Tutorial

QuickCheck for EUnit

Thomas Arts, Simon Thompson

Chalmers, Quviq AB, University of Kent

Property-based testing

Develop higher-level properties of systems, rather than unit tests.

Check with randomly-generated test data.

```
prop_reverse() ->
  ?FORALL({Xs,Ys},
          {list(),list()},
          reverse(Xs++Ys)==
            reverse(Xs)++reverse(Ys)).
```

Testing state-based systems

Specify by finite state machine that describes the allowable sequences of API calls.

Correctness property is that the system runs ok for all legal command sequences.

```
command(S) -> oneof(
 [{call,?MODULE,spawn,…},
  {call,?MODULE,register,…}
  {call,?MODULE,unregister,…},
  {call,erlang,whereis,…}]).
```

Testing

Testing can be used
to reveal the
presence of bugs,
but never used to
prove their absence.

Edsger W. Dijkstra

Testing

Tests as artifacts.

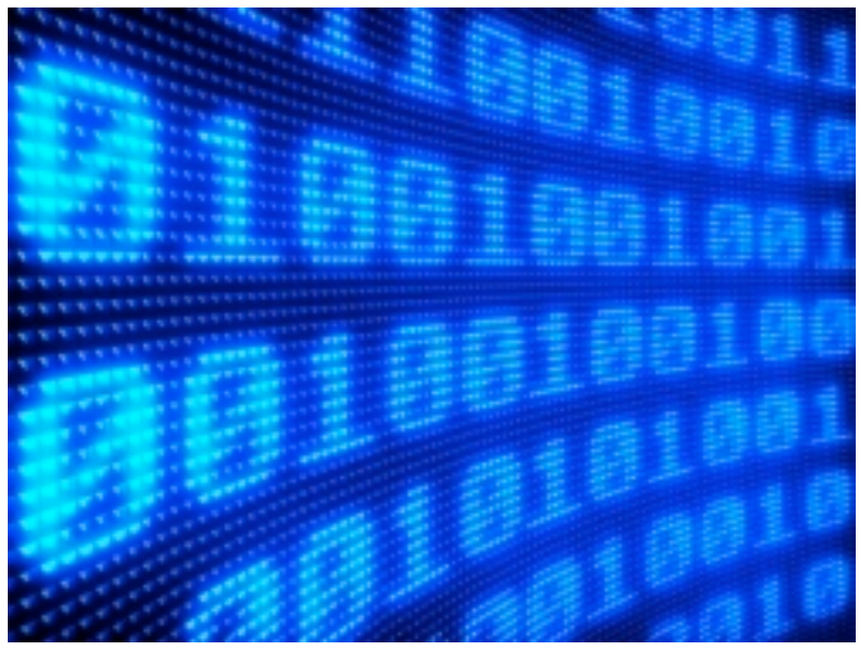
Specifications.

Quality of the tests?

Enough tests?

Role of negative tests.

Tests and models.



Property-based testing

How to write properties?

How to build models?

Use the artifacts that are already written …

Negative tests

I shall not be the last one to admit that negative testing is often a back seat driver in TDD.

Gianfranco Alongi

Used in bluefringe algorithm.

Our approach helps you to develop these tests …

Contribution

Engage with test suite in a different way, either with or without an implementation.

Derive a QuickCheck FSM from an EUnit test suite.

Derive more EUnit tests from that FSM.

