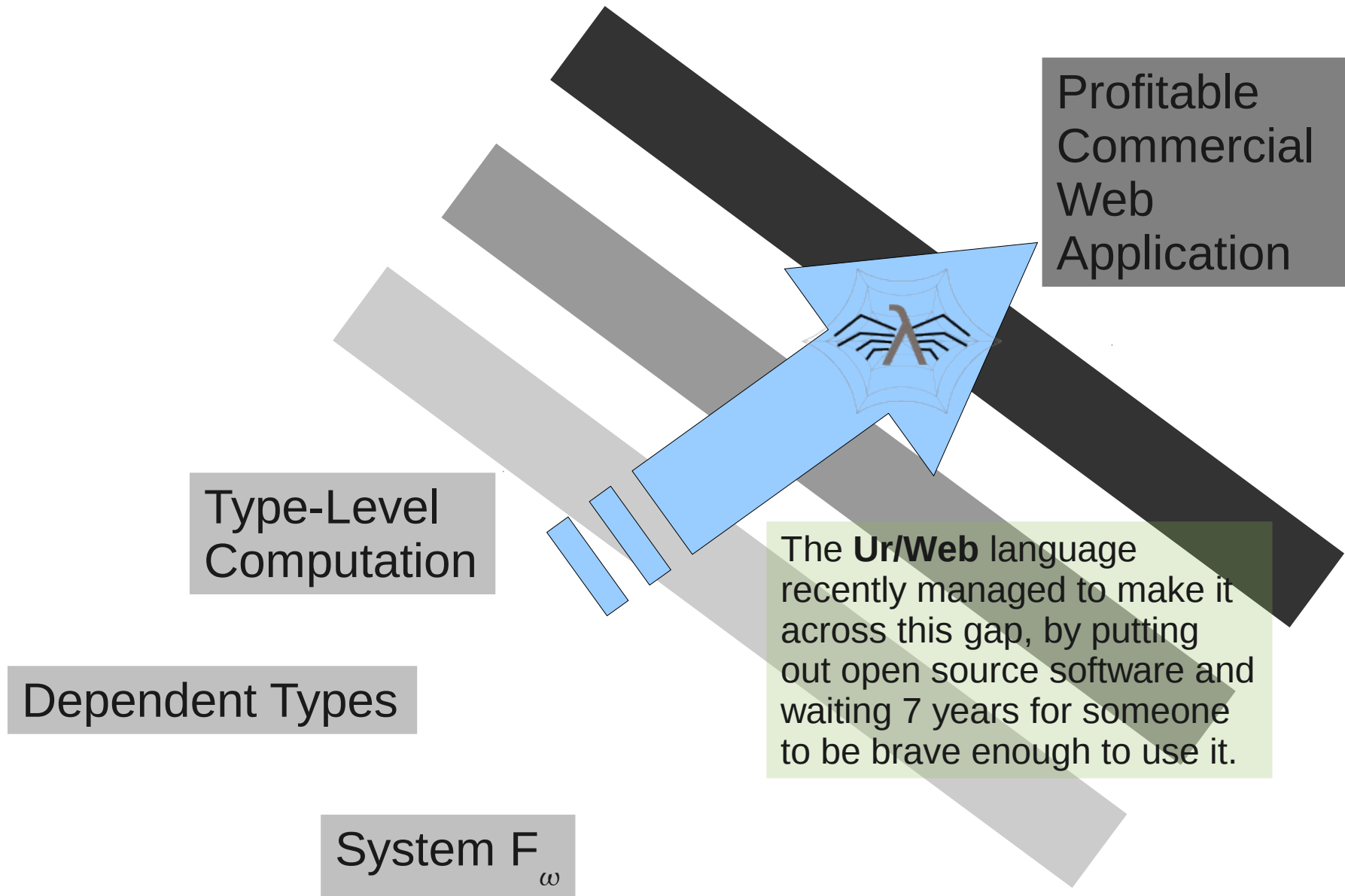


Ur/Web: Taking Syntax Seriously

Adam Chlipala
MIT

SCRIPT
November 13, 2013

A partial map of the programming languages landscape



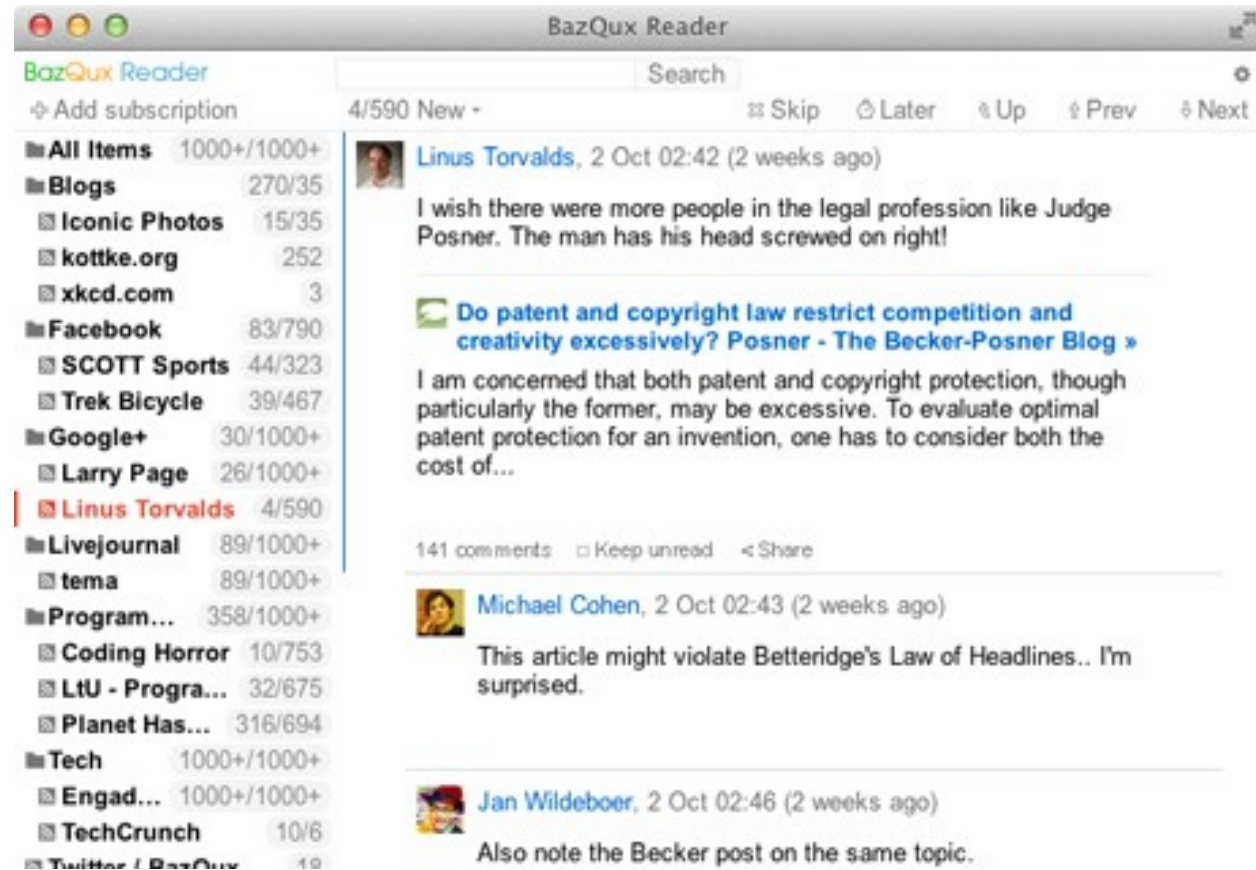
There are at least several users whom I have never met!

*The first commercial
Ur/Web application:
BazQux Reader, by
Vladimir Shabanov*

Feed reader with
comments

<http://www.bazqux.com/>

On the order of 1000
paying users daily



The screenshot shows the BazQux Reader application window. The title bar reads "BazQux Reader". The interface is split into two main sections. On the left is a sidebar with a list of subscriptions, each with a category icon and a count of items. On the right is the main feed area, which includes a search bar, navigation controls, and a list of articles with user comments.

Subscription	Count
All Items	1000+/1000+
Blogs	270/35
Iconic Photos	15/35
kottke.org	252
xkcd.com	3
Facebook	83/790
SCOTT Sports	44/323
Trek Bicycle	39/467
Google+	30/1000+
Larry Page	26/1000+
Linus Torvalds	4/590
LiveJournal	89/1000+
tema	89/1000+
Program...	358/1000+
Coding Horror	10/753
LtU - Progra...	32/675
Planet Has...	316/694
Tech	1000+/1000+
Engad...	1000+/1000+
TechCrunch	10/6
Twitter / BazQux	10

The main feed area shows a comment by Linus Torvalds on October 2nd at 02:42 (2 weeks ago): "I wish there were more people in the legal profession like Judge Posner. The man has his head screwed on right!". Below this is a link to an article titled "Do patent and copyright law restrict competition and creativity excessively? Posner - The Becker-Posner Blog". The article text reads: "I am concerned that both patent and copyright protection, though particularly the former, may be excessive. To evaluate optimal patent protection for an invention, one has to consider both the cost of...". There are 141 comments on this article, with options to "Keep unread" and "Share". Below the article is a comment by Michael Cohen on October 2nd at 02:43 (2 weeks ago): "This article might violate Betteridge's Law of Headlines.. I'm surprised.". At the bottom, a comment by Jan Wildeboer on October 2nd at 02:46 (2 weeks ago) says: "Also note the Becker post on the same topic."

Executive Summary

Ur/Web is a **domain-specific language**
with a **fancy static type system**
with **first-class support** for Web app architecture
including **strong encapsulation**
and **statically checked metaprogramming**
with **security by construction**

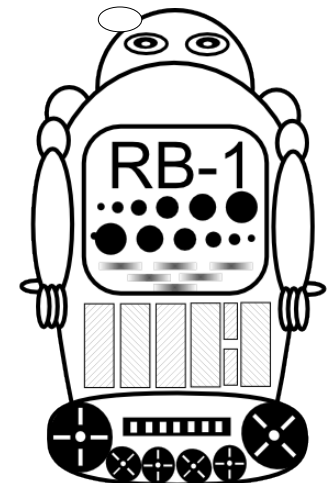
Web Applications: A Steaming Pile of Text

HTML, CSS, JavaScript,
SQL, URLs, JSON, ...



