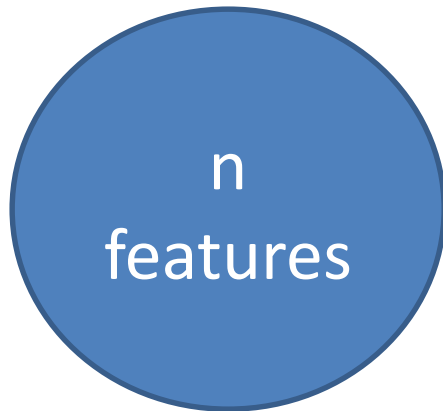


Experiences with QuickCheck: Testing the Hard Stuff and Staying Sane

John Hughes

CHALMERS **QuviQ**

Why is testing hard?



$O(n^2)$ test cases

3—4 tests per
pair of features
per feature

Don't write tests!

Generate them

QuickCheck



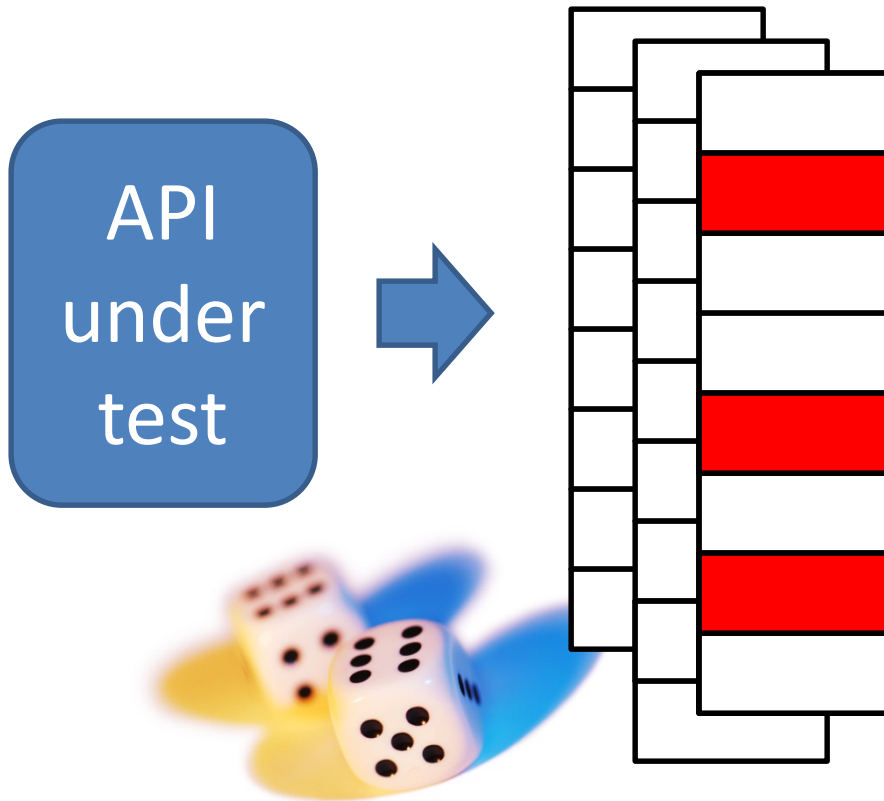
1999—invented by Koen Claessen and myself, for Haskell

