Testing Times

On Model-Based Functional Testing for Real-Time Embedded Systems

Ed Brinksma
University of Twente, NL
University of Aalborg,DK

CISS

University of Twente
The Netherland

Theme

How can theory help to improve
the quality and productivity
of testing conformance of real-life
embedded software systems?

Overview

- Model-based testing
  - model-driven test generation
  - implementation relations
  - input/output systems, quiescence
- Test generation & execution
  - TorX
  - Demo
  - Case studies
- Current and future developments
  - Testing real-time systems
  - Testing and tolerance
  - Test data generation

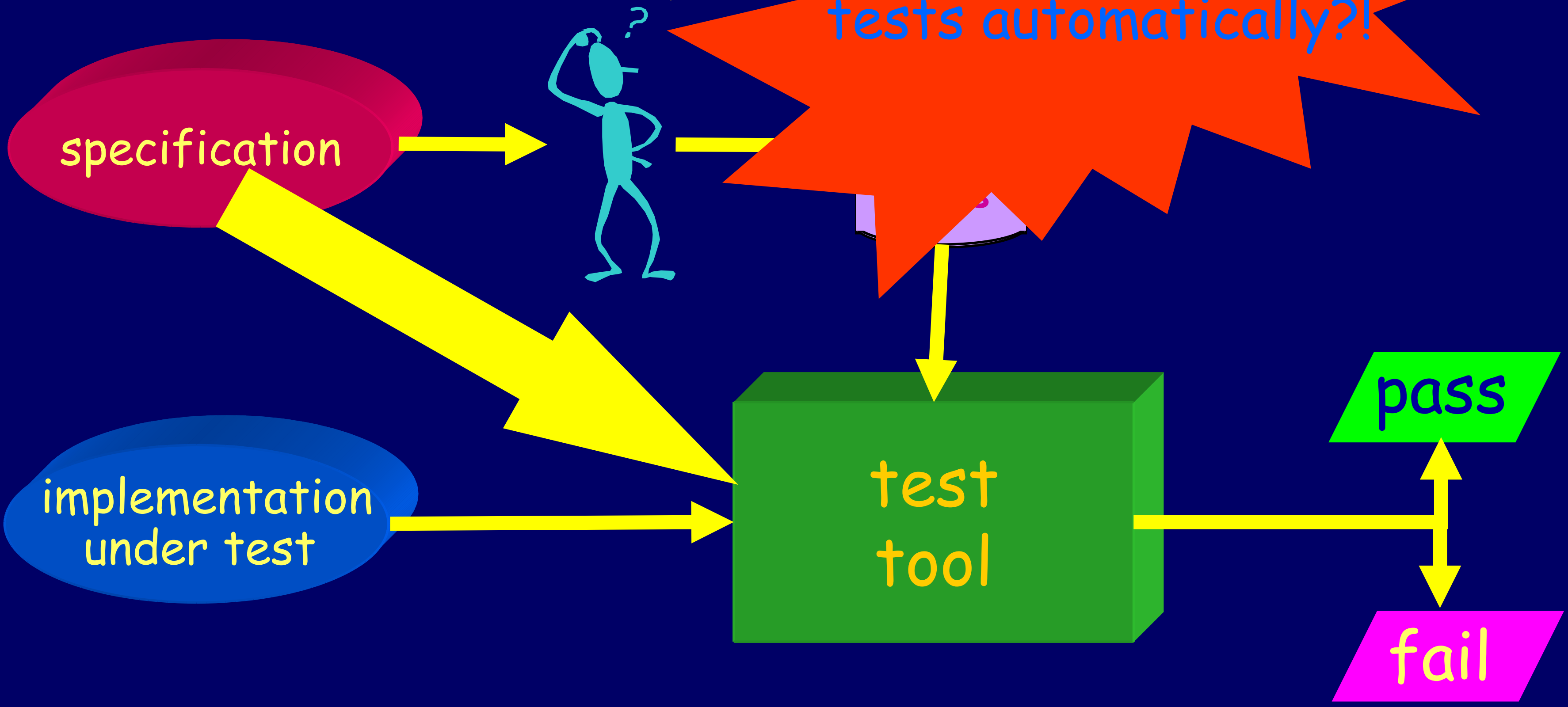
Overview

- Model-based testing
  - model-driven test generation
  - implementation relations
  - input/output systems, quiescence
- Test generation & execution
  - TorX
  - Demo
  - Case studies
- Current and future developments
  - Testing real-time systems
  - Testing and tolerance
  - Test data generation

Test Automation

Traditional test automatio
= tools to execute and

Why not generate tests automatically?!



Our Context

Formal methods:

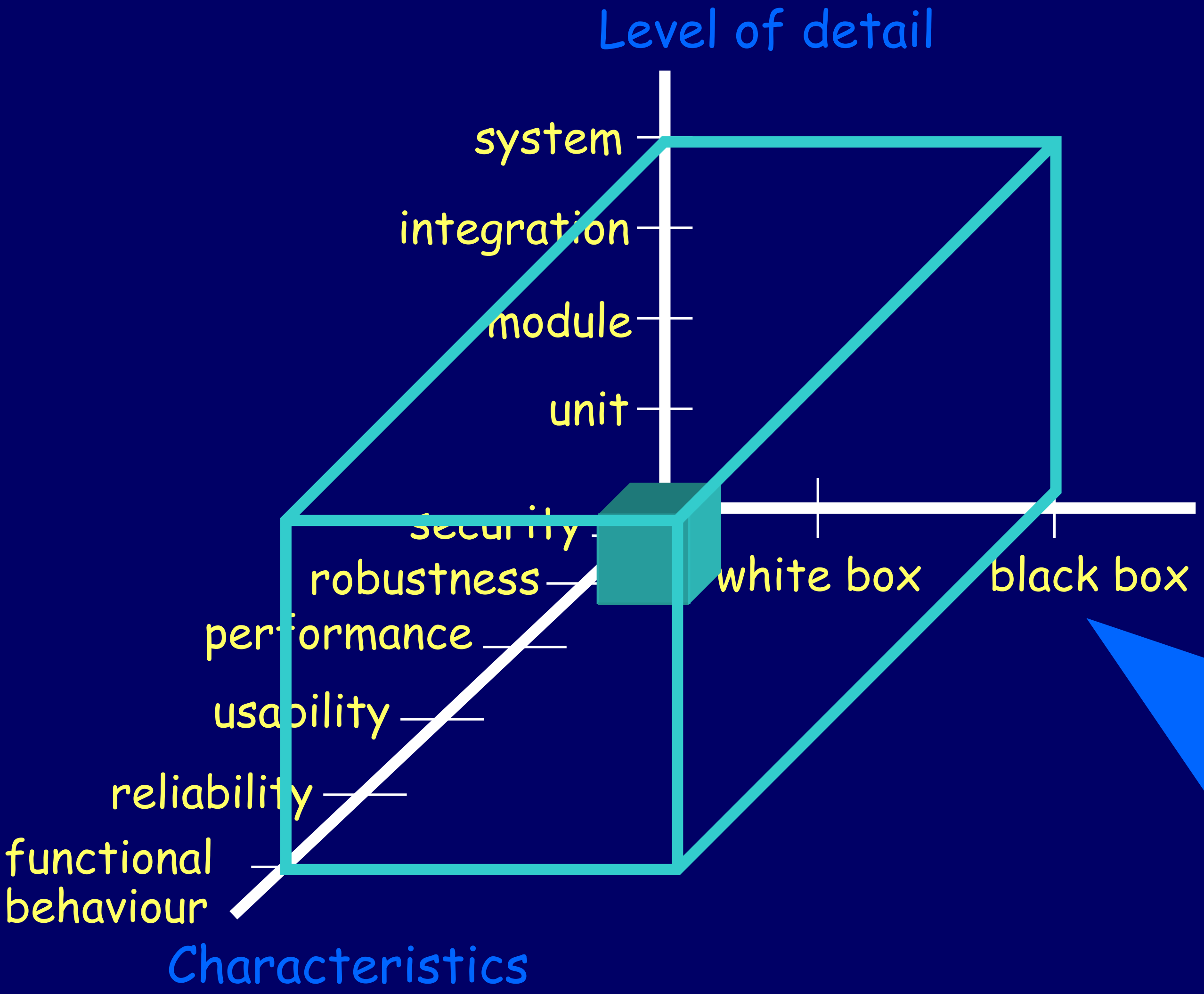
- unambiguous spe ficat -driven”)
- precise notion of
- formal validati
- algorithmic

