Common Pitfalls of Functional Programming and How to Avoid Them: A Mobile Gaming Platform Case Study

Sep 22, 2013

GREE, Inc.

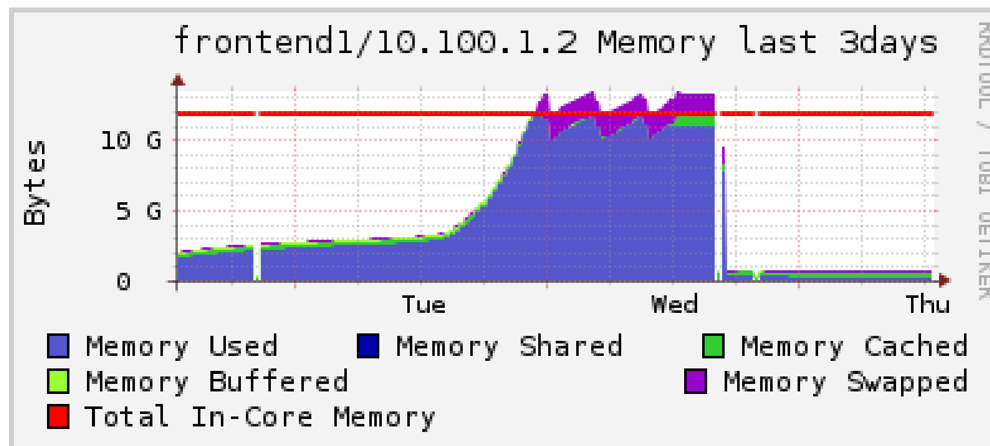
Yasuaki Takebe

Copyright © GREE, Inc. All Rights Reserved.

Introduction

GREE

- Why Functional Programming?
  - Reliability: Eliminate runtime type error, implicit state, ...
  - High performance: 20-60 times faster than Perl, PHP, Ruby, ...
  - Productivity: Powerful and elegant, a vast number of libraries, ...
- However...
  - Memory leak



  - Data lost
  - Performance degradation
  - Crash

Copyright © GREE, Inc. All Rights Reserved.

Contents

GREE

- About ourselves
  - What we developed using functional programming
- Examples of pitfalls
- How to avoid them
  - Testing tool
  - Documentation
  - Technical review
  - Education

Copyright © GREE, Inc. All Rights Reserved.

About GREE (1/3)

- Overview of GREE's services
  - One of the largest mobile game platforms
  - 37.2M users, 2000 games (as of Jun. 2013)
- Business
  - Social games
  - Platform: SNS, 3rd party games
  - Social media: mail magazine, news
  - Advertising and ad network
  - Licensing and merchandising
  - Venture capital

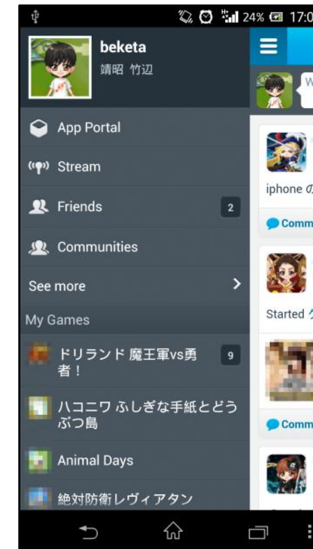
Copyright © GREE, Inc. All Rights Reserved.

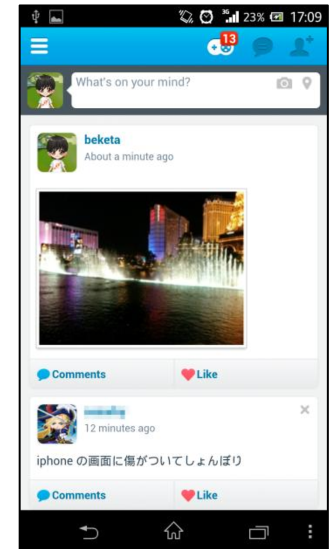
About GREE (2/3)

- Example of GREE's products / services

- Social games
  - Modern War: War simulation game
  - Miniature Garden: Wonder Mail and Animal Island: Gardening game
  - ...

