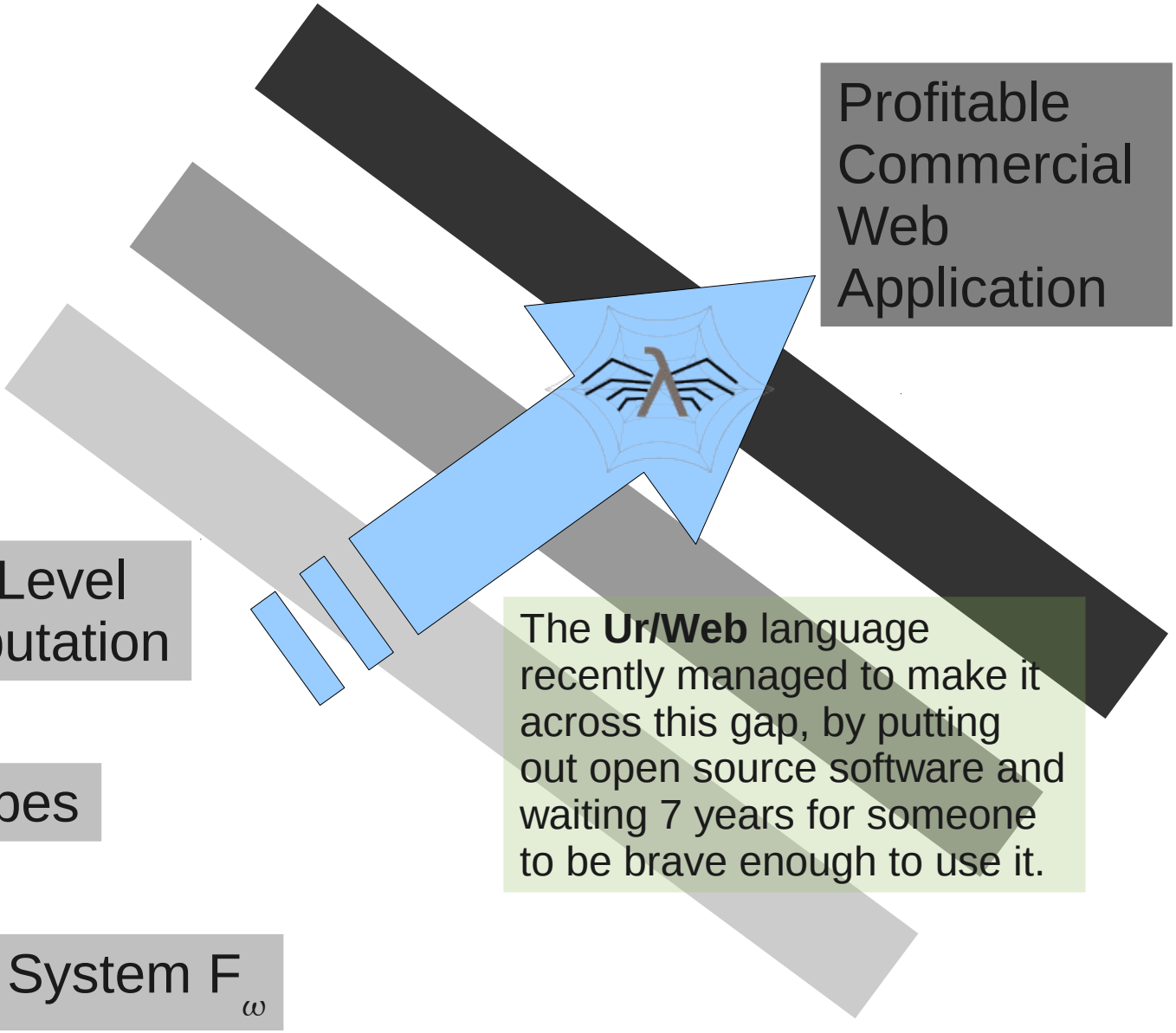
Ur/Web: Taking Syntax Seriously

Adam Chlipala
MIT

SCRIPT
November 13, 2013

A partial map of the programming languages landscape



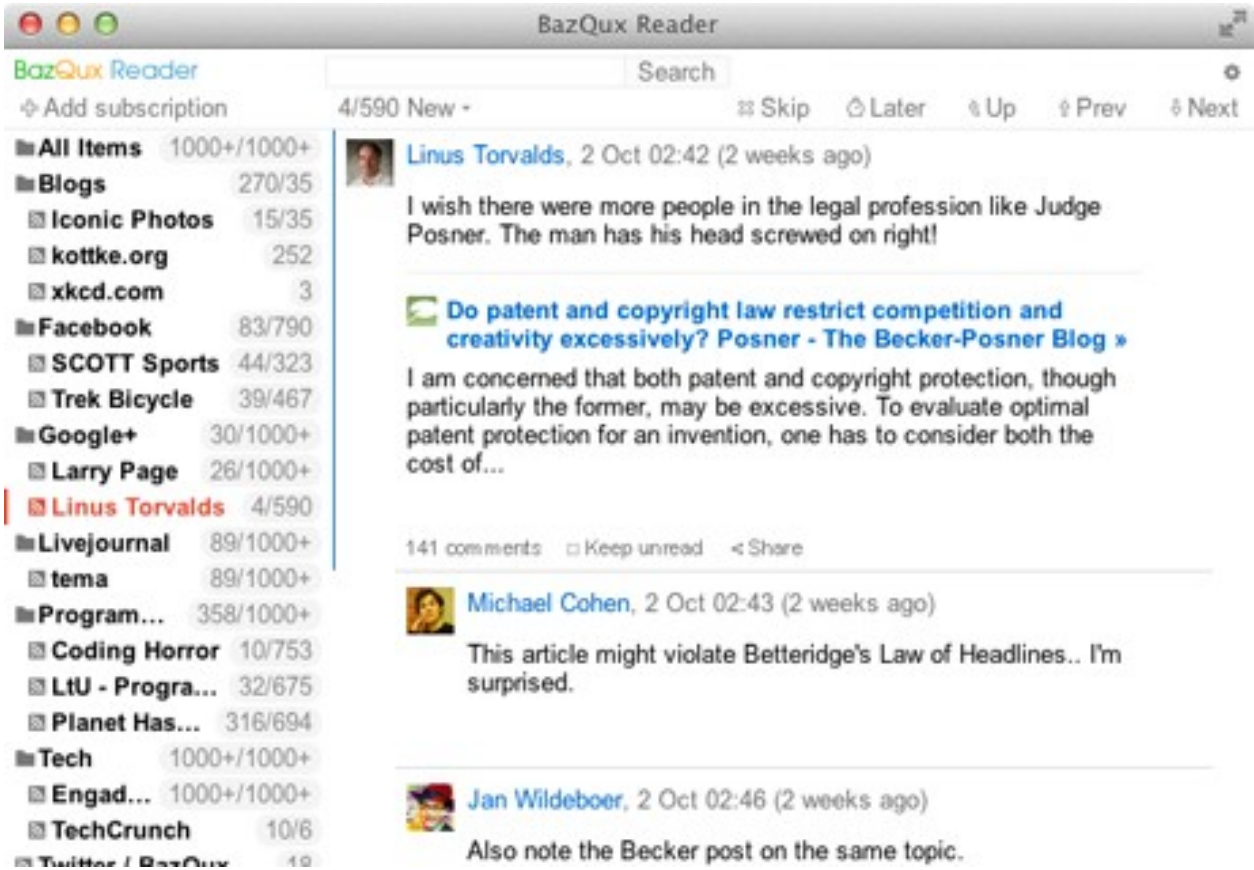
There are at least several users whom I have never met!