Dynamic behavio

- concentrate on c tro behaviour
- concurrency and non-determinism

Models are hard to make, but easier to maintain

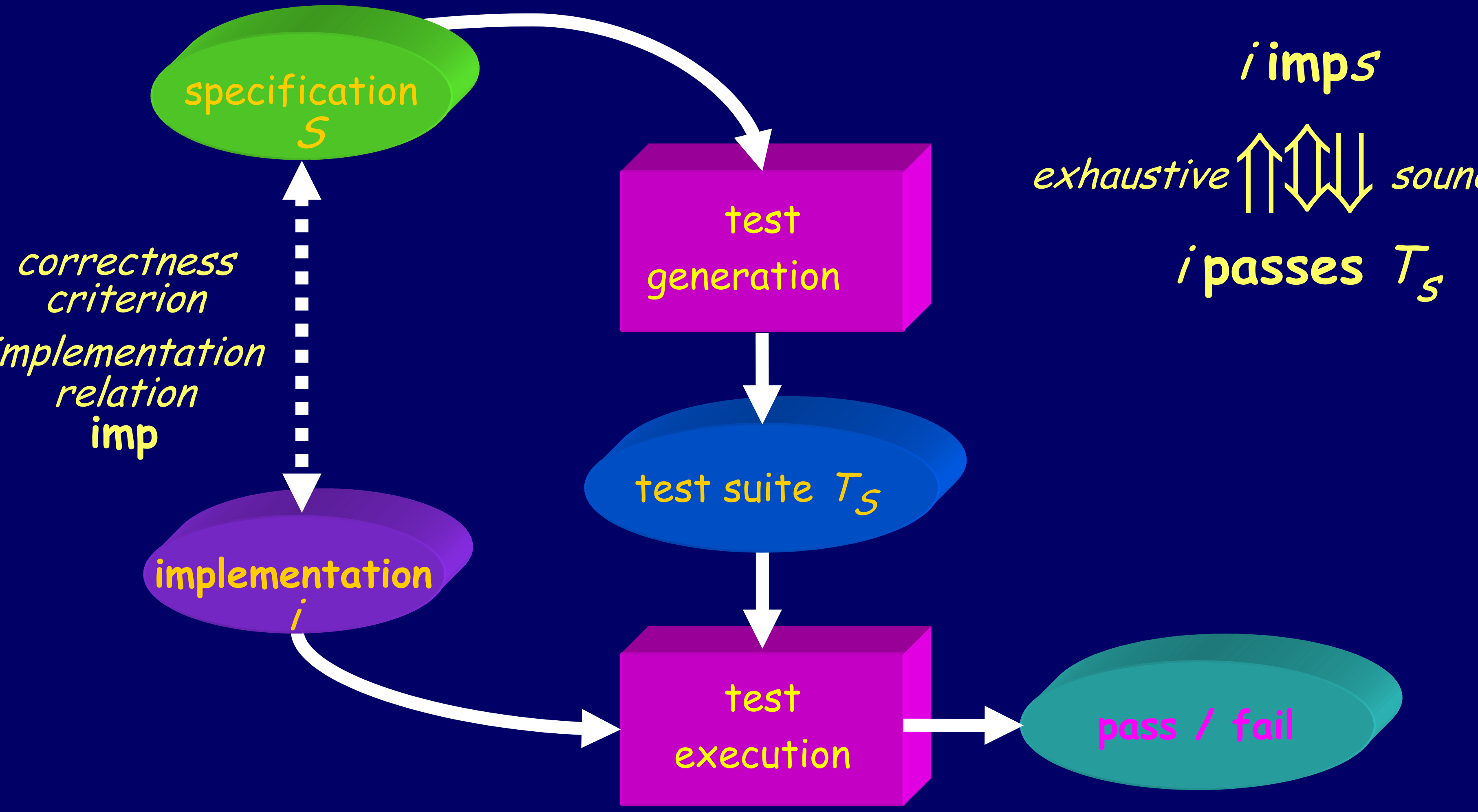
Conformance Testing



Reasons:

- sources inaccessible
- sources unavailable
- simpler models

Formal Testing



specification S

test generation

test suite T_S

implementation i

test execution

pass / fail

correctness criterion

implementation relation

imp

i **imps** *s*

exhaustive ⇑⇕⇓ *soun*

i **passes** T_s

Implementation Relation

IDEA:

Observations = Action Logs (= traces)

including deadlocks

The Qu Machine

Mind the nondeterminism!

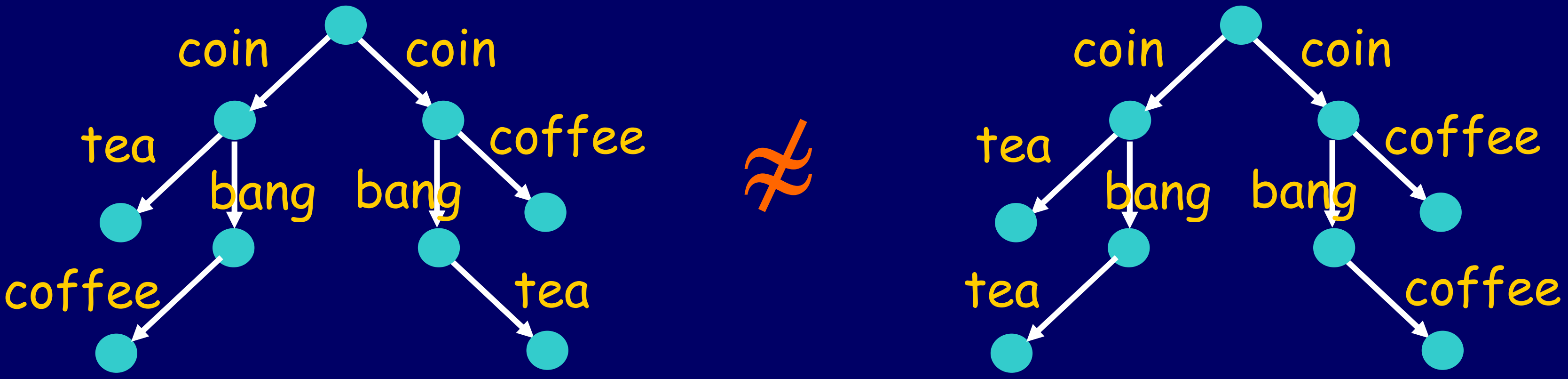
IDEA 2:

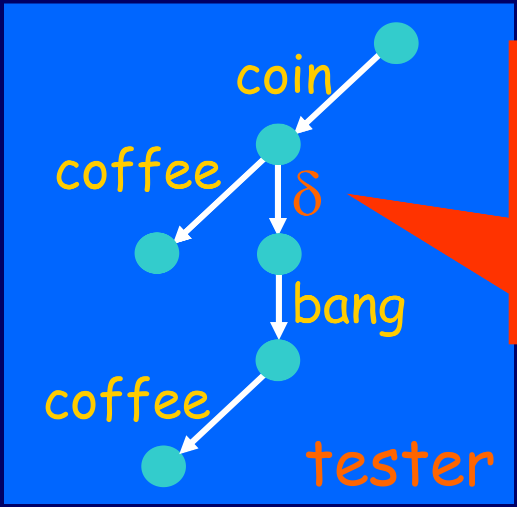
Observations = Action Logs (= traces)

including deadlocks

AND RECOVERY BEHAVIOUR

The Quirky Coffee Machine





δ = deadlock
only enabled
if coffee is not

Input/Output Systems

- testing actions are usually directed, i.e. there are inputs and outputs
- systems can always accept all inputs (input enabledness)
- testers are I/O systems
  - output (stimulus) is input for the SUT
  - input (response) is output of the SUT

Quiescence

- With input enabledness a system S deadlocks with a tester T if and only if:

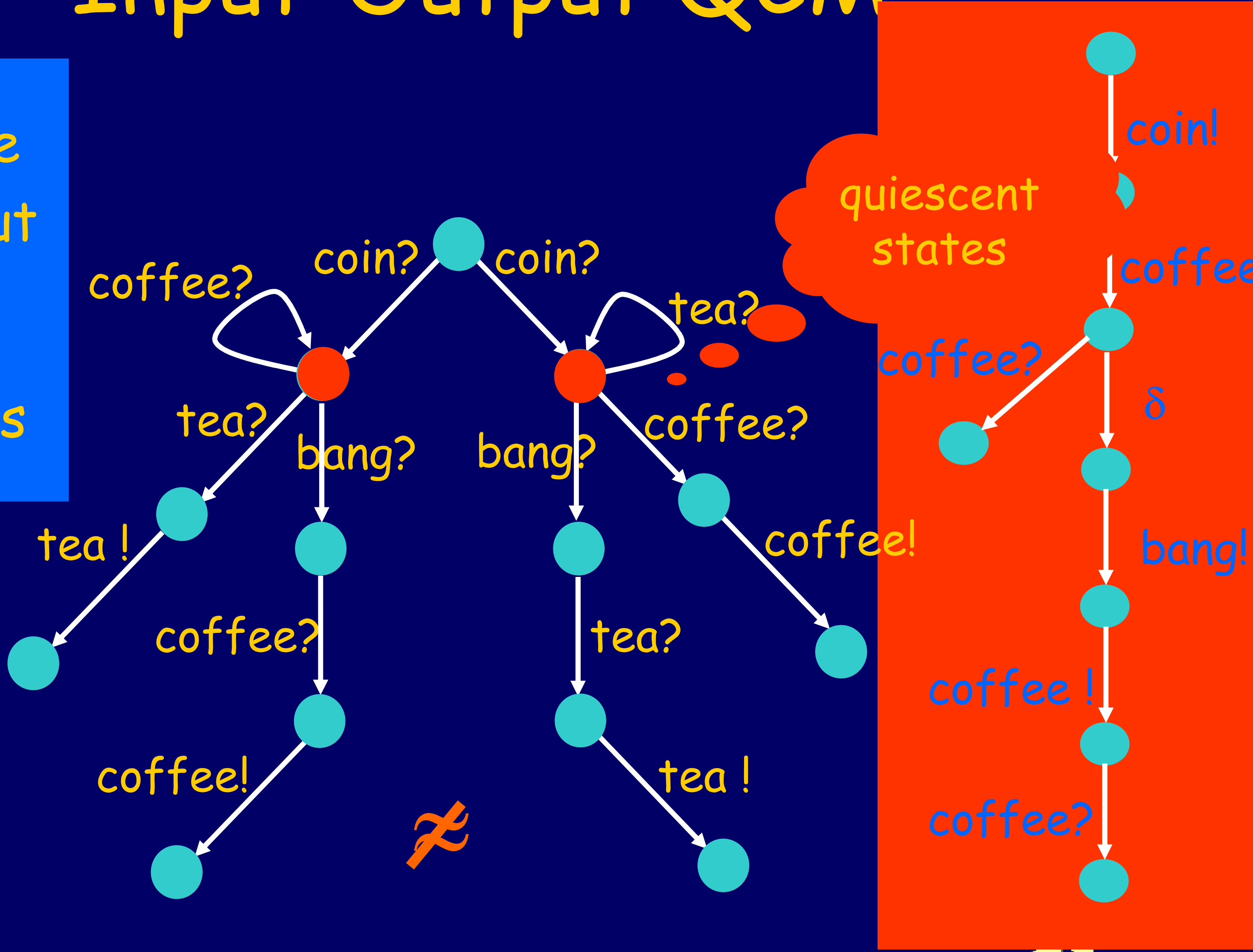
  1. T produces no stimuli, and
  2. S provides no responses

