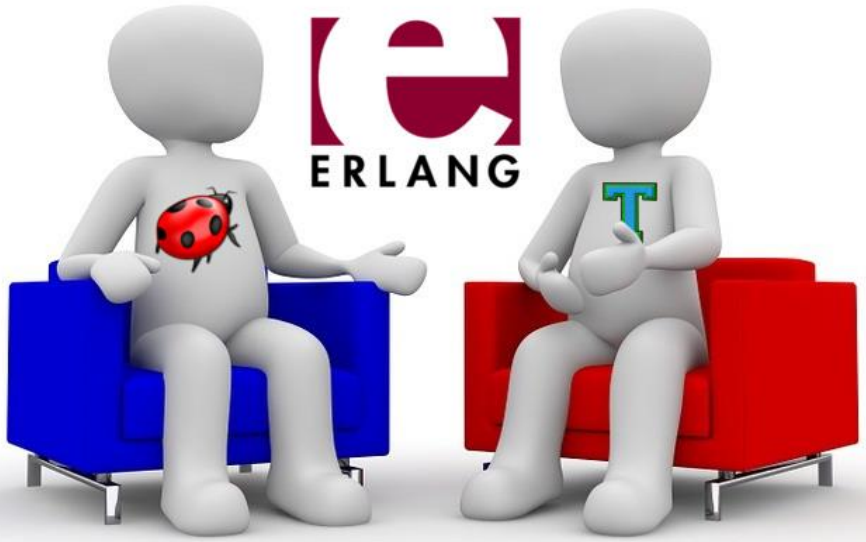


Debugging meets testing in Erlang

Salvador Tamarit, Adrián Riesco,

Enrique Martín-Martín, Rafael Caballero

(U. Complutense & U. Politécnica, Madrid, Spain)

10th International Conference on Tests & Proofs

TAP 2016

5-7 July 2016, Vienna, Austria

Outline

Motivation

Declarative debugging

Unit Testing in Erlang

Our proposal

Conclusions

Outline

Motivation

Motivation

Thinking about the rôle of debugging

Debugging → Most labor-intensive task in software development

1) Many different computations to consider, correct and wrong

2) Complexity: compare

Intended meaning of the each piece of code

The value actually computed

Motivation

Thinking about the rôle of debugging

Debugging → Compare intended and computed values

Motivation

Thinking about the rôle of debugging

Debugging → Sometimes comparing expected and obtained results is not so easy!

Motivation

Thinking about the rôle of debugging

Debugging → Many questions!

Motivation

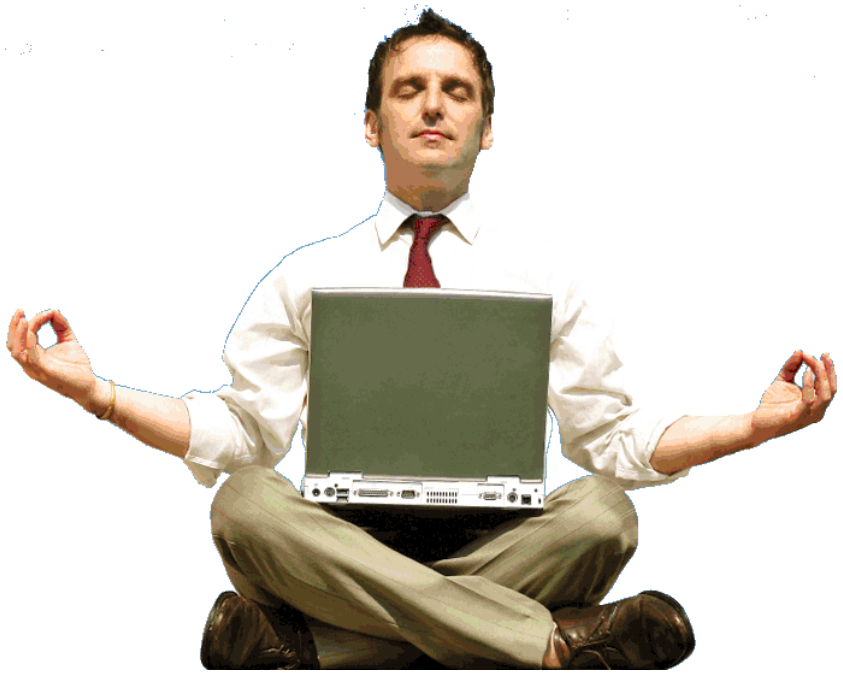
Thinking about the rôle of debugging

Debugging → Goal: find an unexpected result

Motivation

Thinking about the rôle of debugging

Debugging sessions → Intense reflection about the code
→ deep understanding of the program



Motivation

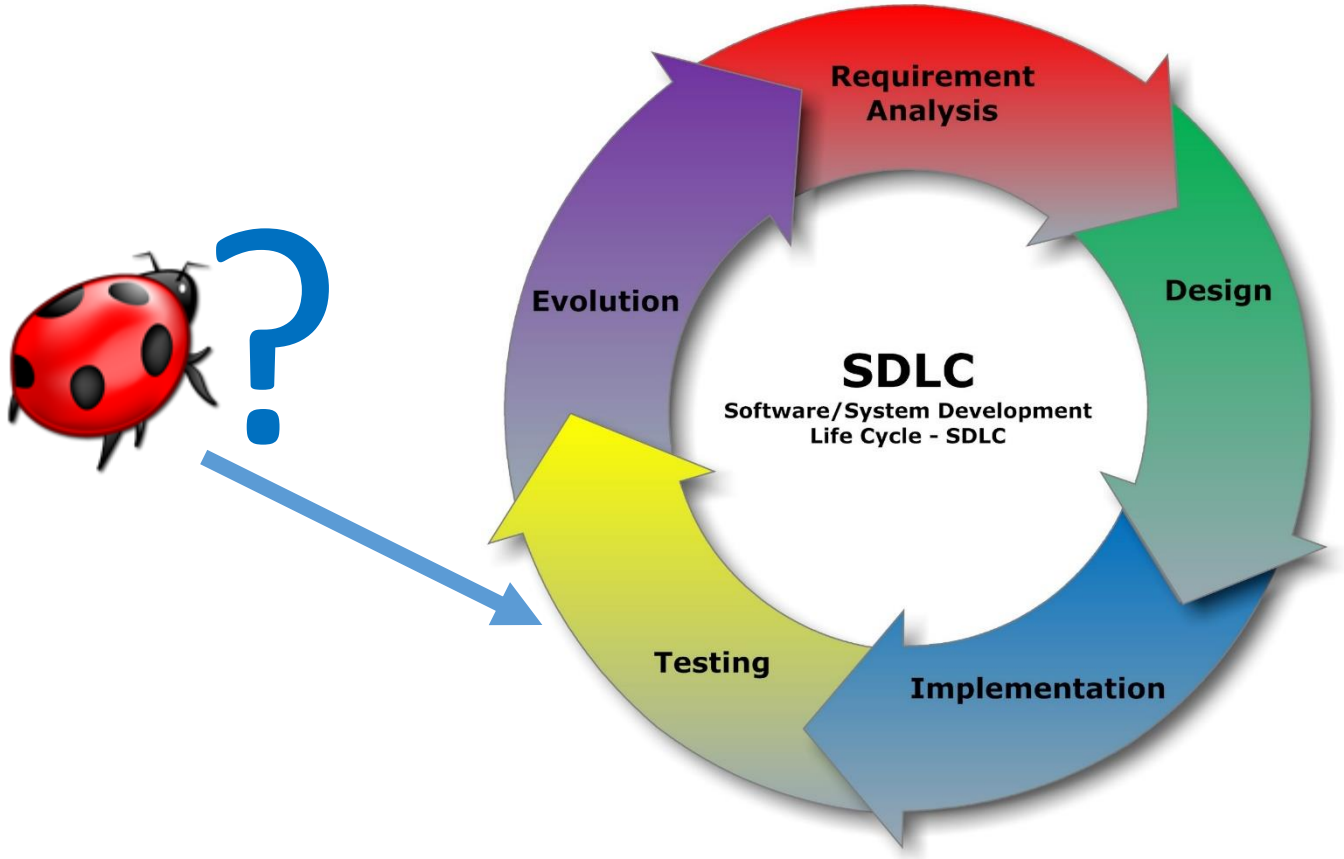
Thinking about the rôle of debugging

Debugging → What happens with all this knowledge once the bug has been found?

