**



# Incorrect Composite Behaviour – 3

- Bugs occur for the following reasons – cont'd
  - **Order of evaluation of guards and preconditions is incorrect or sensitive to the order of evaluation**
    - **Guards behave as if an extra state exists**
  - **Order of guard evaluation produces a side effect in the subclass that is not present in the superclass**
  - **Default name scope resolution results in guard parameters being bound to the wrong subclass or superclass methods**