2006—Quviq founded marketing Erlang version

Many extensions

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

The commonest usage

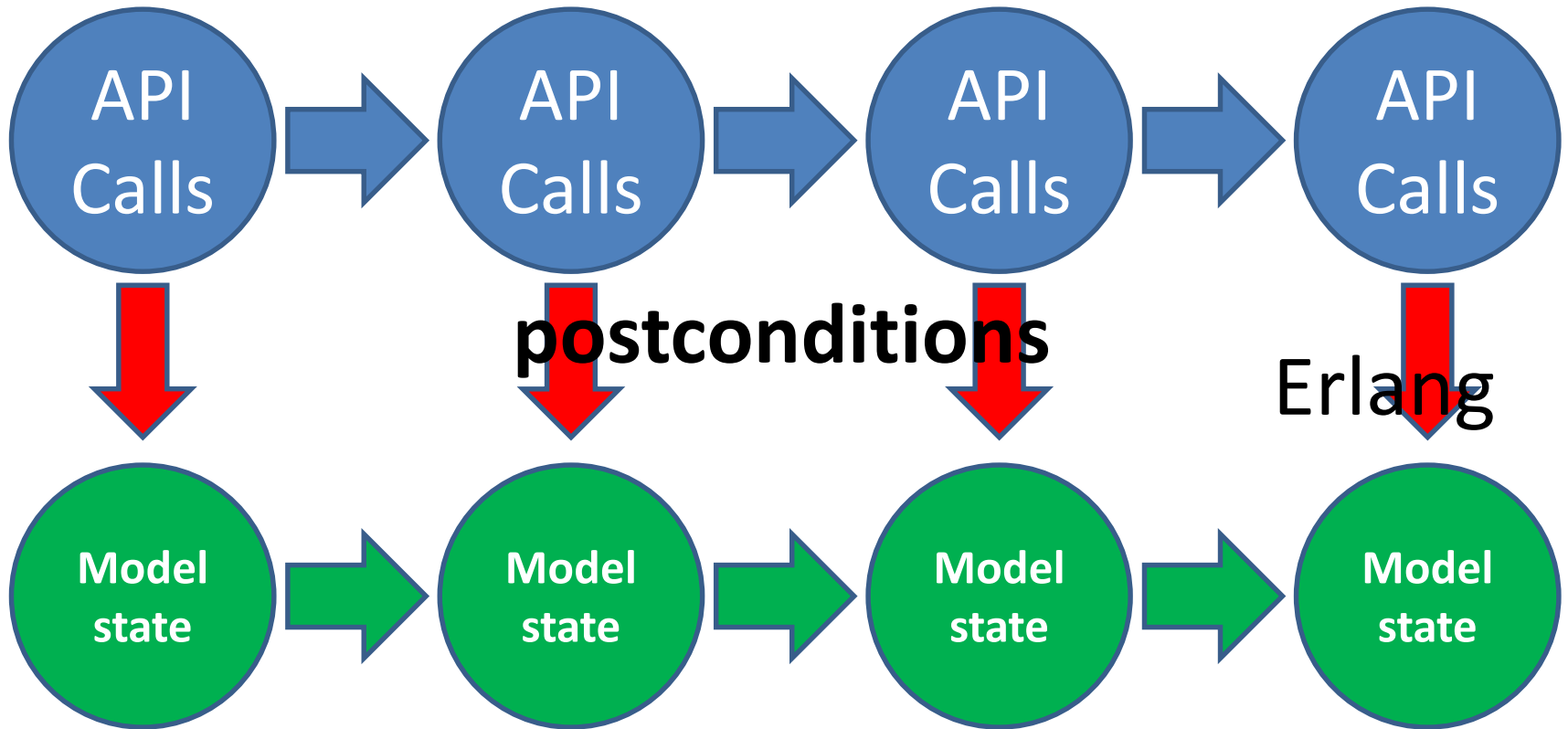


A minimal failing
example

Example: a Circular Buffer

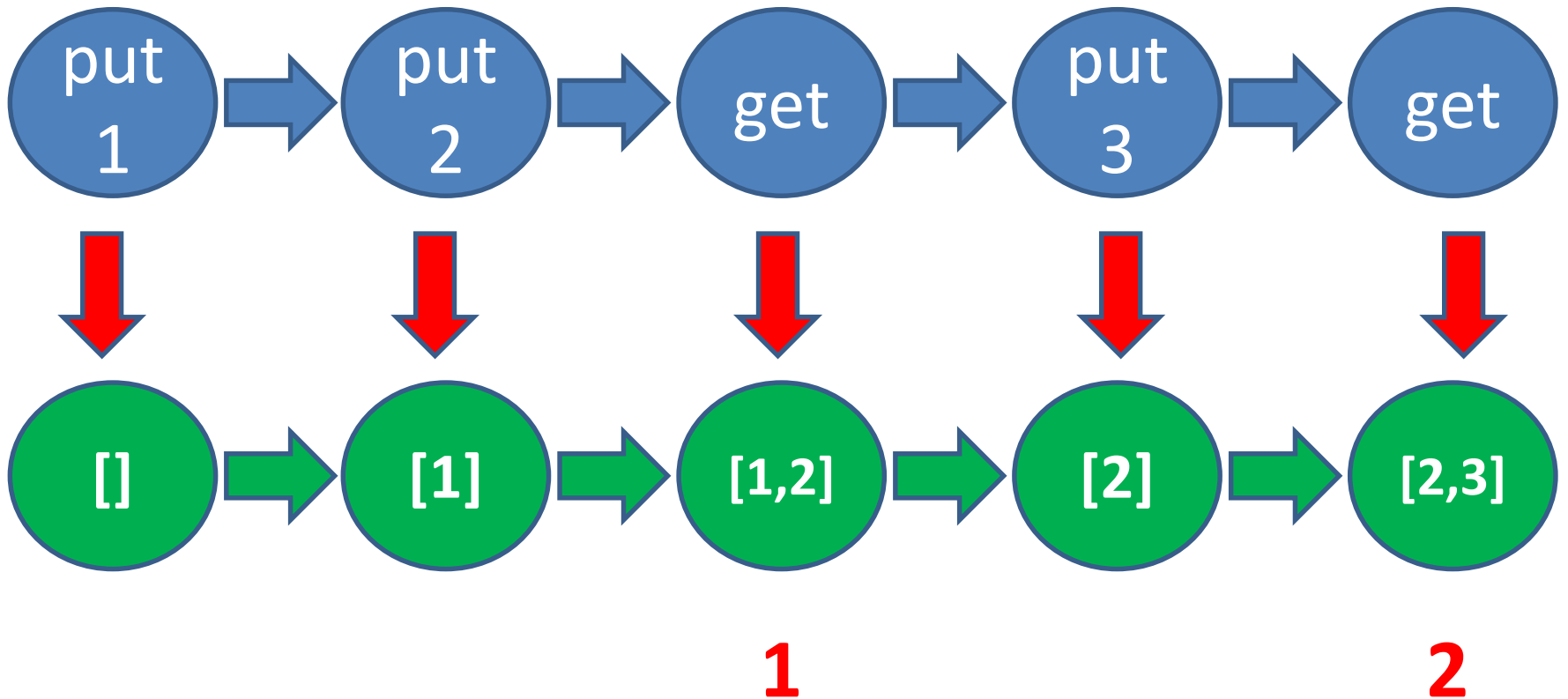
State Machine Models

Erlang



Erlang

Example



Code Fragments: specifying get

```
get_pre(S) ->  
  S#state.ptr /= undefined andalso  
  S#state.contents /= [].
```

Precondition

```
get_next(S, _Value, _Args) ->  
  S#state{contents=tl(S#state.contents)}.
```

State
transition

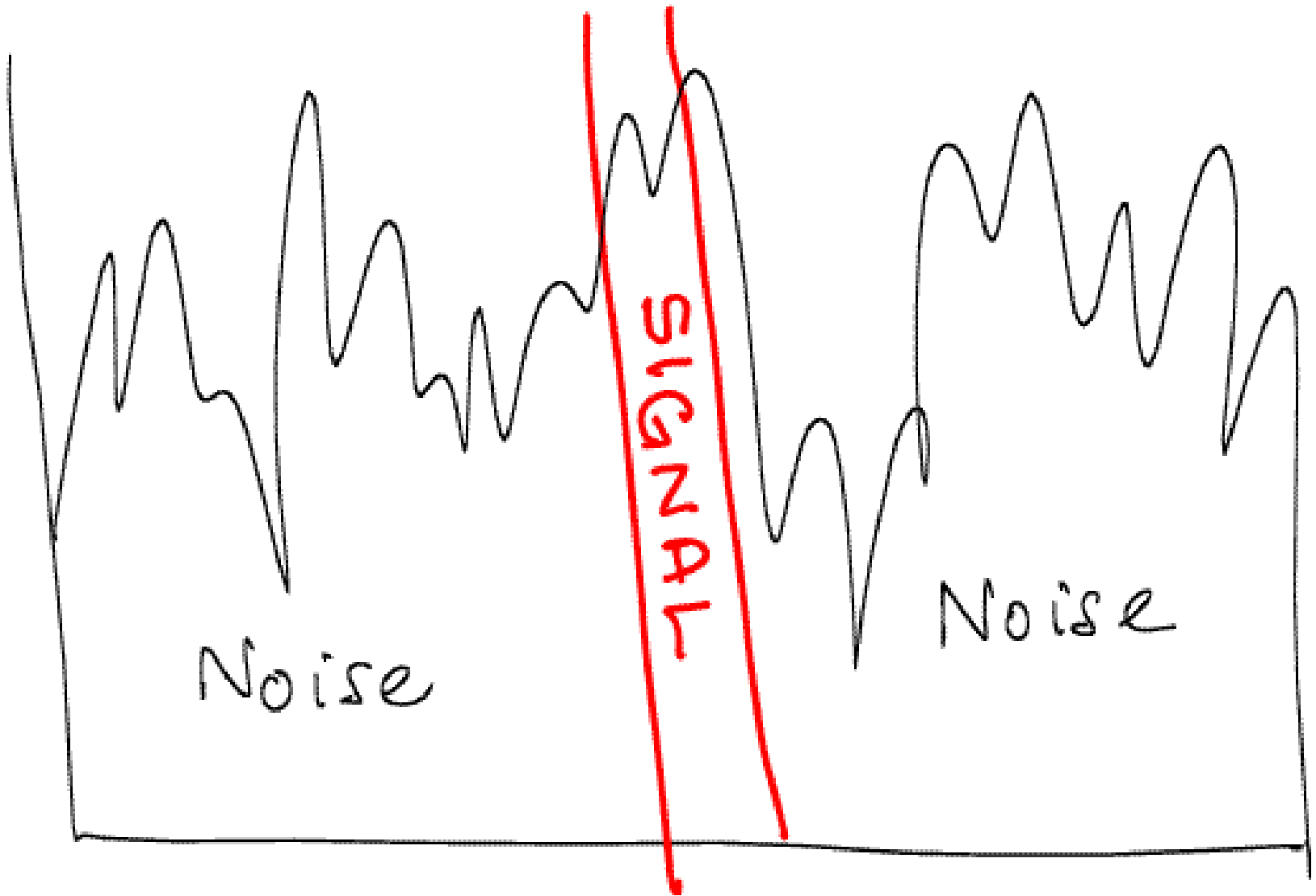
```
get_post(S, _Args, Res) ->  
  eq(Res, hd(S#state.contents)).
```