Motivation

Thinking about the rôle of debugging

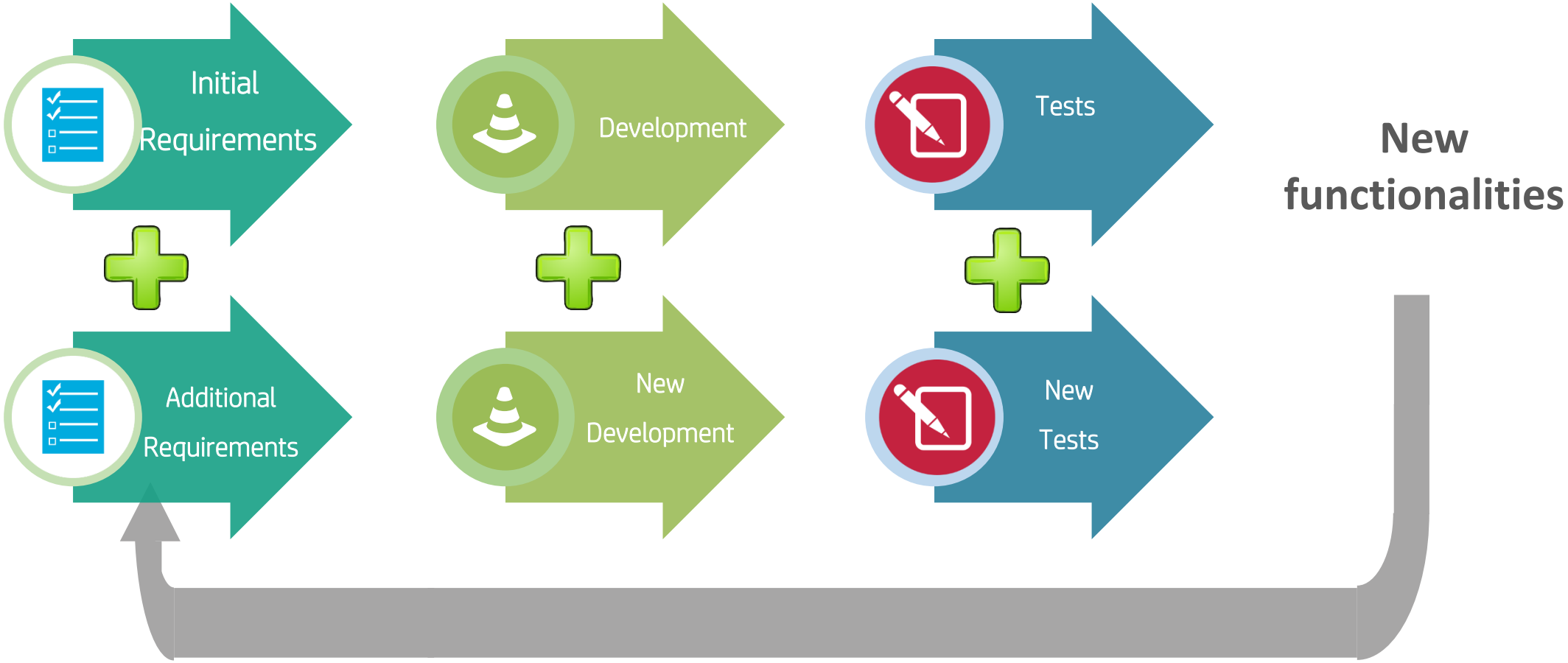
Is debugging really part of the software development life cycle?



Motivation

Thinking about the rôle of debugging

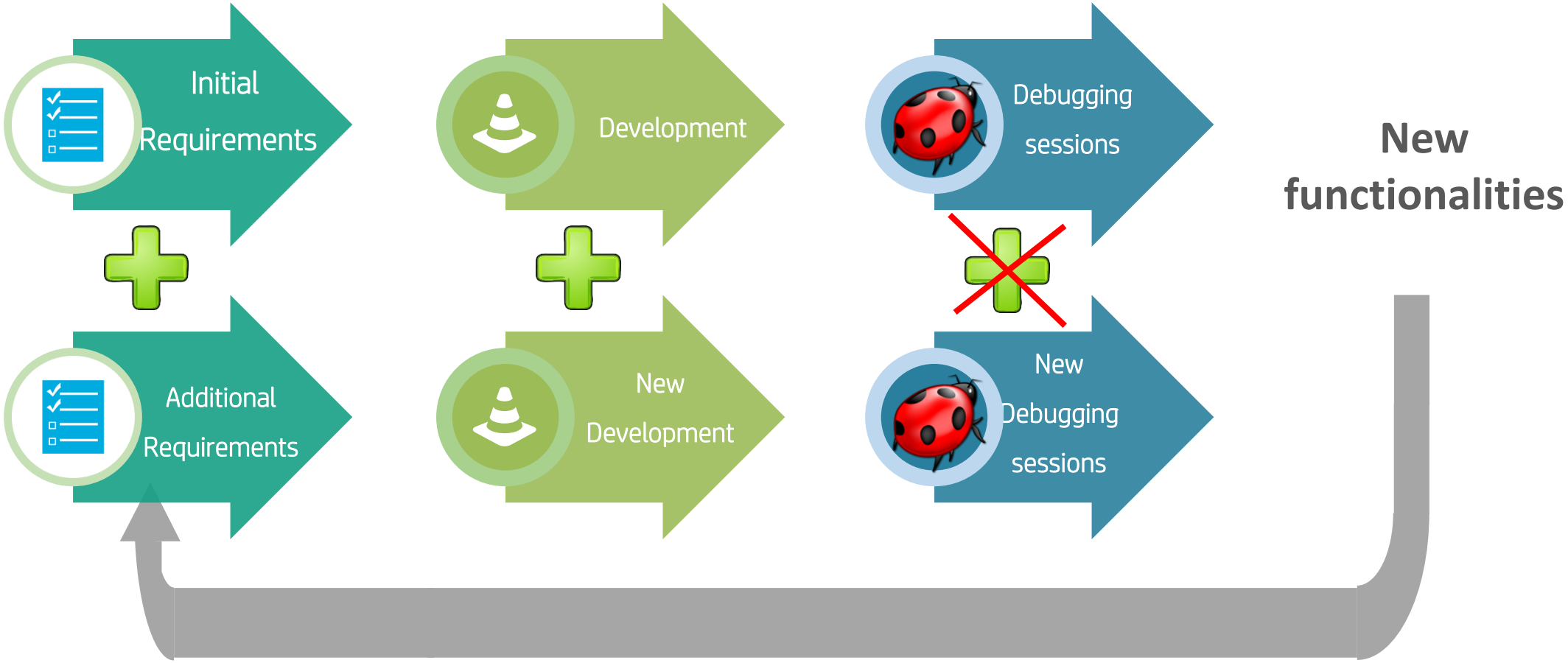
Example: Testing



Motivation

Thinking about the rôle of debugging

What about debugging?



Motivation

Thinking about the rôle of debugging

Our Goal → Integrate **debugging** in the software development life cycle

HowTo

1) Debugging sessions generate tests

2) Tests used during debugging sessions

Outline

Motivation

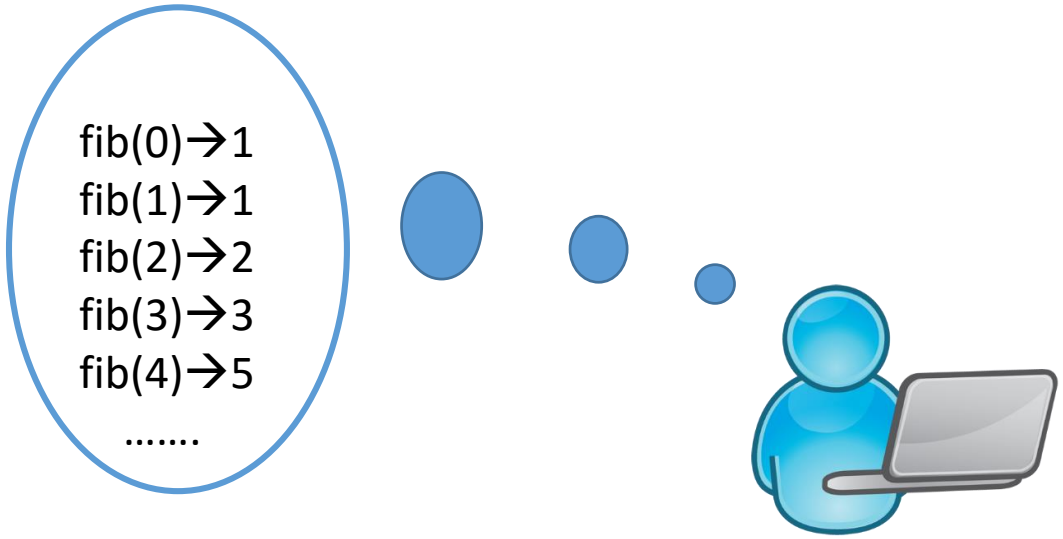
Declarative debugging

Declarative debugging

- ✓ Initial symptom: unexpected result detected by the user
- ✓ Automatically generates a computation tree
  - ✓ Node: Computation steps with its result
  - ✓ Children: Subcomputations needed to obtain the result at the parent node
  - ✓ Root: initial symptom
- ✓ Validity: The user determines the validity of the nodes
- ✓ Goal: Find a buggy node, an invalid node with valid children
  → incorrect piece of code

