

(Advances in)

# Web Programming Languages

Ezra elias kilty Cooper

Sam Lindley

Philip Wadler

Jeremy Yallop

University of Edinburgh

# Why?

Why do we need more languages?

- C++ gives me low-level control
- Java keeps me safe
- Ruby lets me write code quickly

*We want it all.*

The web is different.

# Ways of operating on each list element

C

```
cats = malloc(sizeof(int)*numBlogPosts);  
for (int i = 0; i < numPosts; i++) {  
    cats[i] = blogPosts[i].categoryID;  
}
```

Java

```
Vector<Integer> cats = new Vector();  
for (int p : blogPosts) {  
    cats.add(p.categoryID);  
}
```

Perl

```
@cats = map { $_->categoryID } @blogPosts
```

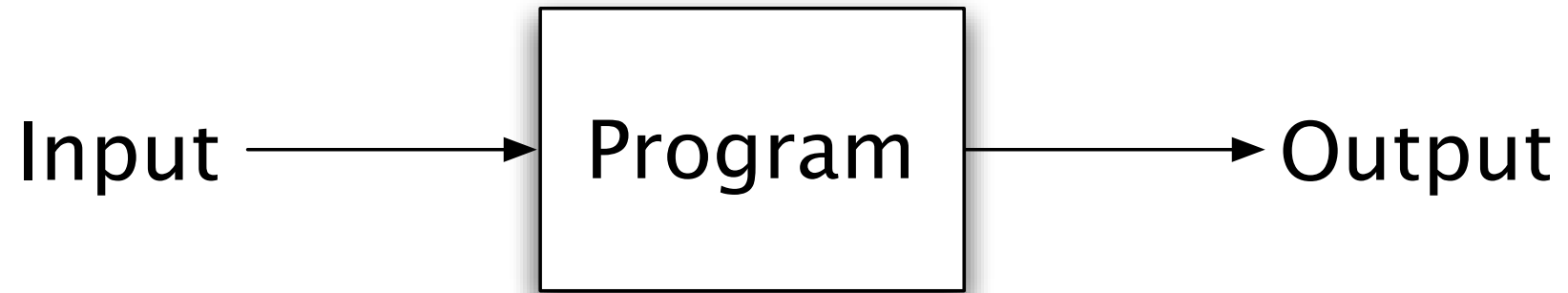
Ruby

```
cats = blogPosts.map { |post| post.categoryID }
```

