Model-Based Testing

There is Nothing More Practical
than a Good Theory

Jan Tretmans

TNO – ESI, Eindhoven, NL
and Radboud University, Nijmegen, NL

Jan Tretmans

Embedded Systems Innovation BY TNO

TNO
Embedded Systems Innovation
Eindhoven
The Netherlands

Groningen
Friesland
Drenthe
Noord-Holland
Flevoland
Overijssel
Utrecht
Gelderland
Zuid-Holland
RU
Noord-Brabant
Zeeland
ESI
Limburg

Radboud University
Nijmegen
The Netherlands

TNO innovation for life

Noldus Information Technology

PHILIPS
Consumer Electronics
Medical Systems
Research
Applied Technologies

NXP founded by Philips

THALES

Technolution AUTOMATION TECHNOLOGY

ASML

FEI COMPANY Tools for Nanotech

océ

VanDerLande INDUSTRIES

CHESS

Imtech ICT

TASS software professionals

DEMCON advanced mechatronics

Universiteit Leiden

TU/e technische universiteit eindhoven

Radboud Universiteit Nijmegen

RuG

TUDelft Delft University of Technology

imec

University of Twente The Netherlands

KATHOLIEKE UNIVERSITEIT LEUVEN

vrije Universiteit amsterdam

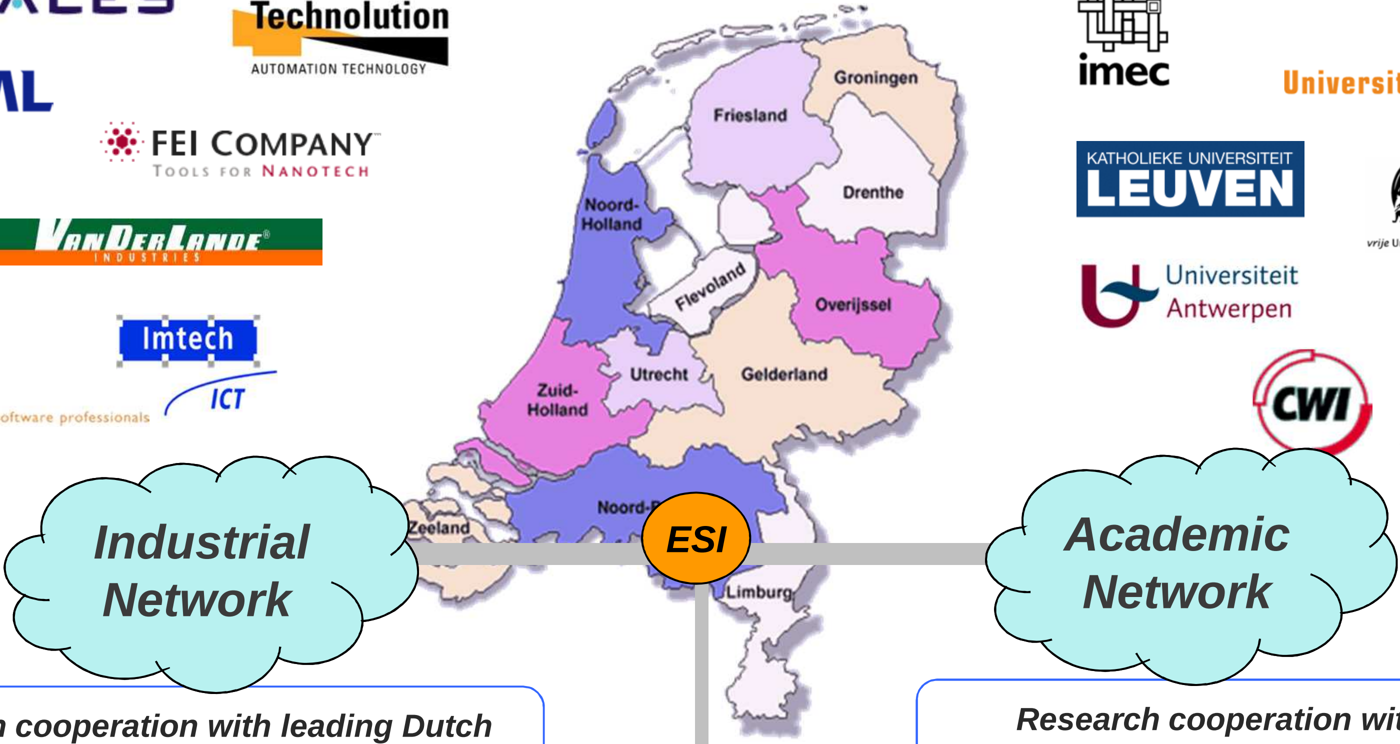
Universiteit Antwerpen

Universiteit Maastricht

CWI

Universiteit van Amsterdam

DESIGN TECHNOLOGY INSTITUTE



Research cooperation with leading Dutch high-tech multinational industries & SME's

Research cooperation with all Dutch universities with embedded systems research

Research cooperation in EU projects

Overview

Model-Based Testing

Theory

Jan Tretmans

- MBT: What and Why
- MBT: A theory with labelled transition systems and ioco
- Variations:
  - Test selection
  - Test-based modelling

Model-Based Testing

Practice

Machiel van der Bijl

- MBT: Practical exercises with Axini Test Manager
- MBT: The difference between theory and practice

Model-Based Testing

(Software) Testing

checking or measuring

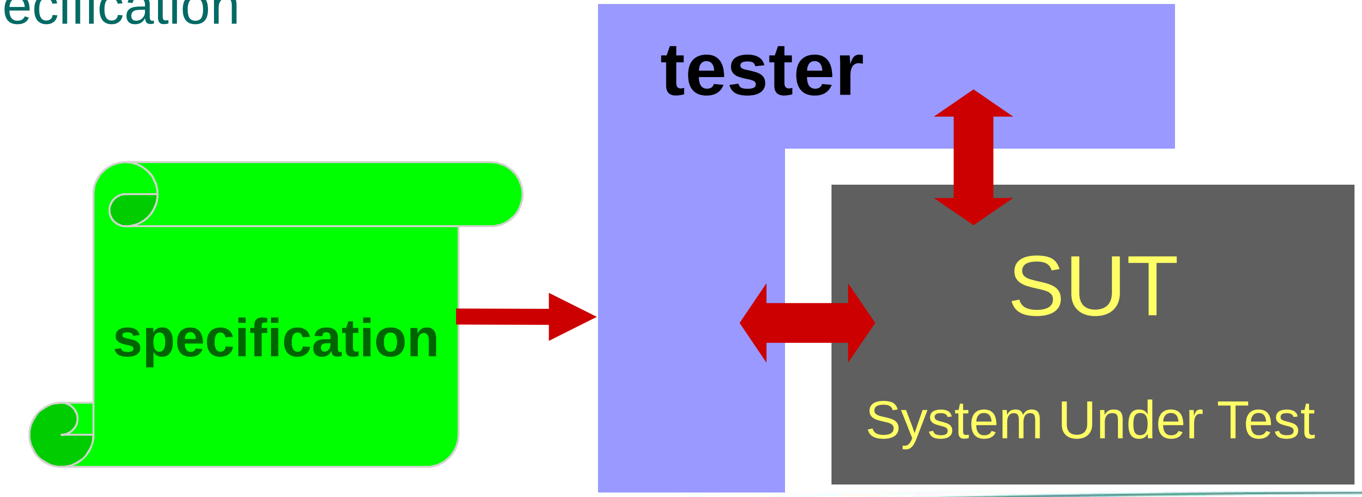
some quality characteristics

of an executing object

by performing experiments

in a controlled way

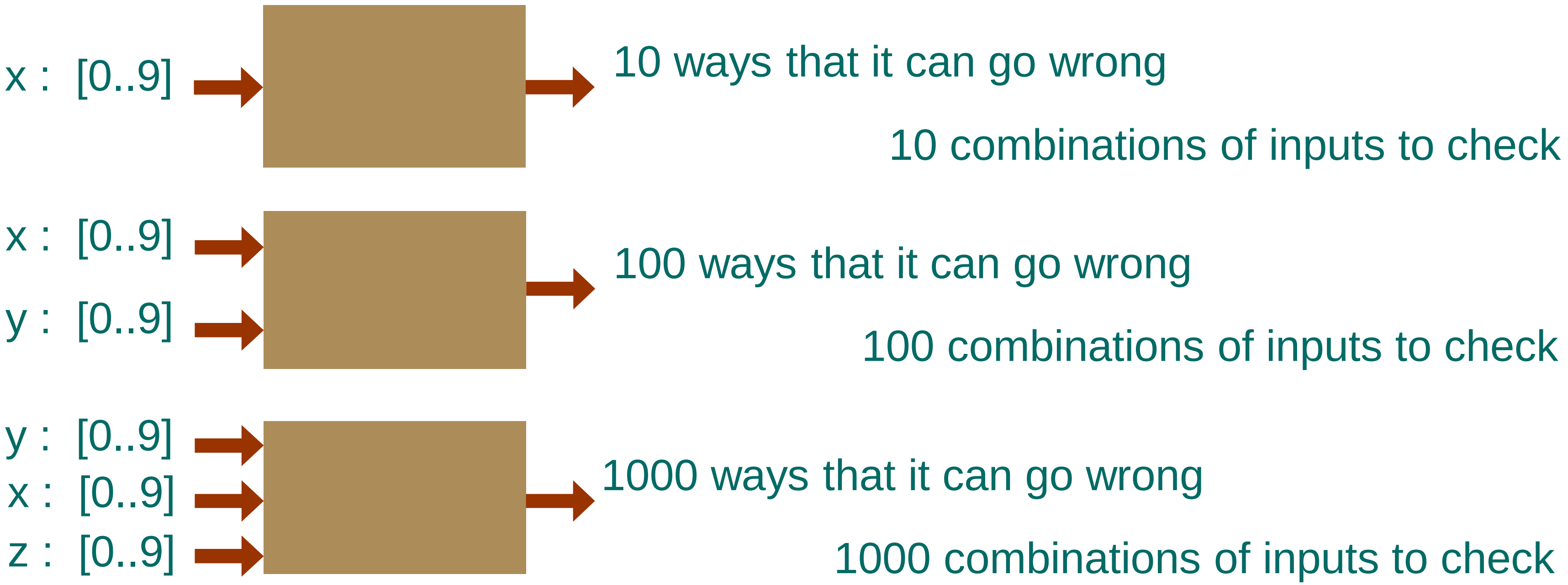
w.r.t. a specification



Testing Complexity

testing effort grows exponentially with system size

testing cannot keep pace with development



x : [0..9]

10 ways that it can go wrong

10 combinations of inputs to check

x : [0..9]

y : [0..9]

100 ways that it can go wrong

100 combinations of inputs to check

y : [0..9]

x : [0..9]

z : [0..9]

1000 ways that it can go wrong

1000 combinations of inputs to check

Automation of testing is necessary

Testing Challenges

- Increasing complexity
  - more functions, more interactions, more options and parameters
- Increasing size
  - building new systems from scratch is not possible anymore
  - integration of legacy-, outsourced-, off-the shelf components
  - abstract from details: models
- Blurring boundaries between systems
  - more, and more complex interactions between systems
  - systems dynamically depend on other systems, systems of systems
- Blurring boundaries in time
  - requirements analysis, specification, implementation, testing, installation, maintenance overlap
  - more different versions and configurations

Model-Based Testing: Why

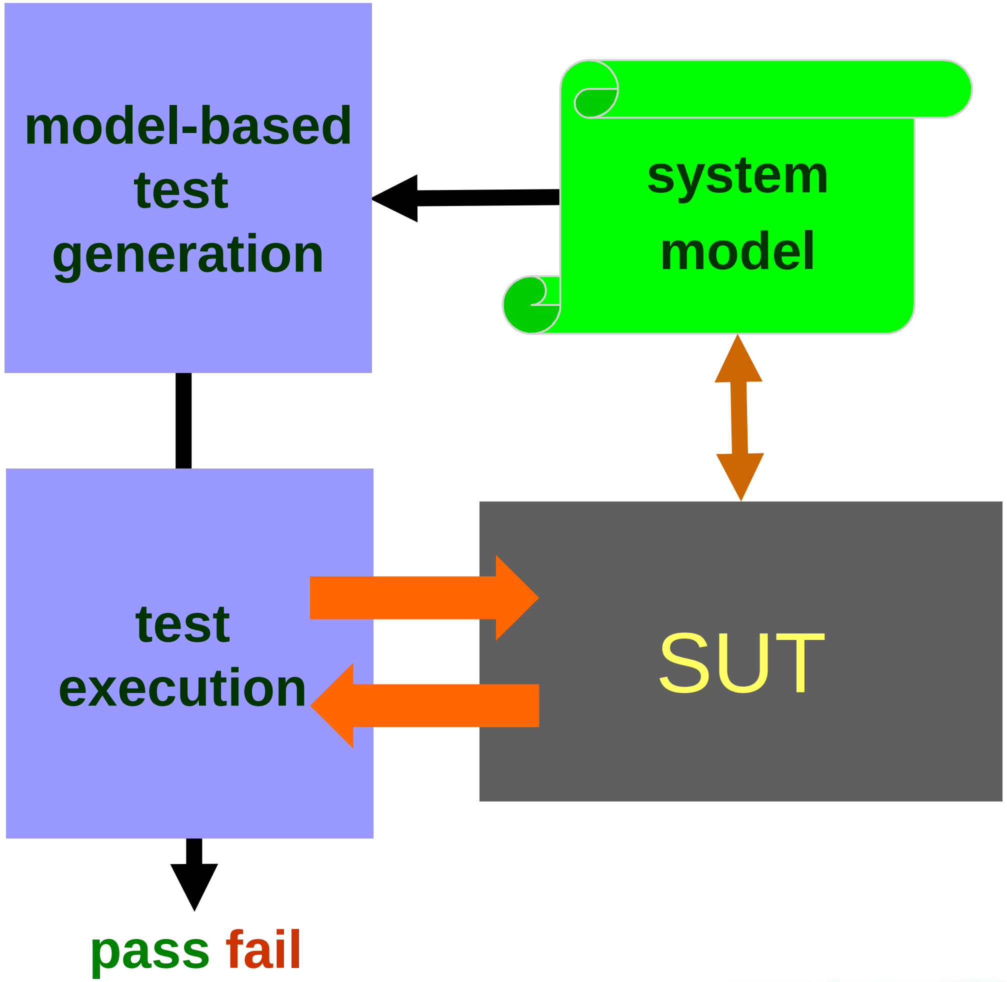
- Mastering increase in complexity, and quest for higher quality
  - testing cannot keep pace with development

Software bugs / errors cost US economy yearly:
$ 59.500.000.000   (www.nist.gov)
$ 22 billion could be eliminated…

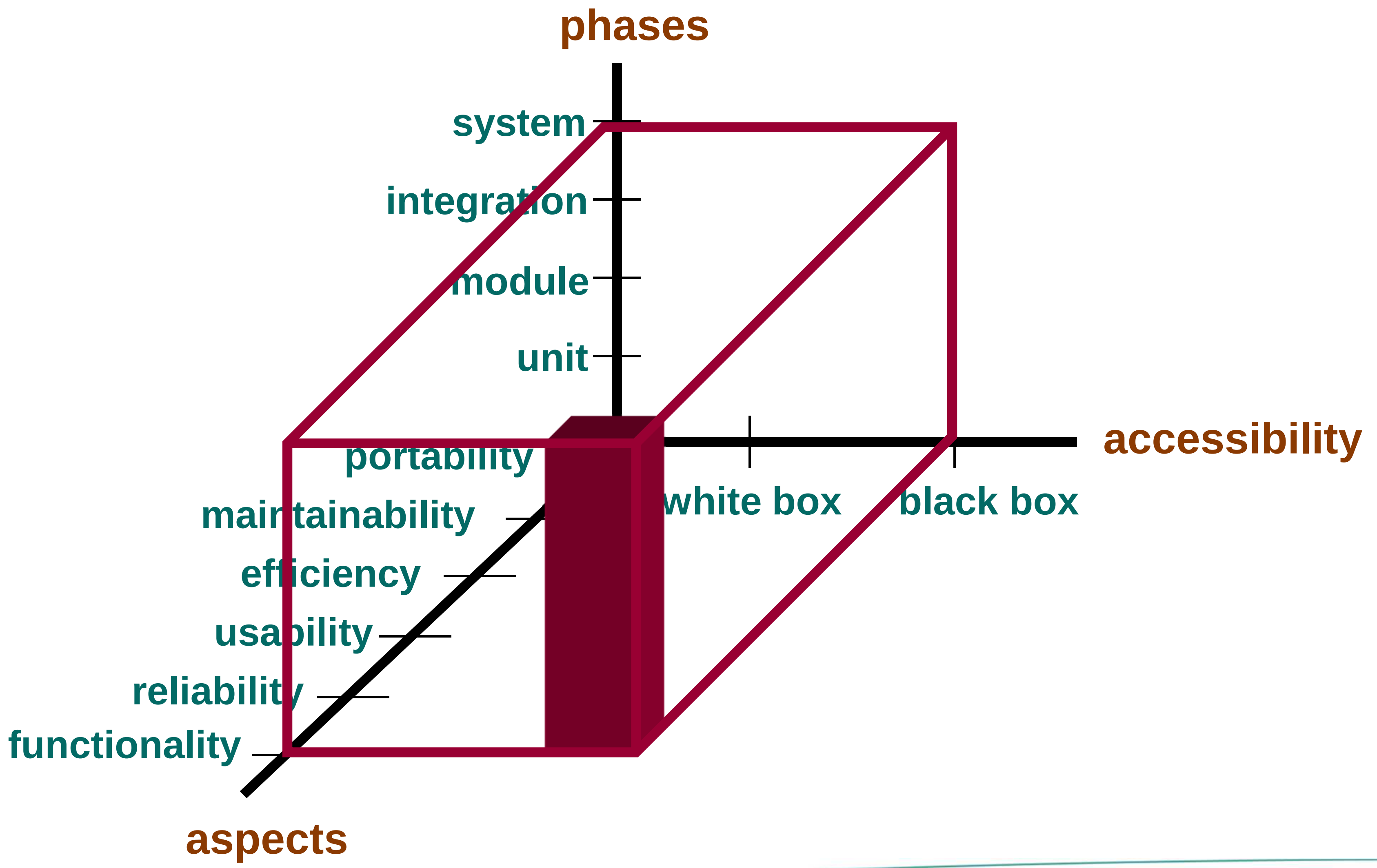
- Dealing with models and abstraction
  - model-based development: UML, MDA, Simulink/Matlab

- Promises better, faster, cheaper testing
  - algorithmic generation of tests and test oracles: tools
  - maintenance of tests through model modification

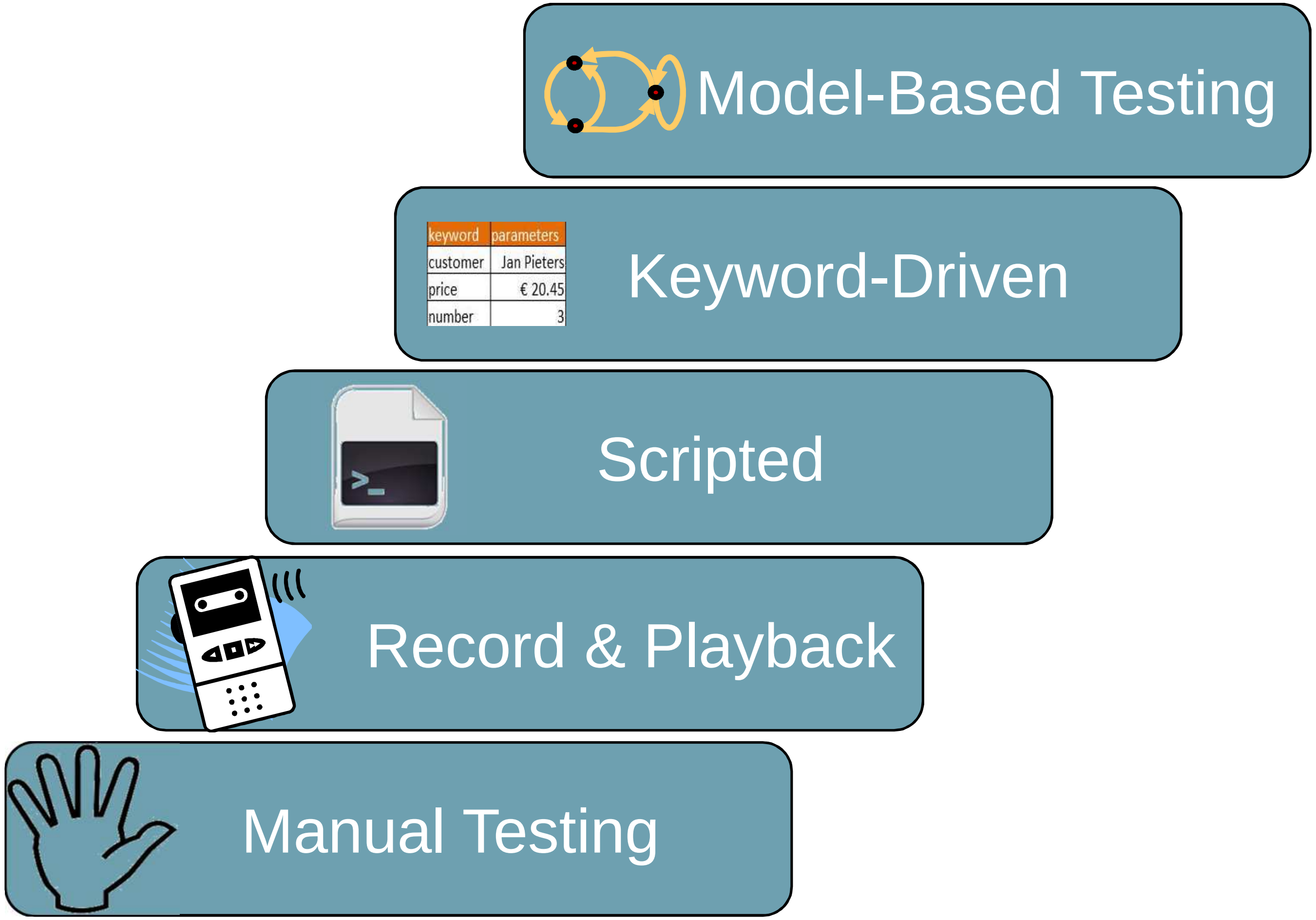
Model-Based Testing (MBT)



