# CS290b – Lecture 2
# Introduction to Rails

Scalable Internet Services, Fall 2013

Jon Walker
Department of Computer Science
University of California at Santa Barbara

## For today…

- Ruby Koans
- Intro to Ruby on Rails
- Web Services
- Announcements

## Ruby Koans

Email grading@cs290.com a .zip or .tar of your Ruby Koans by Friday Oct. 4th

- **Did everyone get Ruby installed and working?**
- **Did everyone get Rails installed and working?**
- **Any questions about Ruby Koans?**
- **Any other questions?**
- **How do Ruby Koans work?**

## Ruby on Rails (Wikipedia)

- **Ruby on Rails, often abbreviated RoR, or just Rails**
  - open source web application framework written in Ruby
  - closely follows the Model-View-Controller (MVC) design pattern
  - strives for
    - Simplicity - convention over configuration
    - DRY – “don’t repeat yourself”
    - Real-world applications in less code than other frameworks
- **Ruby allows for extensive metaprogramming**
  - results in a syntax that many of its users find to be very readable
- **DHH**
  - Ruby on Rails was extracted by David Heinemeier Hansson (DHH) from his work on Basecamp
  - It was first released to the public in July 2004.
  - The current release is 4.0 but most people are using 3.2

JON
WALKER

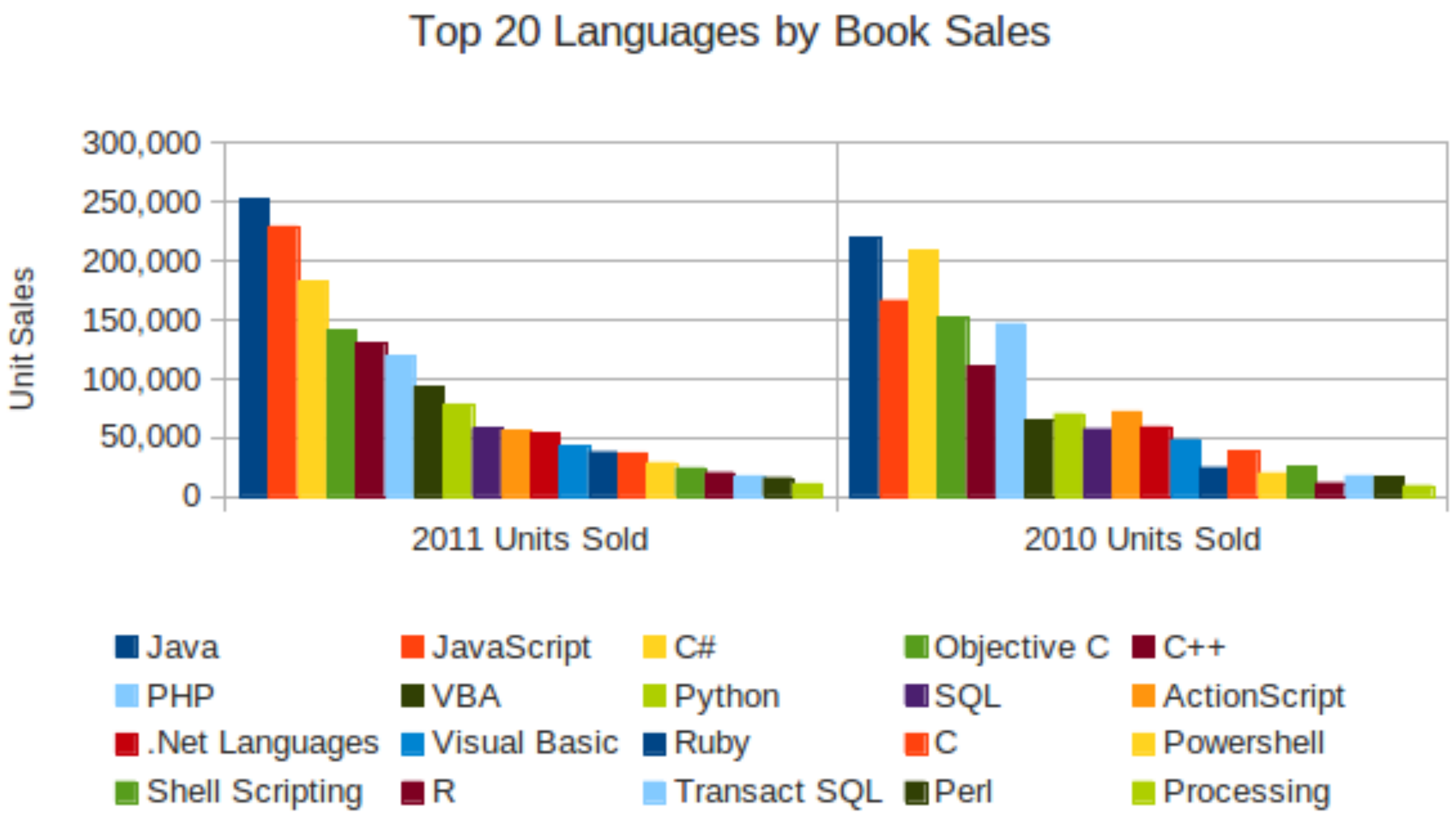
# Ruby

- **Ruby is a reflective, object-oriented programming language**
  - syntax inspired by Perl with Smalltalk-like object-oriented features
  - single-pass interpreted language
  - created by Yukihiro "Matz" Matsumoto, first released in 1995
  - Current stable version is 2.0.0
- **object-oriented**
  - every bit of data is an object, from integers to classes, even `nil`!
  - “duck” typing – huh?
  - single inheritance with dynamic dispatch, mixins and singleton methods
- **multi-paradigm programming language**
  - Procedural: functions/variables outside classes are part of the root 'self' Object
  - Object orientation: everything is an object
  - Functional: anonymous functions, closures, and continuations
  - introspection, reflection and meta-programming, threads
- **According to the Ruby FAQ**
  - "If you like Perl, you will like Ruby and be right at home with its syntax.
  - If you like Smalltalk, you will like Ruby and be right at home with its semantics.
  - If you like Python, you may or may not be put off by the huge difference in design philosophy between Python and Ruby."

