

Engenharia de Software

Prof. Vinicius Cardoso Garcia

vcg@cin.ufpe.br :: [@vinicius3w](https://twitter.com/vinicius3w) :: viniciusgarcia.me

[IF977] Engenharia de Software

<http://bit.ly/vcg-es>

Licença do material

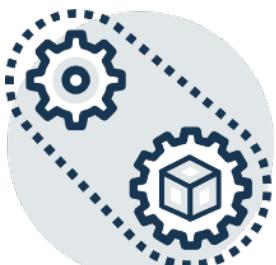
Este Trabalho foi licenciado com uma Licença

Creative Commons - Atribuição-NãoComercial-
Compartilhual 3.0 Não Adaptada



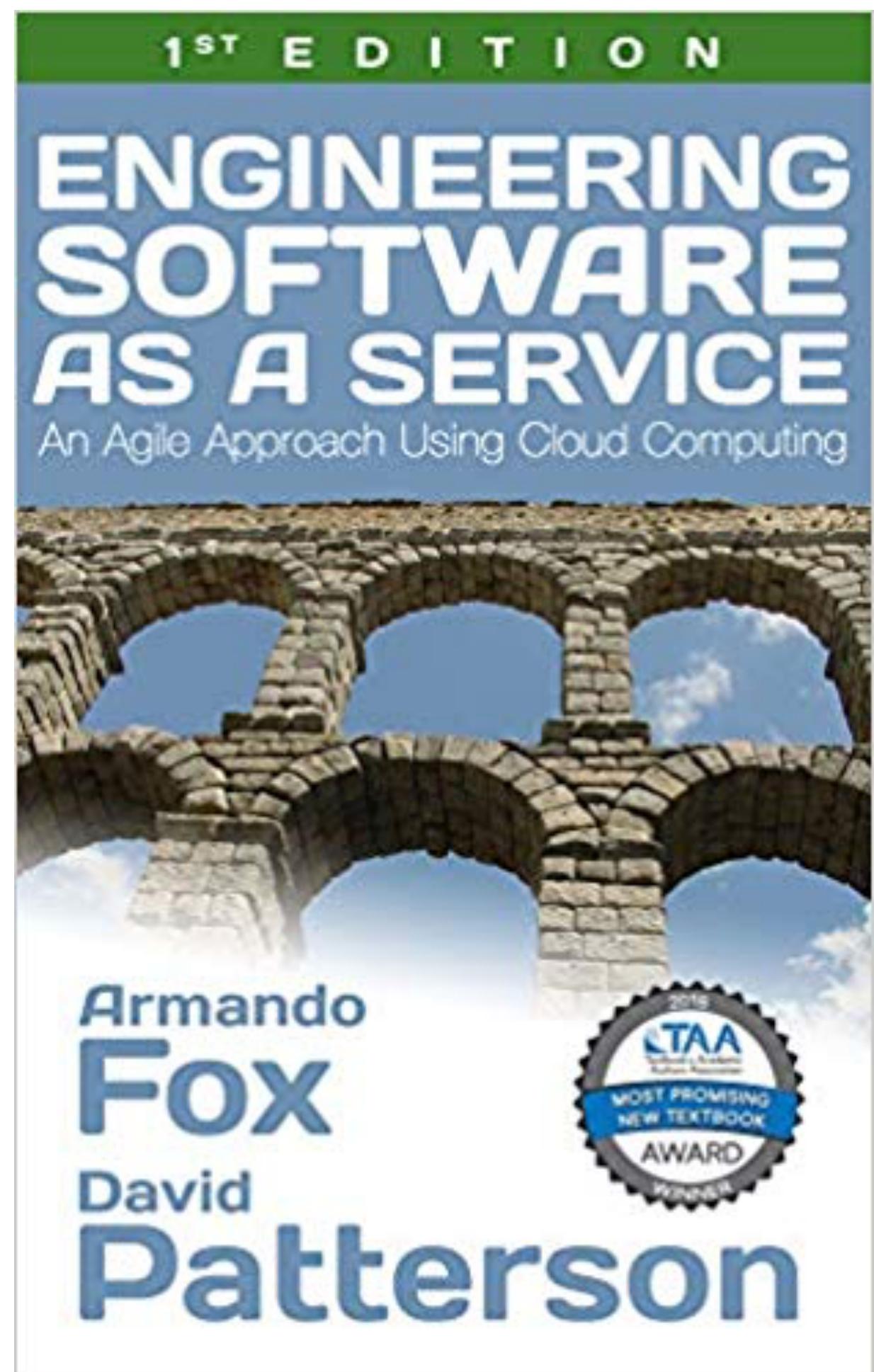
Mais informações visite

[http://creativecommons.org/licenses/by-nc-sa/
3.0/deed.pt](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt)



Referências

- A biblioteca do Desenvolvedor de Software dos dias de hoje
 - <http://bit.ly/31WYK5f>
- SWEBOK: Guide to the Software Engineering Body of Knowledge (SWEBOK)
 - <http://www.computer.org/web/swebok>
- Engineering Software as a Service: An Agile Approach Using Cloud Computing
 - <http://www.saasbook.info/>
- Marco Tulio Valente. Engenharia de Software Moderna
 - <https://engsoftmoderna.info/>



