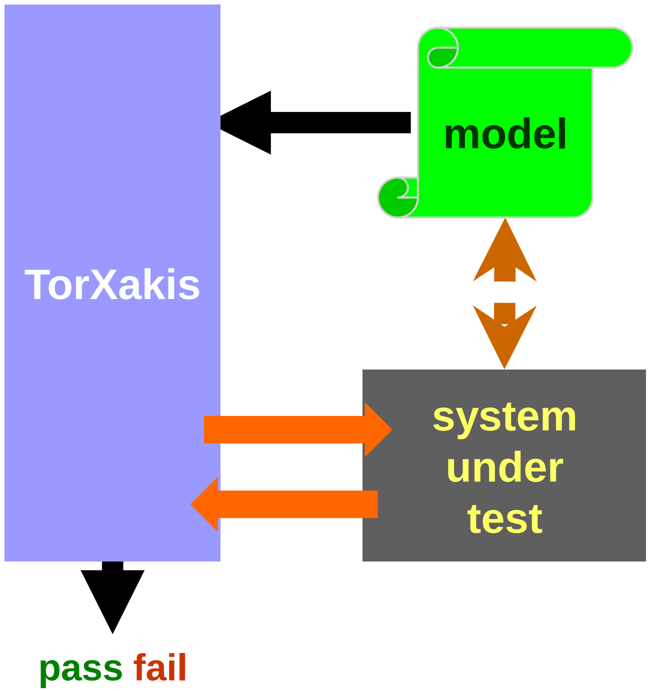
Model-Based Testing

with TorXakis

TestNet WerkGroep
Model-Based testing

Jan Tretmans
Piërre van de Laar
TNO – Embedded Systems Innovation



Embedded Systems Innovation BY TNO

Radboud Universiteit

Model-Based Testing

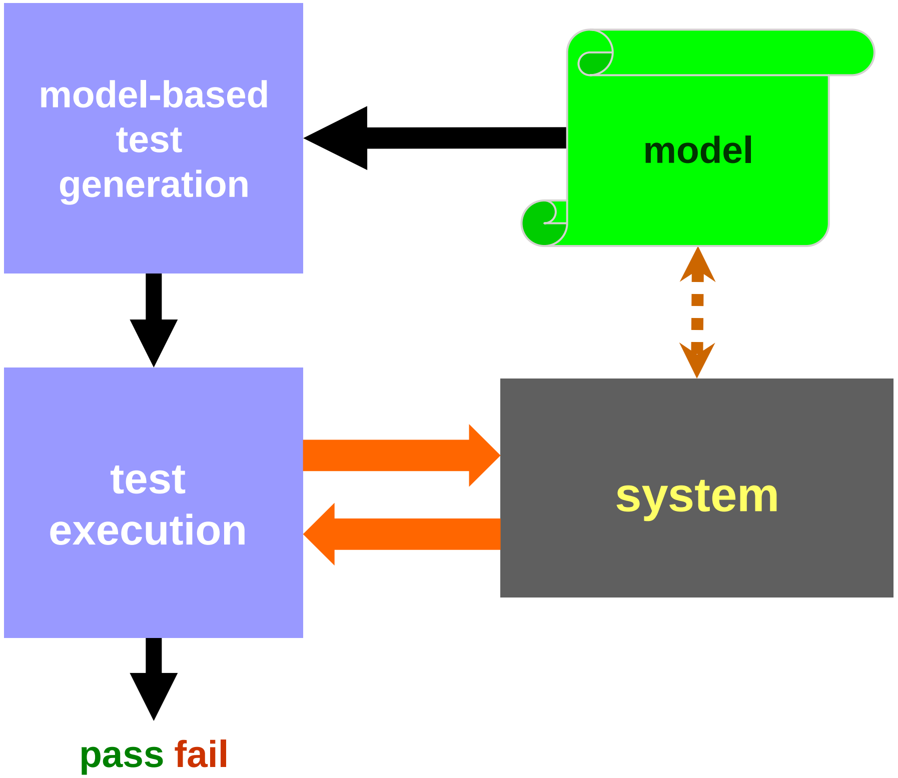
Model-Based Testing

MBT

next step in

test automation

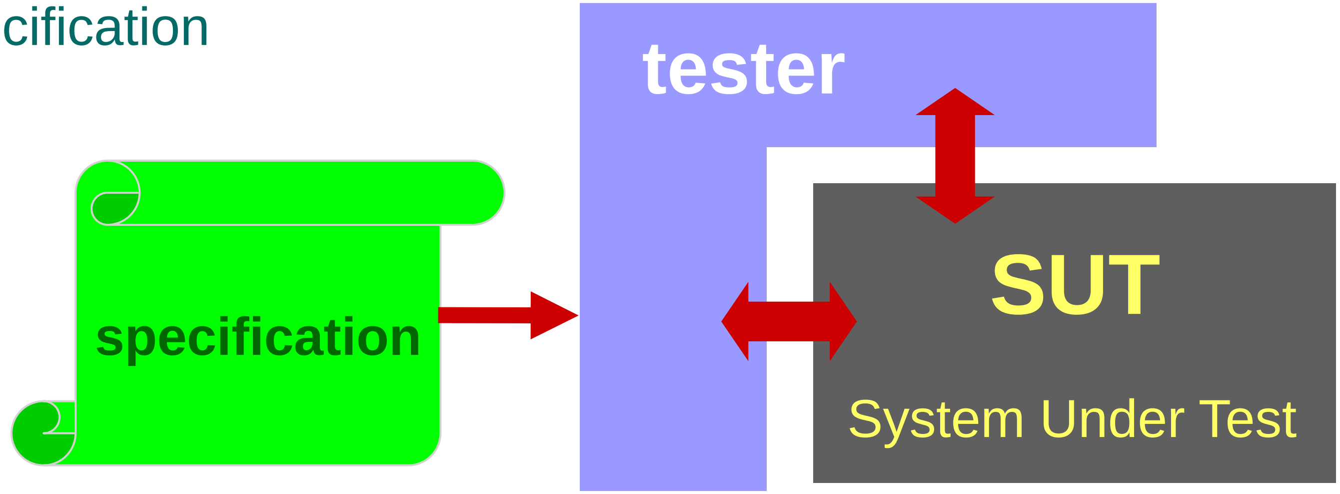
- \+ **test generation**
- \+ **result analysis**



Software & System Testing

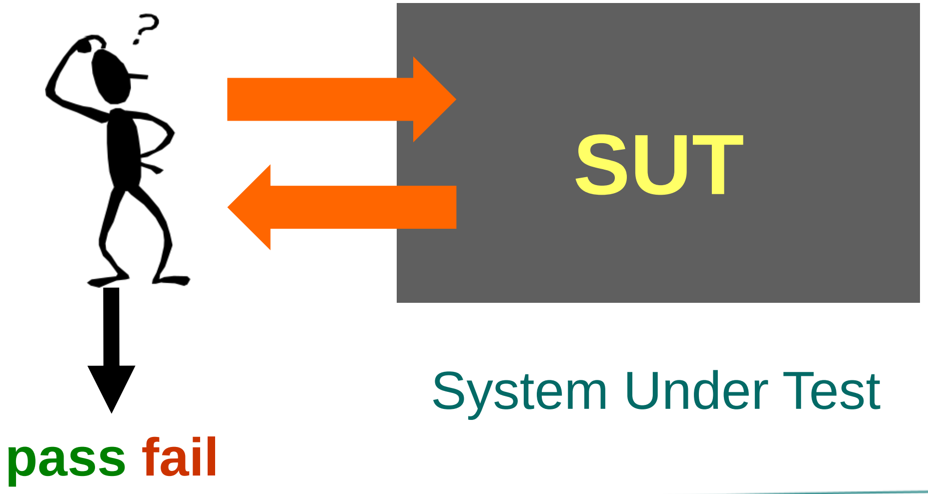
Checking or measuring
some quality characteristics
of an executing software object
by performing experiments
in a controlled way
w.r.t. a specification

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

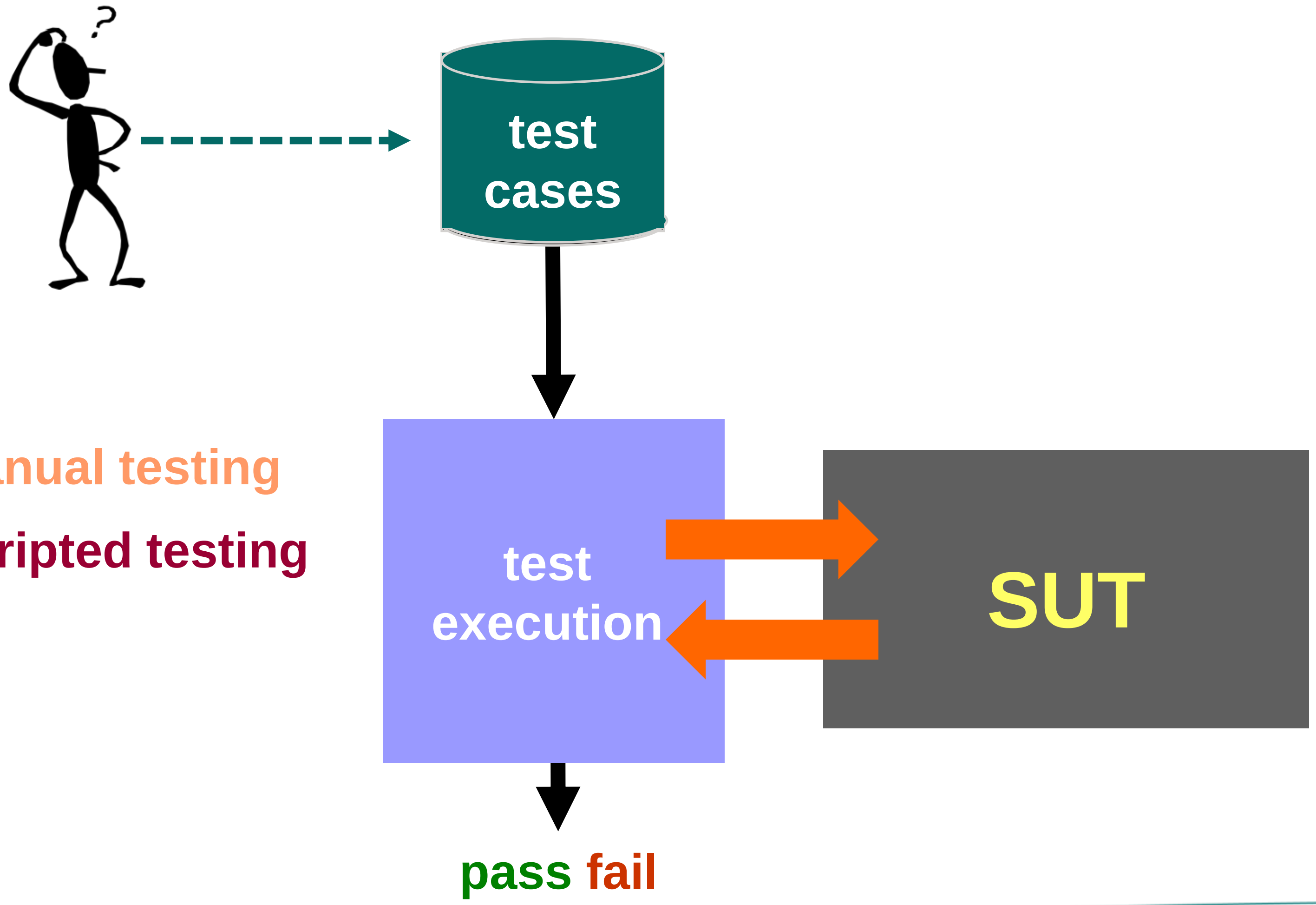


1 : Manual Testing

1. **Manual testing**

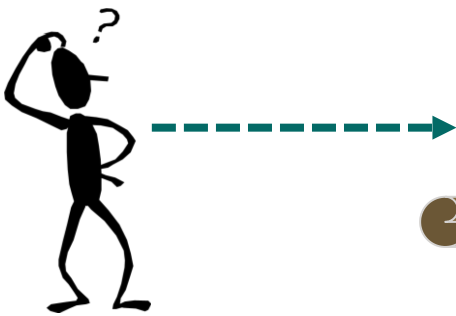


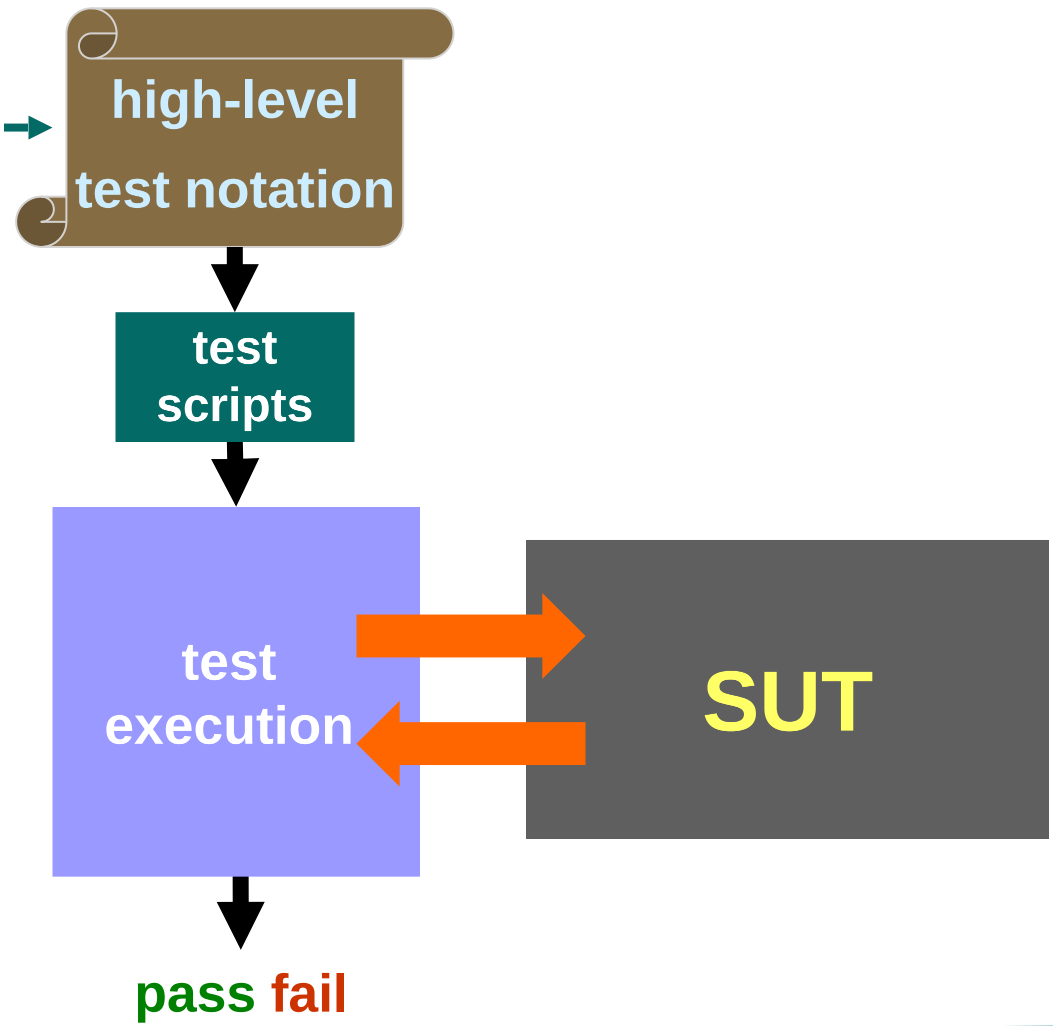
2 : Scripted Testing



1. Manual testing
2. Scripted testing

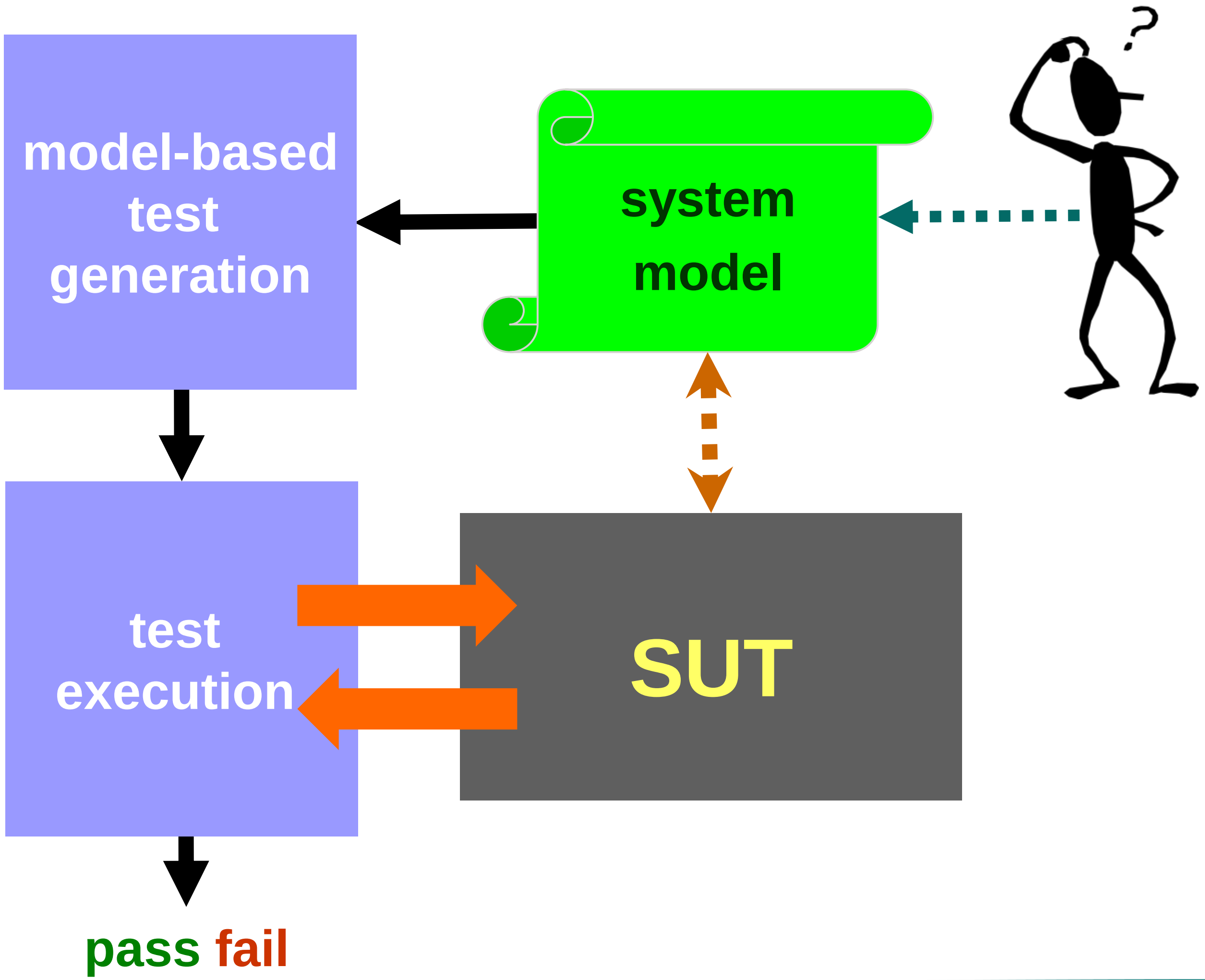
3 : Keyword-Driven Testing





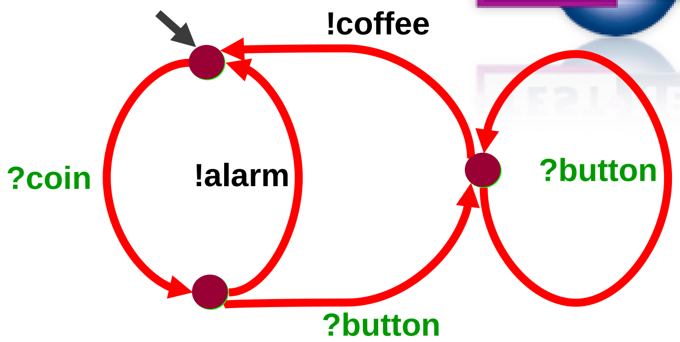
1. Manual testing
2. Scripted testing
3. **Keyword-driven testing**