MBT : Black-Box Testing of Functionality



Evolution of Testing



Model-Based Testing

Keyword-Driven

keyword	parameters
customer	Jan Pieters
price	€ 20.45
number	3

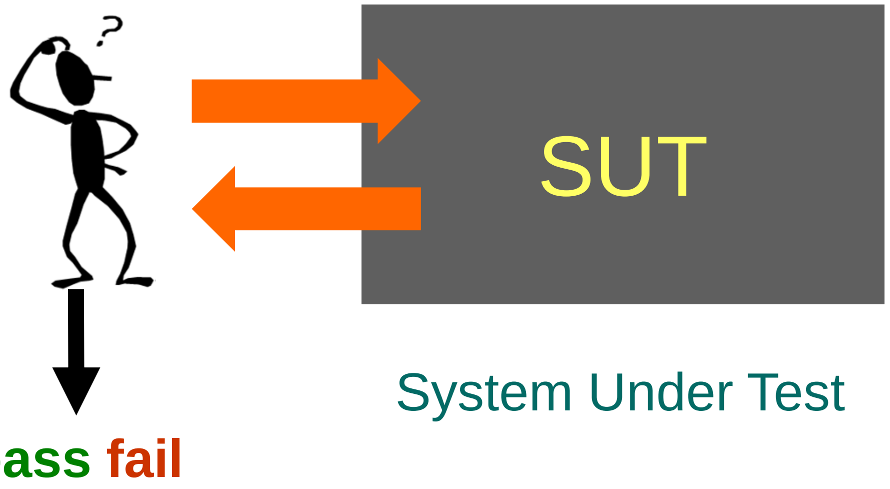
Scripted

Record & Playback

Manual Testing

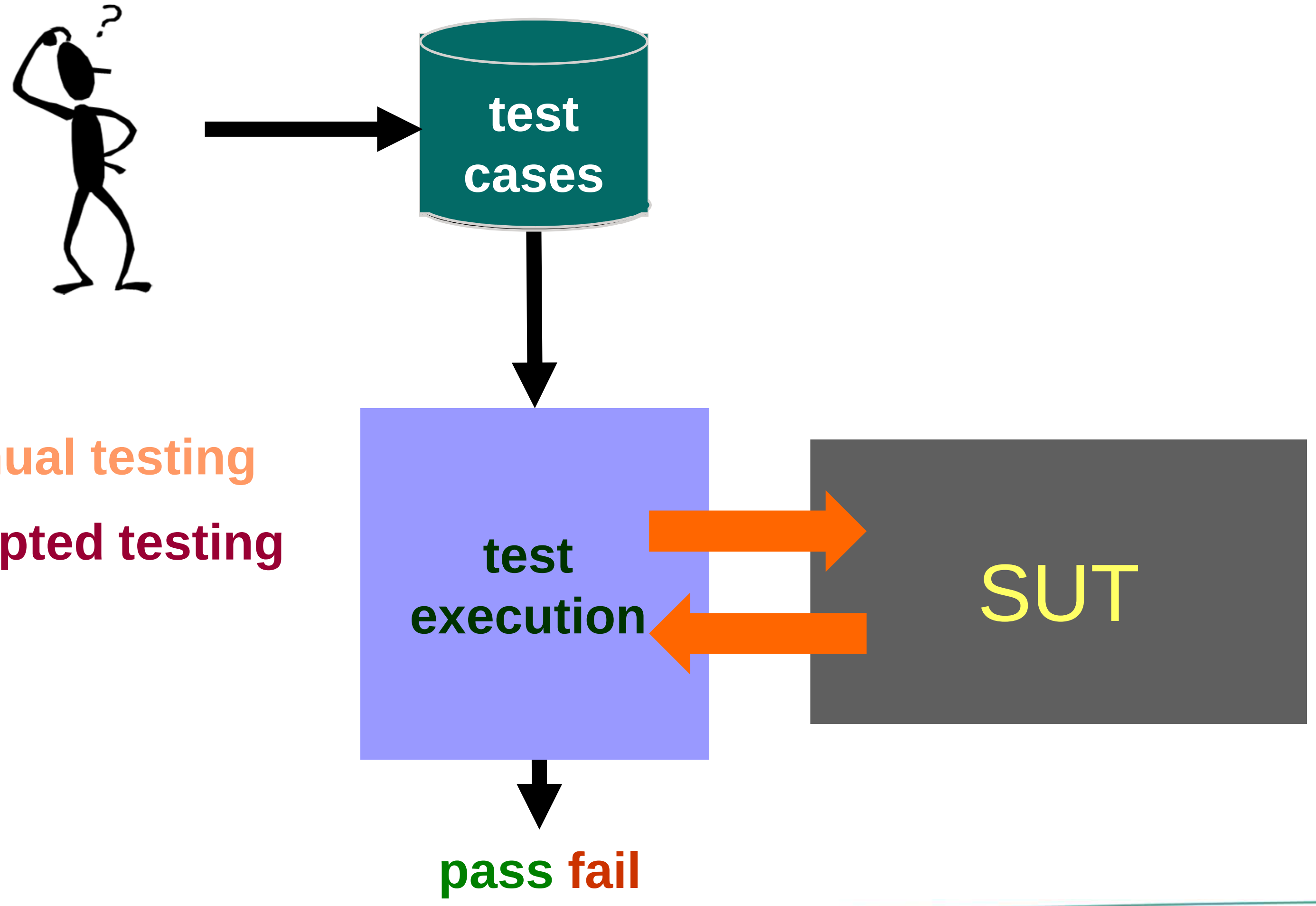
Testing 1 : Manual Testing

1. **Manual testing**

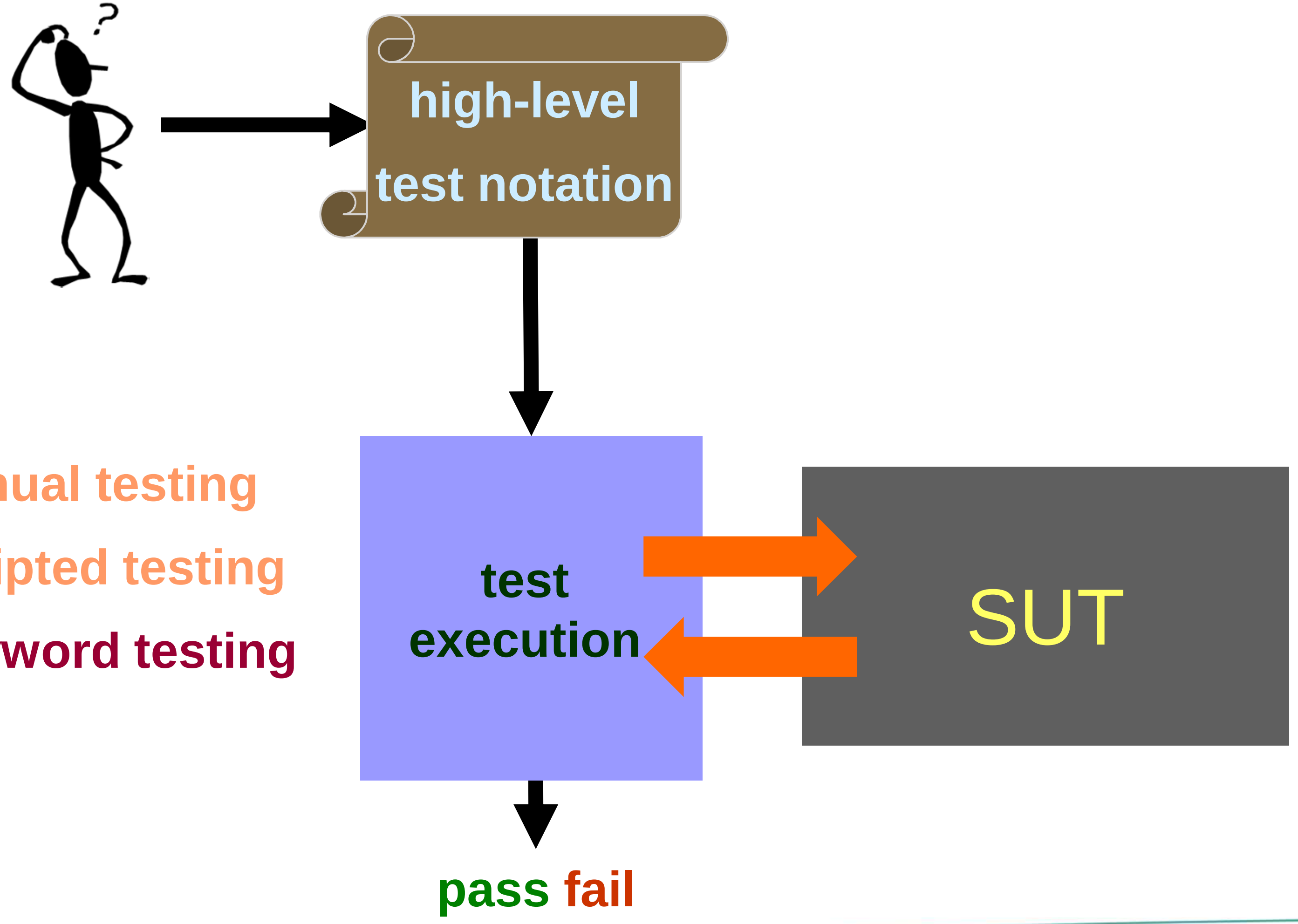


Testing 2 : Scripted Testing

1. Manual testing
2. Scripted testing



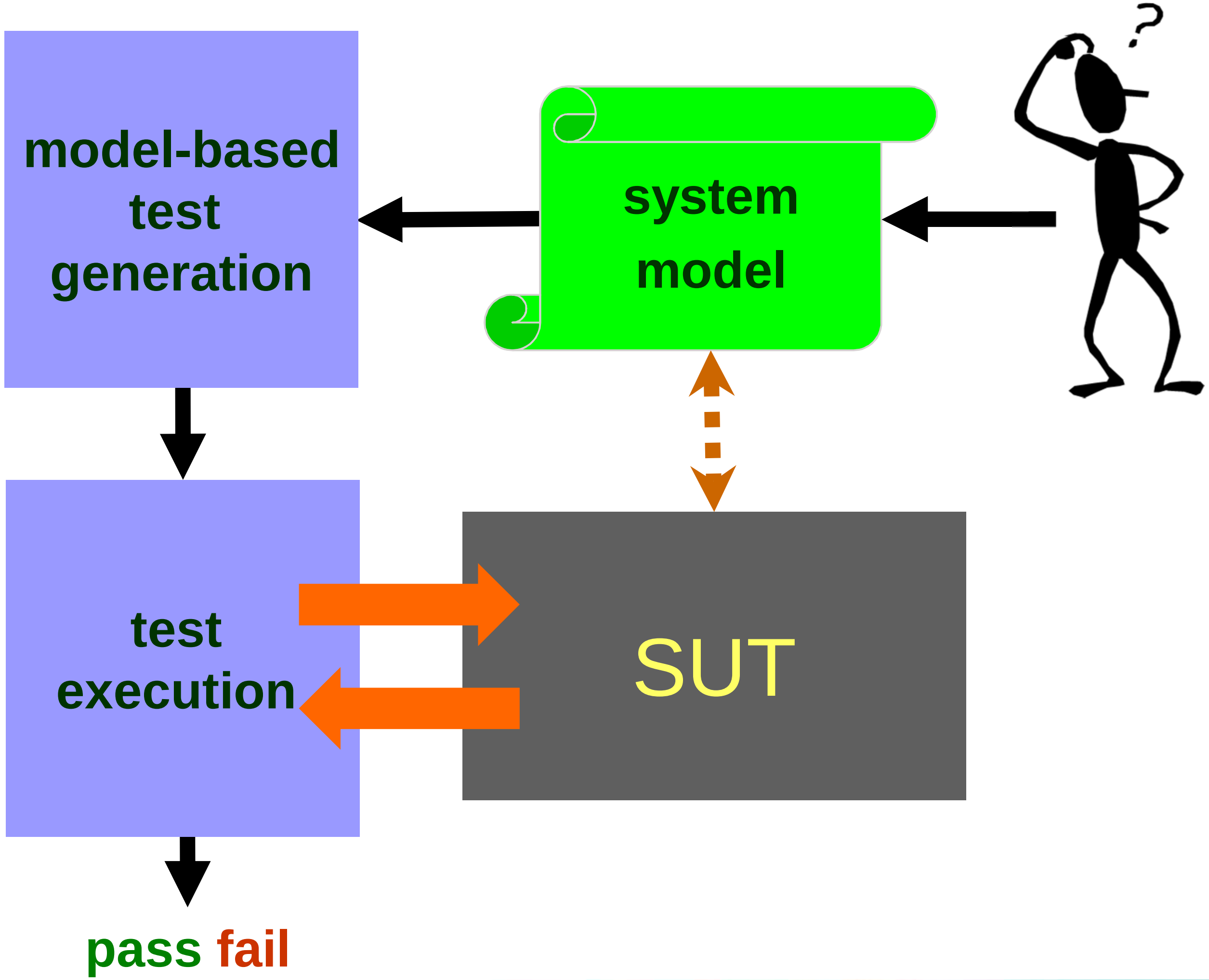
Testing 3 : Keyword-Driven Testing



1. Manual testing
2. Scripted testing
3. **Keyword testing**

Testing 4 : Model-Based Testing

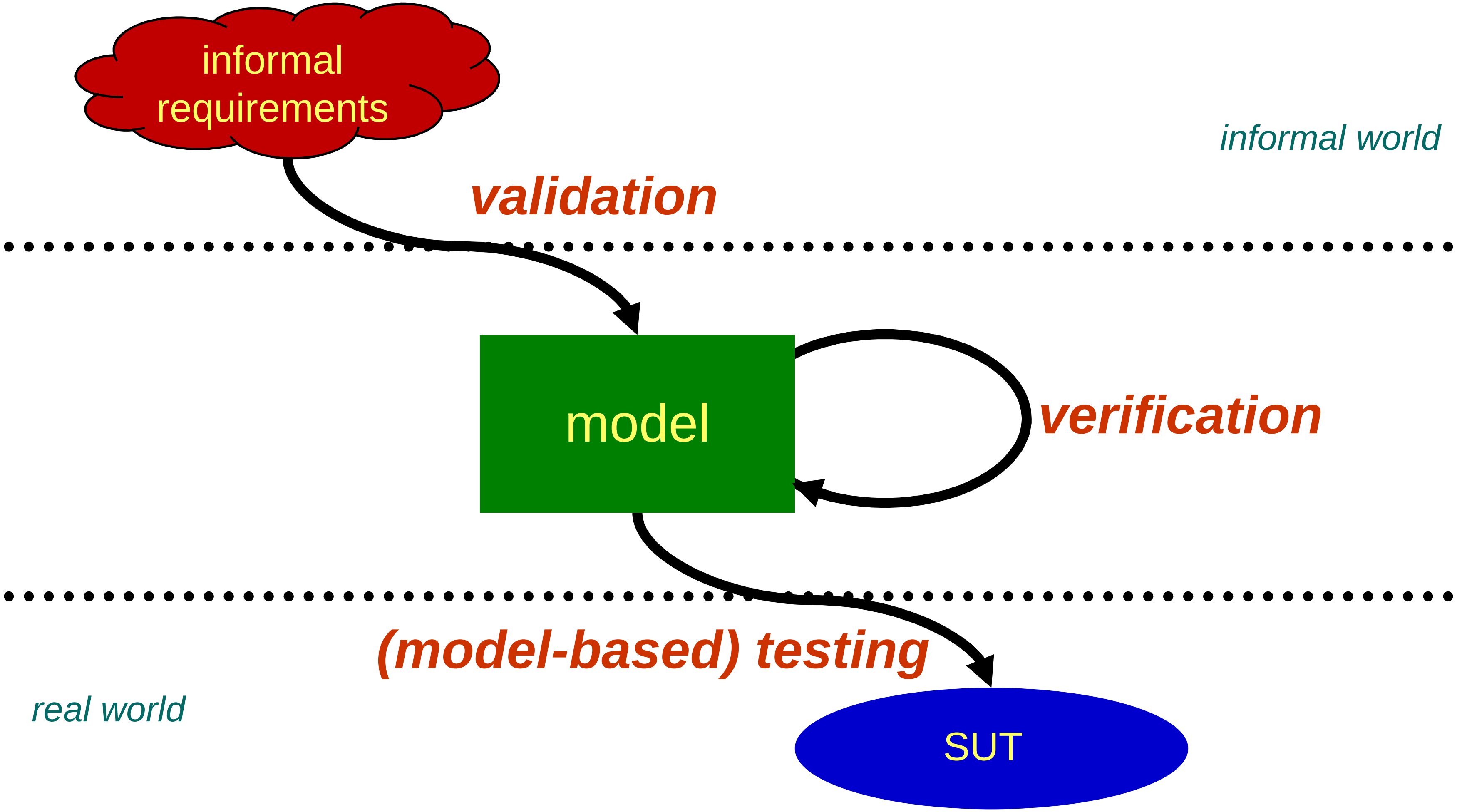
1. Manual testing
2. Scripted testing
3. Programmed testing
4. **Model-based testing**



Model-Based

Verification, Validation, Testing, ...

