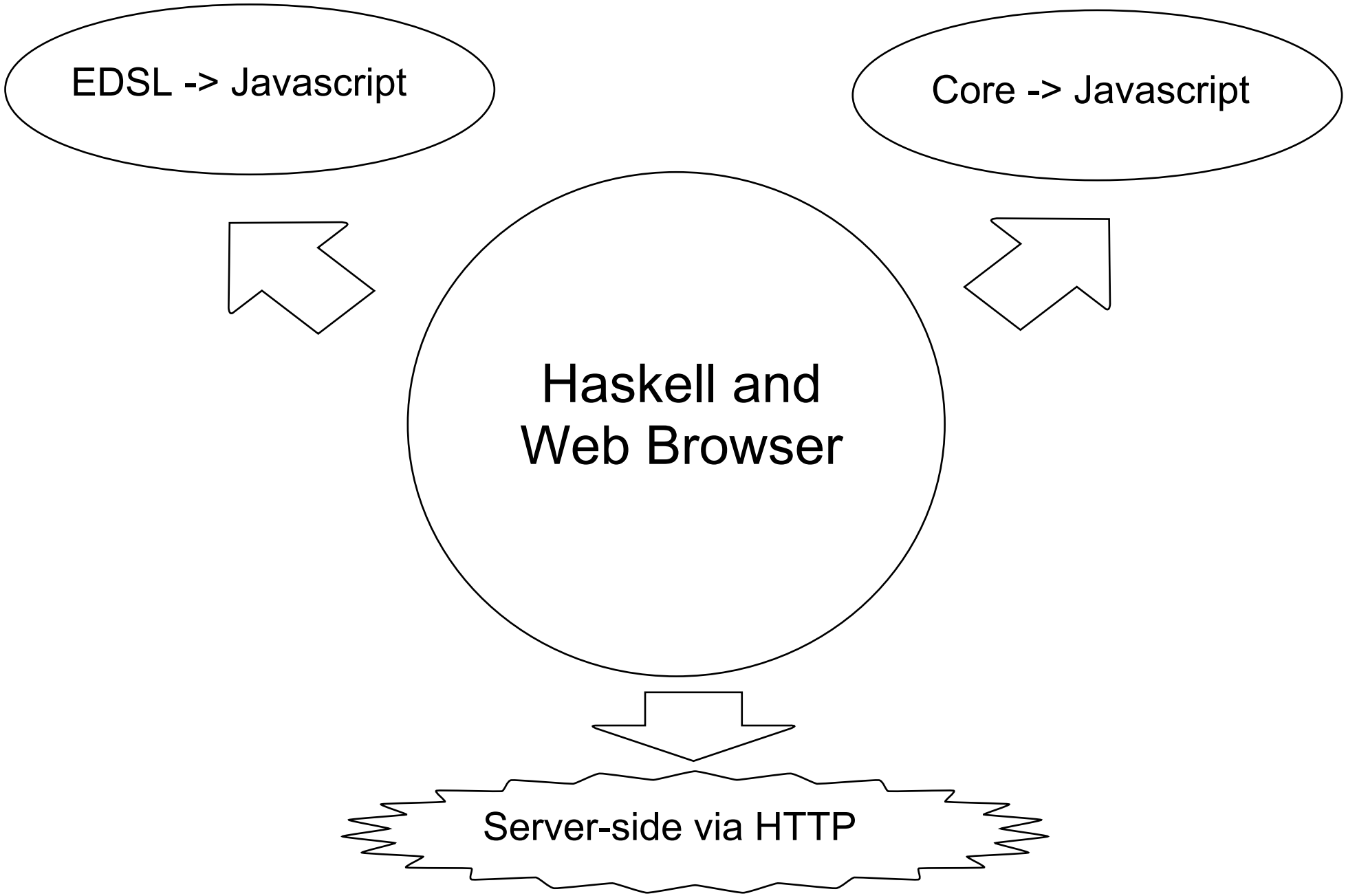
Haskell in Web Browser

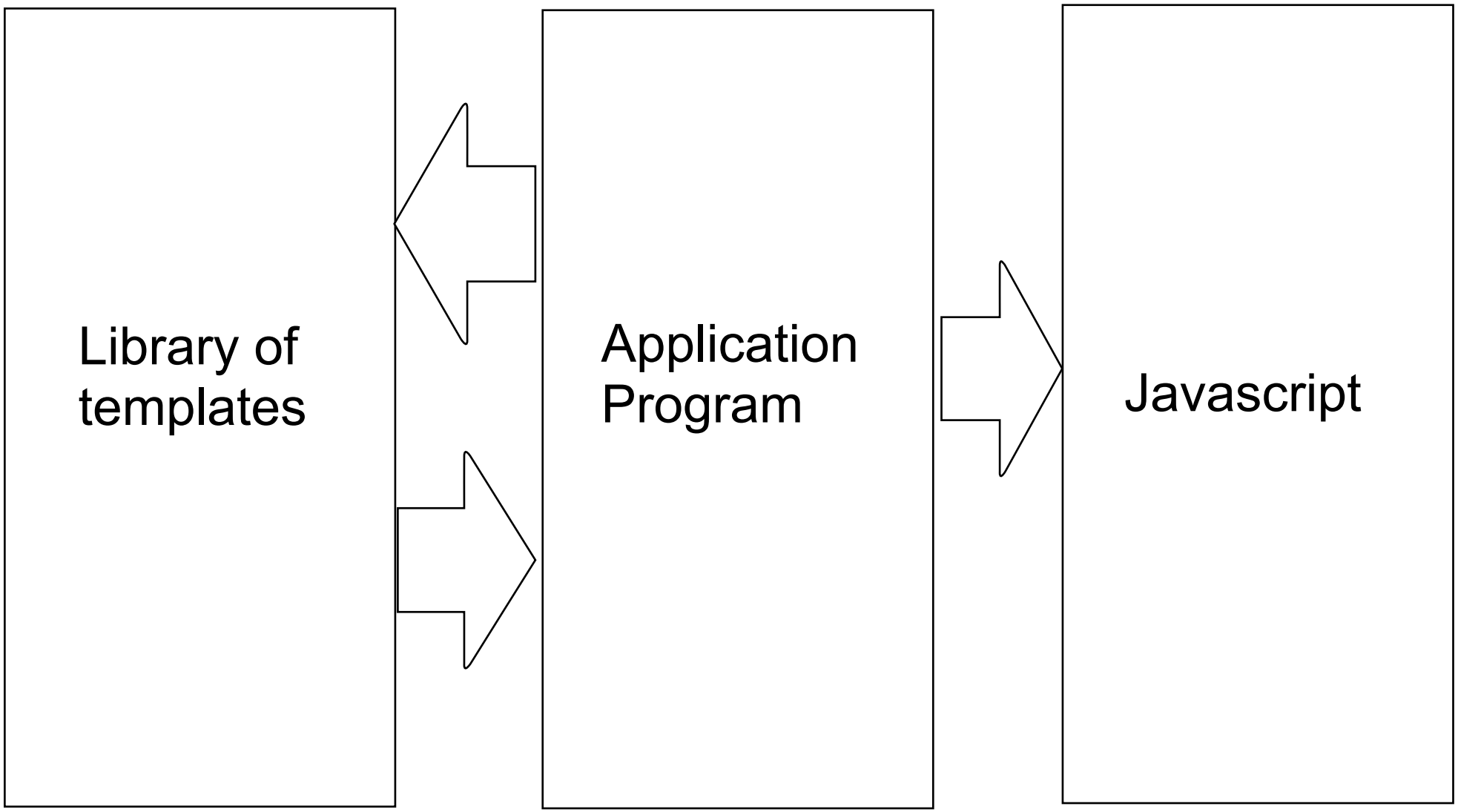
Presented at

Hac φ

by Dmitry Golubovsky
July 25, 2009
Philadelphia, PA


EDSL -> Javascript
Core -> Javascript
Haskell and
Web Browser
Server-side via HTTP

EDSL



Known EDSLs for Javascript

- HJScript [Broberg, Bjornson]
- JSMW [Golubovsky]
- FRP-JS [Visser]

JSMW: Javascript Monadic Writer

An EDSL inspired in part by HJ(ava)Script and HSP aimed at coding in typed Javascript. It uses WebBits as the underlying representation of Javascript.

```
q = do
    d <- htmlDocument
    t <- mkText (string "Hello World") d
    b <- getm'body d
    addChild t b

function main()
{
  window.document.body.appendChild
    (window.document.createTextNode("Hello World"));
}
```

WebBits

- Internal representation of Javascript syntax
- Pretty printer
- A very elegant way to attach type information to Javascript expressions (the *`a'* variable in the code below)

```
data Expression a
  = StringLit a String
  | DotRef a (Expression a) (Id a)
  | CallExpr a (Expression a) [Expression a]
```

WebBits (cont'd)

To encode a method call: **`this.method(arg)`**:

```
CallExpr t (DotRef t this (Id t "method"))
  [arg :: Expression t]
```

The expression above has type: **`Expression t`**

The type of *`t'* defines the type of the whole expression.

