

왜 하스켈인가 ?

키노루

@keynoru

하스켈 프로그래밍 모임

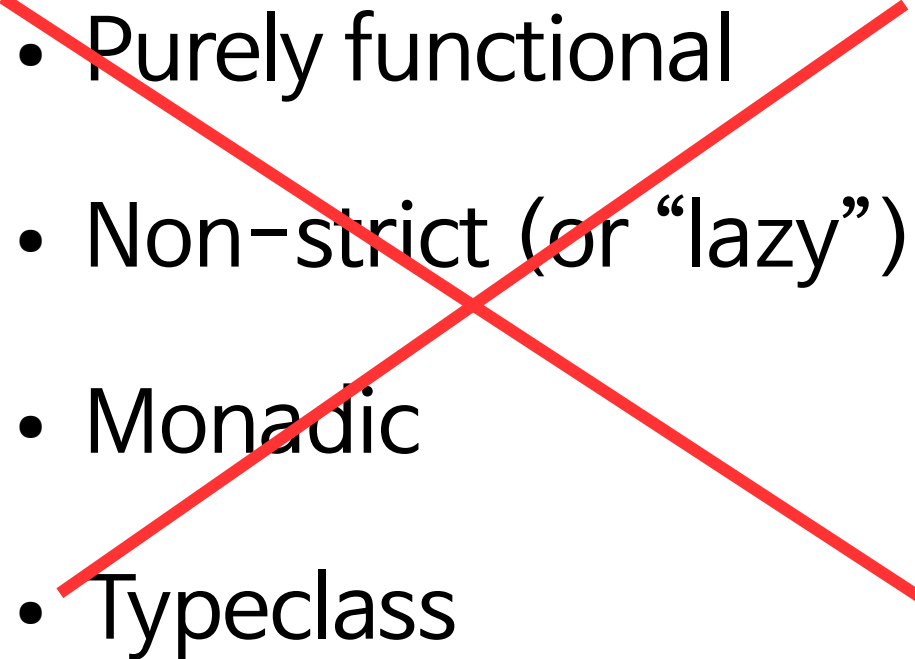
2015-03-11

한양대학교 IT/BT 관

# 대상

- 초급 프로그래머 : “ 프로그래밍 몰라요”
- 중급 프로그래머
- 프로그래밍 기초 지식이 있으면 좋지만 없어도 됨

# 하스켈이란

- Purely functional
  - Non-strict (or “lazy”)
  - Monadic
  - Typeclass
- 

# 현실적 조언

- 하스켈 코딩력 3개월 전까지 모나드라는 말은 보고도 못 본 척 듣고도 못 들은 척, 관심도 갖지 마십시오.
- 진심임.

# 하스켈이란

- Purely functional
  - Non-strict (or “lazy”)
  - Monadic
  - Typeclass
- 고성능의 소프트웨어를 버그 없이 작성할 수 있게 도와주는 현대적인 프로그래밍 언어

# 장점

- 정확성
- 성능
- 동시성
- 조합성
- 테스트링 , 프로파일링
- 커뮤니티와 생태계

정확성  
Correctness

“If it compiles, it works.”

