

QuickCheck at Work

John Hughes



Exercises → Practice

Small scale → Large scale

Property-driven development → Testing legacy code

Trivial inputs → Complex inputs

- Changes the balance of effort



Example: Ericsson Media Proxy

Lots of work to write generators

Megaco response

Many, many parameters, can be 1—2 *pages* per message!

State machine models fit the problem well

CHALMERS

QuviQ

A Media Proxy Bug


- Test adding and removing callers from a call

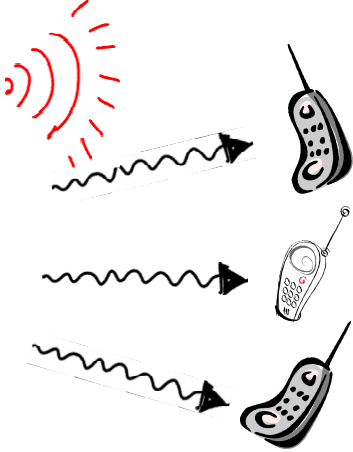
Call Full

CHALMERS

QuviQ

3G Radio Base Station







Setup →

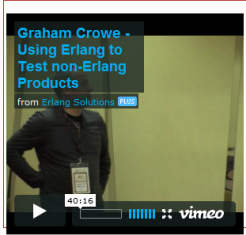
← OK

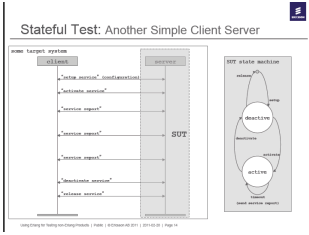
Setup →


← Reject

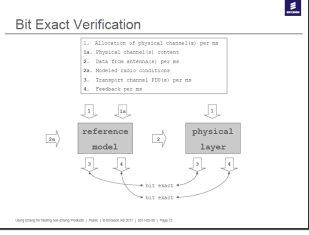




4G LTE Radio Base Stations







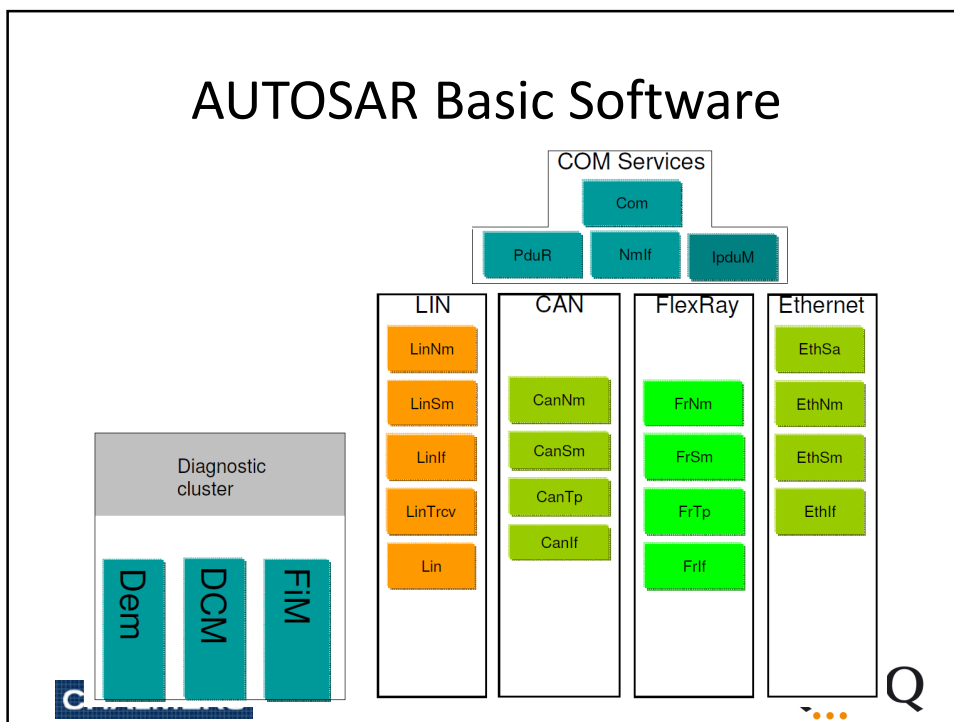






Google:
Graham Crowe,
Erlang Factory



Theory

Car manufacturers should be able to buy code from different providers and have them work seamlessly together