  This is known as quiescence

- We log quiescence and recovery in our observation traces

Input-Output QCM

states have implicit input loops for input enabledness



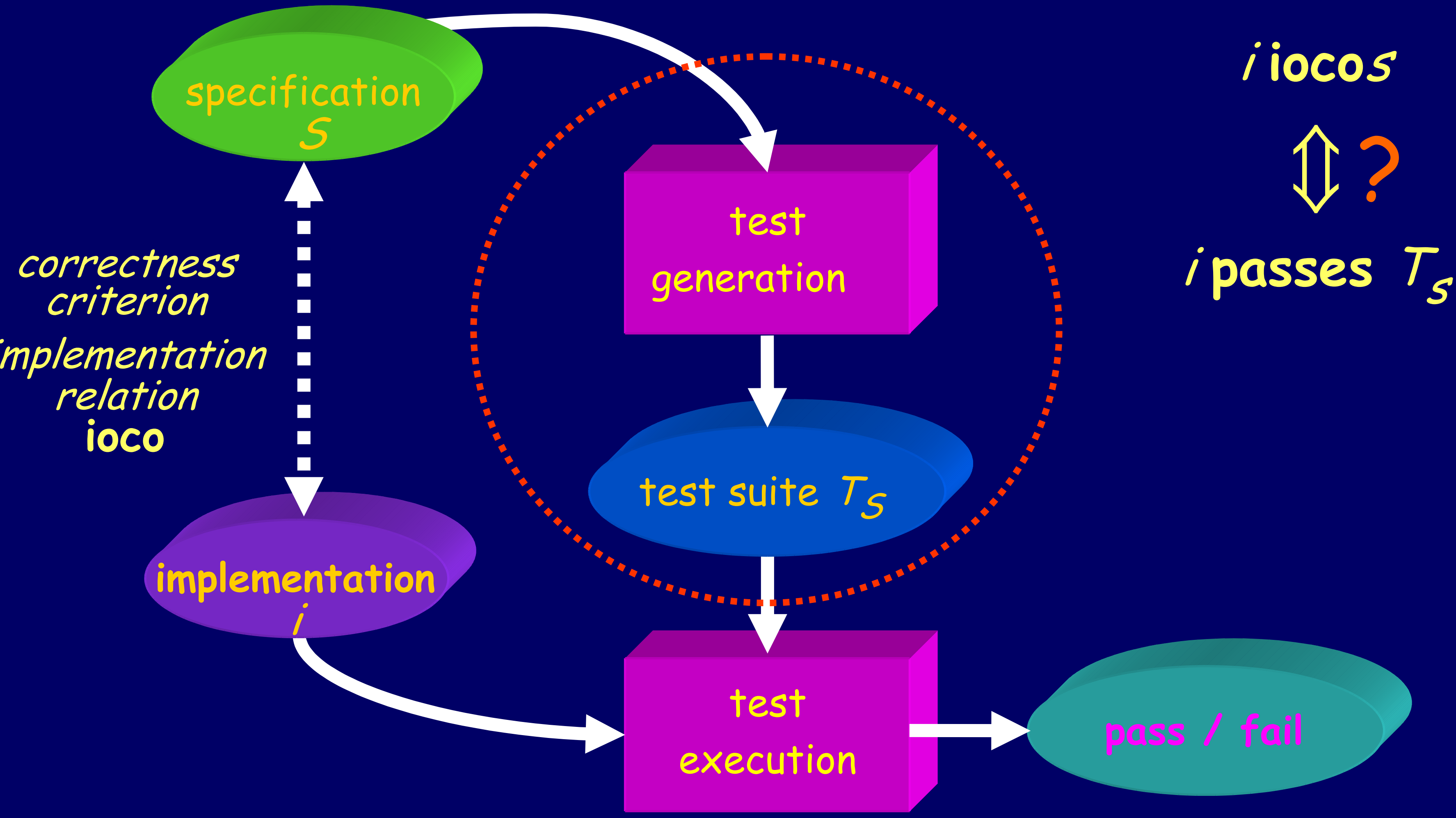
Implementation Relation ioco

To allow under-specification we restrict observations to traces of the specification.

Intuition: I **ioco**-conforms to S, iff

- if I produces output x after a given trace of S, then S can produce x after that trace
- if I cannot produce output after a given trace of S, then it is possible that S cannot produce any output after that trace (*quiescence*)

Formal Testing



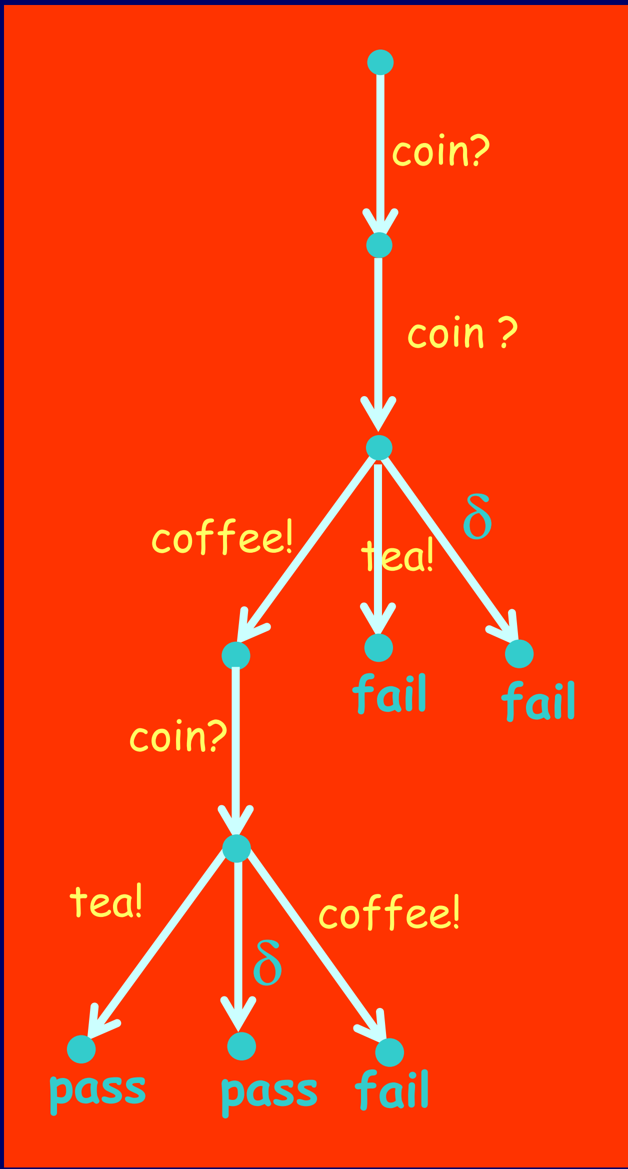
Test Cases

TTCN !

Tes

test case t	
!coin	
!coin ; Start timer1	
?tea	fail
?timer1	fail
?coffee	
!coin ; Start timer1	
?tea	pass
?timer2	pass
?coffee	fail

ail



Test Gener tion rithm:

```
S := {s0};

:

                                          ter a

                                 oi       ates after b
```

SOUND

i.e no correct implementation rejected

&

(limit) COMPLETE

i.e all incorrect implementations

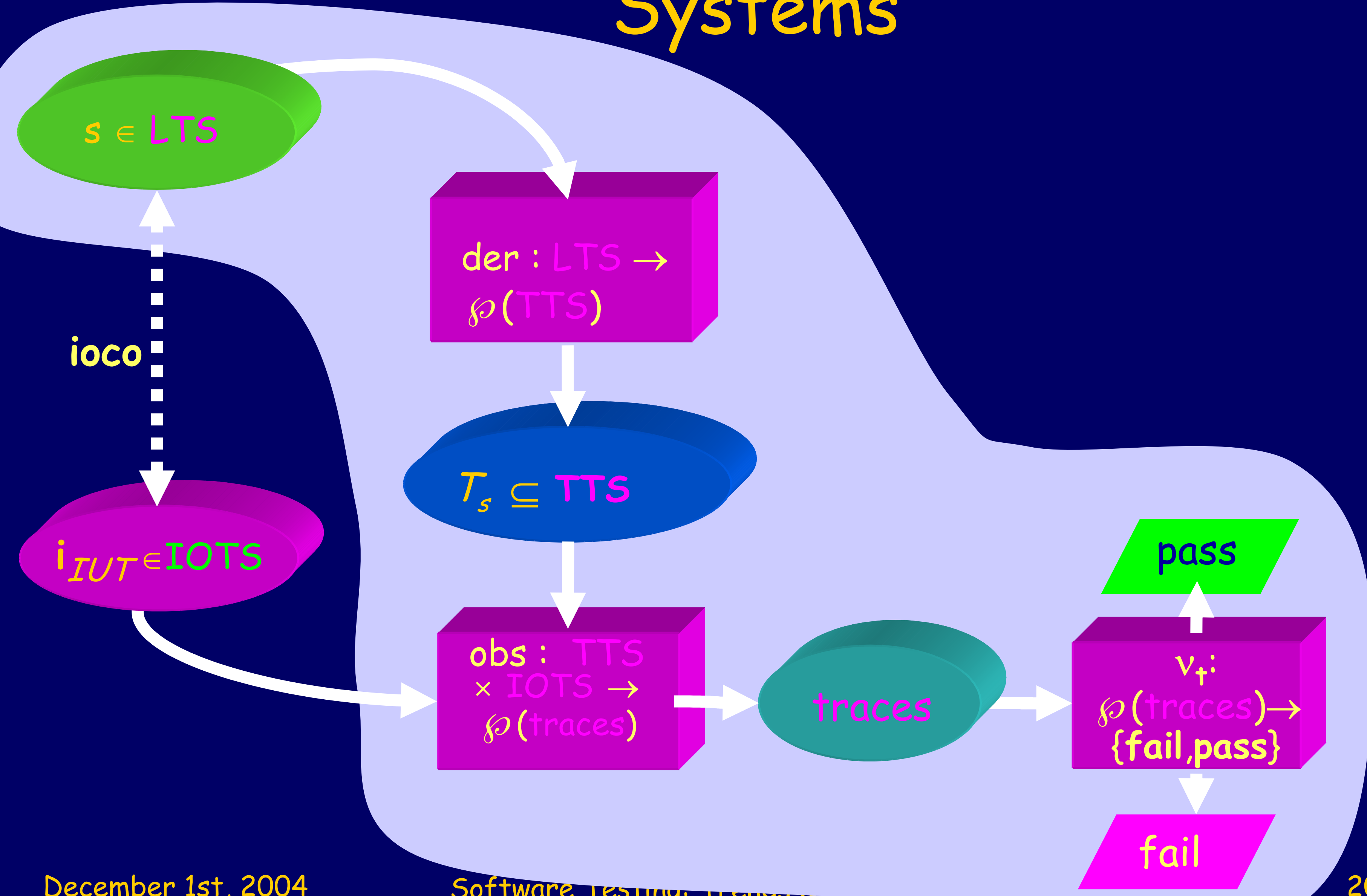
rejected by repeated runs

Every c e run r the algorithm executes a test

Overview

- Model-based testing
  - model-driven test generation
  - implementation relations
  - input/output systems, quiescence
- Test generation & execution
  - TorX
  - Demo
  - Case studies
- Current and future developments
  - Testing real-time systems
  - Testing and tolerance
  - Test data generation

