MODEL BASED TEST DESIGN AT UNITY

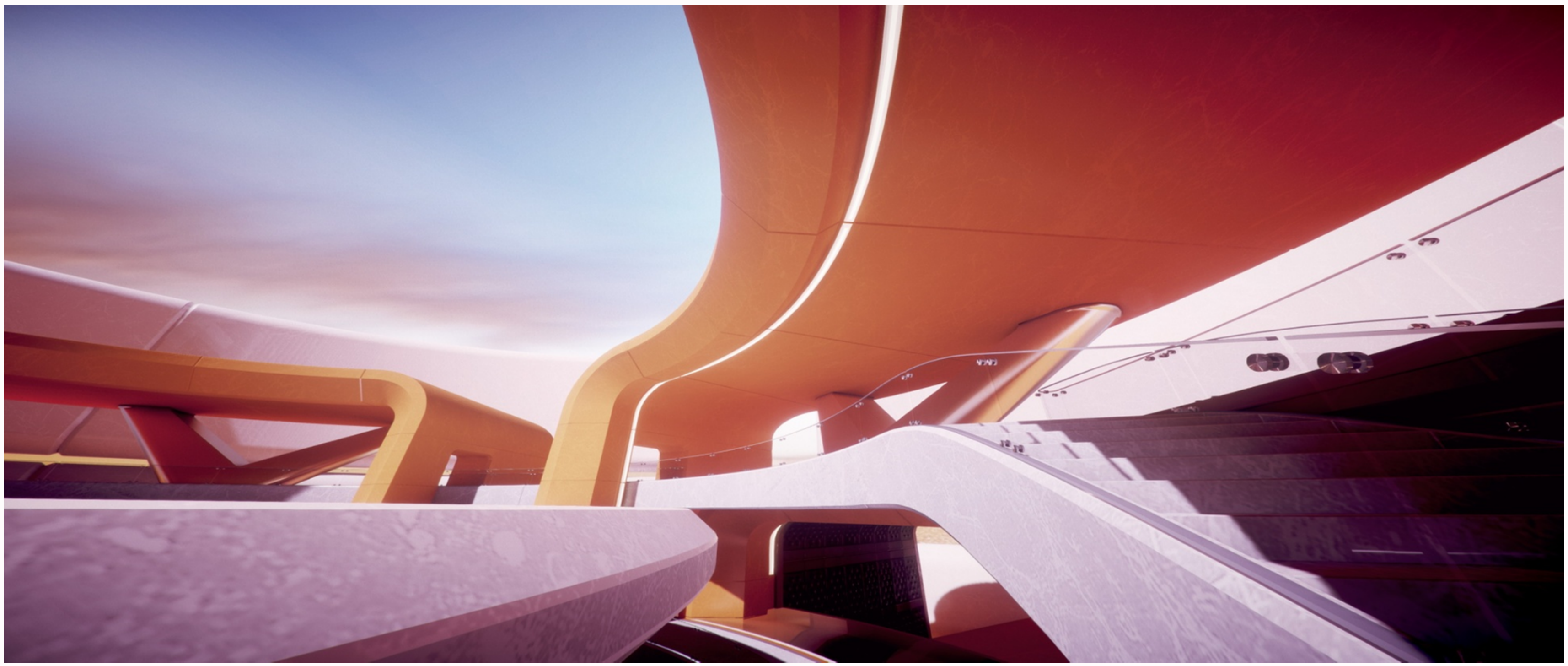
Marek Turski, Ilya Turshatov, Tomasz Paszek

Unity Technologies

© All rights reserved

Unity Technologies

Provider of an integrated development environment for creating games and other interactive virtual content



© All rights reserved

Unity Engine

- A few statistics
  - 100 core product developers
  - 1 million monthly active developers
  - 45k unique titles made with Unity every month
  - 9 million new devices reached every day
  - 20 million new install events per day

© All rights reserved

Unity Engine

- Scaling is challenging
  - Growing team size
  - 23 supported platforms
  - Integrated services
    - Player retention
    - Cloud deployment
    - Multiplayer
    - Asset Store
    - Analytics
    - Ads

© All rights reserved

QA Focus

- Manual Testing (Test Engineers)
- User Experience (UX Researchers)
- User Support
  - Student Workers
  - Support Engineers

© All rights reserved

QA Focus

- Test Automation (Test Developers)
- Test Infrastructure (Toolsmiths)
  - Test runners and frameworks
  - Bug reporting and customer support tools
  - Backend and Reporting
  - Working closely with the Build Infrastructure Team

© All rights reserved

Test Automation Focus

- Unit Tests
  - Code tests, written by developers
- System Integration Tests
  - Components seen as processes
- Sub-System Runtime Tests
  - Components seen as interfaces
- Test Tools
  - Built directly into the Game Engine

© All rights reserved

Test Automation Focus

- Non-Functional Tests
  - Performance/Stress/Load
  - Deployment/Update/Security
  - User Interface
  - Asset Import
  - Graphics

© All rights reserved

Towards Agile Development

Individuals and interactions over Processes and tools
Working software over Comprehensive documentation
Customer collaboration over Contract negotiation
Responding to change over Following a plan

Is Unity agile (enough)?

© All rights reserved

Towards Agile Testing

- Re-evaluation of skills
  - Planning
  - Communication
  - Quality Assistance over Quality Assurance
- Toolbox Clean-up
  - Concepts
  - Techniques
  - Testware

© All rights reserved

Good Requirements Specification

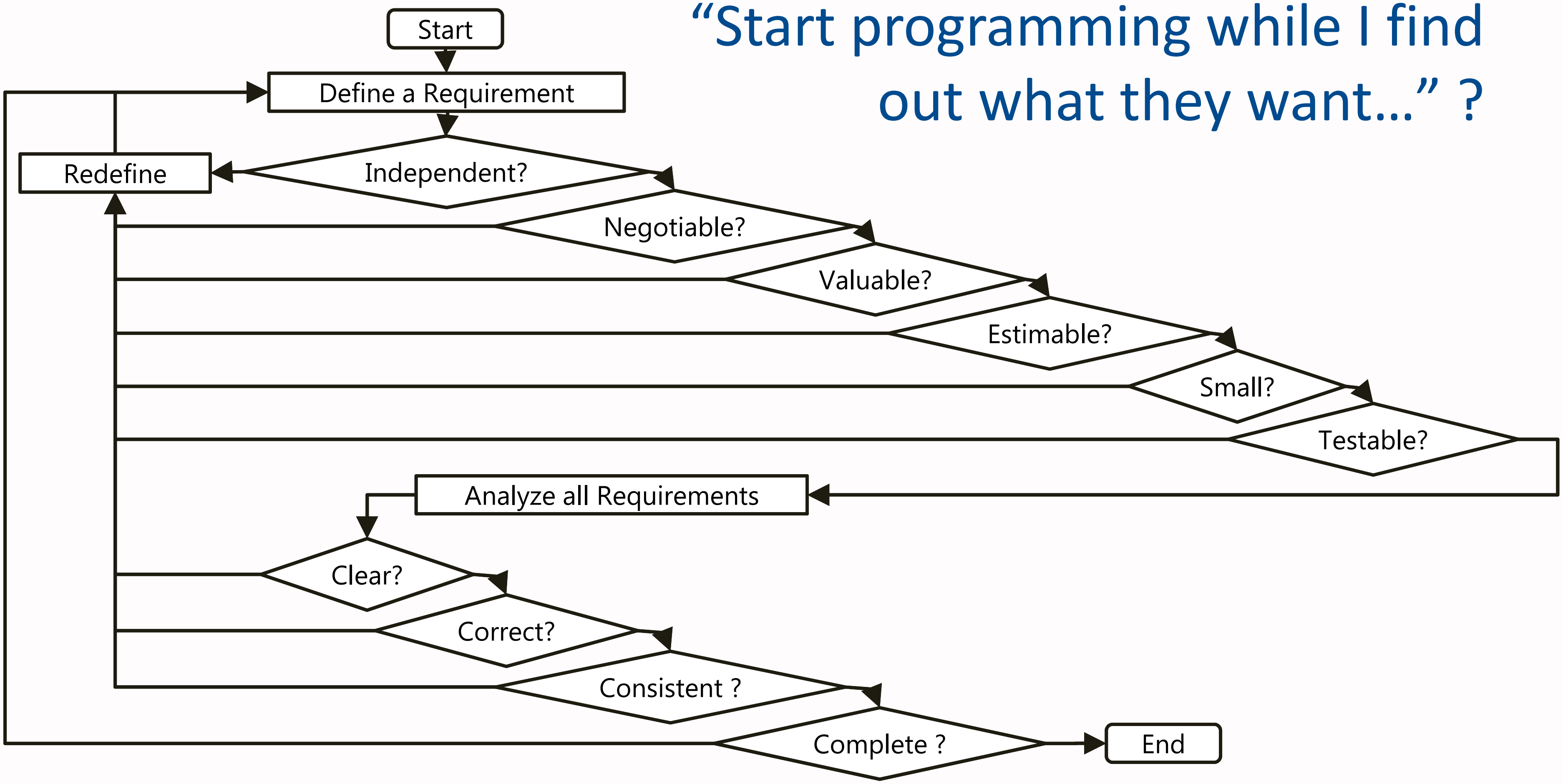
- Traditional guidelines
  - Clarity, Completeness, Correctness, Consistency
- Agile User Stories
  - Small and Independent
  - Estimable and Testable
  - Valuable and Negotiable

As a [*user role*] I want to [*desired feature*] so that [*value/benefit*].

© All rights reserved

Good Requirements Specification

"Start programming while I find out what they want…" ?



© All rights reserved

Model-Based Requirements

- Model-Based Testing
  - Derivation of test cases from a model of the desired system behaviour
- Why use models?
  - The modelling process improves our understanding of the system under test and finds inconsistencies earlier
  - Models become collaborative (and negotiable) test artifacts which visualize and document our view of the system

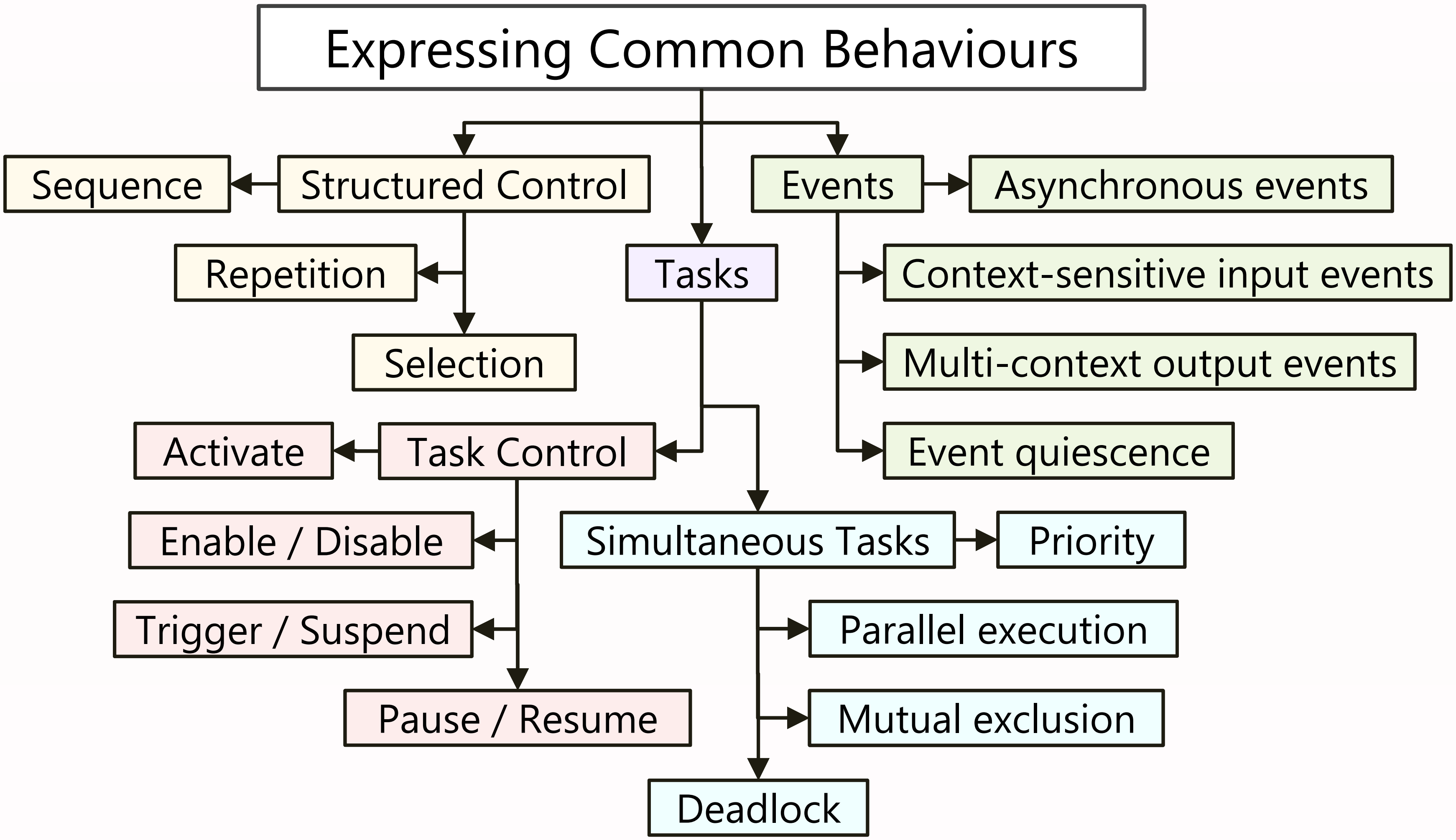
© All rights reserved

Model-Based Requirements

- Properties of a good model-based notation
  - Expressive
    - Represents common process abstractions and control issues
  - Provocative
    - Helps with or drives the discovery of system aspects
  - Processable
    - Can be data-mined or executed
  - Scalable
    - Provides means of functional or data decomposition

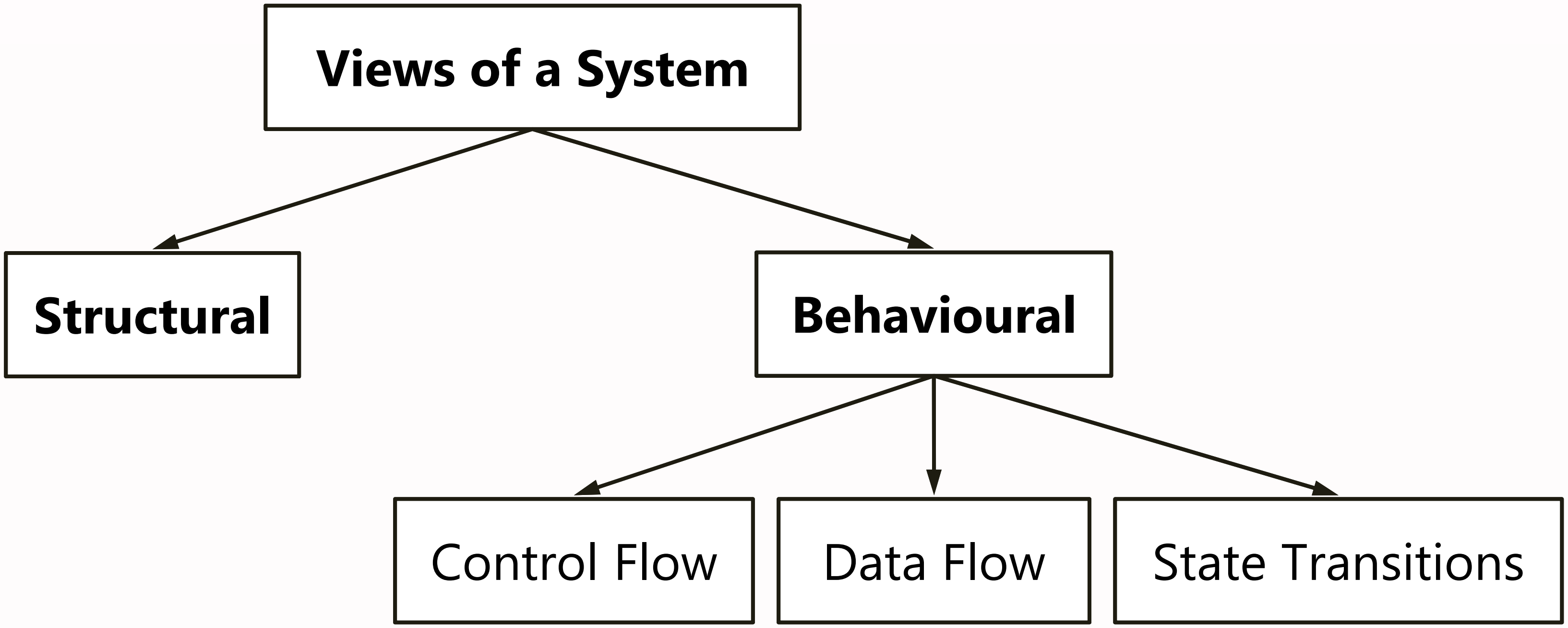
© All rights reserved

Model-Based Requirements



© All rights reserved

Modelling System Requirements

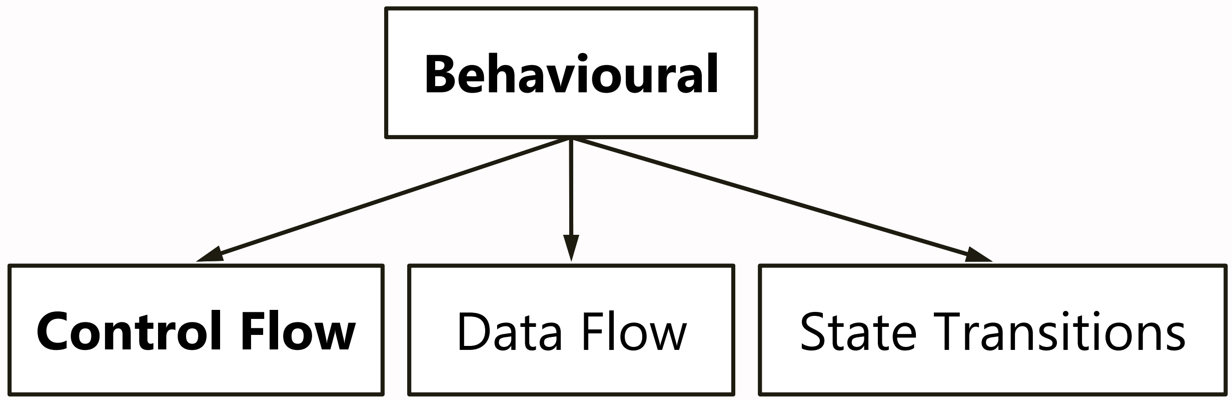


Structural modelling focuses on what the system is.

Behavioural modelling focuses on what the system does.