- 실수를 빠르게 찾아내게 해주는 언어 설계와 지원
- 더 적은 노력으로 올바른 코드를 작성
- 컴파일러 통과하면 웬만하면 맞는 코드



# 어떻게 ?

- 정적 타이핑
- 자료가 기본적으로 불변
- 참조 투명성
- 알고리즘과 입출력의 분리

# 비슷해 보이는 : 파이선 코드

```
def f(a, b, c):  
    if a.startswith(b):  
        c.append(a)
```

# 비슷해 보이는 : 하스켈 코드

```
import Data.ByteString  
f a b c = if isPrefixOf b a  
    then a : c  
    else c
```

- 컴파일러가 확신할 수 있는 범위의 차이

# 명령형

```
x = f(1);
```

```
x = g(x);
```

```
function main() {
```

```
    h(x);
```

```
}
```

# 하스켈

`x = f 1`

`x = g x -- illegal`

`main = h x -- h cannot change x`

# 명령형 : 눈에 불을 켜고 변수 추적

```
x = f(1);
```

```
print(x);
```

```
x = g(x)
```

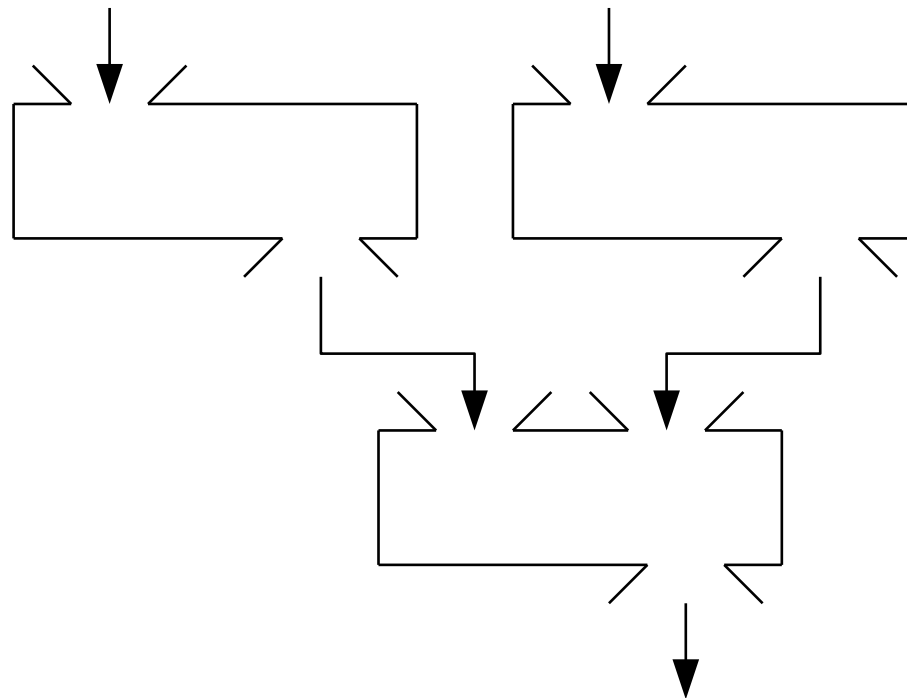
```
print(x);
```

```
h(x);
```

```
print(x); // 매번 다름
```

# 하스켈 : 참조 투명

- 함수에 같은 인자를 주면 반환값도 반드시 같다
- 변수의 변화를 추적할 필요가 없다



# 명령형 : 알고리즘과 입출력의 혼재

```
int f(int a, int b) {
```

```
    print("Hello, World!");
```

```
    return a + b;
```

```
}
```

```
f(1, 2); // ???
```