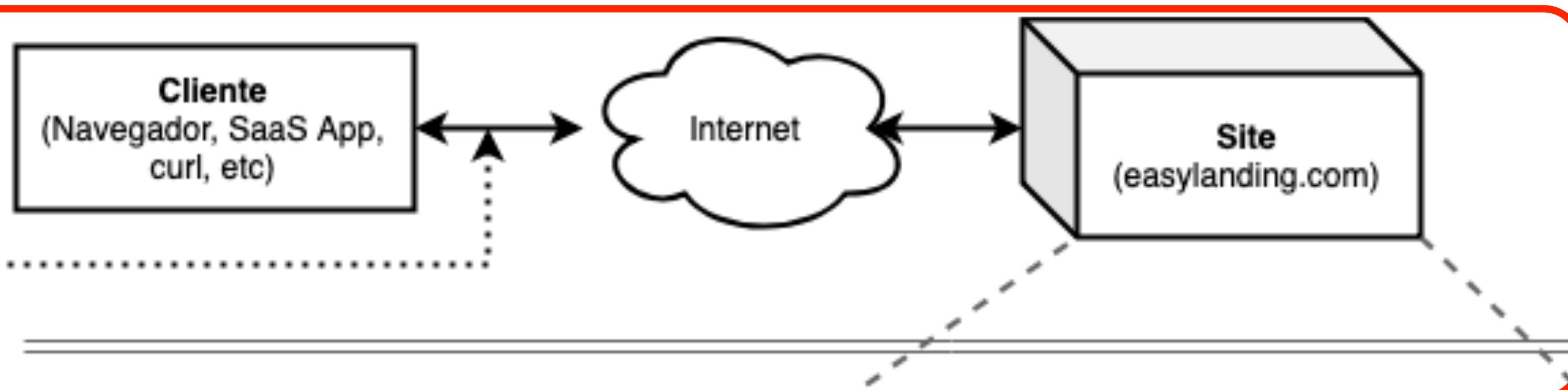


A Web como um sistema cliente-servidor

Introdução ao TCP/IP, HTTP e Rotas

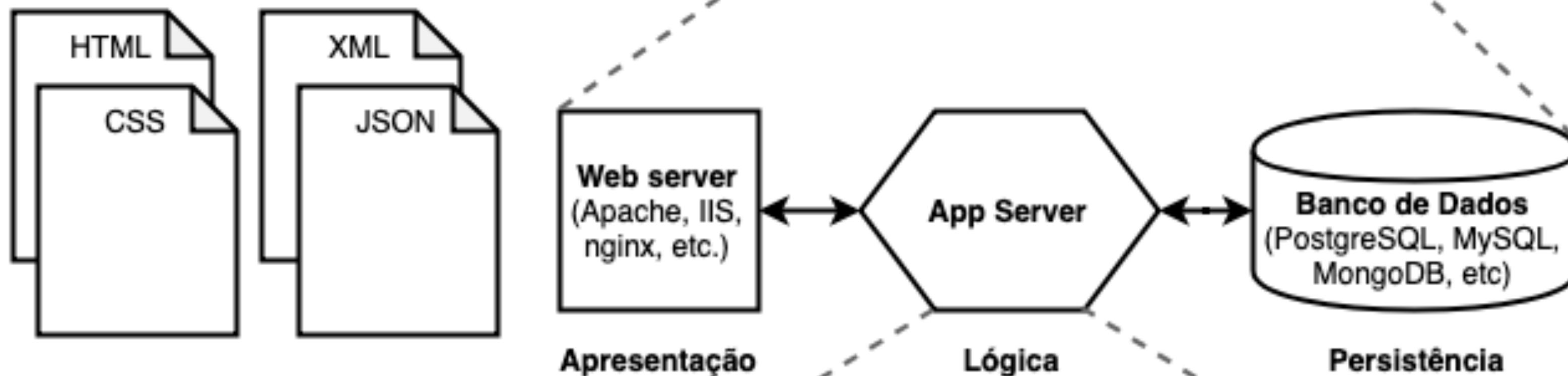


Cliente-Servidor vs P2P



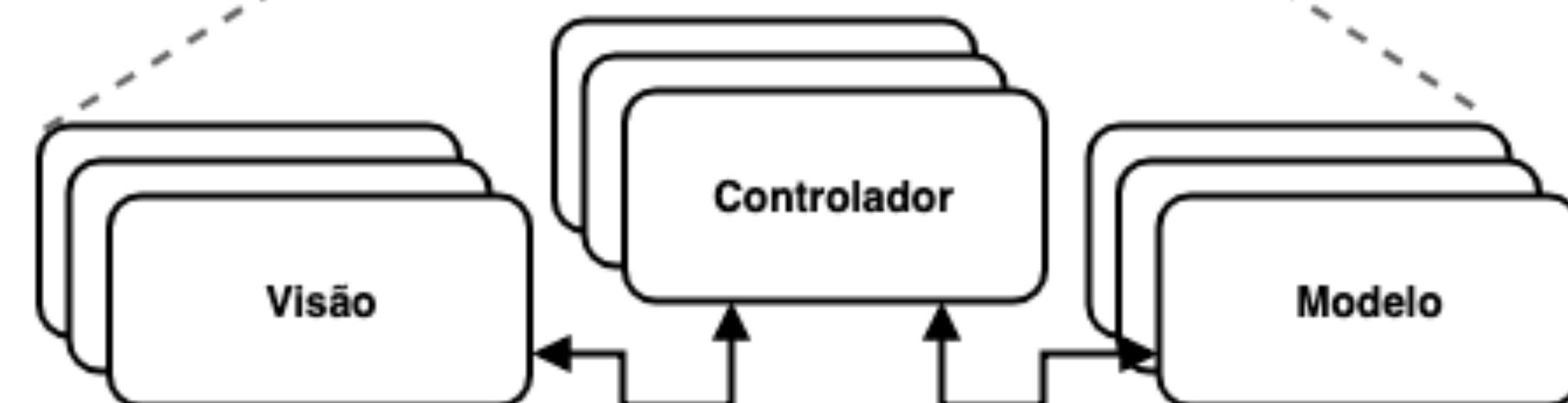
Arquitetura 3 camadas

Escalabilidade



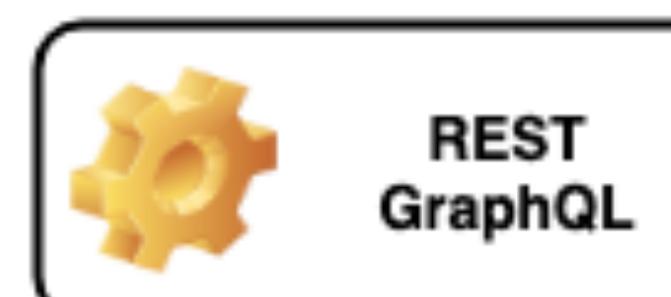
Padrões Arquiteturais

Model-View- Controller



Padrões de Projeto

Idiomas



REST
GraphQL



Template view
Scaffolding

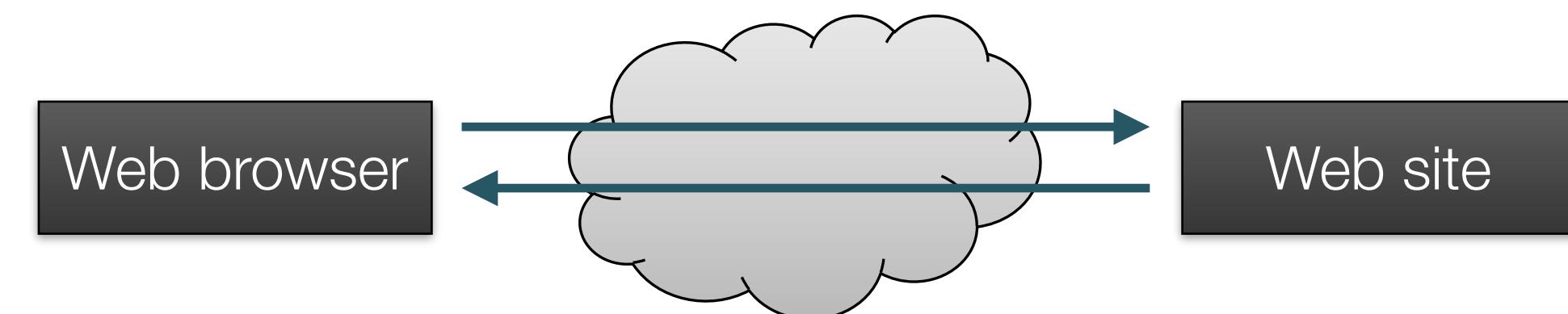


ORM
Active Record
Data Mapper

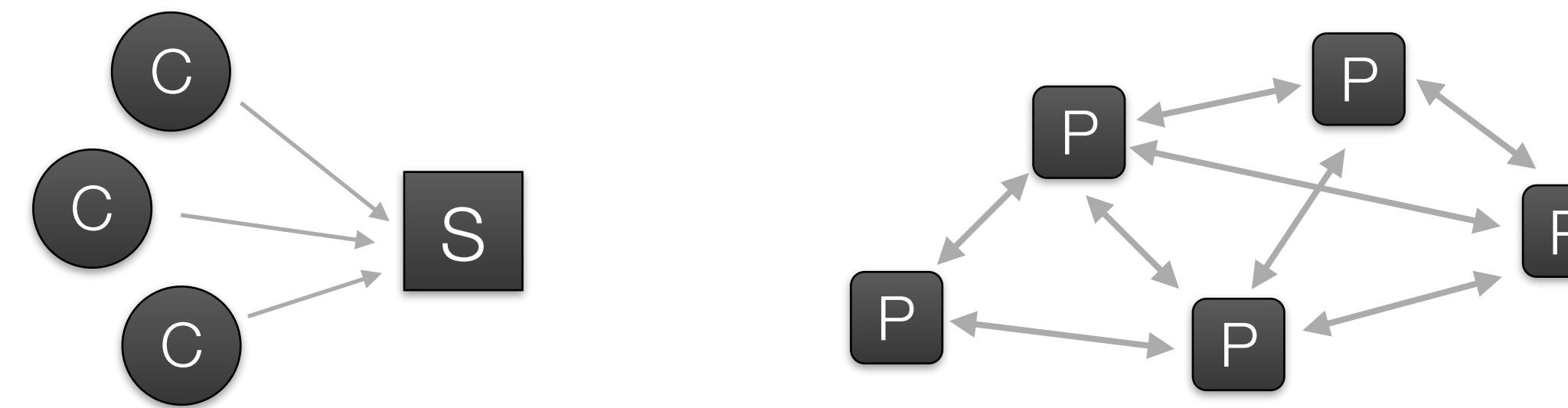


A Web

- A Web é uma arquitetura cliente/servidor
- É fundamentalmente uma estrutura orientada a requisição e resposta



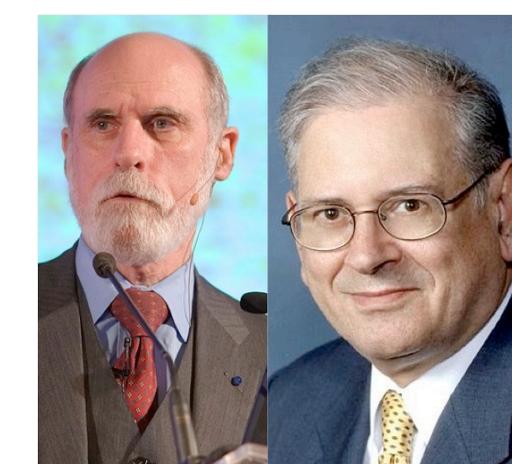
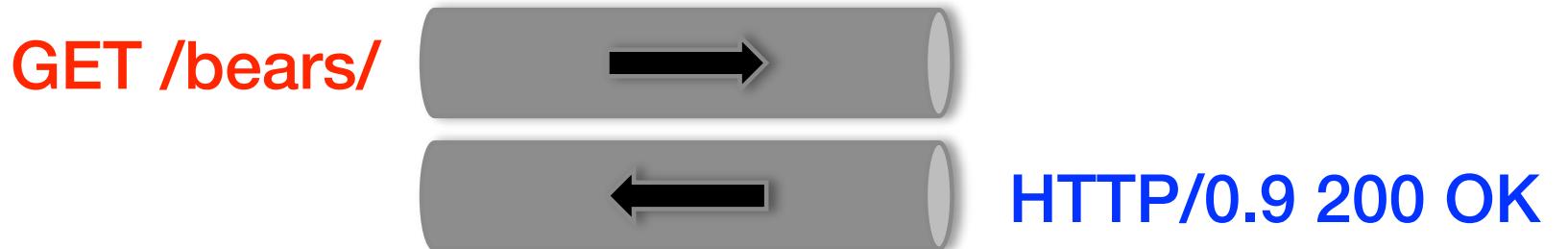
Cliente-Servidor vs. Peer-to-Peer



- Cliente e Servidor são especializados em suas tarefas
 - Cliente: fazer perguntas em nome do usuário
 - Servidor: esperar e responder às perguntas, serve a muitos clientes
- Padrões de projeto (arquitetural, estrutural, computacional, etc) capturam soluções estruturais comuns para problemas recorrentes!
- Cliente/Servidor é um padrão arquitetural!