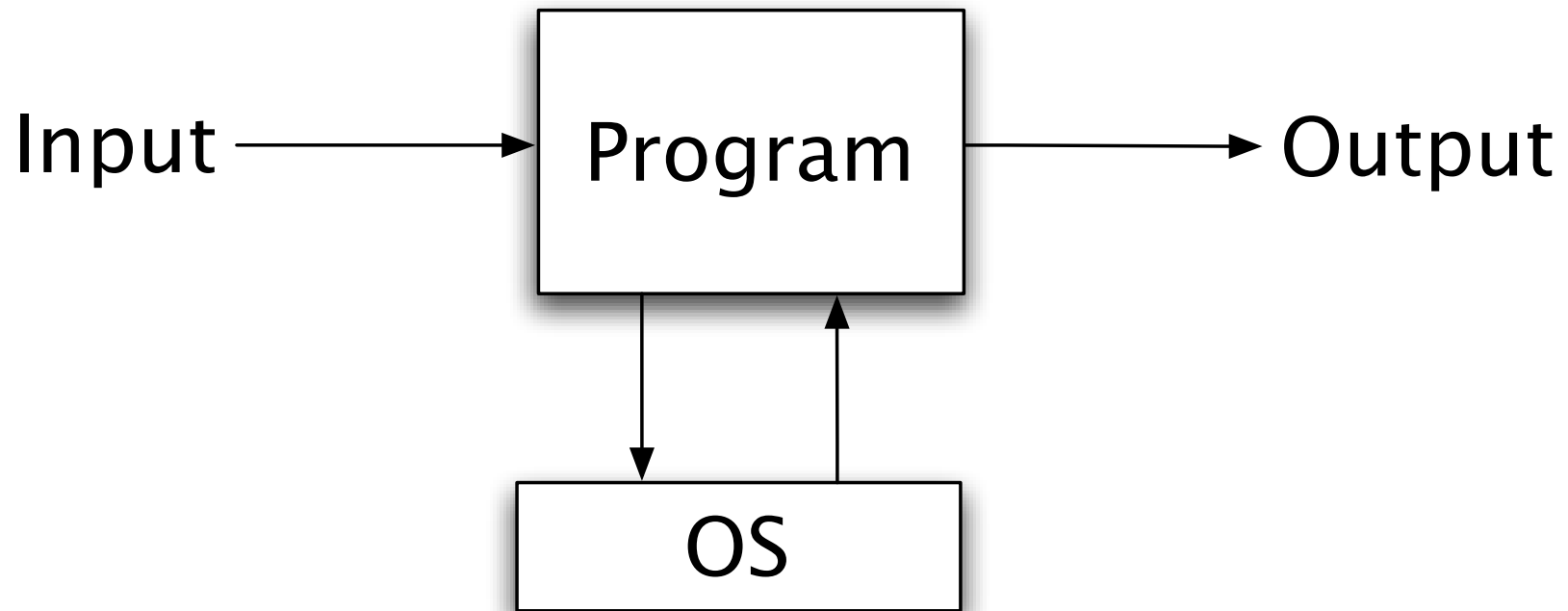
# The web computation model

(Why the web is special.)

