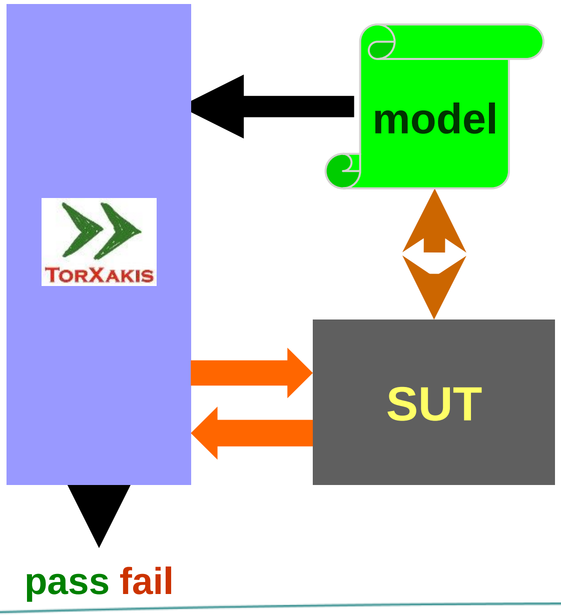
Model-Based Testing



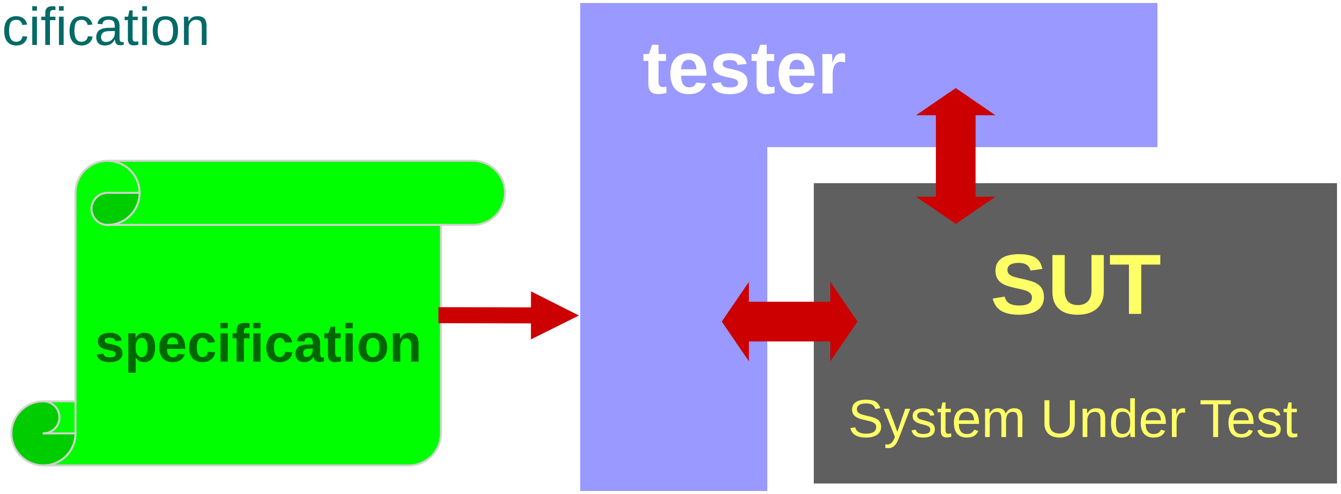
Jan Tretmans

ESI – Embedded Systems Innovation by TNO
Radboud University Nijmegen
Högskolan i Halmstad
jan.tretmans@tno.nl

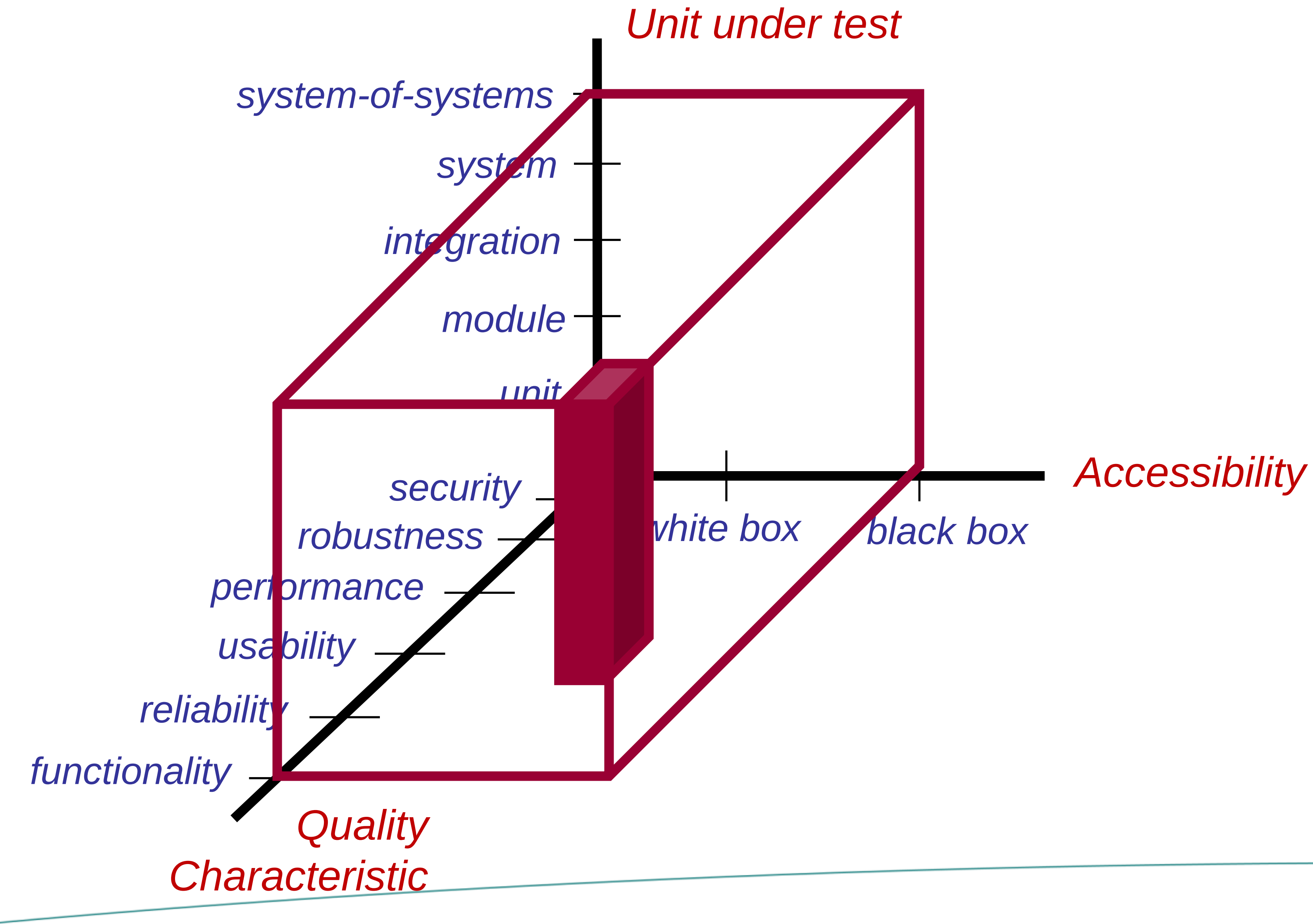
Software Testing

Checking or measuring
some quality characteristics
of an executing software object
by performing experiments
in a controlled way
w.r.t. a specification

*specification-based,
active, black-box
testing of
functionality*



Sorts of Testing



Software Testing

Measuring some quality characteristic
of an executing software object
by performing experiments
in a controlled environment
while comparing actual behaviour
with required behaviour

