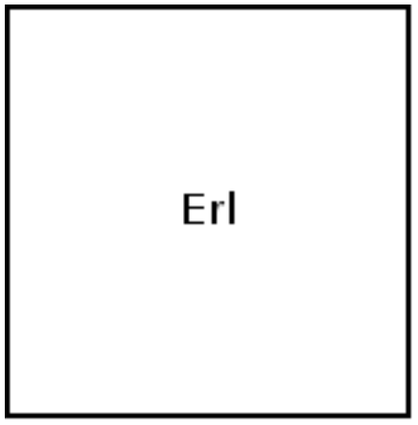
Can I stop testing now??

Test adequacy metrics beyond cover

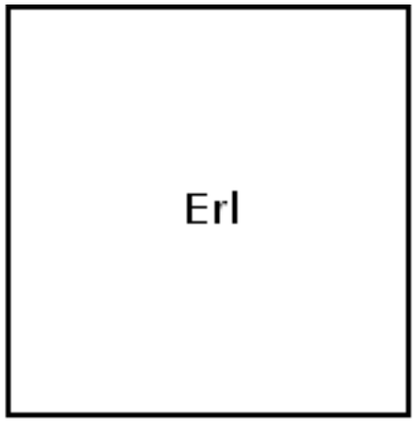
Ramsay Taylor

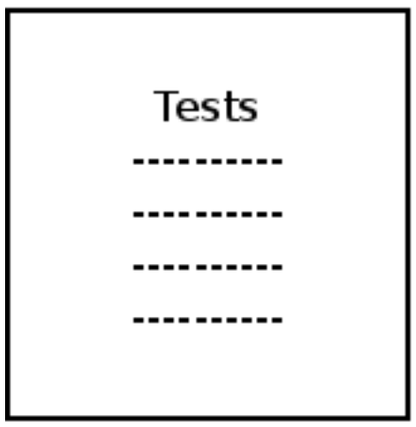
The University Of Sheffield.