Formal Testing with Transition Systems

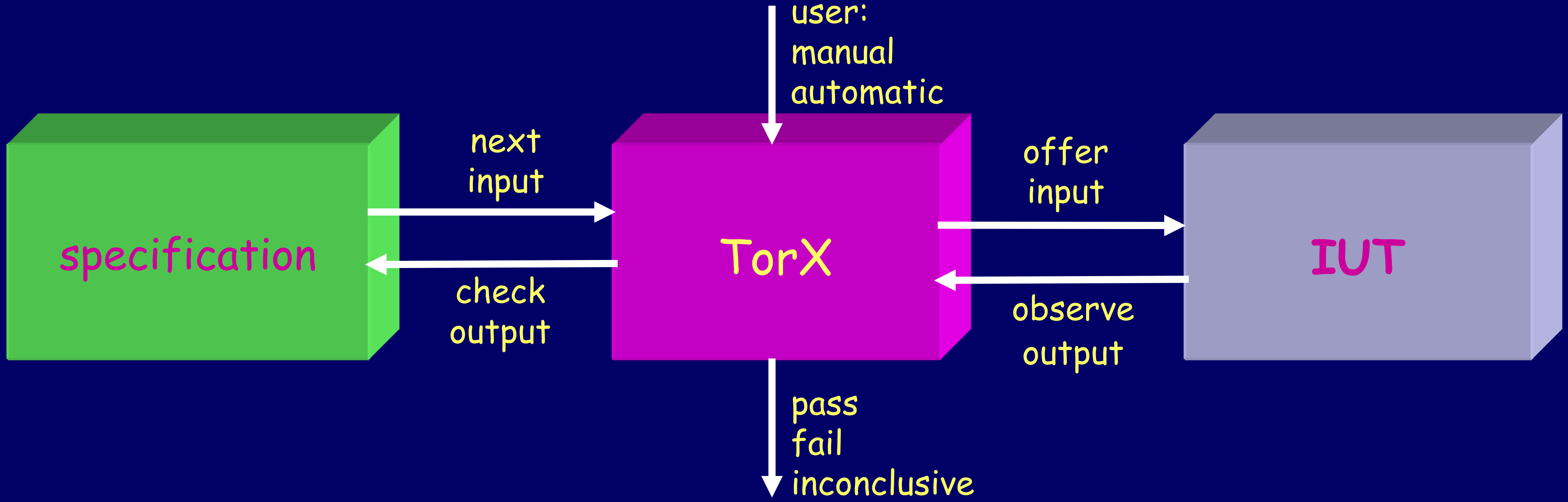


Test Generation Tools for ioco

- TVEDA (CNET - France Telecom)
  - derives TTCN tests from single process SDL specification
  - developed from practical experiences
  - implementation relation R1 ≈ **ioco**
- TGV (IRISA - Rennes)
  - derives tests in TTCN from LOTOS or SDL
  - uses test purposes to guide test derivation
  - implementation relation: unfair extension of **ioco**
- TestComposer
  - Combination of TVEDA and TGV in ObjectGeode
- TestGen (Stirling)
  - Test generation for hardware validation
- TorX (Côte de Resyste)

A Test Tool : TorX

- On-the-fly test generation and test execution
- Implementation relation: **ioco**
- Specification languages: LOTOS, Promela, FSP, Automata, UML



user:
manual
automatic

specification

next
input

check
output

TorX

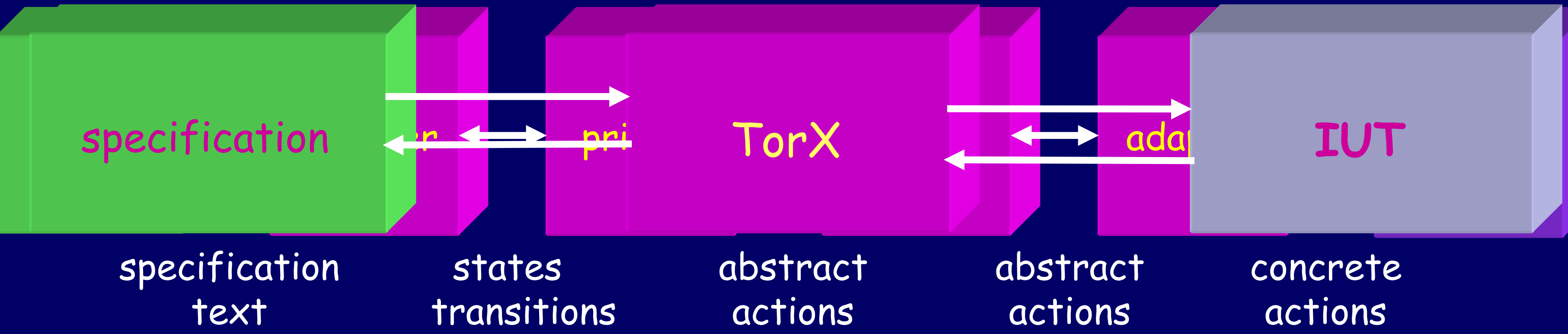
offer
input

observe
output

IUT

pass
fail
inconclusive

TorX Tool Architecture



specification

er

pri

TorX

ada

IUT

specification text

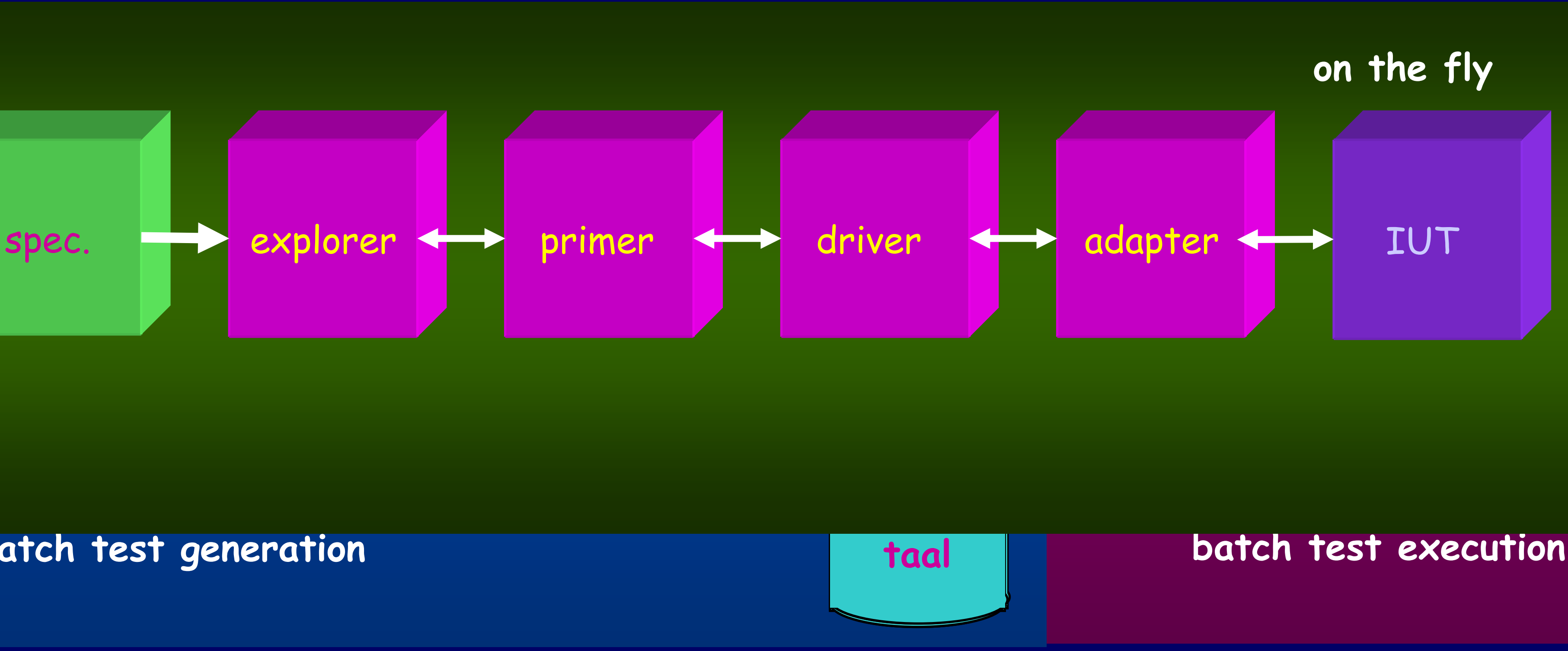
states transitions

abstract actions

abstract actions

concrete actions

On-the-Fly ↔ Batch Testing

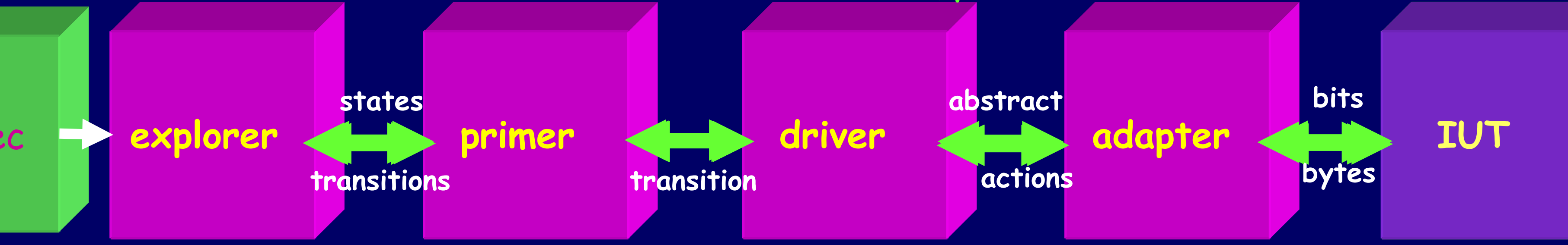


On-the-Fly Testing

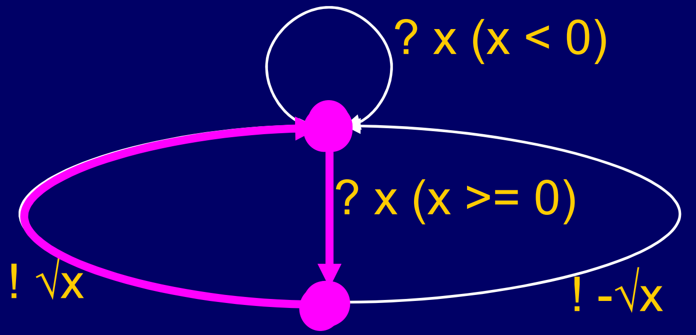
[illegible]

! [illegible] Action Abstract action Concrete action

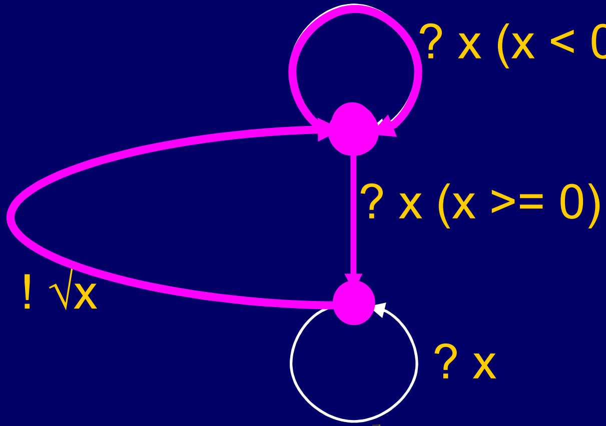
! [illegible] (quiescence) ? [illegible] (quiescence) ? [illegible] (quiescence) ? [illegible]