Tests to FSM and back

Test

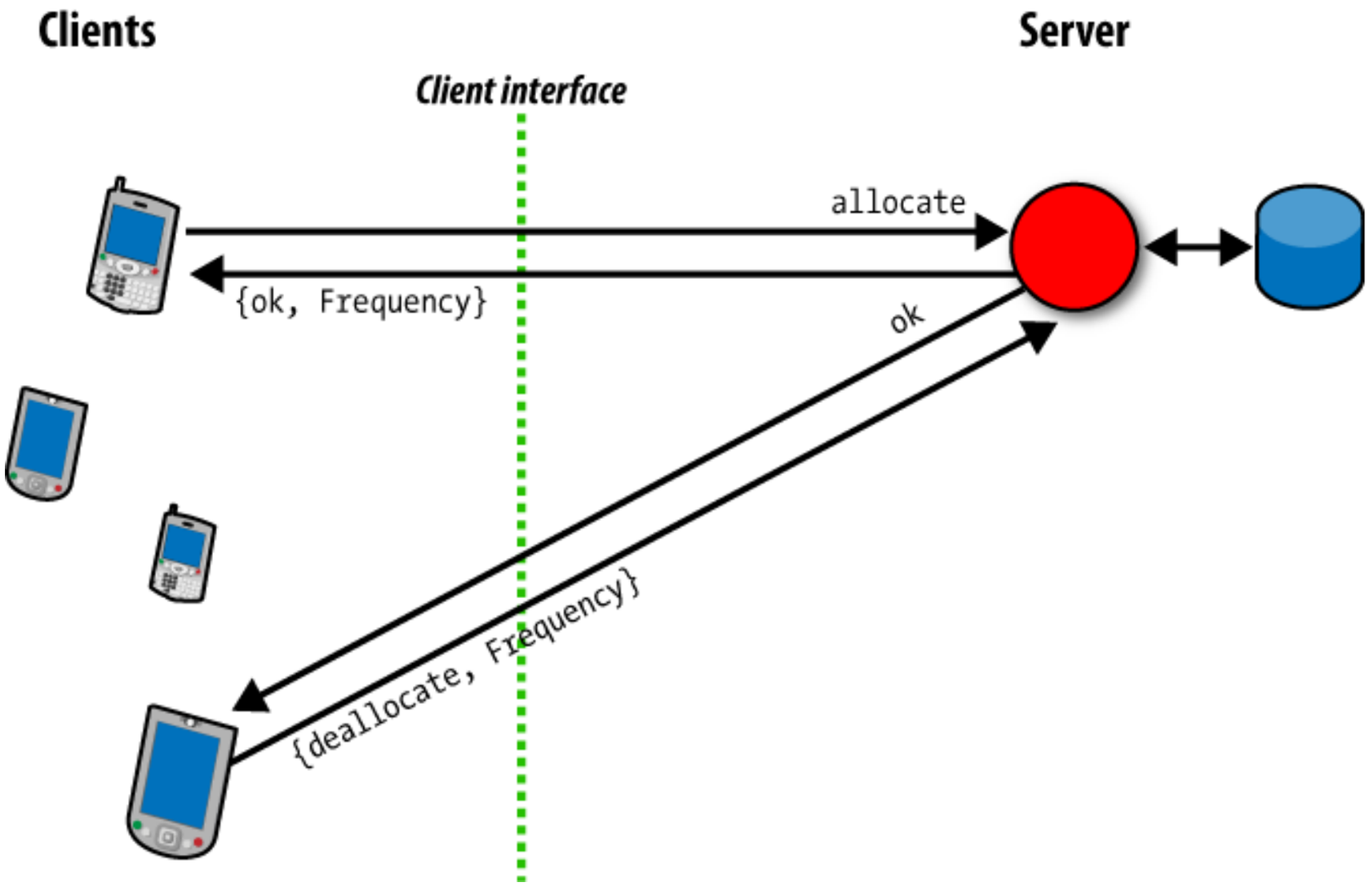
FSM

Implementation

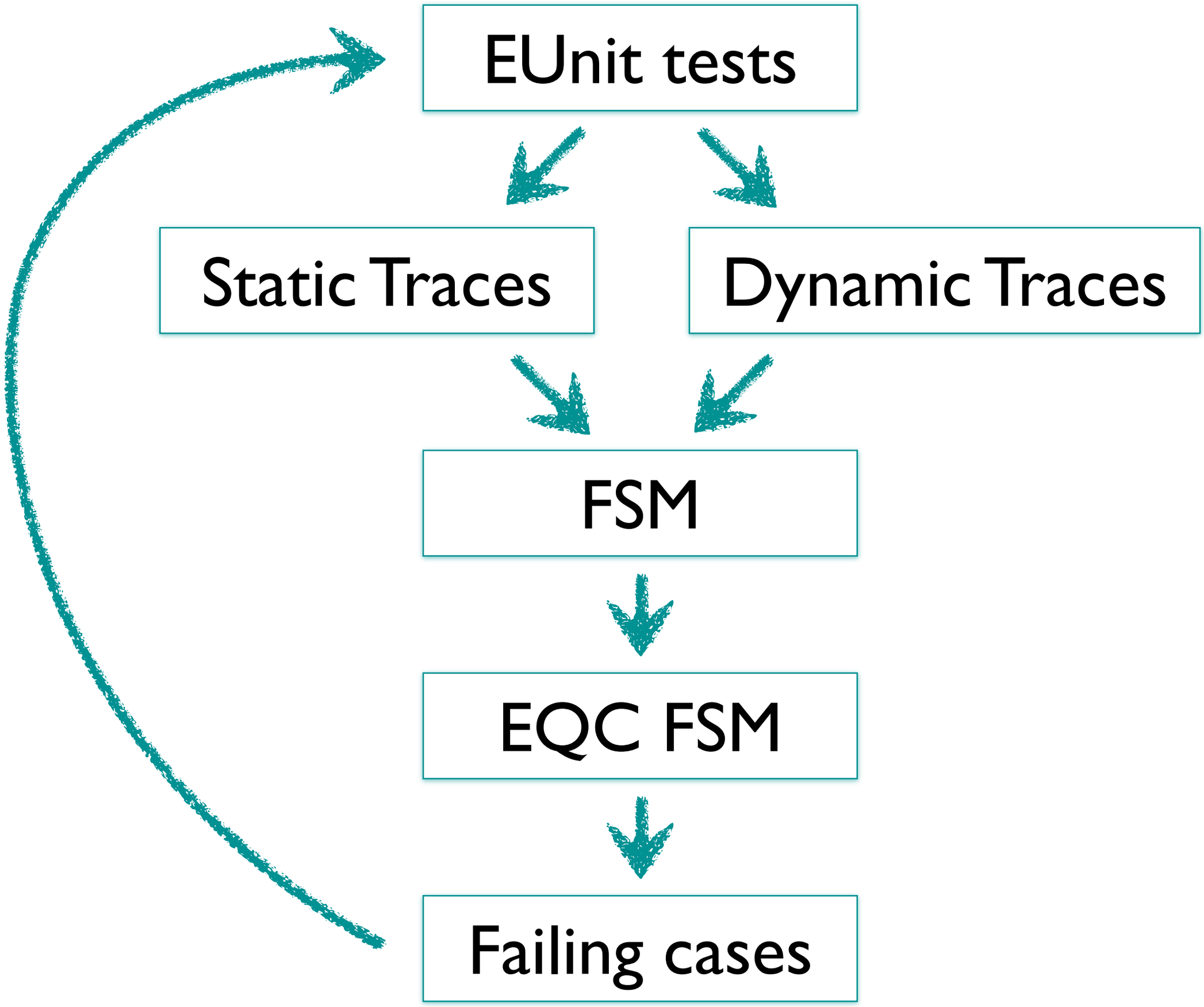
EUnit

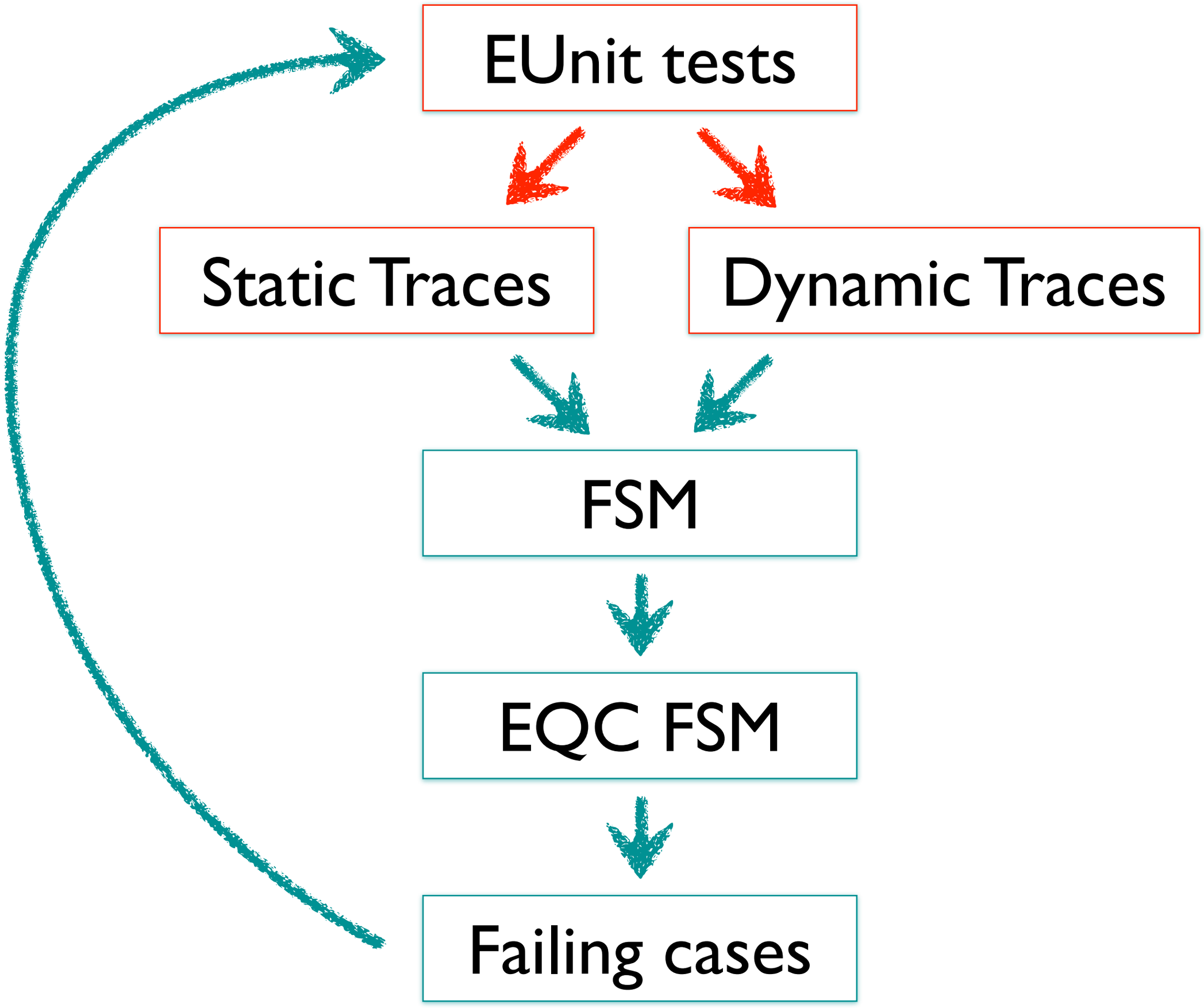
QuickCheck

Example


Clients
Server
Client interface
allocate
{ok, Frequency}
ok
{deallocate, Frequency}

Demo


EUnit tests
Static Traces
Dynamic Traces
FSM
EQC FSM
Failing cases


EUnit tests
Static Traces
Dynamic Traces
FSM
EQC FSM
Failing cases

Positive & negative tests

```
startstop_test() ->
    ?assertMatch(true,start([])),
    ?assertMatch(ok,stop()),
    ?assertMatch(true,start([1])),
    ?assertMatch(ok,stop()).
```

Positive