Validation, Verification, and Testing



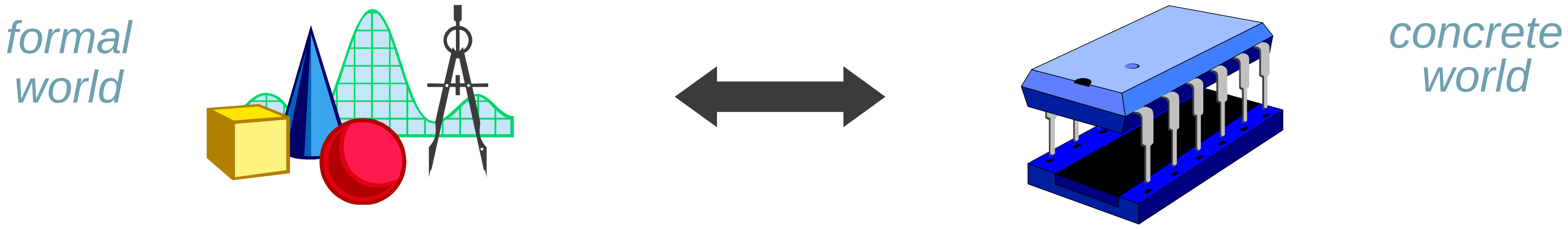
Verification and Testing

Model-based verification :

- formal manipulation
- prove properties
- performed on model

Model-based testing :

- experimentation
- show error
- concrete system

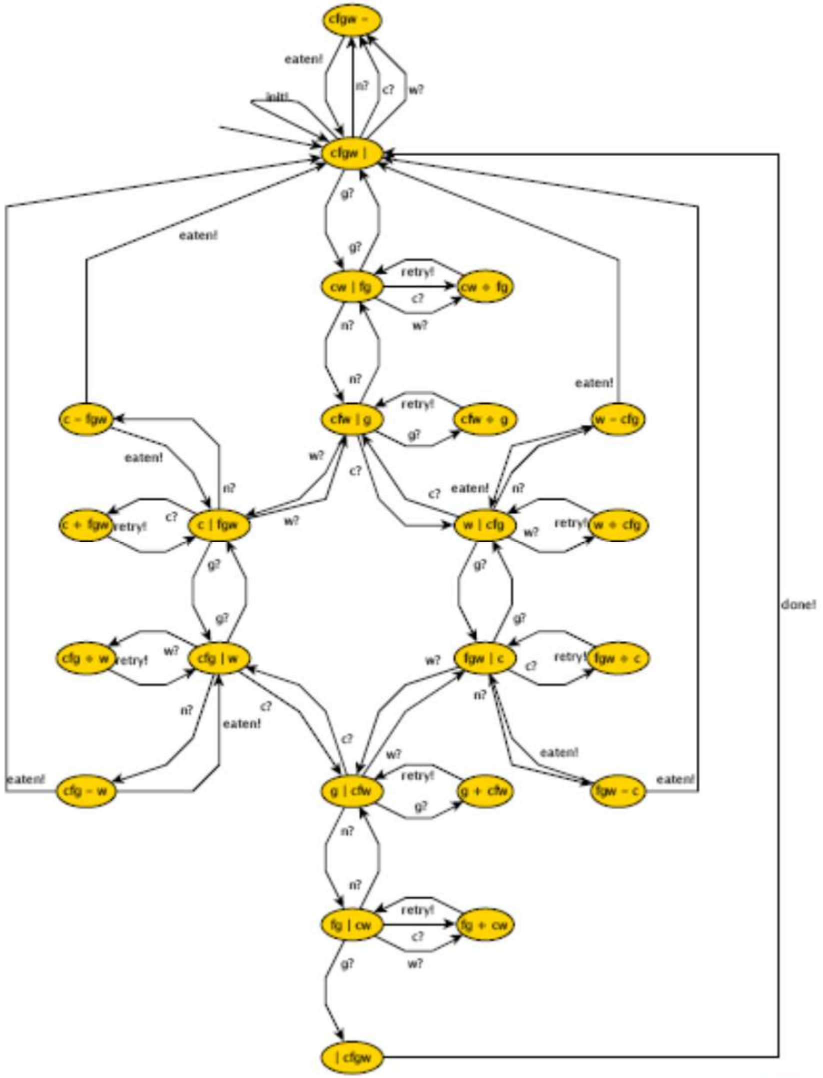


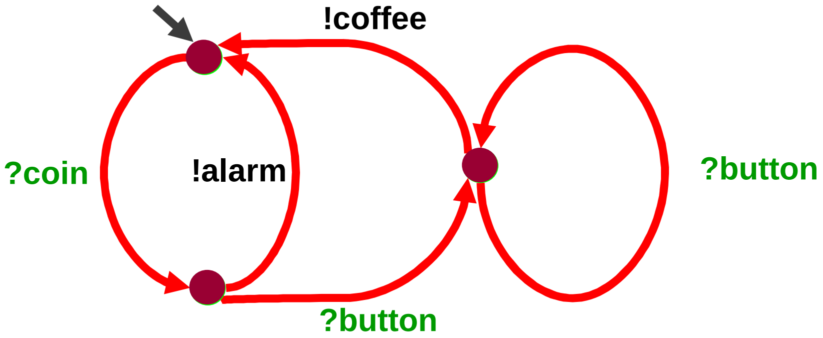
Verification is only as good as the validity of the model on which it is based

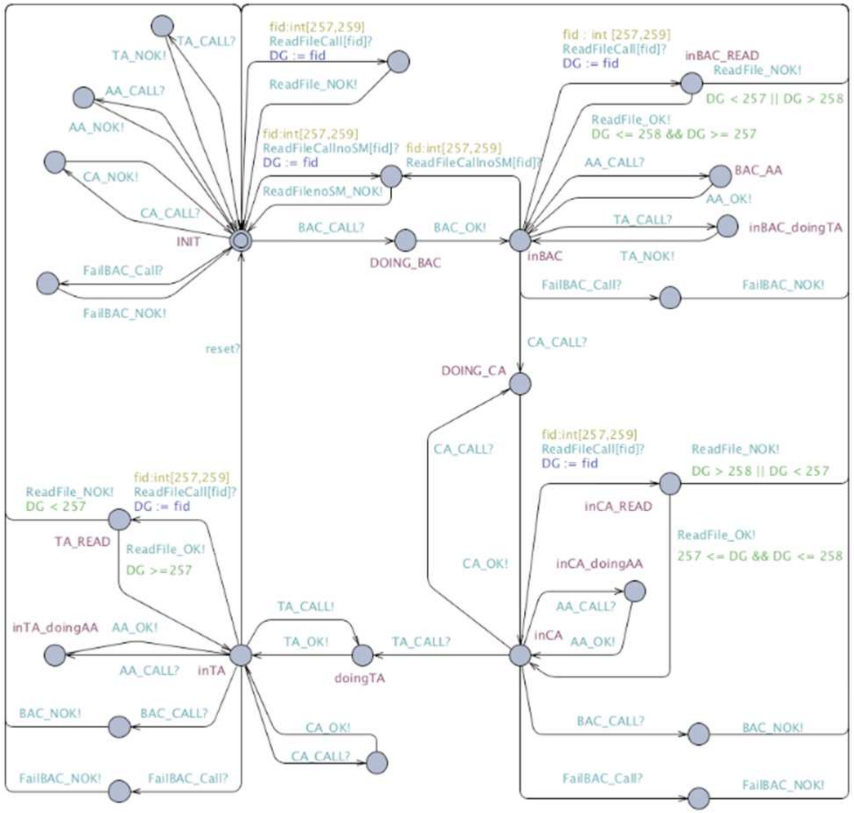
Testing can only show the presence of errors, not their absence

Models

Models


cfgw ~
eaten!
init!
n?
c?
w?
cfgw |
g?
eaten!
cw | fg
retry!
cw + fg
c?
w?
n?
cfw | g
retry!
cfw + g
eaten!
c ~ fgw
w ~ cfg
eaten!
n?
w?
c?
c? eaten!
c + fgw
retry!
c | fgw
w | cfg
w?
retry!
w + cfg
g?
done!
cfg + w
retry!
w?
cfg | w
fgw | c
c?
retry!
fgw + c
n?
eaten!
c?
w?
eaten!
cfg ~ w
g | cfw
retry!
g + cfw
g?
fgw ~ c
eaten!
n?
fg | cw
retry!
fg + cw
c?
w?
g?
| cfgw


!coffee
?coin
!alarm
?button
?button


TA_CALL?
TA_NOK!
AA_CALL?
AA_NOK!
CA_NOK!
CA_CALL?
INIT
FailBAC_Call?
FailBAC_NOK!
reset?
fid:int[257,259]
ReadFileCall[fid]?
DG := fid
ReadFile_NOK!
fid:int[257,259]
ReadFileCallnoSM[fid]?
DG := fid
fid:int[257,259]
ReadFileCallnoSM[fid]?
ReadFilenoSM_NOK!
BAC_CALL?
DOING_BAC
BAC_OK!
inBAC
fid : int [257,259]
ReadFileCall[fid]?
DG := fid
inBAC_READ
ReadFile_NOK!
DG < 257 || DG > 258
ReadFile_OK!
DG <= 258 && DG >= 257
AA_CALL?
BAC_AA
AA_OK!
TA_CALL?
inBAC_doingTA
TA_NOK!
FailBAC_Call?
FailBAC_NOK!
CA_CALL?
DOING_CA
CA_CALL?
fid:int[257,259]
ReadFileCall[fid]?
DG := fid
ReadFile_NOK!
DG > 258 || DG < 257
inCA_READ
ReadFile_OK!
257 <= DG && DG <= 258
CA_OK!
inCA_doingAA
AA_CALL?
inCA
AA_OK!
TA_CALL?
BAC_CALL?
BAC_NOK!
FailBAC_Call?
FailBAC_NOK!
ReadFile_NOK!
DG < 257
fid:int[257,259]
ReadFileCall[fid]?
DG := fid
TA_READ
ReadFile_OK!
DG >=257
TA_CALL!
TA_OK!
doingTA
inTA_doingAA
AA_OK!
AA_CALL?
inTA
BAC_NOK!
BAC_CALL?
CA_OK!
CA_CALL?
FailBAC_NOK!
FailBAC_Call?

Models: Labelled Transition Systems

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

states

input actions

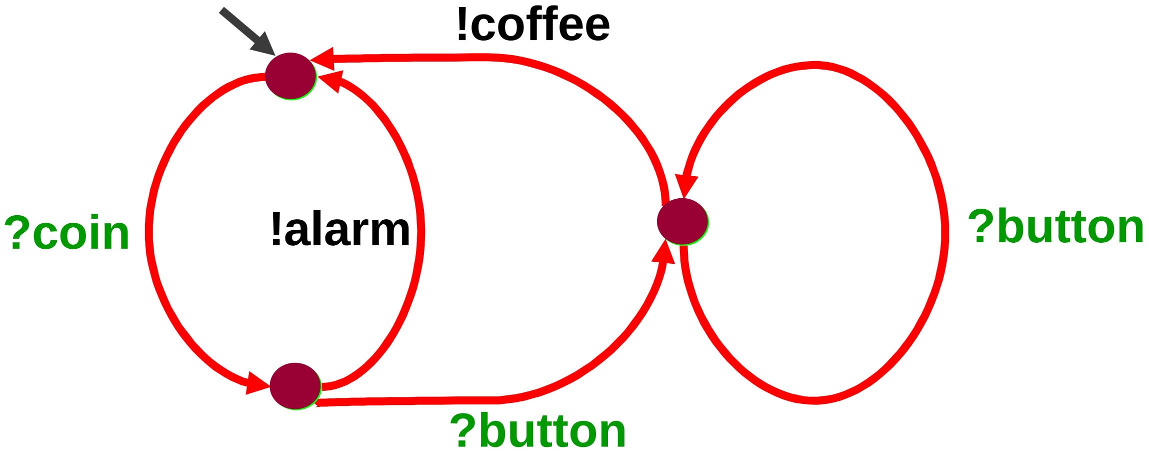
output actions

transitions

initial state

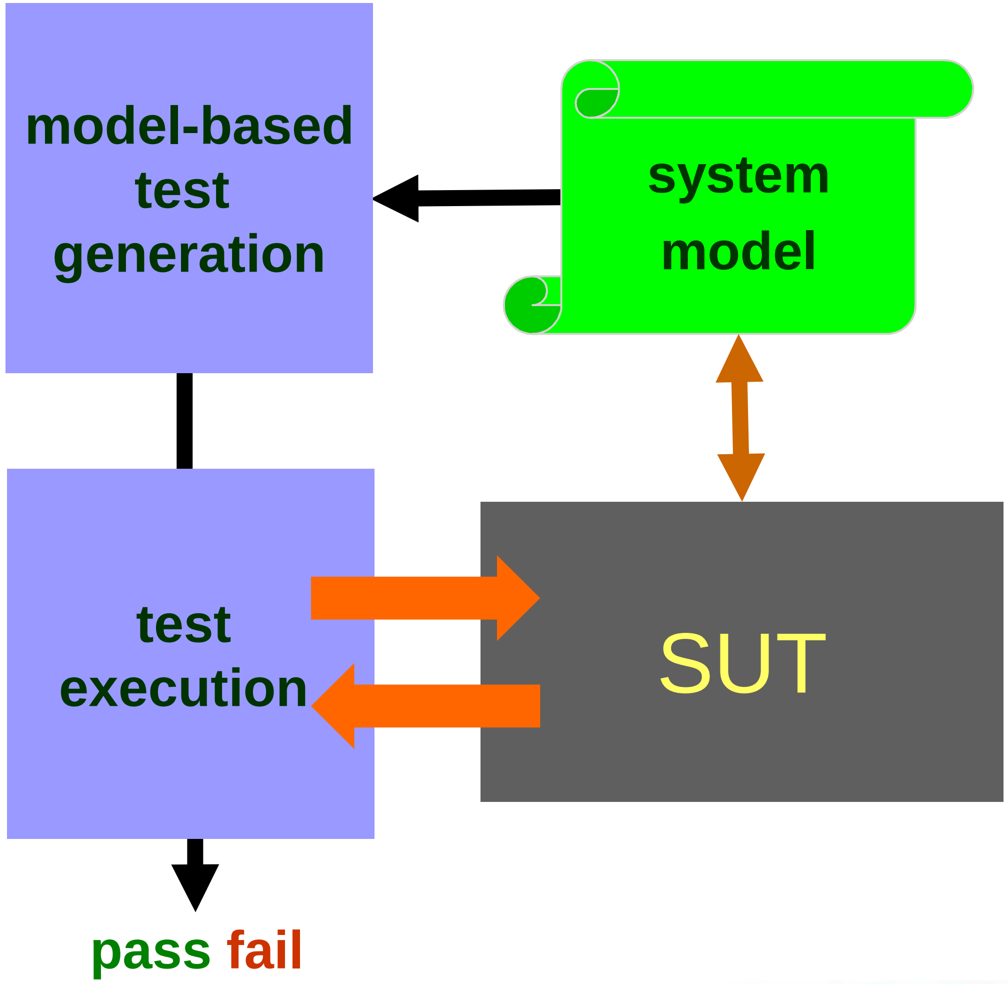
? = input

! = output



A Theory of Model-Based Testing with Labelled Transition Systems

Model-Based Testing

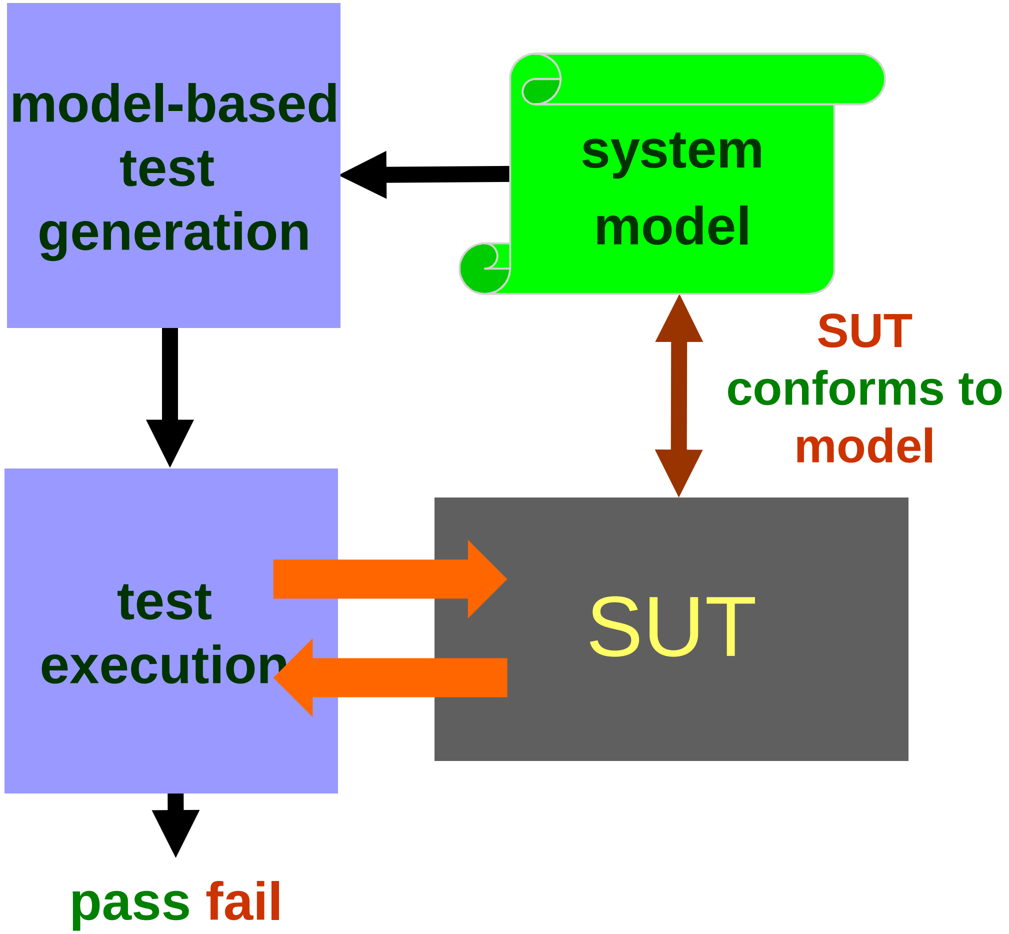


MBT : Validity

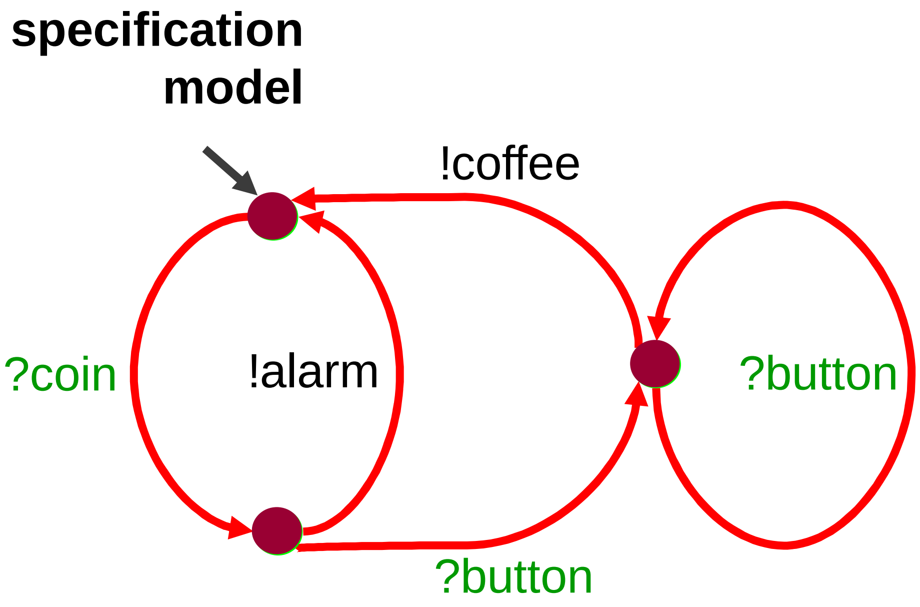
SUT conforms to model

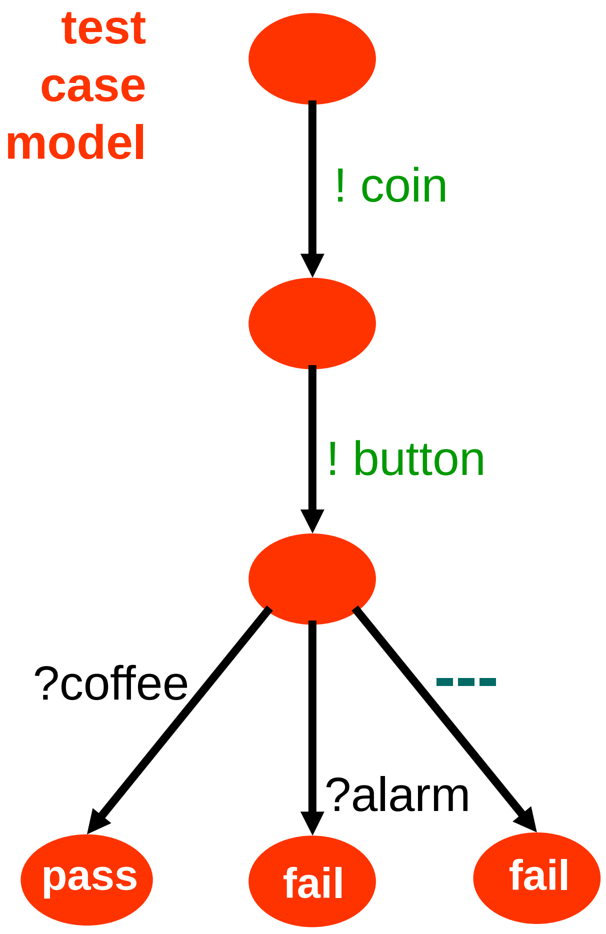
⇕

SUT passes tests

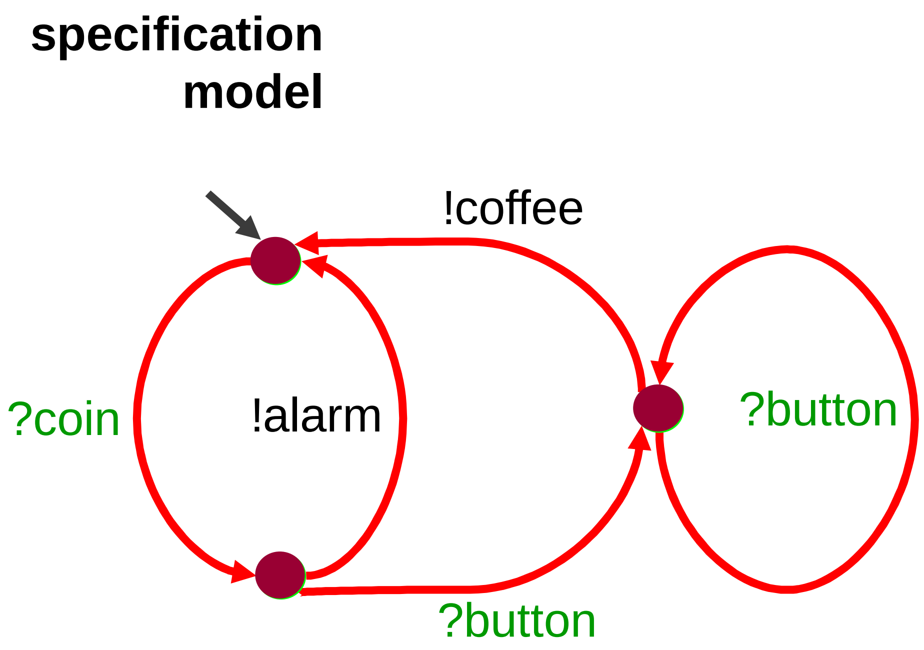


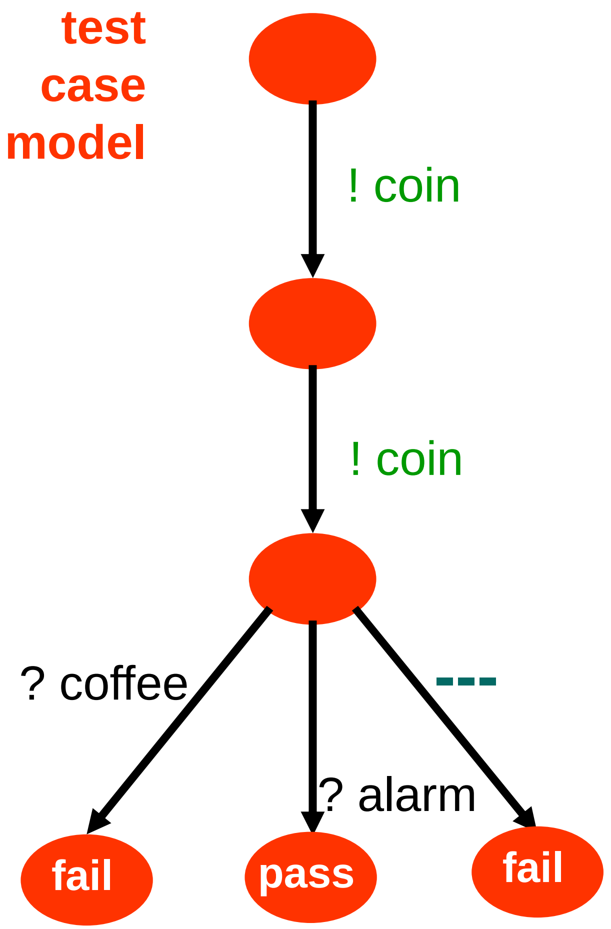
Models: Generation of Test Cases





Models: Generation of Test Cases

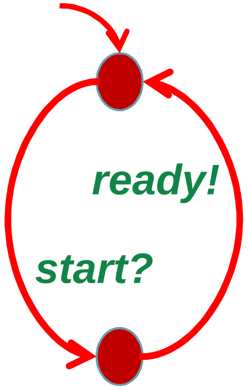




MBT : Abstract from Scheduling Details

- Four components in parallel, in any order

task(start?, ready!)



taskA := task (startA?, readyA!)