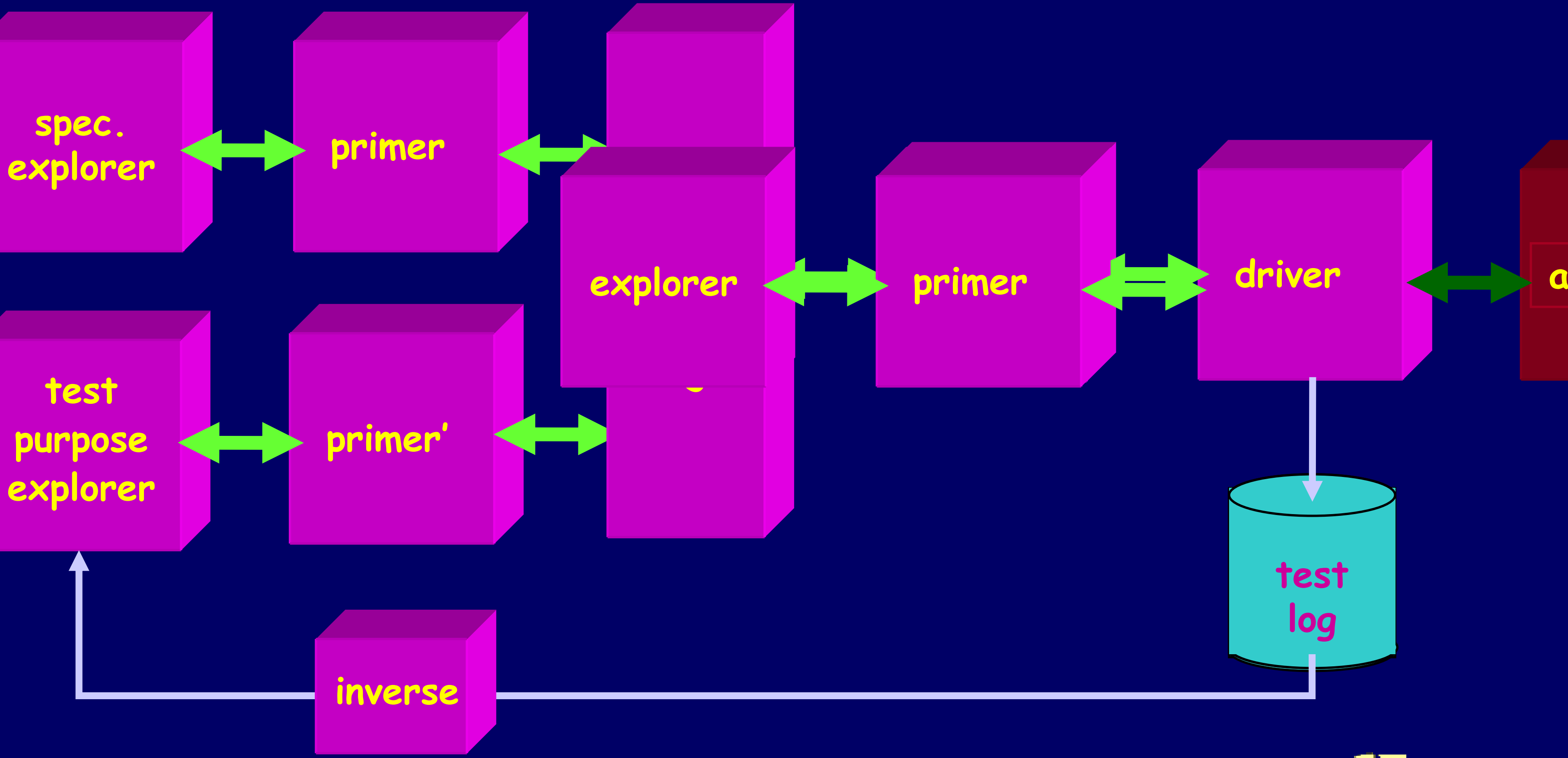
specification



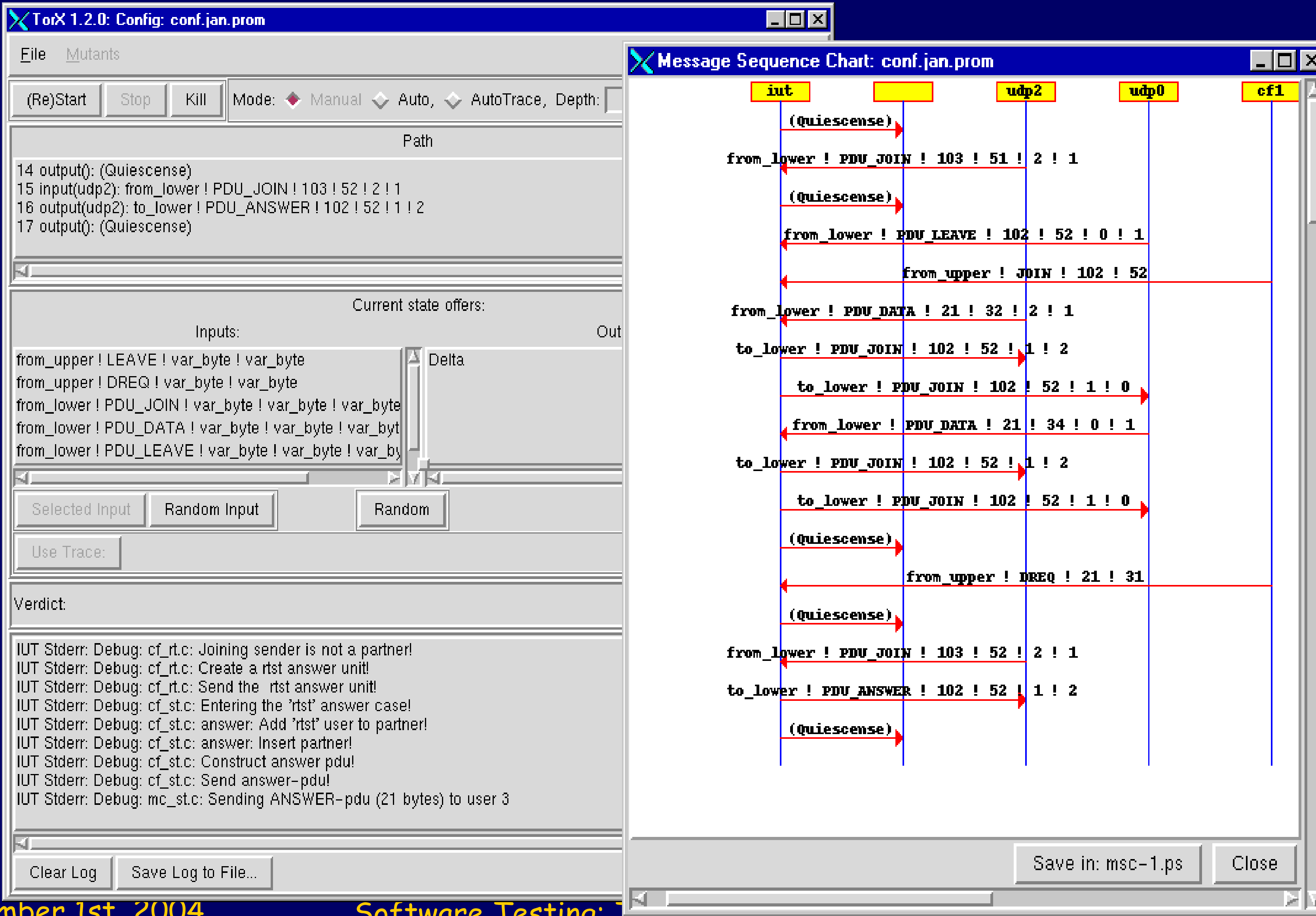
implementatio



TorX : Test Purposes, Selection, ...



TorX



TorX 1.2.0: Config: conf.jan.prom
File
Mutants
(Re)Start
Stop
Kill
Mode: ● Manual ◇ Auto, ◇ AutoTrace, Depth:
Path

```
14 output(): (Quiescense)
15 input(udp2): from_lower ! PDU_JOIN ! 103 ! 52 ! 2 ! 1
16 output(udp2): to_lower ! PDU_ANSWER ! 102 ! 52 ! 1 ! 2
17 output(): (Quiescense)
```

```
from_upper ! LEAVE ! var_byte ! var_byte
from_upper ! DREQ ! var_byte ! var_byte
from_lower ! PDU_JOIN ! var_byte ! var_byte ! var_byte
from_lower ! PDU_DATA ! var_byte ! var_byte ! var_byt
from_lower ! PDU_LEAVE ! var_byte ! var_byte ! var_by
```

```
Delta
```

```
IUT Stderr: Debug: cf_rt.c: Joining sender is not a partner!
IUT Stderr: Debug: cf_rt.c: Create a rtst answer unit!
IUT Stderr: Debug: cf_rt.c: Send the  rtst answer unit!
IUT Stderr: Debug: cf_st.c: Entering the 'rtst' answer case!
IUT Stderr: Debug: cf_st.c: answer: Add 'rtst' user to partner!
IUT Stderr: Debug: cf_st.c: answer: Insert partner!
IUT Stderr: Debug: cf_st.c: Construct answer pdu!
IUT Stderr: Debug: cf_st.c: Send answer-pdu!
IUT Stderr: Debug: mc_st.c: Sending ANSWER-pdu (21 bytes) to user 3
```

```
(Quiescense)
from_lower ! PDU_JOIN ! 103 ! 51 ! 2 ! 1
(Quiescense)
from_lower ! PDU_LEAVE ! 102 ! 52 ! 0 ! 1
from_upper ! JOIN ! 102 ! 52
from_lower ! PDU_DATA ! 21 ! 32 ! 2 ! 1
to_lower ! PDU_JOIN ! 102 ! 52 ! 1 ! 2
to_lower ! PDU_JOIN ! 102 ! 52 ! 1 ! 0
from_lower ! PDU_DATA ! 21 ! 34 ! 0 ! 1
to_lower ! PDU_JOIN ! 102 ! 52 ! 1 ! 2
to_lower ! PDU_JOIN ! 102 ! 52 ! 1 ! 0
(Quiescense)
from_upper ! DREQ ! 21 ! 31
(Quiescense)
from_lower ! PDU_JOIN ! 103 ! 52 ! 2 ! 1
to_lower ! PDU_ANSWER ! 102 ! 52 ! 1 ! 2
(Quiescense)
```