- SNS app
  Features
  - Game portal
  - See what friends are playing
  - Share updates, photos and videos
  - Notification from friends when they like your posts





Copyright © GREE, Inc. All Rights Reserved.

About GREE (3/3)

GREE

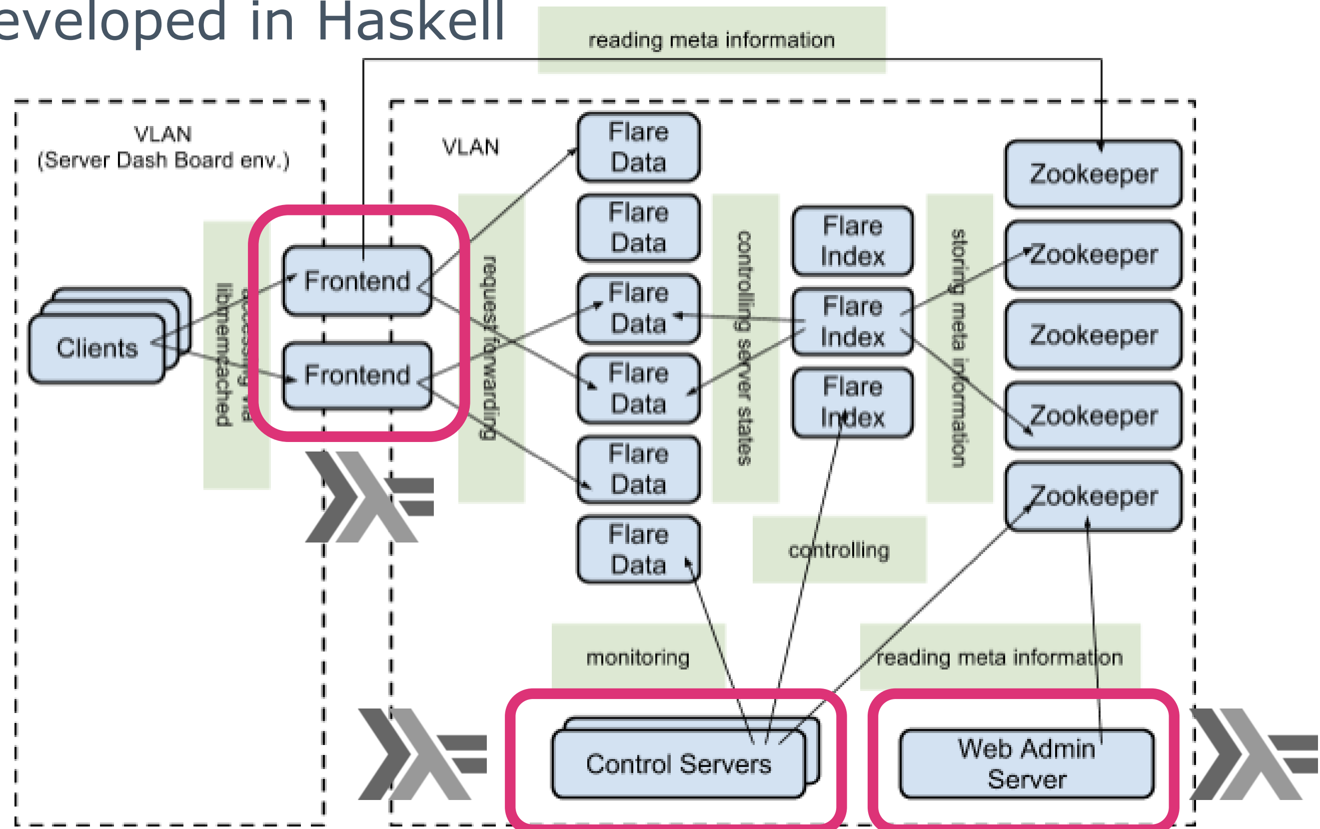
- Company
  - Founded: Dec 7, 2004
  - Employees: 2582 (group, as of Jun. 2013)
- Common architecture
  - Client: Java, Objective-C, JavaScript, Unity/C#, ...
  - Server: PHP, MySQL, Flare (KVS), ...
    - Develop middleware for ourselves
- Functional programming
  - Started: Jun, 2012 (a Haskell project)
  - Engineers: Haskell: 4, Scala: 6