```
stopFirst_test() ->
    ?assertError(badarg,stop()).
```

Negative

Abstraction

```
?assertMatch(true,start([])).
```

```
?assertMatch(true,start([1])).
```

Can view `start` as an operation …

… or `start([])` and `start([1])` as different operations

Need to control the abstraction during extraction: re-implement QSM in Erlang (*vs.* using StateChum).

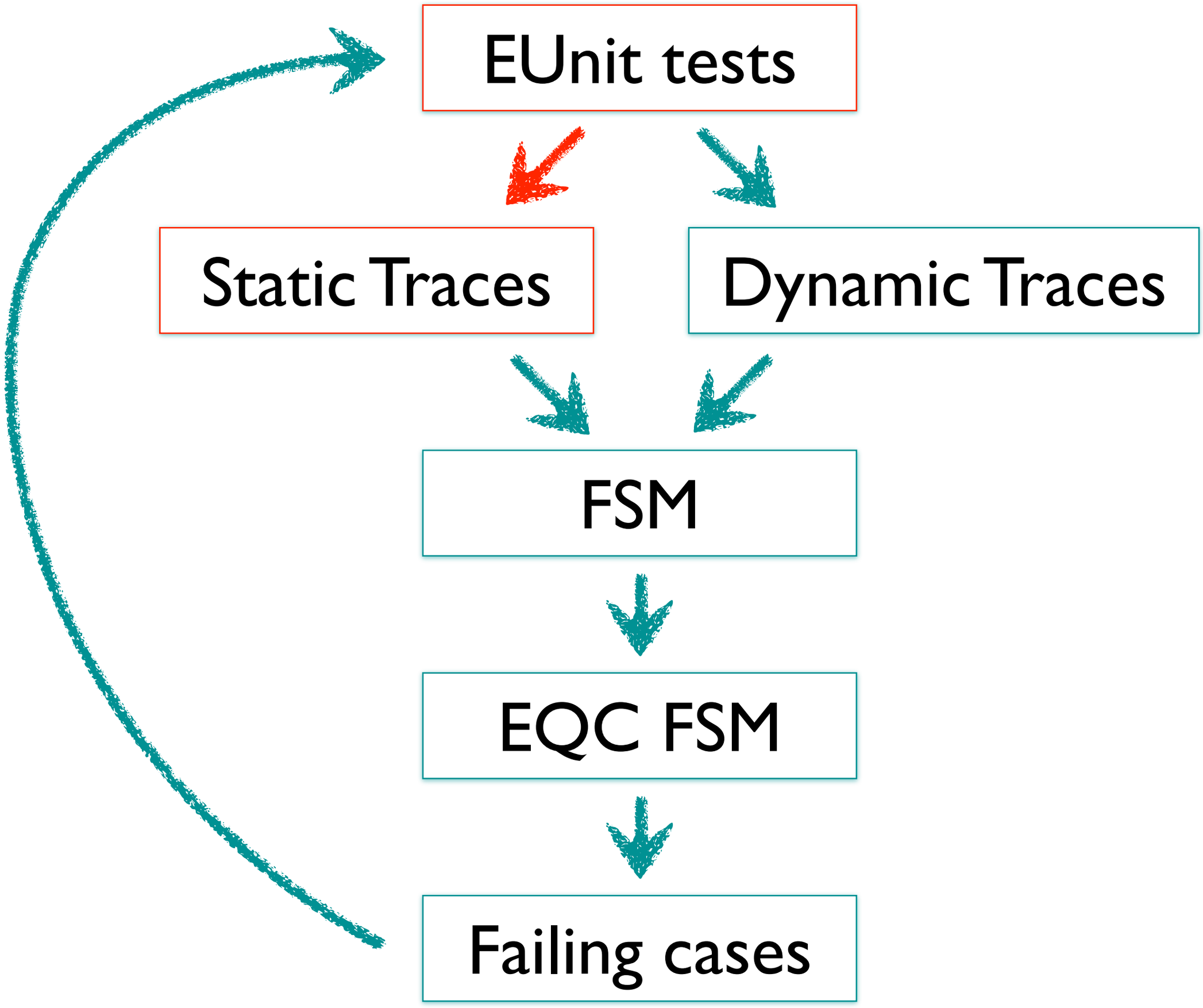
Static *vs.* Dynamic

Static

- No need for an implementation of SUT
- No return values, no dependencies
- Requires stylised tests

Dynamic

- Can record results and dependencies ...
- No need for style conformance.


