Specification Based Testing with QuickCheck

John Hughes

Chalmers University/Quviq AB

CHALMERS

QuviQ

ACM SIGPLAN

Most Influential 2000 ICFP Paper

"QuickCheck:
A Lightweight Tool for Random Testing of Haskell Programs"

by

Koen Claessen

John Hughes

What is QuickCheck?

- A *library* for *writing* and *testing* properties of program code
- **Some code:**

```
reverse([]) ->
    [];
reverse([X|Xs]) ->
    reverse(Xs) ++ [X].
```

- **A property:**

$$\forall xs.\ reverse(reverse(xs)) = xs$$

Properties as Code

$$\forall xs.\ reverse(reverse(xs)) = xs$$

```
prop_reverse() ->
    ?FORALL(Xs,list(int()),
            reverse(reverse(Xs)) == Xs).
```

A quantifier!

A macro!

A set!

A test data generator!

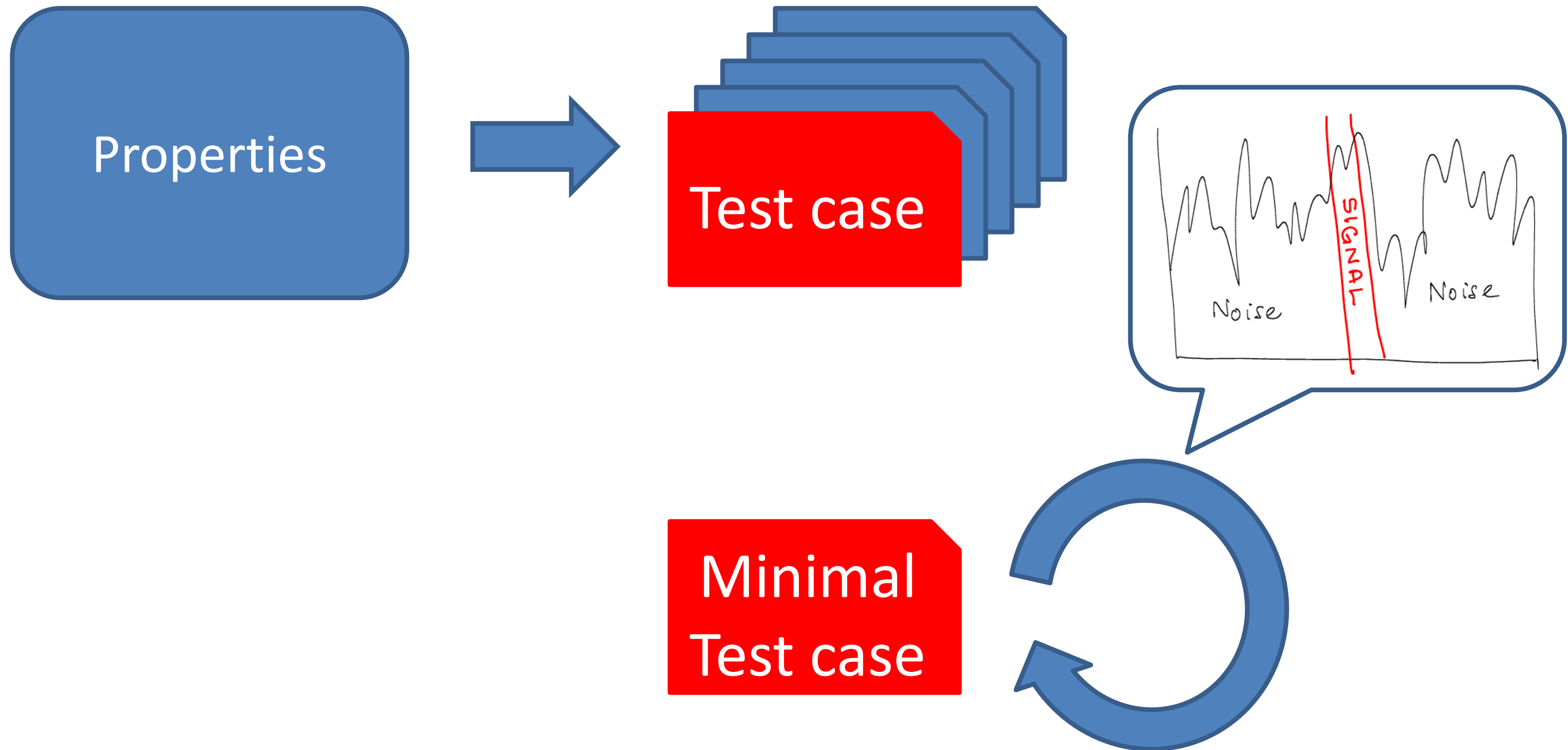
An ordinary function definition!

A predicate!

A boolean-valued expression!

DEMO

QuickCheck in a Nutshell



QuickCheck Properties: things with a counterexample

<bool-exp>

?FORALL(<var>,<generator>,<property>)

?IMPLIES(<bool-exp>,<property>)

conjunction, disjunction

?EXISTS(<var>,<generator>,<property>)

QuickCheck Generators

int(), bool(), real()…

choose(<int>,<int>)

{<generator>**,**<generator>…**}**

oneof(<list-of-generators>)

?LET(<var>,<generator>,<generator>)

Example: Sorted Lists

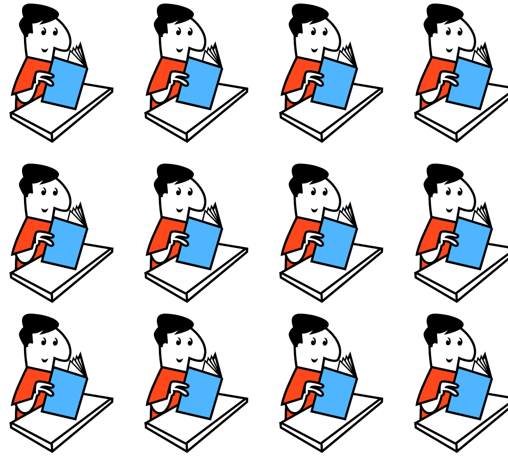
```
sorted_list_int() ->
    ?LET(L,list(int()),
           sort(L)).
```

Benefits

- Less time spent writing test code
  - One property replaces many tests
- Better testing
  - Lots of combinations you'd never test by hand
- Less time spent on diagnosis
  - Failures minimized automagically

