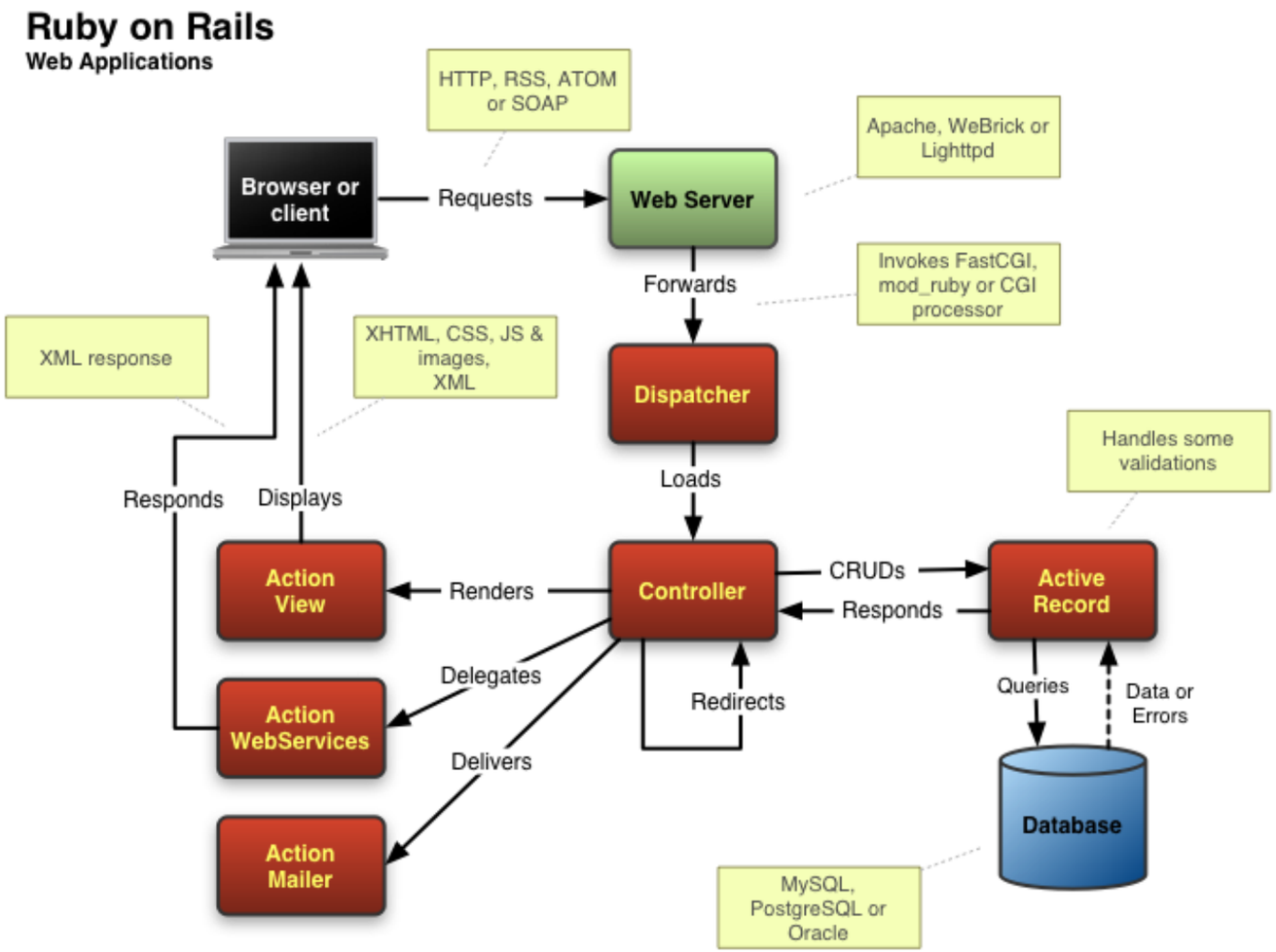
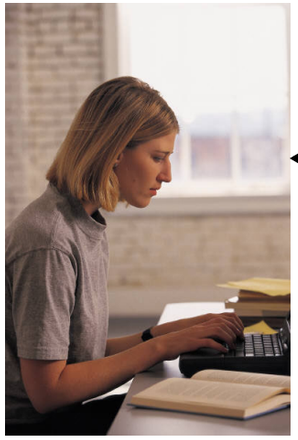
Rails: Models