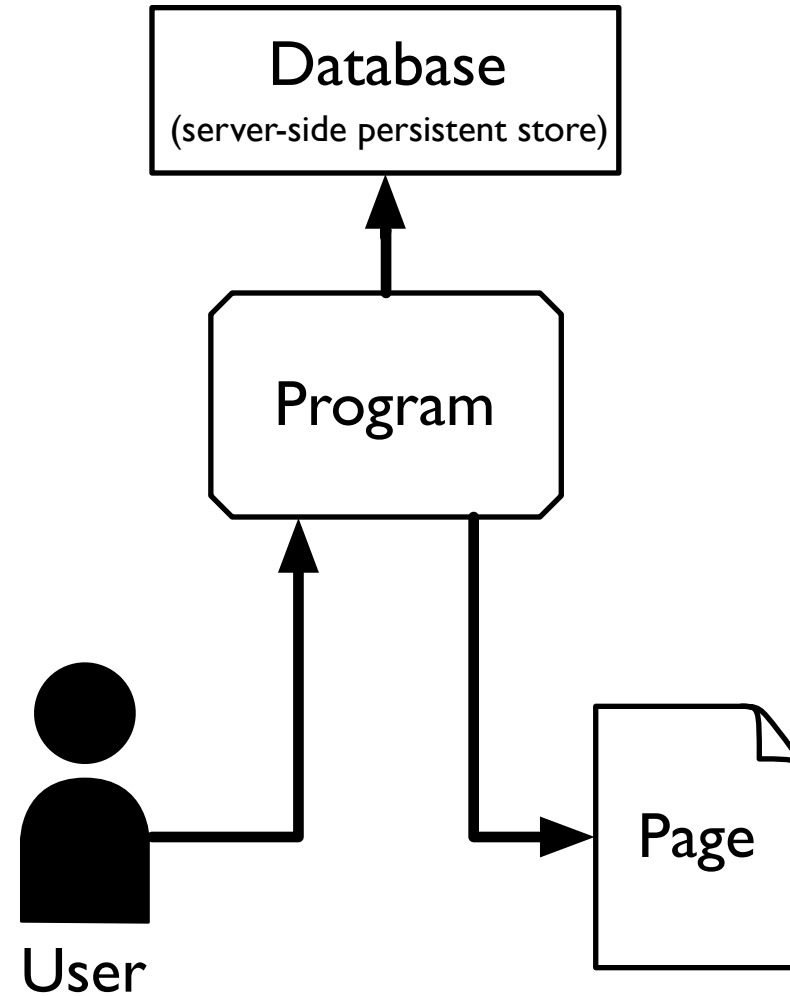
# The I/O model



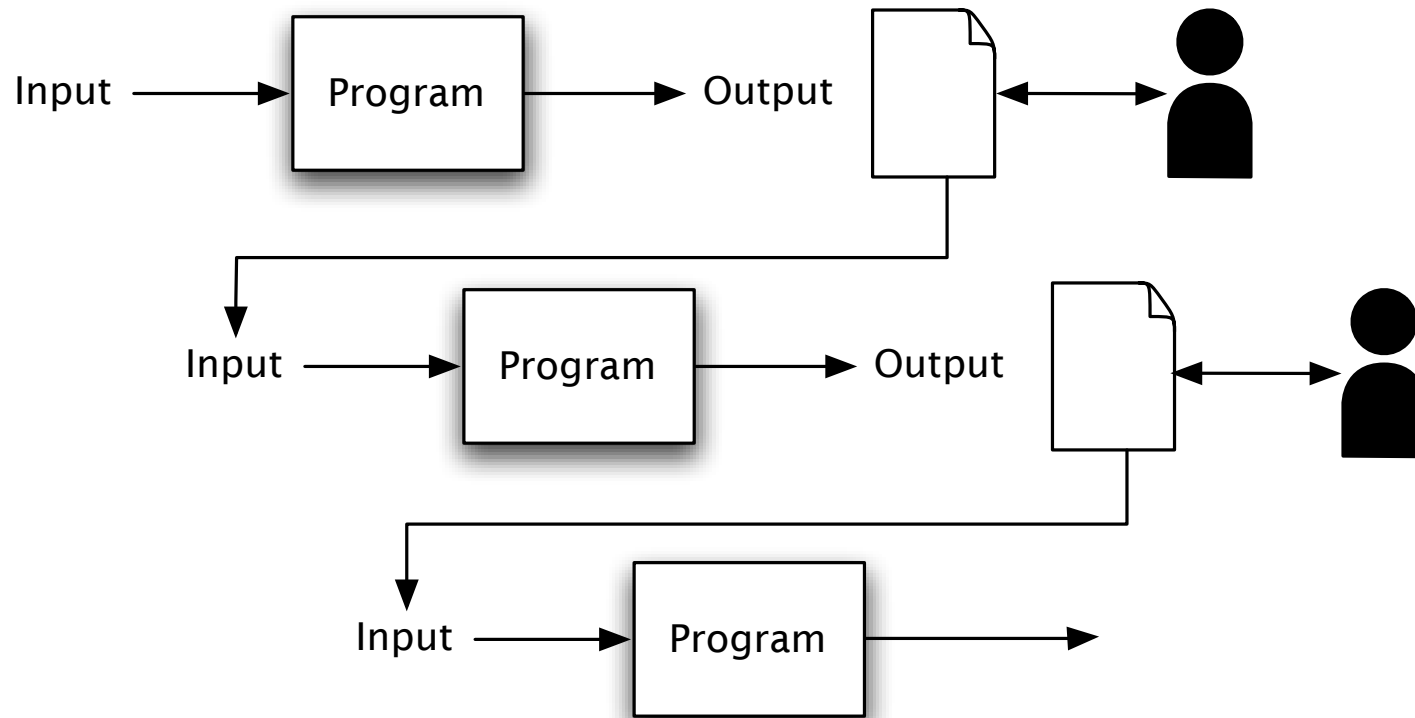
# I/O model revised



# Web Programming



# Building a web experience with the I/O model

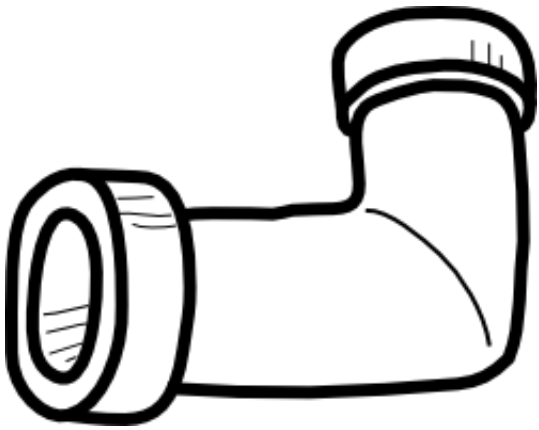


*... a pain!*

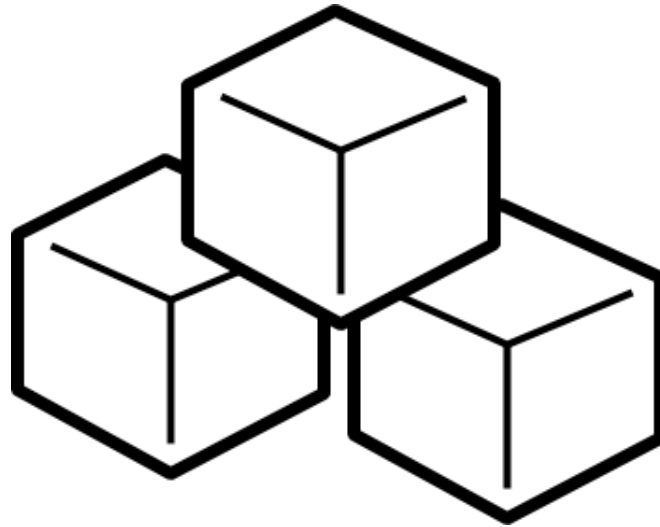
**map** fits tightly around  
many list operations

What fits tightly around  
the web?

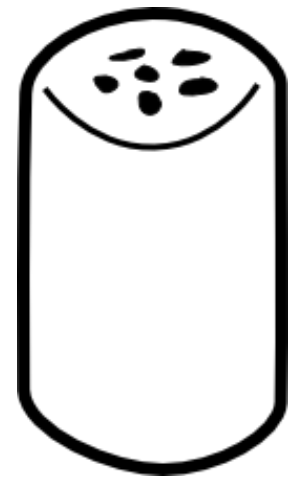
# Links



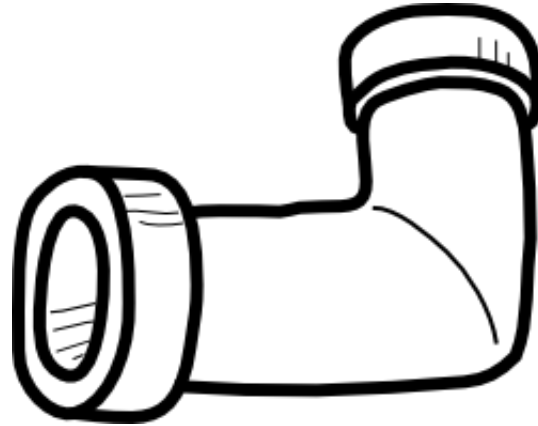
Continuations



Forms

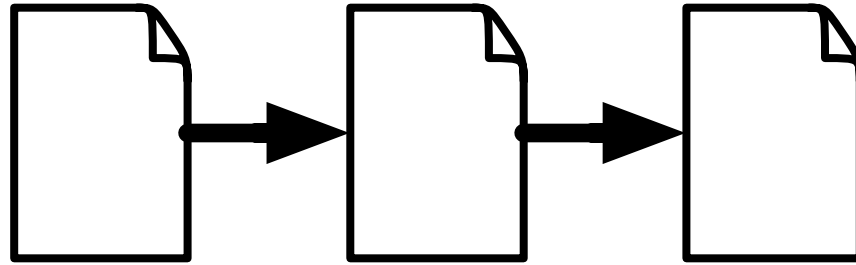


AJAX



# Web continuations

💡 Wizards & pipelines made easy



**A wizard (in Perl)**

# A wizard, in Perl

```
my %dispatchTable = (  
    'wiz1'    => \& wiz1,  
    'wiz2'    => \& wiz2,  
    'finish' => \& finish,  
);  
  
my $defaultAction = \& wiz1;
```

# A wizard, in Perl

```
sub wiz1 {  
    my ($q) = @_  
    # validation goes here  
    useTemplate('wiz1.tmpl', {});  
}
```

```
sub wiz2 {  
    my ($q) = @_  
    # validation goes here  
    useTemplate('wiz2.tmpl',  
                {name => $q->param('name'),  
                 ship_addr =>  
                   $q->param('ship_addr')});  
}
```