EUnit tests
Static Traces
Dynamic Traces
FSM
EQC FSM
Failing cases

Gathering static traces

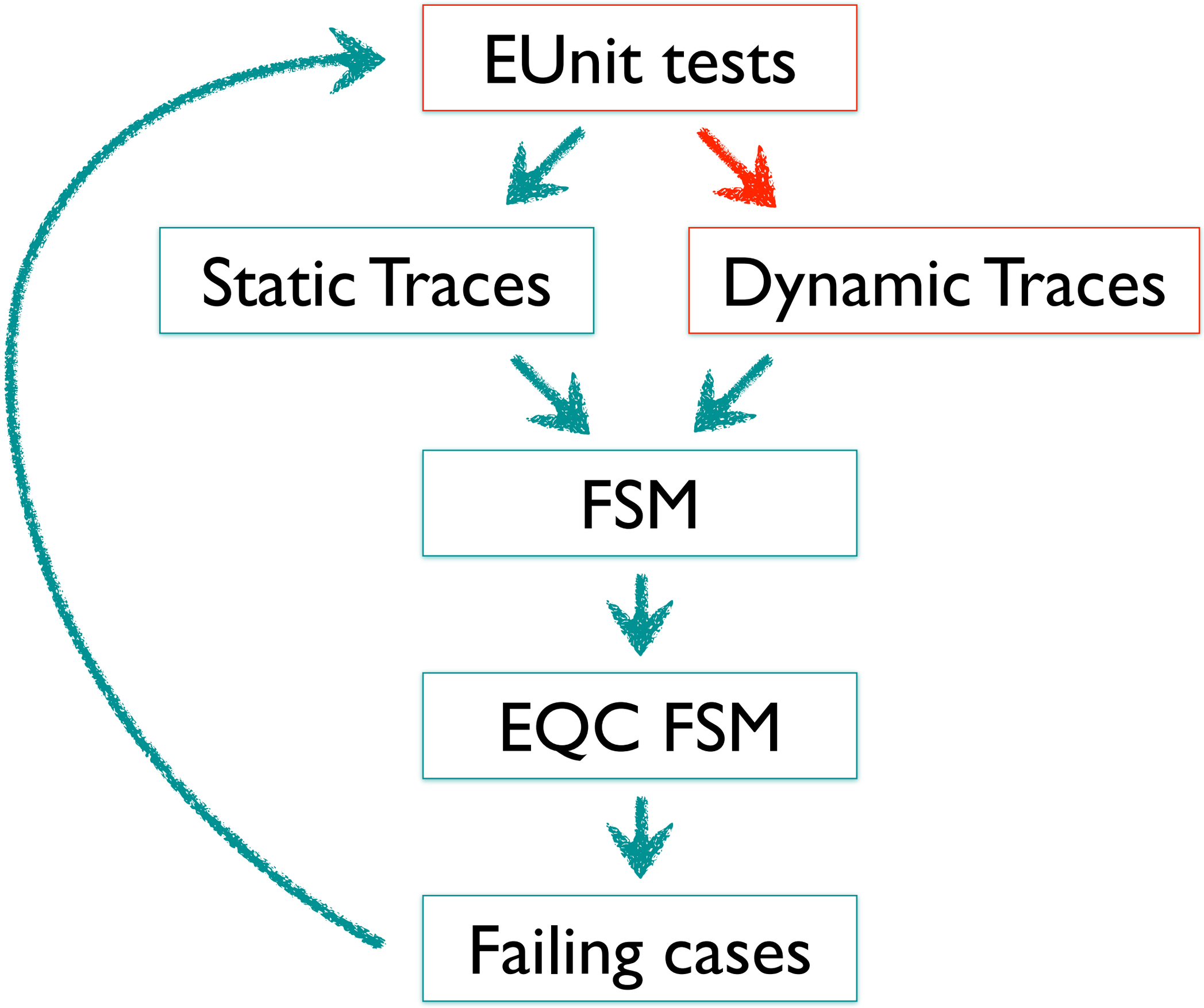
Redefine EUnit macros: `EUNIT_HRL` macro.

Positive *vs.* negative tests.

Parse and recognise

Record function calls in the API …

… and EUnit generators e.g. `foreach`.


EUnit tests
Static Traces
Dynamic Traces
FSM
EQC FSM
Failing cases

Tracing EUnit

```
[{eunit_tracing,open,[inorder]},
 {eunit_tracing,open,[test]},
 {frequency,start,[[]]},
 {eunit_tracing,close,[test]},
 {eunit_tracing,open,[test]},
 {frequency,stop,[]},
 {eunit_tracing,close,[test]},
 {eunit_tracing,open,[test]},
 {frequency,start,[[1]]},
 {eunit_tracing,close,[test]},
 {eunit_tracing,open,[test]},
 {frequency,stop,[]},
 {eunit_tracing,close,[test]},
 {eunit_tracing,close,[inorder]},
 .....
```

Trace BIFs to record API function calls …

… and ‘marker’ functions to delimit the scope of tests and groups.

Parse into individual positive or negative traces.

What (not) to trace?

Trace *API* functions: a subset of those in the `export` statement.

Trace *top-level* API calls: don't trace nested calls to API functions.

Trace calls in all processes, or in the 'home' testing process?

Run EUnit as a black box

Transform tests before running EUnit.

Replace `assertError` by `assertErrorTrace`.

Define `assertErrorTrace` to be like original but augmented with info that test is -ve.

Rewrite tests using `syntax_tools` to contain appropriate wrapping functions.

