WHAT THE USER WANTS...? AND 3 TIERED SHARED NOTHING ARCHITECTURES

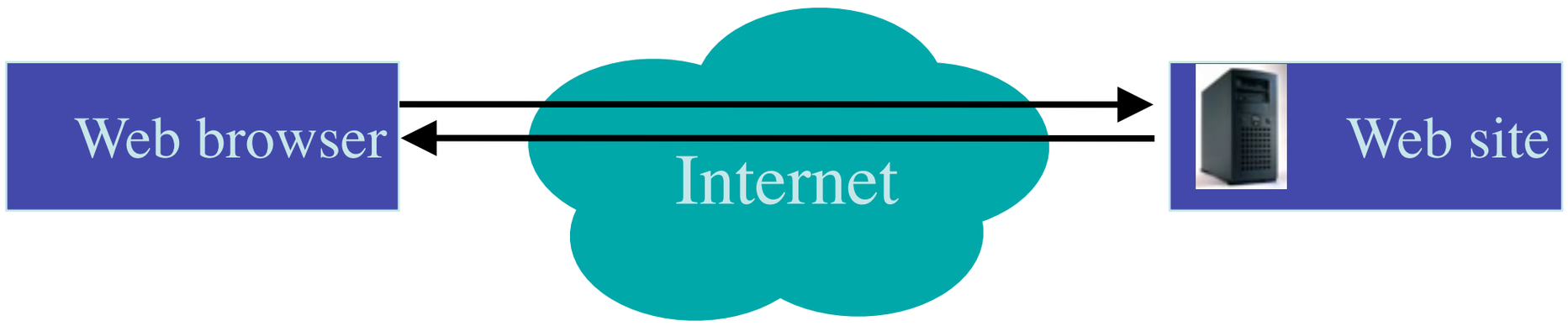
ANNOUNCEMENTS

- Project Iter 0-2 due SUNDAY
  - Lo-fi UI diagrams, storyboards
  - Key user stories
- Homework 2 due Monday 2/20— start early!
  - Gain understanding of MVC arch
  - Get used to the Rails environment
  - Deploy to the cloud

PICKING AND UNDERSTANDING ARCHITECTURES

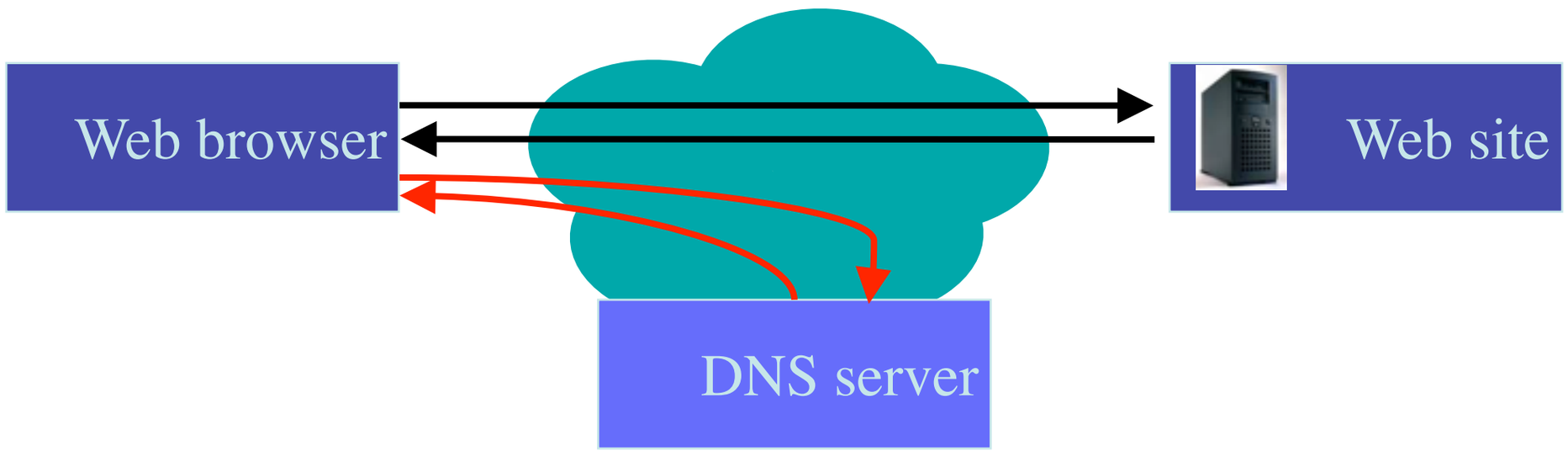
WEB AT 100,000 FEET

- The web is a c*lient/server* architecture
- It is fundamentally *request/reply oriented*



WEB AT 100,000 FEET

- The web is a *client/server* architecture
- It is fundamentally *request/reply oriented*
- Domain Name System (DNS) is another kind of server that maps *names* to *IP addresses*

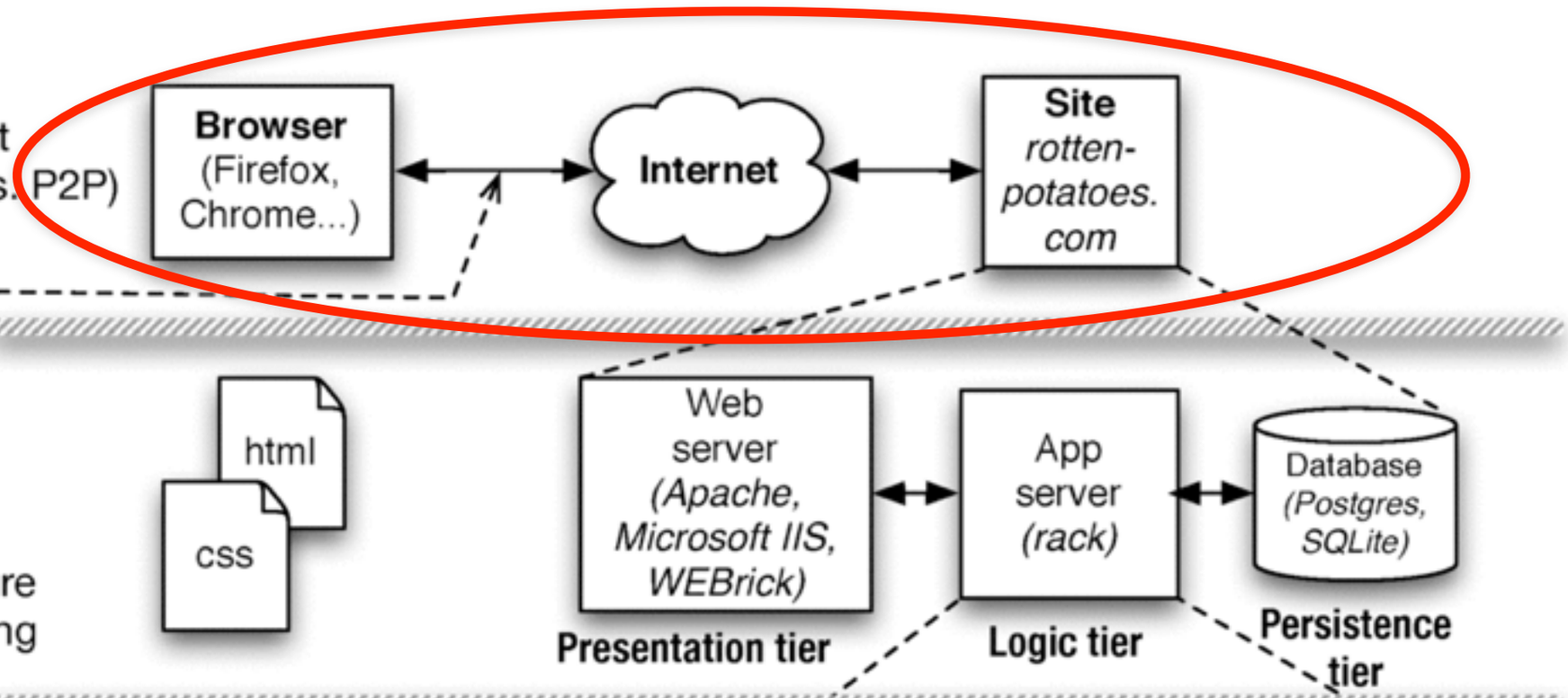


§2.1 100,000 feet
• Client-server (vs. P2P)