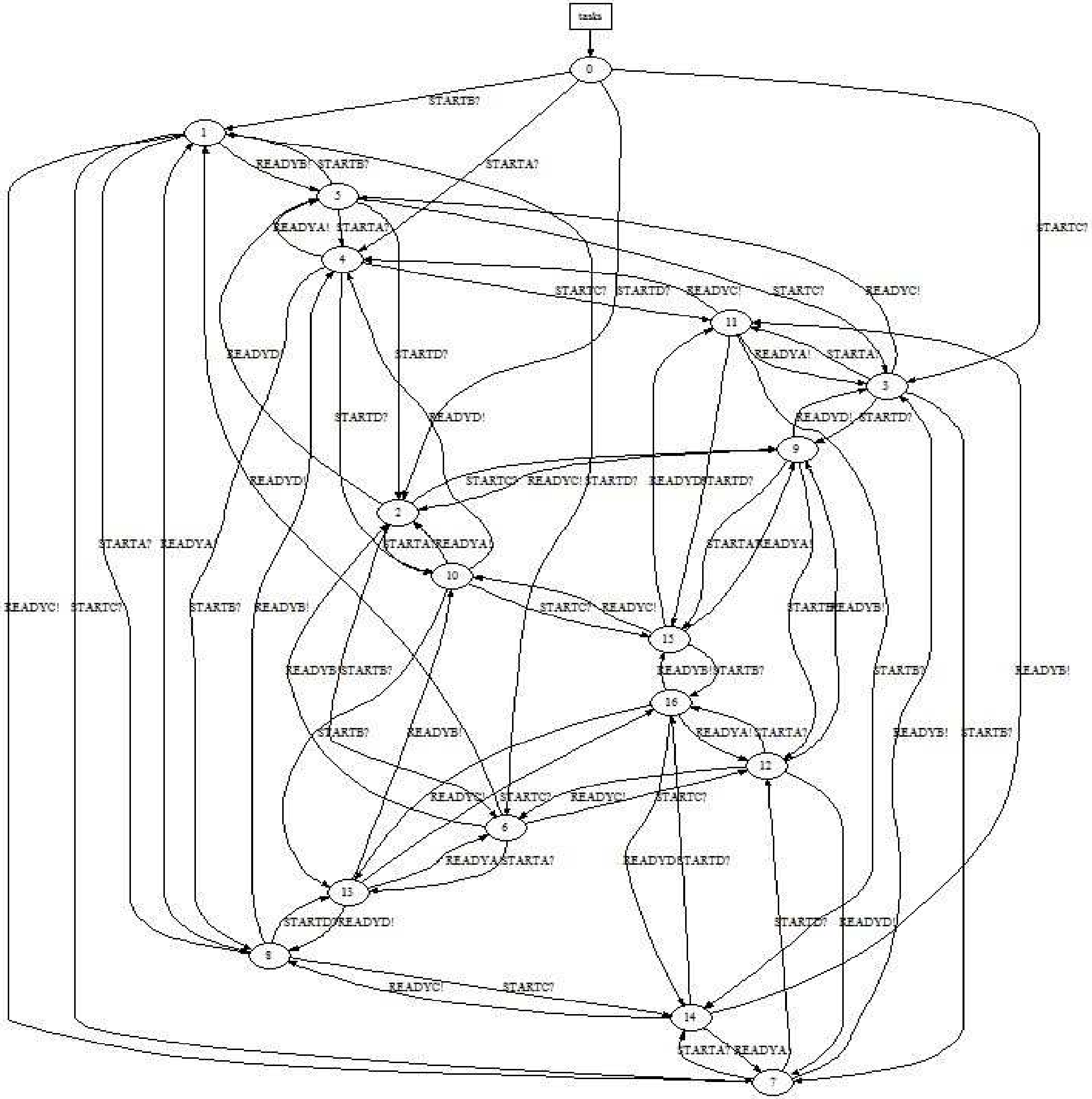
taskB := task (startB?, readyB!)

taskC := task (startC?, readyC!)

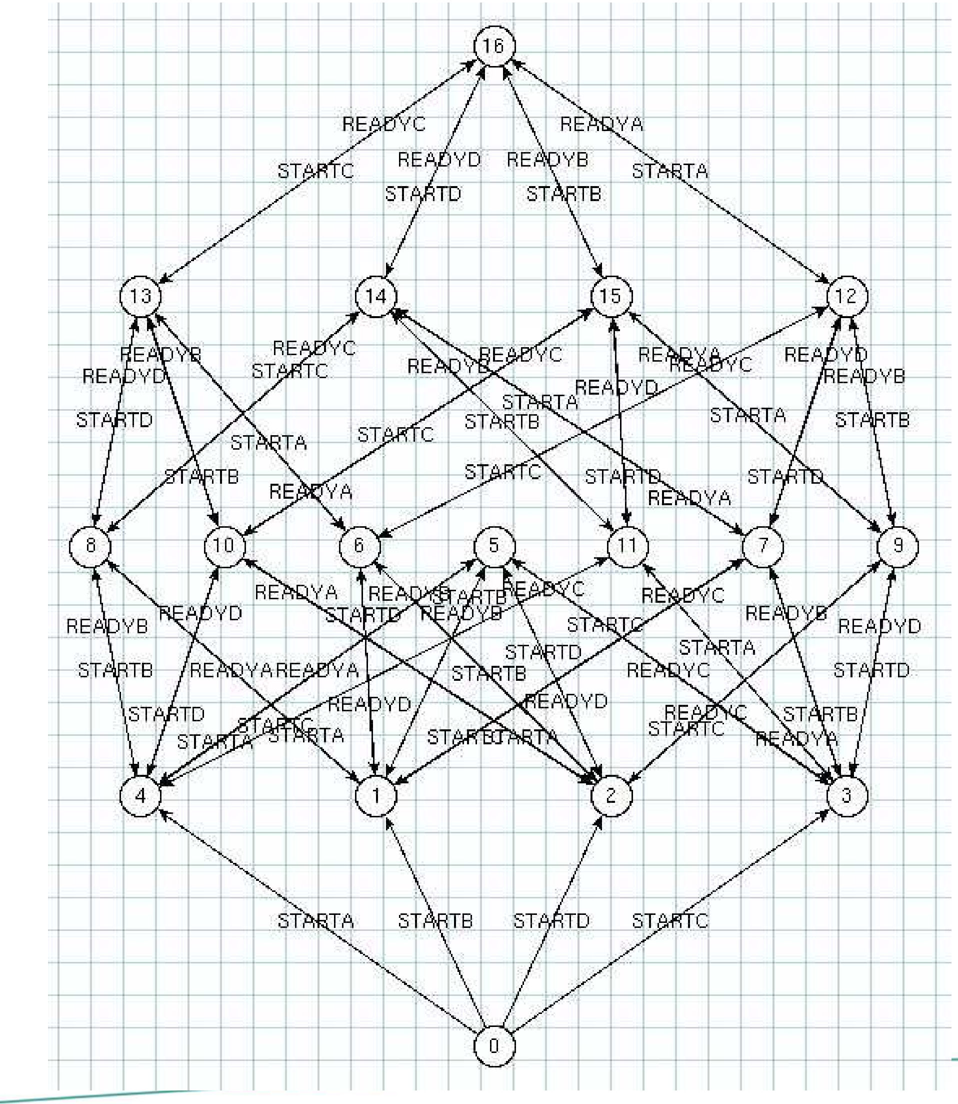
taskD := task (startD?, readyD!)

model := taskA ||| taskB ||| taskC ||| taskD

MBT : Abstract from Scheduling Details

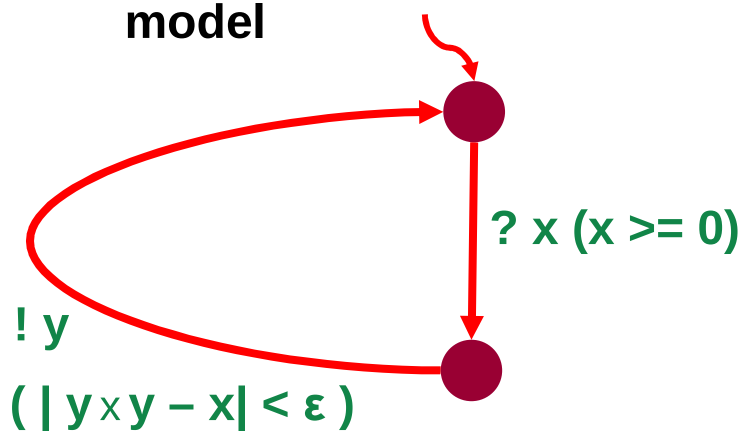


MBT : Abstract from Scheduling Details

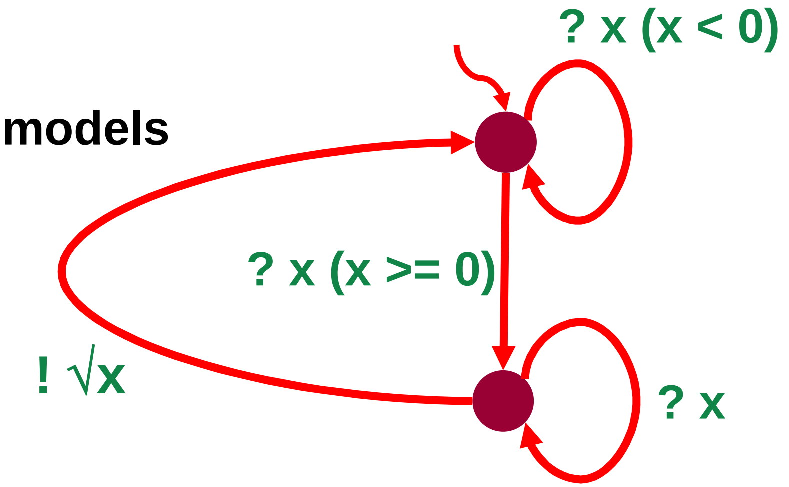


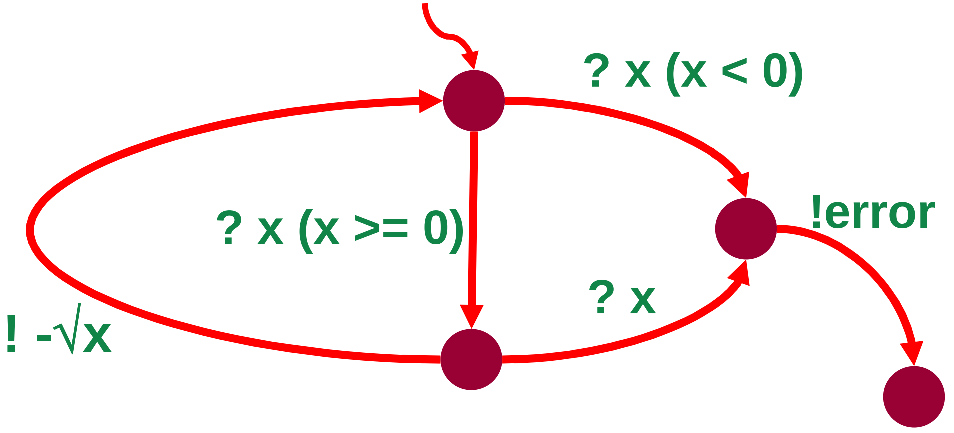
MBT : Nondeterminism, Underspecification

specification model



SUT models





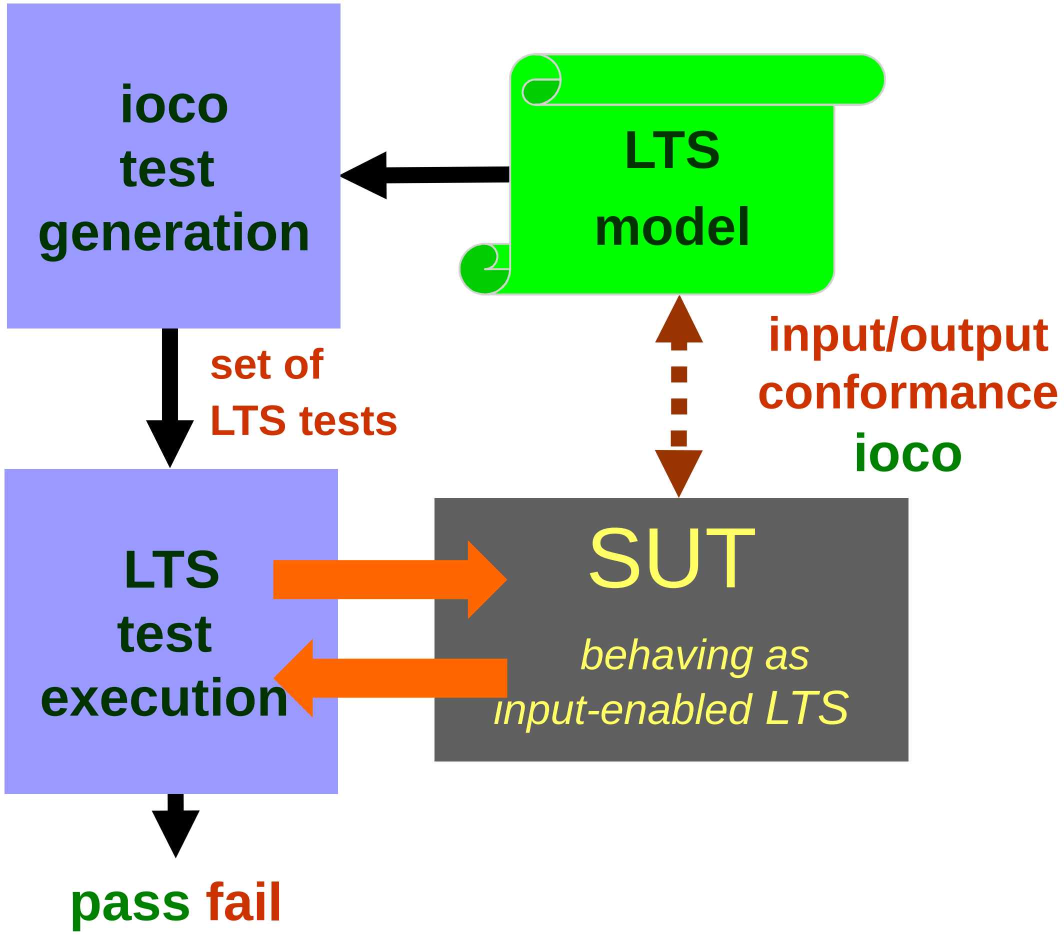
- non-determinism
- under-specification
- specification of properties rather than construction

MBT with LTS and **ioco**

SUT ioco model

sound ⇓ ⇑ *exhaustive*

SUT passes tests



ioco test generation

LTS model

set of LTS tests

input/output conformance ioco

LTS test execution

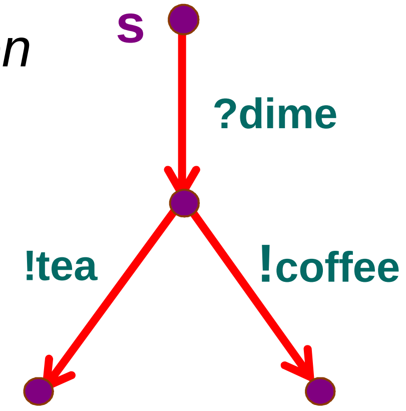
SUT

behaving as input-enabled LTS

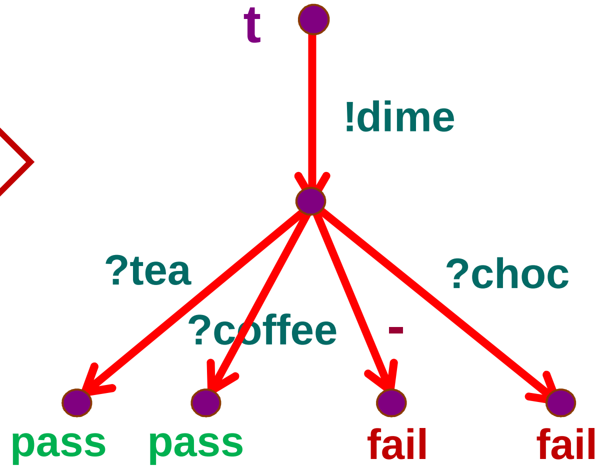
pass fail

MBT : Argue about Validity of Tests

specification model



generated test case

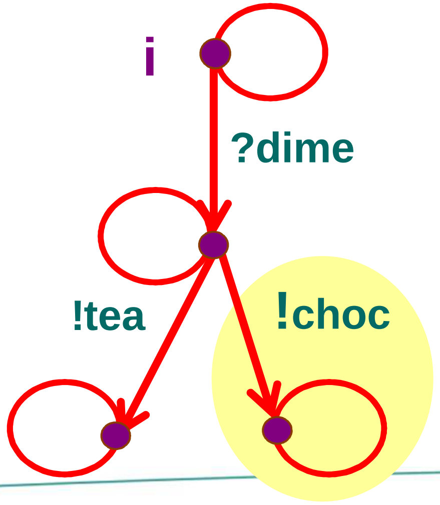


i ~~ioco~~ s

⇕

i fails t

implementation



Model-Based Testing
with Labelled Transition Systems

There is Nothing More Practical

than a Good Theory

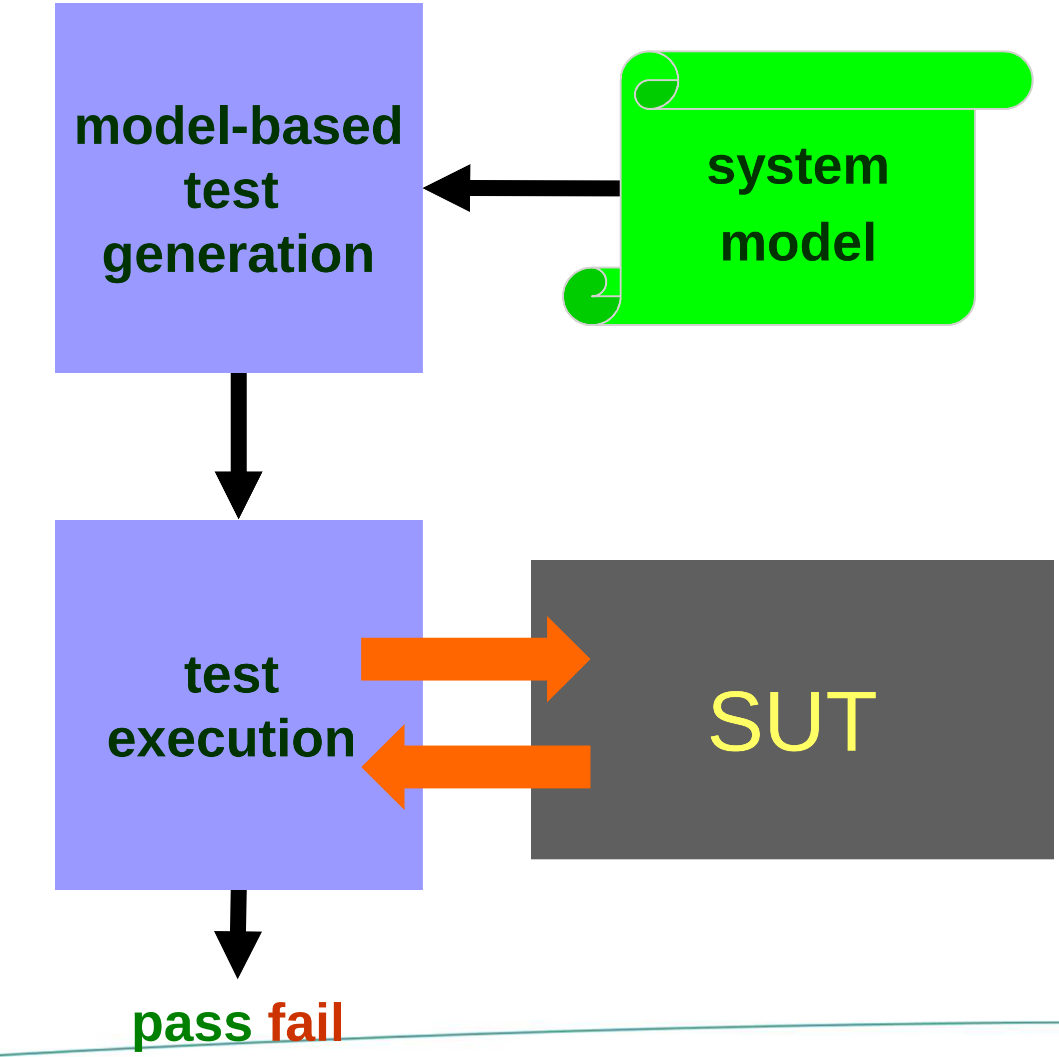
Overview

- MBT: Tools
- MBT: Under-specification
- MBT: Test selection
- MBT: Towards test selection for ioco
- Refinement for ioco
- Test-based modelling = Automata learning