© All rights reserved

Control Flow Modelling



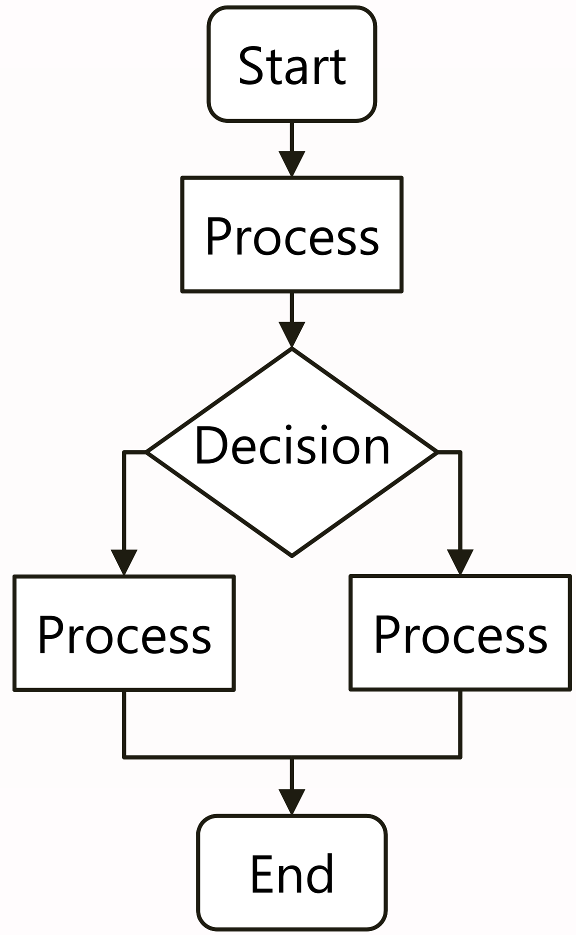
Emphasizes	The exact sequence of steps
De-emphasizes	How inputs to a state are determined
Flow	Control stream - the next step is taken when a previous step (or sequence of steps) has finished

© All rights reserved

Flowcharts

- One of the earliest behavioural models
- Illustrates algorithms, processes and workflows
- Multitude of styles and symbols

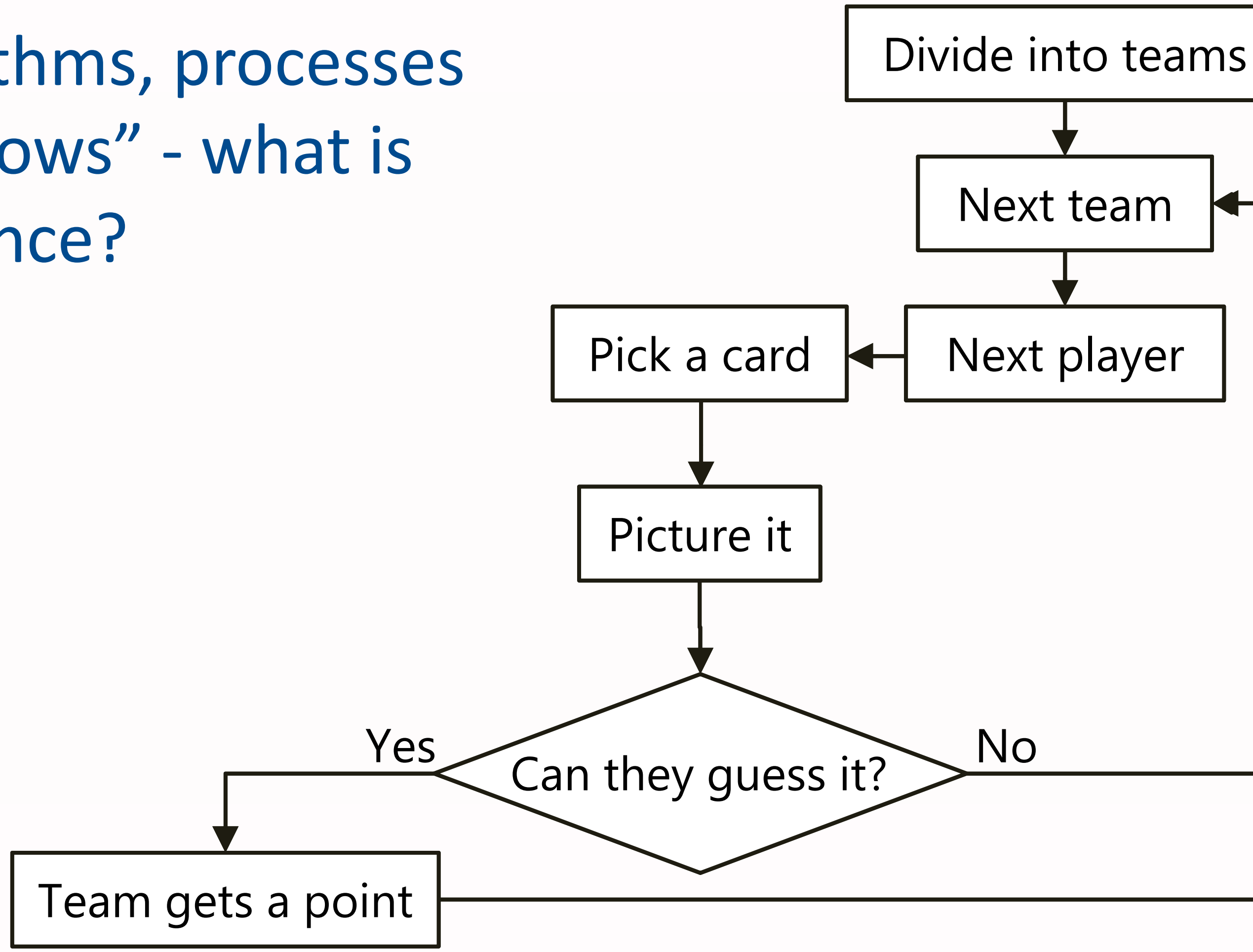
Emphasizes	The exact sequence of steps
De-emphasizes	How inputs to a state are determined
Flow	Control stream - the next step is taken when a previous step (or sequence of steps) has finished



© All rights reserved

Flowcharts

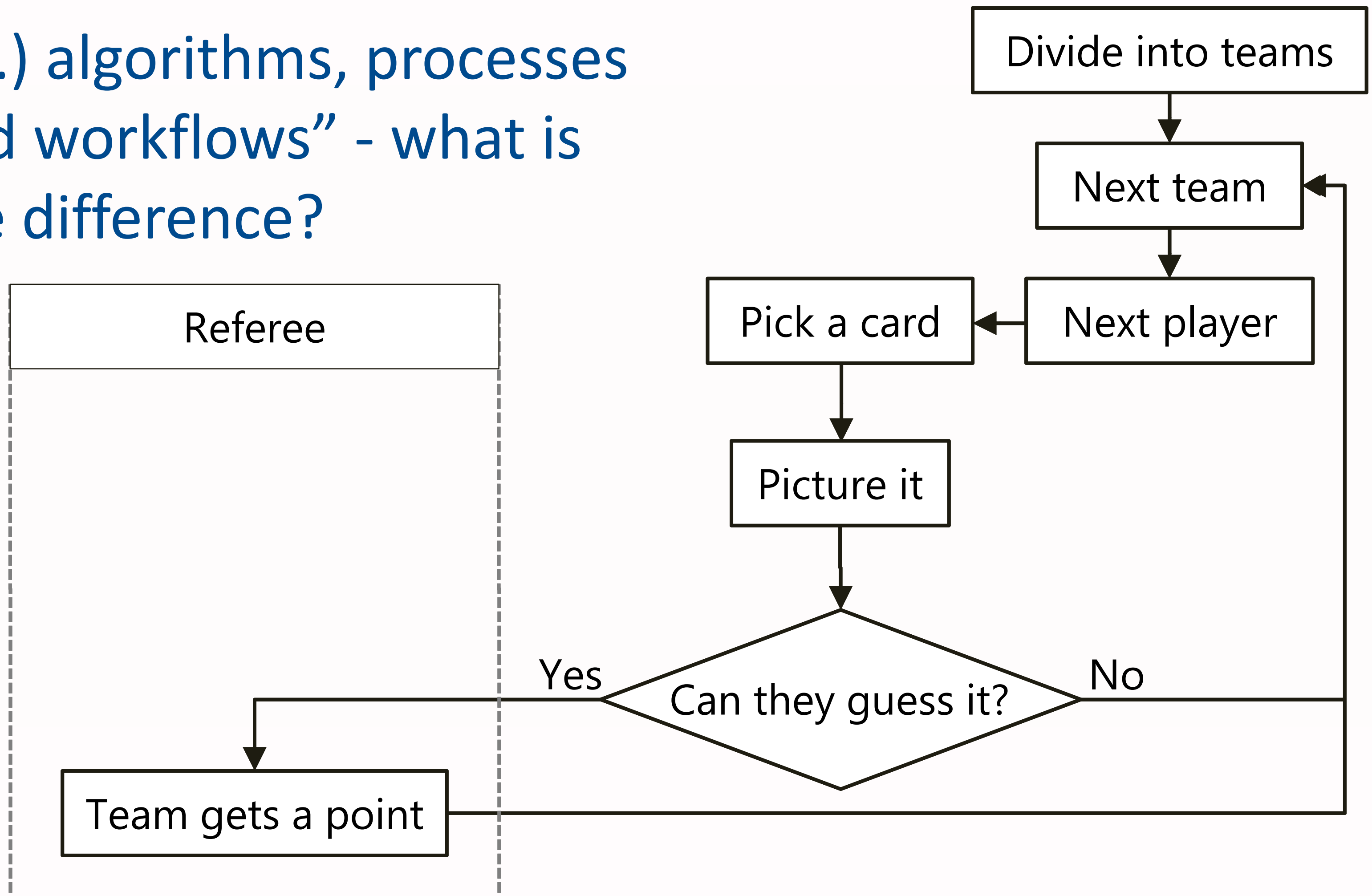
- “(…) algorithms, processes and workflows” - what is the difference?



© All rights reserved

Flowcharts

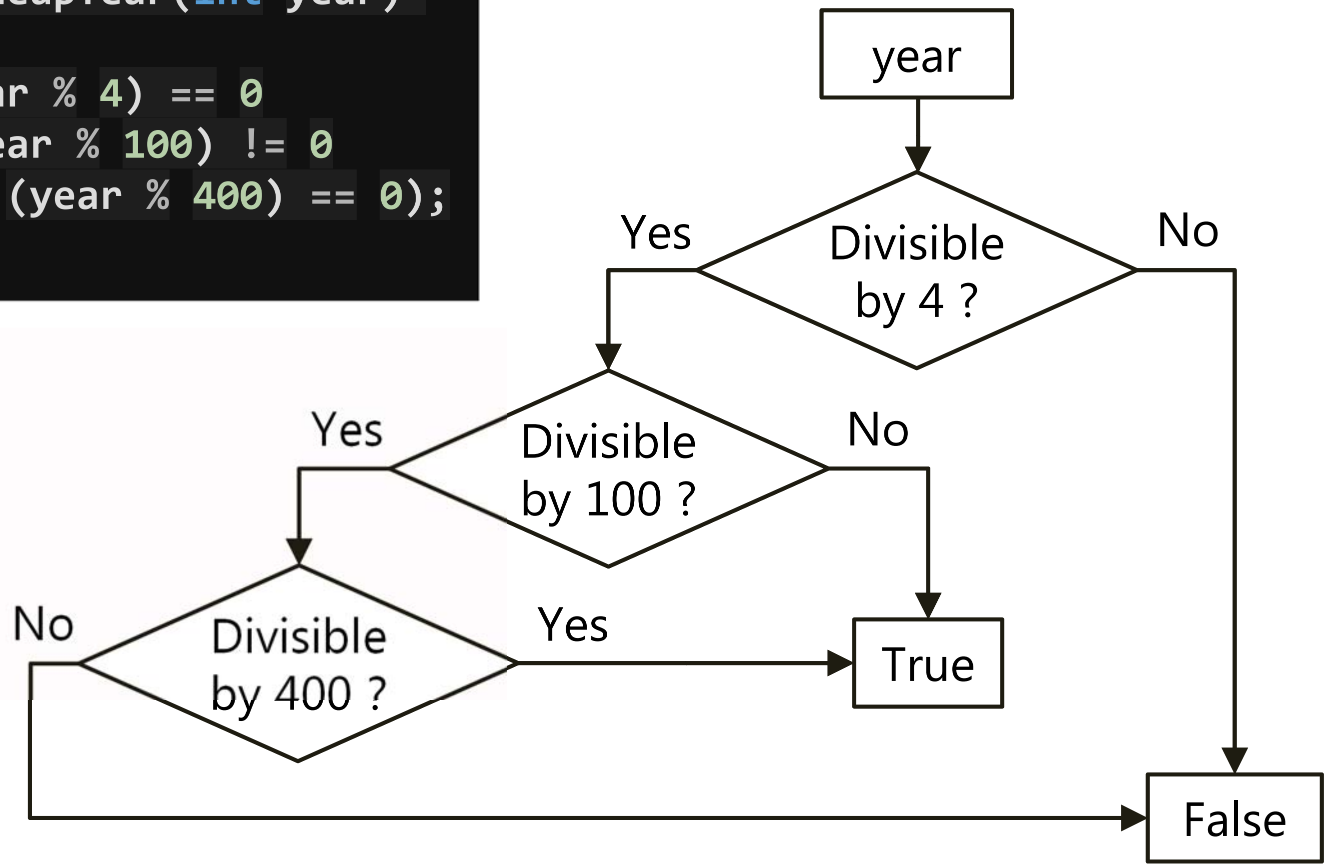
- "(...) algorithms, processes and workflows" - what is the difference?



© All rights reserved

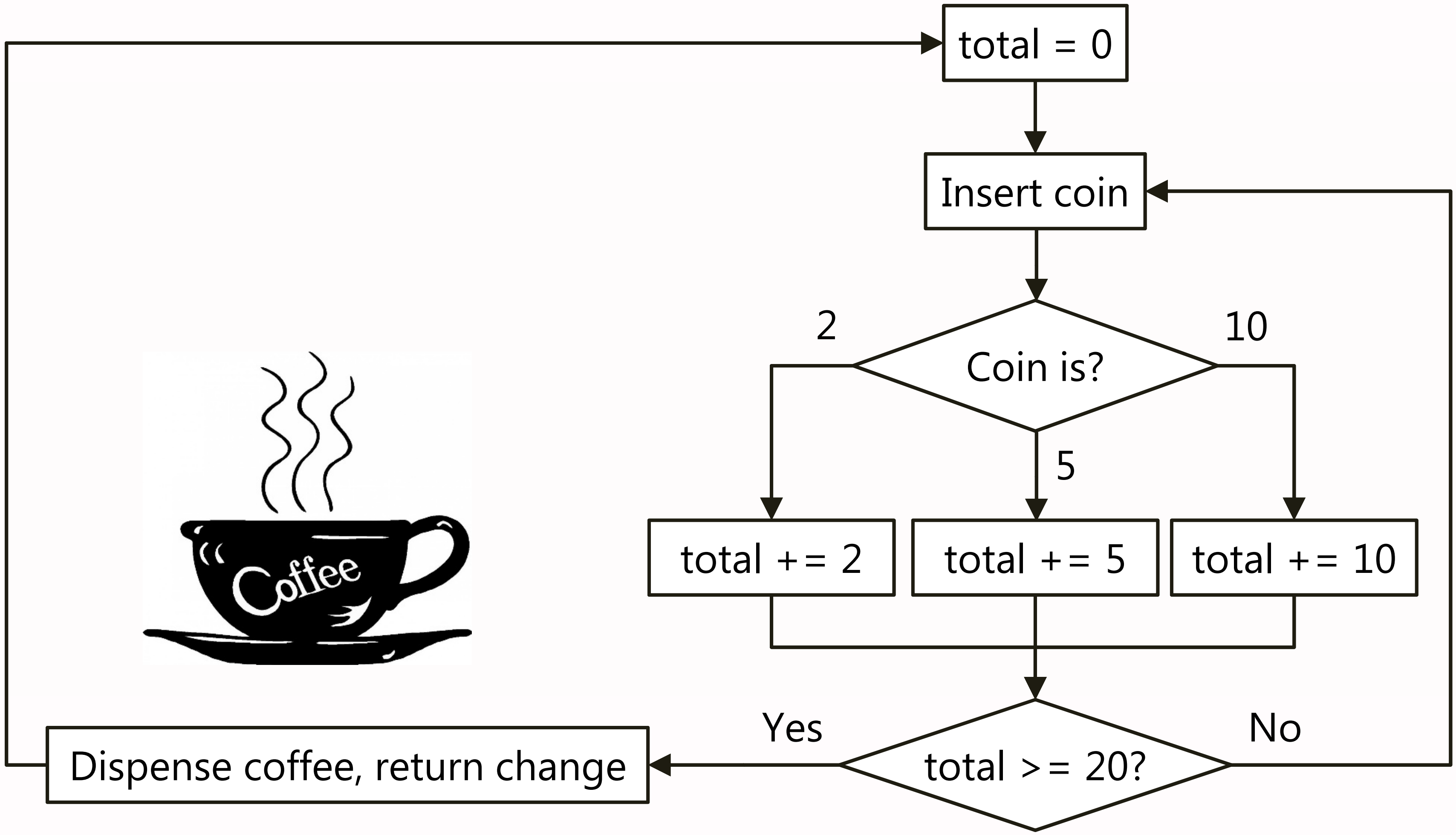
Leap Year Problem flowchart

```
static bool IsLeapYear(int year)
{
    return (year % 4) == 0
        && ((year % 100) != 0
            || (year % 400) == 0);
}
```



© All rights reserved

Coffee Machine Problem flowchart

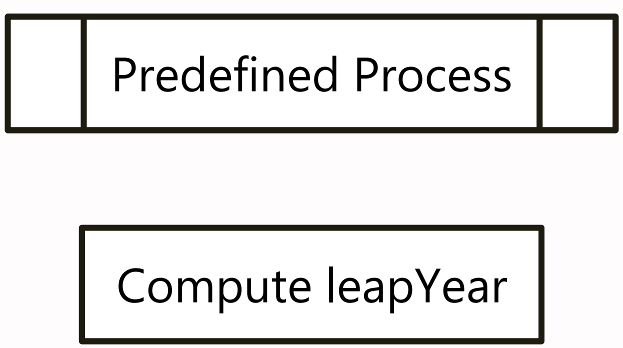


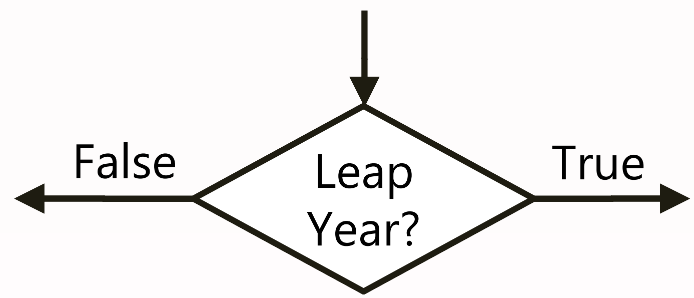
© All rights reserved

Flowcharts – Decomposition

- Layout Decomposition
  - On-page connectors
  - Off-page connectors

- Hierarchy Decomposition





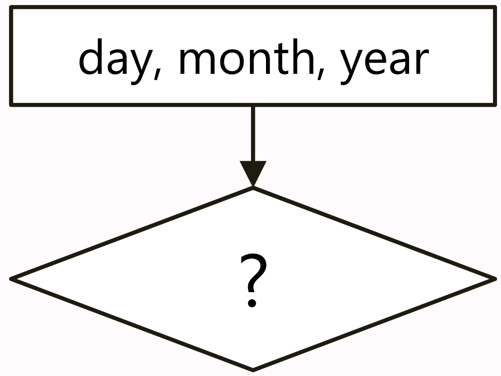
© All rights reserved

Flowcharts – Getting Started

- How to start?
  - Define the process to be illustrated
  - Find a trigger event or a trigger input
  - List known actions, try to keep the order chronological
- Extending the model
  - What could/should happen next?
  - Are all important aspects of the process illustrated?
  - Are all relevant actions taken into account?

