

Haskell で実装した 高速 Web サーバ Mighttpd について

2012.11.14



山本和彦

内容

- Mighttpd とは何か？
- なぜ Haskell か？
 - 安全で軽量なユーザスレッド
- どうやって高速化したか？
 - システムコールの低減
 - 特化と再計算の回避
 - ロックの低減

Mighttpd とは何か？

Haskell で実装した Web サーバ

Mighttpd の特徴

オープン

IIJ-II が開発
Haskell で実装

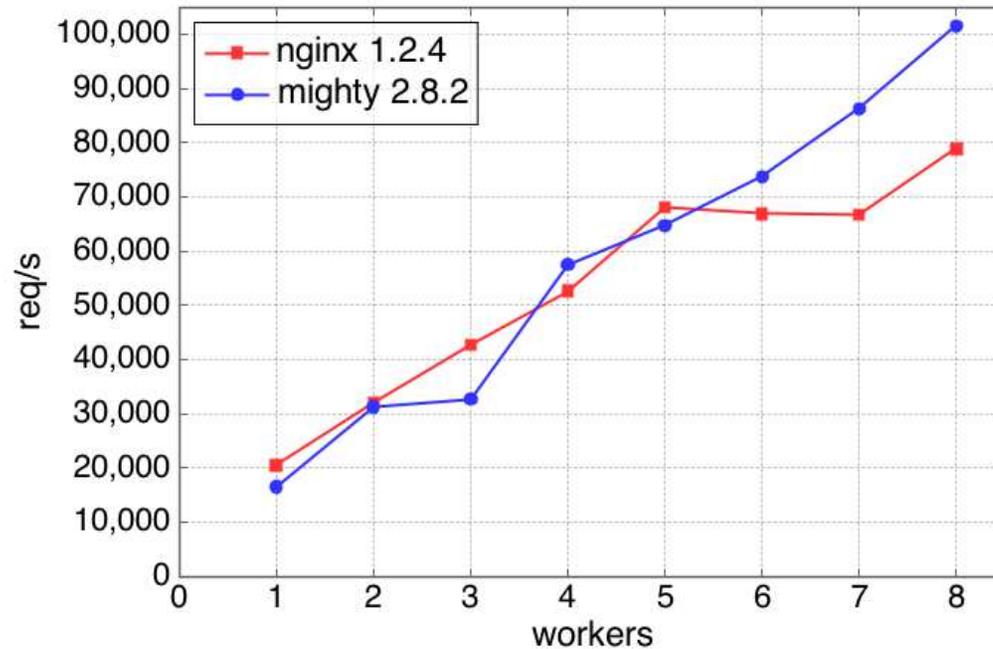
高速

nginx に比肩
機能は少ない

WAI アプリ

Yesod ファミリ
Warp の上に構築

Ping Pong ベンチマーク

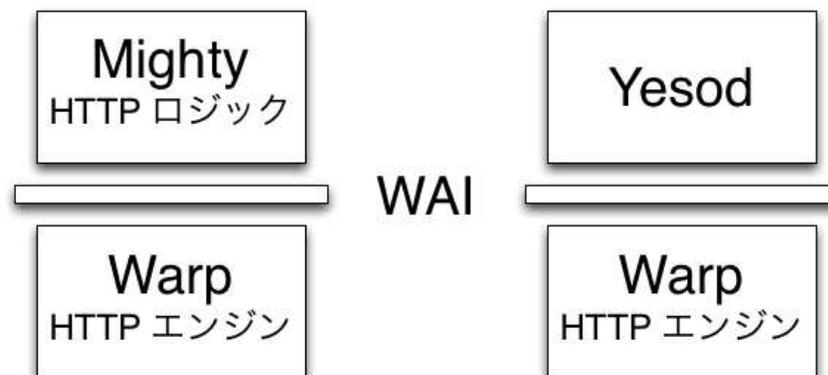


- 12コアマシン、Linux 3.2.0、ハイパーバイザーなし
- `weighttp -n 100000 -c 1000 -t 3 -k http://127.0.0.1:8000/`
- index.html は 151 バイト
- できる限り同じように設定