Copyright © GREE, Inc. All Rights Reserved.

What We Developed Using FP (1/2)

GREE

- KVS management system
  - Setup / destroy KVS nodes in response to hardware fault / sudden access spike
  - Used in a social game app
- Components developed in Haskell
  - Frontend
  - Control server
  - Web admin server

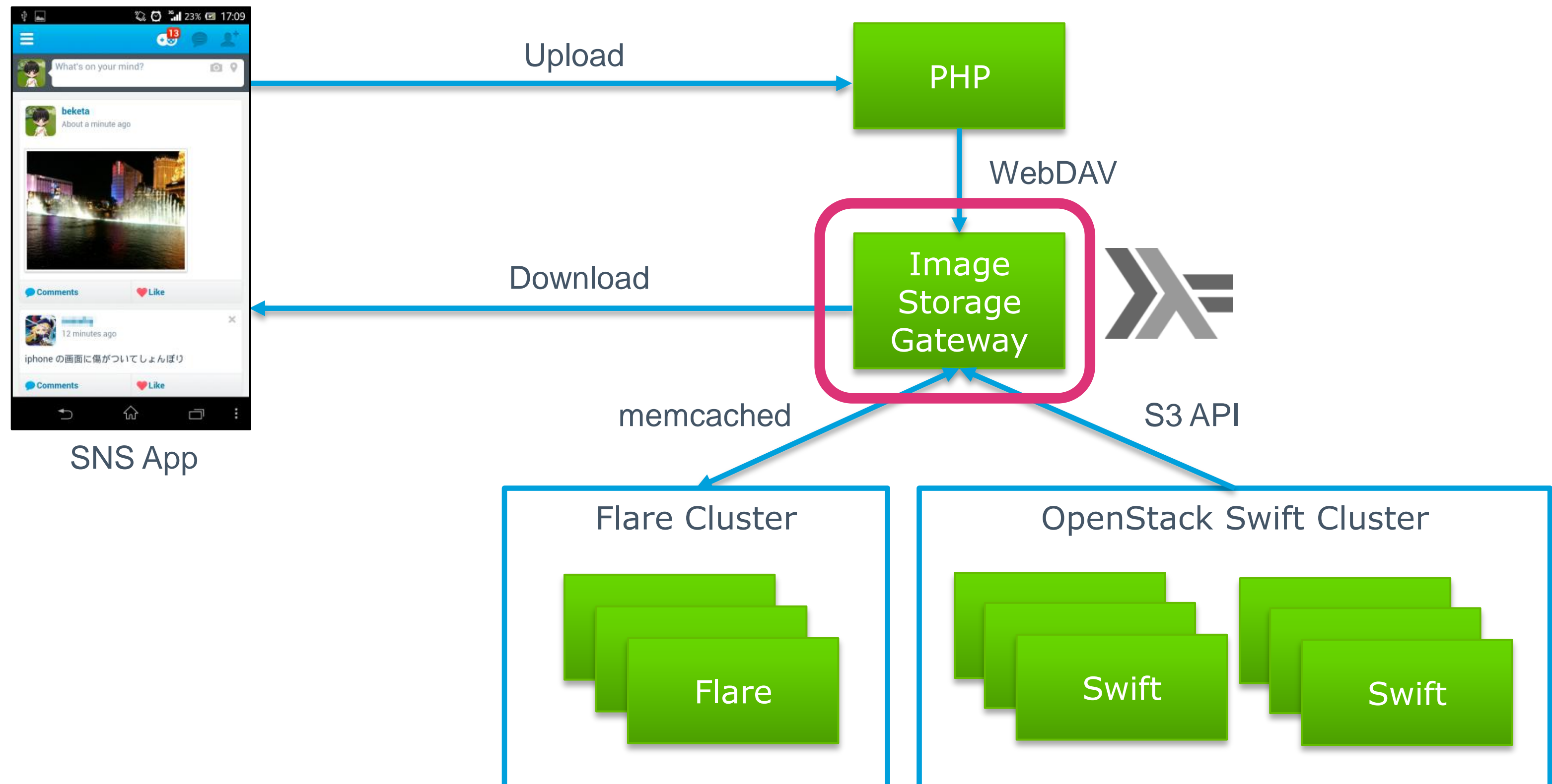


Copyright © GREE, Inc. All Rights Reserved.

What We Developed Using FP (2/2)

GREE

- Image storage gateway
  - Convert WebDAV to memcached / S3 API
  - Used in SNS photo uploader
  - Developed using Warp and http-conduit