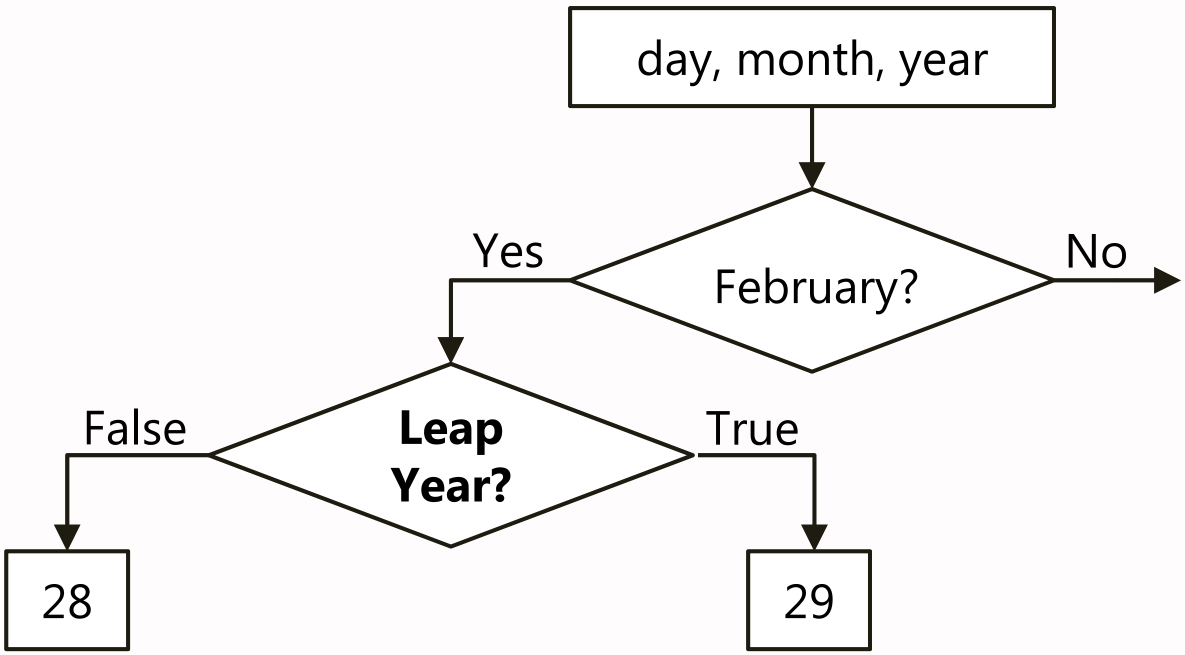
© All rights reserved

Days in a Month flowchart



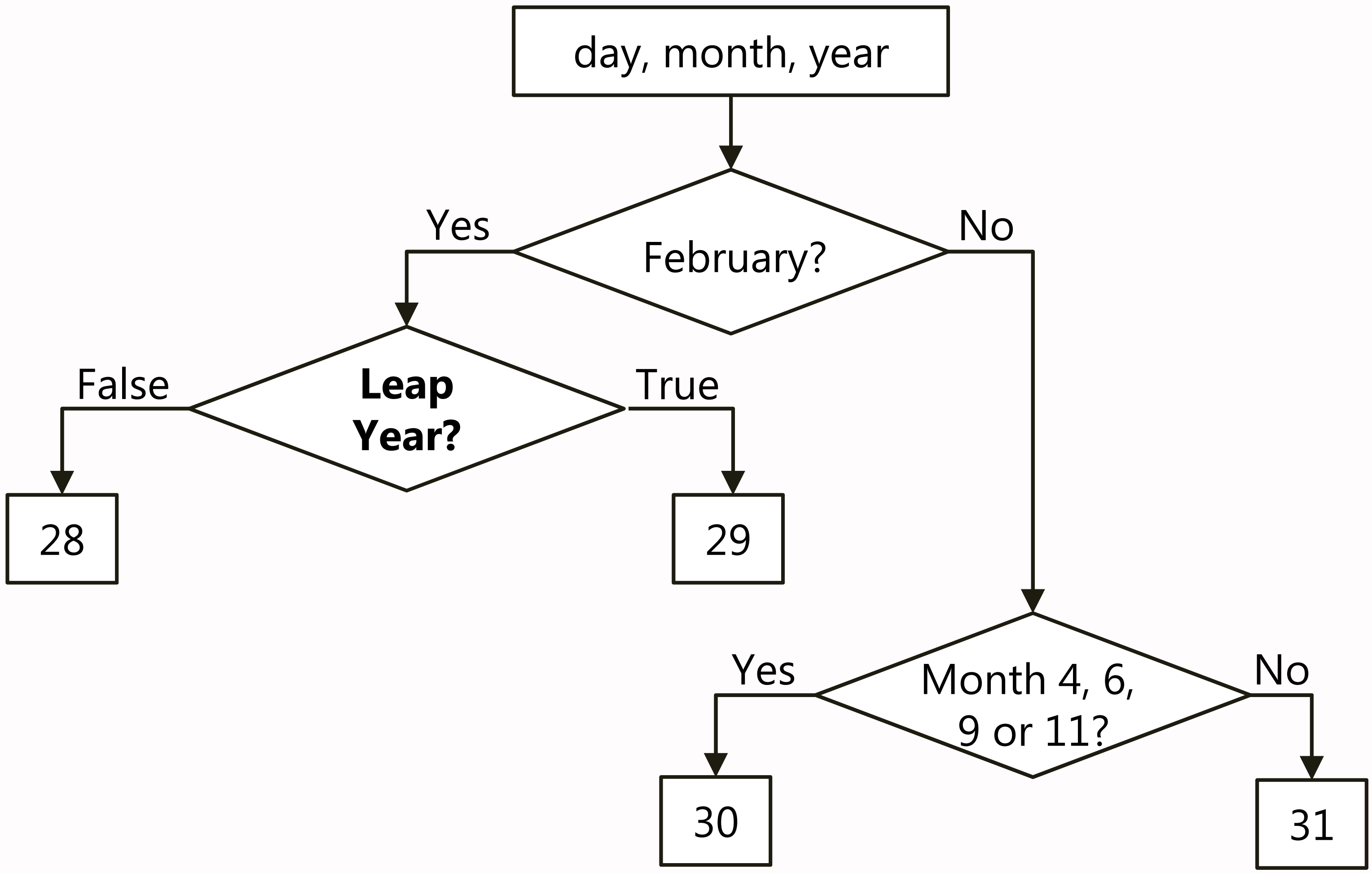
© All rights reserved

Days in a Month flowchart



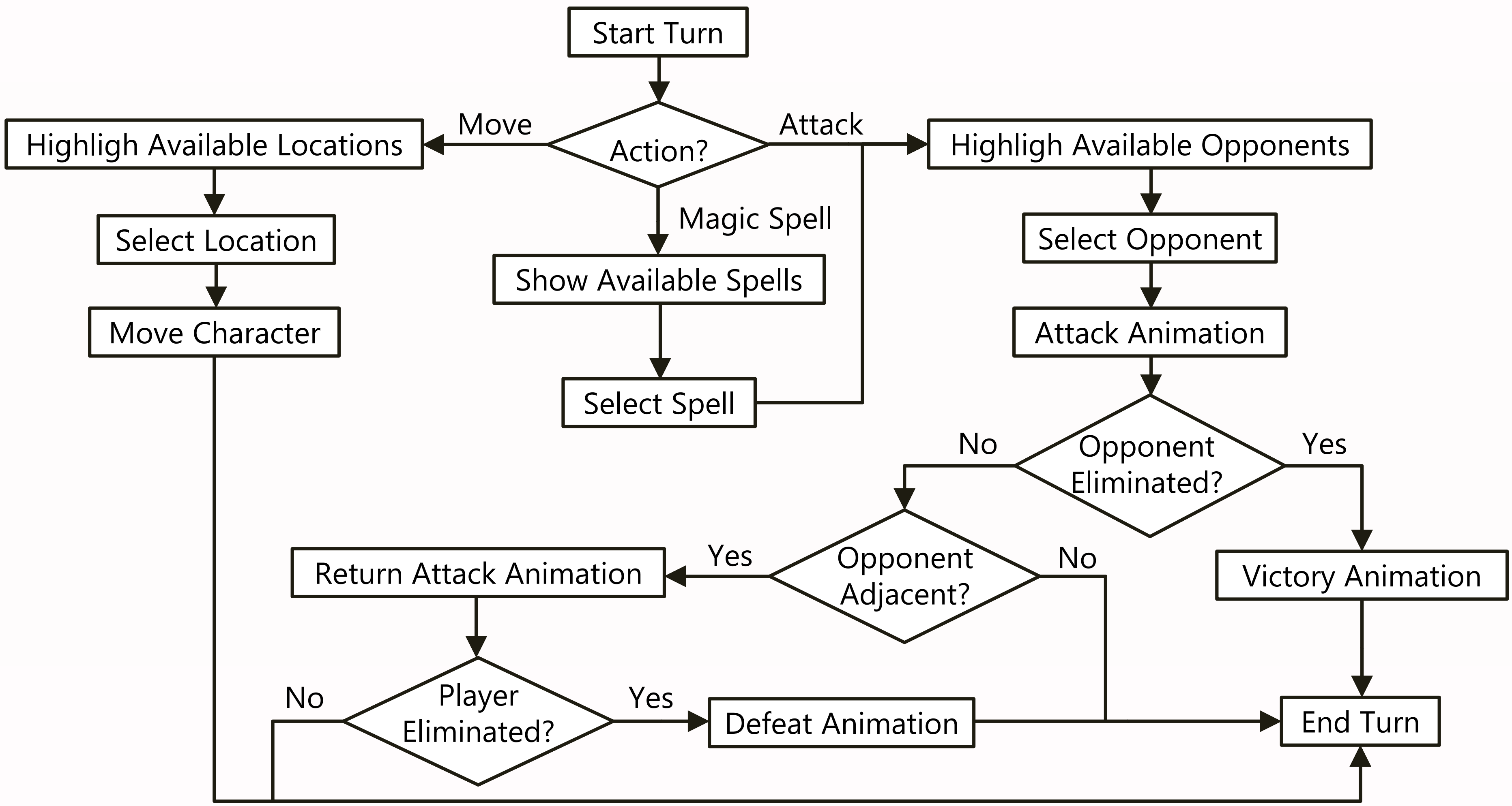
© All rights reserved

Days in a Month flowchart



© All rights reserved

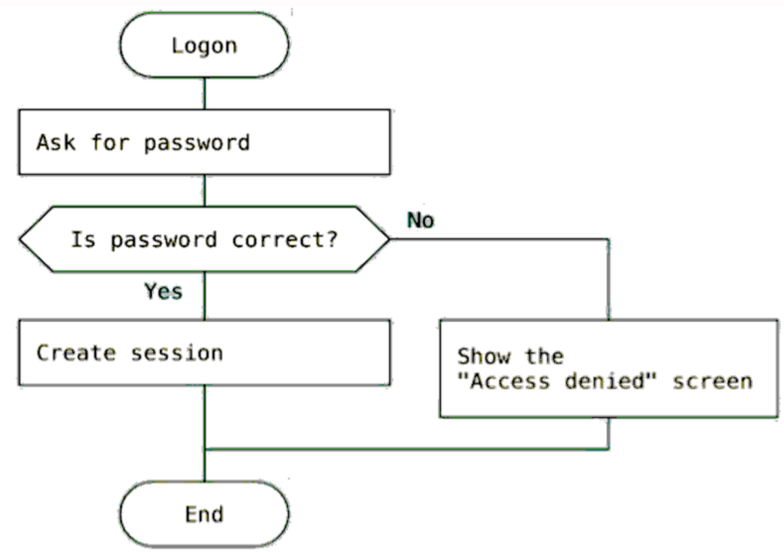
Turn-based game flowchart



© All rights reserved

Flowcharts – Processability

- Well formed flowcharts can be data-mined by business workflow engines
- An executable variation of flowcharts is used by the DRAKON Visual Language
  - Automatic layout
    - Vertical only
    - Unified distances and offsets
    - Logical alignment
  - Silhouette design style
  - Joins and parallel actions

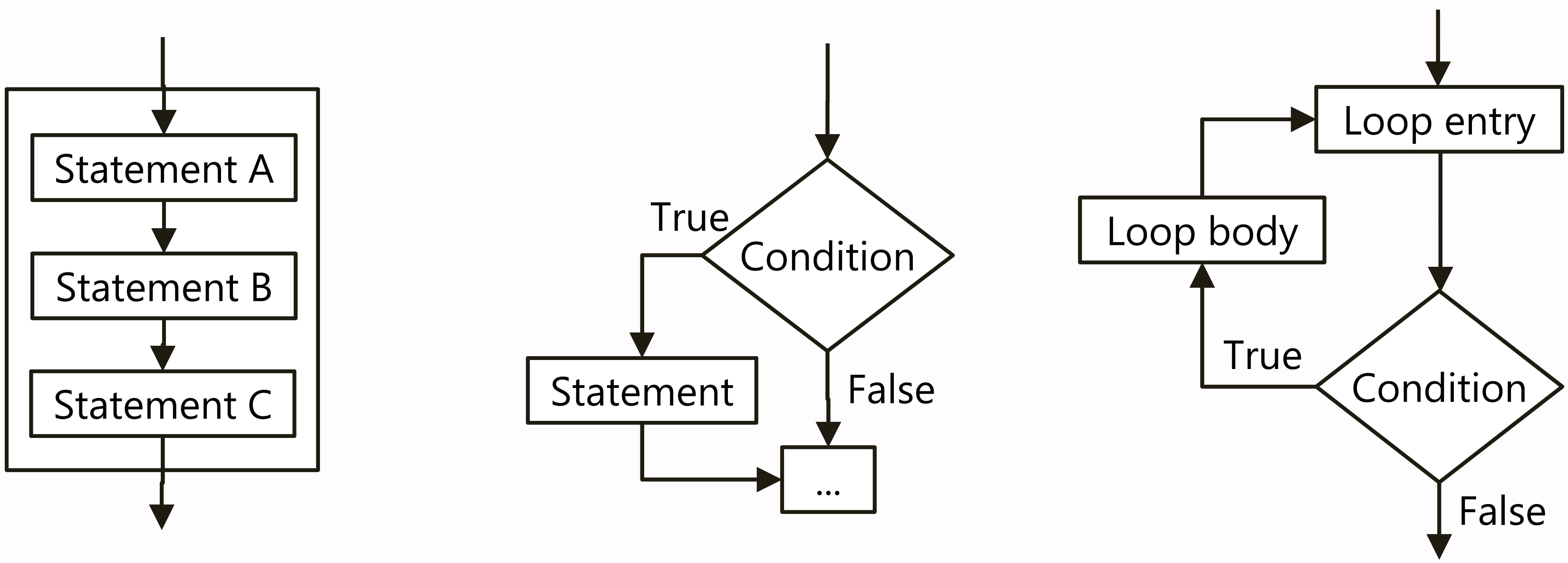


20-22/10/2015
© All rights reserved
User Conference on Advanced Automated Testing
unity

Flowcharts – Expressive power

Structured Control Statements are the core building blocks of a flowchart

- Perfect for representing imperative languages

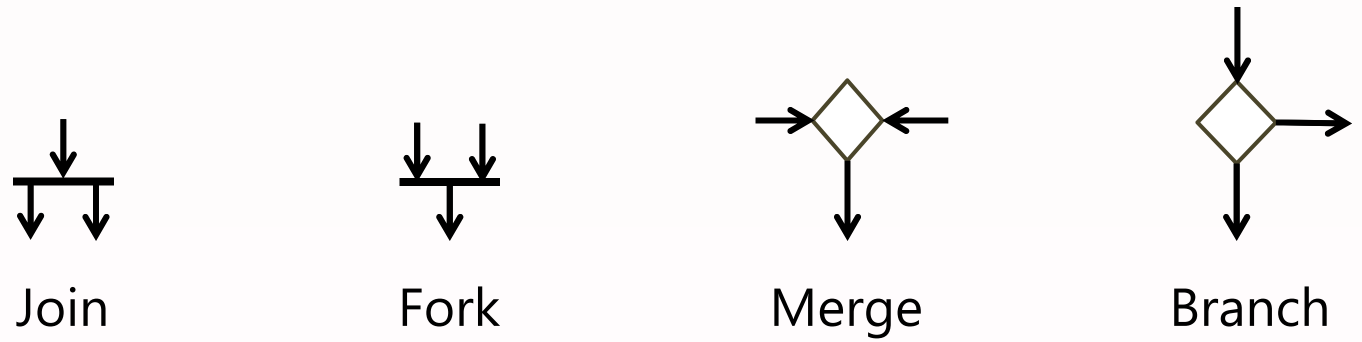


Sequence Selection Repetition

© All rights reserved

Flowcharts – Expressive power

- Other aspects of behaviour must be determined in text or deducted from the flow
  - Mutual exclusion is represented through parallel paths after a decision, but other concurrency concepts are missing
  - A popular extension - Activity Diagrams – addresses this problem by introducing new symbols



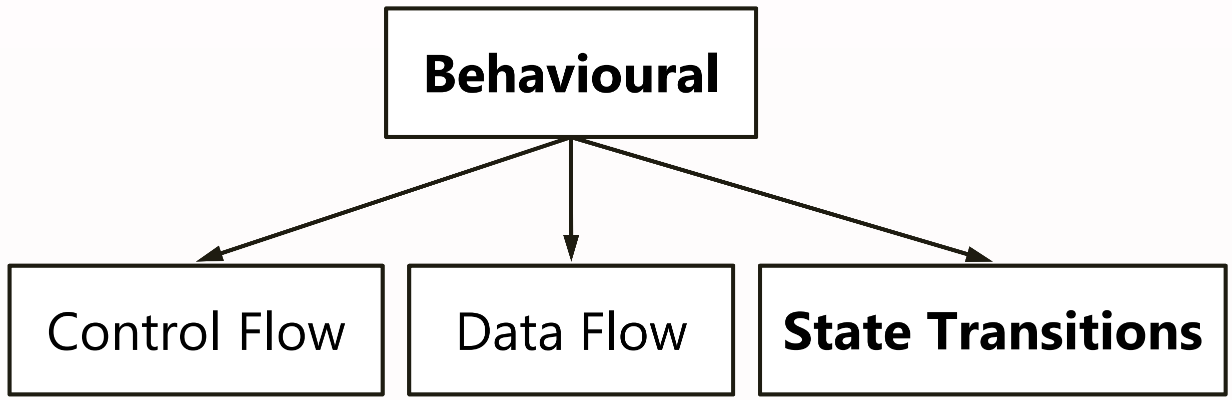
© All rights reserved

Break Time

- 5 minute break
- Q&A
- Flowchart design exercise (optional)

20-22/10/2015
© All rights reserved
User Conference
on Advanced Automated Testing
unity