CHALMERS

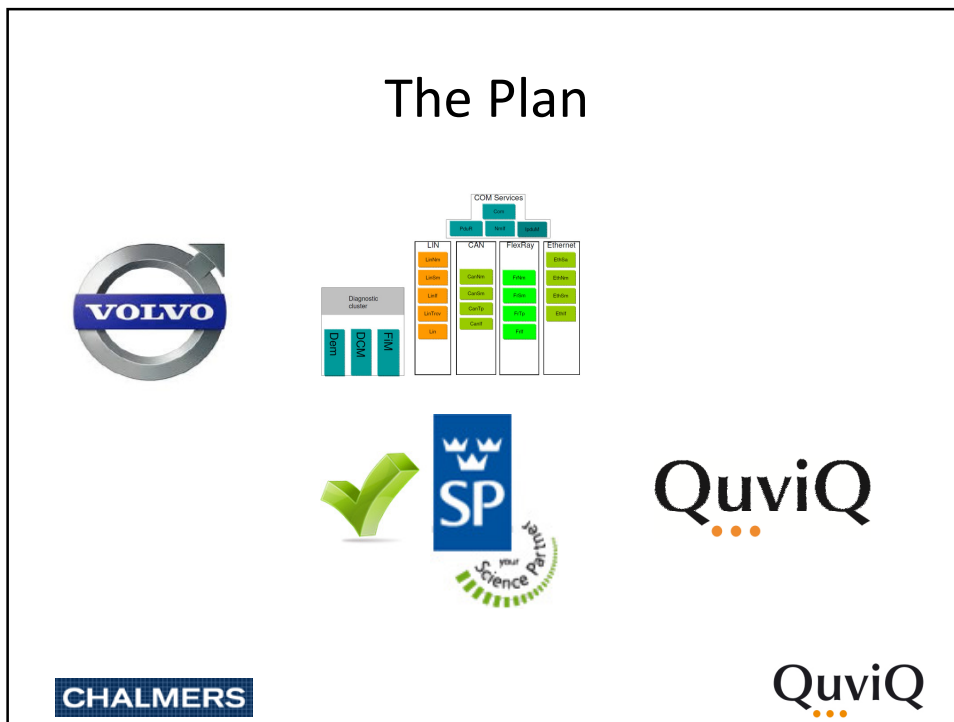
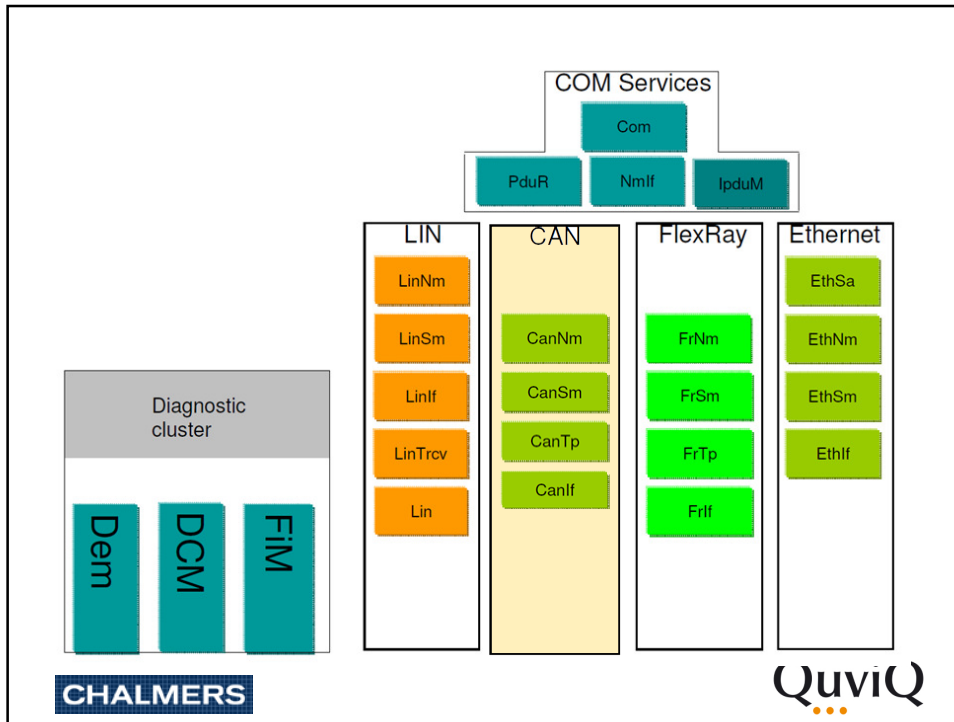
QuviQ

