ARQUITETURA SHARED-NOTHING EM 3 CAMADAS

ENGENHARIA DE SISTEMAS DE INFORMAÇÃO

Daniel Cordeiro

29 de agosto de 2017

Escola de Artes, Ciências e Humanidades | EACH | USP

INFO SOBRE O PROJETO

PROJETO: ITERAÇÕES 1–X

O projeto acontecerá em iterações de 2 semanas. No final da iteração, **cada membro** deve entregar uma avaliação das contribuições de cada um dos membros da equipe. As respostas serão confidenciais e usadas apenas na atribuição de notas individuais dos membros da equipe. Por isso, seja honesto e justo! A entrega, individual, deve conter:

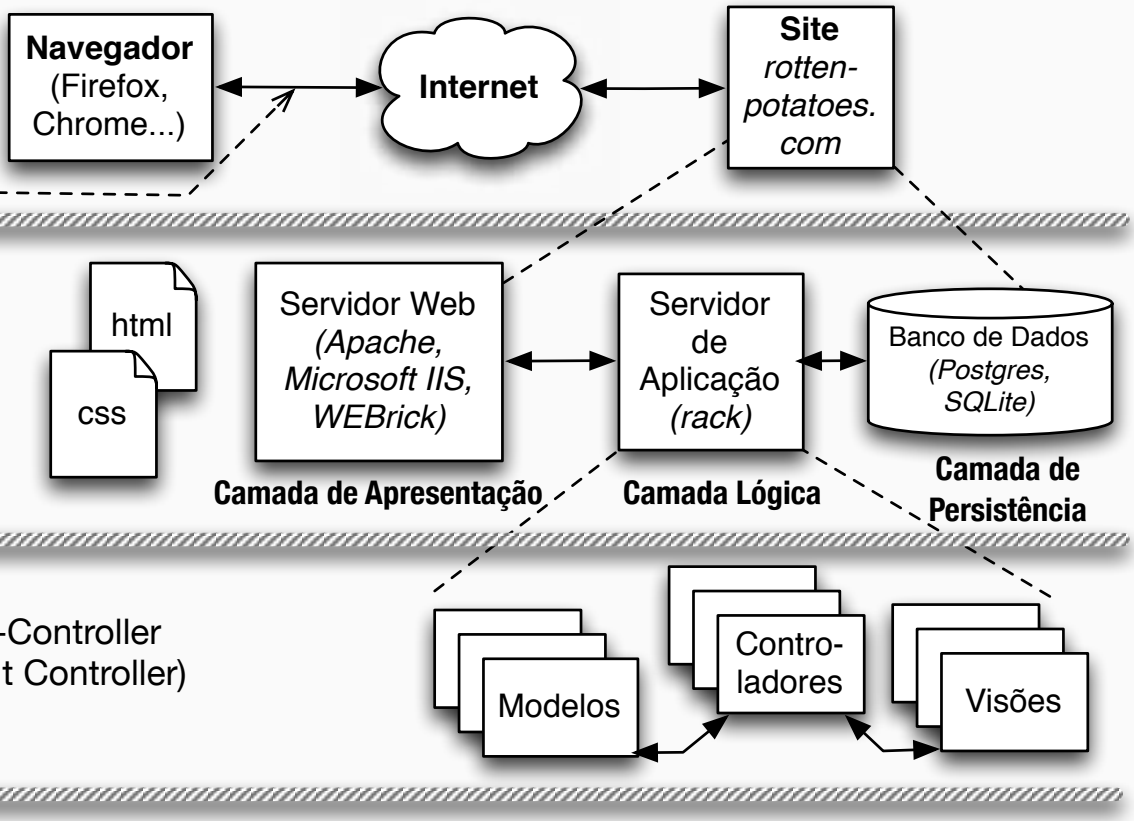
1. link para o projeto no GitHub
2. uma frase que resuma o que você fez nessa iteração
3. para cada outro membro do grupo a resposta da pergunta: "De modo geral, o membro excedeu as expectativas, atendeu as expectativas, fez só o mínimo necessário ou ficou aquém das expectativas do grupo durante a última iteração?"
    - considere todos os fatores que podem contribuir para o desenvolvimento do projeto: essa pessoa se comunicou com o resto da equipe de forma eficiente? Ela tentou fazer a sua parte no trabalho? Ela estava tecnicamente preparada para realizar o trabalho (ela tinha conhecimento do que foi visto em aula e das ferramentas necessárias)?
    - justifique cada avaliação

