Erly Marsh - a Model-Based Testing tool

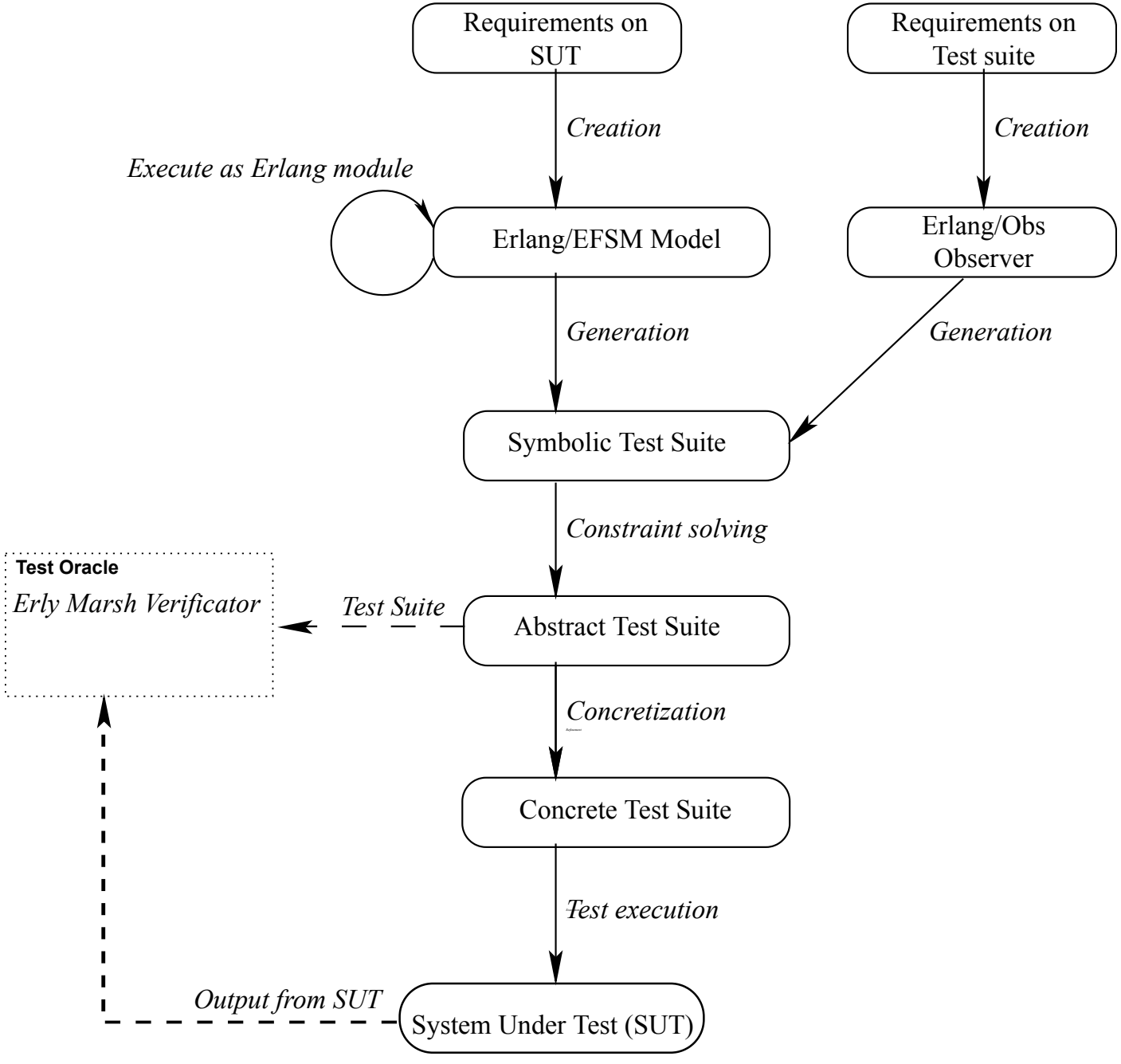
Johan Blom, PhD

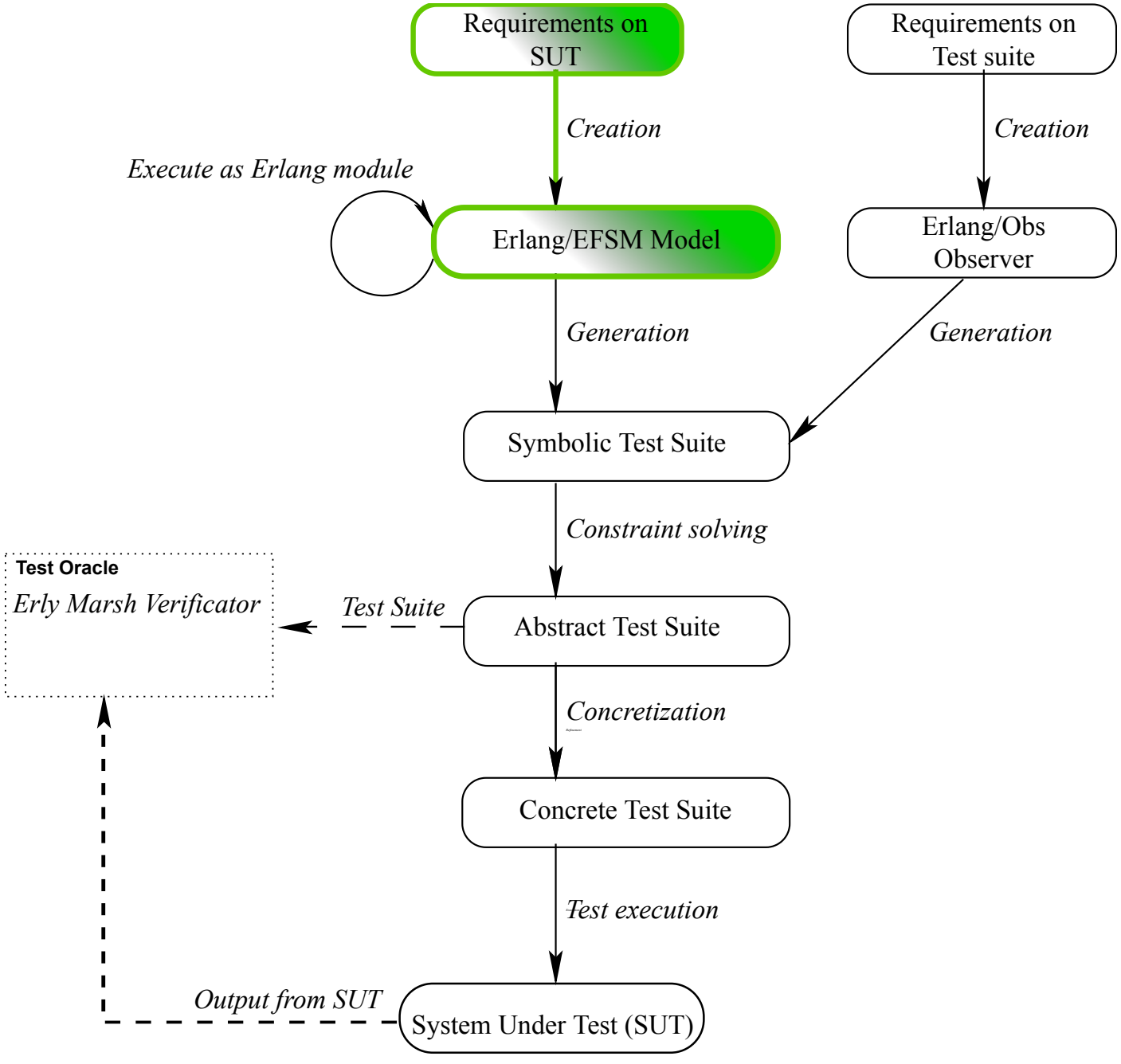
Motivation

- Mobile Arts
  - Develops server software for mobile telecom operators (Location server, SMSC etc.)
  - Implementations rather big and complicated
  - Experienced developers - but no testers
- How and what to test ?
  - Identified set of core modules in the Location server
  - Dialyzer runs revealed close to no problems
  - Manual test suite was running OK
  - So, everything was OK ?
- **NO!** (I thought)