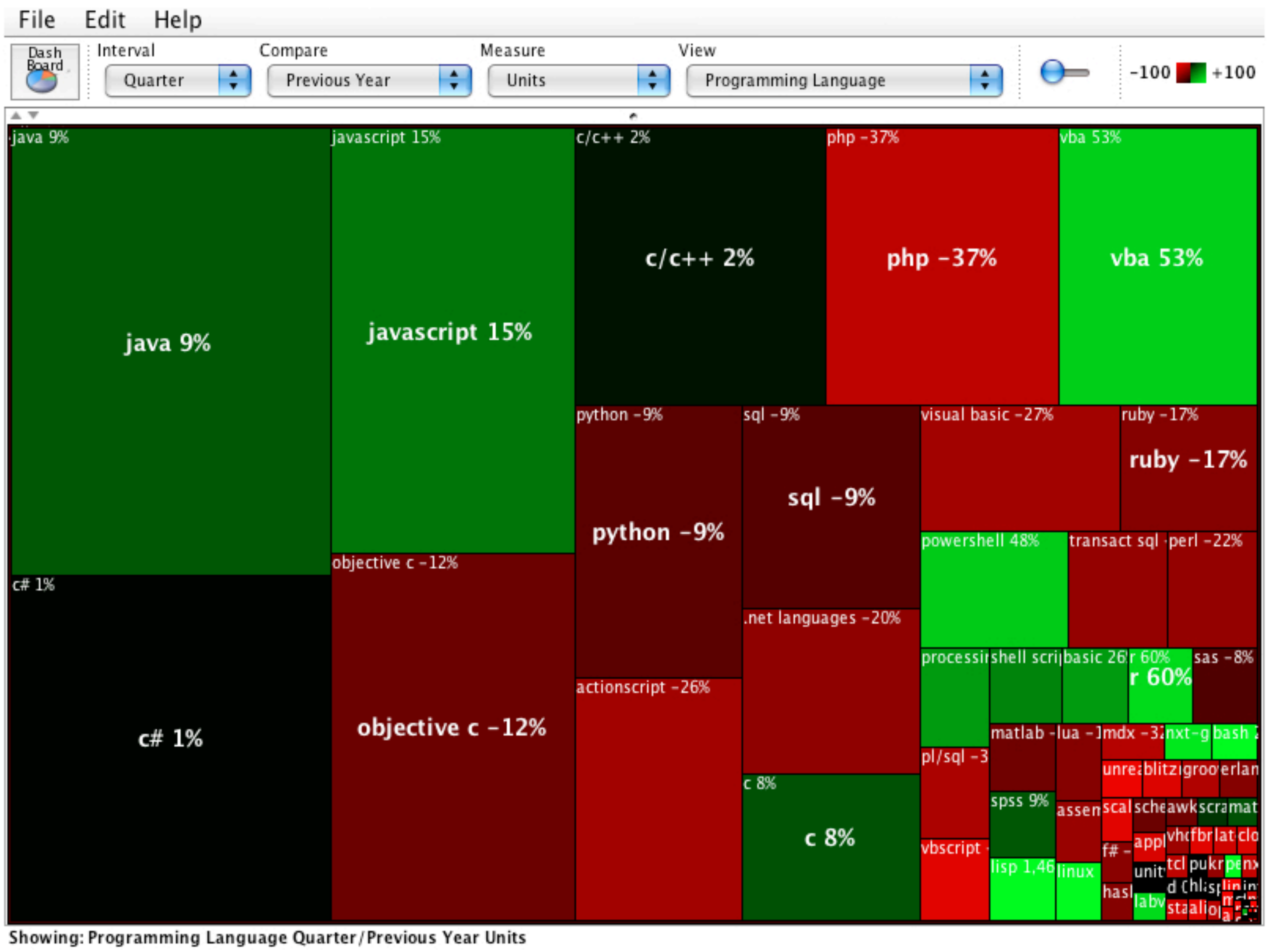
## Question

- Will Ruby be a popular language 10 years from now?

# Prog lang book trends 2012



# Prog lang book trends 2010



Interval: Quarter
Compare: Previous Year
Measure: Units
View: Programming Language

Showing: Programming Language Quarter/Previous Year Units

Note: includes books that are based on a specific prog lang

## RoR Shortcomings

- **Hidden complexity**
  - “Hitting the wall” when automagic stuff fails
  - Frequent change
- **Ruby is slow**
  - interpreter is still evolving
  - its dynamic nature makes it difficult to compile at all
  - need to use external libraries/processes for very compute-intensive things
- **Rails is single-threaded**
  - Run multiple instances, but need to work around pitfalls

## How to learn Ruby on Rails

- **Agile Web Development with Rails 3.2**
  - Dave Thomas and DHH
  - Available in PDF from Pragmatic Programmers
  - 30% class discount - discount code UCSBcs290b
- **Reference “Rdoc” from api.rubyonrails.org**
- **Programming Ruby**
  - Ruby Koans
    - http://rubykoans.com
  - Programming Ruby 2nd edition from Pragmatic Programmers
  - Ruby in 20 minutes
    - http://www.ruby-lang.org/en/documentation/quickstart/
- **Many other resources**
  - Resources page on www.cs290.com

