QuickCheck decreasing your test effort by a factor five

Thomas Arts

Chalmers

Message

Software systems more complex every day...

... features to be released faster

... quality needs to increase

Traditional testing insufficient to keep up with this

We need to change our testing methods!

From test case to property

Instead of specifying one or two test cases to demonstrate that the software fulfills a certain property, we specify ***the property*** and have the tests automatically generated!

Approach is based upon: *controlled random generation of test cases*

On different abstraction level

Model based testing

Model

We specify our model of the software in a programming language called Erlang

We specify the property to hold also in Erlang

Test cases are generated and run against C code

Failing tests are automatically reduced to shorter test case

SMS encode/decode

Encoding SMS messages

GSM standard Technical Specification **03.38**

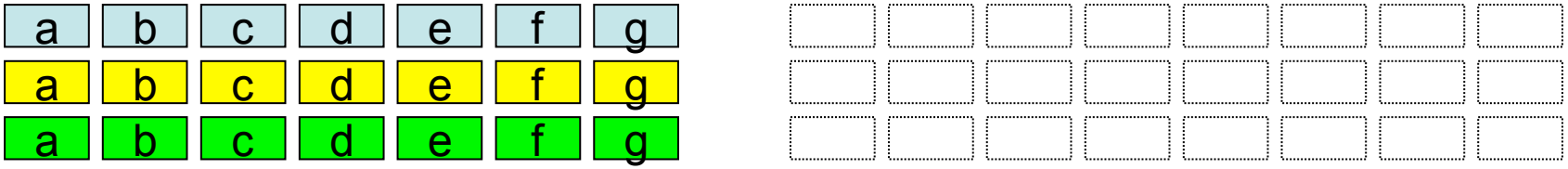
Packing of 7 bit characters

Example

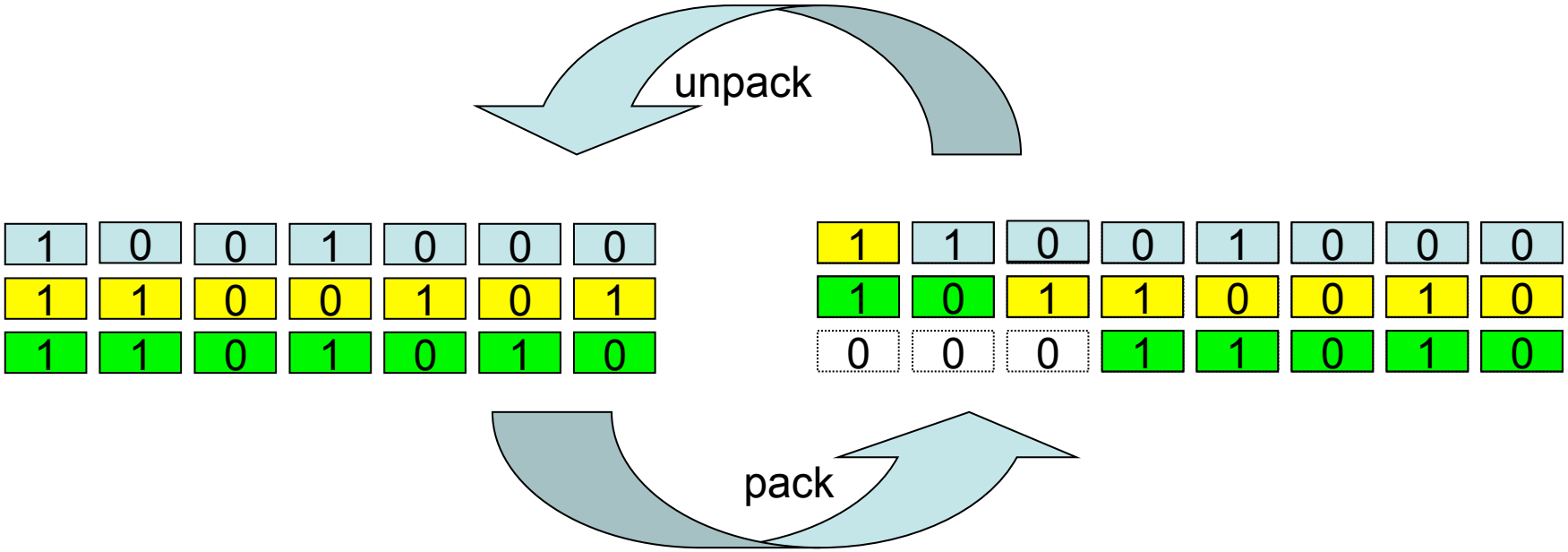
Interpretation

3 characters containing 7 bits

packets 8 bits



Example



“Hej” = [72,101,106]

“È²^Z” = [200,178,26]