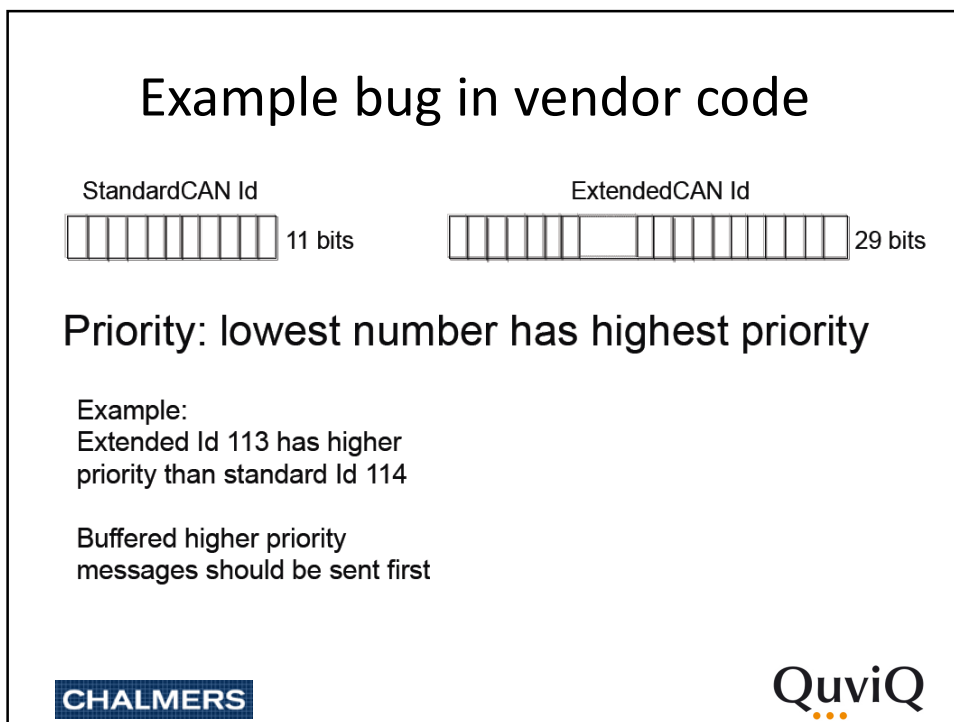
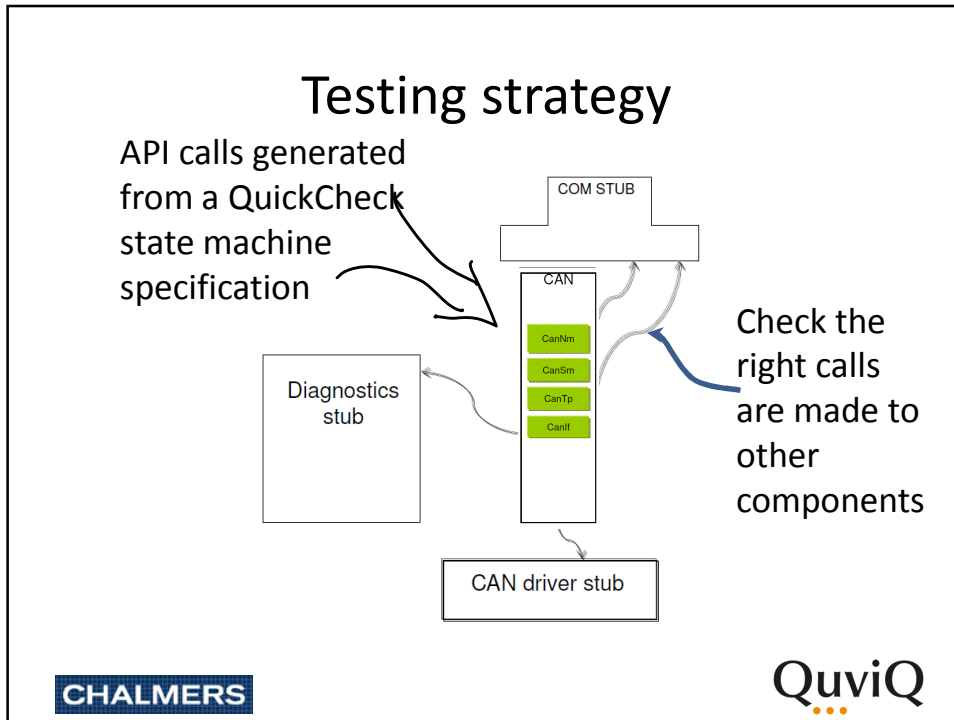
Practice