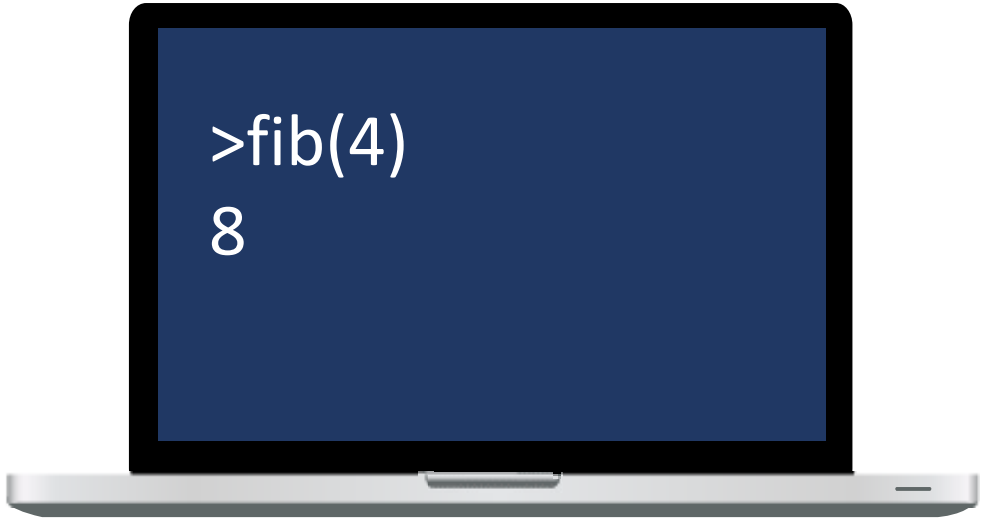
Erlang example: Fibonacci

```
-module(fib).
-export([fib/1]).

fib(0) -> 1;
fib(1) -> 2;
fib(N) -> fib(N-1)+ fib(N-2).
```

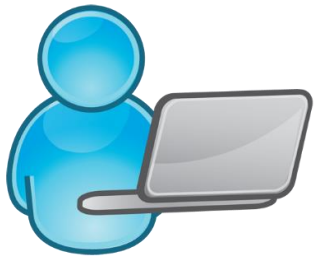


Erlang example: Fibonacci

```
-module(fib).
-export([fib/1]).

fib(0) -> 1;
fib(1) -> 2;
fib(N) -> fib(N-1)+ fib(N-2).
```

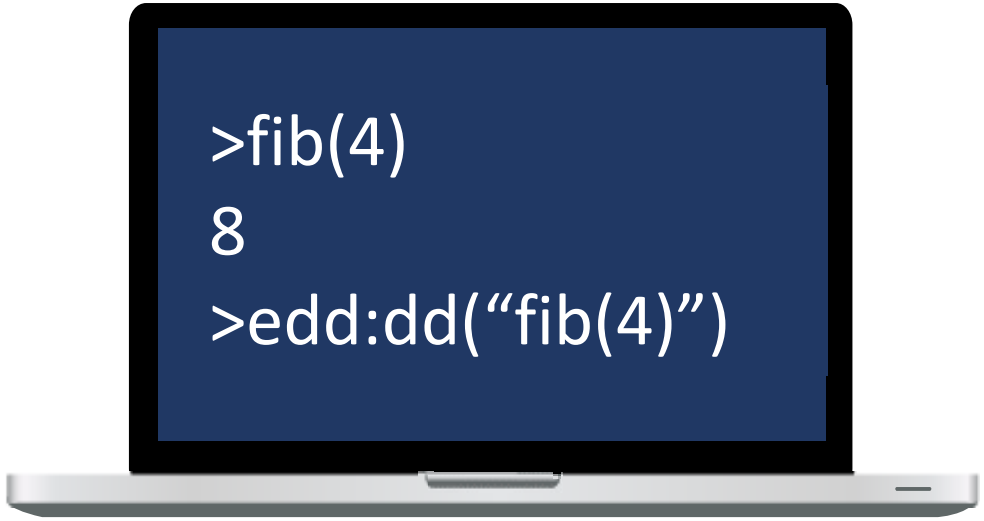


fib(0)→1
fib(1)→1
fib(2)→2
fib(3)→3
fib(4)→5
.....



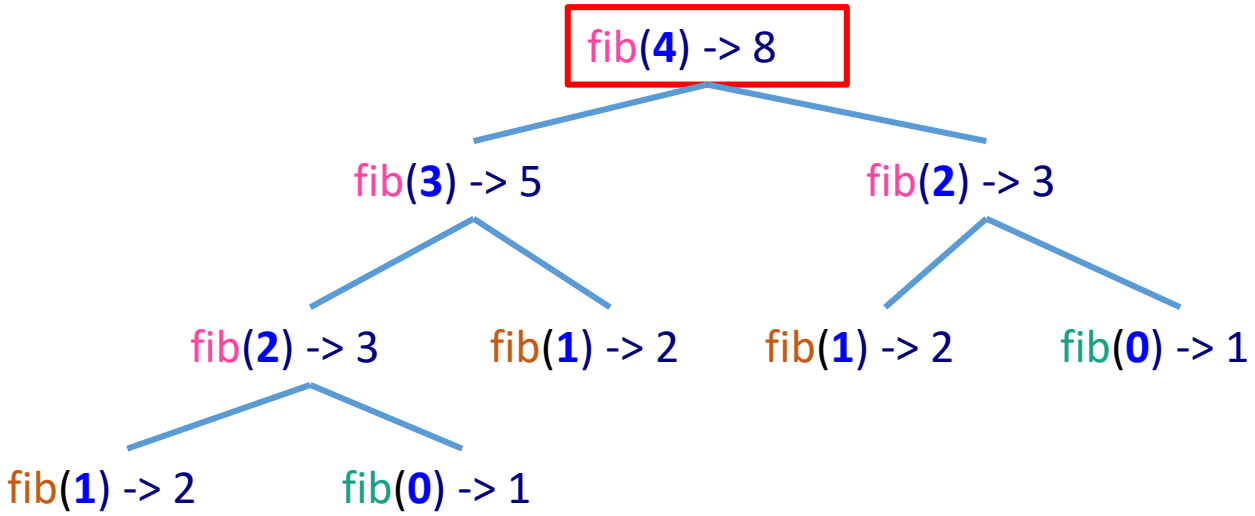
Erlang example: Fibonacci

```
-module(fib).
-export([fib/1]).

fib(0) -> 1;
fib(1) -> 2;
fib(N) -> fib(N-1)+ fib(N-2).
```



Erlang example: Fibonacci

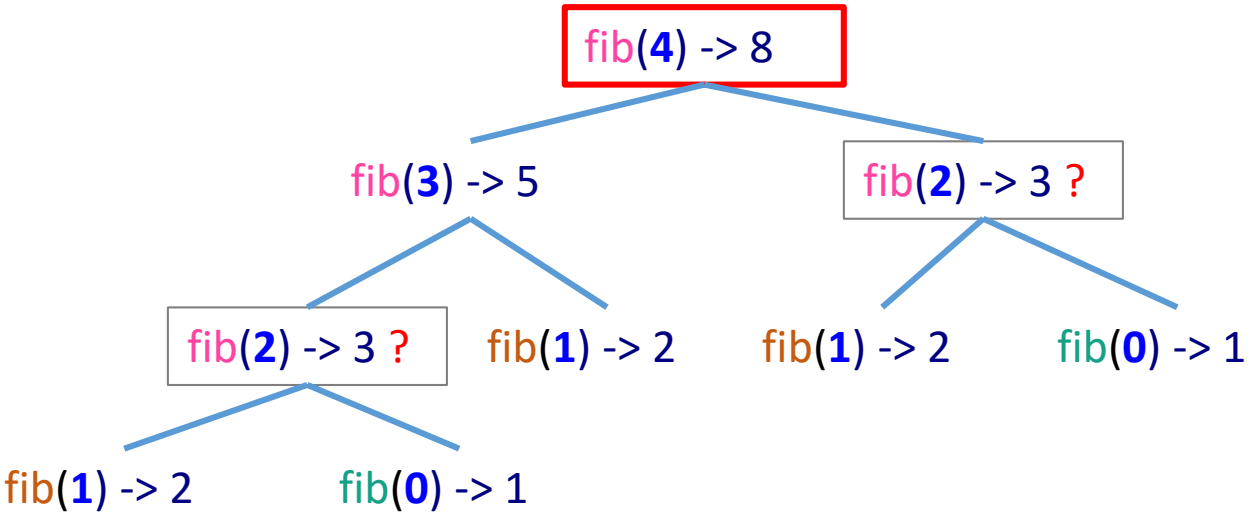
```
-module(fib).
-export([fib/1]).

fib(0) -> 1;
fib(1) -> 2;
fib(N) -> fib(N-1)+ fib(N-2).
```



fib(0)→1
fib(1)→1
fib(2)→2
fib(3)→3
fib(4)→5
.....