Web App Developer

Strings, strings,
strings, ...



Compiler/Interpreter

OWASP Top 10 Security Vulns.*

1. Injection
2. Cross Site Scripting
3. Broken Authentication and Session Mgmt.
4. Insecure Direct Object References
5. Cross Site Request Forgery
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards

*List is a little old at this point!

How Ur/Web Does

1. Injection

Ruled out by type system

2. Cross Site Scripting

3. Broken Authentication and Session Mgmt.

4. Insecure Direct Object References

5. Cross Site Request Forgery

6. Security Misconfiguration

7. Insecure Cryptographic Storage

8. Failure to Restrict URL Access

9. Insufficient Transport Layer Protection

10. Unvalidated Redirects and Forwards

Ruled out by static analysis, if right security policy is written
[more researchy part that won't pop up any more in this talk]

Ruled out by type system, if right security policy is written

Outline

- Big idea: embedded language syntax via fancy types
- Strong encapsulation for the web
- Statically typed metaprogramming
- Demo!

Hello World!

```
fun main () = return <xml>
  <head>
    <title>Hello world!</title>
  </head>

  <body>
    <h1>Hello world!</h1>
  </body>
</xml>
```

Syntactic sugar for building ASTs in a rich type that enforces XML well-formedness

The Big Idea

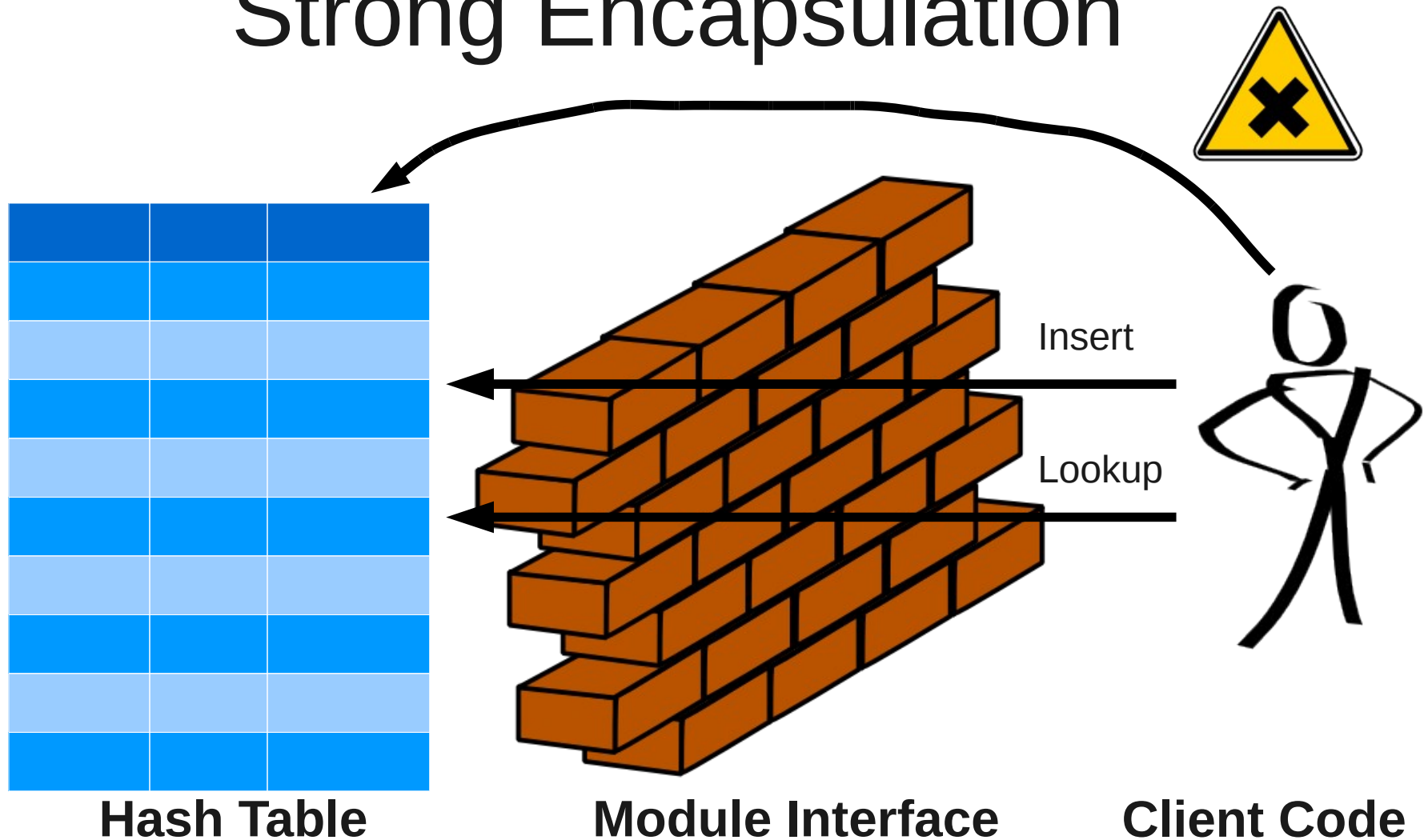
We could design a programming language with built-in static types for hash tables, red-black trees, splay trees, etc., but that's just *no fun at all*.