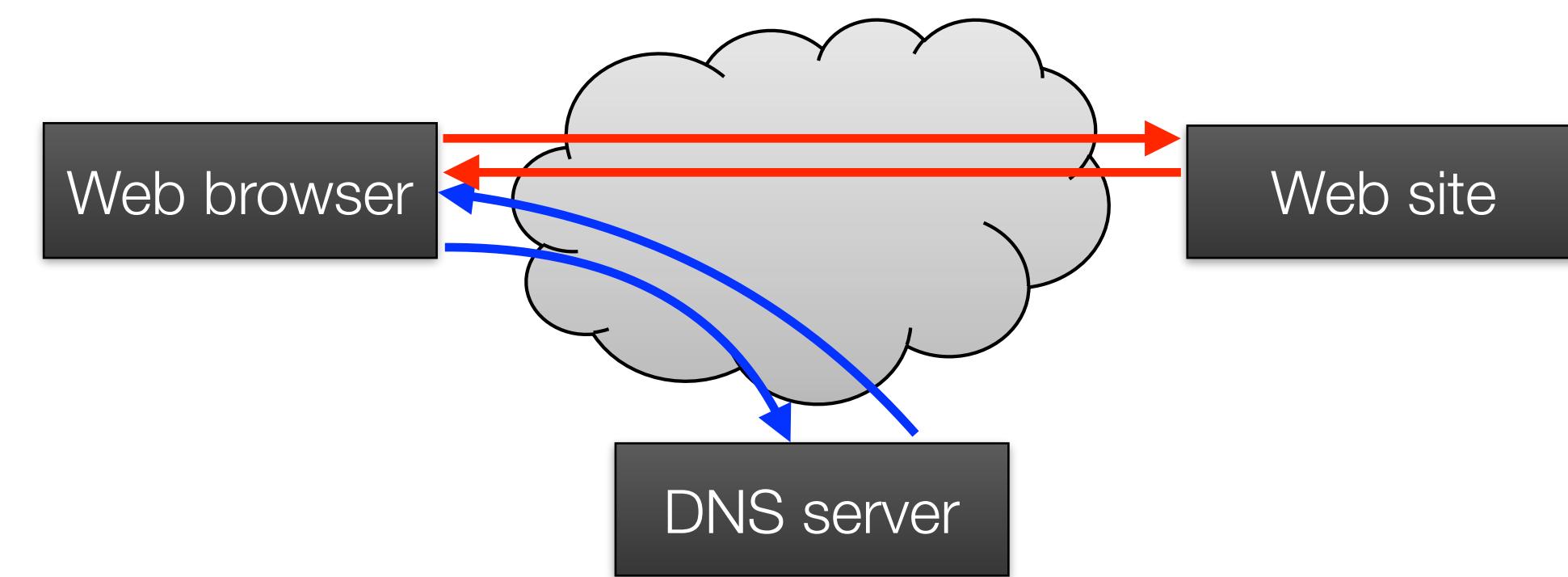
Protocolos TCP/IP

- Um endereço IP (Internet Protocol) identifica uma interface física de rede com quatro octeto, ex. **128.32.244.172**
 - O endereço **127.0.0.1** é “este computador”, chamado de **localhost**, se não estiver conectado na internet!
- TCP/IP (Transmission Control Protocol/Internet Protocol) fornece abstração de um fluxo ordenado e confiável entre endereços IP
 - TCP: torna o IP confiável através da detecção de pacotes “perdidos”, dados que chegam fora de ordem, erros de transmissão, redes lentas, etc, e respondem de forma apropriada
 - Portas TCP permitem múltiplas apps no mesmo computador
- Vint Cerf & Bob Kahn: 2004 Turing Award for Internet architecture & protocols, incl. TCP/IP



DNS

- A Web é uma arquitetura cliente/servidor
- É fundamentalmente uma estrutura orientada a requisição e resposta
- Domain Name System (DNS) é outro serviço que mapeia nomes para endereços IP



Hypertext Transfer Protocol

- Protocolo requisição/resposta baseado em ASCII para transmissão de informação na Web
- Requisição HTTP inclui
 - Método de requisição (GET, POST, etc)
 - Uniform Resource Identifier (URI)
 - Versão do protocolo HTTP entendido pelo cliente
 - Headers – informações extra sobre a requisição
- Resposta HTTP inclui
 - Protocol version & Status Code => => => => => =>
 - Cabeçalho da resposta
 - Corpo da resposta
- HTTP é sem estado! portanto, cookies.

HTTP status codes:

2xx — *all is well*

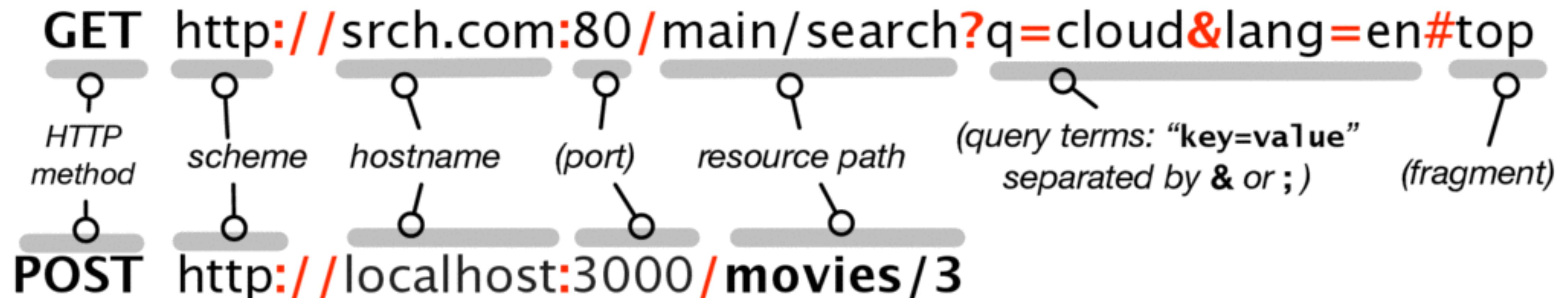
3xx — *resource moved*

4xx — *access problem*

5xx — *server error*



HyperText Transfer Protocol roda sobre TCP/IP



- HTTP method or verb:
 - GET, POST, PUT, PATCH, DELETE, HEAD
 - Idea: GETs should be **side-effect-free**
- Parameters considered separate from path
 - can also be carried in request body if a POST
- **Route = HTTP method + URI**



Cookies

- Problemas da Web 1.0: como guiar o usuário “através” de um fluxo de páginas?
 - Usar endereço IP para identificar retorno de usuários?
 - Computadores públicos, usuário compartilham IP
 - Embarcar identificação de usuários nas URI de busca?
 - Quebra o uso de cache
 - Rapidamente substituído por cookies
 - Frameworks gerenciam as violações evidentes dos cookies para você



Sumário

- O TCP/IP fornece uma abstração de fluxo confiável entre 2 endpoints IP
- HTTP é um dos muitos protocolos de transporte cliente-servidor, requisição-resposta, que usa TCP/IP
- Uma solicitação HTTP (**cliente**→**servidor**) solicita um **recurso** especificando uma **rota** e possivelmente **cabeçalhos**
 - route = método HTTP + URI
- Uma resposta HTTP (**servidor**→**cliente**) fornece **status**, **cabeçalhos** e **corpo** (recurso)



Pergunta

Which statement is true about the two HTTP requests:

GET /foo/bar

POST /foo/bar

- A. Eles são indistinguíveis para um aplicativo SaaS
- B. Eles são distinguíveis e devem ter comportamentos diferentes
- C. Eles são distinguíveis e podem ter comportamentos diferentes
- D. Um determinado aplicativo pode manipular um ou outro, mas não ambos