## DB Migrations

- **Transform the database**

  -
    ```
    class FixRestaurants < ActiveRecord::Migration
      def change
        create_table :products do |t|
          t.string :title

          t.timestamps
        end
      end
    ```
- **Rake tasks (rake = ruby’s make)**
  - `Rake migrate` – applies all *new* migrations to database
  - Rails stores the latest migration version in a `schema_info` table
  - `Rake migrate VERSION=3` – applies migrations forwards or backwards to arrive at migration 3

## Model / View / Controller

- **A web application has 3 parts:**
  - The data model
    - Mapping between relational database and objects
    - Ensuring that the data remains valid & consistent

model

  - The business logic
    - Accepts a request, validates it, decides what happens next
    - Implements the core application logic
    - Fetches or calculates the data to be shown next

controller

  - The presentation layer – generating HTML (or …)
    - Produces the HTML mark-up from data provided by business logic
    - Often created by non-programmers (web designers)

view

## How many M’s, V’s, C’s?

- ***Generally …***
- **Models**
  - One model per database table
  - Use an explicit join table & model for many-to-many relationships
- **Views**
  - One view per response
  - Often view uses several *partials*, reusable page fragments
  - Standard scaffold creates 4 views + 1 partial per controller
- **Controllers**
  - One per model, sometimes more
  - One per list/show/edit/search page-set

## Rails scaffolding: 7 actions

- Index/list
  - List all items
- New
  - Display empty form for new item
- Create
  - Create item based on “new” form
- Show
  - Display item
- Edit
  - Display item in form for editing
- Update
  - Update item based on “edit” form
- Destroy
  - Delete item

## URLs, controllers, actions, IDs

- **Rails provides *routes* to map URLs to actions**
  - From routes.rb:

    ```
    get 'products/:id’, to: 'catalog#view'
    ```

    or

    ```
    # Note: This route will make all actions in every controller accessible via GET requests.
    # get ':controller(/:action(/:id(.:format)))’
    ```

    or

    ```
    resources :products
    ```
  - URLs are divided into parts by “/”

- **Query string or POST parameters**
  - Parsed by default into `params` hash
  - http://host/restaurants/list?city=isla+vista&state=ca
