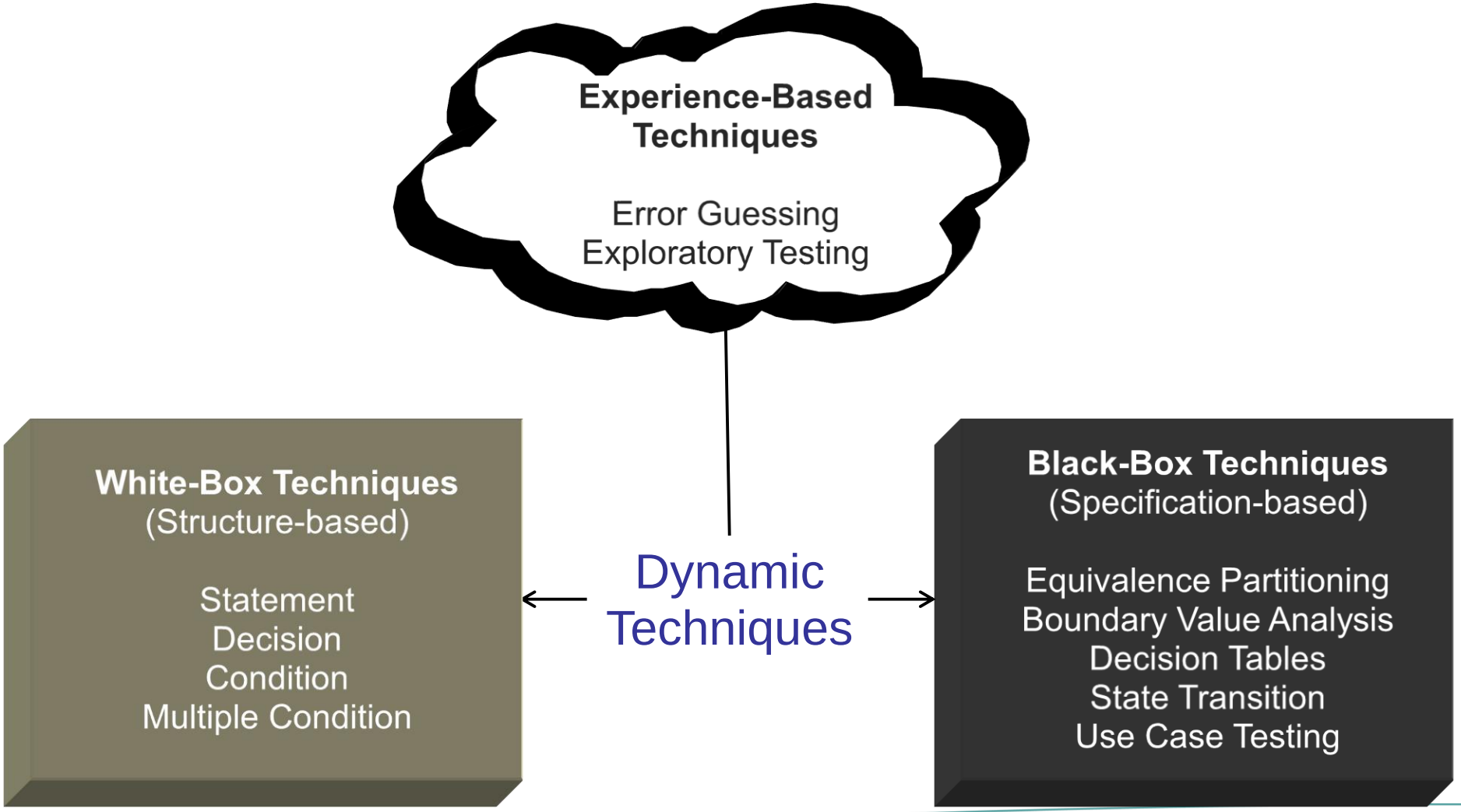
More Black and White Box Testing

Dynamic Testing Techniques



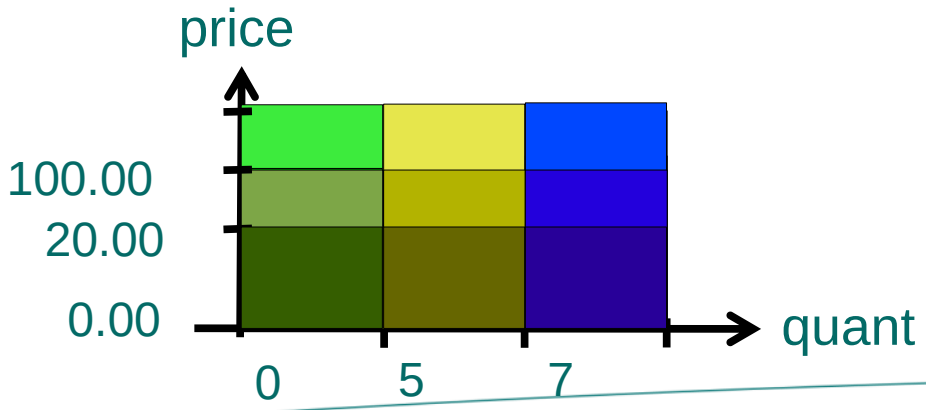
Decision Table Testing

- Equivalence partitioning derives partitions for each input parameter
- A decision table focuses on

  the possible **combinations of input parameters**
- Number of combinations is exponential in the number of parameters
  - further heuristics may be needed for test selection
- Decision table testing better maps to business rules