Example

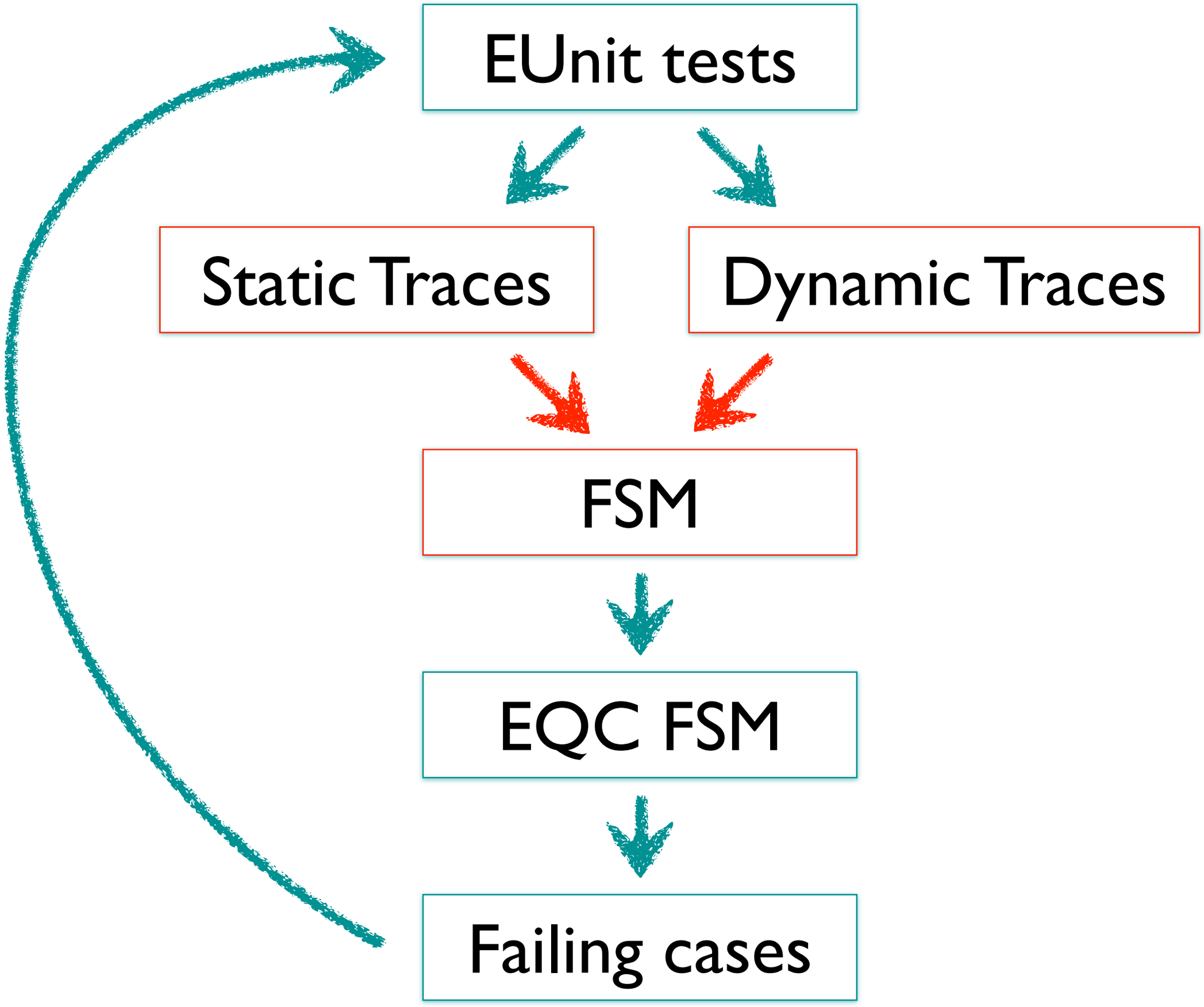
```
startstop_test_() ->
    {inorder,
     [ ?_assertMatch(true,start([])),
       ?_assertMatch(ok,stop()),
       ?_assertMatch(true,start([1])),
       ?_assertMatch(ok,stop())]}.
```

```
startstop_test_() ->
    eunit_tracing:test__wrap(begin
                    {inorder,
                        [?'_assertMatchTrace'(true, (start([]))),
                         ?'_assertMatchTrace'(ok, (stop())),
                         ?'_assertMatchTrace'(true, (start([1]))),
                         ?'_assertMatchTrace'(ok, (stop()))]}
                      end,
                   16).
```

```
test__wrap(F,LineNumber)
  when is_function(F) ->
    case element(2,erlang:fun_info(F,arity)) of
    0 ->
        fun () ->
            test_start(LineNumber),
            Result = F(),
            test_end(),
            Result
        end;
   1 -> ...
```

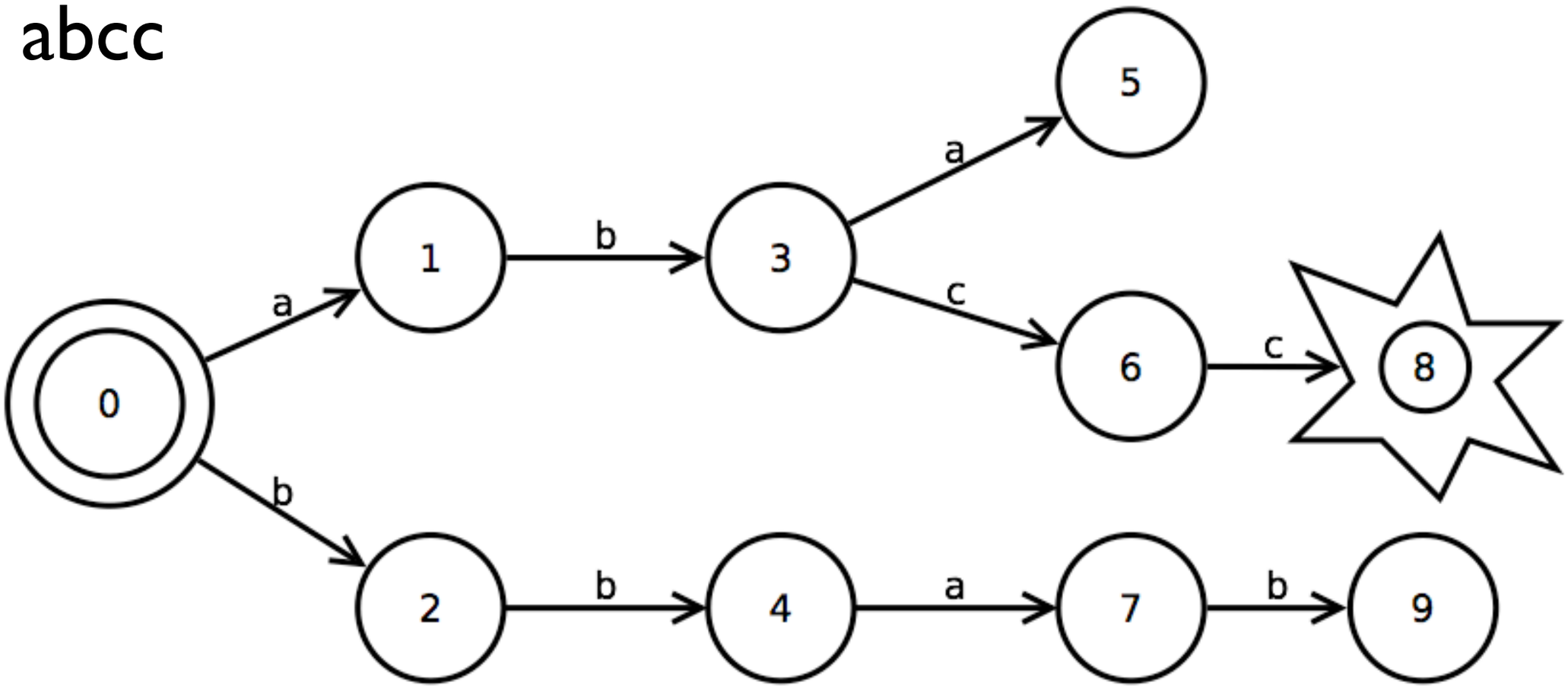
```
startstop_test_() ->
    eunit_tracing:test__wrap(begin
                    {inorder,
                        [?'_assertMatchTrace'(true, (start([]))),
                         ?'_assertMatchTrace'(ok, (stop())),
                         ?'_assertMatchTrace'(true, (start([1]))),
                         ?'_assertMatchTrace'(ok, (stop()))]}
                      end,
                    16).
```

```
-define('_assertMatchTrace'(X, Y),
    eunit_tracing:positive_wrap(??X,?'_assertMatch'(X, Y))).
```