DEMO

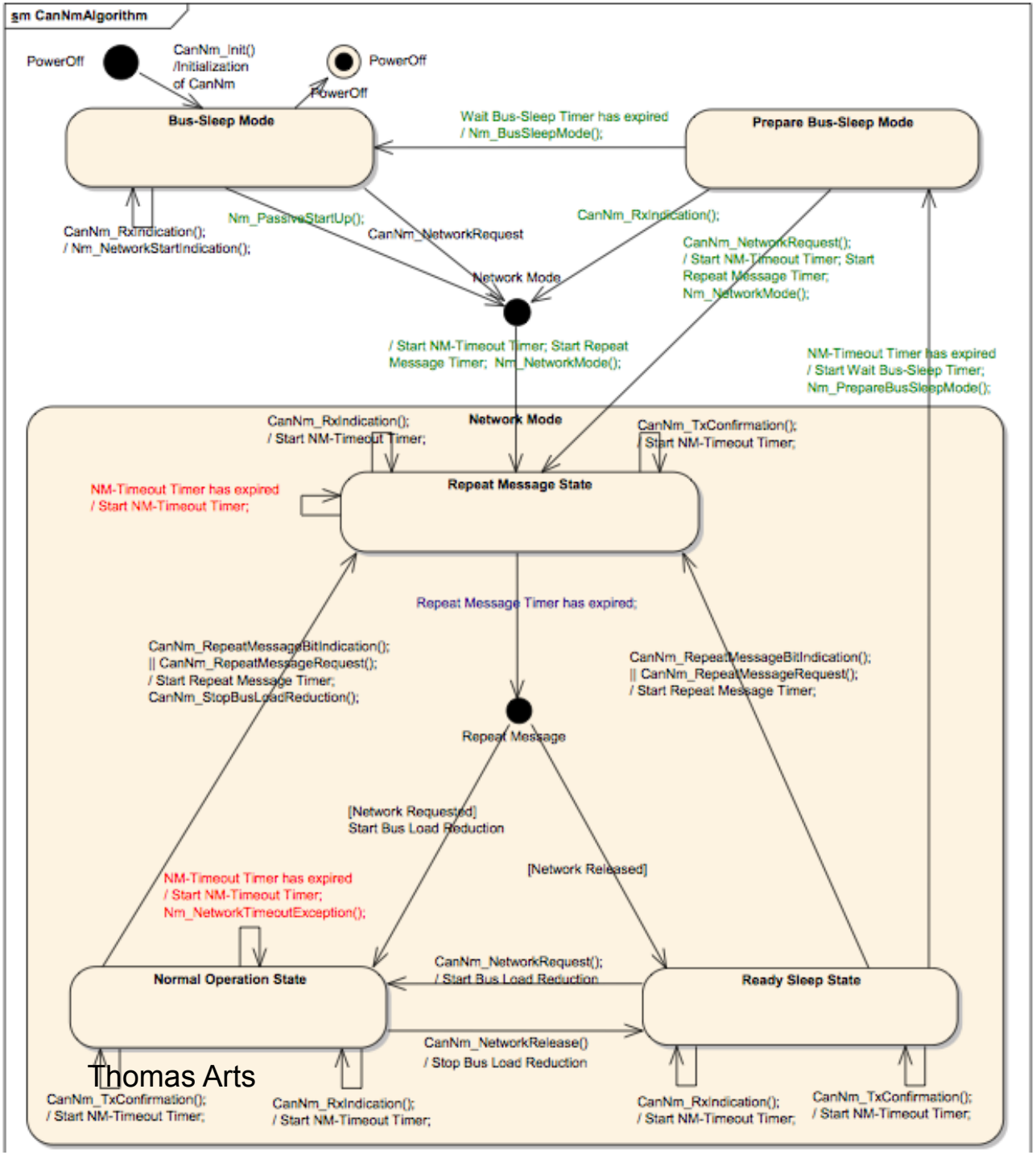
Real software is not as simple ☺

- Ejabberd (XMPP server)
- Ericsson's Media gateway (IMS)
- Ericsson's SIP stack (IP telephony)
- Radio base station for 3G
- Embedded printer software
- SMS/Email gateway
- AUTOSAR components
- ....

Testing effort from 20 to 2 weeks!
Model tests more.
For next iteration, only 1 or 2 days needed