§2.1 100.000 pés
• Cliente-servidor (vs. P2P)

§2.2 50.000 pés
• HTTP e URIs

§2.3 10.000 pés
• XHTML e CSS

§2.4 5.000 pés
• Arquitetura de 3 camadas
• Escalabilidade horizontal



§2.5 1.000 pés—Model-View-Controller
(vs. Page Controller, Front Controller)

§2.6 500 pés: Modelos Active Record (vs. Data Mapper)
§2.7 500 pés: Controladores REST (*Representational State Transfer* para ações auto-contidas)
§2.8 500 pés: Template View (vs. Transform View)

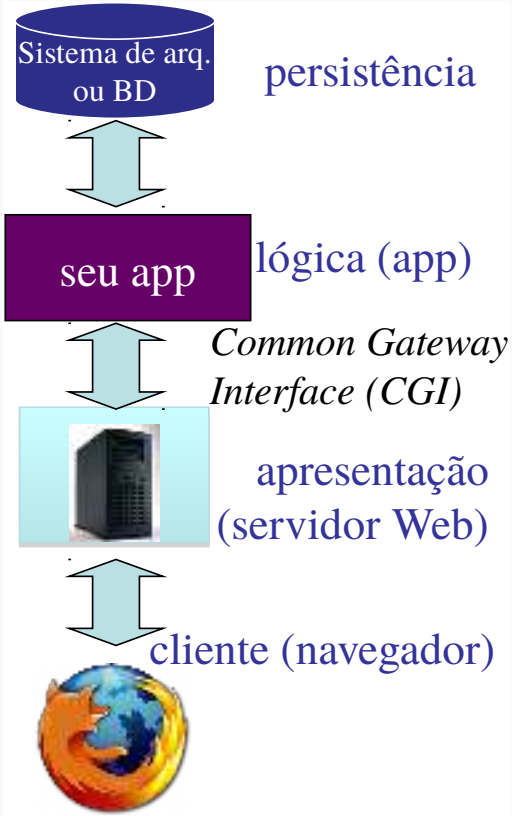
• **Active Record** • **REST** • **Template View**
• Data Mapper • Transform View

CRIAÇÃO DE CONTEÚDO DINÂMICO

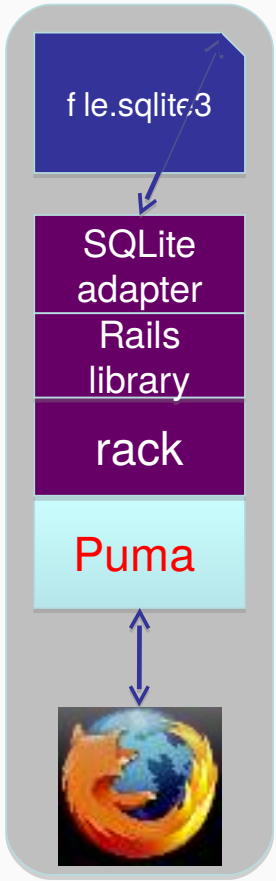
- Antigamente, a maior parte das páginas Web eram (coleções de) arquivos simples
- Mas os sites mais interessantes da Web 1.0/e-commerce executam um programa que cria cada “página”
- Originalmente: *templates* com código embutido (“*snippets*”)
- Eventualmente o código acabou movido para fora do servidor Web

SITES QUE NA VERDADE SÃO PROGRAMAS (SAAS)