    - `params[:city] == “isla vista”`
    - `params[:state] == “ca”`

## Views – ERB - .html.erb

- **"Embed ruby into HTML"**

  -
    ```
    <% @restaurant.ratings.each do |rating| %>
      <p><span class="score"><%= rating.score %></span>
         by
         <span class="user"><%= rating.user.nickname %></span>
      </p>
    <% end %>
    ```
  - *Or*: embed HTML into ruby code!

  -
    ```
    Restaurant.ratings.each do |rating|
      puts '<p><span class="score">'
      puts h(rating.score)
      puts "</span>\nby\n<span class=\"user\">"
      puts h(rating.user.nickname)
      puts "</span>\n</p>"
    end
    ```
- **Security: avoid security issues**
  - `h(stuff)` -> escapes all HTML in `stuff`
  - Default in rails 3.0+ so you don't have to do this anymore

## View variables

- **Variables available in a view**
  - When an action terminates, an auto-generated wrapper calls `render :file =>` "*controller/action*"
    - Unless the action called `render` explicitly
  - `render` is a method of the controller
    - => all controller instance variables are in scope
    - hence idioms such as:
      `@restaurant = Restaurant.find(params[:id])`

## View partials

- **Partials: HTML page fragments**
  - `render :partial => 'form'`
    - Renders `/app/views/`*controller*`/_form.rhtml`
  - Passing in arguments

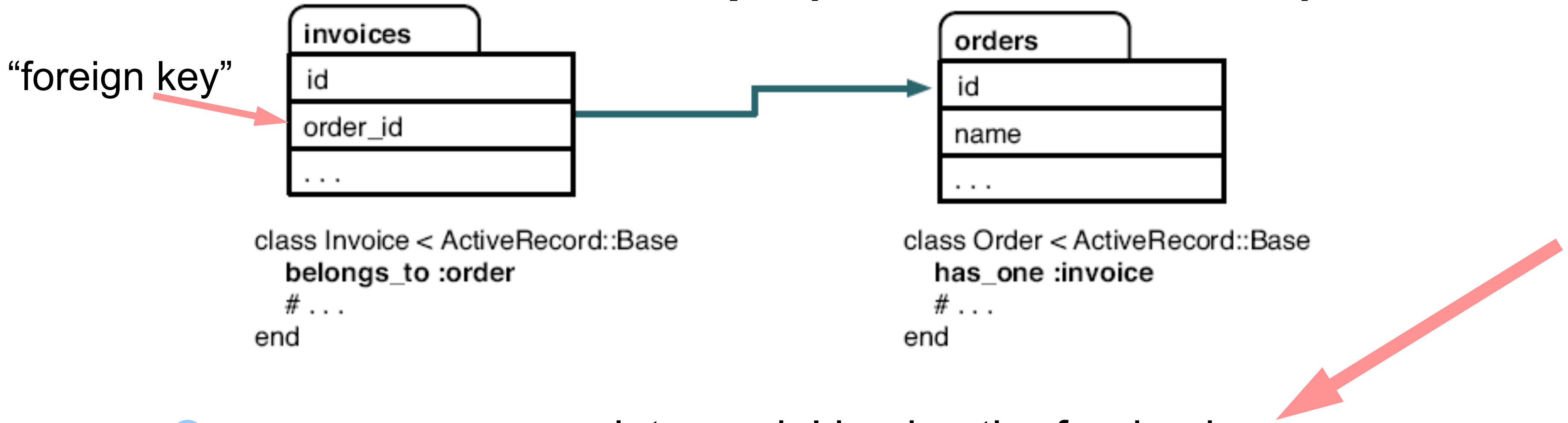
    -
      ```
      render :partial => 'form', :locals =>
        { :size => 10, :title => 'edit' }
      ```
  - Using a partial from another controller
    - `render :partial => 'shared/title', : locals => { … }`
    - Renders `/app/views/shared/_title.rhtml`
- **Options for `render` method**
  - See Rails Rdoc!

## Model validations

- **Assertions on a model's attributes**
  - checked on create and update
  - `validate :title, :description, presence: true`
  - Validate presence, uniqueness, numericality…
- **Validation failures**
  - Prevent the object from being saved to the database
  - Calls `errors.add` on object with error text
  - Access errors using:
    - `<%= error_messages_for(:product) %>`
  - See `ActiveModel::Errors`
- **Notes on model.`save` method**
  - *`model`*`.save` returns false if saving fails
  - *`model`*`.save!` raises a RecordNotSaved exception if saving fails

## Relationships

- **One-to-one relationships (one to zero-or-one)**