State Transition Modelling



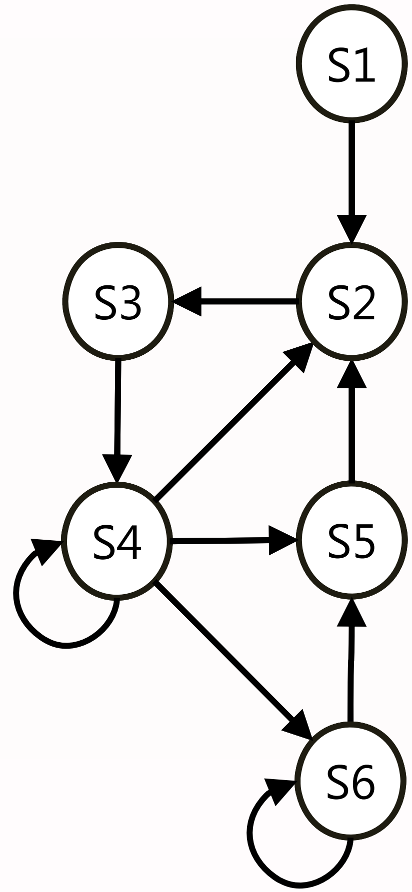
Emphasizes	The state space and state transition stimuli
De-emphasizes	Sequencing of steps
Flow	Event stream – the next step is taken in response to events and the internal state of the system

© All rights reserved

Finite State Machines

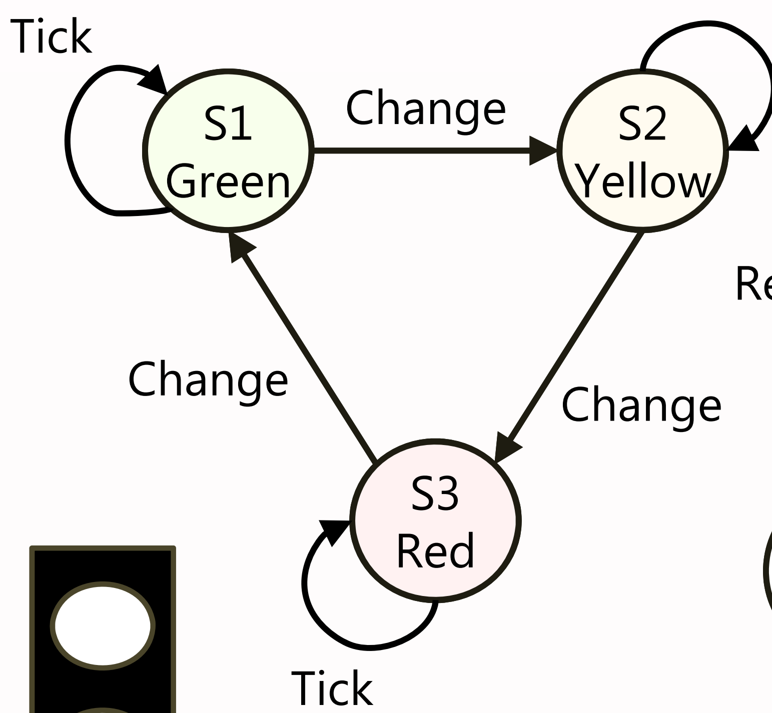
- A mathematical model of computation (a finite automata)
- Illustrate behaviour as a series of events that can occur in one or more possible states of a system

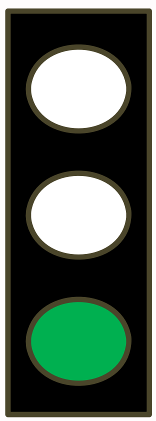
Emphasizes	The state space and state transition stimuli
De-emphasizes	Sequencing of steps
Flow	Event stream – the next step is taken in response to events and the internal state of the system

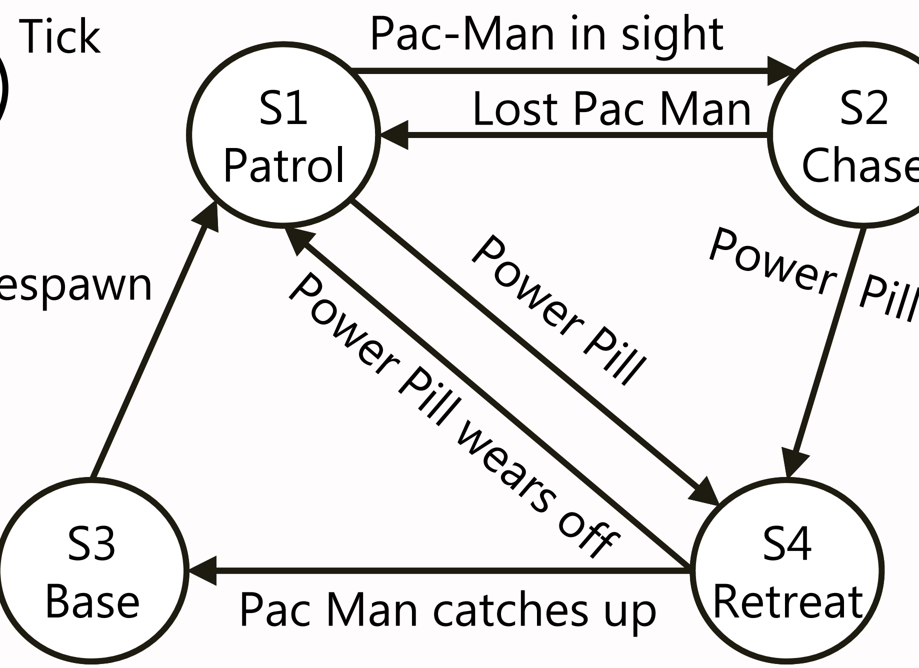


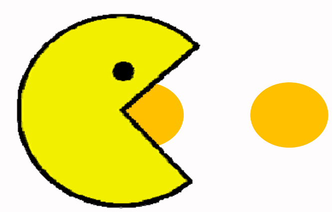
© All rights reserved

Common Finite State Machines







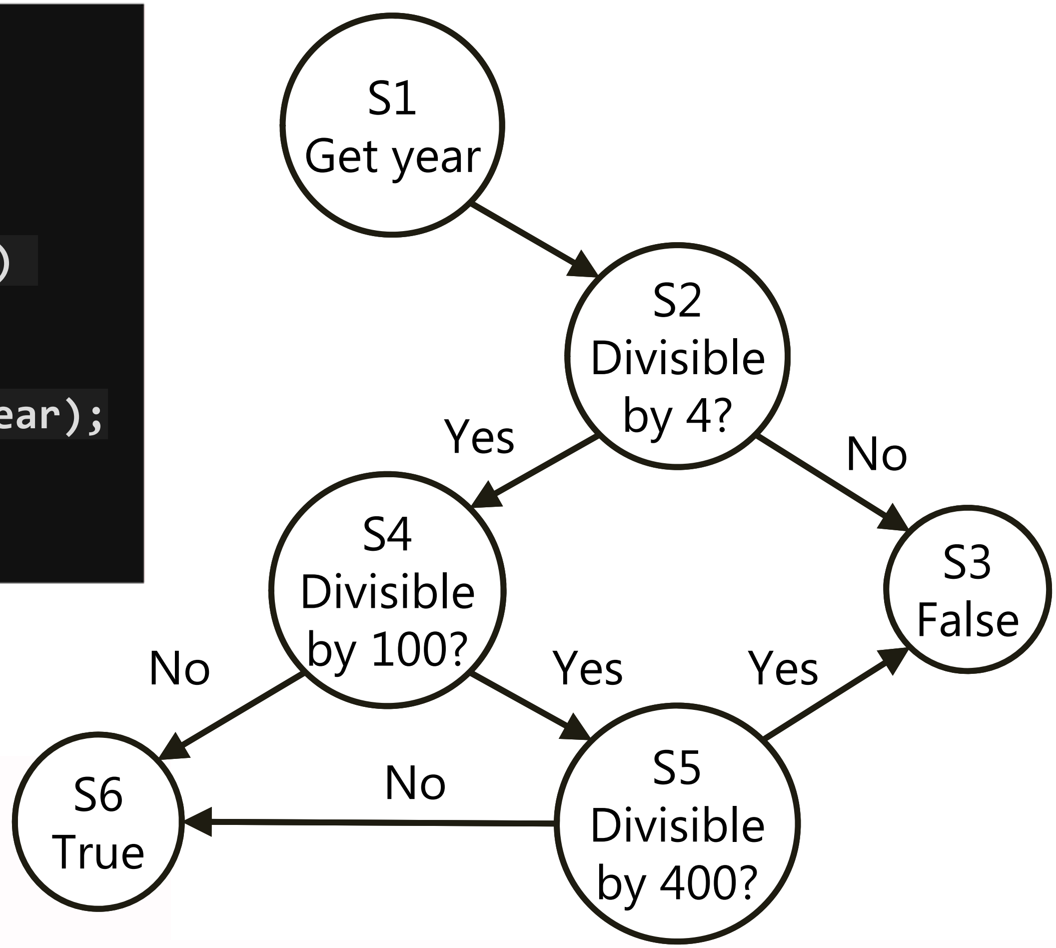


© All rights reserved

Leap Year Problem FSM

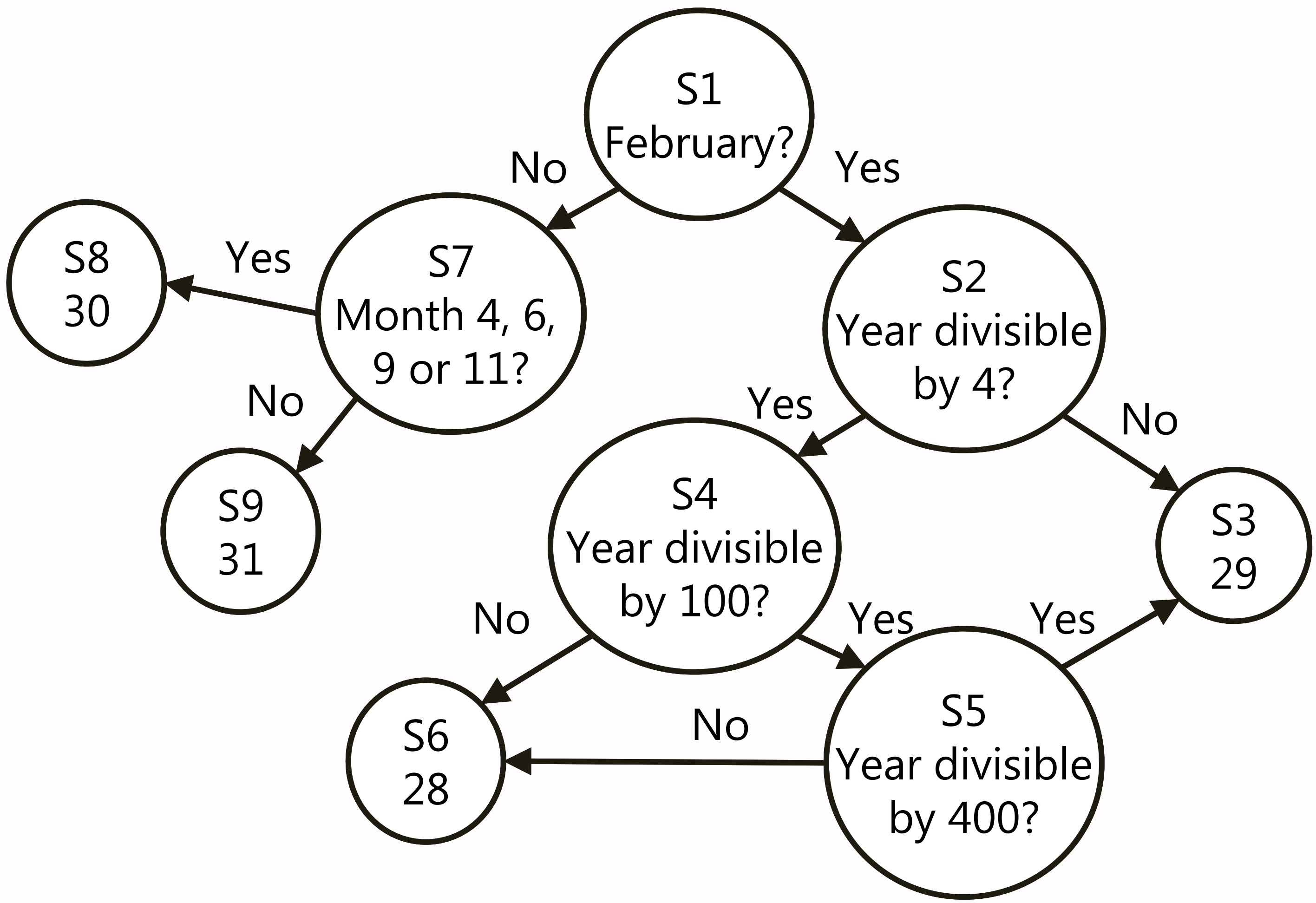
```
class Year
{
    bool m_IsLeapYear;

    public Year(int year)
    {
        m_IsLeapYear =
            IsLeapYear(year);
    }
}
```