Why ?

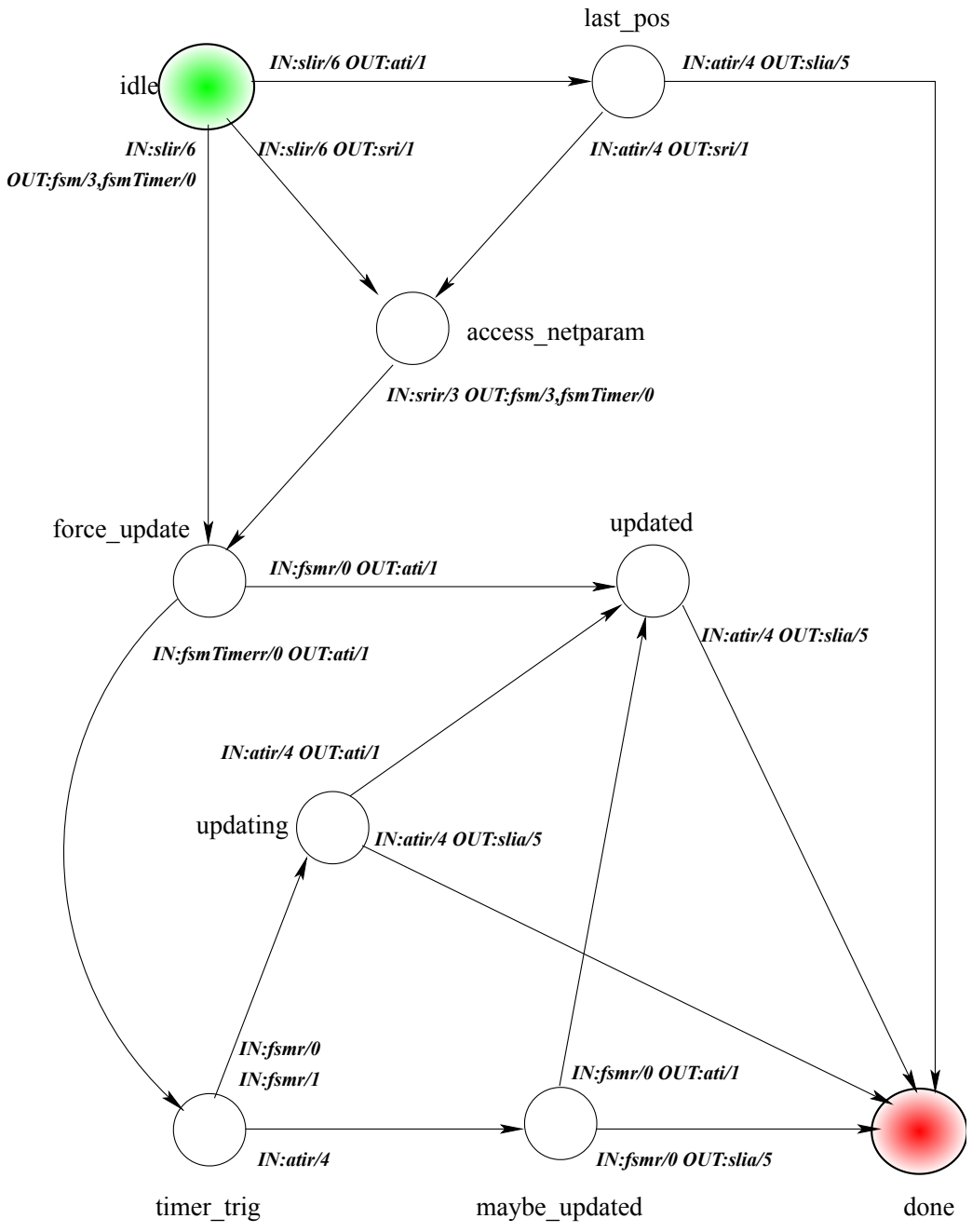
- Lots of the functionality not tested - in core modules and elsewhere
- Products **must** always be *operational* and limit *interference* with other network entities
- Model-Based Testing:
  - Captures *functional* requirements
  - Construct formal *executable* model
  - Test suite *generated* as execution of traces
  - Tool support enables generation of *very large* test suites
  - Implemented in several tools (Qtronic, SpecExplorer, ...)
- Started to develop my own tool - Erly Marsh