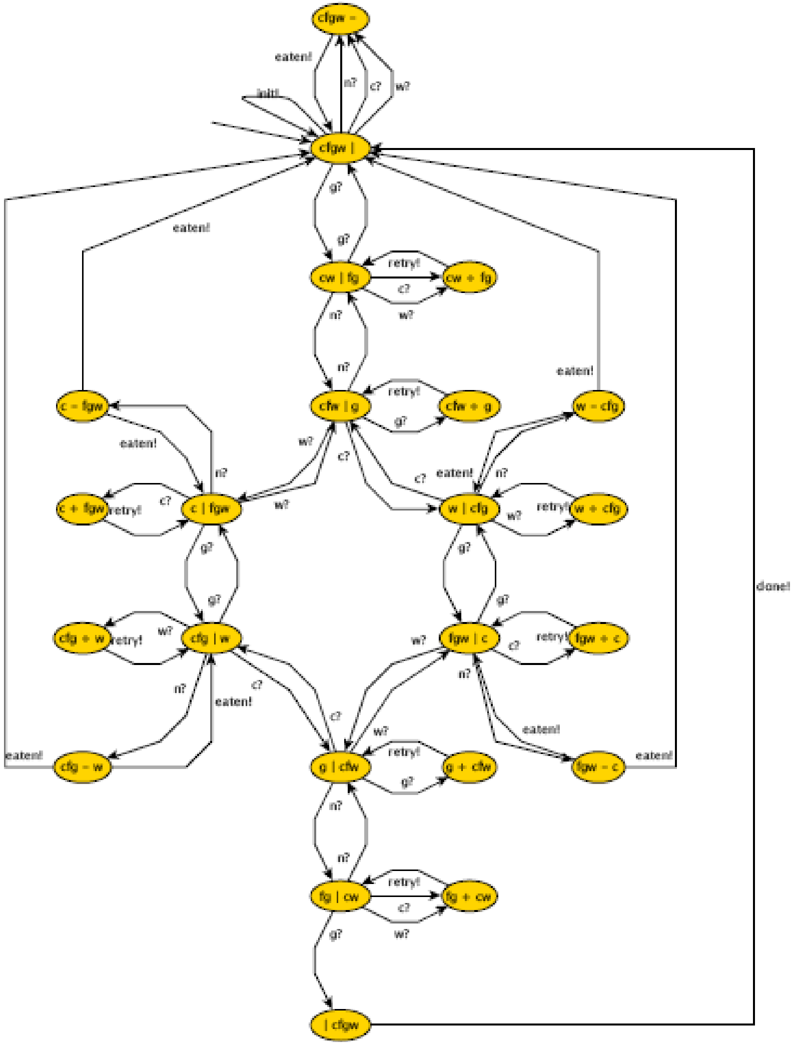
4 : Model-Based Testing

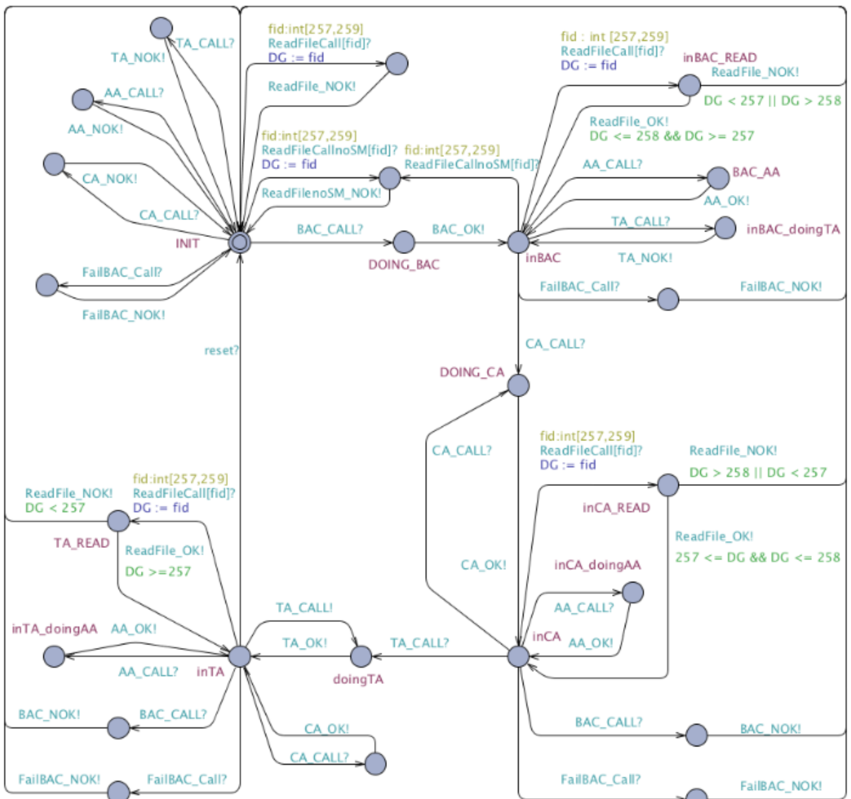


1. Manual testing
2. Scripted testing
3. Keyword-driven testing
4. **Model-based testing**

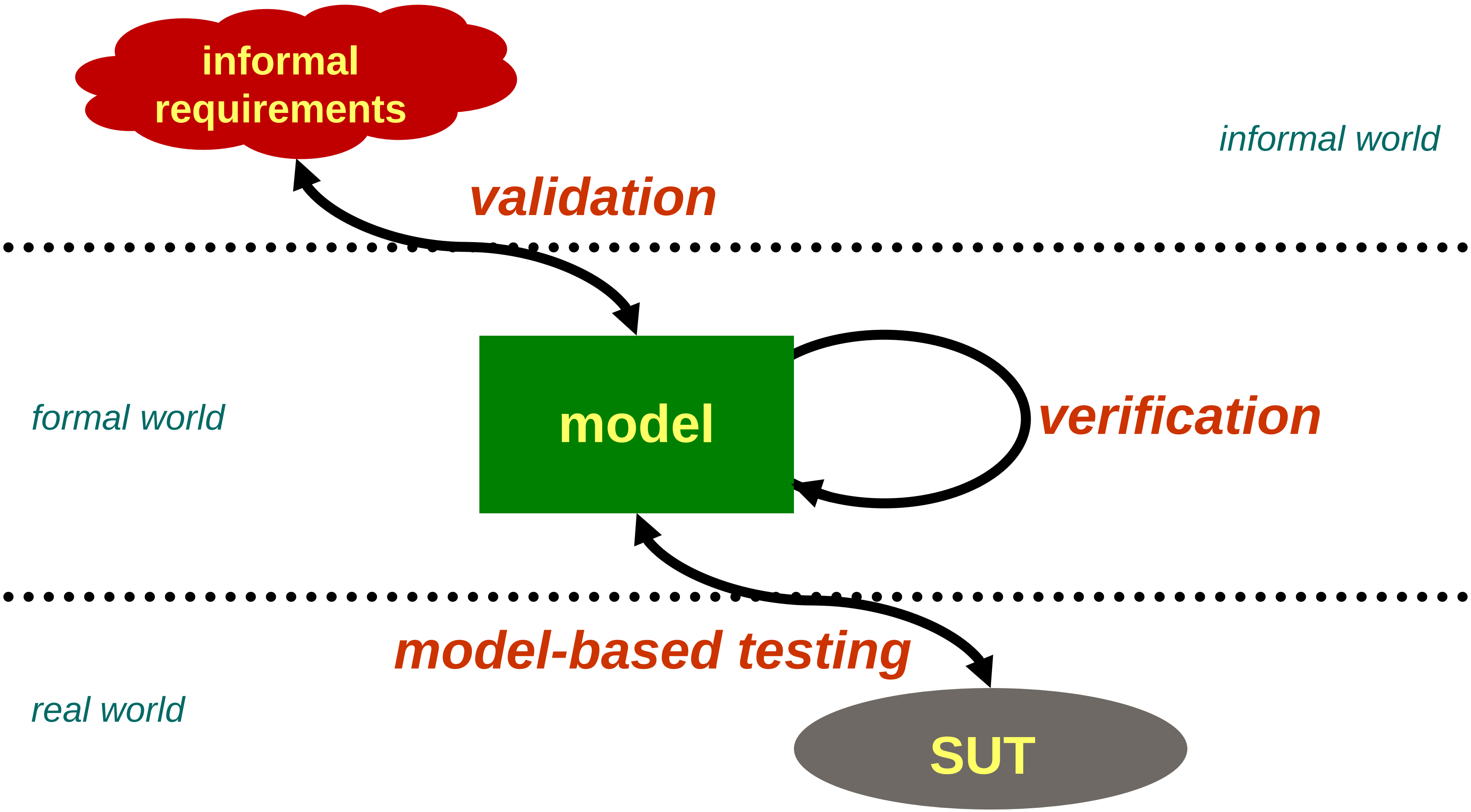
MBT : Example Models



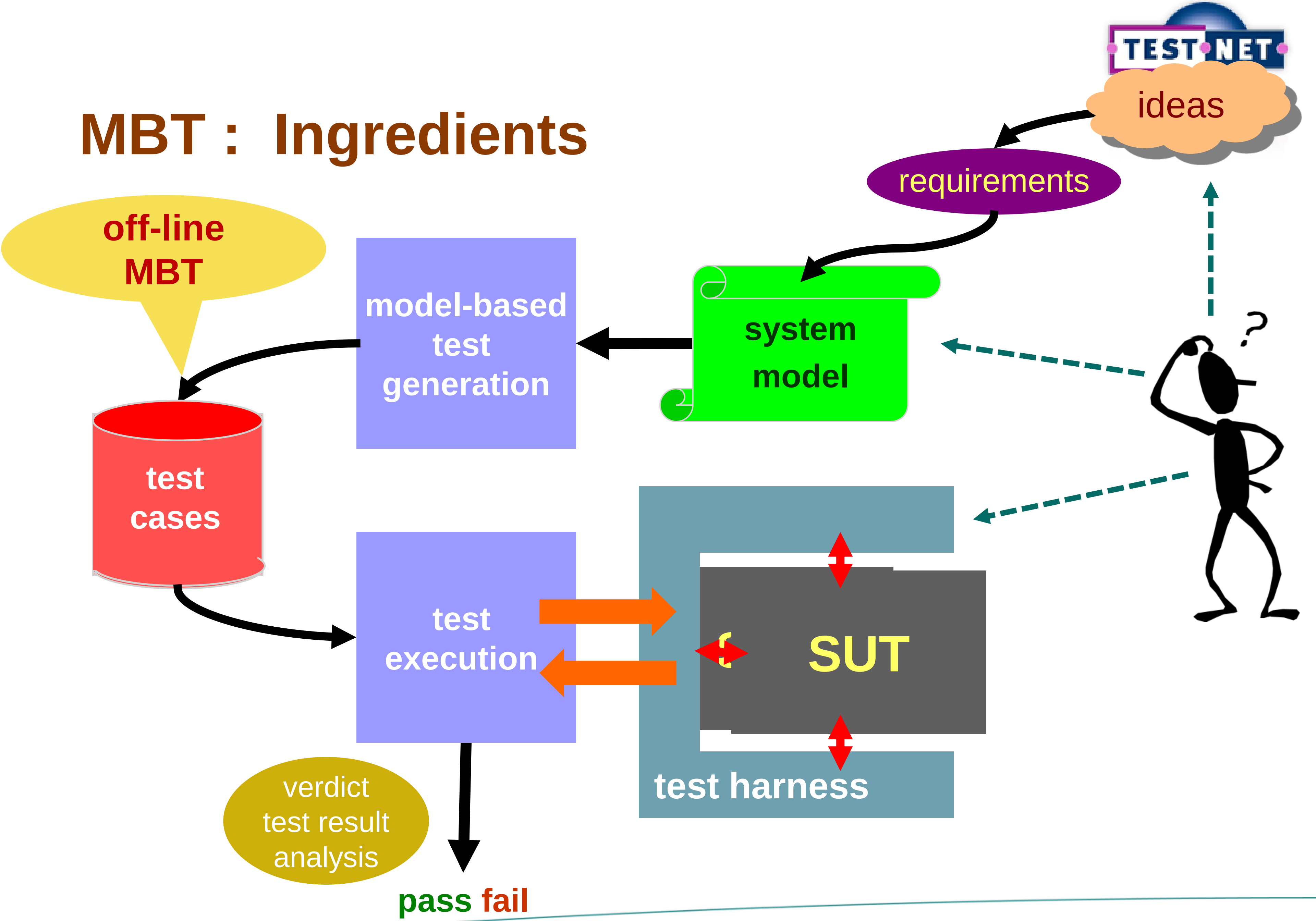




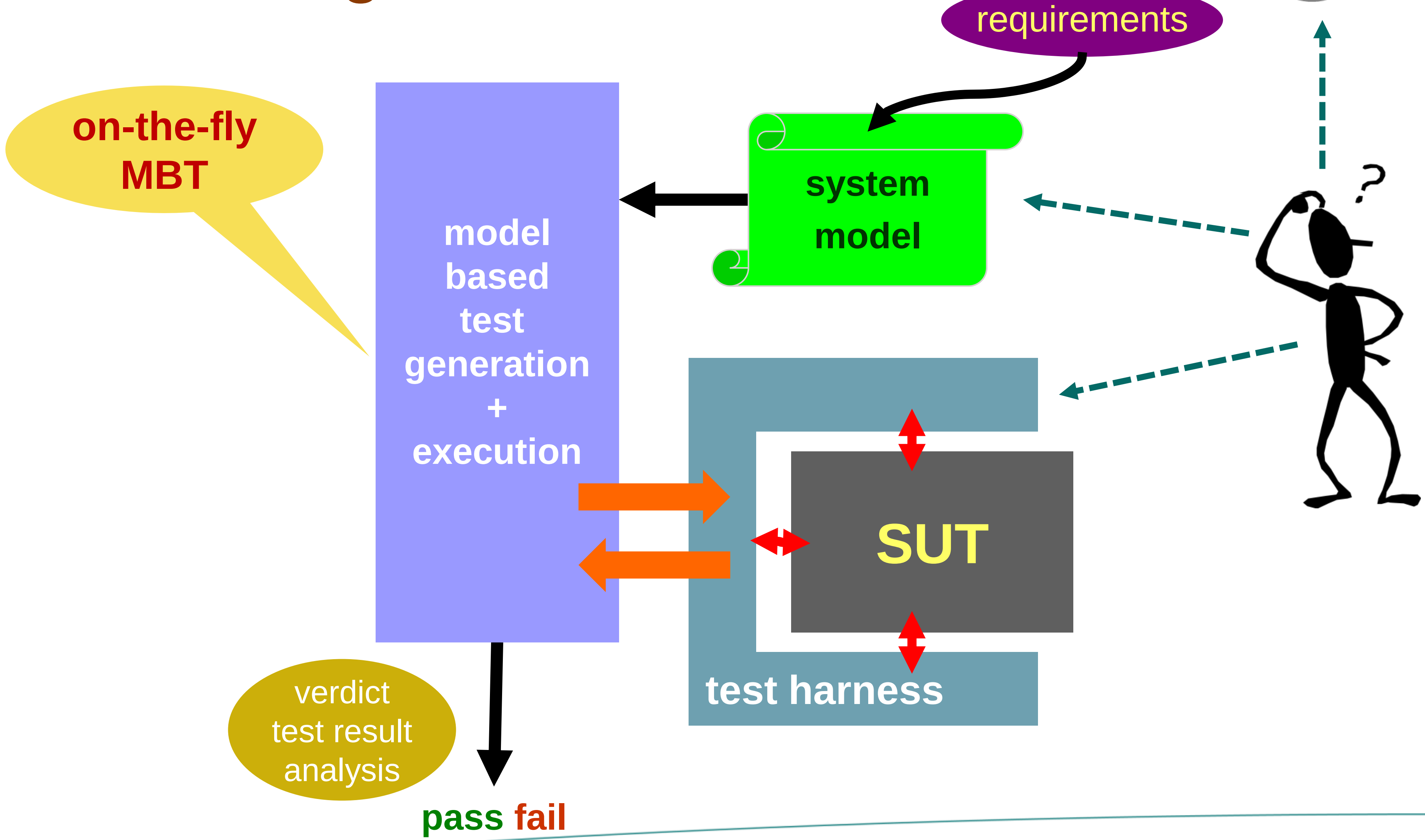
MBT : Validation, Verification, Testing



MBT : Ingredients



MBT : Ingredients



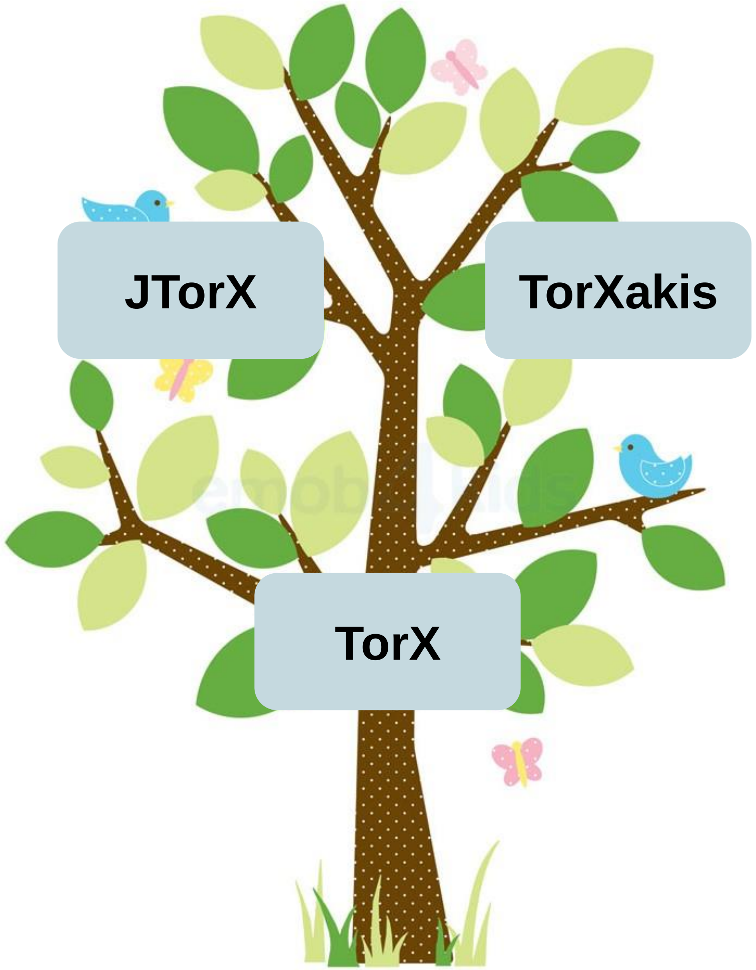
MBT : Many Tools

- AETG
- Agatha
- Agedis
- Autolink
- Axini Test Manager
- Conformiq
- Cooper
- Cover
- DTM
- G∀st
- Gotcha
- Graphwalker
- JTorX
- MaTeLo
- MBT suite
- M-Frame
- MISTA
- NModel
- OSMO
- ParTeG
- Phact/The Kit
- QuickCheck
- Reactis
- Recover
- RT-Tester
- SaMsTaG
- Smartesting CertifyIt
- Spec Explorer
- Statemate
- STG
- Temppo
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TestOptimal
- TGV
- Tigris
- TorX
- TorXakis
- T-Vec
- Uppaal-Cover
- Uppaal-Tron
- Tveda
- . . . .