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



Executive Summary

Ur/Web is a **domain-specific language**

with a **fancy static type system**

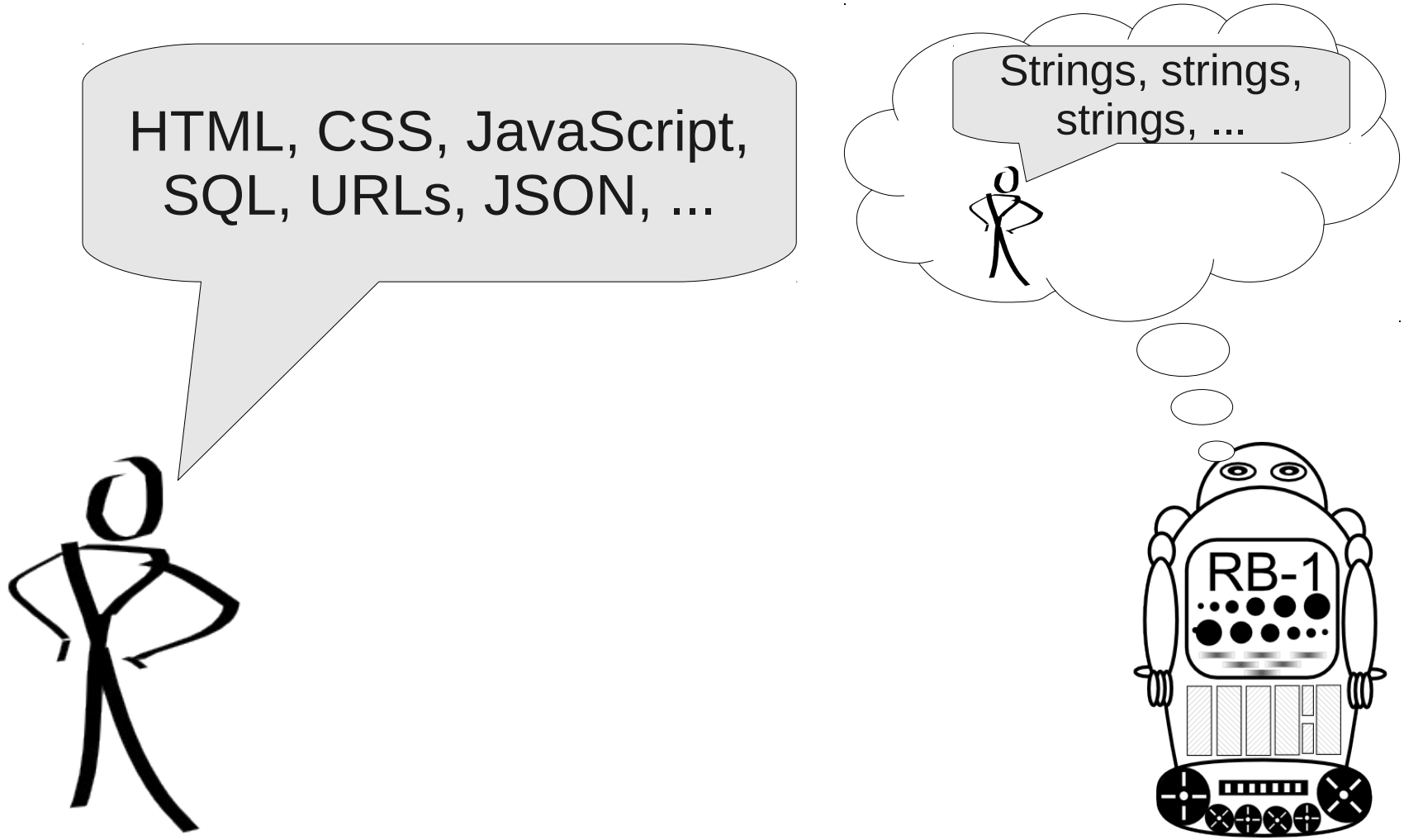
with **first-class support** for Web app architecture

including **strong encapsulation**

and **statically checked metaprogramming**

with **security by construction**

Web Applications: A Steaming Pile of Text



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
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 type system