Postcondition

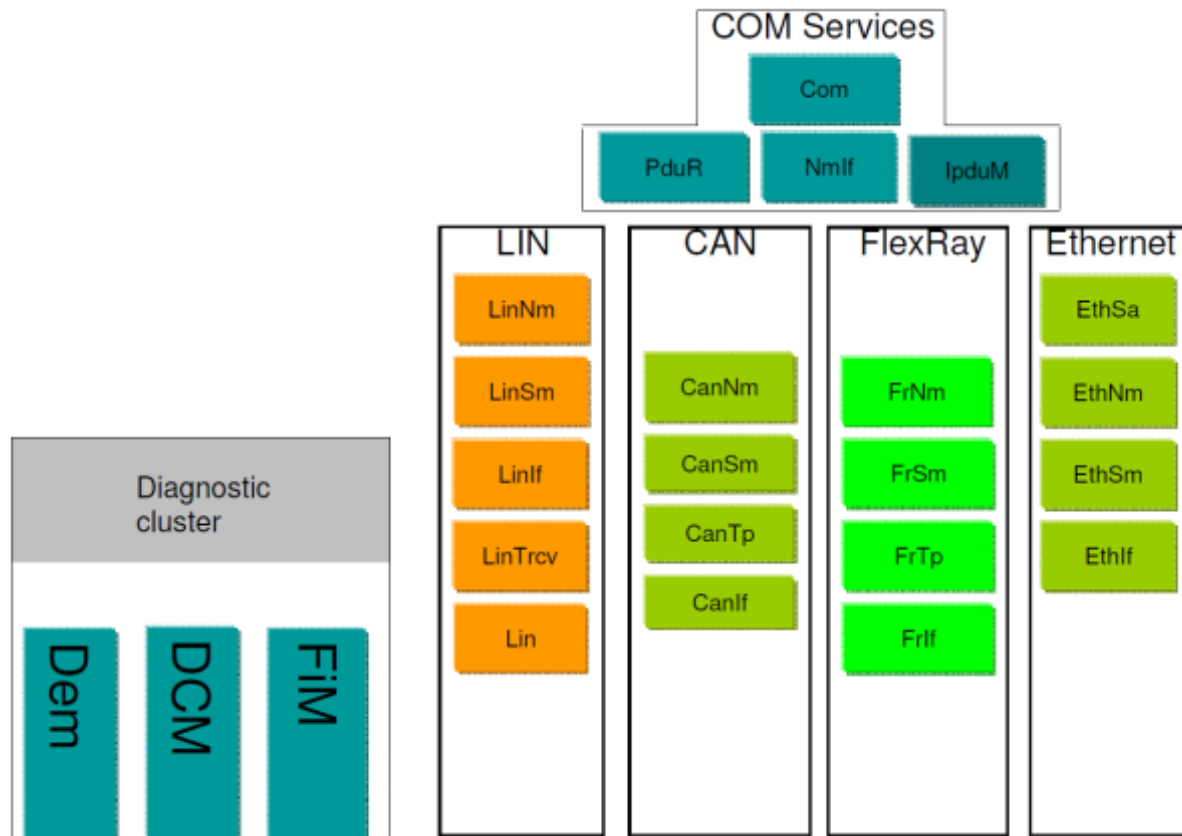
Time for some tests!

Lessons

- The *same* property can find *many* different bugs
- Minimal failing tests make diagnosis easy



Doing it for real...