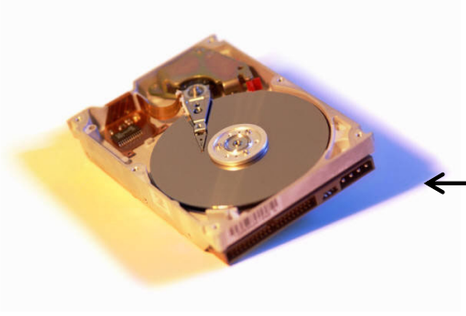
Lecture 27

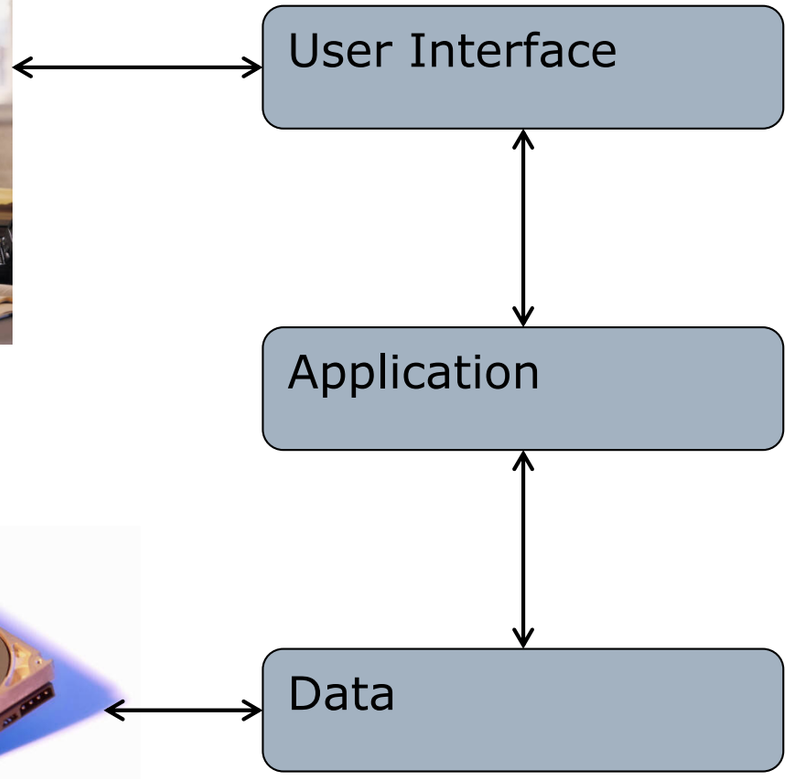
Rails Overview



Architecture: Desktop App







Graphical events (mouse moves, button pushed)