WAI

- Web Application Interface

```
type Application =  
    Request -> ResourceT IO Response
```



なぜ Haskell か？

軽量スレッドがあるから

Haskell の軽量スレッド

軽量

10万個起動しても大丈夫

安全

非同期例外も捕捉できる

プロセスとスレッドと関数型言語



命令型言語が軽量スレッドを提供するのは困難

サーバ・アーキテクチャ

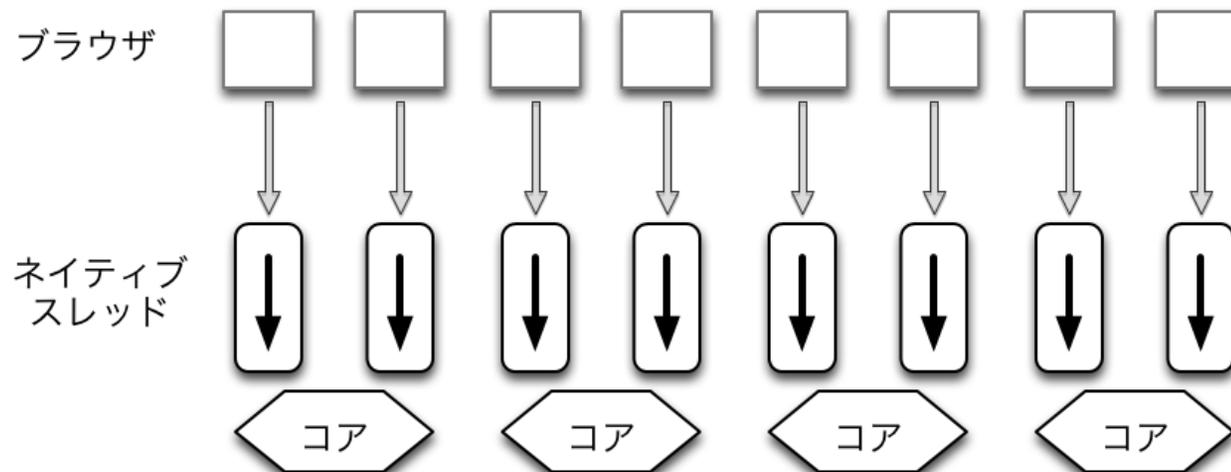