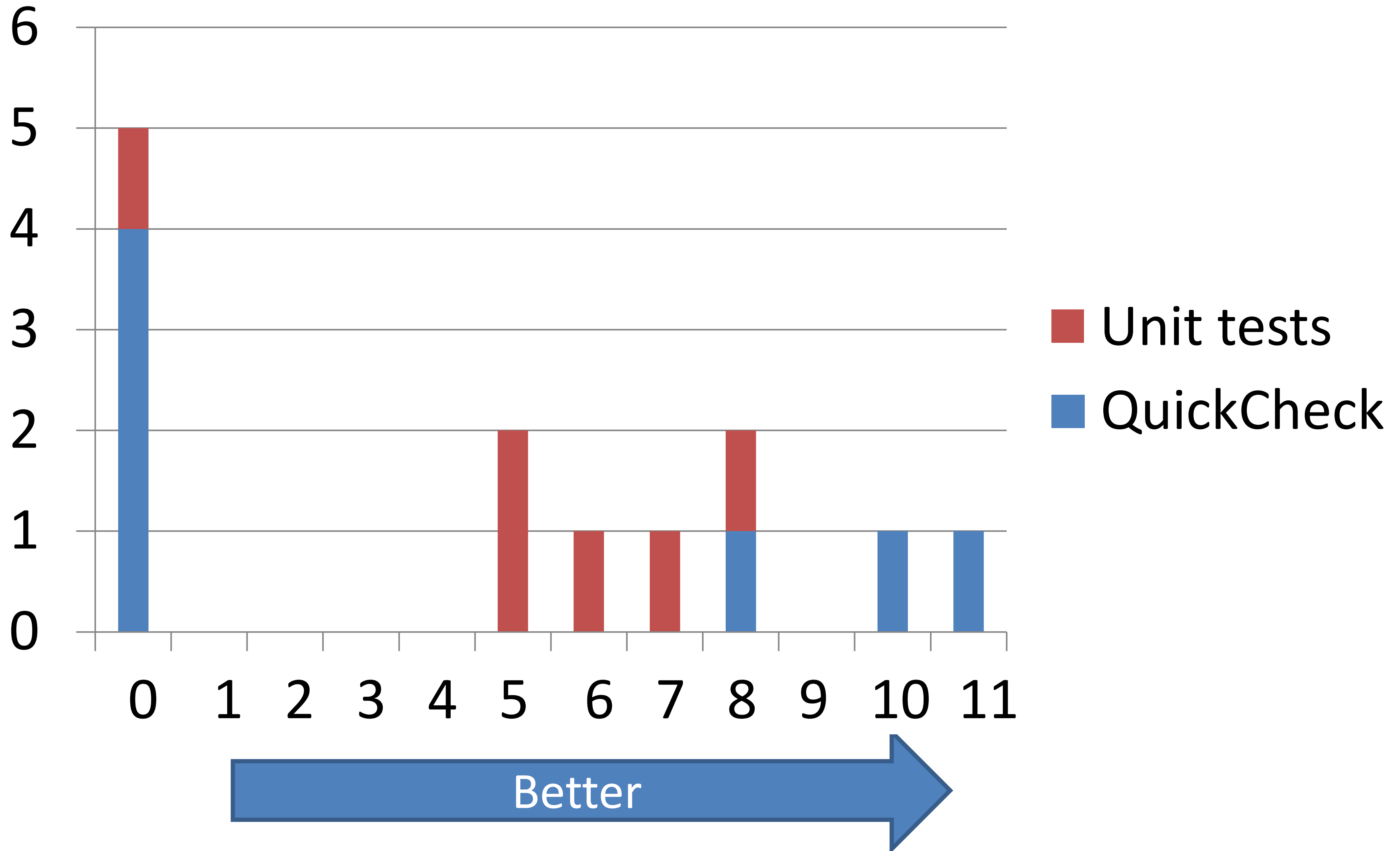
An Experiment

Unit tests



Properties

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



Tests for Base 64 encoding

Expected results

Test cases

```
base64_encode(Config) when is_
    %% Two pads
    <<"QWxhZGRpbjpvcGVuIHNlc2FtZQ==">> =
         base64:encode("Aladdin:open sesame"),

    %% One pad
    <<"SGVsbG8gV29ybGQ=">> = base64:encode(<<"Hello World">>),

    %% No pad
    "QWxhZGRpbjpvcGVuIHNlc2Ft" =
         base64:encode_to_string("Aladdin:open sesam"),

    "MDEyMzQ1Njc4OSFAIzBeJiooKTs6PD4sLiBbXXt9" =
         base64:encode_to_string(
              <<"0123456789!@#0^&*();:<>,. []{}">>),
    ok.
```

Writing a Property

```
prop_base64() ->
  ?FORALL(Data,list(choose(0,255)),
          base64:encode(Data) == ???).
```

Round-trip Properties

```
prop_encode_decode() ->
  ?FORALL(L,list(choose(0,255)),
    base64:decode(base64:encode(L))
      == list_to_binary(L)).
```

```
-define(DECODE_MAP,
          {bad,bad,bad,bad,bad,bad,bad,bad,ws,ws,bad,bad,ws,bad,bad,
           bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,
           ws,bad,bad,bad,bad,bad,bad,bad,bad,bad,62,bad,62,bad,bad,bad,63,
           52,53,54,55,56,57,58,59,60,61,bad,bad,bad,eq,bad,bad,
           ,13,14,
           ,bad,bad,bad,bad,bad,
           35,36,37,38,39,40,
           1,bad,bad,bad,bad,bad,
           ad,bad,bad,bad,bad,bad,bad,bad,bad,
           bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,
           bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,bad,
```

NOT caught by the test suite

Round-trip Properties

```
prop_encode_decode() ->
  ?FORALL(L,list(choose(0,255)),
    base64:decode(base64:encode(L))
      == list_to_binary(L)).
```

117> eqc:quickcheck(base64_eqc:prop_encode_decode()).

................................Failed! Reason: {'EXIT',{badarg,43}}

After 36 tests.

[204,15,130]

Shrinking...(3 times)

Reason: {'EXIT',{badarg,43}}

[0,0,62]

Round-trip Properties

```
prop_encode_decode() ->
  ?FORALL(L,list(choose(0,255)),
    base64:decode(base64:encode(L))
      == list_to_binary(L)).
```

What does this test?

- **NOT** a complete test—will not find a consistent misunderstanding of base64
- **WILL** find mistakes in encoder or decoder

Simple properties find a lot of bugs!

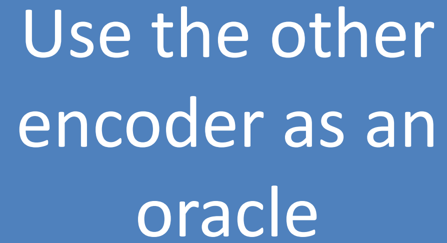
Back to the tests...

Where did these come from?

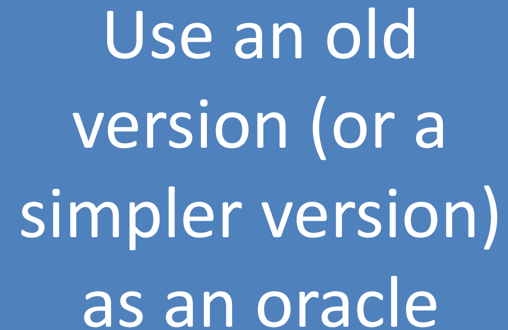
```
base64_encode(Config) when is_list(Config) ->
    %% Two pads
    <<"QWxhZGRpbjpvcGVuIHNlc2FtZQ==">> =
        base64:encode("Aladdin:open sesame"),

    %% One pad
    <<"SGVsbG8gV29ybGQ=">> = base64:encode(<<"Hello World">>),

    %% No pad
    "QWxhZGRpbjpvcGVuIHNlc2Ft" =
        base64:encode_to_string("Aladdin:open sesam"),

    "MDEyMzQ1Njc4OSFAIzBeJiooKTs6PD4sLiBbXXt9" =
        base64:encode_to_string(
            <<"0123456789!@#0^&*();:<>,. []{}">>),
    ok.
```

Possibilities

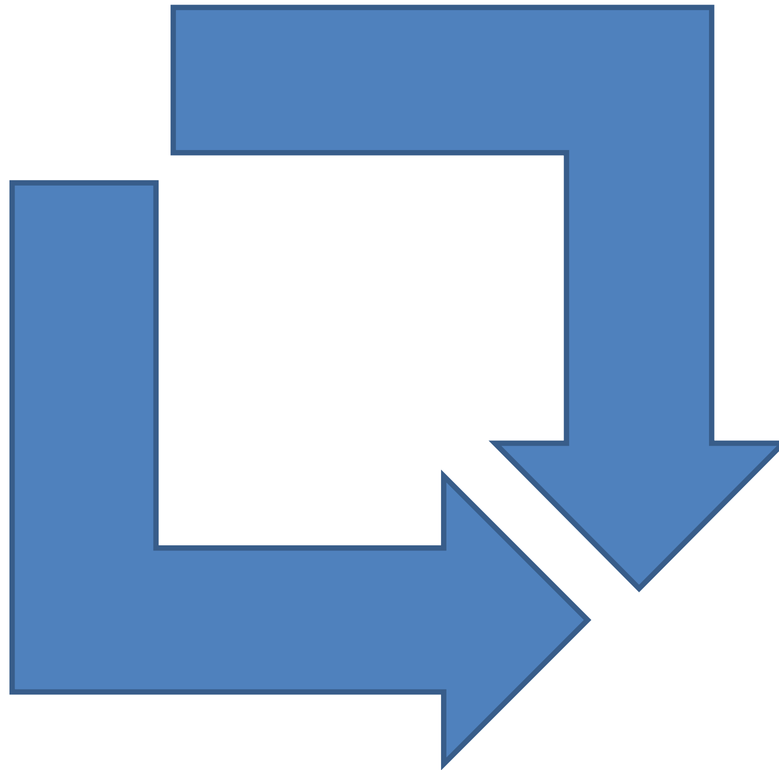
- Someone converted the data



- Another base64 encoder
- The same base64 encoder!
  - Only tests that changes don… t that the result is right