Erlang example: Fibonacci

```
-module(fib).
-export([fib/1]).

fib(0) -> 1;
fib(1) -> 2;
fib(N) -> fib(N-1)+ fib(N-2).
```

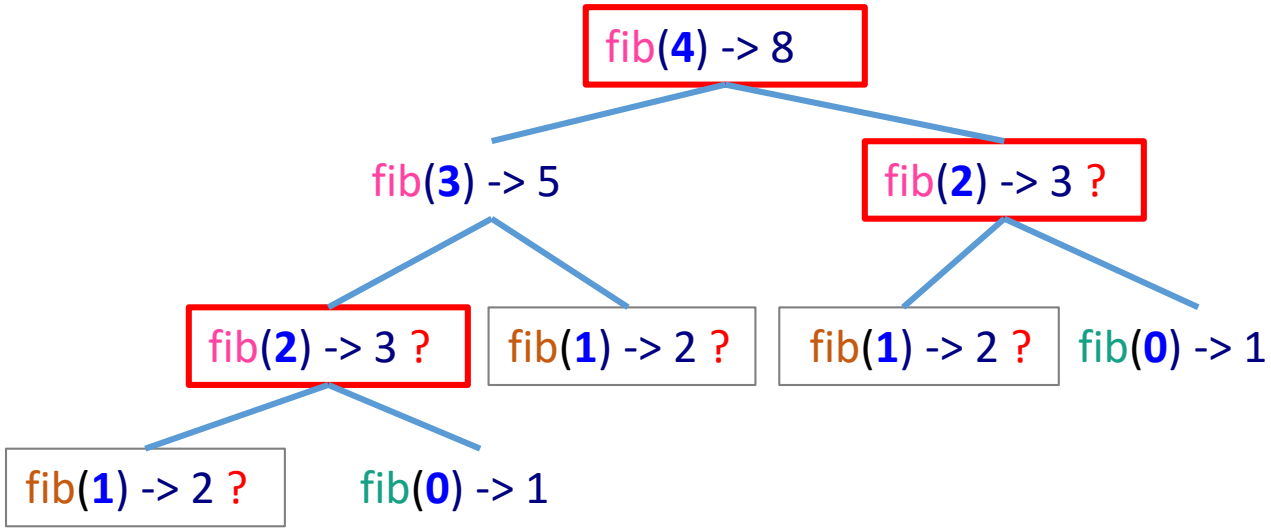


fib(0)→1
fib(1)→1
fib(2)→2
fib(3)→3
fib(4)→5
.....

Invalid !

Erlang example: Fibonacci

```
-module(fib).
-export([fib/1]).

fib(0) -> 1;
fib(1) -> 2;
fib(N) -> fib(N-1)+ fib(N-2).
```

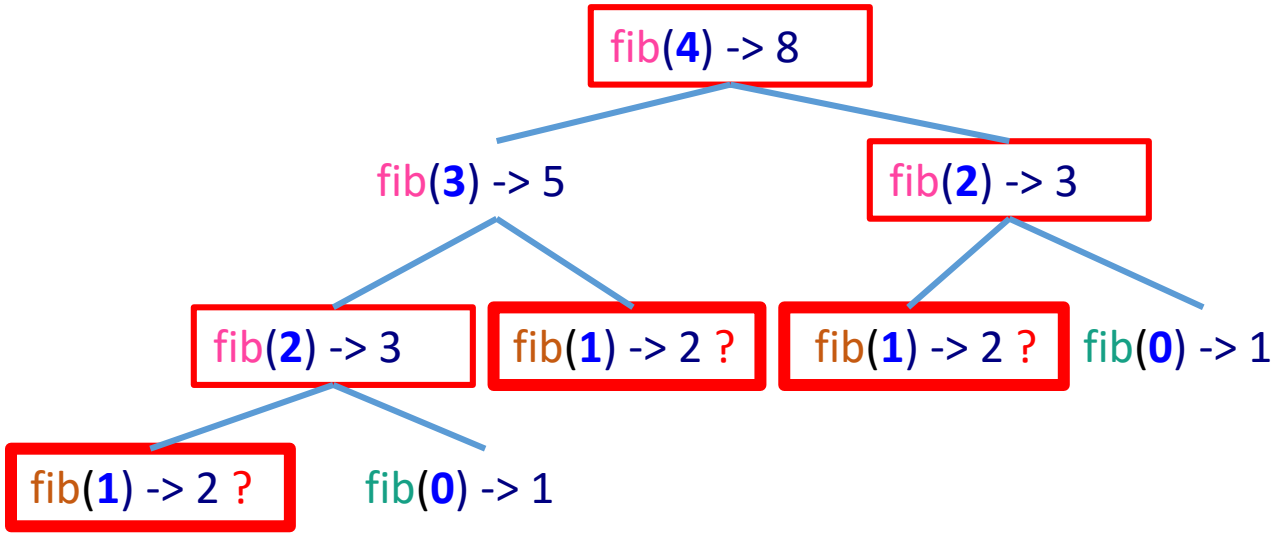


fib(0)→1
fib(1)→1
fib(2)→2
fib(3)→3
fib(4)→5
.....

Invalid !

Erlang example: Fibonacci

```
-module(fib).
-export([fib/1]).

fib(0) -> 1;
fib(1) -> 2;
fib(N) -> fib(N-1)+ fib(N-2).
```



fib(0)→1
fib(1)→1
fib(2)→2
fib(3)→3
fib(4)→5
.....

Buggy nodes !

Outline

Motivation

Declarative debugging

Unit Testing in Erlang

Unit Testing in Erlang

- ✓ Unit tests: Check units of code in (relative) isolation
- ✓ Created by Kent Beck in 1998 (Sunit for Smalltalk)
- ✓ xUnit, a big family: JUnit, Runit, NUnit
- ✓ EUnit: Unit test framework for language Erlang

Unit Testing in Erlang

Erlang

- ✓ Functional: functions as basic pieces of code
- ✓ Concurrent: deals with thousands of processes readily
- ✓ Dynamic Typing: variables declarated without types
- ✓ Hot Swapping, single assignment, eager evaluation…

Erlang example

```
-module(quicksort).
-export([qs/2, leq/2]).

qs(_, []) -> [];
qs(F, [E|R]) -> {A, B} = partition(F, E, R),  qs(F, B) ++ [E] ++ qs(F, A).

partition(_, _, []) -> {[], []};
partition(F, E, [H|T]) ->
        {A, B} = partition(F, E, T),
        case F(H, E) of
                true -> {[H|A], B};
                false -> {A, B}
        end.
leq(A, B) -> A =< B.
```

Unit Testing in Erlang

Erlang EUnit example

```
-include_lib("eunit/include/eunit.hrl").

quicksort_test() ->
    ?assertEqual( qs(fun leq/2, []),          []),
    ?assertEqual( qs(fun leq/2, [1]),         [1]),
    ?assertEqual( qs(fun leq/2, [7,1]),       [1,7]),
    ?assertEqual( qs(fun leq/2, [7,8,1]),     [1,7,8]).
```

```
quicksort:test().
...*failed*
in call from quicksort:quicksort_test/0
  (quicksort.erl, line 22)
**error:{assertEqual,[{module,quicksort},
          {line,22},
          {expression,"[ 1 , 7 ]"},
```

Unit Testing in Erlang

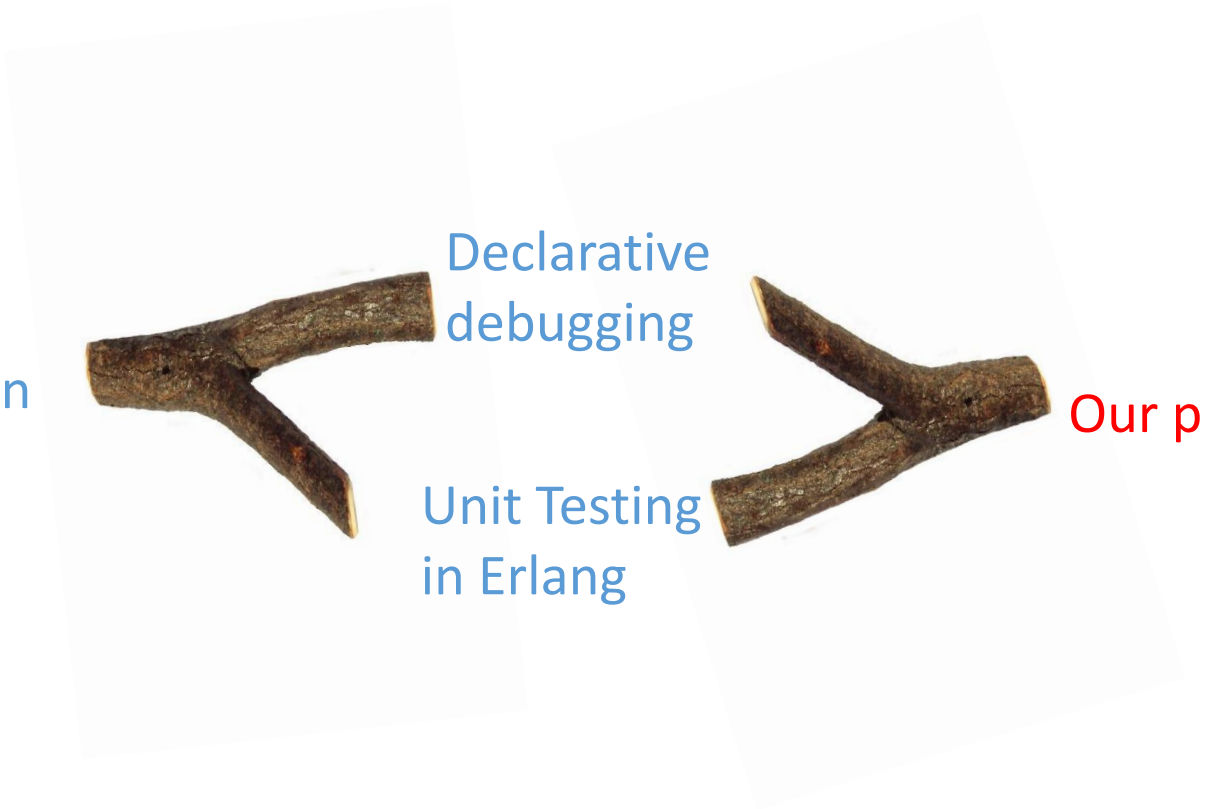
Erlang EUnit example

```
-include_lib("eunit/include/eunit.hrl").

quicksort_test() ->
    ?assertEqual( qs(fun leq/2, []),          []),
    ?assertEqual( qs(fun leq/2, [1]),         [1]),
    ?assertEqual( qs(fun leq/2, [7,1]),       [1,7]),
    ?assertEqual( qs(fun leq/2, [7,8,1]),     [1,7,8]).
```

```
quicksort:test().
...*failed*
in call from quicksort:quicksort_test/0
  (quicksort.erl, line 22)
**error:{assertEqual,[{module,quicksort},
          {line,22},
          {expression,"[ 1 , 7 ]"},
```