Model-Based Testing

with TorXakis

Yet Another MBT Tool : TorXakis

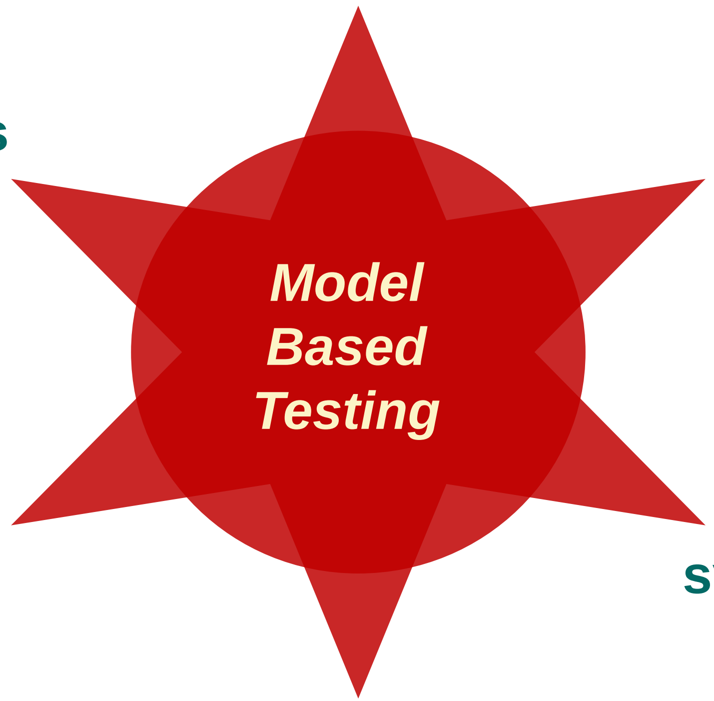


Testing : Trends & Challenges

complexity
size

heterogeneous
components

quest
for
quality

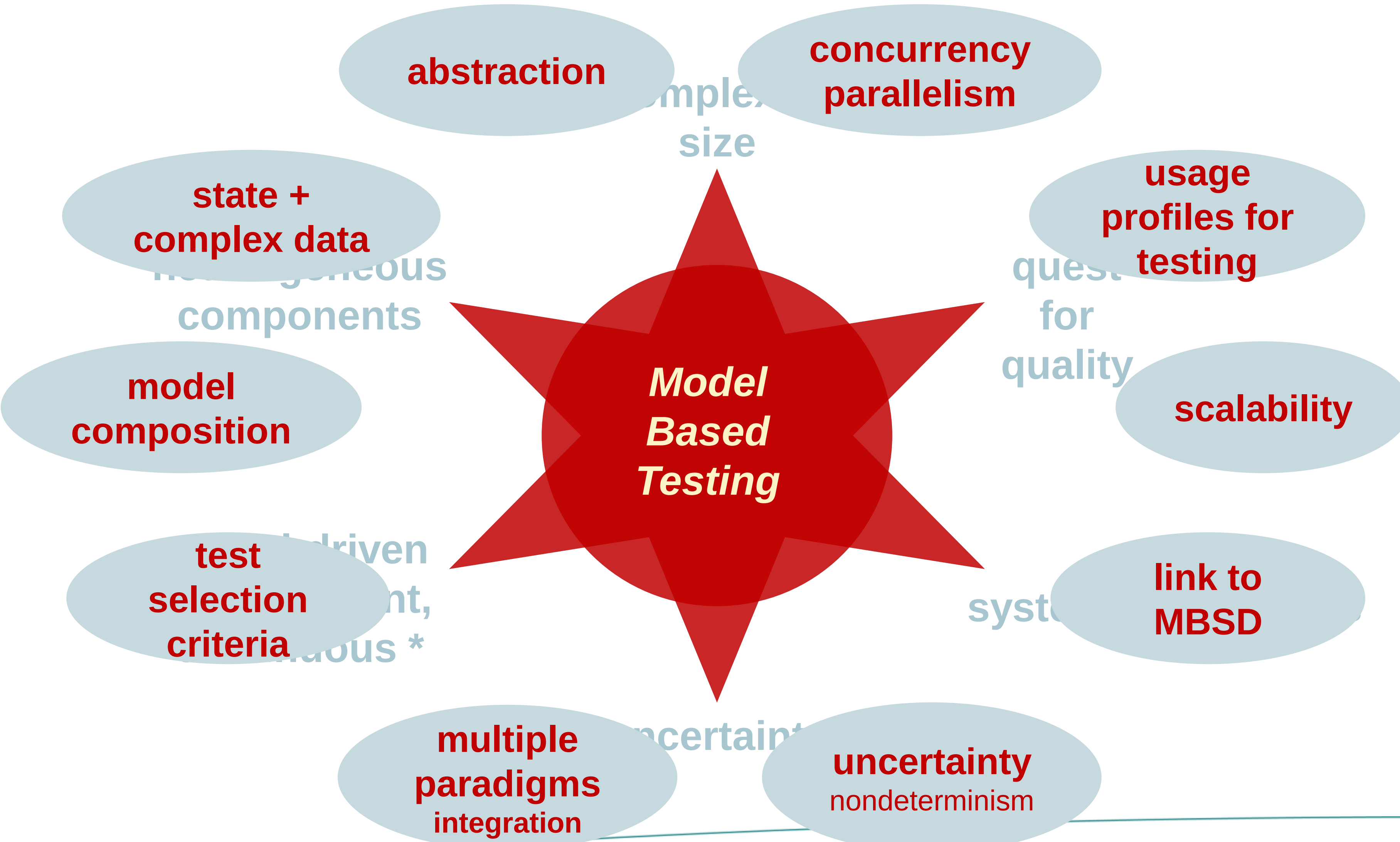


model-driven
development,
continuous *

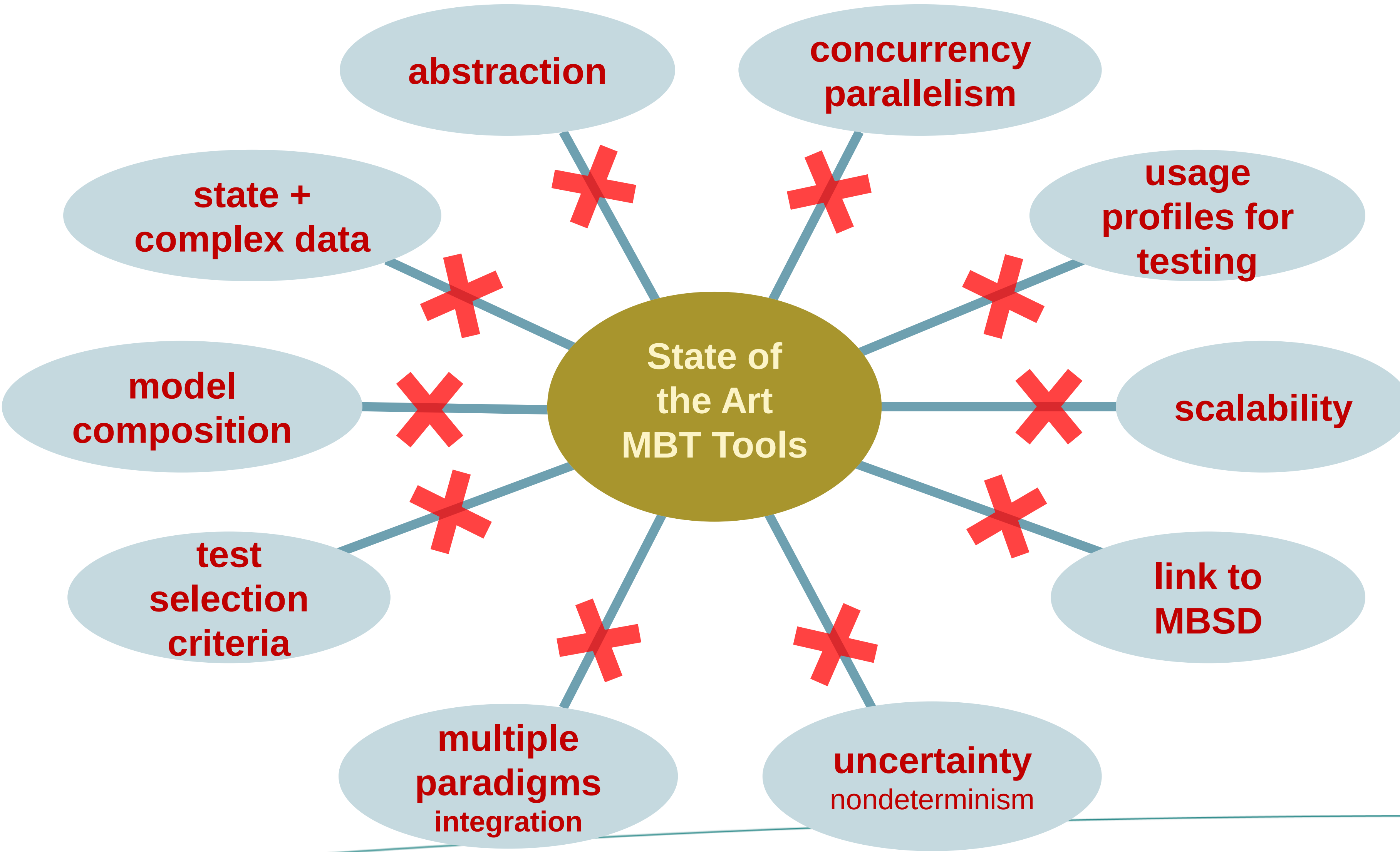
systems-of-systems

uncertainty

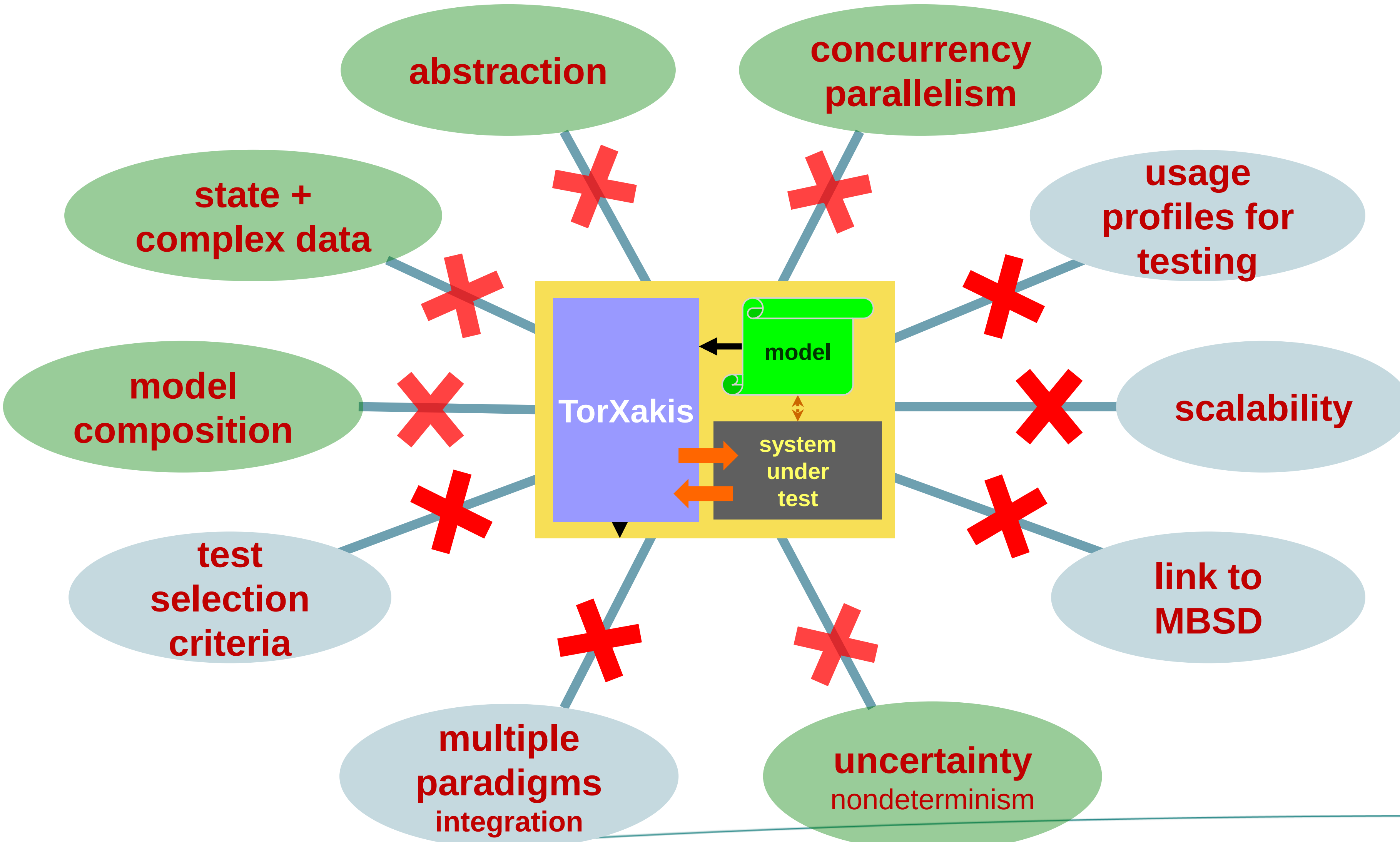
MBT : Next Generation Challenges



MBT : Next Generation Challenges



Next Generation MBT : TorXakis



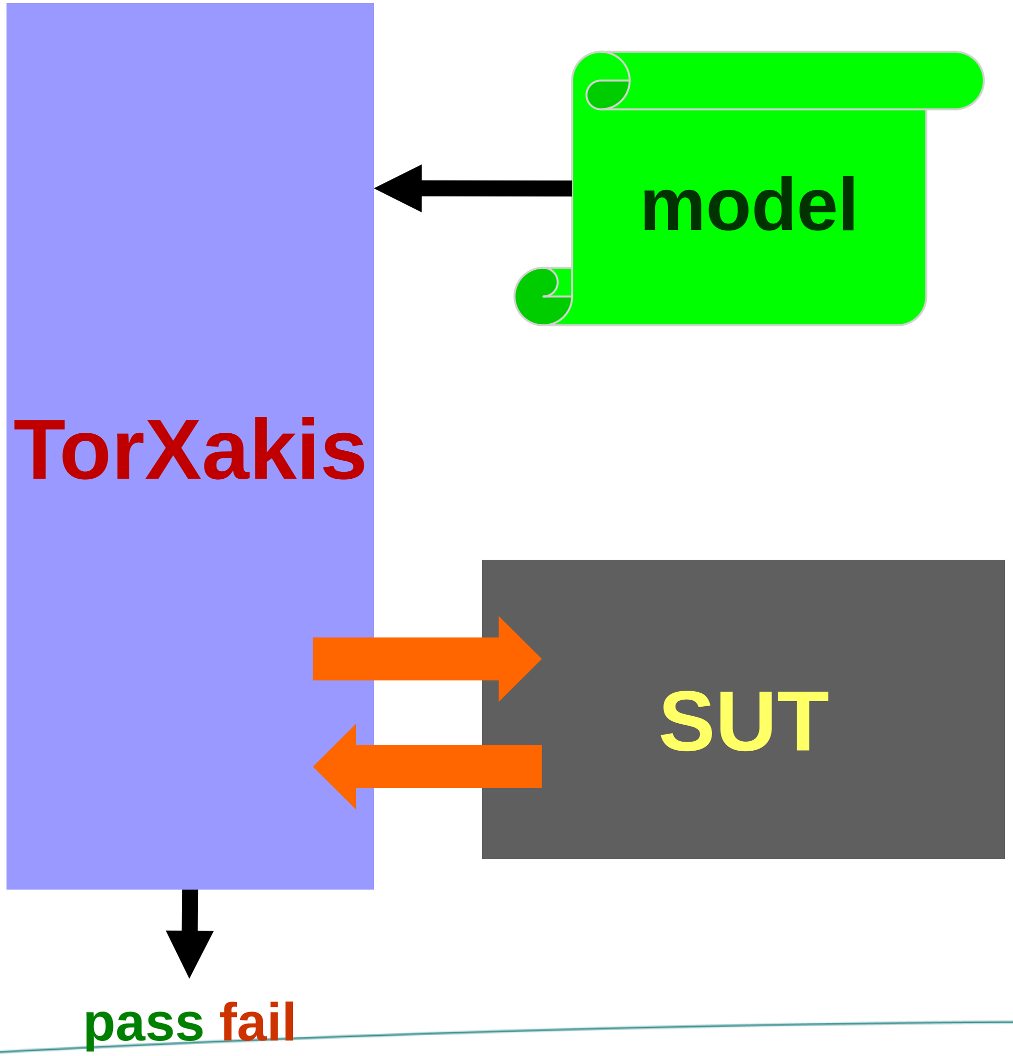
TorXakis

1. My First TorXakis Model
   - SUT
   - Model
   - Adapter

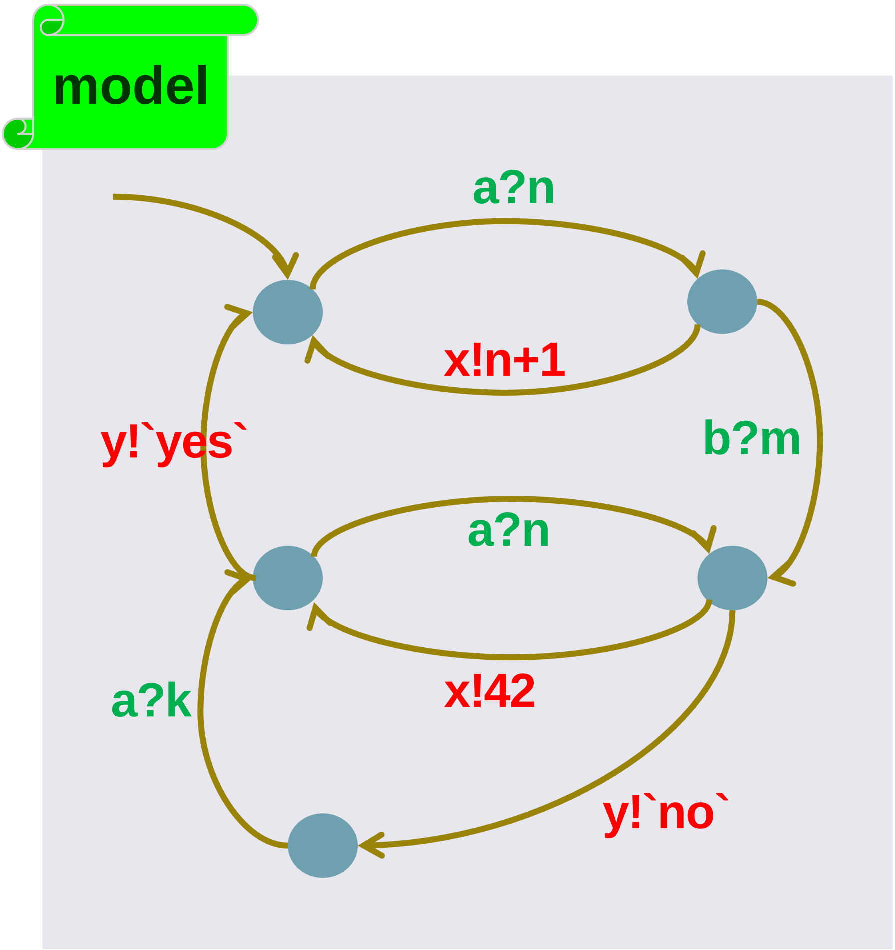
2. My First TorXakis Test Run

3. More TorXakis Models

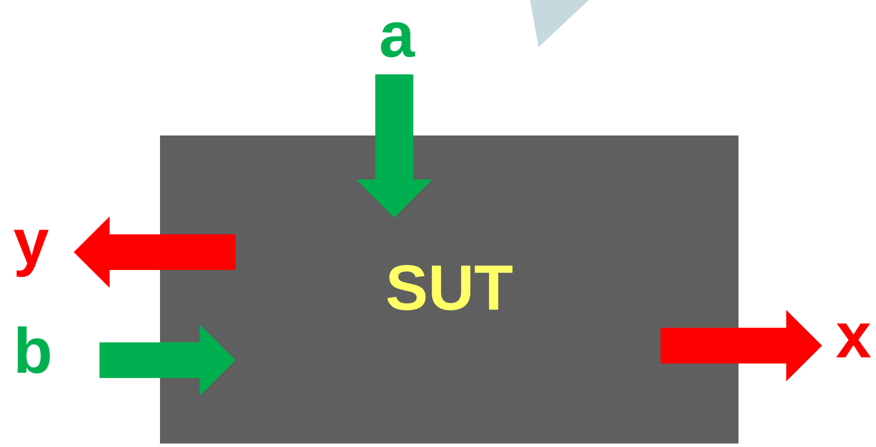
TorXakis : An On-Line MBT Tool



TorXakis : A Black-Box View on Systems

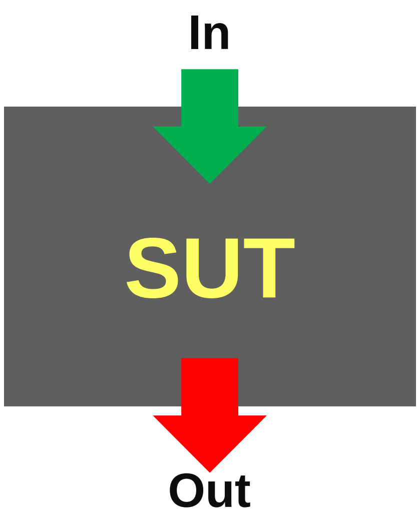


black-box
system view



modelled as
state-transition
system

TorXakis : Definition of SUT



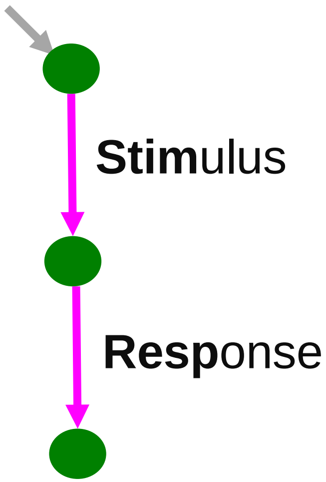
SUT

real, black-box system communicating with its environment via messages on input- and output channels

```
SUTDEF  Sut
   ::=      SUT IN       In    :: String
            SUT OUT      Out :: String

            SOCK IN       In     HOST "localhost"  PORT 7890
            SOCK OUT      Out  HOST "localhost"  PORT 7890
ENDDEF
```