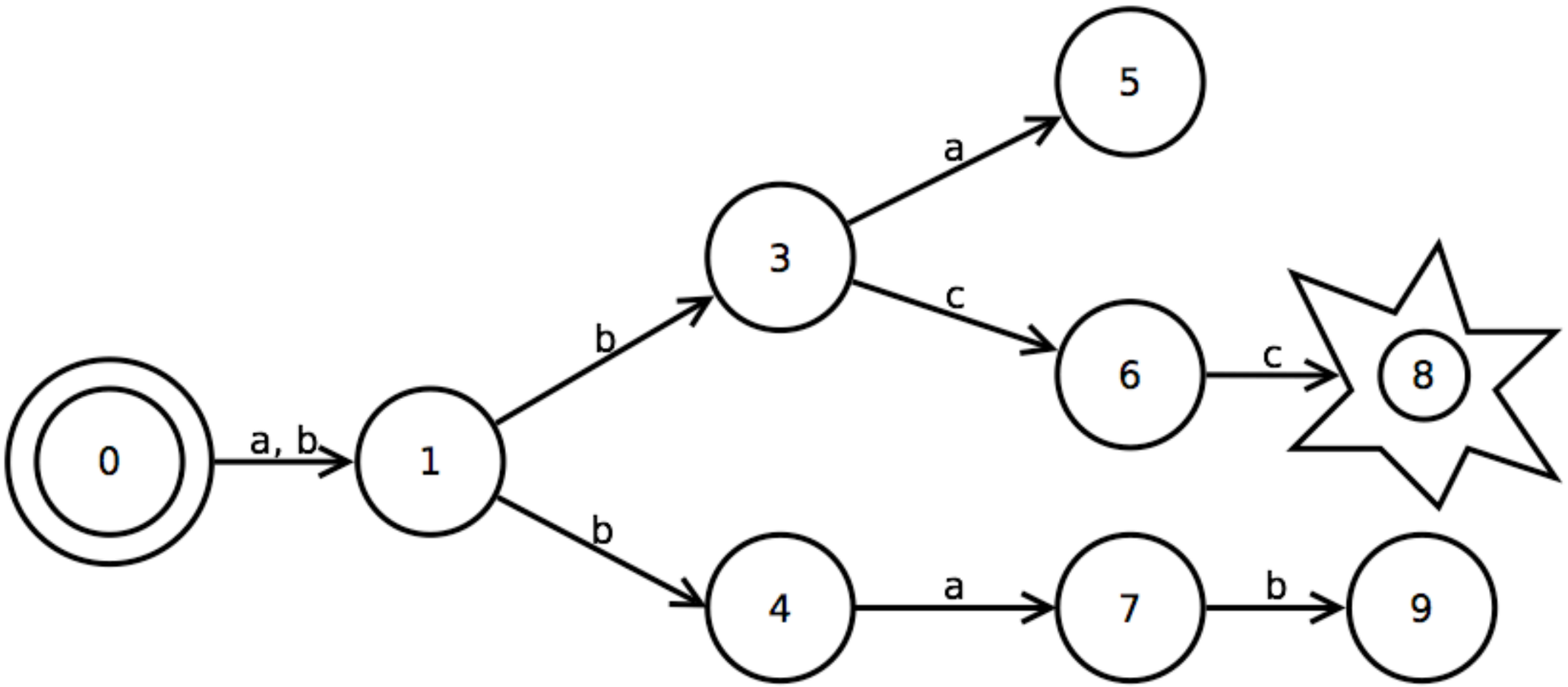

EUnit tests
Static Traces
Dynamic Traces
FSM
EQC FSM
Failing cases

APTA

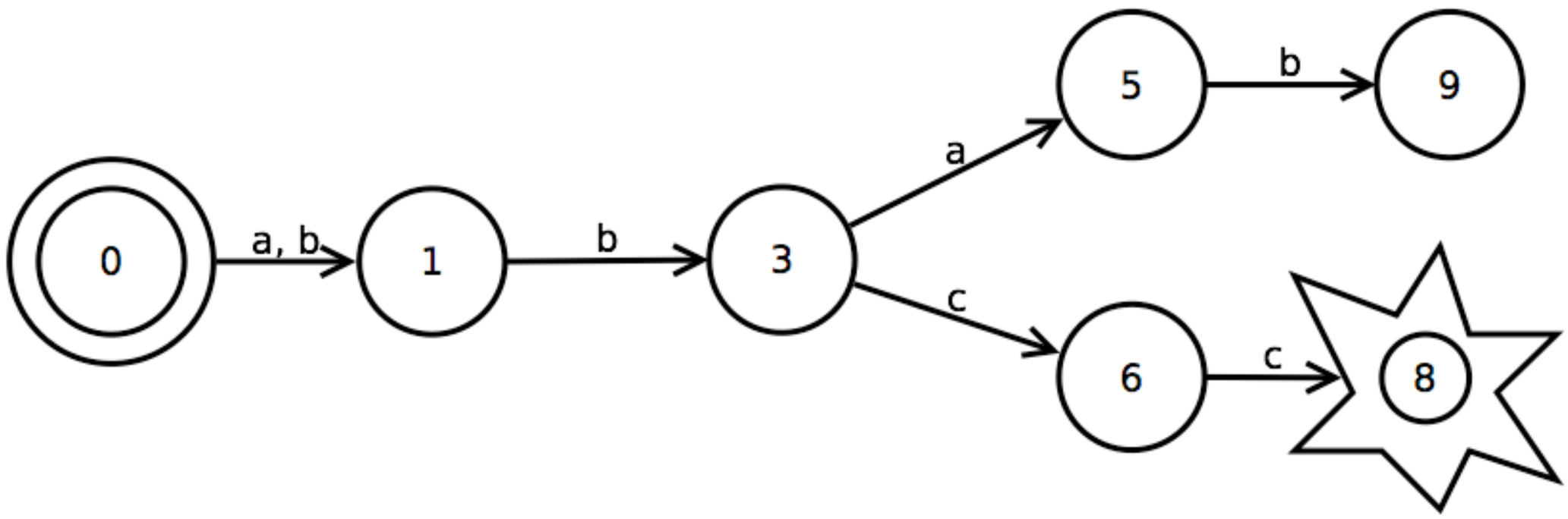
+ aba
  bbab
- abcc



Try merging 1 and 2 …



So merge 3 & 4, 5 & 7 …



Bluefringe algorithm

Merge states

Score # merges to re-make deterministic.