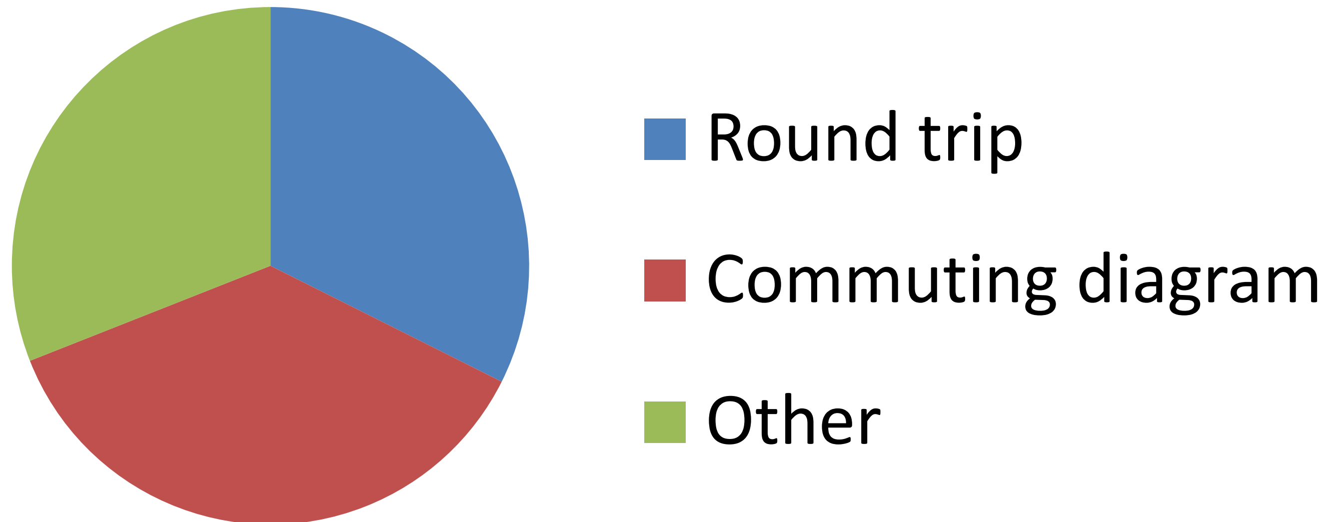


Commuting Diagram Properties



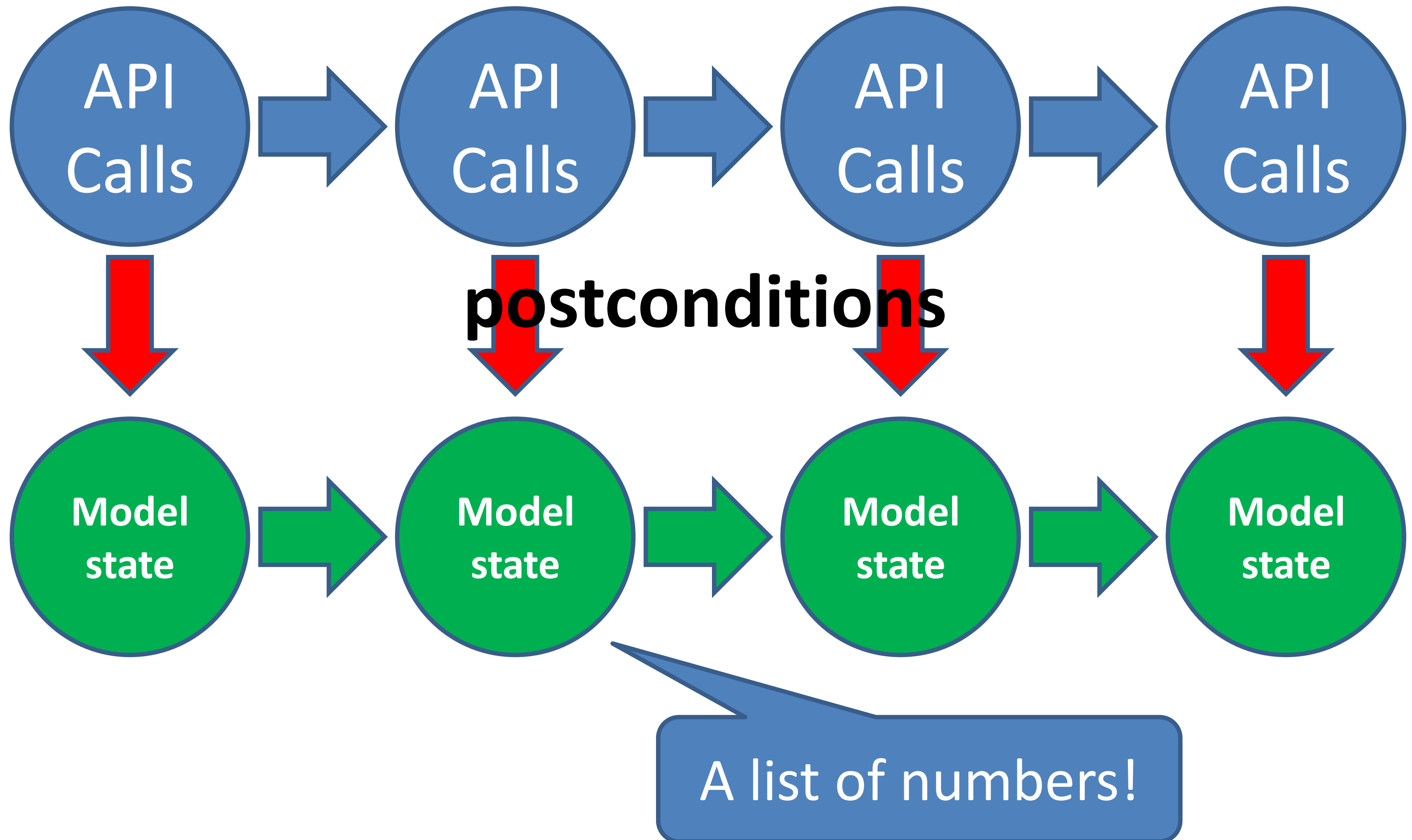
Property Types in Class Examples

- Rex Page: 71 properties in University of Oklahoma courses in Software Engineering, Applied Logic (QuickCheck+ACL2)



Time for some C code…

Testing Stateful Code



A QuickCheck Property

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

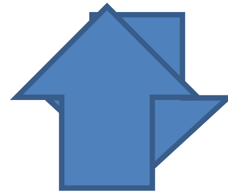
Let’s run some tests…

Exercises → Practice

Small scale	→	Large scale
Property-driven development	→	Testing legacy code
Trivial inputs	→	Complex inputs

Example: Ericsson Media Proxy

Lots of work to write generators

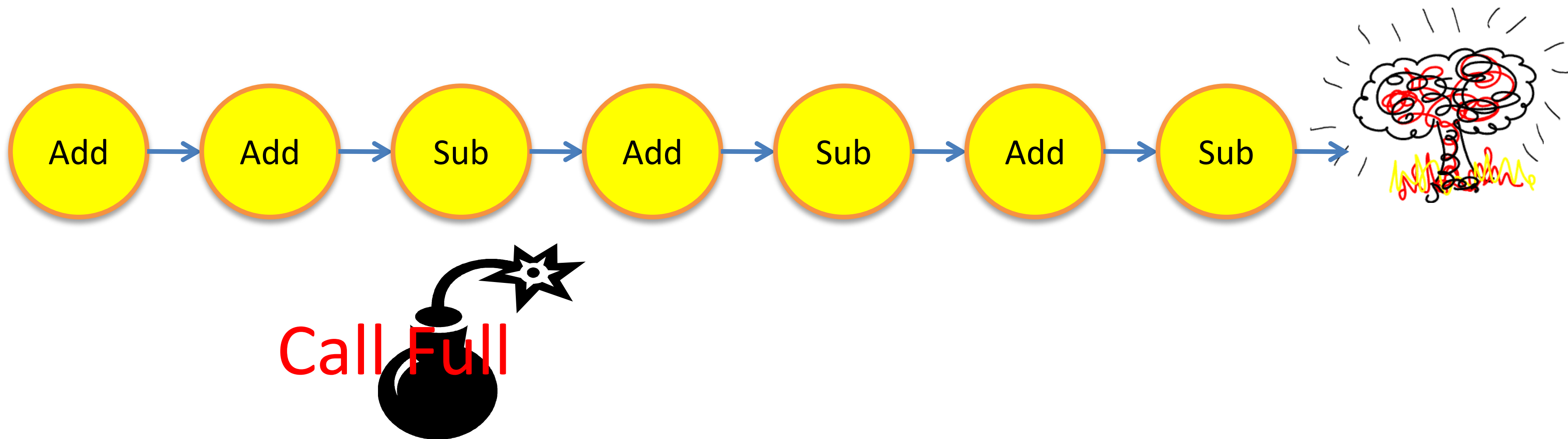


Many, many parameters, can be 1—2 *pages* per message!