Types in JSMW

```
q = do
    d <- htmlDocument
    t <- mkText (string "Hello World") d
    b <- getm'body d
    addChild t b
```

M :: some monad
d :: *D* (type of a HTMLDocument node expression)
b :: *B (*type of a <body> tag node expression)
t :: *T* (type of a text node expression)
q :: *M Q* (type of the toplevel expression)
mkText :: Expression String -> *D* -> *M T*
getm'body :: *D* -> *M B*
addChild :: *T* -> *B* -> *M Q*

Expression Type Cast

Type of the method call expression: the method return type.

We want the method's argument to have an arbitrary type.

```
castExpr :: (Functor x) => b -> x a -> x b
castExpr b e = fmap (const b) e

(/\) :: (Functor x) => x a -> b -> x b
(/\) = flip castExpr
```

To change type of an expression (value remains untouched):

`(e :: Expression a) /\ t` has type **`Expression t`**

Smart Constructors

```
createElement ::
   (Monad mn, CDocument this, CElement zz) =>
  Expression String -> Expression this -> mn (Expression
zz)

createElement a thisp
  = do let et = undefined :: zz
       let r = DotRef et (thisp /\ et)
                         (Id et "createElement")
       return (CallExpr et r [a /\ et])
```

A smart constructor builds a Javascript expression taking care of proper types of method's arguments and return value. The type argument is never evaluated: using of **`undefined`** is safe.

The code above encodes the DOM method:

`document.createElement(tagName)`

Web IDL

- A language derived from OMG IDL
- Provides detailed definition of interfaces used by W3C specifications, e.g. DOM, HTML, CSS
- "Intended ... to provide precise conformance requirements for ECMAScript and Java bindings of such interfaces" *

Web IDL can be used to create Haskell bindings to W3C interfaces as well!

---

* The Web IDL Working Draft: http://www.w3.org/TR/WebIDL/

Haskell and Web IDL

Early work: OMG(ish) IDL parser provided by Haskell Direct.

Used in the Yhc/Javascript experiment to generate bindings to DOM specification, Level 2.

Work in progress: Parser, pretty printer, and JSMW-style backend based on the recent Web IDL specifications. Capable of parsing IDL specifications currently published by the Web Consortium.

Web IDL Example

```
module dom {
...
  interface Document : Node {
...
    readonly attribute DocumentType doctype;
...
    Element createElement(in DOMString tagName)
      raises(DOMException);
...
  };
};
```

The example above shows an interface (Document) inheriting from another (Node) with one read-only attribute and one operation (method).

Web IDL and Haskell Type System

Each Web IDL Interface is reflected by a Haskell class and a data type. Inheritance is reflected by type constraints.

Web IDL:	Haskell:
interface Node { ... };	class CNode a data TNode = TNode instance CNode TNode
interface Element : Node { ... };	class (CNode a) => CElement a data TElement = TElement instance CElement TElement instance CNode TElement

Passing Object References

Web IDL:

```
Node appendChild(in Node newChild)
   raises(DOMException);
```

Haskell:

```
appendChild ::
   (Monad mn, CNode this, CNode newChild, CNode zz) =>
      Expression newChild -> Expression this -> mn (Expression zz)
```

newChild can be a TNode, TElement, THTMLDivElement, whatever is an instance of CNode (IDL: inherits from Node)

Accessing Attributes

Web IDL:

```
readonly attribute Node parentNode;
```

Haskell:

```
get'parentNode ::
   (Monad mn, CNode this, CNode zz) =>
     Expression this -> mn (Expression zz)

getm'parentNode ::
   (Monad mn, CNode this) =>
     Expression this -> mn (Expression TNode)
```

Useful Shortcuts

Shortcut (maker) functions defined for HTML tag nodes: all implemented around **`document.createElement`**, but provide proper type for the resulting expression.

```
mkDiv :: (Monad mn, CHTMLDocument a)
  => Expression a -> mn (Expression THTMLDivElement)

mkButton :: (Monad mn, CHTMLDocument a)
  => Expression a -> mn (Expression THTMLButtonElement)

mkImg :: (Monad mn, CHTMLDocument a) =>
  Expression a -> mn (Expression THTMLImageElement)

mkText :: (Monad mn, CDocument this) =>
  Expression String -> Expression this ->
    mn (Expression TText)
```

A Live Example

A program displays an input field initially set to 0. Pressing "Enter" increments the value; pressing "Shift-Enter" decrements the value.

Page: http://code.haskell.org/yc2js/examples/ex1.html

Source: http://code.haskell.org/yc2js/examples/ex1.hs

Conclusion

Other uses of Web IDL with Haskell:

- Any EDSL may benefit, with proper backend
- Define interfaces to custom client-side libraries
- Expose server-side resources specifying interfaces with IDL

Currently the Haskell IDL tools are in the very early stage, any feedback will be useful.

QUESTIONS?
COMMENTS?