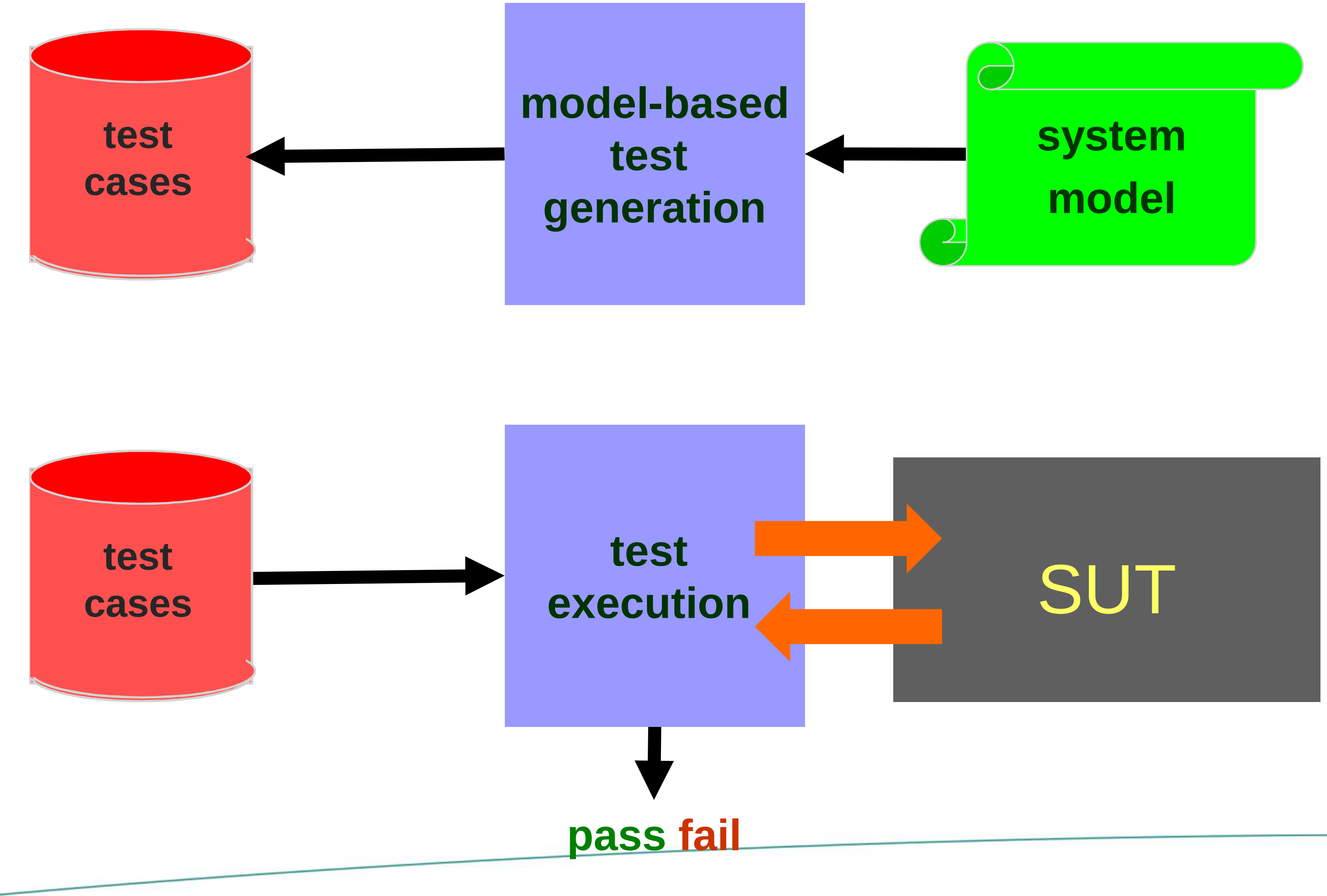
Model-Based Testing

Tools

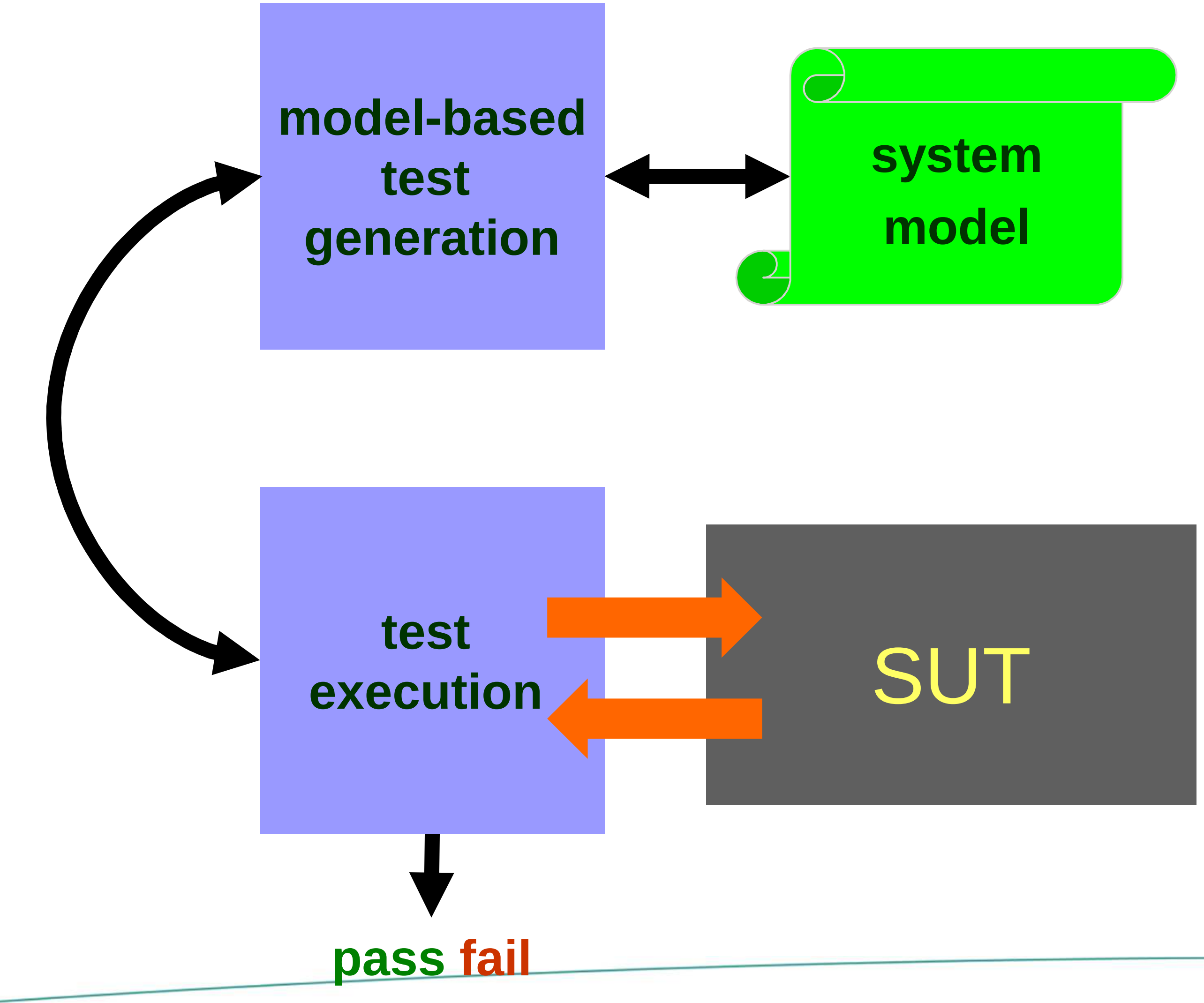
MBT : Off-Line - On-Line



MBT : Off-Line = Batch



MBT : On-Line = On-the-Fly



model-based test generation

system model

test execution

SUT

pass fail

Model-Based Testing :

Variations for Underspecification

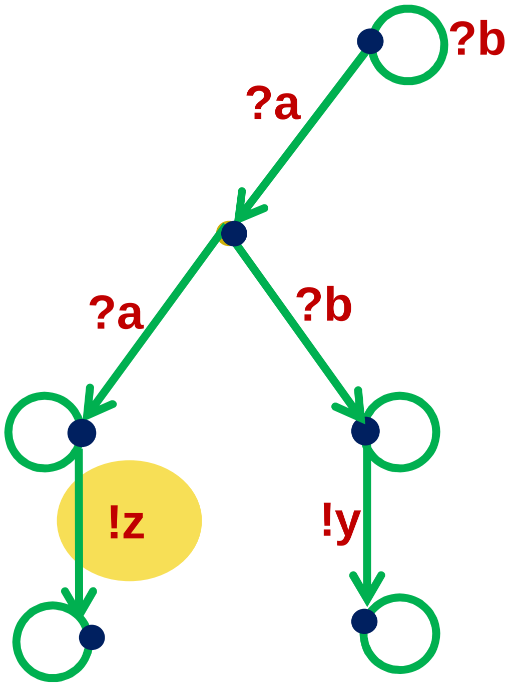
Variations on a Theme

- **i** ioco **s** $\Leftrightarrow$ $\forall \sigma \in$ Straces(s) : out (i after σ) $\subseteq$ out (s after σ)
- **i** $\leq_{ior}$ **s** $\Leftrightarrow$ $\forall \sigma \in$ (L $\cup$ {δ})* : out (i after σ) $\subseteq$ out (s after σ)
- **i** ioconf **s** $\Leftrightarrow$ $\forall \sigma \in$ traces(s) : out (i after σ) $\subseteq$ out (s after σ)
- **i** ioco$_F$ **s** $\Leftrightarrow$ $\forall \sigma \in$ F : out (i after σ) $\subseteq$ out (s after σ)
- **i** uioco **s** $\Leftrightarrow$ $\forall \sigma \in$ Utraces(s) : out (i after σ) $\subseteq$ out (s after σ)
- **i** mioco **s** multi-channel ioco
- **i** wioco **s** non-input-enabled ioco
- **i** eco **e** environmental conformance
- **i** sioco **s** symbolic ioco
- **i** (r)tioco **s** (real) timed tioco (Aalborg, Twente, Grenoble, Bordeaux,.....)
- **i** rioco **s** refinement ioco
- **i** hioco **s** hybrid ioco
- **i** qioco **s** quantified ioco
- **i** poco **s** partially observable game ioco
- **i** stioco$_D$ **s** real time and symbolic data
- . . . . . .

Underspecification: ioco and uioco

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

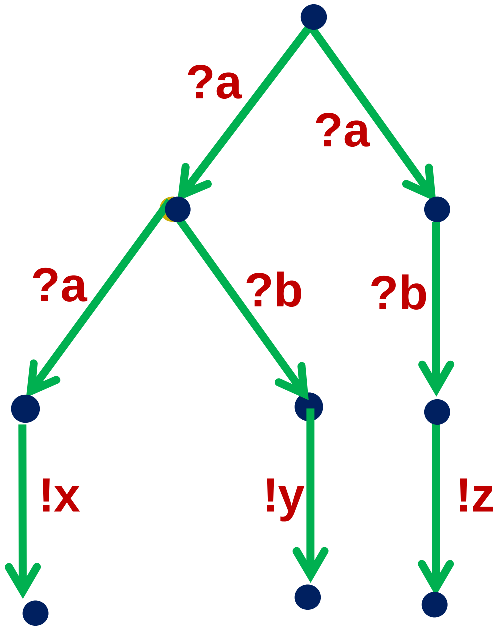
Implementation **i**



i ~~ioco~~ s

i uioco s

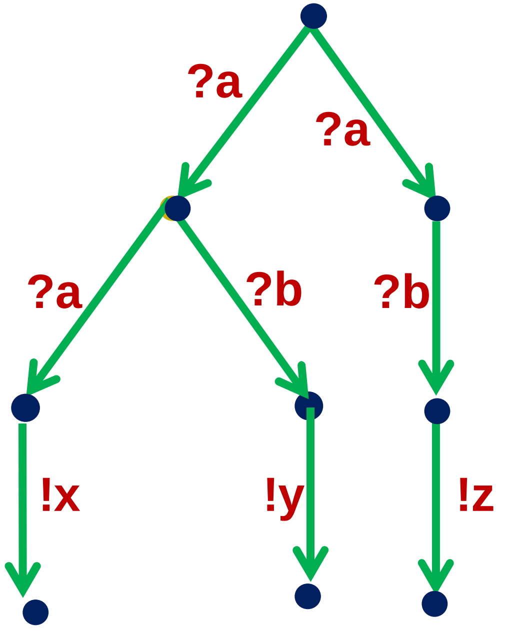
Specification **s**



Underspecification: uioco

$i \; \textbf{ioco} \; s \quad =_{def} \quad \forall \sigma \in \textit{Straces}\,(s) : \; \textit{out}\,(i \text{ after } \sigma) \subseteq \textit{out}\,(s \text{ after } \sigma)$

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



$\textit{Utraces}(s) \; =$

$\{ \; \sigma \in \textit{Straces}\,(s) \; \mid$

$\forall \sigma_1 \, ?b \; \sigma_2 = \sigma :$

$s \text{ after } \sigma_1 \text{ must } ?b \; \}$

$?a \, ?a \in \textit{Straces}\,(s)$

$?a \, ?a \notin \textit{Utraces}\,(s)$

ioco $\subset$ **uioco**

Test Selection

in Model-Based Testing

Test Selection

- Exhaustiveness never achieved in practice
- Test selection = select subset of exhaustive test suite,

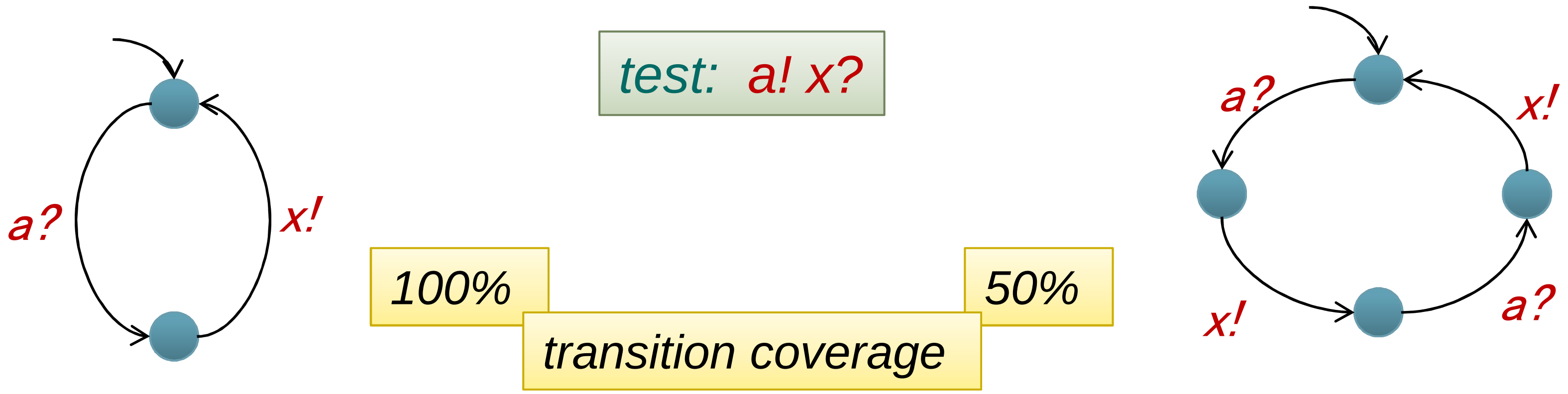
  to achieve confidence in quality of tested product

  - select best test cases capable of detecting failures
  - measure to what extent testing was exhaustive : *coverage*
- Optimization problem

  best possible testing ↔ *within cost/time constraints*

Test Selection: Approaches

1. random
2. domain / application specific: test purposes, test goals, …
3. model / code based: coverage
   - usually structure based



test: a! x?

100%

50%

transition coverage

Towards Test Selection

in the ioco Framework

Test Selection for **uioco**

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

Selection of Sub-Set of UTraces

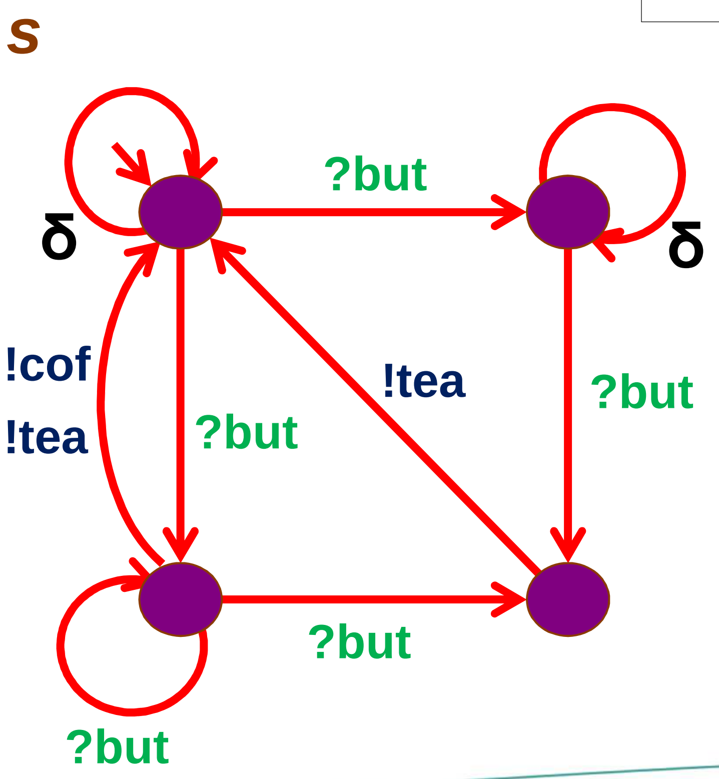
- Select: $\boldsymbol{M} \subset \textit{Utraces}\,(\mathbf{s})$
- Test for: $\mathbf{i}\ \mathbf{uioco}_{\boldsymbol{M}}\ \mathbf{s}$

  $\Leftrightarrow\ \forall\, \sigma \in \boldsymbol{M} :\ \textit{out}\,(\mathbf{i}\ \mathbf{after}\ \sigma) \subseteq \textit{out}\,(\mathbf{s}\ \mathbf{after}\ \sigma)$
- Coverage: $\dfrac{\#\,\boldsymbol{M}}{\#\,\textit{Utraces}\,(\mathbf{s})}$

Test Selection for **uioco**

out (s after ?but δ δ ?but) = *out* (s after ?but δ ?but)

i.e. if already tested for **?but δ ?but**
what does testing for **?but δ δ ?but** *add ?*



out (s after ?but) = { !cof, !tea, δ }

i.e. everything is allowed -
what shall be tested then ?