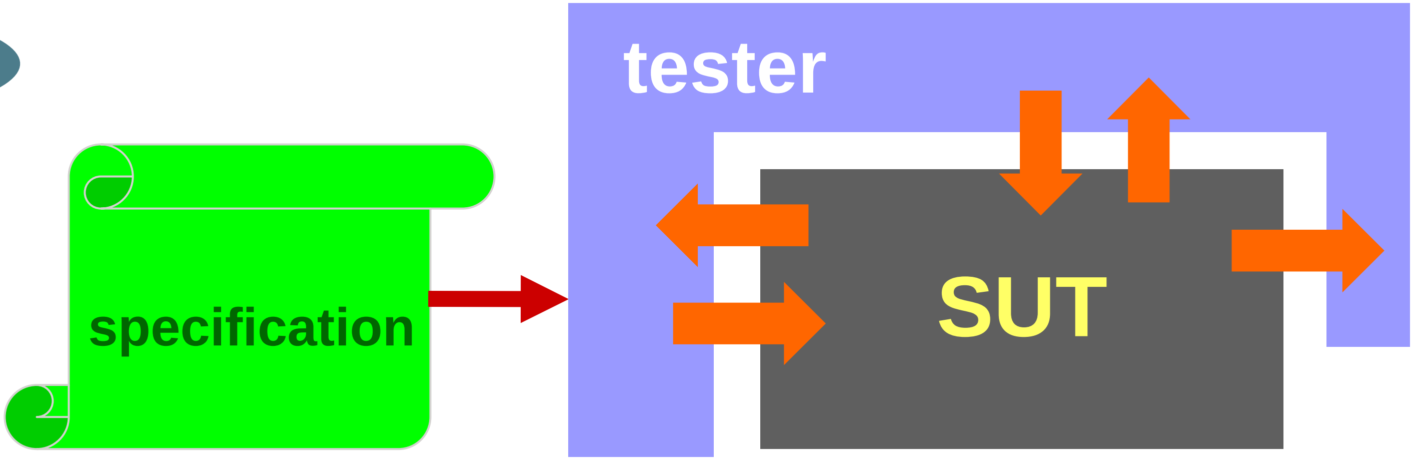
functionality

SUT

tester

specification

*specification-based,
active,
black-box
testing
of functionality*

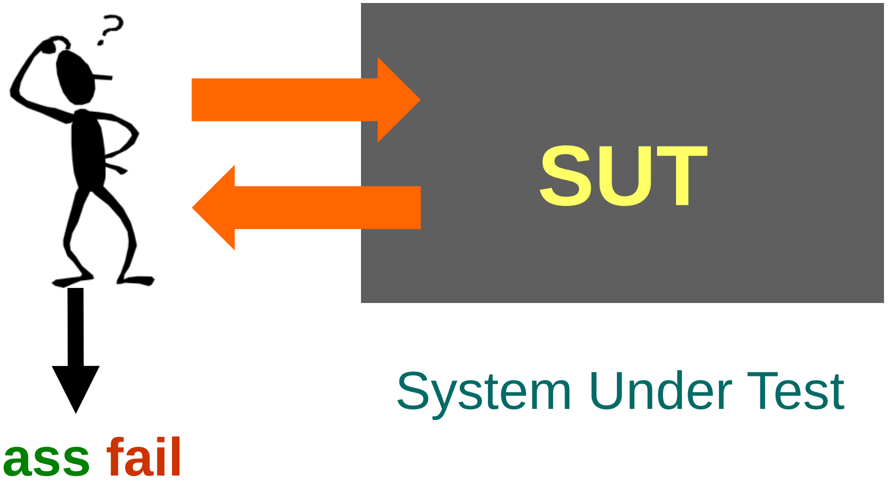


Model-Based Testing

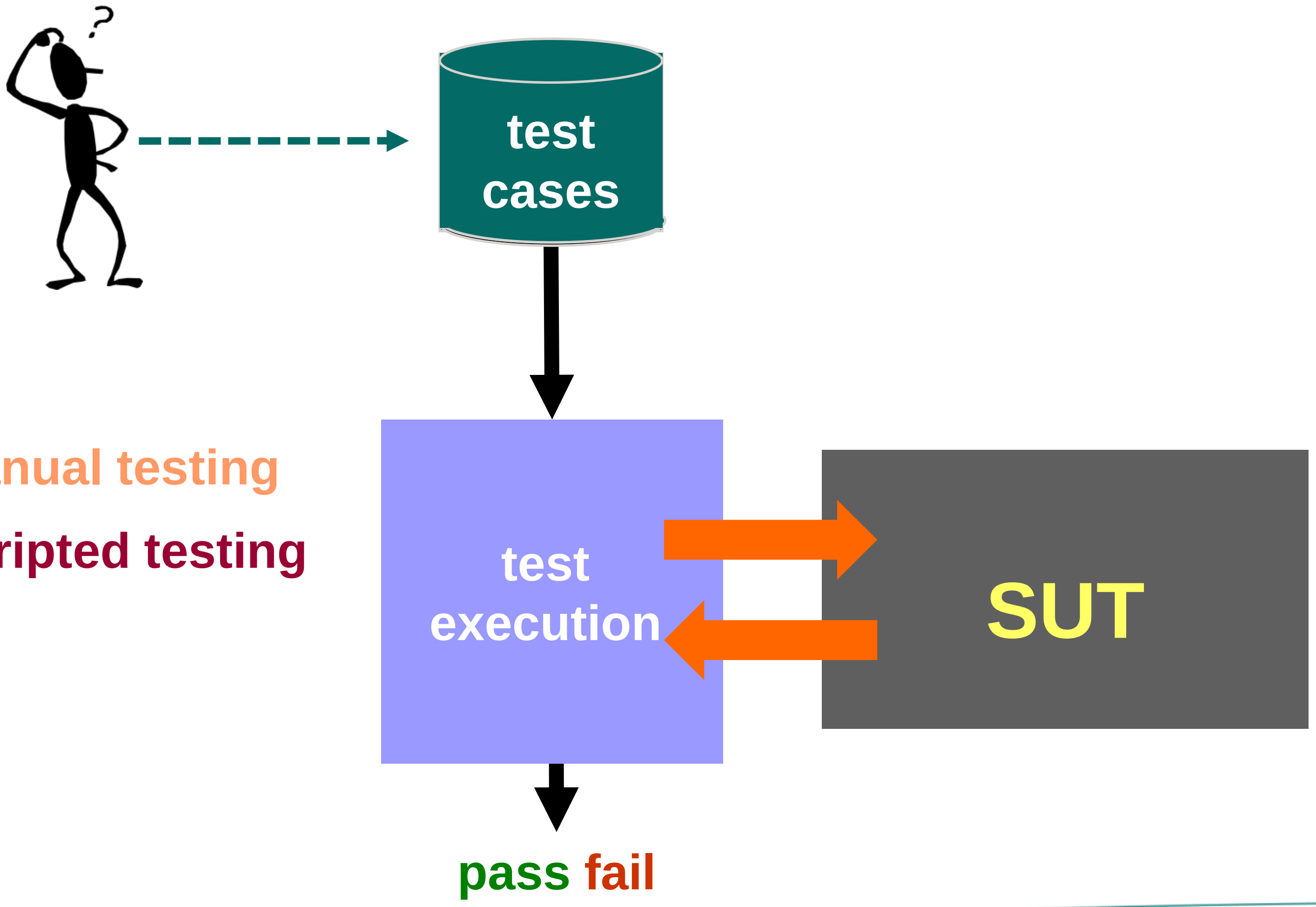
Basics

1 : Manual Testing

1. **Manual testing**

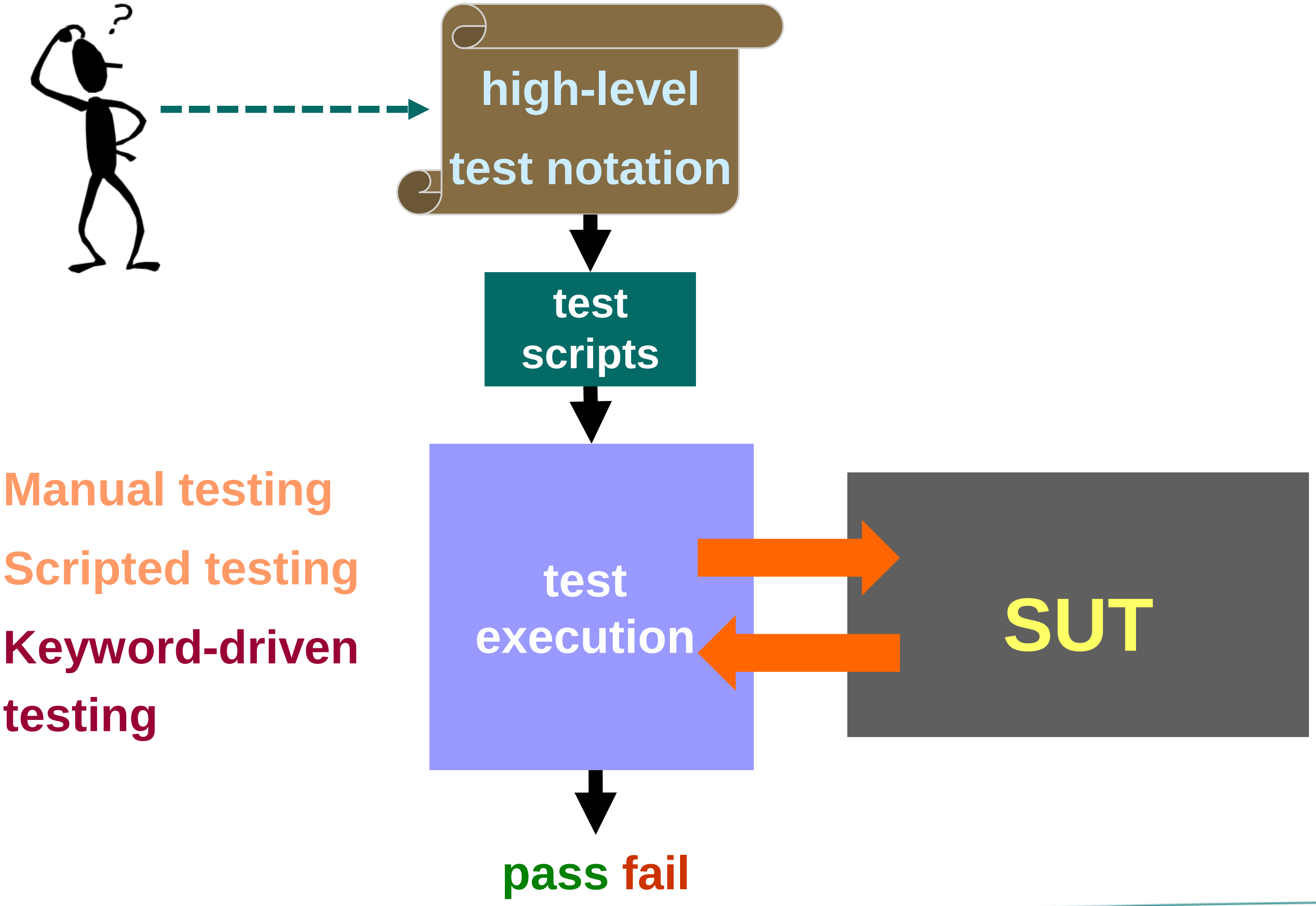


2 : Scripted Testing



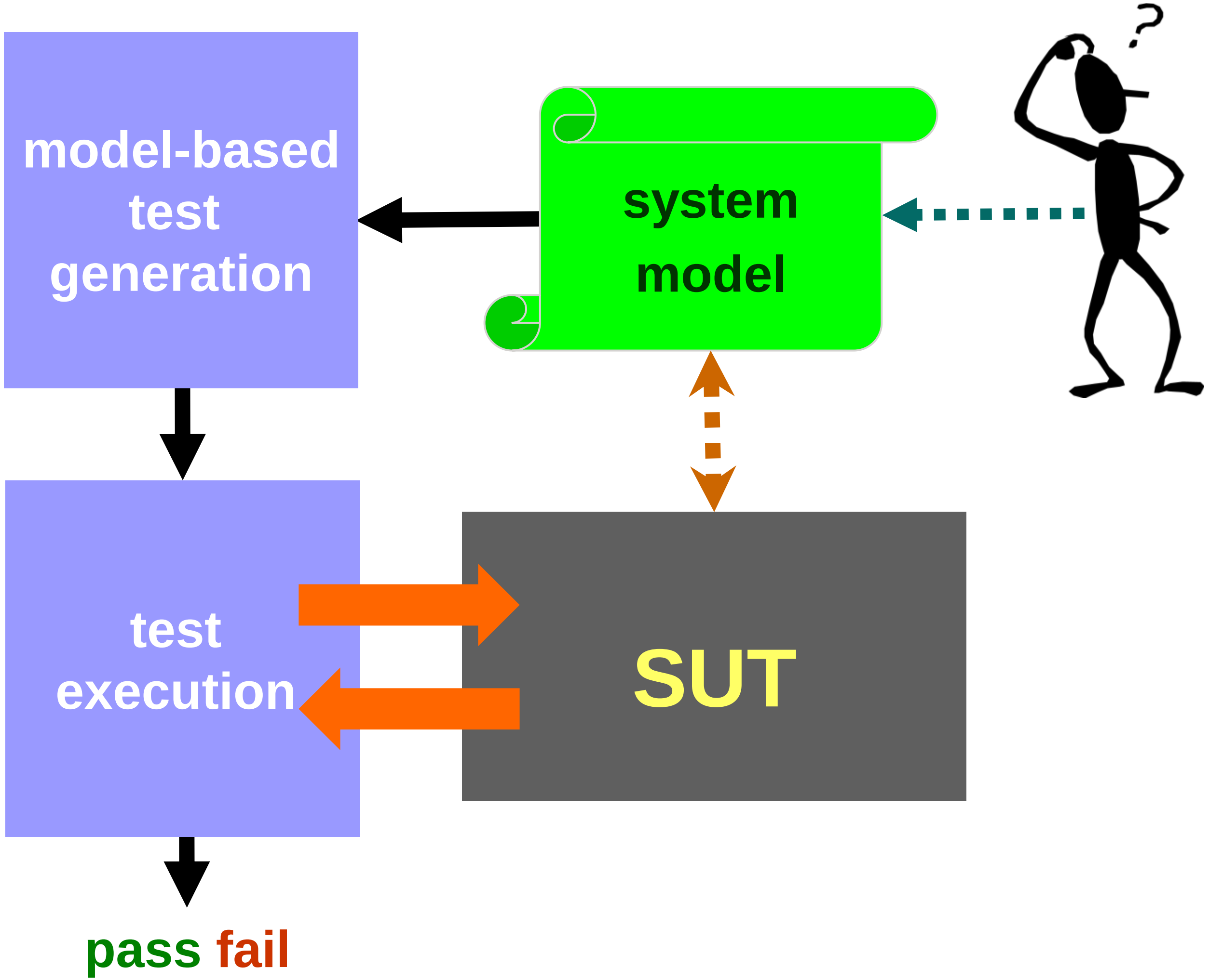
1. Manual testing
2. Scripted testing

3 : Keyword-Driven Testing



1. Manual testing
2. Scripted testing
3. Keyword-driven testing

4 : Model-Based Testing



1. Manual testing
2. Scripted testing
3. Keyword-driven testing
4. Model-based testing

Model-Based Testing

Measuring some quality characteristic
of an executing software object
by performing experiments
in a controlled environment
while comparing actual behaviour
with required behaviour

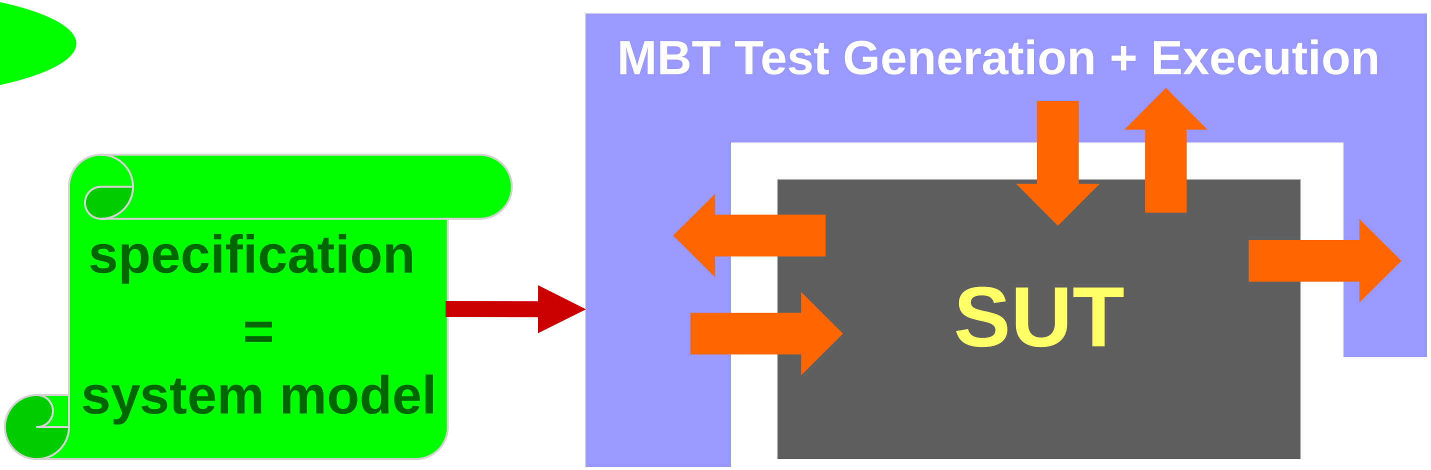
functionality

SUT

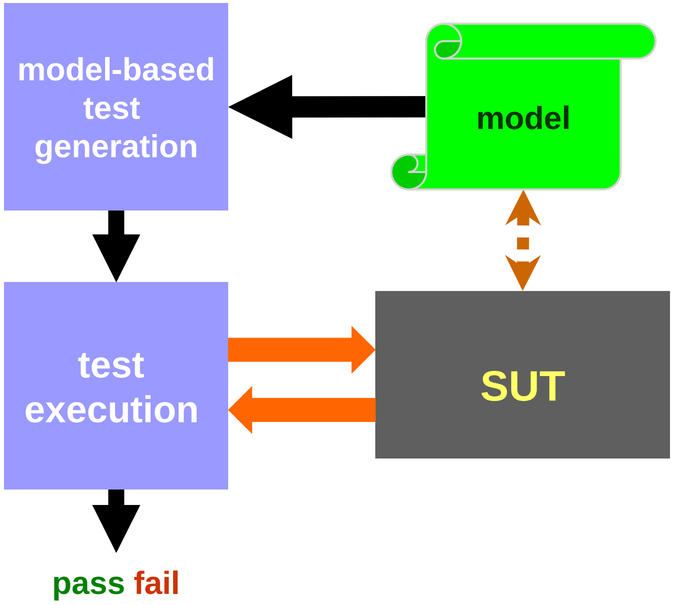
MBT Tester

system model

specification-based,
active,
black-box
testing
of functionality



MBT : Benefits



detecting more bugs

faster and cheaper

MBT: next step in test automation

- **Automatic test generation**

  + test execution + result analysis

- **More, longer, and diversified test cases**

  more variation in test flow and in test data

- **Model is precise and consistent test basis**

  unambiguous analysis of test results

- **Test maintenance by maintaining models**

  improved regression testing

- **Expressing test coverage**

  model coverage

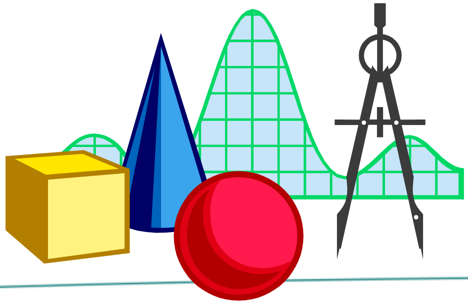
  customer profile coverage

Models

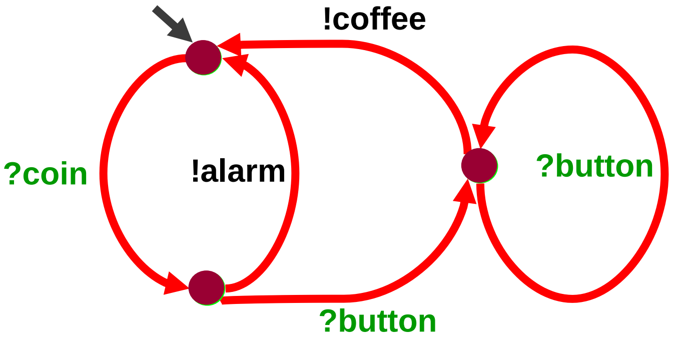
Formal Models

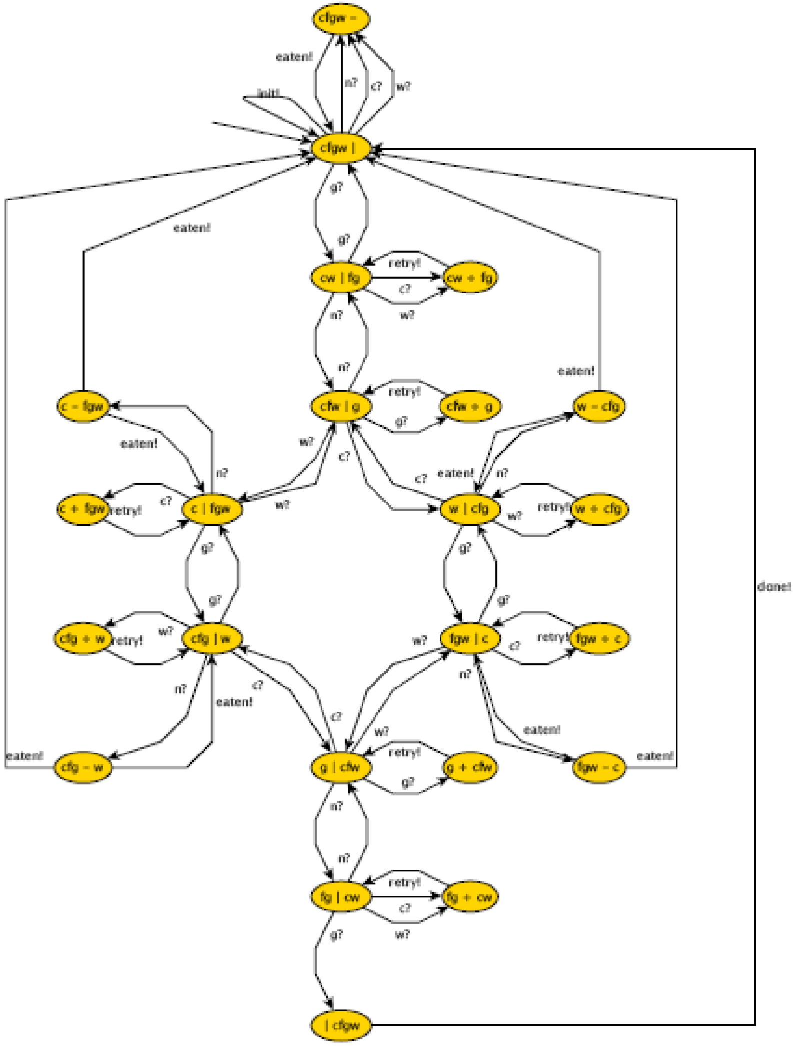
- Use mathematics to model relevant parts of software
- Precise, formal semantics: no room for ambiguity or misinterpretation
- Allow formal validation and reasoning about systems
- Amenable to tools: automation

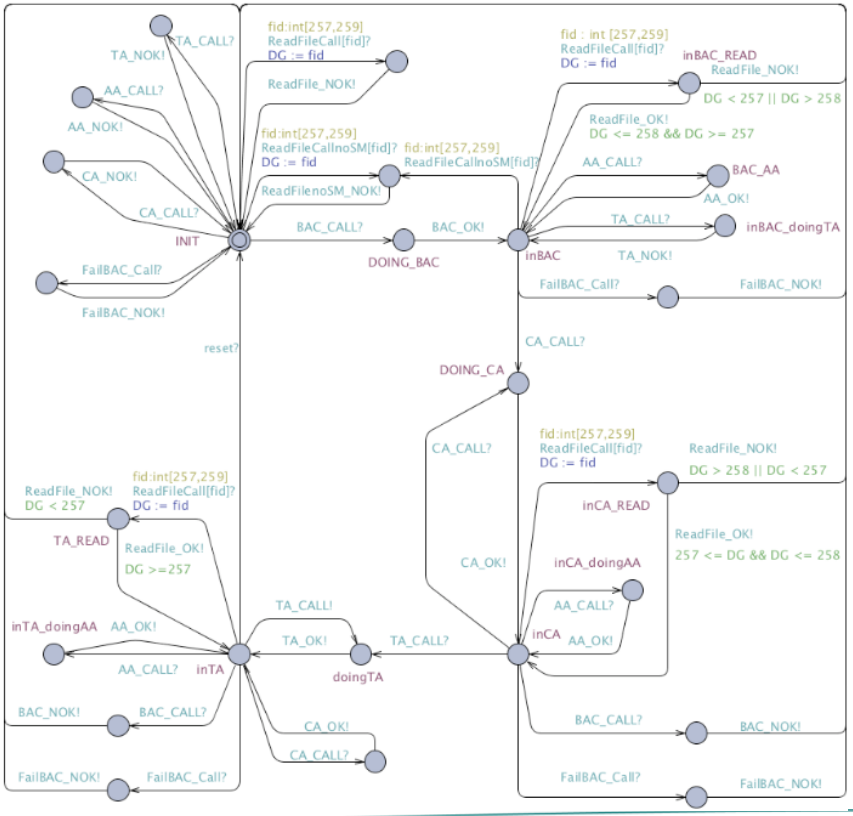
- Examples:
  - Z
  - Temporal Logic
  - First order logic
  - SDL
  - LOTOS
  - Promela
  - Labelled Transition Systems
  - Finite state machine
  - Process algebra
  - ......



MBT : Example Models

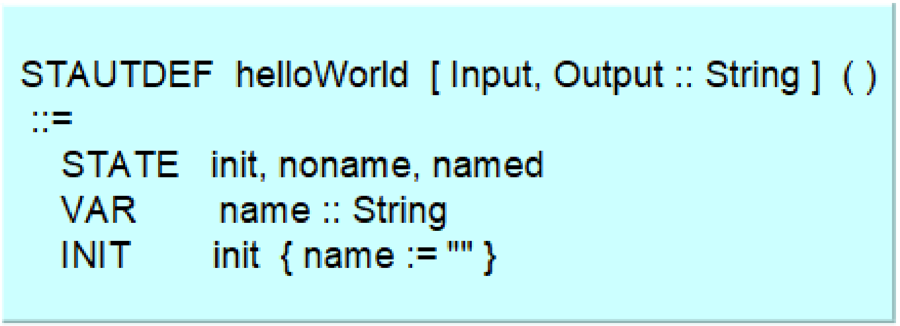




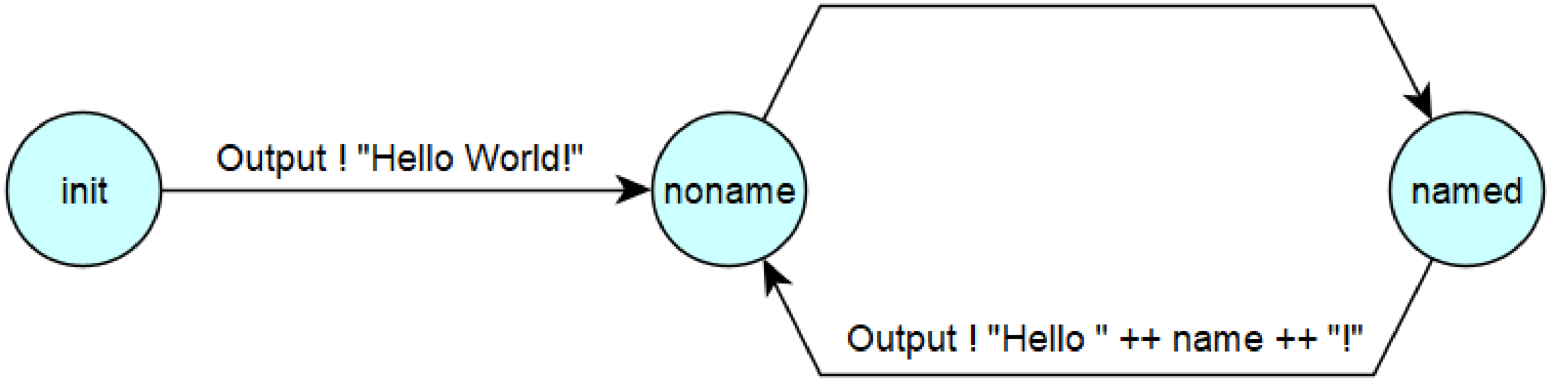


Hello World! Model of Behaviour

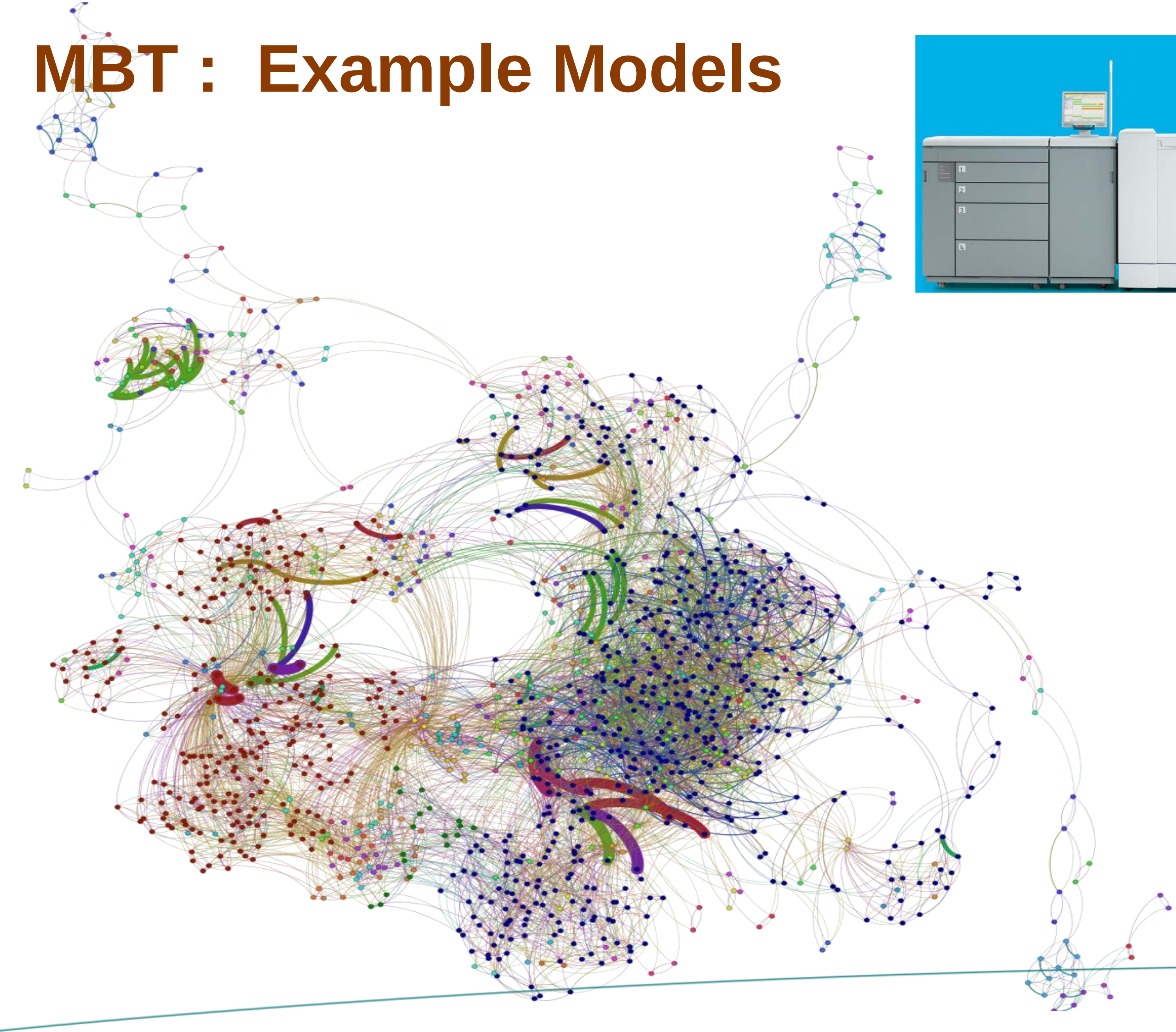
1. Model of interfaces as channels
2. Model of behaviour as state automaton