# 하스켈 : 입출력 격리

```
f x y = x + y
```

```
g x y = do
```

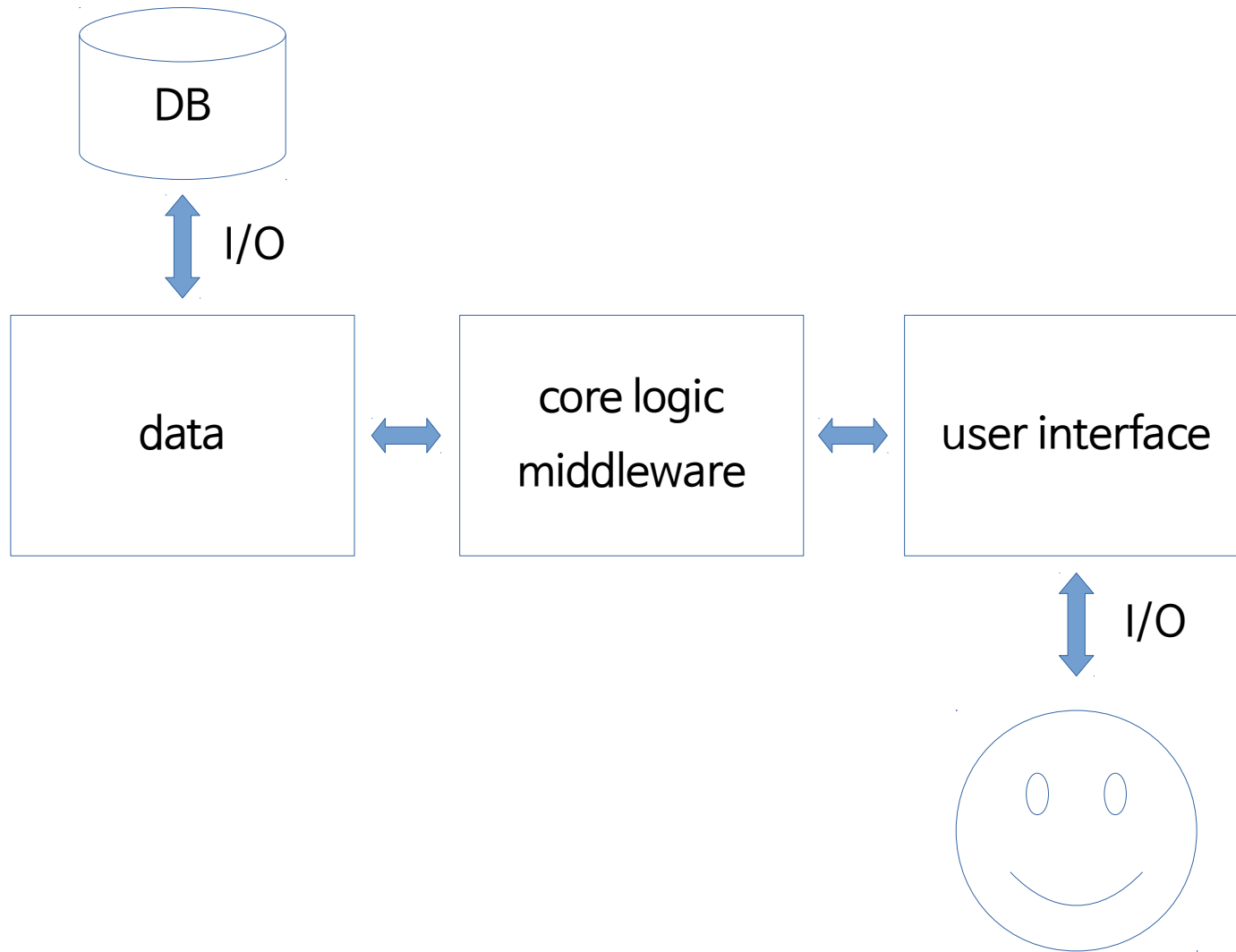
```
    print "Hello, World!"
```

```
    return (x + y)
```

```
f :: Int -> Int -> Int
```

```
g :: Int -> Int -> IO Int
```

# 입출력 격리는 새로운 개념인가 ?

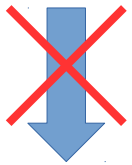


성능  
Performance

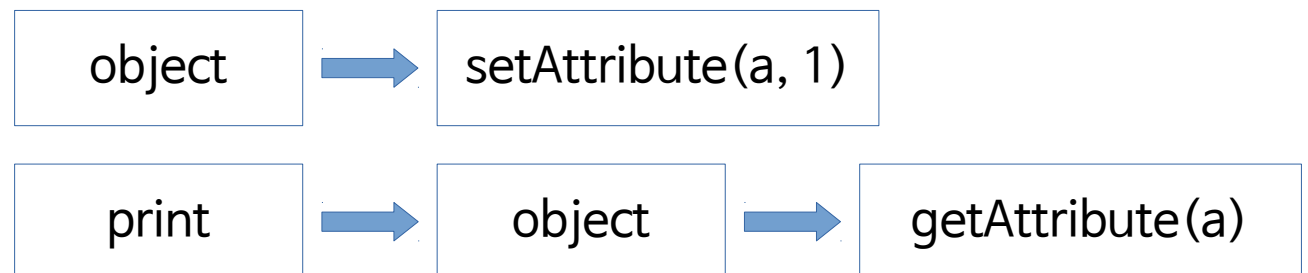
# 동적 언어 : 타입 모름

```
object.a = 1;
```

```
print(object.a);
```



```
print(1);
```



# 하스켈 : 타입 얹

```
data Object =
```

```
    Object { a :: Int }
```

```
myObj = Object { a = 1 }
```

```
main = print (a myObj)
```



```
main = print 1
```

# 정적 타입 시스템의 성능 이점

- 동적 언어보다 훨씬 더 강한 constant folding
- constant 가 되지 않는 경우에도 memory indirection 을 대폭 줄일 수 있음

