SEA Forum
実用期を迎えた関数プログラミング
Haskell 編

Web アプリケーション・フレームワーク
Yesod

2012.3.5

(株) IIJ イノベーションインスティテュート
山本和彦

自己紹介

山本和彦

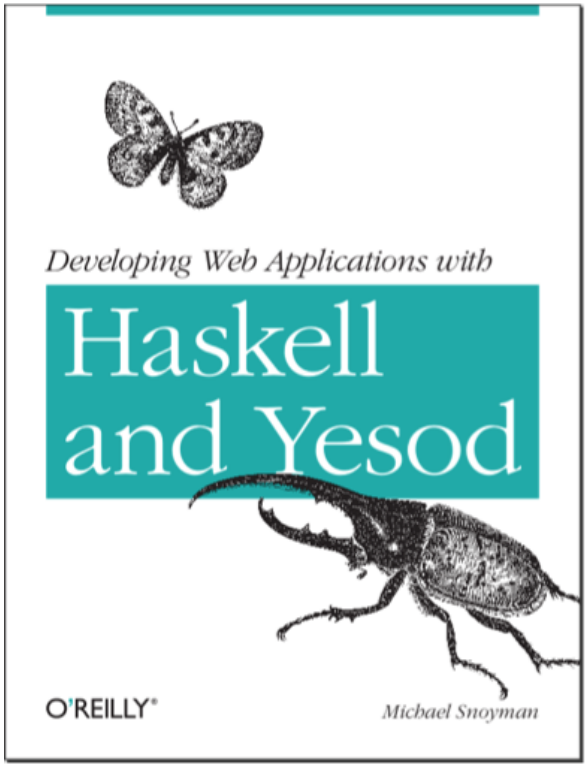
二児の父

Haskell 歴は4年と少し

Mew、 Firemacs、 Mighttpd の開発者

Yesod とは何か？

- Haskell で書かれたフルスタックの
  Web アプリケーション・フレームワーク



安全

No SQL injection
No CSRF
No XSS

高速

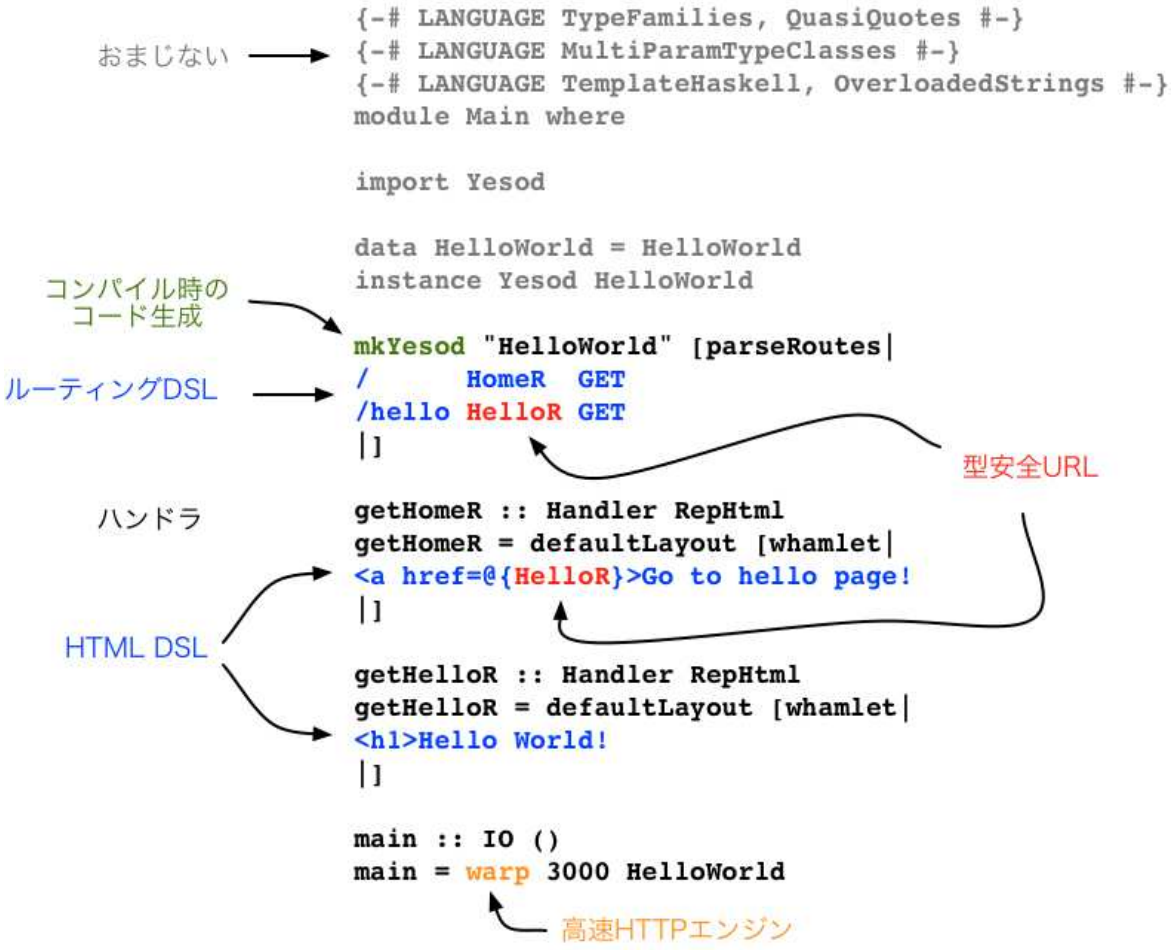
