

(続) Haskell で
Web サーバーを
実装してみました

IIJ イノベーションインスティテュート

山本和彦

Functional language



関数型言語



数学



怖い

Functional



実用的

Functional language



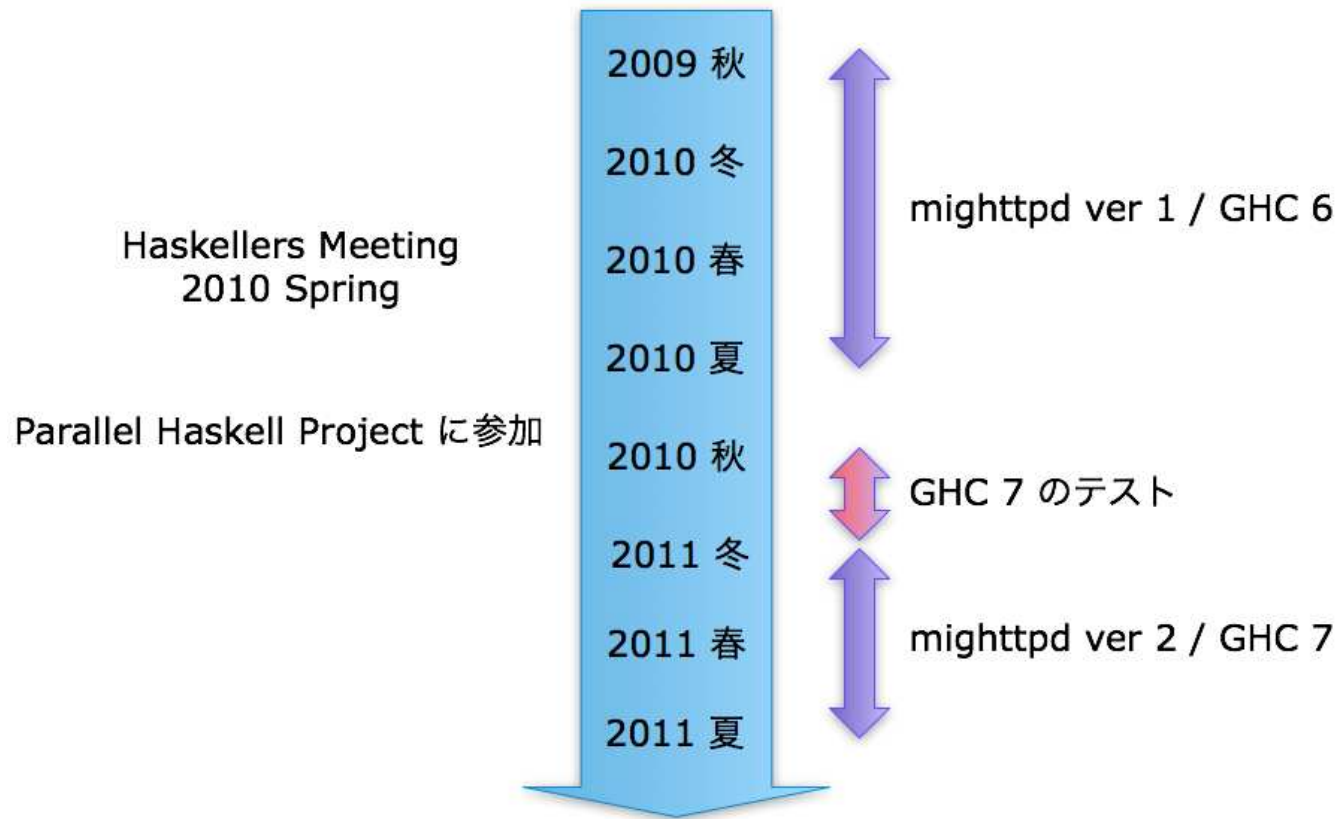
実用的言語

もし Haskell で
世界最速の Web サーバーを
実装できたら



Haskell = 実用的
と言っていいかな？

Mighttpd の歴史




なぜ Web サーバーを実装したのか？



2009年の秋、研究のために、自由に
変更できる Web サーバーが必要になった。



 Apache も考えたが、大きくて複雑。
C を読んだり書いたりするのは、もううんざり。



Haskell で書いた Web サーバーが欲しい。
でも、そんなものの存在は知らなかった。



そこで一から書き始めた。
名前は Mighttpd。Mighty と発音する。

3つのゴール

機能

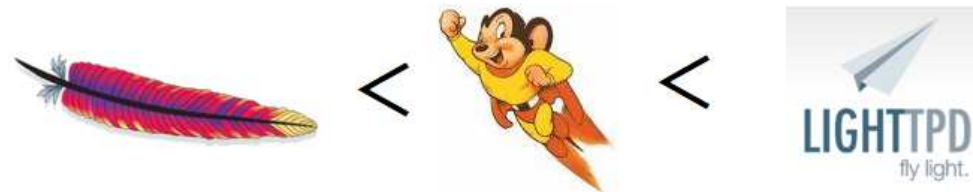
Mighttpd は、僕のドメイン Mew.org の Apache を置き換えるに十分な機能を提供すること。