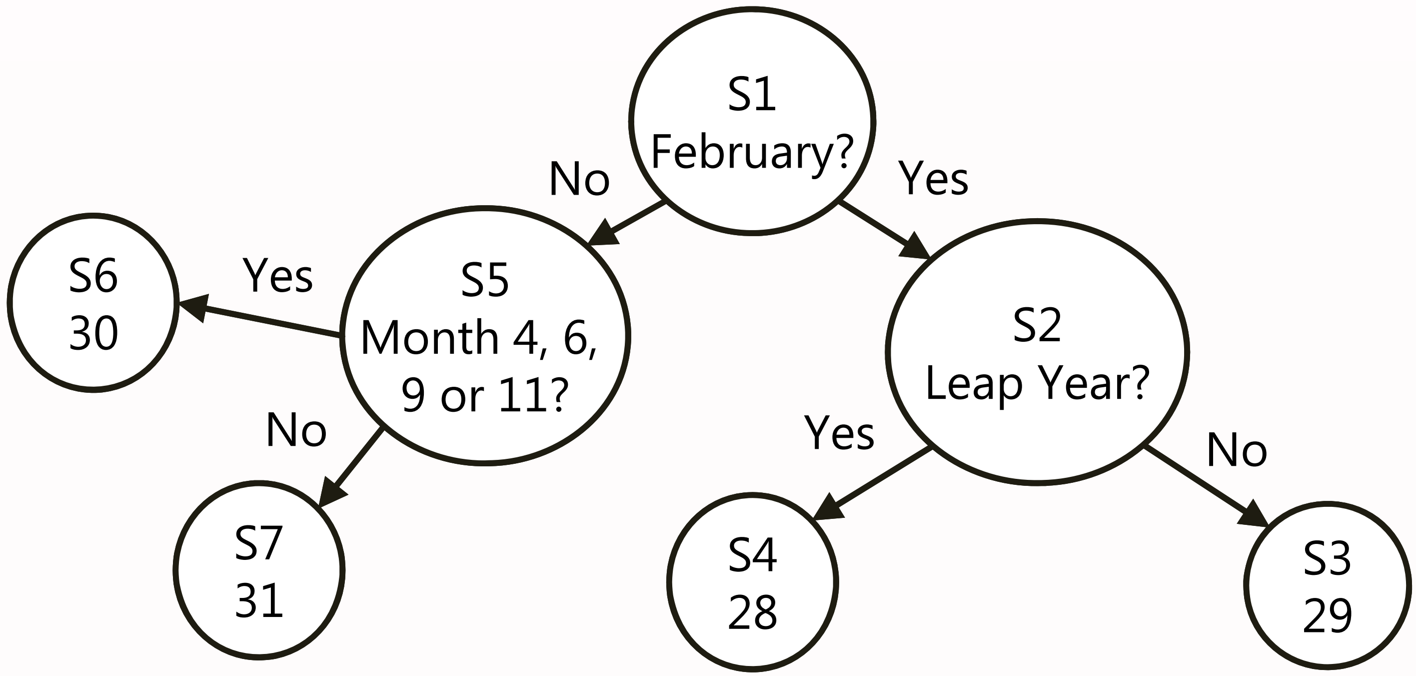
© All rights reserved

Day of Month Problem FSM



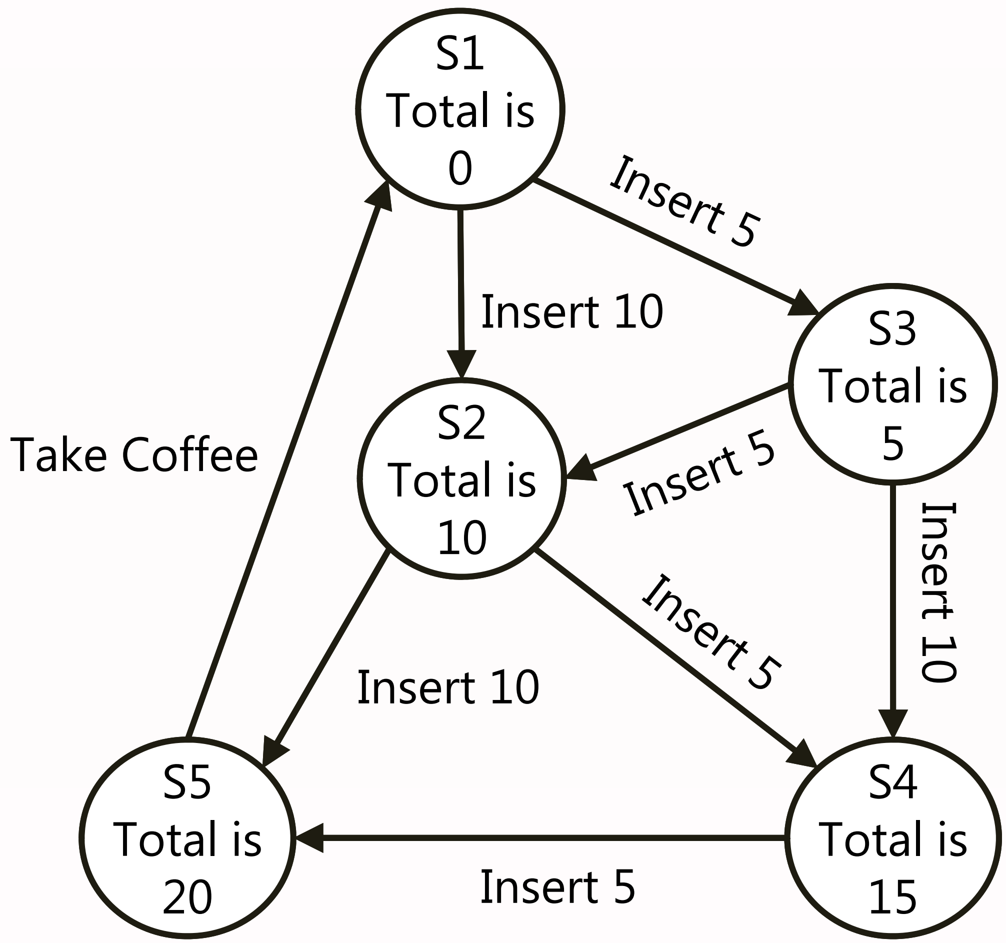
© All rights reserved

Day of Month Problem FSM



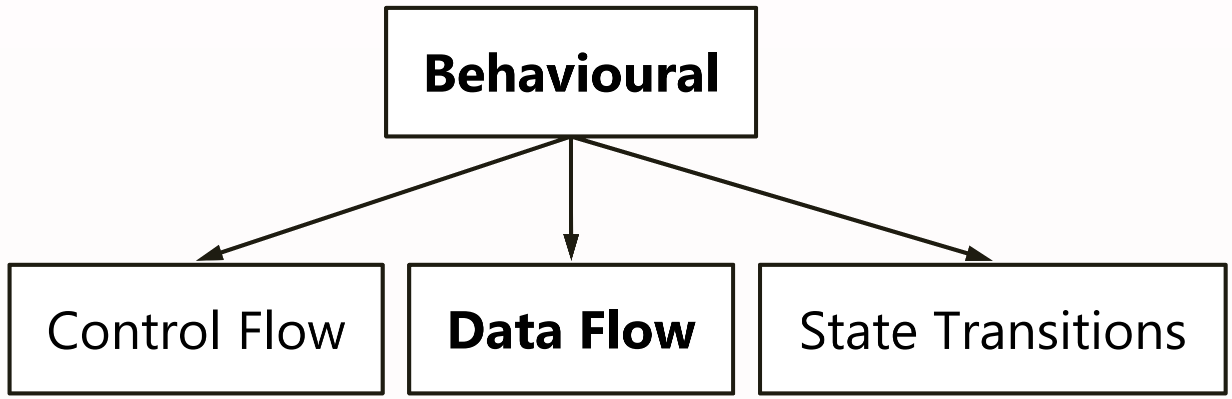
© All rights reserved

Coffee Machine Problem FSM



© All rights reserved

Data Flow Modelling



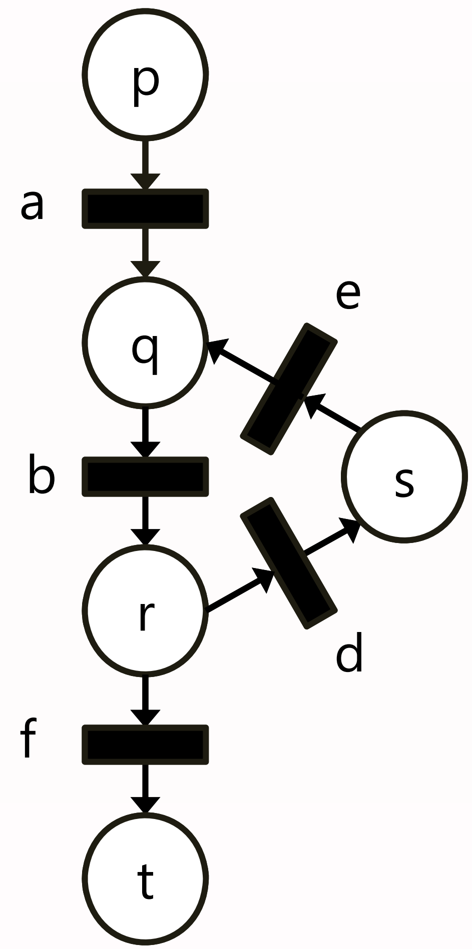
Emphasizes	Exchange and transformation of data
De-emphasizes	Sequencing of steps
Flow	Data stream - the next step is taken when other steps provide its inputs

© All rights reserved

Petri nets

- Graphical notation for step-wise process analysis
- A mathematical modelling language for the description of distributed systems supported by a well developed theory for process analysis

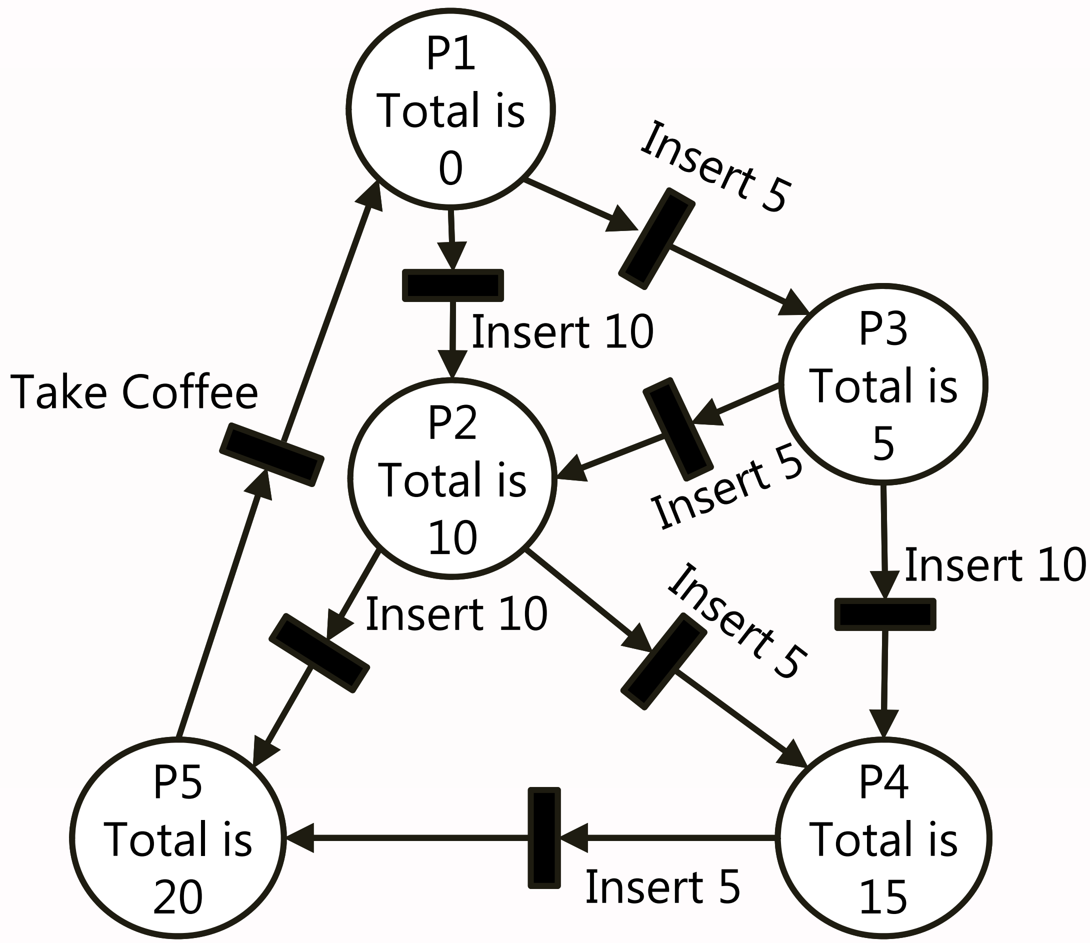
Emphasizes	Exchange and transformation of data
De-emphasizes	Sequencing of steps
Flow	Data stream - the next step is taken when other steps provide its inputs



20-22/10/2015
© All rights reserved
User Conference on Advanced Automated Testing

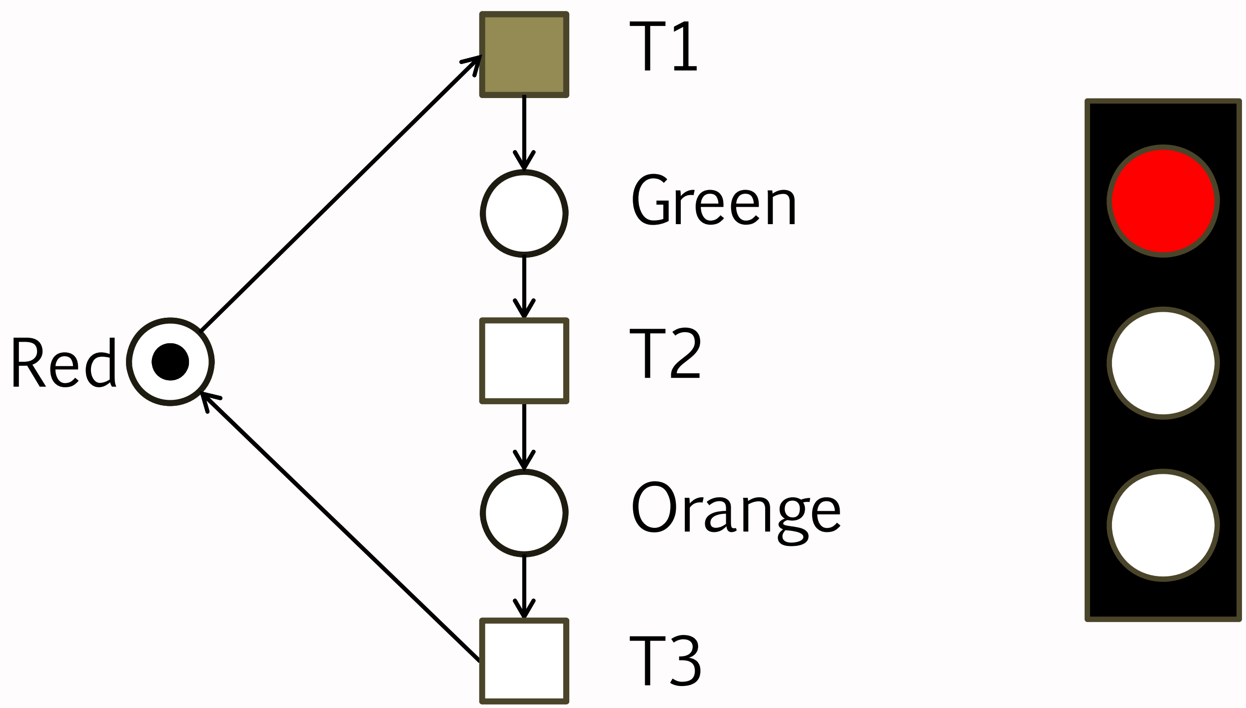
Petri Net - Coffee Machine Problem

Looks familiar?



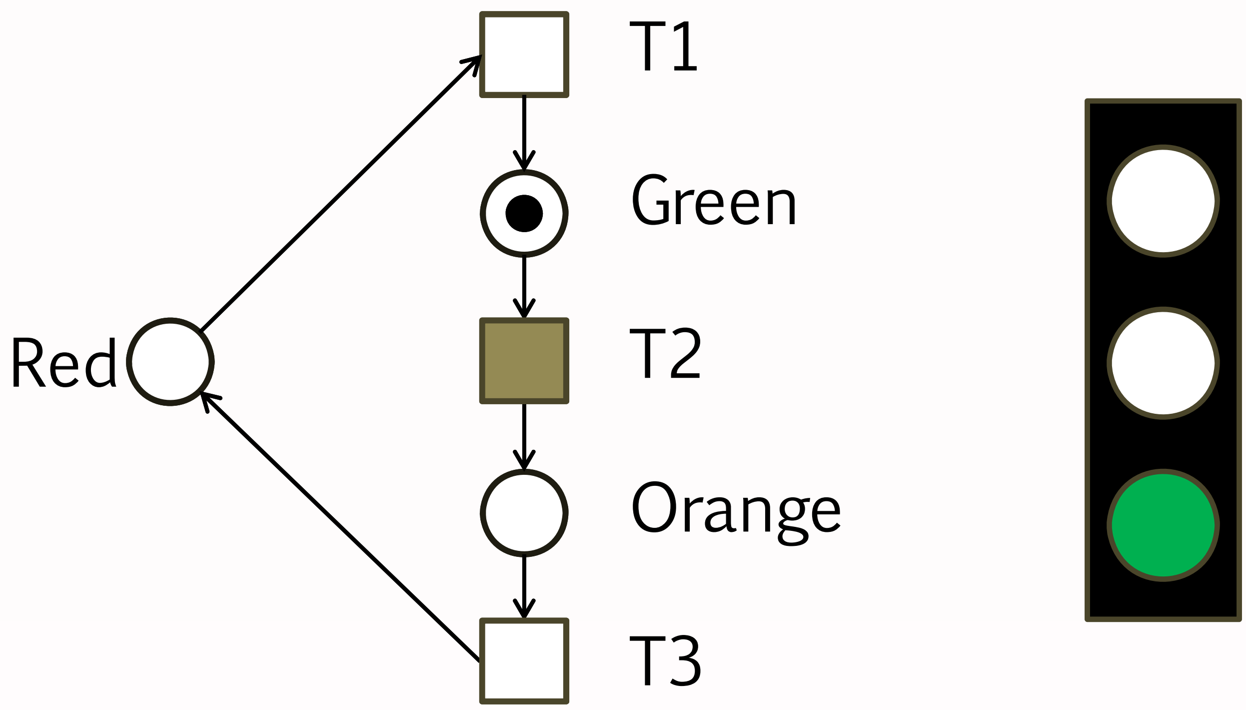
© All rights reserved

Petri Net – Traffic Light Model



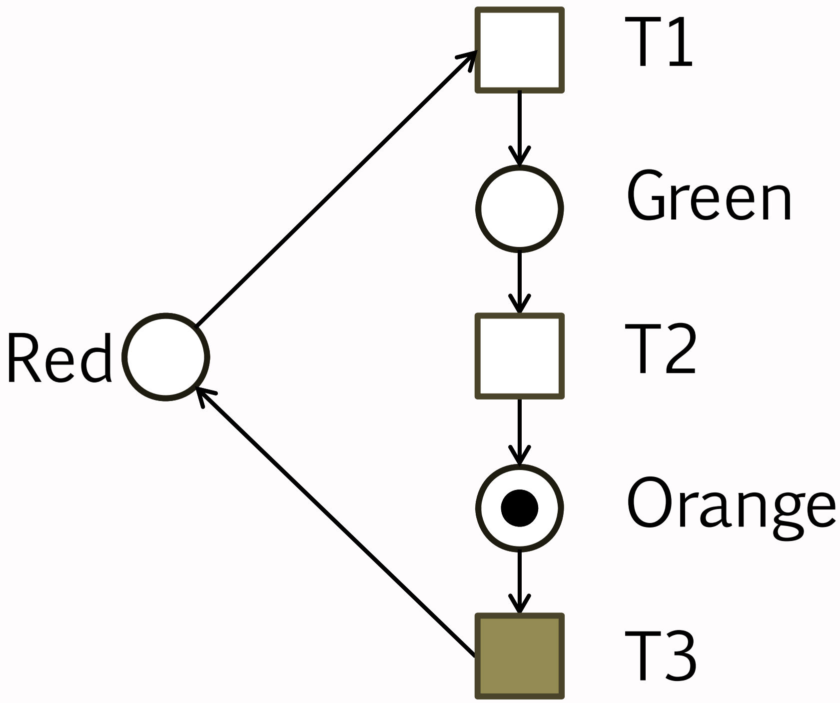
20-22/10/2015
© All rights reserved
User Conference on Advanced Automated Testing