# 네이티브 컴파일

- GHC 는 하스켈 코드를 C-- 이라는 C 의 서브셋으로 컴파일
- C-- 은 ANSI C 의 서브셋이므로 온갖 플랫폼으로 네이티브 컴파일 가능
- 최근 LLVM 백엔드도 발전

# 최적화

- 어떤 값이 Constant folding 이 되는지를 다른 언어에 비해 훨씬 잘 알고 있음
- Lazy lists: 다른 언어의 iterator 와 비슷
- Stream Fusion
- 그밖에 GHC 는 컴파일러 이론 연구자들이 수십 년간 연구하면서 누적된 최적화 덕칠



# 동시성 Concurrency

# 동시성의 중요성

- 프로세서의 클럭 스피드는 한계에 달함
- 코어를 늘리기 시작
- GPU 를 GPGPU 로 활용하기 시작
- 머신을 늘려 클러스터를 만들기 시작
- 대세

# 하스켈의 동시성

- 개어이
- 개이득

# 하스켈의 동시성 층위

- Spark
- Haskell thread
- OS thread

# Spark

```
main = do
```

```
  x <- action1 ...
```

```
  y <- action2 ...
```

```
  v <- h (f x + g y)
```

```
  ...
```

# 어이없는 하스켈의 동시성

```
$ ghc -o main Main.hs -threaded
```

```
$ ./main
```

```
$ ./main +RTS -N4
```

# Haskell thread: forkIO

```
main = do  
    forkIO f  
    ...
```

# Haskell thread

- 쓸 때는 그냥 스레드 쓰듯이 쓴다
- Chan(`Control.Concurrent.Chan`) 등으로 스레드 간 통신 : 그냥 `stdio` 하듯이 읽고 쓰면 동기화 알아서 됨
- 코드가 알아서 `epoll/kqueue` 기반으로 번역됨  
(?!)



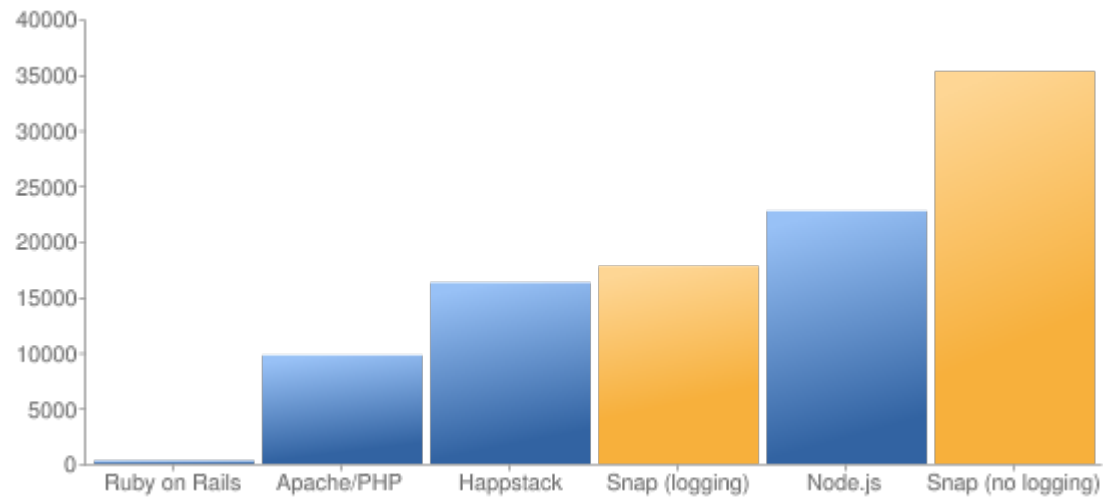
# OS thread: forkOS

```
main = do
```

```
    forkOS f
```