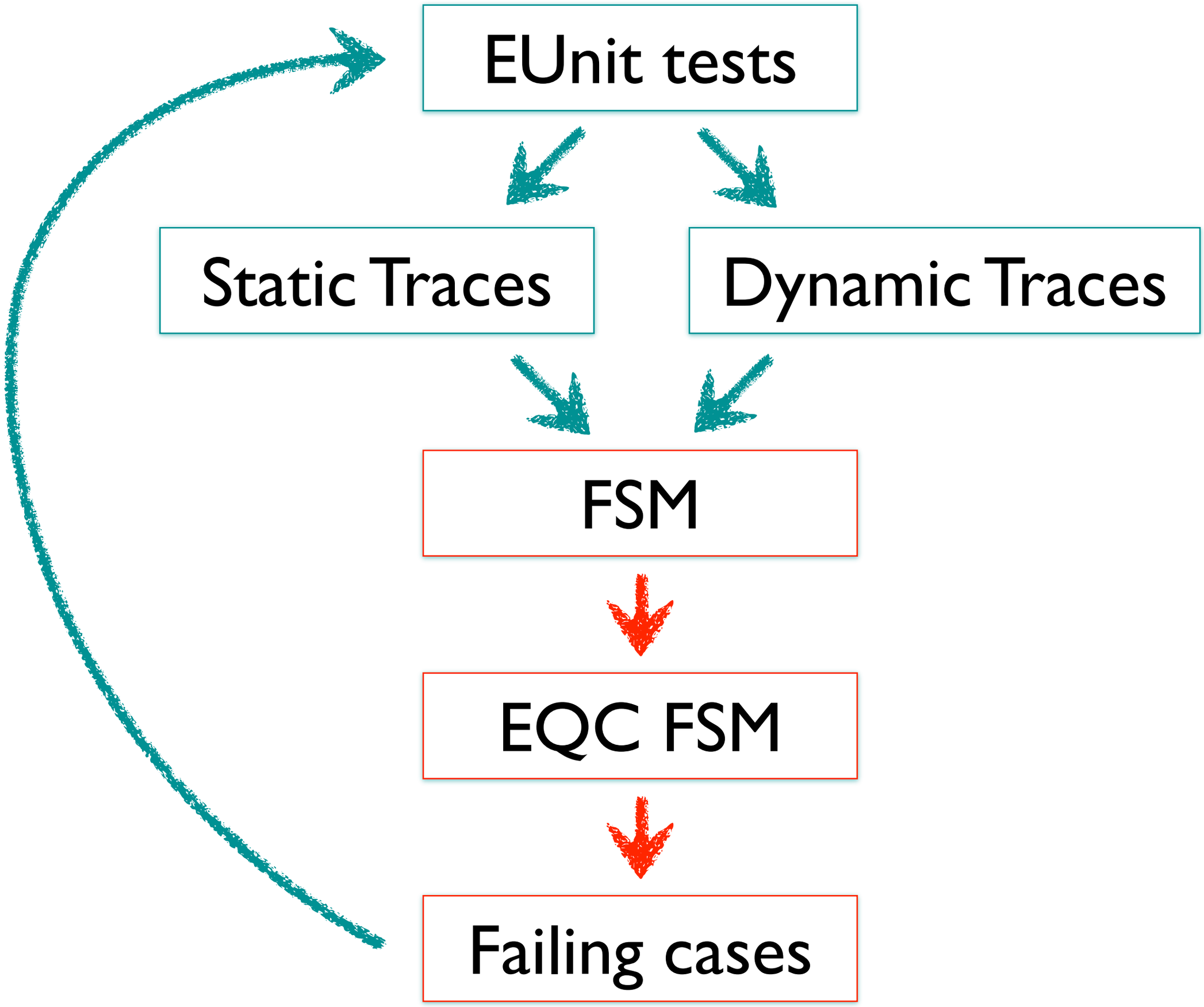
Cannot merge accepting and fail states.

Must(n't) accept original (-ve) +ve sequences.

Bluefringe

Aim to merge from the root outwards.

Maintain red/blue sets: halt when all red.