部品化

Mighttpd は、僕の研究のために簡単に変更できること。

性能

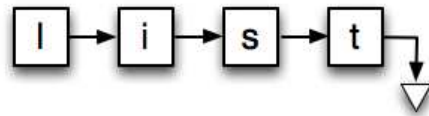
Mighttpd は、静的コンテンツに対して Apache よりも性能を出すこと。



性能を出す2つのアイデア

ByteString

伝統的な Haskell の **String** はとても遅い。



ByteString は C の **char[]** と同程度に速い。

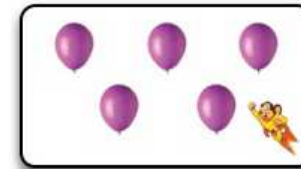


ユーザー
スレッド

カーネルスレッドは重い。



ユーザースレッドは軽い。



HTTP とスレッド

ネットワーク
プロトコル

メッセージ
指向

DNS

ストリーム
指向

SMTP, HTTP

ネットワーク
プログラミング

イベント
駆動

`select, kqueue, epoll`

スレッド

`fork, pthread_create`



ストリーム指向をイベント駆動で書くと見通しが悪い。



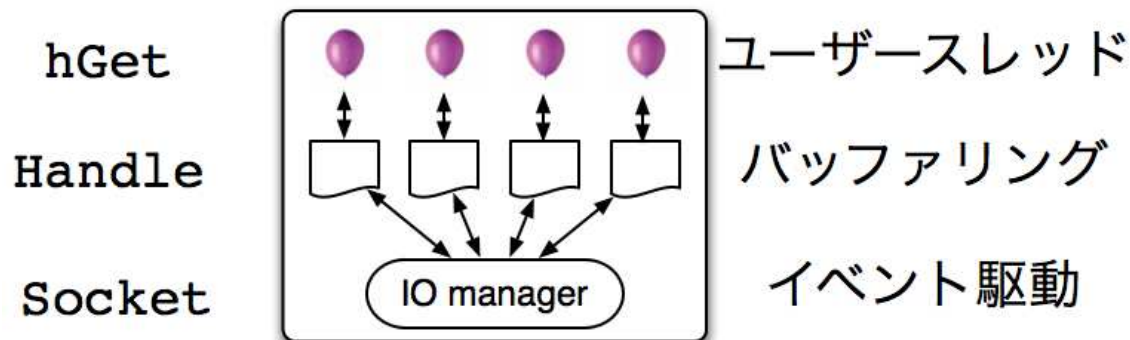
ストリーム指向をスレッドで書くと見通しがよい。






HTTP はスレッドで書きたい。
簡潔なことは、よいことだ。

ユーザースレッドは本当のスレッド

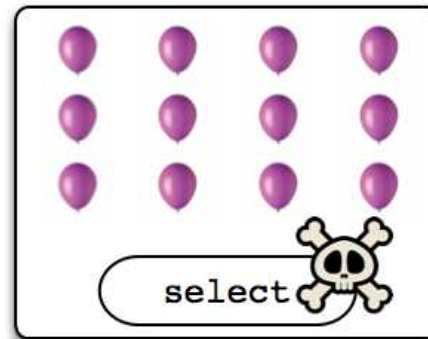
- 🙄 GHC にはユーザースレッドで実装されたイベント駆動の IO マネージャーがある。
- 🙄 バッファリングしたり、ブロックしているユーザースレッドを起こしたり。
- 😊 ユーザースレッドを使っても、真のスレッドプログラミングができる。



1,024 コネクションの壁

-  IO マネージャーは `select` を使っ
て実装されている。
-  `select` は 1,024 以上のファイル/
コネクションを扱えない。
-  もし GHC 6.12 が 1,024 以上のコネ
クションを受け取ると、資源枯渇の例外
が発生する。