Outline

Motivation



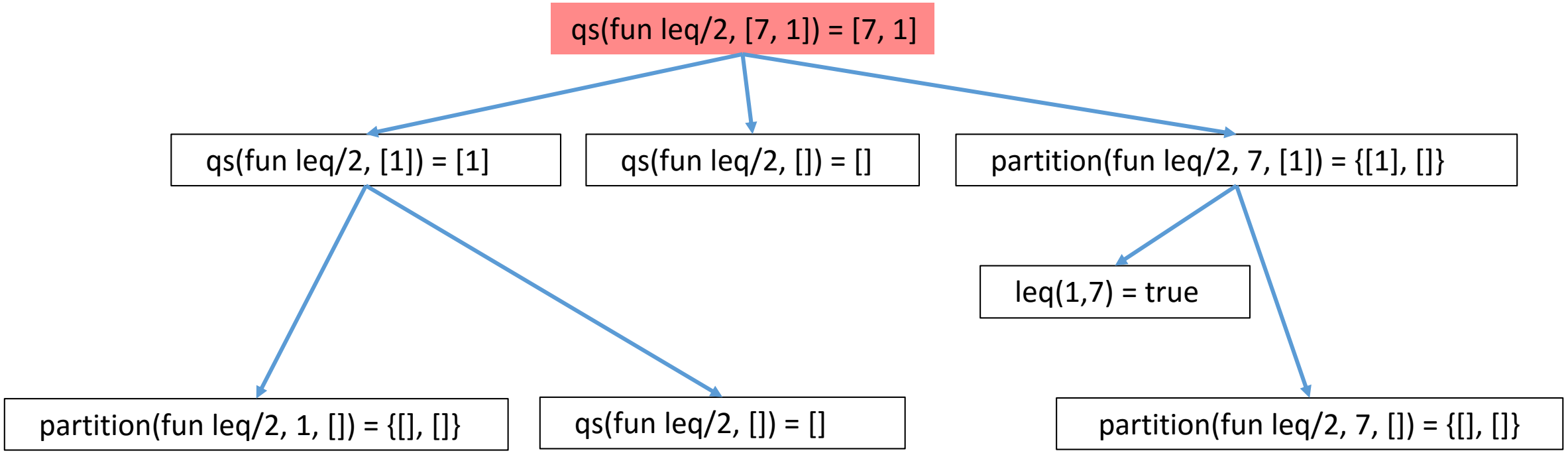
Declarative debugging

Unit Testing in Erlang

Our proposal

Debugging meets testing

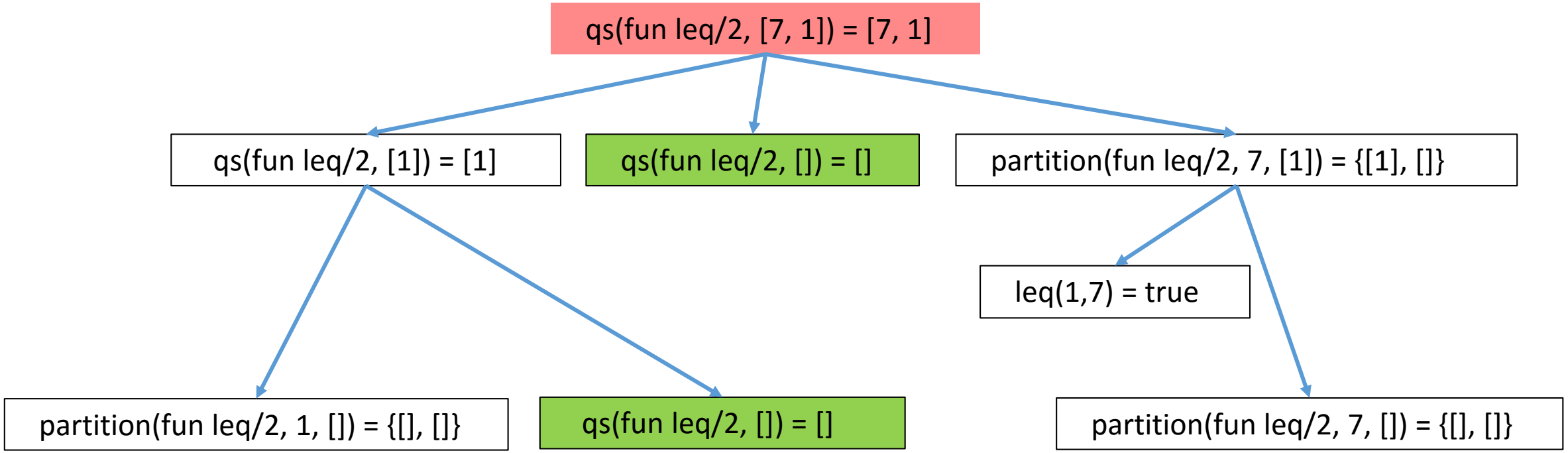
```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,1] )").
```



Debugging meets testing

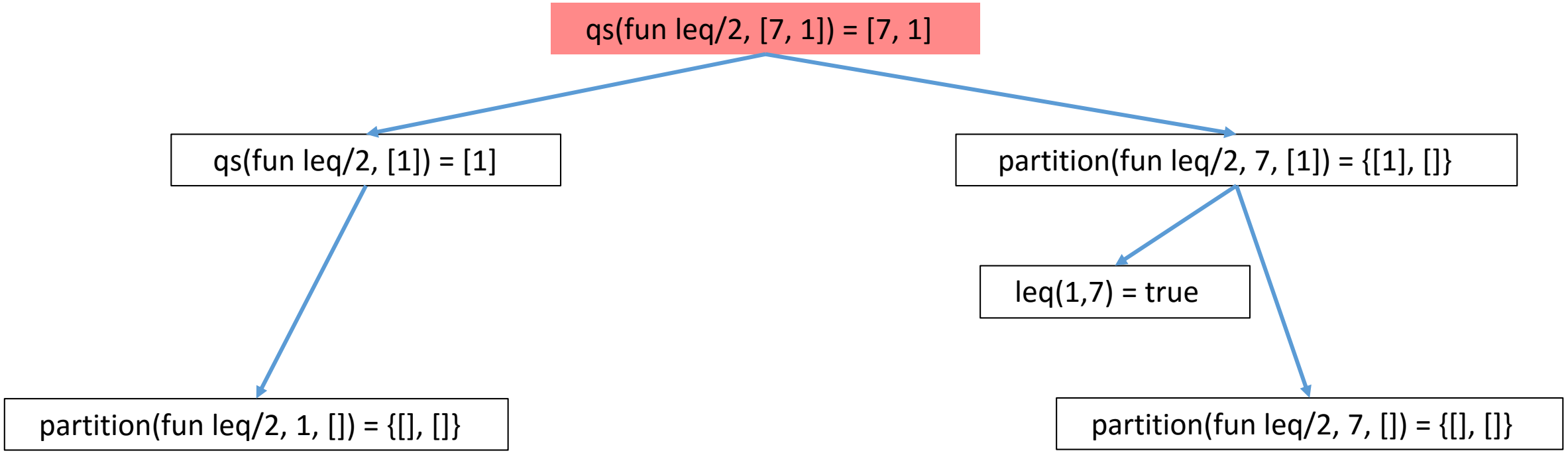
```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,1] )").
```

```
?assertEqual( qs(fun leq/2, []), [])
```