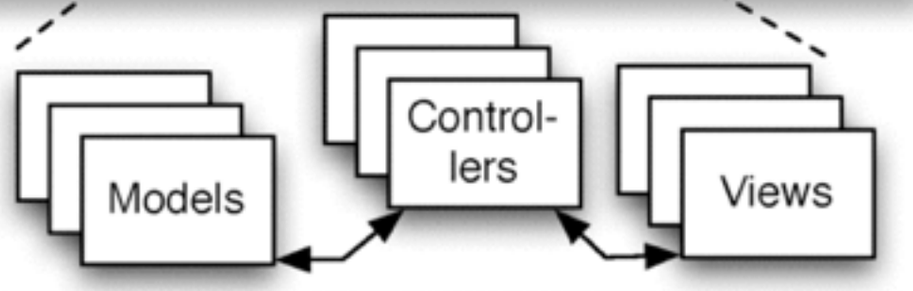
§2.2 50,000 feet
• HTTP & URIs

§2.3 10,000 feet
• XHTML & CSS

§2.4 5,000 feet
• 3-tier architecture
• Horizontal scaling



§2.5 1,000 feet—Model-View-Controller
(vs. Page Controller, Front Controller)



§2.6 500 feet: Active Record models (vs. Data Mapper)
§2.7 500 feet: RESTful controllers (Representational State Transfer for self-contained actions)
§2.8 500 feet: Template View (vs. Transform View)

- **Active Record**
- Data Mapper

- **REST**

- **Template View**
- Transform View

NOW THAT WE'RE TALKING, WHAT DO WE SAY?
HYPERTEXT TRANSFER PROTOCOL

- an *ASCII-based request/reply protocol* for transferring information on the Web
- *HTTP request* includes:
  - *request method* (`GET`, `POST`, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - *headers*—extra info regarding transfer request
- *HTTP response* from server
  - Protocol version & Status code =>
  - Response headers
  - Response body

HTTP status codes:
2xx — *all is well*
3xx — *resource moved*
4xx — *access problem*
5xx — *server error*

TRY THIS OUT

- In a terminal: nc –l 8000 (listen on port 8000)
- In web browser: localhost:8000/la/di/da
- Back to terminal: See what the browser got back- URI it wanted, protocol being used, some cookies (next week), now waiting- you are now playing the web server.
- In the terminal, type “Hello World”
- Back to browser: “Hello World” will appear

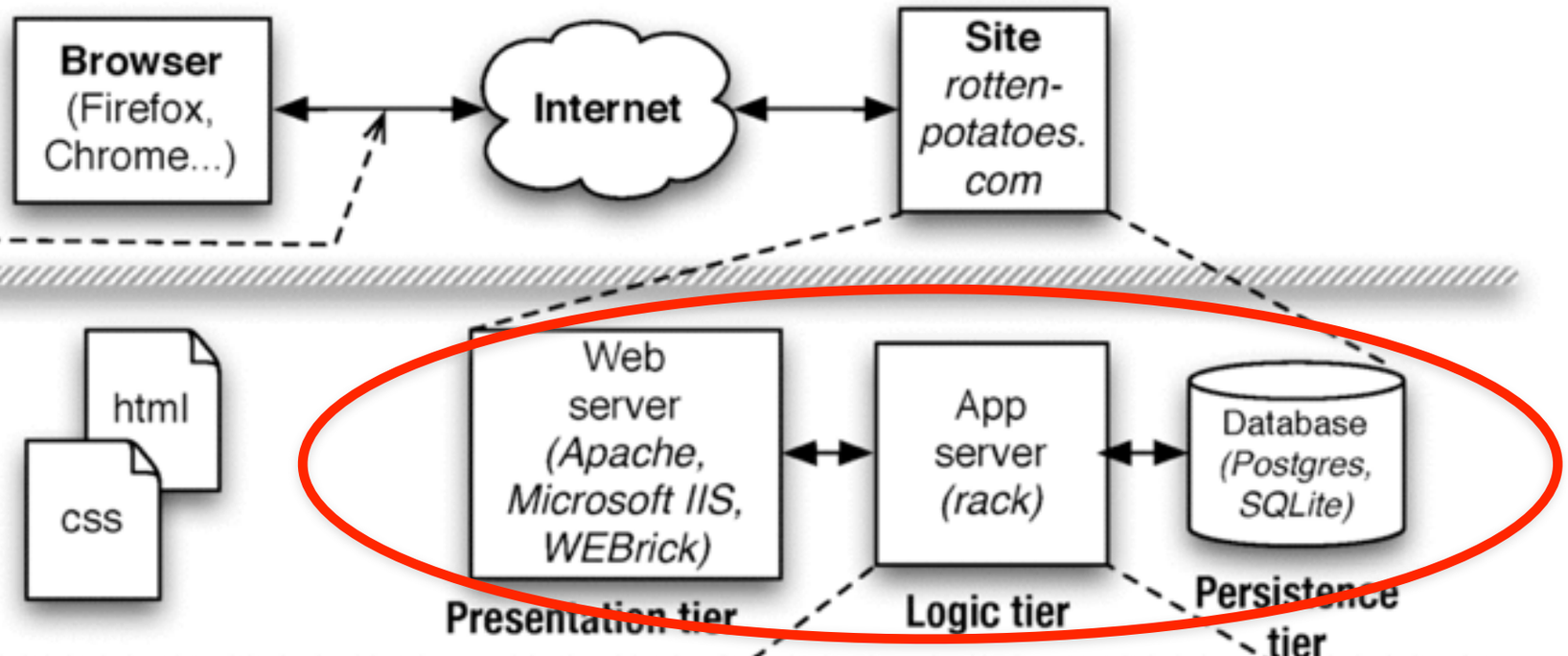
3-TIER SHARED-NOTHING ARCHITECTURE & SCALING

§2.1 100,000 feet
• Client-server (vs. P2P)

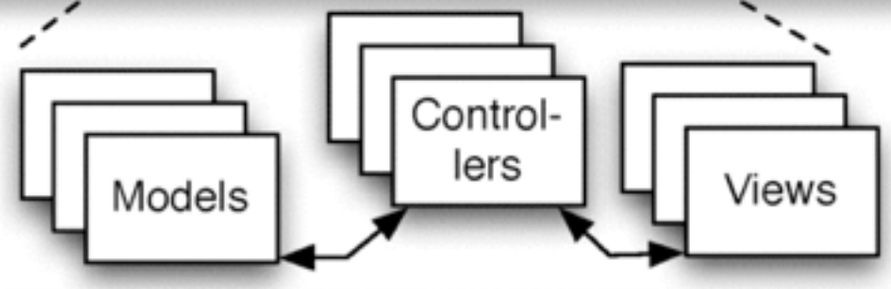
§2.2 50,000 feet
• HTTP & URIs

§2.3 10,000 feet
• XHTML & CSS

§2.4 5,000 feet
• 3-tier architecture
• Horizontal scaling



§2.5 1,000 feet—Model-View-Controller
(vs. Page Controller, Front Controller)



§2.6 500 feet: Active Record models (vs. Data Mapper)
§2.7 500 feet: RESTful controllers (Representational State Transfer for self-contained actions)
§2.8 500 feet: Template View (vs. Transform View)