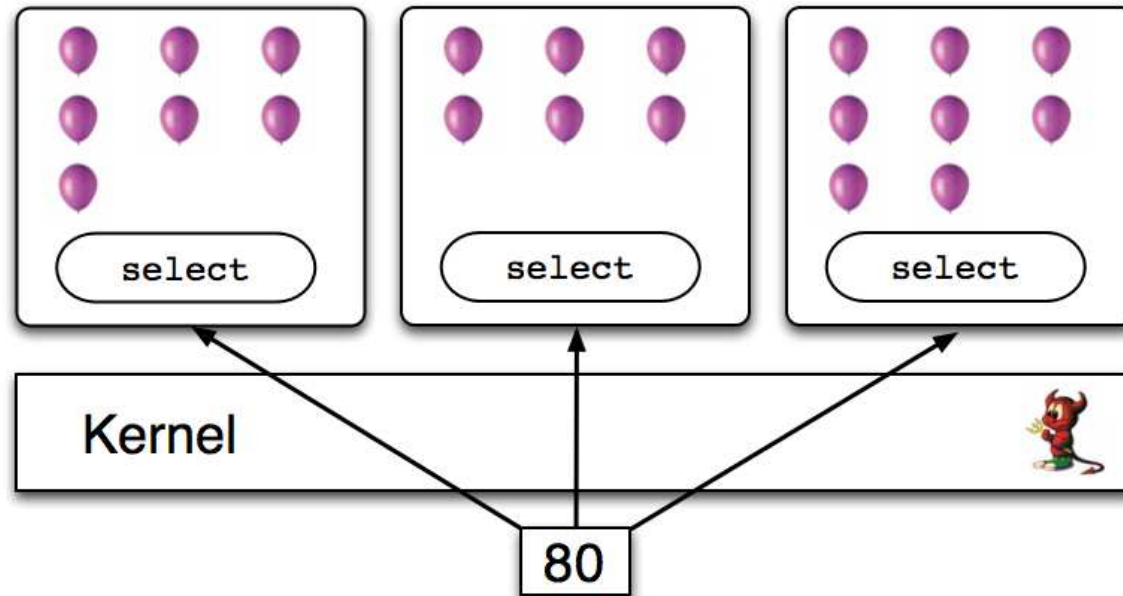
1024



Prefork ライブラリ



Prefork は、fork したプロセス間でリスニングポートを共有する技術。



これで GHC 6.12 でコネクションを何本でも受けられる。

Mighttpd の実装

パッケージ名

mighttpd

File base

KVS base

リリースして
いない

webserver

HTTP, session,
redirect, CGI

c10k

prefork

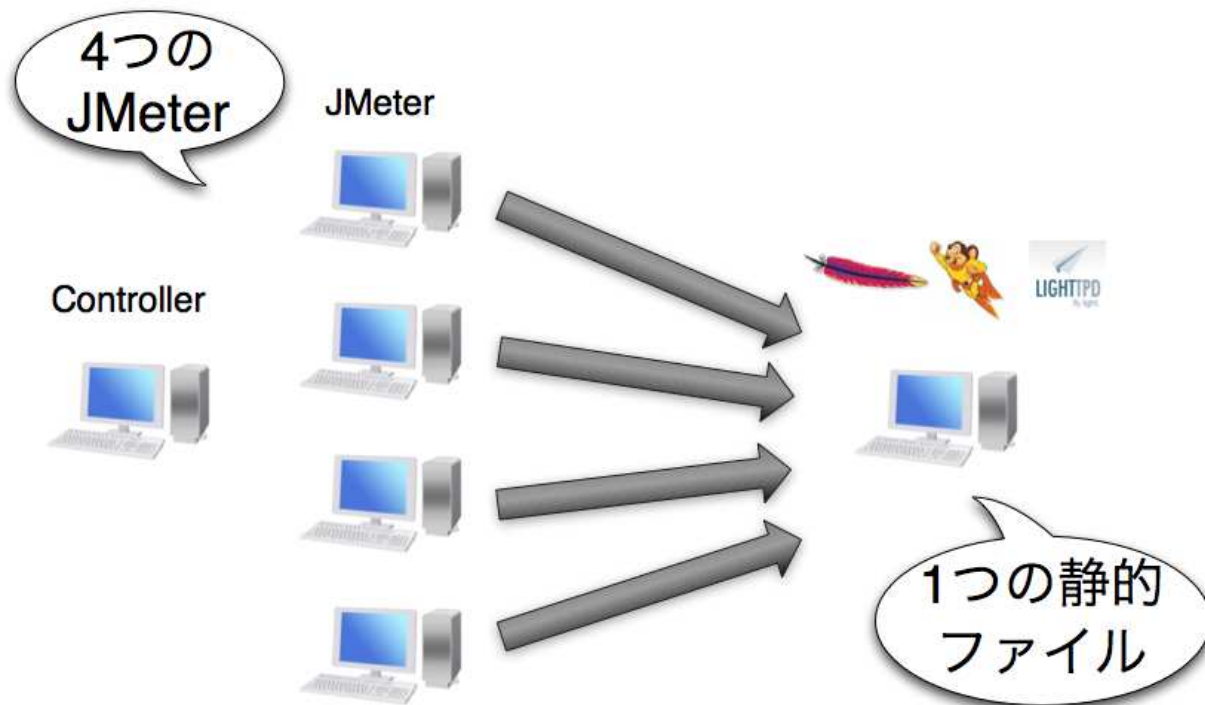
部品化

"webserver" は、いろいろな
ストレージを利用できるように設計。

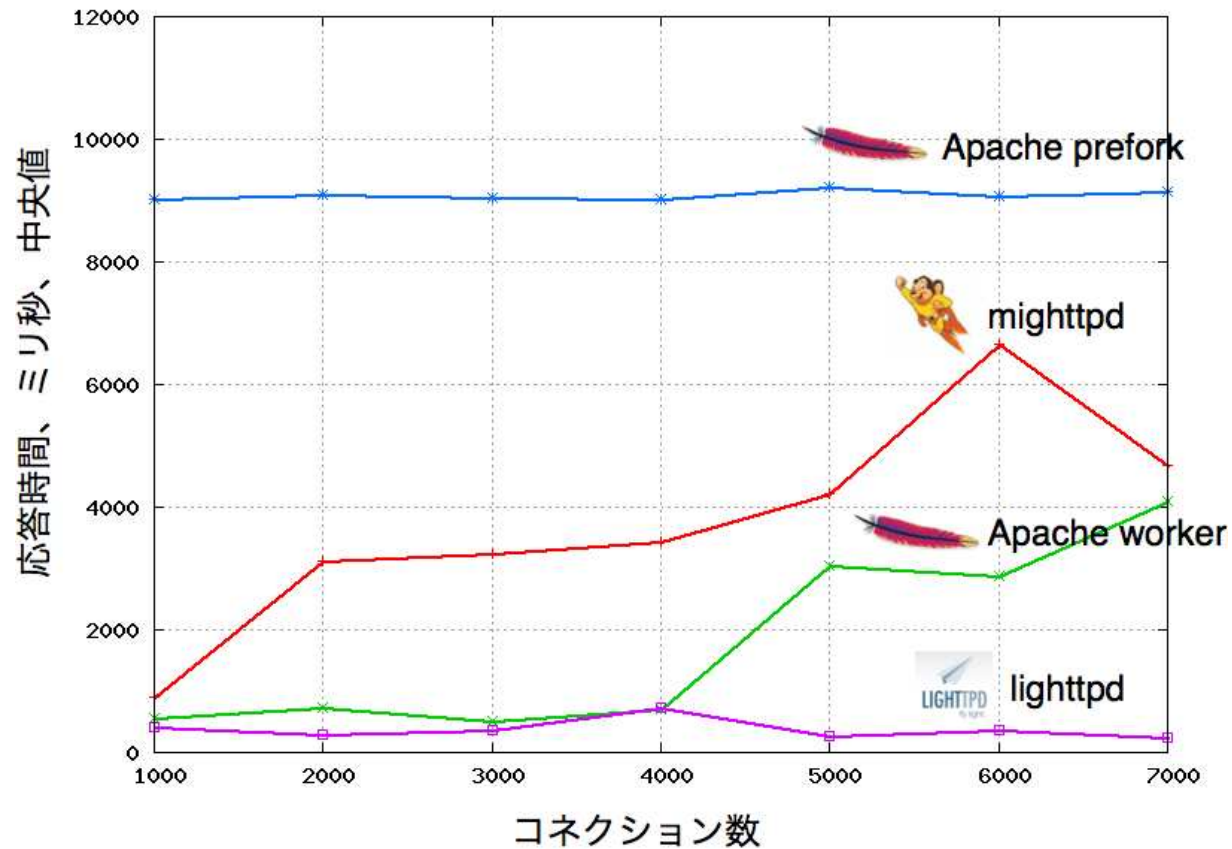
機能

"mighttpd" は、Mew.org で
動いている。

ベンチマーク環境



ベンチマークの結果



■ ベンチマークは不安定なので、あまり信用しないで下さい

Mighttpd 1 と 2 の間

Parallel Haskell