- Can be combined with equivalence partitioning

Decision Table Testing

Conditions									
0 ≤ quant ≤ 4	✓	✓	✓	x	x	x	x	x	x
5 ≤ quant ≤ 6	x	x	x	✓	✓	✓	x	x	x
7 ≤ quant	x	x	x	x	x	x	✓	✓	✓
0 ≤ price ≤ 19.99	✓	x	x	✓	x	x	✓	x	x
20.00 ≤ price ≤ 99.99	x	✓	x	x	✓	x	x	✓	x
100.00 ≤ price	x	x	✓	x	x	✓	x	x	✓
Actions									
Discount (%)	0	0	0	0	5	5	0	5	7



State Transition Testing

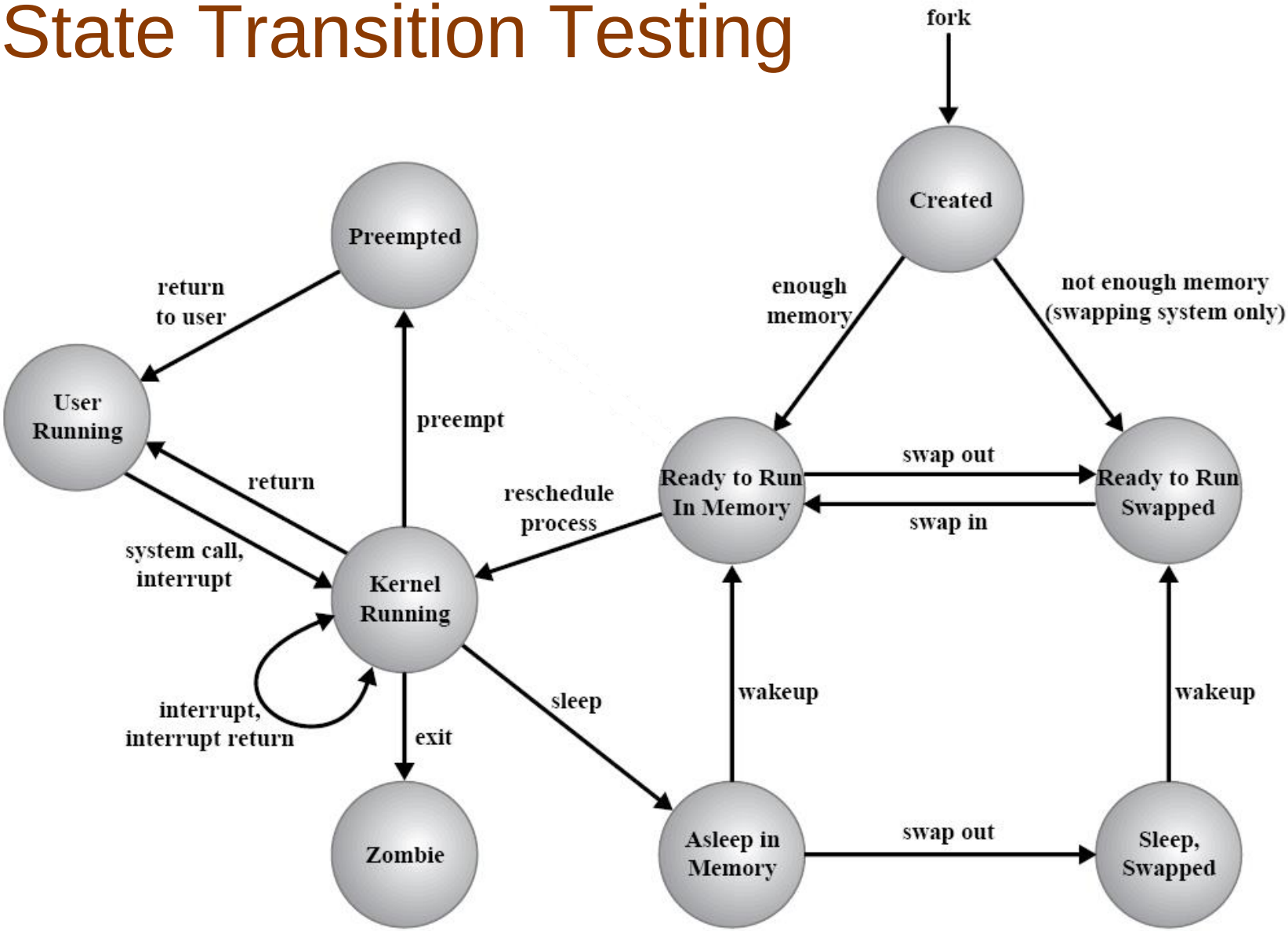
The behavior of a system's functionality may depend on its **state:**