State machine models fit the problem well

Ericsson Media Proxy Bug

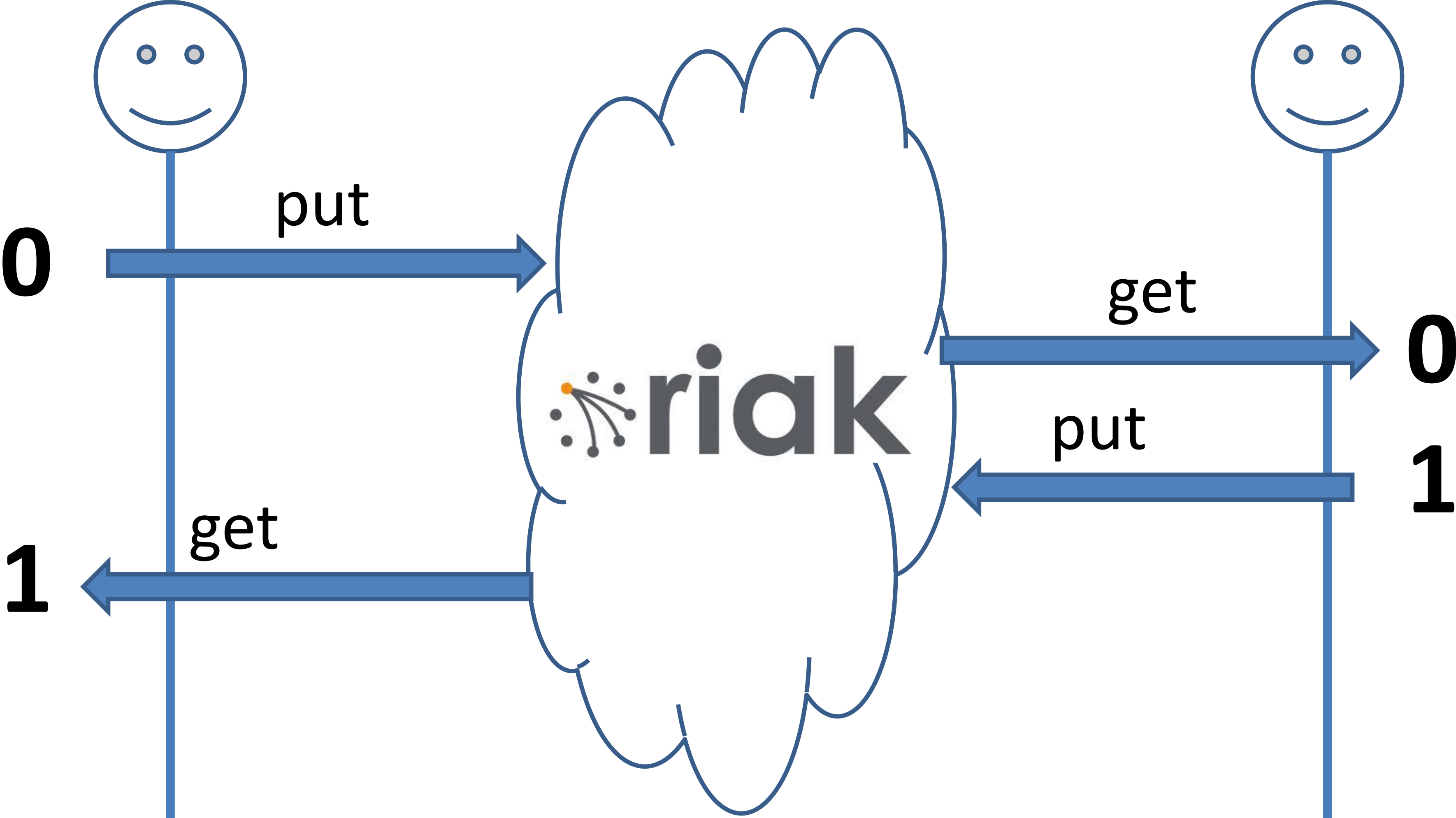
- Test adding and removing callers from a call



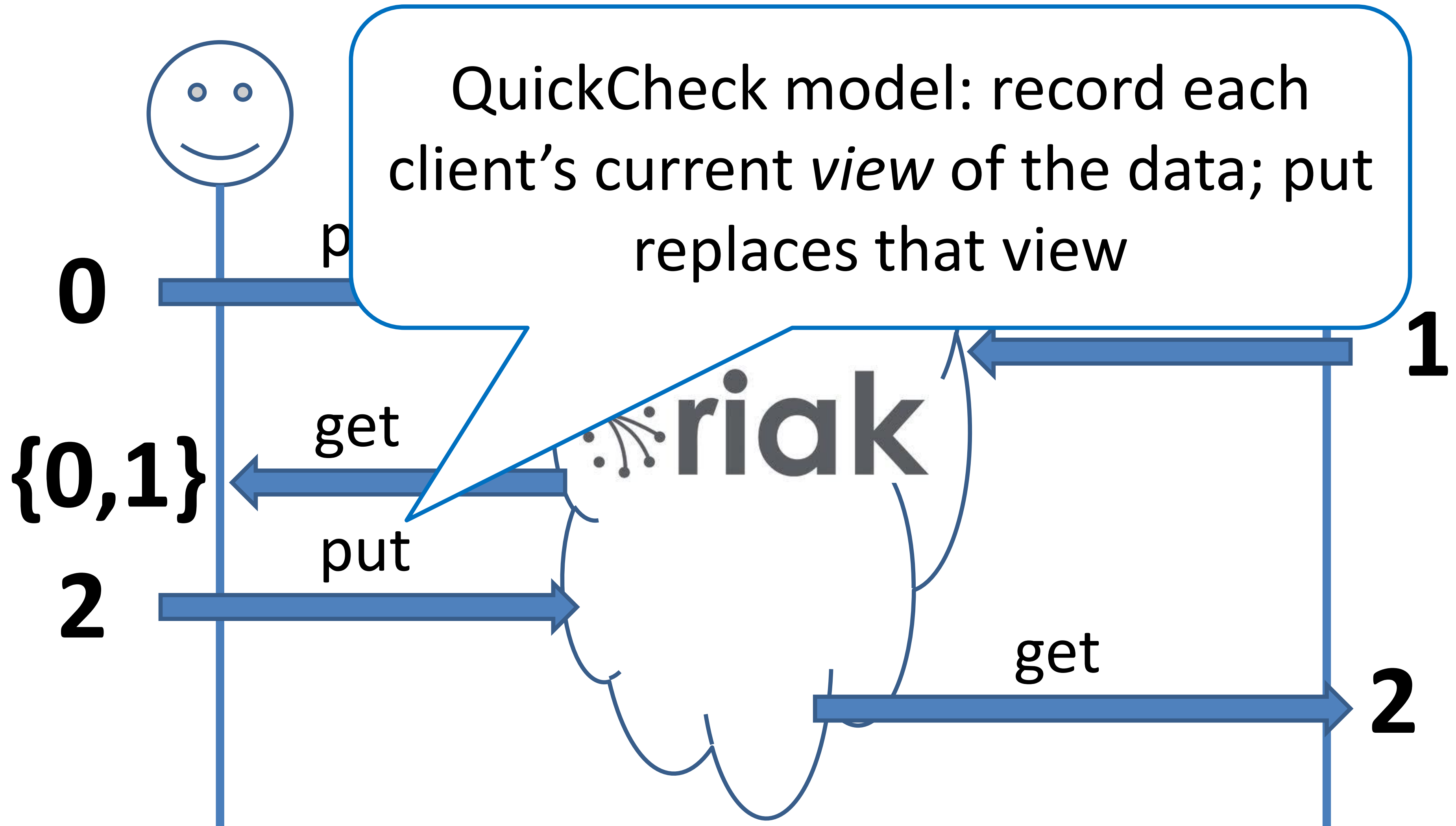
- Relational databases … ’t scale to ”Big Data”
- ”noS … lar alternative

A highly scalable, reliable, available and low-latency distributed key-value store