TorX Case Studies

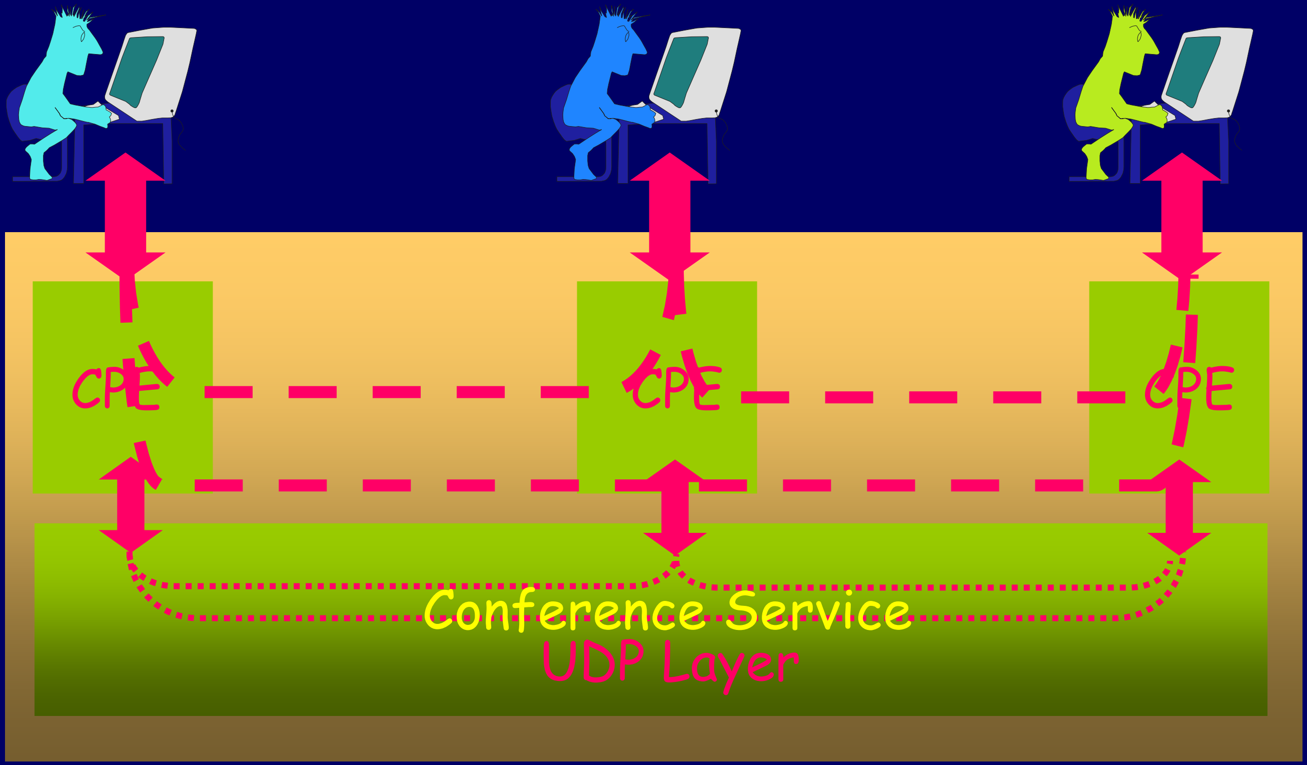
● Conference Protocol	academic
● EasyLink TV-VCR protocol	Philips
● Cell Broadcast Centre component	CMG
● Road Toll Payment Box protocol	Interpay
● V5.1 Access Network protocol	Lucent
● Easy Mail Melder	CMG
● FTP Client	academic
● "Oosterschelde" storm surge barrier-control	CMG
● TANGRAM: testing VLSI lithography machine	ASML

The Conference Protocol Experiment

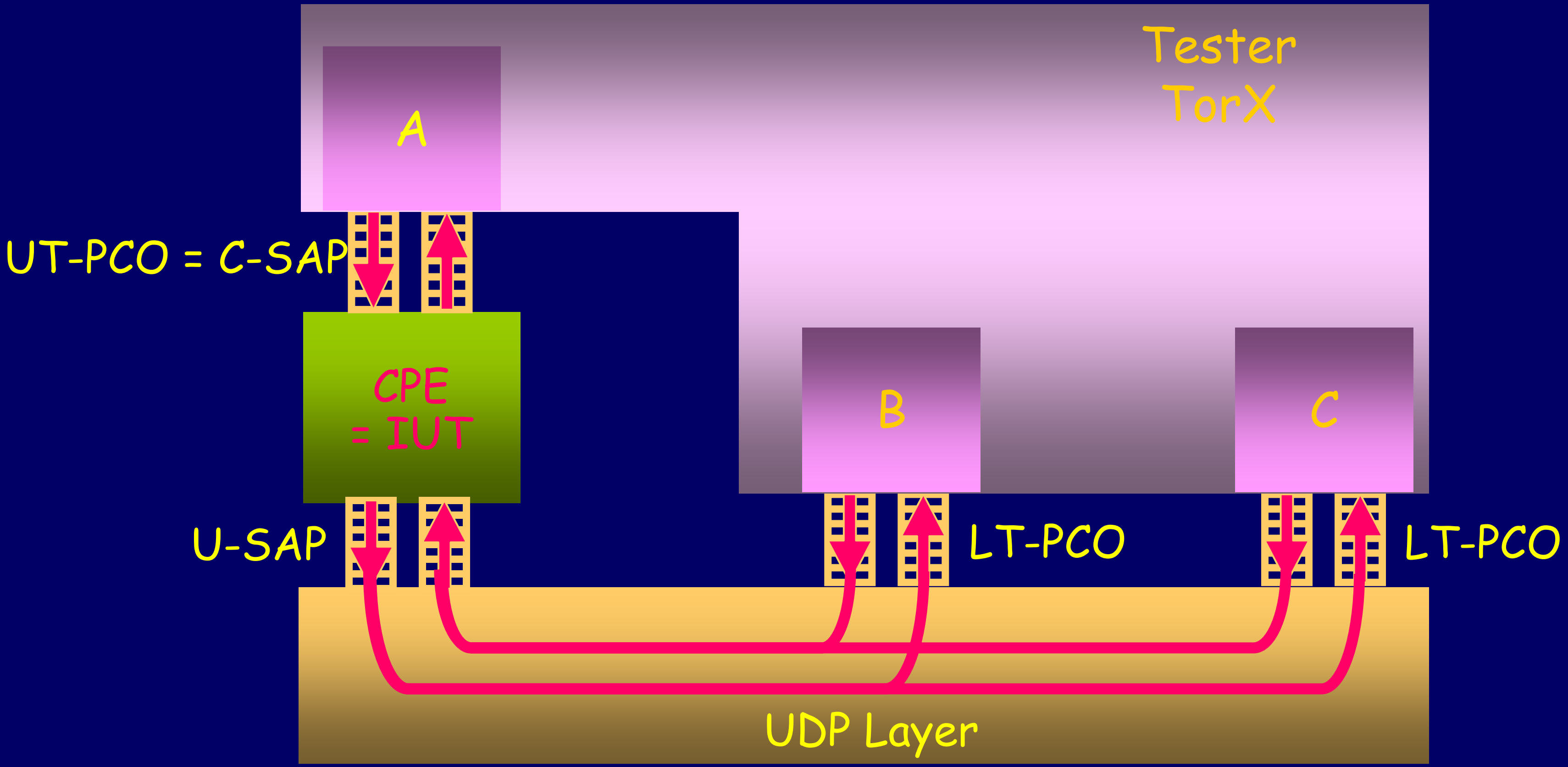
- Academic benchmarking experiment,
  initiated for test tool evaluation and comparison
- Based on really testing different implementations
- Simple, yet realistic protocol (chatbox service)
- Specifications in LOTOS, Promela, SDL, EFSM
- 28 different implementations in *C*
  - one of them (assumed-to-be) correct
  - others manually derived mutants
- http://fmt.cs.utwente.nl/ConfCase

The Conference Protocol

join
leave
send
receive



Conference Protocol Test Architecture



The Conference Protocol Experiments

- TorX - LOTOS, Promela : on-the-fly ioco testing

  Axel Belinfante et al.,
  Formal Test Automation: A Simple Experiment
  IWTCS 12, Budapest, 1999.

- Tau Autolink - SDL : semi-automatic batch testing
- TGV - LOTOS : automatic batch testing with test purposes

  Lydie Du Bousquet et al.,
  Formal Test Automation: The Conference Protocol with TGV/TorX
  TestCom 2000, Ottawa.

- PHACT/Conformance KIT - EFSM : automatic batch testing

  Lex Heerink et al.,
  Formal Test Automation: The Conference Protocol with PHACT
  TestCom 2000, Ottawa.

Conference Protocol Results

Results:	TorX	TorX	PHACT	TGV	TGV
	LOTOS	Promela	EFSM	LOTOS	LOTOS
				random	purposes
fail	25	25	21	25	24
pass	3	3	6	3	4
"core dump"	0	0	1	0	0
pass	000	000	000	000	000
	444	444	444	444	444
	666	666	666	666	666
			289		332
			293		
			398		

