LEARNING-BASED TESTING: AN INTRODUCTION TO LBTEST

Karl Meinke, CSC School, KTH Stockholm

0. Overview of Talk

1. Introduction to LBTest tool
2. Automotive Case Study: Brake-by-Wire (Volvo)
3. Other case studies
   - portfolio compression service (Tri-Optima)
   - e-commerce access server (SDL)
4. Some Current Research
5. Conclusions

Based on:

L. Feng, S. Lundmark, K. Meinke, F. Niu, M.A. Sindhu, P.Y.H. Wong: *Case Studies in Learning-based Testing*, in Proc. ICTSS 2013

K. Meinke and M. Sindhu: *LBTest: A Learning-based Testing Tool for Reactive Systems* in Proc. ISCT-2013

M. Fisher, *An Introduction to Practical Formal Methods Using Temporal Logic*, Wiley-Blackwell, 2011

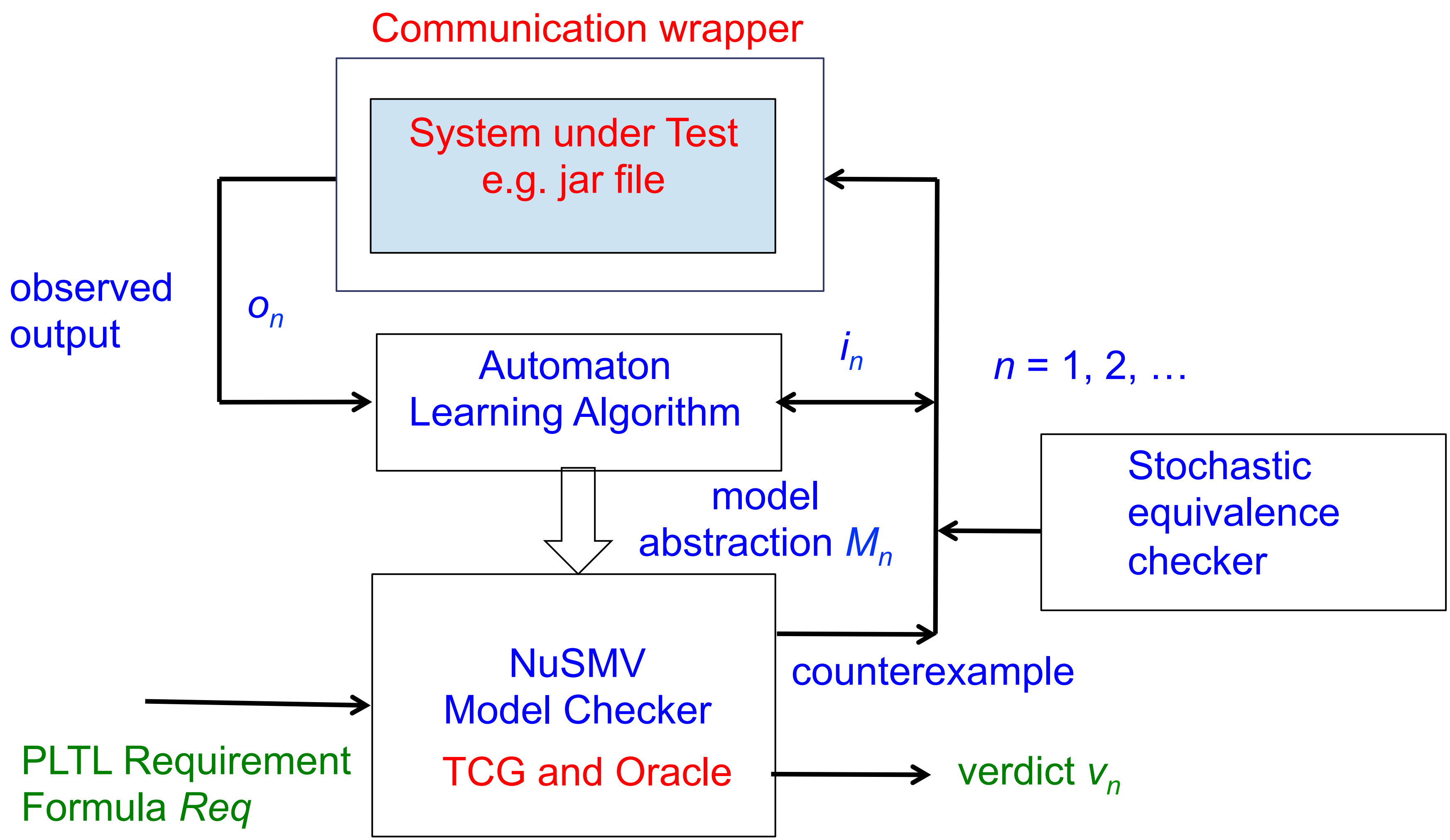
LBTest Tool

- LBTest implements learning-based testing for *embedded and reactive systems* with (more or less) *off-the-shelf components*.