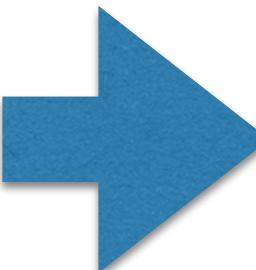


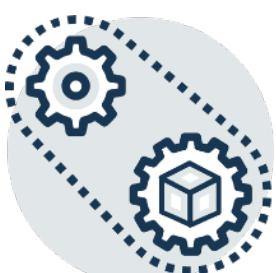
Pergunta

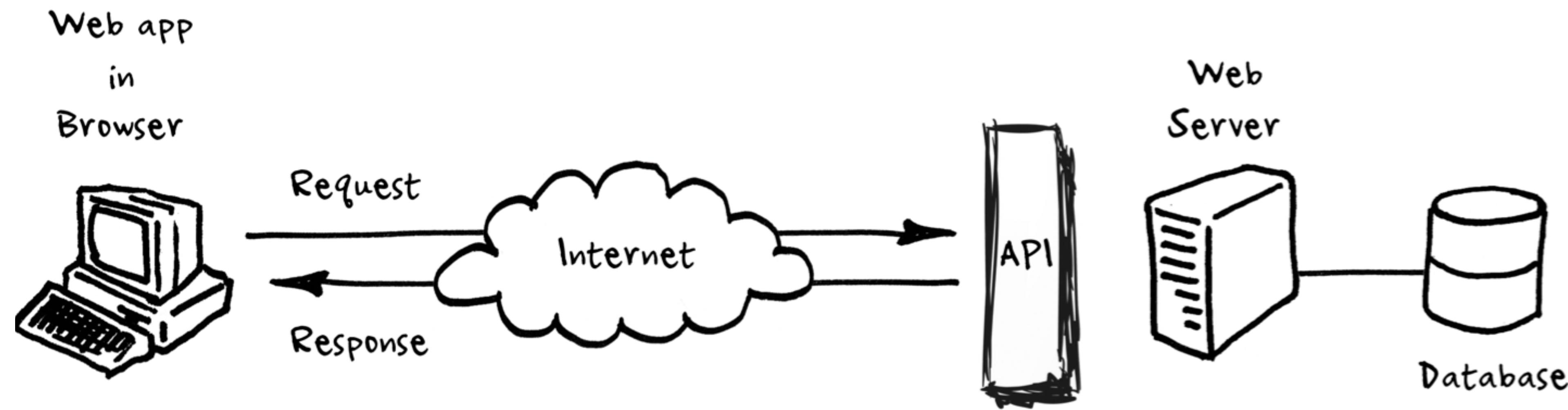
Which statement is true about the two HTTP requests:

GET /foo/bar

POST /foo/bar

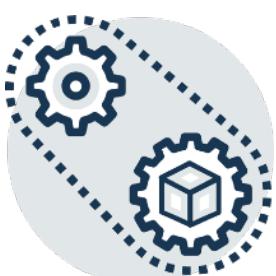
- A. Eles são indistinguíveis para um aplicativo SaaS
-  B. Eles são distinguíveis e devem ter comportamentos diferentes
- C. Eles são distinguíveis e podem ter comportamentos diferentes
- D. Um determinado aplicativo pode manipular um ou outro, mas não ambos