コードの見通し

ネイティブ・スレッド	○	×	○
イベント駆動	×	○	×
1コア1プロセス マッピング	×	○	○
軽量スレッド	○	○	○

コア当たりの性能

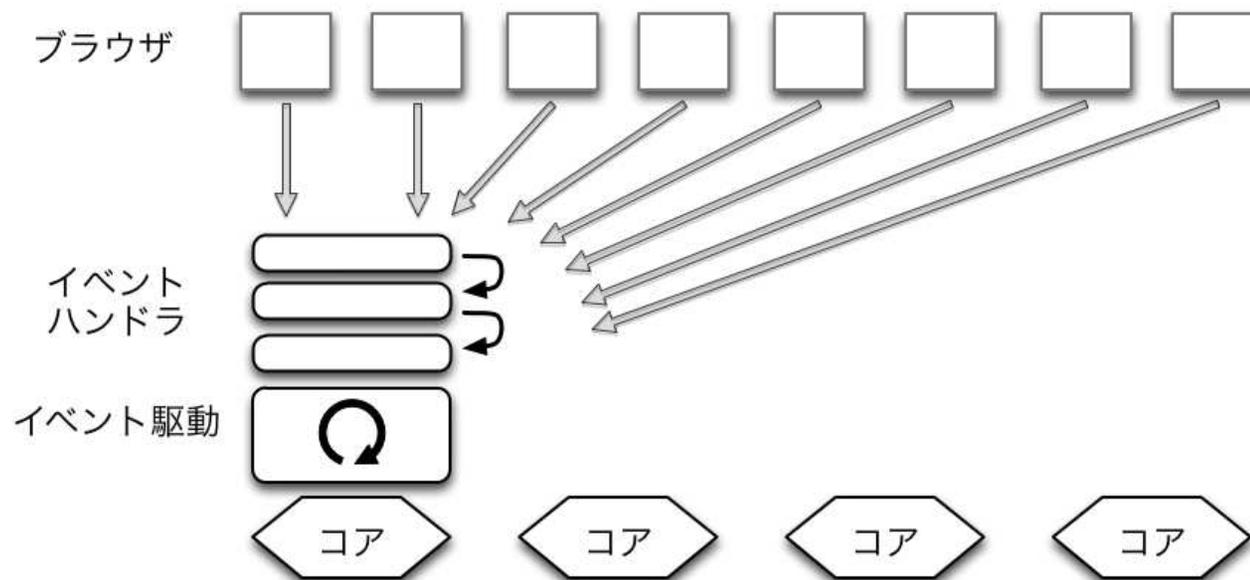
マルチコアの活用

ネイティブ・スレッド



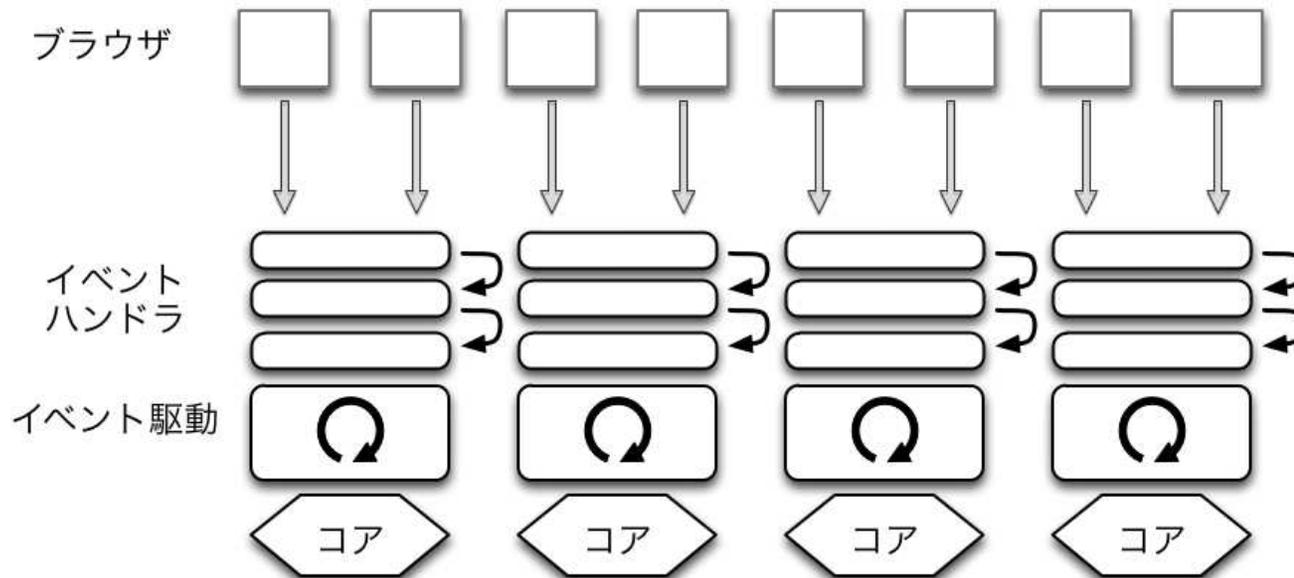
- 利点
 - マルチコアを活用できる
 - コードの見通しがよい
- 欠点
 - 性能が出ない