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

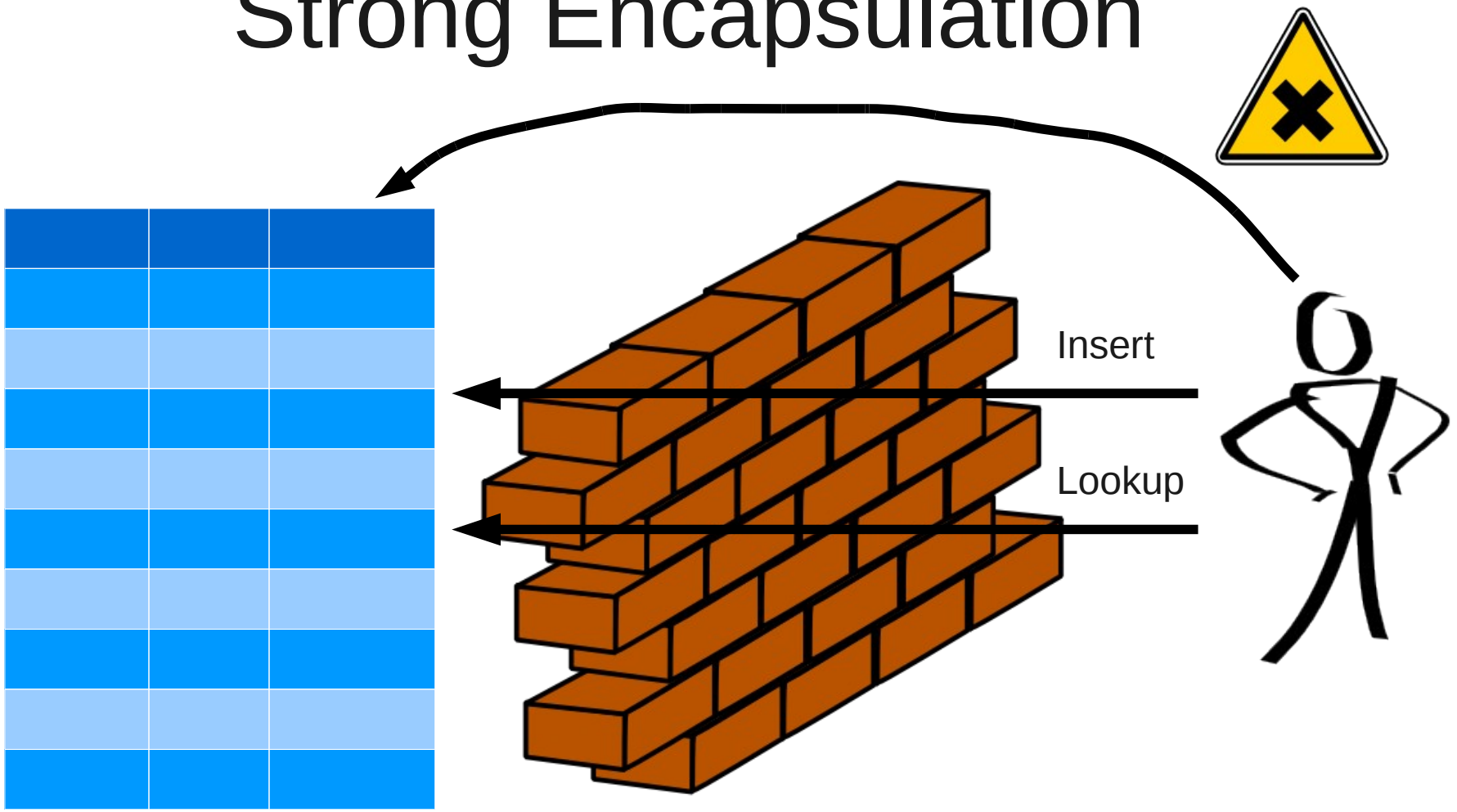
We could design a programming language with built-in static types for hash tables, red-black trees, splay trees, etc., but that's just *no fun at all*.

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

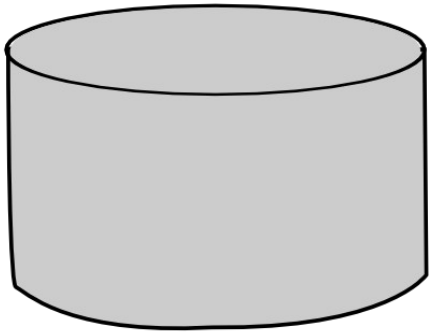
We could design a programming language with built-in static types for XML trees and SQL queries, etc., but that's just *no fun at all*.

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.