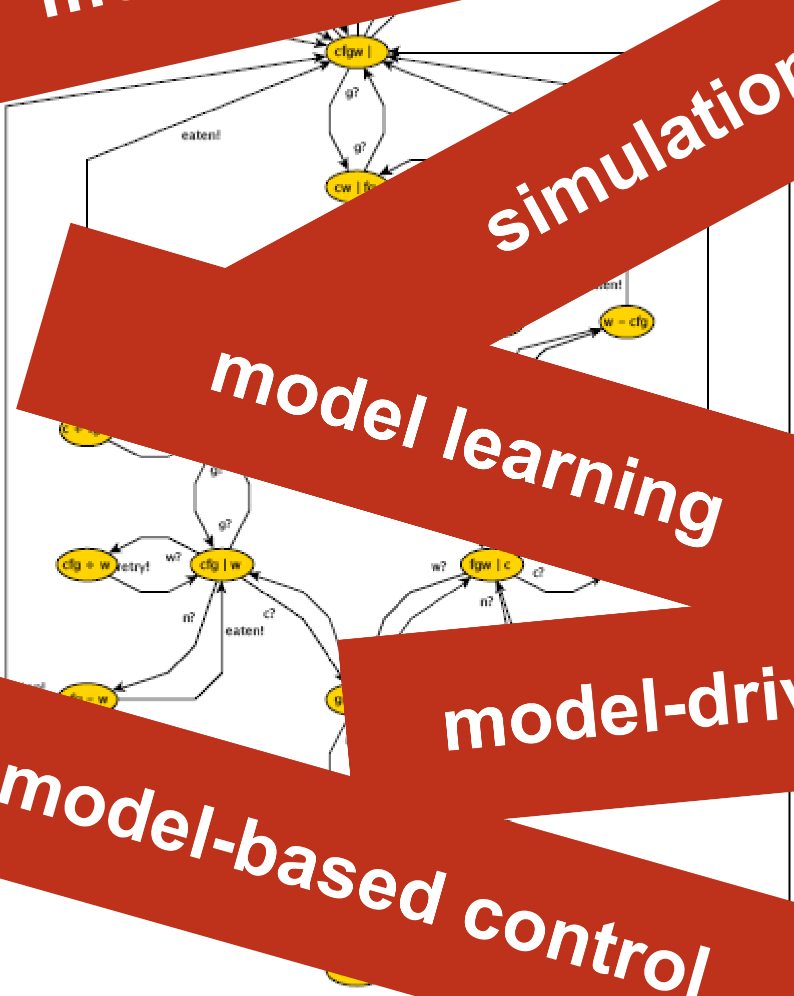
```
STAUTDEF  helloWorld  [ Input, Output :: String ]  ( )
 ::=
   STATE   init, noname, named
   VAR       name :: String
   INIT      init  { name := "" }
```

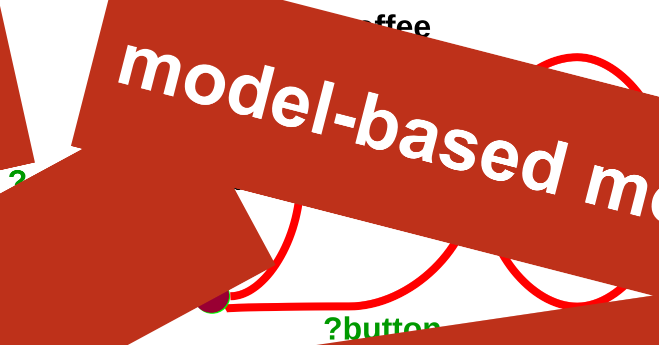

Input ? n
[[strinre(n, REGEX('[A-Z][a-z]+'))]]
{ name := n }
init
Output ! "Hello World!"
noname
named
Output ! "Hello " ++ name ++ "!"

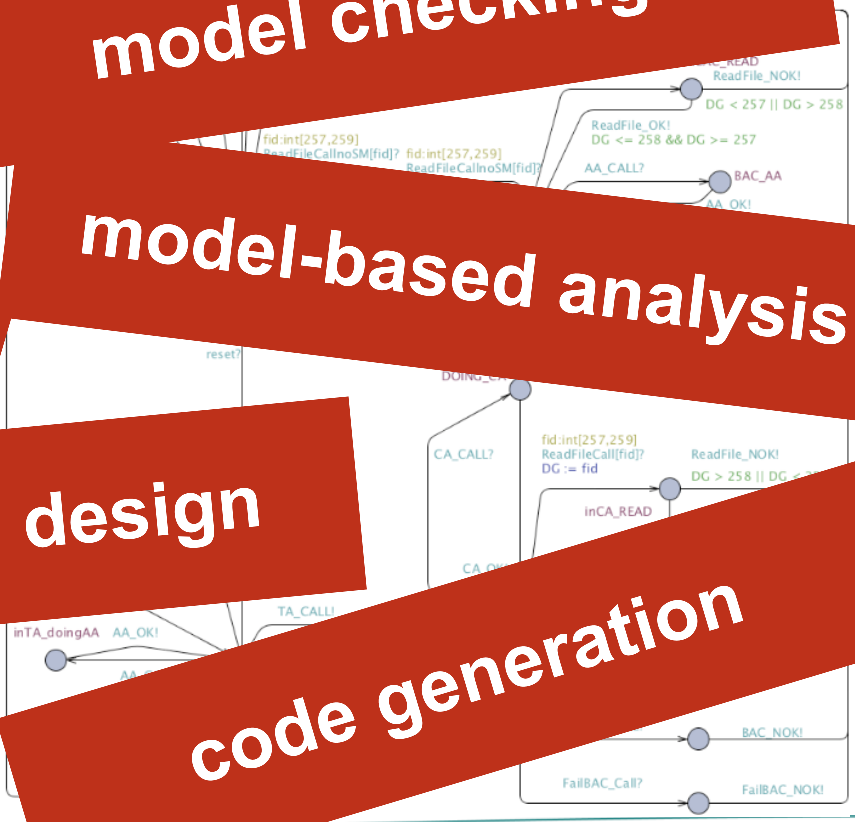
MBT : Example Models



Models







model-based testing

model-based monitoring

simulation

model checking

model learning

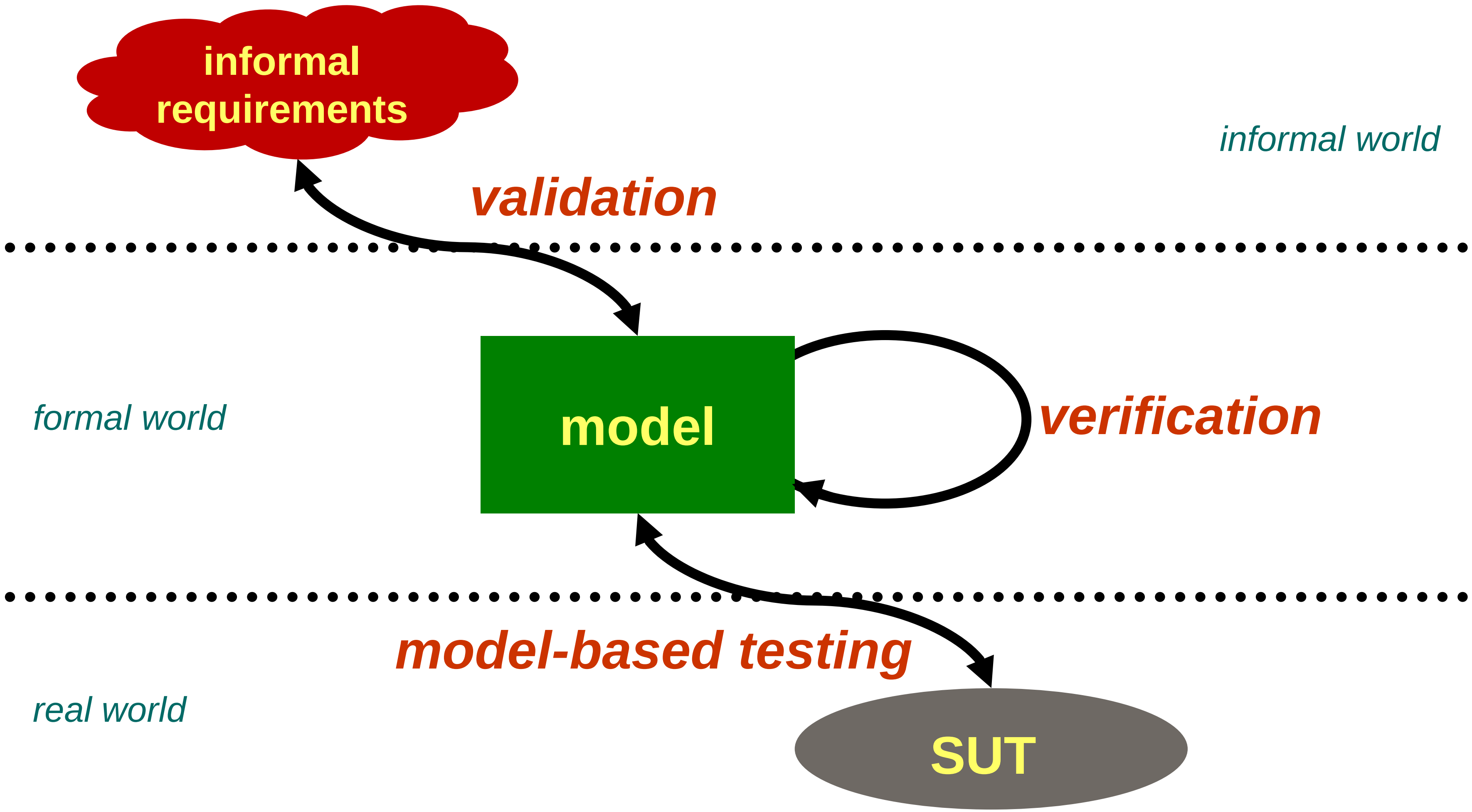
model-based analysis

model-driven design

model-based control

code generation

Validation, Verification, Testing



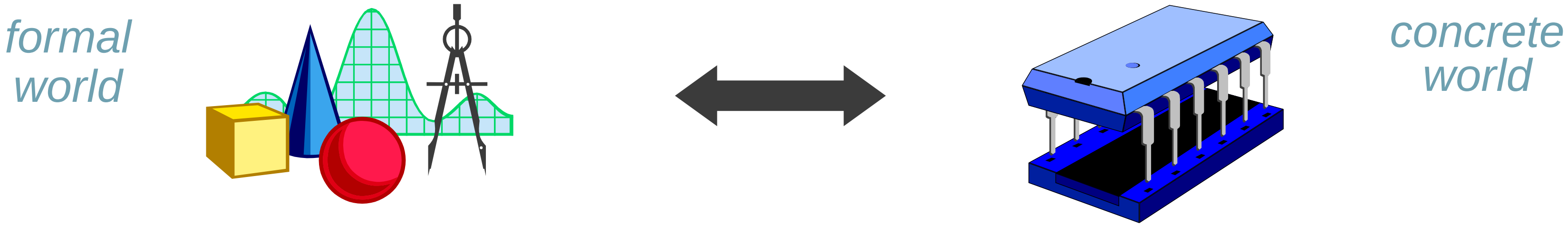
Verification and Testing

Model-based verification :

- formal manipulation
- prove properties
- performed on model

Model-based testing :

- experimentation
- show error
- concrete system



formal world

concrete world

Verification is only as good as the validity of the model on which it is based

Testing can only show the presence of errors, not their absence

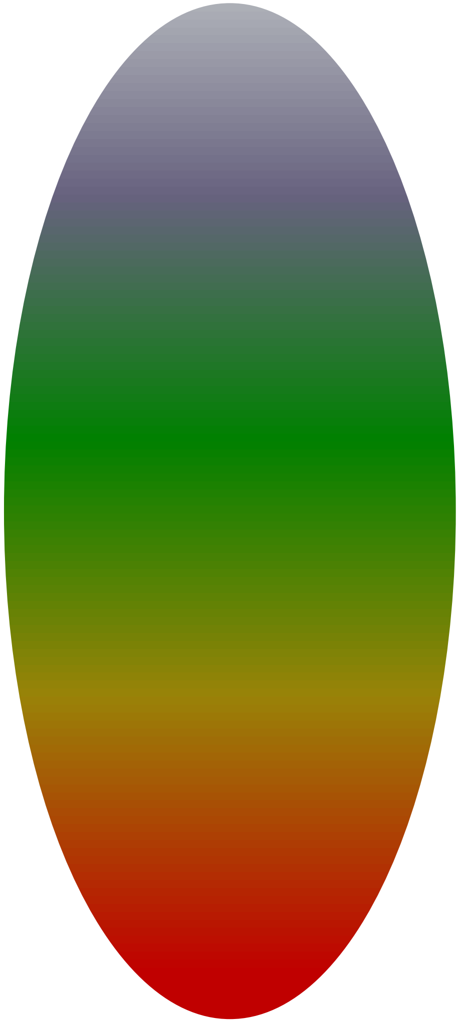
Spectrum of Models

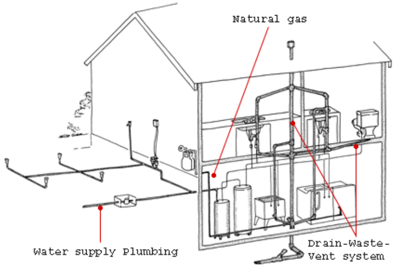
abstract
(test)
models

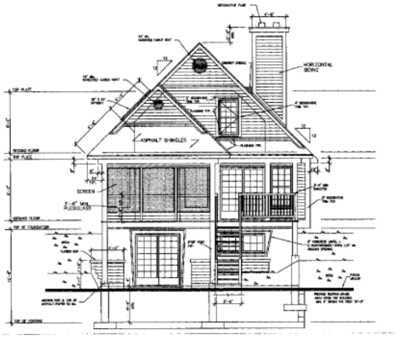
design
models

virtualization

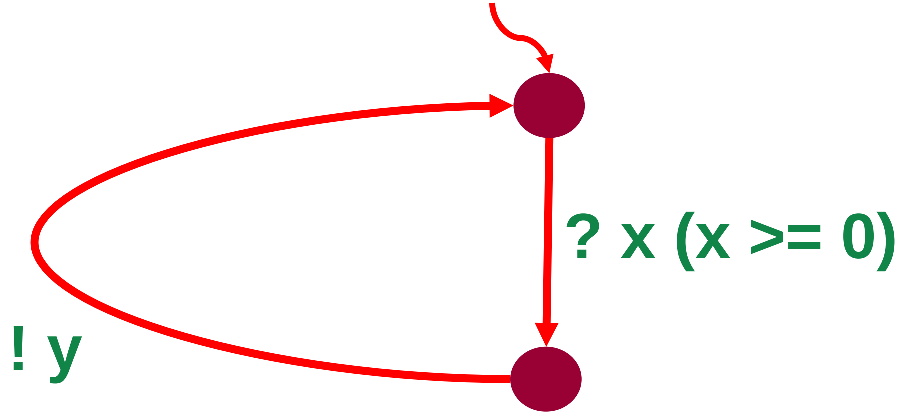
realization







Code Generation from a Model Not Always Possible



? x (x >= 0)