Project

MS Research
well-typed
IJJ-II

GHC 7

新しい IO マネージャ
epoll, kqueue

Web Application
Framework

Snap
HappStack
Yesod

Haskellers Meeting 2010 Spring

- Simon Peyton Jones さんが東京にやってきた
 - 2010年4月16日

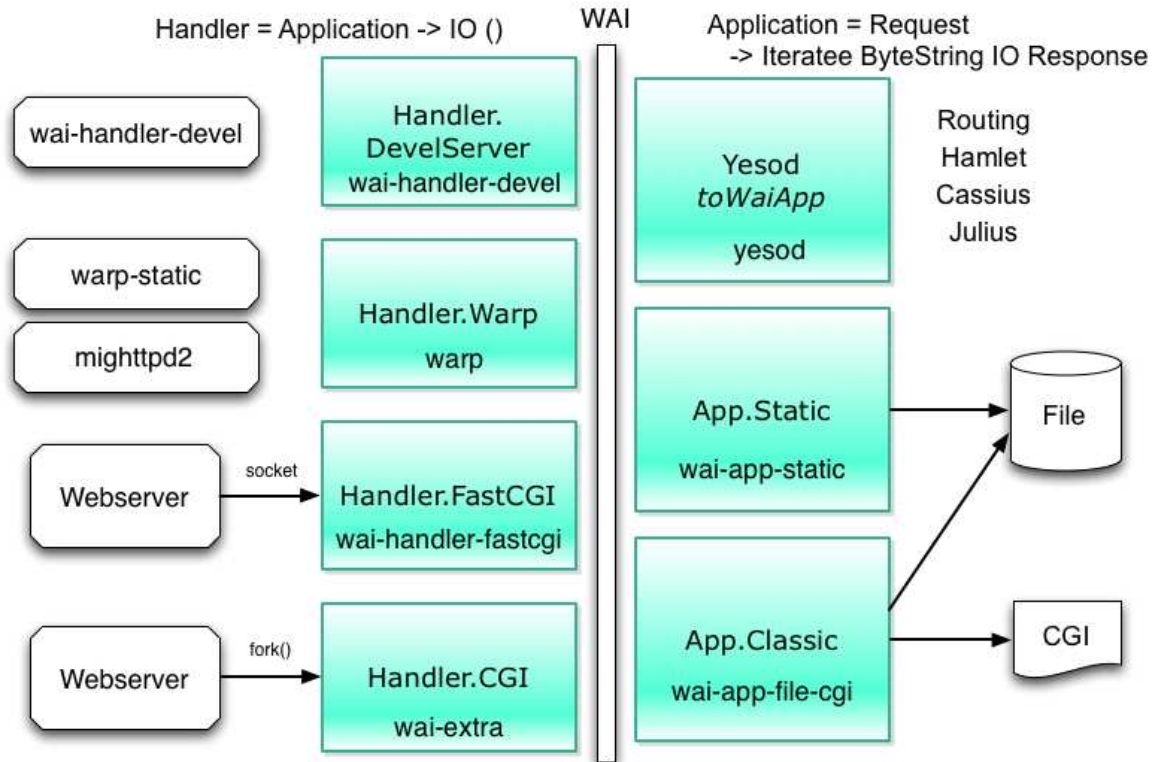
- 発表
 - Haskell で Web サーバーを実装してみました
 - <http://www.mew.org/~kazu/material/2010-mighttpd.pdf>
 - これをきっかけに Parallel Haskell Project に参加

GHC 7 のテスト

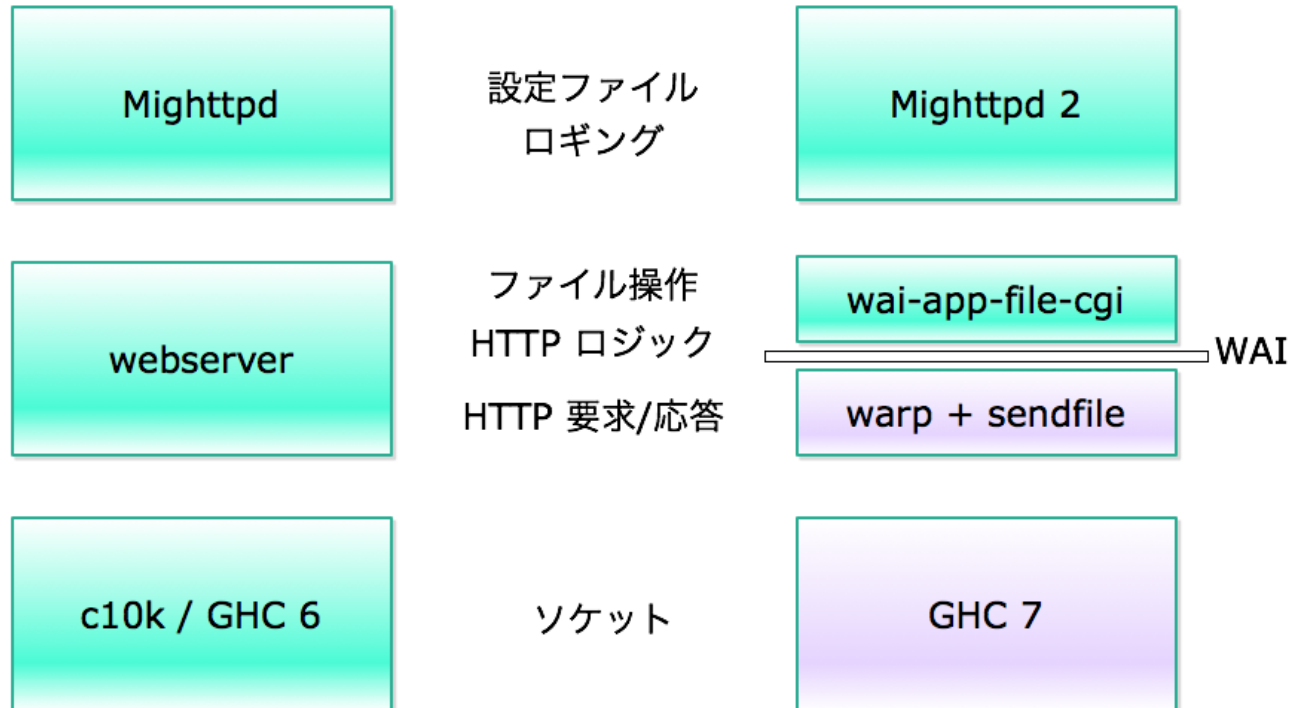
- GHC 7.0.1 の IO マネージャーは不安定だった
 - 僕と well-typed が6つのバグを見付け
 - GHC HQ と well-typed が直した
- バグ
 - Mac でデーモン化すると kqueue ソケットがなくなる
 - <http://hackage.haskell.org/trac/ghc/ticket/4449>
 - シグナルを待てない
 - <http://hackage.haskell.org/trac/ghc/ticket/4504>
 - イベントログが変
 - <http://hackage.haskell.org/trac/ghc/ticket/4512>
 - IO マネージャーがデッドロックに陥る可能性がある
 - <http://hackage.haskell.org/trac/ghc/ticket/4514>
 - getContents の挙動が変
 - <http://hackage.haskell.org/trac/ghc/ticket/4895>
 - hsc2hs は Mac でうまく動かない
 - <http://hackage.haskell.org/trac/ghc/ticket/4852>
- GHC 7.0.2 以降、IO マネージャーは安定した