VOLVO's experience has been that this is often not the case

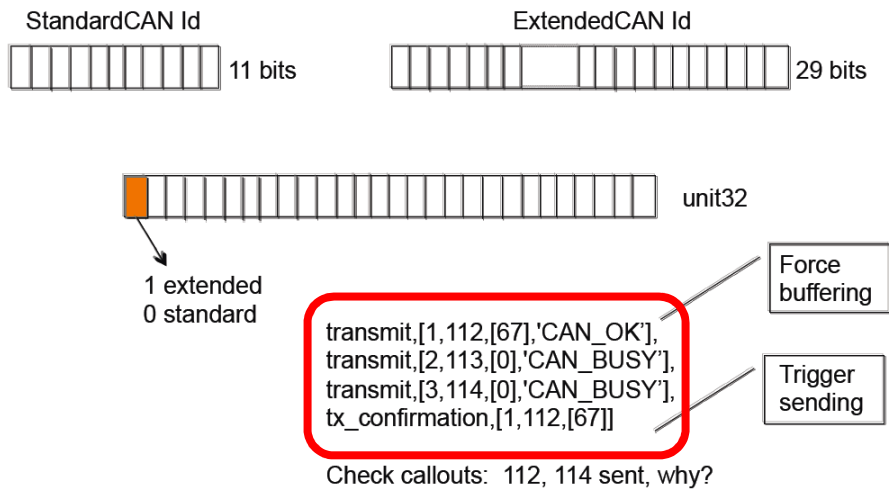
CHALMERS

QuviQ





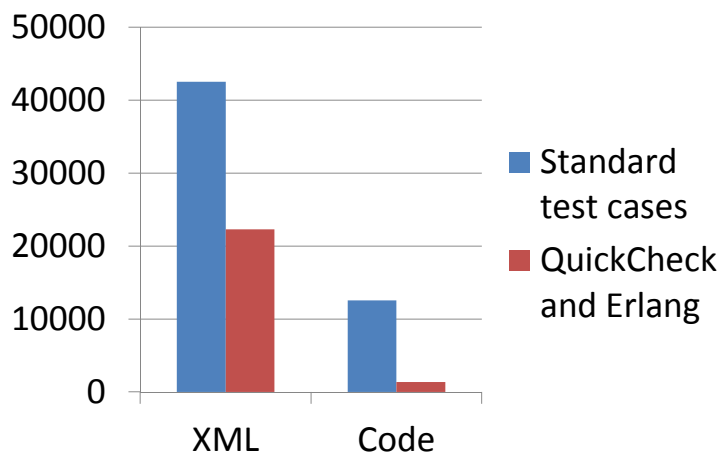
Example bug in vendor code



CHALMERS

QuviQ

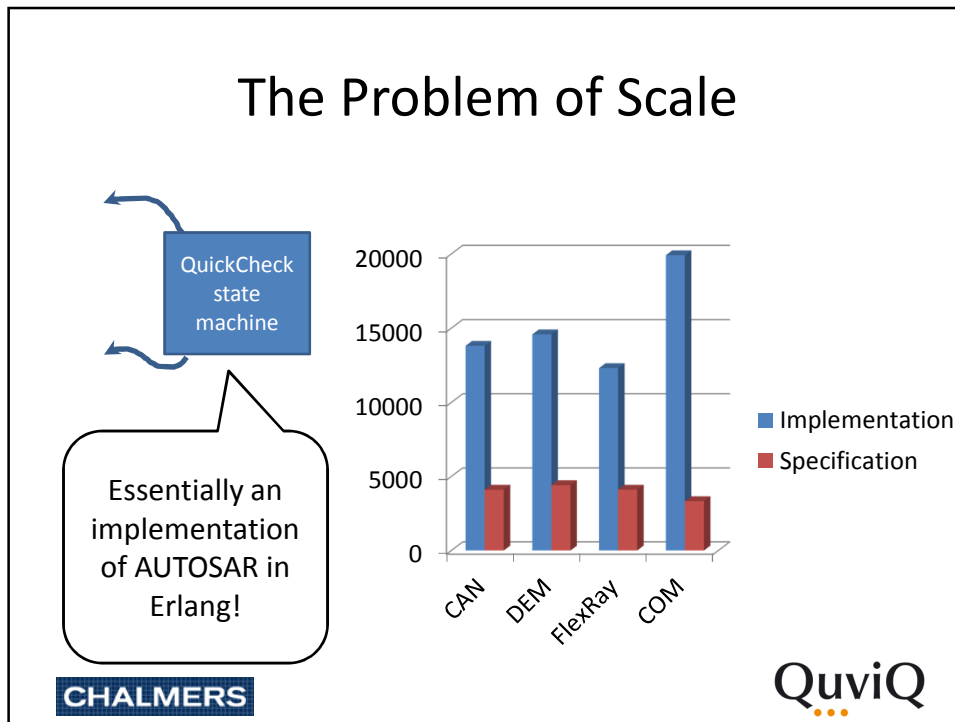
Test code for Flexray Interface



- The code is 16x smaller!

CHALMERS

QuviQ



Results...

- Testing code from 6 suppliers
- **200+ issues identified**
- **100+ bugs identified in the standard**

"We know there is a lurking bug somewhere in the dets code. We have got 'bad object' and 'premature eof' every other month the last year. We have not been able to track the bug down since the dets files is repaired automatically next time it is opened."

Tobbe Törnqvist, Klarna, 2007

CHALMERS

QuviQ

What is it?

300
people in
5 years



 **klarna**

Invoicing services for web shops

Distributed database:
transactions, distribution,
replication

Tuple storage

Race
conditions?

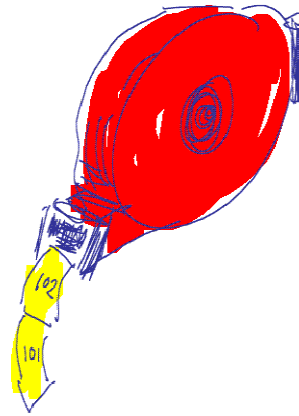
CHALMERS

QuviQ

Imagine Testing This...

dispenser:take_ticket()

dispenser:reset()



