```
data Identity a = Id a

runIdentity :: Identity a -> a
runIdentity (Id x) = x

instance Monad Identity where          (non-strict Monad)
  return :: a -> Identity a
  return x = Id x
  (>>=) :: Identity a -> (a -> Identity b) -> Identity b
  ix >>= f = f (runIdentity ix)
                ↖ postpone unpacking (lazy)
```

---

```
data Eval a = Done a

runEval :: Eval a -> a
runEval (Done x) = x

instance Monad Eval where              (strict Monad)
  return :: a -> Eval a
  return x = Eval x
  (>>=) :: Eval a -> (a -> Eval b) -> Eval b
  (Done x) >>= f = f x
    ↑
  unpacking right away and cache the value  (eager)
```

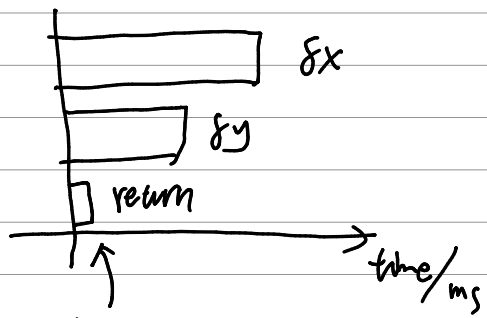
# additional methods for Eval

```
rpar :: a -> Eval a
```
(run in parallel, can be "stealed" by other processors)

```
rseq :: a -> Eval a
```
(before 'return', evaluate a, on the same processor, preserving sequential order)

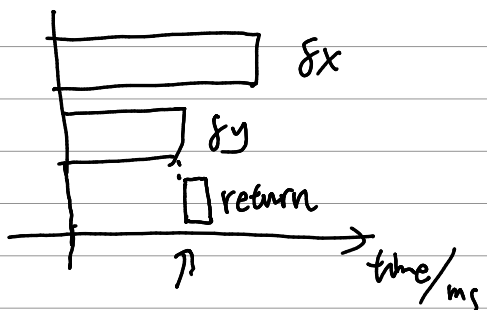
> example

① 
```
runEval $
  do
    x' <- rpar (f x)
    y' <- rpar (f y)
    return (x', y')
```




"return" immediately, non-blocking

② 
```
runEval $
  do
    x' <- rpar (f x)
    y' <- rseq (f y)
    return (x, y)
```




wait until rseq evaluation finishes, "blocking"

③ runEval $
do
x' ← rpar (f x)
y ← rpar (g y)
rseq x'
rseq y'
return (x', y)

← wait until both computation done, before 'return'

---

Similar to Async/Await Syntax in JS.

Any computation wrapped in Promise will be executed right away.

Async function () {
var X = ReadUserData() ← "rpar"

"rseq" → Await X
return X
}

> example : Sudoku Solver (sequential, not in parallel)

Input  solve :: String → Maybe String

main :: IO ()  (not all Sudoku puzzles have solutions)
main = do

IO [String]  [f] ← getArgs

(for simplicity, pattern matching on single element list, * partial function)

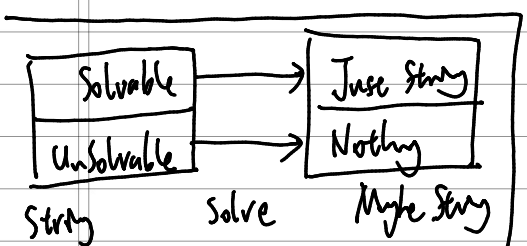
IO String  file ← readFile f

let puzzles = lines file
(puzzles :: [String], lines: parser)

solutions = fmap solve puzzles
(solutions :: [Maybe String])

IO ()  print $ length $ filter isJust solutions

forcing divergence/evaluation, so the puzzles have to be solved by this point




* doesn't validate whether the String is actually the encoding of a legal sudoku puzzle;

⇒ 'unsolvable' encapsulates "random" strings.