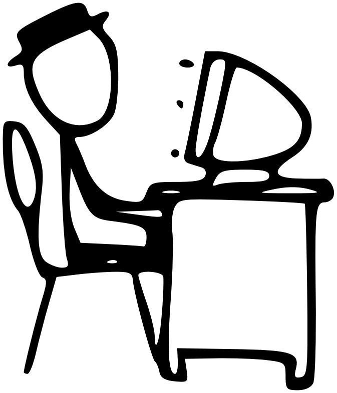
BlazeHtml
Blazingly fast HTML combinators

Jasper Van der Jeugt

March 18, 2011

Hello!

My name is Jasper
Student at UGent
I write Haskell
GhentFPG organizer
`@jaspervdj`
`jaspervdj.be`



Overview

Introduction
Why Haskell?
Haskell web frameworks
Case study: BlazeHtml

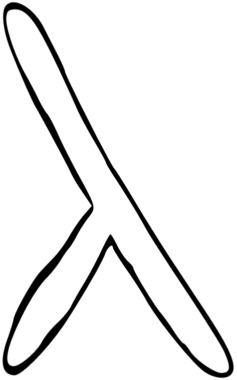
Overview

Web development: languages used

PHP
Ruby
Python

Haskell has an edge

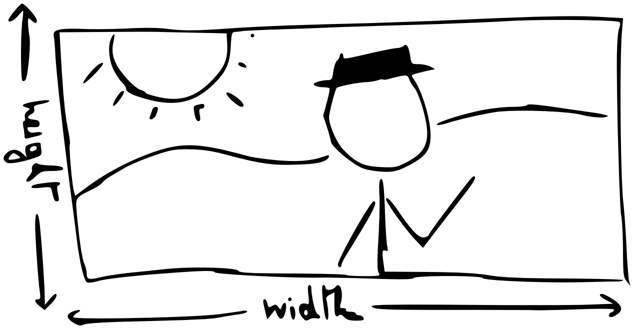
Type-safe
Stateless
Compiled
Highly scalable



Type safety

Is this function error-prone?

```
makeImage ::
      Int -> Int -> Image
```



Type safety

Can prevent many errors

```
newtype Width = Width Int
newtype Height = Height Int
makeImage ::
    Width -> Height -> Image
```

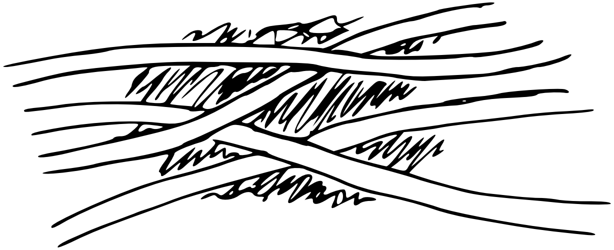
Pure code

Explicit separation of pure and impure code

Pure	Impure
Heavens	Earth
"Functional"	"Imperative"

Impure code can call pure code, but not vice versa

Parallelism & concurrency



Synchronization primitives: `MVar a`
Semi-explicit parallelism
High-performance event manager

Overview

General web programming

Something like

```
app :: Request -> Response
```

Or rather

```
app :: Request -> IO Response
```

Routes

Web framework provides routing, e.g.

```
route
  [ ("",              root)
  , ("user/:id",    user)
  , ("tweet/:id", tweet)
  ]
```

Monadic handlers

Implementation of handlers:

```
user = do
  id' <- getParam "id"
  user <- getFromDataBase id'
  -- Perform pure operations
  save user
  set reponse
```

WAI

Web **A**pplication **I**nterface
Connects server backend to application
Similar to Rack (Ruby)

Happstack

Has been around since 2003
Very mature
Complete stack
Yet flexible

Yesod

Built on *WAI*
Very high-level
Tightly integrated components
Focus on DSL's

Snap Framework

Relatively new

Sensible and clean

Fast and highly concurrent

Aims to be a complete framework

Warp

A very fast web server
Handles 190k req/s
Simple (500 loc)
Backend for *WAI*

Overview

Webapp architecture

HTML generation
Web application server
Data storage layer

BlazeHtml

Embedded in Haskell
Efficient Unicode support
Supports HTML 4 and HTML 5
Pretty fast
Google Summer of Code 2010
By Simon Meier & me

Example

```
head $ do
    title "Title"
body $ do
    div ! class_ "fancy" $ do
        "Literal"
    div ! id "info" $ do
        p "Content..."
        p "More..."
```

Syntactic sugar: do

`do` notation works for every Monad

```
do user <- getUser
   cookie <- getCookie
   touch cookie
   check user cookie
```

Syntactic sugar: do

Translates into plain code using >>, >>= and `return` operators

```
user >>= \user ->
 getCookie >>= \cookie ->
  touch cookie >>
   check user cookie
```

Syntactic sugar: do

This is an incredibly powerful feature for DSL's in Haskell!

