Property-Based Testing with QuickCheck

John Hughes

CHALMERS QuviQ

Why is testing hard?

$O(n^3)$ test cases

3—4 tests per pairs of features

Don’t write tests!

Generate them

QuickCheck

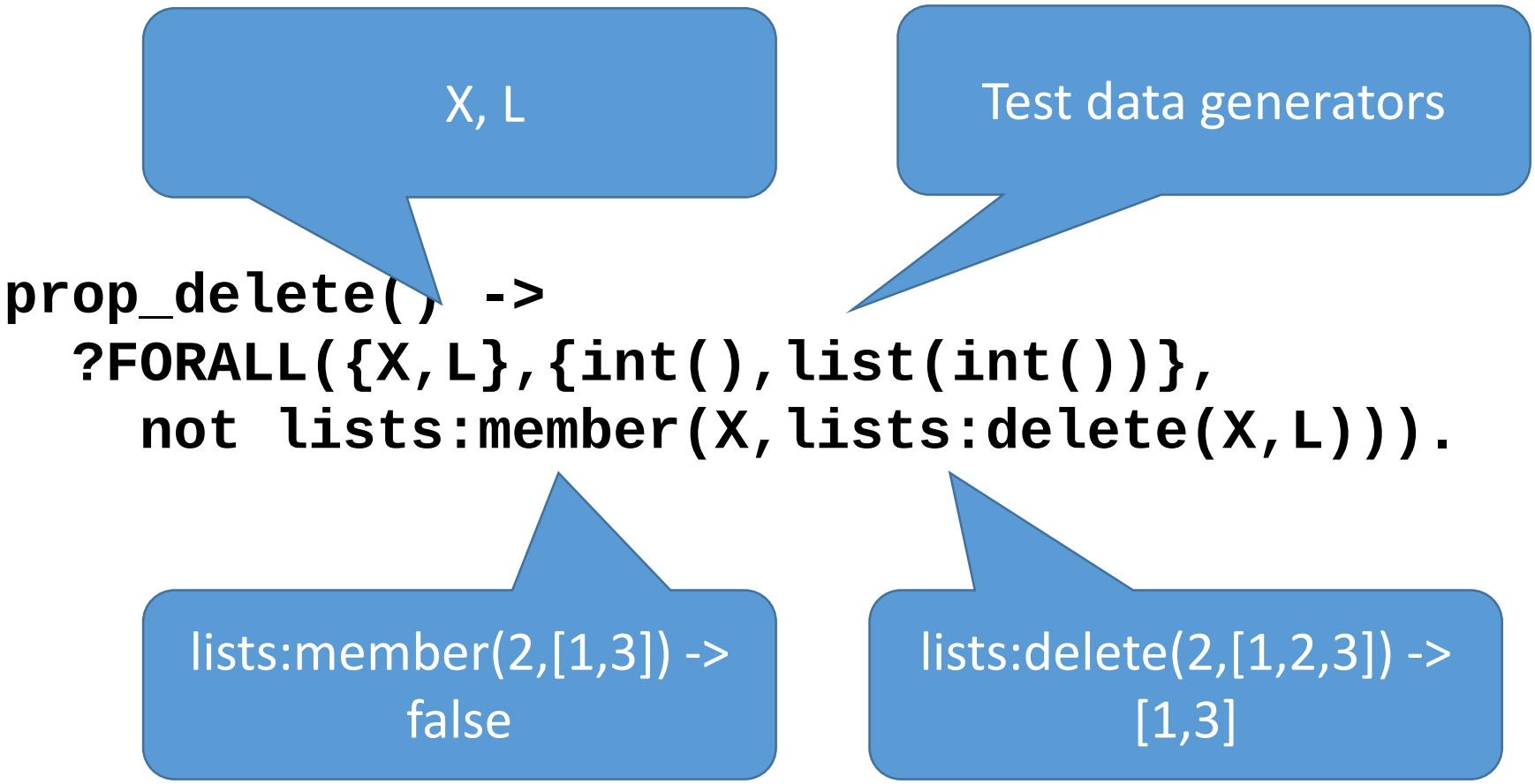
1999—invented by Koen Claessen and myself, for Haskell

2006—Quviq founded marketing Erlang version

Many extensions

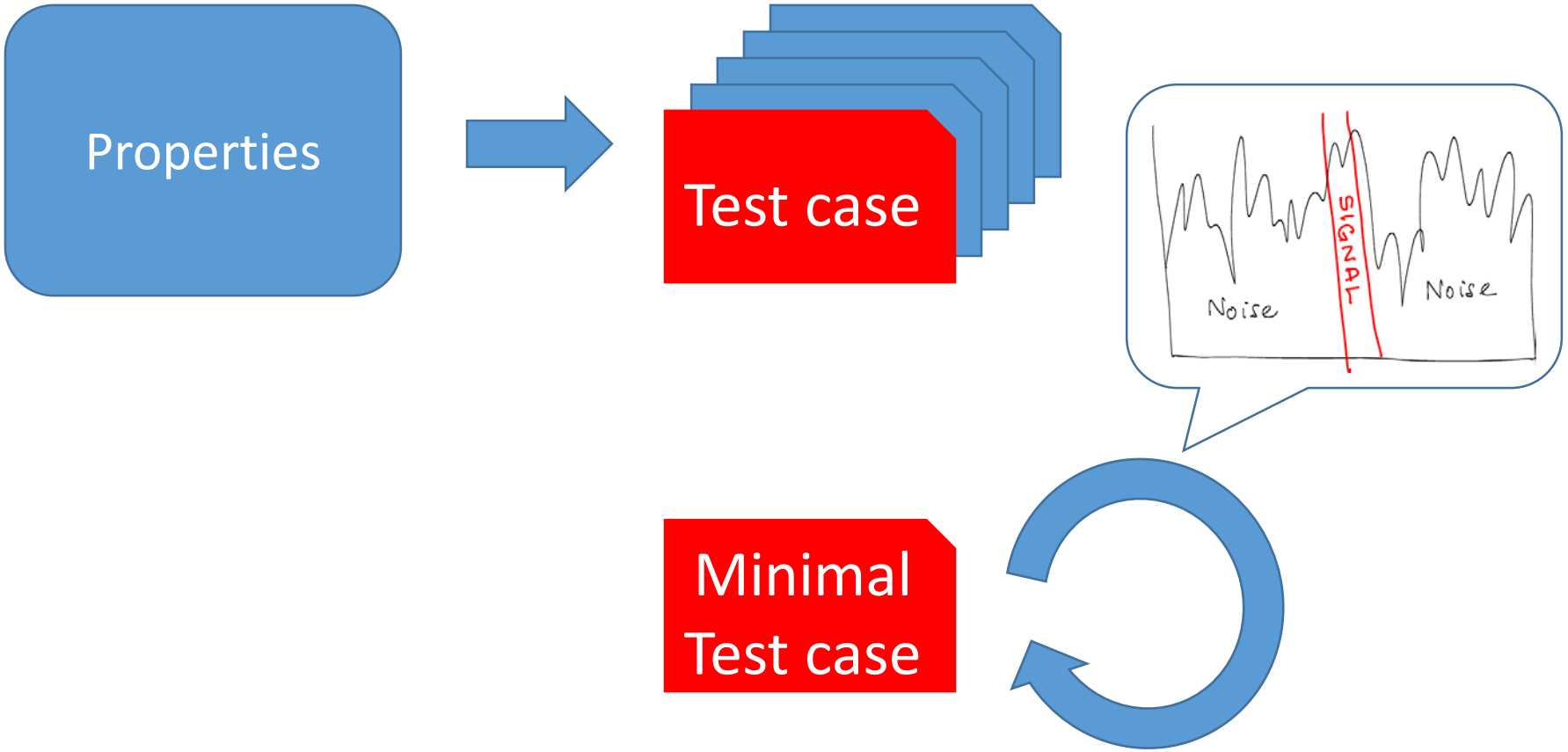
Finding deep bugs for Ericsson, Volvo Cars, Basho, etc…