```
class Invoice < ActiveRecord::Base
   belongs_to :order
   # . . .
end
```

```
class Order < ActiveRecord::Base
   has_one :invoice
   # . . .
end
```

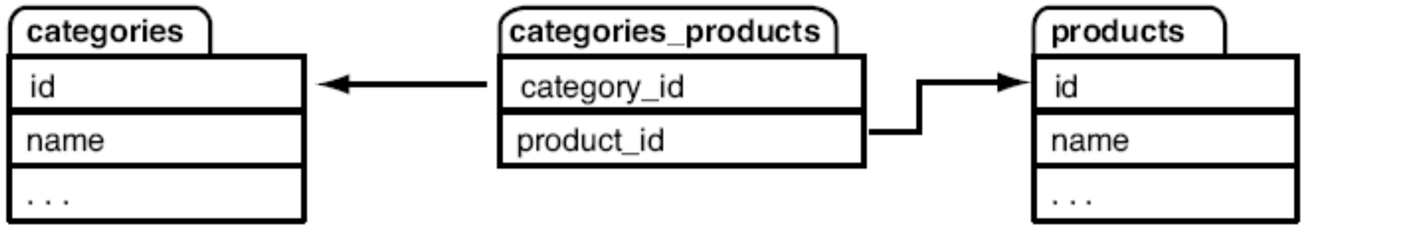
  - `belongs_to` goes into model having the foreign key
- **Rails creates accessor methods**
  - invoice = Invoice.find(1)
  - … = invoice.order.name
  - order = Order.find(23)
  - Order.invoice = …

## One-to-many relationships

- **Example:**

  -
    ```
    class Rating < AR::base              class Restaurant < AR::base
      belongs_to :restaurant               has_many :ratings
      # table has restaurant_id field
      …                                    …
    end                                  end
    ```
- **Rails creates accessor methods**
  - `@restaurant = find(params[:id])`
  - `@restaurant.ratings`
    - returns an array of rating objects
    - In a view, one might use:
      - `<% @ratings.each do |r| %>..<% end%>`
  - Watch out for database queries!