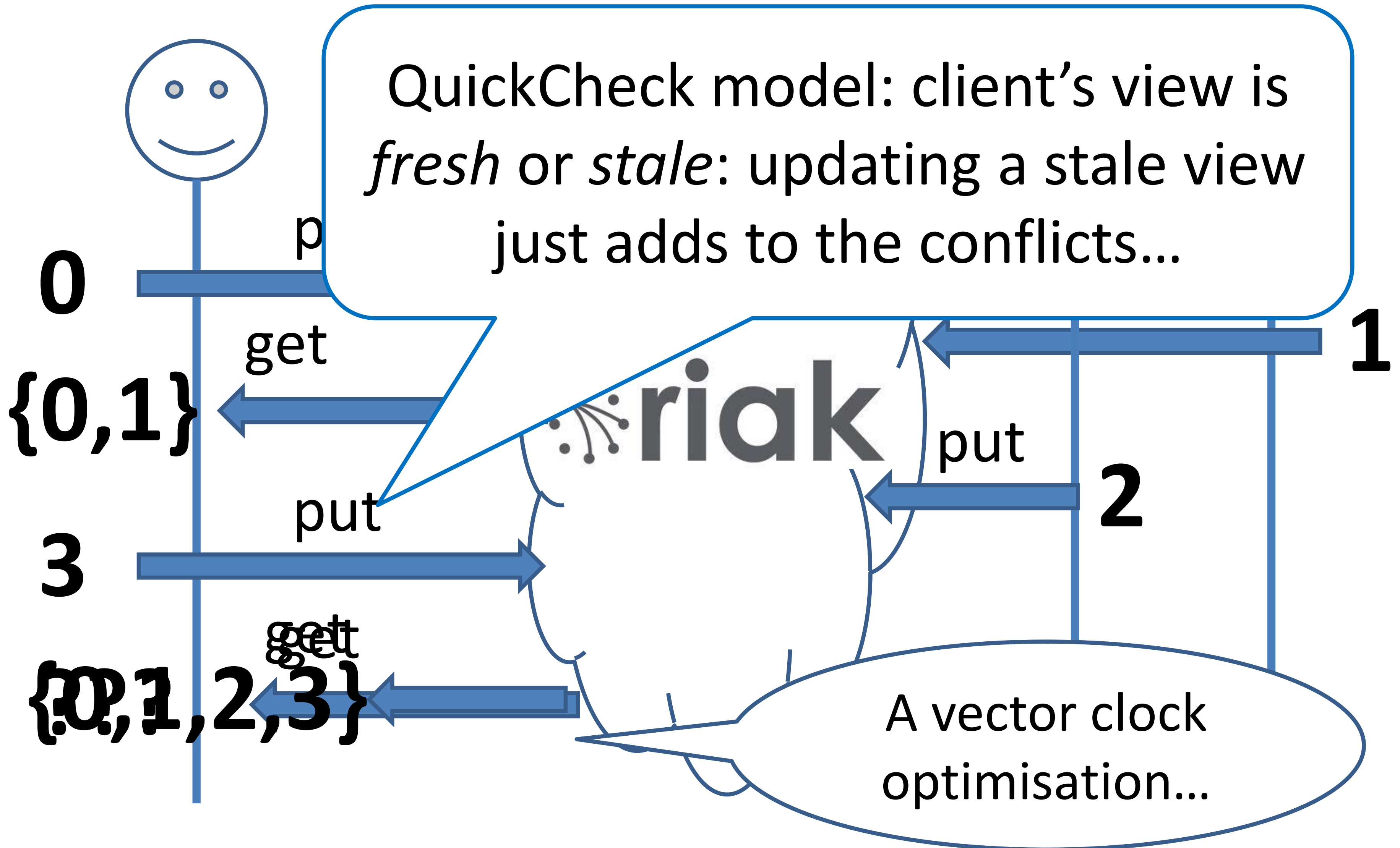
Put and Get



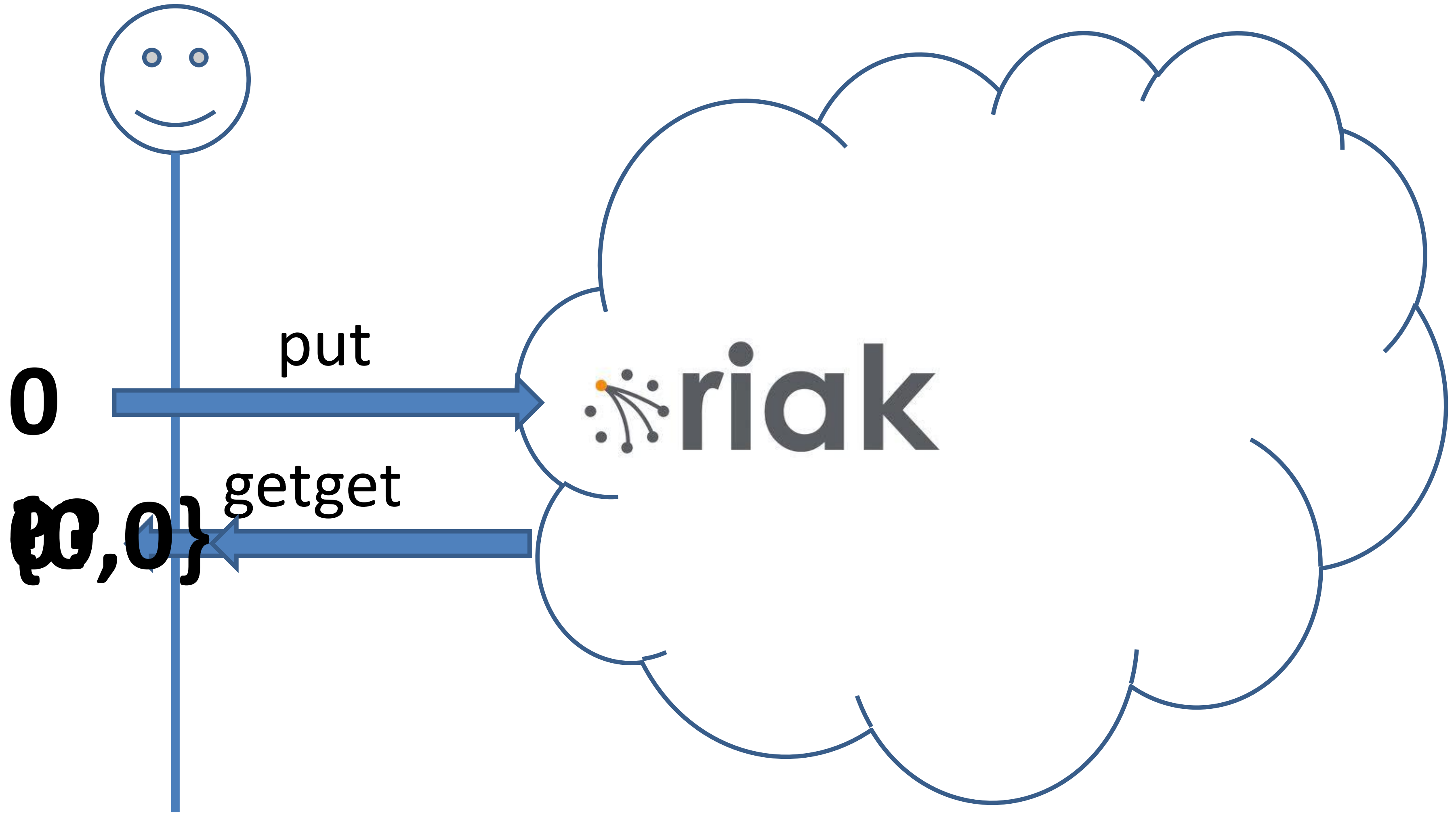
Conflicts



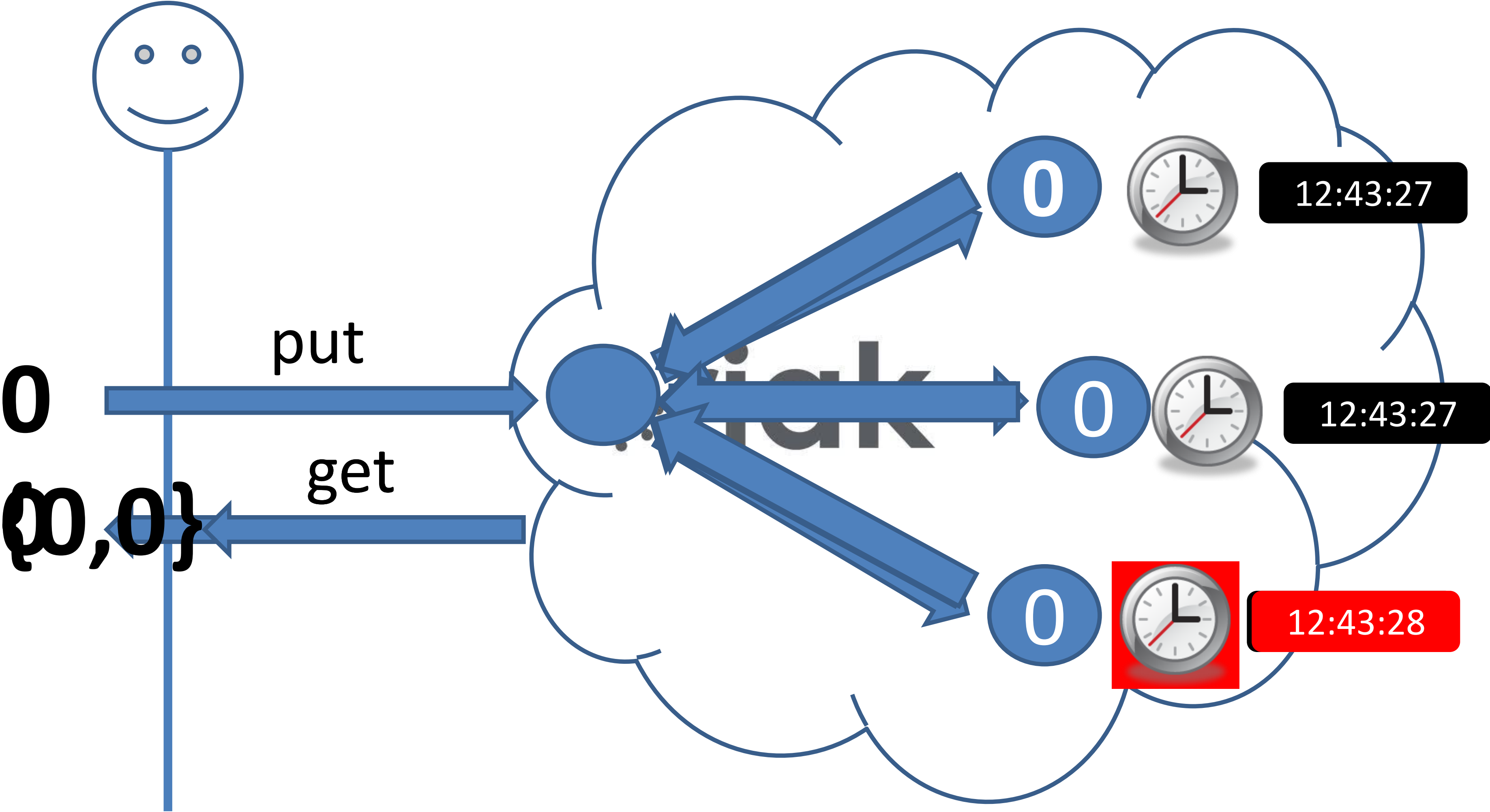
Example



Example



Duplicate value explained



Eventual Consistency

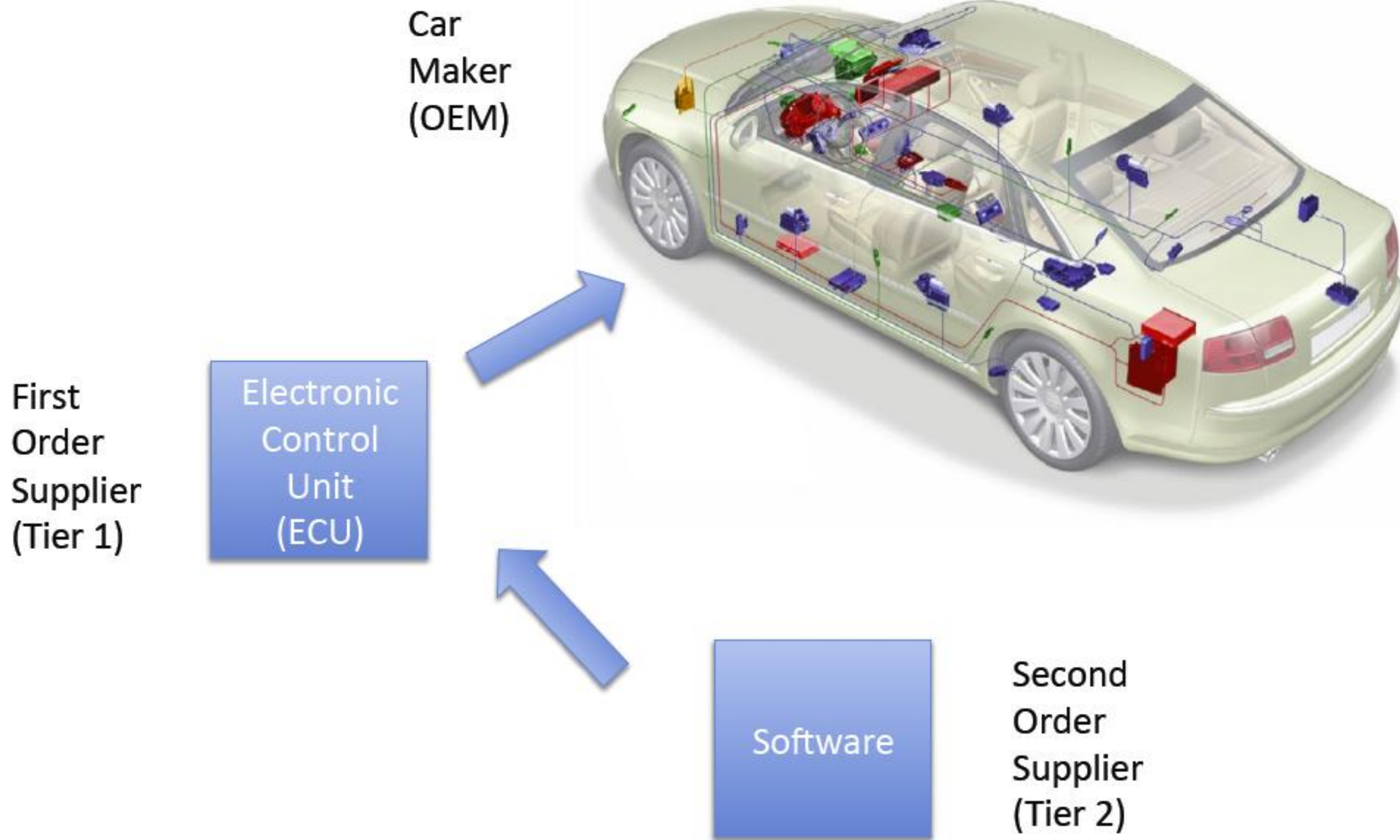
- ”For any sequence of operations, with any node or network failures, Riak *eventually* reaches a consistent state”
  - When is ”eventually”?

- For any sequence of operations sent to any subsets of server nodes (because of failures), completing all Riak’s repair operations results in a consistent state.

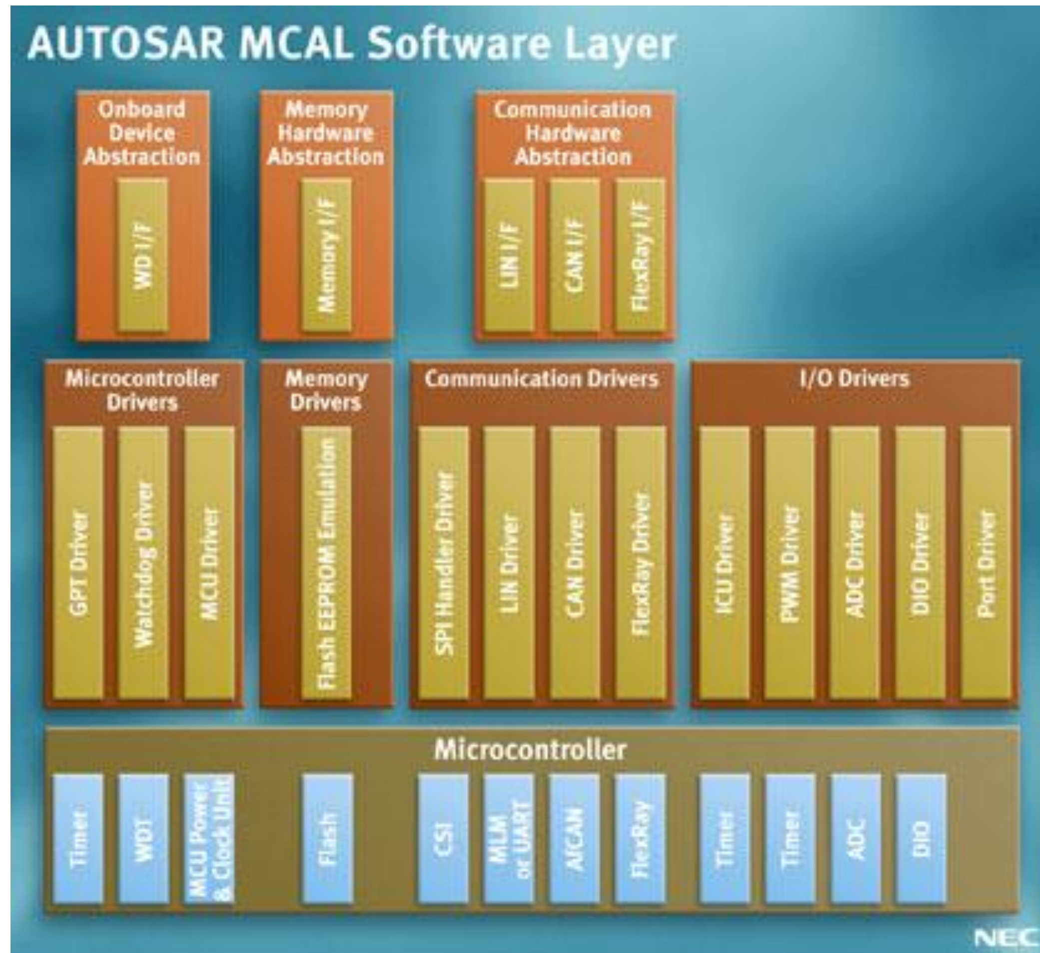
AutoSAR

- Joint project with Quviq, SP, Volvo Cars, Mentor Graphics…

Is the software in different ECUs compatible?



AutoSAR Basic Software



The Story So Far…

- QuickCheck state-machine models for 3 AutoSAR clusters (Com/PDUR, CAN, FlexRay)
- Used to test software from 3 suppliers
- Bugs revealed in all!
  - Plus reinterpretations of the standard

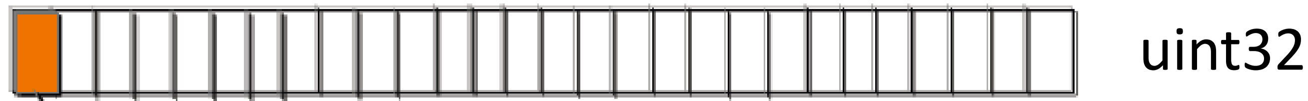
Example: Mixed features

Priority: lowest number has highest priority

Example:
Extended Id 113 has higher priority than standard Id 114

Buffered higher priority messages should be sent first

Example: Mixed features



1 extended
0 standard

```
transmit,[1,112,[67],'CAN_OK'],
transmit,[2,113,[0],'CAN_BUSY'],
transmit,[3,114,[0],'CAN_BUSY'],
tx_confirmation,[1,112,[67]]
```

Force buffering

Trigger sending

Check callouts: 112, 114 sent, why?