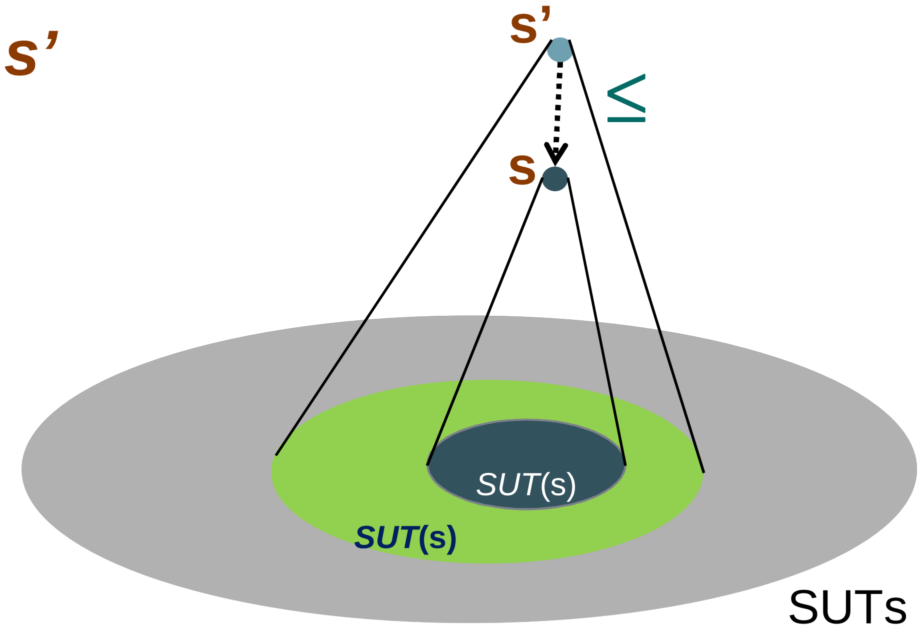
The set *Utraces* is not minimal,
i.e., elements are dependent

Test Selection for **uioco**

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

*Take **weaker** specification **s'***

= *inverse of refinement*



$s\ \leq\ s'$

$\Leftrightarrow\ \ \mathit{SUT}\,(\mathbf{s})\ \subseteq\ \mathit{SUT}\,(\mathbf{s'})$

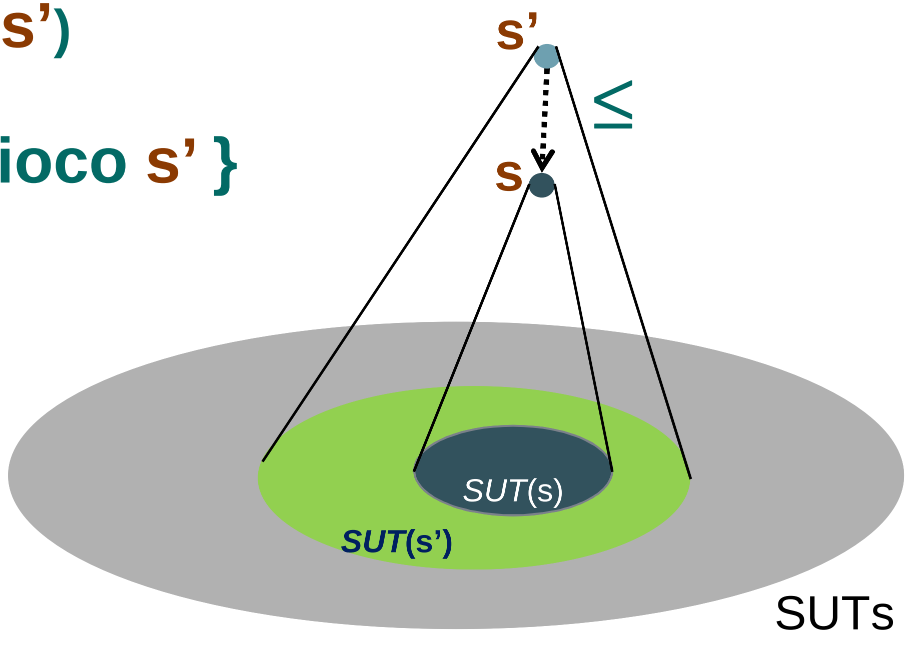
$\Leftrightarrow\ \ \{\ \mathbf{i} \mid \mathbf{i}\ \mathbf{uioco}\ \mathbf{s}\ \}\ \subseteq\ \{\ \mathbf{i} \mid \mathbf{i}\ \mathbf{uioco}\ \mathbf{s'}\ \}$

Test Selection for **uioco**

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

$$s \leq s' \quad \Leftrightarrow \quad \mathit{SUT}(s) \subseteq \mathit{SUT}(s')$$

$$\Leftrightarrow \ \{ i \mid i \ \mathbf{uioco} \ s \} \subseteq \{ i \mid i \ \mathbf{uioco} \ s' \}$$



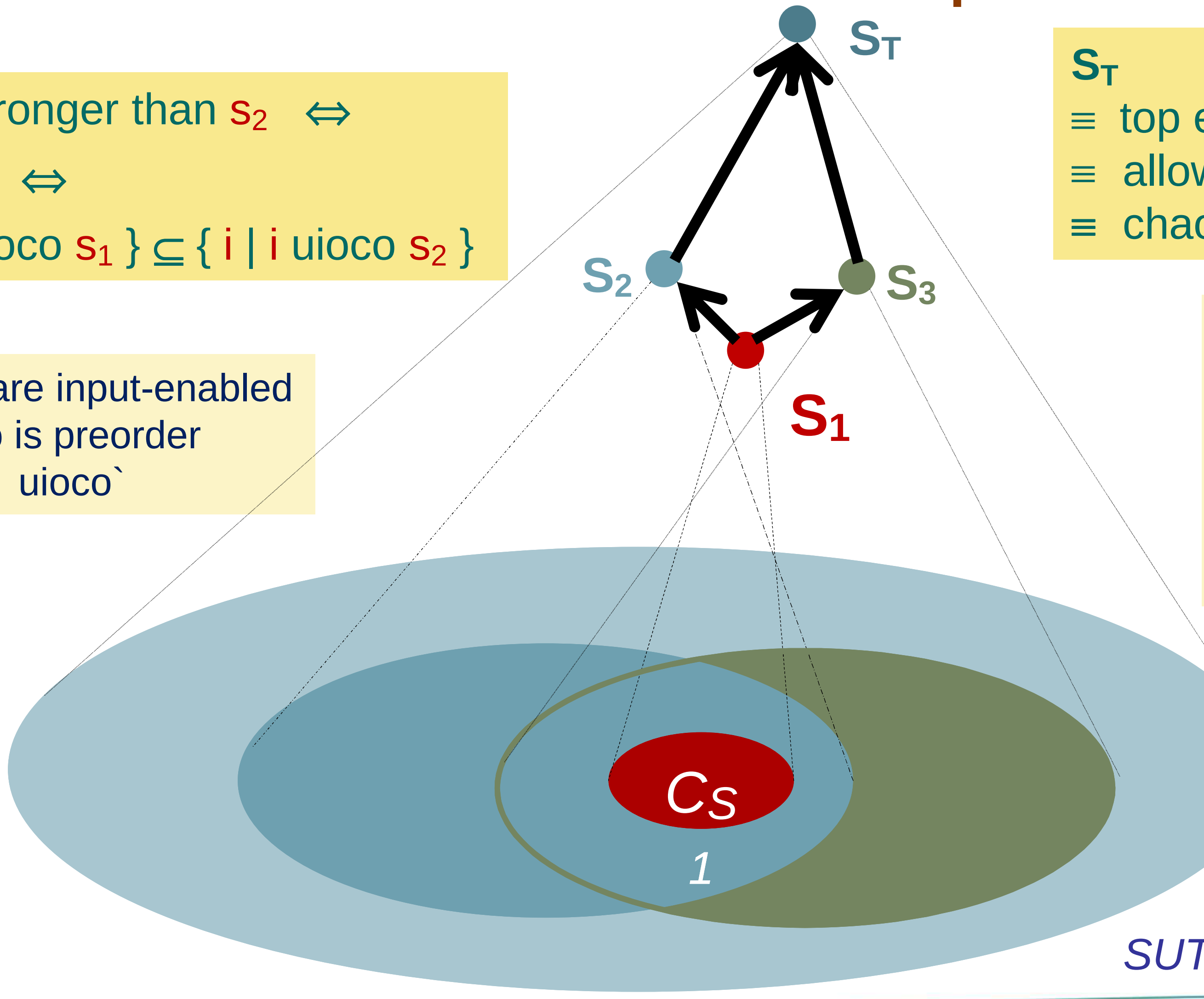
Coverage: $$\frac{\# \ \mathit{SUT}(s)}{\# \ \mathit{SUT}(s')}$$

Test Selection: Lattice of Specifications

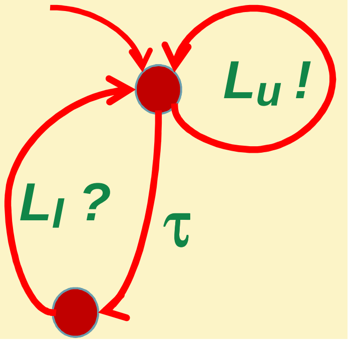
s_1 is stronger than s_2 ⇔

$s_1 \leq s_2$ ⇔

$\{ i \mid i \text{ uioco } s_1 \} \subseteq \{ i \mid i \text{ uioco } s_2 \}$



S_T
≡ top element
≡ allows any impl.
≡ chaos χ



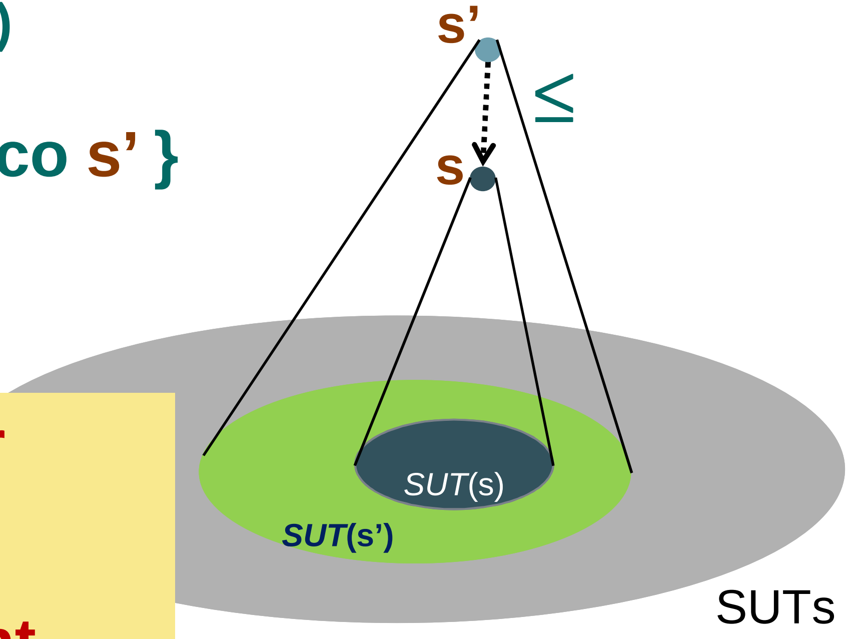
if specs are input-enabled
then ioco is preorder
then ≤ ≡ uioco`

Test Selection for **uioco**

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

$$s \leq s' \quad \Leftrightarrow \quad \mathit{SUT}(s) \subseteq \mathit{SUT}(s')$$

$$\Leftrightarrow \ \{ i \mid i \text{ uioco } s \} \subseteq \{ i \mid i \text{ uioco } s' \}$$



Requires refinement preorder $\leq$ on specifications.

ioco / uioco are not refinement preorders and are only defined for input-enabled implementations

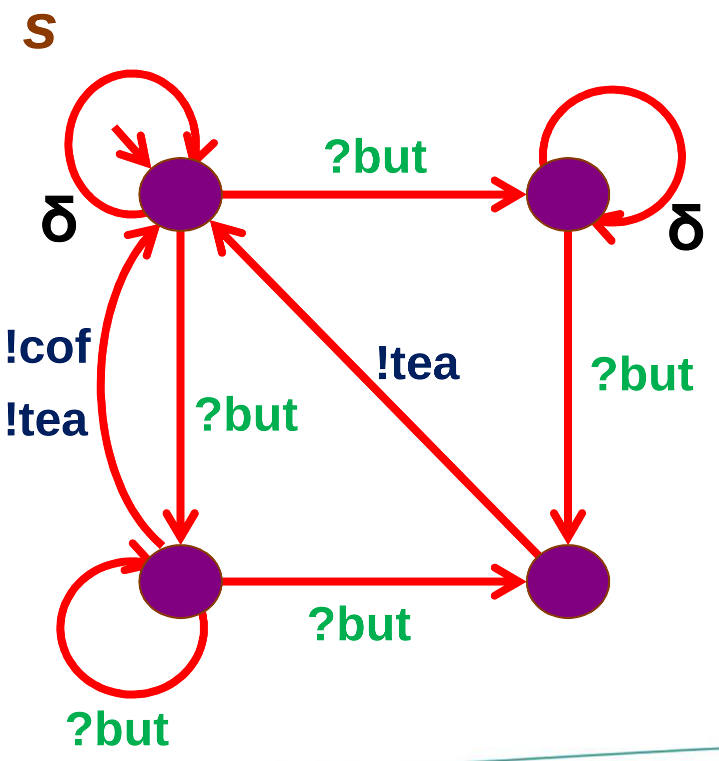
Set of Required Traces

$\mathit{Rtraces}\ (s) \ =_{def} \ \{ \ \sigma \in \underline{\mathit{Utraces}}\ (s) \ \mid$

δ is not a substring of σ,

σ does not end with δ,

out (s after σ) $\neq L_U \cup \{ \delta \} \ \}$



?but δ δ ?but | $\notin$ $\underline{\mathit{Rtraces}}$ (s)

?but |

δ | $\in$ $\underline{\mathit{Utraces}}$ (s)

Set of Required Traces

Rtraces **throw away superfluous traces, and only those**

1. For input enabled implementations:

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

$$\Leftrightarrow\quad \forall\, \sigma \in \textbf{\textit{Rtraces}}\ (\mathbf{s}) :\ \textbf{\textit{out}}\ (\mathbf{i}\ \mathbf{after}\ \sigma)\ \subseteq\ \textbf{\textit{out}}\ (\mathbf{s}\ \mathbf{after}\ \sigma)$$

2. ***Rtraces*** is “minimal” : For $\textbf{\textit{A}}\ \subset\ \textbf{\textit{Rtraces}}\ (\mathbf{s})$ and $\textbf{\textit{A}}\ \neq\ \textbf{\textit{Rtraces}}\ (\mathbf{s})$,

there exists an input-enabled **i** such that

$$\forall\, \sigma \in \textbf{\textit{A}} :\ \textbf{\textit{out}}\ (\mathbf{i}\ \mathbf{after}\ \sigma)\ \subseteq\ \textbf{\textit{out}}\ (\mathbf{s}\ \mathbf{after}\ \sigma)$$

and $\mathbf{i}\ \ \not\mathrel{\mathbf{uioco}}\ \ \mathbf{s}$

From Required Traces to **wioco**

Refinement preorder ≤ is given by **wioco**,
considering superfluous traces and non-input enabledness

s wioco s' $=_{def}$ $\forall \sigma \in$ ***Rtraces*** **(s') :**

1. ***out*** **(s after** σ**)** $\subseteq$ ***out*** **(s' after** σ**)**

2. $\forall \sigma_1 \leq \sigma$ **:** ***in*** **(s after** σ_1**)** $\supseteq$ ***Rin*** **(s' after** σ_1 **)**

in **(s after** σ_1**)** $=_{def}$ $\{$ **a?** $\in$ L_I **|** **s after** σ_1 **must a?** $\}$

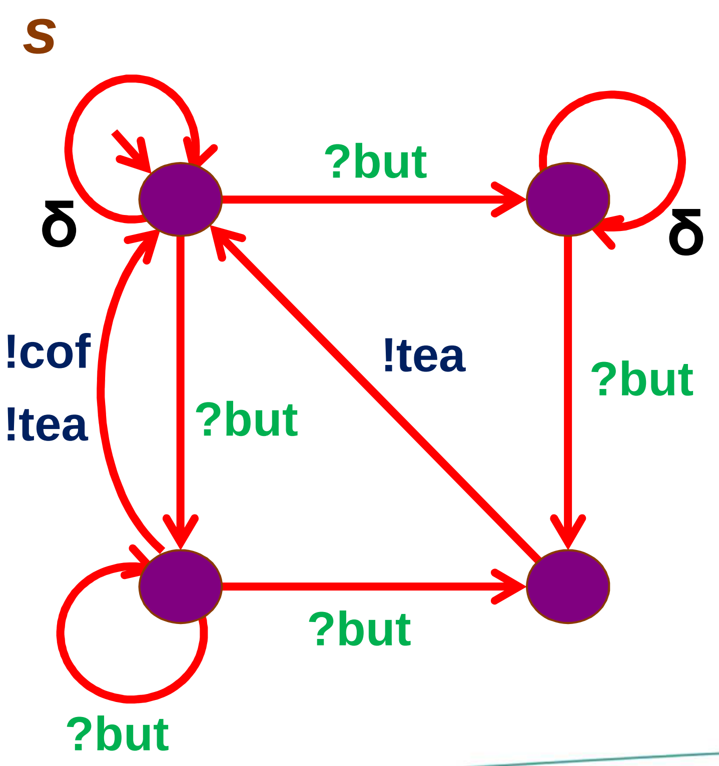
Rin **(s' after** σ_1 **)** $=_{def}$

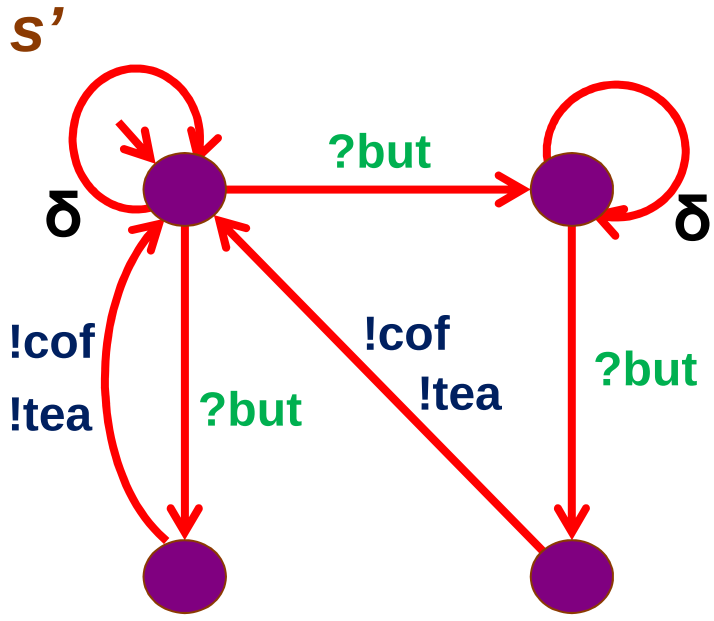
$\{$ **a?** $\in$ ***in*** **(s after** σ_1**)** **|** $\exists \sigma_2 \in$ ***Rtraces*** **(s') :** σ_1 **a?** $\leq \sigma_2$ $\}$

A Weaker Specification through **wioco**

s wioco s' ⇔ *SUT*(s) ⊆ *SUT*(s')

s' is a weaker than s:
- remove inputs
- add outputs

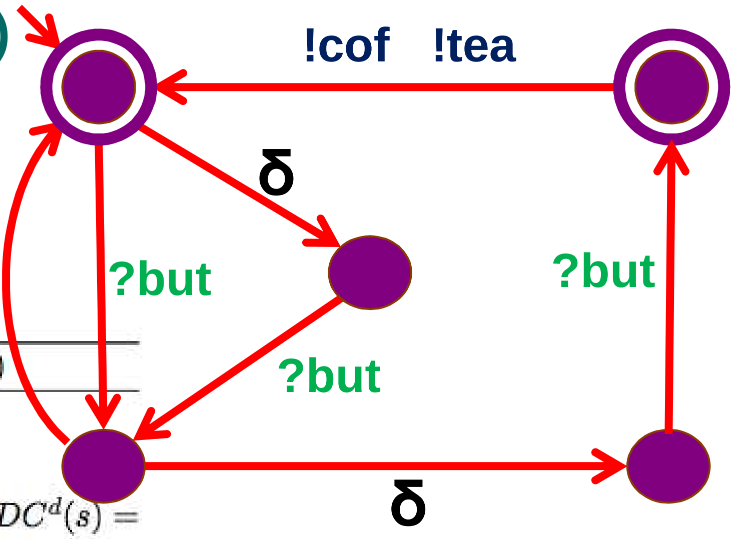


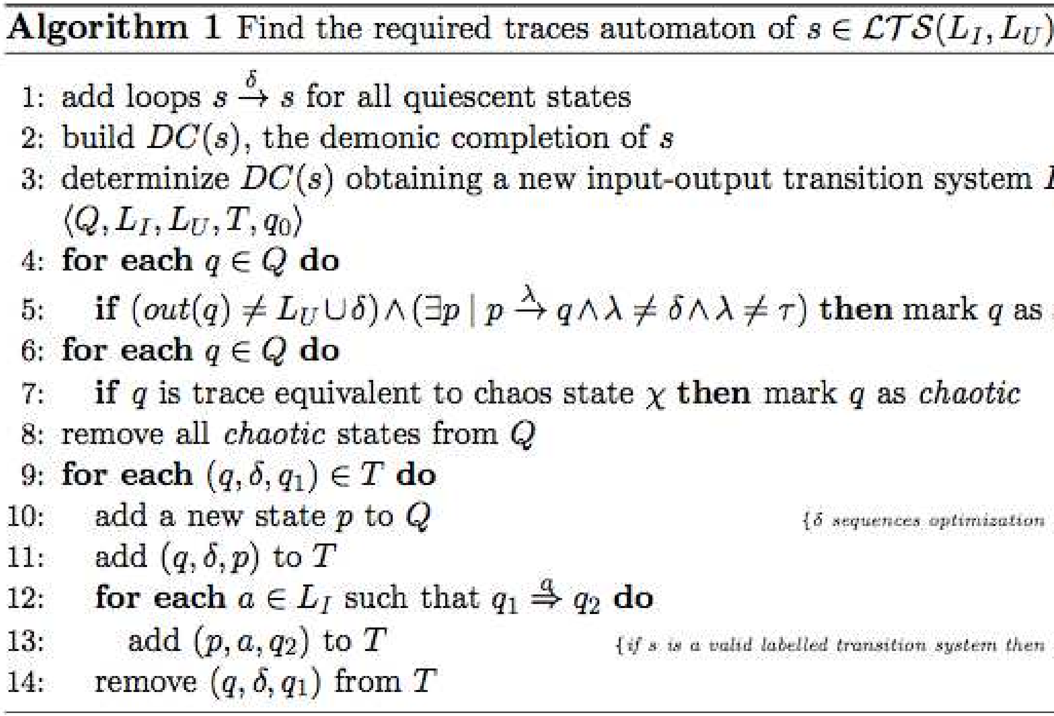


Required Traces Automaton

$\sigma \in$ ***Rtraces*** **(s)**

$\Leftrightarrow$ σ accepted by ***RTA*****(s)**





Algorithm 1 Find the required traces automaton of $s \in \mathcal{LTS}(L_I, L_U)$

```
add loops $s \xrightarrow{\delta} s$ for all quiescent states
build $DC(s)$, the demonic completion of $s$
determinize $DC(s)$ obtaining a new input-output transition system $DC^d(s) = \langle Q, L_I, L_U, T, q_0 \rangle$
for each $q \in Q$ do
    if $(out(q) \neq L_U \cup \delta) \wedge (\exists p \mid p \xrightarrow{\lambda} q \wedge \lambda \neq \delta \wedge \lambda \neq \tau)$ then mark $q$ as accepting
for each $q \in Q$ do
    if $q$ is trace equivalent to chaos state $\chi$ then mark $q$ as chaotic
remove all chaotic states from $Q$
for each $(q, \delta, q_1) \in T$ do
    add a new state $p$ to $Q$                {δ sequences optimization steps 1 and 2}
    add $(q, \delta, p)$ to $T$
    for each $a \in L_I$ such that $q_1 \overset{a}{\Rightarrow} q_2$ do
        add $(p, a, q_2)$ to $T$       {if s is a valid labelled transition system then $out(q) \not\subseteq L_U$}
    remove $(q, \delta, q_1)$ from $T$
```

MBT : Some Tools - ioco

- AETG
- Agatha
- Agedis
- All4Tec MaTeLo
- Autolink
- Axini Test Manager
- Conformiq Qtronic
- Cooper
- G∀st
- Gotcha
- JTorX
- NModel
- ParTeG
- Phact/The Kit
- QuickCheck
- Reactis
- RT-Tester
- SaMsTaG
- SeppMed MBTsuite
- Smartesting CertifyIt
- Spec Explorer
- Statemate
- STG
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TGV
- TorX
- TorXakis
- T-Vec
- Uppaal Cover
- Uppaal-Tron
- Tveda
- . . . . . .