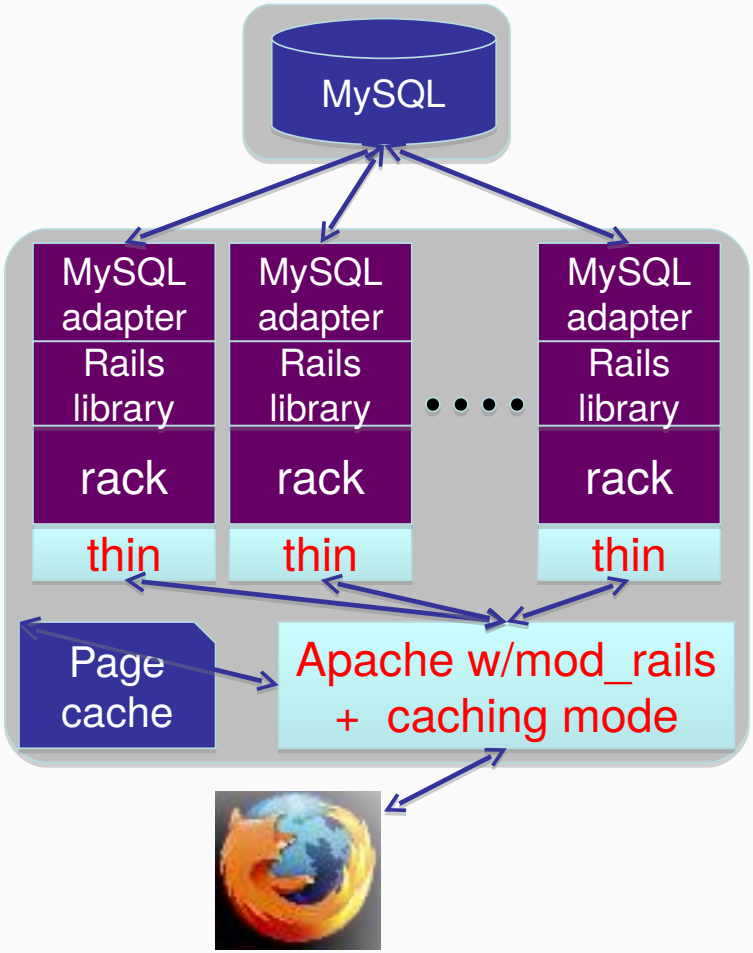
- Como você faz para:
  - "mapear" a URI para o programa & função corretos
  - passar argumentos?
  - invocar programas no servidor?
  - lidar com armazenamento persistente?
  - lidar com cookies?
  - lidar com erros?
  - empacotar a saída para o usuário?
- Usar arcabouços facilita essas tarefas mais comuns