# A wizard, in Perl

```
sub finish {
    my ($q) = @_;
    my $name = $q->param('name');
    my $ship_address = $q->param('ship_address');
    my $bill_address = $q->param('bill_address');
    my $city = $q->param('city');
    my $postcode = $q->param('postcode');
    # validation goes here

    commitOrder($name, $ship_address,
                $bill_address, $city, $postcode);

    useTemplate('finish.tmpl',
                { name          => $name,
                  ship_address => $ship_address,
                  bill_address => $bill_address,
                  city         => $city,
                  postcode     => $postcode });
}
```

# A wizard, in Perl: template (p. 1)

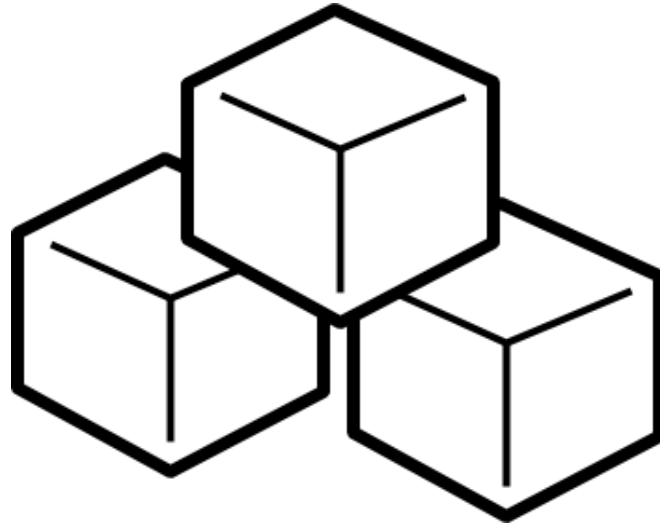
```
<form action="">
  <table>
    <tr>
      <label><td>Billing Address:</td>
      <td><input name="bill_address" /></td></label>
    </tr><tr>
      <label><td>City:</td>
      <td><input name="city" /></td></label>
    </tr><tr>
      <label><td>Postcode:</td>
      <td><input name="postcode" /></td></label>
    </tr><tr>
      <td></td>
      <td><input type="submit" value="Continue" /></td>
    </tr>
  </table>
```

# A wizard, in Perl: template (p. 2)

```
<input type="hidden" name="action" value="finish" />
<input type="hidden" name="name"
value="<TMPL_VAR NAME="NAME">" />
<input type="hidden" name="ship_address"
value="<TMPL_VAR NAME="SHIP_ADDRESS">" />
</form>
```

# A wizard, in Links

```
fun pipeline() {
    var (name=name, ship_addr=ship_addr) =
        sendSuspend(wiz1Tpl);
    # validation goes here
    var (bill_addr=bill_addr, city=city,
        postcode=postcode) =
        sendSuspend(wiz2Tpl);
    # validation goes here
    commitOrder(name, ship_addr, bill_addr, city,
        postcode);
    finishTpl(name, ship_addr, bill_addr, city,
        postcode)
}
```



# Forms abstraction



Building blocks for forms

# Forms abstraction (Links)

**form**

```
<table>
  <tr>
    <label><td>Name:</td>
      <td>{input -> name}</td></label>
  </tr><tr>
    <label><td>Desired Arrival:</td>
      <td>{date() -> arrival}</td></label>
  </tr><tr>
    <td></td>
      <td><input type="submit" value="Continue" /></td>
  </tr>
</table>
```

```
yields {
  (name = name, arrival = arrival)
}
```

# Forms abstraction (Links)

```
sig date : () -> Form (Int, Int)
fun date() {
  form
    <#>{input -> dateStr}</#>
  yields {
    var (dayStr, moStr) = splitAt('/', dateStr);
    (stringToInt(dayStr), stringToInt(moStr))
  }
}
```