- **Active Record**
- Data Mapper

- **REST**

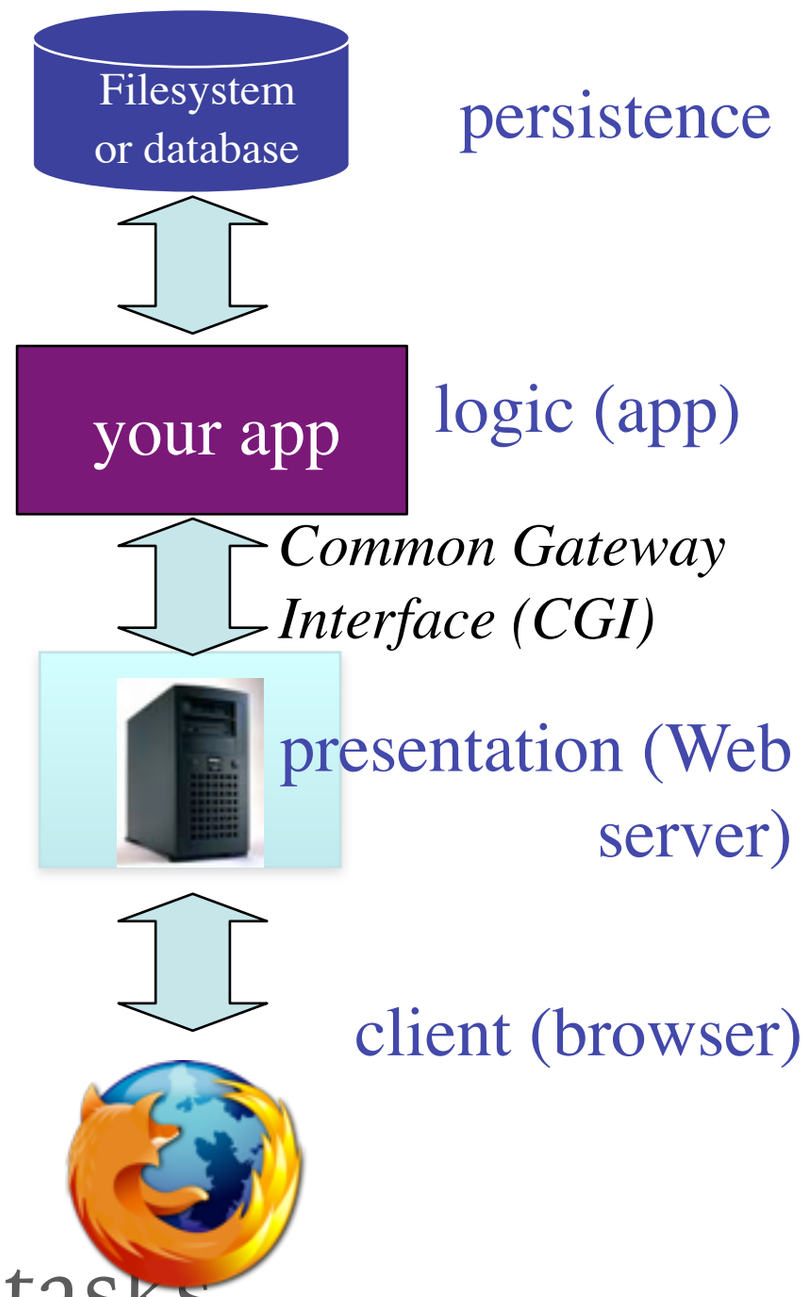
- **Template View**
- Transform View

DYNAMIC CONTENT GENERATION

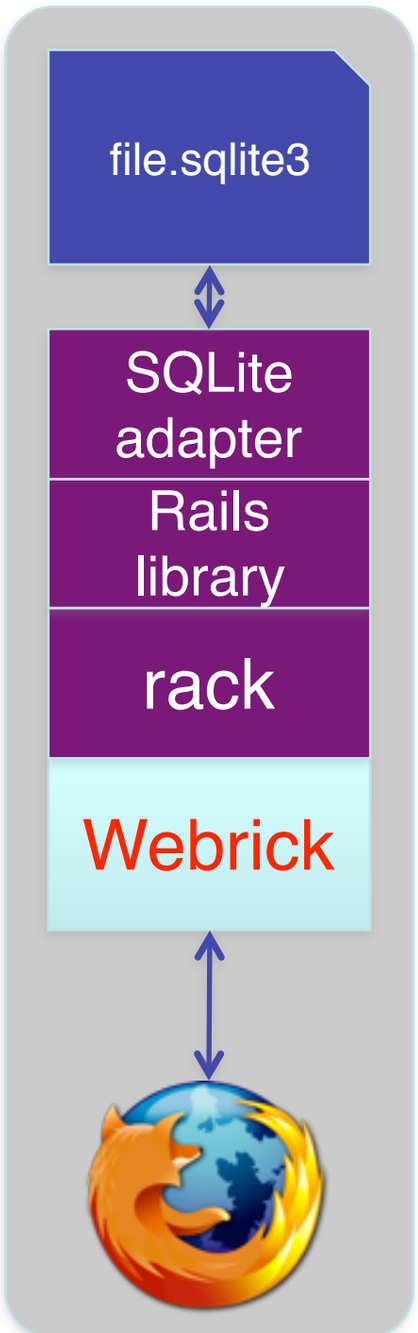
- In the Elder Days, most web pages were (collections of) plain old files
- But most interesting Web 1.0/e-commerce sites actually *run a program* to generate the "page"
- Originally: templates with embedded code "snippets"
- Eventually, code became "tail that wagged the dog" and moved out of the Web server

SITES THAT ARE REALLY PROGRAMS (SAAS)

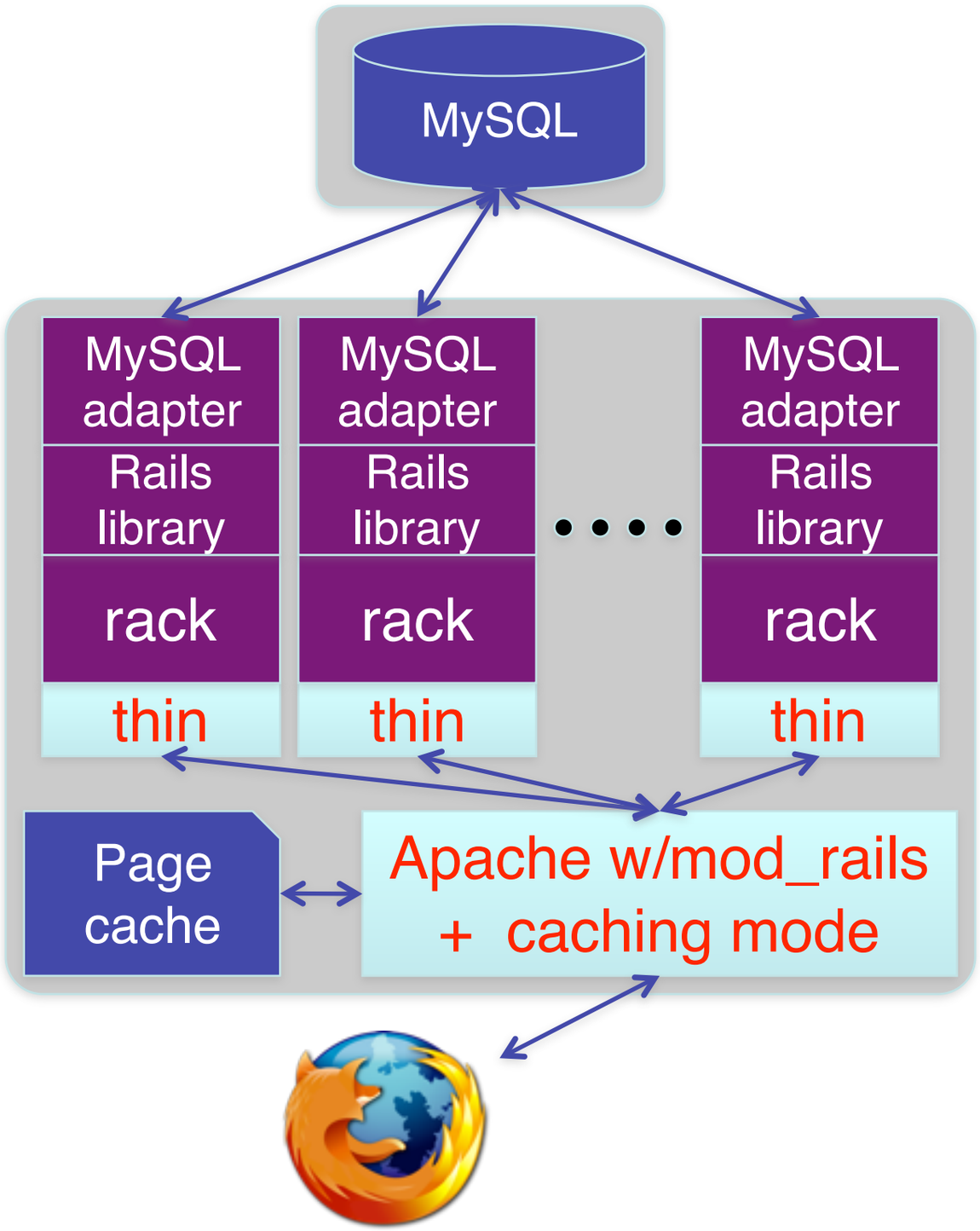
- How do you:
  - "map" URI to correct program & function?
  - pass arguments?
  - invoke program on server?
  - handle persistent storage?
  - handle cookies?
  - handle errors?
  - package output back to user?
- *Frameworks* support these common tasks