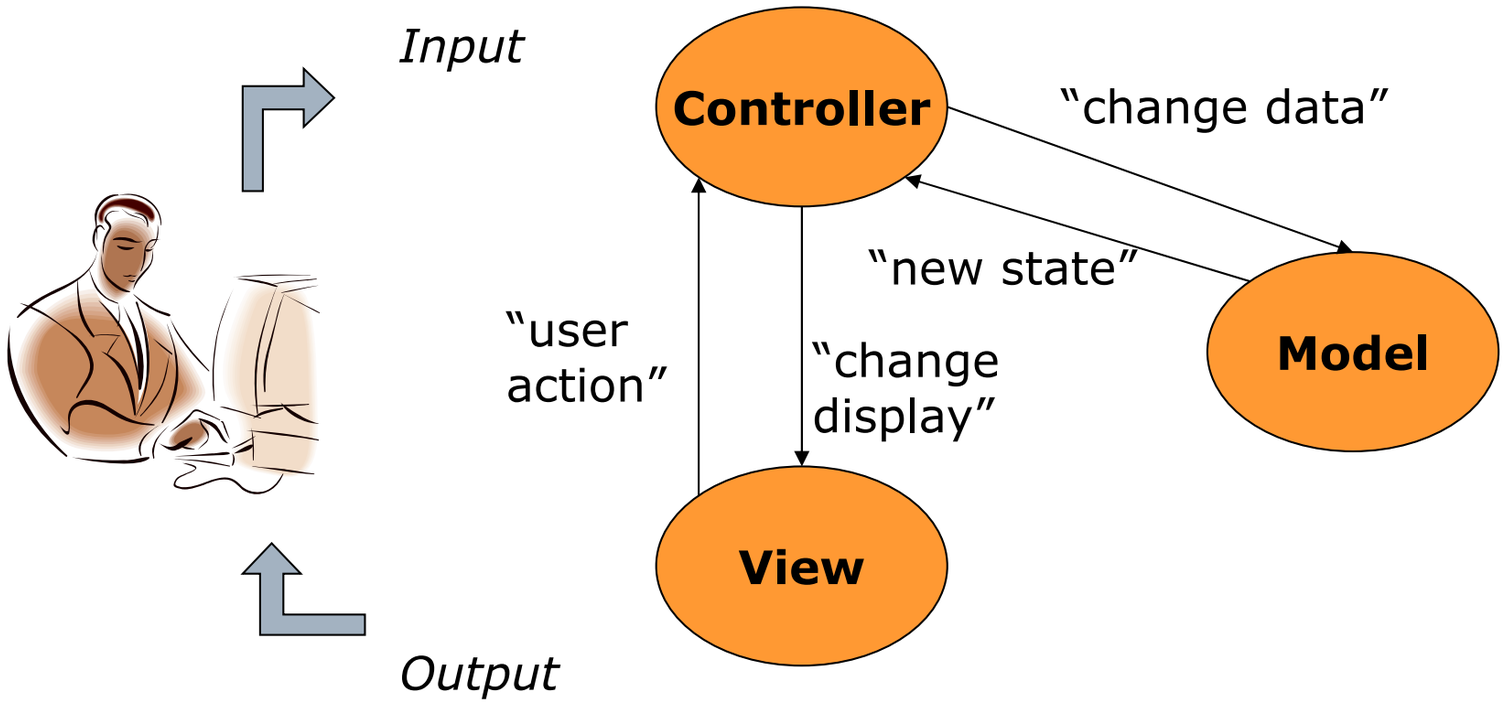
Processing, Calculating

Persistence, Transactions, Triggers

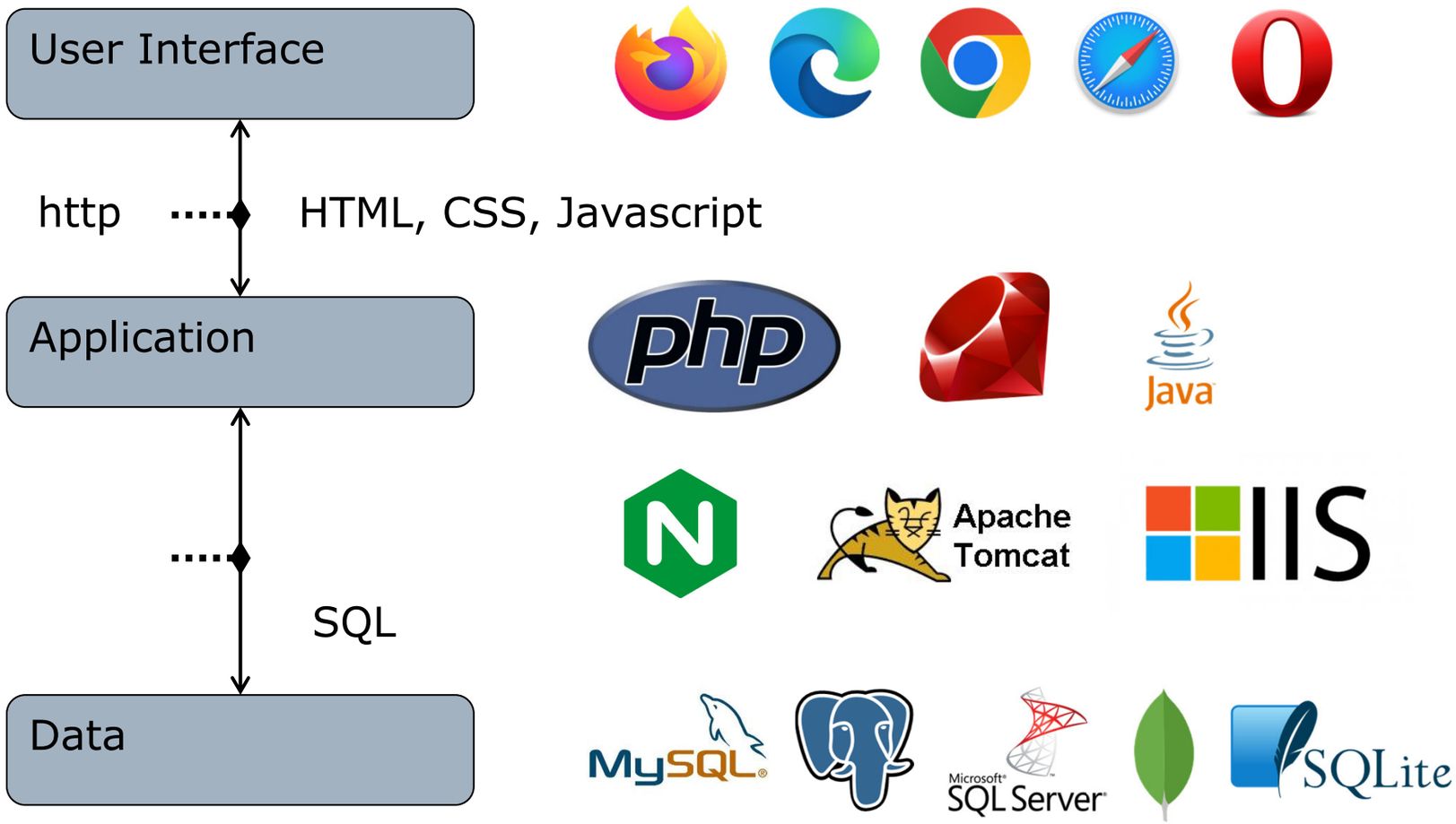
Model-View-Controller Pattern

- Model
  - The data (*i.e.* state)
  - Methods for accessing and modifying state
- View
  - Renders contents of model for user
  - When model changes, view must be updated
- Controller
  - Translates user actions (*i.e.* interactions with view) into operations on the model
  - Example user actions: button clicks, menu selections

