A few improvements to Erlang

Joe Armstrong

EHE 0

```
<h1>hi<h1>

<?e Name="joe","" ?>

<p>Hello my name is <?e Name ?>
```

PHP

```
<h1>Factorial in PHP</h1>

<?php ini_set('display_errors', 'On'); ?>


<?php

for($i=0;$i<200;$i++){
  $j = fac($i);
  print "factorial($i) = $j <br>";
}


function fac($n) {
  if ($n == 0) return 1;
  return $n * fac($n - 1);
}

?>
```

Factorial in PHP

factorial(0) = 1
factorial(1) = 1
factorial(2) = 2
factorial(3) = 6
factorial(4) = 24
factorial(5) = 120
factorial(6) = 720
factorial(7) = 5040
factorial(8) = 40320
factorial(9) = 362880
factorial(10) = 3628800
factorial(11) = 39916800
factorial(12) = 479001600
factorial(13) = 6227020800
factorial(14) = 87178291200
factorial(15) = 1.307674368E+12
factorial(16) = 2.0922789888E+13
...
factorial(170) = 7.25741561531E+306
factorial(171) = INF
factorial(172) = INF

EHE 0

```
<h1>Factorial 1<h1>

<?e Fac = fun(N) ->
    F=fun(F,0)->1;(F,N)->N*F(F,N-1) end,F(F,N) end,
    “”
?>

<p>Factorial 50 is <?e Fac(50)?>
```

```
Factorial 1

Factorial 50 is
30414093201713378043612608166064768844377641568960512000000000000
```

Uuugh

```
1> F=fun(F, 0) -> 1;(F, N) -> N*F(F, N-1) end.
#Fun<erl_eval.12.111823515>
2> F(F, 5).
120
3> Fac = fun(N) -> F(F,N) end.
#Fun<erl_eval.6.111823515>
4> Fac(4).
24

5> Fac2 = fun(N) ->
     F = fun(F,0)->1;(F,N)->N*F(F,N-1) end,
     F(F,N) end.
#Fun<erl_eval.6.111823515>
6> Fac2(4).
24
```

This sucks so we'll generalize it ...

Generalize...

```
<?e

  Y = fun(Func) ->
        F = fun (X) -> Func(fun(A) -> (X(X))(A) end) end,
        F(F)
      end,

  Fac = fun(F) ->
          fun (0) -> 1;
              (N) -> N * F(N-1)
          end
        end,
   ""
?>

<p> Fac 50 is <?e (Y(Fac))(50) ?>
```

But ... that's the ... the ... the ... ...

The Y combinator

There has
to be a better
way ...

Change erl_eval

```
<h1>Factorial2<h1>

<?e def(fac, 1, fun(0) -> 1; (N) -> N * fac(N-1) end),"" ?>

<p>Factorial 50 is <?e fac(50)?>
```

```
Factorial 1

Factorial 50 is
30414093201713378043612608166064768844377641568960512000000000000
```

But that's a hack
there has to be
an even
better way

This is a symptom of a deeper problem....

What's wrong today?

```
-module(foo).
-export(...).

X=42.

fac(0) -> 1;
fac(N) -> N*fac(N-1).
```

In a module
expressions are illegal

```
$ erl
-module(foo).
-export(...).

X=42.

fac(0) -> 1;
fac(N) -> N*fac(N-1).
```

In the shell
forms are illegal

What's wrong with the shell?

```
$erl
1> X = 10.
10
2> Y = quote X + 12.
{var,'Y',{expr,{op,'+'},{var,'X'},{int,12}}}
3> eval(Y).
22
4> define fac = fun(0) -> 1;(N)->N*fac(N-1)
end.
{def,fac,{...}}
5> fac(5).
120
```

What's wrong with Erlang?

- A module is a sequence of **forms**
- An escript is a sequence of **forms**
- The Erlang REPL (the shell) evaluates a sequence of **expressions**.
- What should we do about this?

Redefine forms as expressions with side effects

Define this to be an expression with value parse("-module(foo).")

Define this to be an expression with value parse(...)

Define this to be an expression with value parse(...)

```
$ erl

-module(foo).

-export(...).

X=42.

fac(0) -> 1;
fac(N) -> N*fac(N-1).
```

What is Erlang II?

An erlang generator - running the command `erl2 File.erl2` creates one or more erlang modules.

Don't mess with Erlang. Write a program that writes programs.

erl2 programs create erl1 programs

erl3 programs create erl2 programs

Metaprograming!