Test Adequacy

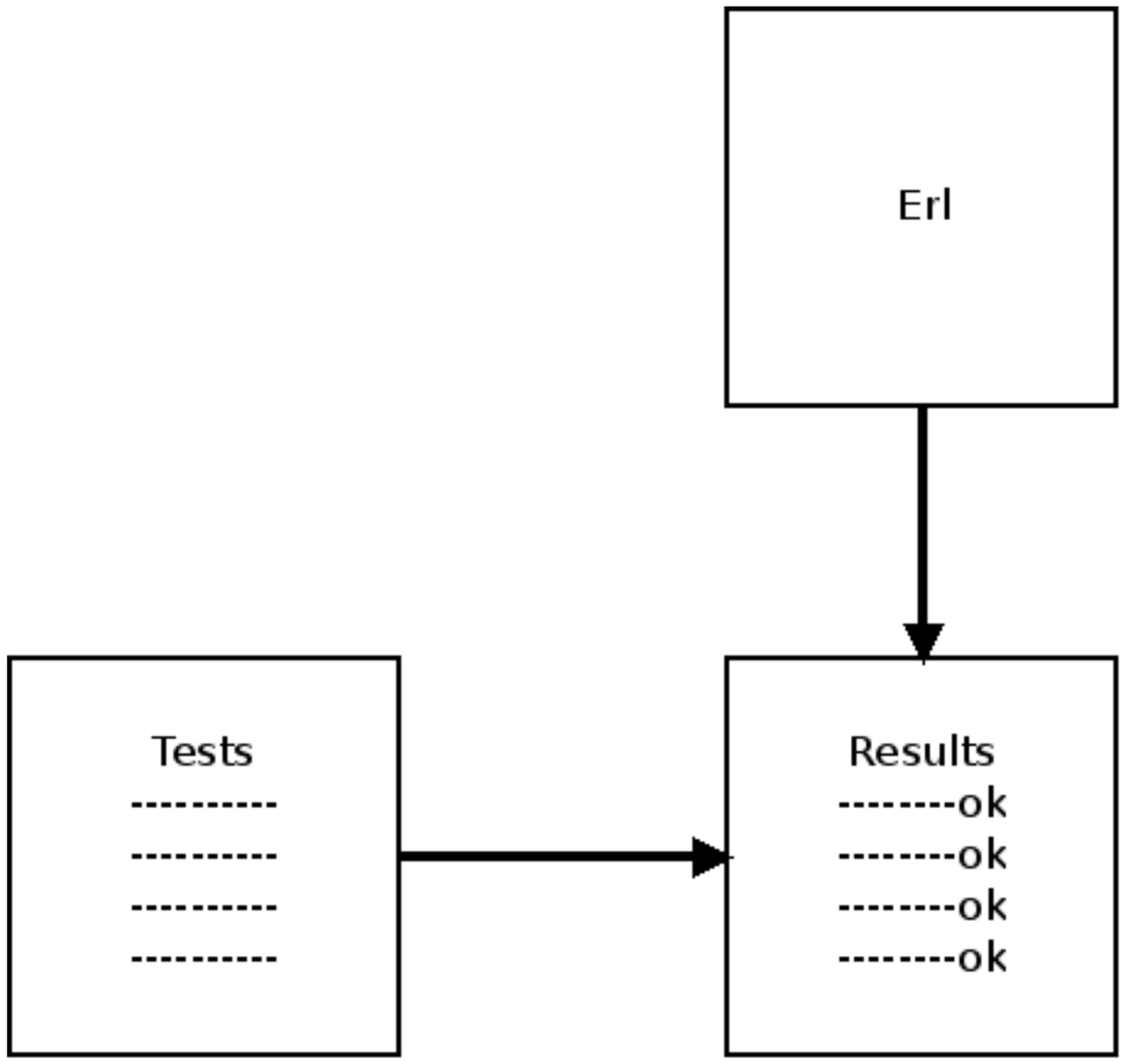


Test Adequacy

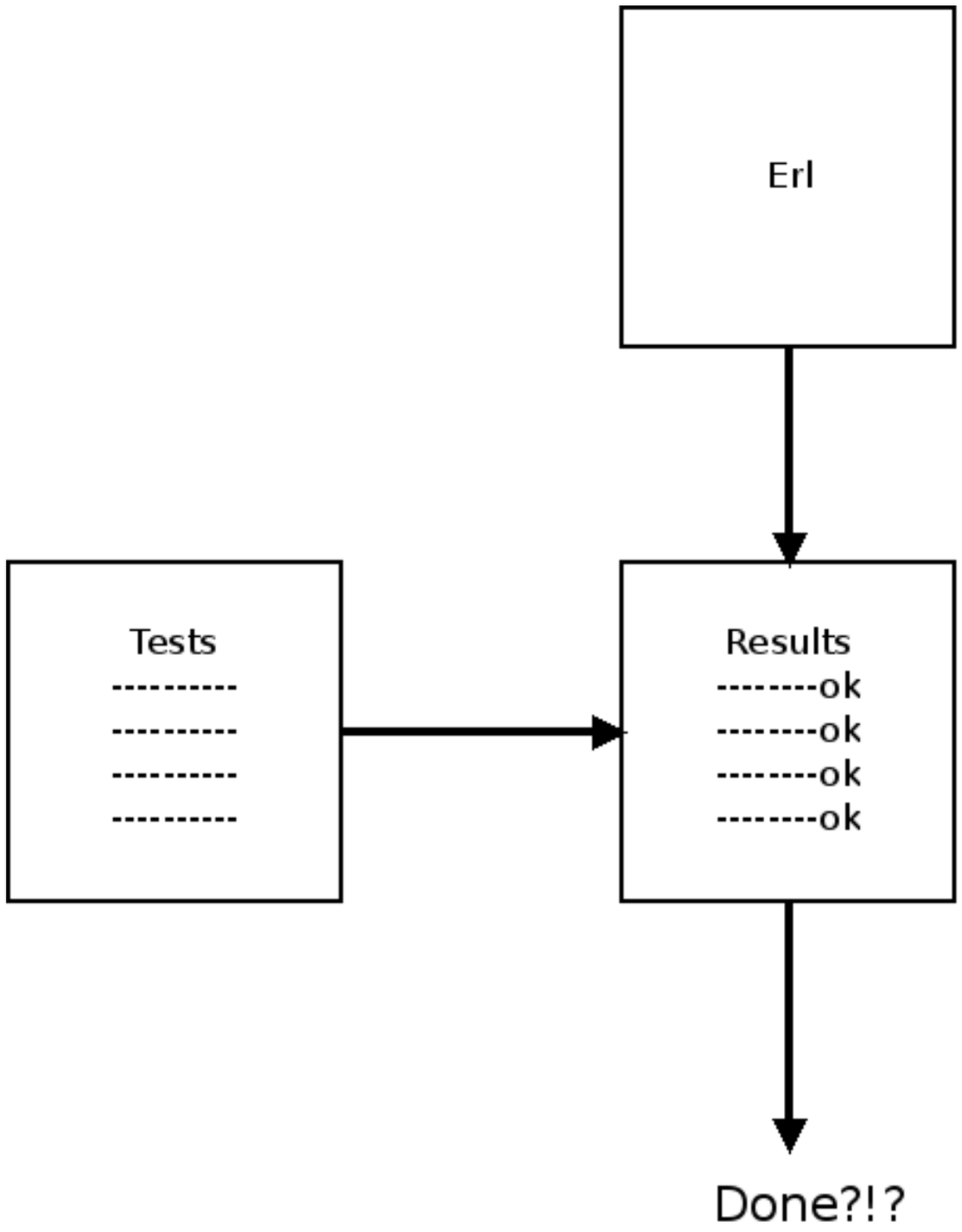




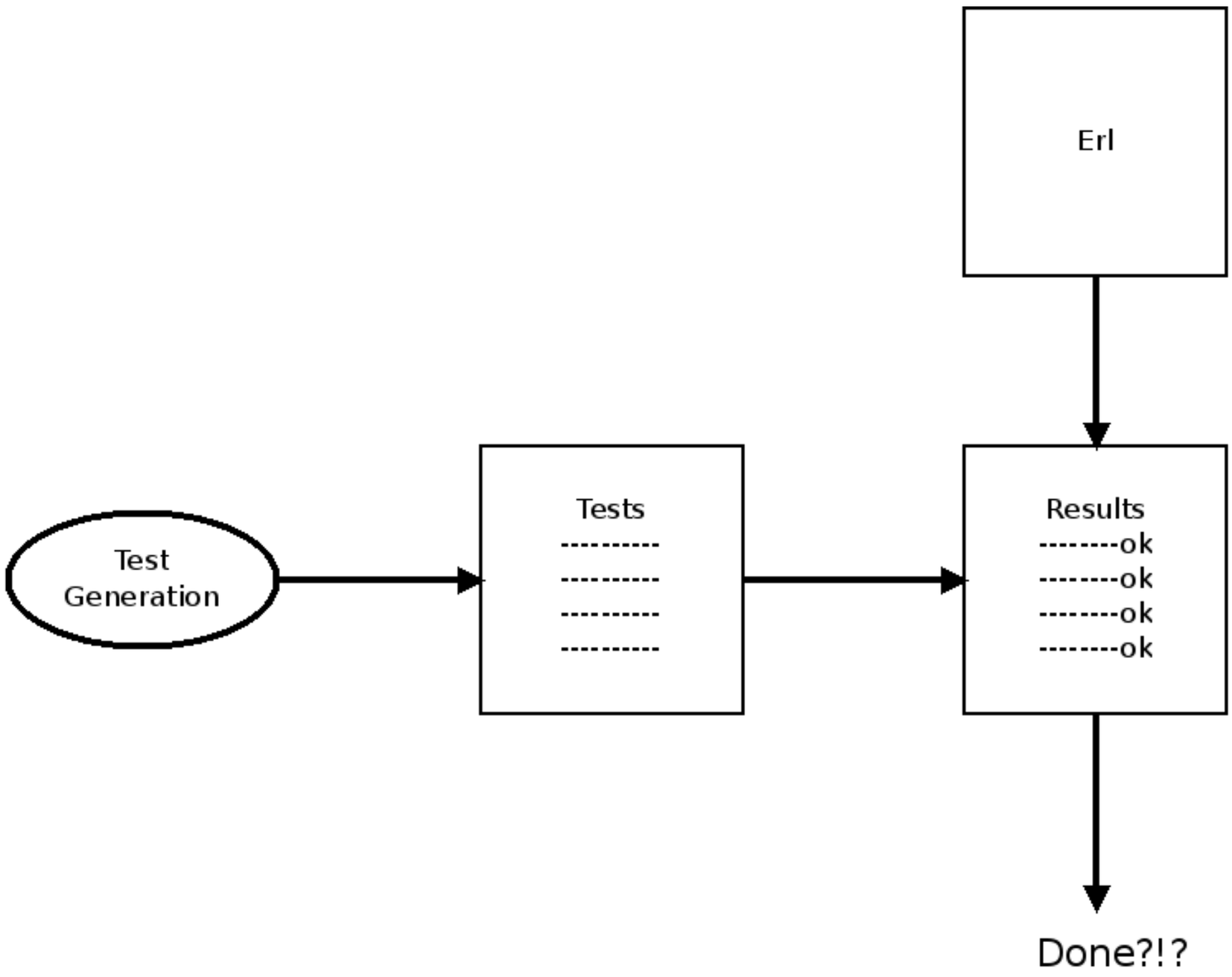
Test Adequacy



Test Adequacy



Test Adequacy



What we want from Test Adequacy?

- Have we tested all of the code?
- Have we tested it in all meaningful ways?
- If the answer to either question is “no”, how can I do better?

In this talk

- Code Coverage
  - Testing all of the code that you have written
  - Testing it in meaningful ways
- Mutation Testing
  - Testing the code you might have written…
  - Testing the code in novel ways
  - Actually checking the answers!
- Model Inference

Code Coverage

```
|   -module(abiftest).