Example: deletion from a list



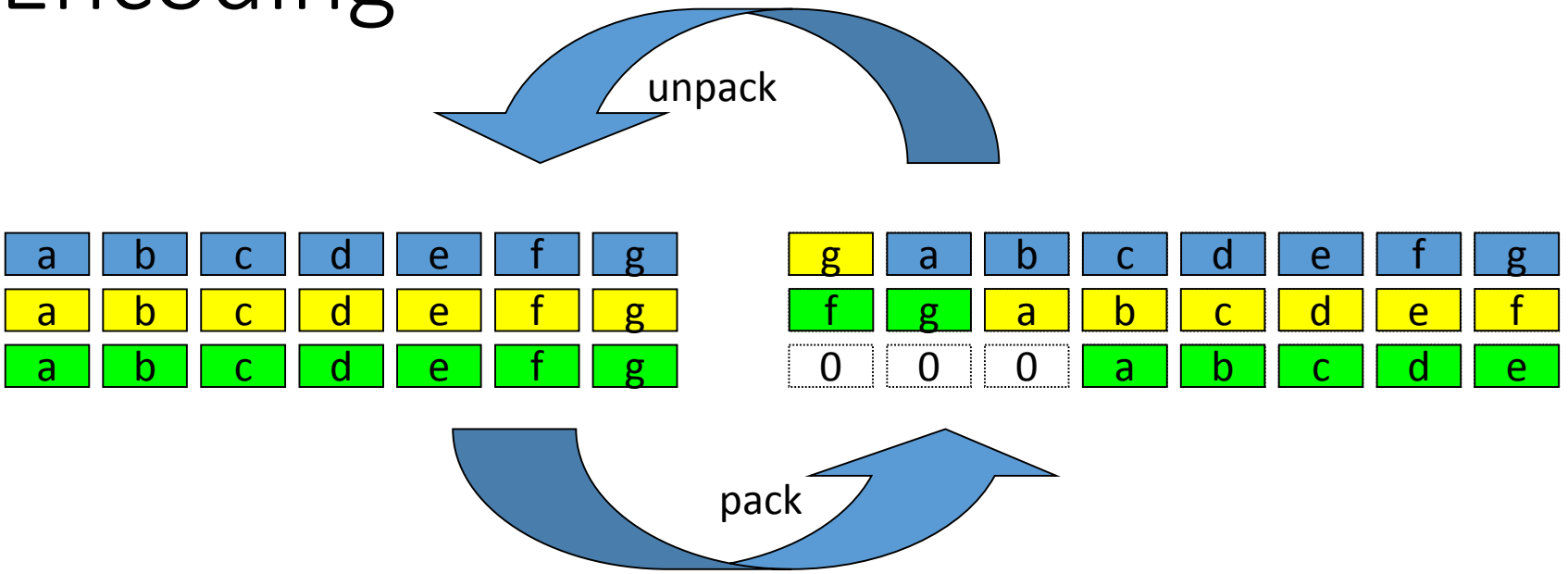
Let’s run some tests…

Property Based Testing



How should we test lists:delete?

Example: GSM Text Message Encoding



Test suite

```
test(S) ->
    T=unpack(pack(S)),
    io:format("unpack(pack(~p)) = ~p~n",[S,T]),
    S=T.

test() ->
    test(""),
    test("1"),
    test("12"),
    test("123"),
    test("1234"),
    test("12345"),
    test("123456"),
    test("1234567"),
    test("12345678"),
    test("123456789"),
    test("1234567890").
```

Round trip property

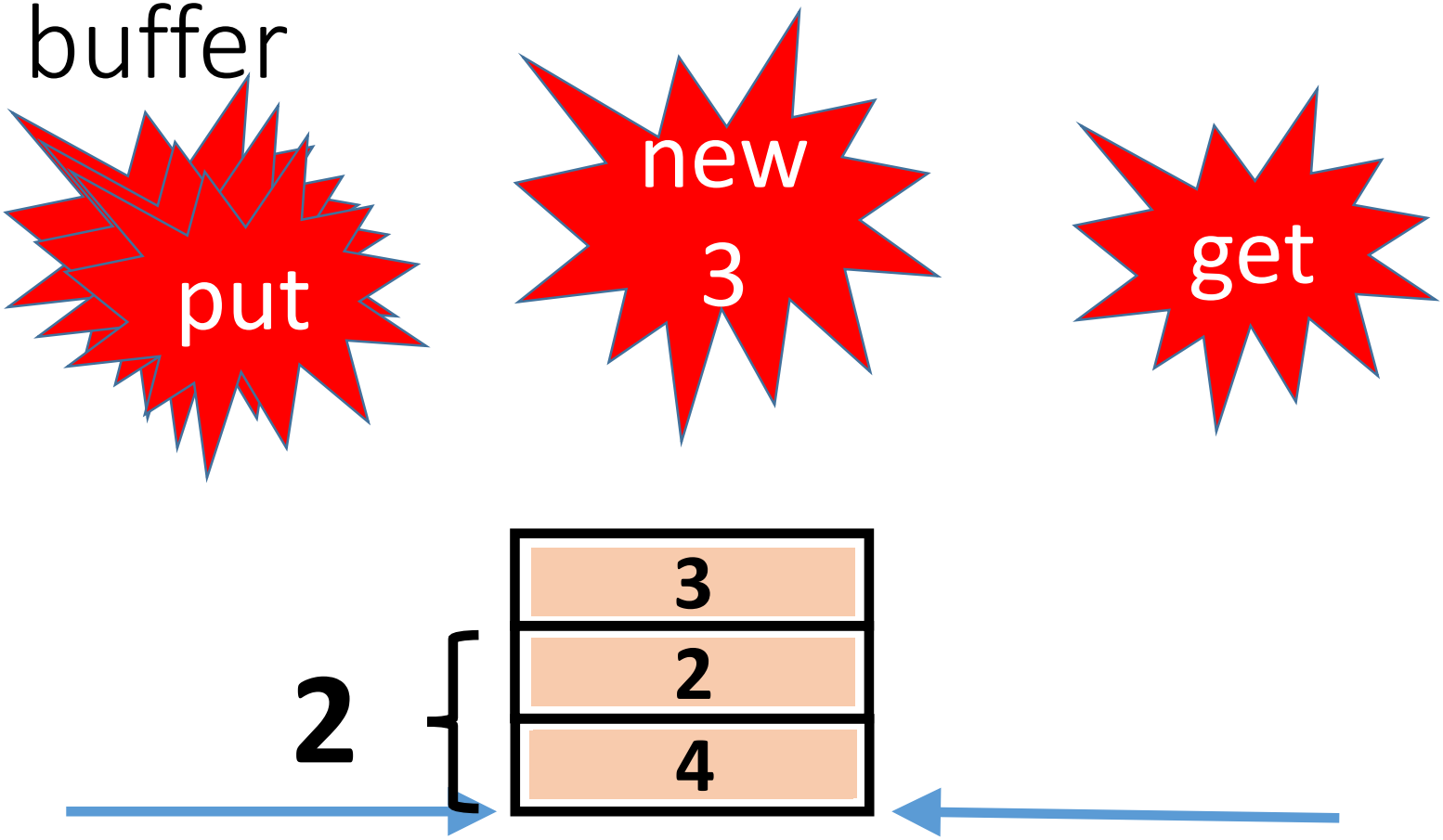
For all messages with 7-bit characters…

```
prop_sms() ->
  ?FORALL(L,list(choose(0,127)),
          unpack(pack(L)) == L).
```