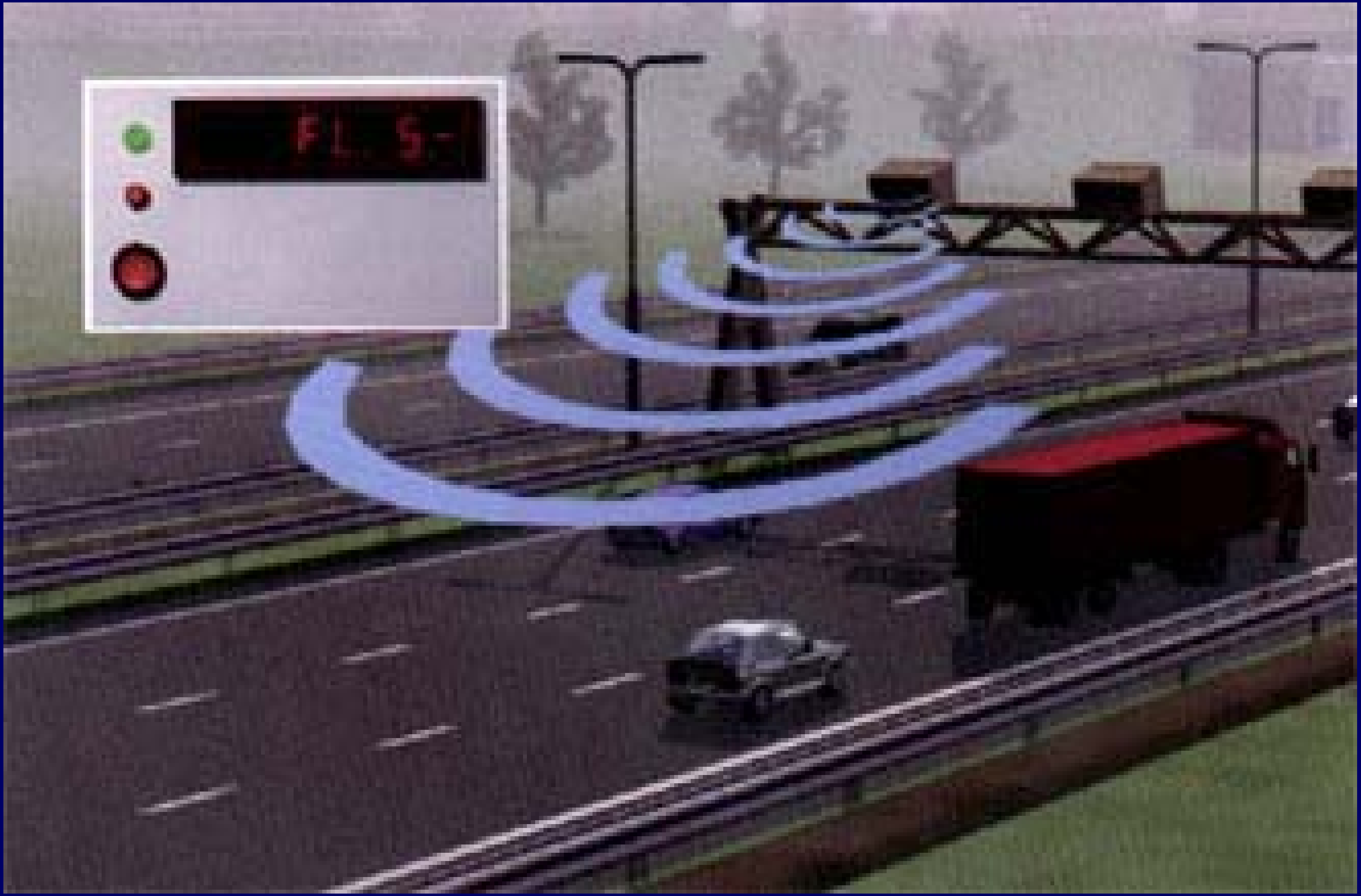
Conference Protocol Analysis

- Mutants 444 and 666 react to PDU's from non-existent partners:
  - no explicit reaction is specified for such PDU's, so **ioco**-correct, and TorX does not test such behaviour
- So, for LOTOS/Promela with TGV/TorX: All **ioco**-erroneous implementations detected
- EFSM:
  - two "additional-state" errors not detected
  - one implicit-transition error not detected

Conference Protocol Analysis

- TorX statistics
  - all errors found after 2 - 498 test events
  - maximum length of tests : > 500,000 test events
- EFSM statistics
  - 82 test cases with “partitioned tour method” (= UIO)
  - length per test case : < 16 test events
- TGV with manual test purposes
  - ~ 20 test cases of various length
- TGV with random test purposes
  - ~ 200 test cases of 200 test events

Interpay
Highway Tolling System

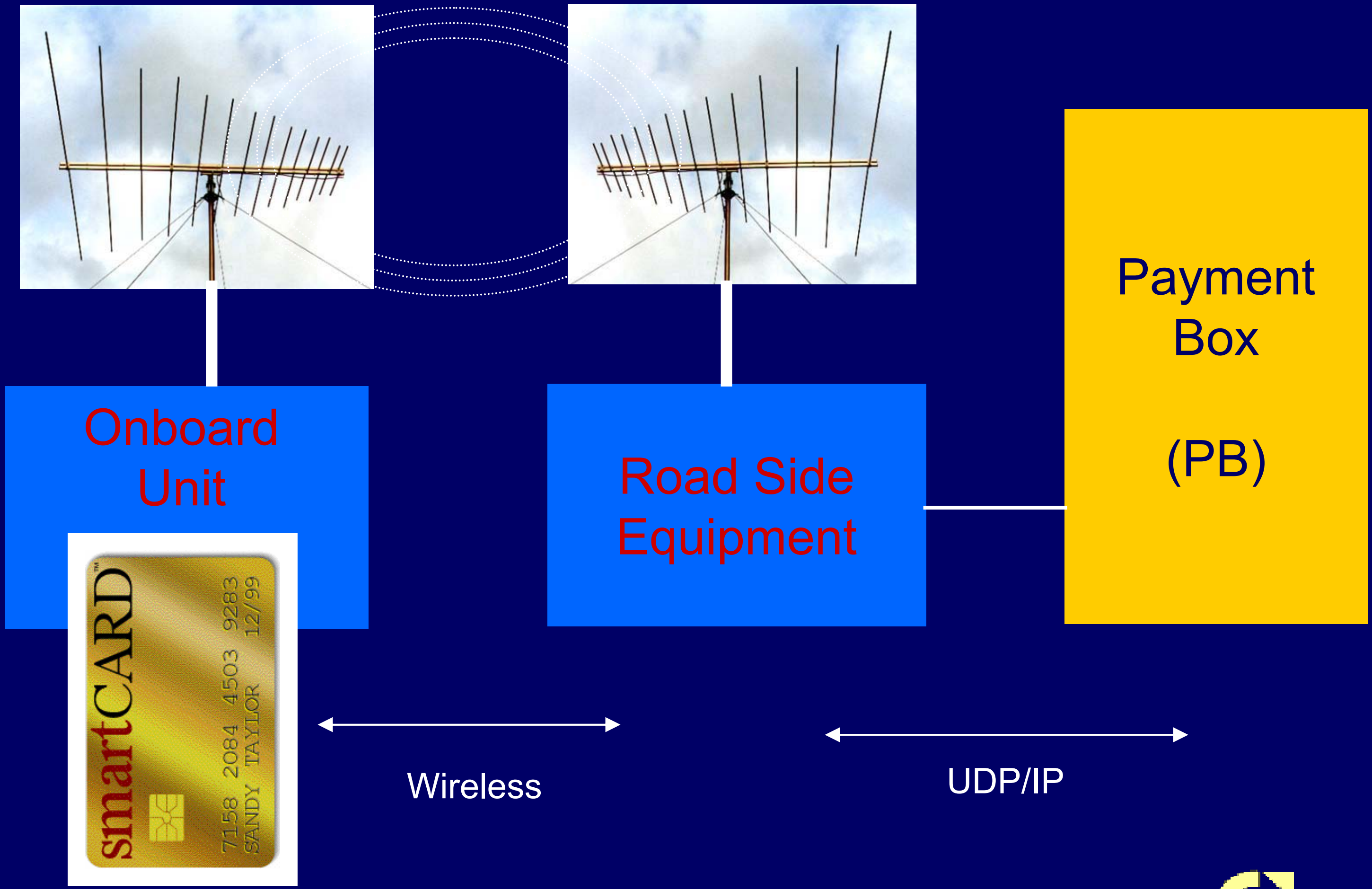


Highway Tolling Protocol

Characteristics :

- Simple protocol
- Parallellism : many cars at the same time
- Encryption
- System passed traditional testing phase

Highway Tolling System



Onboard Unit

Road Side Equipment

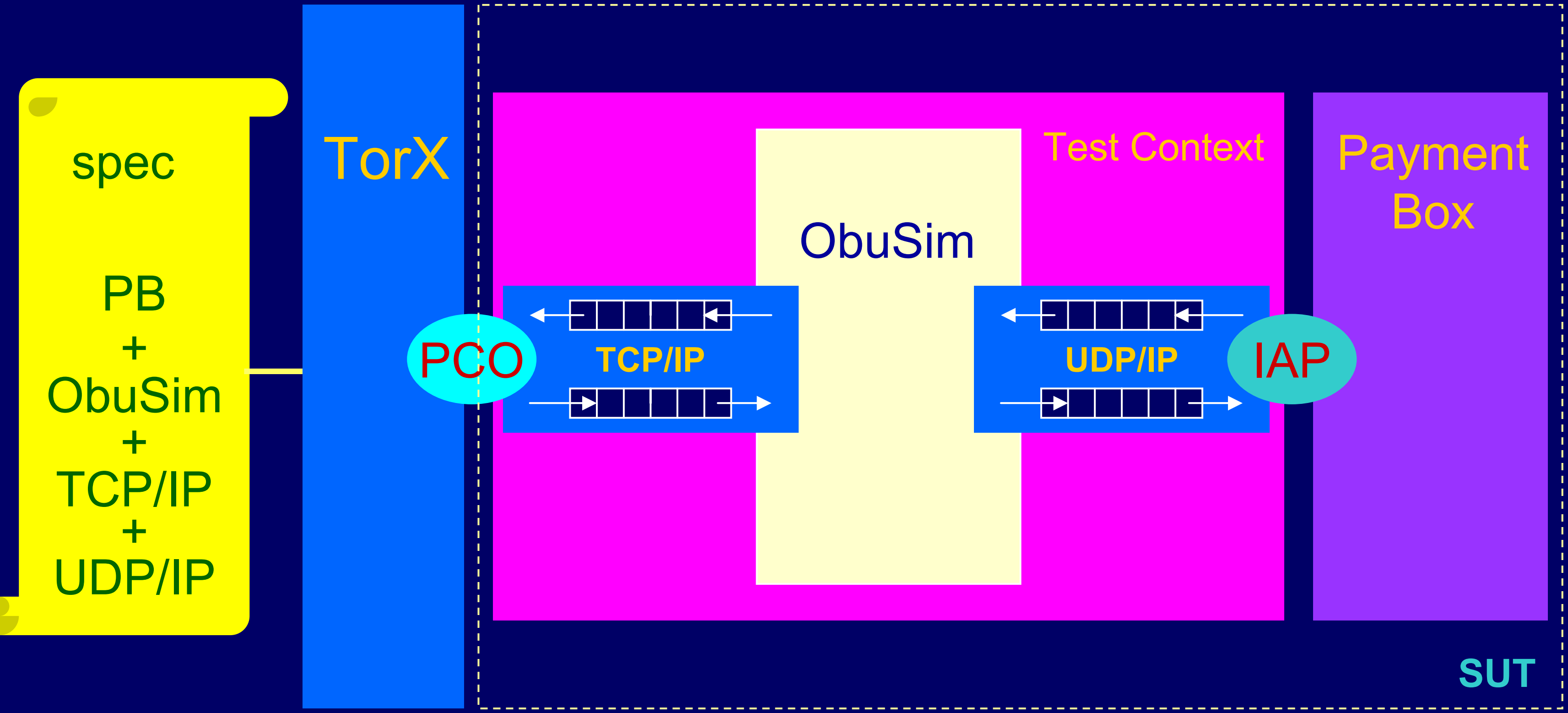
Payment Box

(PB)