Copyright © GREE, Inc. All Rights Reserved.

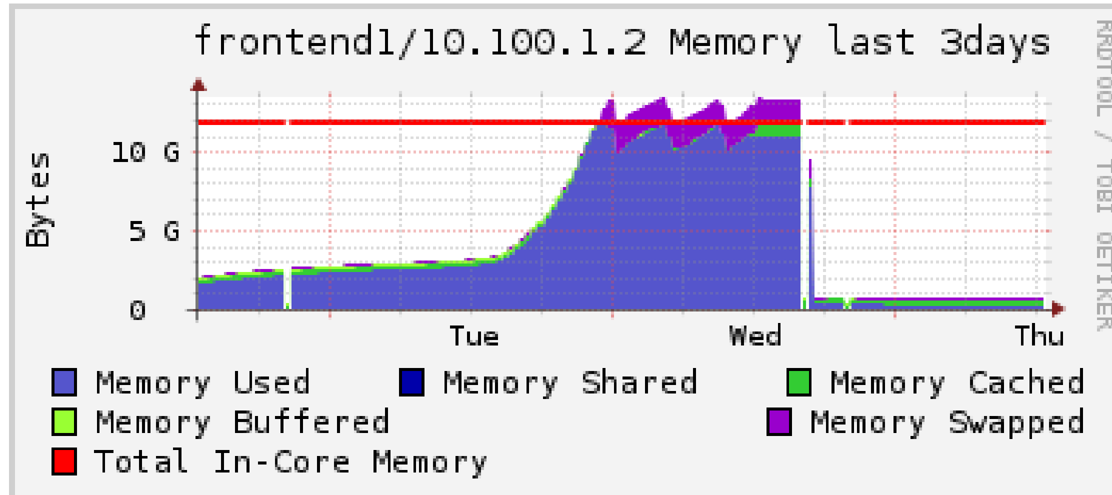
Examples of Pitfalls

Copyright © GREE, Inc. All Rights Reserved.

Pitfall 1: Leak by Lazy Evaluation

GREE

- Issue: Memory leak



- Cause
  - Frontend server keeps a list of active thread IDs in TVar for monitoring
  - Delete from thread ID list

```
modifyTVar' requestThreads $ \threads -> filter (tid /=) threads
```

  - But this reduces thread ID list only to WHNF

Copyright © GREE, Inc. All Rights Reserved.

Pitfall 1: Leak by Lazy Evaluation (Cont.)