Haskell

コンパイルが通れば
だいたい思い通りにプログラムが動く

Yesod

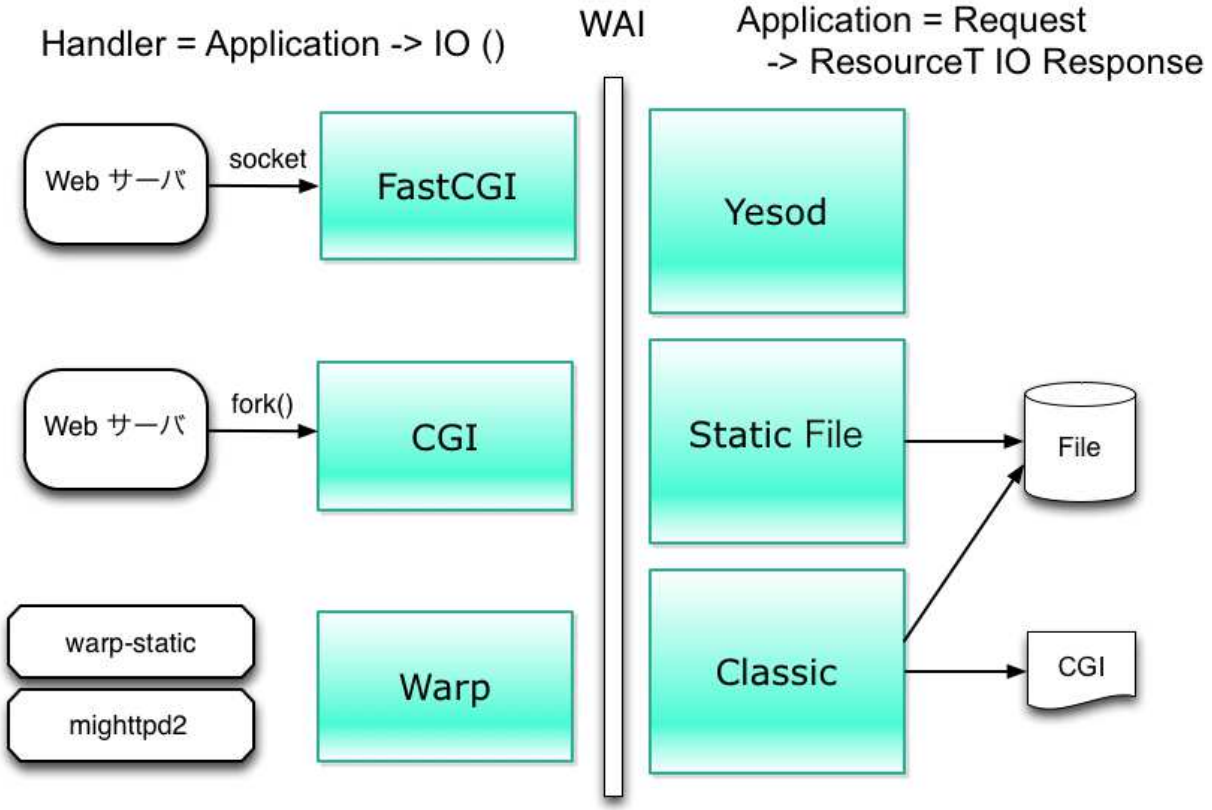
コンパイルが通れば
だいたい思い通りに Web アプリ動く

Hello, world!



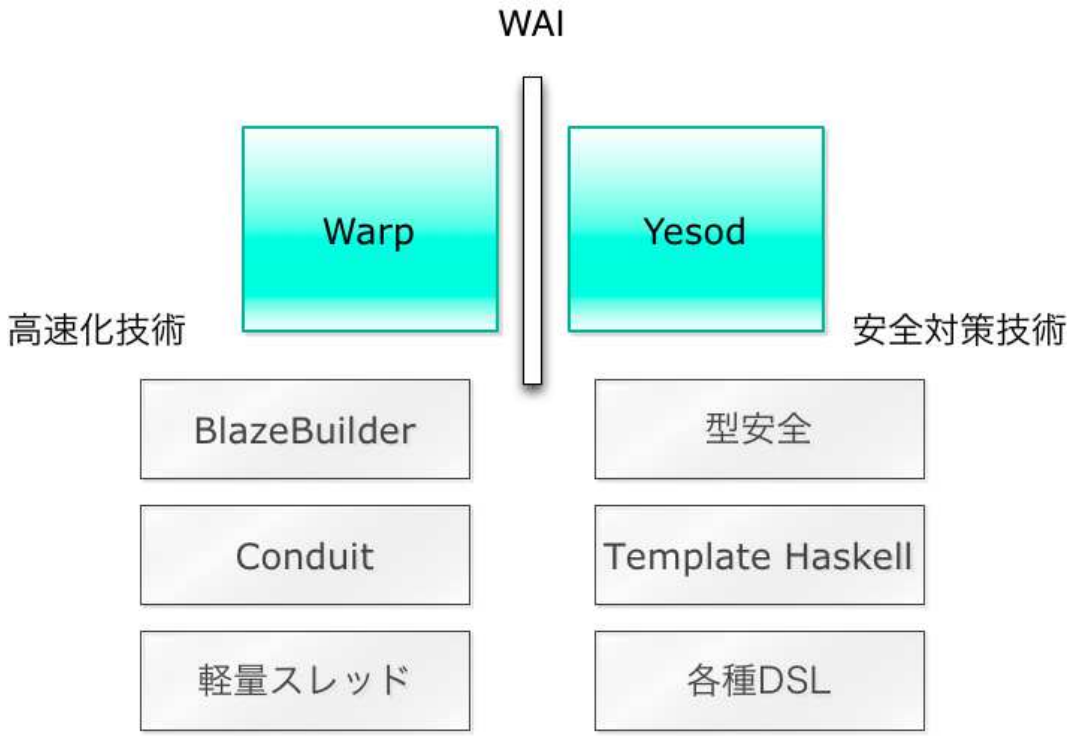

```
{-# LANGUAGE TypeFamilies, QuasiQuotes #-}
{-# LANGUAGE MultiParamTypeClasses #-}
{-# LANGUAGE TemplateHaskell, OverloadedStrings #-}
module Main where

import Yesod

data HelloWorld = HelloWorld
instance Yesod HelloWorld

mkYesod "HelloWorld" [parseRoutes|
/      HomeR  GET
/hello HelloR GET
|]

getHomeR :: Handler RepHtml
getHomeR = defaultLayout [whamlet|
<a href=@{HelloR}>Go to hello page!
|]

getHelloR :: Handler RepHtml
getHelloR = defaultLayout [whamlet|
<h1>Hello World!
|]

main :: IO ()
main = warp 3000 HelloWorld
```