! y

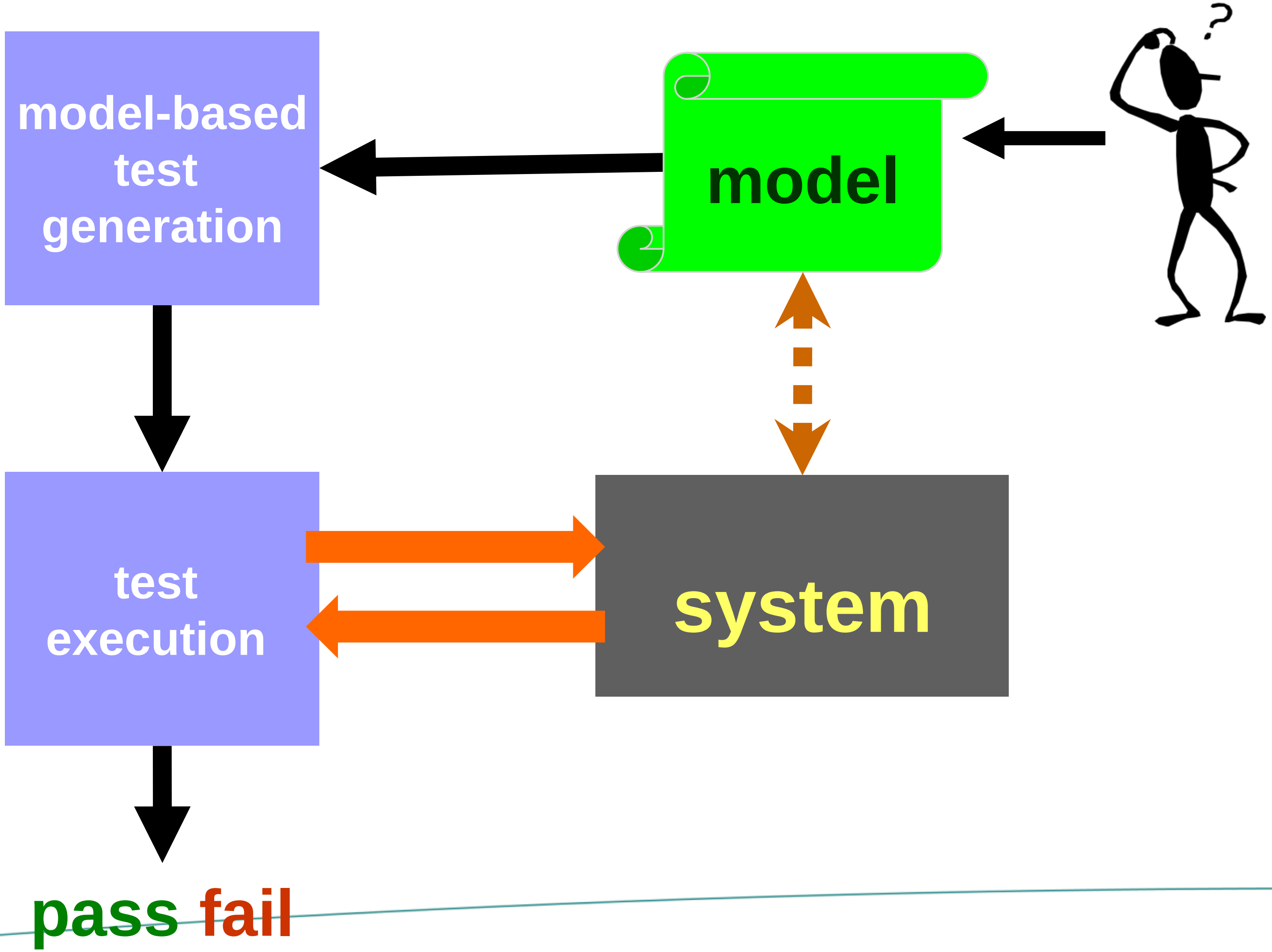
$y \times y = x$

model of $\sqrt{x}$

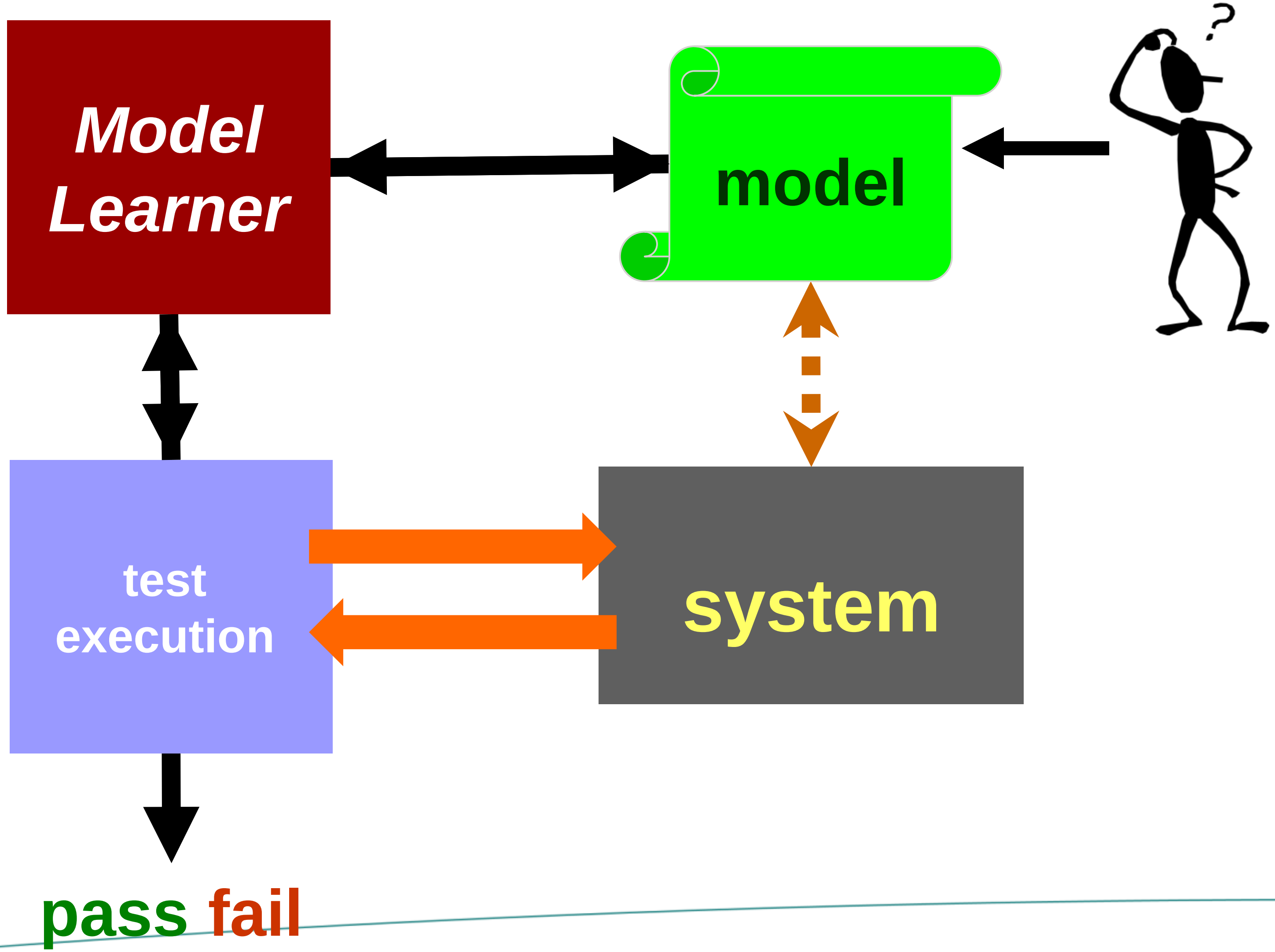
- *specification of **properties** rather than construction*
- ***under-specification***
- ***non-determinism***