Basic Interactions in MVC



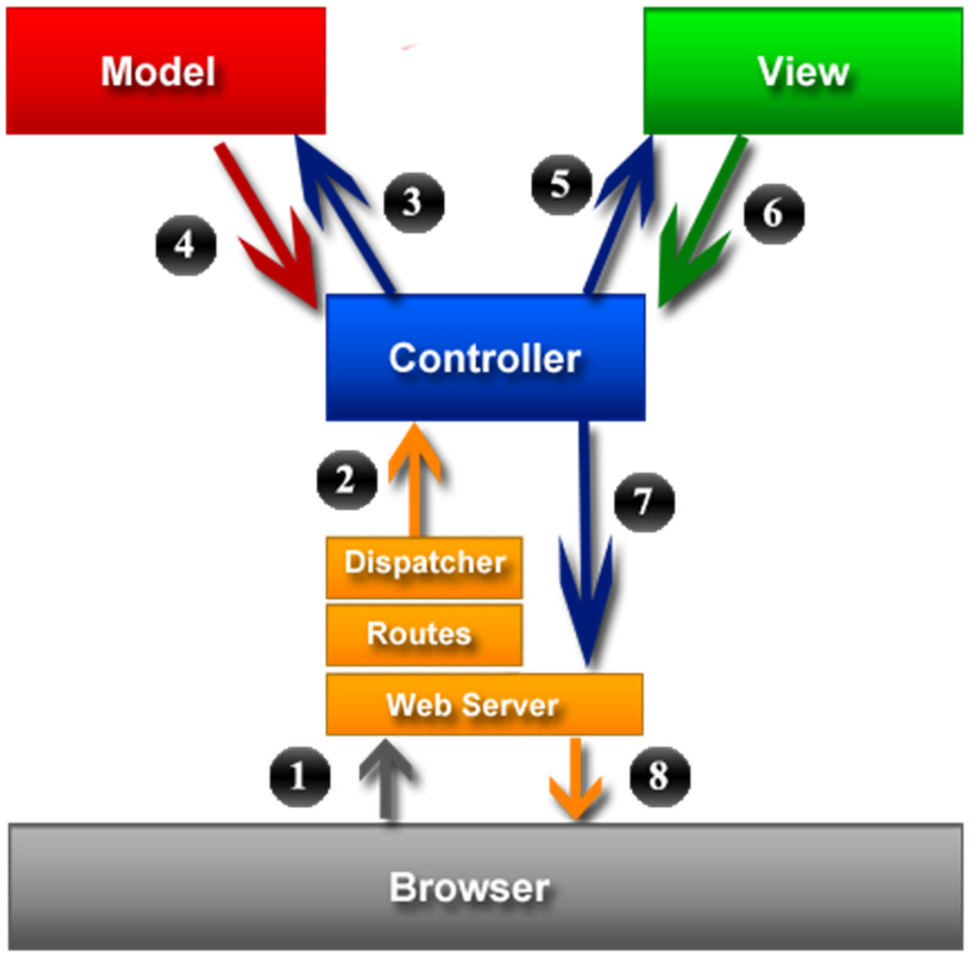
Basic Web App Skeleton: 3-Tier



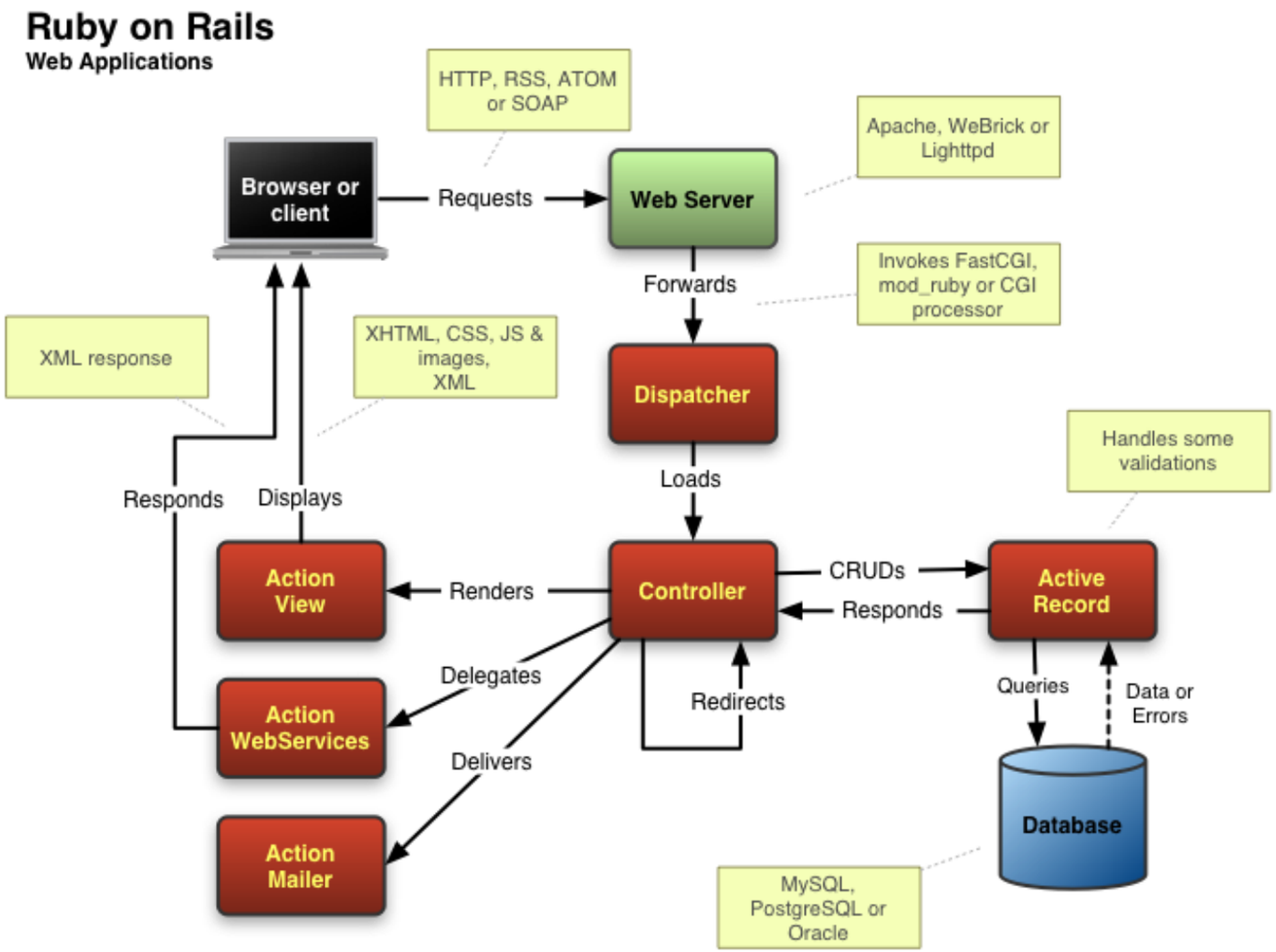
MVC in a Basic Web Application

- Model
  - Database (table with rows)
  - Classes that wrap database operations (class with instances)
- View
  - HTML (+ CSS, JavaScript) files rendered by client's browser
  - Skeleton files used by server to generate these HTML files
- Controller
  - Receives HTTP requests via web server
  - Orchestrates activity (model and view)

MVC with Rails: Major Components



MVC with Rails: More Details



Directory Structure of Rails: MVC


```
depot/
..../app
........./controllers
........./helpers
........./models
........./views
.............../layouts
..../bin
..../config
..../db
..../lib
..../log
..../public
..../storage
..../test
..../tmp
..../vendor
....Gemfile
....Rakefile
....README.md
```