Expressive static type systems make it possible to design **new abstractions** without extending the programming language.

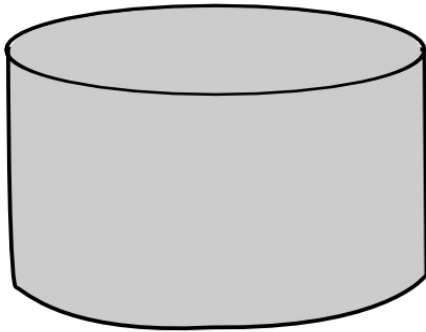
We could design a programming language with built-in static types for XML trees and SQL queries, etc., but that's just *no fun at all*.

An expressive enough static type system allows implementation of XML, SQL, and more, with **sound compile-time checking of syntax and typing rules**, without extending the programming language.

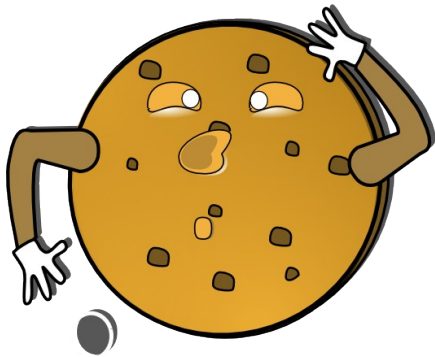
Strong Encapsulation



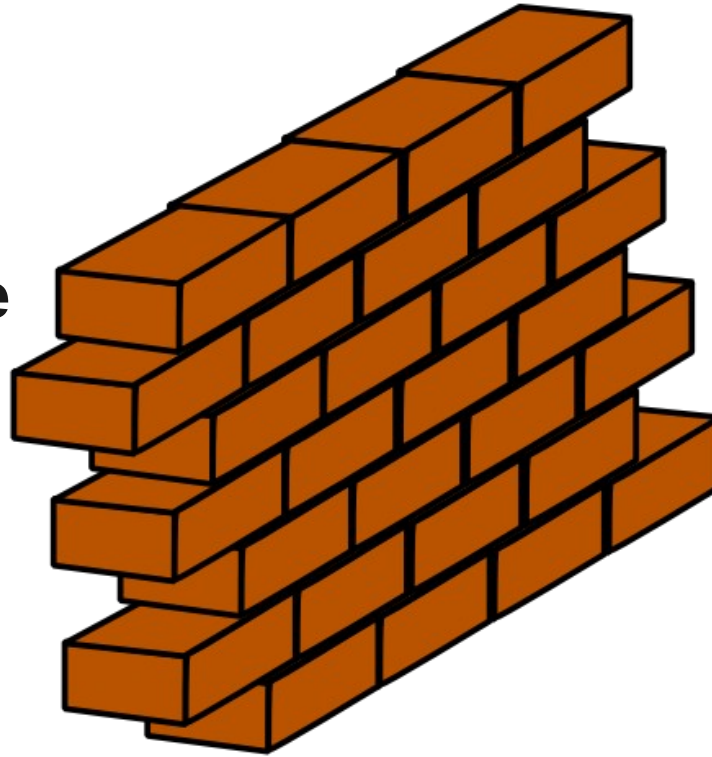
Web 1.0 Encapsulation



Users Database Table



Login Cookie



Module Interface



Client Code

Encapsulation

```
structure Auth
  : sig
    val loginForm : unit -> xbody
    val whoami : unit -> string
  end = struct
table users : {Nam : string, Pw : string}
cookie auth : {Nam : string, Pw : string}

fun rightInfo r =
  oneRow (SELECT COUNT(*) FROM users
        WHERE Nam = {[r.Nam]}
        AND Pw = {[r.Pw]}) = 1 } Syntactic sugar for
                                } construction of ASTs in a
                                } rich type enforcing query
                                } validity

fun login r = if rightInfo r then setCookie auth r
              else error "Wrong info!"

fun loginForm () = (* Form handled by 'login'... *)

fun whoami () = let r = getCookie auth in
                if rightInfo r then r.Nam
                else error "Wrong info!"

end
```

Some Client Code

```
fun main () = return <xml>
  {Auth.loginForm ()}
```

```
  <p>Welcome.  You could
  <a link={somewhere ()}>go somewhere</a>.</p>
</xml>
```