AUTOSAR component
as UML state machine

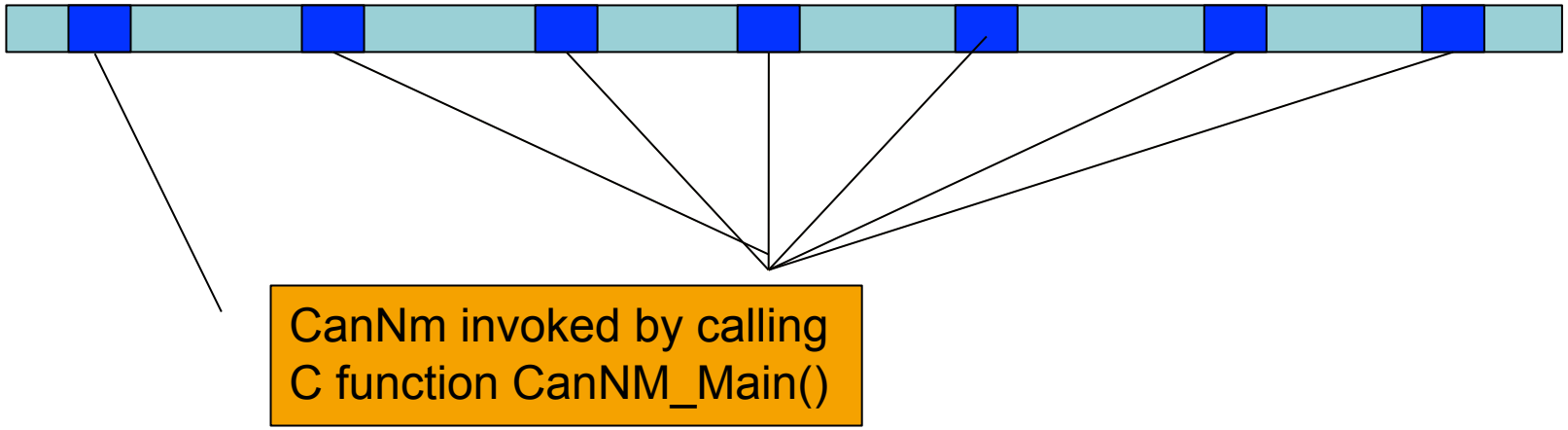
CAN
Network Management



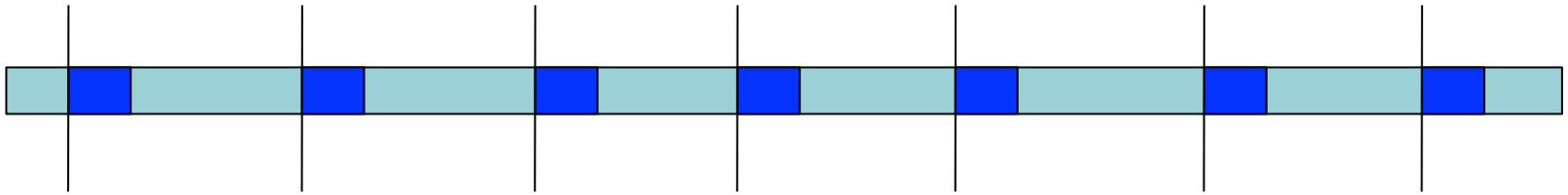
October 14, 2010

Thomas Arts

CanNM is scheduled as one of many tasks



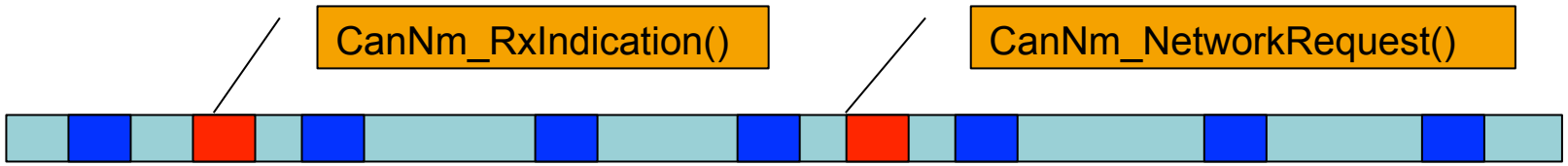
CanNM is scheduled as one of many tasks



Assumption:
One time unit elapses before CanNm_Main() is called

(In fact, C implementation handles the timers, not the scheduler)