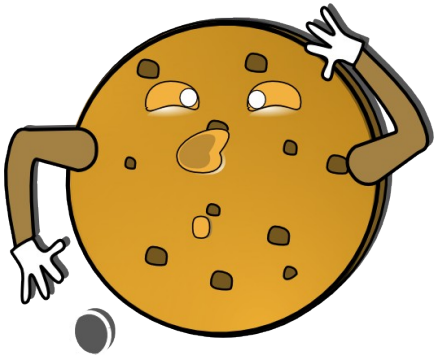
Strong Encapsulation



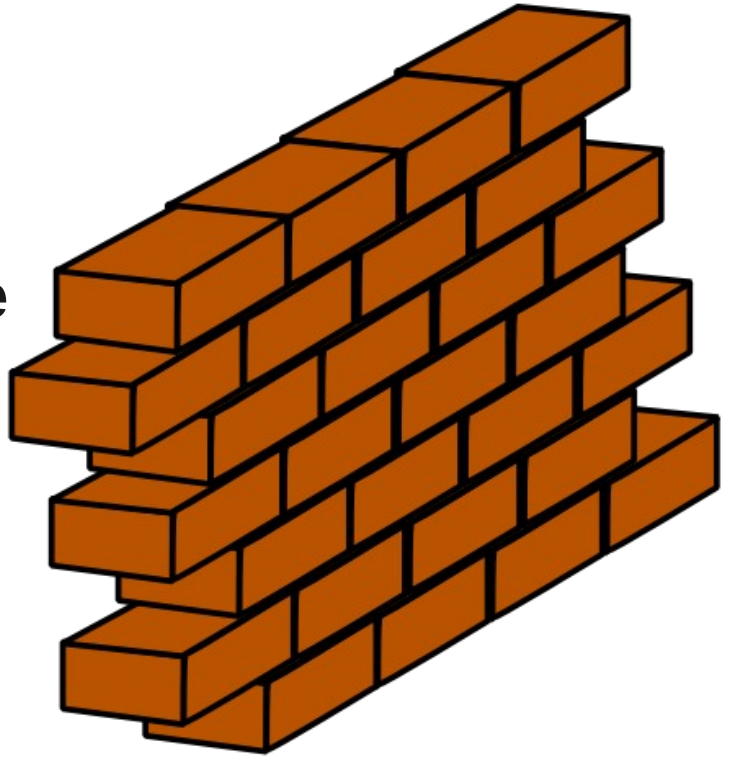
Web 1.0 Encapsulation



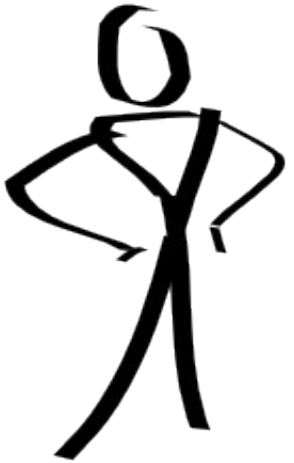
Users Database Table



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

    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
```

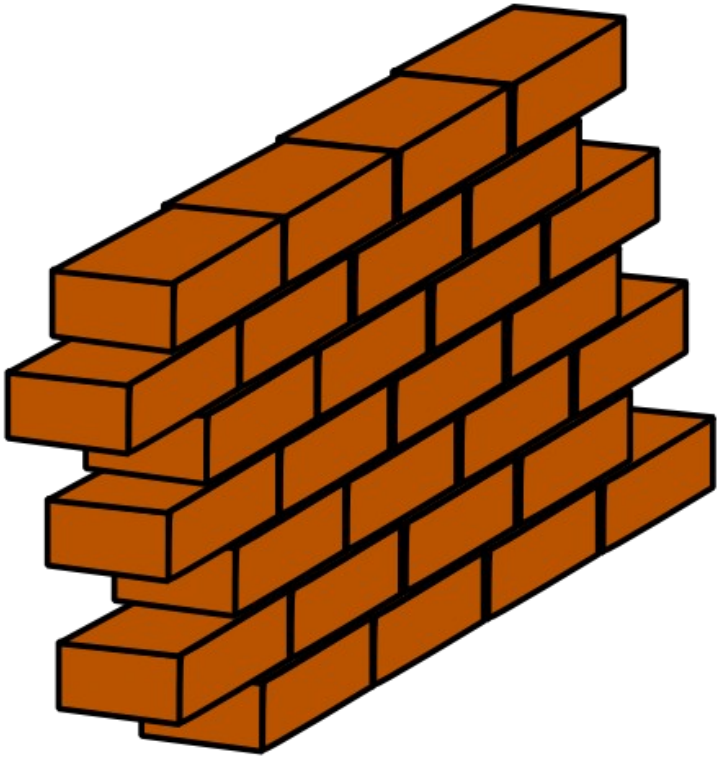
Syntactic sugar for construction of ASTs in a rich type enforcing query validity

Some Client Code

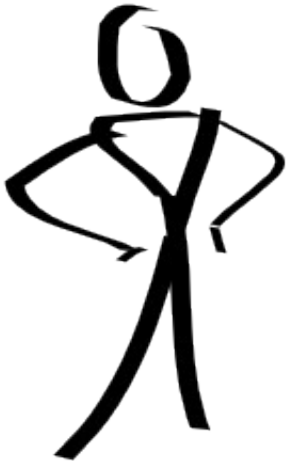
```
fun main () = return <xml>
   {Auth.loginForm ()}