CHALMERS

QuviQ

A Unit Test in Erlang

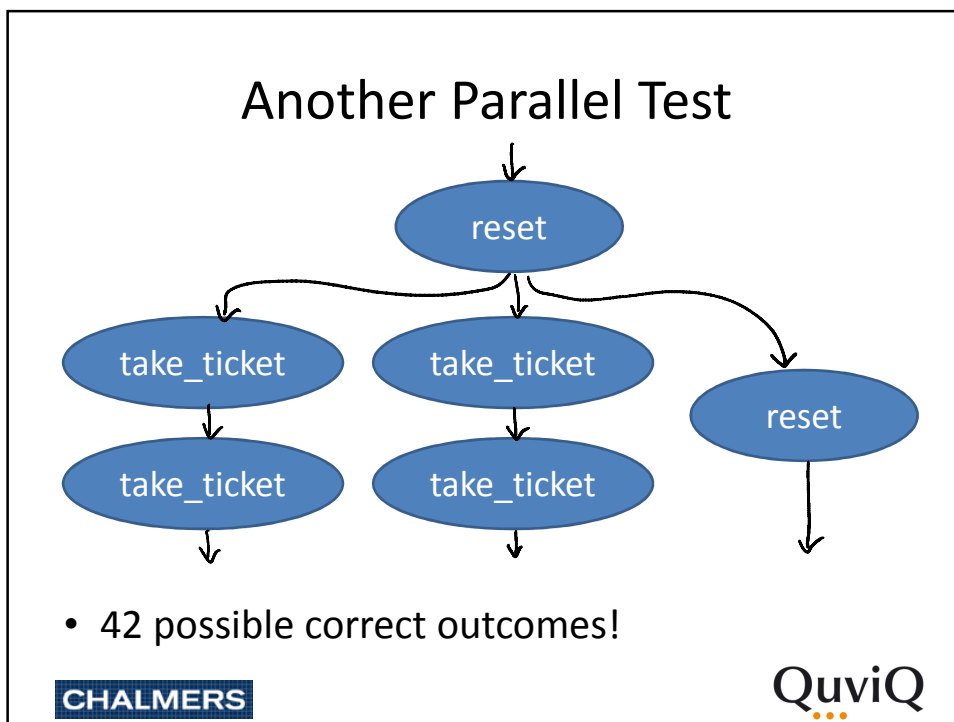
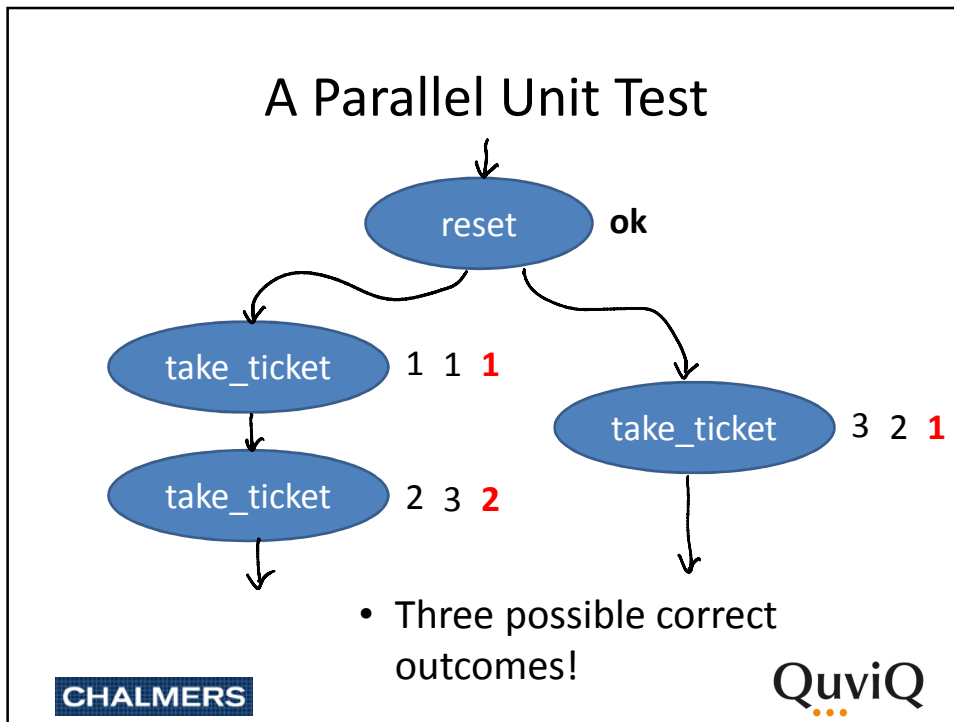
```
test_dispenser() ->
  ok = reset(),
  1 = take_ticket(),
  2 = take_ticket(),
  3 = take_ticket(),
  ok = reset(),
  1 = take_ticket().
```

Expected
results

BUT...