CanNM is scheduled as one of many tasks

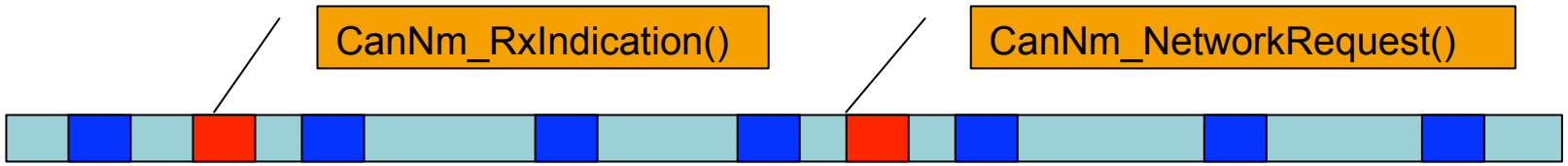


Other tasks communicate by calling CanNM interface functions

These update data structures in memory

Scheduling

CanNM is scheduled as one of many tasks



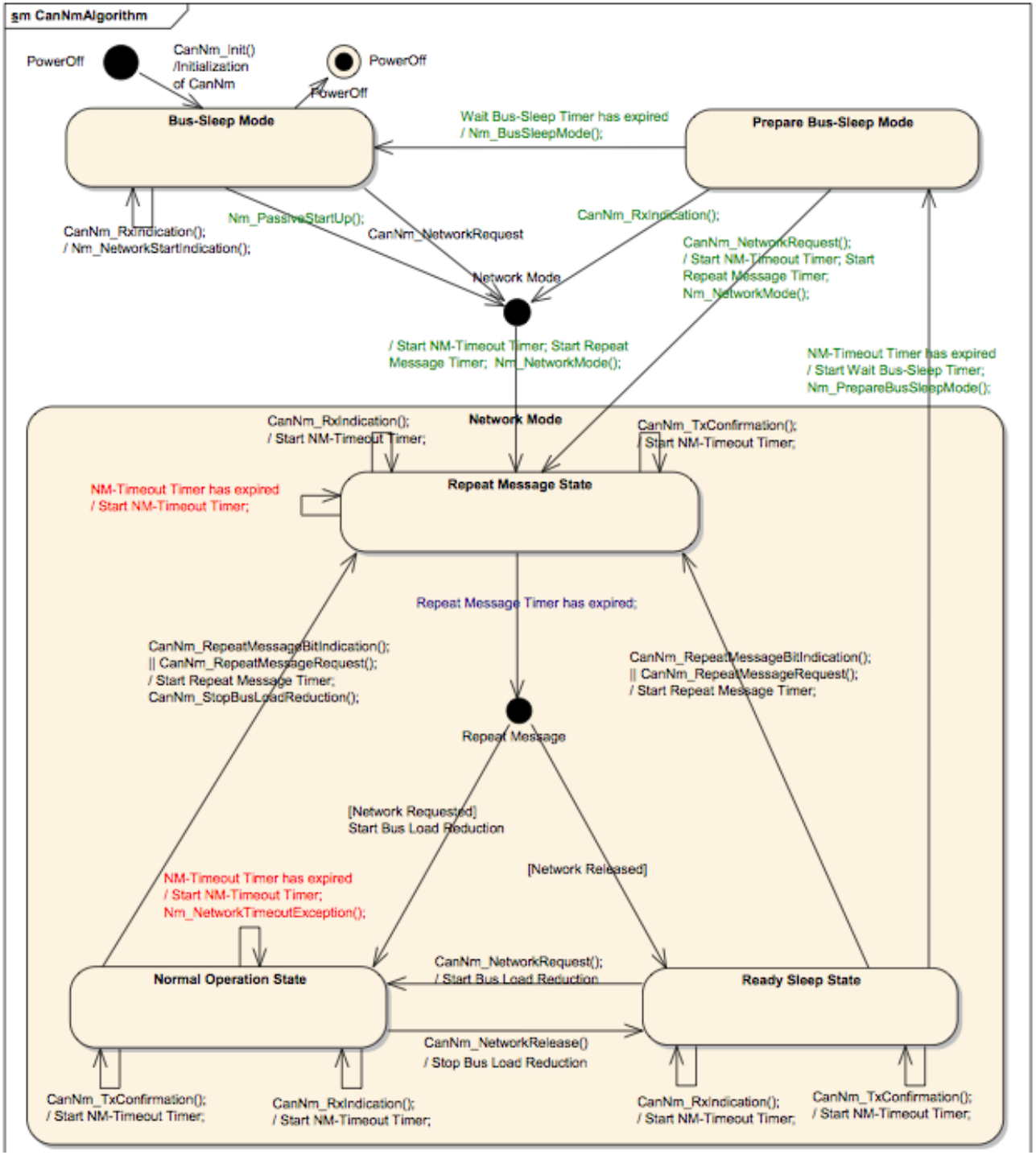
A **Test** is a sequence of events that are provoked by the test

We generate such tests automatically

AUTOSAR component
as UML state machine

CAN
Network Management

Now... make a
QuickCheck model from
this state machine



State transitions as Erlang data structure

```
bus_sleep_mode(_) ->
    [ {power_off,{call,?MODULE,powerOff,[]}},
      {bus_sleep_mode,{call,?MODULE,main,[]}},
      {bus_sleep_mode,{call,?MODULE,'CanNm_RxIndication',[id(),u8()]}},
      {repeat_message_state,{call,?MODULE,'Nm_PassiveStartUp',[]}},
      {repeat_message_state,{call,?MODULE,'CanNm_NetworkRequest',[]}}].

repeat_message_state(_) ->
    [ {normal_operation_state,{call,?MODULE,main,[]}},
      {ready_sleep_state,{call,?MODULE,main,[]}},
      {repeat_message_state,{call,?MODULE,main,[]}},
      {repeat_message_state,{call,?MODULE,'CanNm_RxIndication',[id(),u8()]}},
      {repeat_message_state,{call,?MODULE,'CanNm_TxConfirmation',[id()]}}].
```

QuickCheck model

Model how additional state data changes: timers, network status, ...

```
next_state_data(repeat_message_state,repeat_message_state,S,_V,{_,_,main,_}) ->
    S#can_nm{repeatMessageTimer = S#can_nm.repeatMessageTimer-1,
             nmTimeoutTimer =
               case S#can_nm.nmTimeoutTimer of
                   0 -> ?NMTIMEOUT;
                   N -> N-1
               end};

next_state_data(repeat_message_state,repeat_message_state,S,_V,{_,_,_,_}) ->
   S#can_nm{repeatMessageTimer = S#can_nm.repeatMessageTimer-1,
            nmTimeoutTimer = ?NMTIMEOUT};
```