イベント駆動



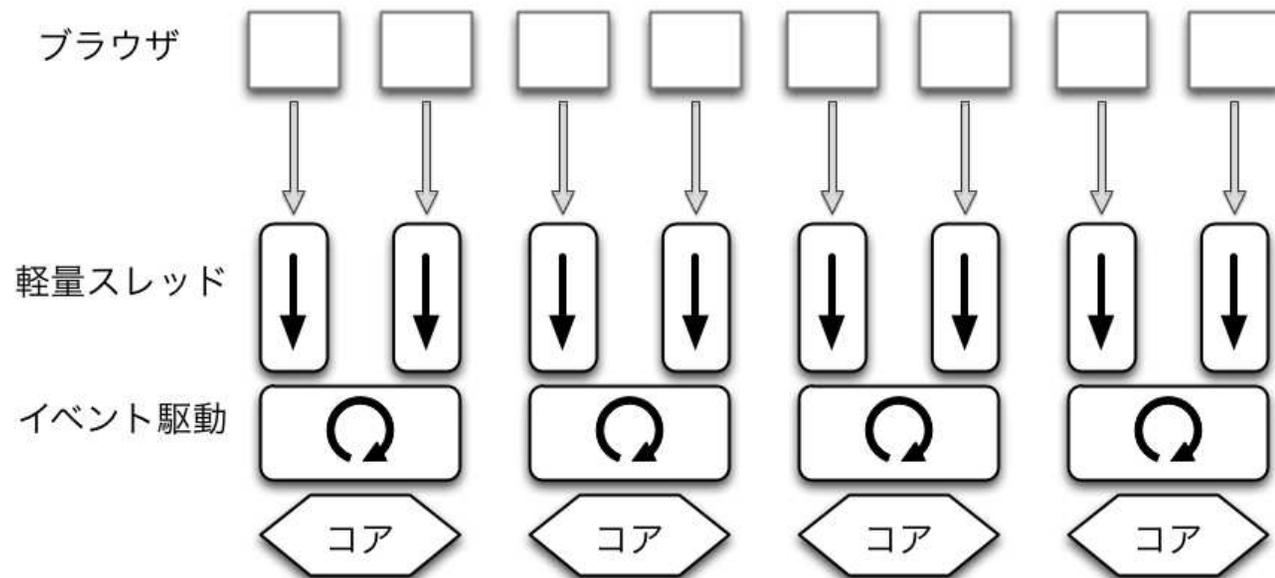
- 利点
 - コア当たりの性能が出せる
- 欠点
 - コードの見通しが悪い
 - マルチコアを活用できない

1コア1プロセス・マッピング



- 利点
 - コア当たりの性能が出せる
 - マルチコアを活用できる
- 欠点
 - コードの見通しが悪い

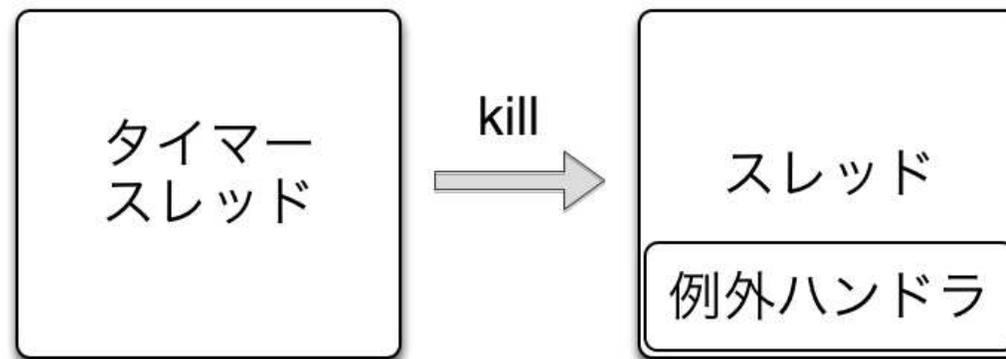
軽量スレッド



- 利点
 - コア当たりの性能が出せる
 - マルチコアを活用できる
 - コードの見通しがよい

非同期例外

- 他のスレッドの行為によって起きる例外
- Haskell では、非同期例外を捕捉できる



- Timeout 関数

```
timeout :: Int -> IO a -> IO (Maybe a)
```

```
recv :: Socket -> Int -> IO String
```

```
timeout 1000000 (recv sock 4096)
```

```
  :: IO (Maybe String)
```