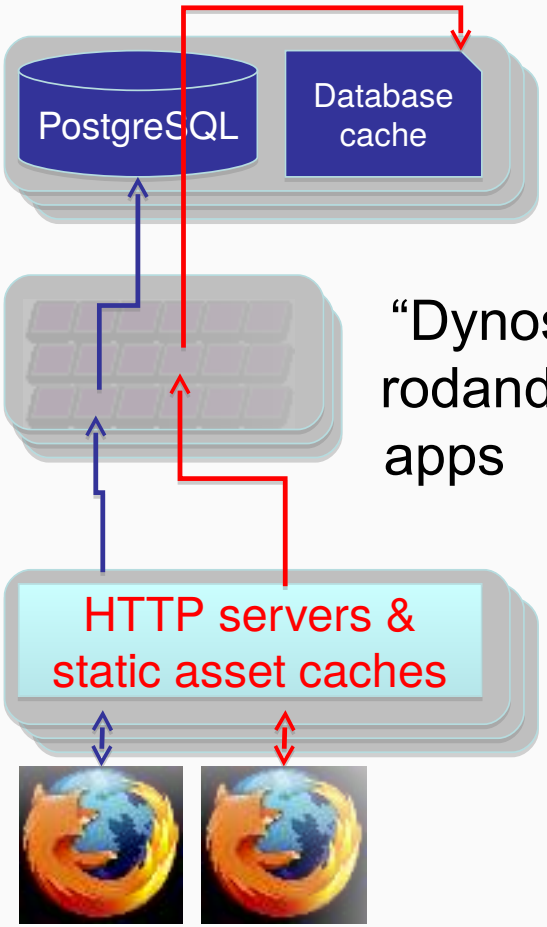
AMBIENTE DO DESENVOLVEDOR VS. IMPLANTAÇÃO DE MÉDIA ESCALA



Desenvolvedor

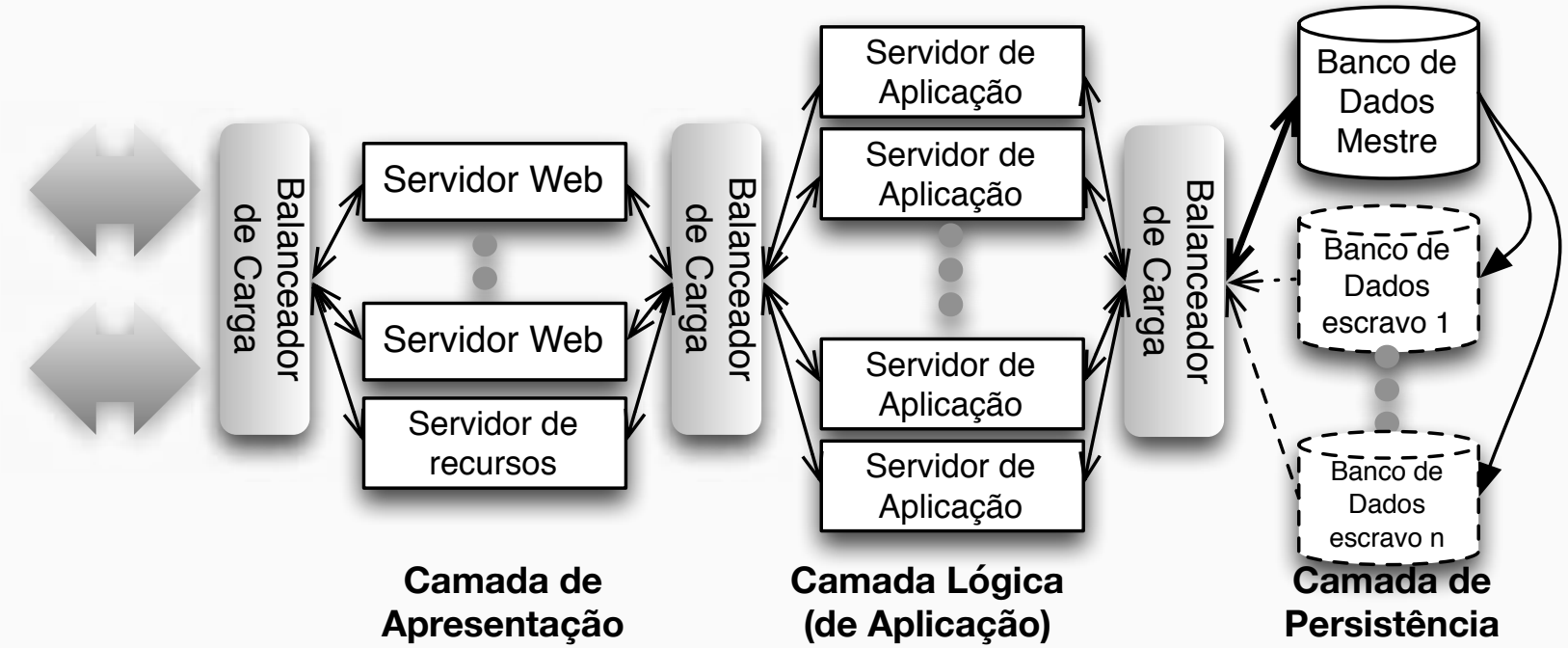


Implantação de média escala



Implantação cuidadosa em larga escala (ex: Heroku)

“SHARED NOTHING”