Web Application Interface

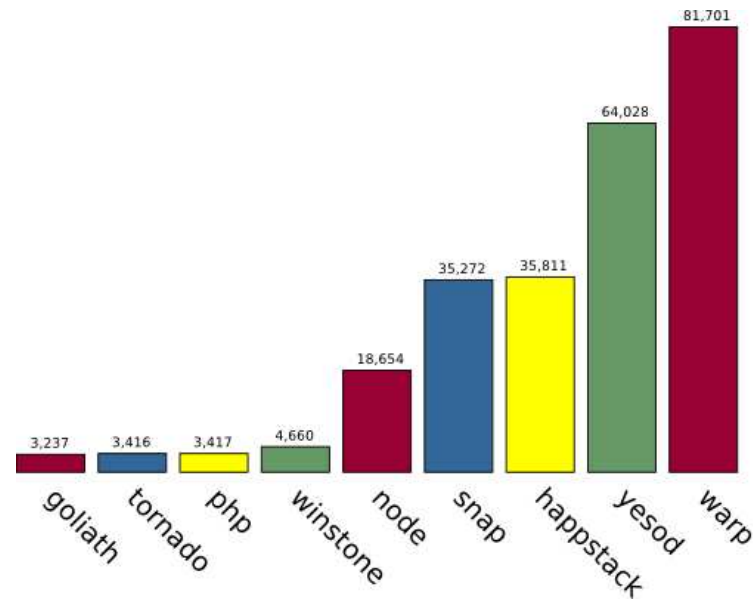
■ Yesod と HappStack の API



Web Application Interface の採用



Warp の性能

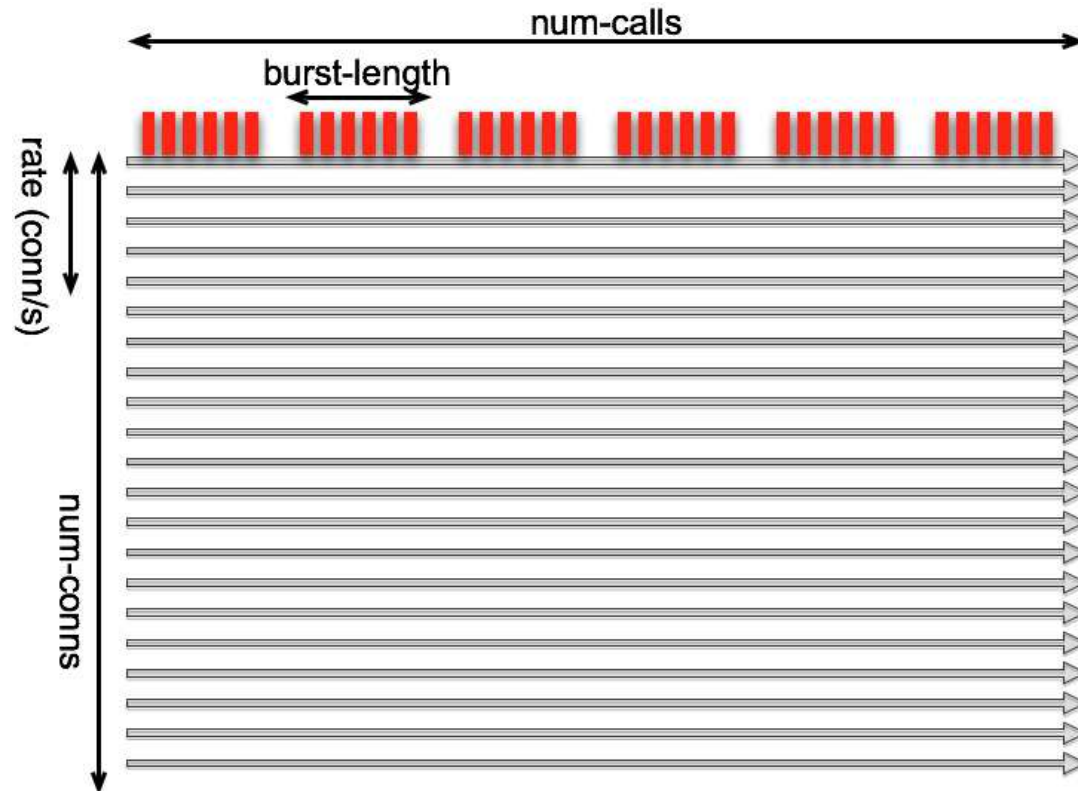


<http://www.yesodweb.com/blog/2011/03/preliminary-warp-cross-language-benchmarks>

■ Warp

- HTTP ロジックなし
- HTTP 要求を解析し HTTP 応答を返すだけ
- Last-Modified: などは処理しない
- ファイルは操作しない

httperf Ping-Pong ベンチマーク



```
httperf --hog --num-conns 1000 --num-calls 1000  
--burst-length 20 --rate 1000 --server localhost  
--port 3000 --uri /
```

Warp と mighttpd 2

- 僕の環境でのベンチマーク
 - ホスト
 - Intel(R) Xeon(R) CPU L5520 @ 2.27GHz x 8, 4 コア/CPU (32 コア)
 - 24G メモリ
 - Ubuntu 10.04, KVM 0.12.3
 - ゲスト
 - 4 コア
 - 1G メモリ
 - Ubuntu 10.10
- Warp (メモリのみ)
 - 23928.1 req/s, 1 コア, ロギングなし
 - 81701.0 req/s (先ほどの値)
- Mighttpd 2 (静的ファイル)
 - 4229.7 req/s, 1 コア, ロギングなし