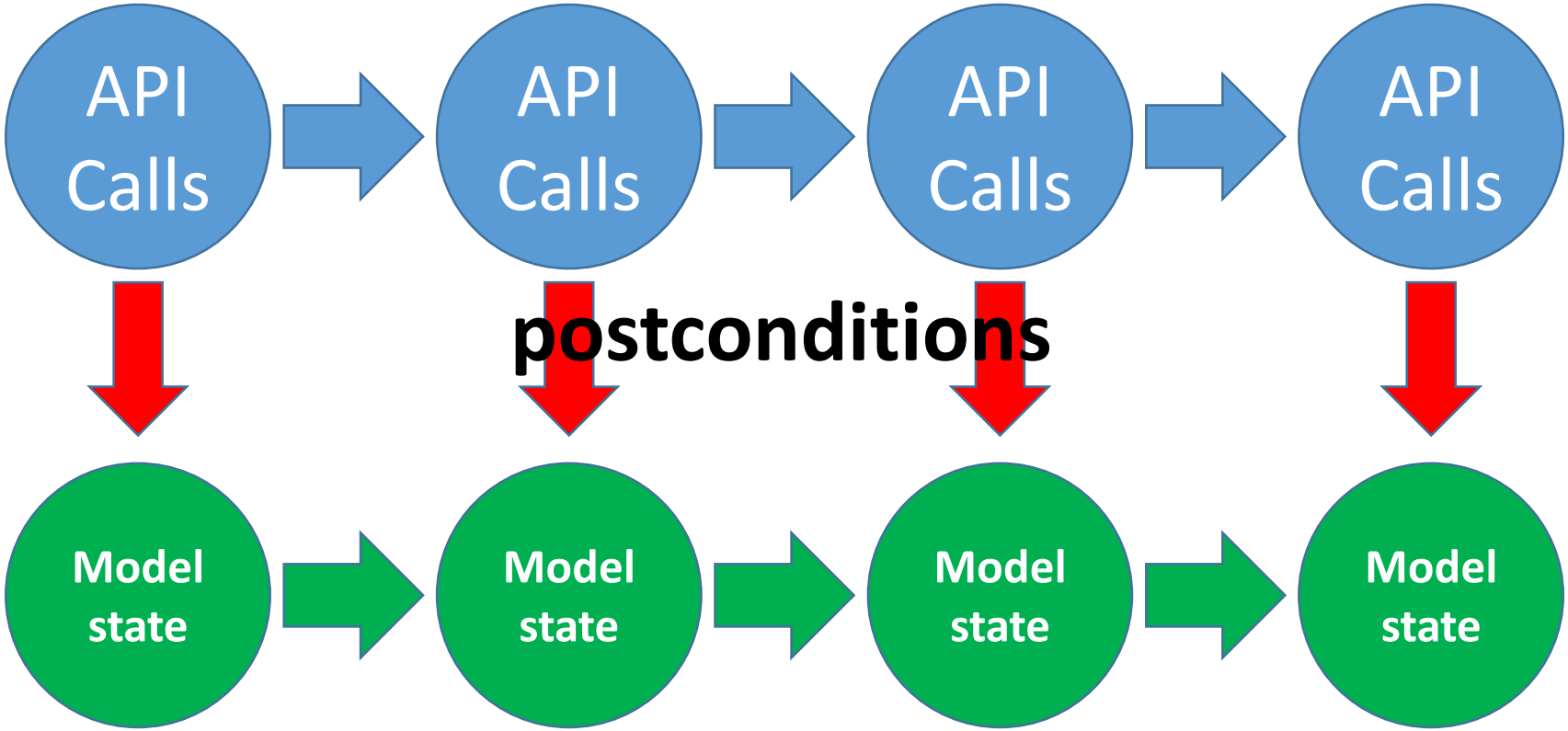
Exercise

- Test the sms encode/decoder, and diagnose any problems

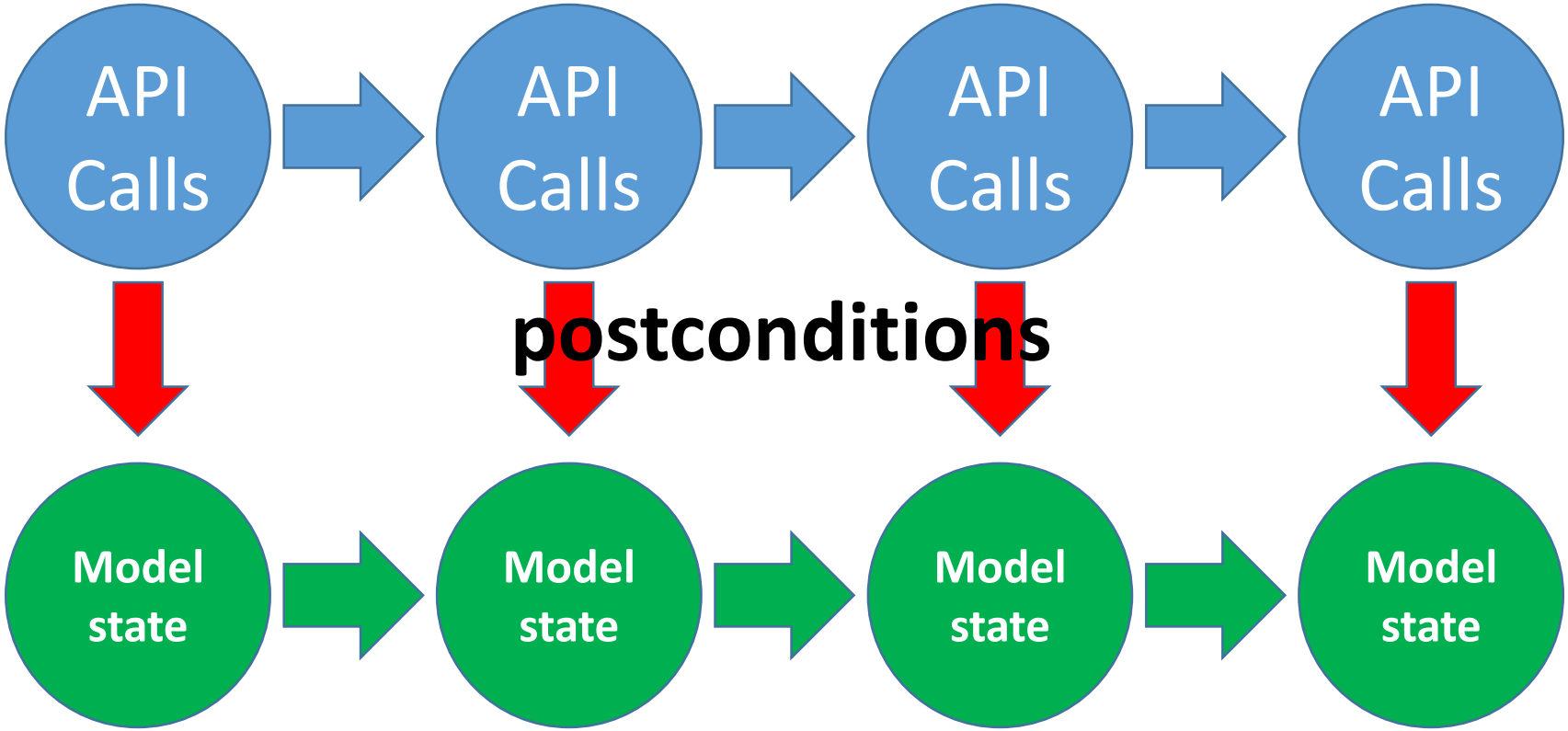
Example with state: a circular buffer



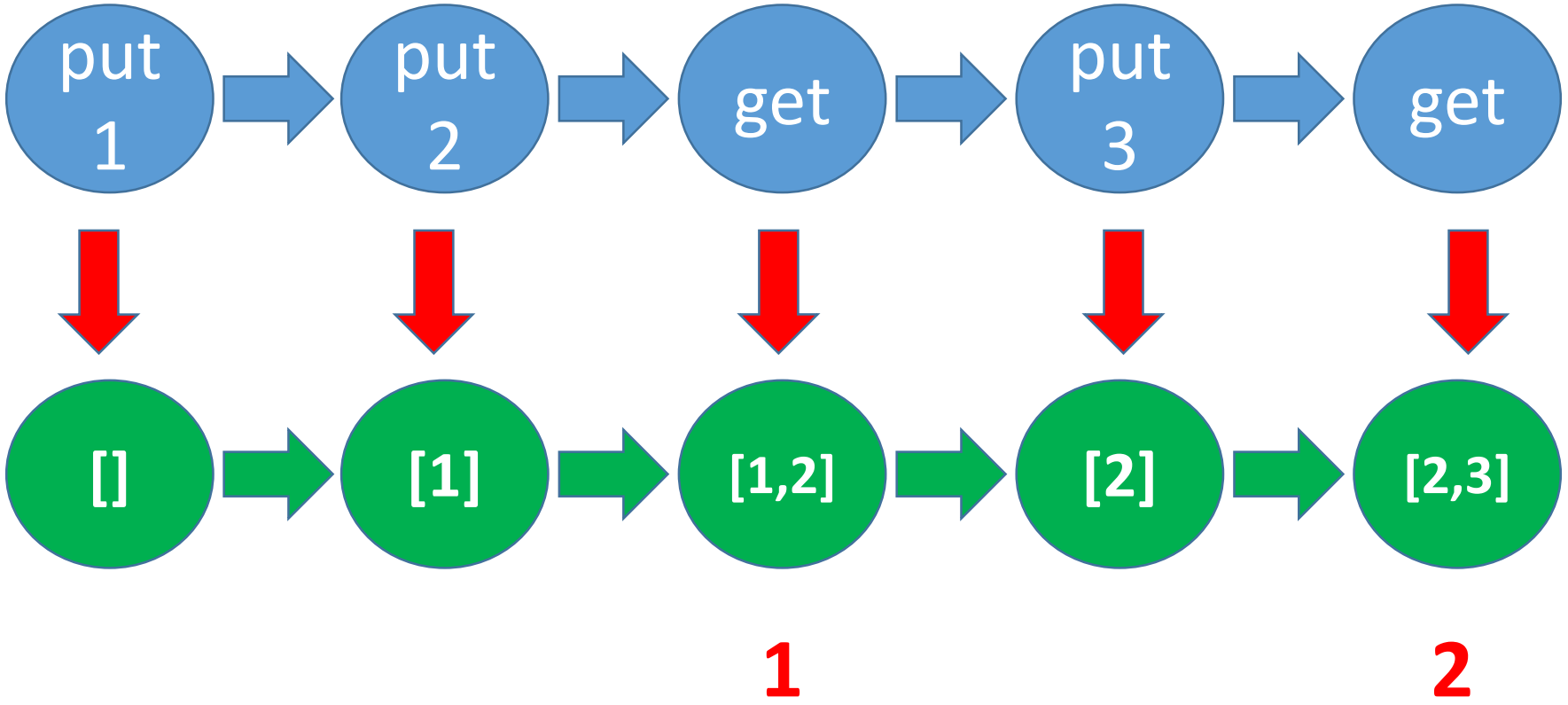
State Machine Models



State Machine Models



Example



The model state

```
-record(state,
    {ptr,
     size,
     elements}).
```

An Erlang record declaration

A pointer to the queue (test data)