RESUMO: SAAS NA WEB 1.0 I

- Navegador requisita um recurso web (URI) usando HTTP
  - HTTP é um protocolo requisição–resposta simples que depende de TCP/IP
  - em SaaS, a maior parte das URIs disparam a execução de um programa
- HTML é usado para codificar o conteúdo, CSS para estilizá-lo
- Cookies permitem que o servidor acompanhe o rastro do usuário
  - o navegador automaticamente passa os cookies para o servidor em cada requisição
  - o servidor pode mudar o cookie em cada requisição
  - uso típico: cookie inclui uma forma de acessar a informação do lado do servidor
  - por isso muitos sites não funcionam quando os cookies estão totalmente desabilitados

RESUMO: SAAS NA WEB 1.0 II

- Arcabouços fazem com que essas abstrações sejam mais convenientes para o programador usar, sem que ele precise entrar em todos os detalhes
- ... e permitem mapear um app SaaS na arquitetura em 3 camadas “shared-nothing”

PERGUNTA

Quais afirmações são corretas sobre as duas requisições abaixo:

```
GET /foo/bar
POST /foo/bar
```

1. Eles são indistinguíveis para a app SaaS
2. Eles são distinguíveis e *devem* ter comportamentos diferentes
3. Eles são distinguíveis e *podem* ter comportamentos diferentes
4. Um dado app pode ser configurado para lidar com um ou com outro, mas não ambos

MODEL–VIEW–CONTROLLER

ATÉ QUE PONTO...

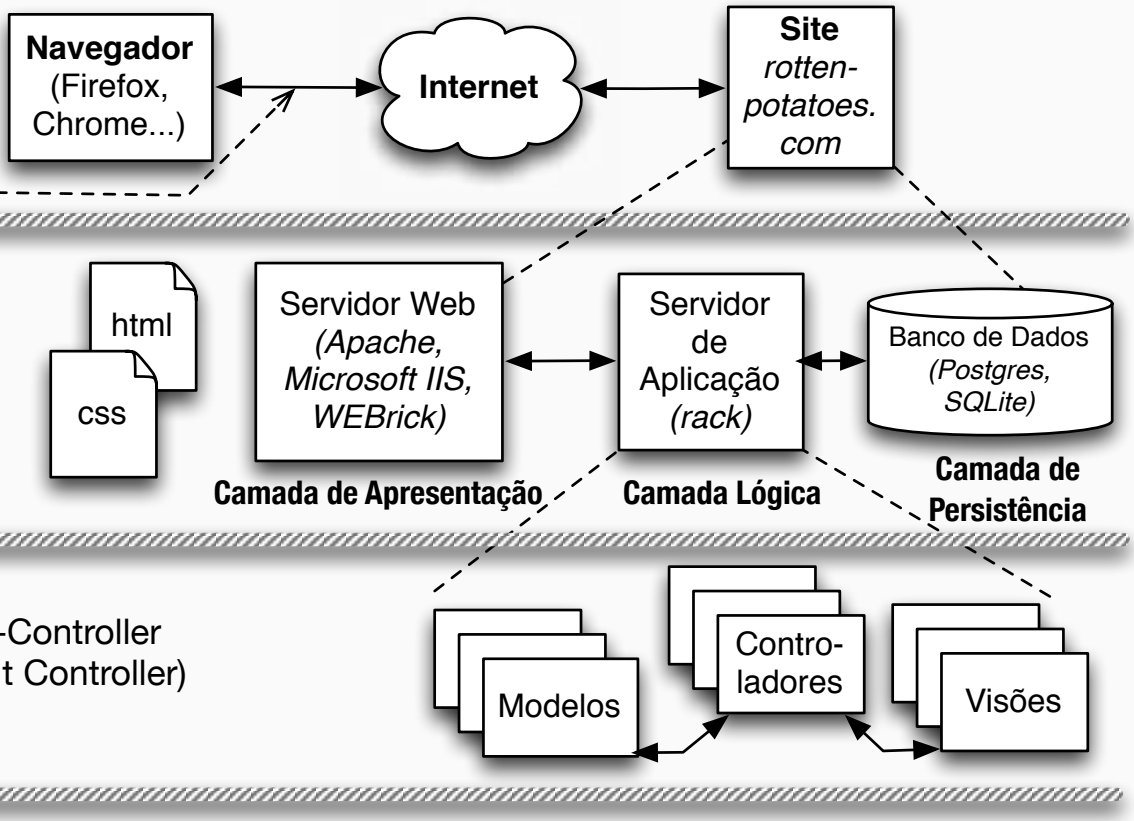
- há alguma estrutura comum em aplicações...
- ... interativas ...
- ... que pode simplificar o desenvolvimento de apps se nós a capturarmos em um arcabouço?