Model Learning

MBT : How to Get these D… Models



Research : Model Learning



Model Learner

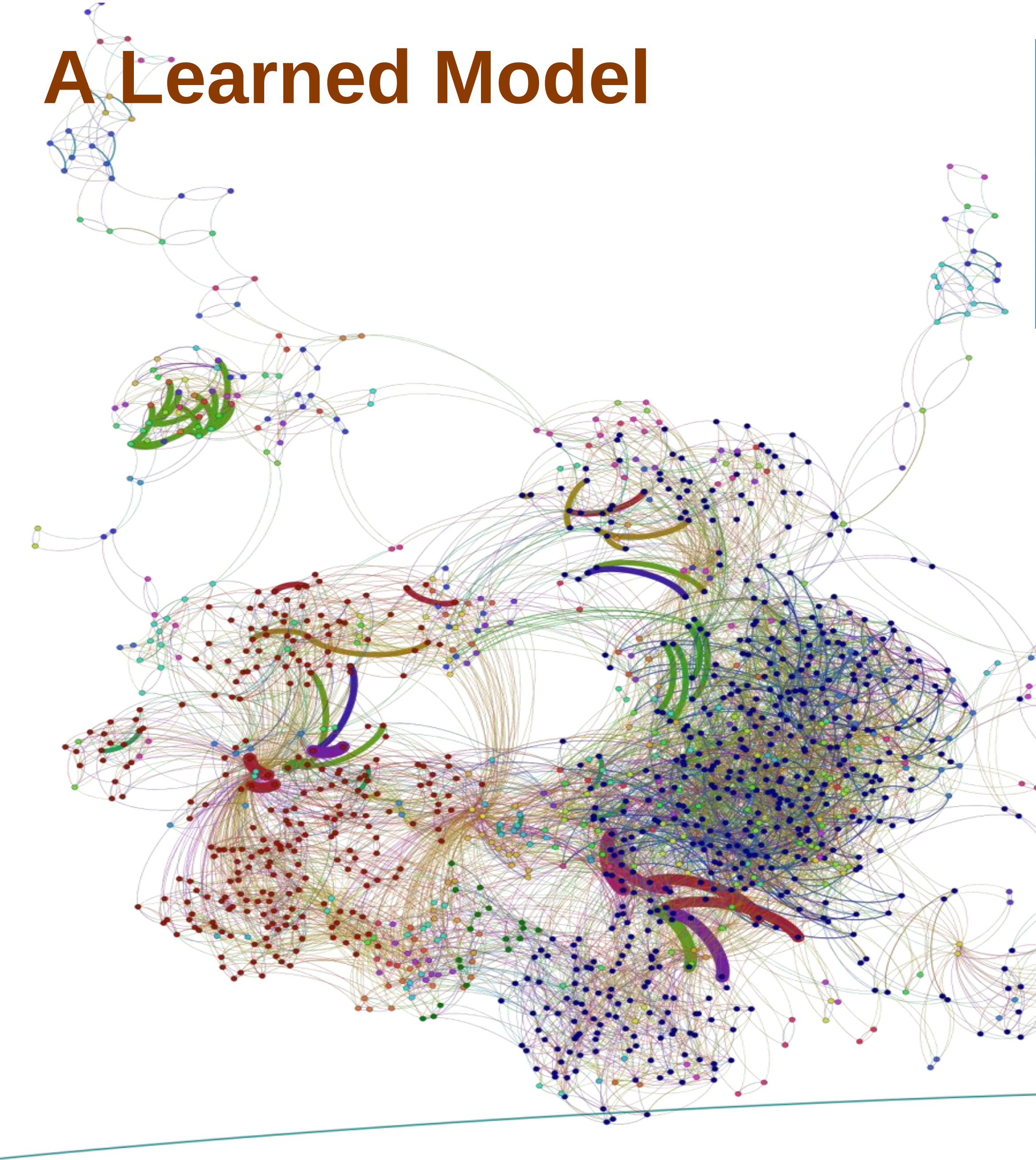
model

test execution

system

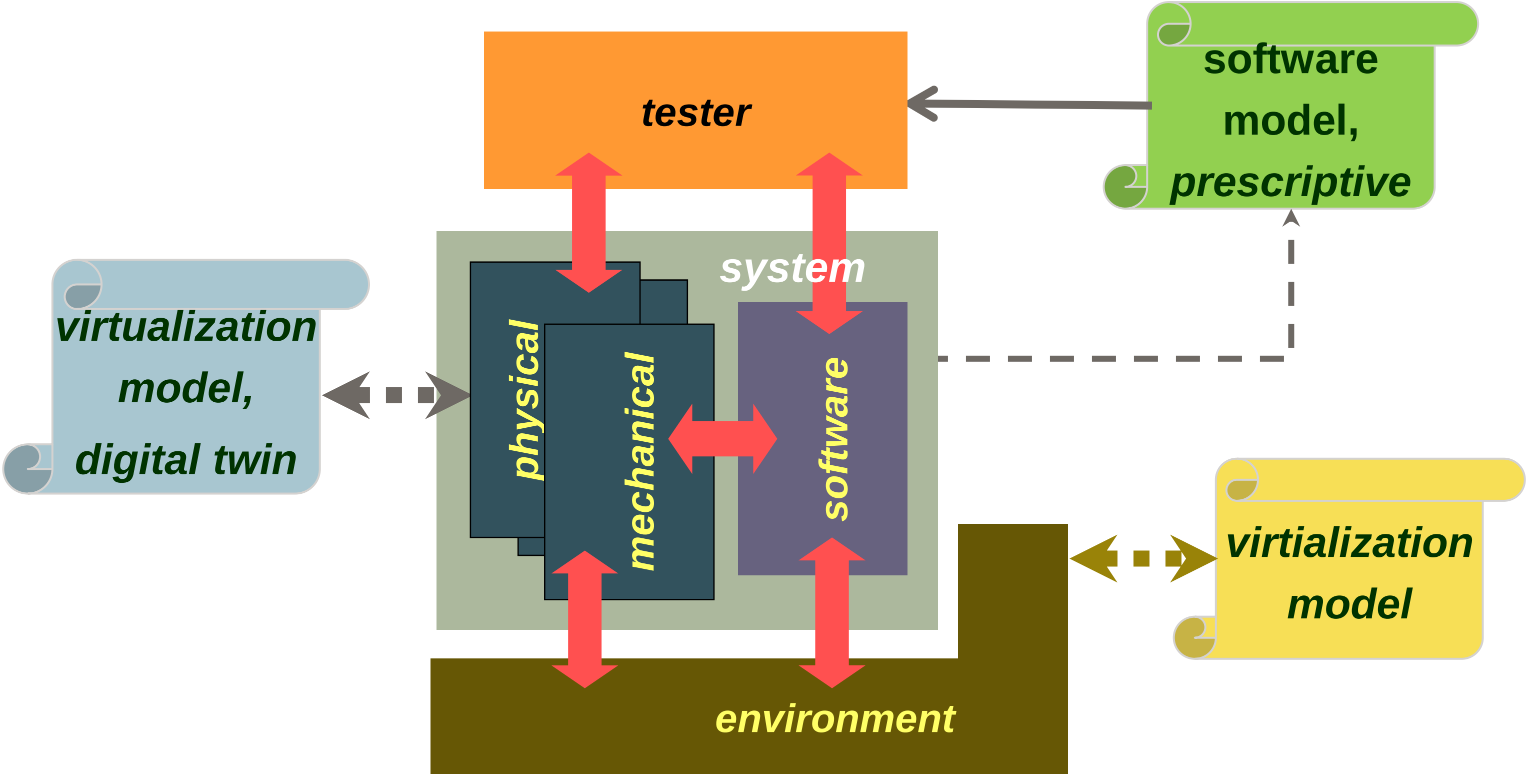
pass fail

A Learned Model



Testing with Models

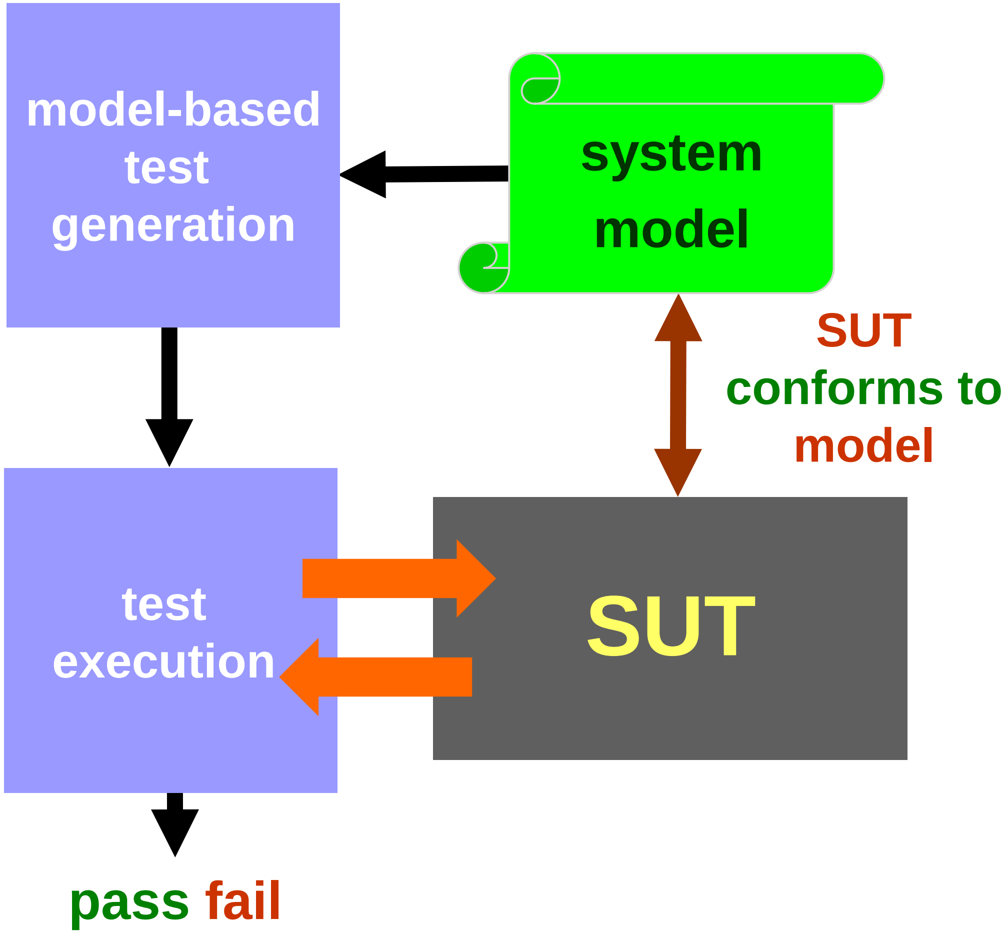
Models for Testing



Model-Based Testing

A Bit of Theory

MBT : Model-Based Testing

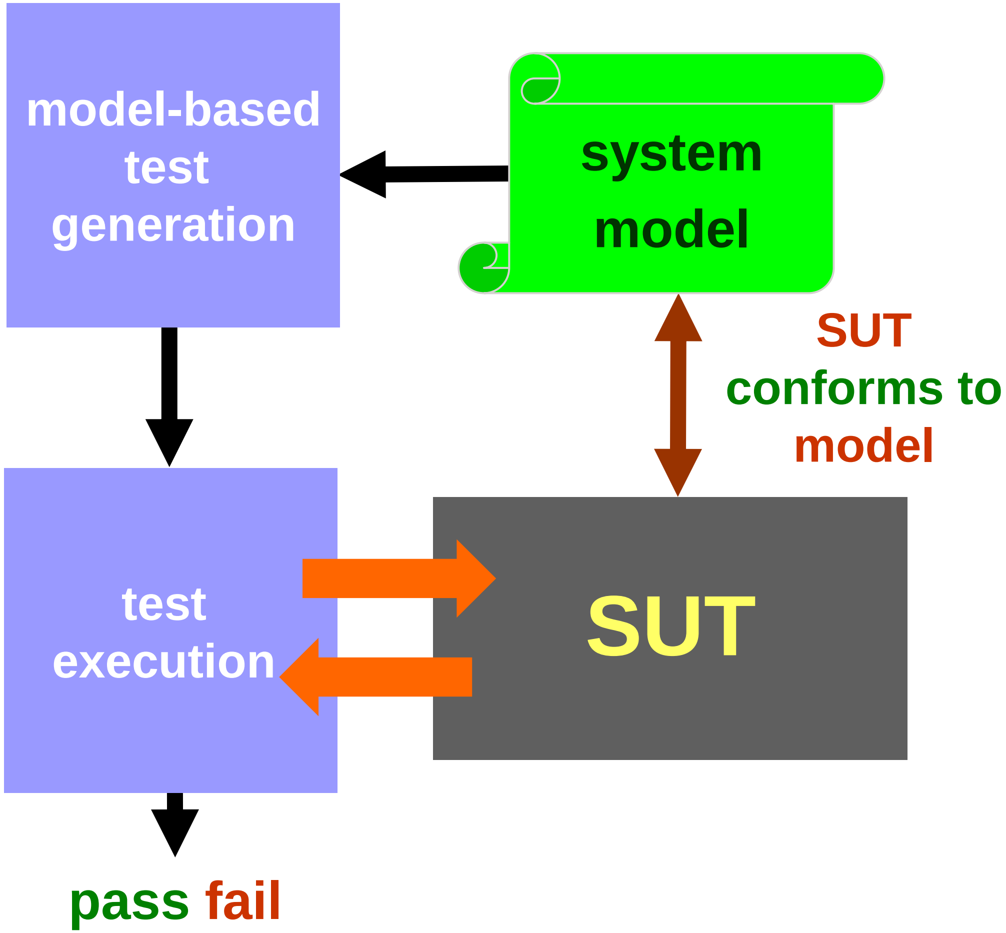


MBT : Model-Based Testing

SUT
conforms to
model

sound ⇓ ⇑ *exhaustive*

SUT passes tests

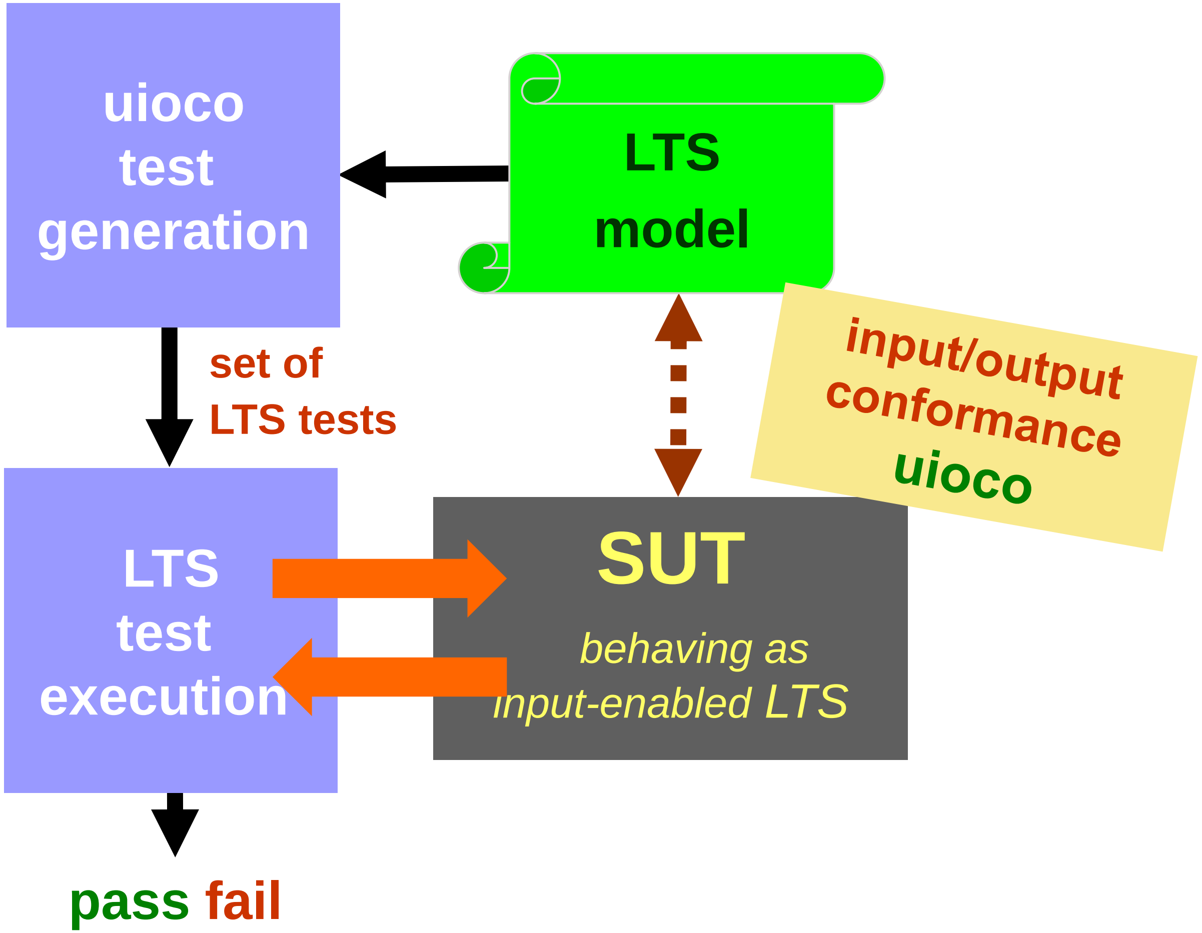


MBT: Labelled Transition Systems & uioco

SUT uioco model

sound ⇓ ⇑ *exhaustive*

SUT passes tests



Models: Labelled Transition Systems

Labelled Transition System: $\langle\ S,\ L_I,\ L_U,\ T,\ s_0\ \rangle$

states

input labels

output labels

transitions

$T \subseteq S \times (L \cup \{\tau\}) \times S$

initial state

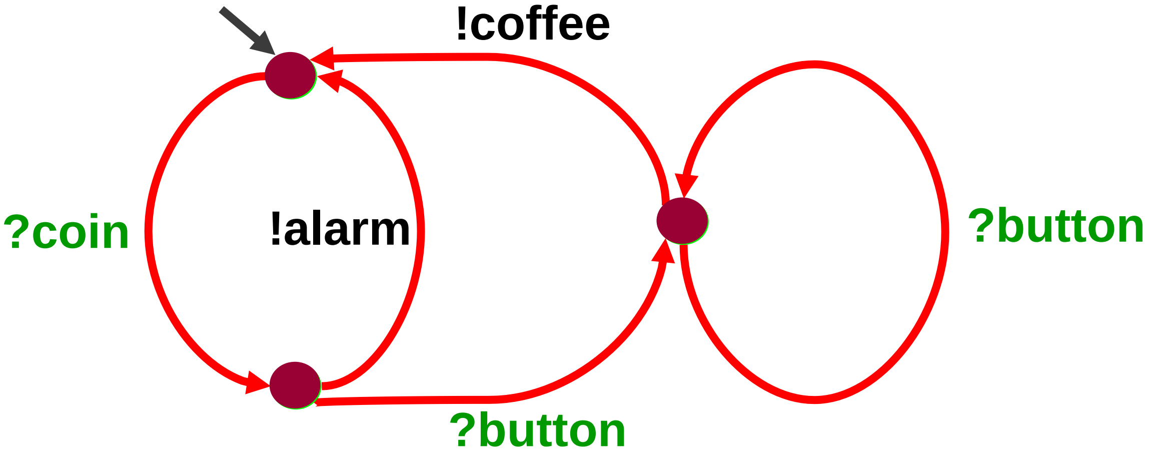
$s_0 \in S$

? = input

! = output

$L = L_I \cup L_U$

$L_I \cap L_U = \emptyset$



Input/Output Conformance : *uioco*

$$i \ \ \textbf{uioco} \ \ s \quad =_{def} \ \forall \sigma \in \textbf{\textit{Utraces}}\,(s) : \ \textbf{\textit{out}}\,(i \textbf{ after } \sigma) \ \subseteq \ \textbf{\textit{out}}\,(s \textbf{ after } \sigma)$$

s is a Labelled Transition System

i is (assumed to be) an input-enabled LTS

$$s \textbf{ after } \sigma \quad = \quad \{ \ s' \mid s \overset{\sigma}{\Longrightarrow} s' \ \}$$

$$s \textbf{ refuses } A \ \Leftrightarrow \ \forall \mu \in A \cup \{\tau\}: \ s \not\xrightarrow{\mu}$$

$$s \xrightarrow{\delta} s \quad \Leftrightarrow \quad s \textbf{ refuses } L_U$$

$$\textbf{\textit{Straces}}\,(s) \quad = \ \{ \ \sigma \ \in (\ L \cup \{\delta\} \ )^* \mid s \overset{\sigma}{\Longrightarrow} \ \}$$

$$\textbf{\textit{Utraces}}\,(s) \quad = \quad \{ \sigma \in \textbf{\textit{Straces}}\,(s) \ \ | \ \ \forall \ \sigma_1 \ ?b \ \sigma_2 = \sigma : \ \textbf{\textit{not}} \ (\ s \ \textbf{ after } \sigma_1 \textbf{ refuses } \ \{?b\} \ ) \ \}$$

$$\textit{out} \ (\ P\ ) \ = \{ \ !x \in L_U \mid \exists \, p \in P : p \xrightarrow{!x} \} \ \cup \ \{ \ \delta \mid \exists \, p \in P : p \xrightarrow{\delta} p \ \}$$

Input/Output Conformance : *uioco*

$$i \ \textbf{uioco} \ s \quad =_{def} \quad \forall \sigma \in \textit{Utraces}(s) : \ \textit{out}(i \text{ after } \sigma) \subseteq \textit{out}(s \text{ after } \sigma)$$

s is a Labelled Transition System

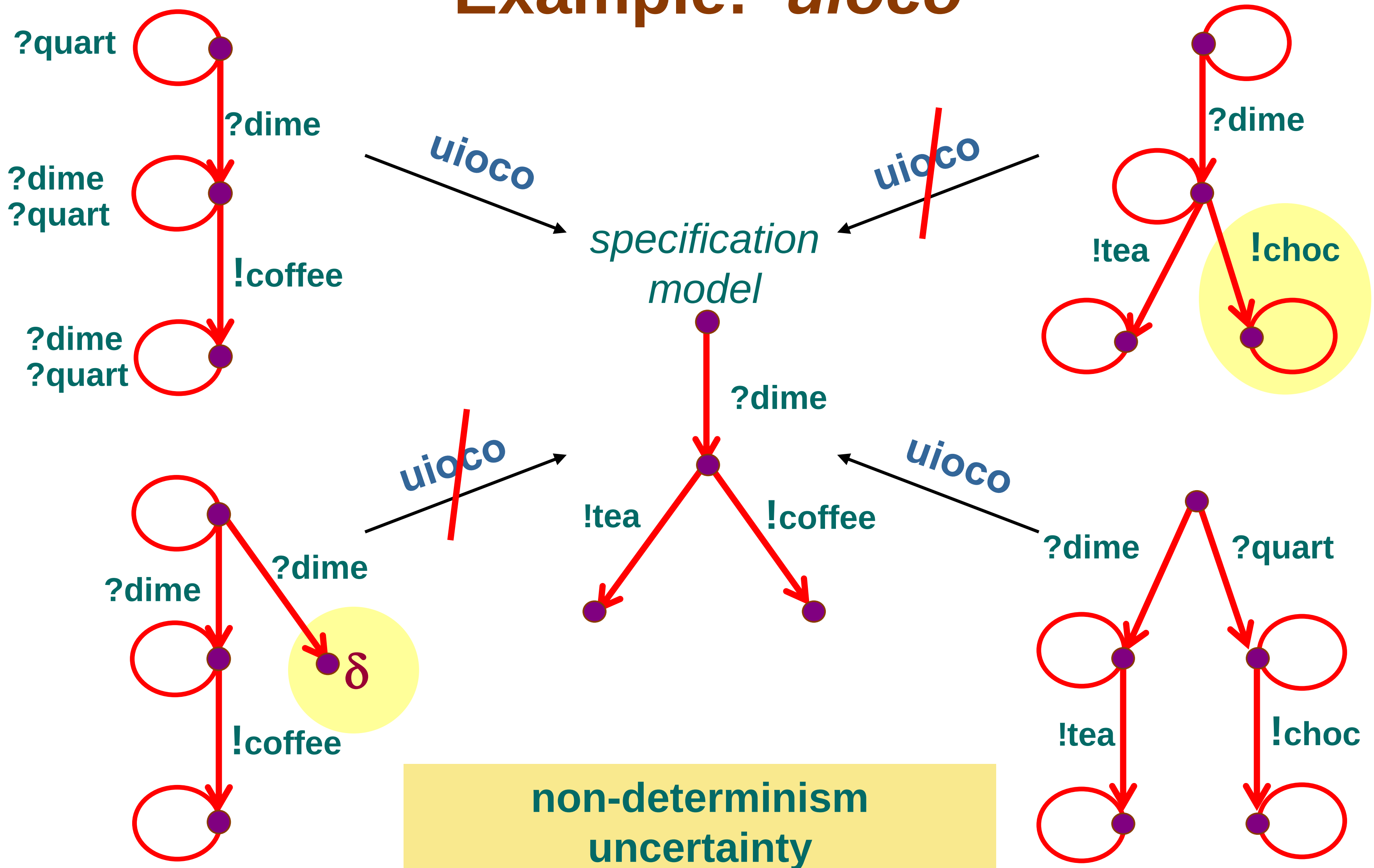
i is (assumed to be) an input-enabled LTS

Intuition:

i **uioco**-conforms to **s**, iff

- if **i** produces output **x** after *U*-trace σ,
  then **s** can produce **x** after σ
- if **i** cannot produce any output after *U*-trace σ,
  then **s** cannot produce any output after σ (called *quiescence* δ)

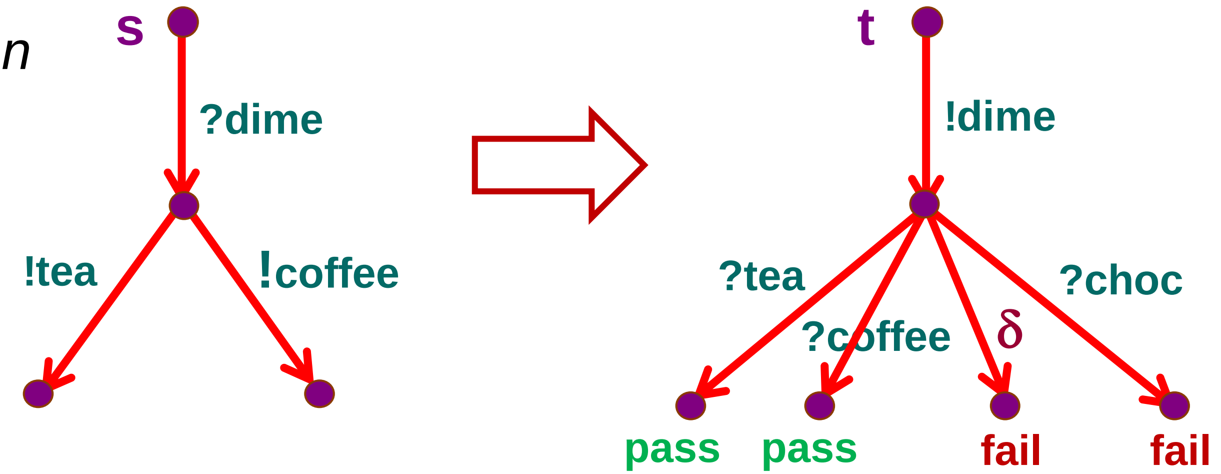
Example: *uioco*



non-determinism
uncertainty
under-specification

Example: *uioco* Test Generation

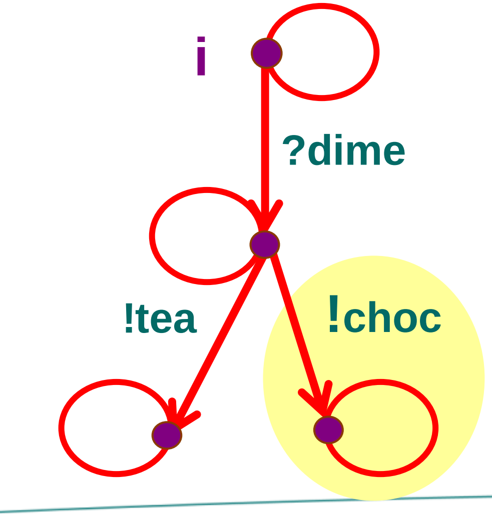
specification model



generated test case

i $\not$uioco s

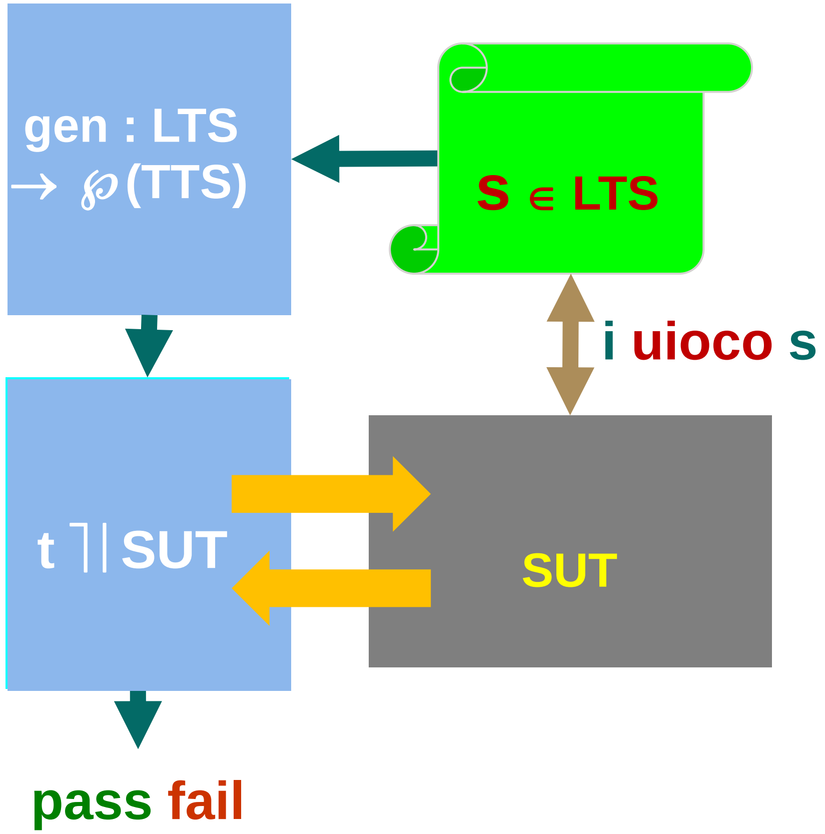
i fails t



implementation

i uioco s $=_{def}$
∀ σ ∈ *Utraces* (s) :
out (i after σ) ⊆ *out* (s after σ)