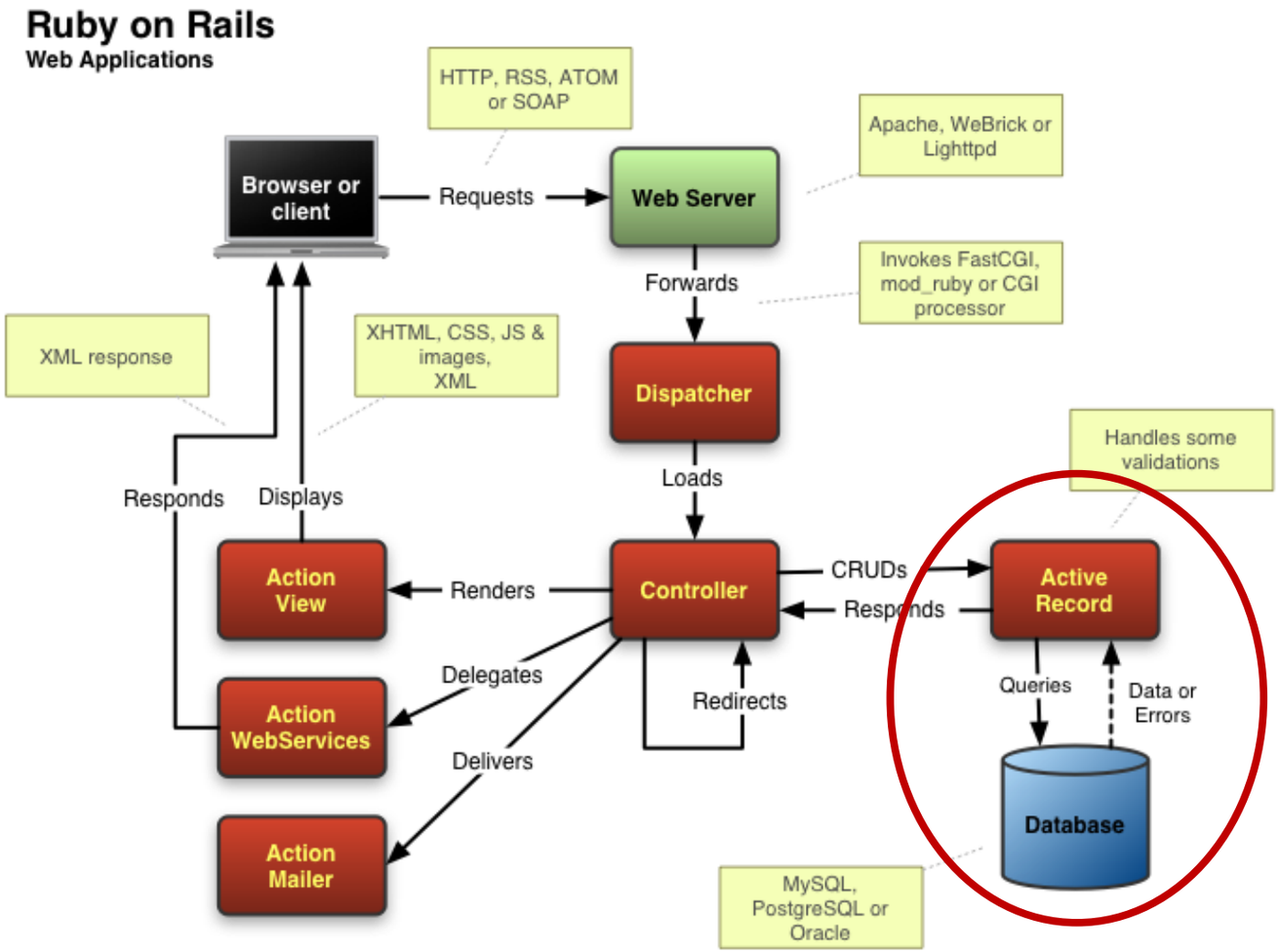
"Convention Over Configuration"

- Use naming & location conventions to wire components together *implicitly*
- Explicit routing too, based on *names* and pattern matching
- Contrast with:
  - Configuration files (*e.g.*, XML)
  - Configuration code (*e.g.*, Swing register listener)
  - Configuration tools (*e.g.*, IDEs to connect GUI widgets to code snippets)