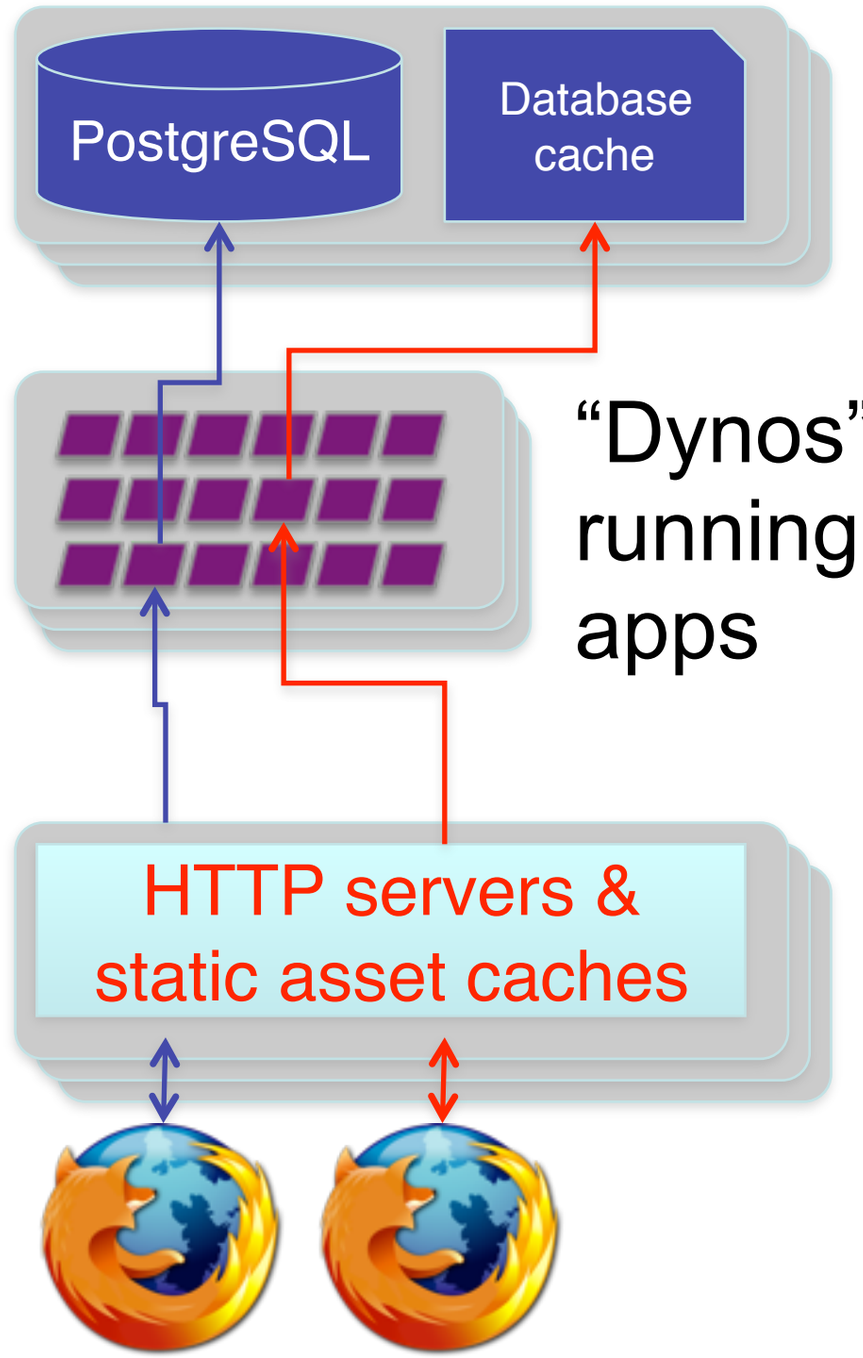
DEVELOPER ENVIRONMENT VS. MEDIUM-SCALE DEPLOYMENT



Developer



Medium-scale deployment



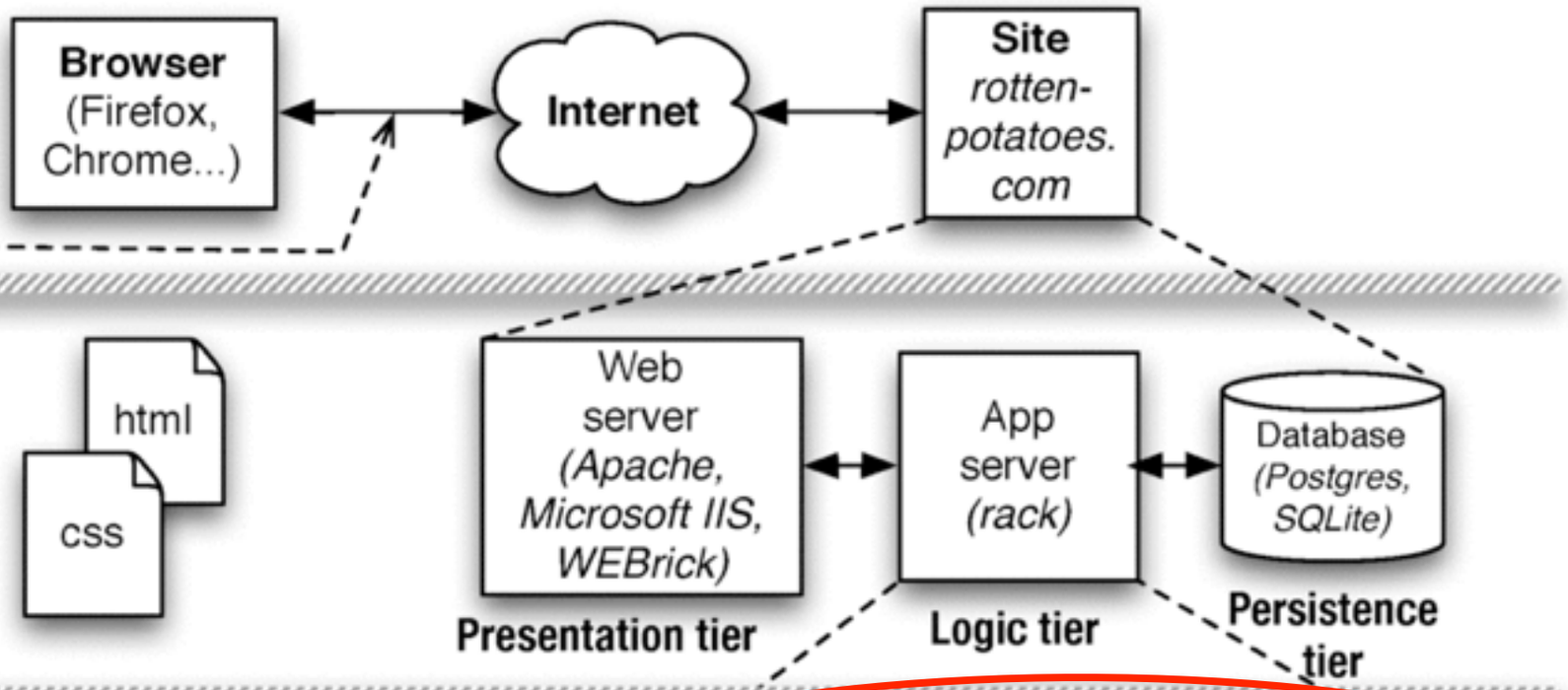
Large-scale curated deployment, e.g. Heroku

§2.1 100,000 feet
- Client-server (vs. P2P)

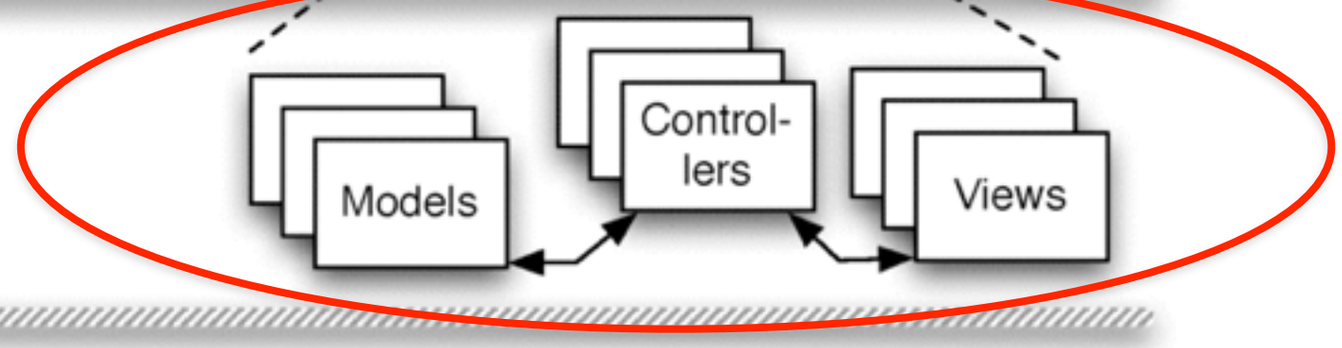
§2.2 50,000 feet
- HTTP & URIs

§2.3 10,000 feet
- XHTML & CSS

§2.4 5,000 feet
- 3-tier architecture
- Horizontal scaling



§2.5 1,000 feet—Model-View-Controller
(vs. Page Controller, Front Controller)