どうやって高速化したか？

システムコールの低減

システムコールはすべての
軽量スレッドを止める

特化と再計算の回避

Haskell のライブラリが
汎用的過ぎて遅い

ロックの低減

ロックはプログラミング
の敵

システムコールの低減

Warp のアーキテクチャ

- システムコールをなるべく使わない
 - システムコールは、すべての軽量スレッドを止める



システムコールのトレース

■ strace

```
accept4(8, {sa_family=AF_INET,...  
recvfrom(9, "GET / HTTP/1.1\r\n...  
sendto(9, "HTTP/1.1 200 OK\r\n...  
sendfile(9, 10, [0], 151)           = 151  
recvfrom(11, "GET / HTTP/1.1\r\n...  
sendto(11, "HTTP/1.1 200 OK\r\n...  
sendfile(11, 10, [0], 151)        = 151
```

■ 不要なシステムコールの発見

- 連続する stat()
- futex() など

■ 知らないシステムコールの発見

- accept4()
 - O_NONBLOCK をセットできる accept()
 - fcntl() を 2 回減らせる

ベンチマークツール

ApacheBench

おススメしない
不安定
select() ベース

httperf

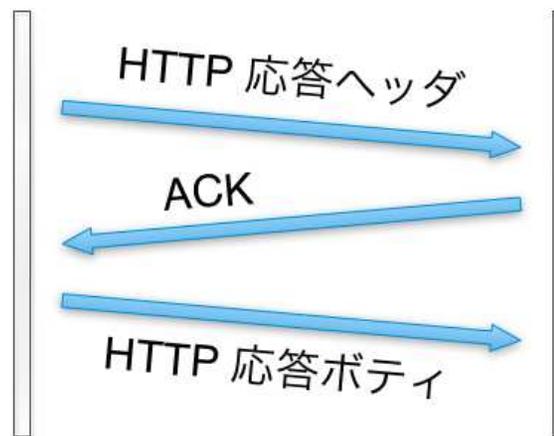
昔のおススメ
まずいメモリ管理
select() ベース

weighttp

今のおススメ
ネイティブスレッドを増やせる
epoll ベース

パケット・ダンプ

- ベンチマークツールでコネクション数を1にするとスループットが悪くなることに気付いた
- tcpdump で観察



HTTP 応答ヘッダとボディ

- 昔の送信方法

 - `writenv(応答ヘッダ1, 応答ヘッダ2, ...)`
 - `sendfile(応答ボディ)`

- 今の送信方法

 - `応答ヘッダ <- compose(応答ヘッダ1, 応答ヘッダ2, ...)`
 - `send(応答ヘッダ, MSG_MORE)`
 - `sendfile(応答ボディ)`