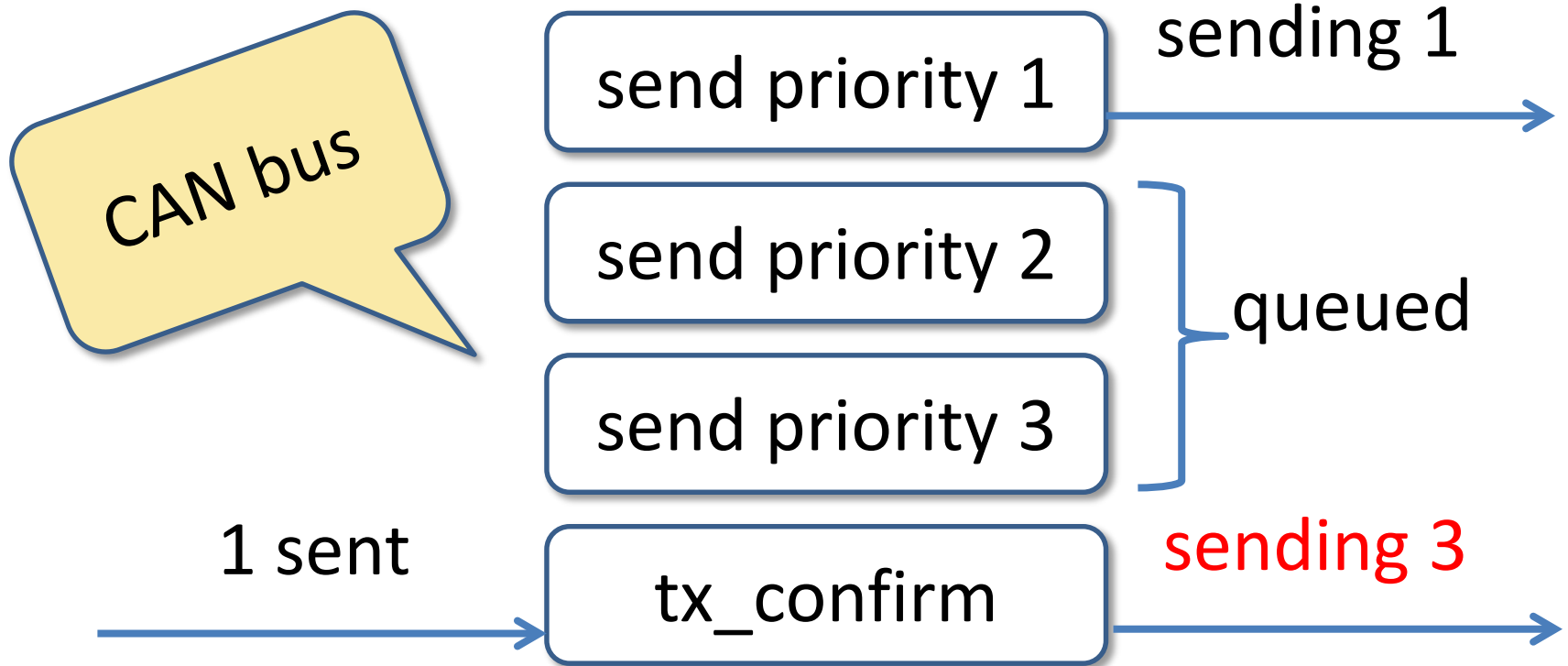
Theory

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

Practice

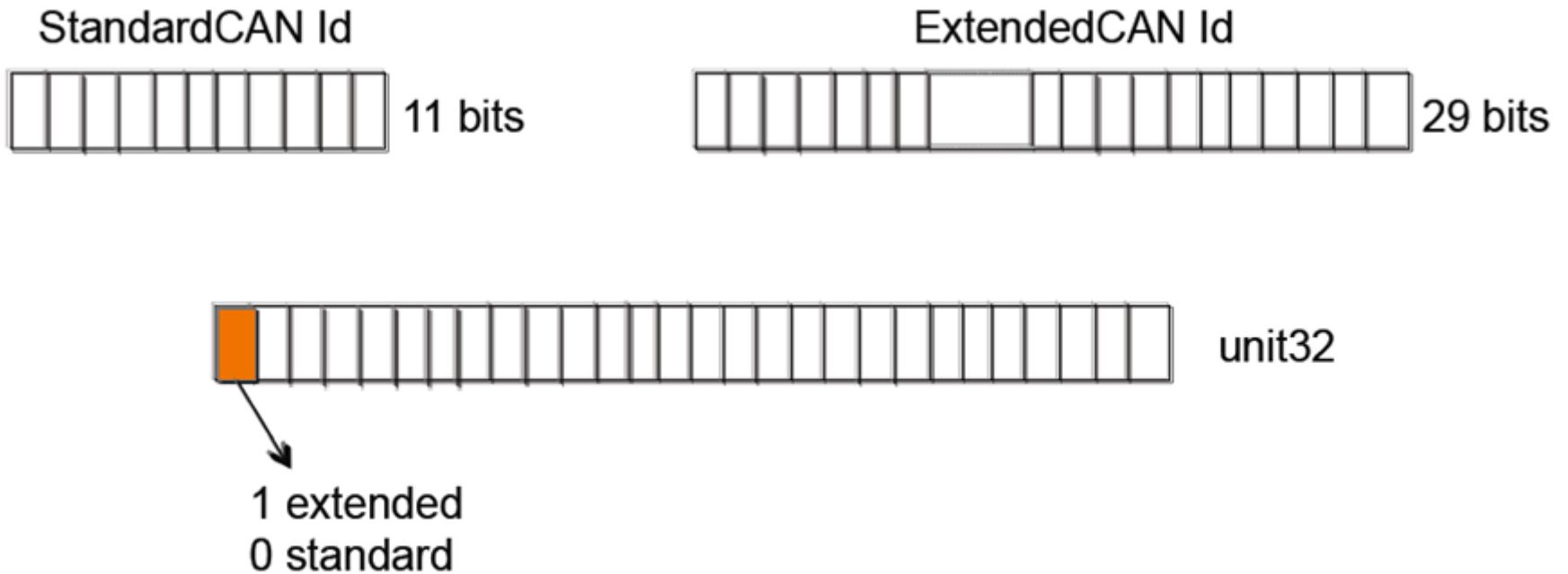
VOLVO's experience has been
that this is often not the case

A Bug in a vendor's CAN stack

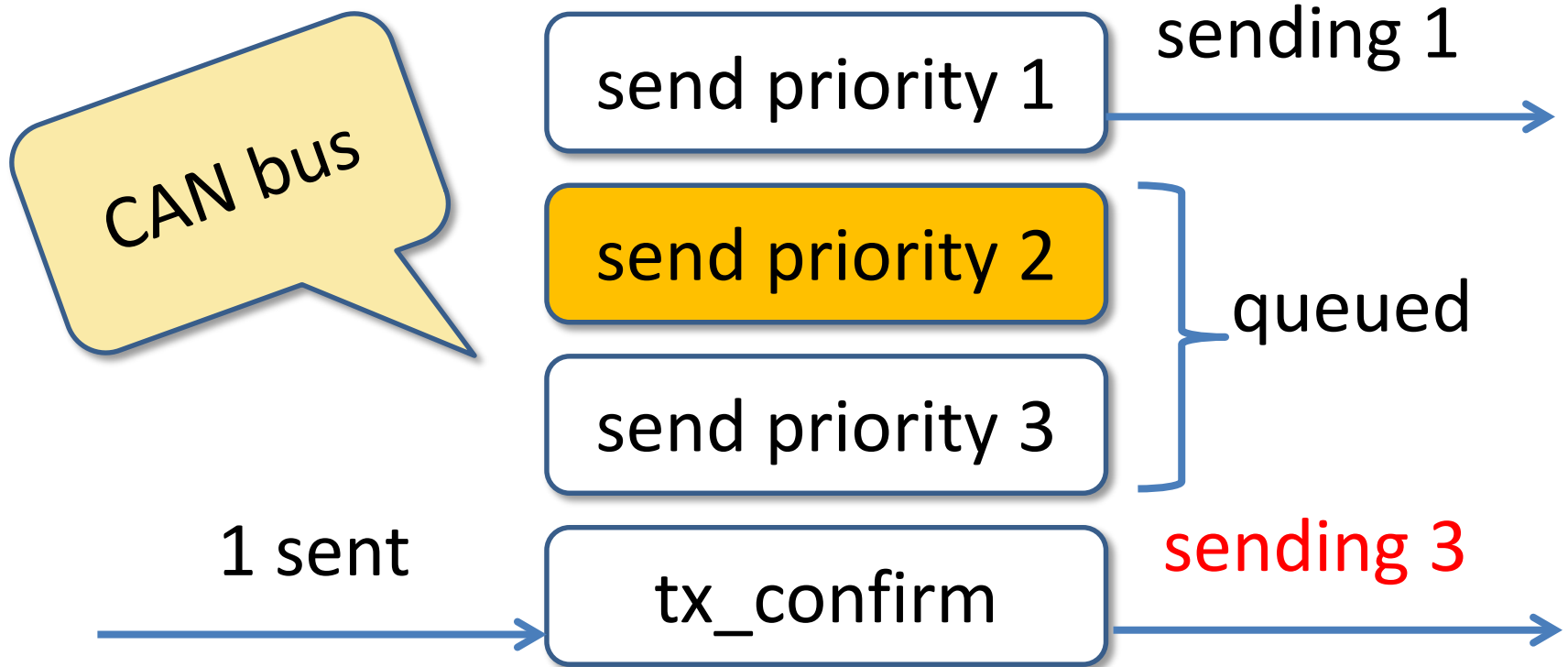


The Problem

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

9X 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

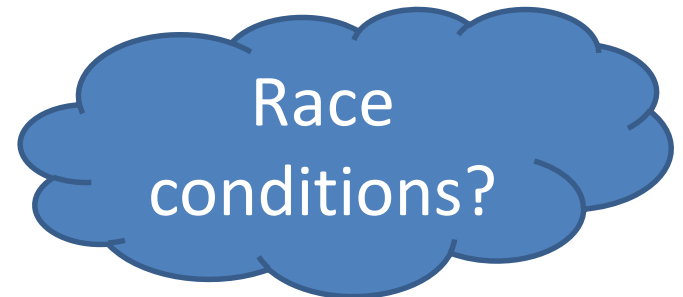
What is it?