Property

For any sequence of permissible events:

after each event:

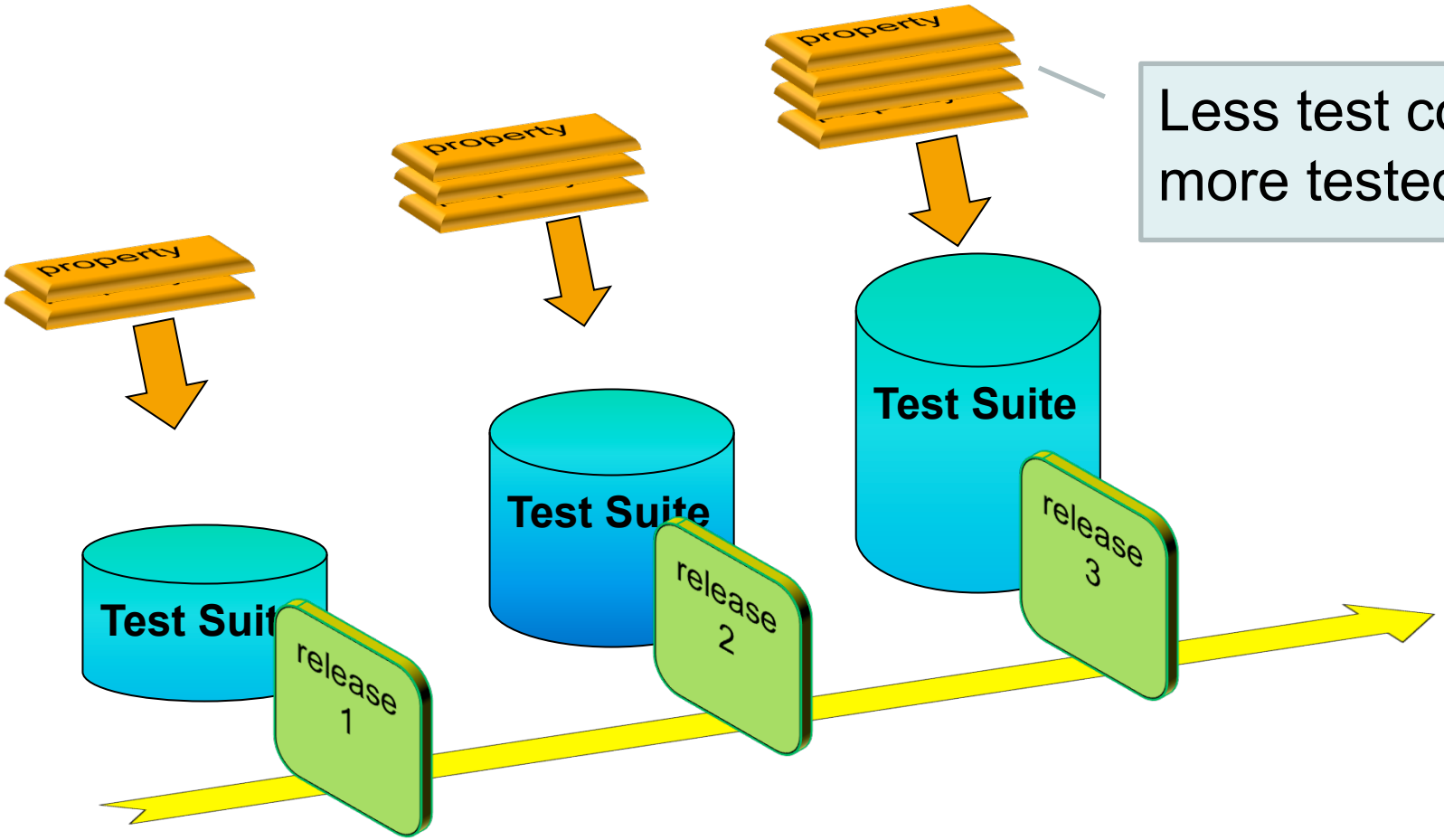
the real system should be in the same state as the model system

What Makes Quickcheck Great?