- How to fix
  - Evaluate to normal form (or evaluate `filters` in this case)
  - In this case we fixed by evaluating length of threads as follows:

```
modifyTVar requestThreads $ \threads ->
  let thread' = filter (tid /=) threads
  in seq (length threads') threads'
```

- Pitfall
  - It is easy to mix up write to TVar / MVar with other IO operations, which evaluate value to normal form
  - Easy to mix up `moditYTVar'`, strict version of `modifyTVar`, with other IO operations which evaluate the value to normal form

Copyright © GREE, Inc. All Rights Reserved.

Pitfall 2: Race Condition

- Issue: Data put in a queue (very rarely) lost
- Cause
  - Queue is implemented using TQueue, which has two TVars of list
  - Dequeue from TQueue is wrapped by timeout, as `readTQueue` blocks forever when no item in queue
  - Definition of `timeout`

```
timeout n f = do
  pid <- myThreadId
  ex  <- fmap Timeout newUnique
  handleJust (\e -> if e == ex then Just () else Nothing)
             (\_ -> return Nothing)
             (bracket (forkIO (threadDelay n >> throwTo pid ex))
                      (killThread)
                      (\_ -> fmap Just f))
```

  - `timeout` invokes another thread which wait n microseconds and an exception to throws current thread
  - Exception might be thrown when evaluation of f (IO action wrapped by timeout) just finished

Copyright © GREE, Inc. All Rights Reserved.

Pitfall 2: Race Condition (Cont.)

- How to fix
  - Do not change state of queue in `timeout`

```
readRequest q = do
  mRequest <- timeout 10 $ atomically $ do
    request <- peekTQueue q
    return request
  case mRequest of
    Just _ -> atomically $ tryReadTQueue q
    Nothing -> return Nothing
```

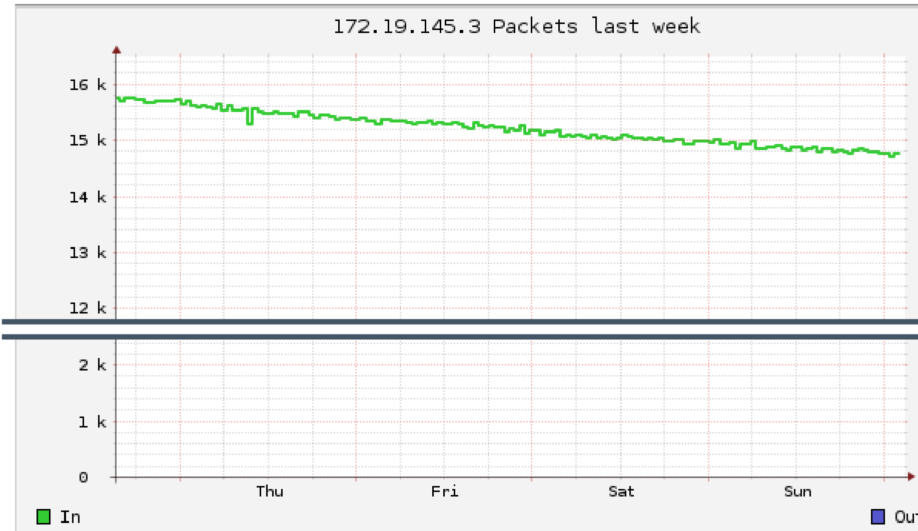
- Pitfall
  - Because `timeout` is implemented as a higher-order function, it is easy to compose with IO action without taking care of internal implementation
  - `timeout` can be used safely only with IO action which does not change data, such as `accept` and `connectTo`

Copyright © GREE, Inc. All Rights Reserved.

Pitfall 3: Library Misuse

GREE

- Issue: Performance degradation



- Cause
  - This program uses http-conduit to connect to backend HTTP servers periodically for health check

```
manager <- newManager def
http req manager
```

  - `newManager` forks thread to repeatedly collect stale connections
  - To finish this thread, `closeManager` must be called (from version 1.2.0)

Copyright © GREE, Inc. All Rights Reserved.