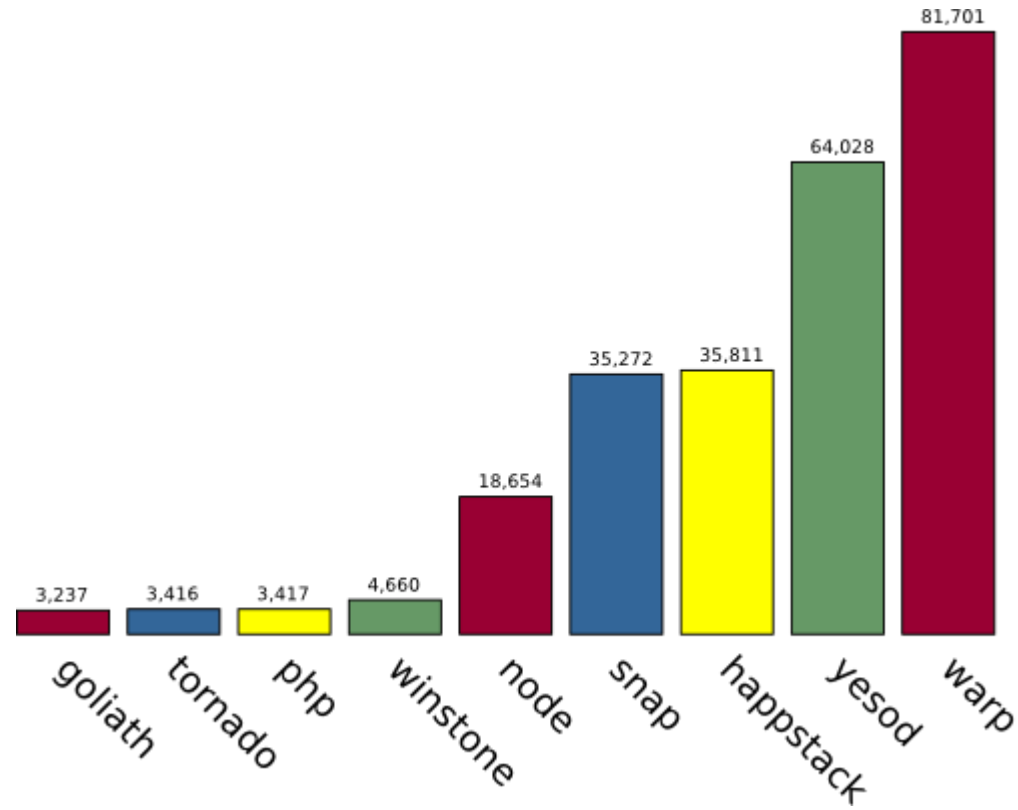
```
    ...
```

# 하스켈 동시성의 위력



<http://snapframework.com/blog/2010/11/17/snap-0.3-benchmarks>

# 하스켈 동시성의 위력



<http://www.yesodweb.com/blog/2011/03/preliminary-warp-cross-language-benchmarks>

# Software Transactional Memory

- 데이터베이스처럼 transaction 이 가능한 공간을 프로그램 내부에 !
- atomic read/write, atomic modify
- 타입 시스템 덕분에 transaction 과 atomicity 를 정확하고 쉽게 기술

# STM 뭐가 좋은가

- DB 만큼 안전하고 DB 만큼 편하면서 DB 보다 빠른 기억 공간
- race condition 해결
- 너무 좋아서 인텔 칩에 Hardware Transactional Memory 생김

# 조합성 Composability

# Regex

`(<.+>)+`

# Parser Combinator

```
many $ do
```

```
  char '<'
```

```
  many anyChar
```

```
  char '>'
```



# 하스켈의 조합성

- 추상 자료형
- 타입클래스로 인터페이스 부여
- 고계함수
- 조합적인 타입클래스들

# 테스팅

- “Haskell has ridiculously good testing support.”
- “QuickCheck is *shockingly* more effective at finding bugs than unit tests.”

