# Scala, Lift and the Real Time Web

**David Pollak**

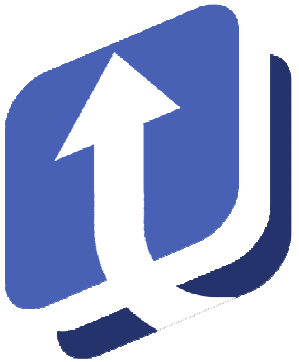
**Benevolent Dictator for Life**

**Lift Web Framework**

**dpp@liftweb.net**

# All about me (David Pollak)

- Sometimes strict, mostly lazy
- Lead developer for Lift Web Framework
- Spreadsheet junky (writing more than using)
- Writing *Beginning Scala*

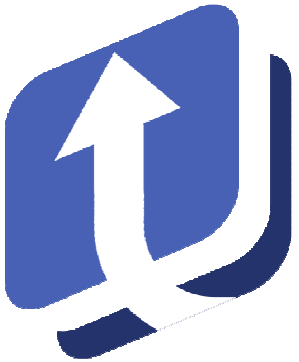



# Oh, the things I'll cram into your brain

- Scala is a Functional/OO hybrid language
  - Compiles to JVM byte-code
  - Runs at native speeds
  - Full Java library interoperability
- Lift is a powerful, simple Web Framework
  - Best framework for building interactive web sites
  - More concise than Ruby on Rails
  - More type-safe than anything you've ever used (except Happs)
- Scala leads to Lift

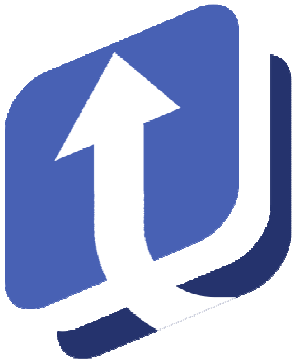
# World Wide Web: In the Beginning

- Antisocial
- Person ↔ Machine
  - Shopping
  - Banking
  - CRUD
- Browser == Green Screen



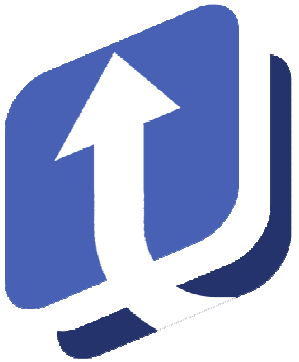
# Web 2.0

- More Social
- Different Flavors
  - Person ↔ Machine ↔ Machine: Mashups
  - Person ↔ Person: Facebook, Twitter
  - Machine ↔ Machine → Person: Microformats
- Internet becomes Postman



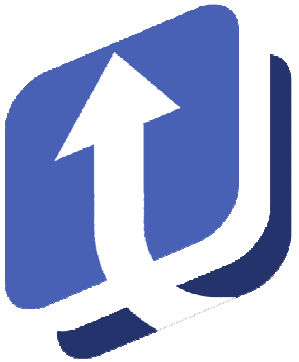
# Real Time Web

- We are social Animals that love instant gratification
- Real Time
  - Games
  - Chat
  - Everything
- Next wave: Real Time Web



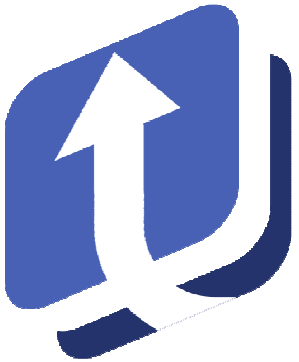
# Punch Line

- Scala → Lift
- Lift → Real Time Web
- Real Time Web → Awesome User Experience



# Real-time Chat in Lift: Messages

- `case class Messages(msgs: List[String])`

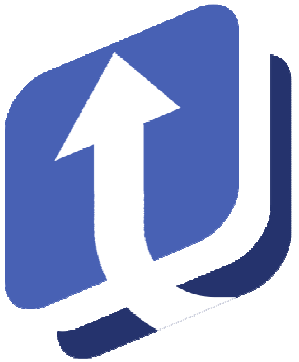


# Real-time Chat in Lift: Server

```
object ChatServer extends Actor with
      ListenerManager {
  private var msgs: List[String] = Nil

  protected def createUpdate = Messages(msgs)

  override def highPriority = {
    case s: String if s.length > 0 =>
      msgs ::= s
      updateListeners()
  }
  this.start
}
```



# Real-time Chat in Lift: Comet

```
class Chat extends CometActor with CometListenee {
  private var msgs: List[String] = Nil

  def render =
  <div>
    <ul>{msgs.reverse.map(m => <li>{m}</li>)}</ul>
    {ajaxText("", s => {ChatServer ! s; Noop})}
  </div>

  protected def registerWith = ChatServer

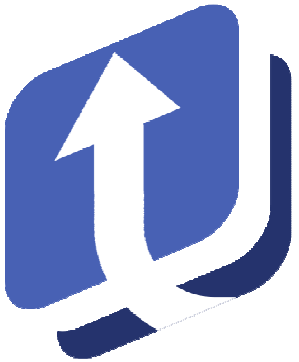
  override def highPriority = {
    case Messages(m) => msgs = m ; reRender(false)
  }
}
```