- amount of money inserted → push “coffee” button
- account balance → withdraw money
- user status → login into system
- order state → ship goods
- number entered in GUI → push OK button
- stack size → pop from stack
- filled form fields → submit form
- process state → swap out process
- . . . .

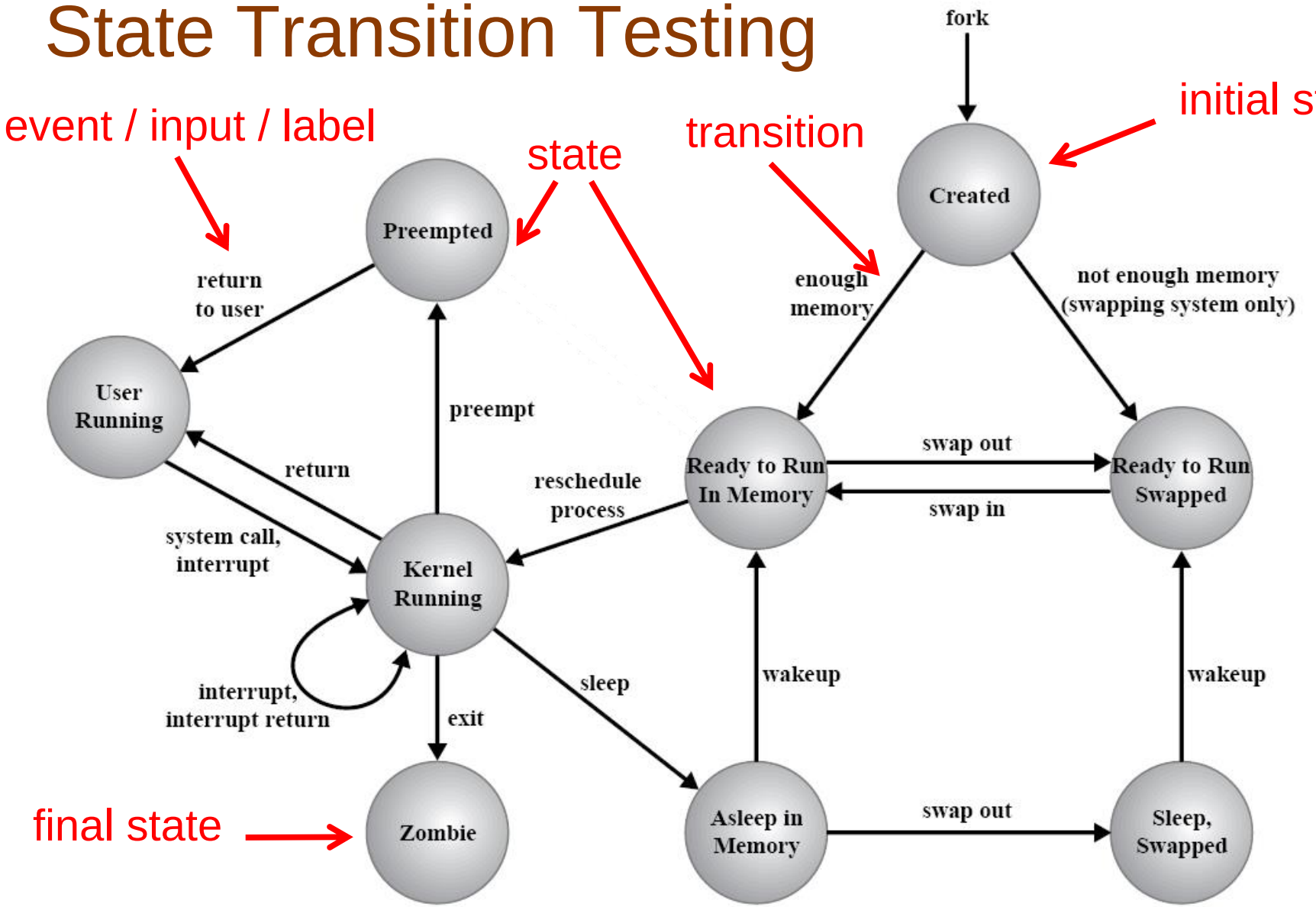
State Transition Testing

- Several kinds of state machines have been defined:
  - **Automata / Formal Languages theory:**
    - Finite Automata (FA, DFA, FSM)
    - Nondeterministic Finite Automata (NFA)
    - Push-Down Automata (PDA)
    - Turing Machine (TM)
    - Labeled Transition System (LTS)
    - Moore Machine
    - Mealy Machine
  - **System Modeling:**
    - State Charts
    - UML State Machines
    - SysML
    - BPMN
    - . . . . .

State Transition Testing



State Transition Testing

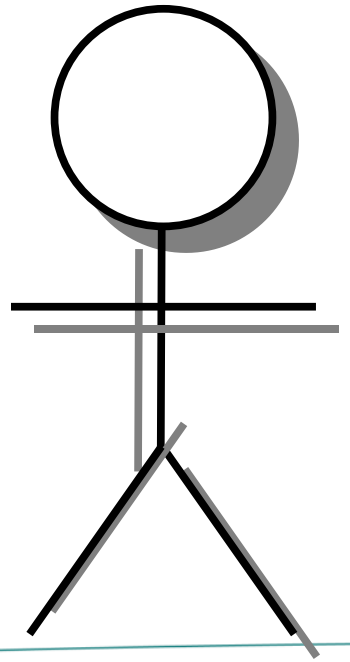


State Transition Testing

- A model is an **abstraction** of the system
- When there is a circular path in the model,
  there are infinitely many paths to be potentially tested, e.g.
  - . . . , (swap out, swap in)*, . . .
- Thus, simpler (finite) heuristics are needed in practice, e.g.:
  - cover each state
  - cover each event
  - cover each transition
  - cover transition pairs, triples, ...
  - test purposes (interesting paths)
  - . . . . .

Use Case Testing

- A **Use Case**
  - captures a **functional requirement** of a system
  - describes the interaction between a primary **actor** and the system
  - is a sequence of simple steps
  - consists of:
    - preconditions and postconditions
    - main scenarios and alternative scenarios