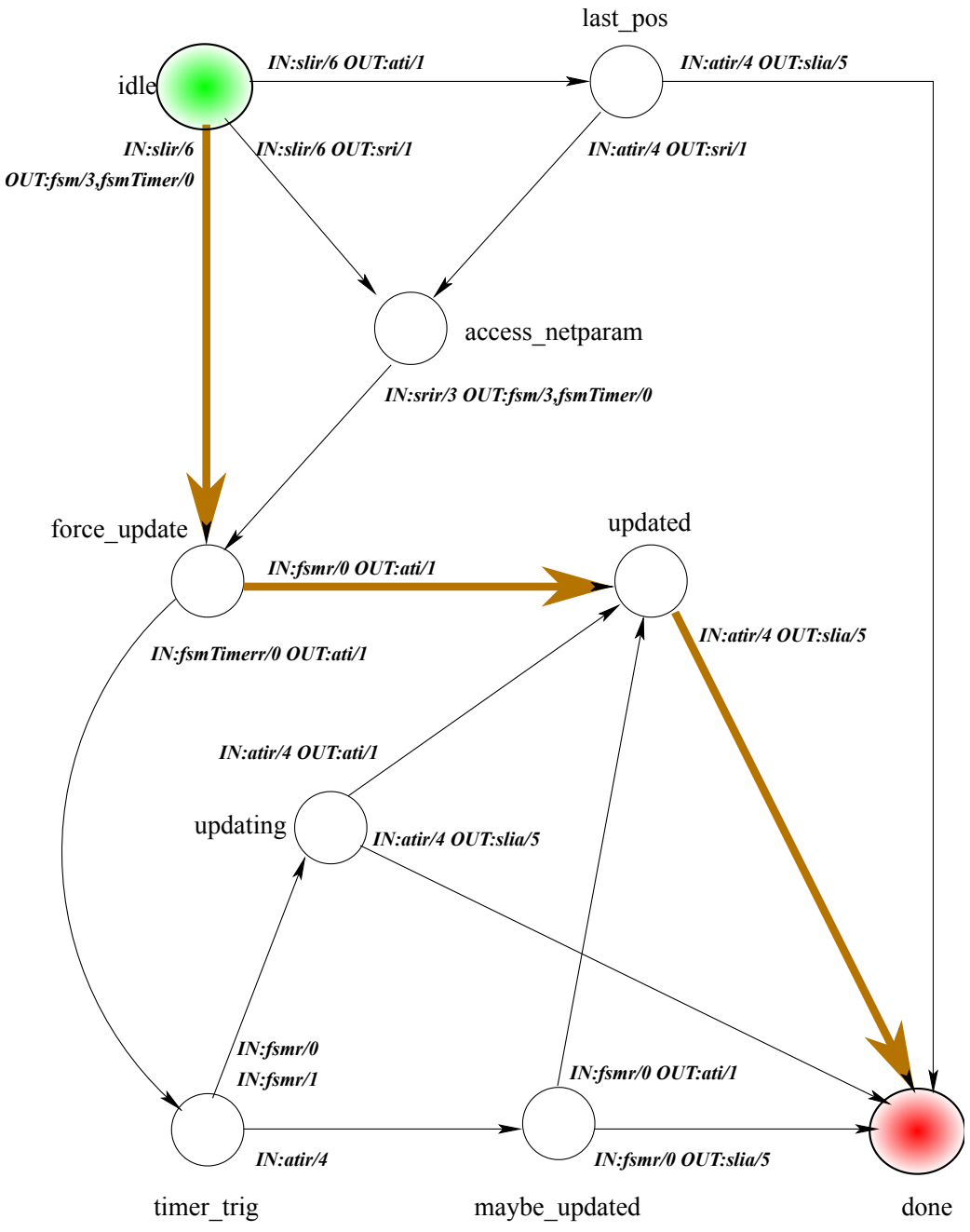

Requirements on SUT
Requirements on Test suite
Creation
Creation
Execute as Erlang module
Erlang/EFSM Model
Erlang/Obs Observer
Generation
Generation
Symbolic Test Suite
Constraint solving
Test Oracle
Erly Marsh Verificator
Test Suite
Abstract Test Suite
Concretization
Concrete Test Suite
Test execution
Output from SUT
System Under Test (SUT)


Requirements on SUT
Requirements on Test suite
Creation
Creation
Execute as Erlang module
Erlang/EFSM Model
Erlang/Obs Observer
Generation
Generation
Symbolic Test Suite
Constraint solving
Test Oracle
Erly Marsh Verificator
Test Suite
Abstract Test Suite
Concretization
Concrete Test Suite
Test execution
Output from SUT
System Under Test (SUT)

Modeling Language

- **Idea:**
  - Represent *abstract* view of System Under Test (SUT)
  - Cover *functional* properties
  - Specifically protocols, i.e., modeled by *state machines*
  - Based on Erlang:
    - Suitable constructs: pattern matching, dynamic typing, single assignment variables, ...
    - Lower learning threshold for Erlang developers
    - Allows reuse of Erlang tool infrastructure
- **Approach:**
  - Introduce Erlang/EFSM as a specification language


last_pos
idle
IN:slir/6 OUT:ati/1
IN:atir/4 OUT:slia/5
IN:slir/6
OUT:fsm/3,fsmTimer/0
IN:slir/6 OUT:sri/1
IN:atir/4 OUT:sri/1
access_netparam
IN:srir/3 OUT:fsm/3,fsmTimer/0
force_update
updated
IN:fsmr/0 OUT:ati/1
IN:fsmTimerr/0 OUT:ati/1
IN:atir/4 OUT:slia/5
IN:atir/4 OUT:ati/1
updating
IN:atir/4 OUT:slia/5
IN:fsmr/0
IN:fsmr/1
IN:fsmr/0 OUT:ati/1
IN:atir/4
IN:fsmr/0 OUT:slia/5
timer_trig
maybe_updated
done