Coined as:

The programmable semicolon

Syntactic sugar: strings

`OverloadedStrings` allows you to use literal strings wherever you want

E.g. what is the type of

```
"Hello world"
```

Syntactic sugar: strings

REGEX

ROUTE

```
"Hello world"
```

BINARY

PATTERN

HTML

Multiple renderers

`String` (= `[Char]`)
`ByteString` (UTF-8? Latin-1?)
`Text` (Lazy or Strict?)

Lazy evaluation

```
let x = sum [1 .. 20000]
in if var then x else y
```

Lazy evaluation

`x` is calculated when...

```
if x > 5 then ...
print x
seq x y
...
```

Lazy renderers

Keep more than strictly necessary

```
data StaticString = StaticString
    { string :: String -> String
    , utf8   :: S.ByteString
    , utf16  :: Text
    }
```

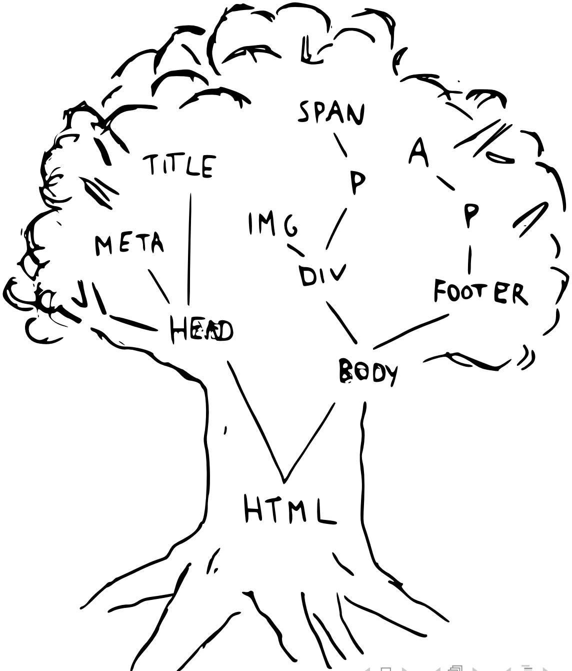
Build your own abstractions

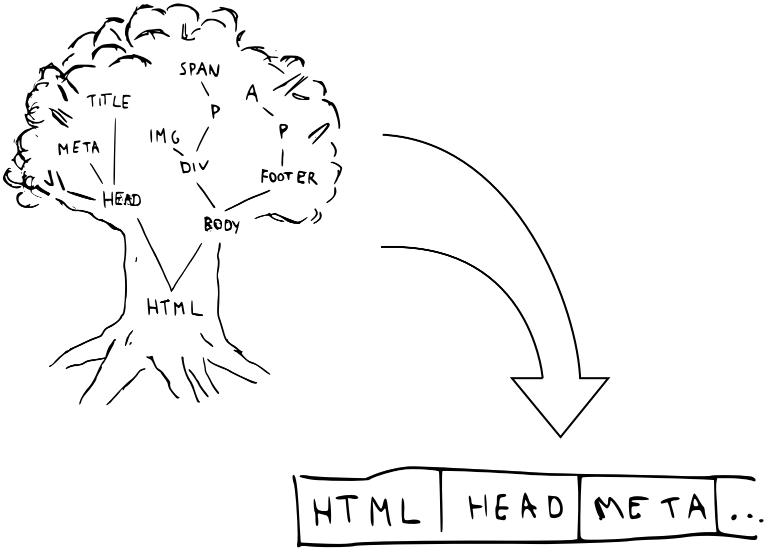
```
includeJS source =
  script ! type_ "text/javascript"
         ! src source
         $ mempty

includeJS "jquery.min.js"
```

HTML representation

```haskell
data Tree a
    = Node a (Tree a) (Tree a)
    | Empty
```

A simple, immutable data structure


SPAN
A
TITLE
P
P
IMG
META
DIV
FOOTER
HEAD
BODY
HTML



SPAN
A
TITLE
P
P
IMG
META
DIV
FOOTER
HEAD
BODY
HTML
HTML
HEAD
META
...