COM Component

- 500 pages of standard
- 250 pages of C
- 25 pages of QuickCheck

"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

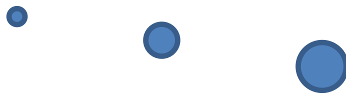
What is it?

>500 people in 5 years

Invoicing services for web shops

Distributed database: transactions, distribution, replication

Tuple storage

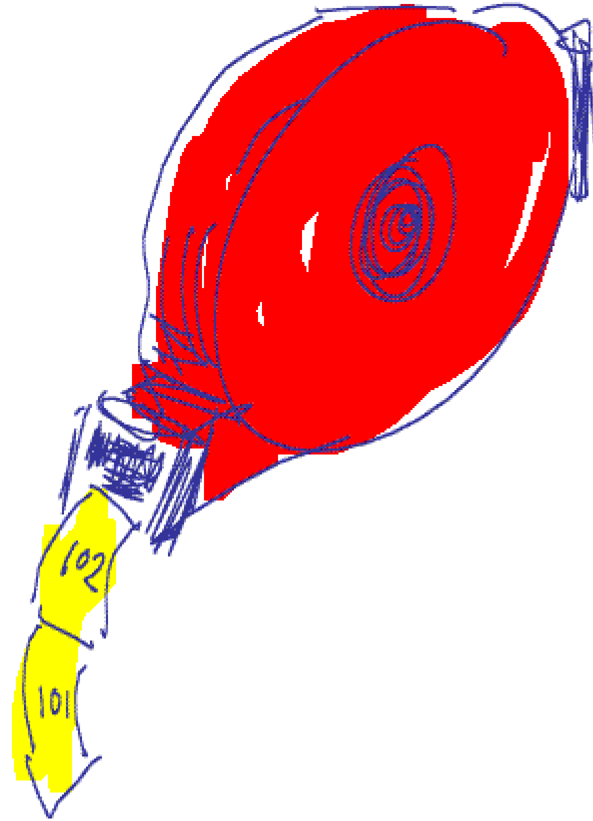


Race conditions?

Imagine Testing This…

dispenser:take_ticket()

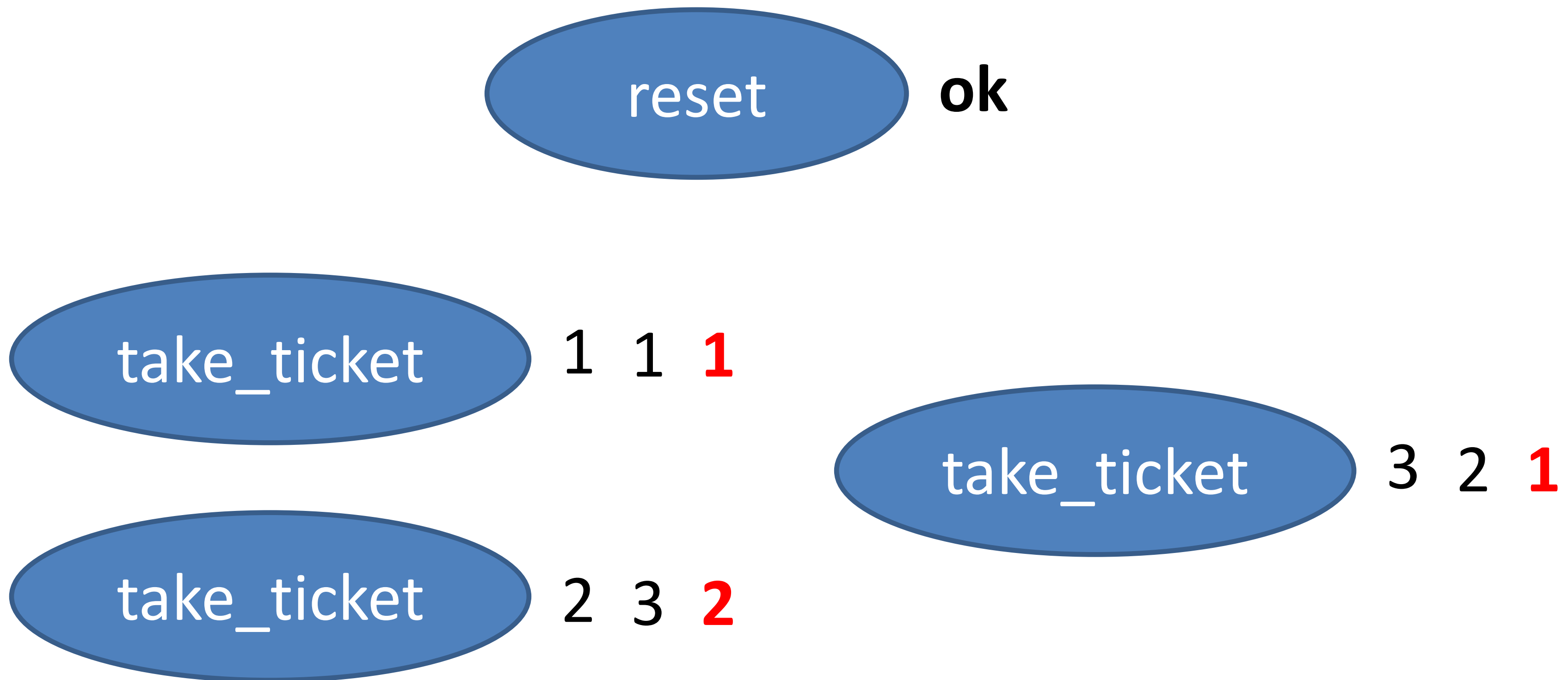
dispenser:reset()



A Unit Test in Erlang

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

A Parallel Unit Test



- Three possible correct outcomes!

Another Paral l T

A killer app
for
properties!

take_ticket

take_ticket

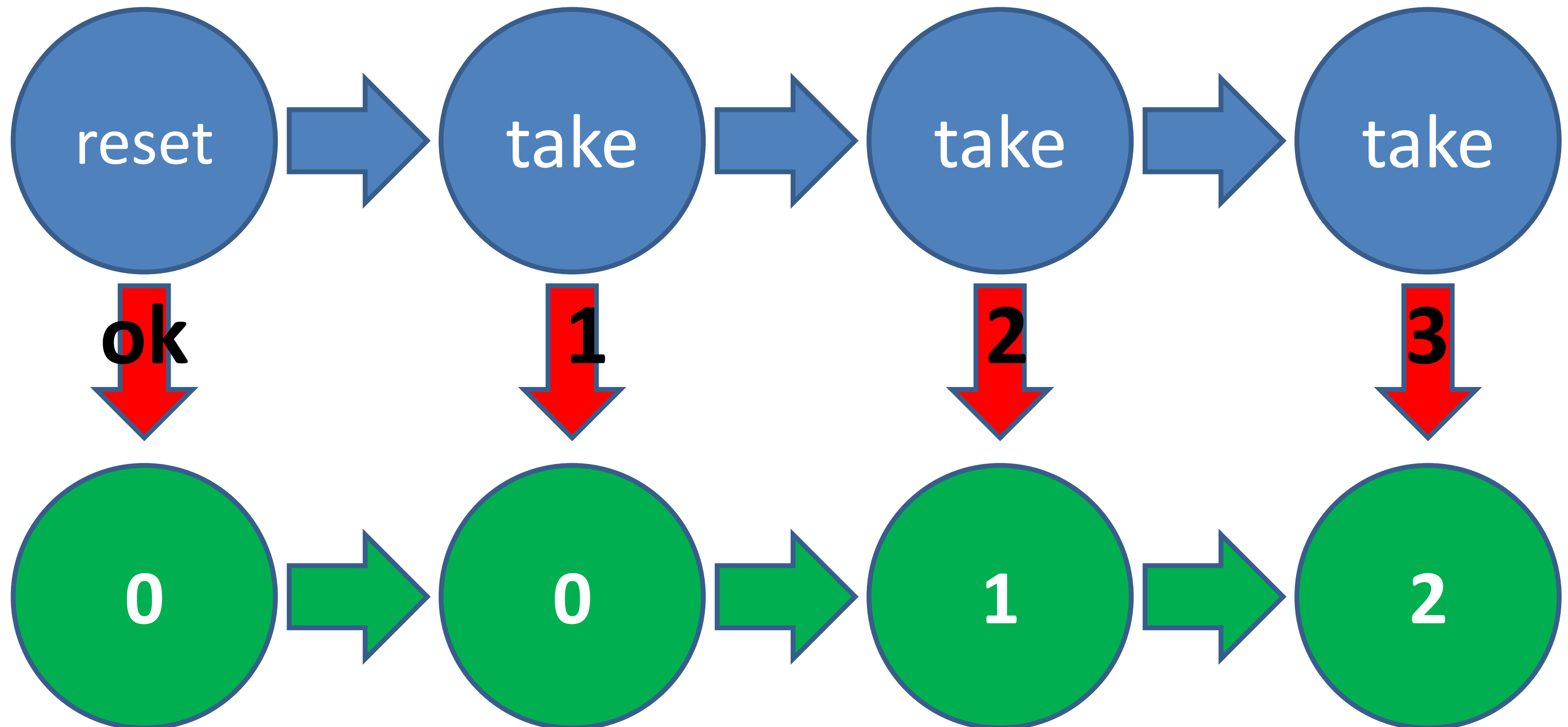
reset

take_ticket

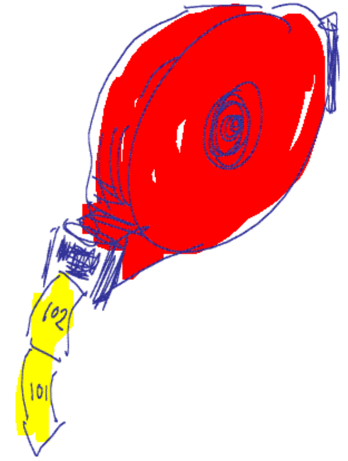
take_ticket

- 42 possible correct outcomes!