TorXakis : Definition of Model



MODEL

labelled transition system

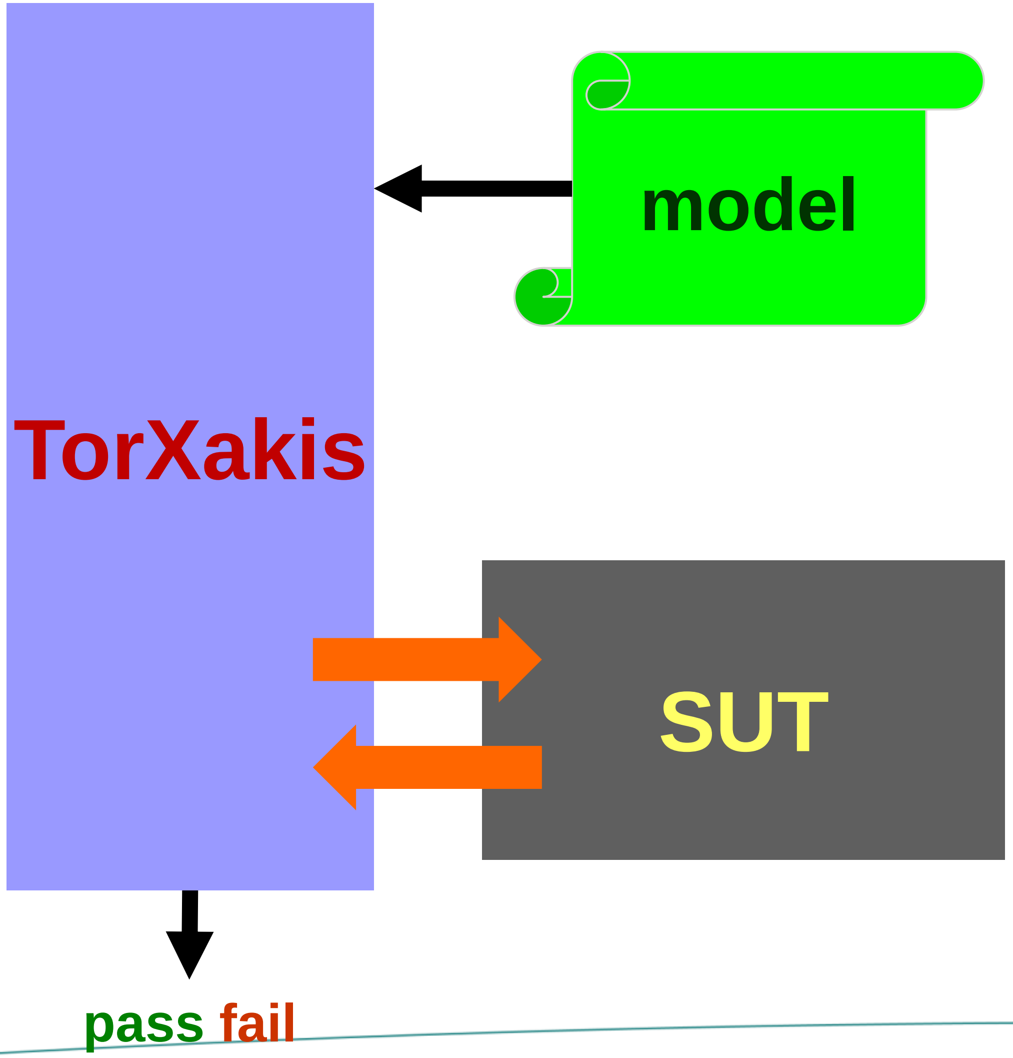
with actions on input-

and output channels

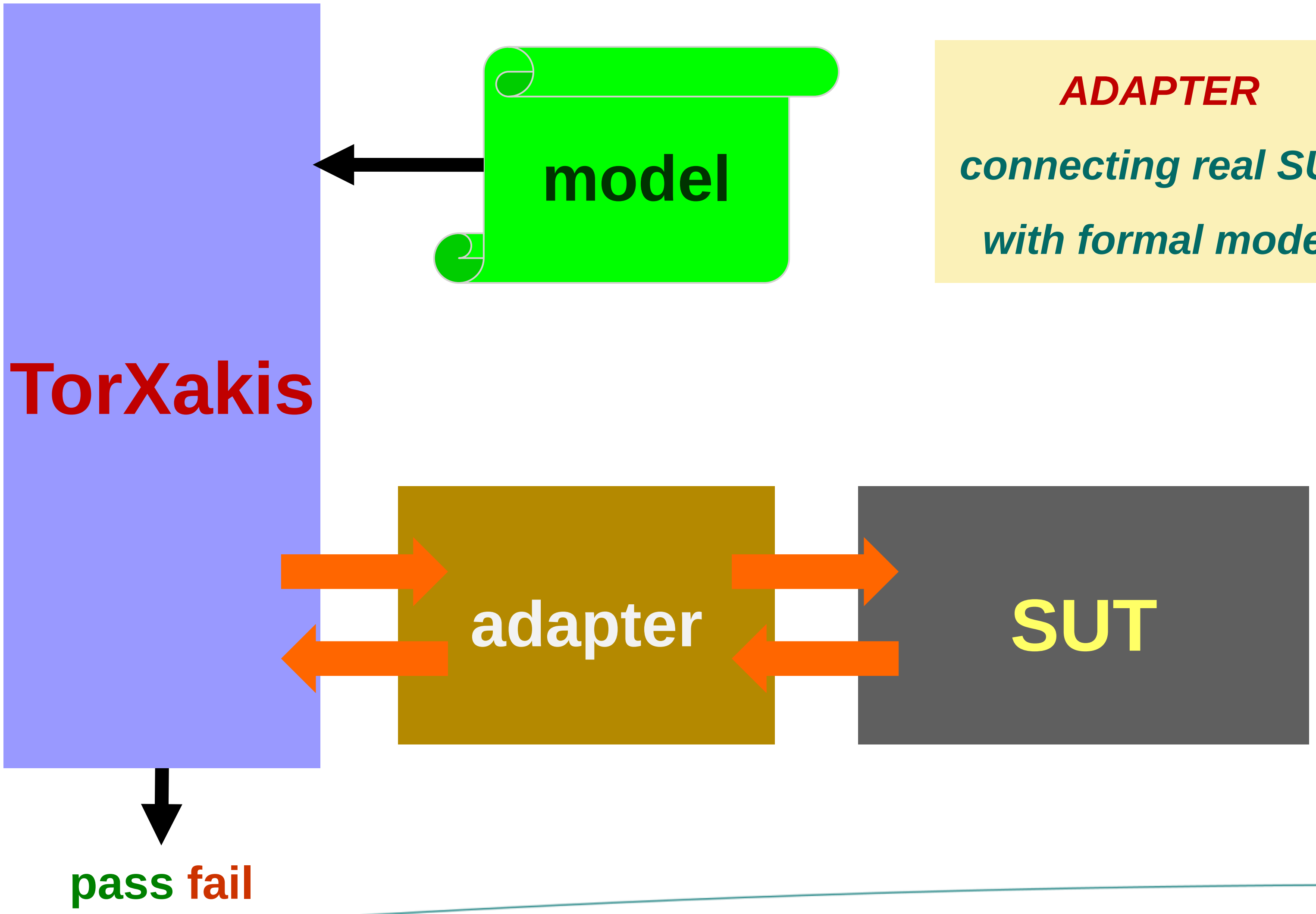
```
SPECDEF  Spec
   ::=
              CHAN IN            Stim
              CHAN OUT           Resp

              BEHAVIOUR          Stim   >->   Resp
ENDDEF
```

TorXakis : An On-Line MBT Tool



Adapter: Connecting TorXakis and SUT



ADAPTER

connecting real SUT

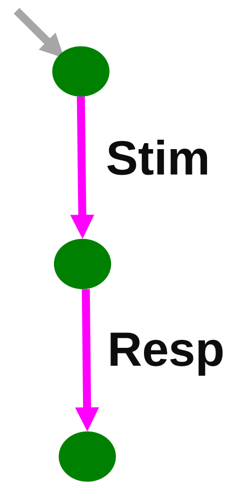
with formal model

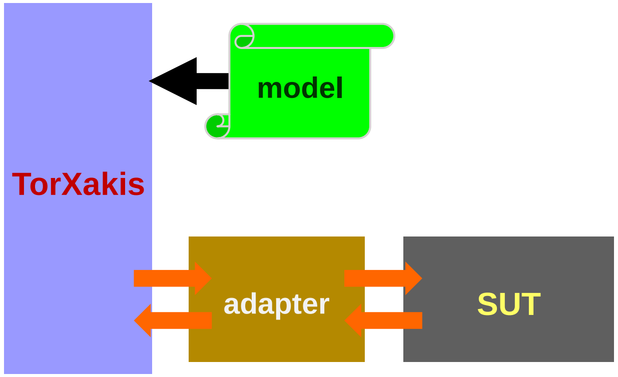
TorXakis : Definition of Adapter

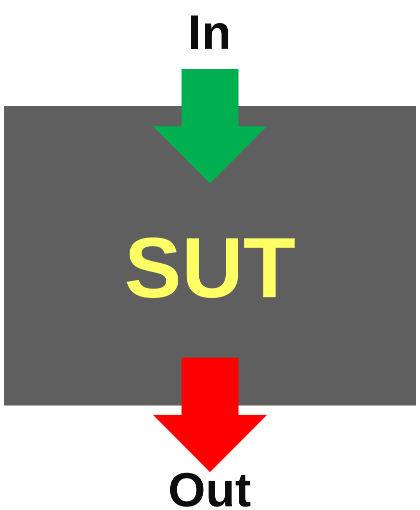
ADAPTER

connecting real SUT

with formal model








```
ADAPDEF  Adap
  ::=      CHAN IN    Stim            SUT IN     In   :: String
           CHAN OUT   Resp            SUT OUT    Out :: String

           MAP IN     Stim       ->    In ! "stim"
           MAP OUT    Out ? s    ->    Resp
ENDDEF
```

TorXakis

1. My First TorXakis Model
   - SUT
   - Model
   - Adapter

.....\examps.testnet\StimulusResponse\model\SRfinite.txs

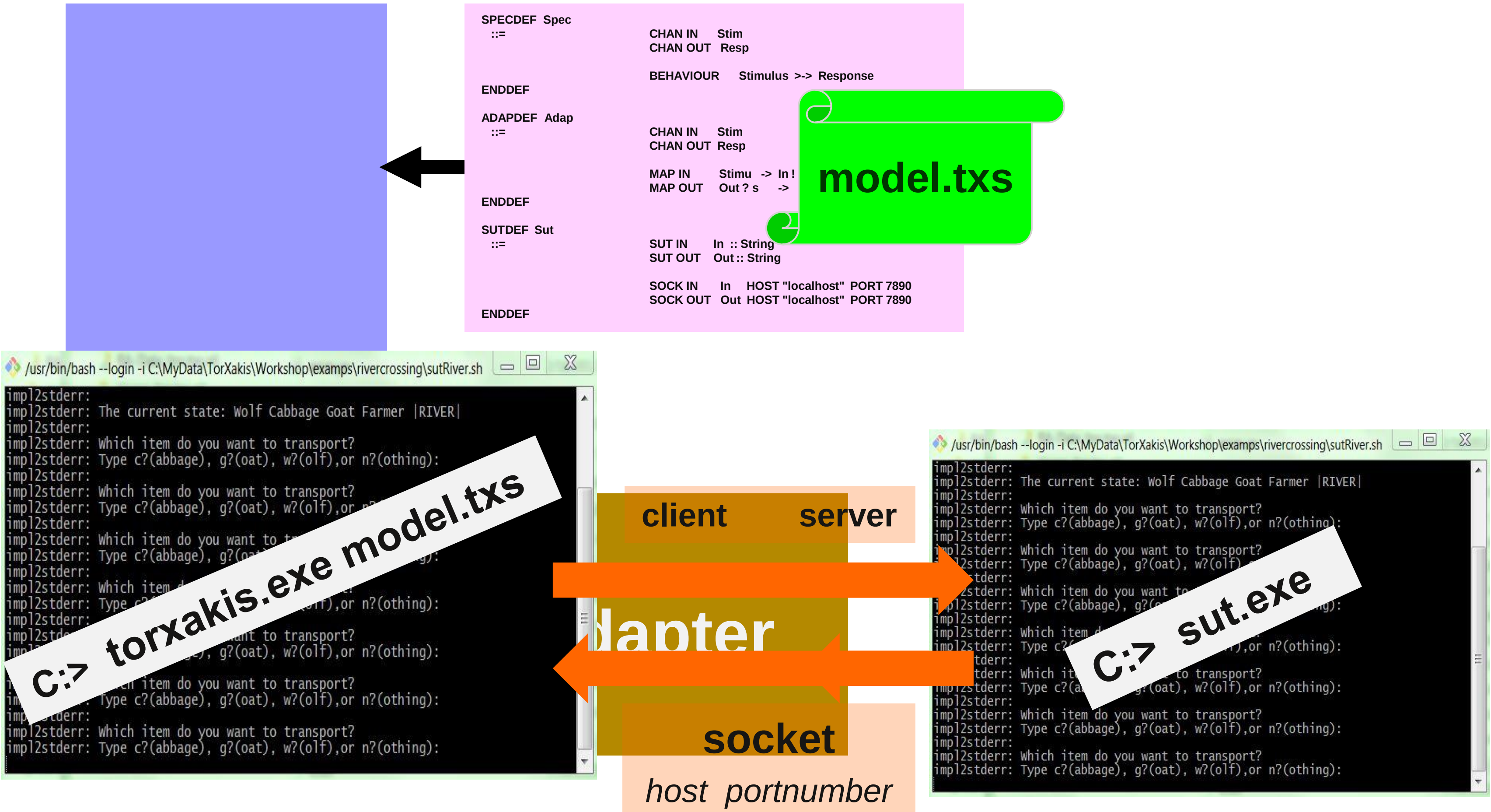
TorXakis

1. My First TorXakis Model
   - SUT
   - Model
   - Adapter

2. **My First TorXakis Test Run**

3. More TorXakis Models

Running TorXakis and SUT



TorXakis : Running a Test (Windows)

1. Start two Command Prompt windows, one for TorXakis, one for the SUT
2. There are two versions of SUTs: pre-compiled executables and Java sources
3. For pre-compiled, go to ...\examps.testnet\StimulusResponse\winexe
   and run the SUT: C:> SRfinite.exe <port nr>
4. For Java – JDK required* - go to: ...\examps.testnet\StimulusResponse\java
   and compile and run the SUT C:> javac SRfinite.java
   C:> java SRfinite
5. In the TorXakis window, go to ...\examps.testnet\StimulusResponse\model
6. Start TorXakis with the model file: C:> torxakis.exe SRfinite.txs
7. Try some TorXakis commands: TXS >> help

* http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Running TorXakis and SUT

```
$ torxakis.exe StimulusResponse.txs

TXS >>   TorXakis :: Model-Based Testing

TXS >>   TorXakis :: Input File parsed ...

TXS >>   TorXakis :: SMT Solver (Z3) initialized ...

TXS >>   TorXakis :: Internal Environment initialized ...

TXS >>  help
```

Running TorXakis and SUT

```
TXS >> spec Spec
TXS >> adap Adap
TXS >> sut Sut
TXS >> test 4

1   >> Input: ActIn  { { ( Stimulus, [] ) } }

2   >> Output: ActOut { { ( Response, [] ) } }

3   >> Output: Delta

4   >> Output: Delta

PASS

TXS >>
```

TXS >>> help

- quit, q : stop TorXakis completely
- exit, x : exit command run
- help, h, ? : show help

- spec <spec-name> : (re)set the state
- adap <adap-name> : (re)set the adapter
- sut <sut-name> : (re)set the sut

TXS >>> help

- `const <const-def>              : define a constant`
- `func <func-def>                : define a function`
- `var <var-decl>                 : declare variables`
- `val <value-def>                : define values`
- `eval <value-expr>              : evaluate value-expression`
- `satsolve <value-expr> : solve value expression (open, boolean)`
- `ransolve <value-expr> : solve randomly`
- `unisolve <value-expr> : solve uniquely`

TXS >>> help

- state : show current state number
- btree [<state>] : show internal state
- goto [<state>] : goto <state> number
- back [<n>] : go back <n>/[one] steps
- init : go to initial state
- path : path from initial state
- trace : show action trace from the initial state

TXS >>> help

• menu [<state>]	: give possible actions
• step [<n>]	: make <n>/[one] random steps
• step <action>	: make a step <action>
• test <action>	: make test step identified by (visible) input <action>
• test	: make a test step by observing output
• test <n>	: make <n> random test steps

TXS >>> help

- sutin <action> : send input <action> to sut
- sutout : receive output from sut
- adapin <action> : send input <action> through adapter to sut
- adapout : receive output action from sut
- <command> '$<' <file> : read command arg from <file>
- <command> args '$>' <file> : write output of <command> to <file>
- run <file> : run script from <file>

TorXakis: Exercise

Use Client.exe and Server.exe in ...\utils to test manually via socket (with Client.exe) or simulate a SUT (Server.exe) [or use telnet or nc]

Test SUT = SRfinite.java/.exe

against

MODEL = SRfinite.txs

Test SUT = SRnone.java/.exe

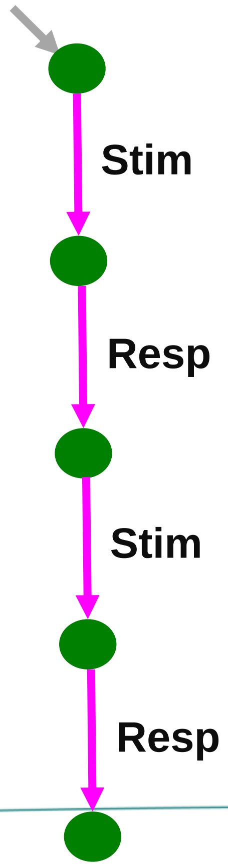
against

MODEL = SRfinite.txs

Adapt SRfinite.txs such that there are

two Stimuli and two Responses;

test with SUT = SRfinite.java/.exe



TorXakis

1. My First TorXakis Model
   - SUT
   - Model
   - Adapter

2. My First TorXakis Test Run

3. **More TorXakis Models**

Input for TorXakis

TorXakis input

= model

= list of definitions

behaviour / labelled transition system

- PROCESS — PROCDEF

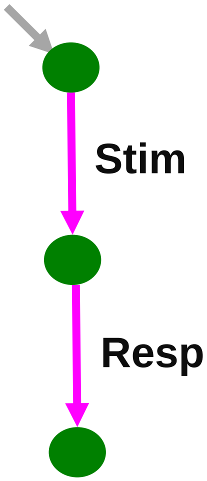
test architecture

- SPECIFICATION — SPECDEF
- ADAPTER — ADAPDEF
- SUT — SUTDEF

data

- TYPE — TYPEDEF
- FUNCTION — FUNCDEF
- CONSTANT — CONSTDEF

TorXakis: Process Definition



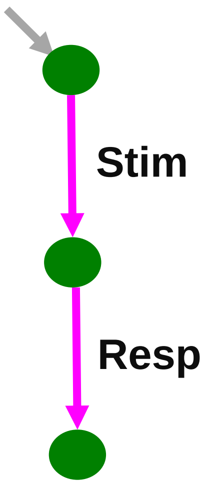

```
SPECDEF  Spec
   ::=
            CHAN IN       Stim
            CHAN OUT   Resp

            BEHAVIOUR

                              Stim   >-> Resp

ENDDEF
```

TorXakis: Process Definition




```
PROCDEF  stimresp  [ Stim, Resp ]  ( )
   ::=
             Stimulus  >-> Response
ENDDEF
```

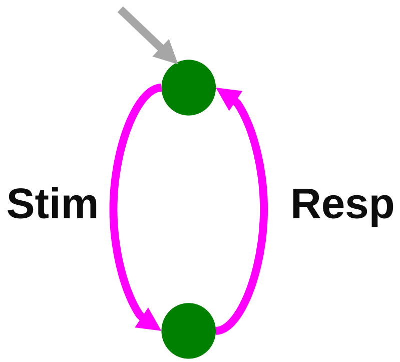
```
SPECDEF  Spec
   ::=
             CHAN IN            Stim
             CHAN OUT           Resp

             BEHAVIOUR

                      stimresp [ Stim, Resp ]  ( )

ENDDEF
```

TorXakis: Process Definition




```
{-  Cyclic Stimulus-Response  -}

PROCDEF stimuRespo [ Stim, Resp ] ( )
   ::=
                    Stim
              >-> Resp
              >-> stimuRespo [ Stim, Resp ] ( )
ENDDEF

SPECDEF Spec
   ::=        CHAN IN      Stim
              CHAN OUT   Resp

              BEHAVIOUR
                    stimuRespo [ Stim, Resp ] ( )
ENDDEF
```

TorXakis: Exercise

Try to a model for the looping StimulusResponse system

(or look at SRloop.txs)

Test SUT = SRloop.java/.exe

against your looping StimulusResponsemodel (SRloop.txs)

Test the finite system SUT = SRfinite.java/.exe

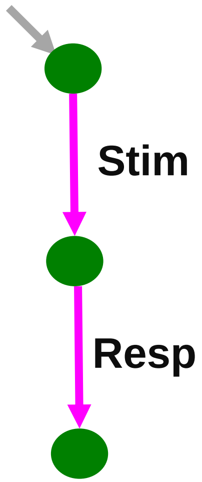
against your looping StimulusResponsemodel.

Repeat for the looping SUT = SRloop.java/.exe

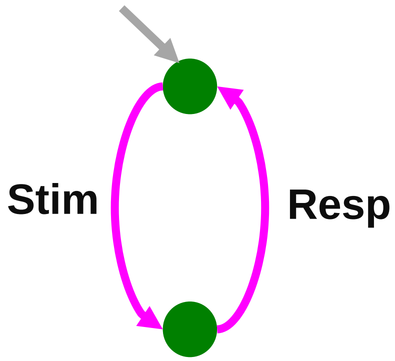
against the old model = SRfinite.txs.

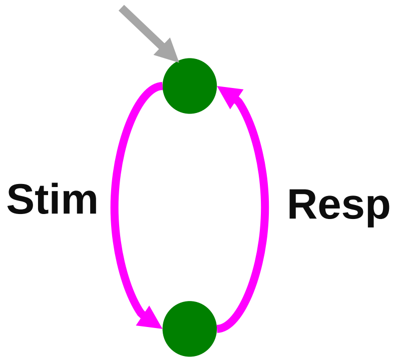
Explain the results.