MBT with *uioco* is Sound and Exhaustive



gen : LTS → ℘(TTS)

s ∈ LTS

i uioco s

t ⌉| SUT

SUT

pass fail

Testability assumption :

$\forall \mathbf{SUT} \in IMP \,.\, \exists \mathbf{m}_{\mathbf{SUT}} \in IOTS \,.$

$\forall t \in TESTS \,.$

$\mathbf{SUT\ passes}\ t \Leftrightarrow \mathbf{m}_{\mathbf{SUT}}\ \mathbf{passes}\ t$

Prove soundness and exhaustiveness:

$\forall \mathbf{m} \in IOTS \,.$

$(\ \forall t \in gen(\mathbf{s}) \,.\, \mathbf{m\ passes}\ t\ )$

$\Leftrightarrow \quad \mathbf{m\ uioco\ s}$

SUT comforms to s

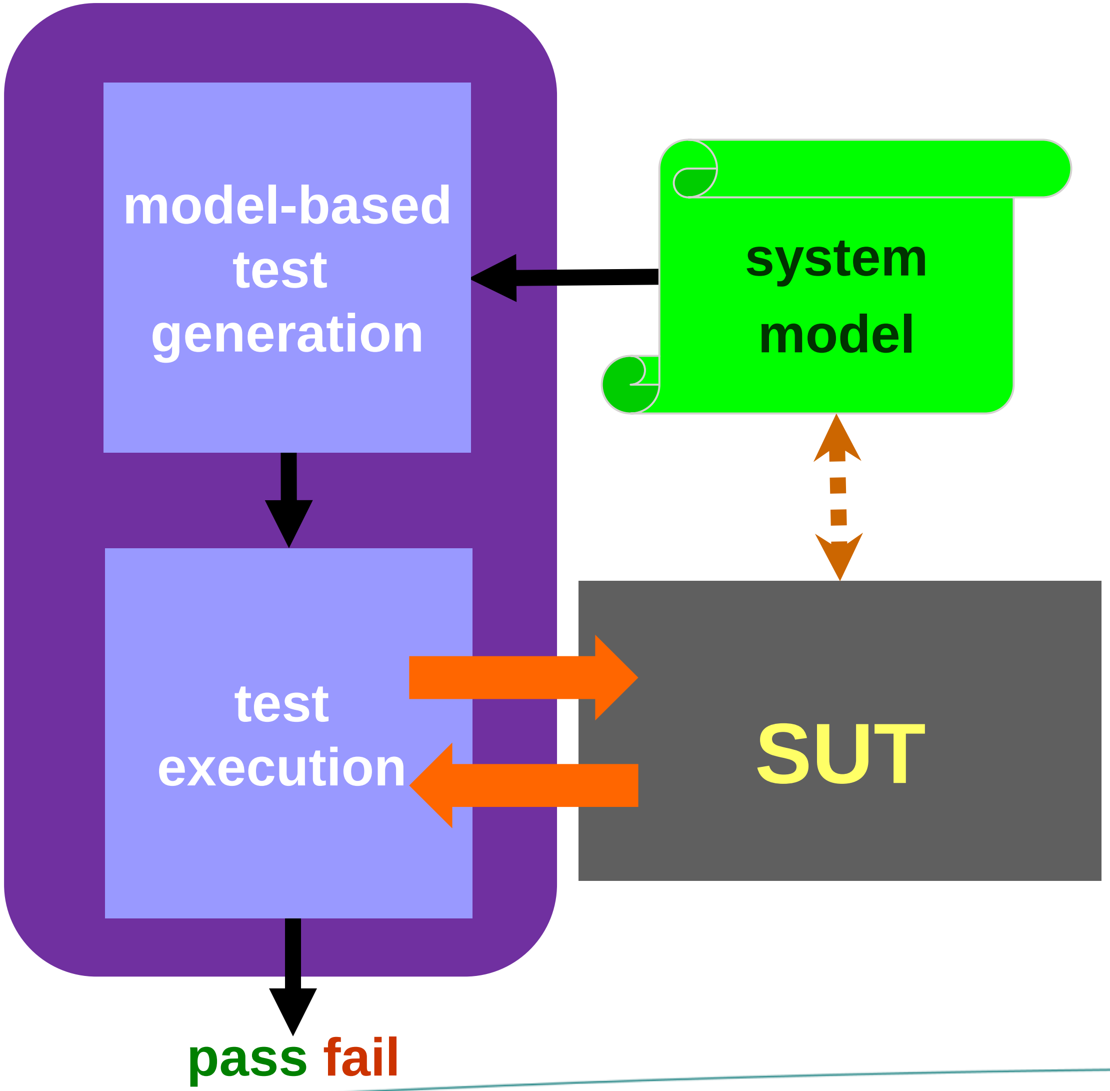
exhaustive ⇑ ⇓ *sound*

SUT passes gen(s)

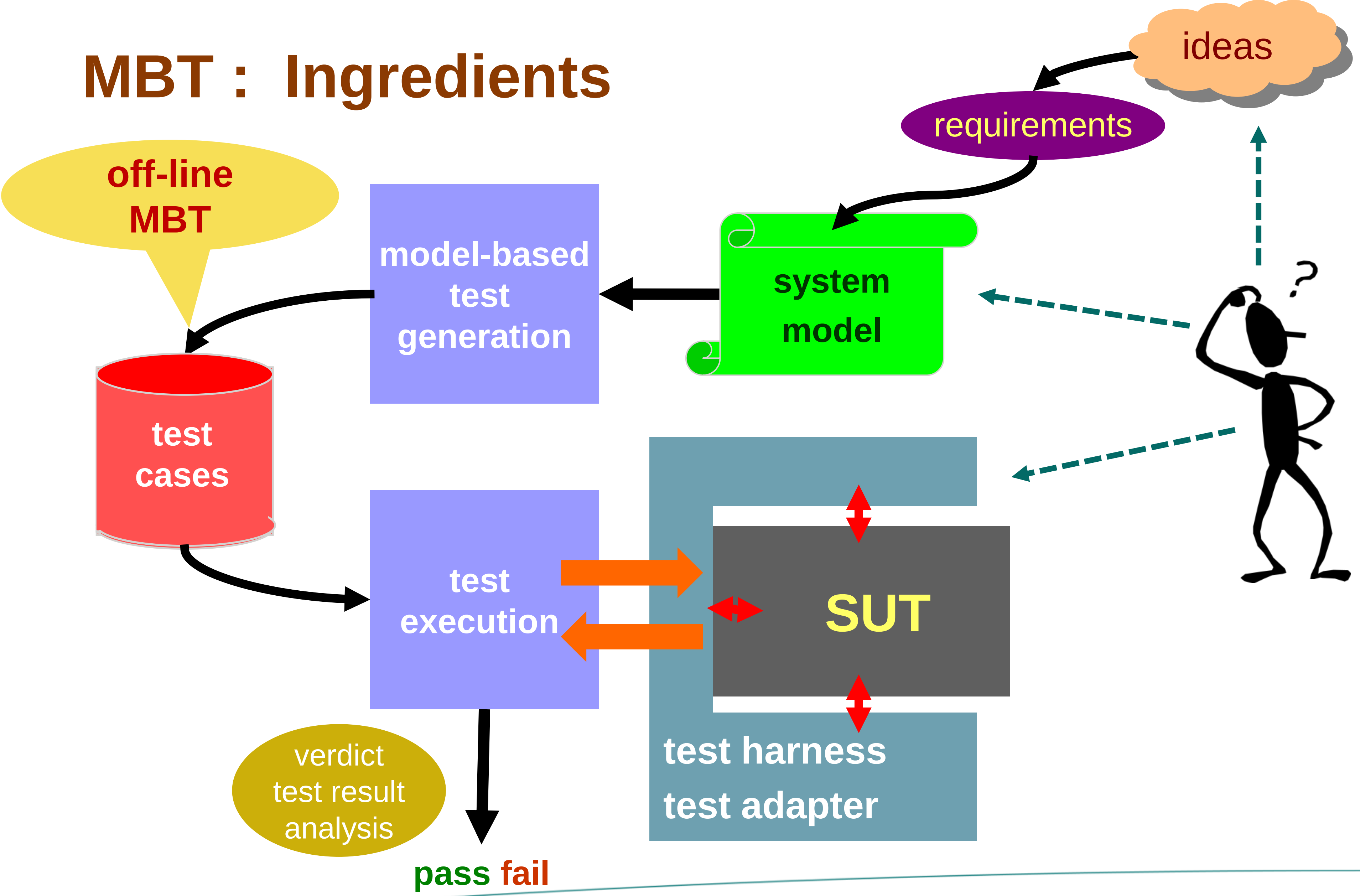
Model-Based Testing

A Tool’s View

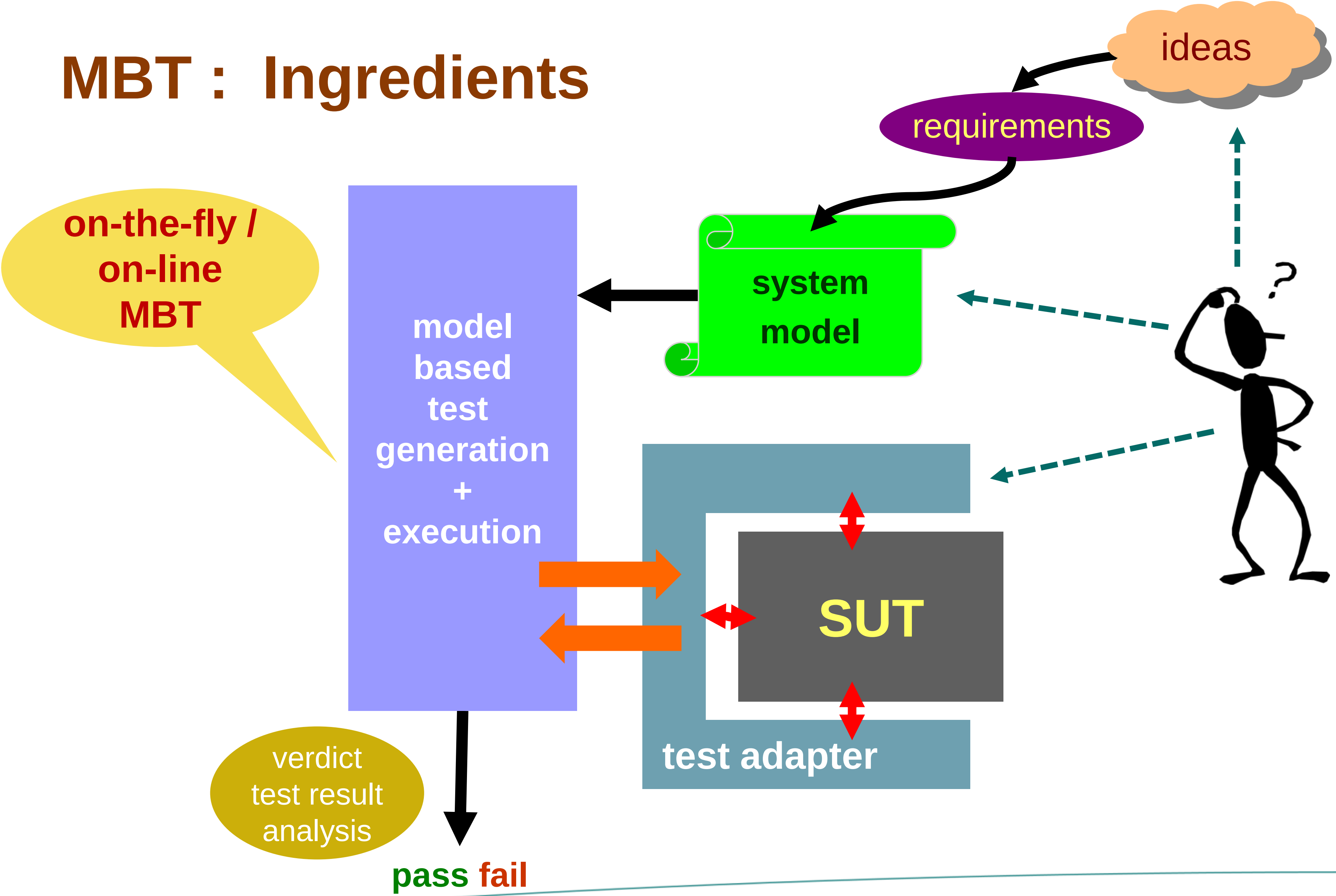
MBT Tool


model-based test generation
system model
test execution
SUT
pass fail

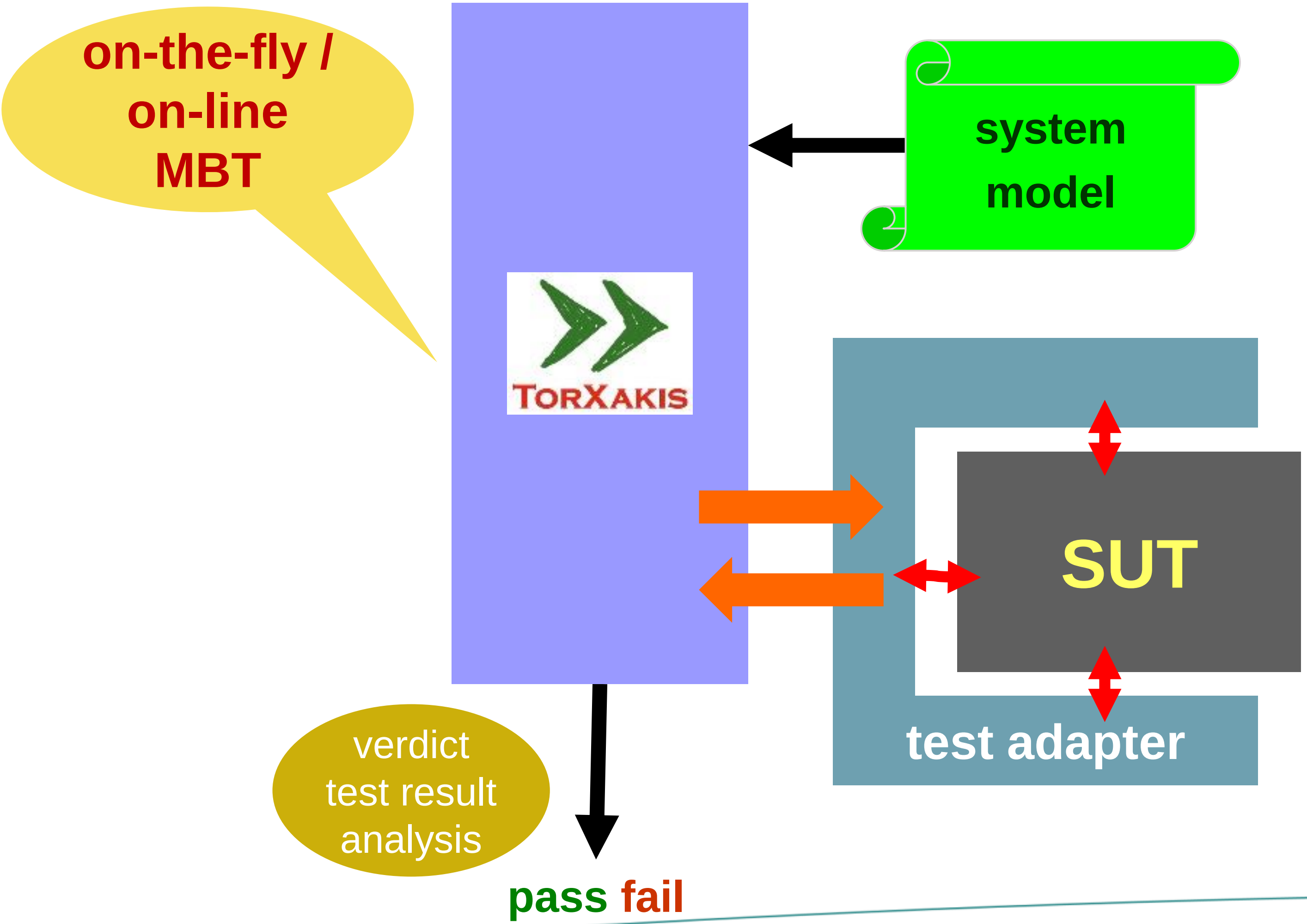
MBT : Ingredients



MBT : Ingredients



TorXakis : An On-the Fly MBT Tool



TorXakis : Overview

Models

- state-based control flow and complex data
- support for parallel, concurrent systems
- composing complex models from simple models
- non-determinism, uncertainty
- abstraction, under-specification

Applications

- several high-tech systems companies
- experimental level

But ....