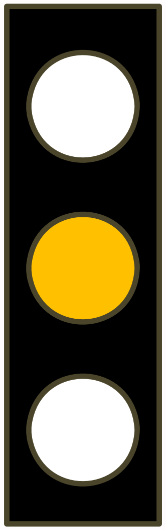
Petri Net - Traffic Light Model



© All rights reserved

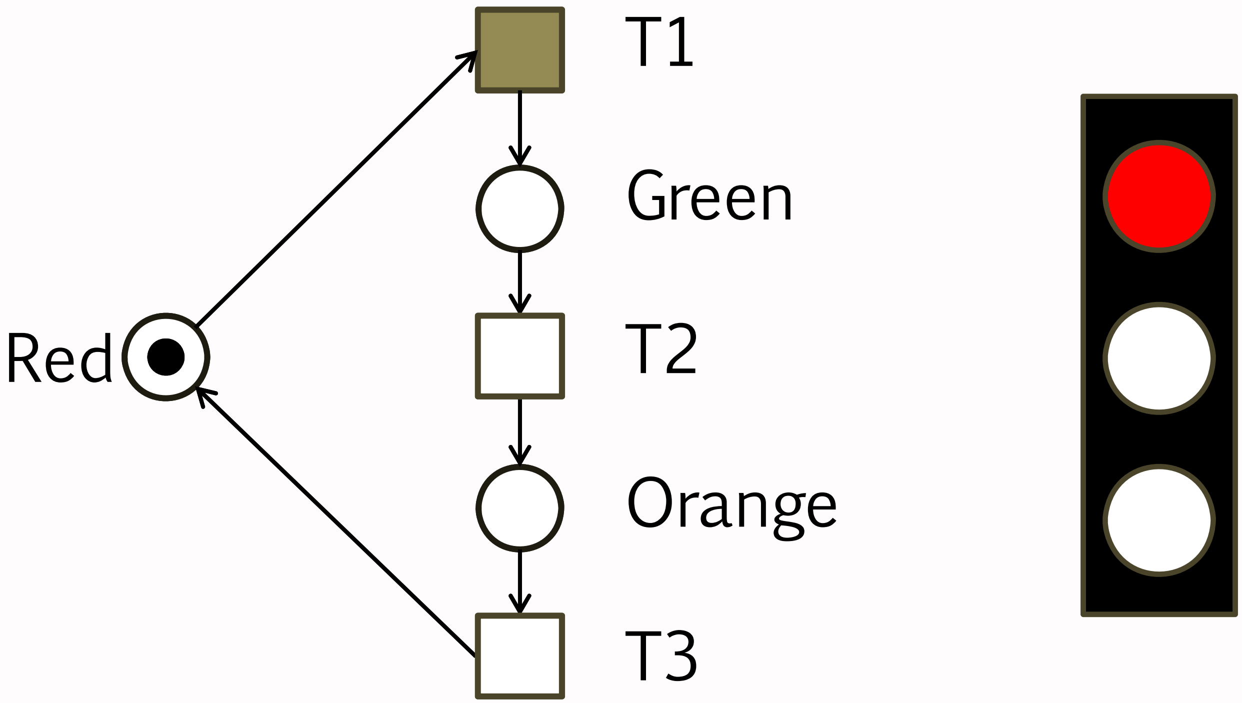
Petri Net – Traffic Light Model





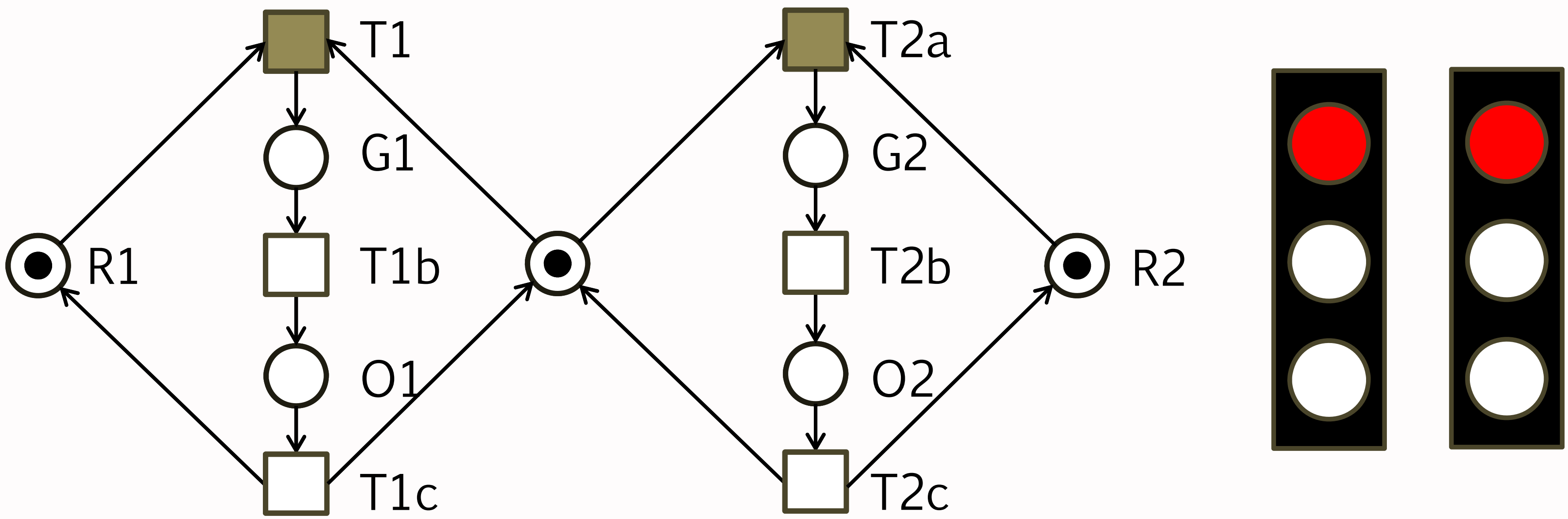
© All rights reserved

Petri Net – Traffic Light Model



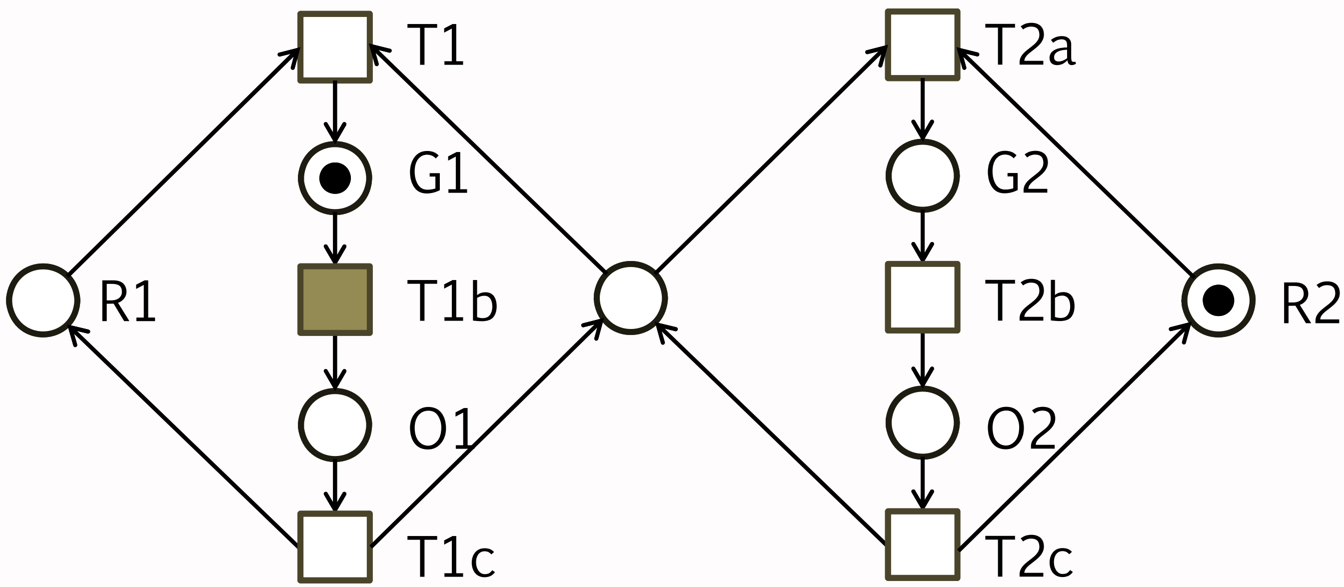
© All rights reserved

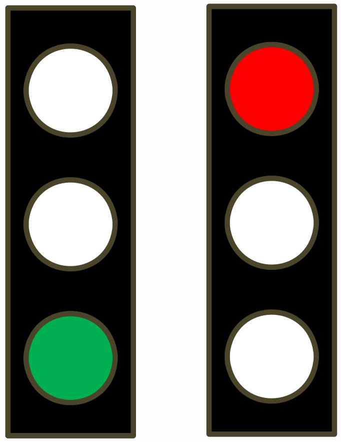
Petri Net – Two Traffic Light Model



© All rights reserved

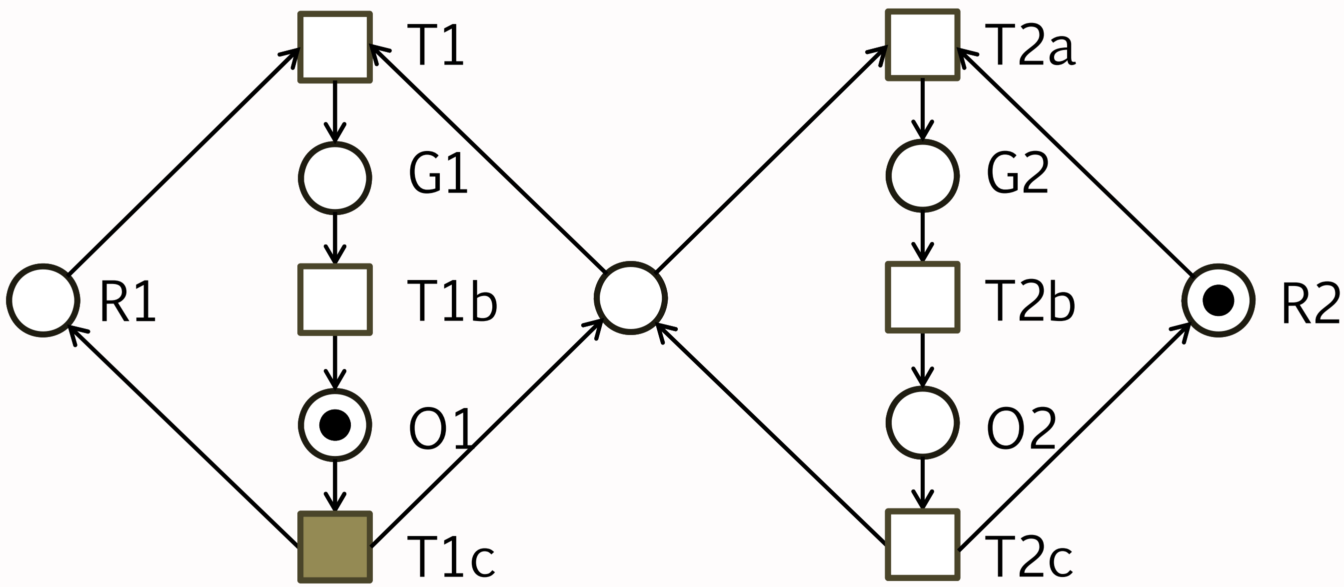
Petri Net – Two Traffic Light Model

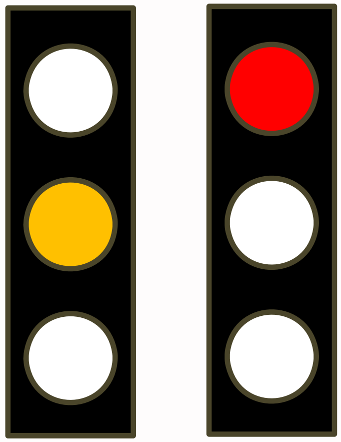




© All rights reserved

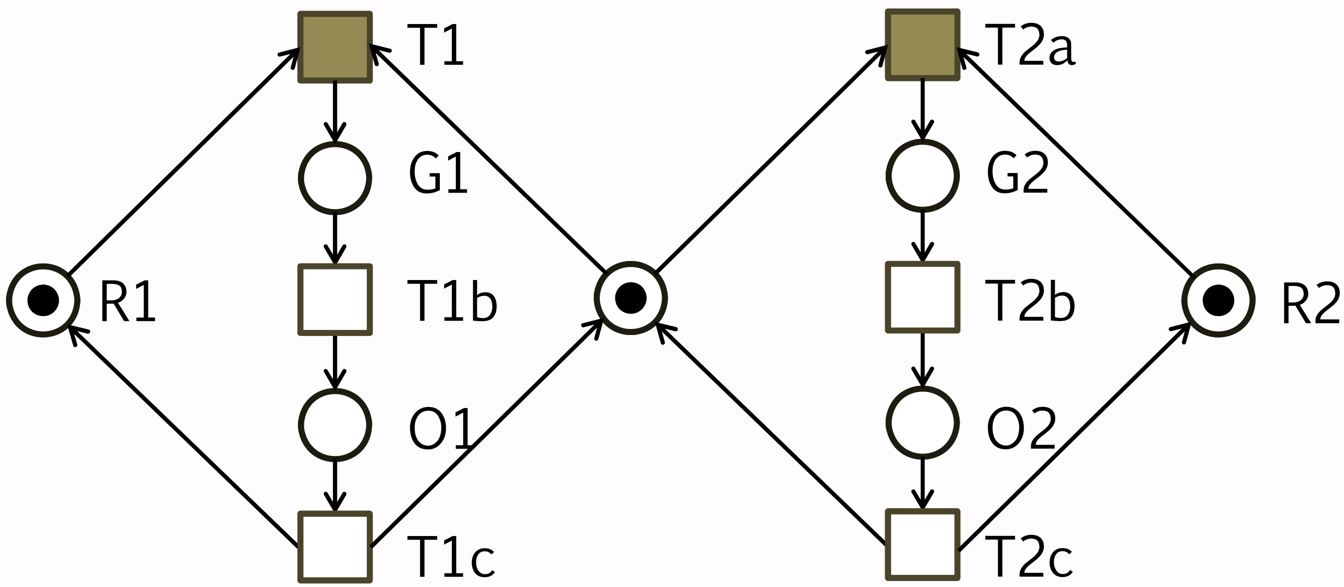
Petri Net – Two Traffic Light Model

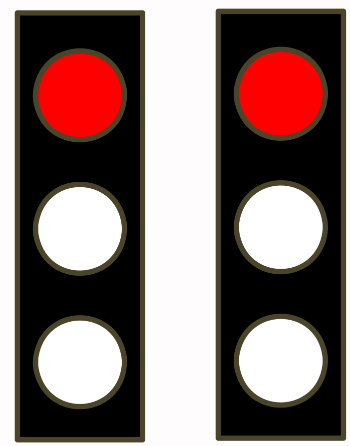




© All rights reserved

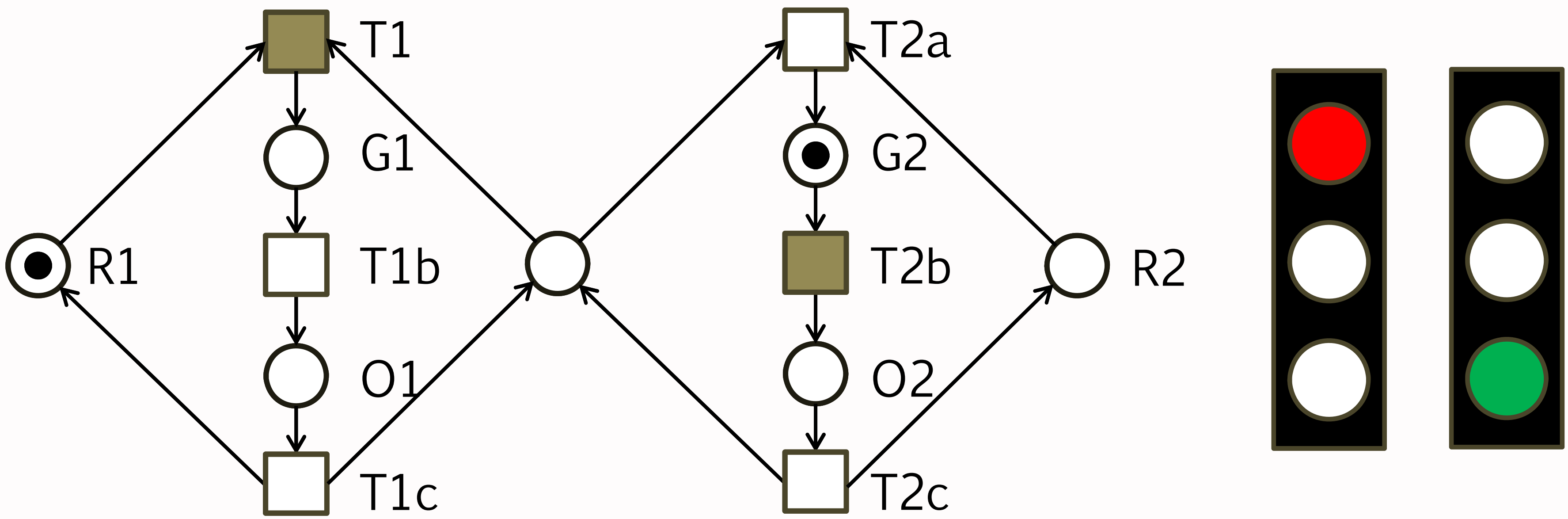
Petri Net – Two Traffic Light Model





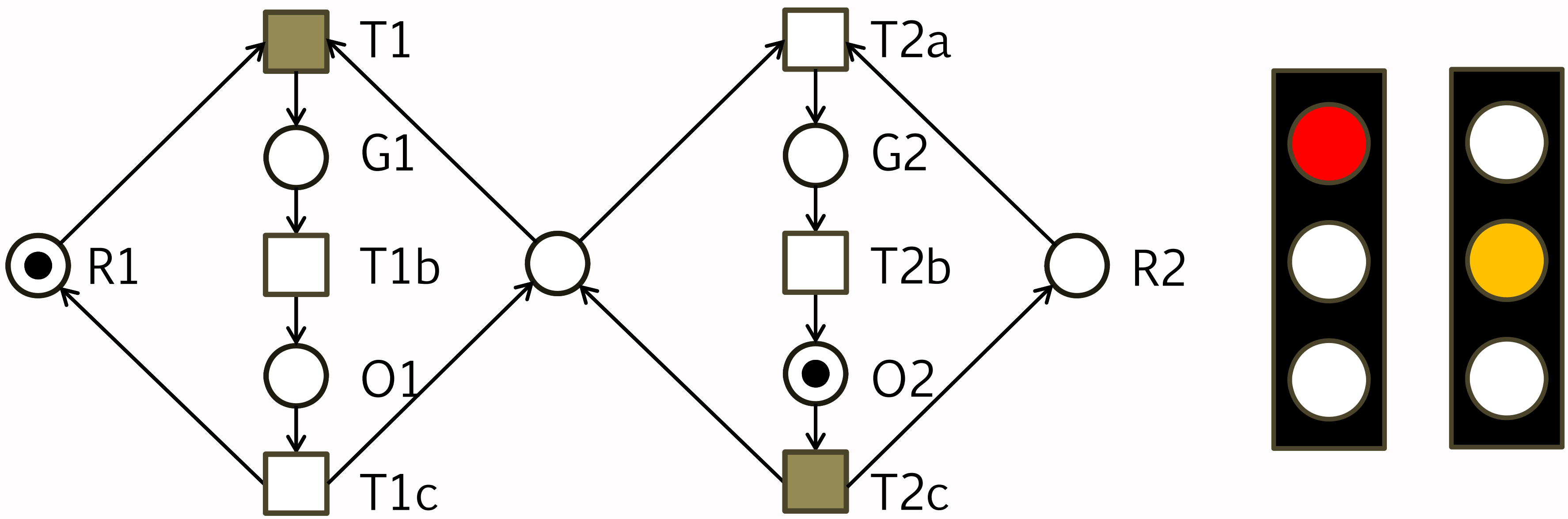
© All rights reserved

Petri Net – Two Traffic Light Model



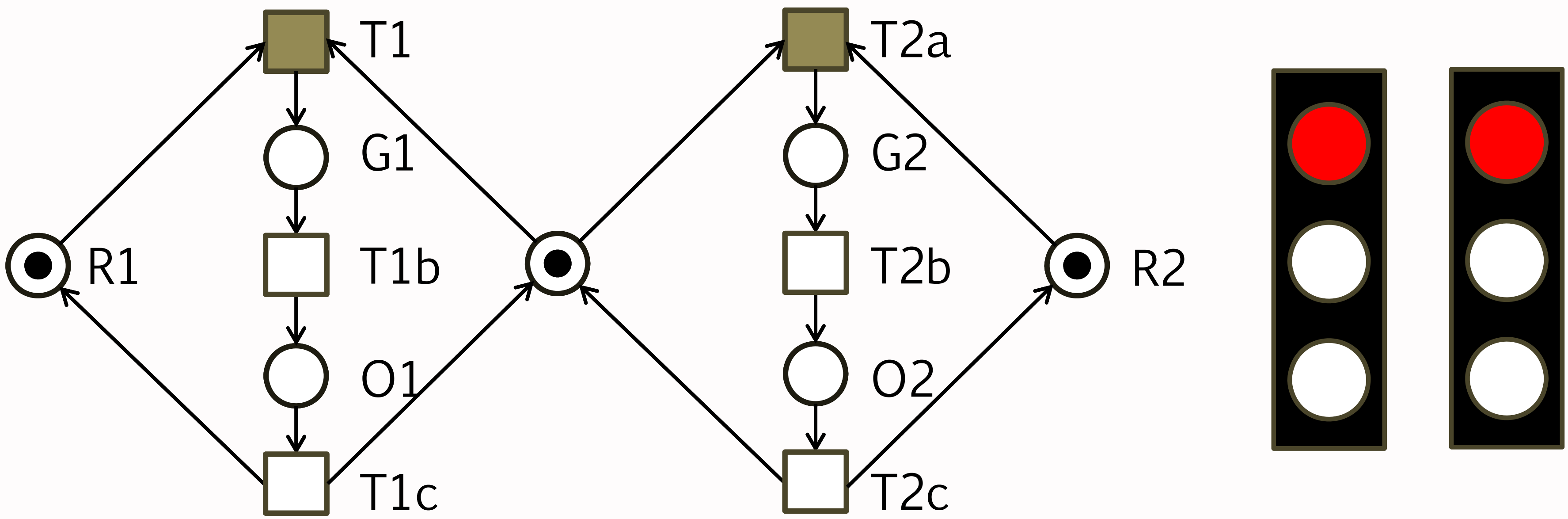
© All rights reserved

Petri Net – Two Traffic Light Model



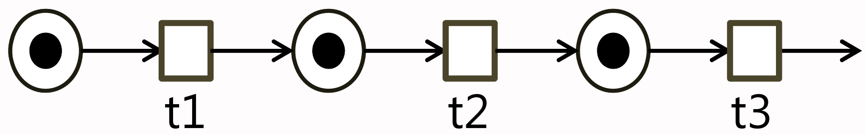
© All rights reserved

Petri Net – Two Traffic Light Model

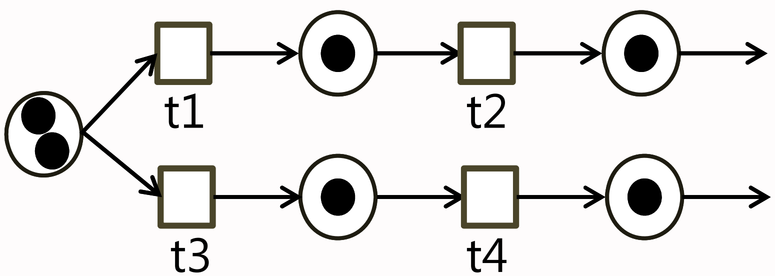


© All rights reserved

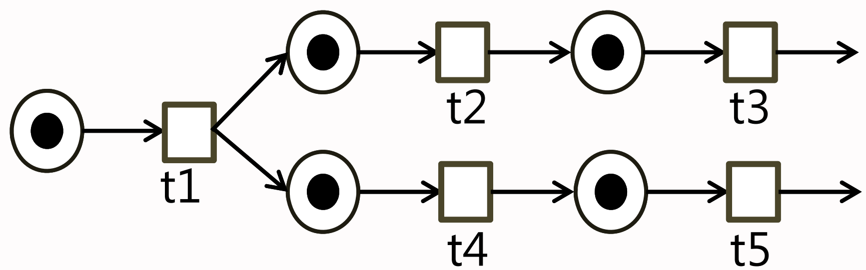
Petri Nets – Expressive Power



Events/actions sequence



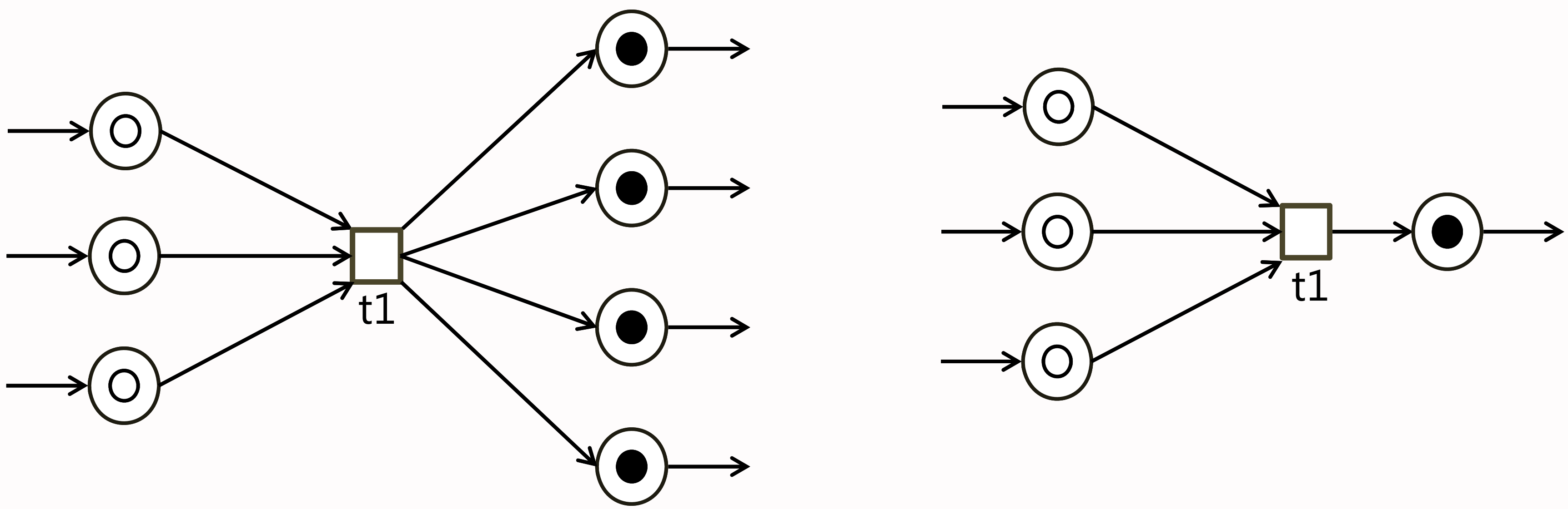
Non-deterministic events - conflict, choice or decision



Concurrent executions

© All rights reserved

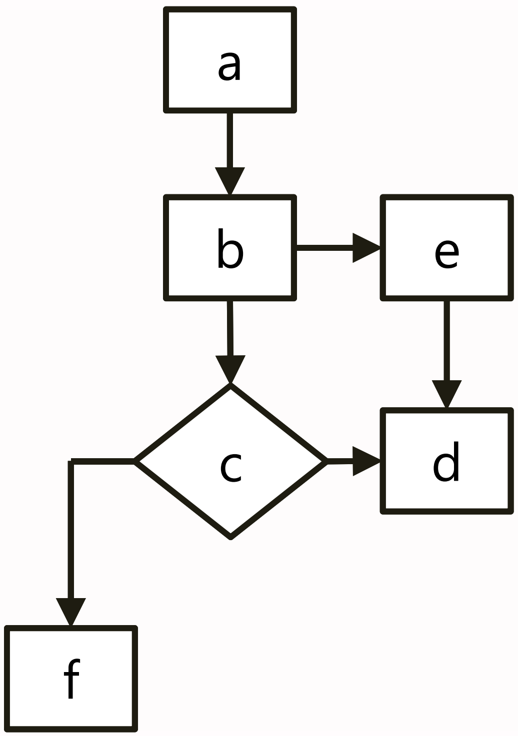
Petri Nets – Expressive Power



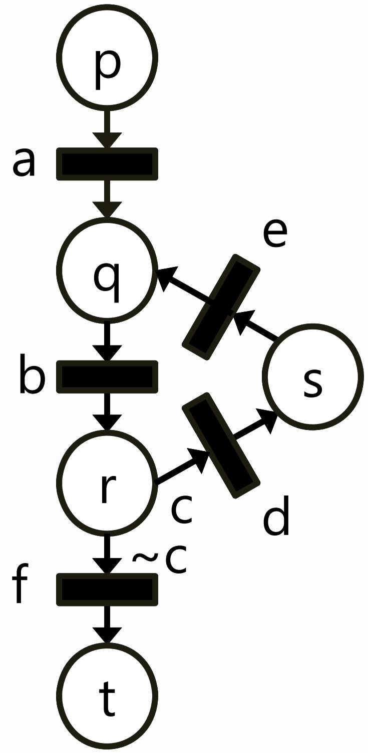
Synchronization and Concurrency

© All rights reserved

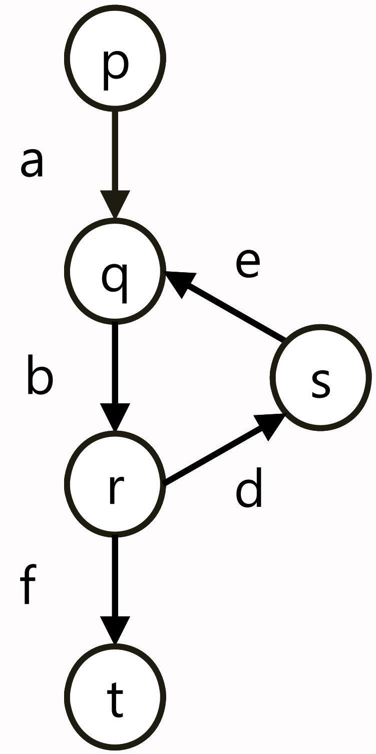
Summary – Model Notations



Flow Chart



Petri Nets



State Machines

© All rights reserved

Summary – Model Expressive Power

	Flowcharts	Finite State Machines	Petri Nets
Sequence	Yes	Yes (unnatural)	Yes (unnatural)
Selection	Yes	Yes	Yes
Repetition	Yes	Yes	Yes
Activate	No	No	Yes
Enable/Disable	No	No	Yes
Trigger/Suspend	No	No	Yes
Pause/Resume	No	No	Yes
Priority	No	No	Yes
Parallel execution	No	No	Yes
Mutual execution	No*	No	Yes
Deadlock	No	No	Yes
Context-sensitive input events	No	Yes	No*