# Forms abstraction: *Two* dates

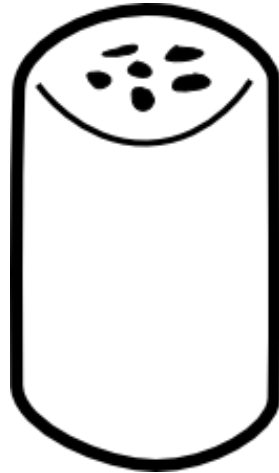
**form**

```
<table>
  <tr>
    <label><td>Arrival:</td>
      <td>{date() -> arrival}</td></label>
  </tr><tr>
    <label><td>Departure:</td>
      <td>{date() -> departure}</td></label>
  </tr><tr>
    <td></td>
      <td><input type="submit" value="Continue" /></td>
  </tr>
</table>
```

**yields {**

```
(arrival = arrival, departure = departure)
```

**}**



# Simplified AJAX



Treat AJAX calls as simple RPC calls

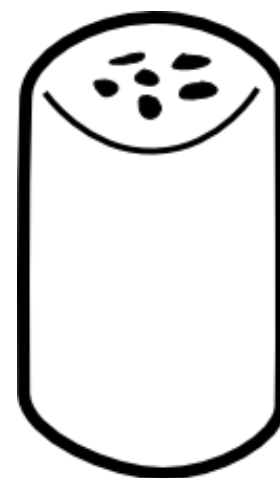
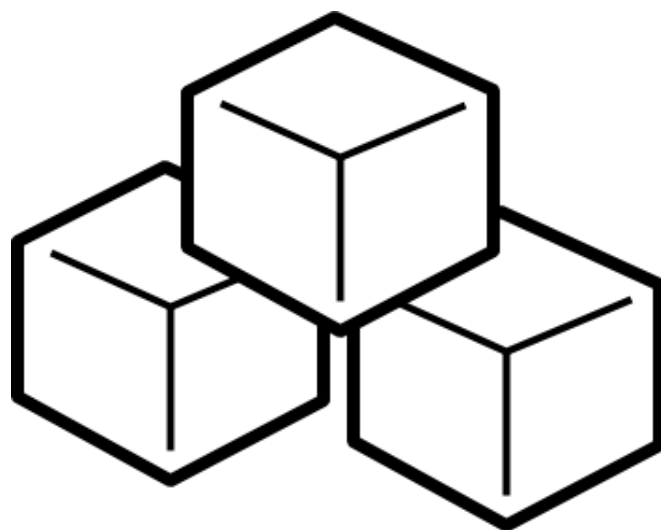
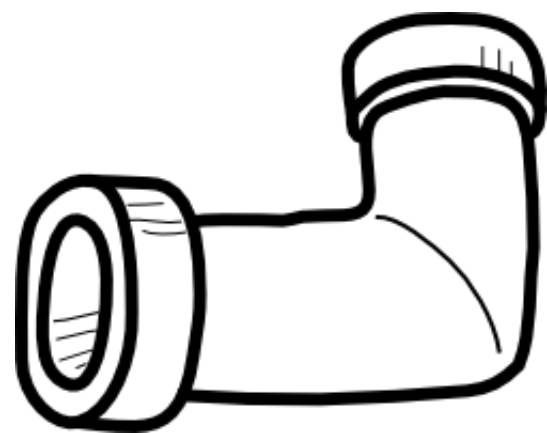
# Simplifying AJAX

```
fun list(items) client {
  <ul>
    {for (var x <- items)
      <li id="{x}">
        
        {stringToXml(x)}
      </li>
    }
  </ul>
}
```