last_pos
idle
IN:slir/6 OUT:ati/1
IN:atir/4 OUT:slia/5
IN:slir/6
OUT:fsm/3,fsmTimer/0
IN:slir/6 OUT:sri/1
IN:atir/4 OUT:sri/1
access_netparam
IN:srir/3 OUT:fsm/3,fsmTimer/0
force_update
updated
IN:fsmr/0 OUT:ati/1
IN:fsmTimerr/0 OUT:ati/1
IN:atir/4 OUT:slia/5
IN:atir/4 OUT:ati/1
updating
IN:atir/4 OUT:slia/5
IN:fsmr/0
IN:fsmr/1
IN:fsmr/0 OUT:ati/1
IN:atir/4
IN:fsmr/0 OUT:slia/5
timer_trig
maybe_updated
done

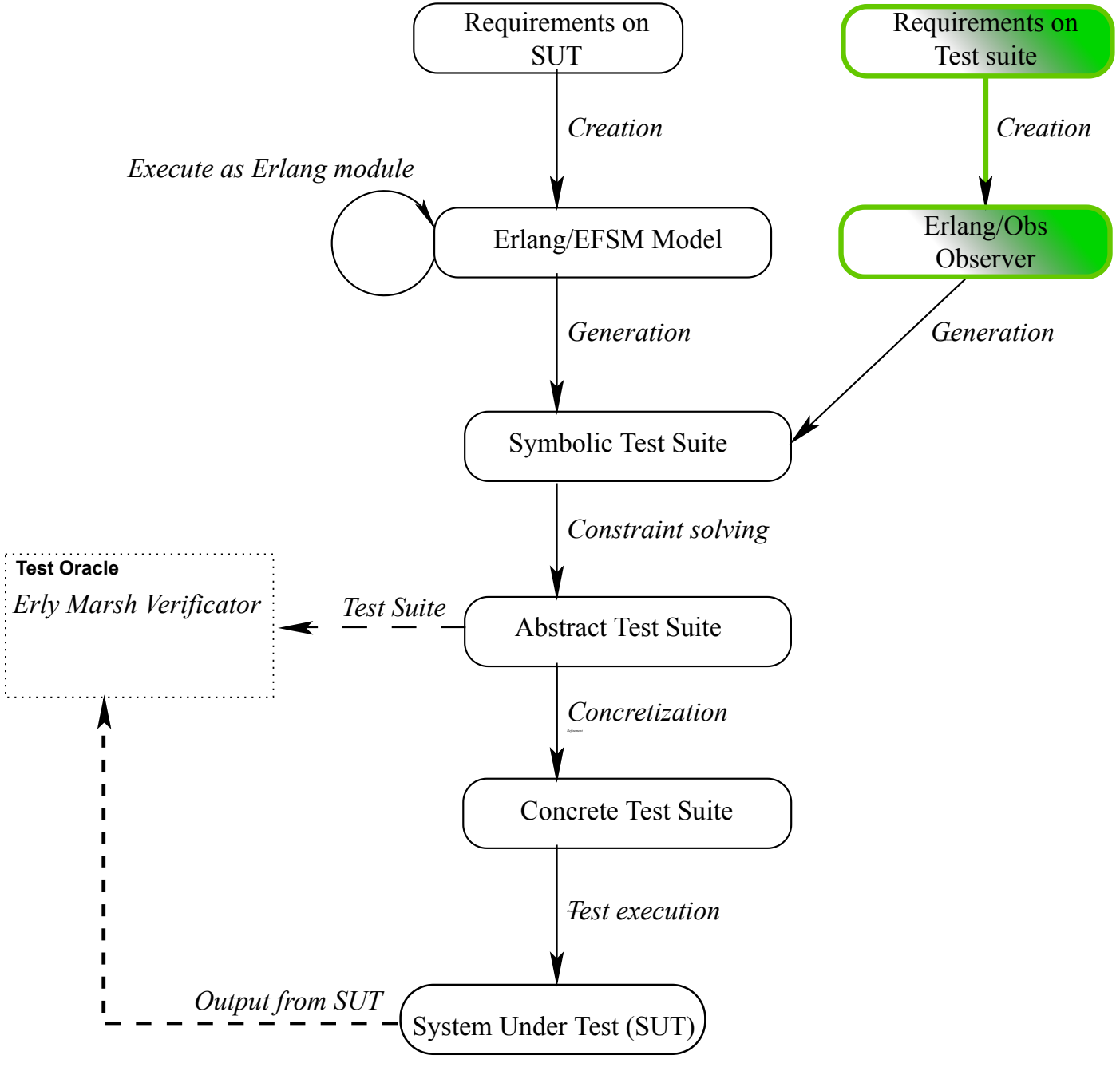
Erlang/EFSM transition clause

```
idle(slir,MSISDN,LocType,MaxAge,SMSTextType,NetPar,EnforcePSI)
  when EnforcePSI==false,
       hlrlocMethod(MSISDN)==ati ->
    if
      is_record(NetPar,netparam) ->
        send_fsm(NetPar),
        fsmTimerr(),
        {next_state,force_update};
      true ->
        send_sri(MSISDN),
        {next_state,access_netparam}
    end.

send_fsm(NetP) ->
    fsm(NetP#netparam.mscid,NetP#netparam.imsi,NetP#netparam.lmsi).

send_sri(MSID) ->
    sri(MSID)
```

Partial evaluation of transition clauses gives us Erlang/EFSM edge clauses.