CHALMERS

QuviQ



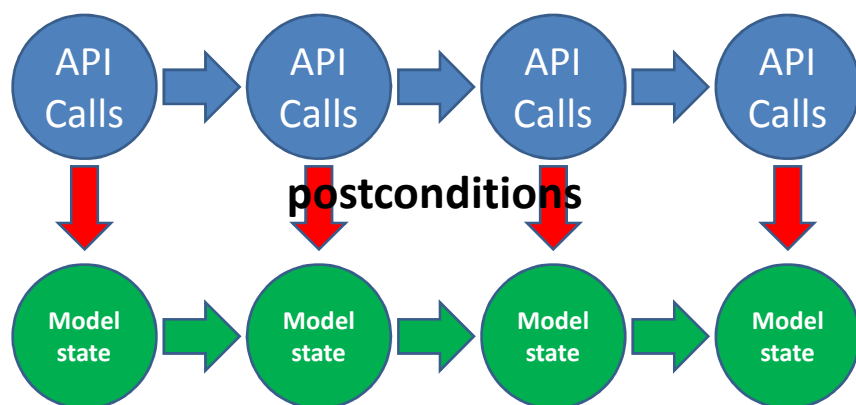
Property-Based Testing to the rescue!

- Enumerating possible results is impractical!
- Only *properties* of the correct results make sense
 - e.g. `sort([A,B,C]) == [1,2,3]`
- Reuse *sequential models* to decide if parallel tests pass

CHALMERS

QuviQ

Sequential testing...



CHALMERS

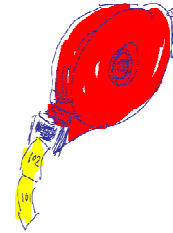
QuviQ

The Model

- Just an integer!

- State transitions

```
next_state(S, _V, {call, _, reset, _}) ->
    0;
next_state(S, _V, {call, _, take_ticket, _}) ->
    S+1.
```



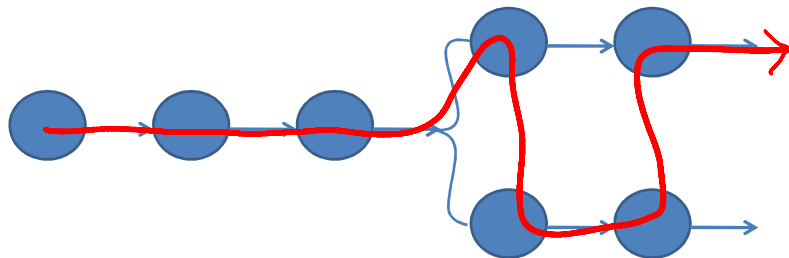
- Postconditions

```
postcondition(S, {call, _, take_ticket, _}, Res) ->
    Res == S+1;
```

CHALMERS

QuviQ

Parallel Test Cases



- Use the *same* model!

CHALMERS

QuviQ

Generate parallel test cases


```

prop_parallel() ->
  ?FORALL(Cmds, parallel_commands(?MODULE),
    begin
      start(),
      {H, Par, Res} =
        run_parallel_commands(?MODULE, Cmds),
      Res == ok
    end) .

```

Run tests, check for a matching serialization

CHALMERS



Prefix:

Parallel:

1. take_ticket() --> 1
2. take_ticket() --> 1


Result: no_possible_interleaving

```

take_ticket() ->
  N = read(),
  write(N+1),
  N+1.