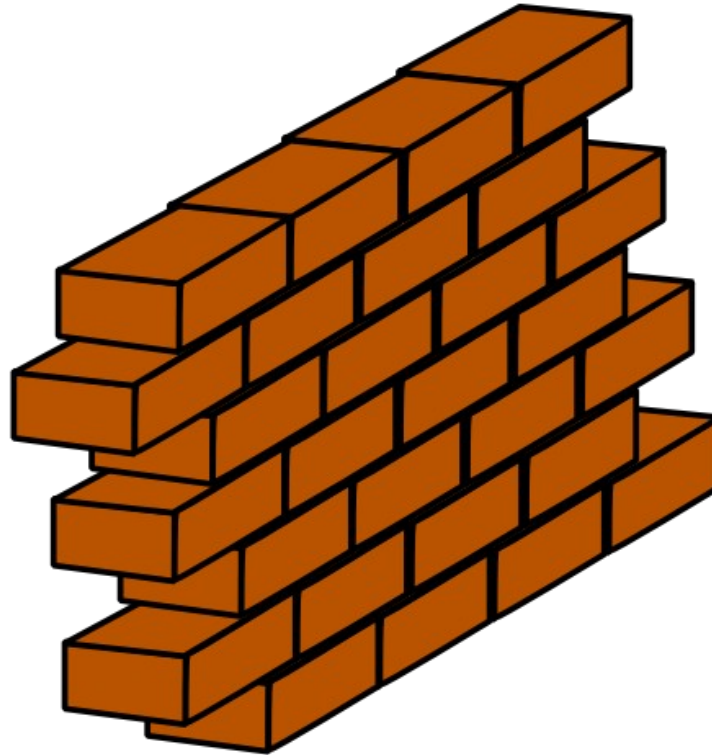
```
and somewhere () =
  user <- Auth.whoami ();
  if user = "Fred Flintstone" then
    return <xml>Yabba dabba doo!</xml>
  else
    return <xml>Boring</xml>
```

```
and whoami () =
  Row (SELECT Pw
        FROM Auth.user
        WHERE Name = 'Fred Flintstone')
```