チューニング

Parsec lib
↓
attoparsec lib

Data.Map for
Content-Type:
↓
static-hash lib

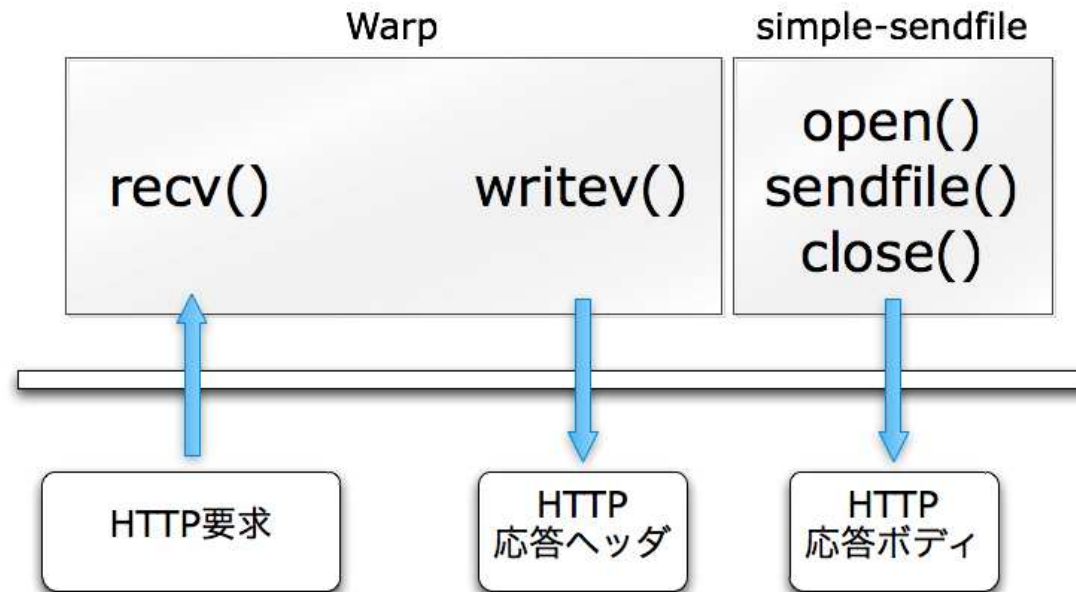
Date.Time
↓
http-date lib

システムコール
↓
使用を避ける

getFileStatus
↓
キャッシュ

sendfile lib
↓
simple-sendfile
lib

システムコール



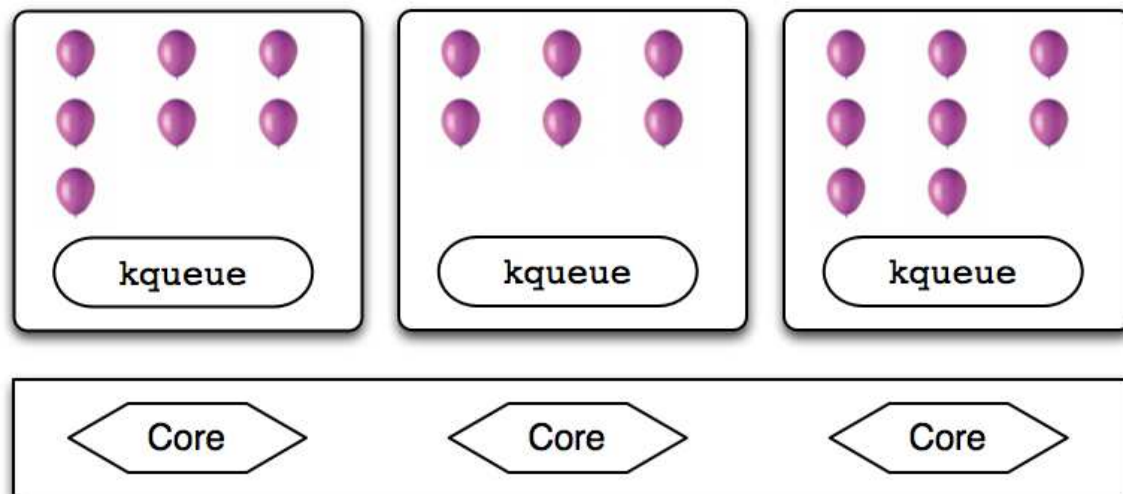
もちろんすべてノンブロッキングで！

1コア上でのベンチマーク

- nginx
 - 22713.3 req/s, 1 コア, ロギングなし
- Warp (メモリーのみ)
 - 23928.1 req/s, 1 コア, ロギングなし
- mighttpd2 (静的ファイル)
 - 21601.6 req/s, 1 コア, ロギングなし
 - 4229.7 req/s, 1 コア, ロギングなし, チューニング前

マルチコアでのスケールさせる

- IO マネージャーは単一のカーネルスレッド
 - +RTS -Nx としても、マルチコアではスケールしない
 - +RTS -Nx は forkProcess と一緒には使えない
 - 再び Prefork を導入した