Brian O'Sullivan

<https://bos.github.io/strange-loop-2011/talk/talk.html#%2829%29>

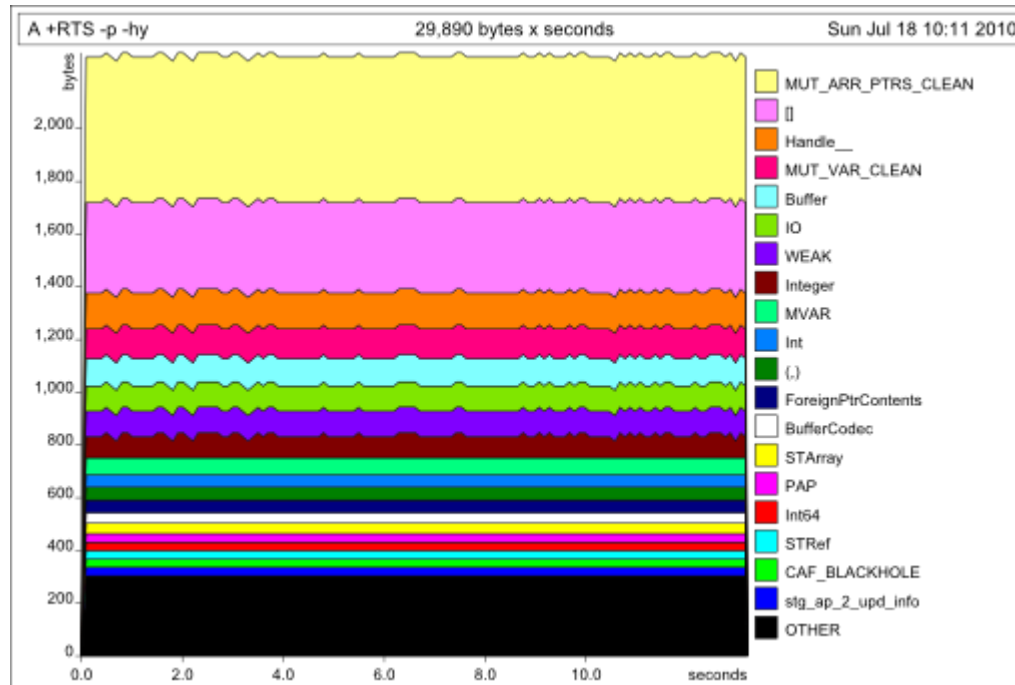
# QuickCheck

- 함수를 기술한다
- 이 함수가 만족해야 할 조건을 기술한다
- 퀵чек이 알아서 온갖 입력을 집어넣으며 그 조건을 과연 만족하는지 시험

# 프로파일링

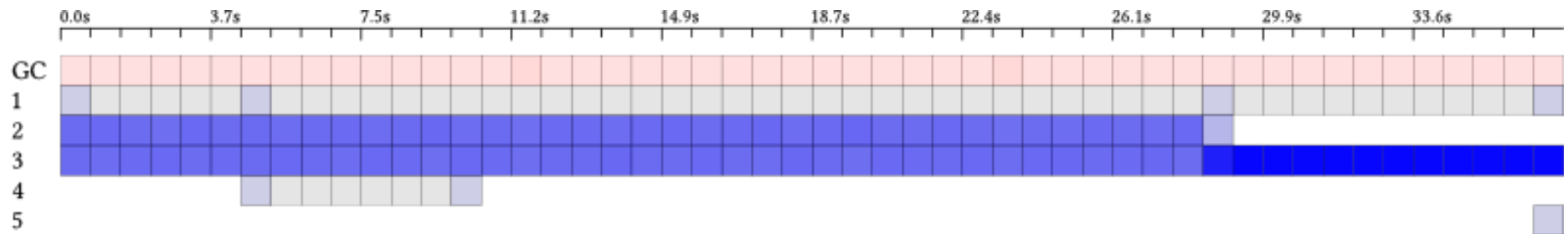
- runtime statistics
- time profiling
- heap profiling
- thread analysis
- core analysis
- comparative benchmarking
- GC tuning