Pitfall 3: Library Misuse (Cont.)

- How to fix
  - Call `closeManager` or use `withManager`

```
withManager $ (\manager -> http req manager)
```

- Pitfall
  - Specification of `newManager` was changed from 1.2.0
  - Haskell libraries are often developed very actively

Copyright © GREE, Inc. All Rights Reserved.

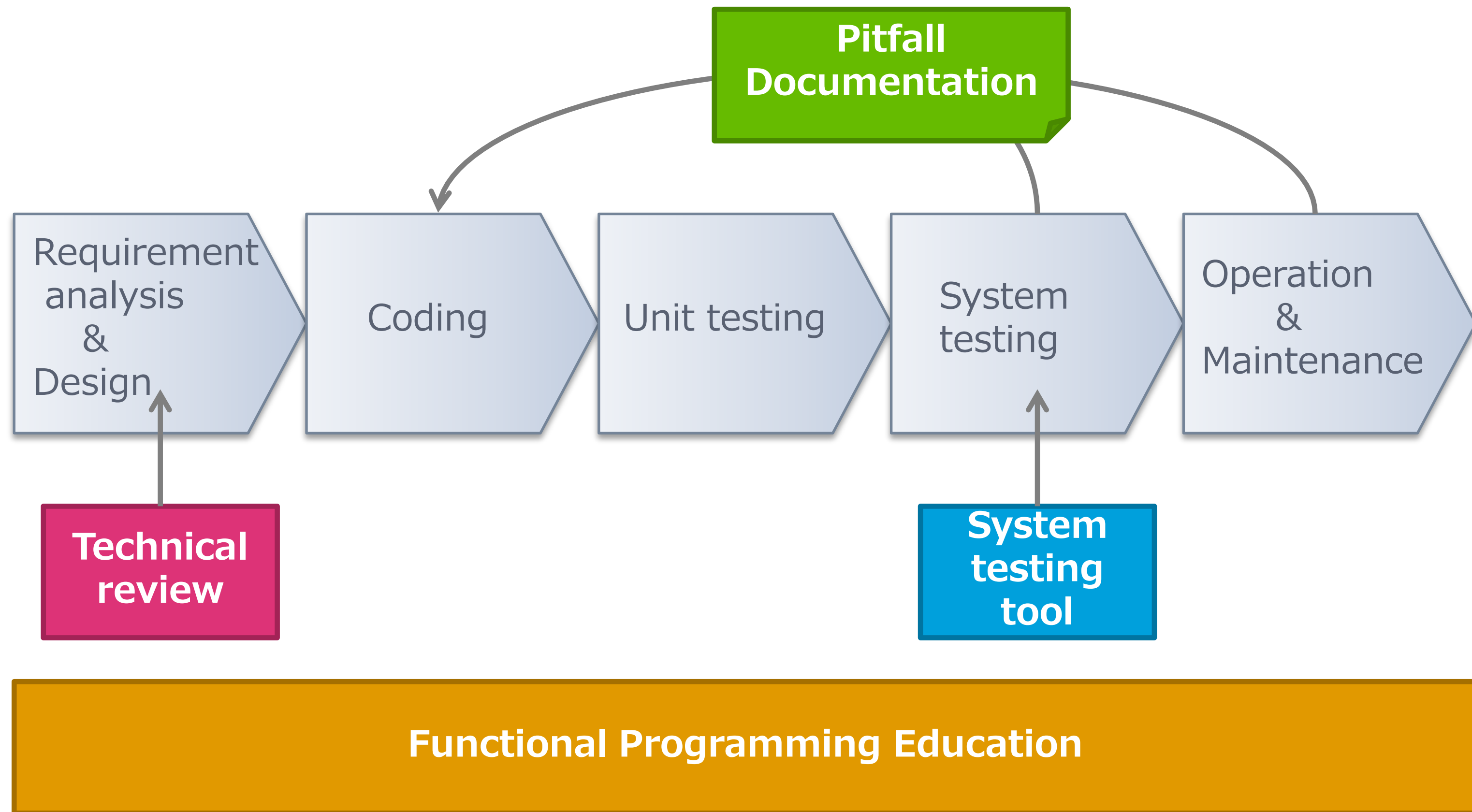
How to Avoid Pitfalls

Copyright © GREE, Inc. All Rights Reserved.