   <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>

a      i ()      Row (SELECT Pw
                      Auth.user
                 WHERE Nam      in')
```

Web 2.0 Encapsulation

Subtree of Dynamic Page Structure



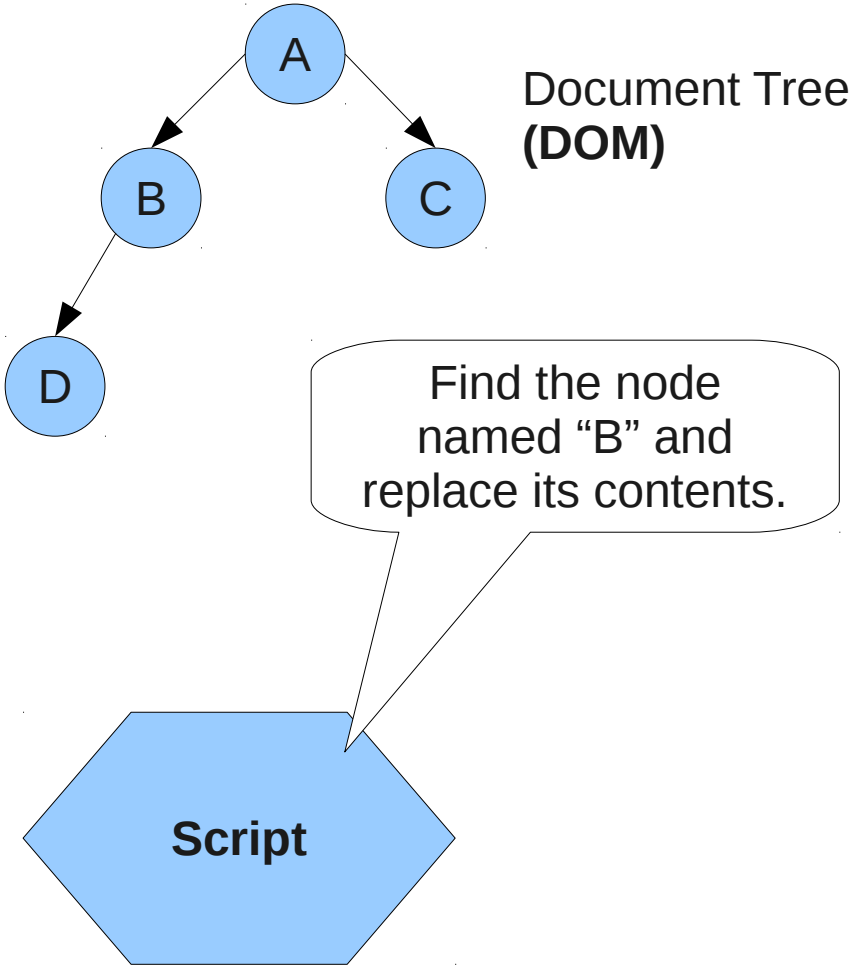
Module Interface



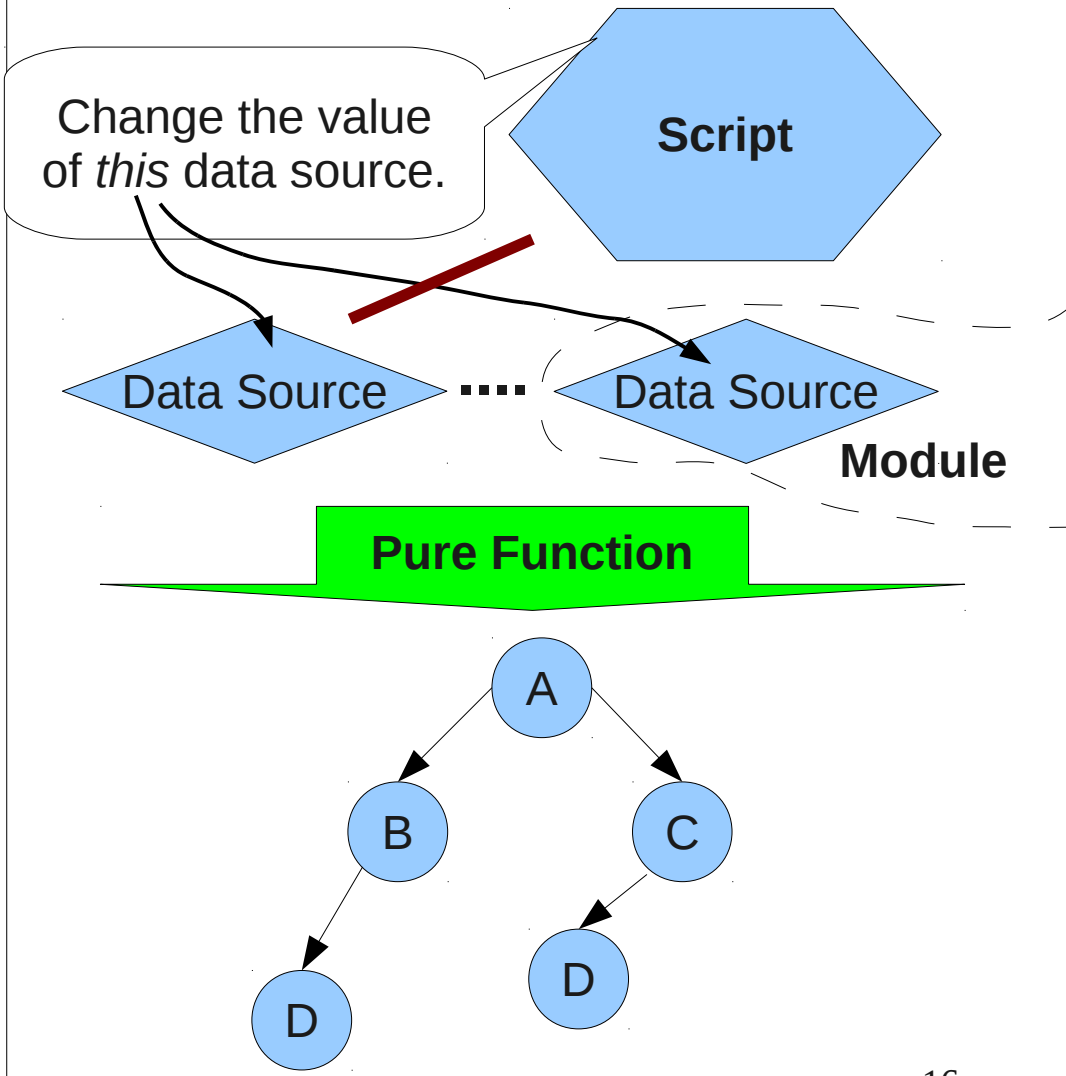
Client Code

Functional-Reactive GUIs

The Status Quo:



The Reactive Way:



A Client-Side Counter

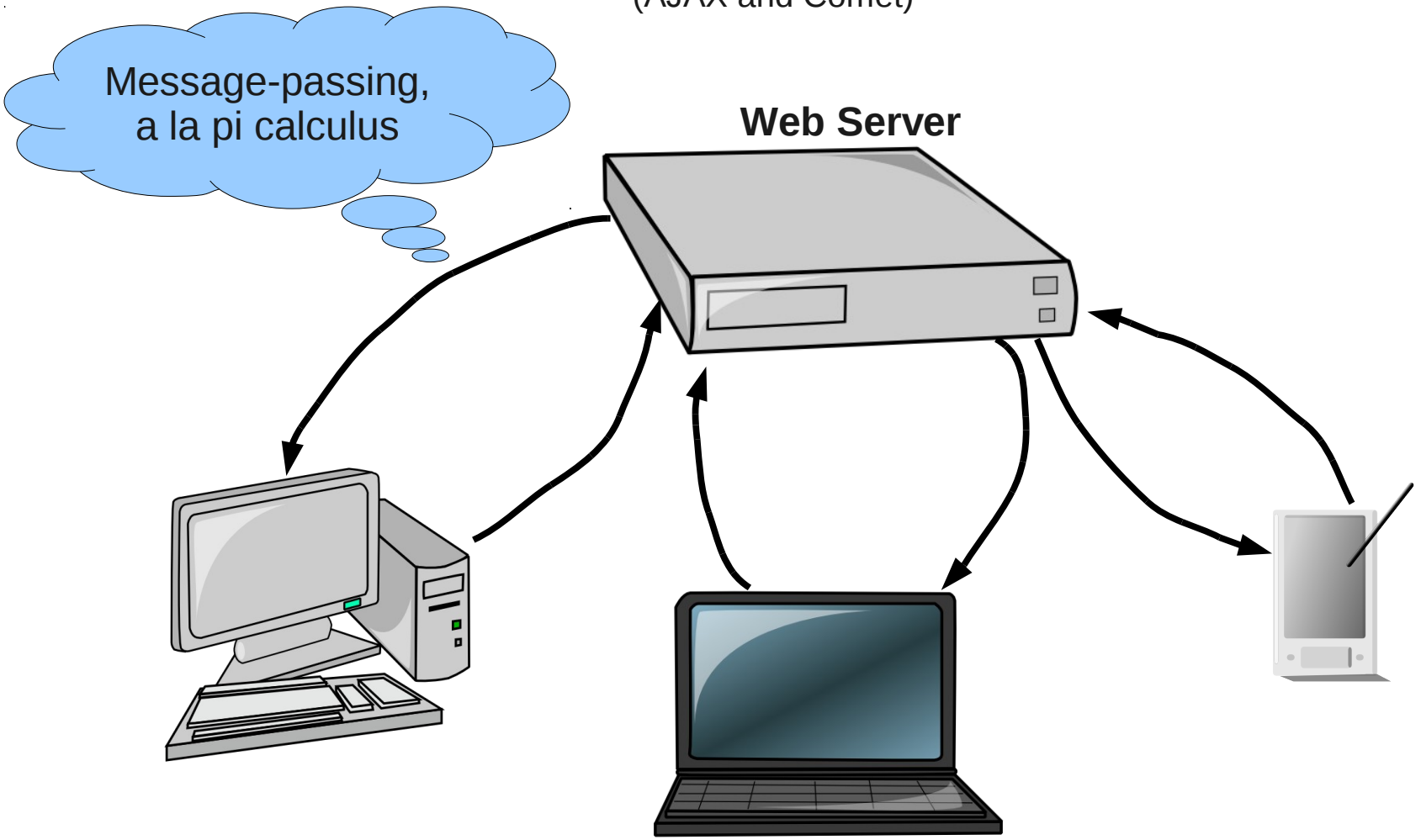
```
structure Counter : sig
                      type t
                      val new : unit -> t
                      val increment : t -> unit
                      val render : t -> xbody
                    end = struct
  type t = source int