Debugging meets testing

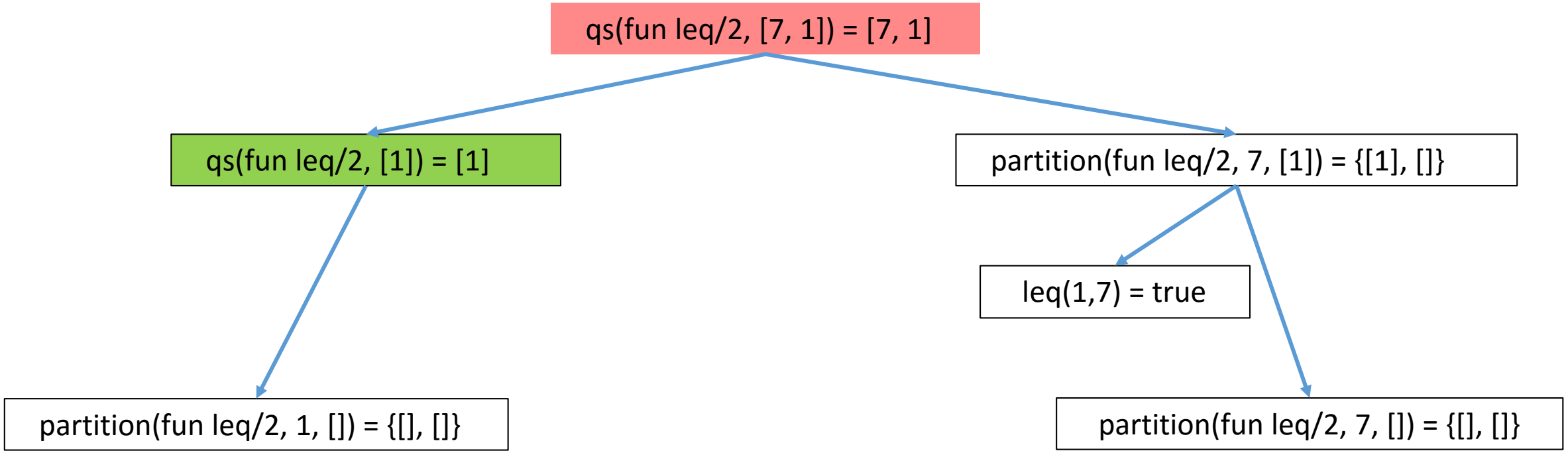
```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,1] )").
```



Debugging meets testing

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,1] )").
```

```
?assertEqual( qs(fun leq/2, [1]), [1])
```



Debugging meets testing

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,1] )").
```

```
?assertEqual( qs(fun leq/2, [7,1]), [1,7]),
```

8 .- qs(fun leq/2, [7, 1]) = [7, 1]

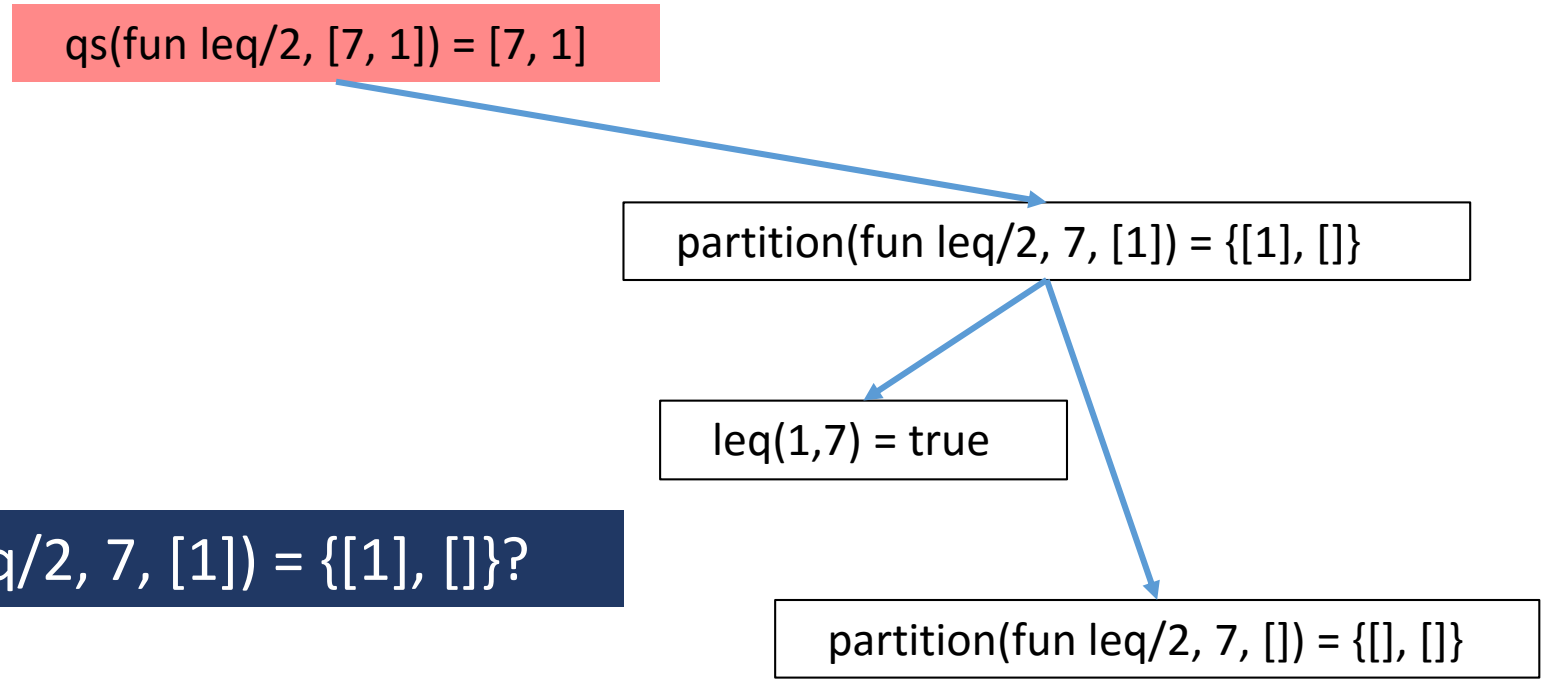
2 .- partition(fun leq/2, 7, [1]) = {[1], []}

1 .- leq(1,7) = true

0 .- partition(fun leq/2, 7, []) = {[], []}

Debugging meets testing

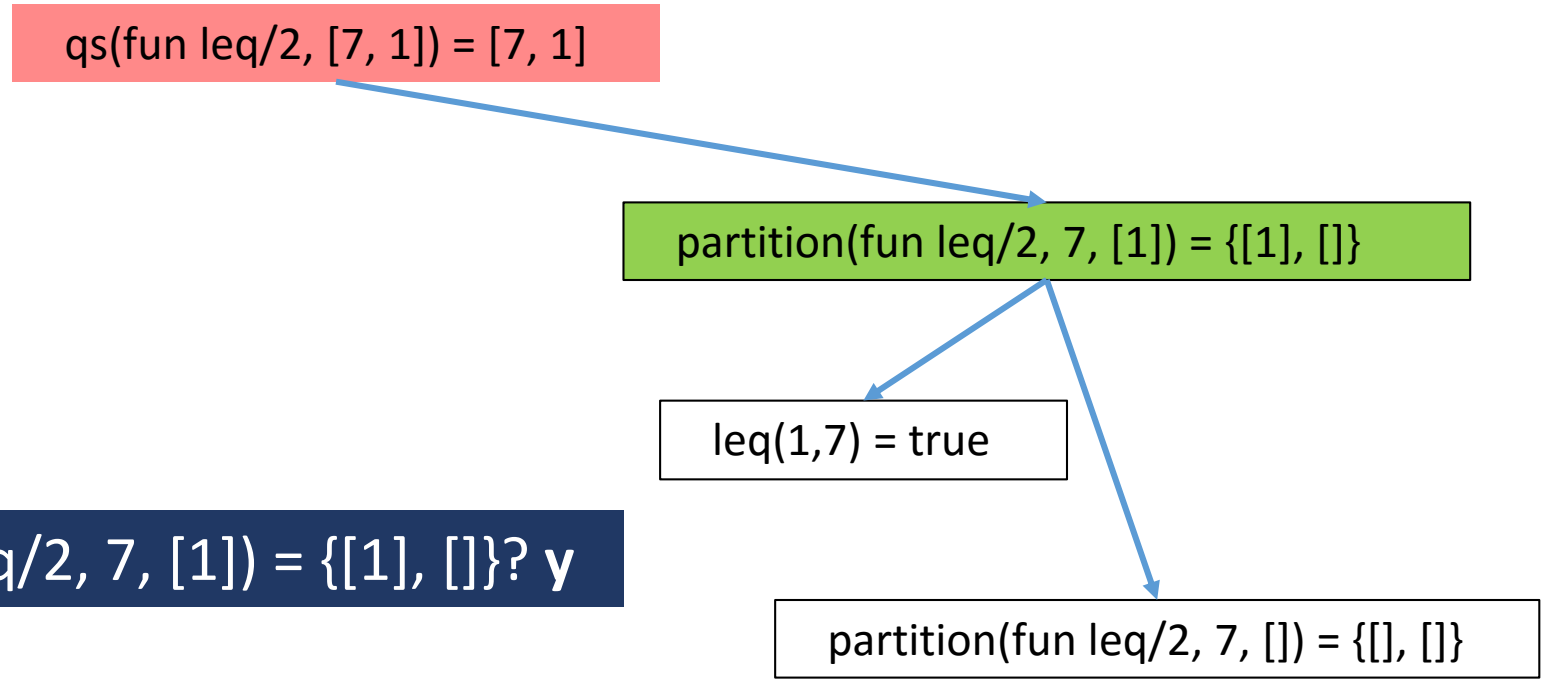
```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,1] )").
```




```
> partition(fun quicksort:leq/2, 7, [1]) = {[1], []}?
```