Concatenating

StringBuilder:

```
builder.append(someString)
```

Builder monoid

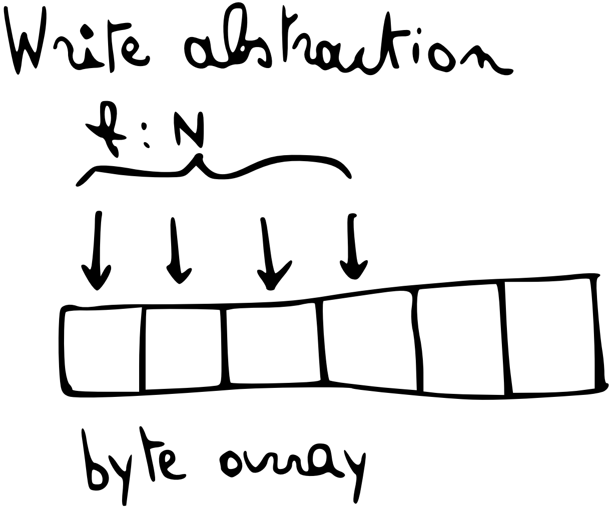
```
builder1 `mappend` builder2
```

Builder Monoid

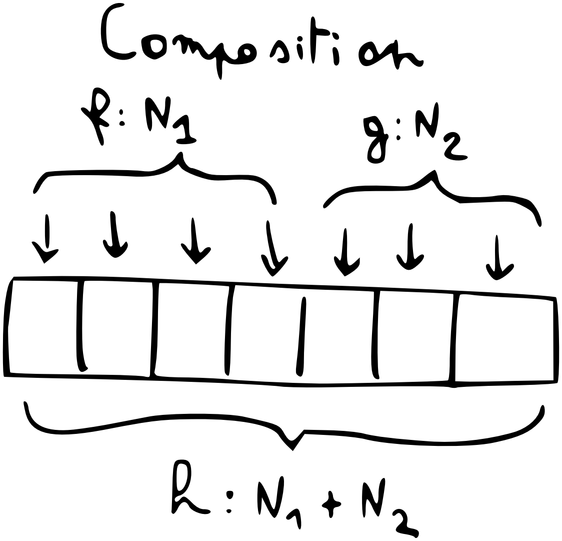
Simple interface:

```
mempty  ::
    Builder

mappend  ::
    Builder -> Builder -> Builder
```

Implementation: Write



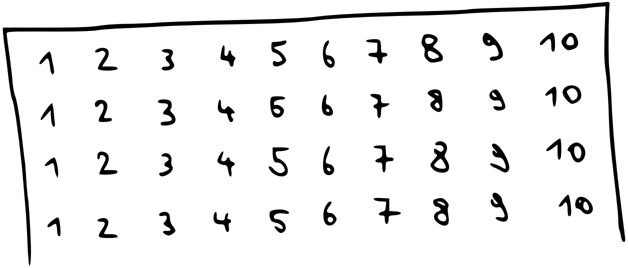
Implementation: Write



Implementation: tricky

$$\begin{tikzcd}
\text{extract data} \arrow[rr, bend right, "\text{Fresh buffers}"'] & & \text{Builder} \arrow[ll, bend right, "\text{Signals}"']
\end{tikzcd}$$

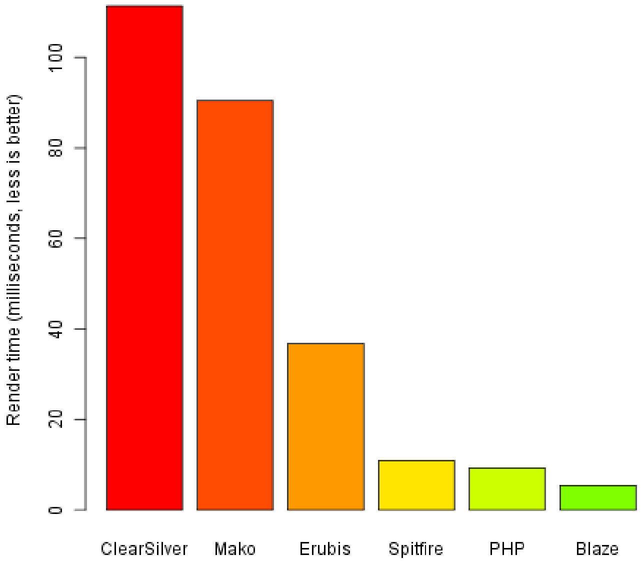
BigTable benchmark



1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

```
bigTable = table $
  replcateM_ 1000 $ tr $
    forM_ [1 .. 10] $ \c -> td $
      toHtml c
```

BigTable benchmark



Migrating

```
blaze-from-html
   -s -e -v html5
```

Questions?

`jaspervdj.be/blaze`