TorXakis: Exercise Result

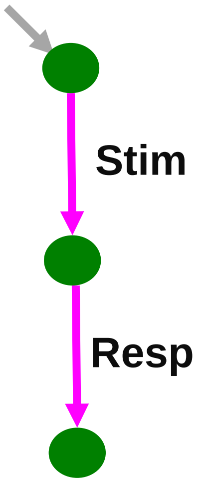


~~implements~~





implements



Input for TorXakis

TorXakis input

= model

= list of definitions

behaviour / labelled transition system

- PROCESS — PROCDEF

test architecture

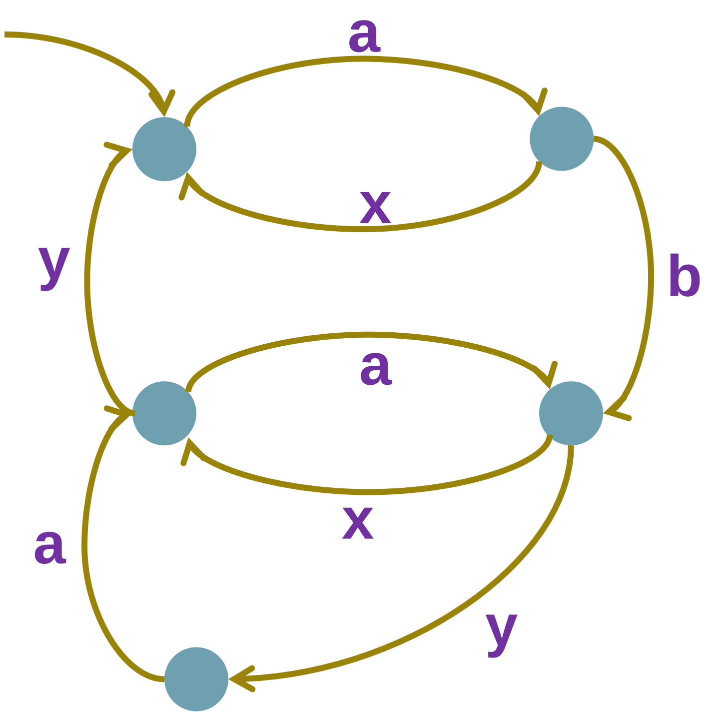
- SPECIFICATION — SPECDEF
- ADAPTER — ADAPDEF
- SUT — SUTDEF

data

- TYPE — TYPEDEF
- FUNCTION — FUNCDEF
- CONSTANT — CONSTDEF

TorXakis : Defining Behaviour

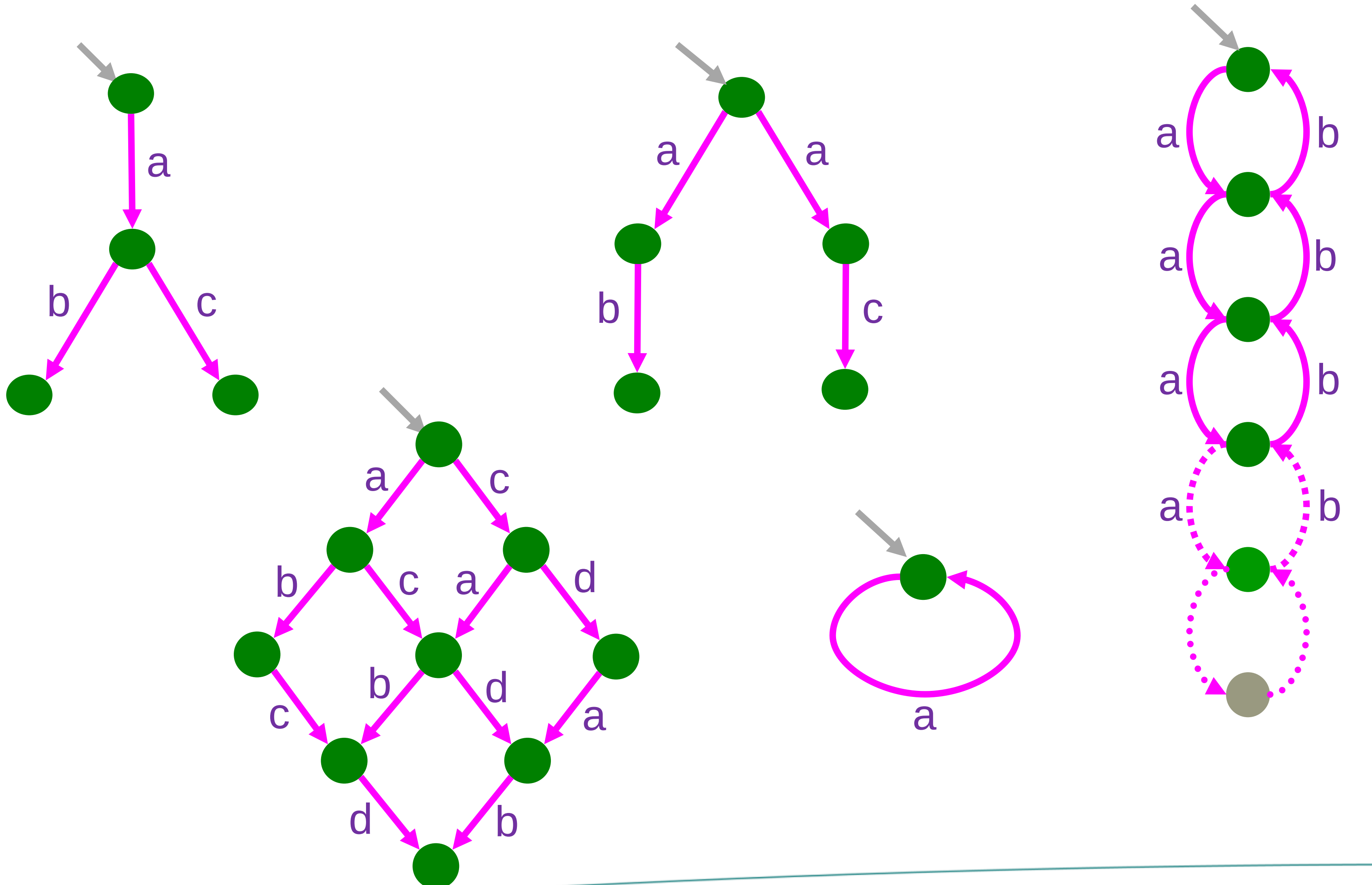
basic behaviour
= transition system



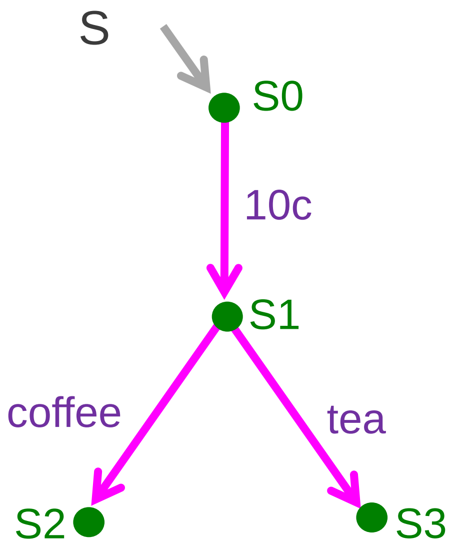
complex behaviour
= combining transition systems

- named behaviour definition
- named behaviour use
- sequence
- choice
- parallel
- communication
- exception
- interrupt
- hiding

Basis : Labelled Transition Systems - LTS



Representation of LTS



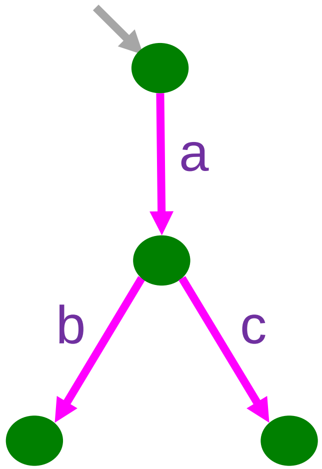
- Explicit : ⟨ { S0, S1, S2, S3 } ,
  {10c,coffee,tea} ,
  { (S0,10c,S1),
  (S1, coffee, S2),
  (S1,tea,S3) } ,
  S0 ⟩

- Transition tree / graph

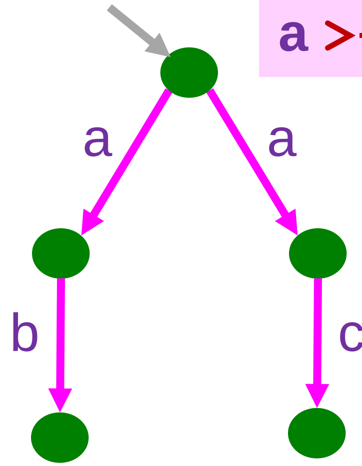
- Language :

  S ::= 10c >-> (coffee ## tea)

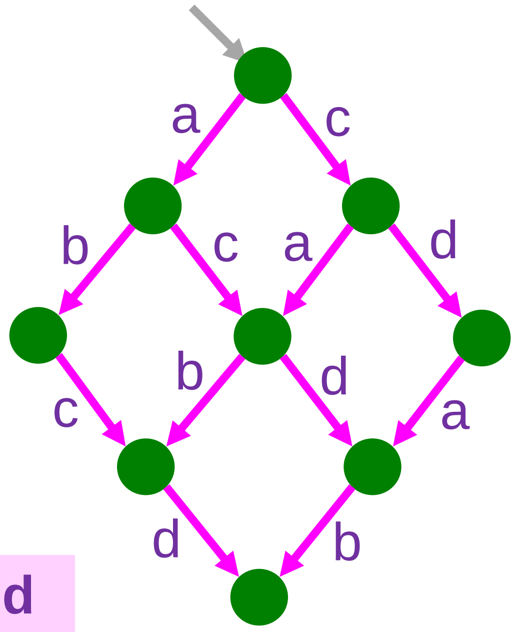
Representation of LTS



a >-> (b ## c)

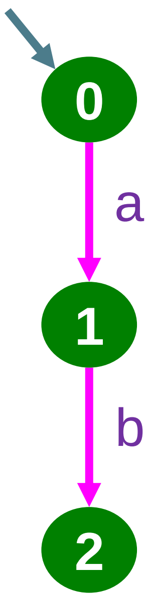


a >-> b ## a >-> c

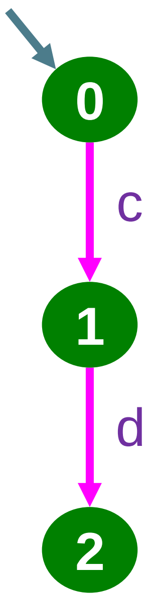


a >-> b ||| c >-> d

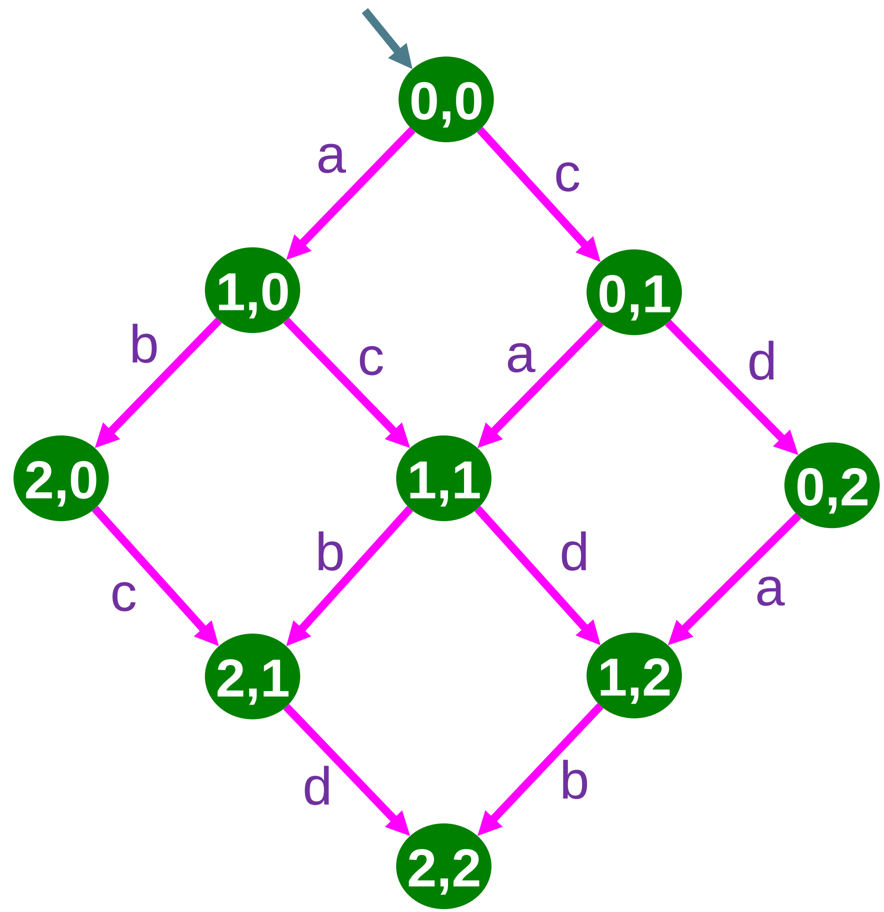
Representation of LTS



a >-> b

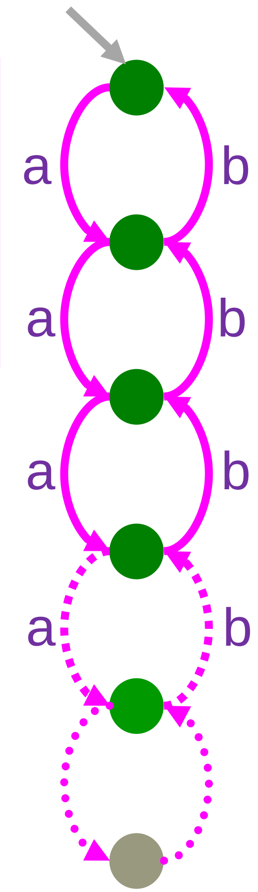


c >-> d



a >-> b ||| c >-> d

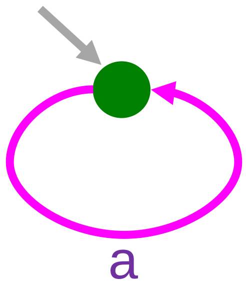
Representation of LTS



Q

where

Q ::= a >-> (b ||| Q)

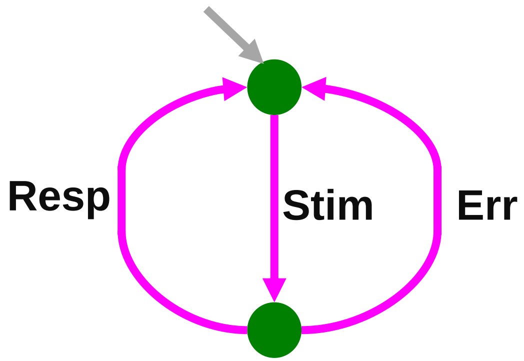


P

where

P ::= a >-> P

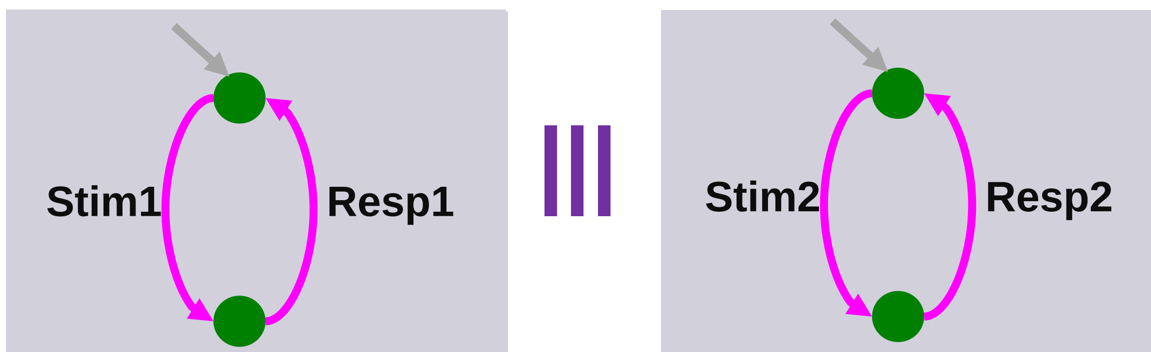
TorXakis: Choice

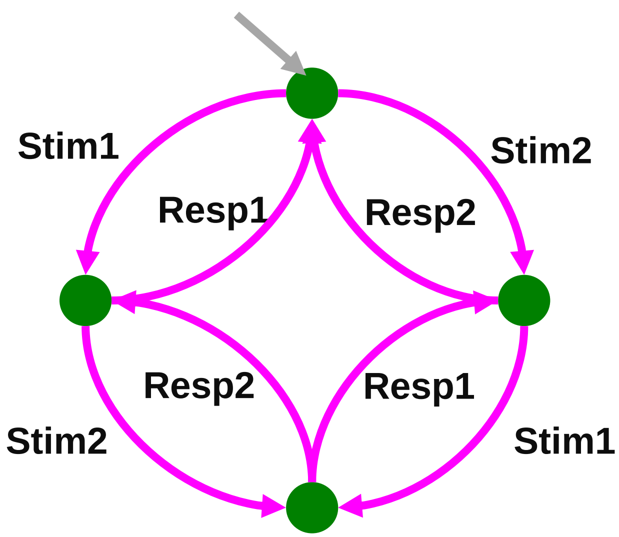



```
--  Stimulus-Response with Error

PROCDEF  errSR  [ Stim, Resp, Err ]  ()
  ::=
       Stim   >->
            (    Resp  >->    errSR [Stim,Resp,Err] ()
              ##
                Err      >->    errSR [Stim,Resp,Err] ()
            )
ENDDEF
```

TorXakis: Parallel Interleaving

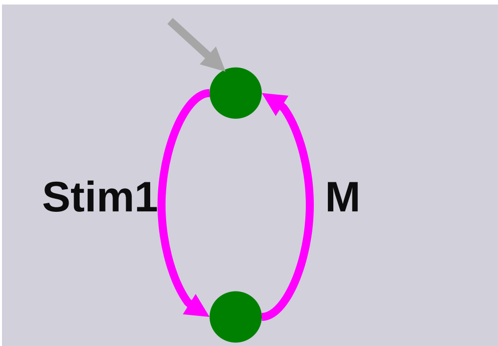




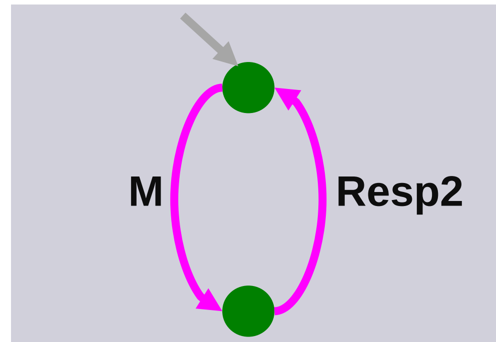
-- Parallelism with interleaving:

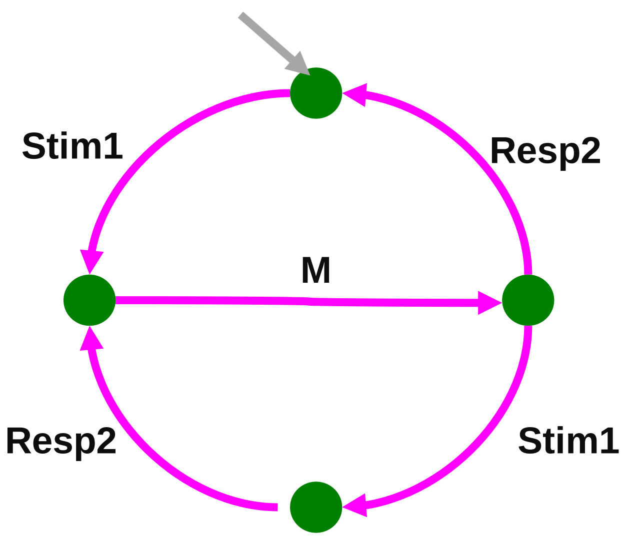
StimResp1 ||| StimResp2

TorXakis: Parallel Communication



|[M]|

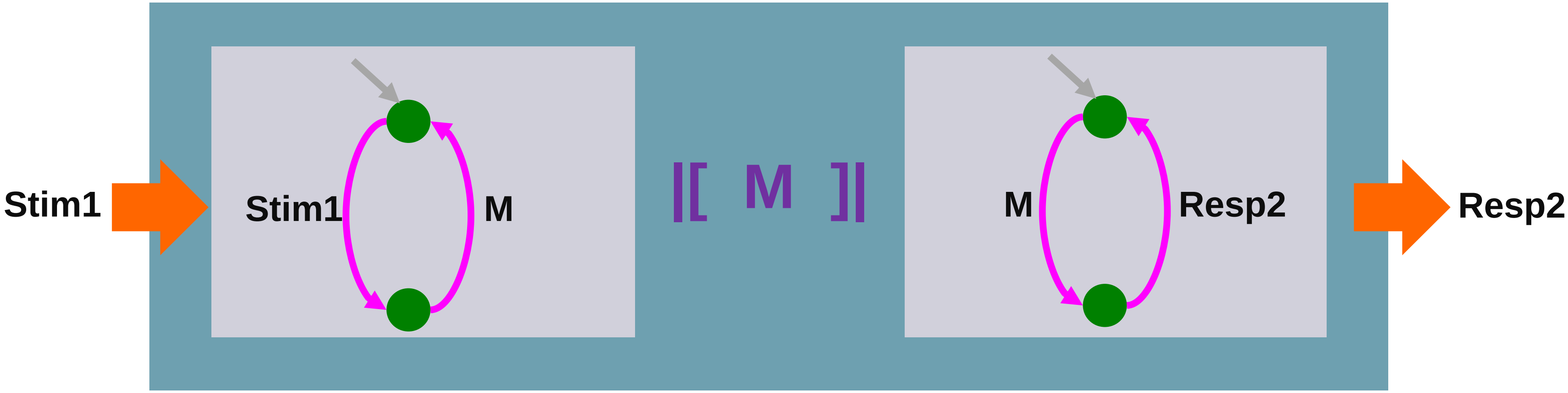


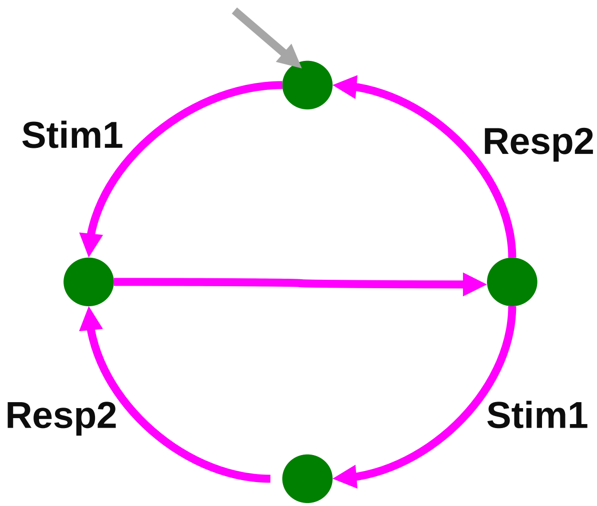


-- Parallelism with communication:

StimResp1 |[M]| StimResp2

TorXakis: Communication + Hiding






```
--  Communication + Hiding:

HIDE [ M ]
IN
      StimResp1  |[ M ]|   StimResp2
NI
```

TorXakis: Exercise

Experiment with, using testing or stepping, with

SRfinite.txs, SRlnone.txs, SRloop,txs,

SRnone, SRparallel.txs and corresponding SUTs.