 - `setsockopt(TCP_CORK)` は遅かった

特化と再計算の回避

Glasgow Haskell Compiler

コンパイラー

```
% ghc foo.hs  
→ foo
```

インタープリター

```
% ghci  
> 1 + 1  
2
```

スクリプト

```
% runghc foo.hs
```

GHC のプロファイリング

- コンパイル・オプション

- `-rtsopts -prof`

- 実行時オプション

- `+RTS -p`

- 例

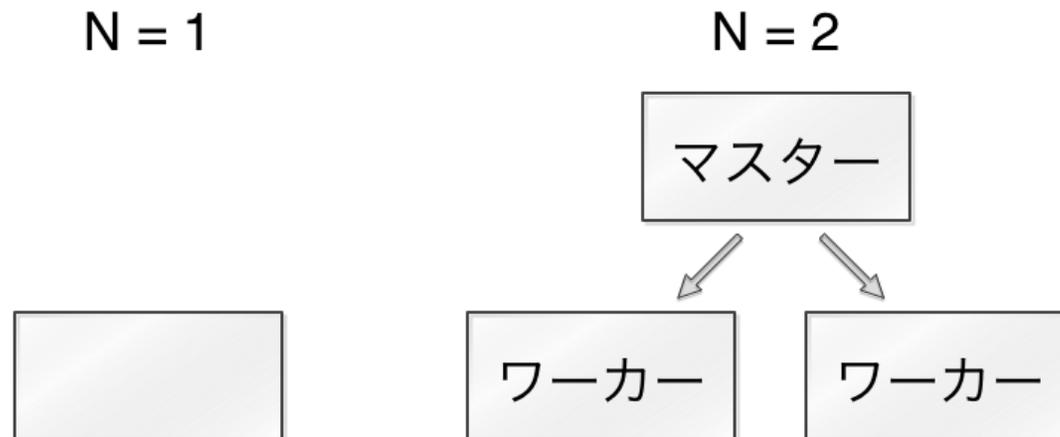
COST CENTRE	MODULE	%time
>>=	Control.Monad.Trans.Resource	8.0
sendloop	Network.Sendfile.Linux	4.1
==	Data.CaseInsensitive	2.7
parseFirst	Network.Wai.Handler...	2.6

- 制約

- フォアグラウンド・プロセスであること
- プロセス数が1であること

Mighty 側の対応

- デバッグ・オプション
 - Yes ならフォアグラウンド・プロセス
 - No ならバックグラウンド・プロセス
- ワーカー数
 - N が 2 以上なら、N 個の子プロセスを起動する
 - N が 1 なら、子プロセスを起動しない



- nginx は N = 1 でも子プロセスを起動

衝撃の事実

日付文字列の作成が遅い
30～40% の CPU 時間を消費

日付文字列

- 標準は Data.Time

- 非効率なリスト・プログラミング
- 汎用的過ぎる

- http-date ライブラリ

- 29 文字、GMT に特化したライブラリ

Date: Wed, 24 Oct 2012 07:31:18 GMT

Last-Modified: Wed, 01 Sep 2010 08:51:36 GMT

- unix-time ライブラリ

- ロギングのためにタイムゾーンを考慮する
- strftime() へのラッパー

192.0.2.0.1 - - [18/Oct/2012:21:11:10 +0900] ...

日付文字列のキャッシュ

- それでも日付文字列の生成は遅い
 - http-date も unix-time も速いがコストは0ではない
 - 同じ時刻に何度も同じ日付文字列を生成するのは無駄
 - gettimeofday() は遅い
- キャッシュ戦略
 - 一秒間に一回、HTTP 日付とログ日付を生成してキャッシュ
 - HTTP 応答コンポーザやロガーは、キャッシュされた日付文字列を利用

ロックの低減

ロック

- 大きなメモリの確保
 - `pthread_mutex_lock()` が使われている
- MVar
 - Haskell でのプリミティブなロック
 - 同期通信の仕組み

GHC eventlog

- 時刻とイベント名の記録

- コード例

```
create :: Int
        -> (Ptr Word8 -> IO ())
        -> IO ByteString

create l f = do
    traceEventIO "before mallocByteString"
    fp <- mallocByteString l
    traceEventIO "after mallocByteString"
    withForeignPtr fp $ \p -> f p
    return $! PS fp 0 l
```