The maximum size of the queue

The elements currently in the queue

Specification of get, part I

```
get_pre(S) ->
  S#state.ptr /= undefined.

get_args(S) ->
  [S#state.ptr].

get(Q) ->
  q:get(Q).

get_pre(S,[Q]) ->
  S#state.elements /= [].
```

Precondition parameterized on the model state

How to generate the argument list for get

How to call get

Precondition parameterised on the state *and arguments*

Specification of get, part II

Model state transition function

Erlang record selection and update

```
get_next(S,Result,[Q]) ->
  S#state{elements=tl(S#state.elements)}.

get_post(S,[Q],Result) ->
  eq(Result,hd(S#state.elements)).
```

Postcondition, gets state *beforehand*, args and result

The property—almost boilerplate

Generate a list of commands from callbacks in this module

```
prop_q() ->
  ?FORALL(Cmds,commands(?MODULE),
    begin
      {H,S,Res} = run_commands(?MODULE,Cmds),
      check_commands(?MODULE,Cmds,{H,S,Res})
    end).
```

Let’s run some tests…

Exercise—modelling the process registry

- **`spawn()`**—create a new process, return its *process identifier* (Pid)
- **`register(Name,Pid)`**—register the pid with this name
- **`whereis(Name)`**—return the pid registered with a name
- **`unregister(Name)`**—remove a pid from the registry

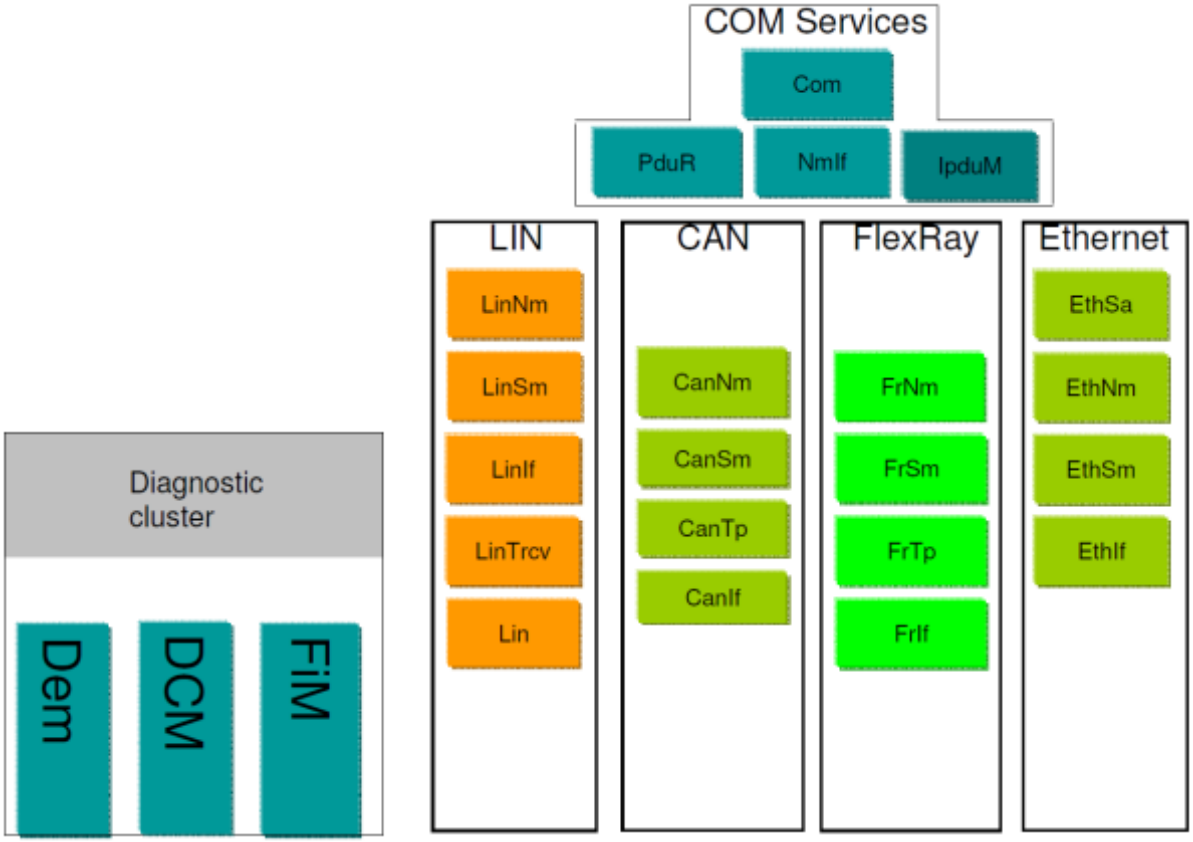
Reverse engineer the right *preconditions* to prevent exceptions being raised

See registry_eqc.erl (includes instructions)

Reflections

- Reverse engineering specifications—how realistic is that?
- Does any of this scale?