Invoicing services for web shops

Distributed database:
transactions, distribution,
replication

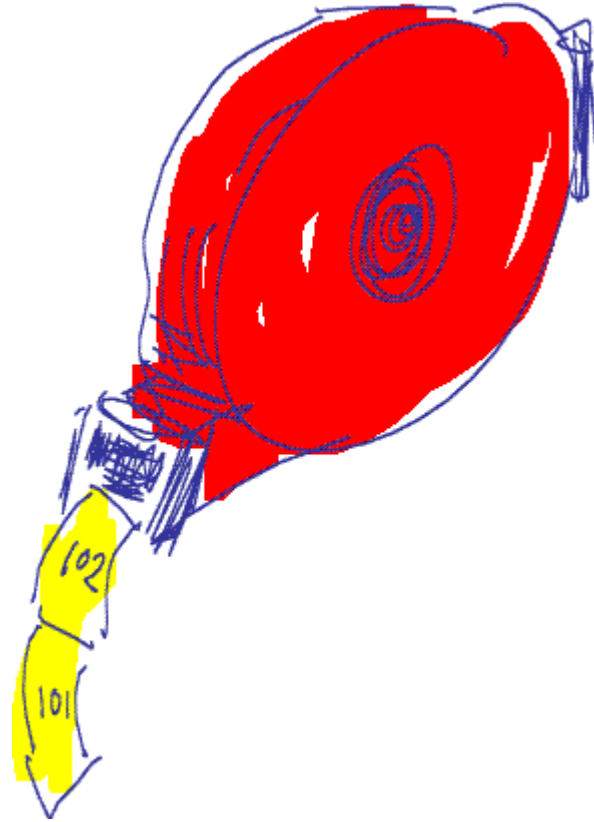
Tuple storage



Imagine Testing This...