Web 2.0 Encapsulation



**Subtree of
Dynamic Page
Structure**



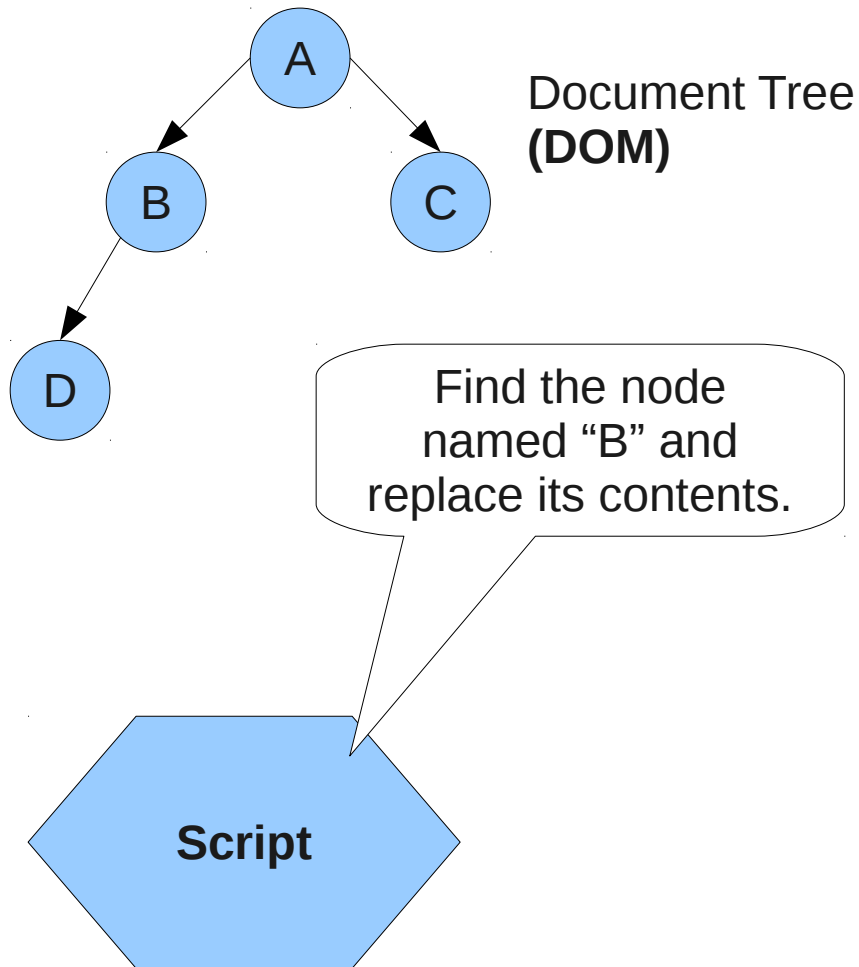
Module Interface



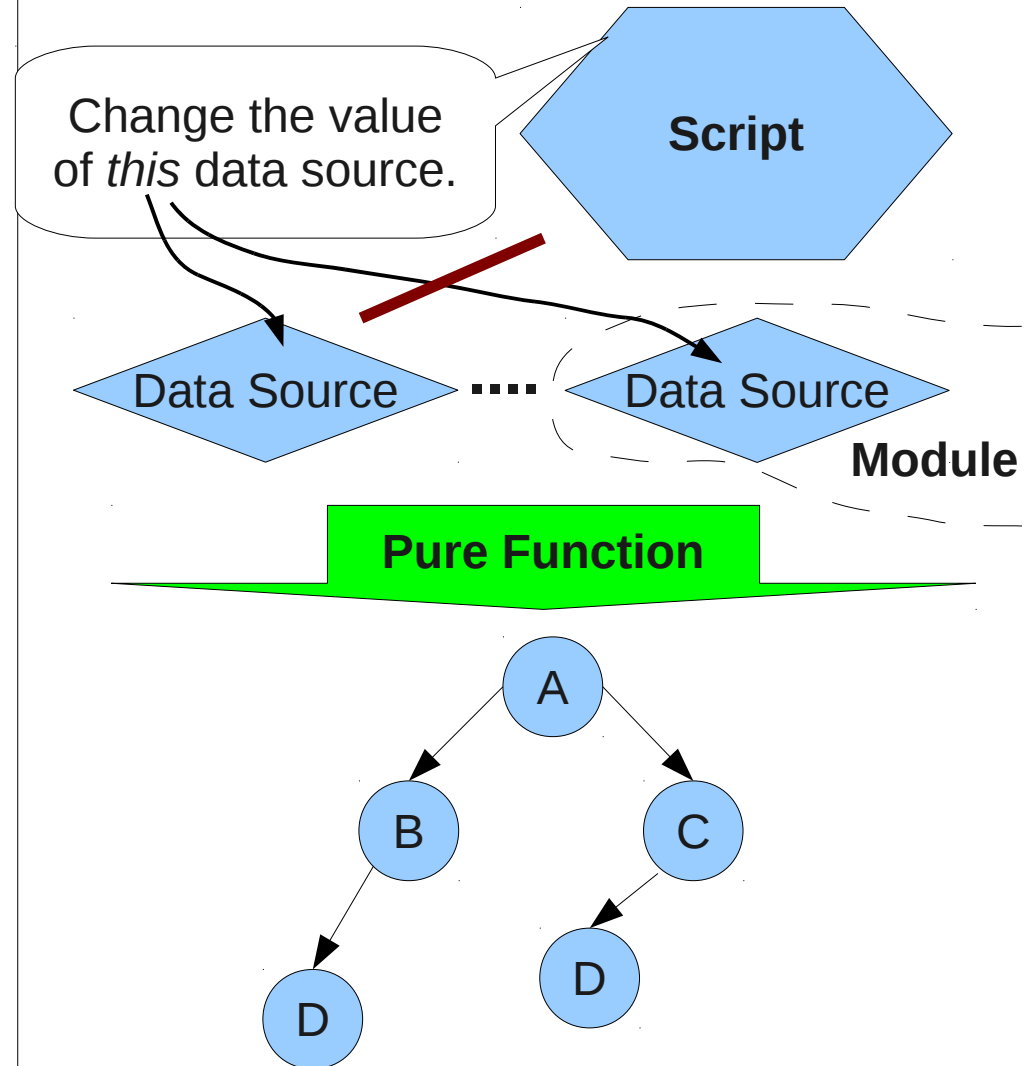
Client Code

Functional-Reactive GUIs

The Status Quo:



The Reactive Way:



A Client-Side Counter

```
structure Counter : sig
    type t
    val new : unit -> t