If you have JDK installed you can make *mutants*,

i.e., small modifications/errors in the Java SUTs,

and see whether you can detect the errors.

Input for TorXakis

TorXakis input

= model

= list of definitions

behaviour / labelled transition system

- PROCESS — PROCDEF

test architecture

- SPECIFICATION — SPECDEF
- ADAPTER — ADAPDEF
- SUT — SUTDEF

data

- TYPE — TYPEDEF
- FUNCTION — FUNCDEF
- CONSTANT — CONSTDEF

TorXakis: Data Types

- Standard types: Int, Bool, String
- Algebraic data types

```
TYPEDEF   Colour   ::=      Red  |  Yellow  |  Blue

TYPEDEF   IntList   ::=     Nil
                          | Cons  {   hd  ::  Int
                                  ,   tl   ::  IntList
                                  }
```

TorXakis: Fu

- Functions: name,
- Overloading
- Standard functions

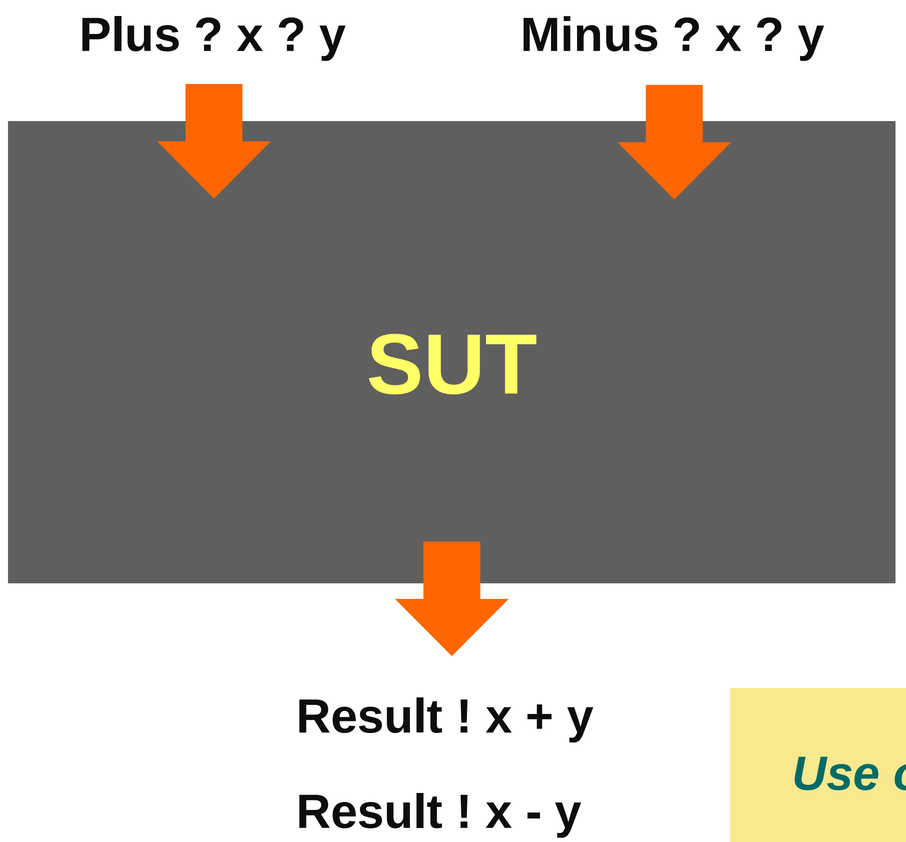
```
TYPEDEF   Colour   ::=      Red  |  Yellow  |  Blue


TYPEDEF   IntList   ::=      Nil
                           | Cons   {   hd  ::  Int
                                    ,   tl   ::  IntList
                                    }
```

```
FUNCDEF   add  ( x, y  ::  Int )  ::  Int    ::=     x + y


FUNCDEF   add  ( x :: Int;  lst :: IntList  )  ::  IntList
    ::=
          IF    isNil ( lst )
          THEN   Cons ( x, Nil )
          ELSE   Cons ( hd ( lst ),  add ( x, tl ( lst ) ) )
          FI
```

TorXakis: Adder



Use of Data in processes

TorXakis: Exercise

Test with the Adder model in SRadder.txs and corresponding SUTs.

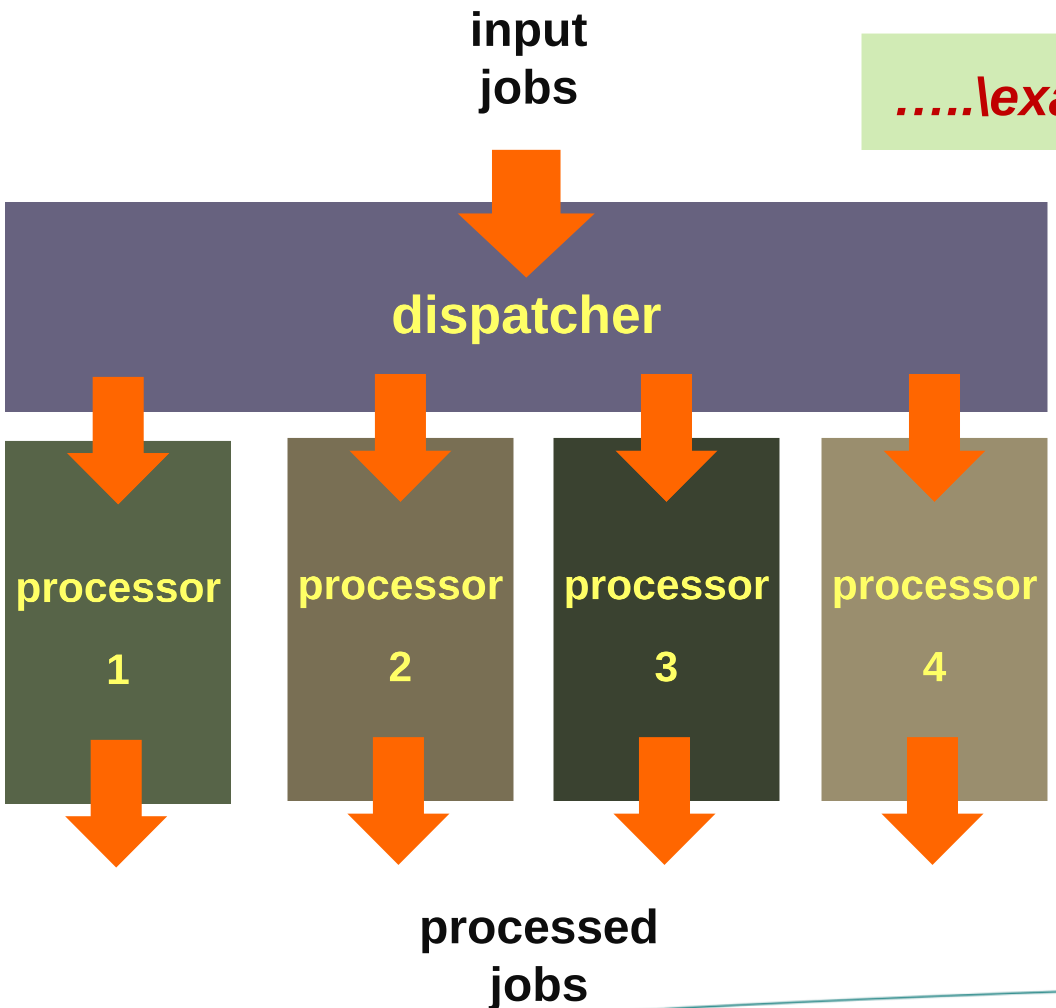
Make a mutant of SRadder.java.

Investigate the possibilities with Strings and Regular Expressions

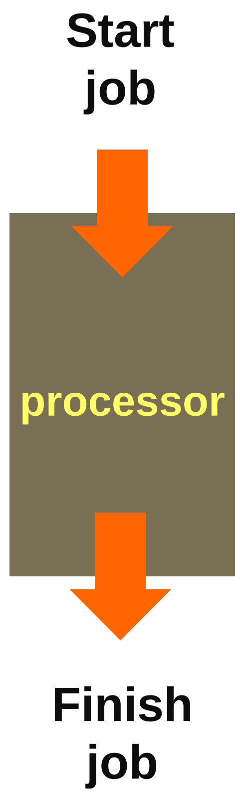
using the model SRrr.txs and corresponding SUTs.

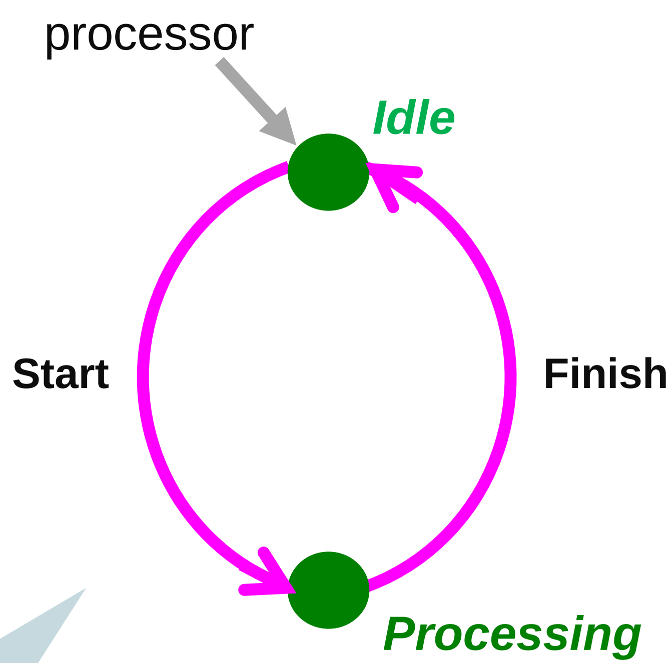
A More Elaborate Example:
The Dispatcher-Processing System