`dispenser:take_ticket()`

`dispenser:reset()`



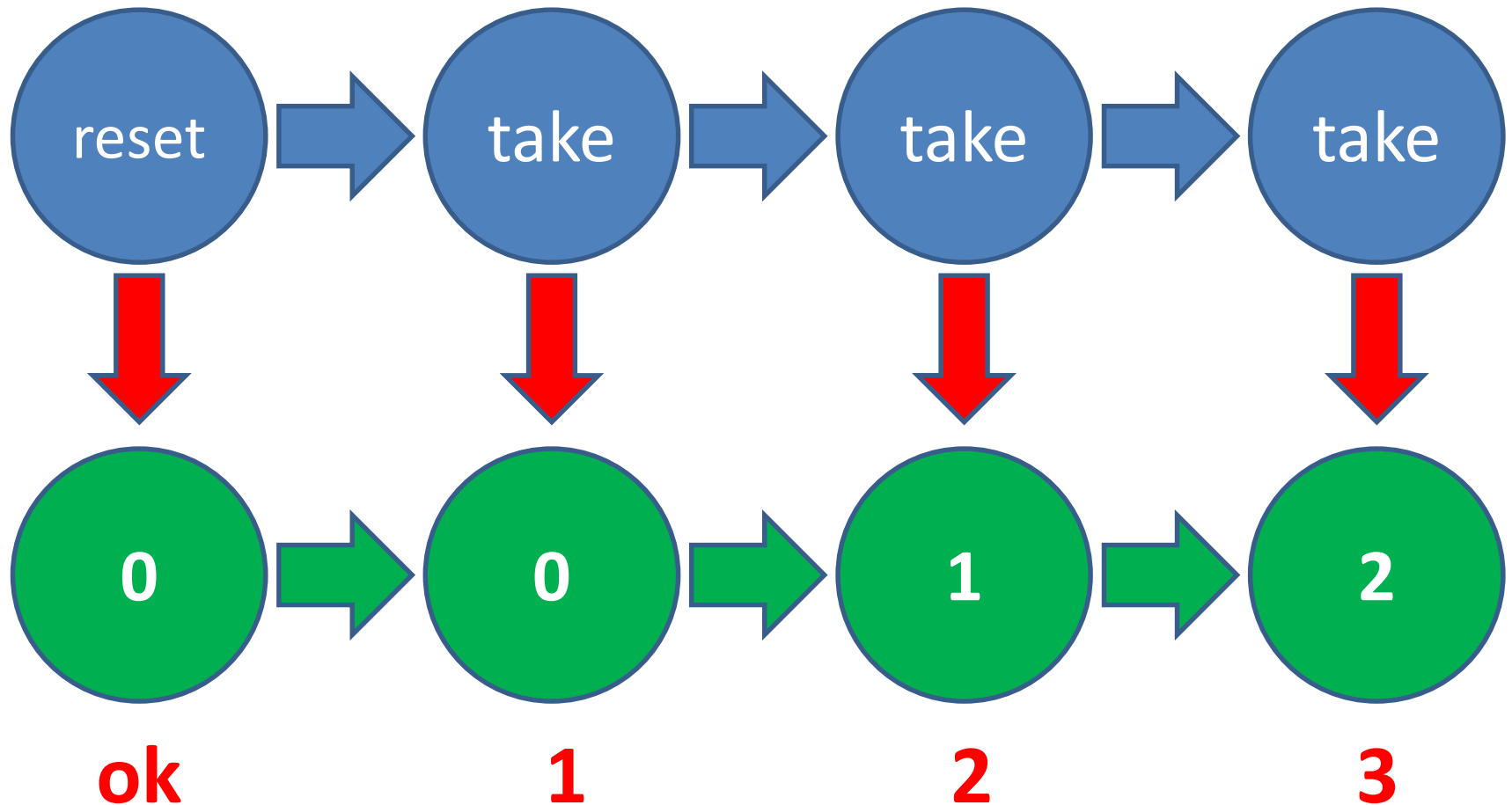
A Unit Test in Erlang

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

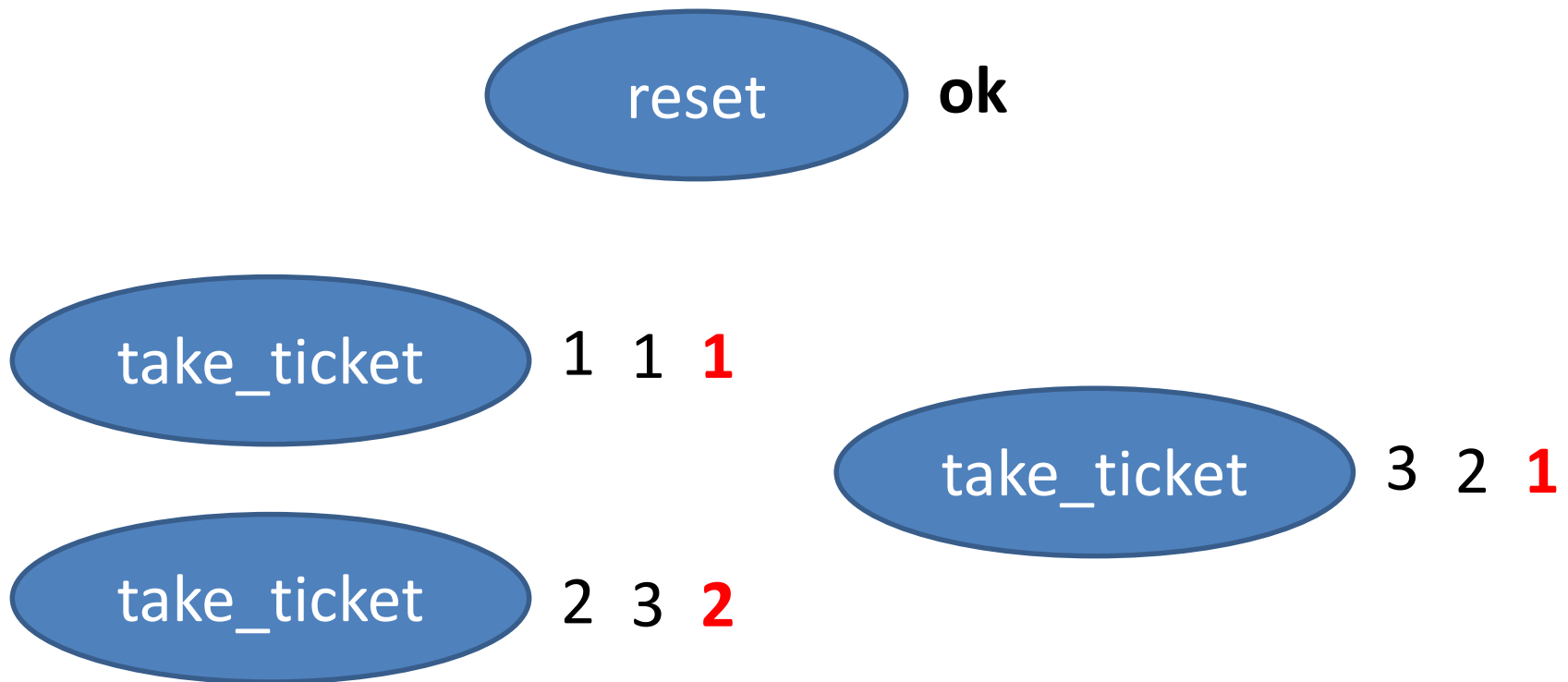


Expected
results

Modelling the dispenser

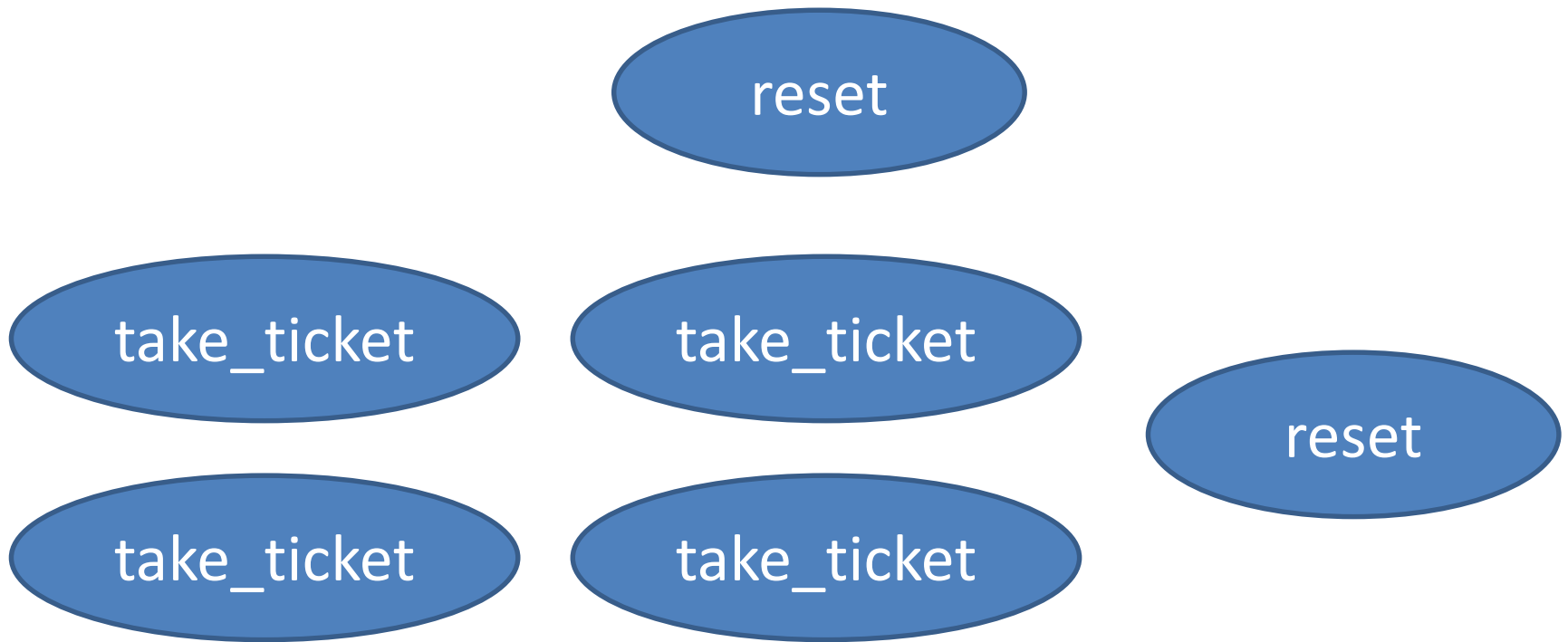


A Parallel Unit Test



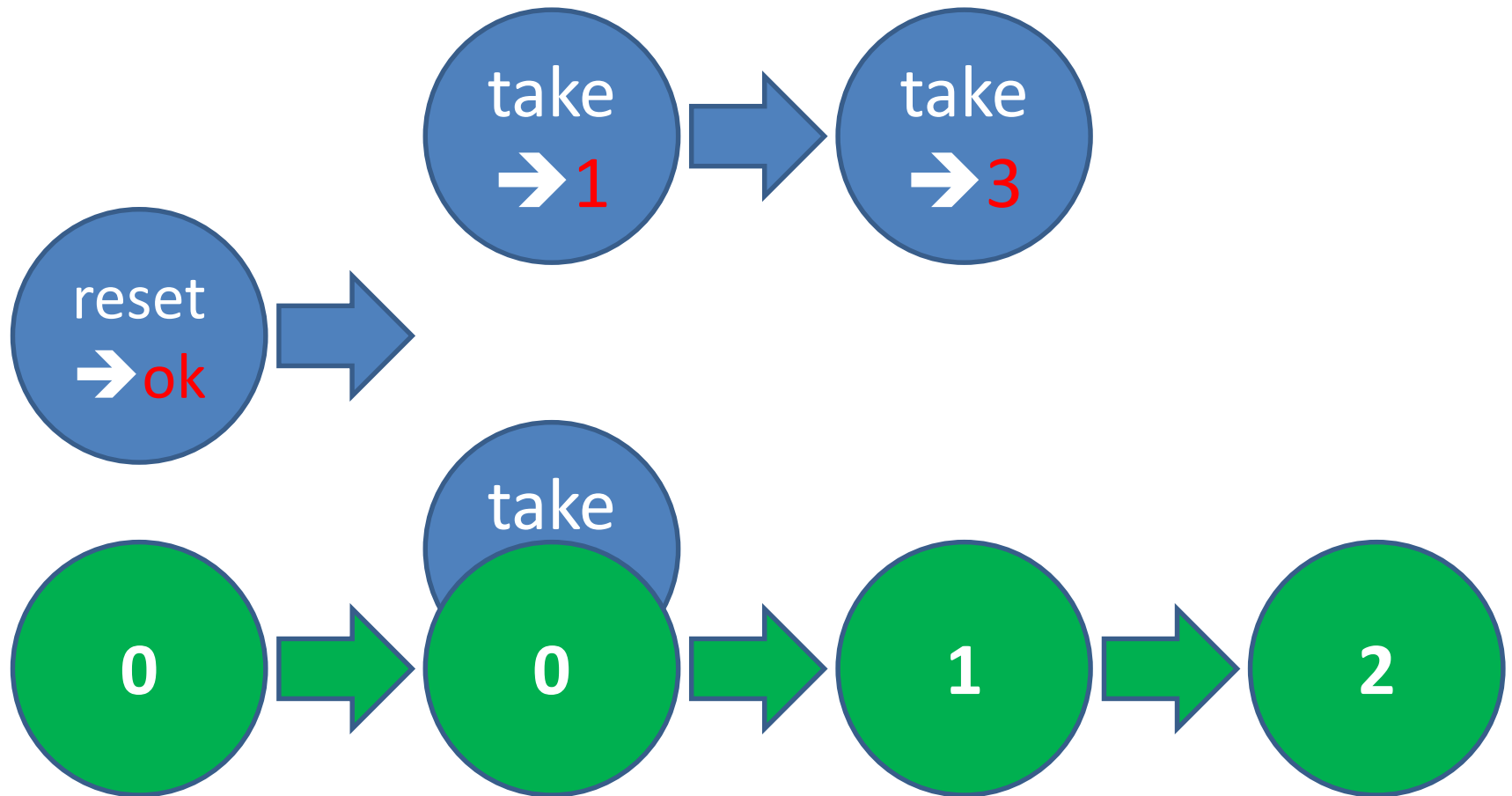