Why this approach?

- Long time to fix all the erlang 1 tools emacs/eclipse/dialyzer/quickcheck/typer/ rebar ...
- All the old tools will still work
- Difficult to reach agreement on Erlang 1 changes - need to balance lots of different interests

Ideas from

- Partial evaluation
- Literate programming
- Unit testing
- Program synthesis
- Refactoring
- LISP
- Git

All of these are added in a very simple form

So for example

```
120 = fac(5)
```

is the only form of unit test - ie no unit test frameworks

Goals

- Get rid of bad stuff (Macros, records)
- Add good stuff (partial evaluation, structs)
- Without breaking the old stuff
- Have fun

Compile time optimization

```
-module(example7).
-export([test/1]).
-local([fac/1]).

fac(0) -> 1;
fac(N) -> N*fac(N-1).

120 = fac(5).

Fac20 = fac(20).

test(X) ->
    X*Fac20.
```

example7.erl2

```
$./erl2 example7.erl2
Created:example7.erl
```

This is a local function - don't export it

```
-module(example7).
-export([test/1]).

test(X) ->
   Fac20 = 2432902008176640000,
   X * Fac20.
```

example7.erl

Unit tests

```
-module(example8).
-export([test/1]).
-local([fac/1]).

-spec fac(int()) -> int().

fac(0) -> 1;
fac(N) -> N+fac(N-1).

24 = fac(4).

Fac20 = fac(20).

test(X) ->
    X*Fac20.
```

example8.erl2

```
$./erl2 example8.erl2
Created:"gen/exprs.tmp"
Error:{{badmatch,24},
[{erl_eval,expr,11,[]}]}
Compile failed
```

This is buggy - though type correct

When we process `example8.erl2` the unit test fails so no code is generated.

The code is not compiled if it fails the unit tests.

Letrec

```
beginFunc f1/1 end.
  f1(X) -> foo(X).
  foo(X) -> {foo1, X}.
endFunc.

beginFunc f2/1 end.
  f2(X) -> foo(X).
  foo(X) -> {foo2, X}.
endFunc.
```

foo/1 is a local to f1/1

```
f1(X) ->
    gen_5_foo(X).

f2(X) ->
    gen_14_foo(X).

gen_5_foo(X) ->
    {foo1,X}.

gen_14_foo(X) ->
    {foo2,X}.
```