Modelling the dispenser



The Model



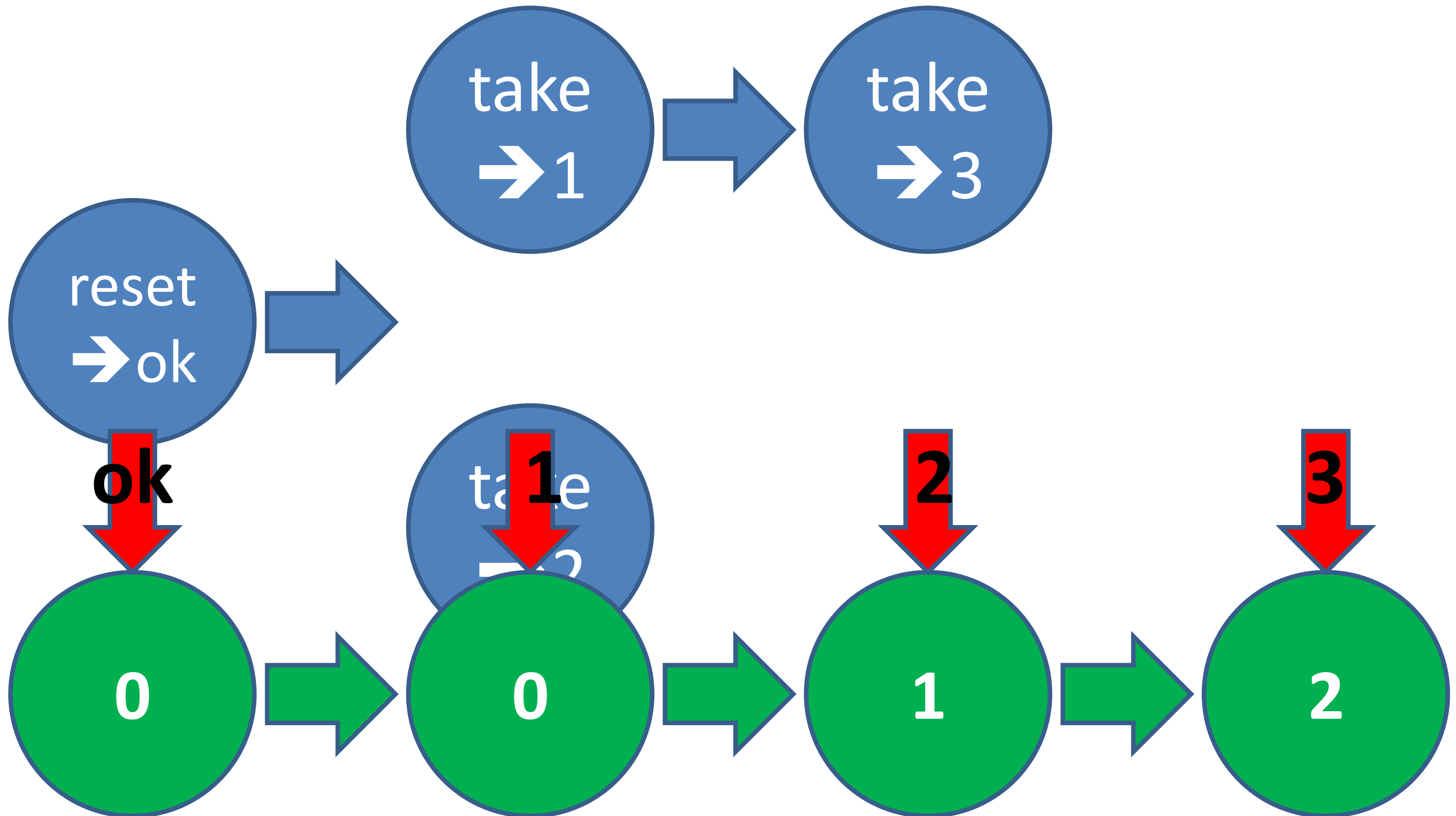
- State transitions

```
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;
```

Parallel Test Cases



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

Generate parallel test cases

Run tests, check for a matching serialization

DEMO

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

Prefix:

Parallel:

1. take_ticket() --> 1

2. take_ticket() --> 1

Result: no_possible_interleaving

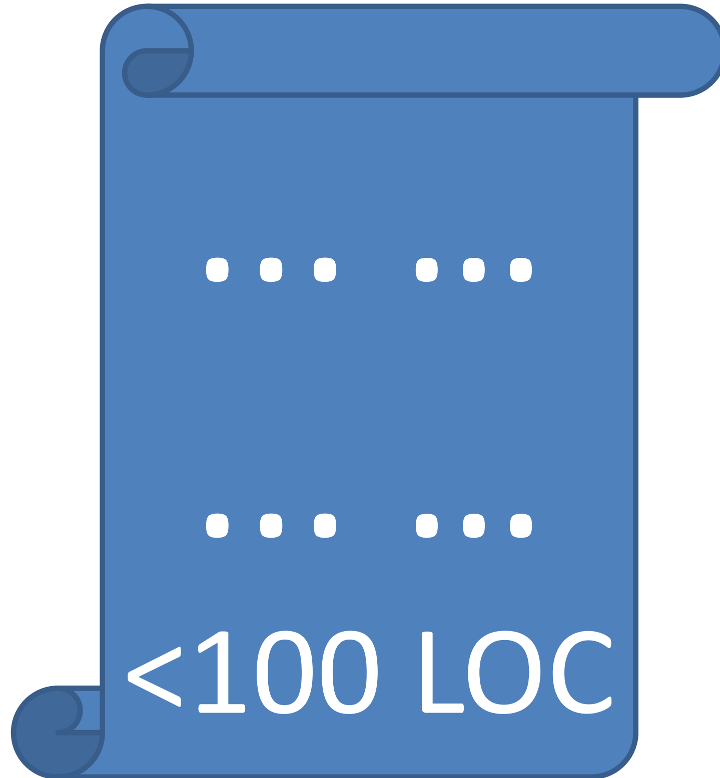
dets

- 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

DEMO

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
```

What’s going on?

Reordering and concurrency!

Is the file corrupt?

Bug #4

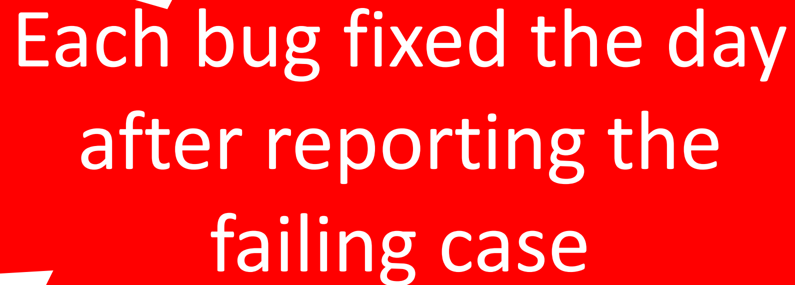
```
Prefix:
   open_file(dets_table,[{type,bag}]) --> dets_table
   close(dets_table) --> ok
   open_file(dets_table,[{type,bag}]) --> dets_table

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."

T[illegible] Tö[illegible]st, Kl[illegible]na, 2007



How come?

- The bugs weren't found earlier?
  - despite > 6 weeks of work
- Hypotheses
  - …files of over 1GB?
  - …rehashing could be the problem?
  - Diagnosing races in production is hopeless
- The bugs weren't found in testing?
  - Unit tests for races are hard to write…so people don't!
  - Races=feature interaction ➔ impractically many tests

Race conditions should be found by *unit testing* with *generated tests*

Reflections

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

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

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

QuickCheck…

- …is very widely applicable
- …almost always finds bugs in real systems!
- …is particularly good at spotting *interactions* that conventional test cases miss
- …makes diagnosis simple by *shrinking*
- …makes testing more intellectually challenging *and fun!!*