- コンパイル・オプション

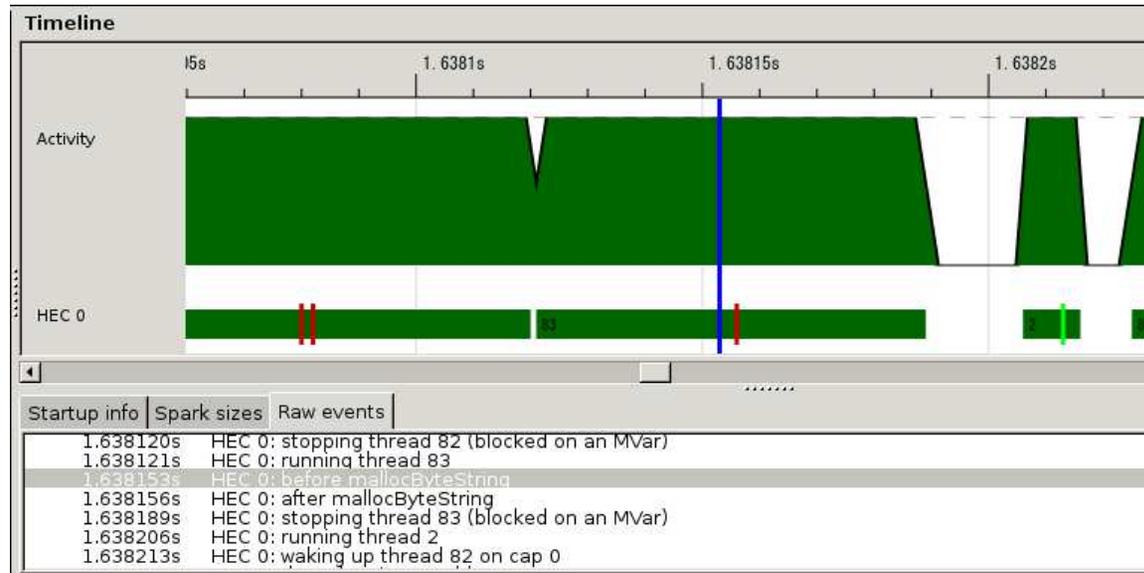
- `-rtsopts -eventlog`

- 実行時オプション

- `+RTS -ls`

ThreadScope

- eventlog を可視化するツール



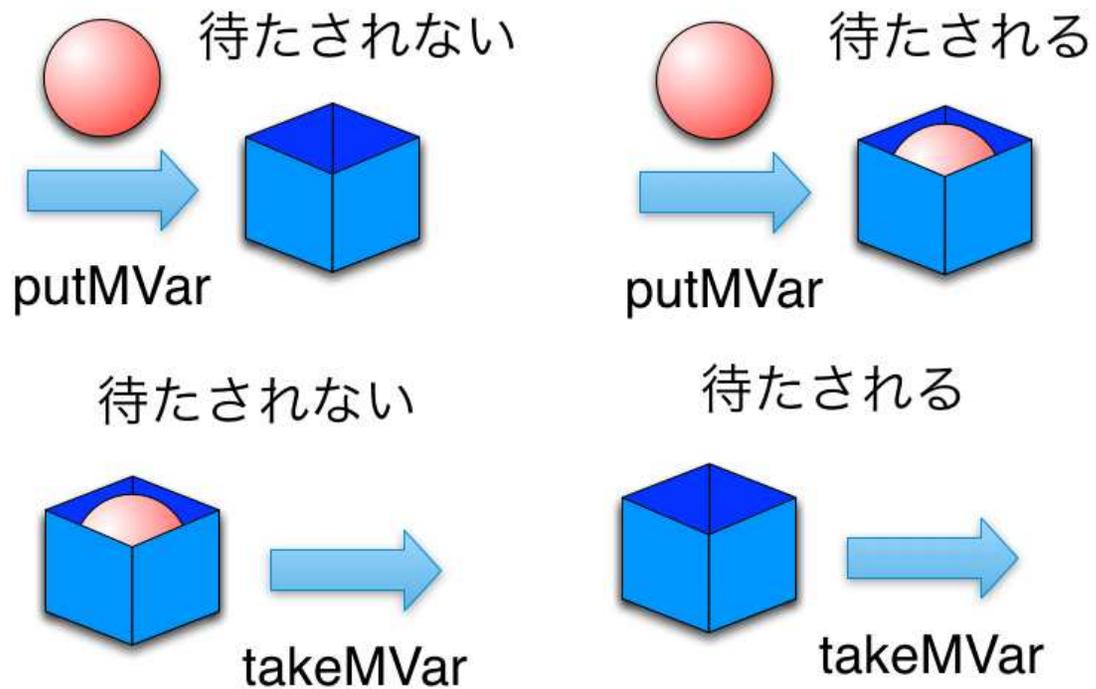
- pthread_mutex_lock() がセッションの 1/10 を消費

Timeout

- slowloris 攻撃への対策
 - 一定時間、適量の通信が発生しなければコネクションを切る
- Warp の timeout の歴史
 - timeout で実装したがスケールしなかった
 - MVar で実装したが遅かった
 - 二重の IORef で解決