Letrec

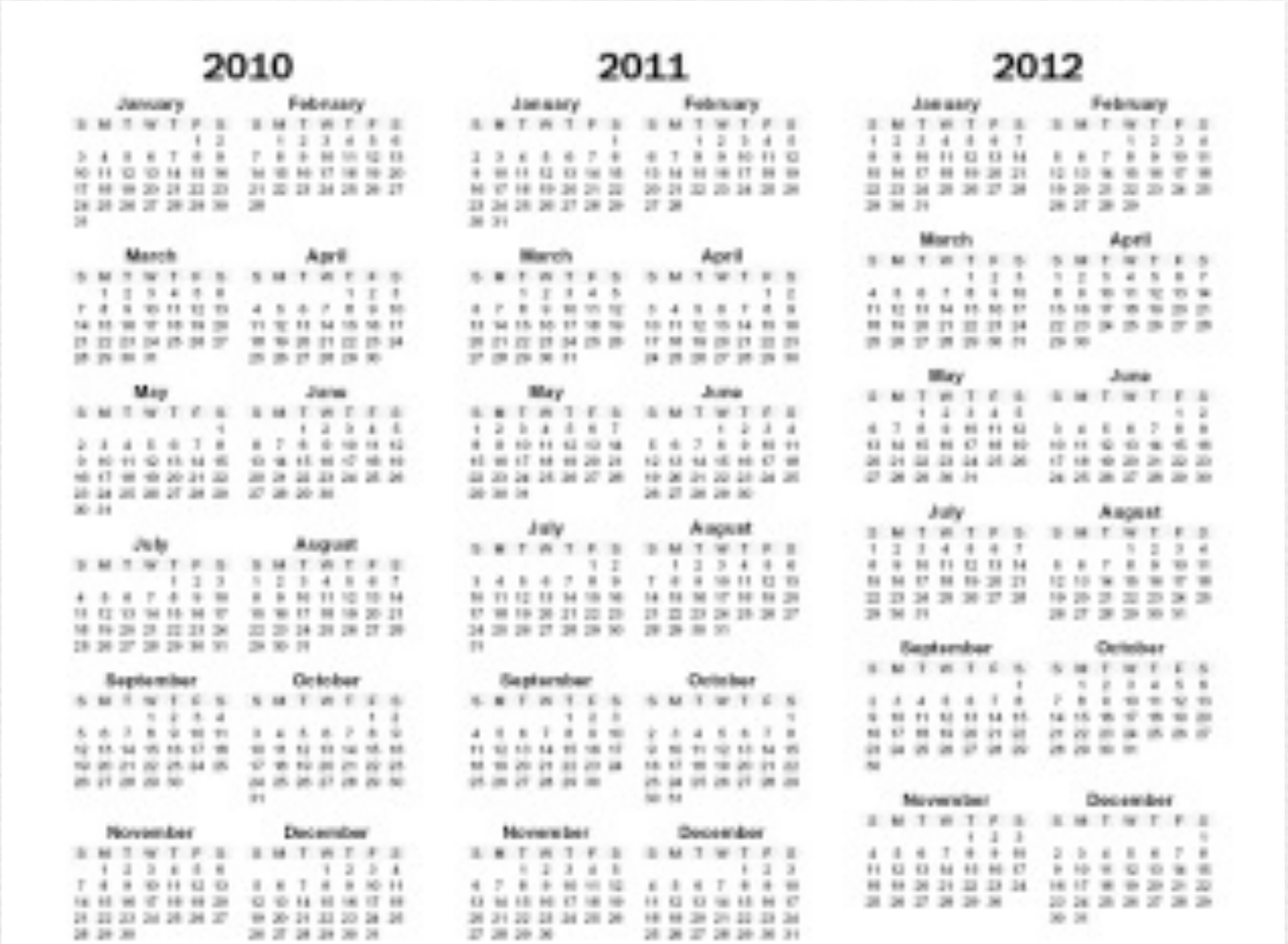
foo/1 is a local

this:foo/1 is in the outer scope

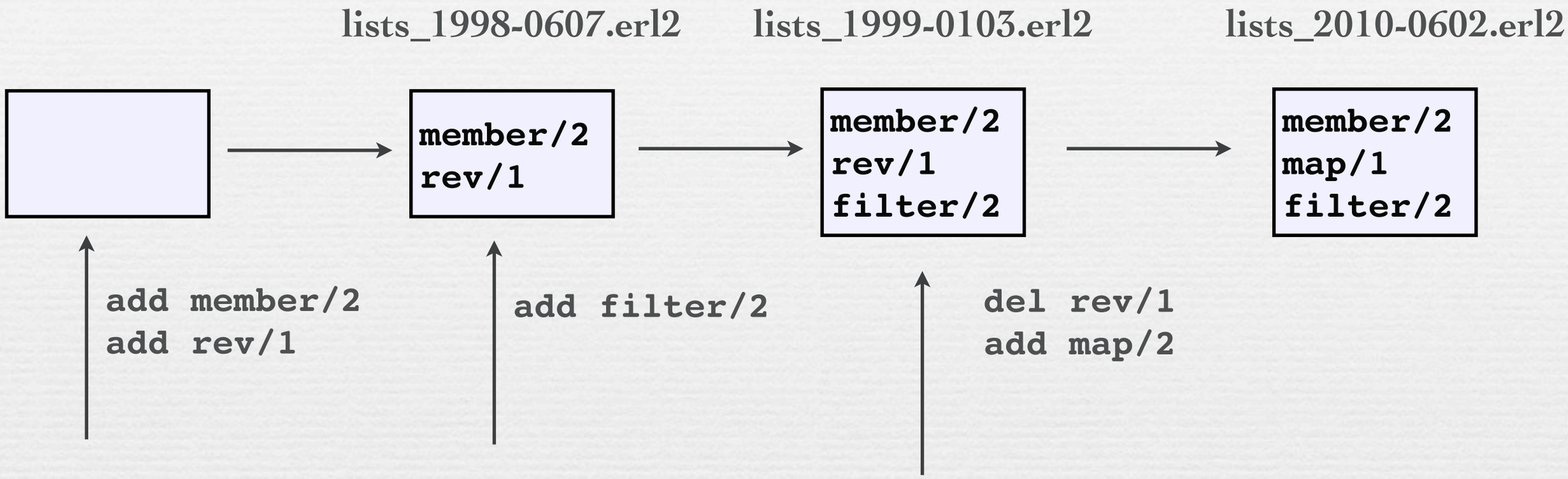
```
beginFunc f3/1 f4/1 end.
   f3(X) -> foo(this:f1(X)),this:foo(X).
   f4(X) -> foo(example9:f2(X)).
   foo(X) -> {foo3, X}.
endFunc.
```

```
f3(X) ->
    gen_23_foo(f1(X)),
    foo(X).

f4(X) ->
    gen_23_foo(example9:f2(X)).

gen_23_foo(X) ->
    {foo3,X}.
```