- Traditional Testing
  - Each sequence is
    - Hard-coded
    - Picked manually
- Test cases tend to be large to cover tricky cases

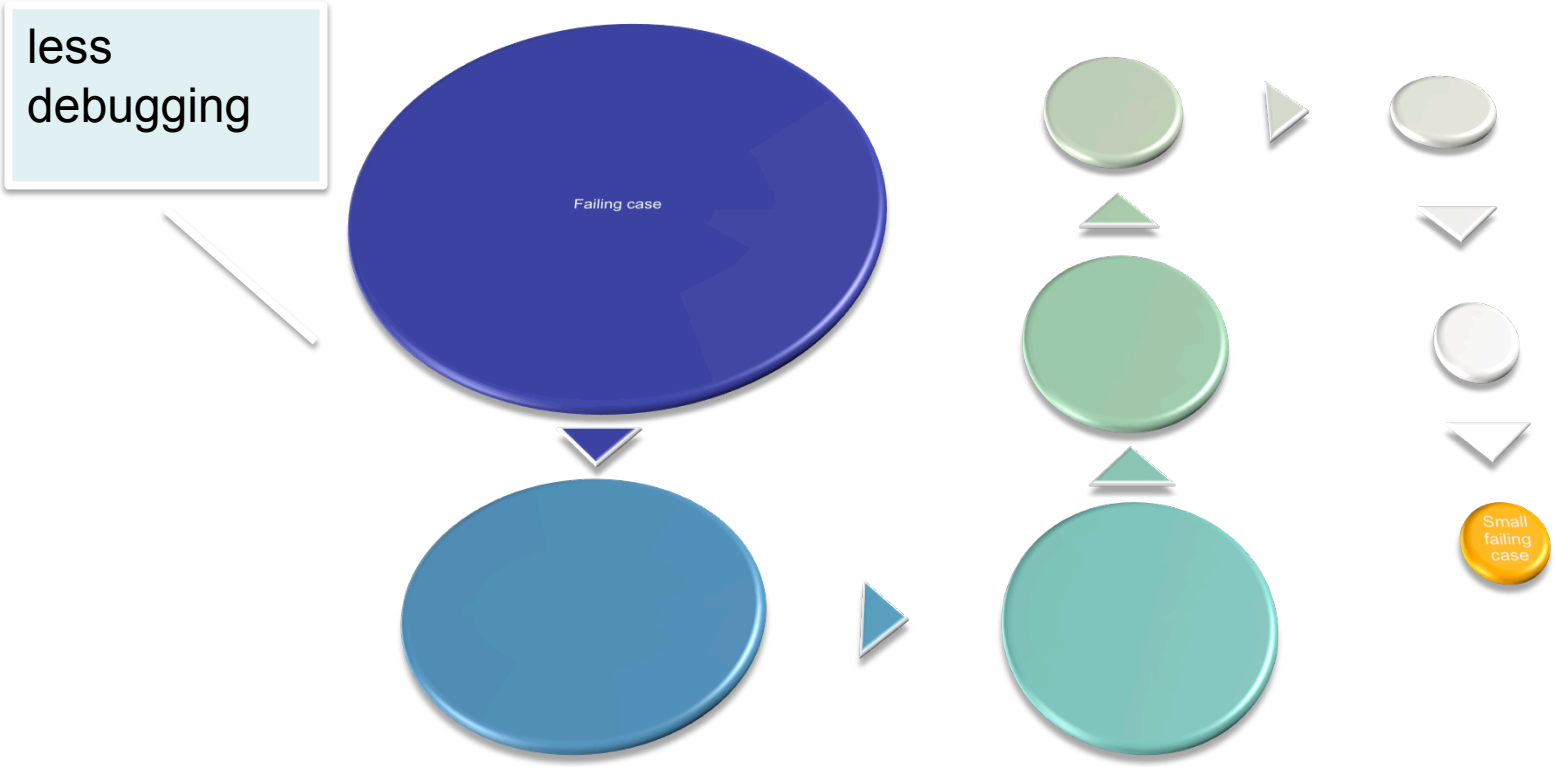
- Quickcheck testing
  - Each testis
    - Randomly chosen
    - Generic
- Test cases shrink to small failing test cases