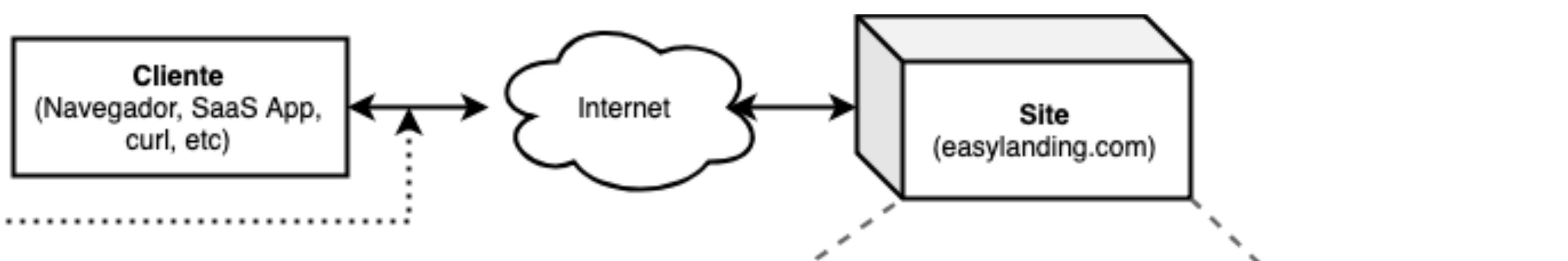


Service-Oriented Architecture

or
From Web sites to Microservices



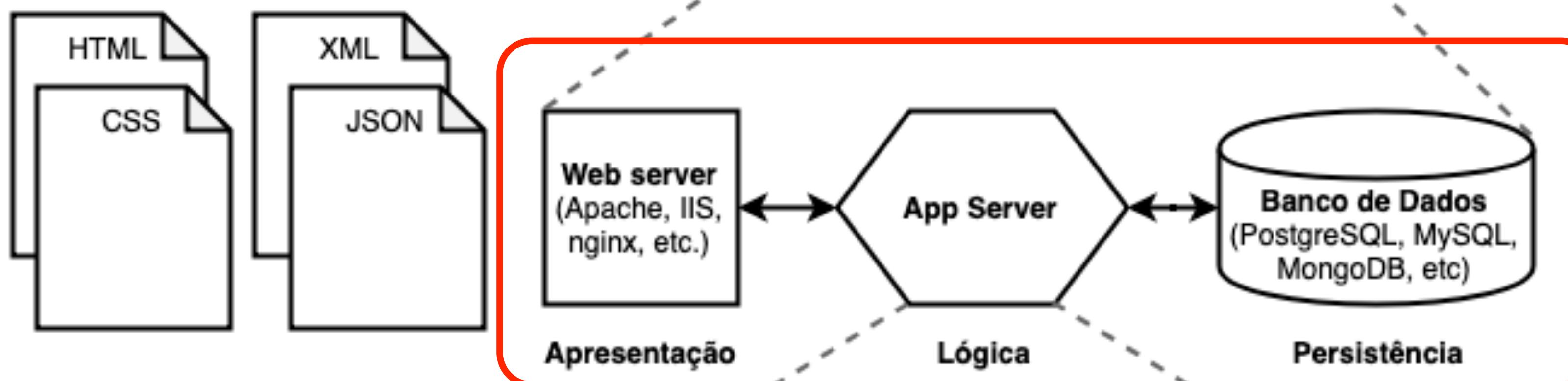
Cliente-Servidor vs P2P



HTTP & URIs

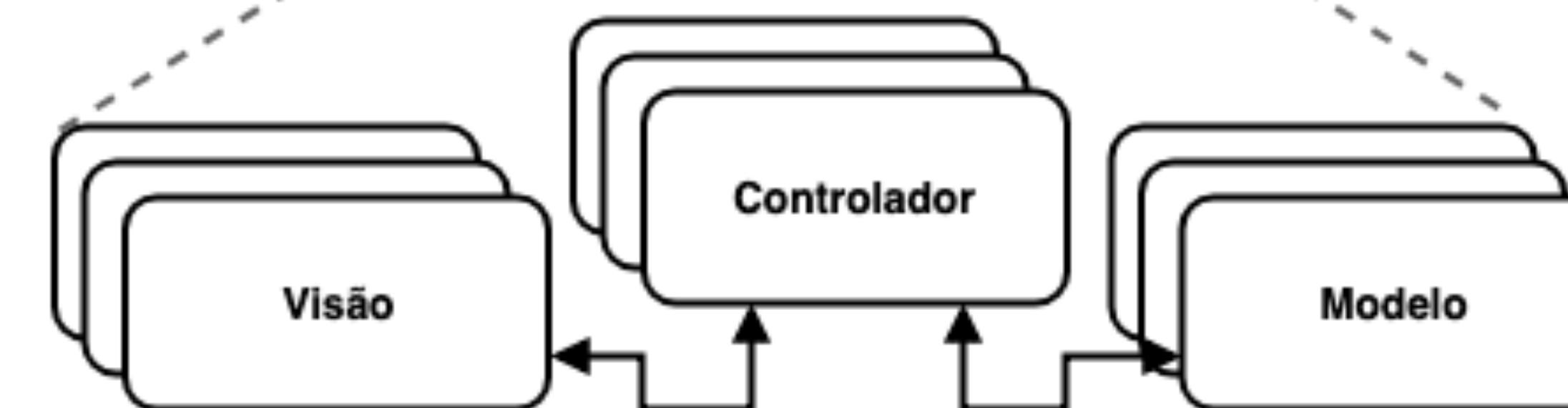
Arquitetura 3 camadas

Escalabilidade



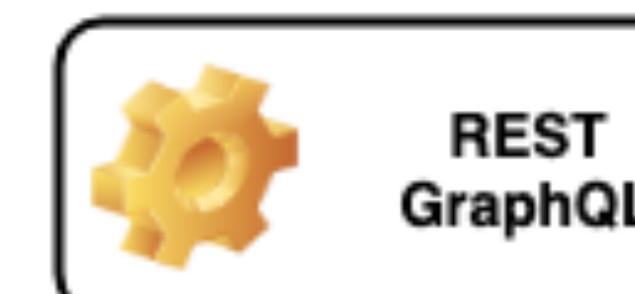
Padrões Arquiteturais

Model-View- Controller

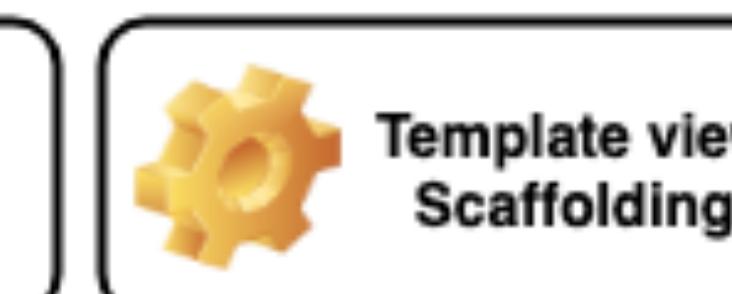


Padrões de Projeto

Idiomas



REST
GraphQL



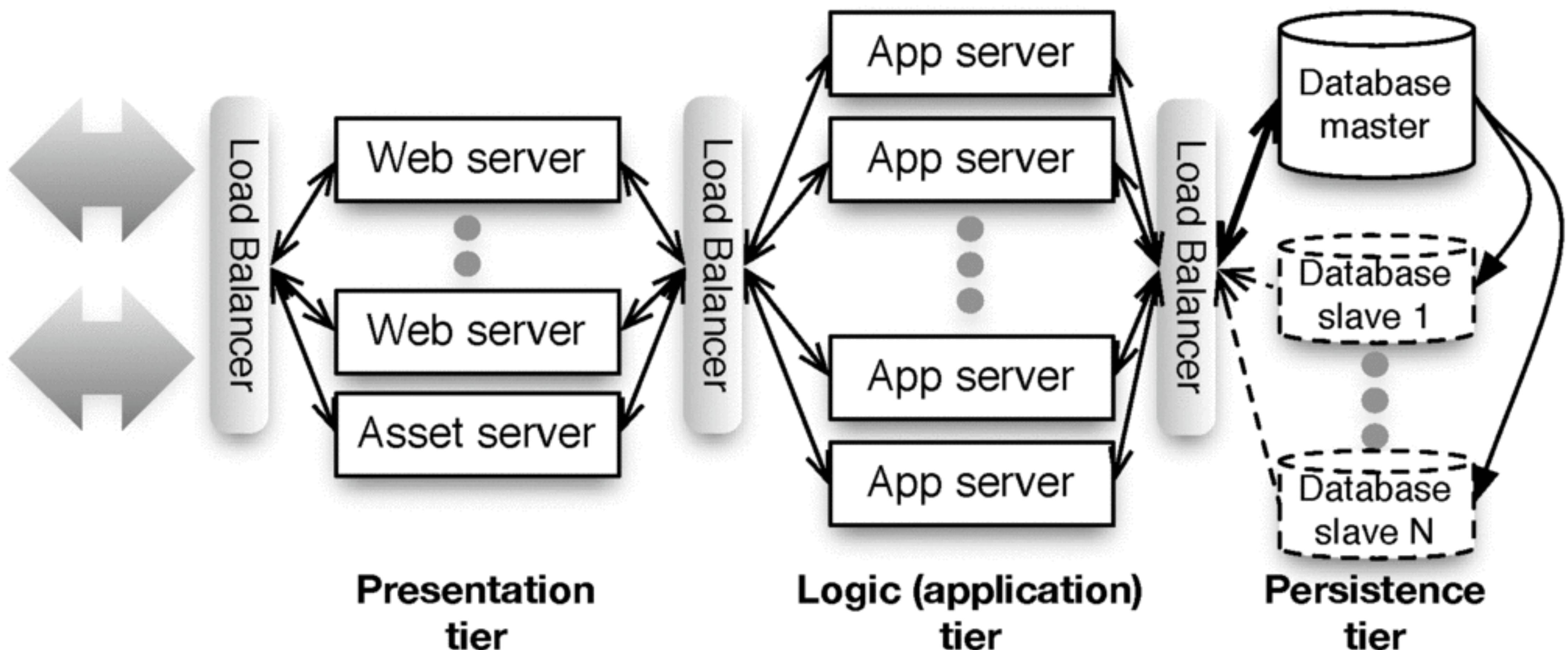
Template view
Scaffolding



ORM
Active Record
Data Mapper

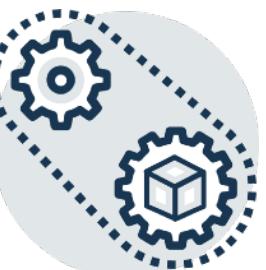


“Shared-nothing”



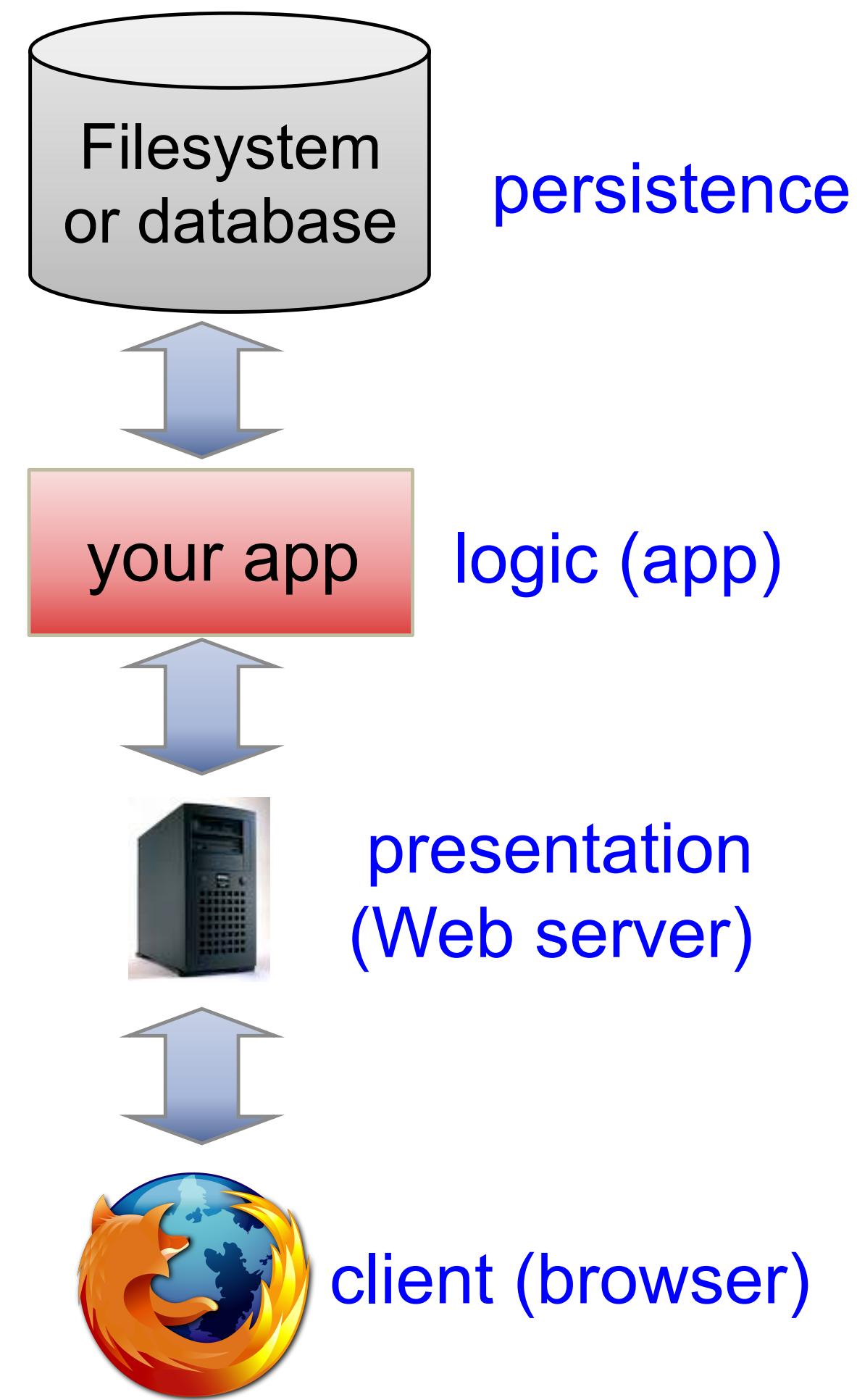
Geração de conteúdo dinâmico

- Antigamente a maioria das páginas da web eram (coleções de) arquivos antigos simples
- Mas os sites mais interessantes da Web1.0/e-commerce, na verdade, executavam um programa para gerar a "página"
- Originalmente: templates com "trechos" de código incorporados
- Eventualmente, o código tornou-se "rabo que abanou o cachorro" e mudou-se para fora do servidor Web