Modules change with time

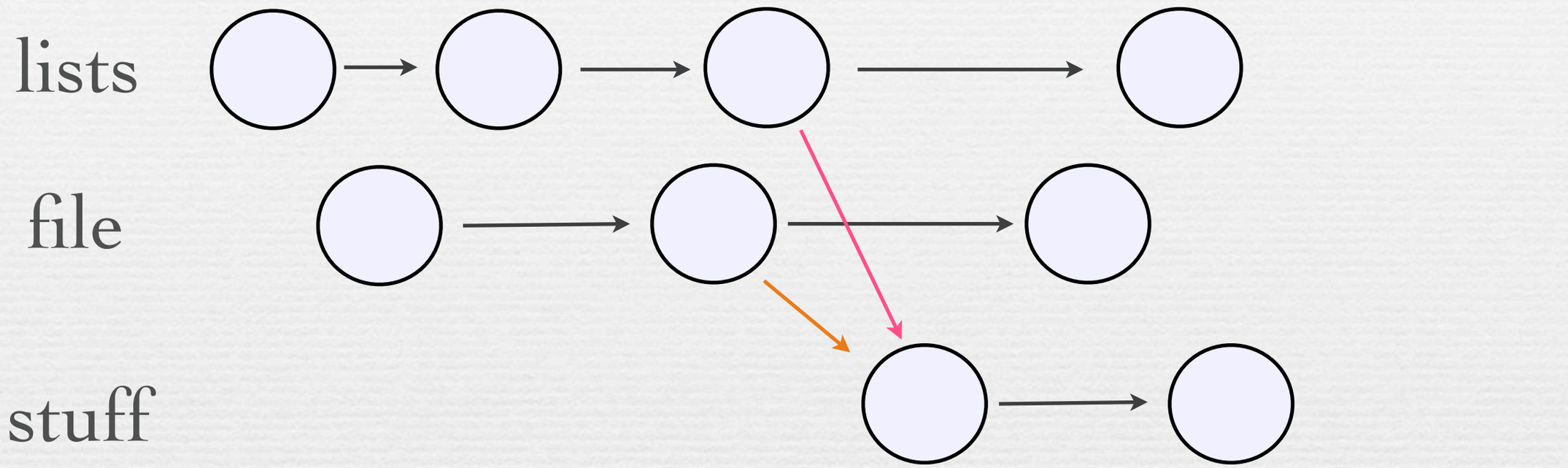


lists.erl changes with time



The module lists is a K-V database whose content changes with time

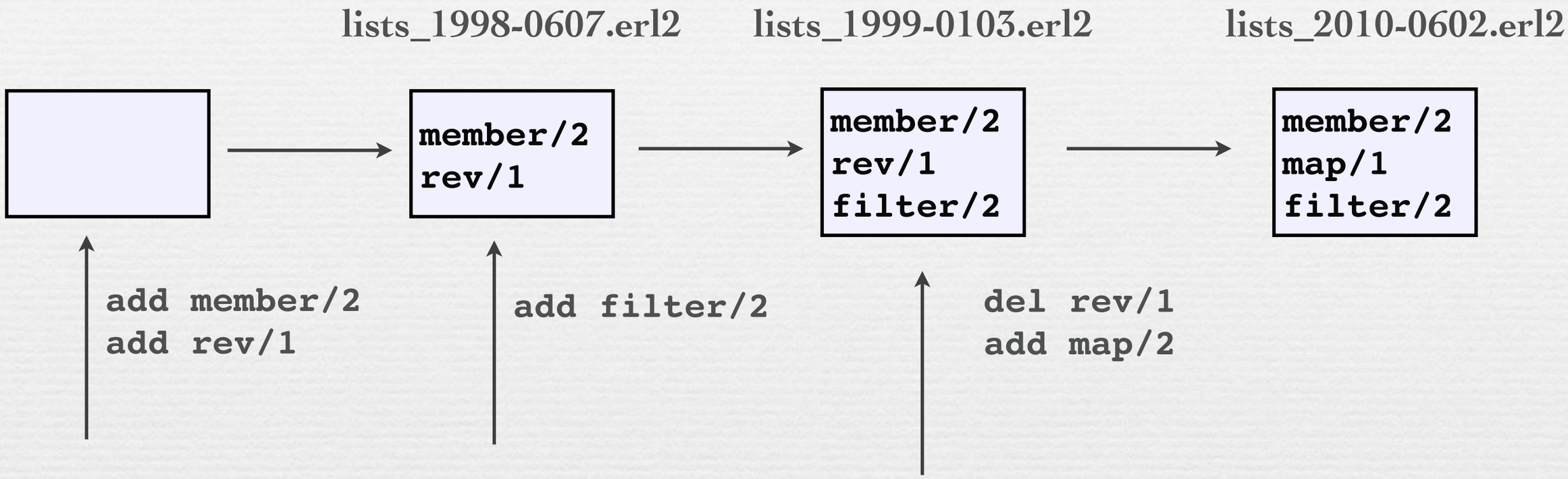
System evolution



Time

Describe the Deltas and why

lists.erl changes with time



The module lists is a K-V database whose content changes with time

```
-module(lists, "19860612").

reverse(X) -> reverse(X, []).

reverse([], L)    -> L;
reverse([H|T], L) -> reverse(T, [H|L]).
```

```
-module(lists, "19870212").
-insert(lists, "19860612").

map([], _) -> [];
map([H|T], F) -> [F(H)|map(T, F)].
```

```
-module(lists, "19880417").
-insert(lists, "19870212").

sort(...) -> ...
```

lists_19860612.erl2

Old version

lists_19860612.erl2

Delta

lists_19880417.erl2

```
-module(lists, "19920612").
-insert(lists, "19880417").

remove sort/1.

sort(...) -> ...
```

lists_19920612.erl2

```
-module(stuff, "19920612").
-insert(lists, "19880417").
-insert(file, ...).

blaa(...) -> ...
```

Patch files express *intentionality* binary diffs and checksummed blobs (git etc.) do not.

Knuth was right.
Source+List of patches
is easier to understand that
a sequences of source files.

Three operators

- add Func/Arity
- delete Func/Arity
- clone Mod

The bigger picture

- Code is part of a bigger picture

Real-world Haskell

Our function's type signature indicates that it accepts a single string, the contents of a file with some unknown line ending convention. It returns a list of strings, representing each line from the file. 2 comments

```
-- file: ch04/SplitLines.hs
splitLines [] = []
splitLines cs =
    let (pre, suf) = break isLineTerminator cs
    in  pre : case suf of
                ('\r':'\n':rest) -> splitLines rest
                ('\r':rest)      -> splitLines rest
                ('\n':rest)      -> splitLines rest
                _                -> []