Debugging meets testing

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,1] )").
```




```
> partition(fun quicksort:leq/2, 7, [1]) = {[1], []}? y
```

Testing meets debugging

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,1] )").
```

qs(fun leq/2, [7, 1]) = [7, 1]

partition(fun leq/2, 7, [1]) = {[1], []}

```
> partition(fun quicksort:leq/2, 7, [1]) = {[1], []}? y
```

leq(1,7) = true

partition(fun leq/2, 7, []) = {[], []}

```
quicksort_test() ->
    ?assertEqual( qs(fun leq/2, []),          []),
    ?assertEqual( qs(fun leq/2, [1]),         [1]),
    ?assertEqual( qs(fun leq/2, [7,1]),       [1,7]),
    ?assertEqual( qs(fun leq/2, [7,8,1]),     [1,7,8])
    ?assertEqual(  partition(fun leq/2, 7, [1] ), {[1], [ ]} )
```

Debugging meets testing

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,1] )").
```

qs(fun leq/2, [7, 1]) = [7, 1]

```
error: quicksort:qs(fun quicksort:leq/2, [7, 1]) = [7, 1]
Please, revise the second clause:
qs(F, [E | R]) -> {A, B} = partition(F, E, R), qs(F, B) ++ [E] ++ qs(F, A).
```

Error found with one question

(at least 3 questions without tests)

Erlang example

```
-module(quicksort).
-export([qs/2, leq/2]).

qs(_, []) -> [];
qs(F, [E|R]) -> {A, B} = partition(F, E, R),  qs(F, A) ++ [E] ++ qs(F, B).

partition(_, _, []) -> {[], []};
partition(F, E, [H|T]) ->
        {A, B} = partition(F, E, T),
        case F(H, E) of
                true -> {[H|A], B};
                false -> {A, B}
        end.
leq(A, B) -> A =< B.
```

Testing meets debugging

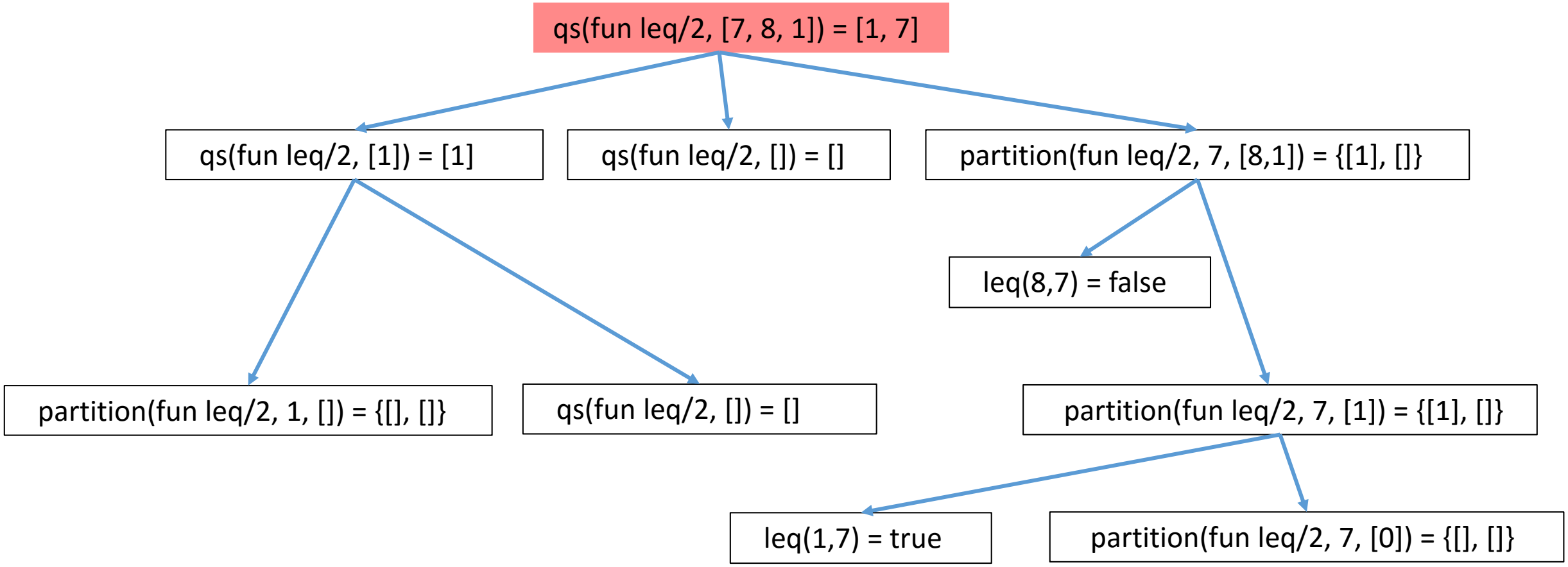
After correcting the error we try again the tests

```
quicksort:test().
...*failed*
in call from quicksort:quicksort_test/0
(quicksort.erl, line 23)
**error:{assertEqual,[{module,quicksort},
            {line,23},
            {expression,"[ 1 , 7 , 8 ]"},
            {expected,[1,7]},
            {value,[1,7,8]}]}
```

Another test case failing!!! → another error

Testing meets debugging

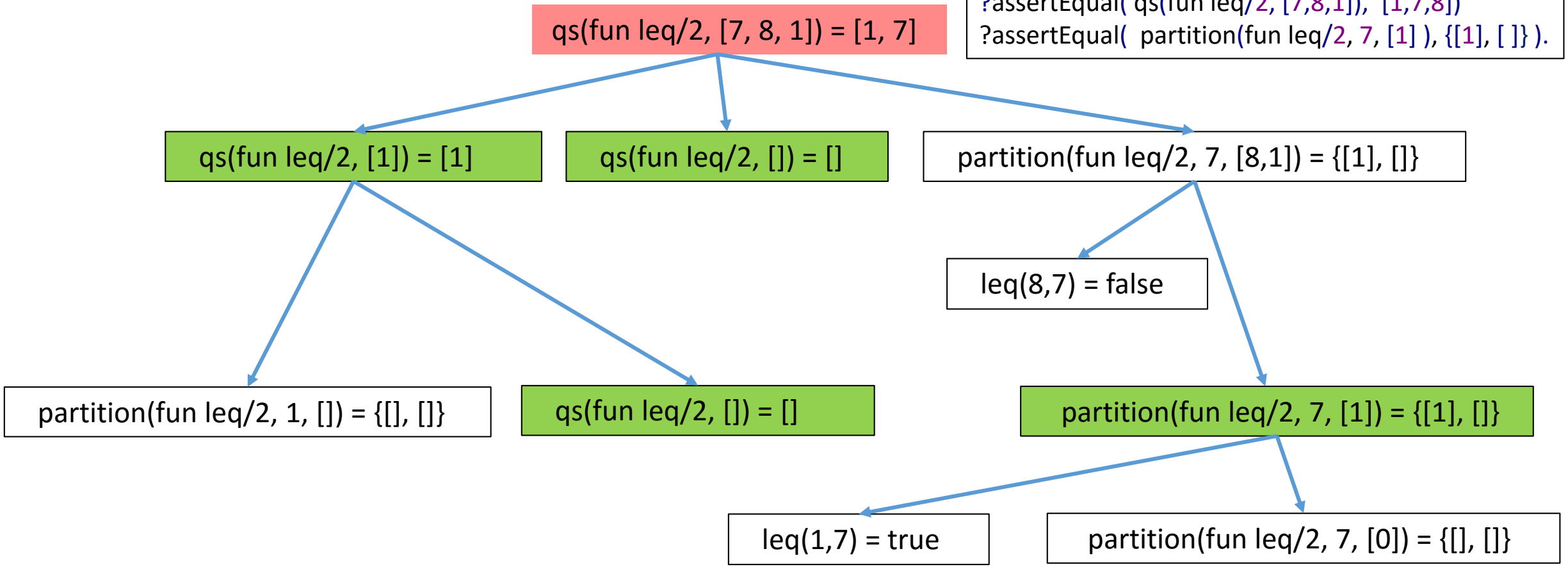
```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,8,1] )").
```



Testing meets debugging

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,8,1] )").
```

```
?assertEqual( qs(fun leq/2, []),          []),
?assertEqual( qs(fun leq/2, [1]),         [1]),
?assertEqual( qs(fun leq/2, [7,1]),       [1,7]),
?assertEqual( qs(fun leq/2, [7,8,1]),     [1,7,8])
?assertEqual(  partition(fun leq/2, 7, [1] ), {[1], [ ]} ).
```