Doing it for real…



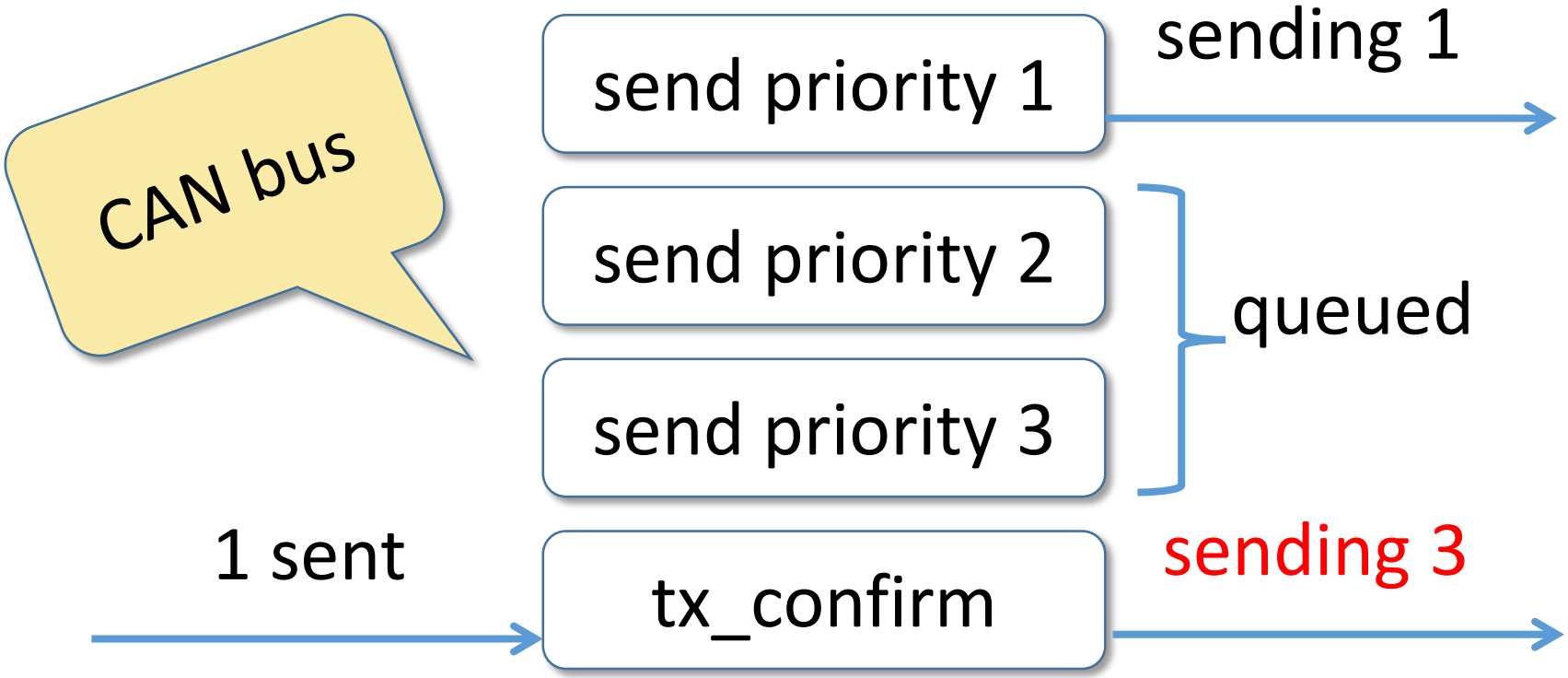
Theory

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

Practice

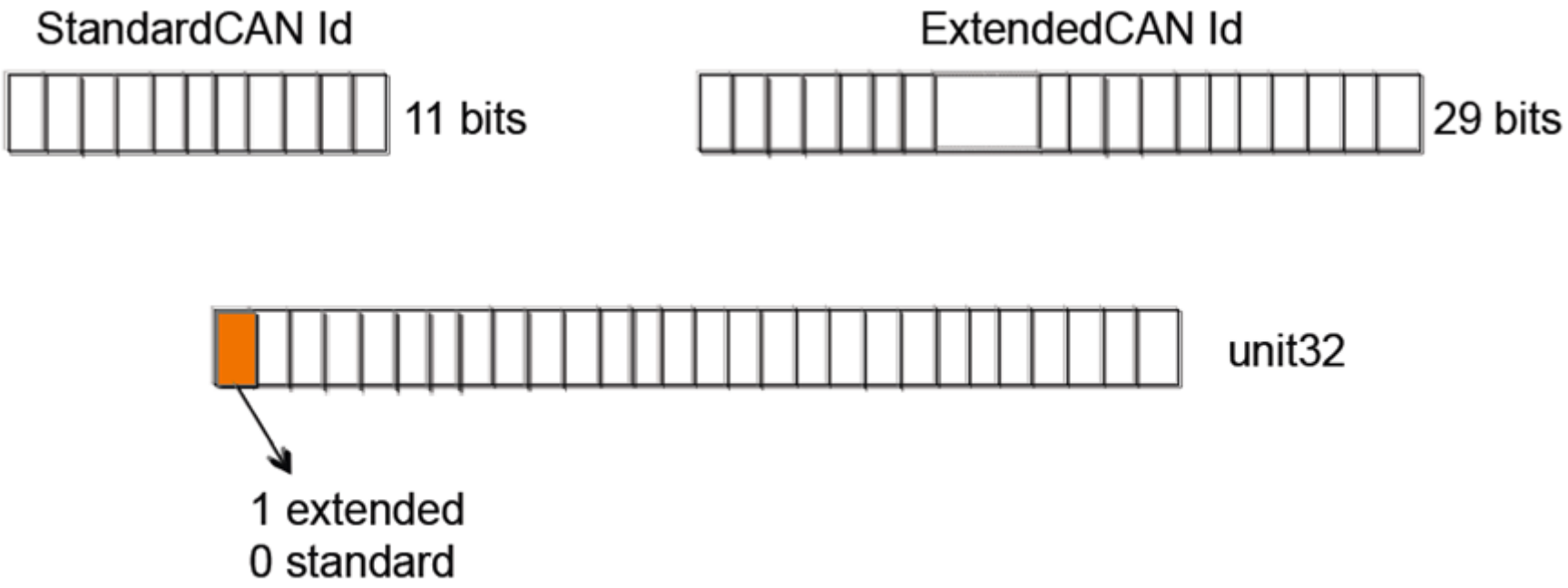
VOLVO's experience has been that this is often not the case

A Bug in a vendor's CAN stack

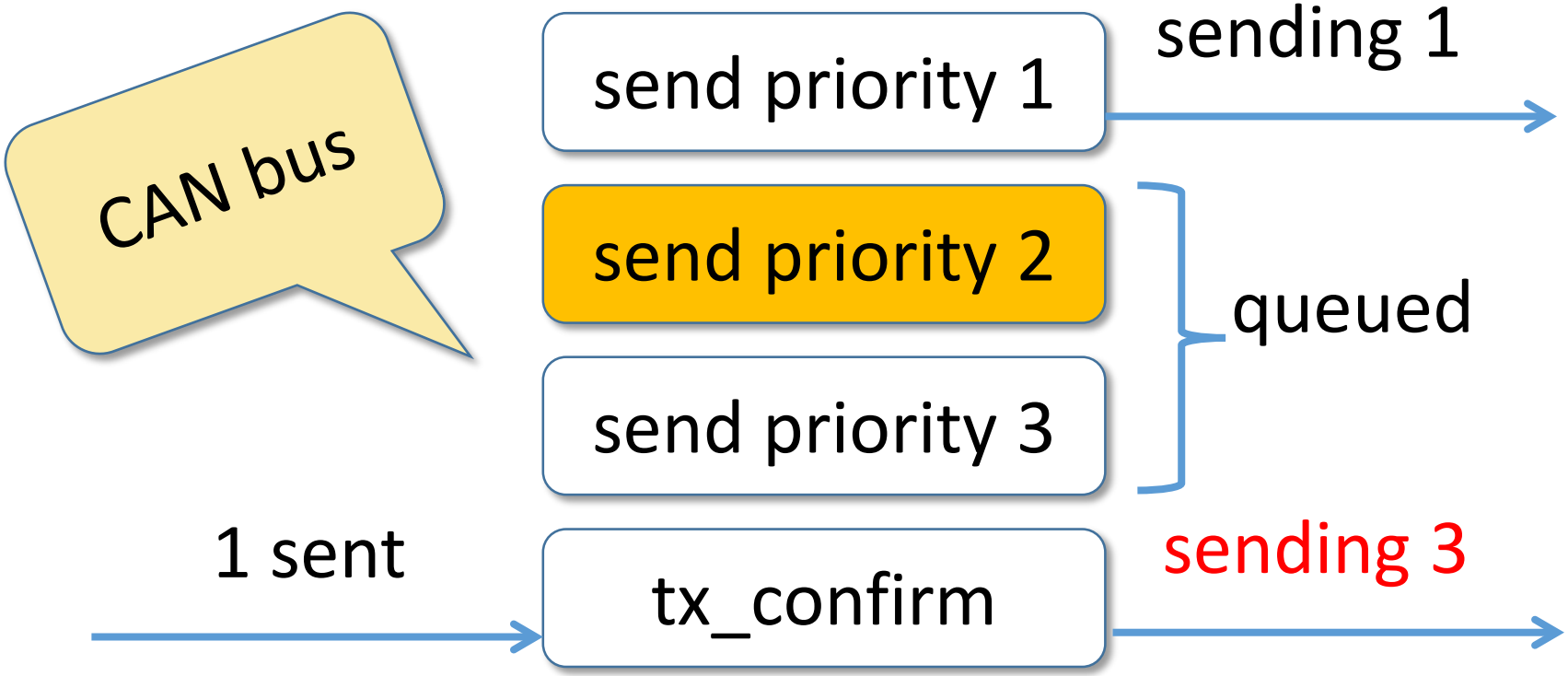


The Problem

CAN bus identifiers determine bus priority



A Bug in a vendor's CAN stack



Failed to mask off the top bit before comparing priorities

3,000 pages of specifications

20,000 lines of QuickCheck

1,000,000 LOC, 6 suppliers

200 problems

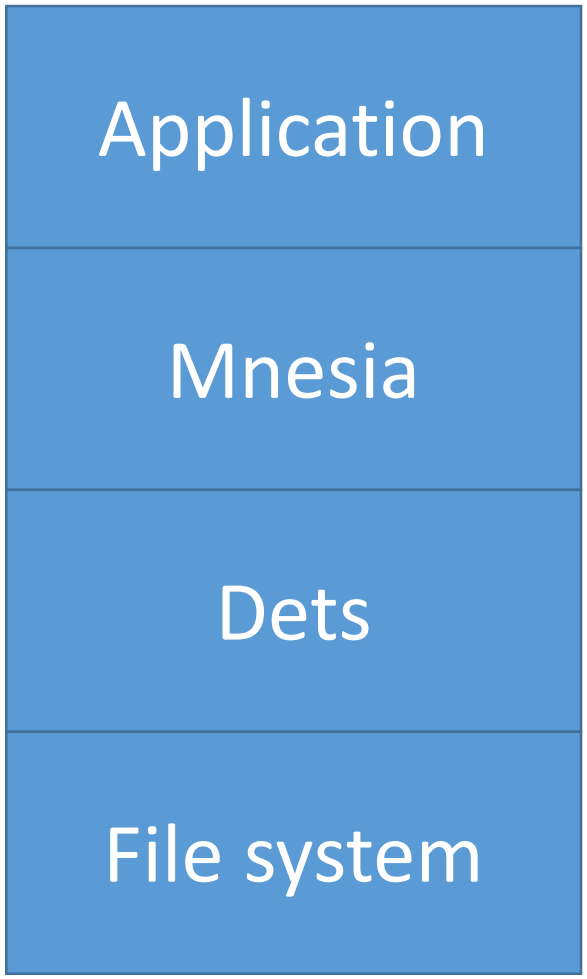
100 problems in the standard

10x shorter test code

"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

What is it?