§2.1 100.000 pés
• Cliente-servidor (vs. P2P)

§2.2 50.000 pés
• HTTP e URIs

§2.3 10.000 pés
• XHTML e CSS

§2.4 5.000 pés
• Arquitetura de 3 camadas
• Escalabilidade horizontal



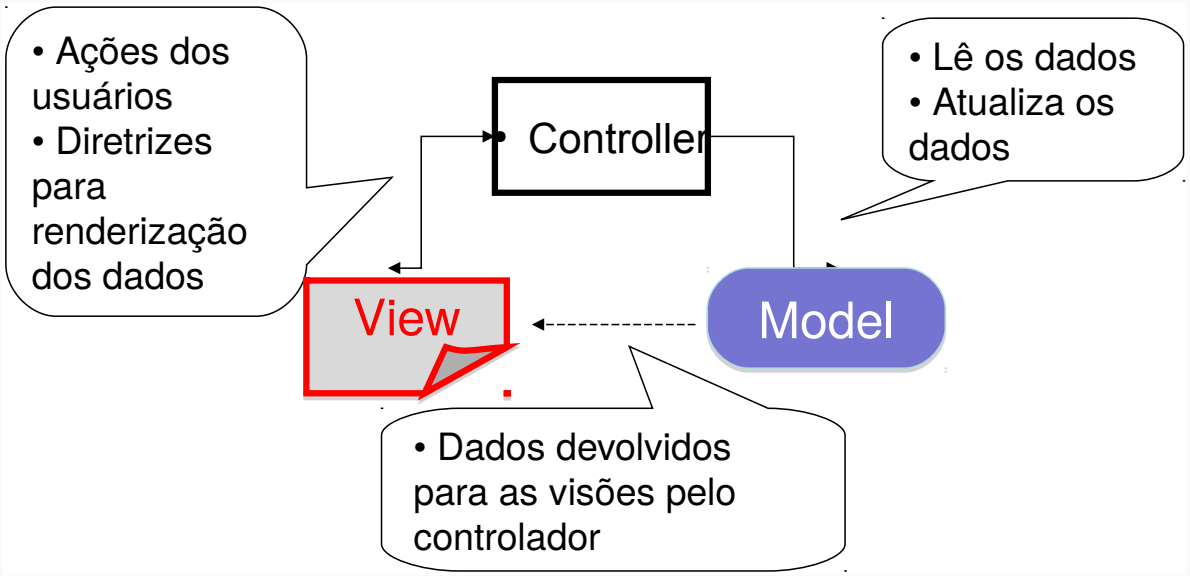
§2.5 1.000 pés—Model-View-Controller
(vs. Page Controller, Front Controller)

§2.6 500 pés: Modelos Active Record (vs. Data Mapper)
§2.7 500 pés: Controladores REST (*Representational State Transfer* para ações auto-contidas)
§2.8 500 pés: Template View (vs. Transform View)