WAI (Web Application Interface)



- Yesod アプリ + Warp → Web アプリ
- Yesod アプリ + FastCGI ハンドラ → FastCGI アプリ
- Classic + Warp → mighttpd2

Yesod を支える技術



安全対策技術

型安全

- ルーティング情報

```
mkYesod "HelloWorld" [parseRoutes|
/      HomeR  GET
/hello HelloR GET
|]
```

- 型の自動生成

```
data Route = HomeR | HelloR
```

  - 実際にはもっと複雑ですが...

- 型で守るURL
  - @{} は Route 型を要求する

```
getHomeR :: Handler RepHtml
getHomeR = defaultLayout [whamlet|
<a href=@{HelloR}>Go to hello page!
|]
```

- アプリ内ではリンク切れなし！

Template Haskell

- 関数型言語ではパーサの作成が簡単
  - パーサコンビネータのおかげ
  - 気軽に DSL を作れる
- 準クオート
  - "[parser|" と "|]" の間に DSL を書く

```
mkYesod "HelloWorld" [parseRoutes|
/       HomeR  GET
/hello HelloR GET
|]
```

- 展開
  - コンパイル時に構文木を生成する

```
data Route = HomeR | HelloR
...
```

  - 型検査の対象となる

No No No

No SQL injection	SQL 要求では prepare を使う
No CSRF	セッションごとに 異なる nonce を使う
No XSS	型安全なエスケープ処理

似て非なるものの区別

- Echo アプリ

```
mkYesod "HelloWorld" [parseRoutes|
/echo/#Text EchoR GET
|]

getEchoR :: Text -> Handler RepHtml
getEchoR = defaultLayout . toWidget . echoHtml

echoHtml :: Text -> a -> Html
echoHtml txt = [hamlet|<h1>#{txt}|]
```

- URL

```
%3Cscript%3Ealert%28%22Danger
%21%22%29%3B%3C%2Fscript%3E
```

- Text

```
<script>alert("Danger!");</script>
```

- Html

```
&lt;script&gt;alert(&quot;Danger!&quot;);
&lt;/script&gt;
```

各種DSL

- Hamlet (HTML, Html 型)

```
<h1 .page-title>#{pageTitle}
<p>This is the home page of...
```

- Cassius (CSS, Css 型)