|   -export([dv/2]).
|
|   dv(A,B) ->
0..|       if (A == 0) and (B > 4) ->
0..|            B;
   |          true ->
0..|            B / A
   |       end.
```

Code Coverage

```
|    -module(abiftest).
|    -export([dv/2]).
|
     |    dv(A,B) ->
1..  |        if (A == 0) and (B > 4) ->
1..  |               B;
     |           true ->
0..  |               B / A
     |        end.
```

dv(0,5)

Code Coverage

```
|    -module(abiftest).
|    -export([dv/2]).
|
|    dv(A,B) ->
2..|        if (A == 0) and (B > 4) ->
1..|               B;
   |           true ->
1..|               B / A
   |        end.
```

dv(0,5)

dv(5,5)

Code Coverage

** exception error: an error
occurred when evaluating an
arithmetic expression
    in function abiftest:dv/2
(abiftest.erl, line 8)

dv(0,5)
dv(5,5)
dv(0,2)

Modified Condition/ Decision Coverage

- Instrument not just what got called, but in what way
- Focus on decision points not large blocks of sequential lines
- Measure/require all (reasonable) ways of taking or not taking a branch

MC/DC

```
-module(abiftest).
-export([dv/2]).

dv(A,B) ->
    if (A == 0) and (B > 4) ->
            B;
        true ->
            B / A
    end.
```

dv(0,5)

dv(5,5)

MC/DC

```
-module(abiftest).
-export([dv/2]).

dv(A,B) ->
    if (A == 0) and (B > 4) ->
            B;
       true ->
            B / A
    end.
```

dv(0,5)
dv(5,5)
dv(5,0)