- LBTest implements:
  - Test Case Generation (ATCG)
  - Test execution (online testing)
  - Verdict construction (pass/fail/warning/exception)

- Achieves high state space coverage quite quickly.
- Uses probabilistic convergence, (PAC learning).

LBTest Architecture



Technical & Process Advantages

- Well suited to agile development
- Model is always synchronised to actual code
- No false positives or false negatives due to wrong/ outdated models
- Avoid manual model construction and maintenance

Off-the-shelf Algorithms

- **Learners**
- L*Mealy
- IKL (Incremental Kripke learner) (Meinke, Sindhu 2011)
- (Kearn's algorithm)

- **Model checker**
- NuSMV … BDD and BMC/SAT methods

- **Equivalence checker**
- First / longest / shortest difference

Requirements Modeling

- Modeling reactive systems needs a time concept
- LBTest uses *propositional linear temporal logic* (PLTL)
- PLTL = “Boolean logic + time”
- Conventional model-based testing (conformance testing) is the *next-only part* of PLTL.

- Could interface LTL to *visual requirements modeling* languages
  - Statecharts (conventional MBT)
  - Message Sequence Charts
  - Sequence Diagrams
  - Live sequence charts (Harel)

Linear Temporal Logic LTL (.smv syntax)

- Boolean variables

  A, B, …, X, Y, .. MyVar, ...

- Boolean operators

  !(ϕ), (ϕ & φ), (ϕ | ϕ), (ϕ -> ϕ) …

- Temporal (time) operators

  F(ϕ) (sometime in the future ϕ)

  G(ϕ) (always in the future ϕ)

  (ϕ U φ) (φ holds until ϕ holds)

  X(ϕ) (next ϕ holds)

- Write $X^n(\phi)$ for **X(X(… X(ϕ)))** (ϕ holds in n steps)

Examples

Right now it is Wednesday

Wednesday

Tomorrow is Wednesday

X (Wednesday)

Thursday (always) immediately follows Wednesday

G(Wednesday -> X (Thursday))

Saturday (always) follows Wednesday

G(Wednesday -> F(Saturday))

- Exercise: Define the sequence of days precisely, i.e. just one solution
- Question: Are there any English statements you can't make in LTL?
- Question: Can you express use cases or state machines in LTL?

Safety Properties

- A safety property describes a situation that shall not occur in any state.
- "*Something bad never happens*"
- To verify, all states must be checked exhaustively

- Safety properties usually have the form

$$G\,!\,\varphi$$

where φ defines the "*bad thing*" (invariant)

- Counterexamples (test cases) are finite

Liveness Properties

- A liveness property describes a behavior that must eventually hold on specific execution paths
- "Something good eventually happens"

- Liveness properties often have the form

$$F(\varphi) \quad \text{or} \quad G(\phi \;{-}{>}\; X^n\varphi\;) \quad \text{or} \quad G(\phi \;{-}{>}\; F\varphi\;)$$

where φ describes the "good" thing and ϕ is some event trigger needed for it to occur.

- Counterexamples are usually infinite (why?)
- LBTest performs liveness testing!!

Approximate Models

- Real-world SUTs are *infinite state systems*
- LBTest constructs finite state approximations through *finite partition sets*.
- Example: $\Re$ can be partitioned into
- $\{ x \in \Re : x < 0.0 \}, \{0.0\}, \{x \in \Re : x > 0.0 \}$
- As an input partition we *choose* 3 elements
  - E.g. -100.0, 0.0, 100.0
- As an output partition we *map* outputs to symbolic values
  - negative, zero, positive

- Output partitioning is implemented in the wrapper
- Gives a limited quantifier-free *first-order extension* to PLTL.