§2.6 500 feet: Active Record models (vs. Data Mapper)
§2.7 500 feet: RESTful controllers (Representational State Transfer for self-contained actions)
§2.8 500 feet: Template View (vs. Transform View)

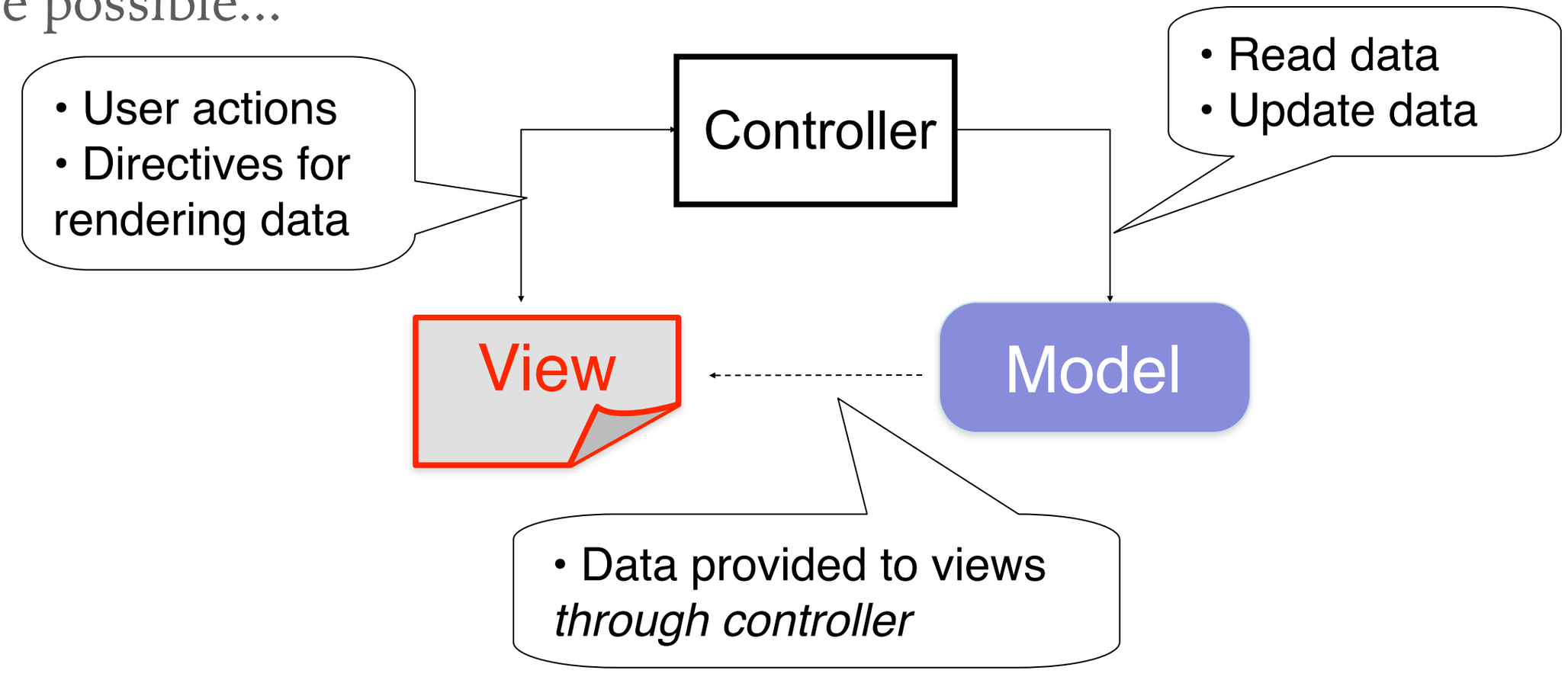
- **Active Record**
- Data Mapper

- **REST**

- **Template View**
- Transform View

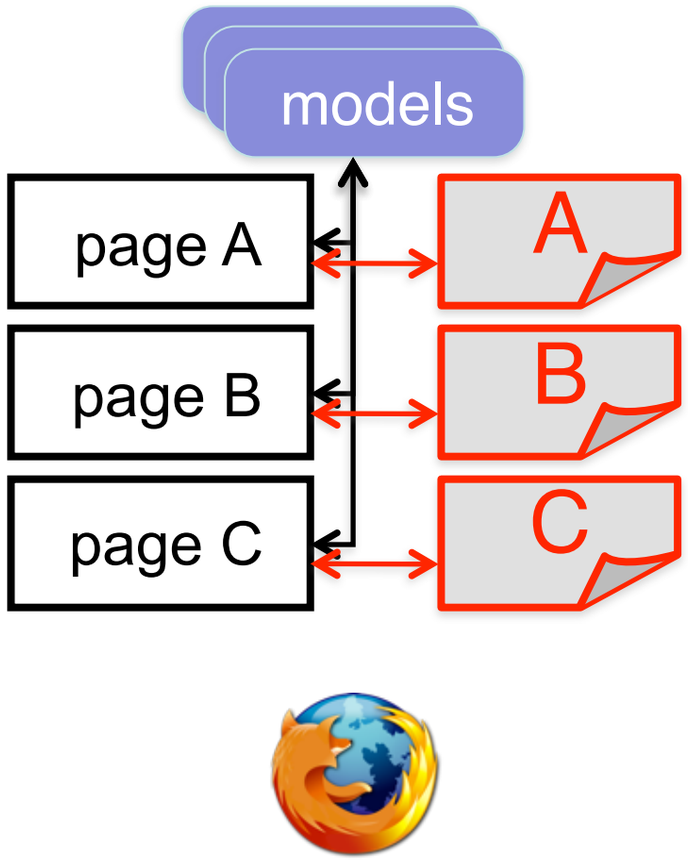
THE MVC DESIGN PATTERN

- Goal: separate organization of data (model) from UI & presentation (view) by introducing *controller*
  - mediates user actions requesting access to data
  - presents data for *rendering* by the view
- Web apps may seem "obviously" MVC by design, but other alternatives are possible...

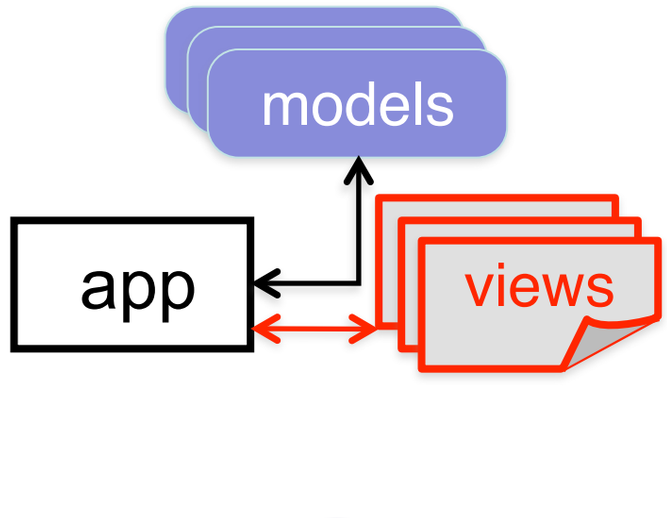


ALTERNATIVES TO MVC

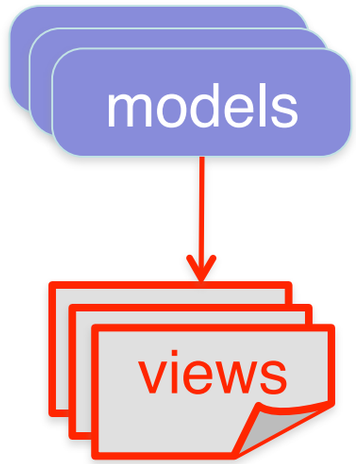
Page Controller
(Ruby Sinatra)



Front Controller
(J2EE servlet)



Template View
(PHP)



Rails supports SaaS apps structured as MVC, but other architectures may be better fit for some apps.