isLineTerminator c = c == '\r' || c == '\n'
```

27 comments

Our function's type signature indicates that it accepts a single string, the contents of a file with some unknown line ending convention. It returns a list of strings, representing each line from the file. 2 comments

Farkface 2008-12-31 8926

Should be "line-ending convention."

Tom Mitchell 2009-09-28 11002

Not "unknown" but one of the two most common possible line-ending conventions. Other conventions are possible and given the state of string handling related bugs other conventions make sense to explore and use.

This takes us back to test cases and "strict type checking" in once sense a "line" is a type of data. Show how Haskel can validate input by dealing with varied line-ending conventions and manage this common data type in a strict and reliable way.

Or leave it as a problem for the student.

Comment
[help]

Your name **Required** so we can give you credit

Your URL **Optional** link to blog, home page, *etc.*

Remember you? ☑

Submit Comment

Writing a program

- We do loads of research
- We write the code
- We throw away the research
- The research is >> the program
- Nobody can understand or reproduce what we have done later

Solution

- An environment that stores the code and together with the research

How?

- In browser environment
- Integrated commenting/transformation system
- Integrated chat
- Integrated data store
- Websockets, HTML5, Erlang

Work in progress ...

https://github.com/joearms/erl2.git

QUESTIONS

Additional material

Literate programming

```
addMod a.

f1() -> ...

addMod b.
...

addMod c.
...

addMod a
...
```

```
Text ...

a:f1() -> ...
Text ...

b:f2() -> ...

c:foo() -> ...

a:bar() -> ...
```

Batch refactoring

```
-module(x).
foo(X) -> ...
bar(A,B) -> ...
```

x.erl2

```
-module(y).
-include("x.erl2").
clone x as z.
delete z:bar/2.
add z:bar = fun(..
...
```