Multi-context output events	No	Yes	No*
Asynchronous Events	No	No	No*
Event quiescence	No	No	No*

© All rights reserved

Break Time

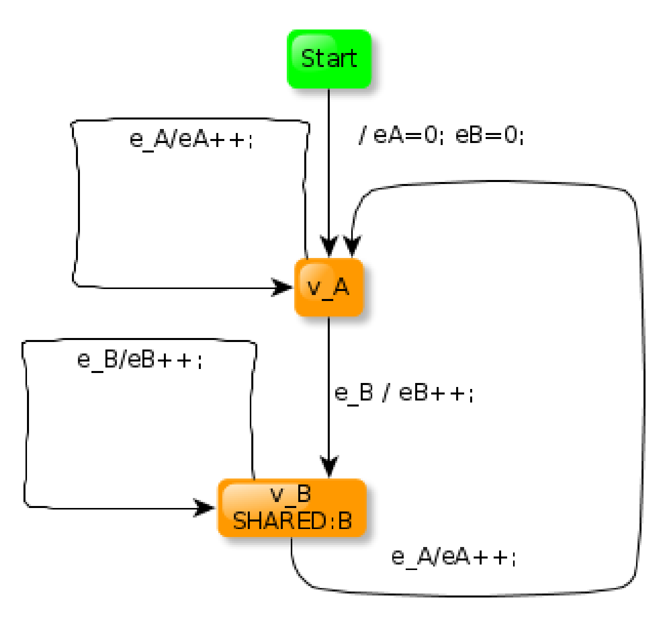
- 5 minute break
- Q&A
- Installing and setting up Unity (optional)

© All rights reserved

From Models to Test Tools

The built-in scripting engine makes Unity easy to extend with custom tools, including test tools

- Integrated NUnit and Integration Test Frameworks
- Model Based Test Designer
  - Uses the open-source GraphWalker tool at the core (ver. 3.3)
    - Extended Finite State Machine
    - Intuitive workflow and notation
  - Integrated test designer interface
  - Builds offline tests that can be executed using the other frameworks



© All rights reserved

From Models to Test Tools

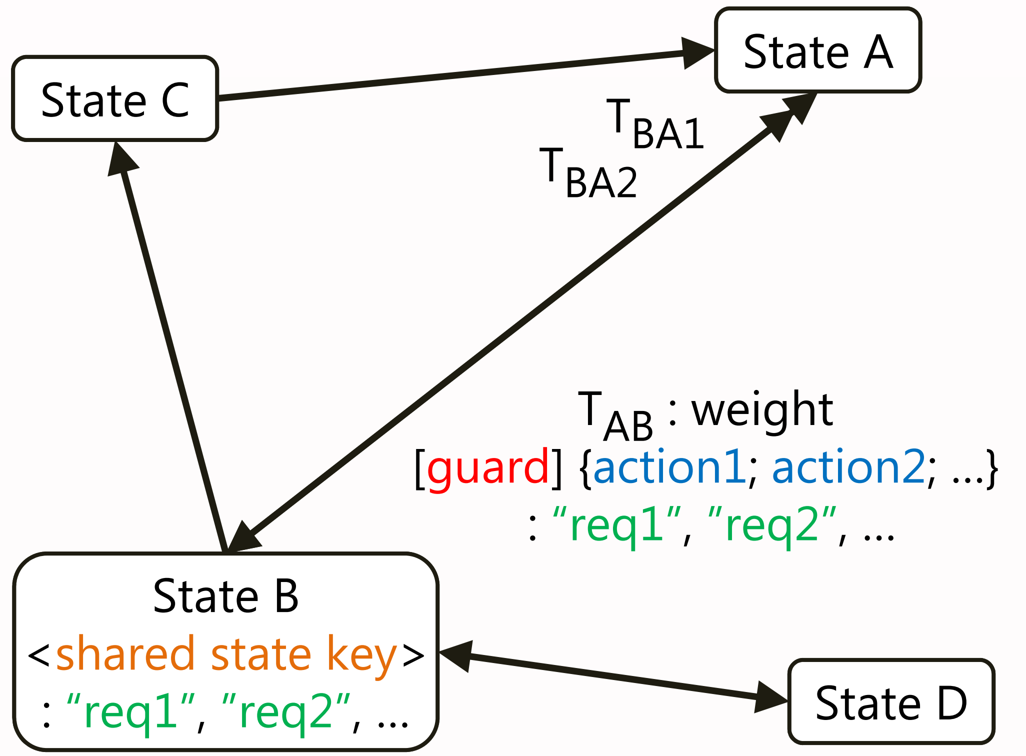
- Actions and Guards
  - Executable annotations that allow for flexible control over path traversals through the model
- Shared State
  - Represents the same system state in multiple models (contexts of the same model)
  - The abstraction allows for hyperlink-like jumps between models, where every next model can extend the represented behaviour's scope or detail level

© All rights reserved

From Models to Test Tools

Model notation

- **Model**
  - Start State
  - Start Actions
  - Requirements
- **State**
  - Shared State Key
  - Requirements List
- **Transition**
  - Weight
  - Guard
  - Actions
  - Requirements



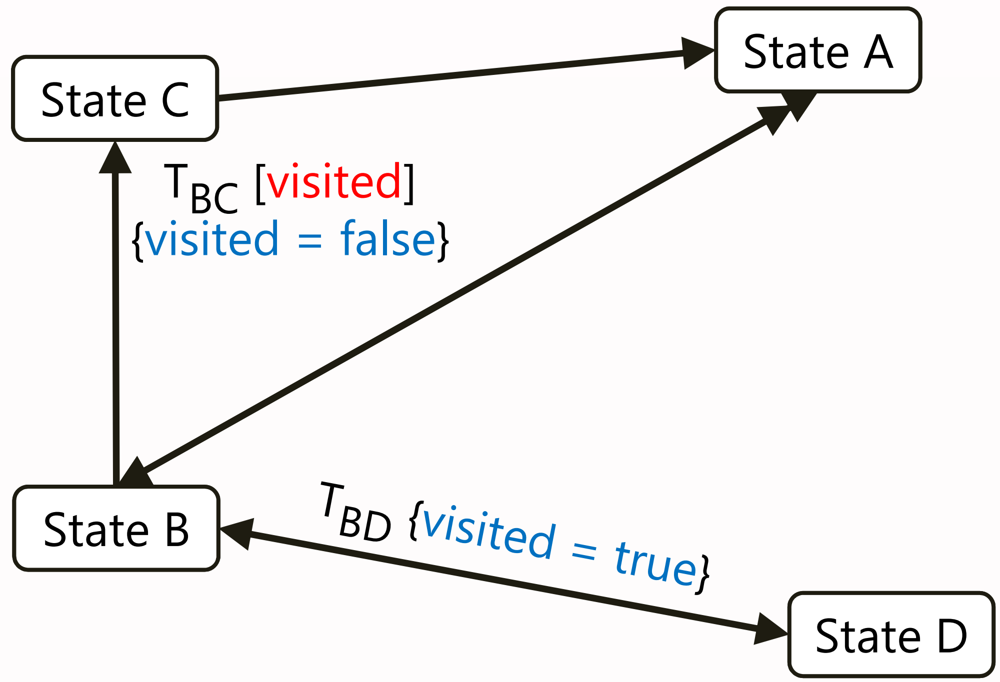
Model M <start state> {action1; action2; …}
: "requirement1", "requirement2", …

© All rights reserved

From Models to Test Tools

Model notation

- Model
  - **Start Actions**
- Transition
  - **Guard**
  - **Actions**



Make **ABC** illegal - always visit **D** before **C**.

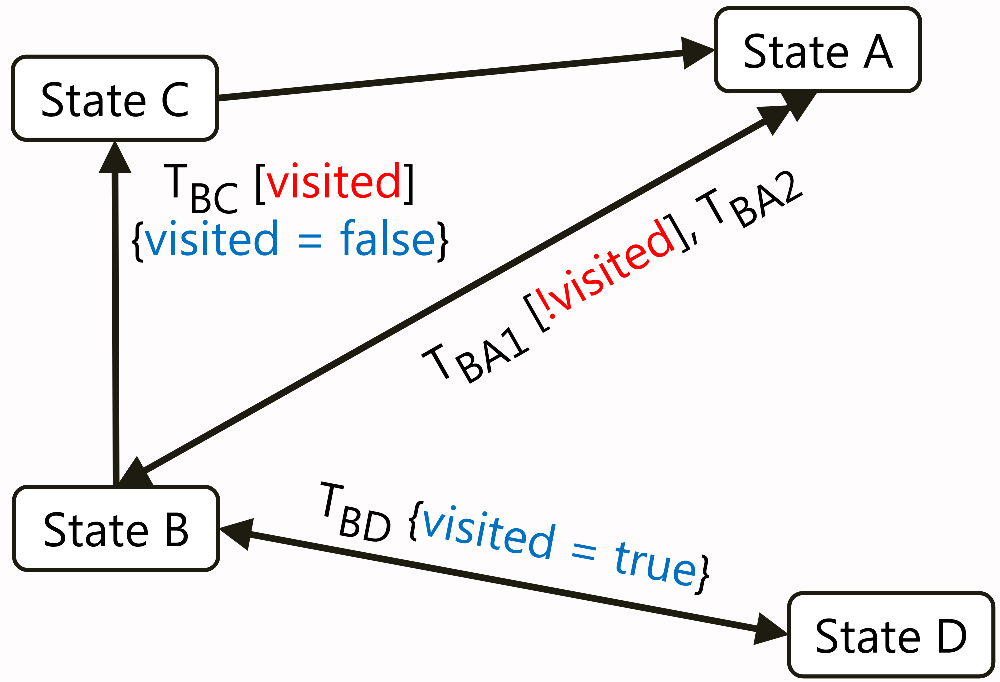
Model M <State A> {var visited = false}

20-22/10/2015
© All rights reserved
User Conference on Advanced Automated Testing

From Models to Test Tools

Model notation

- Model
  - **Start Actions**
- Transition
  - **Guard**
  - **Actions**



Also, make $\mathbf{T_{BA1}}$ feedback illegal between **D** and **C**.