klarna

Invoicing services for web shops

Distributed database: transactions, distribution, replication

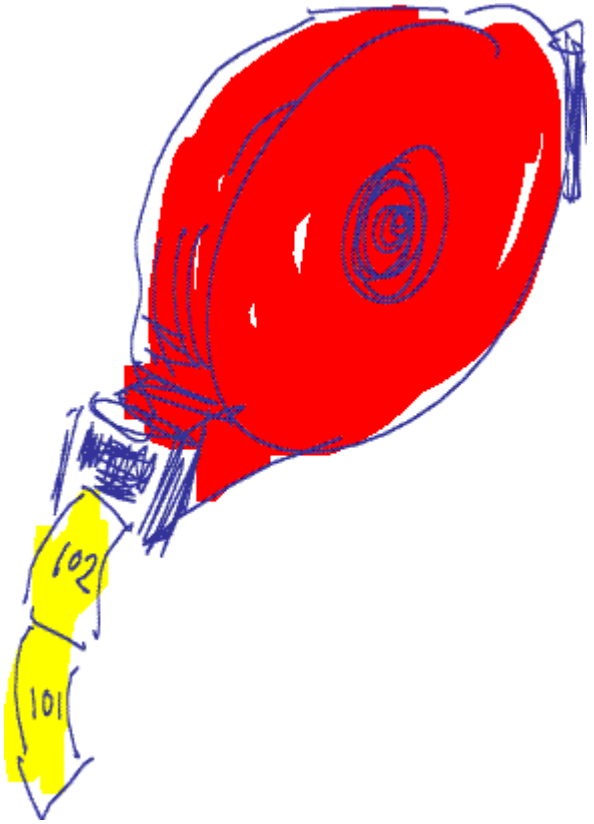
Tuple storage

Race conditions?

Imagine Testing This…

dispenser:take_ticket()

dispenser:reset()

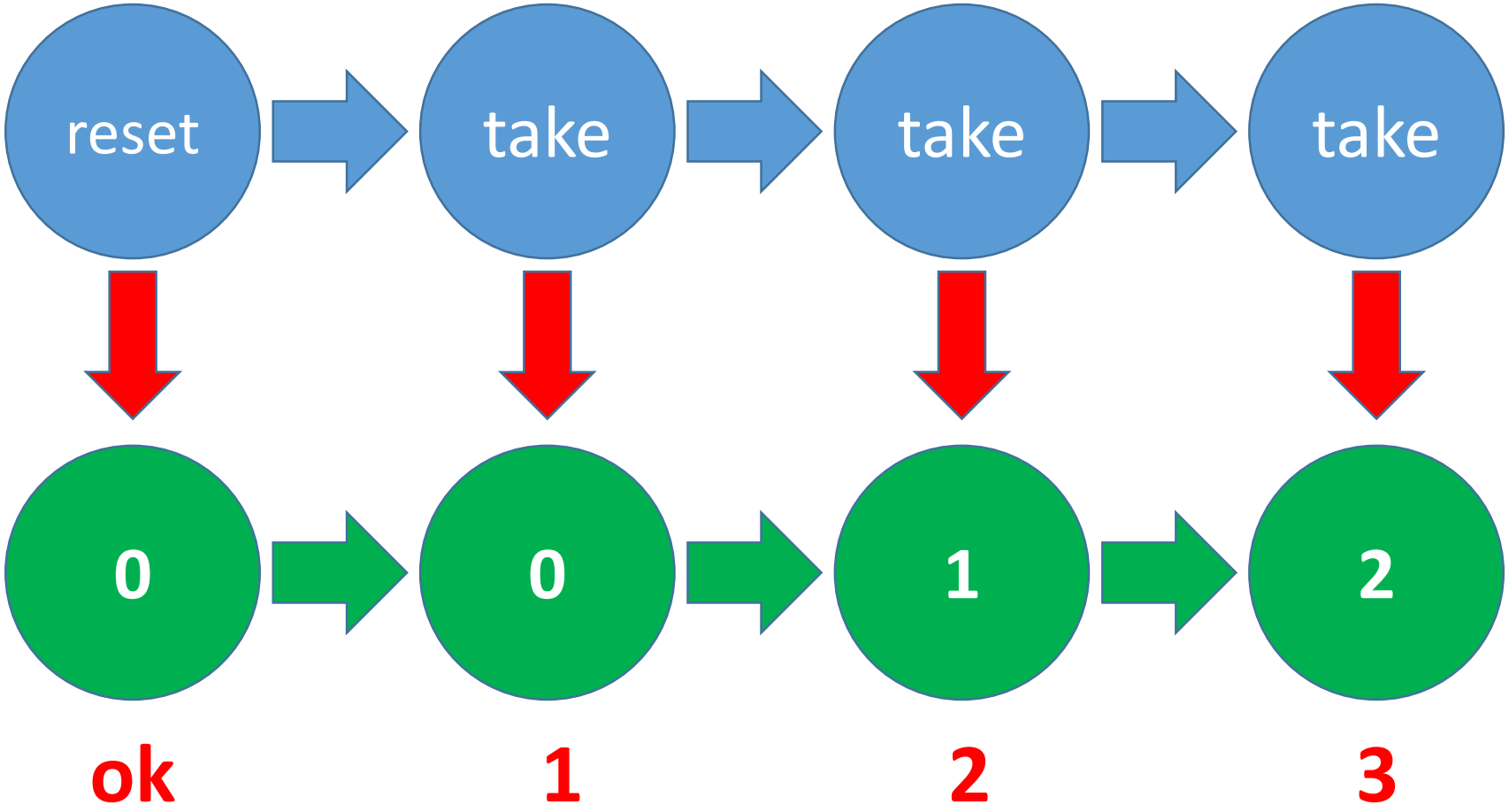


A Unit Test in Erlang

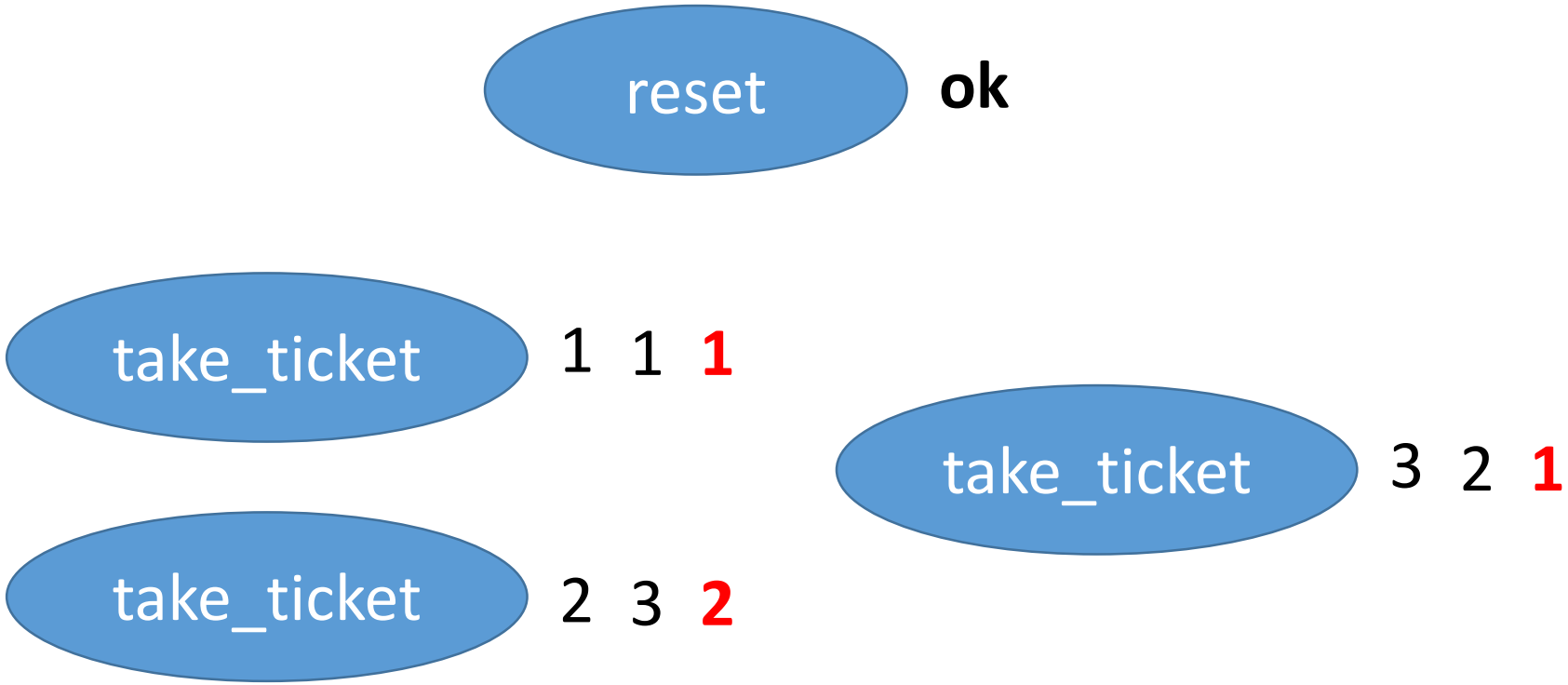
```
test_dispenser() ->
      ok =  reset(),
      1  =  take_ticket(),
      2  =  take_ticket(),
      3  =  take_ticket(),
      ok =  reset(),
      1  =  take_ticket().
```