  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

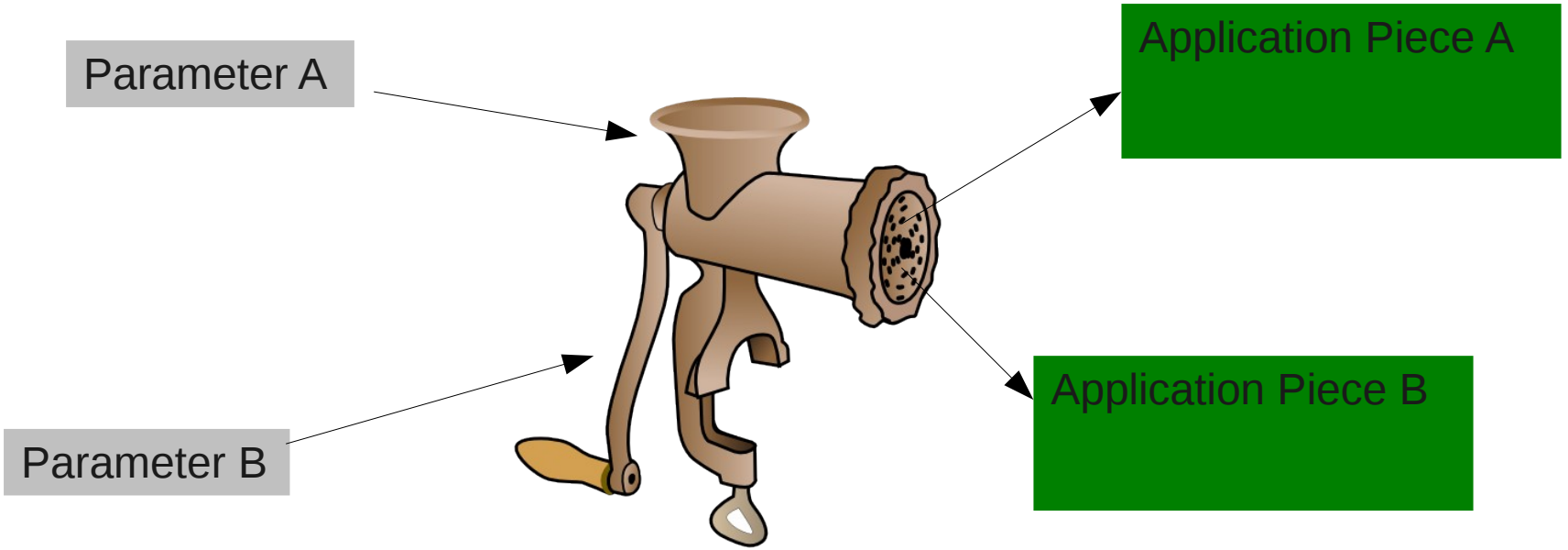
```
fun main () =
   c <- Counter.new ();
   return <xml>
      {Counter.render c}
      <button onclick={Counter.increment c}/>
      Backup copy: {Counter.render c}
      <button onclick={set c -1}/>
   </xml>
```

Client-Server Communication

(AJAX and Comet)