## Many-to-many relationships



```
class Category< ActiveRecord::Base
  has_and_belongs_to_many :products
  # . . .
end
```

```
class Product< ActiveRecord::Base
  has_and_belongs_to_many :categories
  # . . .
end
```

- **HABTM (has_and_belongs_to_many)**
  - Will use join table "behind the scenes"
  - `category.products` returns array of product objects
  - `product.categories << new_category`
    - "<<" operator is "append to array"
    - Don't forget `product.save`

## Explicit many-to-many

- **Through relationships using explicit join model**

  -
    ```
    class Categories < ActiveRecord::base
      has_many :categorizations
      has_many :products, :through => :categorizations
    end
    ```

  -
    ```
    class Products < ActiveRecord::base
      has_many :categorizations
      has_many :categories, :through => :categorizations
    end
    ```

  -
    ```
    class Categorization < ActiveRecord::base
      belongs_to :restaurant
      belongs_to :category
    end
    ```
  - `categorizations` table contains `foreign keys:`
    - `category_id` and `product_id`
  - `has_many` enables
    - `product.categorizations.collect{|c| c.category}`
  - `:through` enables `product.categories`

## Questions?

## Break

- 10 minute break

## Web services

- **Most common “protocols”:**
  - XML-RPC
  - REST (not really a protocol)
  - SOAP
  - … and JSON-RPC

## XML-RPC

- **XML-RPC: a remote procedure call protocol**
  - uses XML to encode its calls
  - uses HTTP as a transport mechanism
  - very simple
    - defines only a handful of data types and commands
    - the entire description can be printed on two pages of paper
    - see http://www.xmlrpc.com/spec
- **Data types**
  - Scalars: integer, double, boolean, date/time, string, base64, nil
  - Structures: struct, array
  - See http://en.wikipedia.org/wiki/XML-RPC

## Example

- **Users, magazines, subscriptions**
  - Data model:
    - Users table (name, address, …)
    - Magazines (name, publisher, …)
    - Subscriptions (user, magazine, start date, end date, …)
  - Subscription is a many-to-many relationship between users and magazines
- **Operations**
  - List user, show user, update user, create user, delete user
  - List magazine, show magazine, …
  - Subscribe, unsubscribe, …

## Users/magazines in XML-RPC

- **URI: /rpc**
- **Methods:**
  - Show_user(user_id) -> { name, email, … }
  - Update_user(user_id, new_name, new_email, …) ->
  - Subscribe(user_id, magazine_id)
  - Create_user, list_magazines, unsubscribe, …
    - <methodCall>
      - <methodName>create_user</methodName>
    - </methodCall>
- **Data:**
  - update user example:
    - <params>
      <param><value><integer>142</integer></value></param>
      <param><value><string>Jon Walker</string></value></param>
      <param><value><string>jwalker@cs.ucsb.edu</string></value></param>
      …
      </params>

## REST

- **Roy Fielding: Representational State Transfer**
  - Set of architectural principles for transferring information over the web
  - *REST* strictly refers to a collection of architectural principles
  - The term is also often used in a looser sense to describe any simple interface that uses XML (or YAML, JSON, plain text) over HTTP without an additional messaging layer such as SOAP

- **RESTful services in the HTTP+XML context**
  - Application state and functionality is exported as set of resources (?)
  - Every resource is uniquely addressable using a URL (remember: Uniform Resource Locator)
  - All resources share a uniform interface for the transfer of state between client and resource, consisting of:
    - 4 standard operations: GET, POST, PUT, DELETE
    - Content types defined as MIME types
  - The protocol is: stateless, cacheable, layered

## Users/magazines in REST

- **Resources ("nouns"):**
  - `/users` (the collection), `/user/<user_id>` (a specific user)
  - `/magazines`, `/magazine/<magazine_id>`
- **Operations ("verbs"):**
  - POST  = Create
    - POST /users -> create new user and return user_id
  - GET = Retrieve
    - GET /users -> list of users, GET /user/143 -> get one user
  - PUT = Update
    - PUT /user/143 -> update user 143
  - DELETE = Delete
    - DELETE /user/143 -> delete user 143
- **Note similarity to database CRUD operations**

## REST subscriptions

- **How does a user subscribe to a magazine?**
  - Need an additional resource type: subscription
  - `POST /subscriptions` to create new subscription
    - The data contains the user_id and the magazine_id
  - `DELETE /subscription/7489` to delete subscription
  - `GET /subscriptions/user/143` to get user 143’s subscriptions

## REST data

- **There is no standard data representation**
  - REST is not a protocol, it is an architecture style
  - REST suggests to use MIME types to drive representation
    - HTTP Content-Type header for data sent
    - HTTP Accept-Type header for requesting desired data type
  - Data should (must?) contain URLs to further resources

    -
      ```
      <user>
        <name>Jon Walker</name>
        <subscriptions>http://subscriptions.com/subscriptions/user/143
        </subscriptions>
      </user>
      ```
- **Data:**
  - update user example:

    -
      ```
      <params>
        <param><value><integer>142</integer></value></param>
        <param><value><string>Jon Walker</string></value></param>
        <param><value><string>jwalker@cs.ucsb.edu</string></value></param>
        …
      </params>
      ```

## SOAP

- **S.O.A.P. = Simple Object Access Protocol**
  - Now SOAP is no longer an acronym…
- **Typically**
  - Protocol to perform RPCs over HTTP
  - (SMTP transport also defined, not-RPC also possible)
- **Benefits**
  - Fully specified protocol, with many companion protocols
  - Flexible (arbitrary?) XML data representation
  - Supports lots of automation, lots of programming tools available
- **Drawbacks**
  - Complexity

## SOAP Example

- **Request**