Expected results

Modelling the dispenser

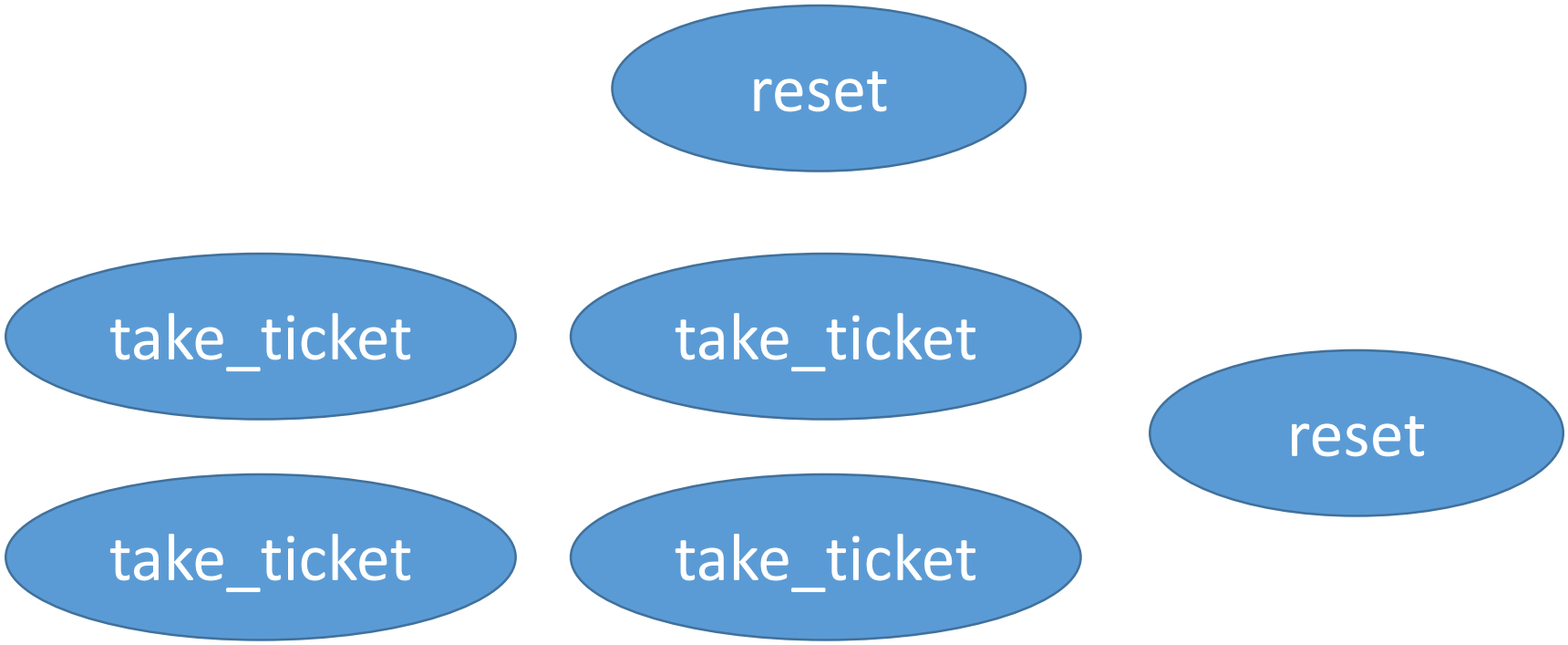


A Parallel Unit Test



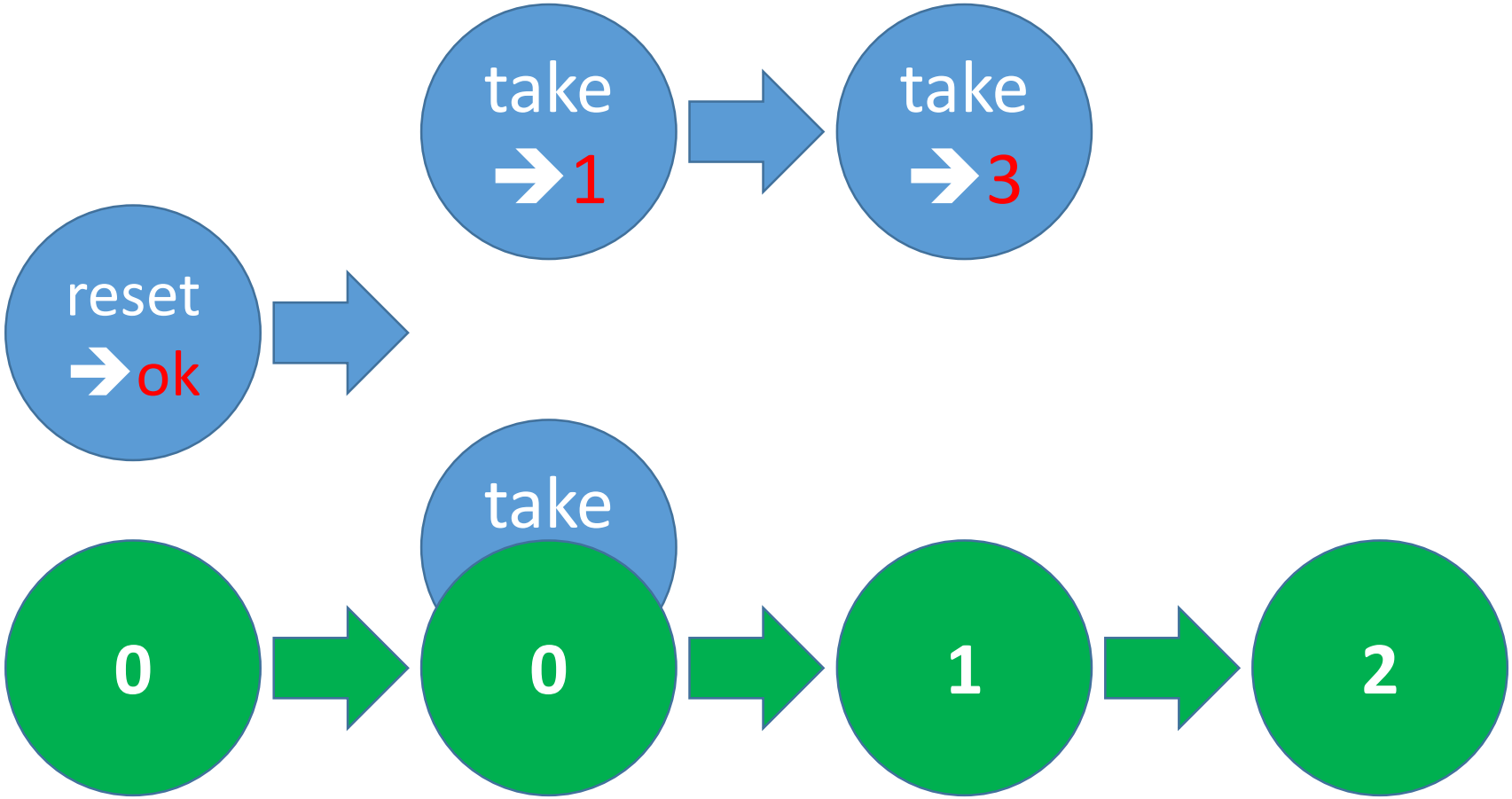
- Three possible correct outcomes!

Another Parallel Test



- 30 possible correct outcomes!