MC/DC

```
-module(abiftest).
-export([dv/2]).

dv(A,B) ->
    if (A == 0) and (B > 4) ->
           B;
       true ->
           B / A
    end.
```

(A == 0) and (B > 4)

- matched: 1
- non-matched: 2

When false:

	matched	non-matched
A == 0	0	2
B > 4	1	1

Pattern Matching

```
-module(abtest).
-export([dv/2]).

dv(0,5) ->
    5;
dv(A,B) ->
    B / A.
```

[0,5]

- matched: 1
- non-matched: 2

When non-matched:

	matched	non-matched
0	0	2
5	1	1

Pattern Matching

```
-module(abcasetest).
-export([dv/2]).

dv(A,B) ->
    case {A,B} of
        {0,5} ->
            B;
        _ ->
            B / A
end.
```

{0,5}

- matched: 1
- non-matched: 2

When non-matched:

	matched	non-matched
0	0	2
5	1	1

Pattern Matching

```
-module(ablisttest).
-export([dv/1]).

dv(Arg) ->
     case Arg of
          [0,5] ->
               5;
          [A,B] ->
               B / A
end.
```

[0,5]

- matched: 1
- non-matched: 2

When non-matched:

	matched	non-matched
0	0	2
5	1	1
empty_list	0	N/A
list_size_mismatch	0	N/A
not_a_list	0	N/A

Pattern Matching

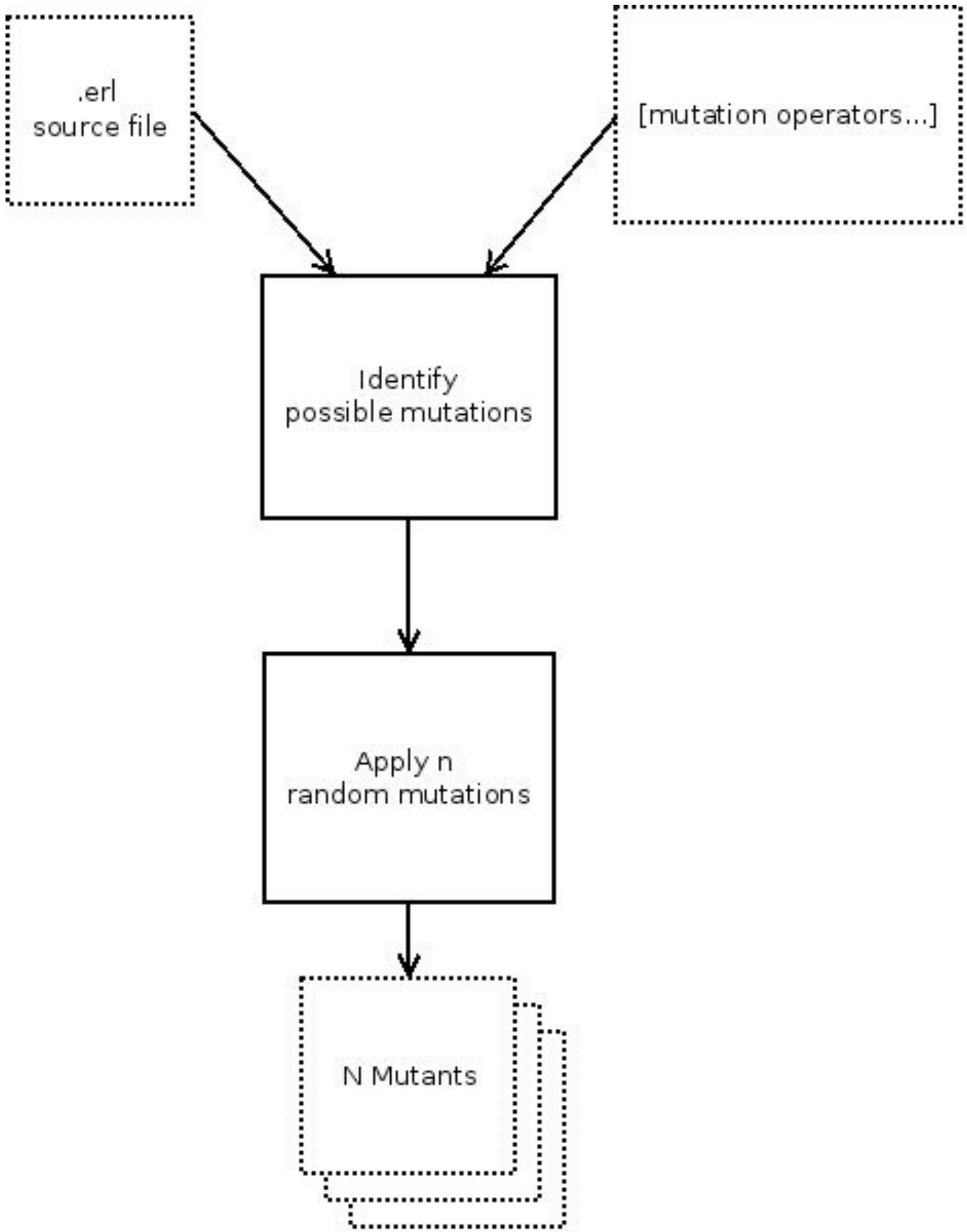
```
dv_proc() ->
    receive
        {From, {A,B}} when (A == 0) and (B > 4) ->
            From ! B;
        {From, {A,B}} ->
            From ! B / A
end.
```