Erlang/EFSM edge clauses

```
idle(slir,MSISDN,LocType,MaxAge,SMSTextType,NetPar,EnforcePSI)
 when EnforcePSI==false,
      hlrlocMethod(MSISDN)==ati,
      is_record(NetPar,netparam) ->
  fsm(NetPar#netparam.mscid,NetPar#netparam.imsi,NetPar#netparam.lmsi),
  fsmTimerr(),
  {next_state,force_update};

idle(slir,MSISDN,LocType,MaxAge,SMSTextType,NetPar,EnforcePSI)
 when EnforcePSI==false,
      hlrlocMethod(MSISDN)==ati,
      not is_record(NetPar,netparam) ->
  sri(MSISDN),
  {next_state,access_netparam}.
```


Requirements on SUT
Requirements on Test suite
Creation
Creation
Execute as Erlang module
Erlang/EFSM Model
Erlang/Obs Observer
Generation
Generation
Symbolic Test Suite
Constraint solving
Test Oracle
Erly Marsh Verificator
Test Suite
Abstract Test Suite
Concretization
Concrete Test Suite
Test execution
Output from SUT
System Under Test (SUT)

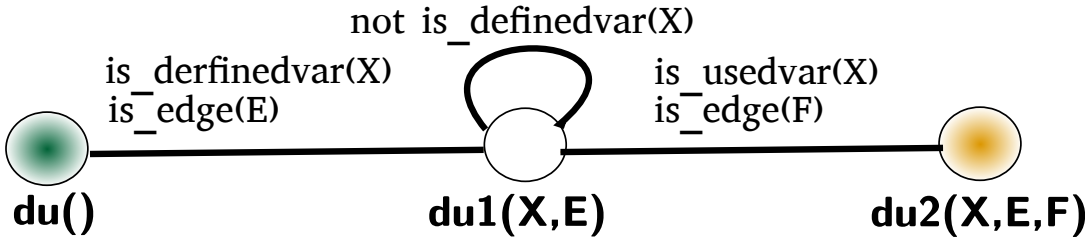
Specifying Test Case Selection

- Typically a tool can generate *huge* numbers of test cases from a model
- Often practical limits on test suite size
  - Complicated environments
  - SUT treated as a black box
- How to generate "best" test suite of bounded size ?

Observers

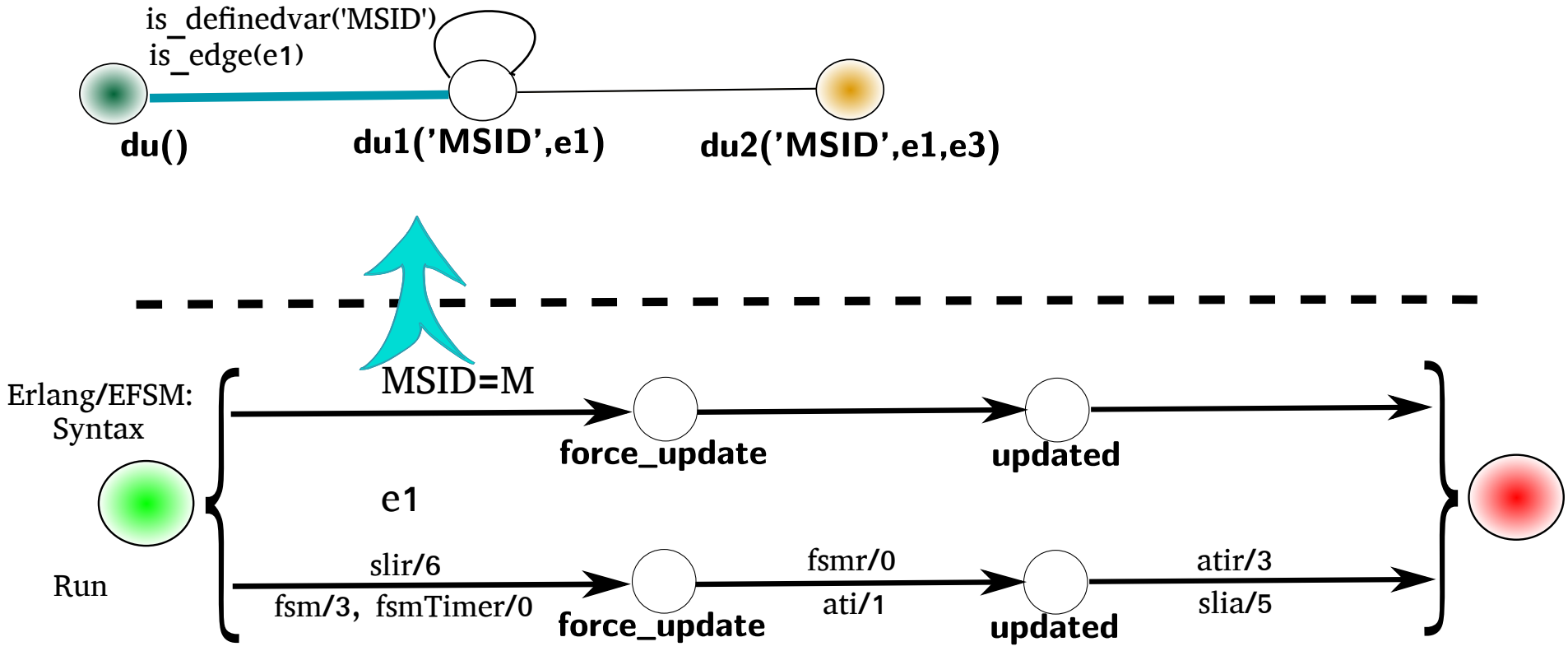
- **Observation:** Many existing coverage criteria in the litterature
  - A coverage criterion is a set of *requirements* (called coverage items) on test suite
  - Examples: *All locations*, *All edges* and *All Definition-Use pairs*
  - Unclear when and what coverage criteria to use
- **Motivation:**
  - Make it possible to specify *any* coverage criteria
- **Approach:**
  - Specify a coverage criterion by the use of (another) state machine
  - Introduces Erlang/Obs as a specification language for observers