Verdict Construction (Oracle step)

- On-the-fly verdict construction
- Compares two behaviours:

(1) a predicted (bad) behaviour in model

(2) an observed behaviour in SUT

  - Prediction == Observation -> Fail/Warning
  - Prediction != Observation -> Pass
  - No Observation -> Exception/Timeout error

2. Recent Case Studies

- Does LBT actually work?

- Can we make a simple tool with off-the-shelf algorithms and components?

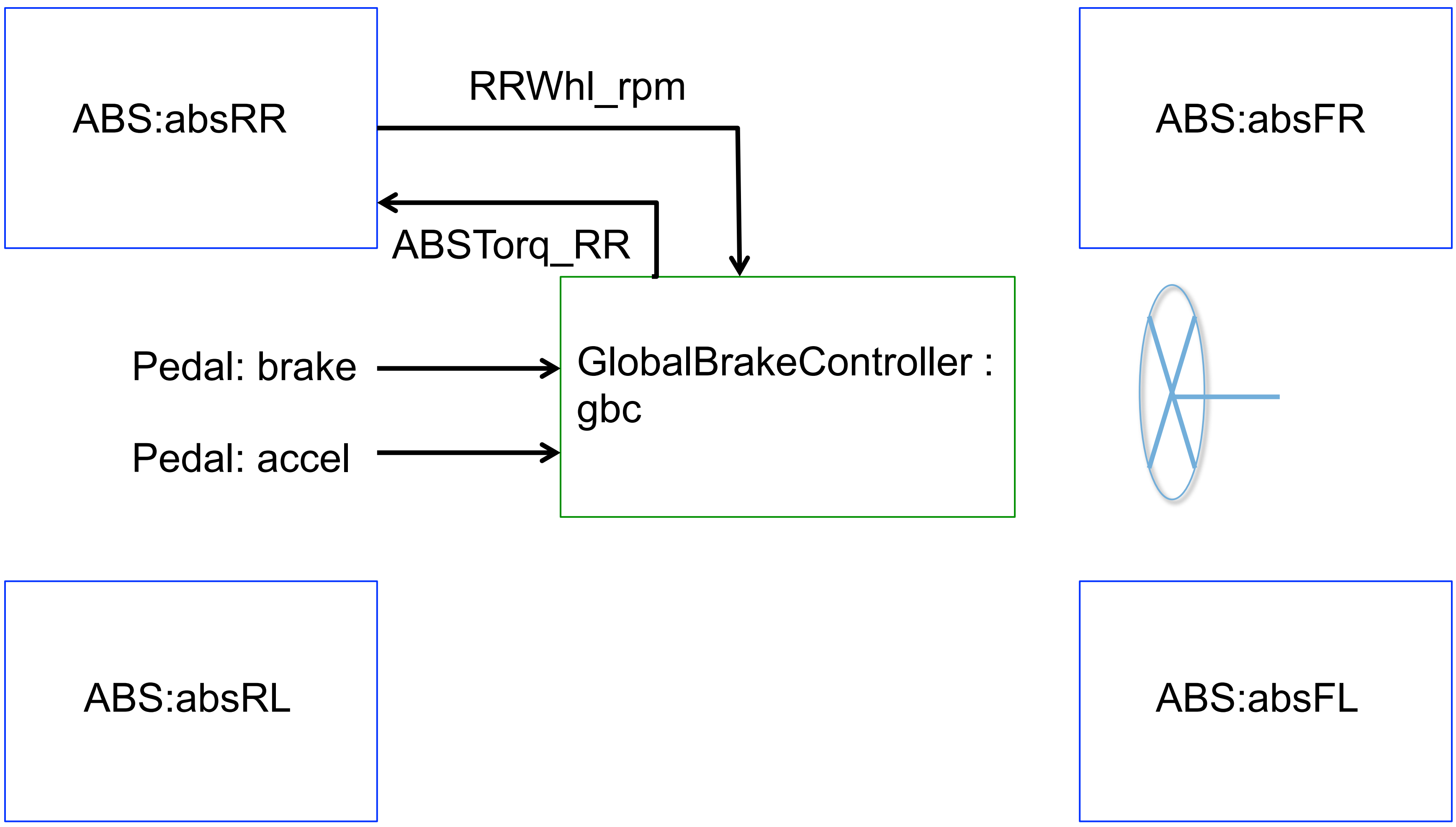
- How does LBT scale to large real-world systems?
- Where are the bottlenecks?
  - Learning?
  - Model checking?
  - Equivalence checking?
  - SUT?