Code coverage limitations

- Only assess the code that you have written, not the code you should have written…
- Says nothing except that the code has been executed and maybe didn't crash.

Mutation Testing

- Deliberately break the code and see if the tests "notice"
- Try to simulate common faults
  - with the system
  - with the programmer…


.erl
source file
[mutation operators...]
Identify
possible mutations
Apply n
random mutations
N Mutants

Test results per mutant

- Fails - Good! It found the fault
- Passed - Bad! It didn't notice the change
  - unless its "semantically equivalent"

mu2 Framework

- Allows domain-specific operators to be supplied
- Uses the Wrangler refactoring library to allow rich and subtle mutation operators

mu2 Operators

```
{plus_to_minus,
 ?MUTATION_MATCH("X@ + Y@"),
 ?MUTATION_EXCHANGE("X@ + Y@", "X@ - Y@")}
```

mu2 Operators

```
{swap_case_order,
?MUTATION_MATCH("if Guards@@@ -> Body@@@ end"),
?MUTATION("if Guards@@@ -> Body@@@ end",
    begin
     A = random:uniform(length(Guards@@@)),
     B = random_not_n(length(Guards@@@), A),
     NewGuards@@@ = swap(Guards@@@, A, B),
     NewBody@@@ = swap(Body@@@, A, B),
         ?TO_AST("if NewGuards@@@ -> NewBody@@@ end")
    end)
}
```