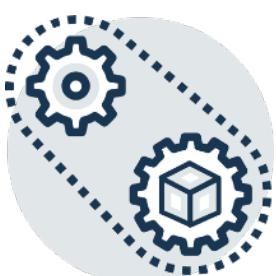
Sites que são programas (SaaS)

- Como você:
 - “mapeia” a URI para o programa e função correta?
 - Passa argumentos?
 - Evoca programas em um servidor?
 - Trata armazenamento persistente?
 - Trata cookies?
 - Trata erros?
 - Empacota saídas de volta para o usuário?
- Frameworks dão suporte a estas tarefas comuns



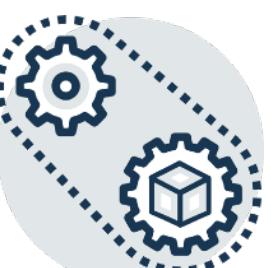
Sumário: Web 1.0 SaaS

- Browser requisita recursos (URI) usando HTTP
 - HTTP é um protocolo simples de requisição-resposta que depende do TCP/IP
 - Em SaaS, a maioria das URIs disparam eventos de uma aplicação
- HTML é utilizado para codificar conteúdo, CSS para estilizar o visual
- Cookies permitem ao servidor rastrear os clientes
 - O browser automaticamente envia o cookie para o servidor em cada requisição
 - O servidor pode atualizar o cookie em cada resposta
- Frameworks abstraem tudo isso de maneira conveniente para os desenvolvedores
- ...e ajuda a mapear SaaS para uma arquitetura 3-camadas, shared-nothing



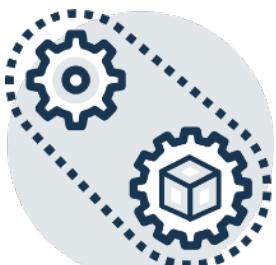
AJAX chegou (Asynchronous JavaScript And XML, a/k/a “Web 2.0”)

- Antes: a resposta HTTP contém conteúdo a ser exibido no navegador
- Depois: a resposta HTTP contém dados
 - O navegador organiza os dados de resposta para uma função de manipulador JavaScript específica
 - Essa função pode usar dados para modificar o que já está sendo exibido (“aplicativo de página única”)
- Então requisição/resposta HTTP <-> chamando uma função!
- Web como conjunto de serviços compostos independentes:
Arquitetura orientada a serviços



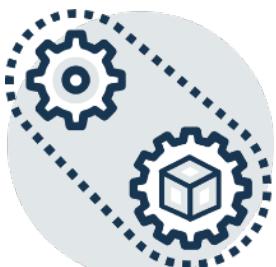
Arquitetura de Software

Você pode/consegue **projetar** um software de forma que seja possível **recombinar** módulos **independentes** para ofertar **diferentes** apps sem muito **esforço** de programação?



Arquitetura Orientada a Serviço

- SOA: Arquitetura de Software onde todos os componentes são projetados como serviço
- Apps são compostas por serviços interoperáveis
 - Fácil de derivar uma nova versão para um conjunto de usuários
 - Também é fácil para se recuperar de um erro de projeto
- Contraste para “SW silo” sem uma API interna



CEO: Amazon shall use SOA!

1. “All teams will henceforth expose their data and functionality through service interfaces.
2. “Teams must communicate with each other through these interfaces.
3. “There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.



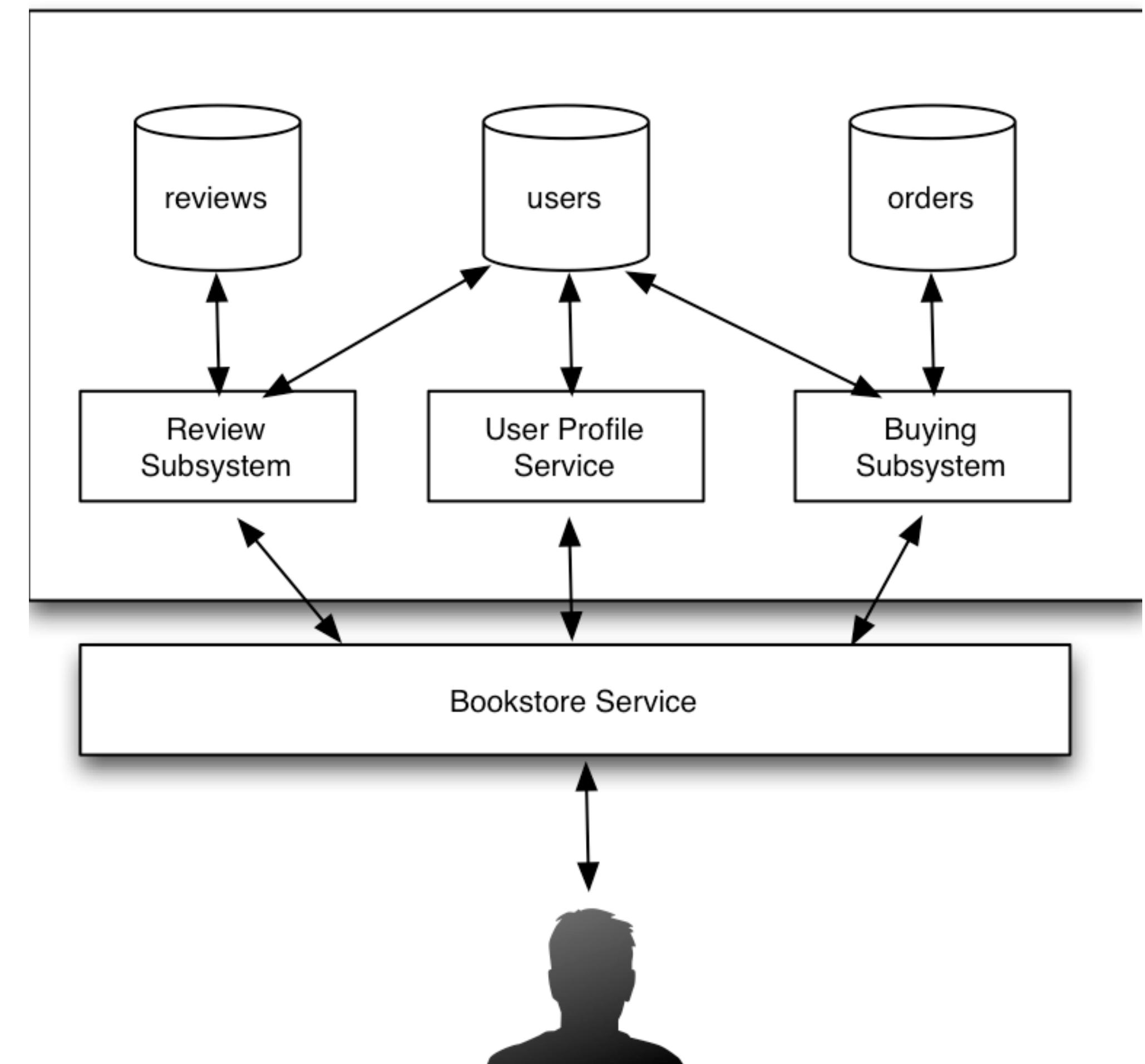
CEO: Amazon shall use SOA!

4. “It doesn't matter what [API protocol] technology you use.
5. “Service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. “Anyone who doesn't do this will be fired.
7. “Thank you; have a nice day!”



Amazon: início de SOA!