How to Avoid Pitfalls

GREE

- Overview of recurrence prevention method



Copyright © GREE, Inc. All Rights Reserved.

System Testing Tool (1/4)

GREE

- Haskell has great unit testing framework
  - HUnit, QuickCheck
- Unit testing is not enough to find critical bugs
  - System testing
  - Stress testing
  - Aging testing (long-running stress testing)
- test-sandbox
  - System testing framework
  - Write system tests using HUnit or QuickCheck
  - Can be used for network applications and CUI tools

Copyright © GREE, Inc. All Rights Reserved.

System Testing Tool (2/4)

GREE

- Example: memcached test (HUnit)

```
setup = do          -- Register memcached using free TCP port to env
  port <- getPort "memcached"
  register "memcached" "/usr/bin/memcached" [ "-p", show port ] def

test1 = sandboxTest "Store" $ do
  -- Send commant through registered TCP port
  output <- sendTo "memcached" "set key 0 0 5\r\nvalue\r\n" 1
  assertEqual "item is stored" "STORED\r\n" output

main =
  defaultMain
    [ sandboxTests "Example" $ do
        setup                  -- Setup env accessible from all tests
        start "memcached"
        sandboxTestGroup "All" [ test1, test2, ... ]
    ]
```

Copyright © GREE, Inc. All Rights Reserved.

System Testing Tool (3/4)

GREE

- Example: memcached test (QuickCheck)
  - For any string s, get(set s) == s

```
sandboxTest "Get and set" $ quickCheck $ do
  -- Take any string
  str <- pick arbitrary :: PropertyM Sandbox String

  -- Get and set string
  _ <- run $ sendTo "memcached"
         (printf "set key 0 0 %d\r\n%s\r\n" (length str) str) 20
  output <- run $ sendTo "memcached" "get key\r\n" 20

  -- Check that we get the same string
  assert $ printf "VALUE key 0 %d\r\n%s\r\nEND\r\n" (length str) str
    == output
```

Copyright © GREE, Inc. All Rights Reserved.

System Testing Tool (4/4)

- Applied to
  - KVS management system
  - Flare (KVS written in C++)
- # of tests
  - Frontend server
    - 49 property tests
    - 103 system tests
  - Control server
    - 45 system tests
    - 5000+ assertions
  - Flare
    - Found many bugs
    - > 7000 tests

http://hackage.haskell.org/package/test-sandbox

Copyright © GREE, Inc. All Rights Reserved.

Documentation of Pitfalls (1/4)

- Problem report
  - Describe details of problem
  - Linked from bug tracking system

  - Timeline of issue
  - Temporary measure
  - Extent of influence
  - Detailed cause and how to fix
  - Recurrence prevention
  - …

  - Scattered among a lot of other problem reports

- **Other FP programmers don't read them**

Copyright © GREE, Inc. All Rights Reserved.

Documentation of Pitfalls (2/4)

GREE

- Aggregated document
  - Collect problems caused by functional programming
  - Summarize cause and how to fix for each item
  - "Writing Middleware in Haskell"
- Contents
  - Lazy evaluation and memory leak
  - Preforking and load balancing
  - Concurrent programming
  - Libraries
  - Profiling and optimization
  - Test and debug
- **Other FP programmers still won't read it**

Copyright © GREE, Inc. All Rights Reserved.