Deciding a Parallel Test



Let’s run some tests

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

Prefix:

Parallel:

1. dispenser:take_ticket() --> 1

2. dispenser:take_ticket() --> 1

Result: no_possible_interleaving

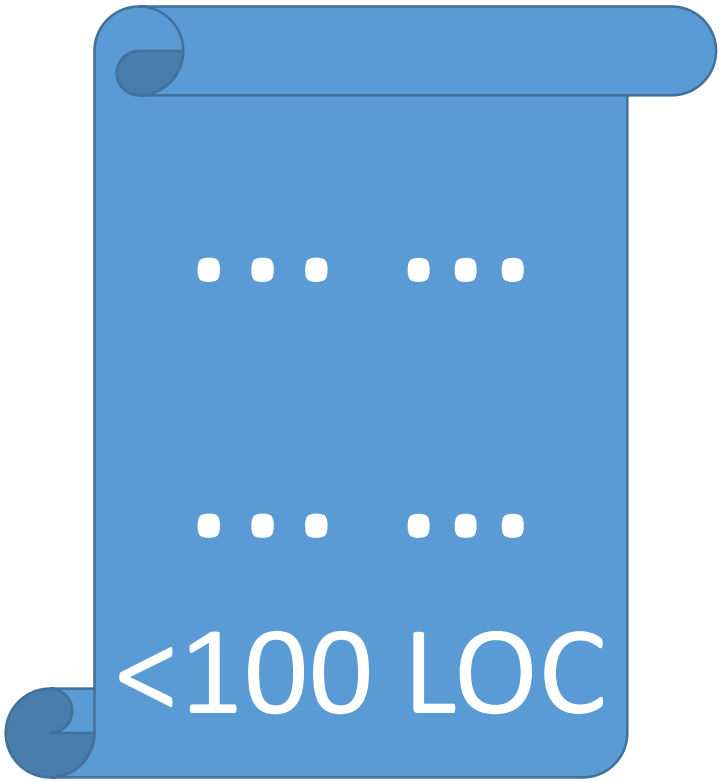
dets

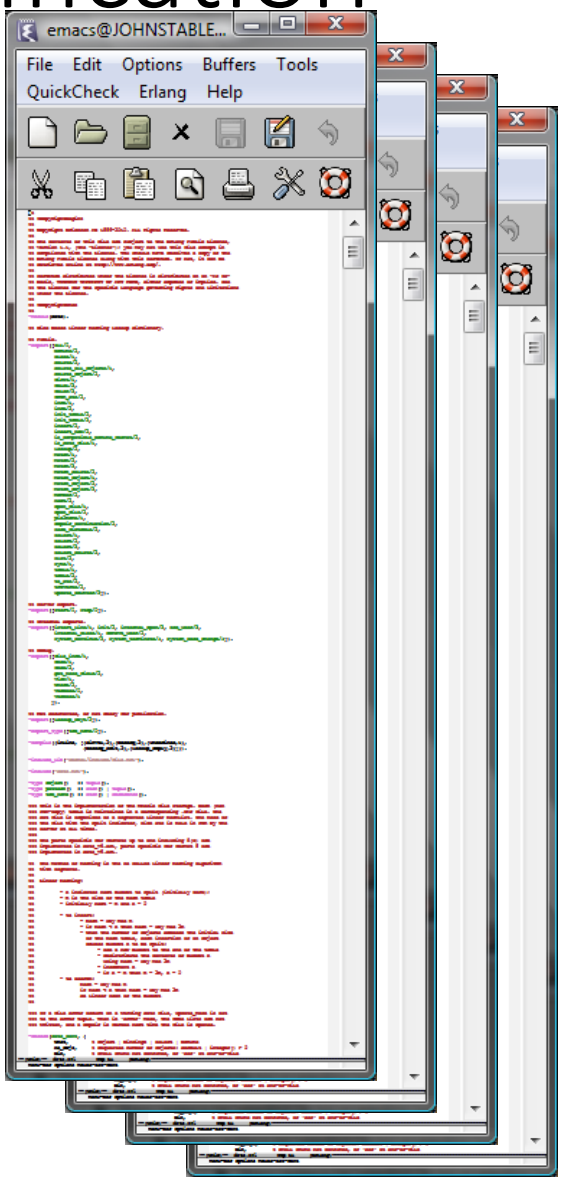
- Tuple store:

  {Key, Value1, Value2…}

- Operations:
  - insert(Table,ListOfTuples)
  - delete(Table,Key)
  - insert_new(Table,ListOfTuples)
  - …
- Model:
  - List of tuples (almost)

QuickCheck Specification





> 6,000 LOC

Bug #1

insert_new(Name, Objects) -> Bool

Types:
Name = name()
Objects = object() | [object()]
Bool = bool()

```
Prefix:
   open_file(dets

Parallel:
1. insert(dets_ta

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

Result: no_possible_interleaving
```

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

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
```

Is the file corrupt?

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
```

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
```

"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."

Tobbe Törnqvist, Klarna, 2007

Each bug fixed the day after reporting the failing case

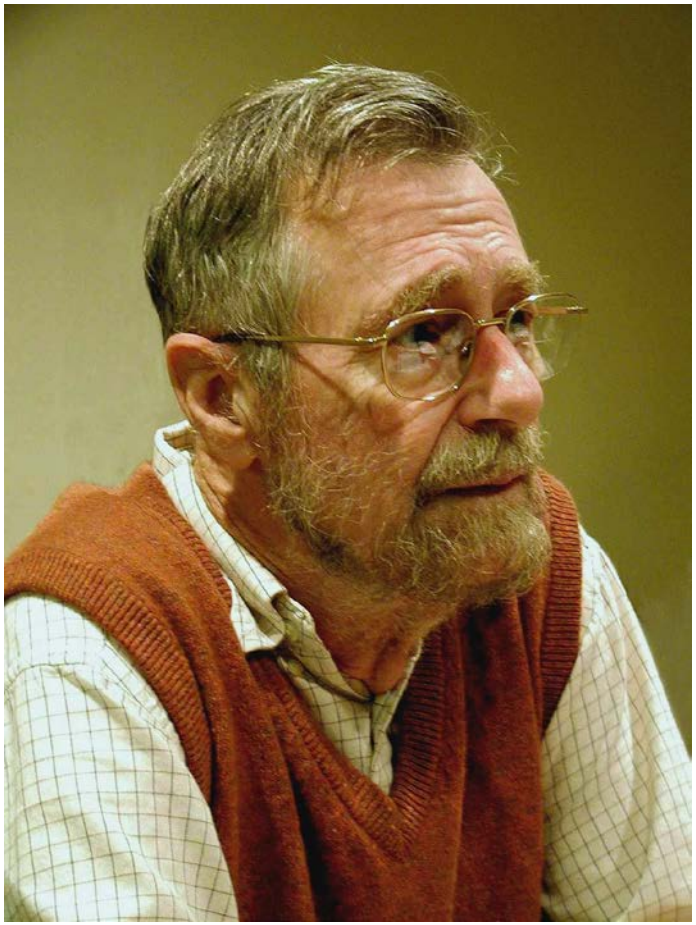
Before

- Files over 1GB?
- Rehashing?
- > 6 weeks of effort!

After

- Database with *one* record!
- 5—6 calls to reproduce
- < 1 day to fix

Reflections



”Testing can never demonstrate the *absence* of bugs in software, only their presence”

COMPCERT

COMPILERS YOU CAN *FORMALLY* TRUST

- An entire optimising C compiler, verified in Coq
  - Enormously impressive *tour de force!*

8x more costly than conventional compilers

Not bug free

100x fewer bugs than conventional compilers

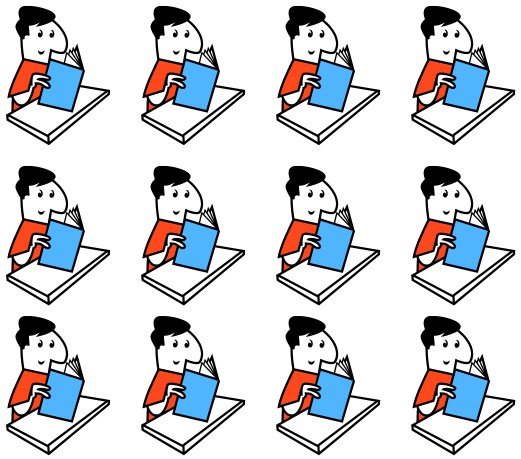
Can formal proofs demonstrate the absence of bugs in software?

Specifications are almost always wrong

Testing is both cheaper and vastly more effective than it used to be

A final thought

Unit tests



Properties

How good were the tests at find bugs—in *other* students' code?