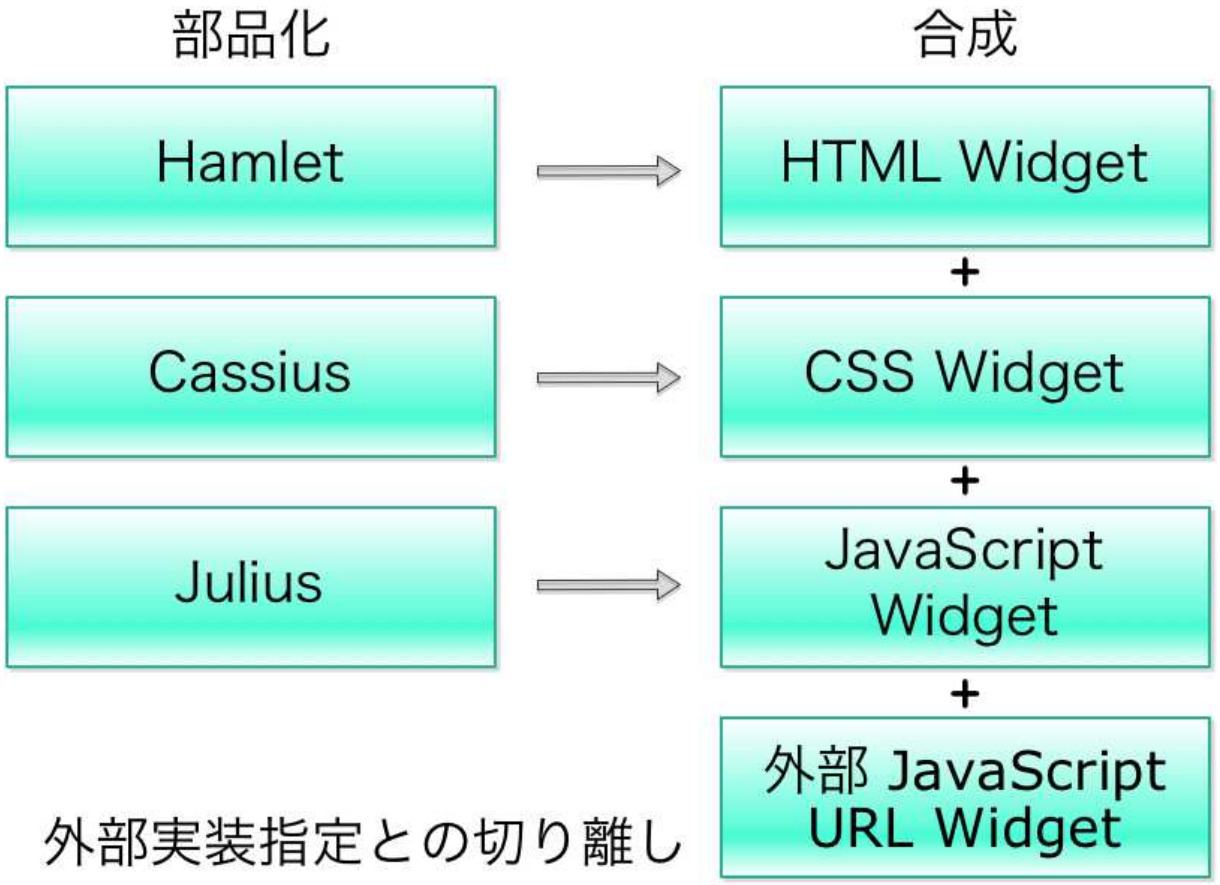
```
#myid
    color: #{red}
    font-size: #{bodyFontSize}
```

- Julius (JavaScript, Javascript 型)

```
$(function(){
    $("section.#{sectionClass}").hide();
    $("#mybutton").click(function(){
        document.location = "@{SomeRouteR}";
    });
    ^{addBling}
});
```

- 型を考慮して変数を埋め込める(interpolate)