Documentation of Pitfalls (3/4)

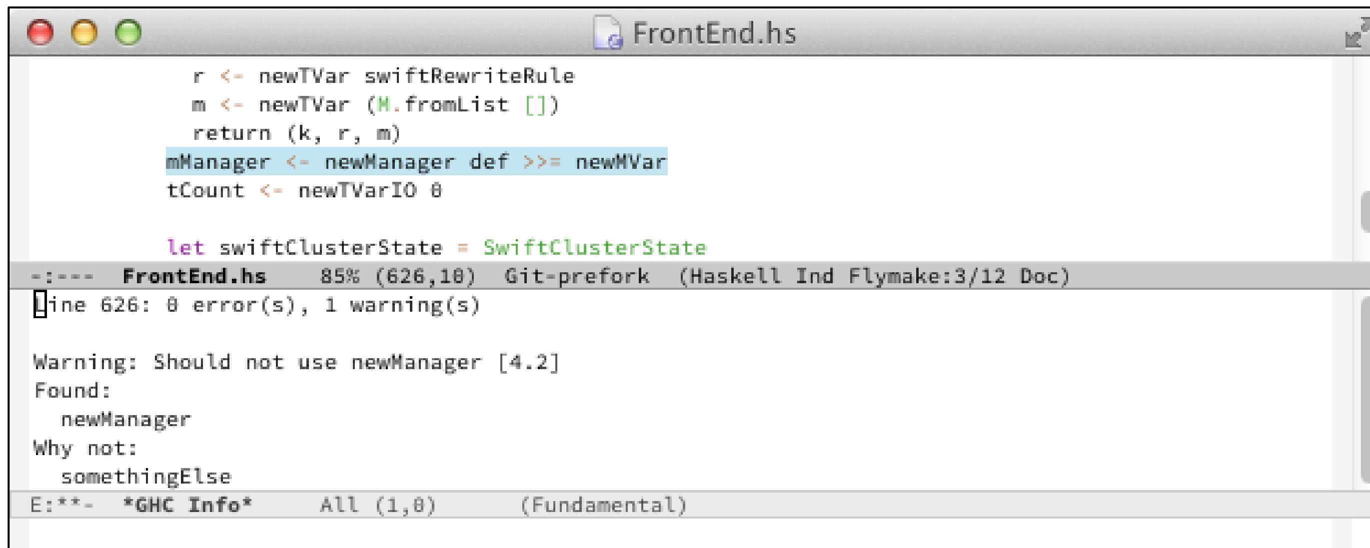
GREE

- Automated check using hlint
  - Customize hlint to check pitfall
  - Put item number of aggregated document in hlint comment

```
warn "Non-strict TVar [1.1]" = modifyTVar ==> modifyTVar'

warn "Should not use timeout with STM [3.1]" =
  timeout x (atomically f) ==> somethingElse
```

- Check from Emacs



Copyright © GREE, Inc. All Rights Reserved.

Documentation of Pitfalls (4/4)

GREE

- Problems of hlint method
  - Not all pitfalls can be detected by hlint
  - High level design issue
  - Library issue (Ex. Version of http-conduit, hashable)

Copyright © GREE, Inc. All Rights Reserved.

Technical Review

GREE

- Established technical review process
  - Check feasibility of new technologies such as functional programming by managements and other teams

Copyright © GREE, Inc. All Rights Reserved.

Education

- Brown bag FP meeting
  - Once or twice in a month
  - Scala and Haskell topics
  - "Make GREE a better place through the power of FP"
- Education program for new graduate
  - Haskell code puzzle from Project Euler

Copyright © GREE, Inc. All Rights Reserved.

Conclusion

GREE

- Functional programming is great
  - We develop some key components of our services using FP
- But there are many pitfalls
  - Lazy evaluation, race condition, library misuse, ...
- We should avoid them
  - Testing tool
  - Documentation
  - Technical review
  - Education

Copyright © GREE, Inc. All Rights Reserved.

Copyright © GREE, Inc. All Rights Reserved.