    val increment : t -> unit
    val render : t -> xbody
end = struct

type t = source int

fun new () = source 0

fun increment c = n <- get c; set c (n + 1)

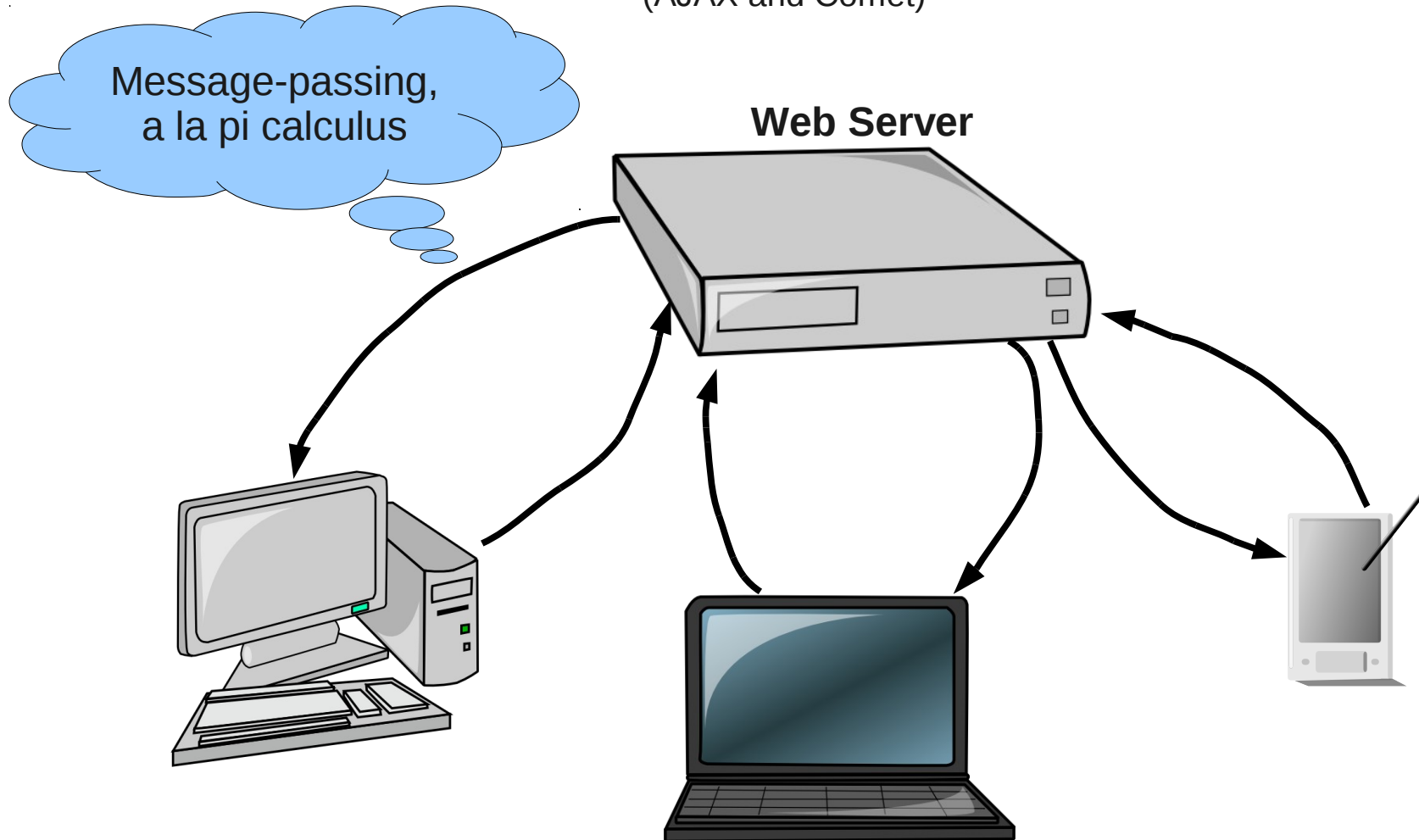
fun render c = <xml>
    <dyn signal={n <- signal c;
        return <xml><b>{[n]}</b></xml>} />
</xml>
end
```