Testing meets debugging

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,8,1] )").
```

qs(fun leq/2, [7, 8, 1]) = [1, 7]

Debugging session

```
partition(fun leq/2, 7, [8, 1]) = {[1], []}?
```

partition(fun leq/2, 7, [8,1]) = {[1], []}

leq(8,7) = false

Testing meets debugging

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,8,1] )").
```

qs(fun leq/2, [7, 8, 1]) = [1, 7]

Debugging session

```
partition(fun leq/2, 7, [8, 1]) = {[1], []}?   v
```

partition(fun leq/2, 7, [8,1]) = {[1], []}

leq(8,7) = false

Meaning: *"this is wrong, and I know the expected value"*

Testing meets debugging

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,8,1] )").
```

qs(fun leq/2, [7, 8, 1]) = [1, 7]

partition(fun leq/2, 7, [8,1]) = {[1], []}

leq(8,7) = false

partition(fun leq/2, 7, [8, 1]) = {[1], []}? v
What is the value you expected? {[1],[8]}

A new positive test case is generated!

Testing meets debugging

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,8,1] )").
```

qs(fun leq/2, [7, 8, 1]) = [1, 7]

partition(fun leq/2, 7, [8,1]) = {[1], []}

leq(8,7) = false

```
partition(fun leq/2, 7, [8, 1]) = {[1], []}? v
What is the value you expected? {[1],[8]}
leq(8, 7) = false? t
```

"I trust this function, mark all the questions about leq as valid"

Testing meets debugging

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,8,1] )").
```

qs(fun leq/2, [7, 8, 1]) = [1, 7]

partition(fun leq/2, 7, [8,1]) = {[1], []}

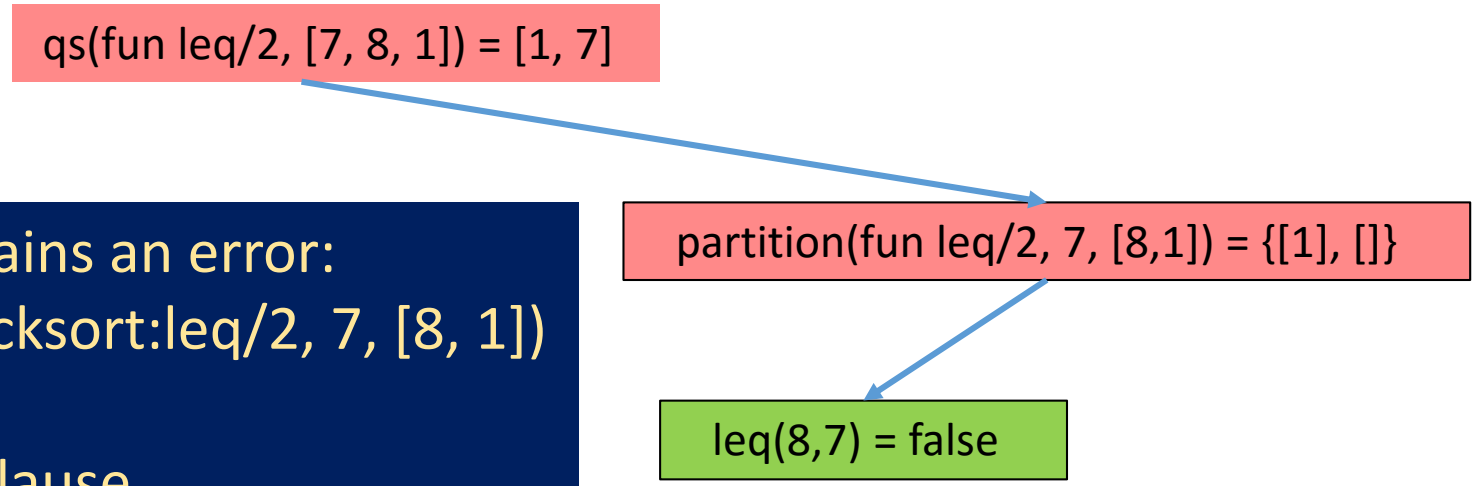
leq(8,7) = false

```
partition(fun leq/2, 7, [8, 1]) = {[1], []}? v
What is the value you expected? {[1],[8]}
leq(8, 7) = false? t
```

2 new assertions / test cases!

Testing meets debugging

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,8,1] )").
```

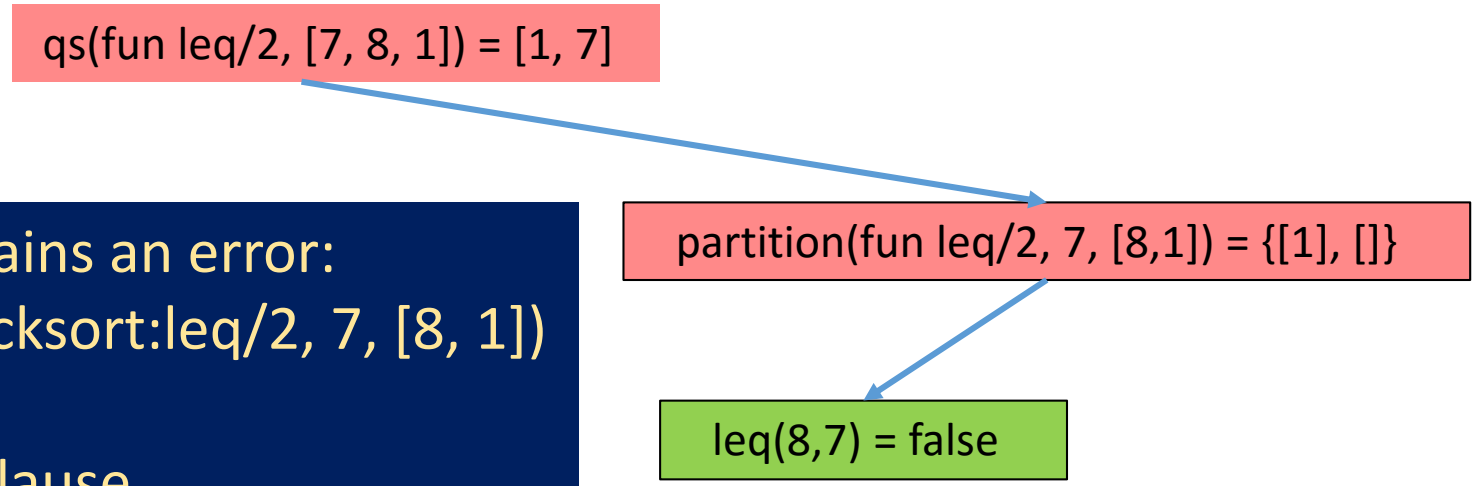


Call to a function that contains an error:
quicksort:partition(fun quicksort:leq/2, 7, [8, 1]) = {[1], []}
Please, revise the second clause

```
partition(F, E, [H | T]) ->
  {A, B} = partition(F, E, T),
  case F(H, E) of
    true -> {[H | A], B};
     false -> {A, B}
  end.
```

Testing meets debugging

```
> edd:dd("quicksort:qs( fun quicksort:leq/2, [7,8,1] )").
```



Call to a function that contains an error:
quicksort:partition(fun quicksort:leq/2, 7, [8, 1]) = {[1], []}
Please, revise the second clause

```
partition(F, E, [H | T]) ->
  {A, B} = partition(F, E, T),
  case F(H, E) of
    true -> {[H | A], B};
     false -> {A, [H | B]}
  end.
```

Testing meets debugging

After correcting the error we try again the tests

```
quicksort:test().
Test passed
```

No more bugs…at least in this talk (hopefully)

Pros and Cons

Pros

- ✓ Debugging becomes part of the software dev. life cycle
- ✓ No need of initial tests → they will be generated during debbuging

Cons

- ✓ Only “deterministic” functions (for instance no input operations)
- ✓ The user answer really matter → an erroneous answer becomes part of the test suite

Outline

Motivation

Declarative debugging

Unit Testing in Erlang

Our proposal

Conclusions

Conclusions

- ✓ Debugging: A lot of useful information thrown away
- ✓ Declarative debugging: store the information as Unit Tests
- ✓ Unit Testing: saves questions in declarative debugging
- ✓ General approach: presented for Erlang but can be seen as a general result

Thanks for your attention