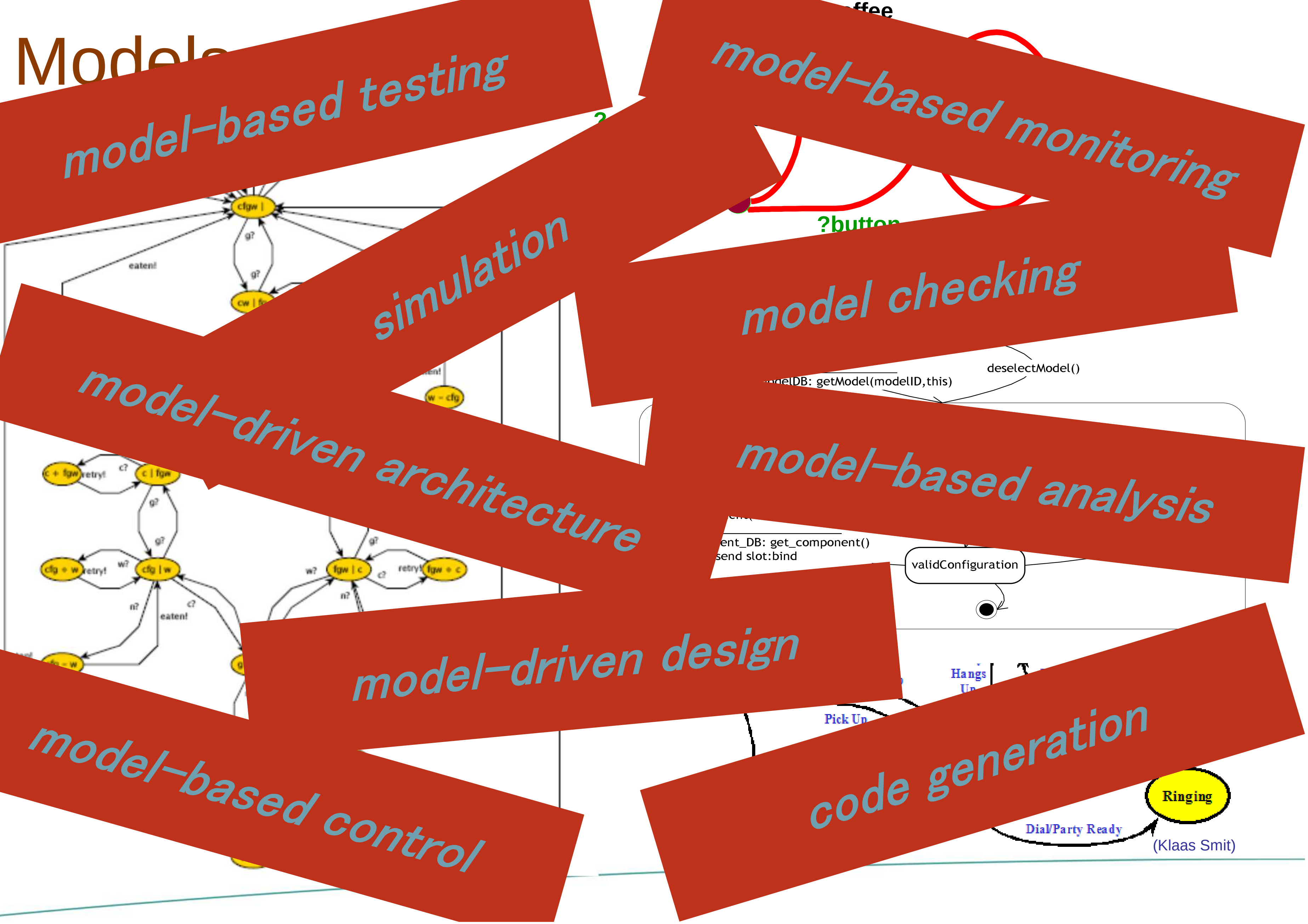
MBT : Some Tools - commercial

- AETG
- Agatha
- Agedis
- All4Tec MaTeLo
- Autolink
- Axini Test Manager
- Conformiq Qtronic
- Cooper
- G∀st
- Gotcha
- JTorX
- NModel
- ParTeG
- Phact/The Kit
- QuickCheck
- Reactis
- RT-Tester
- SaMsTaG
- SeppMed MBTsuite
- Smartesting CertifyIt
- Spec Explorer
- Statemate
- STG
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TGV
- TorX
- TorXakis
- T-Vec
- Uppaal-Cover
- Uppaal-Tron
- Tveda
- ……

Learning

Test-Based Modelling

Models



(Klaas Smit)

model-based testing

model-based monitoring

simulation

model checking

model-driven architecture

model-based analysis

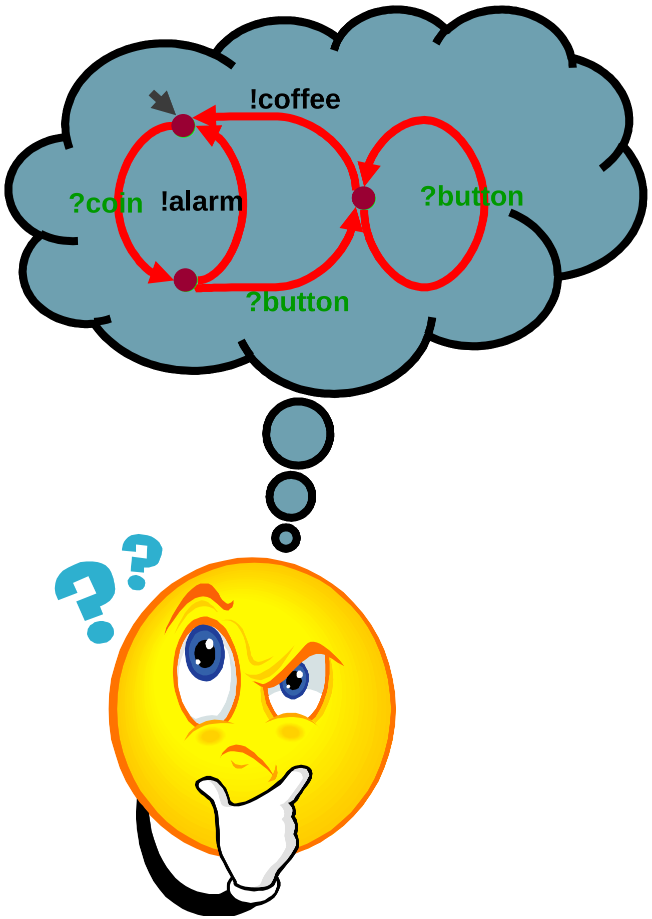
model-driven design

model-based control

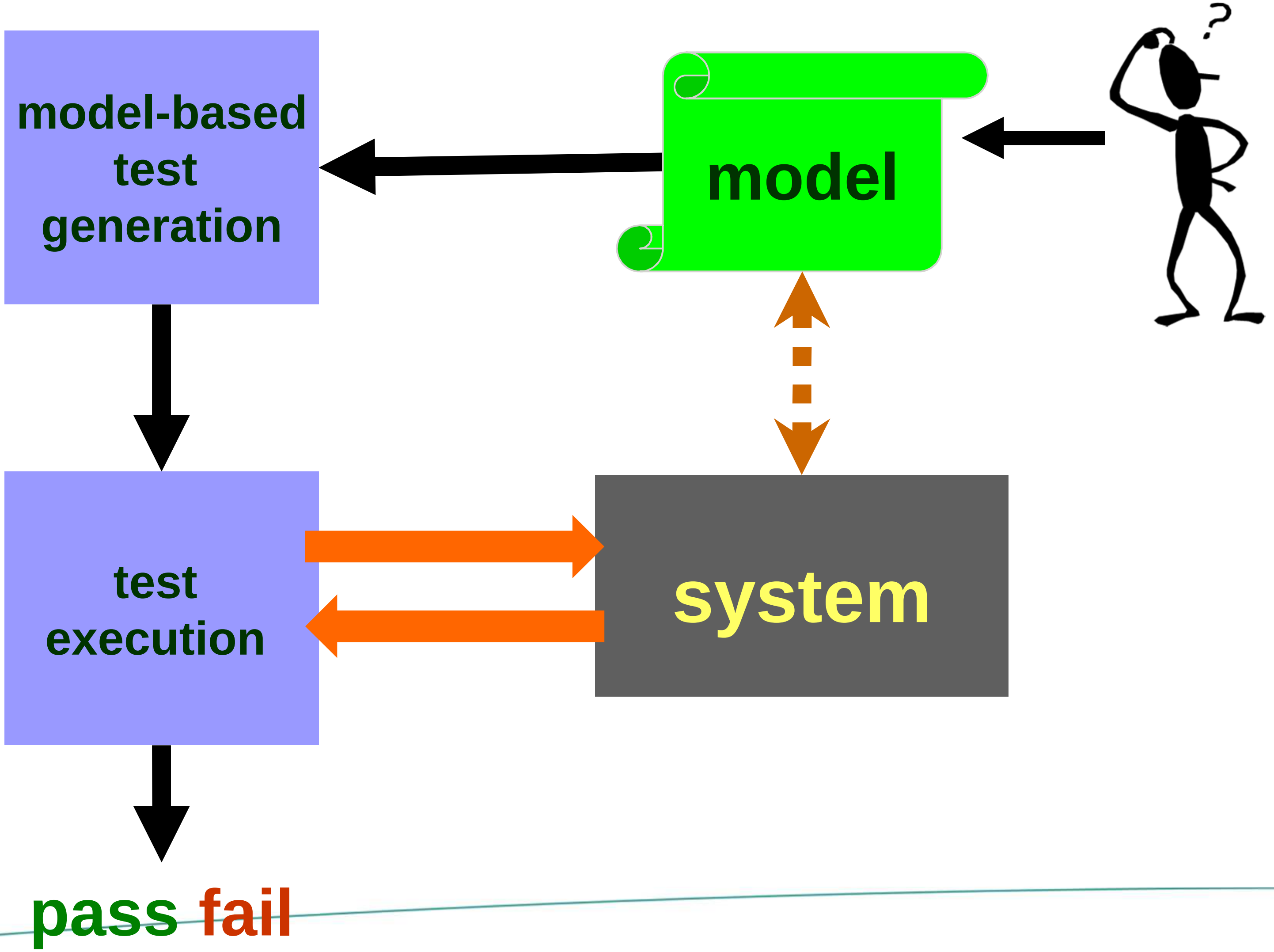
code generation

Models

- Everybody wants models
- Doing nice things with models
  - model checking, simulation, ...
- **How to get these models?**
  - *in particular for*: legacy, third-party, out-sourced, off-the-shelf, ... components
- **Does the model correspond with the real system?**



Testing : Model-Based Testing


model-based test generation

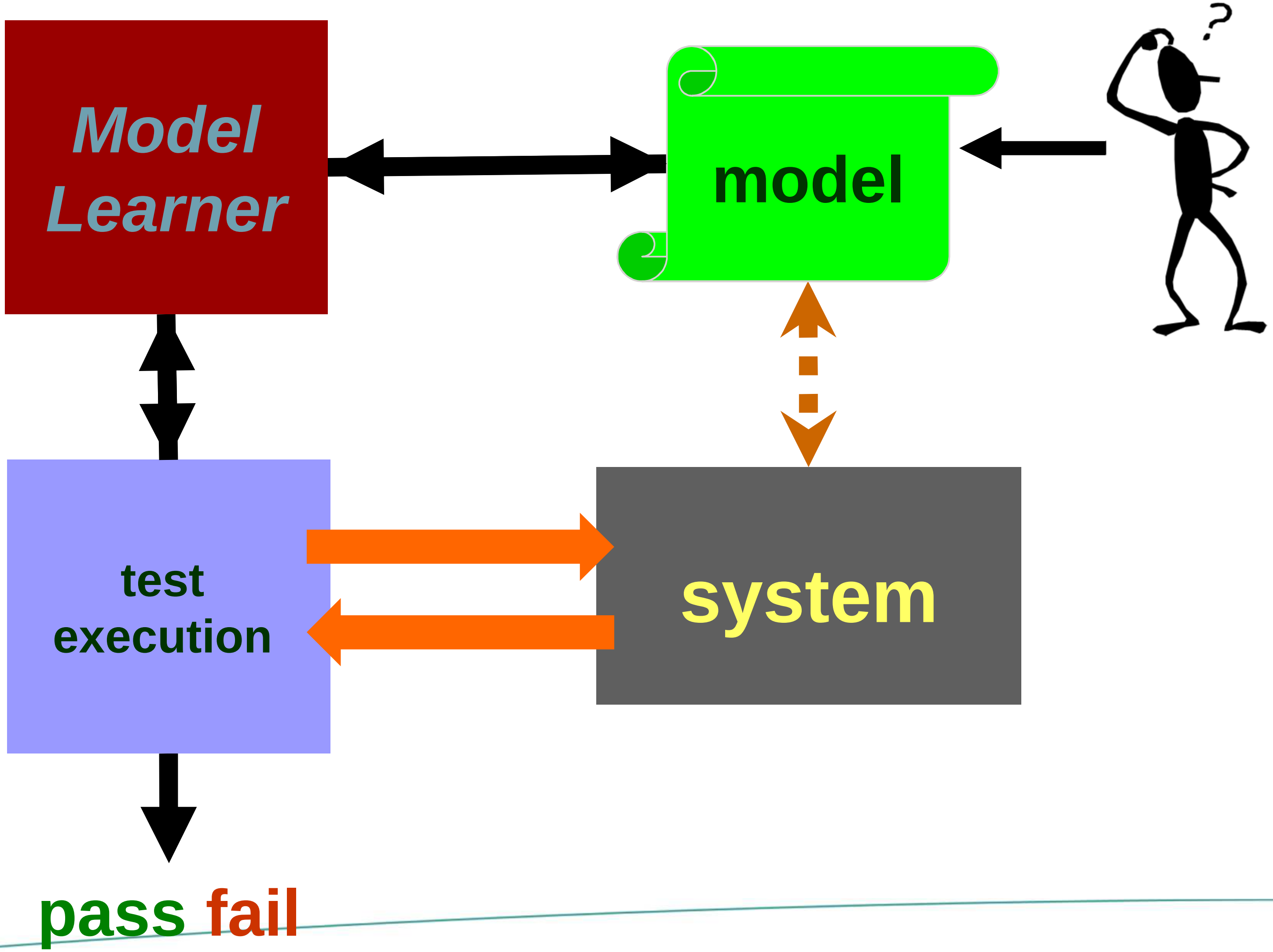
model

test execution

system

pass fail

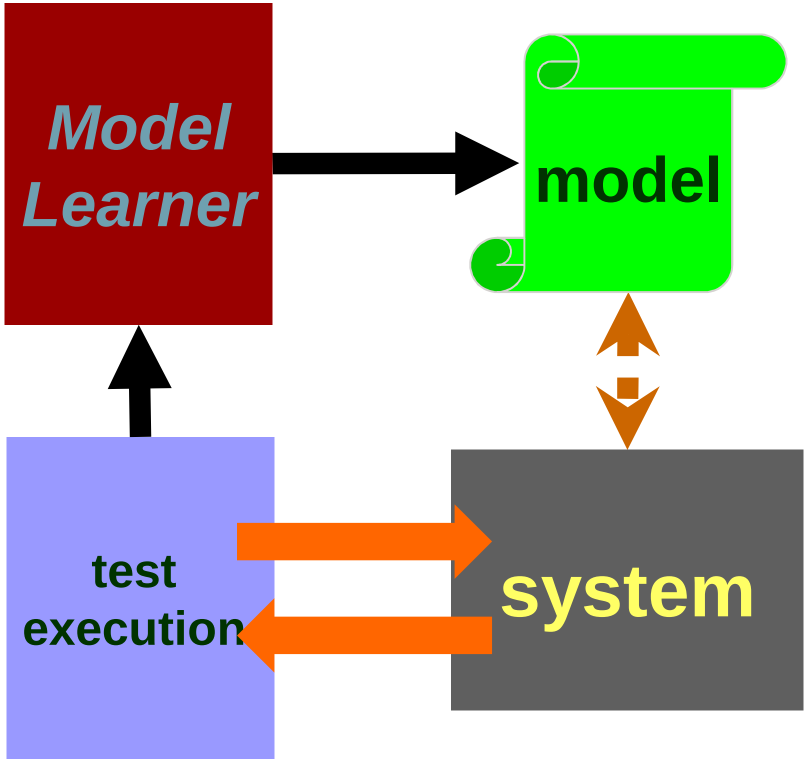
Test-Based Modeling



Test-Based Modeling

Automatically learning a model of the behavior of a system from observations made with testing

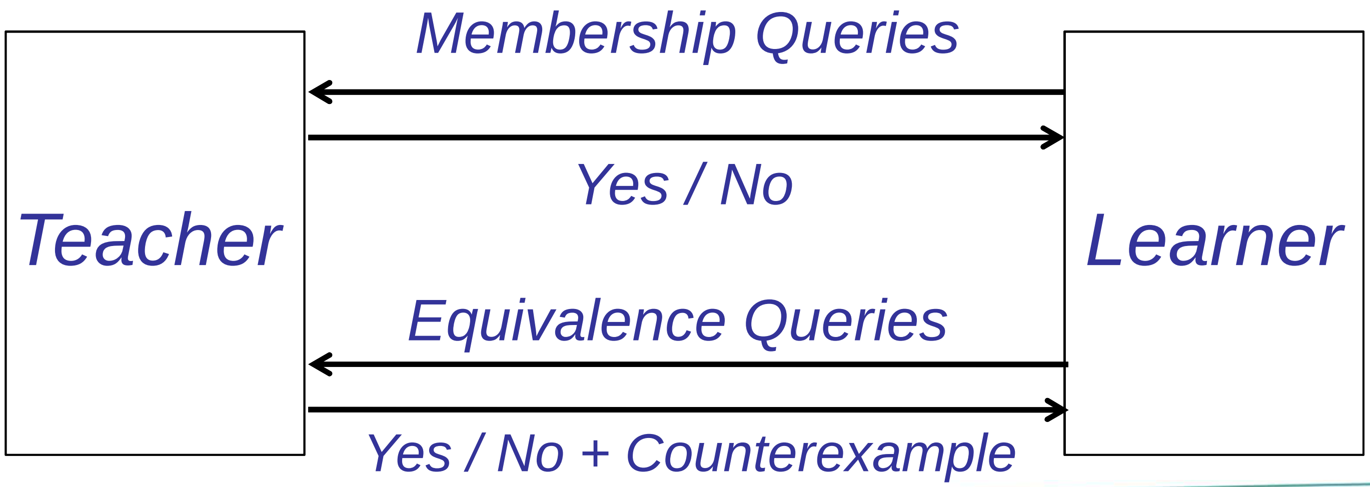
- *test-based modeling*
- *automata learning*
- *black-box reverse engineering*
- *observation-based modeling*
- *behavior capture and test*
- *grammatical inference*