- Can temporal logic be used in real-life?
- Pedagogical examples – technology uptake

3. Brake-by-Wire (BBW) Case Study

- A Case Study with Volvo
- From ARTEMIS project MBAT
- Joint work with Volvo

- BBW is a distributed system of 5 ECUs
- 4 ABS ECUs (1 per wheel)
- 1 central controller with brake/accelerator inputs
- Controller calculates specific brake torque requests to each wheel ABS in real-time
- Floating point data types need partitioning



BBW Architecture

System variables (5 and 20 ms clocks)

25 floating point registers:

0: driverBrake;

1: GlobalTorque;

2-5: RRWhl_rpm, RLWhl_rpm, FRWhl_rpm, FLWhl_rpm;

6-9: RRWhl_torq, RLWhl_torq, FRWhl_torq, FLWhl_torq;

10: Veh_Spd_Est;

11-14: ABSTorq_RR, ABSTorq_RL, ABSTorq_FR, ABSTorq_FL;

15: Veh_Spd_Real;

16: AccPedalPos;

17-20: estimated SlipRate of four wheels

21-24: real slip rate of four wheels.

Fourteen Black-box Requirements

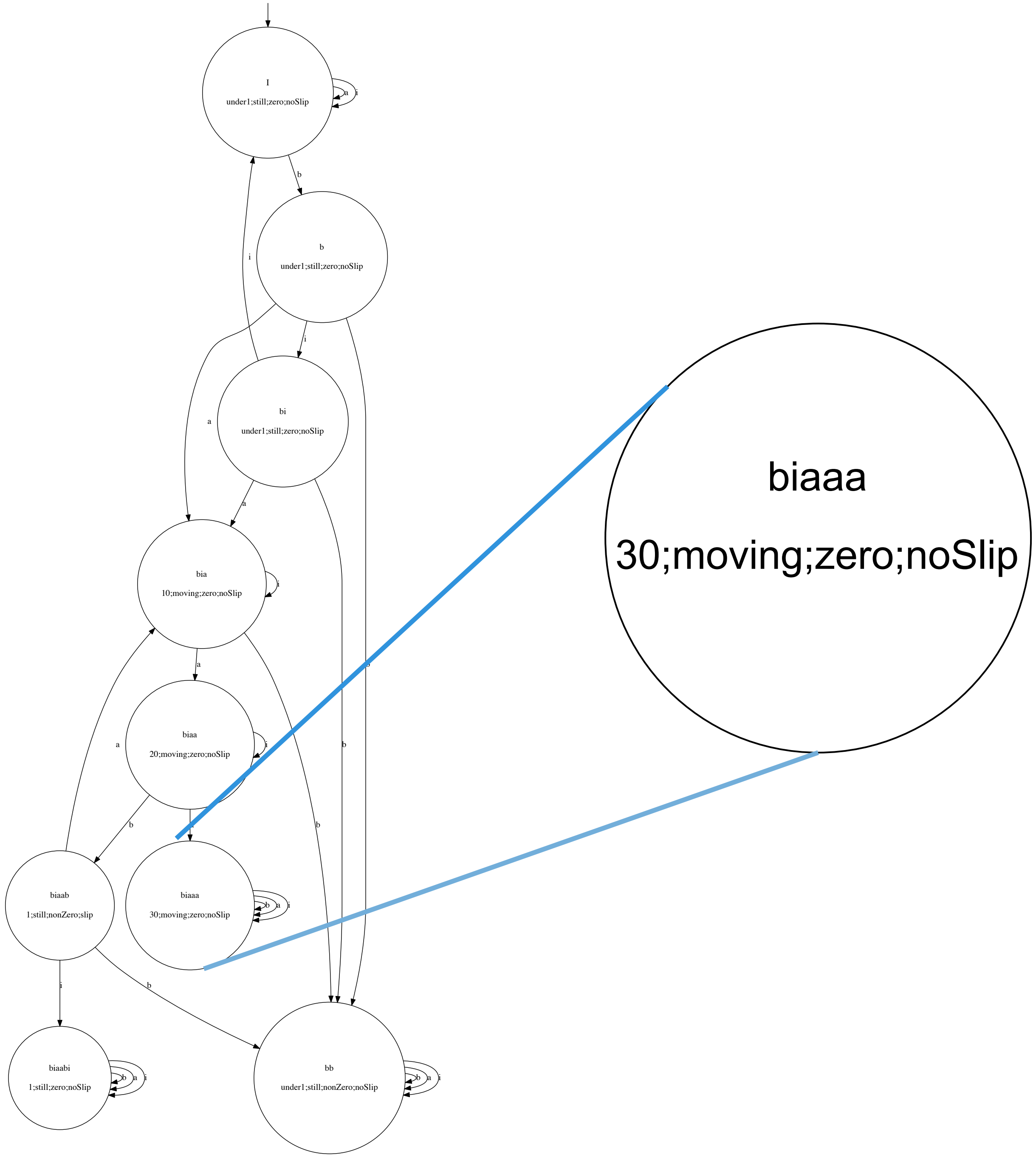
REQ-4 If the brake pedal is pressed and the actual speed of the vehicle is larger than 10 km/h and the slippage sensor shows that the (front right) wheel is slipping, this implies that the corresponding brake torque at the (front right) wheel should be 0.