INTRODUCTION TO BEHAVIOR-DRIVEN DESIGN AND USER STORIES

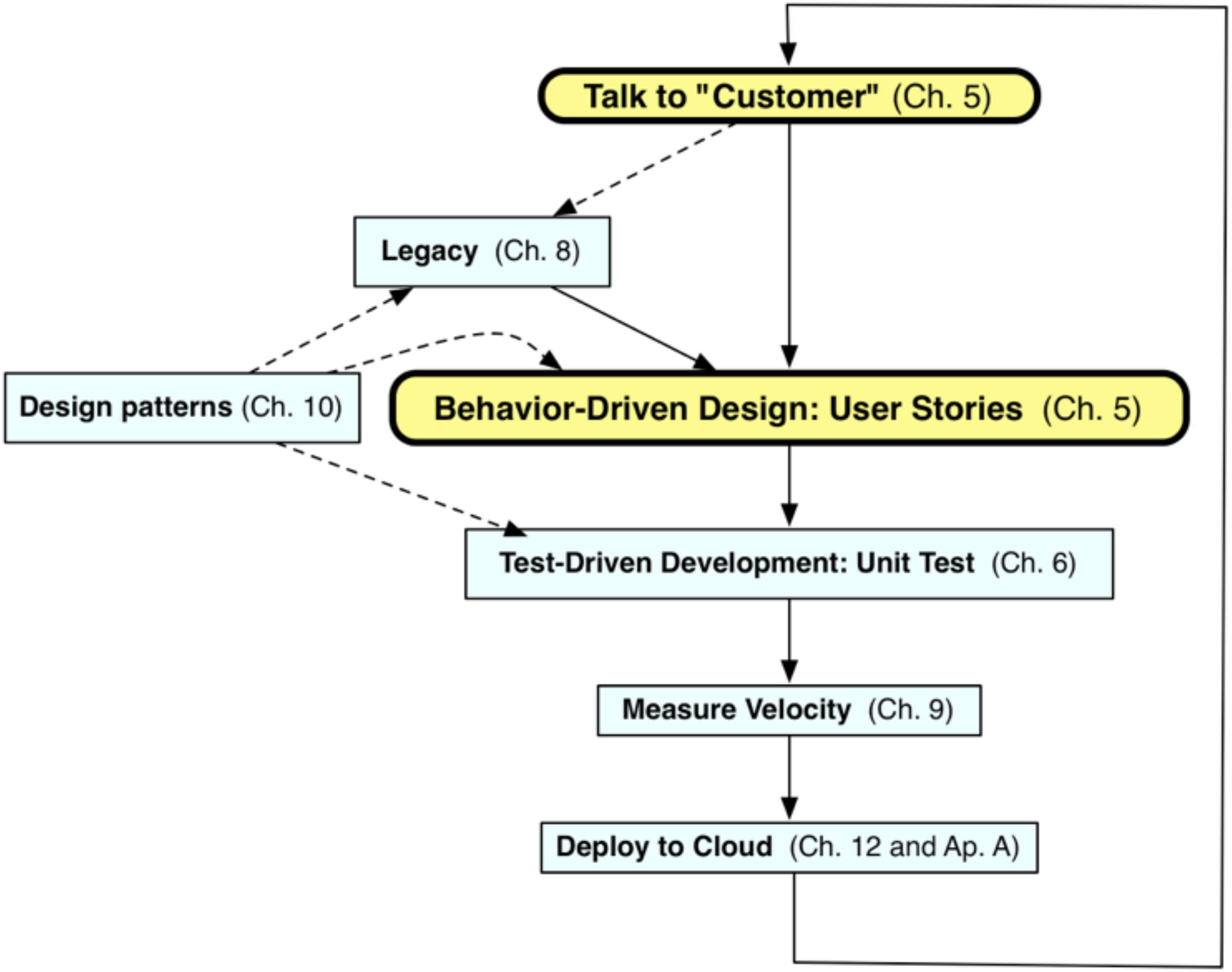
WHY DO SW PROJECTS FAIL?

- Don't do what customers want
- Or projects are late
- Or over budget
- Or hard to maintain and evolve
- Or all of the above
- Inspired Agile Lifecycle

AGILE LIFECYCLE

- Work closely, continuously with stakeholders to develop requirements, tests
  - Users, customers, developers, maintenance programmers, operators, project managers, …
- Maintain working prototype while deploying new features every **iteration**
  - Typically every 1 or 2 weeks
  - Instead of 5 major phases, each months long
- Check with stakeholders on what's next,
  to validate building right thing (vs. verify)

AGILE ITERATION

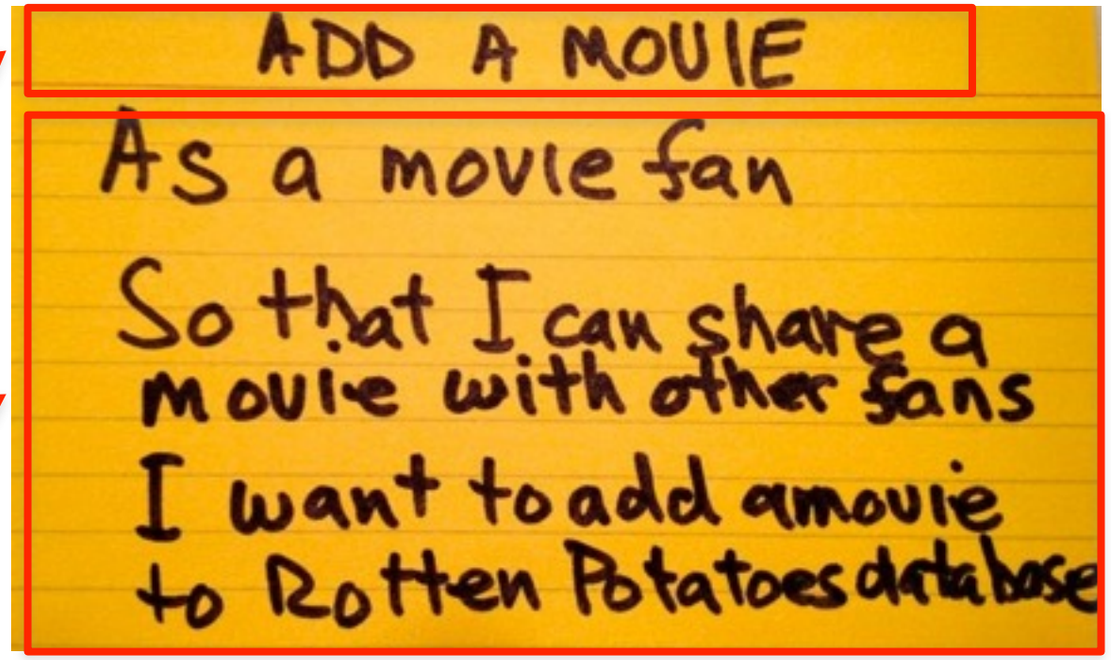


BEHAVIOR-DRIVEN DESIGN (BDD)

- BDD asks questions about behavior of app *before and during development* to reduce miscommunication
- Requirements written down as *user stories*
  - Lightweight descriptions of how app used
- BDD concentrates on *behavior* of app vs. *implementation* of app
  - Test Driven Design or TDD (next chapter) tests implementation

USER STORIES

- 1-3 sentences in everyday language
  - Fits on 3” x 5” index card
  - Written by/with customer
- “Connextra” format:
  - Feature name
  - As a [kind of stakeholder],
    So that [I can achieve some goal],
    I want to [do some task]
  - 3 phrases must be there, can be in any order



- Idea: user story can be formulated as *acceptance test before* code is written

WHY 3X5 CARDS?