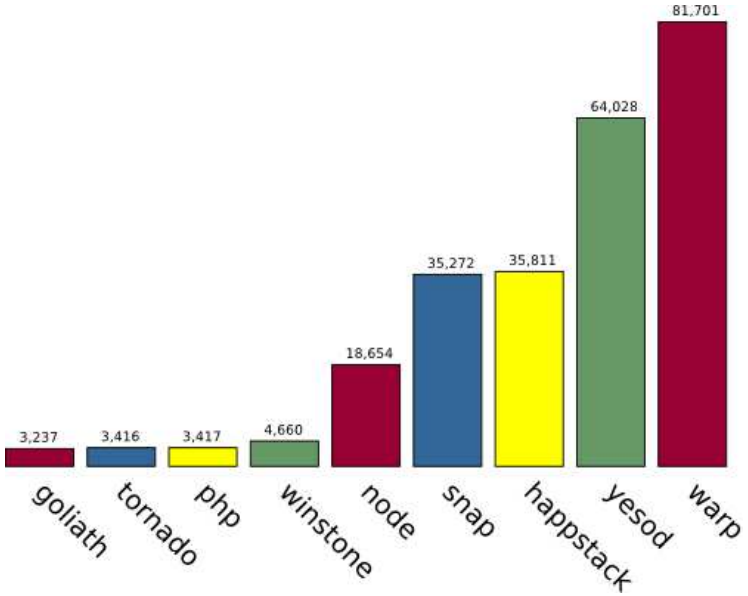
Widget


部品化
合成
Hamlet
HTML Widget
+
Cassius
CSS Widget
+
Julius
JavaScript
Widget
+
外部 JavaScript
URL Widget
外部実装指定との切り離し

高速化技術

Yesod/Warp の速度

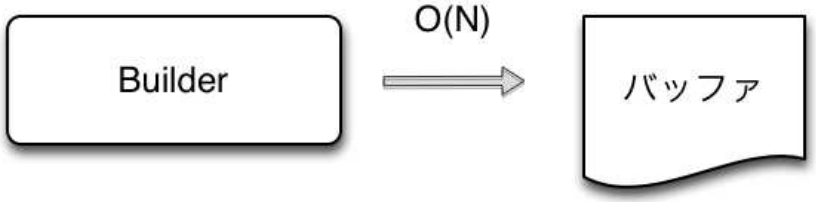
- Ping pong ベンチマーク



高速文字列処理

- BlazeBuiler
  - 差分リストのアイディアに基づいた
    高速文字列連結ライブラリ
- 文字列連結
  - Haskell の文字列は変更できないので、安全に共有できる
  - 関数合成は O(1)
  - どう関数合成しても右から左へ関数が評価される

- 出力
  - 出力の際に、はじめて出力バッファにコピーされる



Conduit とは何か？

- Unix のパイプに似ている

生産者と消費者の分離 独立に実装できる	合成可能 固定長のバッファを利用
生産者の抽象化 ファイル、ソケット、リスト...	消費者の抽象化 ファイル、ソケット、リスト...
生産者での資源管理 ファイル記述子...	消費者での資源管理 ファイル記述子...

- 第二世代の Iteratee

合成可能とは何か？

- 悪しき一枚岩による実装

```
replicate :: Int -> a -> [a]
replicate 0 _ = []
replicate n x = x : replicate (n - 1) x
```

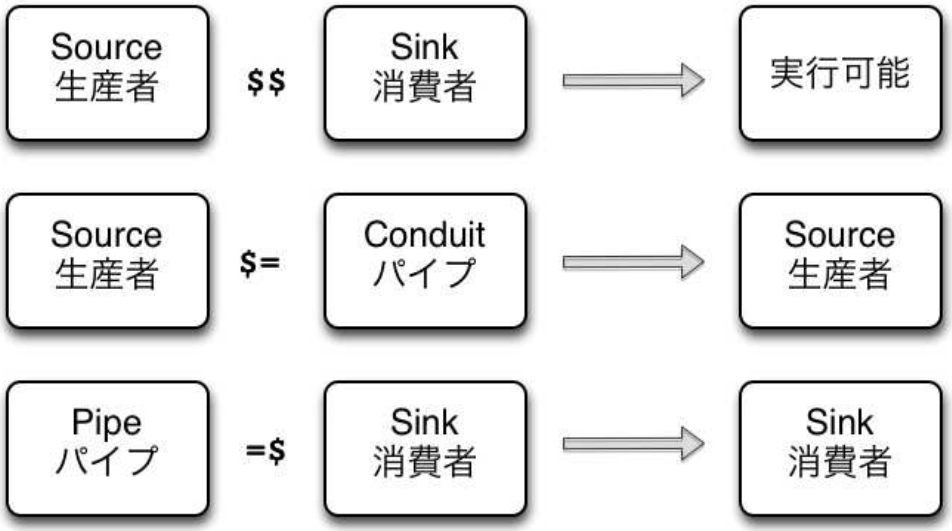
- 合成による実装

```
replicate n x = take n $ repeat x
```