# Heap profile



<http://stackoverflow.com/a/3276557/2079797>

# ghc-events-analyze



<http://www.well-typed.com/blog/86/>

# 하스켈러의 특징

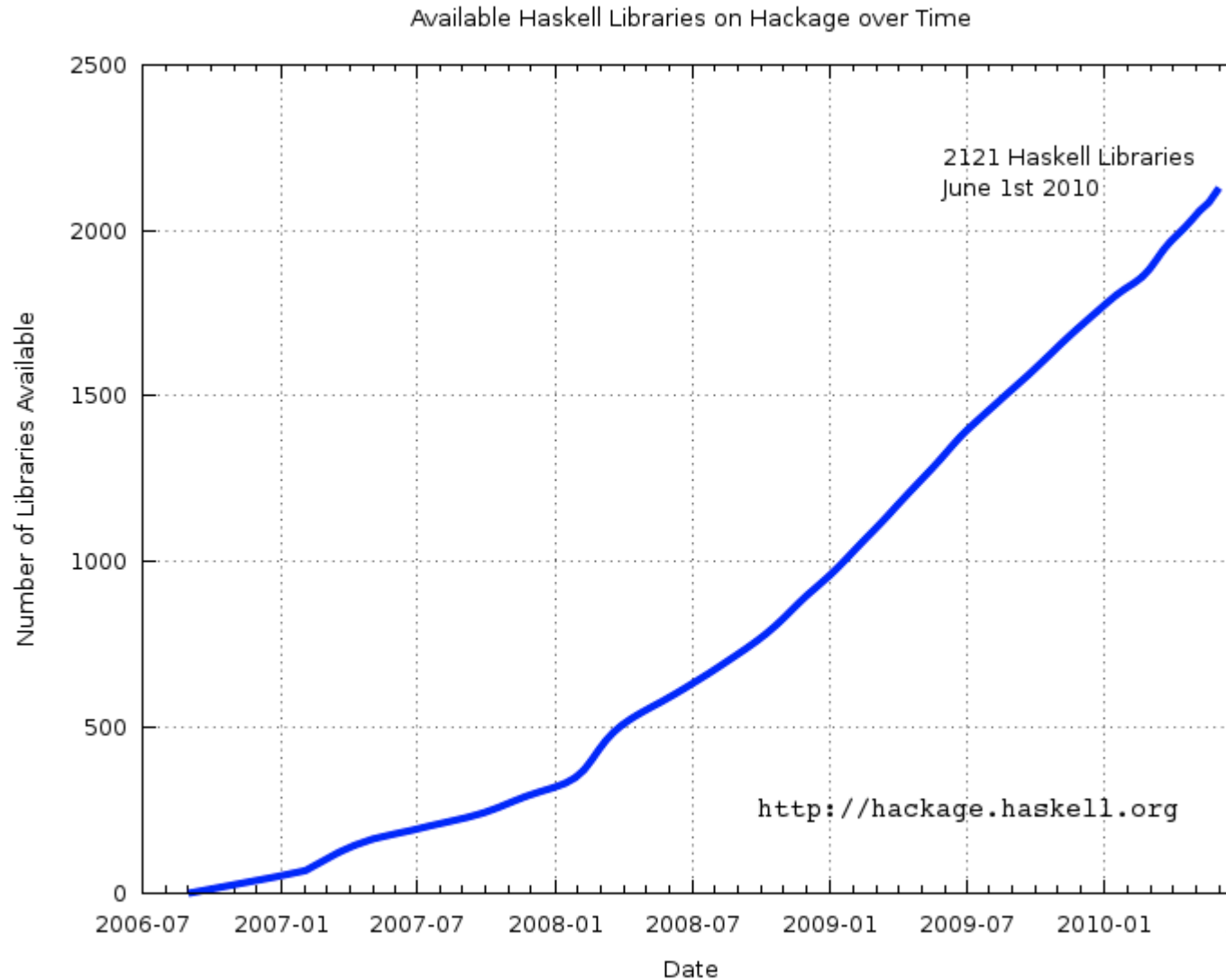
- 아무개가 하스켈을 한다
  - 지적 호기심이 있다
  - 초기 진입 장벽을 돌파할 만큼 근성이 있다
  - 아주 멍청하지는 않음

# 하스켈 커뮤니티의 특징

- 뭔가 물어보면 기뻐함 (?)
- 잘 도와줌
- Stack Overflow, Reddit, IRC, ...



# 생태계



<https://donsbot.files.wordpress.com/2010/05/hackage-june-2010.png>

# 하스켈 라이브러리

- 시스템 (POSIX, Win32, ...), 웹 (warp, snap, yesod, heist, ...), 병렬 처리 (vector, repa, Accelerate, DPH, ...), 네트워크 (wreq, tls, secure-sockets. ...), 파서 (Parsec, attoparsec, ...), ...
- 각종 외부 라이브러리 바인딩 : 그래픽스 , 데이터베이스 , ...

# 책

- Learn You a Haskell for Great Good  
<http://learnyouahaskell.com/>
- Real World Haskell  
<http://book.realworldhaskell.org/read/>