- (from User Interface community)
- Nonthreatening => all stakeholders participate in brainstorming
- Easy to rearrange => all stakeholders participate in prioritization
- Since stories must be short, easy to change during development
  - As often get new insights during development

DIFFERENT STAKEHOLDERS MAY DESCRIBE BEHAVIOR DIFFERENTLY

- *See which of my friends are going to a show*
  - As a theatergoer
  - So that I can enjoy the show with my friends
  - I want to see which of my Facebook friends are attending a given show

---

- *Show patron's Facebook friends*
  - As a box office manager
  - So that I can induce a patron to buy a ticket
  - I want to show her which of her Facebook friends are going to a given show

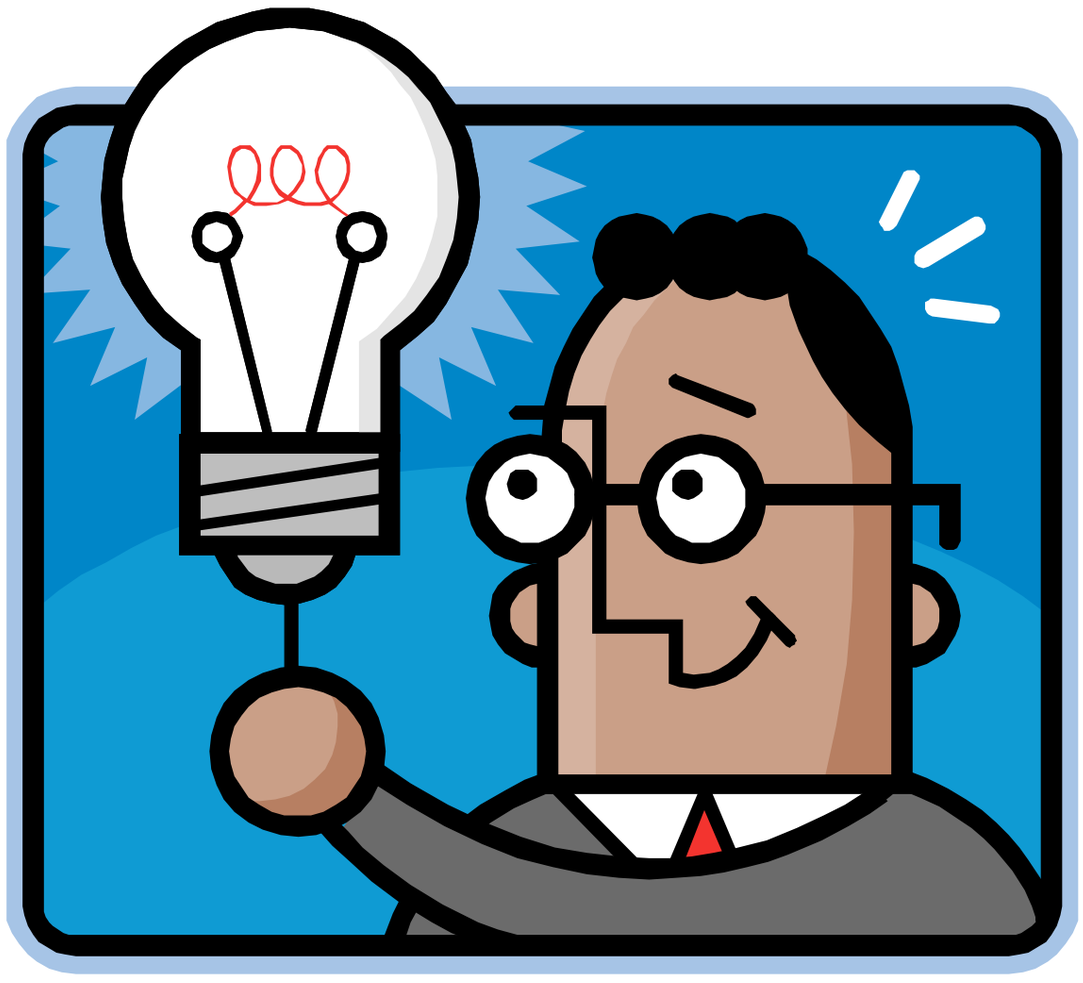
PRODUCT BACKLOG

- Real systems have 100s of user stories
- *Backlog*: User Stories not yet completed
  - (We'll see Backlog again with Pivotal Tracker)
- Prioritize so most valuable items highest
- Organize so they match SW releases over time

SMART USER STORIES

SMART STORIES

- **S**pecific
- **M**easurable
- **A**chievable
  (ideally, implement in
  1 iteration)
- **R**elevant
  (“the 5 why’s”)
- **T**imeboxed
  (know when to give up)



SPECIFIC & MEASURABLE

- Each scenario testable
  - Implies known good input and expected results exist
- Anti-example:
  "UI should be user-friendly"
- Example: Given/When/Then.
  - *Given* some specific starting condition(s),
  - *When* I do X,
  - *Then* one or more specific thing(s) should happen

ACHIEVABLE

- Complete in 1 iteration
- If can't deliver feature in 1iteration, deliver subset of stories
  - Always aim for working code @ end of iteration

TIMEBOXED

- Estimate what's achievable using *velocity*
  - Each story assigned *points* (1-3) based on difficulty
  - Velocity
    - = Points completed / iteration
  - Use measured velocity to plan future i
    points per story
- Pivotal Tracker (later) tracks velocity

RELEVANT: "BUSINESS VALUE"

- Ask "Why?" recursively until discover business value, or kill the story:
  - Protect revenue
  - Increase revenue
  - Manage cost
  - Increase brand value
  - Making the product remarkable
  - Providing more value to your customers

STORIES ARE SMART—
BUT FEATURES SHOULD BE RELEVANT

- Specific & Measurable: can I test it?
- Achievable? / Timeboxed?
- Relevant? use the “5 whys”
- *Show patron’s Facebook friends*