Counter Client

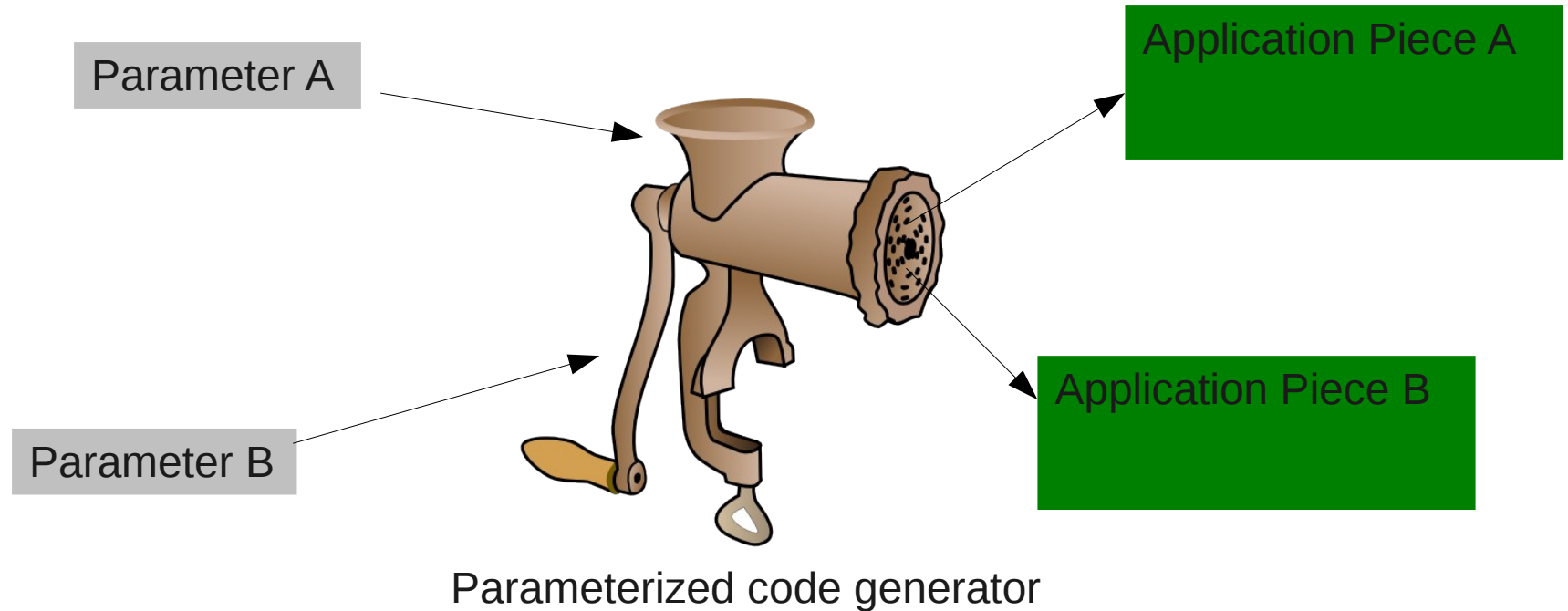
```
fun main () =  
  c <- Counter.new ();  
  return <xml>  
    {Counter.render c}  
    <button onclick={Counter.increment c}/>  
    Backup copy: {Counter.render c}  
    ton onclick={set c -1}/  
</xml>
```

Client-Server Communication

(AJAX and Comet)



Metaprogramming for the Web



Already very popular in Ruby on Rails and other frameworks!