- Three possible correct outcomes!

Another Parallel Test



- 30 possible correct outcomes!

Deciding a Parallel Test



Let's run some tests

Prefix:

Parallel:

1. dispenser:take_ticket() --> 1

2. dispenser:take_ticket() --> 1

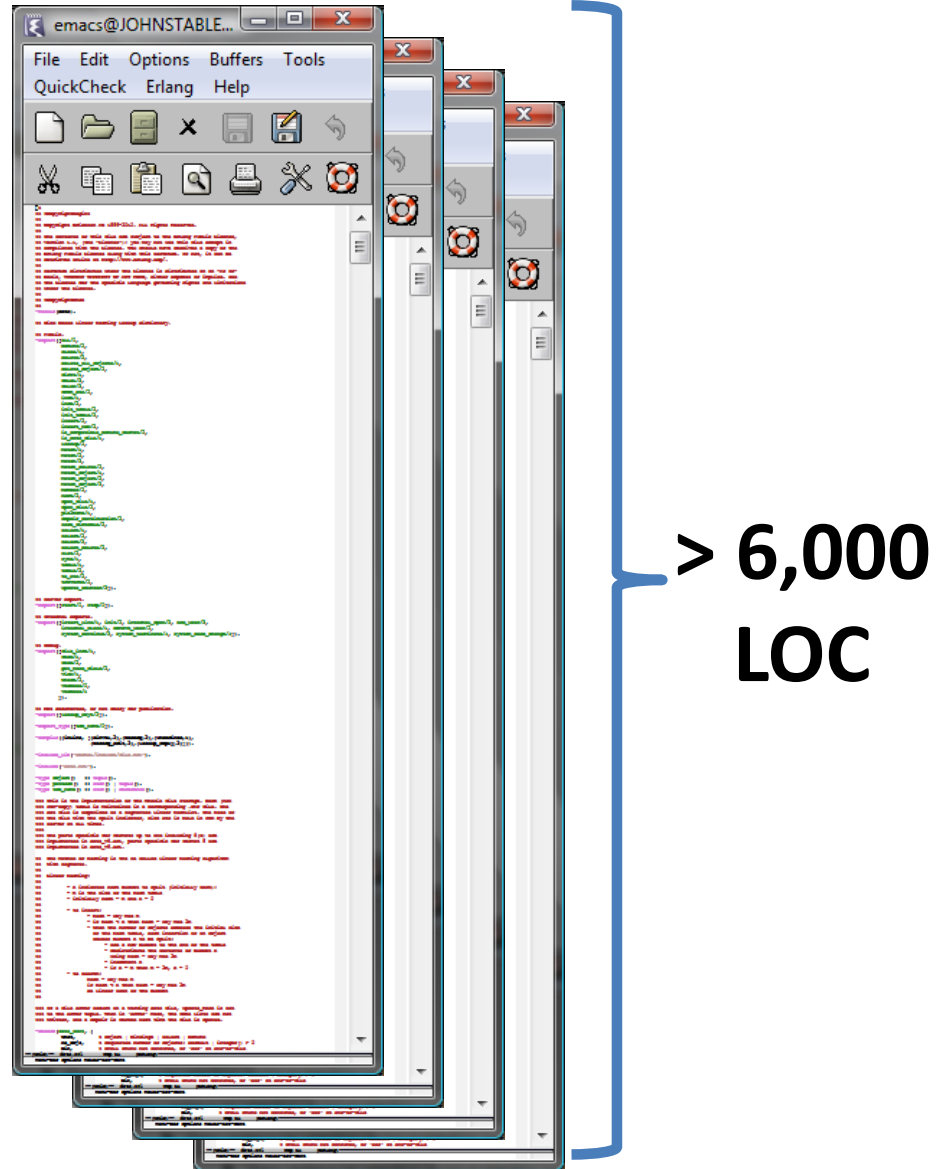
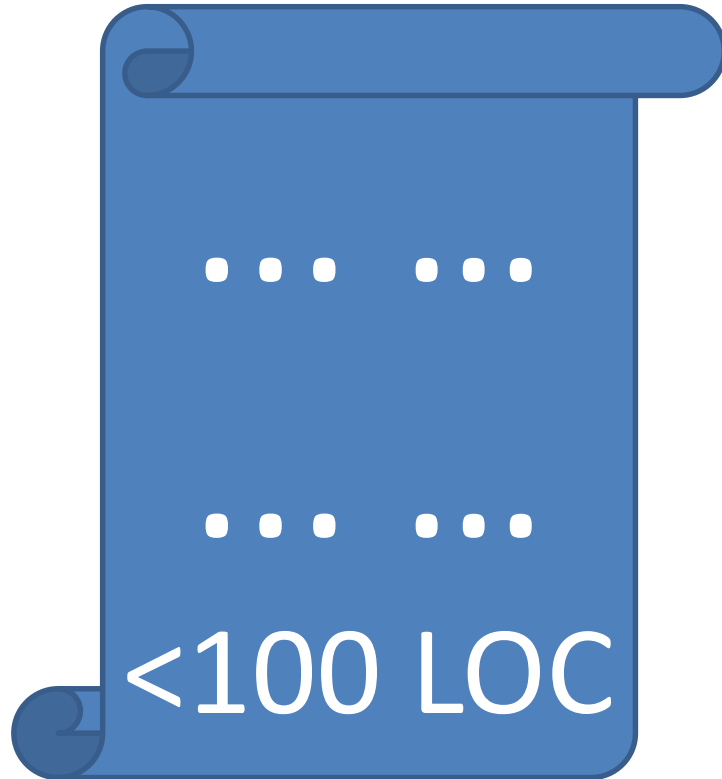
Result: no_possible_interleaving