```

CHALMERS



dets

- Tuple store:
 {Key, Value1, Value2...}
- Operations:
 - insert(Table,ListOfTuples)
 - delete(Table,Key)
 - insert_new(Table,ListOfTuples)
 - ...
- Model:
 - List of tuples

200 LOC
vs.
6.3 KLOC

CHALMERS

QuviQ

Bug #1

`insert_new(Name, Objects) -> Bool`

Prefix:

`open_file(dets`

Parallel:

1. `insert(dets_t`

2. `insert_new(dets_table, []) --> ok`

Result: `no_possible_interleaving`

Types:

`Name = name()`

`Objects = object() | [object()]`

`Bool = bool()`

CHALMERS

QuviQ

Bug #2

Prefix:

```
open_file(dets_table, [{type, set}]) --> dets_table
```

Parallel:

```
1. insert(dets_table, {0,0}) --> ok
```

```
2. insert_new(dets_table, {0,0}) --> ...time out...
```



=ERROR REPORT===== 4-Oct-2010::17:08:21 ===

** dets: Bug was found when accessing table dets_table

CHALMERS

QuviQ

Bug #3

Prefix:

```
open_file(dets_table, [{type, set}]) --> dets_table
```

Parallel:

```
1. open_file(dets_table, [{type, set}]) --> dets_table
```

```
2. insert(dets_table, {0,0}) --> ok
```

```
get_contents(dets_table) --> []
```

Result: no_possible_interleaving



CHALMERS

QuviQ

Bug #4

Prefix:

```
open_file(dets_table, [{type,bag}]) --> dets_table
close(dets_table) --> ok
open_file(dets_table, [{type,bag}]) --> dets_table
```

Parallel:

1. lookup(dets_table,0) --> []
2. insert(dets_table,{0,0}) --> ok
3. insert(dets_table,{0,0}) --> ok

Result: ok

premature eof

CHALMERS

QuviQ

Bug #5

Prefix:

```
open_file(dets_table, [{type,set}]) --> dets_table
insert(dets_table, [{1,0}]) --> ok
```

Parallel:

1. lookup(dets_table,0) --> []
delete(dets_table,1) --> ok
2. open_file(dets_table, [{type,set}]) --> dets_table

Result: ok
false

bad object

CHALMERS

QuviQ

"We know there is a lurking bug somewhere in the dets code. We have got 'bad object' and 'premature eof' every other month the last year."

7. The Test at Klarna, 2007

Each bug fixed the day
after reporting the
failing case

CHALMERS

QuviQ

How come?

- dets is *mature software* that has been heavily used in production for >15 years
- Race conditions are *hard* to write test cases for
 - So people don't!
 - Usually left until *integration testing*
- If it's not tested, why should it work?

CHALMERS

QuviQ

Property-Based Testing

==

Lightweight Formal Methods

- Formal Methods can bring great benefits, but are *too costly* or *impractical* in many situations, because proofs are so very hard
- PBT lets you *use* formal methods
 - Write a formal spec, and check it against the code!
 - Cheaply—often *cheaper* than ordinary testing!
- It's working, at Ericsson, Basho, Motorola, ...

CHALMERS

QuviQ

The Initial Phases

- Lots of work to develop specification
 - Understanding and generating test inputs
- Many errors to fix in the *specification*, due to...
 - New code is buggy
 - Misunderstandings of the informal spec
 - Undocumented features of the system
 - Undocumented *limitations* of the system
 - "happy case" programming

CHALMERS

QuviQ

Making Progress

- QuickCheck tends to find the *same* problem in every run
 - There is a "most likely bug"
 - Other bugs usually *shrink* to the most likely one
- To make progress, the most likely bug must be excluded
 - *Bug preconditions* document the limitations of the system

CHALMERS

QuviQ

The Payoff

- Once the spec is corrected, and limitations accounted for, *real* bugs start to appear
- Each extension to the spec yields a *non-linear* improvement in the variety of tests
- The *same* spec can find many, many bugs

CHALMERS

QuviQ

Property-Based Testing Makes Quality Affordable

CHALMERS

QuviQ