Automatic Admin Interface

Id	A	B	C	D

SQL Table Schema

Metaprogram

Crud1

ID	A	B	C	D	
115	2	1	65	True	[Update] [Delete]
123	10	1000	100	True	[Update] [Delete]

A:

B:

C:

D:

In-Browser Spreadsheet

No sort		Id	A	B	C	D	E	F	2A	Link
Update	Delete	138	1	4	False	default		NULL	2	Go
Update	Delete	137	0		True	default		NULL	0	Go
Save	Cancel	136	56	qqq	<input checked="" type="checkbox"/>	default ▾		NULL ▾
Update	Delete	135	0		False	further		NULL	0	Go
Update	Delete	134	7657		True	default		NULL	15314	Go
Update	Delete	140	0	141	False	other		NULL	0	Go
Update	Delete	157	0		False	default		NULL	0	Go
Aggregates			7715		False					
Filters										

Pages: 1 | 2

New row

Refresh

Ad-Hoc Code Generation?

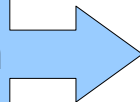
Edit some source files to customize....
Now change the database schema....



Id	A	B	C	D

SQL Table Schema

Metaprogram



```
script/generate scaffold T  
  Id:int A:string B:float  
  C:string D:int
```

app

models

post.rb

views

posts

index.html.erb

show.html.erb

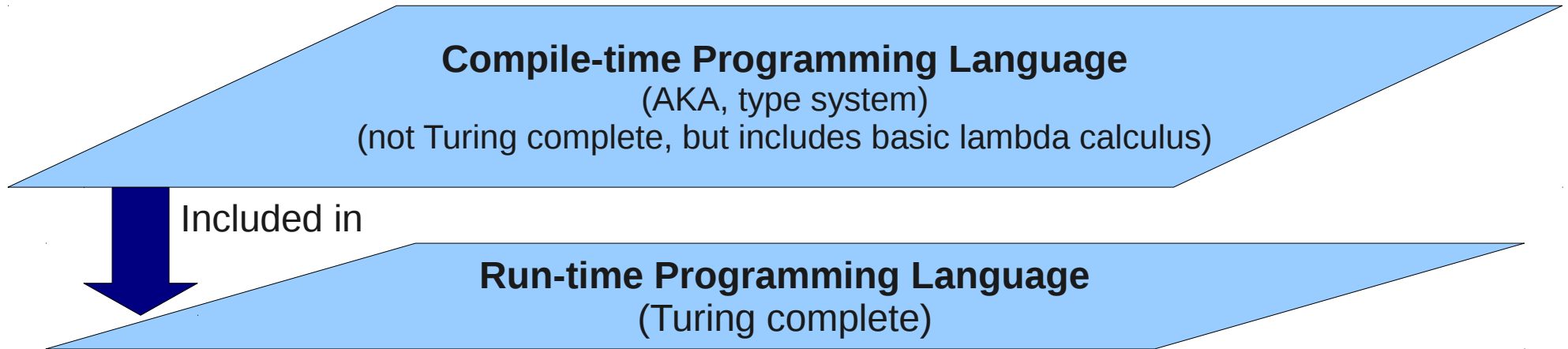
....

config

routes.rb

....

Metaprogramming in a Nutshell



Key Property 1:

These types are expressive enough to guarantee absence of code injection, abstraction violation, etc..

Key Property 2:

Effective static checking that the metaprogram really has the claimed type

Metadata controlling code generation options

Metadata's type

Metaprogram

Type-level program

Sub-Web

Sub-Web's type

One Slide for Type System Geeks

Ur's type system is:

F_{ω}

+

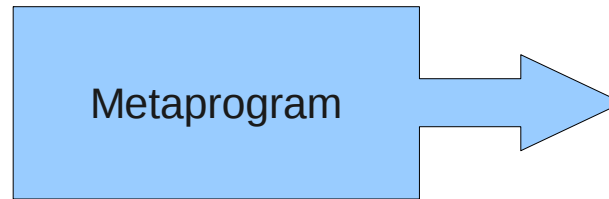
Type-level records

+

Automatic use of inductive
theorems about records during
type inference

An Example Application

Id	A	B	C	D



Crud1

ID	A	B	C	D	
115	2	1	65	True	[Update] [Delete]
123	10	1000	100	True	[Update] [Delete]

A:

B:

C:

D:

SQL Table Schema

```
table t1 : {Id : int, A : int, B : string,  
           C : float, D : bool}  
PRIMARY KEY Id
```

```
open Crud.Make(struct  
    val tab = t1  
  
    val title = "Crud1"  
  
    val cols = {A = Crud.int "A",  
               B = Crud.string "B",  
               C = Crud.float "C",  
               D = Crud.bool "D"}  
end)
```

Ur/Web Available At:

`http://www.impredicative.com/ur/`

Including online demos that I'll present now