# Singletons

- `object ChatServer extends Actor with ListenerManager`
- ChatServer is a singleton
- One instance per JVM
- Can be passed as parameter… it's an instance
- Composition of Actor and ListenerManager

# Case classes

- `case class Foo(bar: String,`
  `baz: List[Foo])`
- For Free:
  - `bar` and `baz` properties (immutable by default)
  - `toString`, `hashCode`, and `equals`
  - Pattern matching with parameter extraction
- 20 lines of boilerplate reduced to 1 line



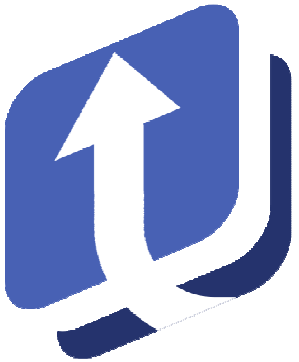
# Pattern Matching

- ```
  case Messages(m) => msgs = m
  case s: String if s.length > 0 =>
                                  msgs ::= s
  ```
- Match against case classes
  - Extract parameters
  - Test against parameters: `case Person(name, 35) =>`
  - Great for message/event handling
- Type-safe casting
- Awesome declarative way of expressing logic


  ```

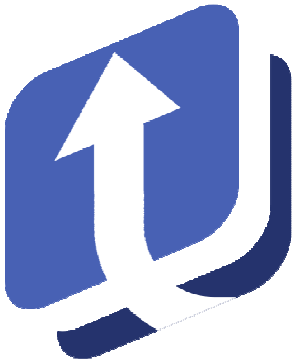
# Traits and Composition

- `class Chat extends CometActor with CometListenee`
- Traits are interfaces plus data and logic
- Composition
  - `object sally extends Person("Sally") with Female with Runner`
  - `def womansRun(who: Female with Runner) -> womansRun(sally)`
- Benefits of multiple inheritance w/o diamond problem



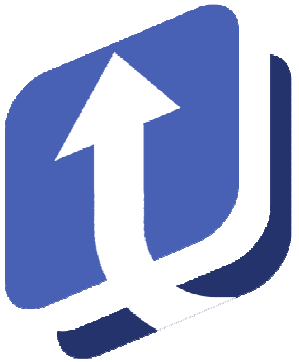
# Immutable Data Types

- `var msgs: List[String] = Nil`
  `<div>Hello</div>`
- Immutability your long-time friend: String
  - Never have to say synchronized
  - Never have to make a copy “just in case”
  - Great for hash keys
- Leads to transformational thinking
- Better for garbage collector



# Function passing

- `msgs.reverse.map(m => <li>{m}</li>)`
  `ajaxText("",`
  `s => {ChatServer ! s; Noop})`
- `map` takes a function as a parameter
  - Transforms `String` to `Elem`
  - Applied to each `String` in msgs
  - The function is strongly typed: m is a `String`
- Functions are instances and can be passed
- Functions can be put in `Map`s for later use



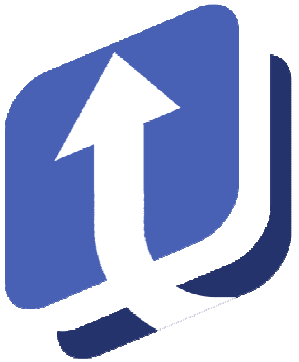
# XML Literals and Support

- `<ul>{msgs.reverse.map(m =>`
  `<li>{m}</li>)}</ul>`
- XML first-class in Scala, like Strings in Java
- Library-level XPath-style operators
  - `xml \ "li"` – find all the child <li> tags
  - `for {p <- x \\ "p"; ca <- p \ "@class"`
    `c <- ca.text.split(" ")} yield c`

    Find all the classes used by <p> tags
- Immutable, like Strings

# Actor Library

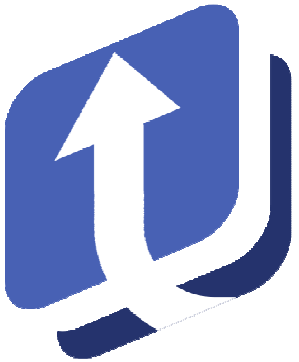
- Real Time means events
- Threadless, stackless event handlers
- With very nice syntax (Erlangish)



# Feeling RESTful

```
case Req(ApiPath :: "statuses" ::
                    "public_timeline" :: Nil,
        this.method, GetRequest) => publicTimeline

  def publicTimeline(): Box[TwitterResponse] = {
     val statusList =
       Message.findAll(OrderBy(Message.id,
                       Descending),
                       MaxRows(20)).
         map(msgData _)
     Full(Right(Map("statuses" ->
                ("status", statusList) )))
   }
```



# Conclusion

- Scala's object model is a superset of Java's
- Scala's traits: super-powerful class composition
- Scala is more type-safe than Java
- Scala is syntactically simpler than Java or Ruby
- Scala is as concise as Ruby
- Scala has awesome libraries including Actors
- What if Java, Ruby, and Haskell has a love-child?
- Scala's design led to Lift's design
- Lift's design makes the Real Time Web super-simple

# Questions