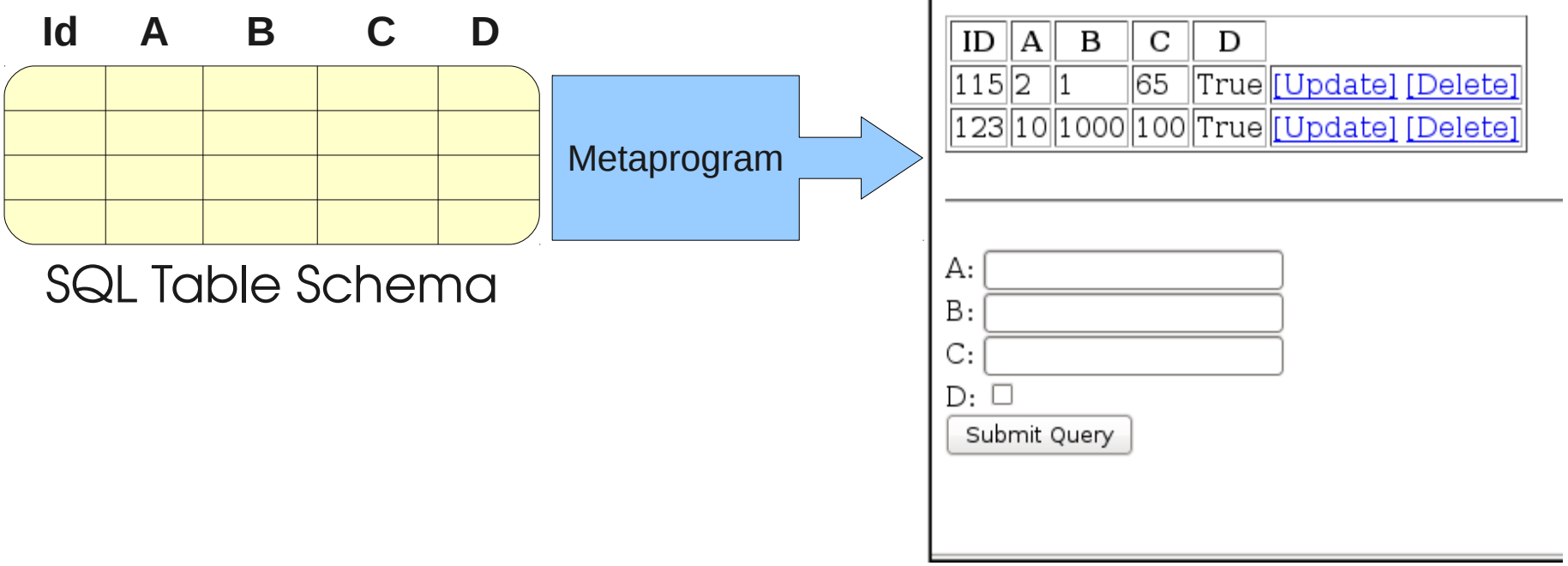
Metaprogramming for the Web



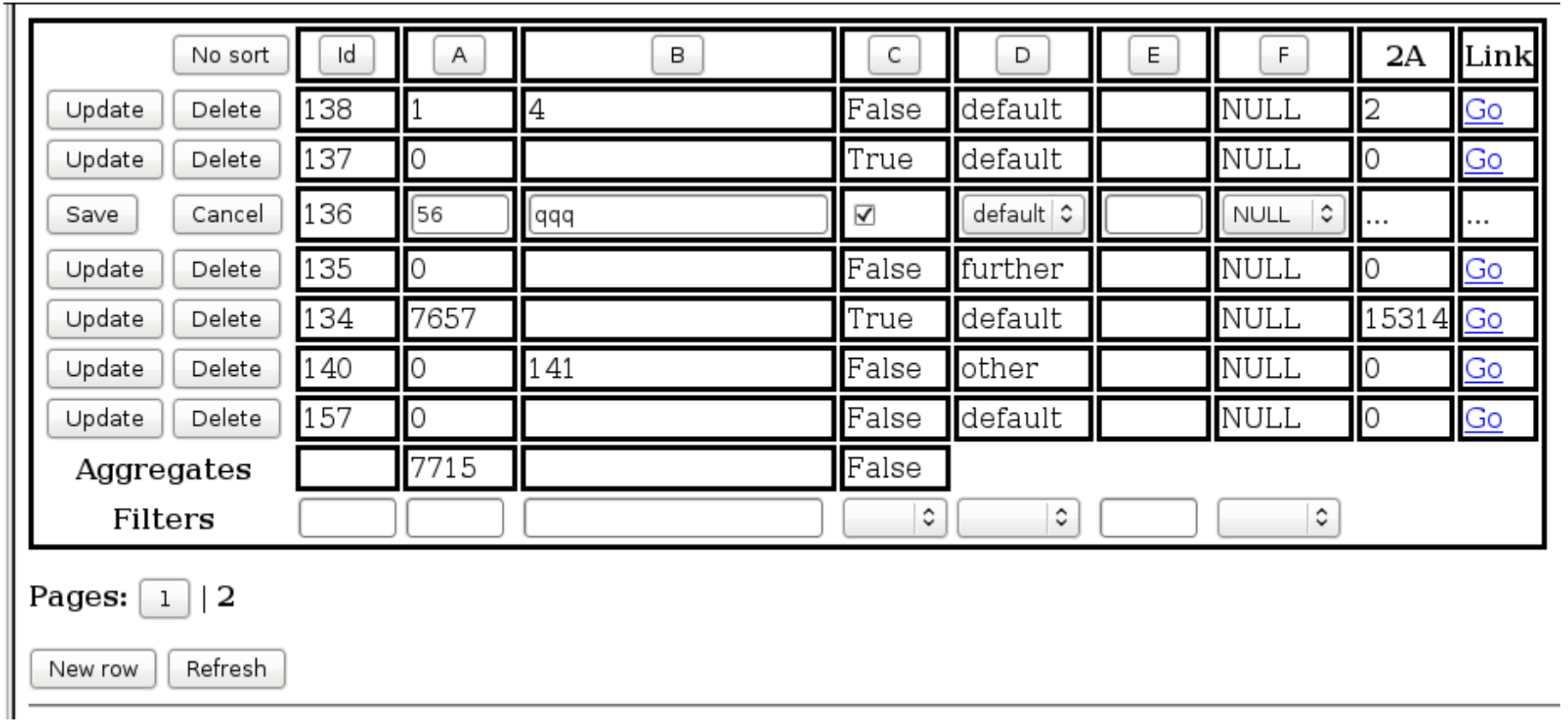
Parameterized code generator

Already very popular in Ruby on Rails and other frameworks!

Automatic Admin Interface



In-Browser Spreadsheet



Ad-Hoc Code Generation?