PLTL modeling

9 of 14 Volvo requirements could be modeled in PLTL

REQ-4 G(input = brake & motion = moving & slipRR = slip -> torqueRR = zero)

Model #3
after 400 msec



Other Industrial Case Studies

Portfolio Compression Software (Finance)

A compression cycle has 4 stages:

Preparation

Sign up

Linking

Live execution

619,000 LoC (Python including large dependencies like Django).
100+ databases! Reset was expensive!

Tested authentication and authorization features.

Requirement 2: If Bank A is not logged in, and does log in, then Bank A should become logged in.

Requirement 3: Cycle signup should be prohibited until a bank adheres to the protocol.

Requirement 5: If bank A adheres to the protocol, then cycle signup for bank A should always be allowed.

Requirement 5, was tested with two 7 hour testing sessions. Both terminated with a “pass” verdict after about 86000 SUT executions and hypothesis sizes of up to 503 states. The log files were manually checked and contained no errors.

Distributed Access Server (FAS)

- Distributed, concurrent OO system developed by web company that provides search and merchandising services
- Developed and evolved over 12 years. Its various modules have been tested with automated and manual techniques.
- SUT was implementation of the SyncClient, 6400 LoC (Java), 44 classes and 2 interfaces
- Tested interaction between a SyncClient and a ClientJob

- 11 user requirements could be expressed in LTL

- **Requirement 8**: If it is not in the End state then every schedule that the SyncClient possesses will eventually be executed as a replication job.

- **Requirement 9**: The SyncClient cannot modify its underlying file system (files =readonly) unless it is in state WorkOnReplicate .

- All requirements passed except #8 and #9.
- #9 was a requirement error (U replaced by W)
- #8 was a true negative.