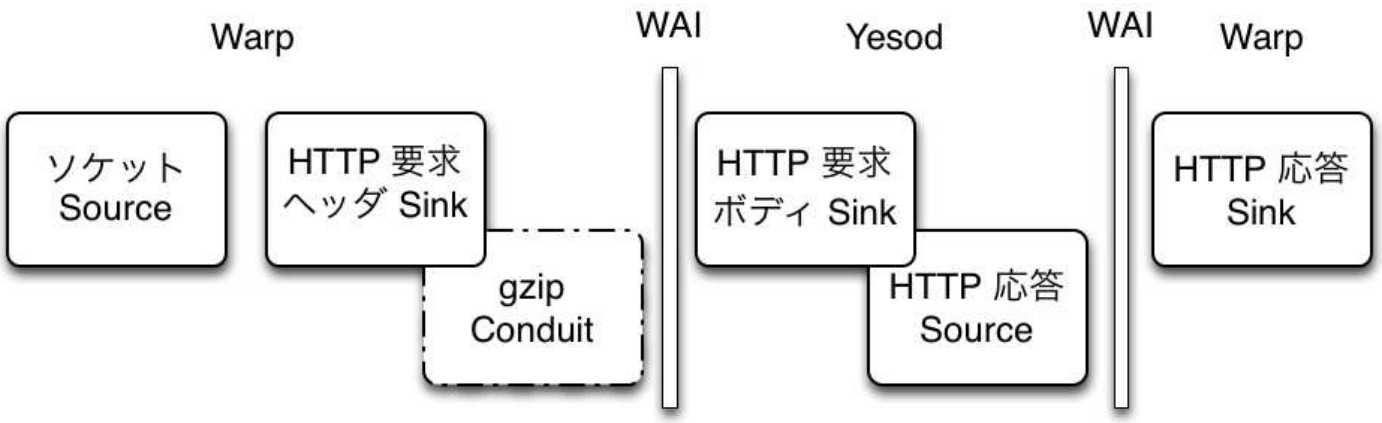
- Conduit でのファイルのコピー

```
copyFile :: FilePath -> FilePath -> IO ()
copyFile src dst = runResourceT $
    CB.sourceFile src $$ CB.sinkFile dst
```

Conduit での合成

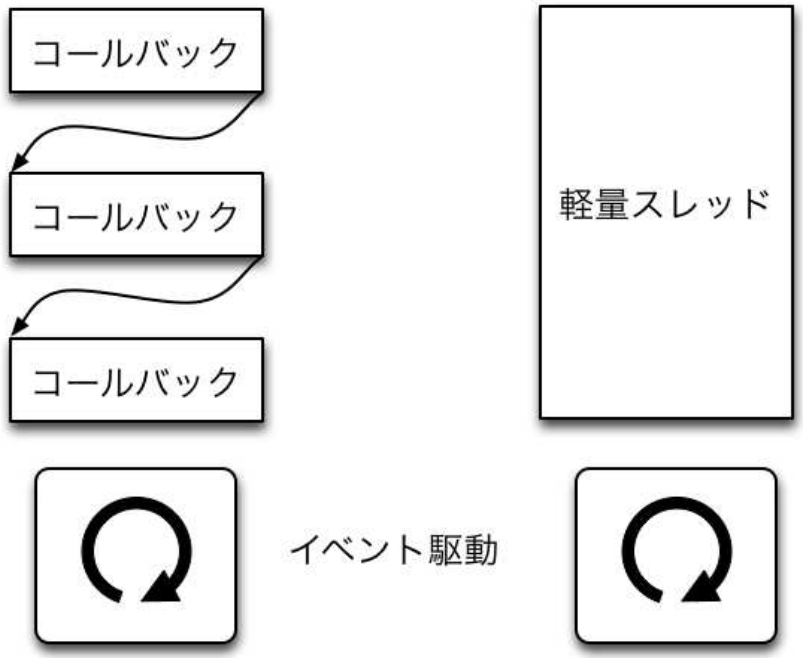


Warp と Conduit



高速サーバ

- 高速化にはイベント駆動が必須
  - プロセス・プール、ネイティブ・スレッド・プールでは遅い
- 多くの言語ではコールバック (イベント・ハンドラ)
- Haskell では軽量スレッド