Learning Models of Automata

- Active learning is an active research area:
  - D. Angluin (1987) : *L**-algorithm
  - *LearnLib* : Tool for FSM learning
  - *.....*

Learning Finite Automata with L *** :**



Teacher

Membership Queries

Yes / No

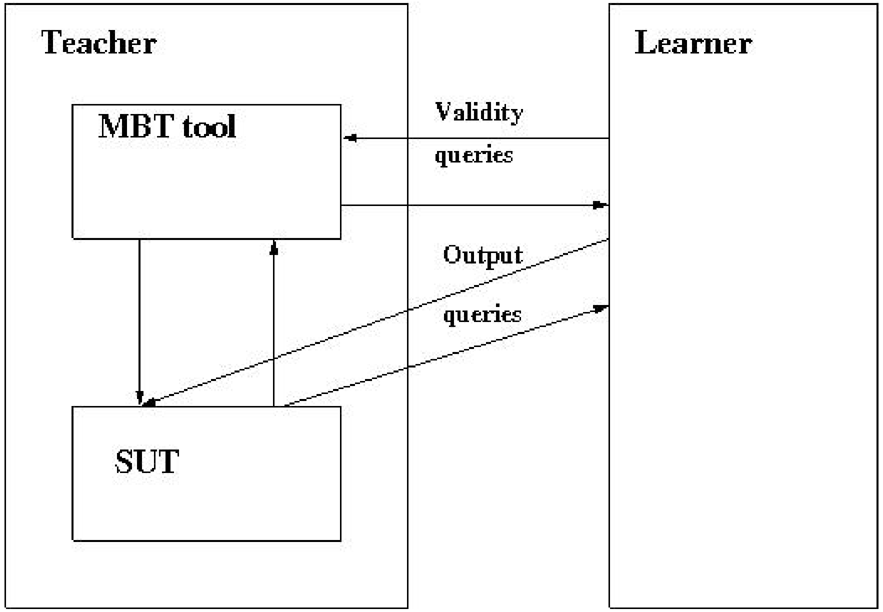
Equivalence Queries

Yes / No + Counterexample

Learner

Learning Models of Reactive Systems

- Tool for active learning of Finite State Machines : **LearnLib**
- Developed by group B. Steffen (U. Dortmund)
- Able to learn models with up to 10.000 states



- Learner:
  formulate
  a hypothesis FSM

- Equivalence query
  replaced by
  model-based testing
  of hypothesized model

Application: Banking Cards: Learning the EMV protocol

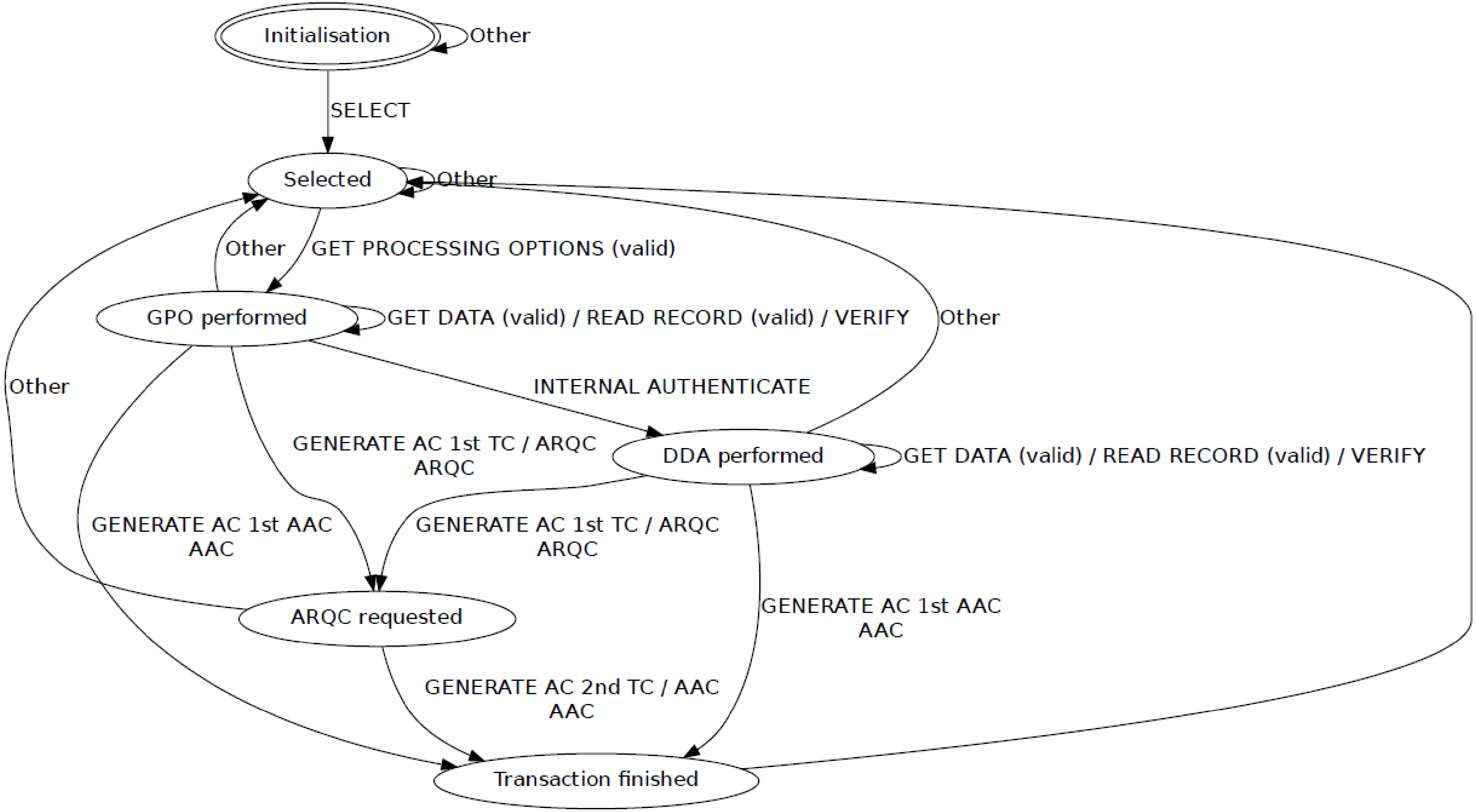
Fides Aarts, Erik Poll, and Joeri de Ruiter

- EMV =
  Europay, Mastercard and Visa
- Models from black-box implementations
- Learn behaviour blindly
- Security: absence of unwanted functionality
- Correctness/conformance:
  presence of required functionality

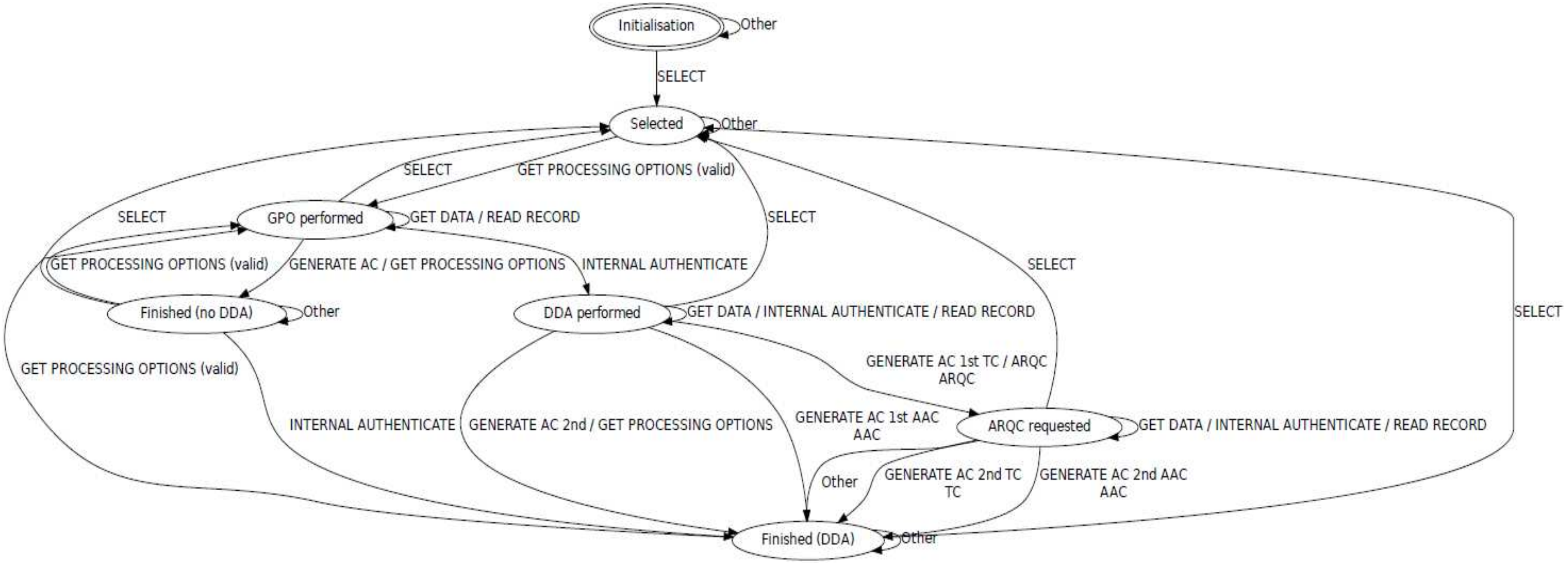
S€PA

Single Euro Payments Area

Model of Maestro app on Dutch banking card

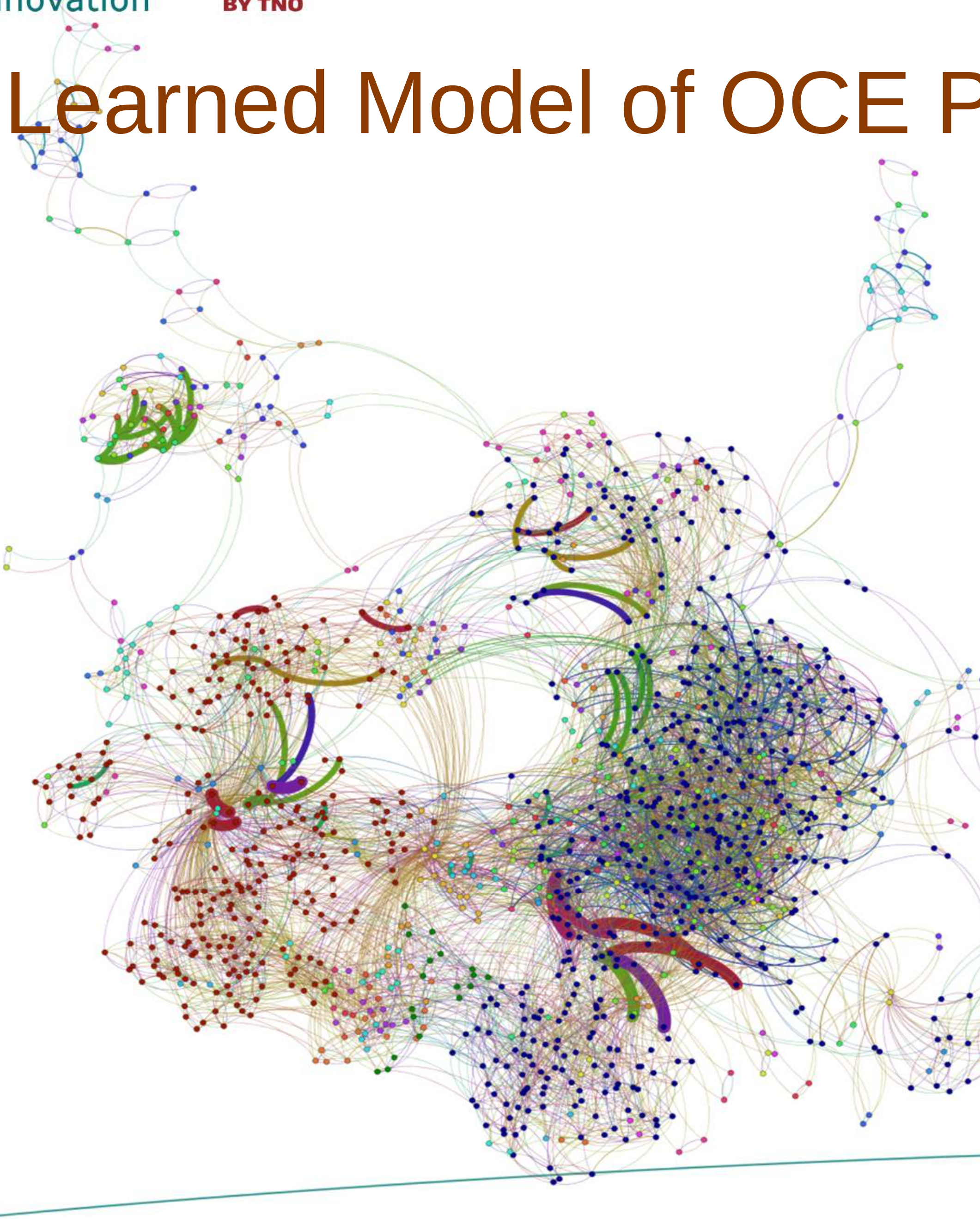


Model of Maestro app on German banking card

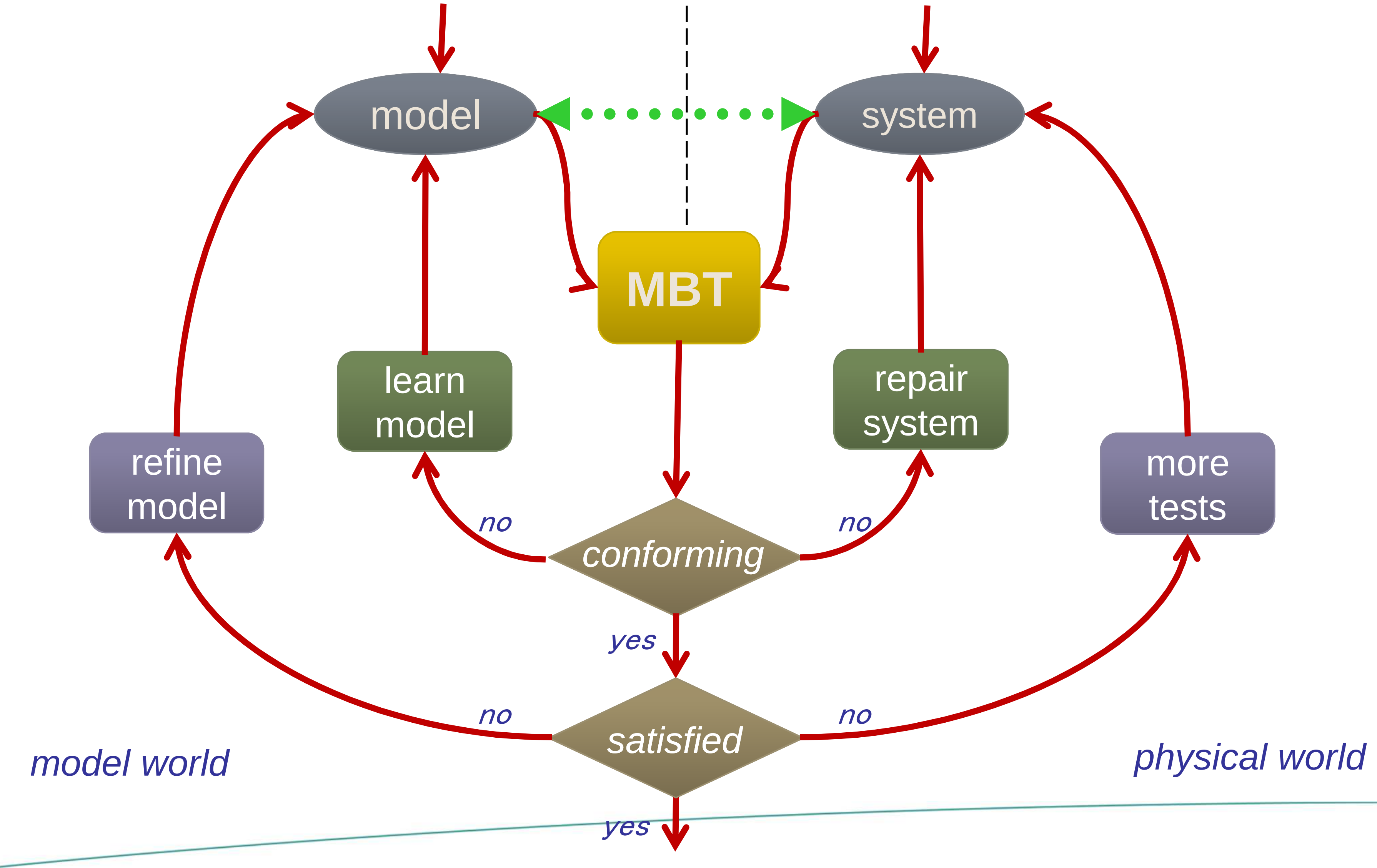


- Dutch vs. German banking card: different handling of errors

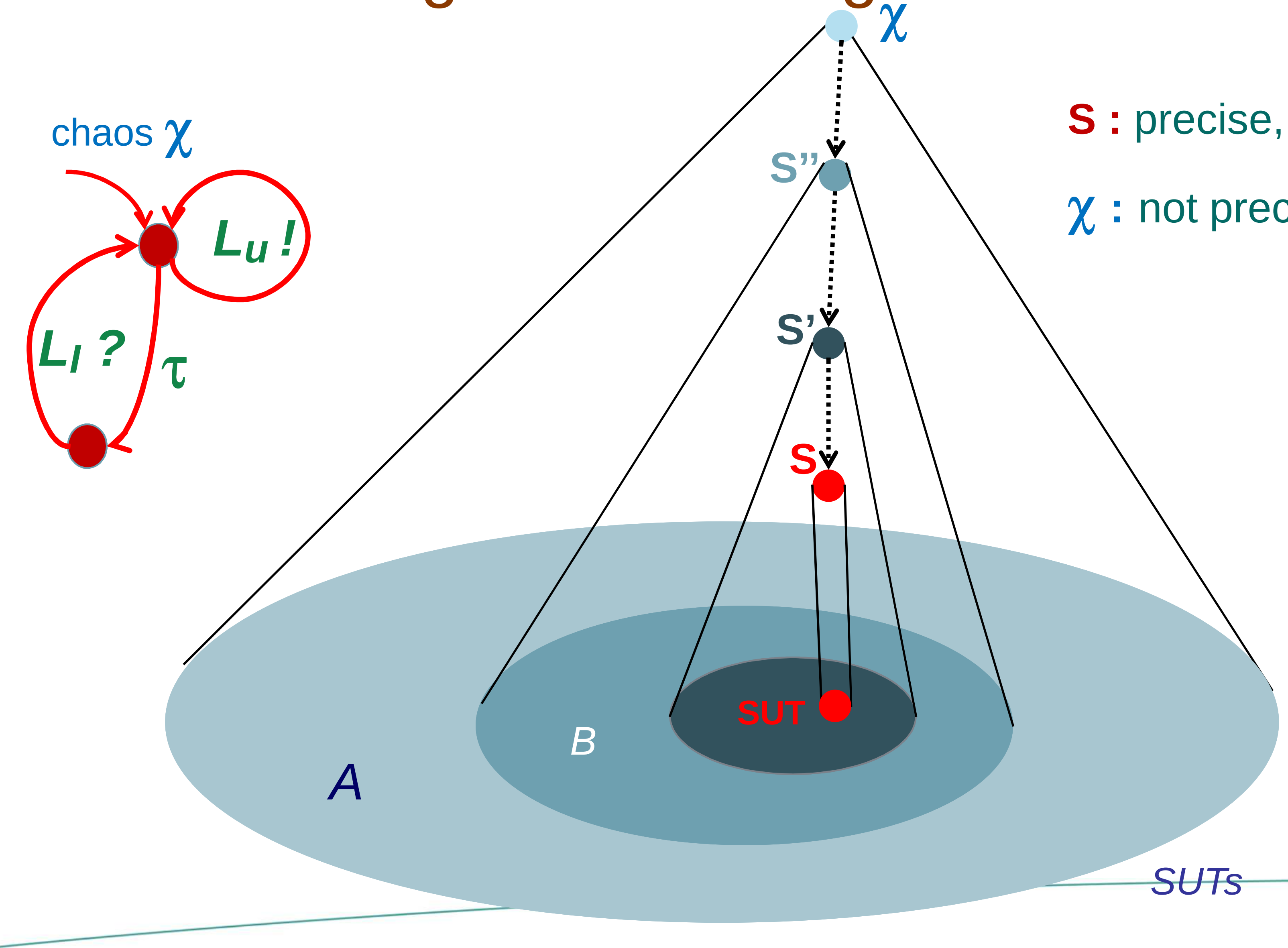
Learned Model of OCE Printer Module



Model-Based Testing & Test-Based Modeling



Test Coverage = Learning Precision



S : precise, expensive

χ **:** not precise, cheap

Model-Based Testing

There is Nothing More Practical

than a Good Theory