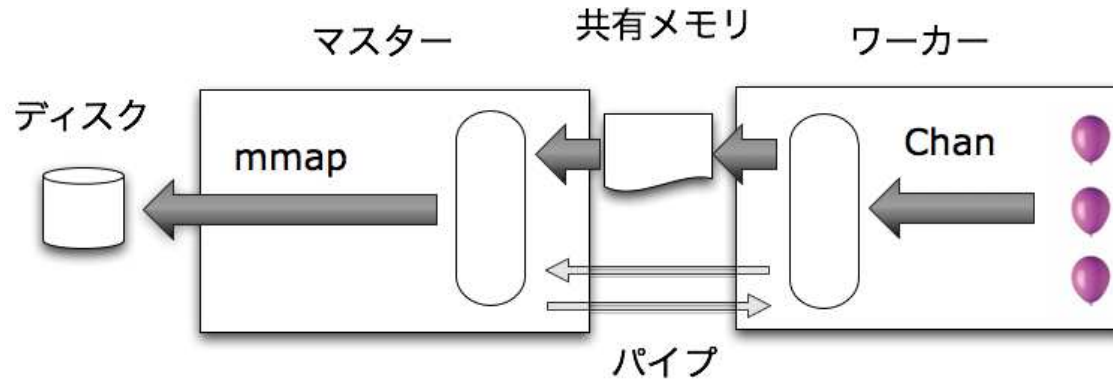


3コア上でのベンチマーク

- nginx, ワーカー3つ
 - 30471.2 req/s, 3 コア, ロギングなし
 - 22713.3 req/s, 1 コア, ロギングなし
- mighttpd2, ワーカー3つ
 - 61309.0 req/s, 3 コア, ロギングなし
 - 21601.6 req/s, 1 コア, ロギングなし

ロギング ～最大のボトルネック～

- 思いつく限りのアイデアを試した



- 一番単純な実装が一番速かった orz



ロギングありのベンチマーク

- nginx, ワーカー3つ
 - 25035.2 req/s, 3 コア, ロギングあり
 - 30471.2 req/s, 3 コア, ロギングなし
- mighttpd2, ワーカー3つ
 - 31101.5 req/s, 3 コア, ロギングあり
 - 61309.0 req/s, 3 コア, ロギングなし
- ロギングに改良の余地あり？

スタート Mighttpd

```
% cabal install mighttpd2
```

```
http://mew.org/~kazu/proj/mighttpd/
```

最後に宣伝

関数プログラミングの集い

Clojure, Erlang, F#,
Haskell, OCaml, Scala

2011年9月17日(土)

会場はここです