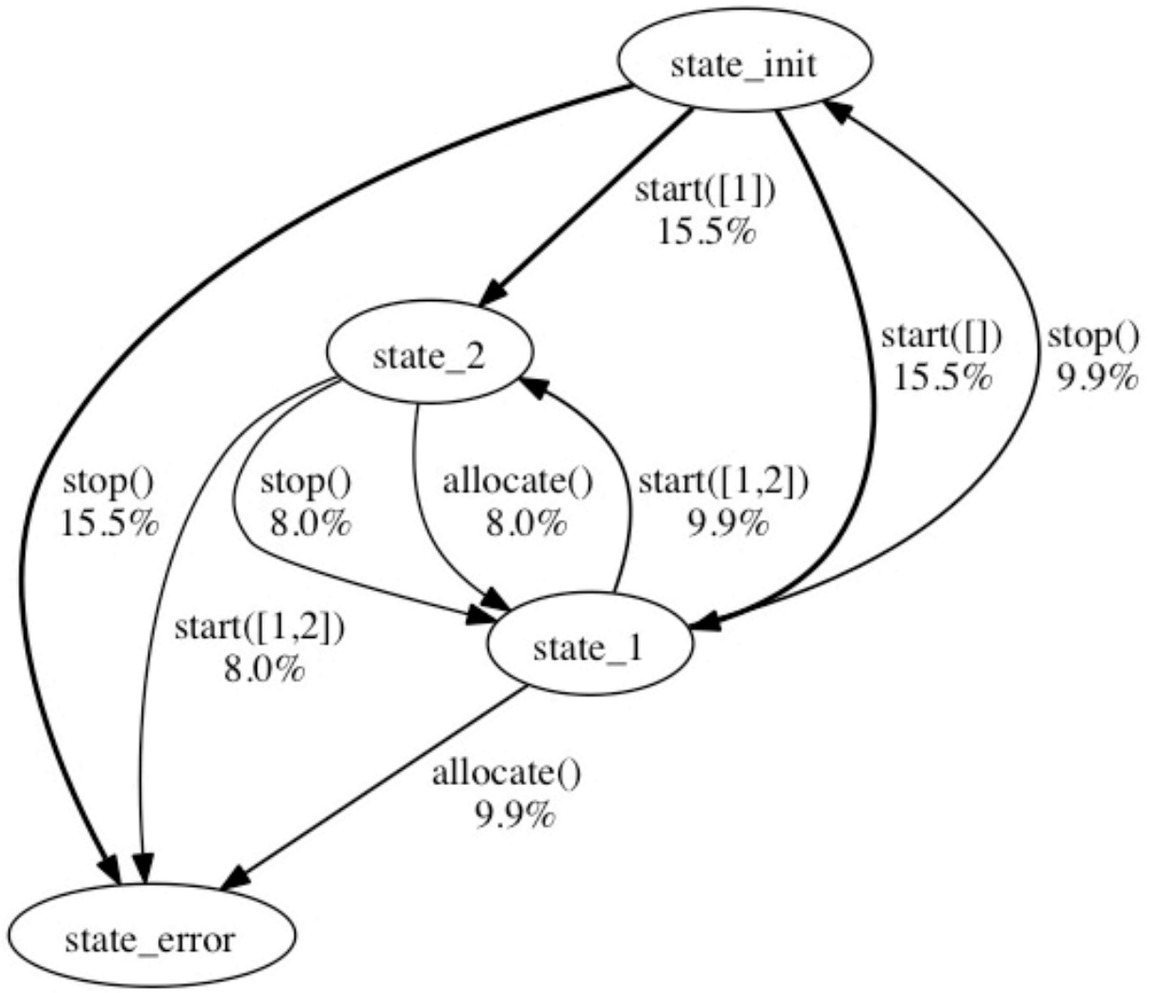

EUnit tests
Static Traces
Dynamic Traces
FSM
EQC FSM
Failing cases

QuickCheck FSM

Standard FSM

Generate and …

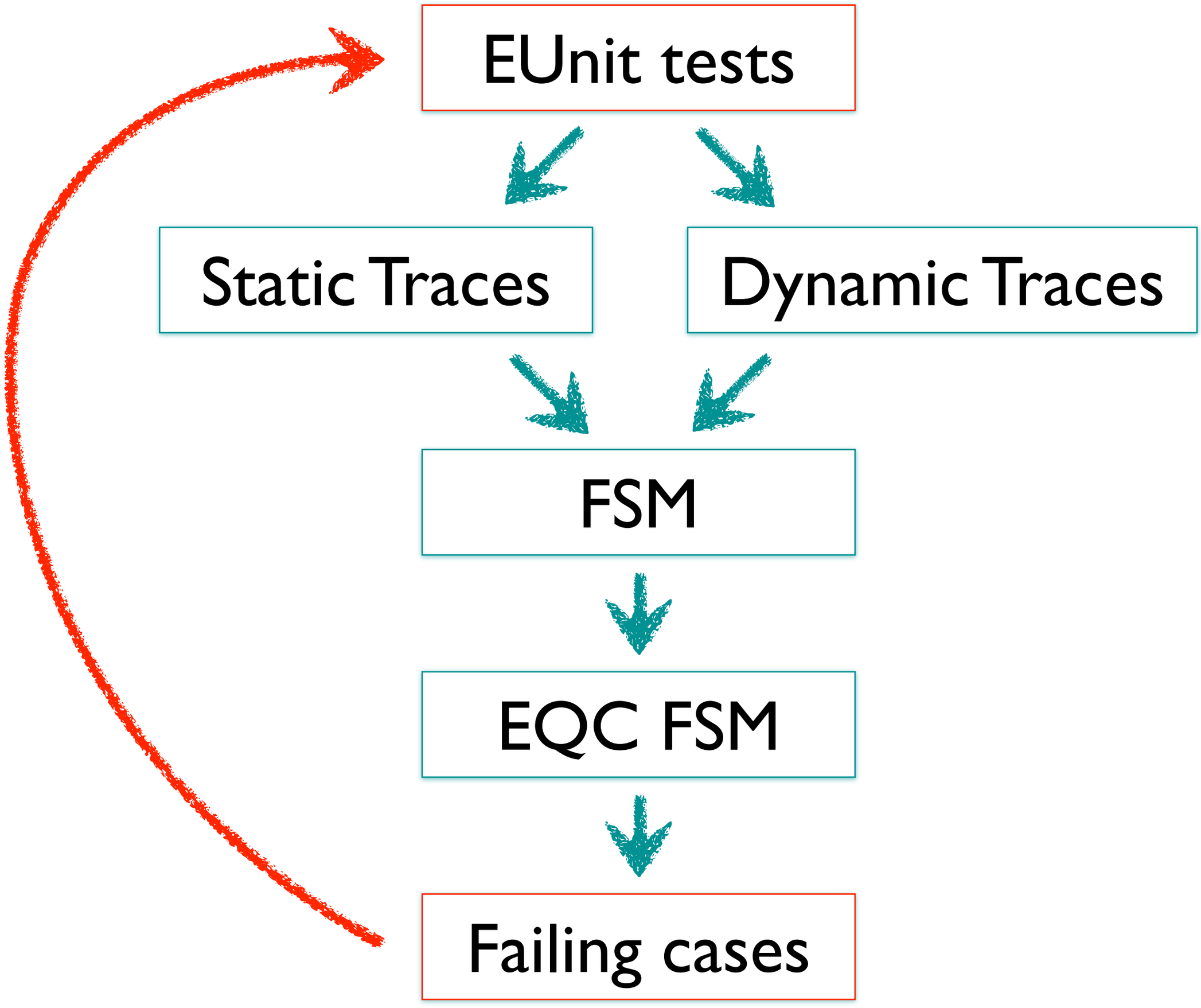
… shrink failing cases



QuickCheck FSM

```
-module(frequency_eqc).

-include_lib("eqc/include/eqc.hrl").
-include_lib("eqc/include/eqc_fsm.hrl").

-compile(export_all).

initial_state() -> state_init.

state_init(_) ->
  [{state_error, {call,?MODULE,stop,[]}},
   {state_1,
     {call,?MODULE,start,[oneof([[],
[1]])]}}].

state_1(_) ->
  [{state_init, {call,?MODULE,stop,[]}},
   {state_error, {call,?MODULE,start,
[[]]}}].

state_error(_) -> [].
```

```
postcondition(_, state_error, _S, _Call, R) ->
    case R of
      {'EXIT', _} -> true;
      _ -> false
    end;
postcondition(_, _, _S, {call,_,start,[_]}, _R)
->
    true;
postcondition(_, _, _S, {call,_,stop,[]}, _R) ->
    true.

prop_frequency() ->
  ?FORALL(Cmds, (commands(?MODULE)),
    begin
      {_History, _S, Res} =
        run_commands(?MODULE,Cmds),
      Res == ok
    end).
```


EUnit tests
Static Traces
Dynamic Traces
FSM
EQC FSM
Failing cases

New tests

```
Shrinking.(1 times)
[{set,{var,1},{call,frequency_eqc,start,[[1]]}},
 {set,{var,2},{call,frequency_eqc,stop,[]}},
 {set,{var,3},{call,frequency_eqc,stop,[]}}]
```

```
new_test() ->
  ?assertMatch(true,frequency:start([1])),
  ?assertMatch(ok,frequency:stop()),
  ?assertError(badarg,frequency:stop()).
```

Cleanup: need system-specific information …

… add new `cleanup/0` callback function?

Future work

Other frameworks, e.g. Common Test , …

Other algorithms, e.g. grammar inference, …

Dealing with abstraction, state data, …

Questions?

`git@github.com:ThomasArts/Visualizing-EUnit-tests.git`

ProTest
property based testing