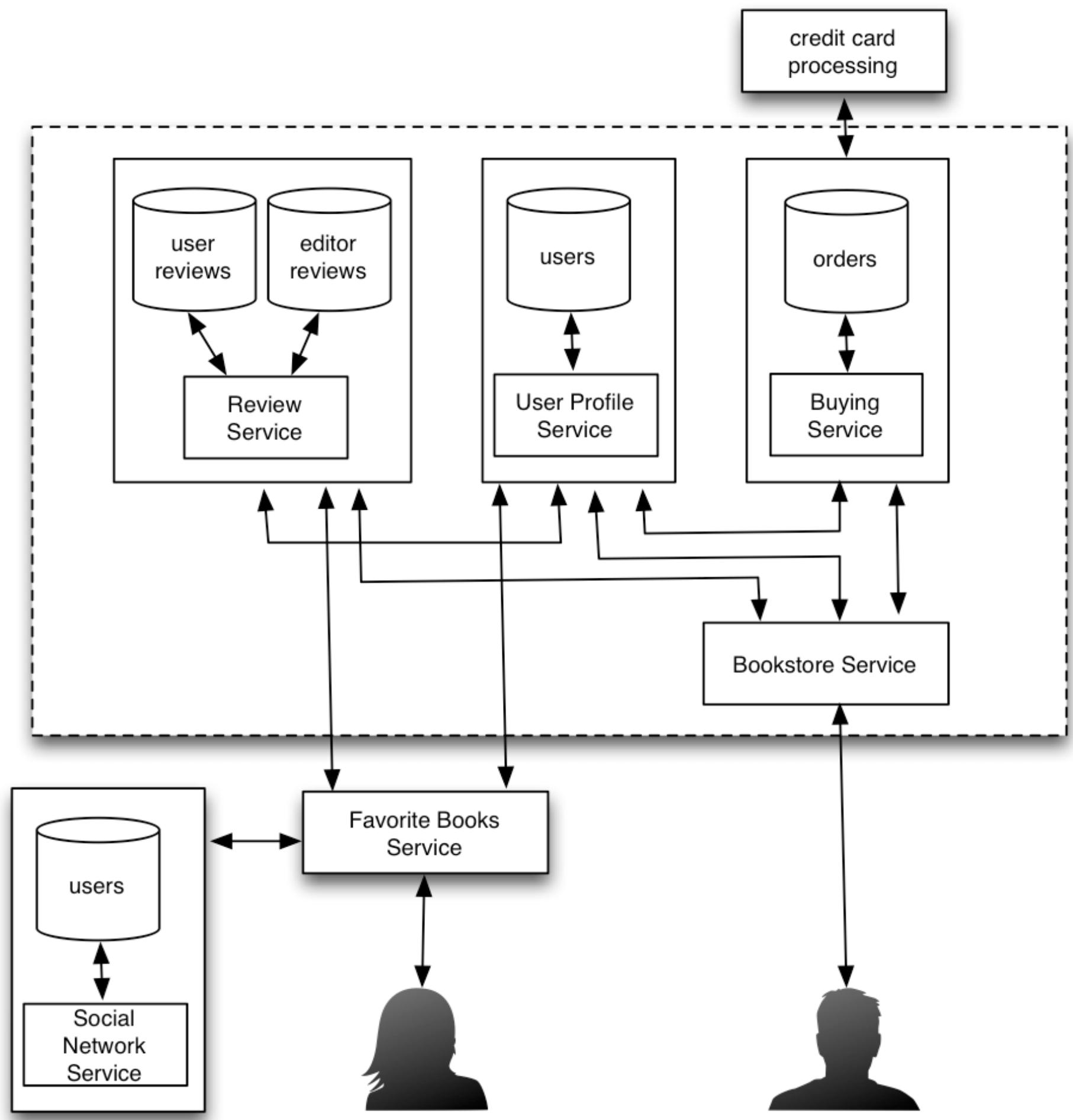
- Subsistemas internos podem compartilhar dados diretamente
- User profile Service
- Todos os subsistemas “dentro” de uma única API (“Bookstore”)



(Figure 1.3, Engineering Long Lasting Software by Armando Fox and David Patterson, Beta edition, 2012.)

Amazon: início de SOA!

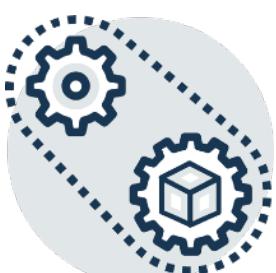
- Subsistemas independentes, como se estivessem em datacenters separados
- User Service API
- Pode recombinar serviços para criar novos serviços (“Favorite Books”)



(Figure 1.4, *Engineering Long Lasting Software* by Armando Fox and David Patterson, Beta edition, 2012.)

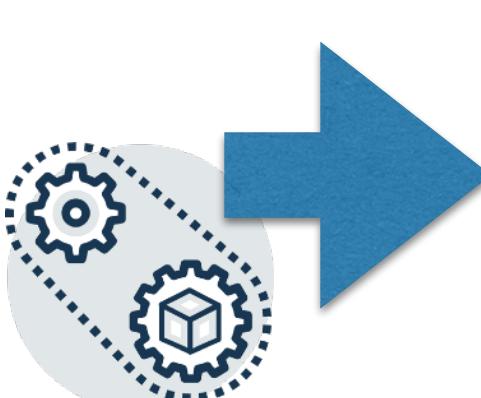
Pergunta

- Quando dois serviços são compostos em uma SOA, quais condições abaixo **são** necessárias, se houver?
 - Ambos devem ser escritos na mesma linguagem/estrutura
 - O código que “une” as informações dos dois serviços deve ser código JavaScript em execução no navegador
 - Os dois serviços não devem exigir uma etapa de autenticação (login)
 - Nenhuma das opções acima são condições necessárias



Pergunta

- Quando dois serviços são compostos em uma SOA, quais condições abaixo **são** necessárias, se houver?
 - Ambos devem ser escritos na mesma linguagem/estrutura
 - O código que “une” as informações dos dois serviços deve ser código JavaScript em execução no navegador
 - Os dois serviços não devem exigir uma etapa de autenticação (login)
 - Nenhuma das opções acima são condições necessárias



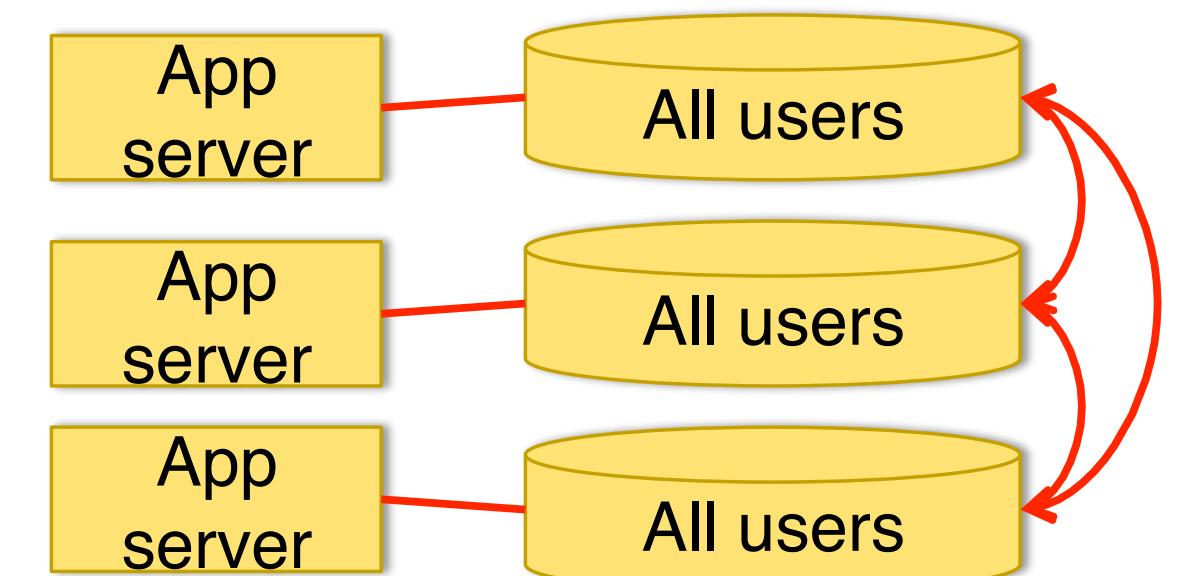
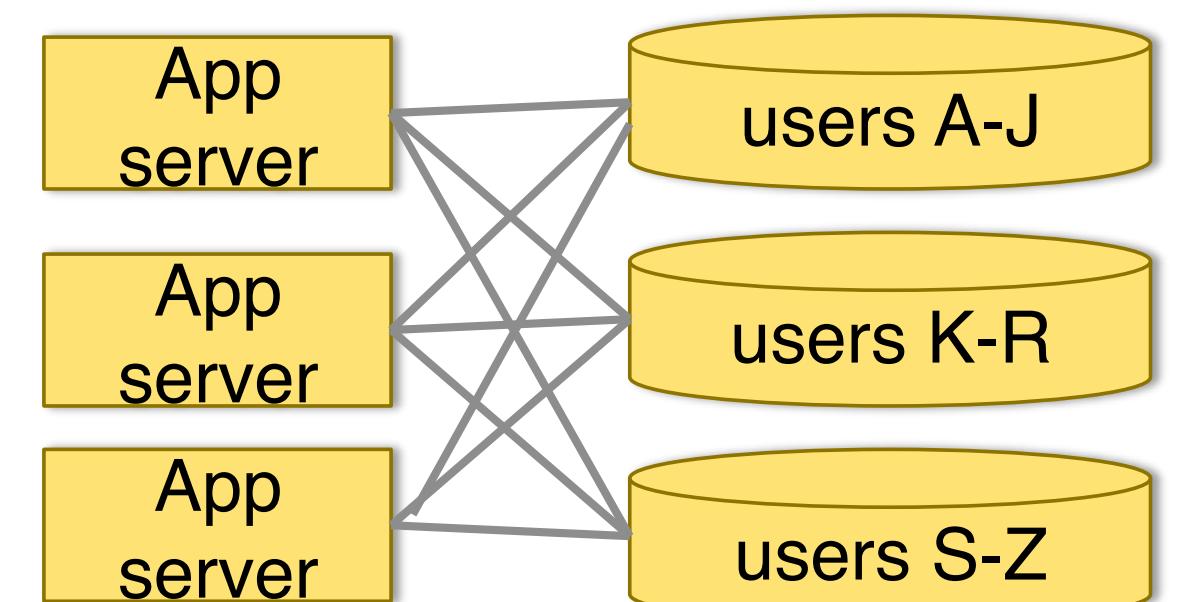
Pós e Contras de SOA