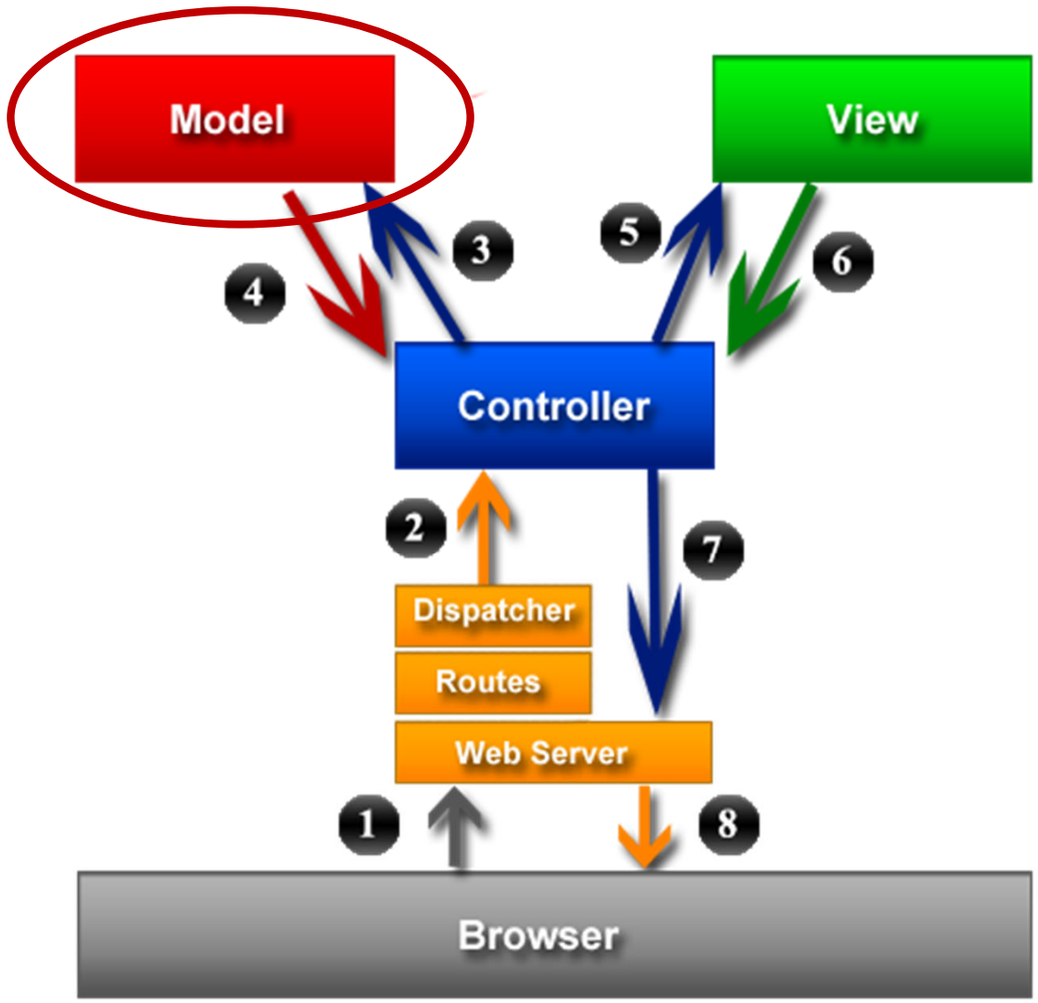
Wiring Parts Together in Rails

- Example: Event → Controller wiring
  - HTTP GET request for URL /say/hello gets routed to controller:
    - Class called `SayController`
    - File `say_controller.rb` in app/controllers
    - Method `hello`
- Example: Controller → View wiring
  - HTTP response formed from:
    - File app/views/say/hello.html.erb
- Example: Model → Database wiring
  - Class `Order` maps to database table "`orders`"
  - Attributes of `Order` map to *columns* of table
  - Instances of `Order` map to a *rows* of table

Models in Rails Architecture



Models in Rails Architecture (Alternate)



Mapping Tables to Objects

- General strategy for OO languages
  - Table in database -- a class
  - Table columns -- attributes of the class
  - Table rows -- instances of class (objects)
- Application works with database using ordinary language syntax
  - Class methods for finding row(s) in table
- Example: Java POJOs, Rails models

Database Tables

- A database is a collection of *tables*
  - Naming convention: Table names plural
- Each table has a list of *columns*
- Each column has a *name* and a *type*
- A table has a list of *rows*

students

fname (string)	lname (string)	buckid (integer)
Marco	Pantani	22352022
Primo	Carnera	334432
	Cher	34822039

Models

- Programmatic way for application to interact with database
  - Model = a Ruby class
  - Extends **`ApplicationRecord`**
  - Found in app/models
- Each class corresponds to a table
  - Note: Models are *singular* (tables are *plural*)
  - Includes attributes corresponding to columns *implicitly*

```
class Post < ApplicationRecord
  # attr_accessible :author,:title,:cont
end
```

Class Methods for Models

- Create a new instance with **new**

```
p1 = Post.new
p2 = Post.new author: 'Xi', title: 'Hola'
```

  - Warning: this only creates the model (object) it does *not* modify the database

- Create instance *and* add it to database

```
p3 = Post.create author: 'Zippy'
```

- Retrieve particular row(s) from table

```
p = Post.find 4 # search by id
p = Post.find_by author: 'Xi'
s = Student.find_by buckid: 543333
blog = Post.all
post = Post.first
post = Post.last
```

Instance Methods for Models

- To save a model (object) as a row in the database

```
p = Post.new author: 'Xi'
p.save # commits change to database
```

- Read/write attributes like an ordinary Ruby class

```
p = Post.find_by author: 'Xi'
p.title #=> nil
p.title = 'A Successful Project'
p.save # don't forget to save!
```

- To delete a row from the table

```
p.destroy # no save needed
```

Directory Structure of Rails: Models


```
depot/
..../app
........./controllers
........./helpers
........./models
........./views
.............../layouts
..../bin
..../config
..../db
..../lib
..../log
..../public
..../storage
..../test
..../tmp
..../vendor
....Gemfile
....Rakefile
....README.md
```

A Bit of Configuration

- Which database to use?
  - SQLite is the easiest (no setup!)
  - MySQL and PostgreSQL have better performance
- Different environments: development, test, production
  - Default (for rake command) is development
- See config/database.yml

```
default: &default
  adapter: sqlite3
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") {5} %>
  timeout: 5000
development:
  <<: *default
  database: db/development.sqlite3
```

Database Column Types

SQLite	Postgresql	MySQL
blob	bytea	blob
boolean	boolean	tinyint(1)
date	date	date
datetime	timestamp	datetime
decimal	decimal	decimal
float	float	float
integer	integer	int(11)
varchar(255)	character varying	varchar(255)
text	text	text
datetime	time	time
datetime	timestamp	datetime

Table Constraints

- Invariants on table entries beyond type information
  - "lname is not null"
  - "buckid is unique"
- Often useful to have a unique identifier for each row (a *primary key*)
  - Easy: Include an extra (integer) column
  - Database responsible for assigning this value every time a row is added
  - No way to change this value after creation

Primary Key With Autoincrement

students

id (key)	fname (string)	lname (string)	buckid (integer)
1	Marco	Pantani	22352022
3	Primo	Carnera	334432
4		Cher	34822039