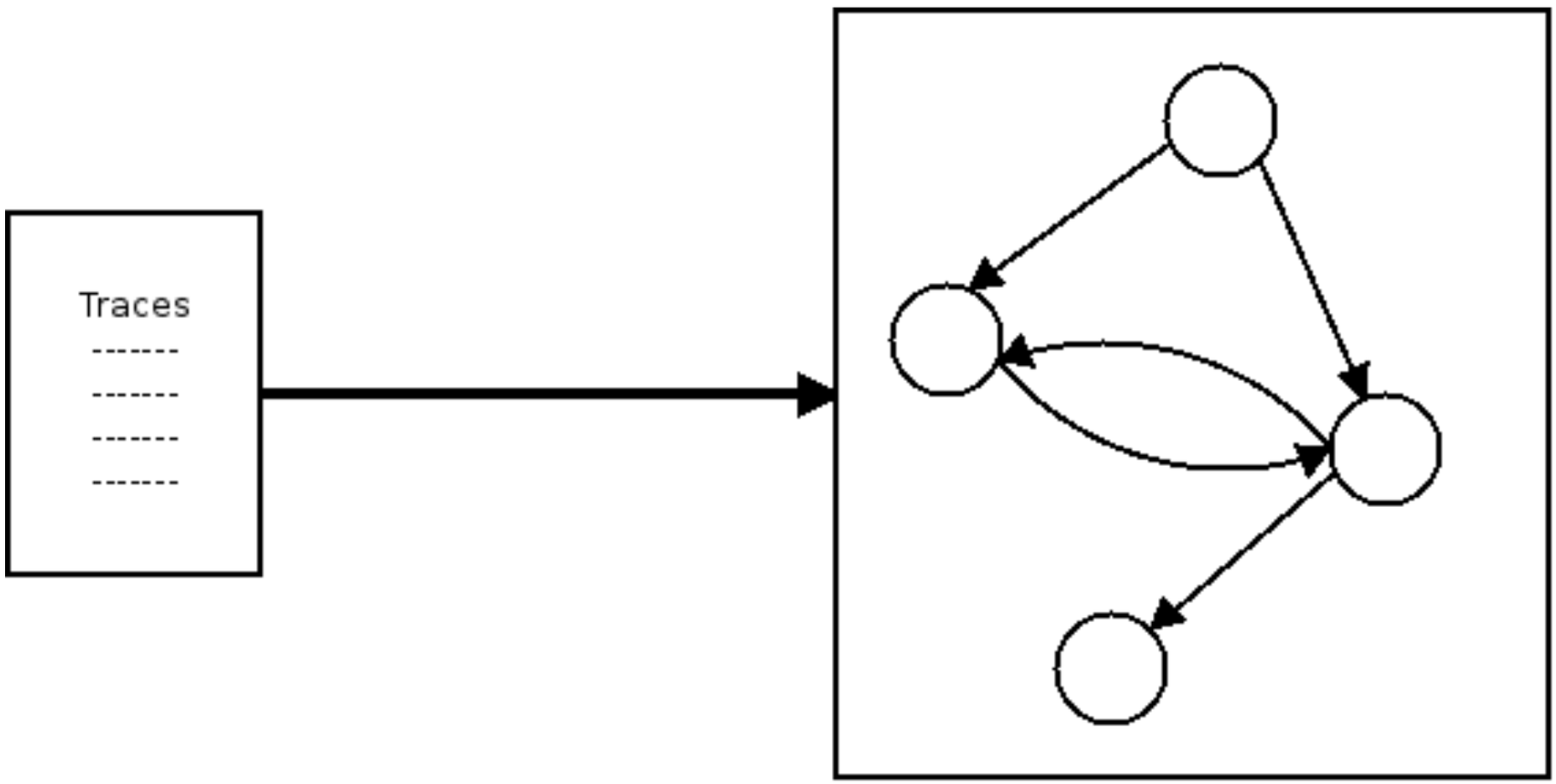
mu2 Operators

```
{decrease_timeout,
?MUTATION_MATCH("receive
                    Pats@@@ when Guards@@@ -> Body@@@
                    after APats@@@ -> ABody@@@
                    end"),
?MUTATION("receive
              Pats@@@ when Guards@@@ -> Body@@@
              after APats@@ -> ABody@@ end",
          begin
             NewAPats@@ = lists:map(fun(Pat@) ->
                                      ?TO_AST("(Pat@ / 100)")
                                    end,
                                    APats@@),
             ?TO_AST("receive
                        Pats@@@ when Guards@@@ -> Body@@@
                        after NewAPats@@ -> ABody@@
                        end")
          end)}
```

Mutation testing limitations

- Have to compile lots of mutants
- Have to run the test set lots of times

Model Inference



Conclusions

- You should be testing your tests
  - but don't ask me to recurse again ;)
- Code coverage is cheap so use it
  - but do it properly!
- Mutation testing is a useful complement
  - but its expensive so use it wisely…
- Model inference is cool!
  - look into it

Prototypes…

https://github.com/ramsay-t/Smother

https://github.com/ramsay-t/mu2

http://statechum.sourceforge.net/

Questions?