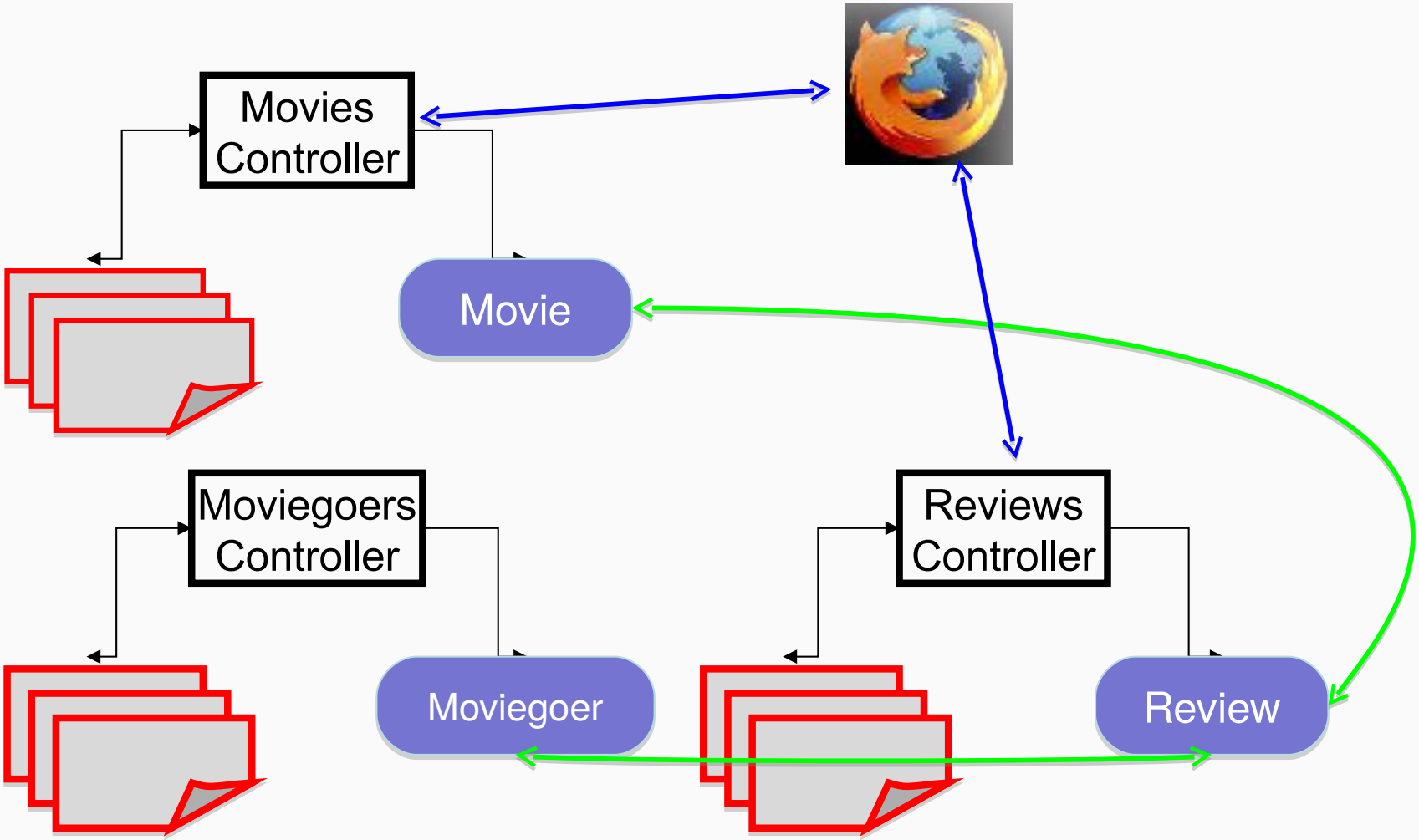
• **Active Record** • **REST** • **Template View**
• Data Mapper • Transform View