Linking Tables

- Different tables can be related to each other
  - "Each student has exactly 1 major"
  - "Each student can own 1 (or more) vehicles"
- Keys are used to encode this relationship
  - Include a column in table X containing keys from table Y (*foreign keys*)
  - For examples:
    - Students table includes a column identifying a student's major
    - Vehicles table includes a column identifying a (student) owner
- Association is an invariant between tables

Association: Students & Vehicles

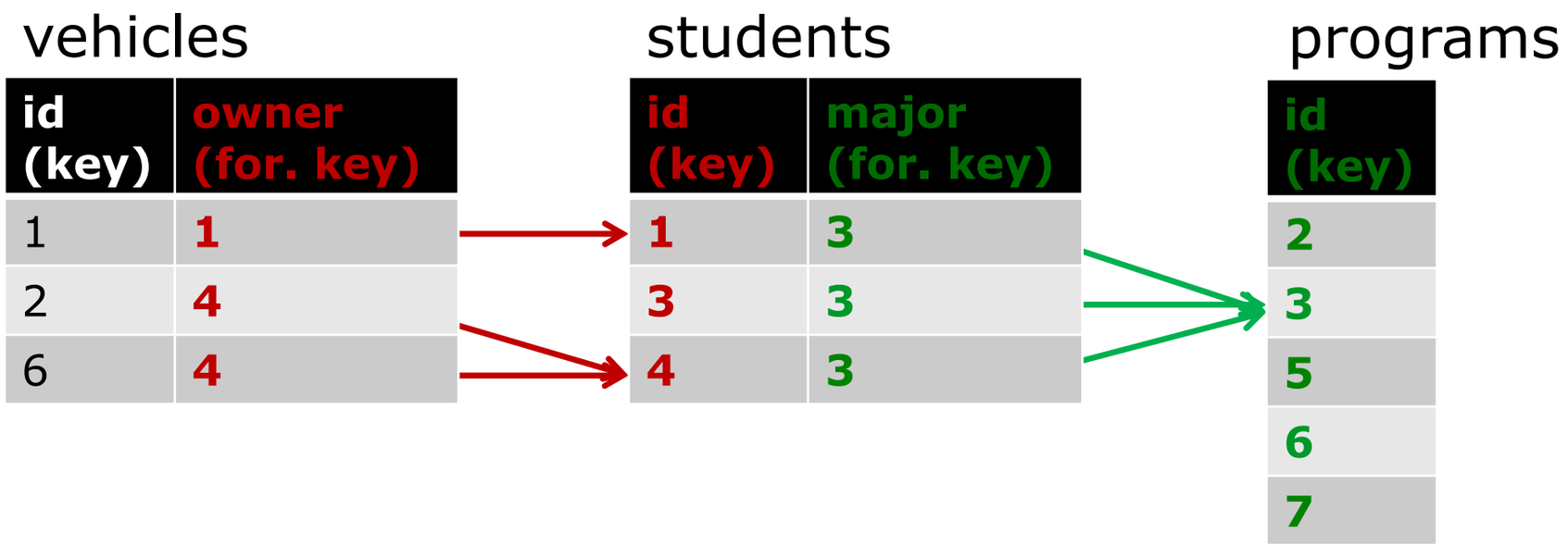
students

id (key)	fname (string)	lname (string)	buckid (integer)	major (foreign key)
1	Marco	Pantani	22352022	3
3	Primo	Carnera	334432	3
4		Cher	34822039	3

vehicles

id (key)	owner (foreign key)	license (string)
1	1	K3F 443L
2	4	F8L 220J
6	4	GOHBUX

Associations



Schema

- Definition of table structure
  - Table name
  - Column names and types
  - Constraints
- Usually database manager-specific
- See db/schema.rb for *Ruby-based* schema description
  - Allows independence from particular DB manager
  - Schema is versioned by timestamp (really by *migration*…)

Example schema.rb


```
ActiveRecord::Schema.define(version:
                            2023_03_19_144259) do

  create_table "students", force: :cascade
                           do |t|
    t.string   "fname"
    t.string   "lname"
    t.integer  "buckid"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
  end

end
```

Migrations

- Q. Who writes schema.rb?
  - A. It is generated!
  - Golden rule: Never edit schema.rb directly
  - Instead, write a *migration*
- A migration is Ruby code (a class) that represents a *change* in schema
  - Create new tables (including column names and column types)
  - Modify existing tables (adding/removing columns, or changing associations)
  - Delete ("drop") existing tables

Migration Classes

- See db/migrate
- Filename consists of
  - Timestamp (UTC) of creation
  - Class name (descriptive of delta)
  - Example: class **`CreatePosts`** in 20230319145307_create_posts.rb
- Consequence: Migrations are run in a consistent order
  - Deltas do not commute, so order is important
- Class extends **`ActiveRecord::Migration`**
  - Contains method change
  - This method invoked by **`rails db:migrate`**

Example Migration Class


```
class CreatePosts < ActiveRecord::Migration
  def change
    create_table :posts do |t|
      t.string :name
      t.string :title
      t.text :content

      t.timestamps
    end
  end
end
```

Result of Running This Migration

:posts

:id (key)	:name (string)	:title (string)	:content (text)	:created_at (datetime)	:updated_at (datetime)

Column Type Mappings

Migration	Ruby	SQLite	Postgresql	MySQL
:binary	String	blob	bytea	blob
:boolean	Boolean	boolean	boolean	tinyint(1)
:date	Date	date	date	date
:datetime	Time	datetime	timestamp	datetime
:decimal	BigDecimal	decimal	decimal	decimal
:float	Float	float	float	float
:integer	Integer	integer	integer	int(11)
:string	String	varchar(255)	character varying	varchar(255)
:text	String	text	text	text
:time	Time	datetime	time	time
:timestamp	Time	datetime	timestamp	datetime