Wireless

UDP/IP

Highway Tolling: Test Architecture



Highway Tolling: Results

- Test results :
  - 1 error during validation (design error)
  - 1 error during testing (coding error)
- Automated testing :
  - beneficial: high volume and reliability
  - many and long tests executed (> 50,000 test events)
  - very flexible: adaptation and many configurations
- Step ahead in formal testing of realistic systems

Storm Surge Barrier Control

Oosterschelde Stormvloedkering (OSVK)

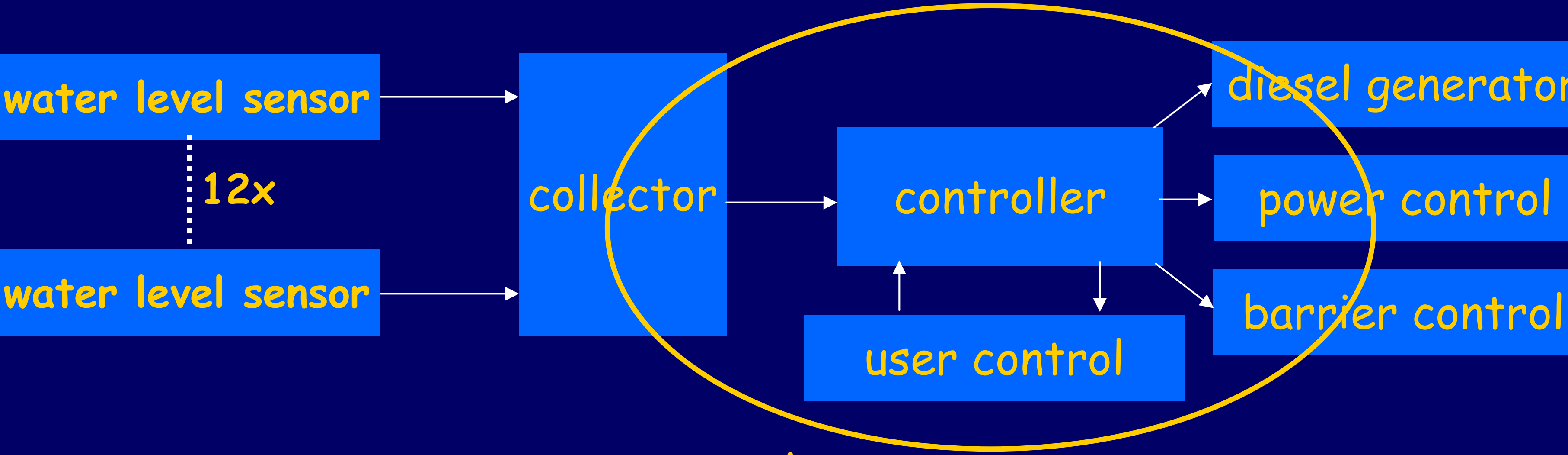
SVKO Emergency Closing System

- Collect water level sensor readings (12x, 10Hz)
- Calculate mean outer-water level and mean inner-water level
- Determine closing conditions

```
 if (closing_condition)
 {notify officials
  start diesel engines
  block manual control
  control local computers}
```

- Failure rate: 10^{-4}/closing event

Testing SVKO



signal wire communication

- test controller (Unix port)
- many timed observations
  - shortest timed delay: 2 seconds
  - longest timed delay: 85 minutes

Results

- real-time control systems can be tested with TorX-technology
  - addition of discrete real time
  - time stamped actions
- quiescence action is not used
  - time spectrum of 3 orders of magnitude
  - deterministic system
- adhoc implementation relation

Overview

- Model-based testing
  - model-driven test generation
  - implementation relations
  - input/output systems, quiescence
- Test generation & execution
  - TorX
  - Demo
  - Case studies
- Current and future developments
  - Testing real-time systems
  - Testing and tolerance
  - Test data generation

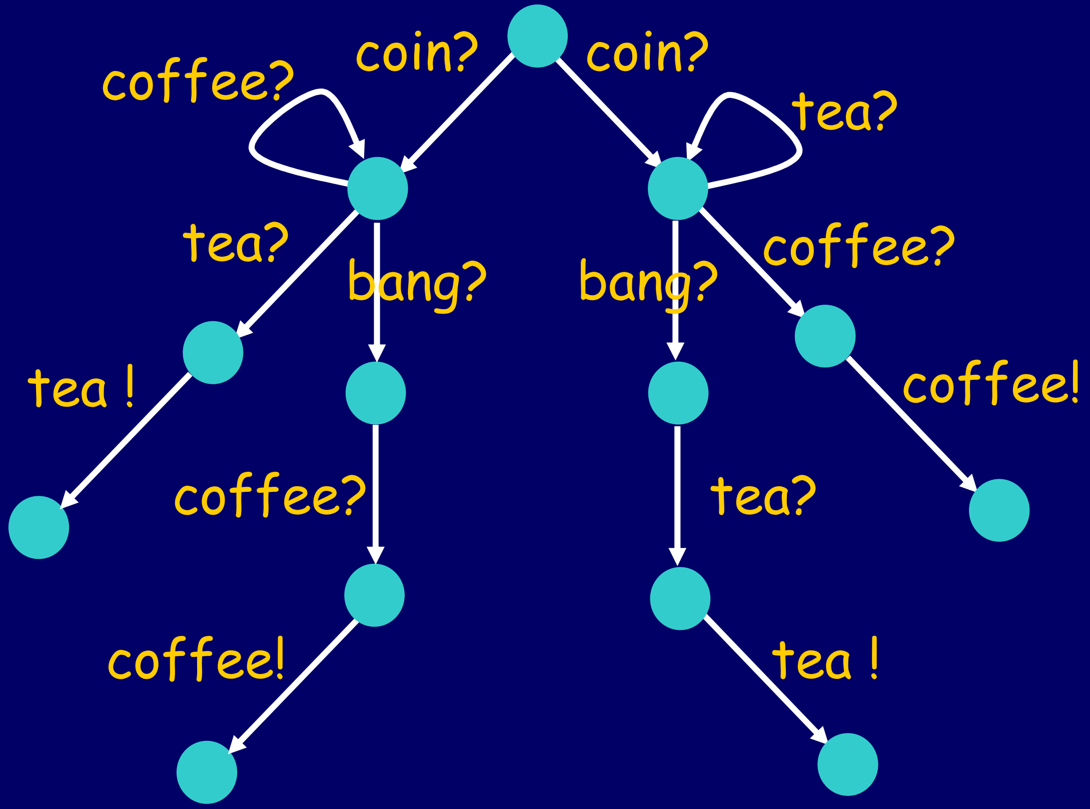
Real-time Testing and I/O Systems

- can the notion of repetitive quiescence be combined with real-time testing?
- is there a well-defined and useful conformance relation that allows sound and (limit) complete test derivation?
- can the TorX test tool be adapted to support real-time conformance testing?

Do We Still Need Quiescence?

Yes!

the example processes should also be distinct in a real-time context



Real-Time and Quiescence

The testing framework can be extended
to real-time processes
if we make an additional assumption:

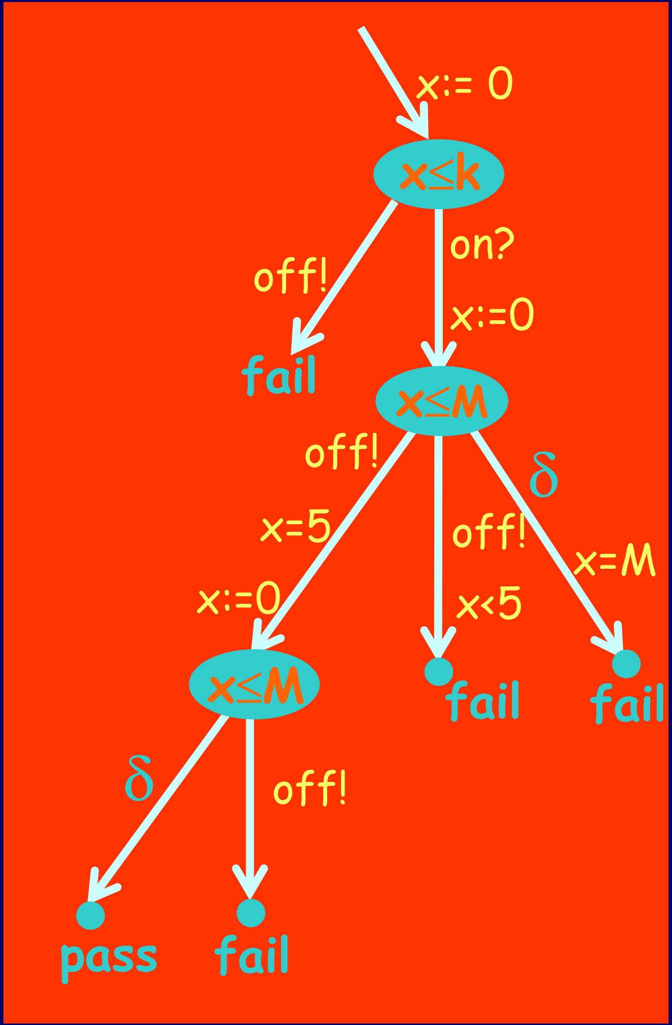
quiescence of implementations is
observable in finite time

i.e. there exists an M>0 such that
for all reachable states s that can be reached
by letting time pass for M time units, s is quiescent

Real-Time Test Cases

Real-time test cases are/have:

- tree-structured
- finite, deterministic
- final states **pass** and **fail**
- from each state ≠ **pass**, **fail**
  - choose input i? and a time k ; apply i? at k, accepting all outputs o! occurring earlier; or
  - or wait for time accepting all outputs o! and δ



Real-Time Test Generation

- the non-timed generation algorithm can be adap
  sound real-time test cases
- test generation is complete

  for every erroneous trace it can generate a

  test that exposes it
- test generation is not limit complete

  because of continuous time there are uncount y n
  traces and only countably many test are gen ated by repeated runs
- test generation is almost limit complete

  repeated test geration runs will eventually generate a test case that will expose one of the non-spurious errors of a non-conforming implementation

non-spurious errors
=
errors with a positive probability of occurring

Current Work

- Extension of the framework
  - M as a function of the specification state/output channel
  - integration with symbolic data generation
  - test action refinement
  - robustness & tolerance in real-time testing
- Extending TorX environment using CORBA IDL
  - generate abstract TorX actions
  - generate TTCN-3 signatures
  - generate adapter code
- Practical application
  - TANGRAM project: testing control software for VLSI lithography machines (ASML)
  - smooth transition between timed & untimed testing

Future Work

- stochastic systems
- quality of service
- hybrid systems
- coverage measures
- integration white/black box spectrum
- ...

For more information

fmt.cs.utwente.nl/research/testing