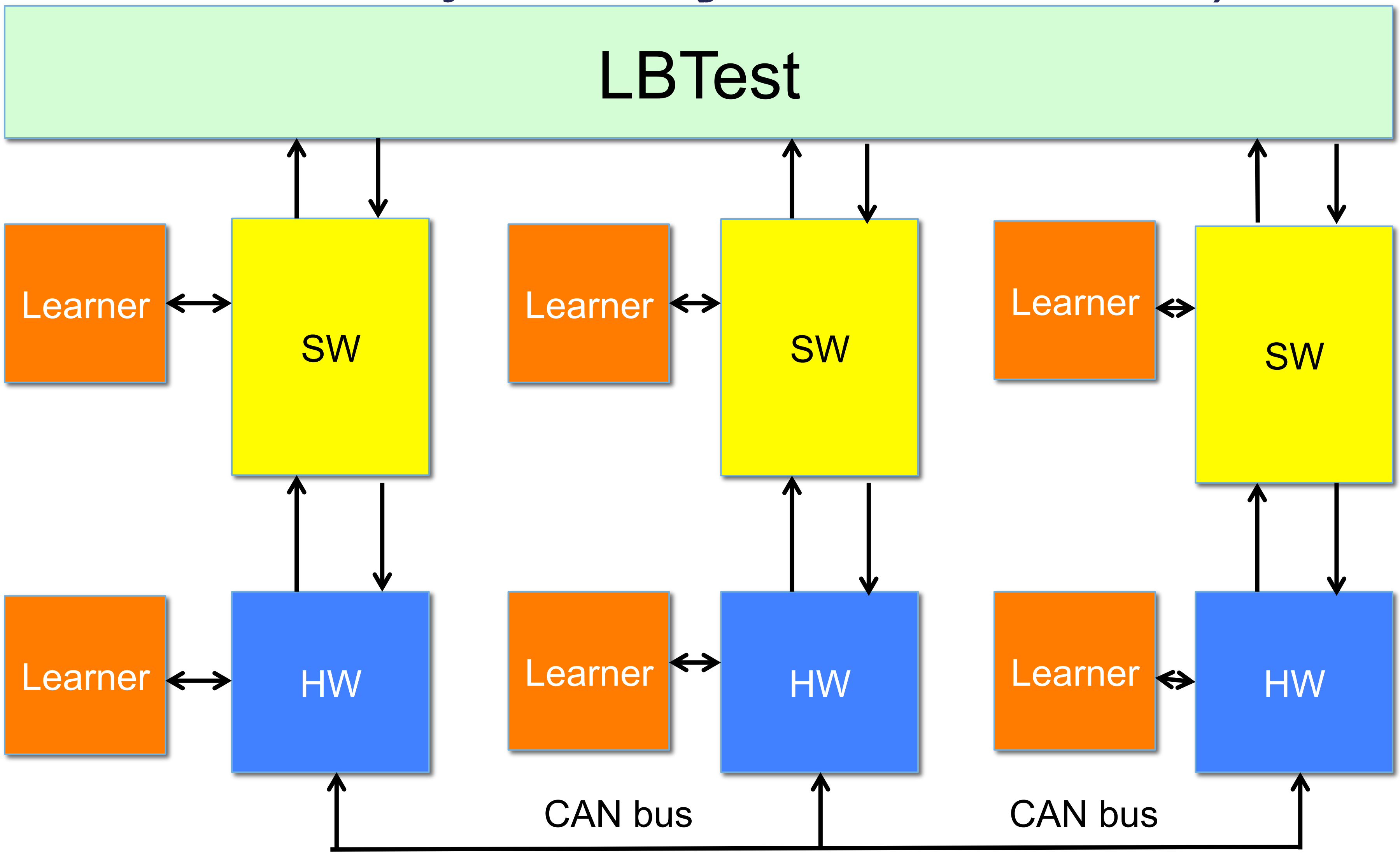
Some Current Research

- *Software hardware co-testing* with virtualised hardware
- Joint with Scania and Hojat Khosrowjerdi
- Motivated by ISO 26262 standard

- *Distributed systems fault injection* from LBTest
- Joint with Tri-Optima and Peter Nycander

- *Testing avionics mode systems*
- Joint with SAAB Aerospace and Sebastian Stenlund

Software/hardware co-testing for distributed systems (joint with Scania)



Conclusions

- LBTest found errors in all 3 industrial case studies
- Worked across a range of industrial domains

- Repeating these experiments today leads to much better performance. Not yet reached theoretical limits.

- **Future research**
- More efficient learning / model checking
- Parallel testing
- Virtualised Environments

Configuration File (Server side ADTs)

output_types = [speed, motion, torqueRR, slipRR];

output_values = { under1:speed, 1: speed, 10:speed, 20:speed, 30:speed, 40:speed, 50:speed, 60:speed, 70:speed, 80:speed, 90:speed, 100:speed, 110:speed, over120:speed,

still: motion, moving: motion,

zero: wheelRotateRR, nonZero: wheelRotateRR,

zero: torqueRR, nonZero: torqueRR,

slip: slipRR, noSlip: slipRR };

inputs = { a=acc, b=brake, i=idle };

SUT Wrapper code (client side)

```
if( inChar == 'a’ ) {              // full accelerate
        register[0] = 0.0;         // brake pedal
        register[16] = 100.0;      // gas pedal

} else if ( inChar == ’b’ ){       // full brake
        register[0] = 100.0;
        register[16] = 0.0;

} else if ( inChar == 'i’ ) {      // idle
        register[0]  = 0.0;
        register[16] = 0.0;
}
```

SUT Wrapper code (client side)

```
if ( Veh_Spd_Real > 10.0 ) { dOut[1] = "moving”; }
else { dOut[1] = "still”; }

if ( ABSTorq_RR > 0.0 ) { dOut[2] = "nonZero”; }
 else { dOut[2] = "zero”; }

if ( slipRateRR > 0.2 ) { dOut[3] = "slip”; }
else { dOut[3] = "noSlip”; }

if ( 120.0 <= Veh_Spd_Real ) dOut[0] = "over120";
```