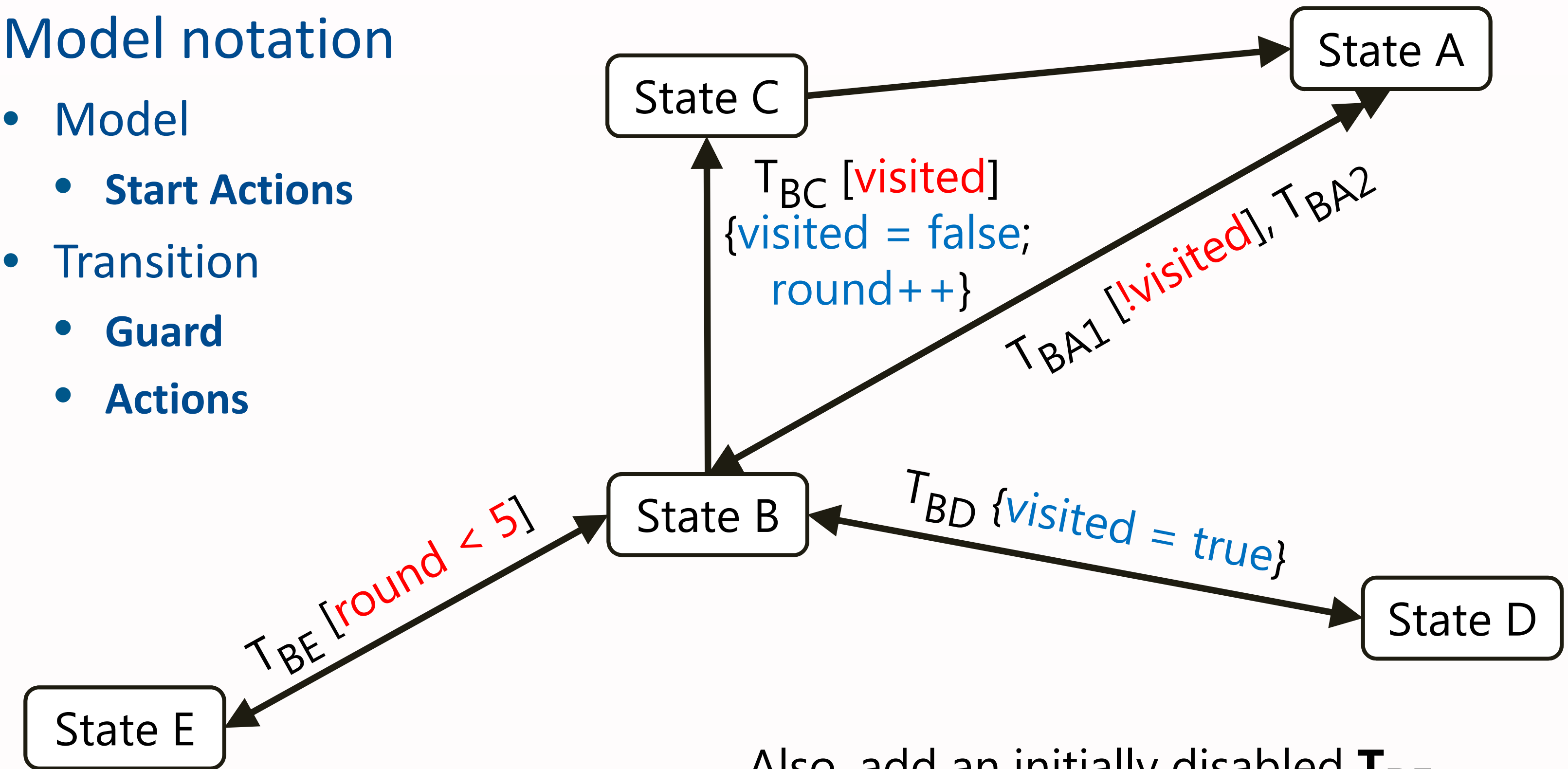
Model M <State A> {var visited = false}

© All rights reserved

From Models to Test Tools

Model notation

- Model
  - **Start Actions**
- Transition
  - **Guard**
  - **Actions**



Also, add an initially disabled **T_{BE}**.

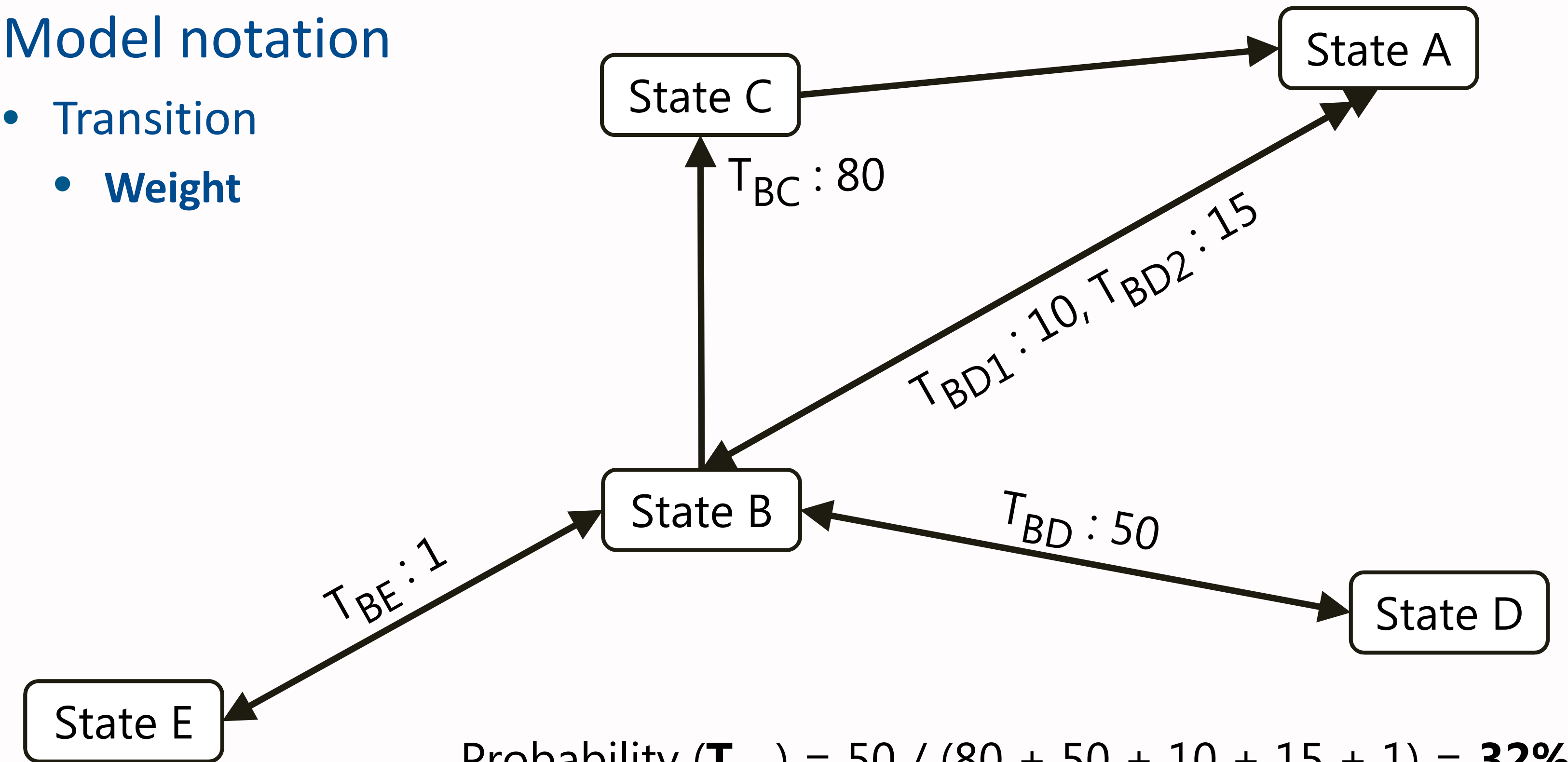
Model M <State A> {var visited = false; var round = 0}

© All rights reserved

From Models to Test Tools

Model notation

- Transition
  - **Weight**



Probability ($\mathbf{T_{BD}}$) = 50 / (80 + 50 + 10 + 15 + 1) = **32%**

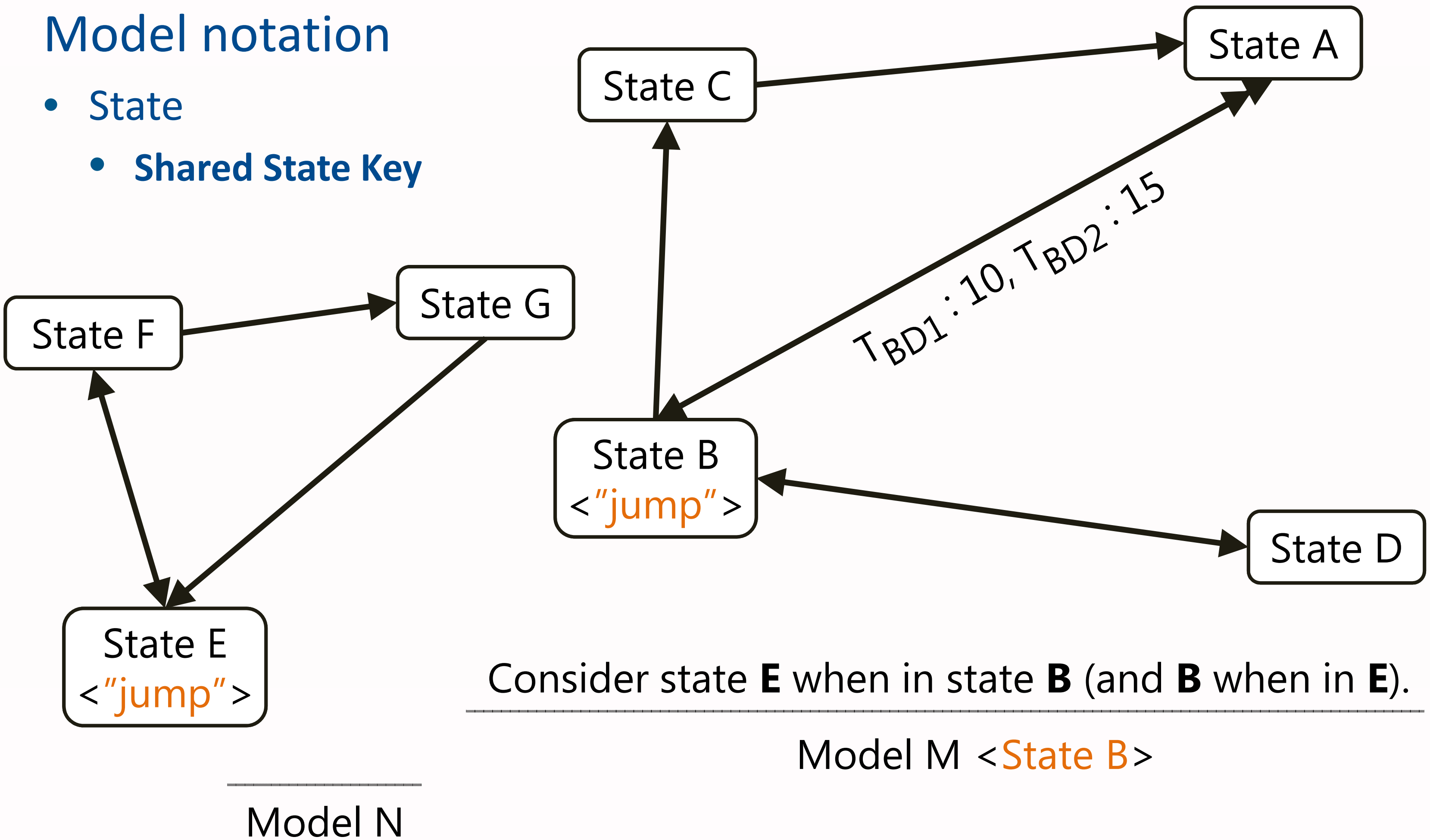
Model M <State B>

© All rights reserved

From Models to Test Tools

Model notation

- State
  - **Shared State Key**

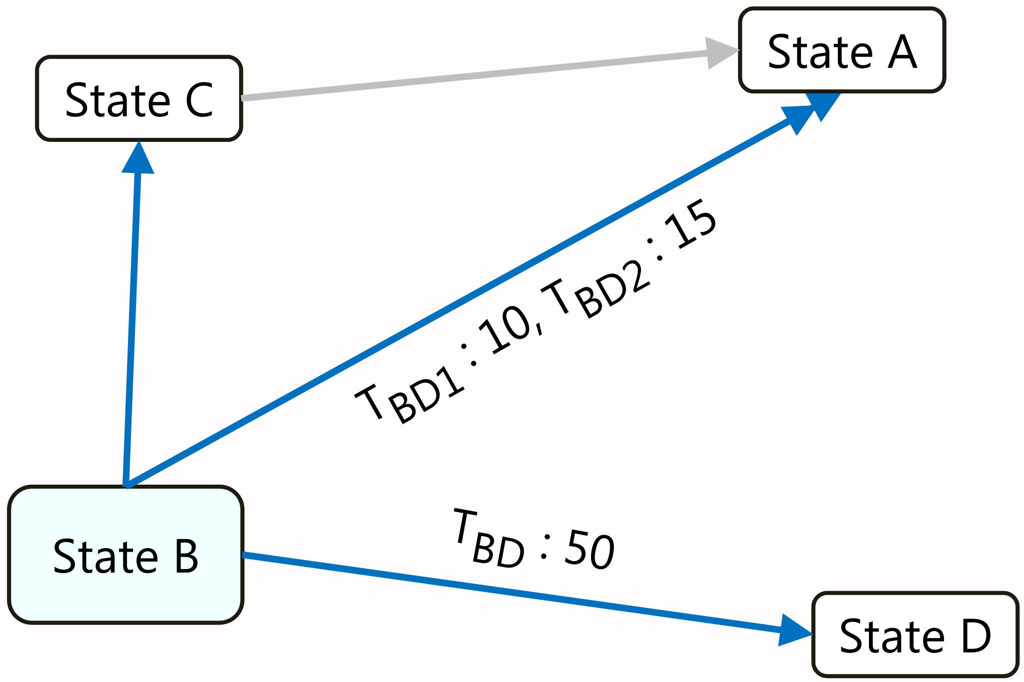


© All rights reserved

From Models to Test Tools

Path generators

- Random
- Quick Random
- Weighted
- A Star



What's the next step?

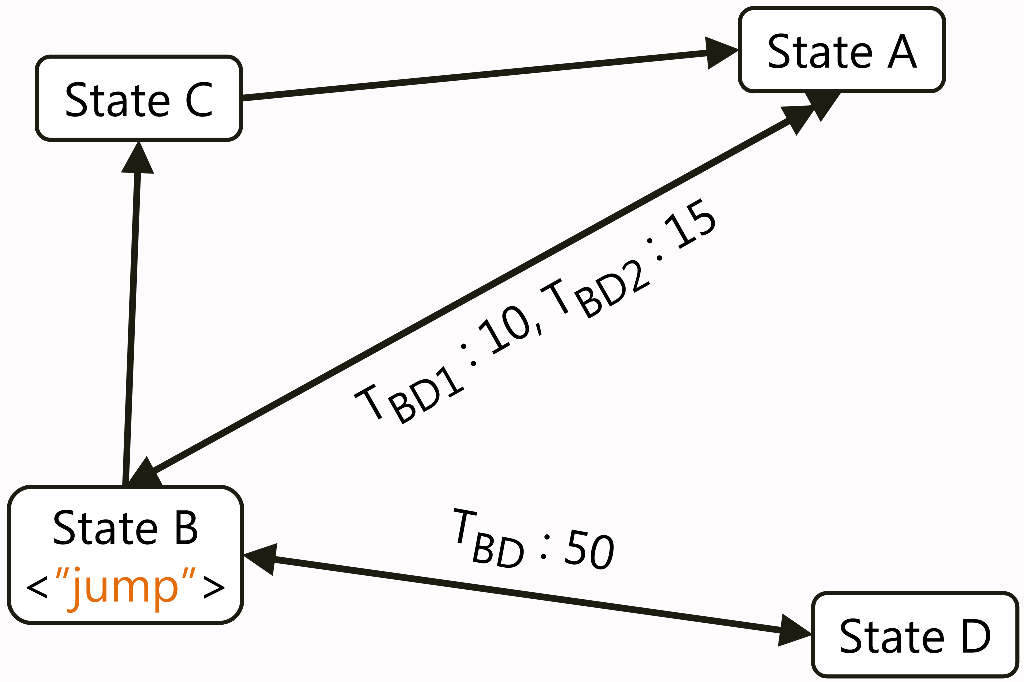
Model M <State B>

© All rights reserved

From Models to Test Tools

Stop Conditions

- Coverage
  - States
  - Transitions
  - Requirements
- Reached Target
  - (Shared) State
  - Transition
  - Assertion
- Time Duration
- Path Length
- Composite
  - AND/OR Tree

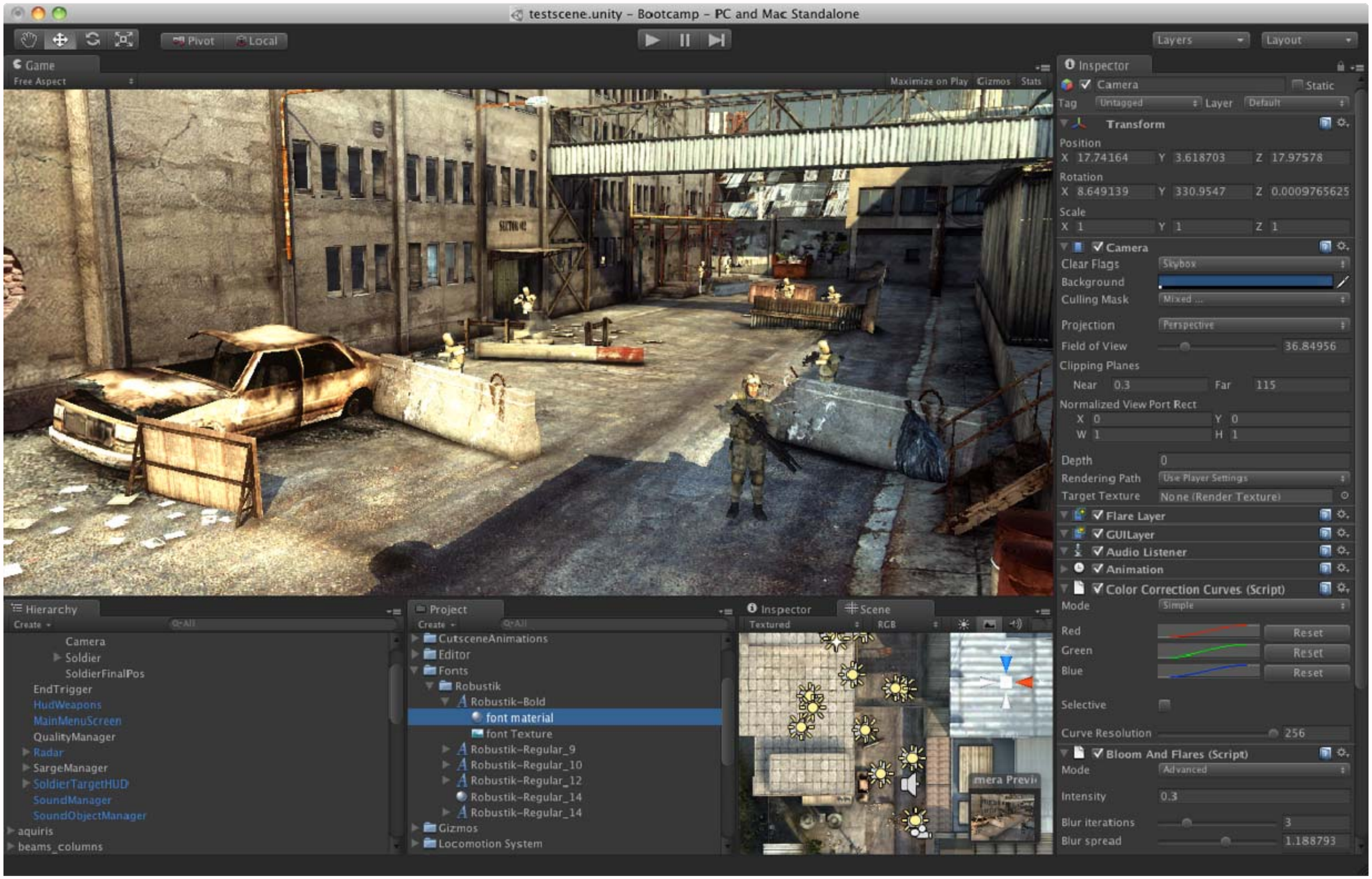


Pick at **Random** until
Path Length = 4 AND Reached "State D"

Model M <State B>

© All rights reserved

Let’s see it in action!



© All rights reserved

Thank You!

Q & A

© All rights reserved