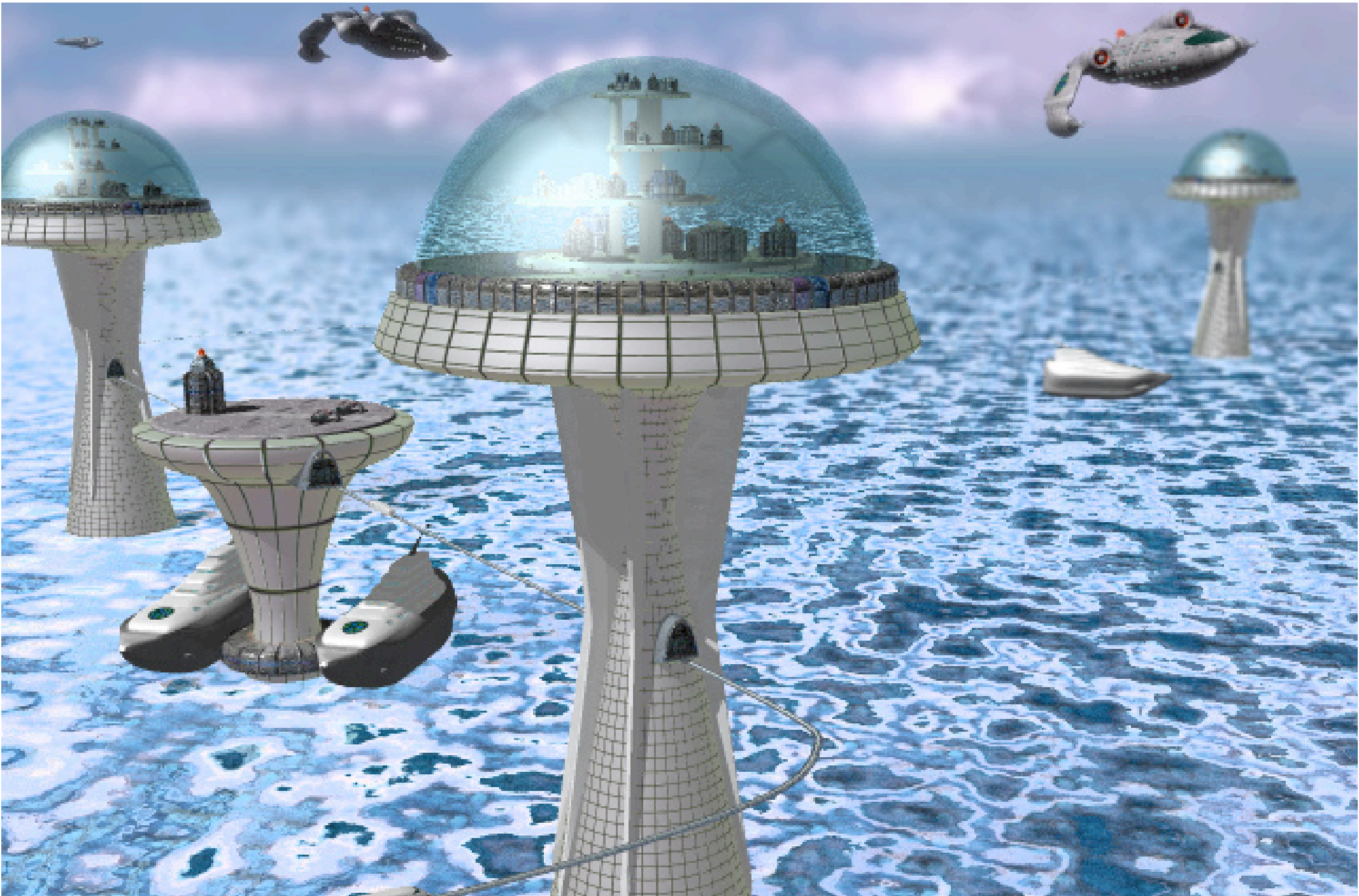
# Simplifying AJAX

```
fun removeFromDB(item) server {  
    var todoTable = table "todo"  
        with (item : String)  
        from (database "ezra");  
    delete (var itemRow <-- todoTable)  
        where (itemRow.item == item)  
}
```

```
fun remove(item) client {  
    removeNode(getNodeById(item));  
    removeFromDB(item)  
}
```