As a box office manager

So that I can induce a patron to
buy a ticket

I want to show her which Facebook
friends are going to a given show

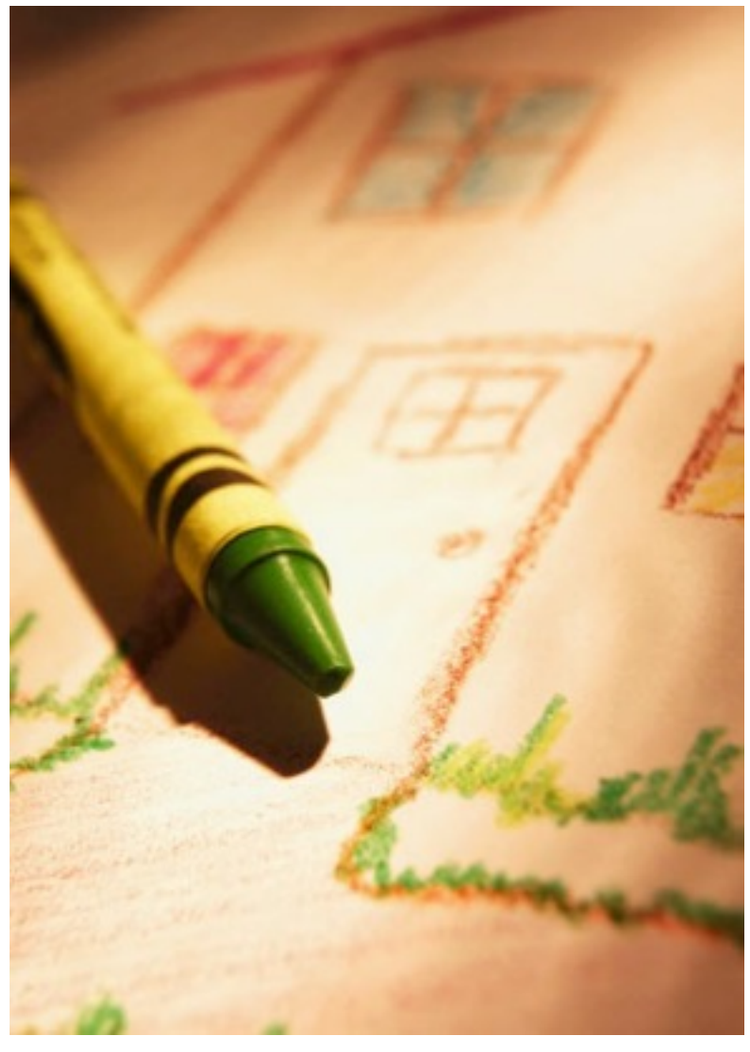
LO-FI UI SKETCHES AND STORYBOARDS

SAAS USER INTERFACE DESIGN

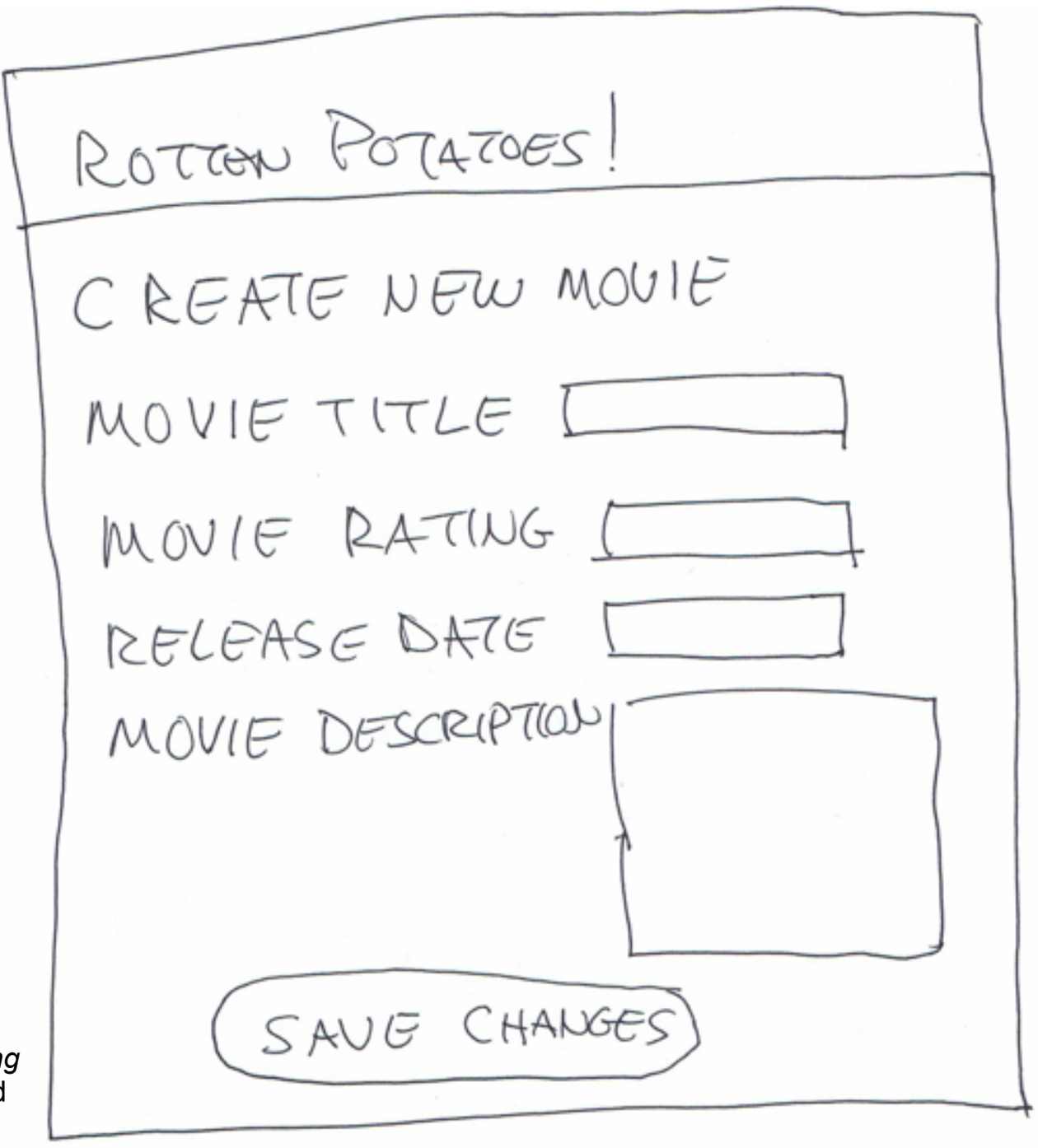
- SaaS apps often faces users

⇒User stories need User Interface (UI)

- Want *all* stakeholders involved in UI design
  - Don't want UI rejected!
- Need UI equivalent of 3x5 cards
- Sketches: pen and paper drawings or "Lo-Fi UI"



LO-FI UI EXAMPLE

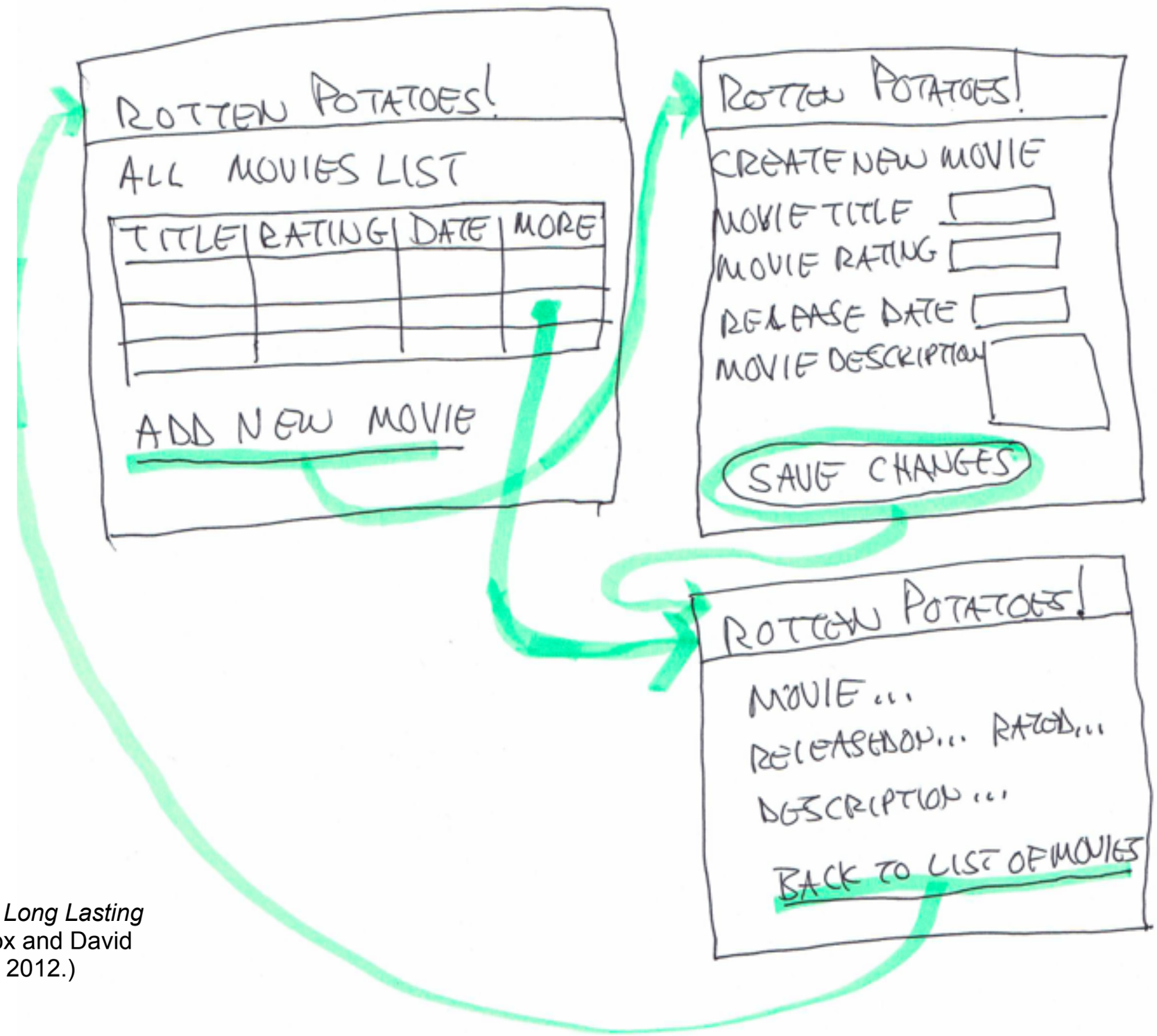


(Figure 4.3, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

STORYBOARDS

- Need to show how UI changes based on user actions
- HCI => “storyboards”
- Like scenes in a movie
- But not linear

EXAMPLE STORYBOARD



(Figure 4.4, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Alpha edition, 2012.)

LO-FI TO HTML

- Tedious to do sketches and storyboards,
  but easier than producing HTML!
  - Also less intimidating to nontechnical stakeholders =>
    More likely to suggest
    changes to UI if not code behind it
  - More likely to be happy with ultimate UI
- Next steps: More on CSS (Cascading Style Sheets) and Haml
  - Make it pretty *after* it works