  -
    ```
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
      <getProductDetails xmlns="http://warehouse.example.com/ws">
        <productID>827635</productID>
      </getProductDetails>
    </soap:Body> </soap:Envelope>
    ```
- **Response**

  -
    ```
    <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
      <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
        <getProductDetailsResult>
          <productName>Toptimate 3-Piece Set</productName>
          <productID>827635</productID>
          <description>3-Piece luggage set. Black Polyester.</description>
          <price>96.50</price>
          <inStock>true</inStock>
        </getProductDetailsResult>
    </getProductDetailsResponse> </soap:Body> </soap:Envelope>
    ```

## WSDL, UDDI, XML Schema, WS-Security, …

- **WSDL (“wiz-dull”)**
  - Web Services Description Language
  - Describes the public interface to the web service
    - the protocol bindings and message formats
    - supported operations and messages
  - W3C primer: http://www.w3.org/TR/wsdl20-primer/
- **UDDI**
  - Universal Description, Discovery, and Integration
  - Directory to locate services, returns WSDL descriptions
  - and much more…
- **XML Schema**
  - Formal language to define XML schemas, used for data representation in SOAP
  - W3C primer: http://www.w3.org/TR/xmlschema-0/
- **WS-Security**

## Amazon Web Services

- **Elastic Compute Cloud (EC2)**
  - “web hosting by Amazon”
  - Rent a “1.7Ghz” Xeon box w/1.7GB memory & 160GB disk
  - Anytime, starts within 3-5 minutes
  - $0.060/hr (Linux) $0.091/hr (windows) + $0.12/GB transferred out up to 10TB/month (first 1GB/month free)
  - Uses Xen virtual images
  - Elastic Block Store (EBS) volumes for persistent storage
- **Simple Storage Service (S3)**
  - “highly reliable and scalable data storage”
  - Web service interface to put and retrieve “objects”
  - Object: arbitrary text key + binary data up to 5TB (hashtable)
  - $0.14/GB/mo up to 1TB + $0.12/GB transferred out up to 10TB/month (first 1GB/month free)

## Announcements

- Did everyone get the email about the discount code for the book?
  - If not, make sure to send an email Yifan to add you to the list
- Did everyone get access to the class website?
- Assignments
  - Email .zip or .tar of your completed RubyKoans by Friday 10/4 to grading@cs290.com
  - Read Chapters 1-5+ of Agile Web Development With Rails (AWDWR)
  - Add project ideas to class website
- Text editor options?
  - TextMate
  - RubyMine
    - http://www.jetbrains.com/ruby/
    - Academic license $39 – mention cs290b at UCSB
  - Eclipse
    - Aptana Eclipse plugin (I haven't used this)
  - Rails VIM
- Lab – Phelps 1401
  - First lab on Monday 10/7 at 5pm
- GSWC – Friday Oct. 4th – Corwin Pavillion

## Questions?