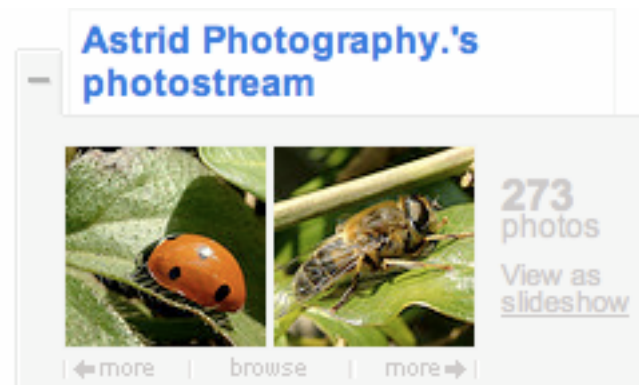
# The Future



# HTML+CSS

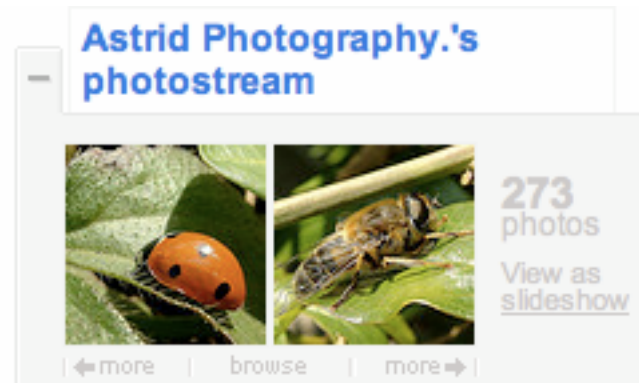
A declarative page-specification language.

# Declarative & Interactive?

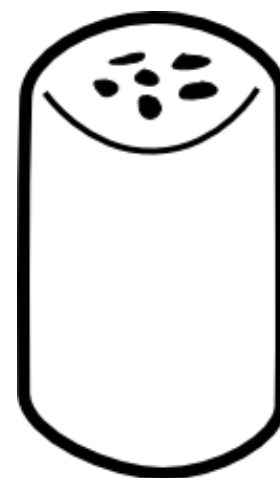
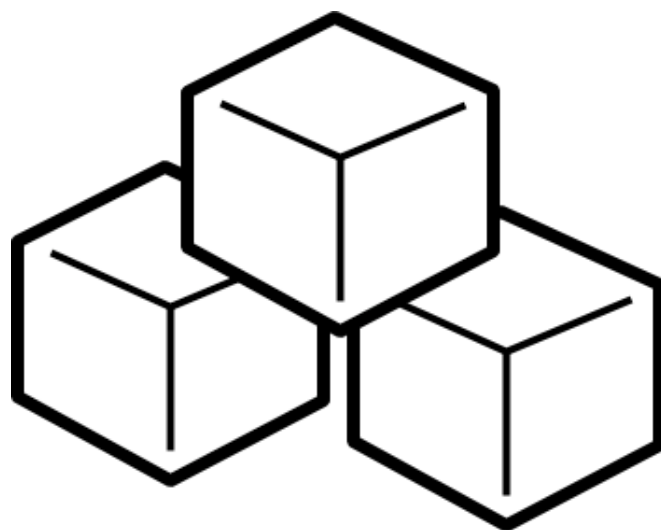
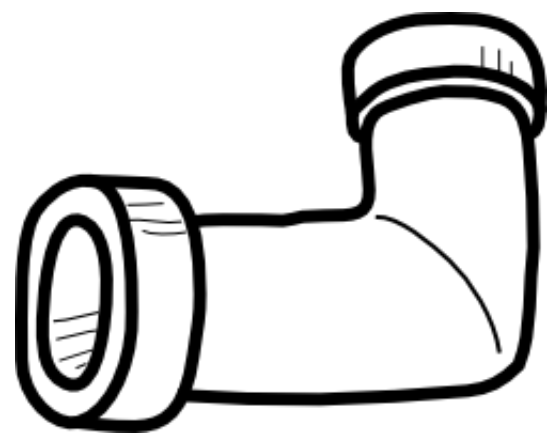


How do we make a declarative language that can express all the bells & whistles of Flickr?

# Functional Reactive Programming



- On-screen objects behave as functions of (time-varying) input parameters



# Links: Web Programming Without Tiers

Try it:

[groups.inf.ed.ac.uk/links](http://groups.inf.ed.ac.uk/links)

**Thank You**