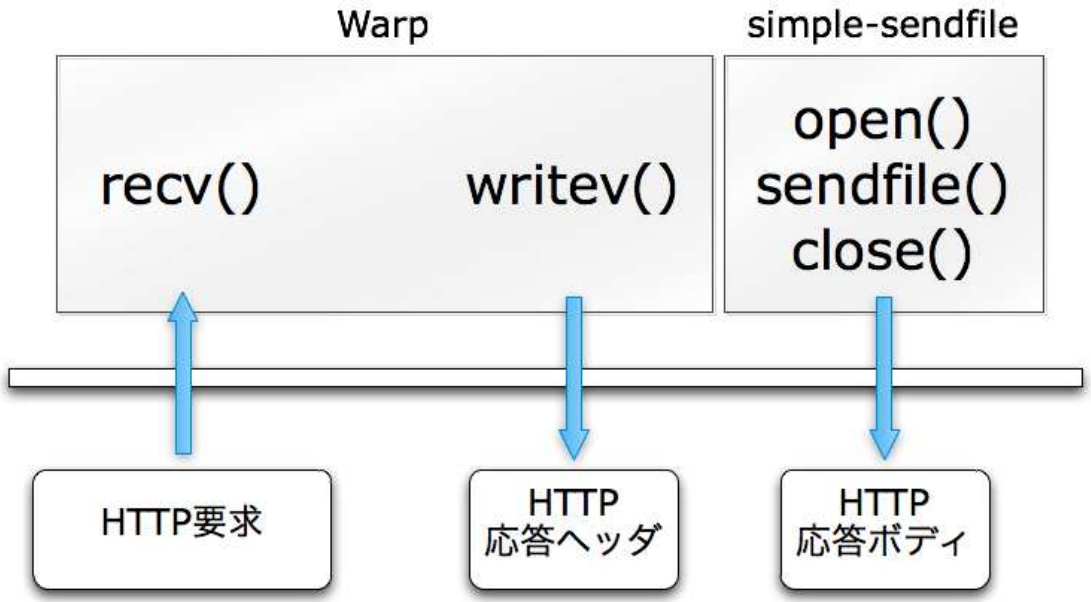


軽量スレッド

- ユーザ空間で実装されたスレッド
  - ネイティブ・スレッドとは異なる
- 軽量
  - 10万個作成しても大丈夫
  - 軽量スレッドの切り替え時には、
    カーネルへのコンテキスト・スイッチが発生しない
- Haskell に特有の機能
  - ライブラリはノンブロッキングとなるよう実装されている
  - 外部言語呼び出しでブロックしても大丈夫
    - 外部言語呼び出しにはネイティブ・スレッドを使う
  - 非同期例外
    - ある軽量スレッドが、他の軽量スレッドを kill できる
    - ほとんどの計算が純粋なので、突然例外を受け取ってもよい

Warp が発行するシステムコール

- 軽量スレッドの敵はシステムコール
  - カーネルにコンテキストスイッチして、すべての軽量スレッドが止まる
- なるべくシステムコールは発行しない
  - stat() で得た情報などはキャッシュすべき



参考文献

- Yesod のサイト
  - http://www.yesodweb.com/
- Yesod のチュートリアル
  - http://yannesposito.com/Scratch/en/blog/Yesod-tutorial-for-newbies/
- Haskell and Yesod
  - オライリーから近日発売
- Mighttpd のサイト
  - http://mew.org/~kazu/proj/mighttpd/en/
  - Monad.Reader へ投稿した記事へのリンクあり