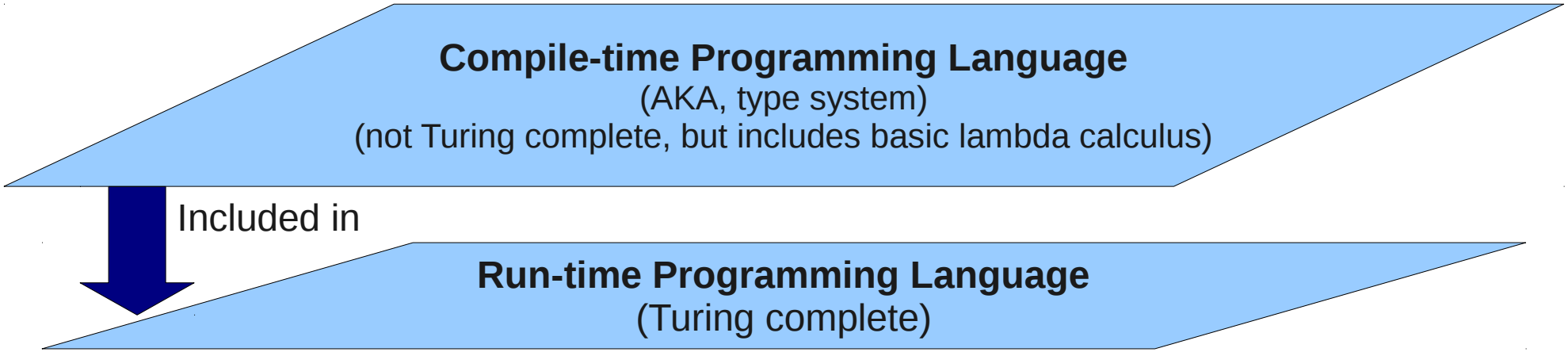
Edit some source files to customize....
Now change the database schema....

SQL Table Schema

Metaprogram

```
app
  models
    post.rb
  views
    posts
      index.html.erb
      show.html.erb
      ....
  config
    routes.rb
  ....
```

Metaprogramming in a Nutshell

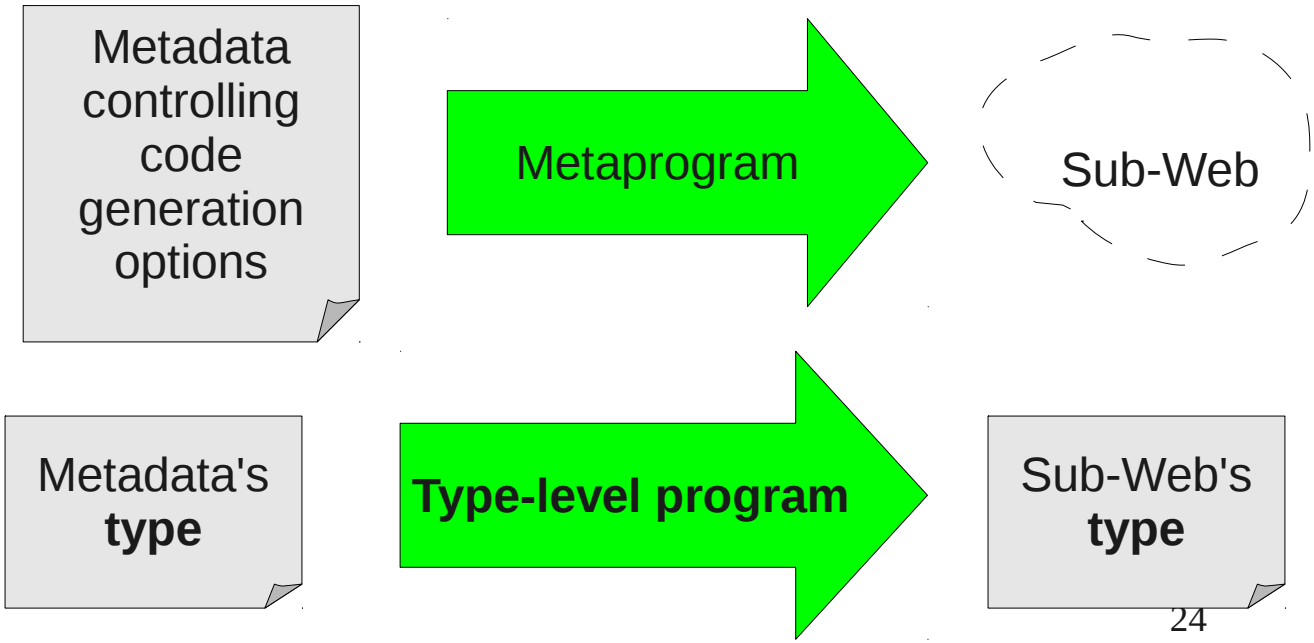


Key Property 1:

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



One Slide for Type System Geeks

Ur's type system is:

F_ω

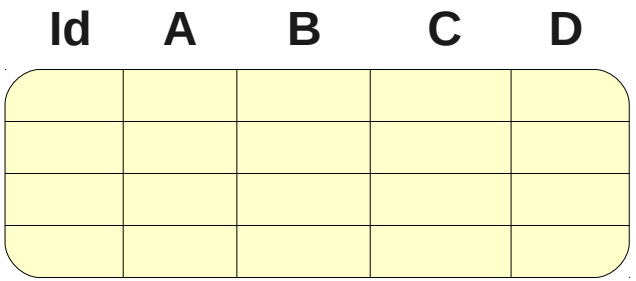
+

Type-level records

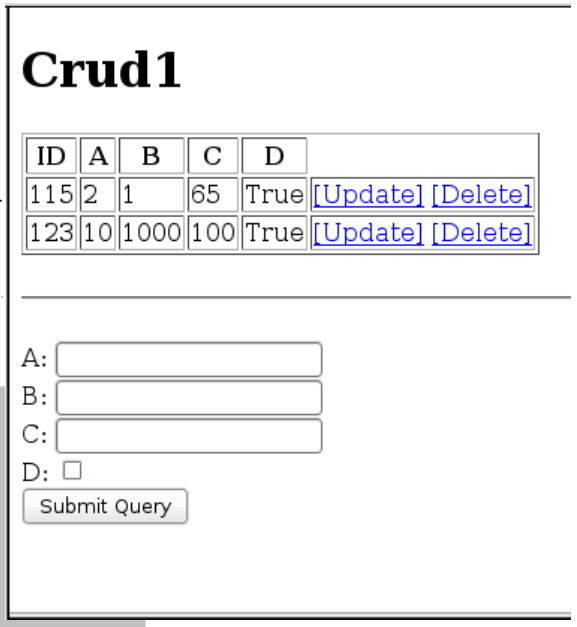
+

Automatic use of inductive theorems about records during type inference

An Example Application



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