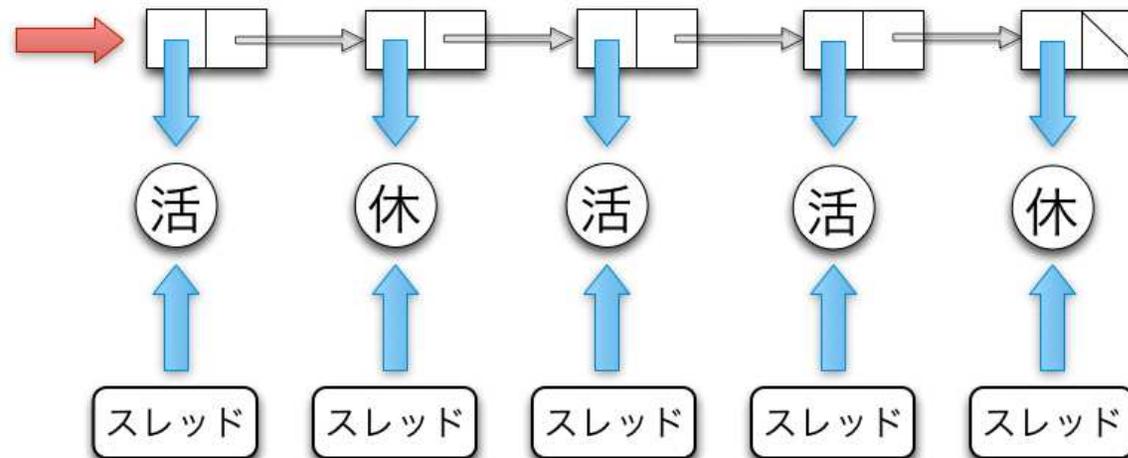
MVar

- 同期通信の仕組み
 - GHC 自家製のスピロックで実装

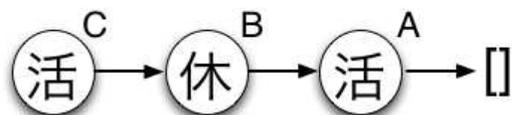


高速 timeout

- IORef
 - 破壊的代入ができる参照
- 二重 IORef
 - スレッドが定期的書き換える参照とリストへの参照を分ける
- リストの変更はアトミックに
 - CAS (compare and swap)



CAS で更新

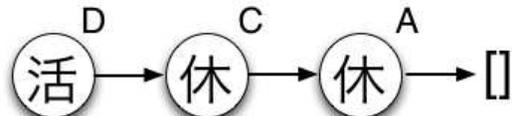


アトミックに空リストと入れ替える

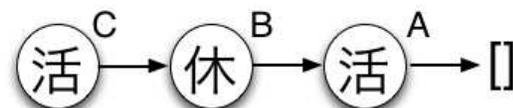
[]



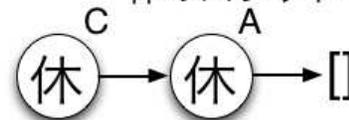
アトミックにマージ



マネージャ



活を休みへ
休のスレッドを殺す



参考文献

- 高速WebサーバMighttpdのアーキテクチャ
 - <http://www.iij.ad.jp/company/development/tech/activities/mighttpd/>
- Future work to improve the performance of Warp
 - <http://www.yesodweb.com/blog/2012/10/future-work-warp>
 - 7つの記事