Schema Deltas In Migrations

- In addition to creating tables, the change method can also *change* existing tables
  - Modify columns of an existing table

    `add_column, remove_column, rename_column, change_column`
  - Modify and delete tables

    `change_table, drop_table`
- Example: xxx_add_author_to_posts.rb

```
class AddAuthorToPosts <
                    ActiveRecord::Migration
  def change
    add_column :posts, :author, :string
  end
end
```

Migrations as History

- Change defined by migration can be undone
  - Migrations give a *linear* history of deltas
  - Schema is the result of applying them (in order)
- Can move forward/backward in history
  - Create database only (no schema) defined in config/database.yml

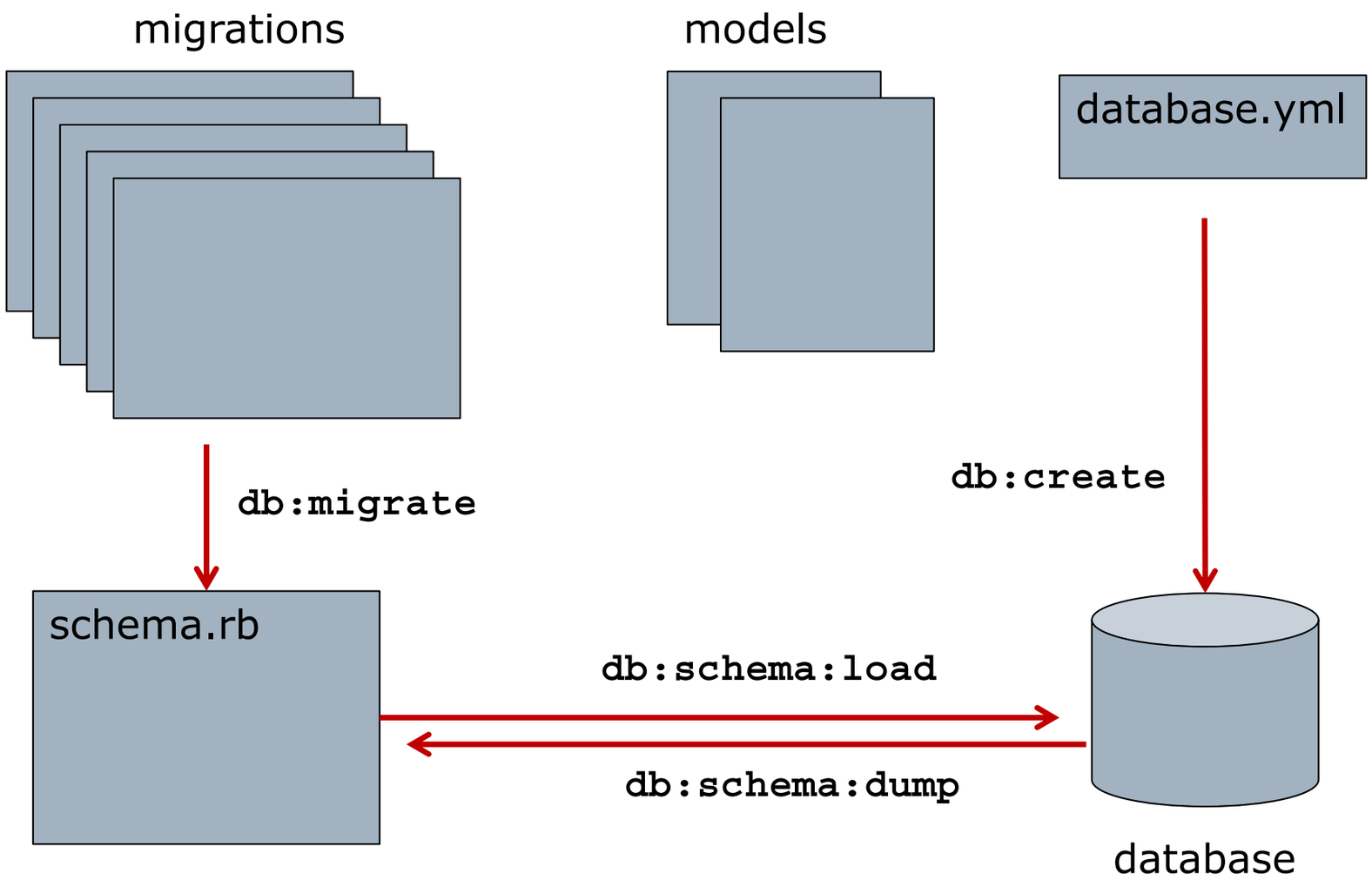
    ```
    $ rails db:create
    ```
  - Update schema.rb (compare its version number to list of migrations) and apply to database

    ```
    $ rails db:migrate
    ```
  - Rollback schema.rb to earlier point in history

    ```
    $ rails db:rollback
    ```
  - Load schema defined in db/schema.rb

    ```
    $ rails db:schema:load
    ```

Schemas, Migrations, Models



Migrations vs Schema

- Golden rule: Never edit schema.rb
  - It is regenerated every time you do a migration
  - *Every* change in schema means writing a migration
- Commit schema.rb to version control
  - Deployment in fresh environment means loading schema, not reliving the full migration history
- Commit migrations to version control
  - Once a migration has been shared, to undo it you should create a *new* migration (preserve the linear history)

Summary: Model Basics

- Databases: Tables, columns, rows
  - Structure defined in a schema
  - Rails uses Ruby code to generate schema
- Models
  - Ruby classes that mirror database tables
  - Class names from table (singular vs plural)
  - Attributes from columns
- Migrations
  - Ruby code describing change to schema
  - Syntax look declarative