.....\examps.testnet\DispatchProcs



Example: Dispatcher-Processing System

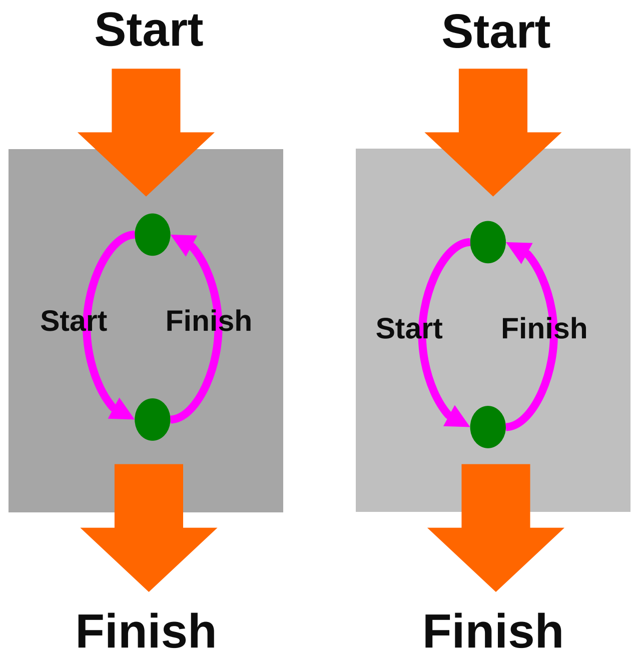




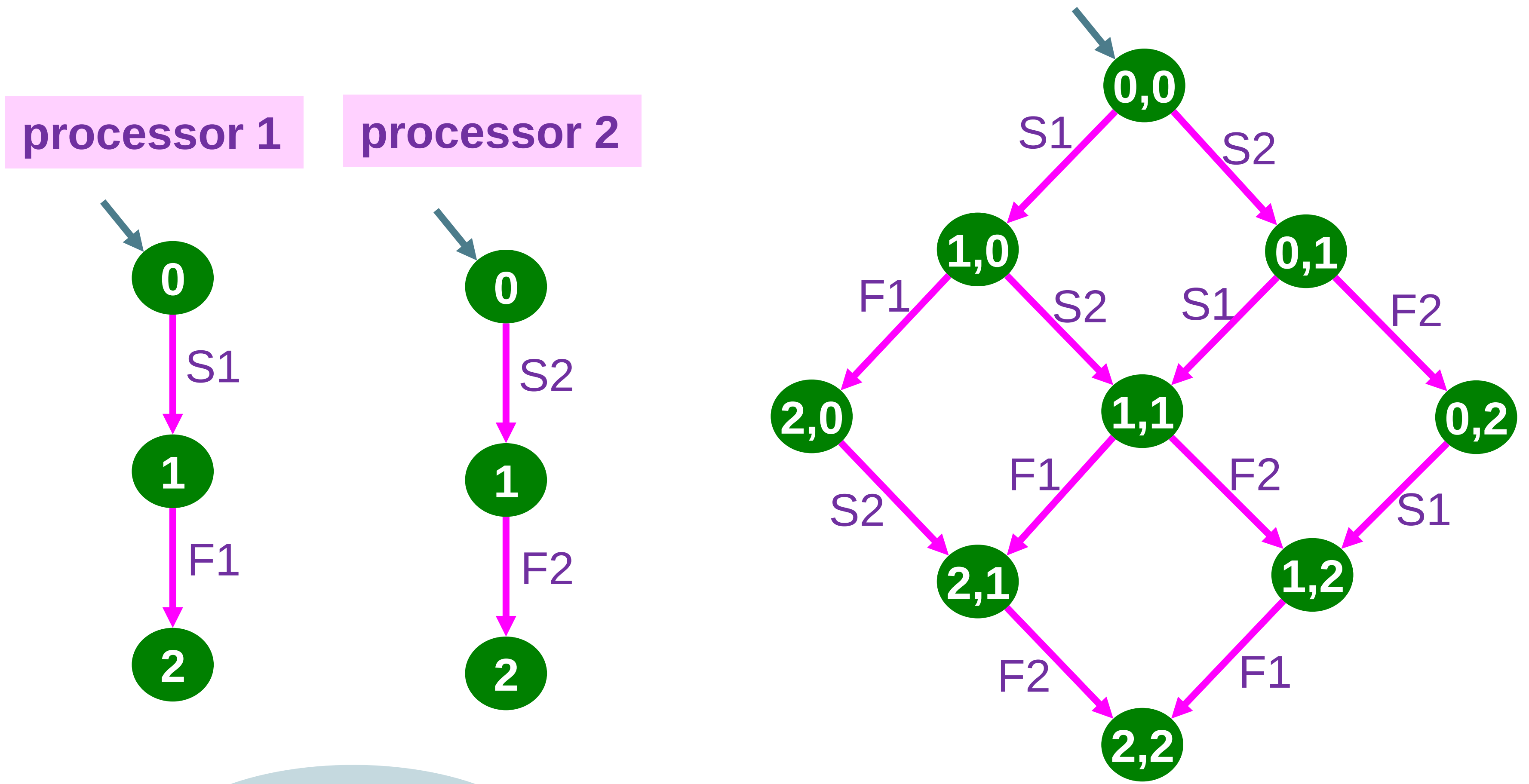
state transition system

DisPro01-proc.txs

Example: Two Parallel Processors



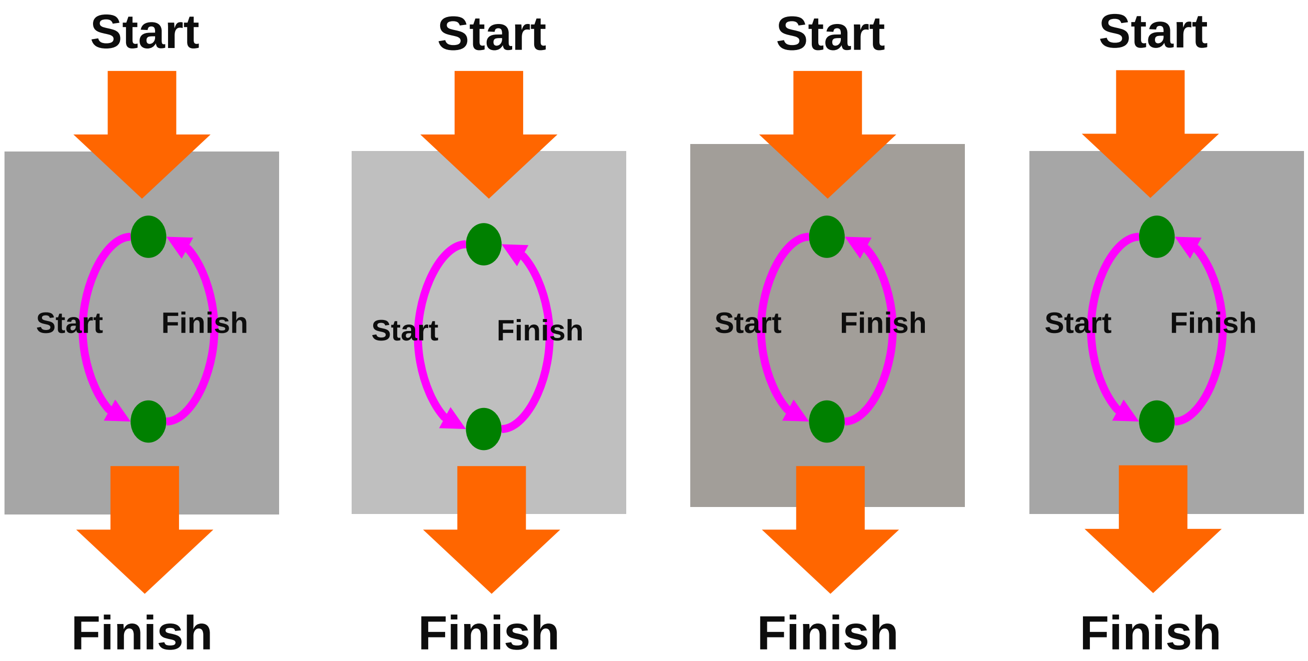
Example: Two Parallel Processors



parallelism

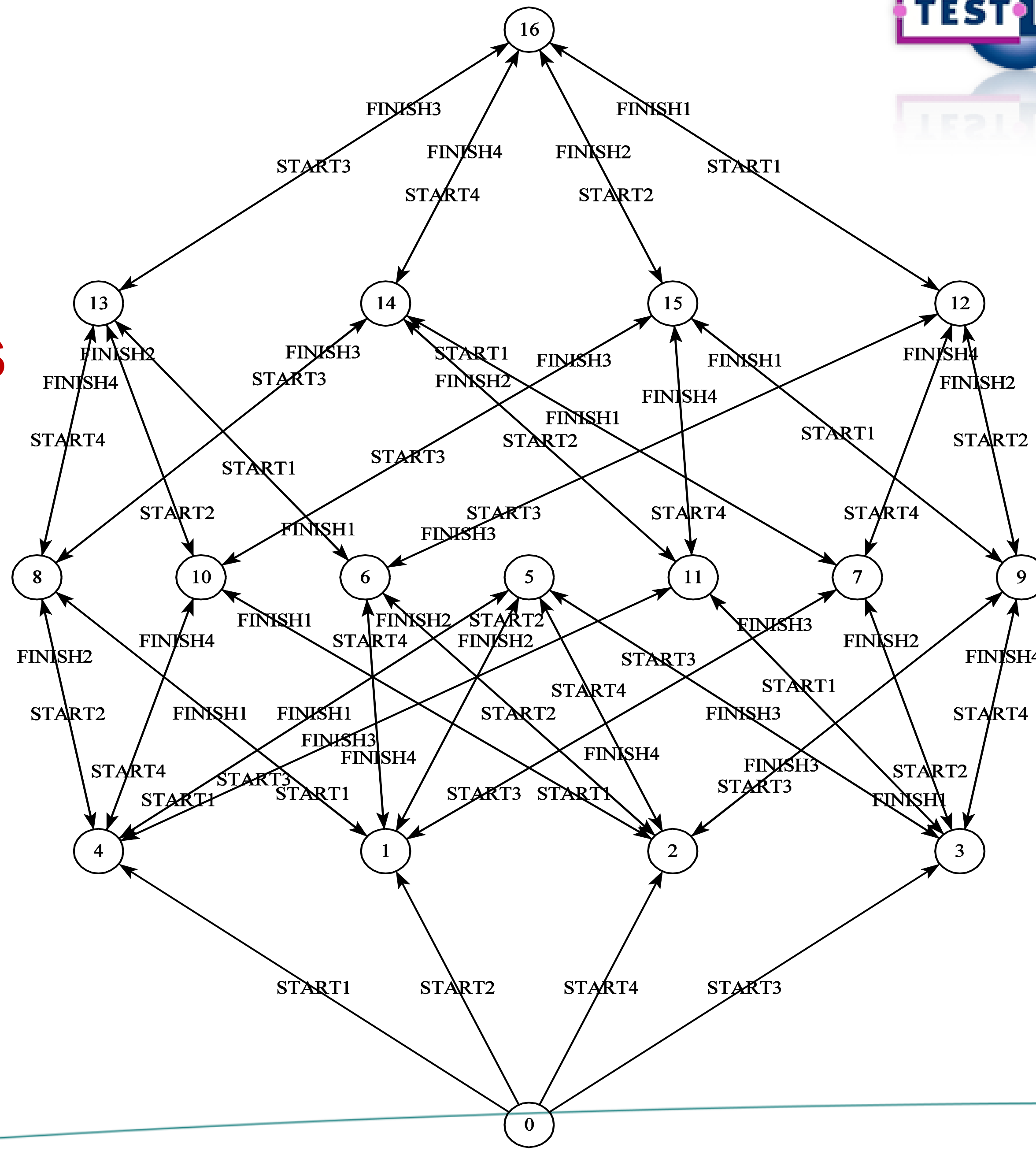
processor 1 ||| processor 2

Example: Four Parallel Processors



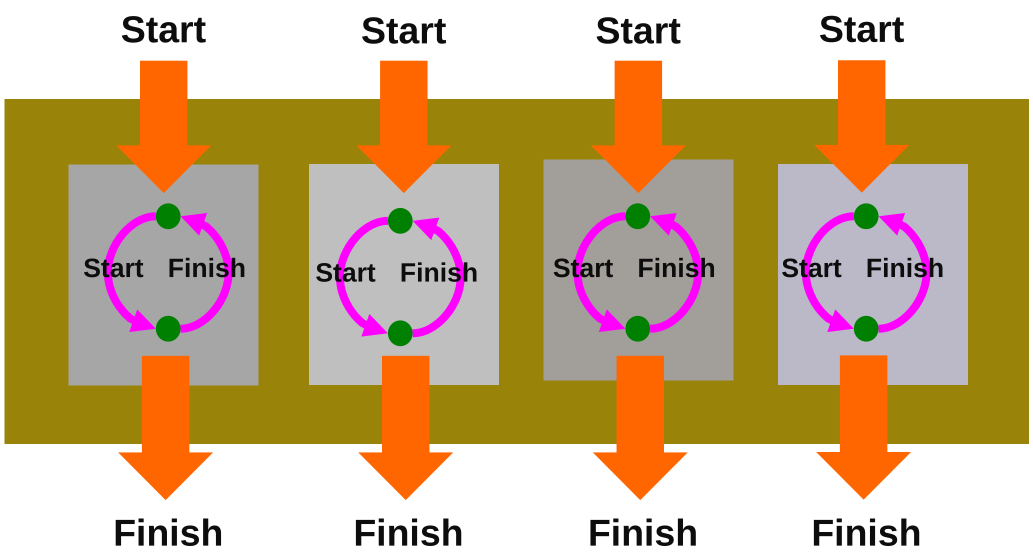
DisPro03-procs.txs

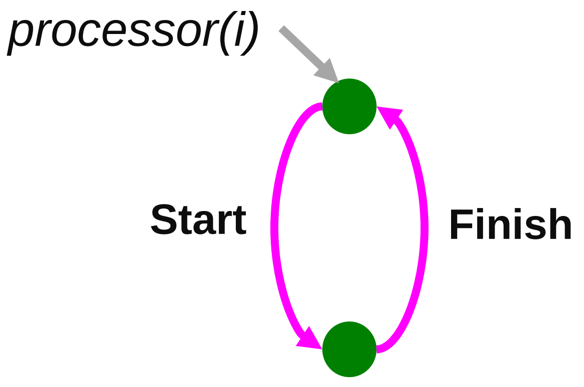
Example:
Four Parallel Processors



parallelism

Example: Dispatcher-Processing System

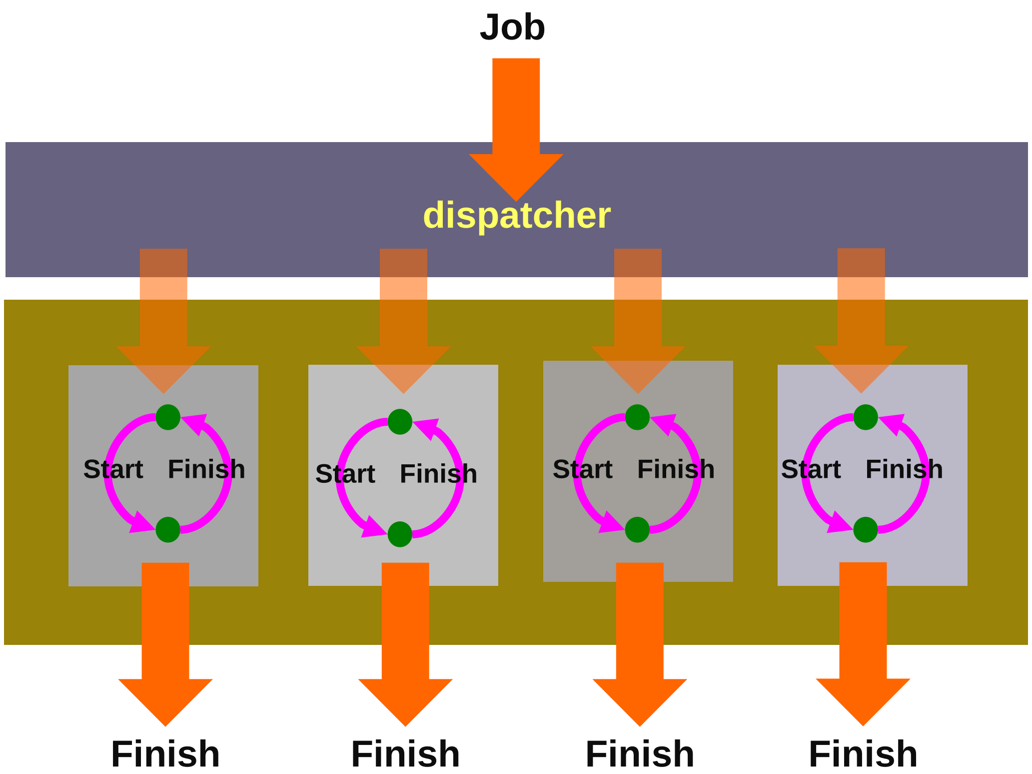




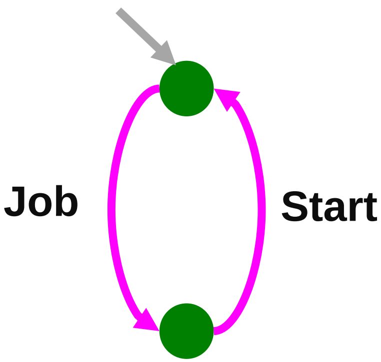
composition

processors
::=
processor(1) ||| processor(2) ||| processor(3) ||| processor(4)

Example: Dispatcher-Processing System



DisPro03-procs.txs

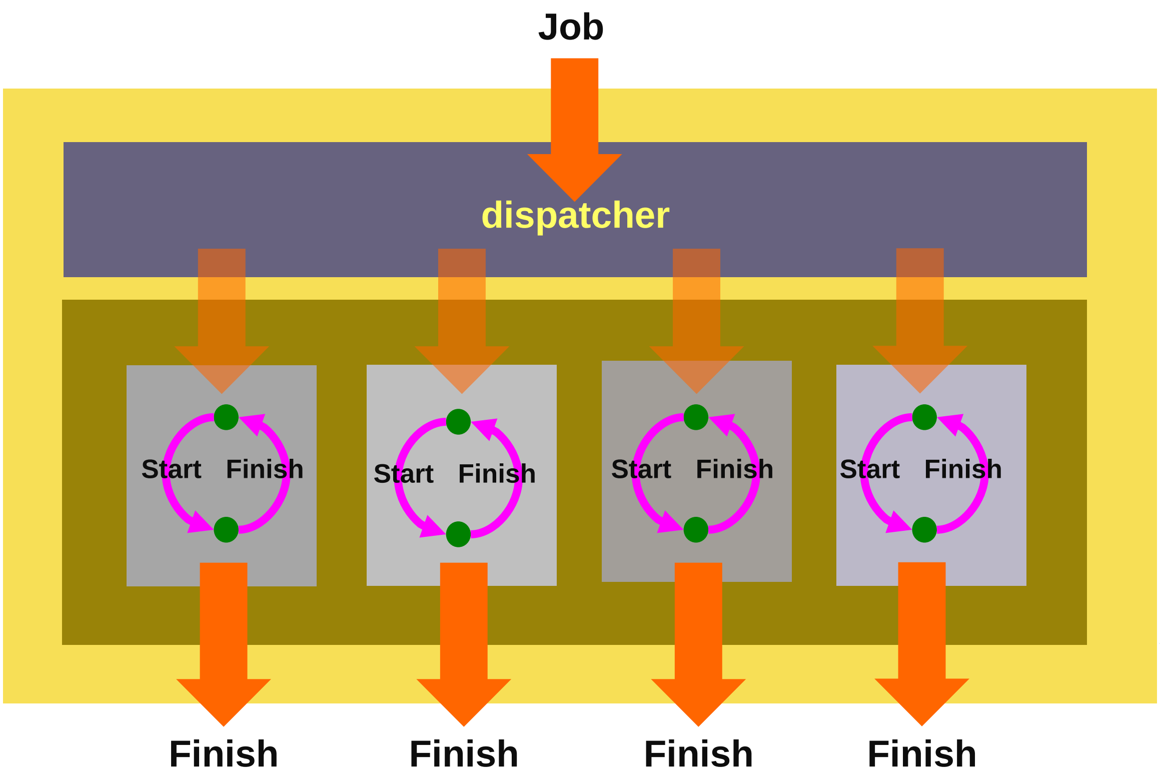


dispatcher

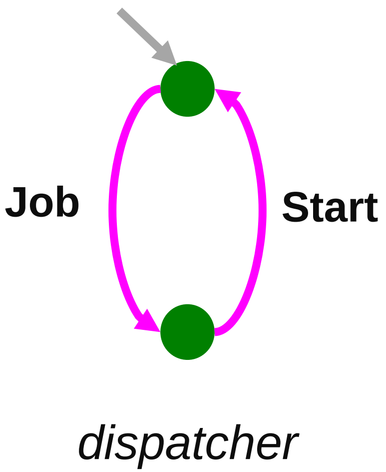
composition

dispatch_procs
::=
processors |[Start]| dispatcher

Example: Dispatcher-Processing System



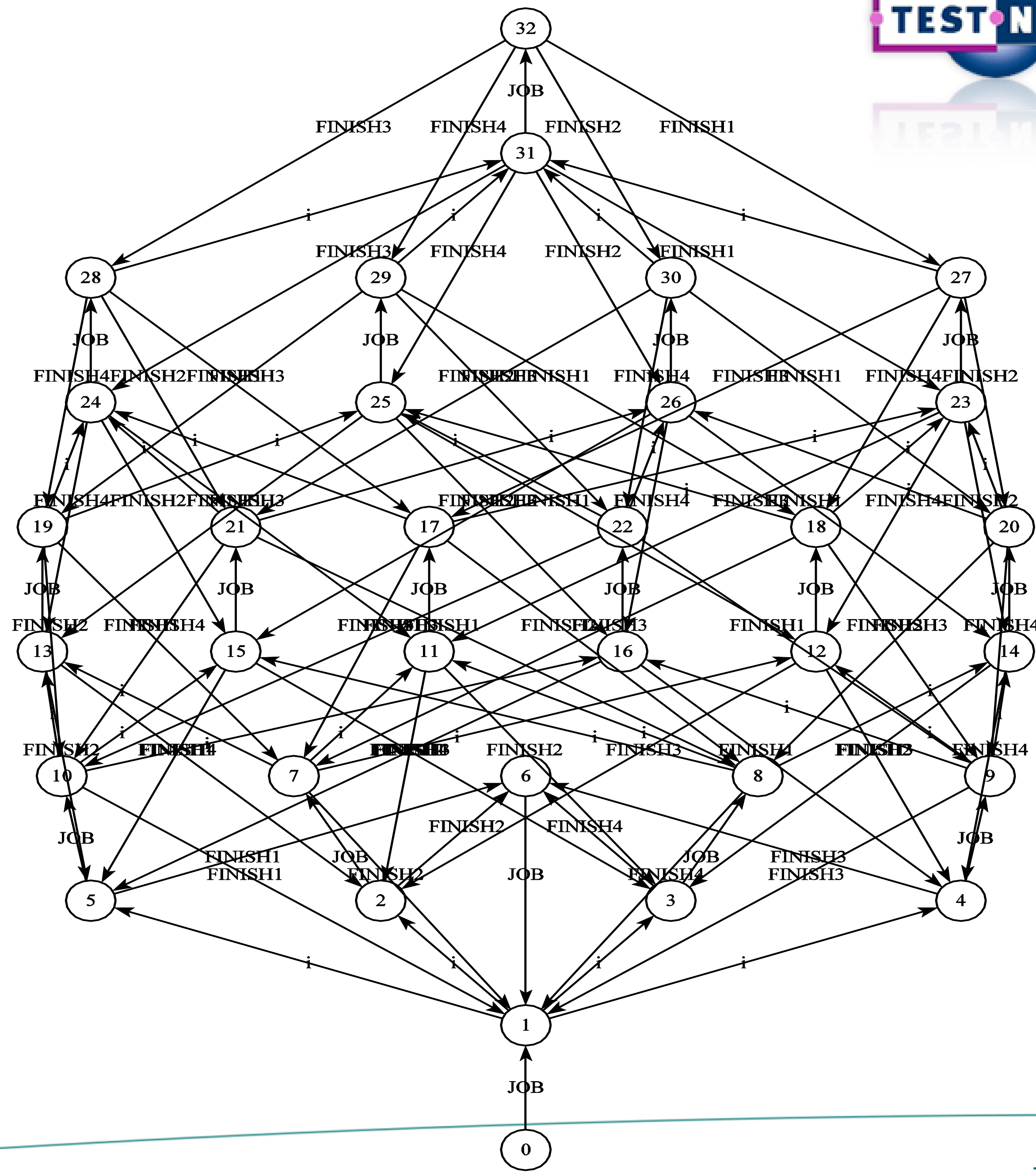
DisPro04-hide.txs



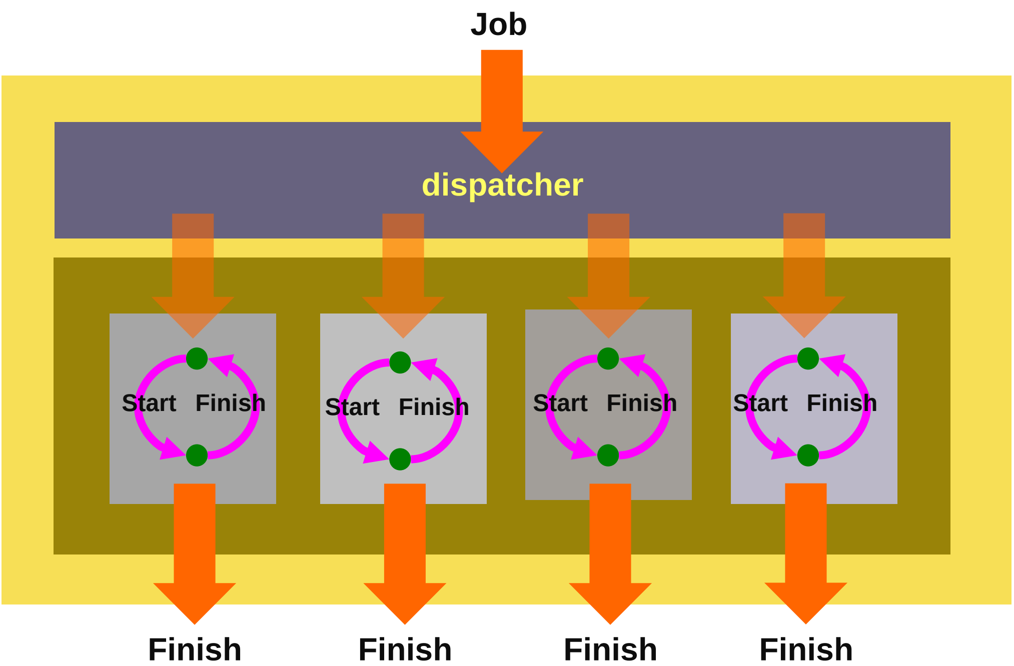

```
dispatch_procs
 ::=  HIDE  [ Start ]
      IN   processors  |[ Start ]|  dispatcher
      NI
```

abstraction

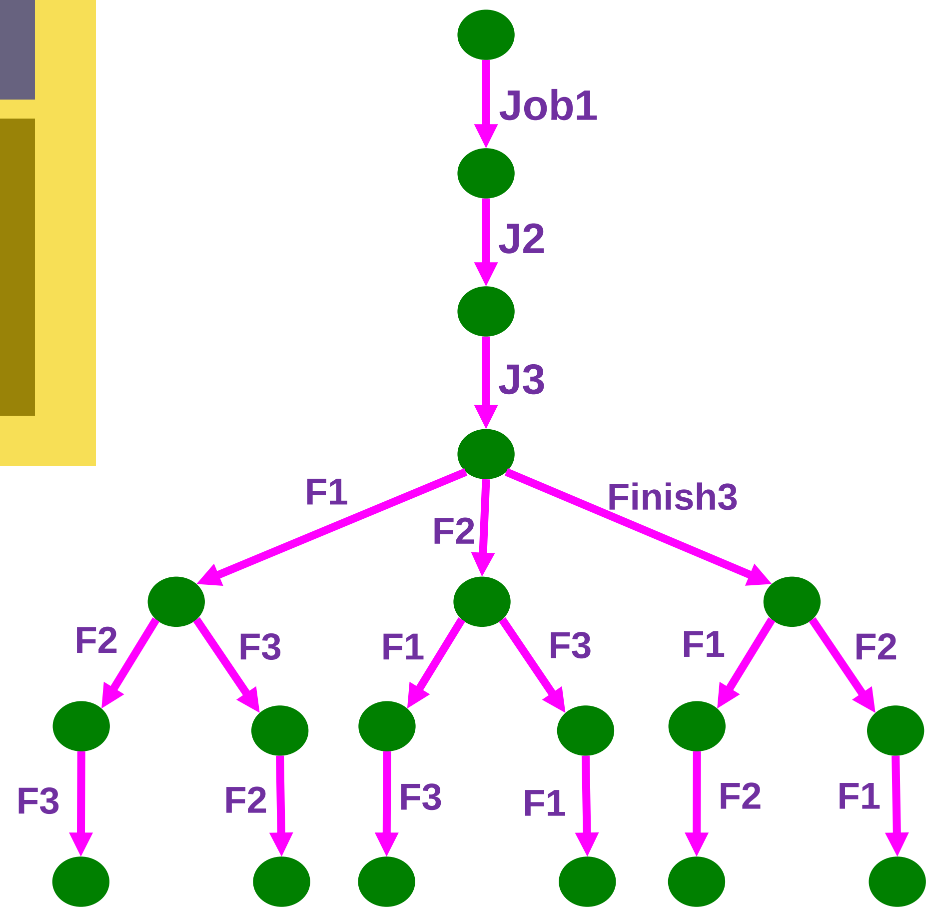
Example: Dispatcher Processing System



Example: Dispatcher-Processing System



Inputs: Job1, Job2, Job3:



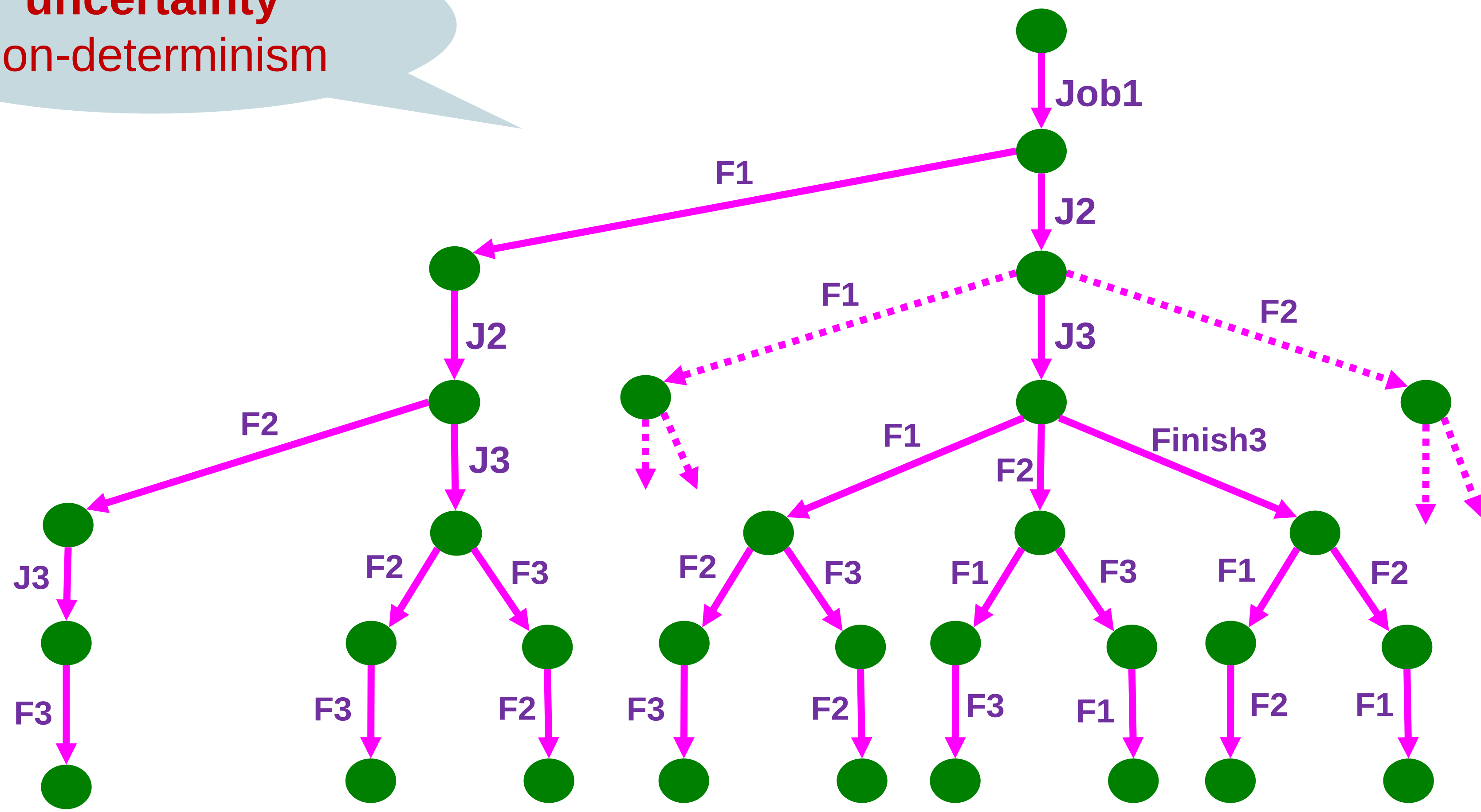
uncertainty

no unique expected result

Example: Dispatcher-Processing System

uncertainty
non-determinism

Inputs: Job1, Job2, Job3:



TorXakis: Exercise

Use TXS >> step to step through model DisPro04-hide.txs
and check possible execution sequences.