- research prototype
- poor usability

Tool

- on-line MBT tool

Current Research

- test selection
- partial models & variability

Under the hood

- powerful constraint/SMT solvers (Z3, CVC4)
- well-defined semantics and algorithms
- **ioco** testing theory
  for symbolic transition systems
- algebraic data-type definitions

MBT : Many Tools

- AETG
- Agatha
- Agedis
- Autolink

Axini Test Manager

- Conformiq
- Cooper
- Cover
- DTM
- fMBT
- G∀st
- Gotcha
- Graphwalker
- JTorX
- MaTeLo
- MBTsuite

- M-Frame
- MISTA
- NModel
- OSMO
- ParTeG
- Phact/The Kit
- PyModel
- QuickCheck
- Reactis
- Recover
- RT-Tester
- SaMsTaG
- Smartesting CertifyIt
- Spec Explorer
- StateMate
- STG

- tedeso
- Temppo
- TestCompass
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TestOptimal
- TGV
- Tigris
- TorX

TorXakis

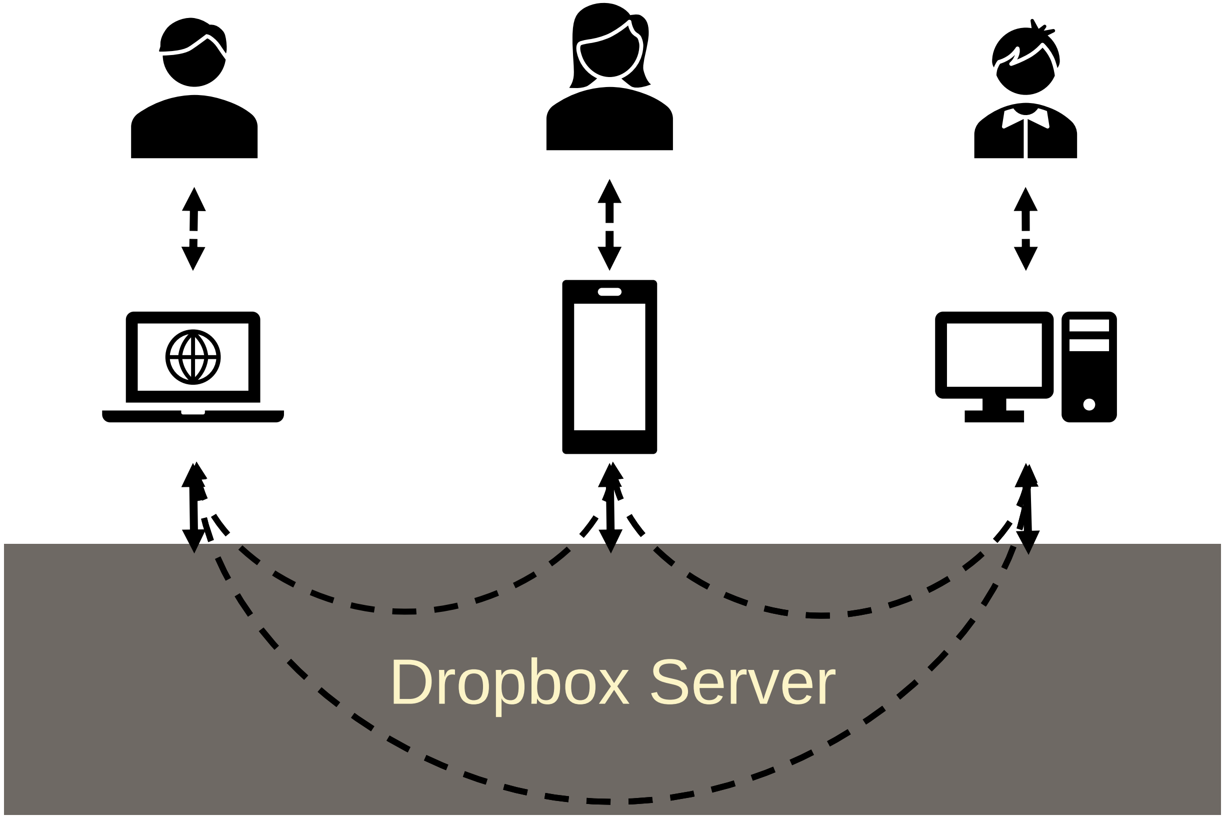
- T-Vec
- Tveda
- Uppaal-Cover
- Uppaal-Tron
- . . . .

Model-Based Testing

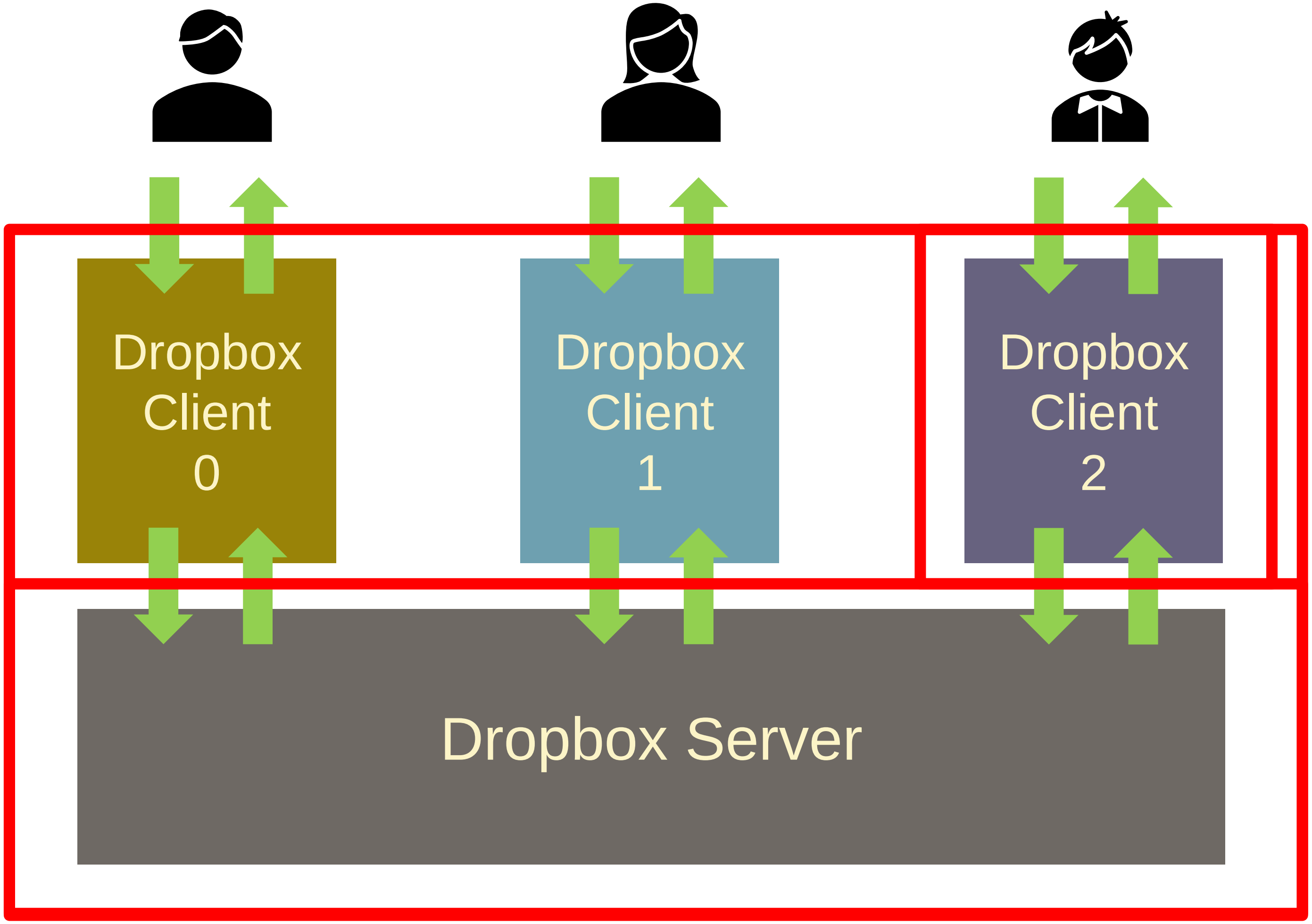
Example :

Testing *Dropbox* with *TorXakis*

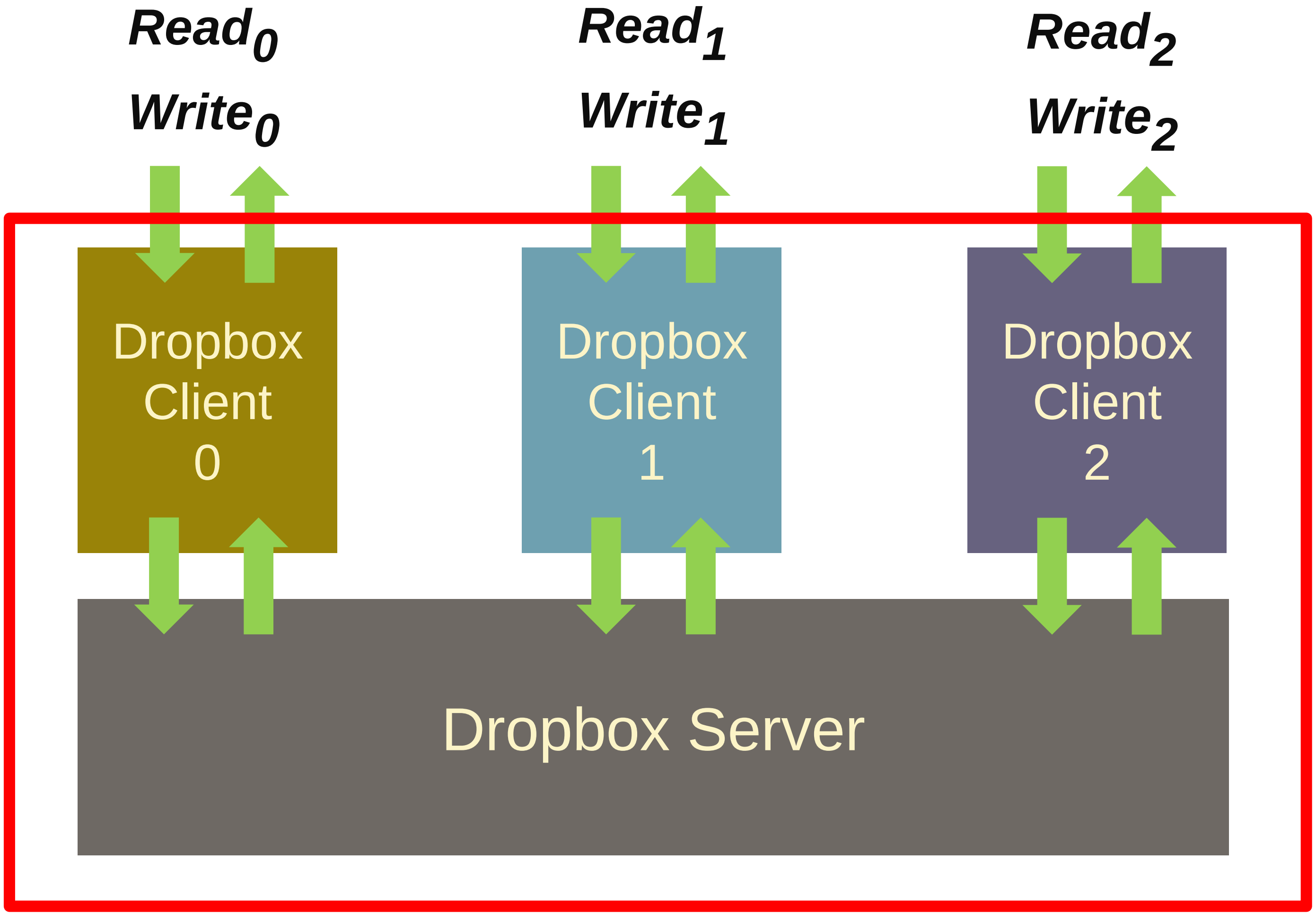
Dropbox : A File Synchronization Service



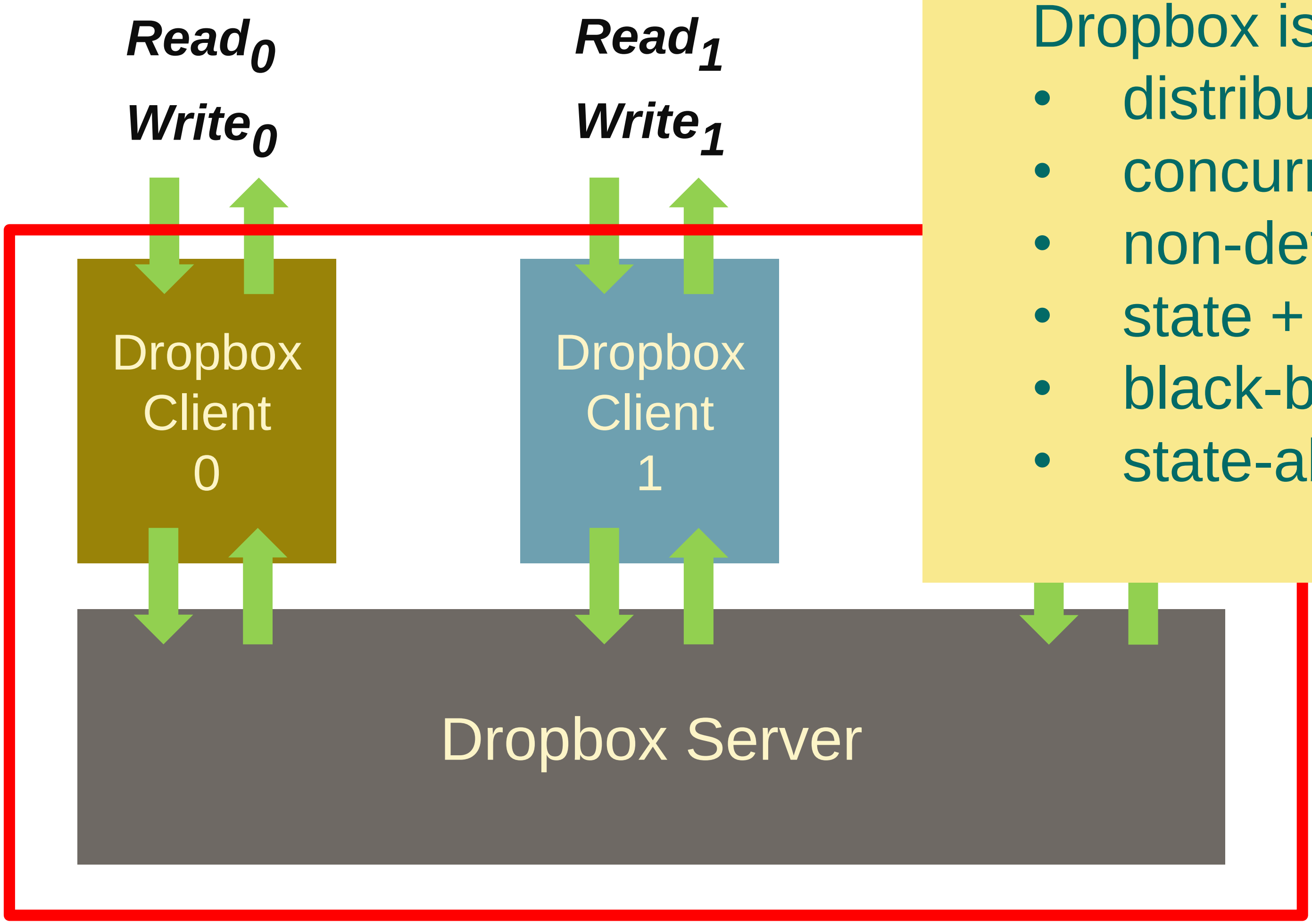
Testing Dropbox



Testing Dropbox



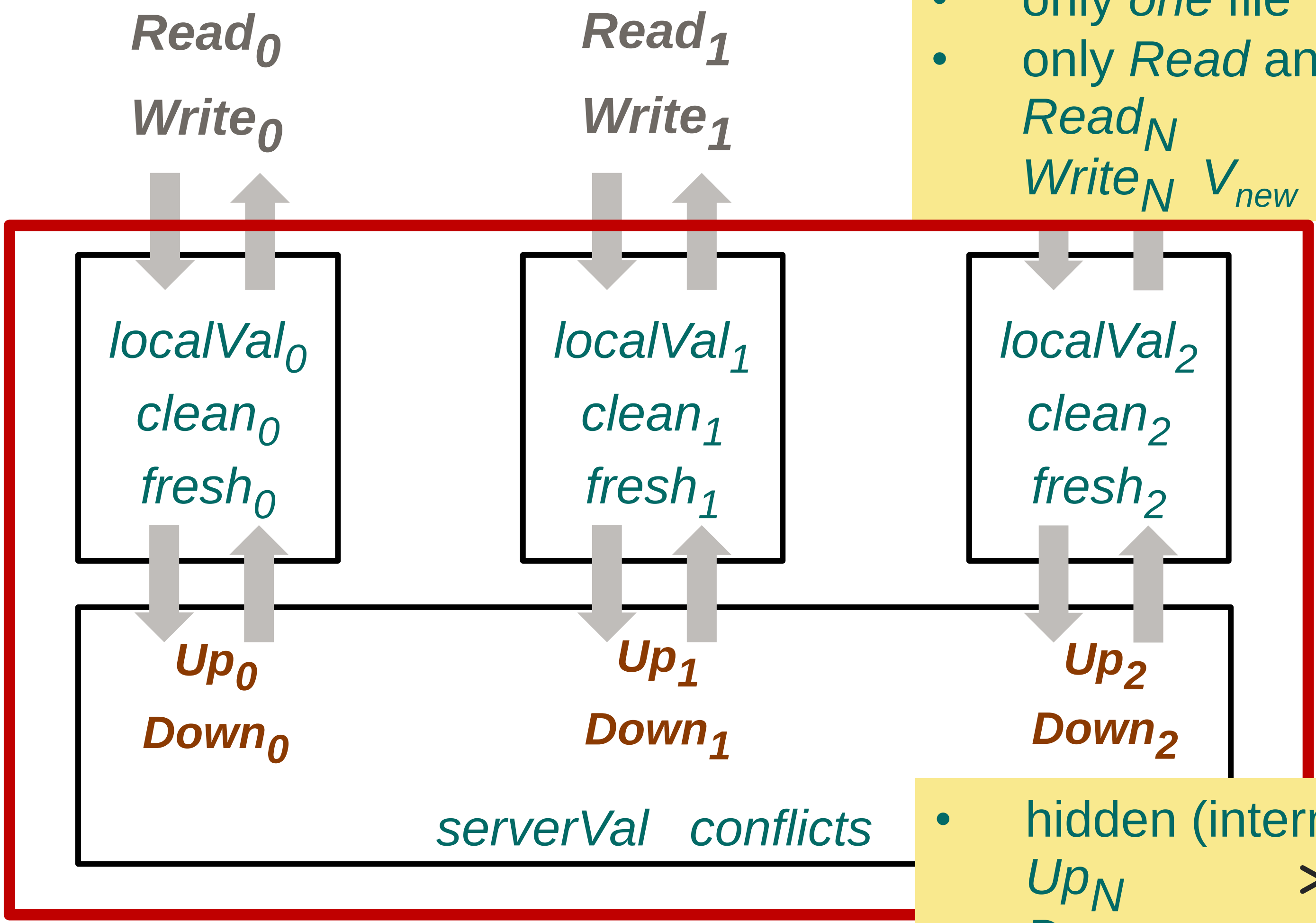
Testing Dropbox



Dropbox is

- distributed
- concurrent
- non-deterministic
- state + data
- black-box
- state-abstracted

A Dropbox Model (*Hughes et al.*)



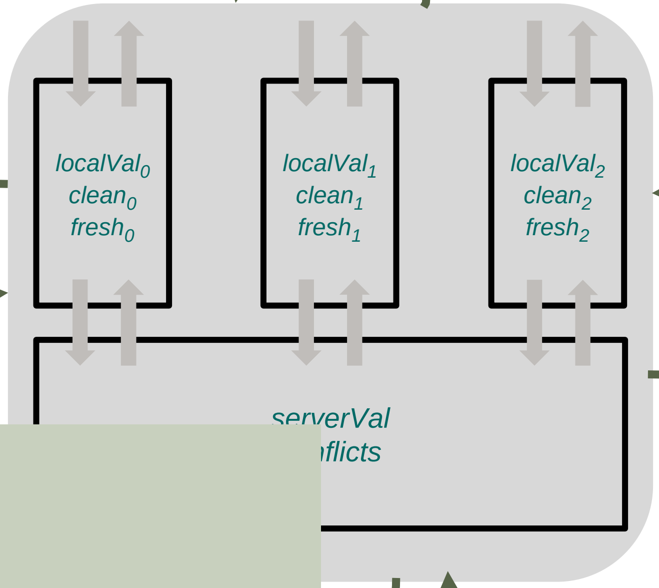
- only *one* file
- only *Read* and *Write* :
  $Read_N$ >–> *file value*
  $Write_N$ V_{new} >–> *old file value*