Write **Properties** instead of Test Cases

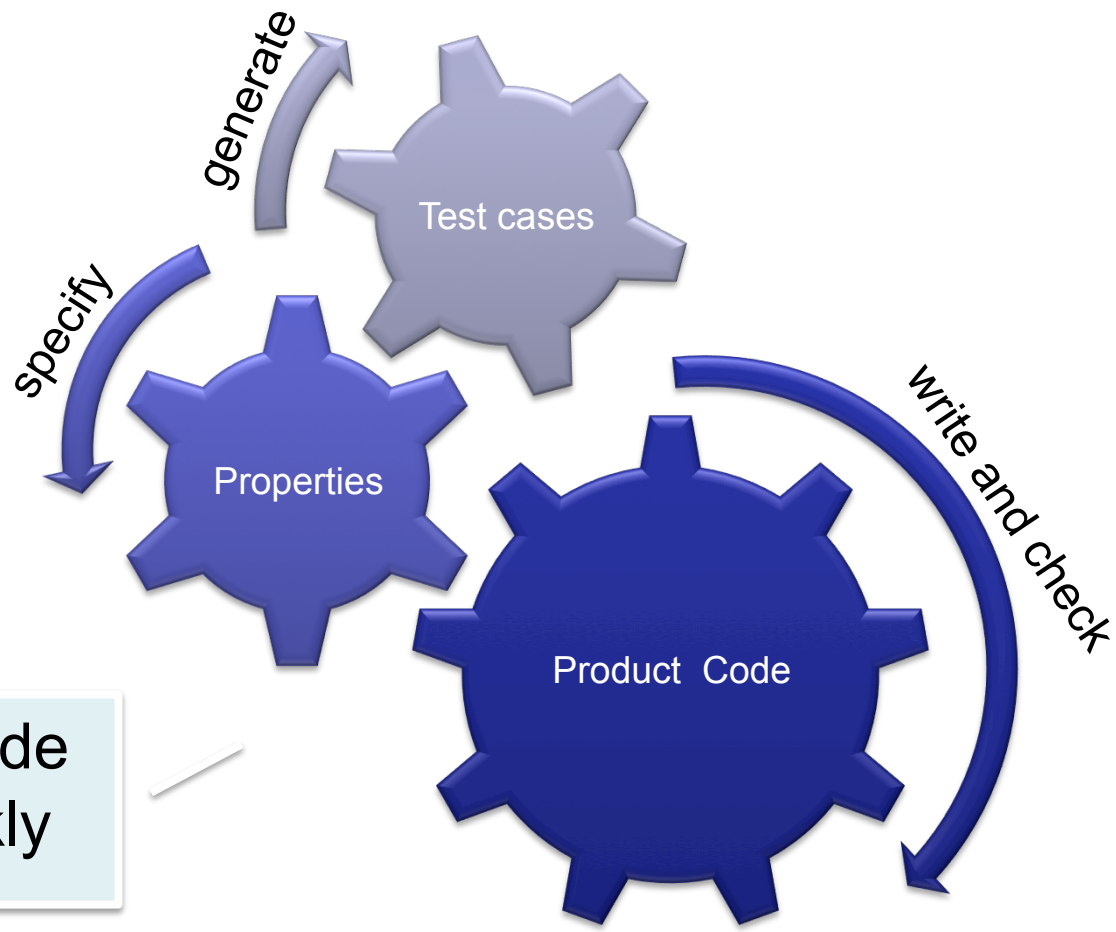


Less test code, more tested

Find smaller failing tests automatically



Testing and Property specification part of Coding



Get tricky code correct quickly

Five times more productive

QuickCheck shows:

With model based testing one can gain at least five times more productivity in testing

Model based testing makes software development more effective (feedback loop)

Models are easier to communicate than test cases

Creating models is a creative exercise and a lot of fun.