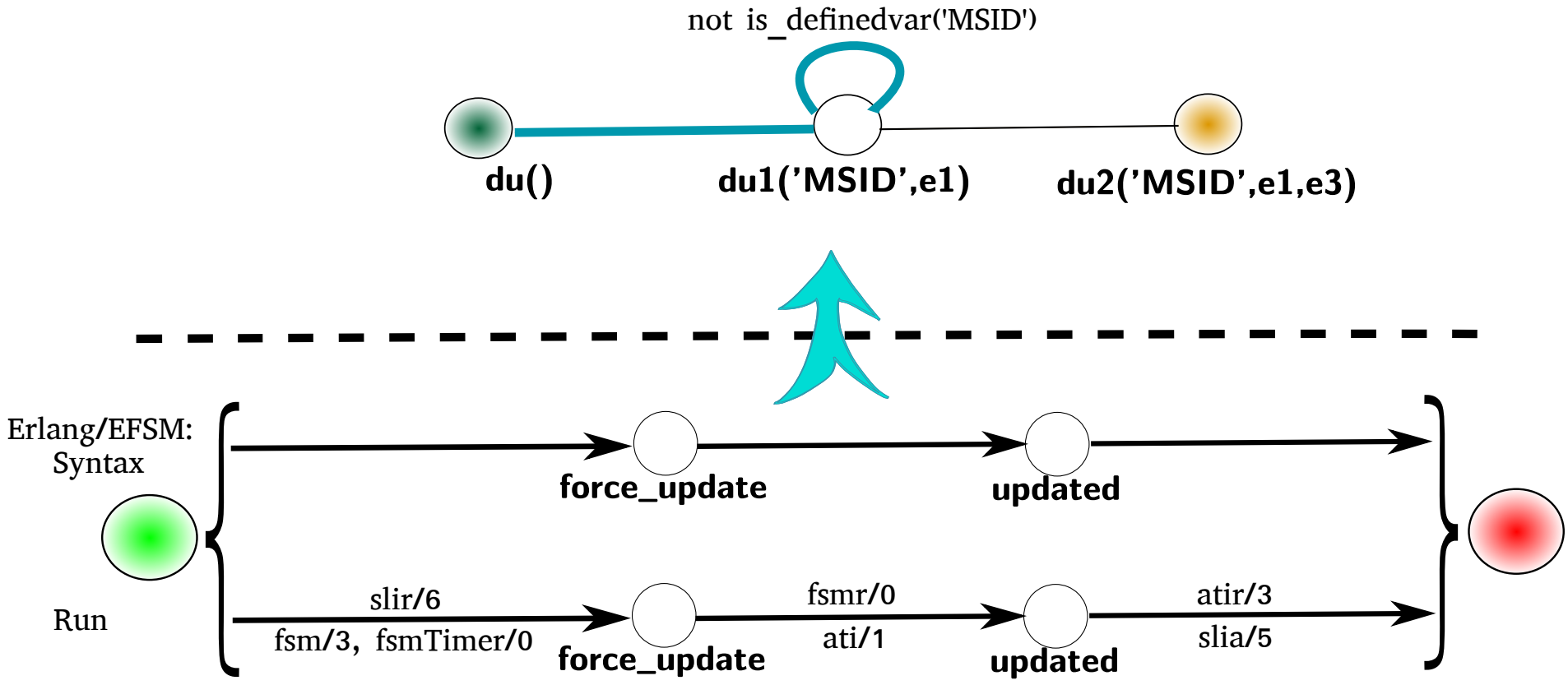
Erlang/Obs Definition-Use

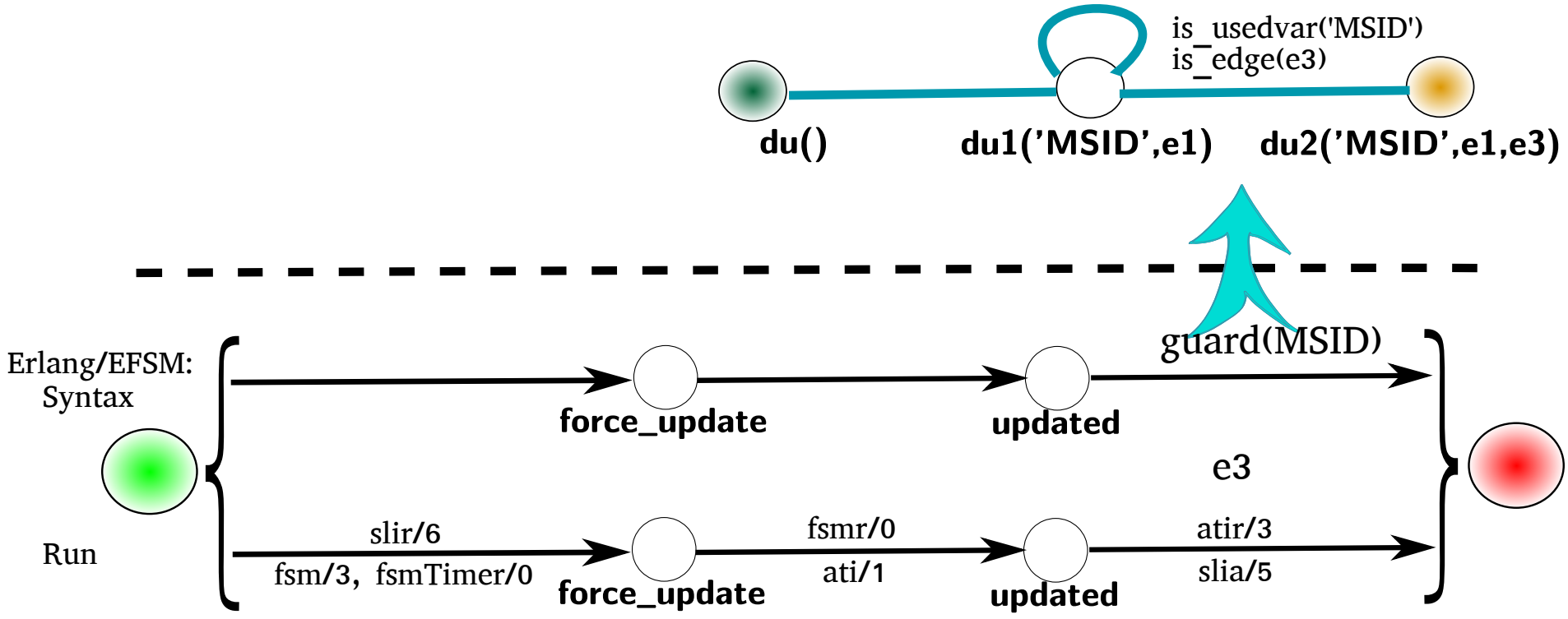


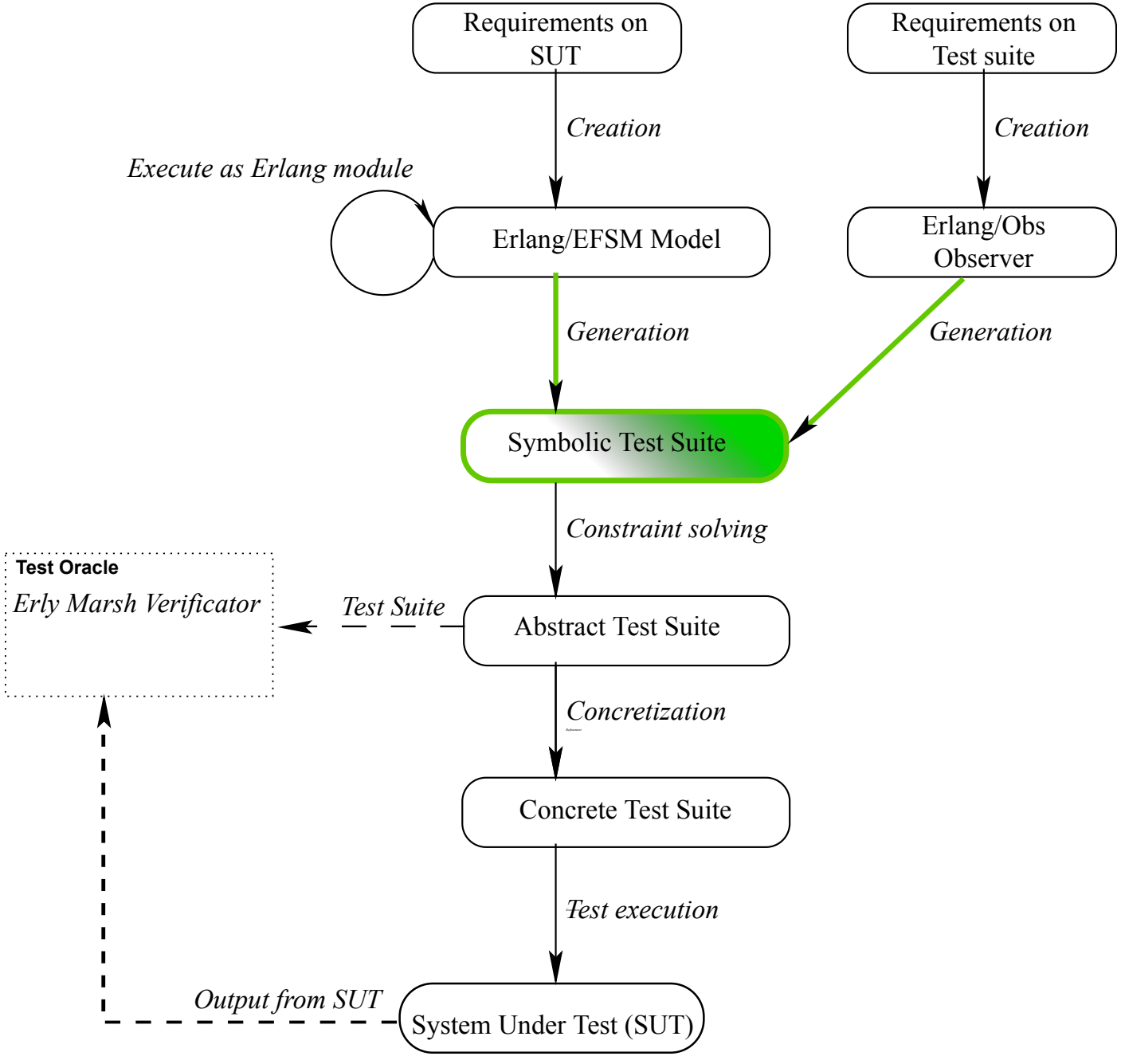

```
du() when is_definedvar(X),is_edge(E) ->
    {next_state,du1(X,E)}.

du1(X,E) when not is_definedvar(X) ->
    {next_state,du1(X,E)};
du1(X,E) when is_usedvar(X),is_edge(F) ->
    {accept_state,du2(X,E,F)}.
```

- is_definedvar(X), is_edge(E) etc. are *observer predicates*
- X and E etc. are variables that are bound by need (compare with Prolog)
- Observer predicates are user definable


is_definedvar('MSID')
is_edge(e1)
du()
du1('MSID',e1)
du2('MSID',e1,e3)
MSID=M
Erlang/EFSM:
Syntax
force_update
updated
e1
Run
slir/6
fsm/3, fsmTimer/0
force_update
fsmr/0
ati/1
updated
atir/3
slia/5