- hidden (internal) actions
  Up_N >–> *upload file*
  $Down_N$ >–> *download file*

A Dropbox Model

$In\ ?\ Read_N$
$Out\ !\ localVal_N$

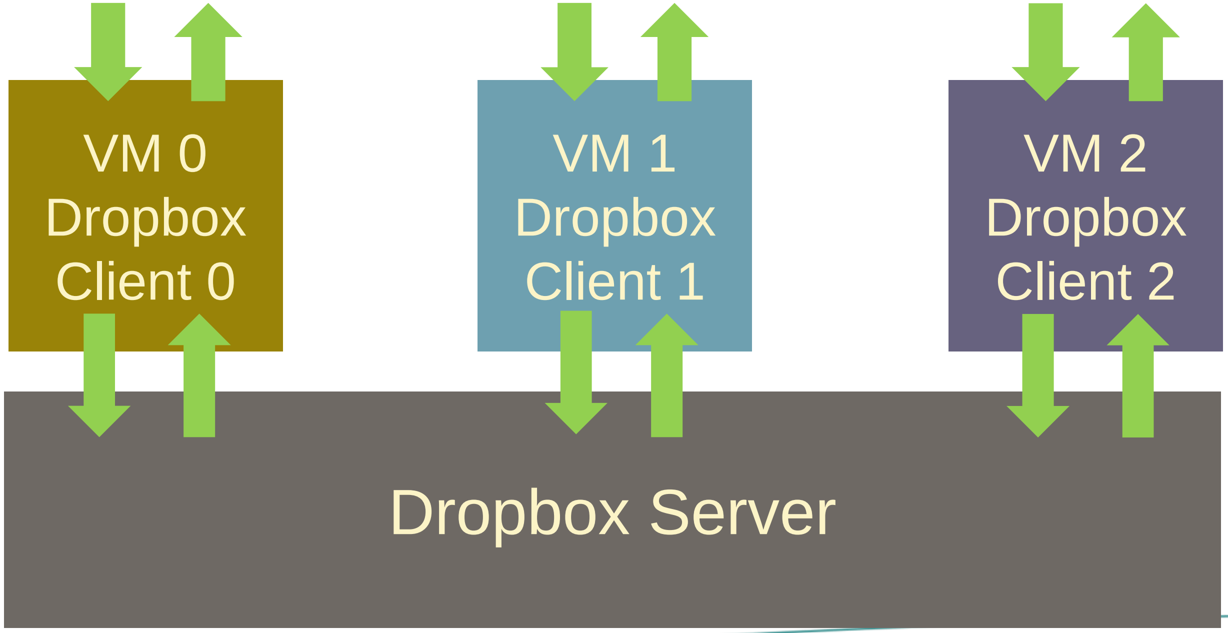
$In\ ?\ Write_N\ V_{new}$
$Out\ !\ localVal_N$
$\quad localVal_N\ ::=\ V_{new}$
$\quad clean_N\ ::=\ False$



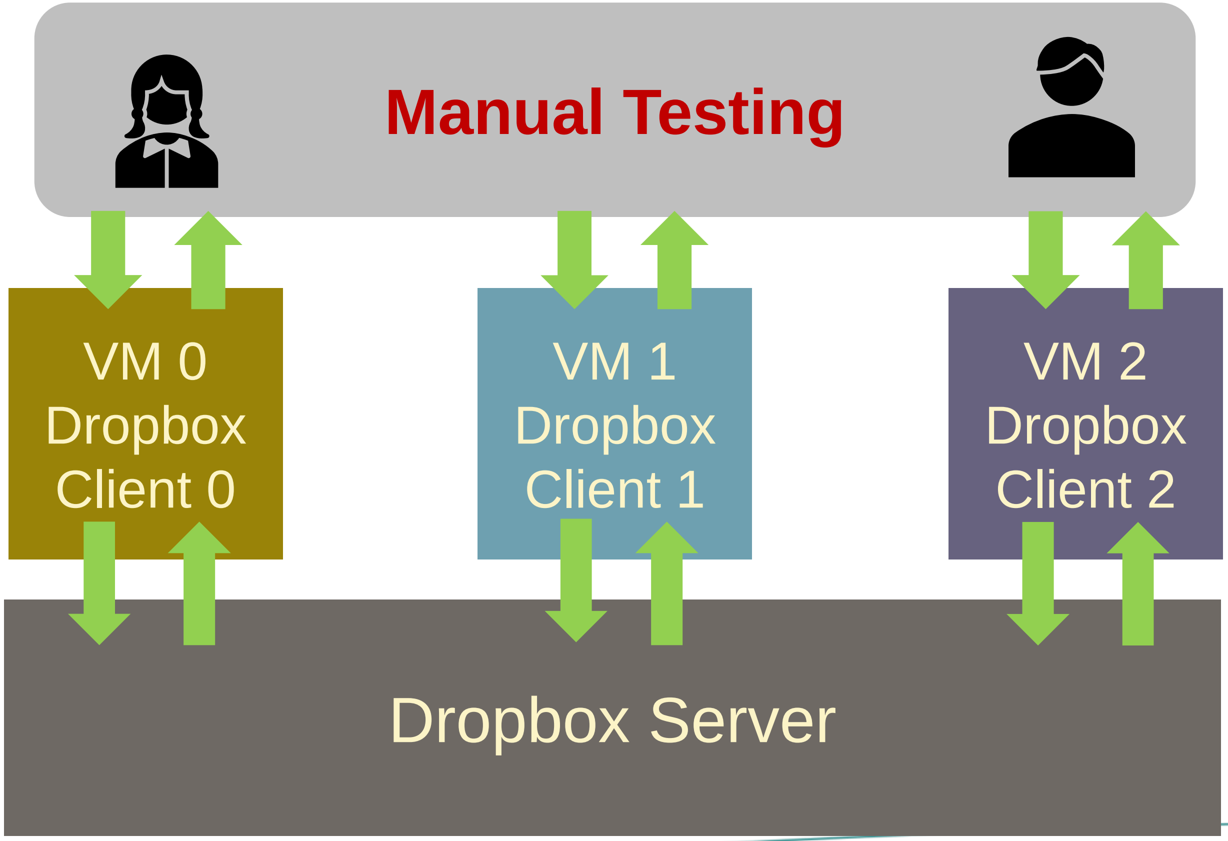
```
$\neg\ clean_N \rightarrow$
$Up_N$
    $clean\ ::=\ True$
    $if\ fresh_N\ then$
      $if\ localVal_N \neq serverVal\ then$
        $fresh_{N'}\ ::=\ False\ \ for\ all\ N' \neq N$
        $serverVal\ ::=\ localVal_N$
  $else\ if\ localVal_N \notin \{\ serverVal, \perp\}\ then$
          $conflicts\ ::=\ conflicts \cup \{\ localVal_N\}$
```

$\neg\ fresh_N \wedge clean_N \rightarrow$
$Down_N$
$\quad localVal_N\ ::= serverVal$
$\quad fresh_N\ ::=\ True$

Testing Dropbox



Testing Dropbox


Manual Testing

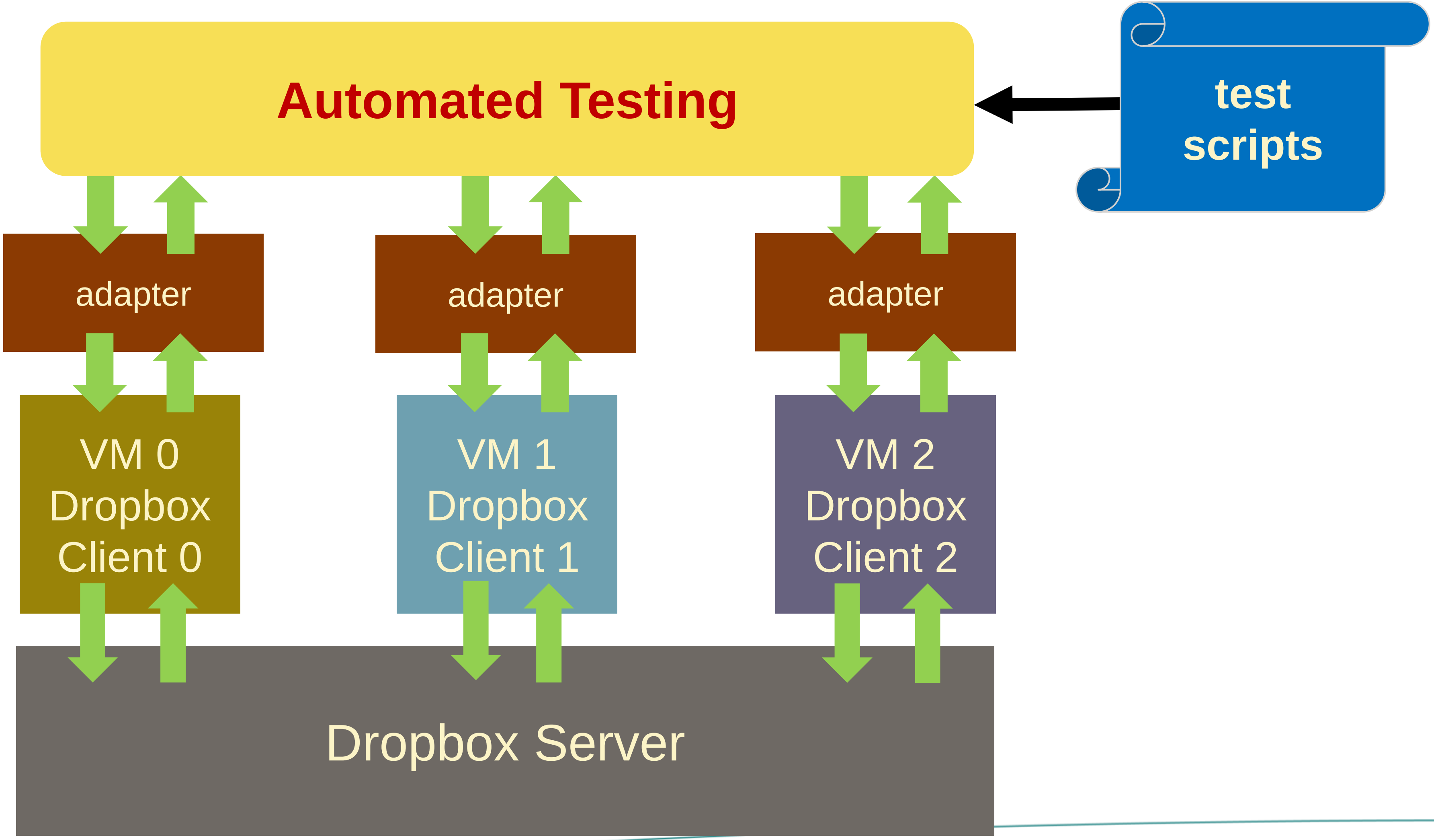
VM 0
Dropbox
Client 0

VM 1
Dropbox
Client 1

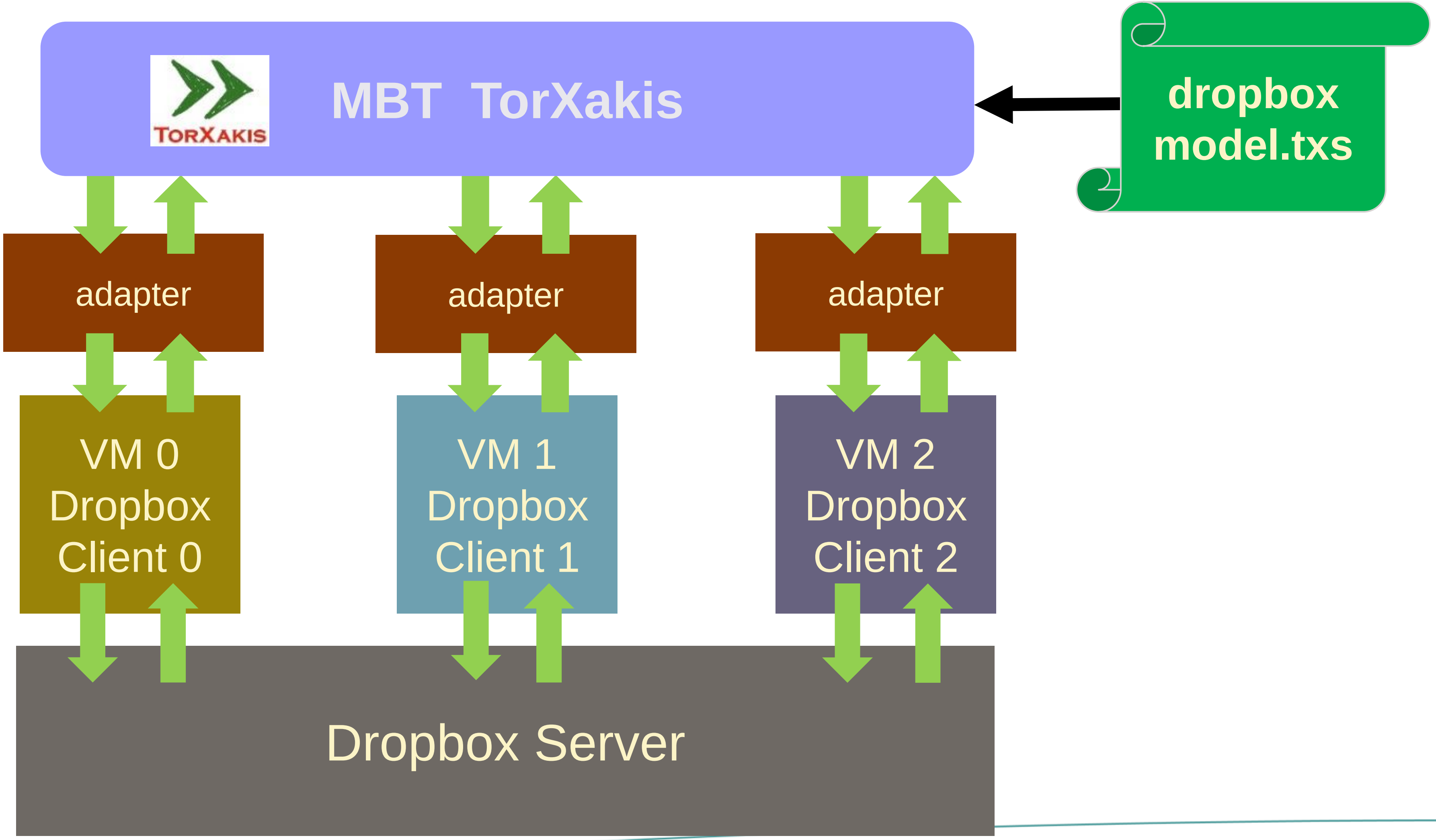
VM 2
Dropbox
Client 2

Dropbox Server

Testing Dropbox



Testing Dropbox



Dropbox Test Run

```
TXS >> ...5: IN:   In1  ! Write(Value("P"))
TXS >> ...6: OUT:  Out1 ! File(Value("$"))
TXS >> ...7: IN:   In0  ! Write(Value("SHK"))
TXS >> ...8: OUT:  Out0 ! File(Value("$"))
TXS >> ...9: IN:   In1  ! Read
TXS >> ..10: OUT:  Out1 ! File(Value("P"))
```

Dropbox Test Run

```
TXS >> ..11: IN:   In0  ! Write(Value("X"))
TXS >> ..12: OUT:  Out0 ! File(Value("SHK"))
TXS >> ..13: IN:   In2  ! Write(Value("A"))
TXS >> ..14: OUT:  Out2 ! File(Value("$"))
TXS >> ..15: IN:   In2  ! Write(Value("SP"))
TXS >> ..16: OUT:  Out2 ! File(Value("A"))
TXS >> ..17: IN:   In1  ! Write(Value("BH"))
TXS >> ..18: OUT:  Out1 ! File(Value("P"))
TXS >> ..19: IN:   In2  ! Read
TXS >> ..20: OUT:  Out2 ! File(Value("SP"))
TXS >> ..21: IN:   In0  ! Read
TXS >> ..22: OUT:  Out0 ! File(Value("X"))
TXS >> ..23: IN:   In2  ! Write(Value("PXH"))
TXS >> ..24: OUT:  Out2 ! File(Value("X"))
```

Dropbox Test Run

```
TXS >>  ..77: IN:   In0  ! Stabilize
TXS >>  ..78: OUT:  Out0 ! File(Value("P"))
TXS >>  ..79: OUT:  Out0 ! File(Value("L"))
TXS >>  ..80: OUT:  Out0 ! File(Value("TK"))
TXS >>  ..81: OUT:  Out0 ! File(Value("P"))
TXS >>  Expected:
( { Out0[ $"Out0"$1266 ] }
, ( IF isFile($"Out0"$1266)
    THEN isValueInList(["PH","H"],value($"Out0"$1266))
    ELSE False
    FI
  )
)
TXS >>  FAIL:  Out0 ! File(Value("P"))
```

An On-the-Fly MBT Tool : TorXakis

Installation

Follow: *TorXakis : Getting Started*

in: *Model-Based Testing and TorXakis – A Tutorial*

or on: https://torxakis.org → Getting Started

Radboud Universiteit

HÖGSKOLAN I HALMSTAD

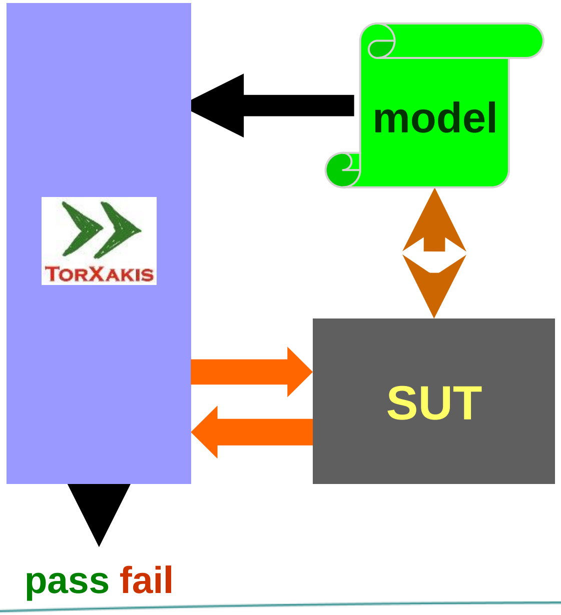
ESI

Model-Based Testing

The Next Step in Test Automation ! ?

New software testing methodologies are needed if testing shall keep up with software development.

Model-based testing may be one of them.



Jan Tretmans
jan.tretmans@tno.nl

TNO innovation for life