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

dets

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

QuickCheck Specification



Bug #1

insert_new(Name, Objects) -> Bool

Prefix:

`open_file(dets`

Types:

Name = name()

Objects = object() | [object()]

Bool = bool()

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



premature eof

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

"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!

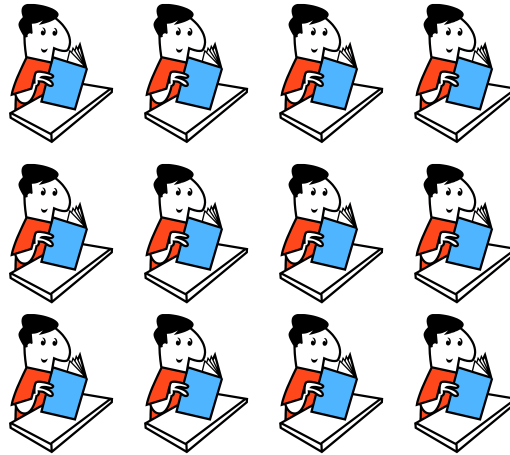
After



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

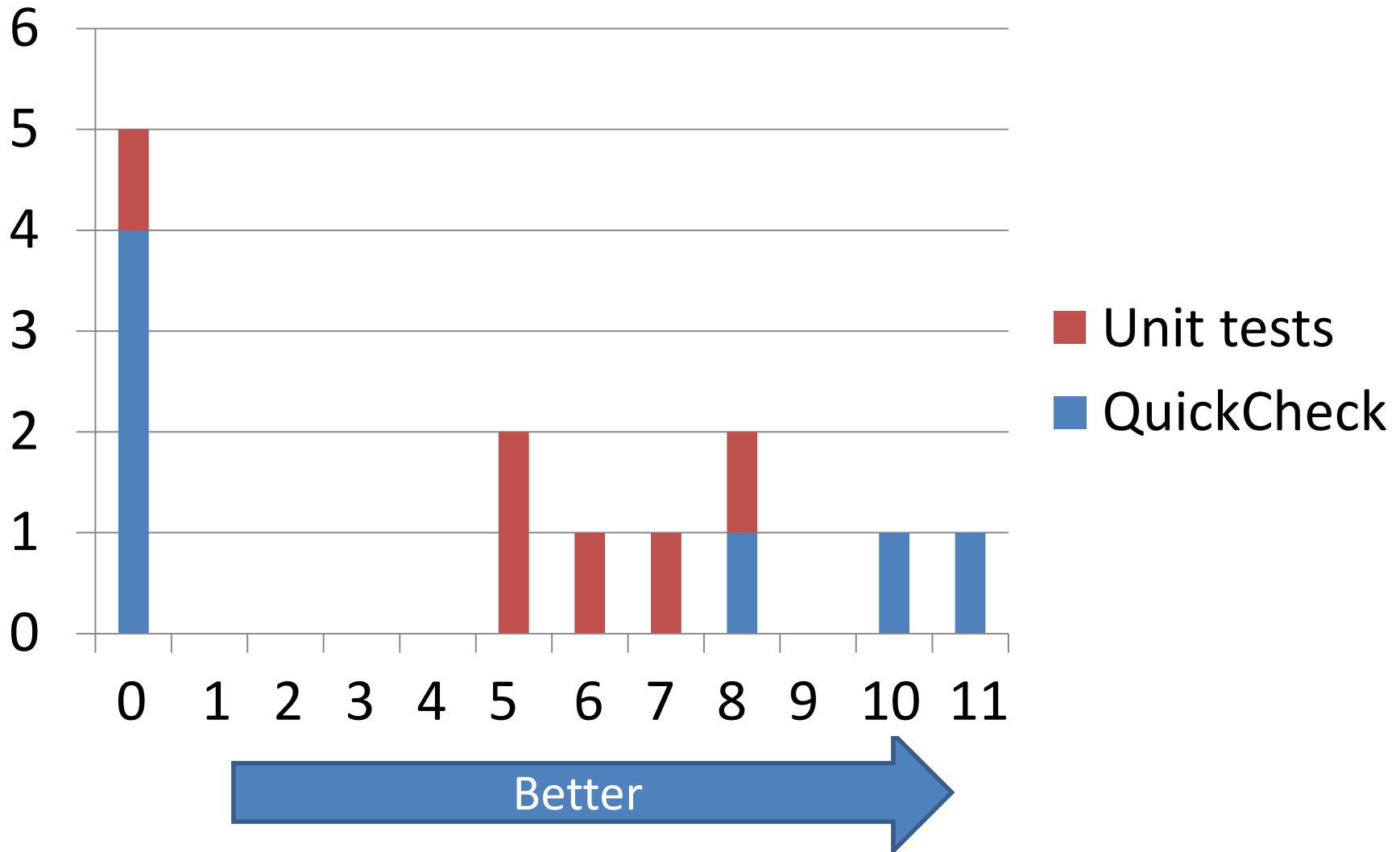
An Experiment

Unit
tests



Properties

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



Property Based Testing

...finds more bugs with less effort!

Don't *write* tests...

Generate them!

Where to find out more...

- Haskell QuickCheck
 - The original... ICFP 2000
- Quviq QuickCheck
 - Commercial tool & services www.quviq.com
- Many, many tools inspired by QuickCheck
 - test.check, ScalaCheck, FsCheck (F#)...