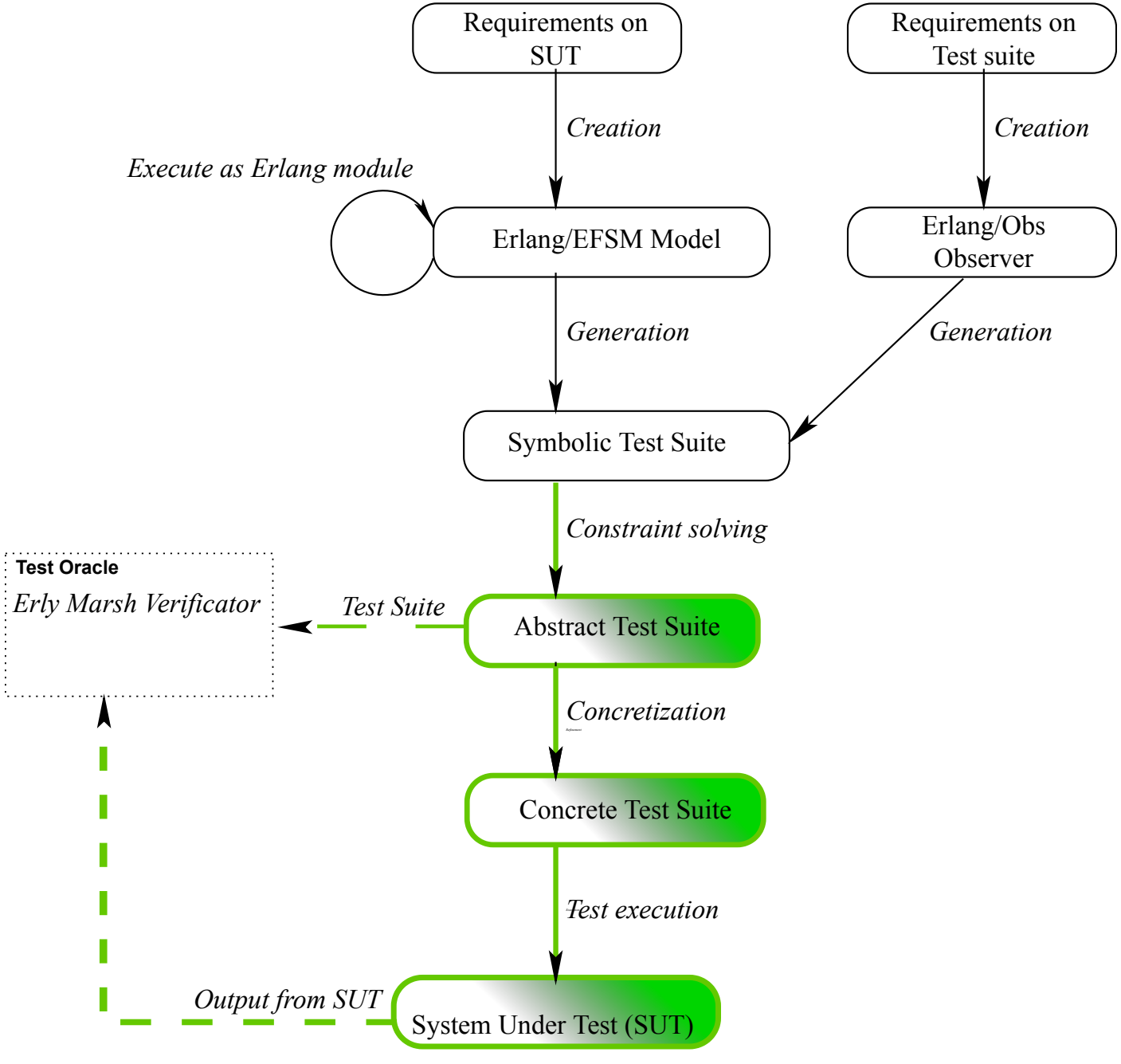

not is_definedvar('MSID')
du()
du1('MSID',e1)
du2('MSID',e1,e3)
Erlang/EFSM:
Syntax
force_update
updated
Run
slir/6
fsm/3, fsmTimer/0
force_update
fsmr/0
ati/1
updated
atir/3
slia/5


is_usedvar('MSID')
is_edge(e3)
du()
du1('MSID',e1)
du2('MSID',e1,e3)
guard(MSID)
Erlang/EFSM:
Syntax
force_update
updated
e3
Run
slir/6
fsm/3, fsmTimer/0
force_update
fsmr/0
ati/1
updated
atir/3
slia/5


Requirements on SUT
Requirements on Test suite
Creation
Creation
Execute as Erlang module
Erlang/EFSM Model
Erlang/Obs Observer
Generation
Generation
Symbolic Test Suite
Constraint solving
Test Oracle
Erly Marsh Verificator
Test Suite
Abstract Test Suite
Concretization
Concrete Test Suite
Test execution
Output from SUT
System Under Test (SUT)

Symbolic Test Suite

- Too many excutions of model $\rightarrow$ Use *symbolic* execution
- Symbolic excutions handles *sets of* executions
- Adapted to handle Erlang/EFSM and Erlang/Obs specifications
- ...skip lots of theoretical stuff ...
- Results in Test Case with symbolic parameters


Requirements on SUT
Requirements on Test suite
Creation
Creation
Execute as Erlang module
Erlang/EFSM Model
Erlang/Obs Observer
Generation
Generation
Symbolic Test Suite
Constraint solving
Test Oracle
Erly Marsh Verificator
Test Suite
Abstract Test Suite
Concretization
Concrete Test Suite
Test execution
Output from SUT
System Under Test (SUT)

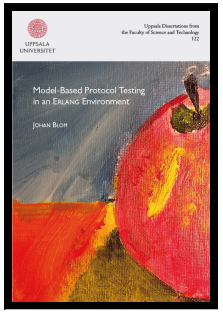
Abstract and Concrete Test Suite

- **Abstract Test Suite**
  - Symbolic parameters can possibly be instantiated in many ways
  - Select each abstract test case from symbolic test case
- **Concrete Test Suite**
  - Concretize generated abstract test suite to format SUT expects
  - SUT specific (handled by Erlang call-back module).
  - In general many concrete possible for each abstract test case

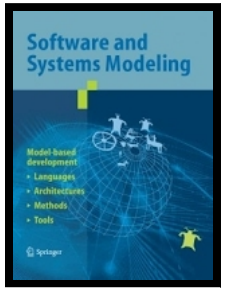
Summary

- Erly Marsh is a Model-Based Testing tool
- Use an abstract model to capture requirements to be tested
  - Introduced Erlang/EFSM for specification of protocols
  - Share syntax and semantics with Erlang
  - Straight forward to transform Erlang/EFSM specification into an executable Erlang module
- Use observers as a tool for flexible specification of coverage criteria
  - Introduced Erlang/Obs, for specification of observers
- Found over 100 faults in that Location server...

References



PhD thesis:
Model-Based Protocol Testing in an Erlang Environment



Software and Systems Modeling:
Special Theme on Model-Based Testing

Github: `http://joh-blo.github.io/erly_marsh/`