Investigate, using stepping, the use of Data and Type Definitions in

- DisPro05-adder.txs : for modelling an Adder
- DisPro06-constr.txs : for modelling input constraints and a non-deterministic result
- DisPro-07-nprocs.txs : for modelling an arbitrary number of processors

Test with model TXDisPro08-gcd.txs and SUT SutGcd.java.

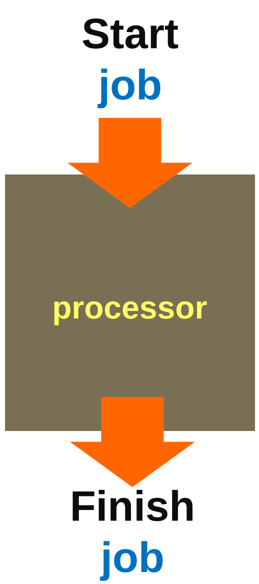
Repeat with model TXDisPro09-purp.txs. Explain the difference.

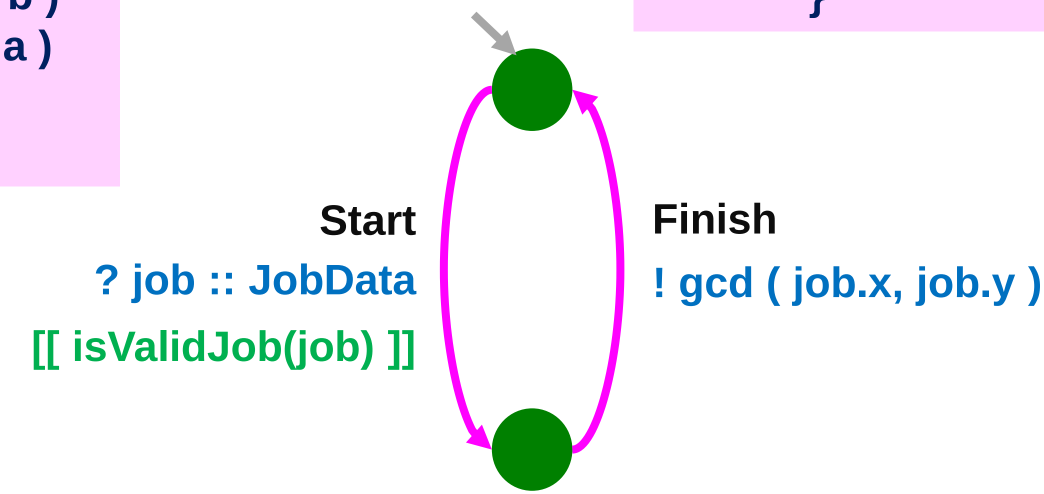
Example: Dispatcher-Processing System

```
FUNCDEF  gcd ( a, b :: Int ) :: Int
 ::=
       IF a == b
       THEN  a
       ELSE  IF a > b
                THEN  gcd ( a - b, b )
                ELSE  gcd ( a, b - a )
                FI
       FI
```

state + data

```
TYPEDEF  JobData
  ::=  JobData
          {  jobId       :: Int
          ;  jobDescr :: String
          ;  x, y          :: Int
          }
```






```
FUNCDEF  isValidJob ( jobdata :: JobData )  :: Bool
  ::=
          jobdata.jobId > 0
      /\  strinre ( jobdata.jobDescr, REGEX('[A-Z][0-9]{2}[a-z]+') )
      /\  . . . . .
```

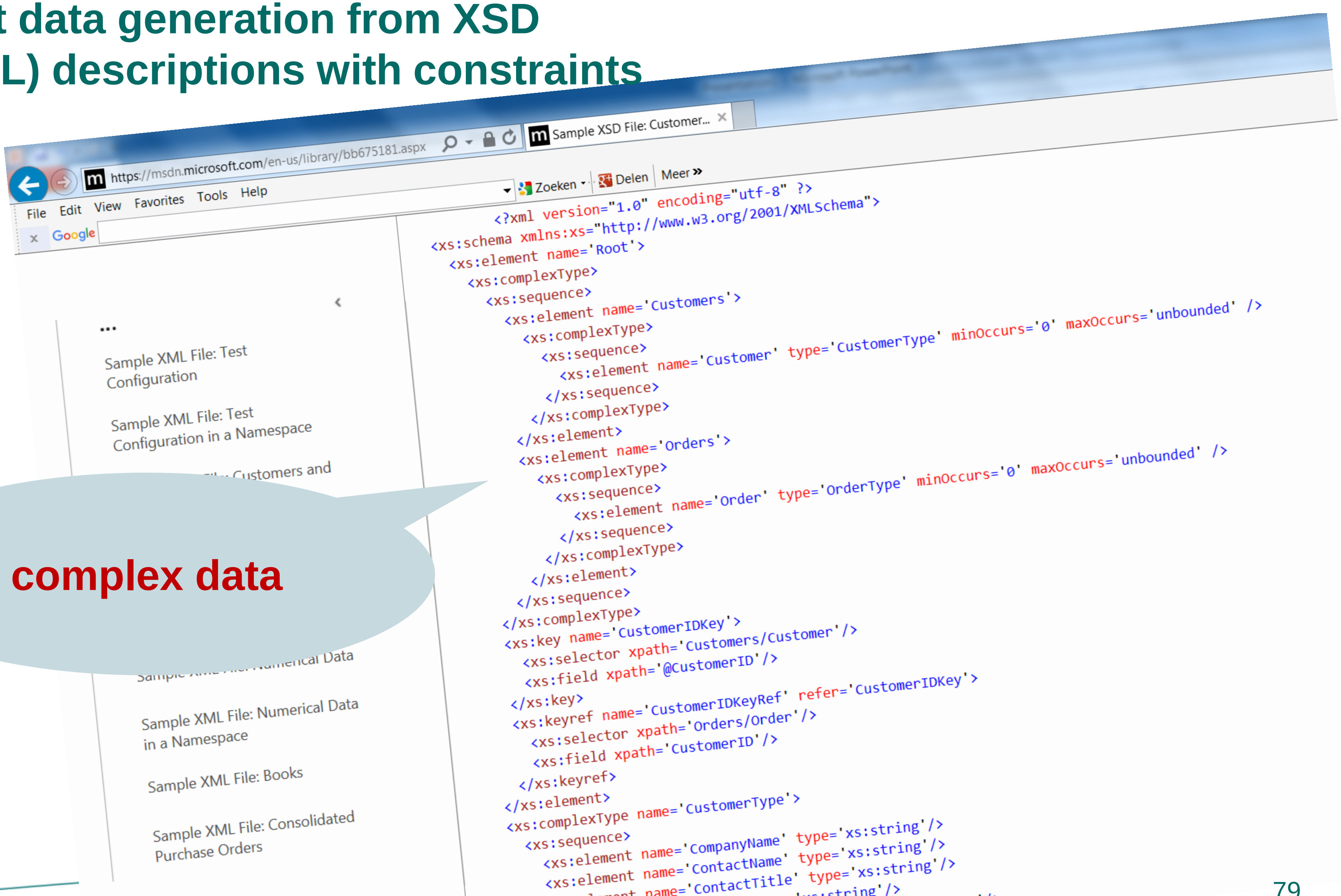
TorXakis: Exercise

Test with model TXDisPro10-jobdata.txs and SUT SutJobData.java.

Repeat with SUT model SutWithError.java. What is the Error?

More Complex Data

Test data generation from XSD (XML) descriptions with constraints



complex data

TorXakis: Exercise

Go to .....\examps.testnet\ReadWriteConflict.

Consider the model ReadWrite.txs.

This model has finite behaviour, i.e., it stops after 7 steps (deadlocks).

From he model, argue what the last action is.

Check your argumentation by stepping through the model.

A Goat, a Wolf, Cabbage, and a Cat

…..\examps.testnet\RiverCrossing

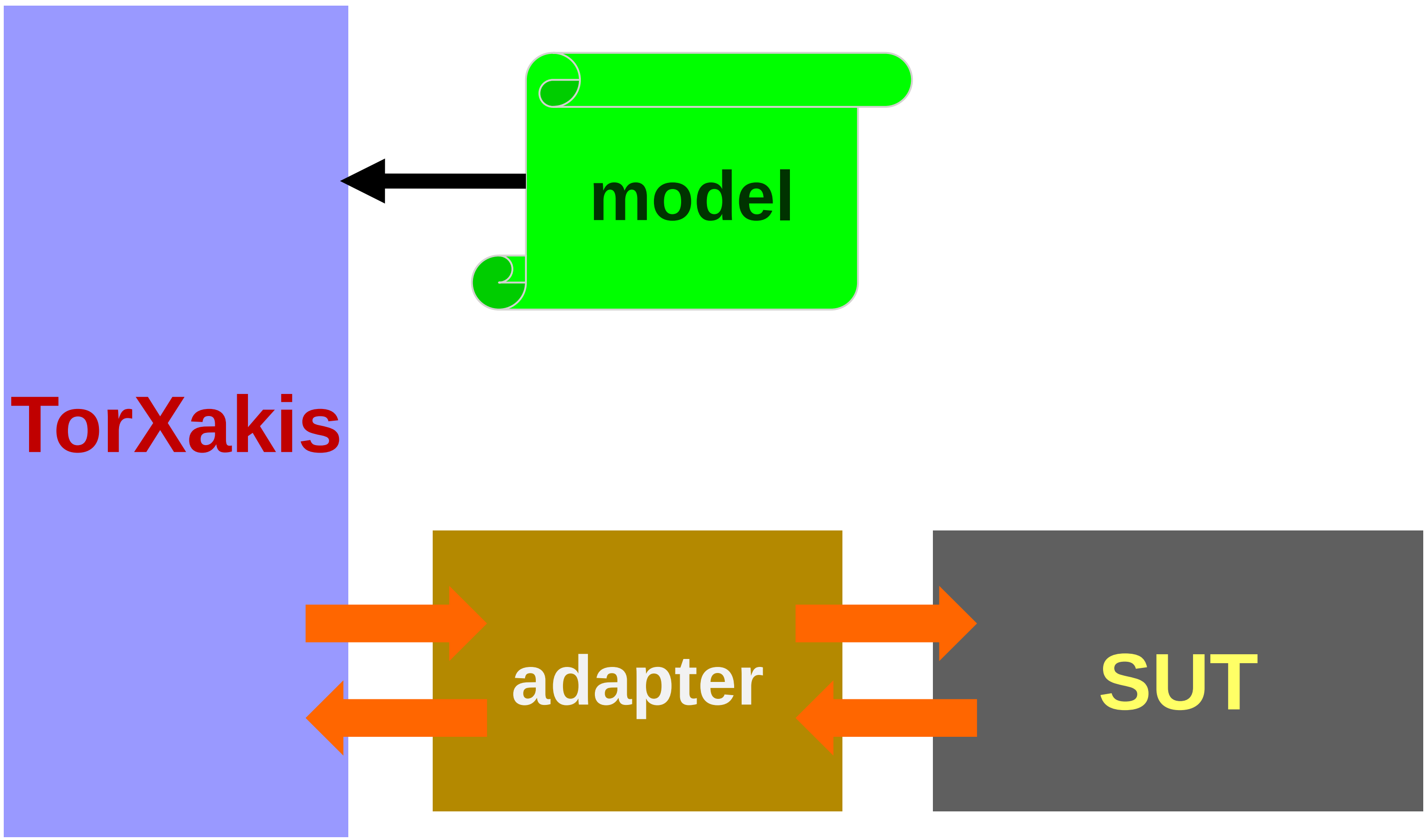
Model-Based Testing of a Little Game

Input via standard IO

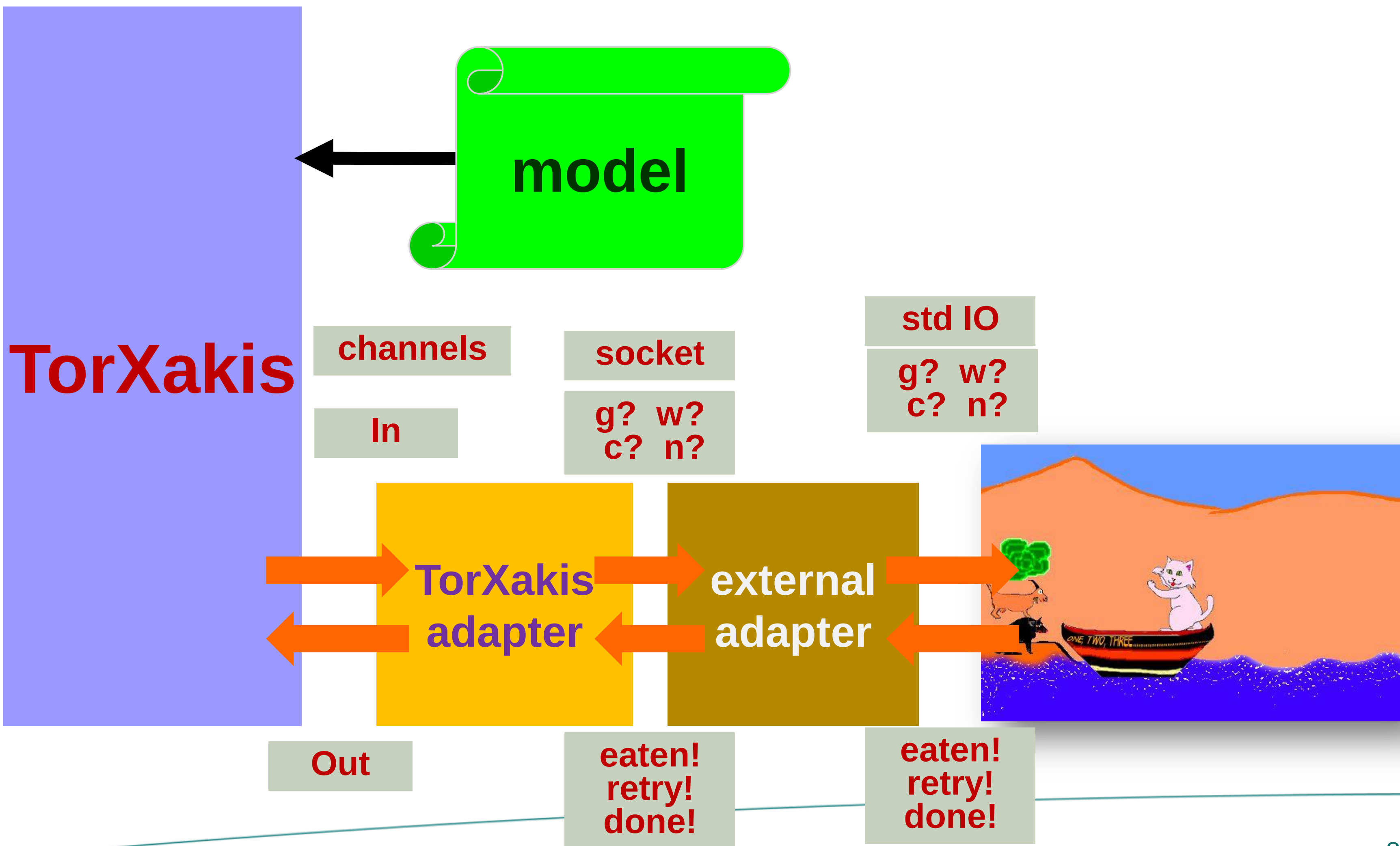
Inputs:
g? w? c? n?

Outputs:
eaten! retry! done!

A Goat, a Wolf, Cabbage, and a Cat



A Goat, a Wolf, Cabbage, and a Cat



TorXakis: Exercise

Go to .....\examps.testnet\RiverCrossing.

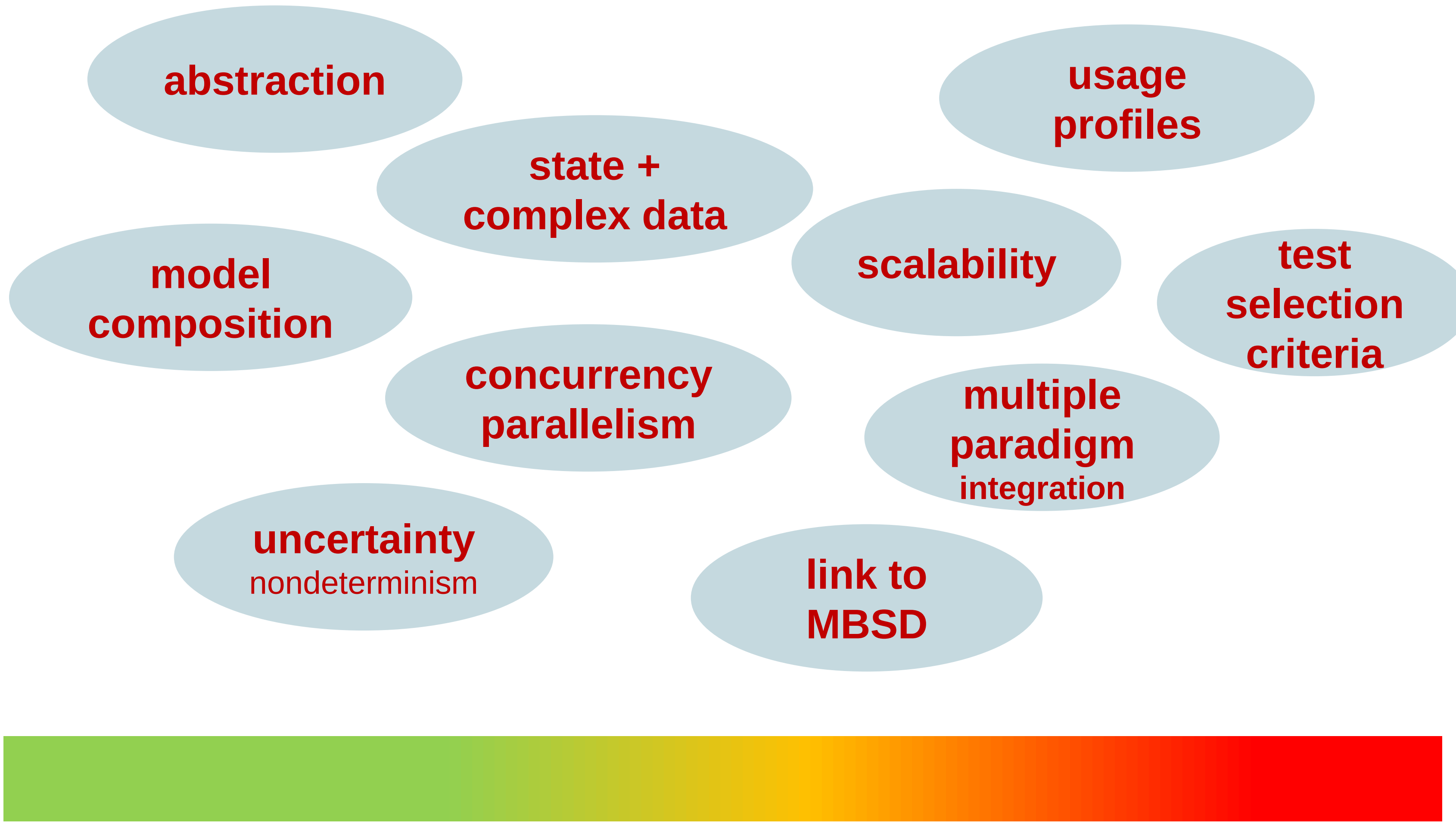
Consider the model RiverState.txs (visualization in RiverState.pdf)

The SUT is sut.bat or sut.sh.

Does the behaviour of SUT correspond to the model?

Explain .....

Next Generation MBT : TorXakis Status



TorXakis

Questions?