O PADRÃO DE PROJETO MVC

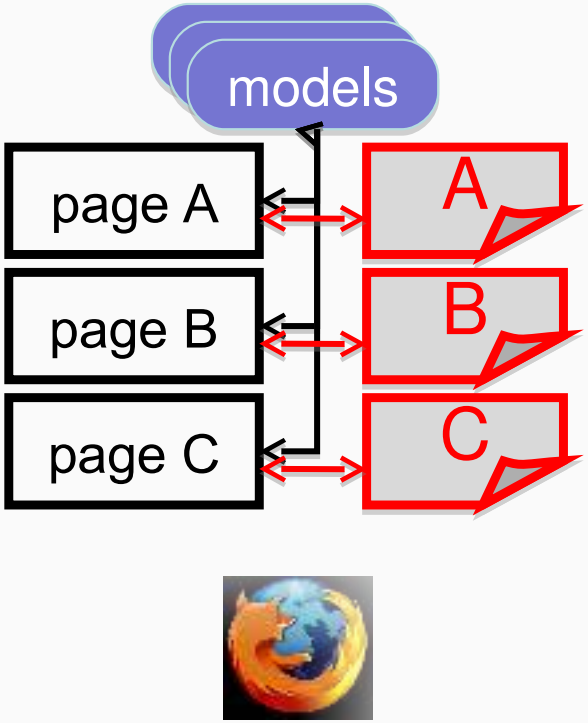
- Objetivo: separar os dados (*modelo*) da UI & apresentação (*visão*) com o uso de um *controlador*
  - intercede as ações dos usuários que pedem acesso aos dados
  - expõe os dados para a renderização (a ser realizada pela visão)
- Apps Web podem parecer "obviamente" MVC por definição, mas outras alternativas são possíveis...



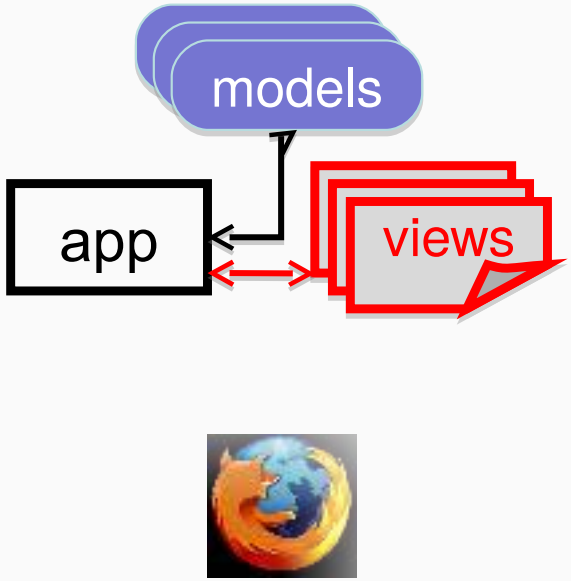
CADA ENTIDADE TEM UM MODELO, CONTROLE & CONJUNTO DE VISÕES



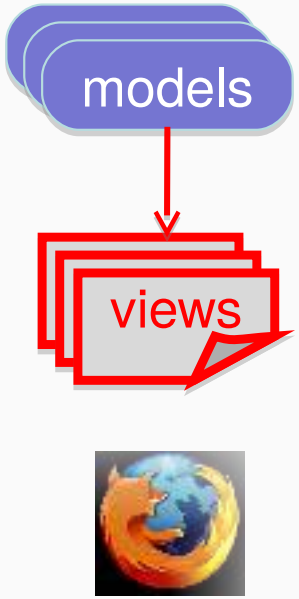
Page Controller
(Ruby Sinatra)



Front Controller
(J2EE servlet)



Template View
(PHP)



Rails

Usa por padrão apps estruturados como MVC, mas outras arquiteturas podem ser melhores para certos apps.

PERGUNTA

Qual afirmação não é verdade sobre o padrão arquitetural Model–View–Controller:

- Em apps SaaS na Web, o conteúdo relacionado às ações do controlador e visão são transmitidos via HTTP
- Todos os apps MVC possuem uma parte “cliente” (ex: navegador) e uma parte “na nuvem” (ex: app Rails em uma plataforma de computação em nuvem)
- Model–View–Controller é só um dentre vários modos possíveis de estruturar uma app SaaS
- Apps Peer-to-Peer (o contrário de apps cliente–servidor) podem ser estruturados com Model–View–Controller