Use Case Testing

Use Case ***Write and Send Email***

Main Scenario:
Precondition: none

1. Click “Compose”
2. Enter single valid recipient
3. Enter subject
4. Enter body text
5. Click “Send”

Postcondition: email is sent to recipient and stored in *Sent*-folder

Alternative Scenarios:
several recipients, empty subject, CCs, BCCs, invalid recipient, ...

Use Case Testing

- Use Cases are
  - close to business scenarios
  - most suited for system- and acceptance test levels
  - can uncover defects in the process flow
  - can spot integration defects

- **Principle approach**
  - test the main scenario and all alternative scenarios

- **Alternative:** BDD (*Behaviour Driven Development*)

  GIVEN . . .
  WHEN . . .
  THEN . . .

White Box Testing

- ISTQB glossary:

  white-box test design technique
  (structural test design technique)

- *A procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.*

More Coverage...

- Function coverage
- Call coverage
- Linear Code Sequence and Jump (LCSAJ) coverage
- Data flow coverage
- Object code branch coverage
- Loop coverage
- Race Coverage
- Relation operator coverage
- Weak mutation Coverage
- Table coverage
- . . . . .

Structural Test Case Design

- Structural test case design asks questions like:
  - how can I reach this part of the code?
  - how can I make this expression true?
- Many interesting properties of programs are hard, or even **undecidable**, for instance reachability of code
- Structural test case design is in general a difficult task
- A hot research topic
  (symbolic testing, concolic testing, software model checking, ...)