- Reusabilidade
- Melhor ferramenta para o trabalho
- Mais fácil para testas
- Mais ágil e amigável
- Você constrói, você roda
- Estado dividido entre serviços
- Todos os dados serializados na fronteira
- Mais difícil de testar
- Gerenciamento de desempenho e falhas
- Dev/ops vs. trabalhos separados

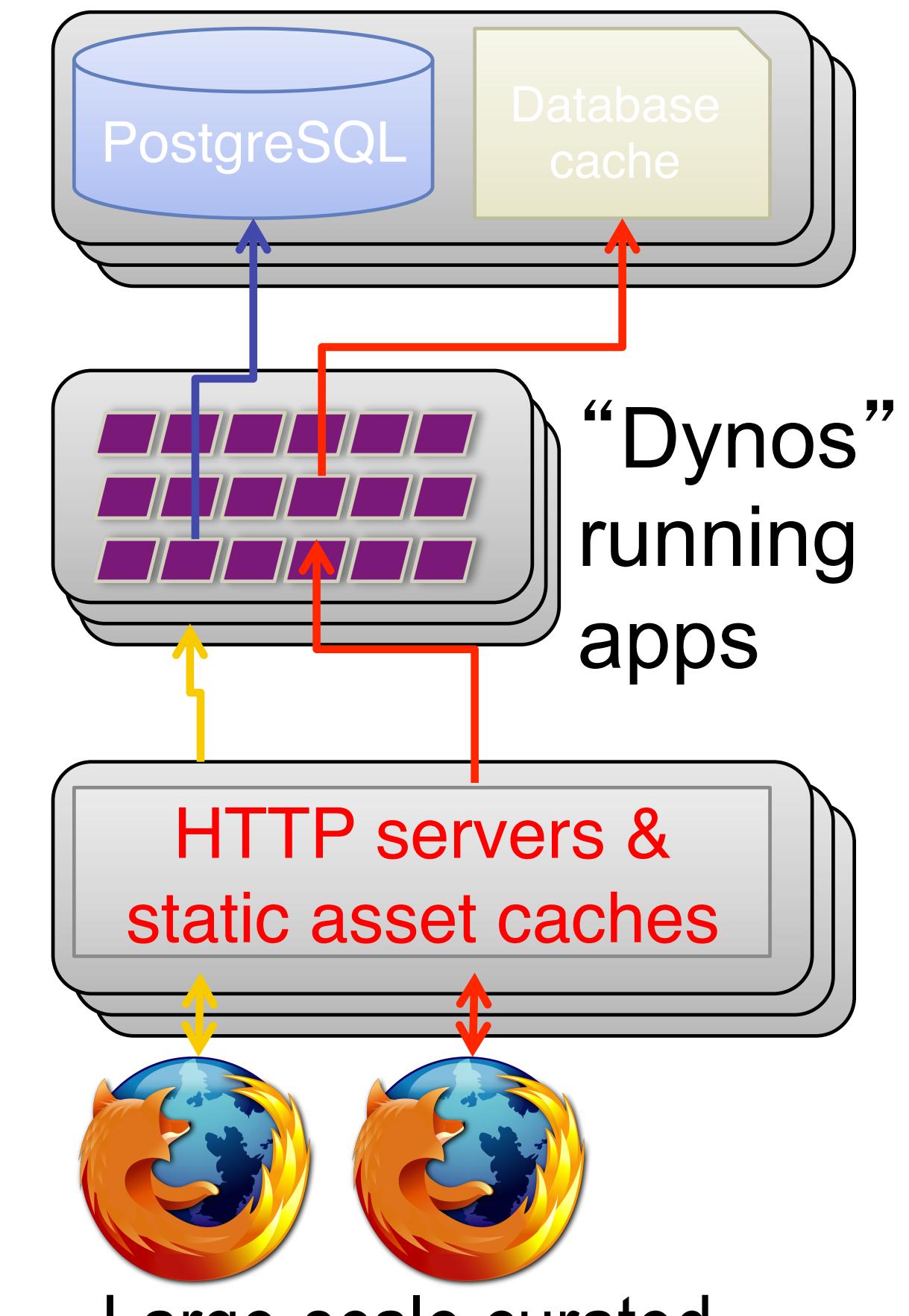
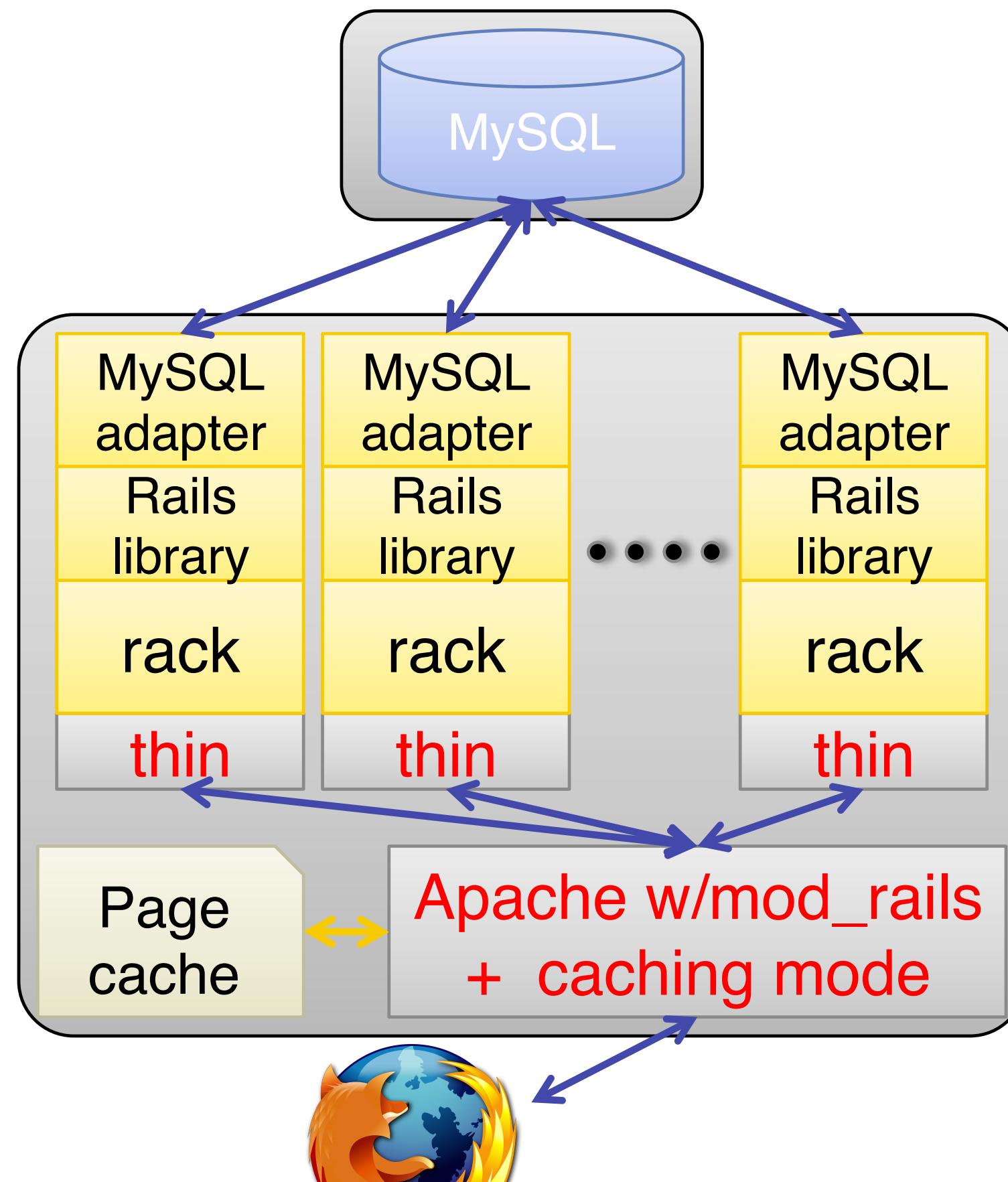
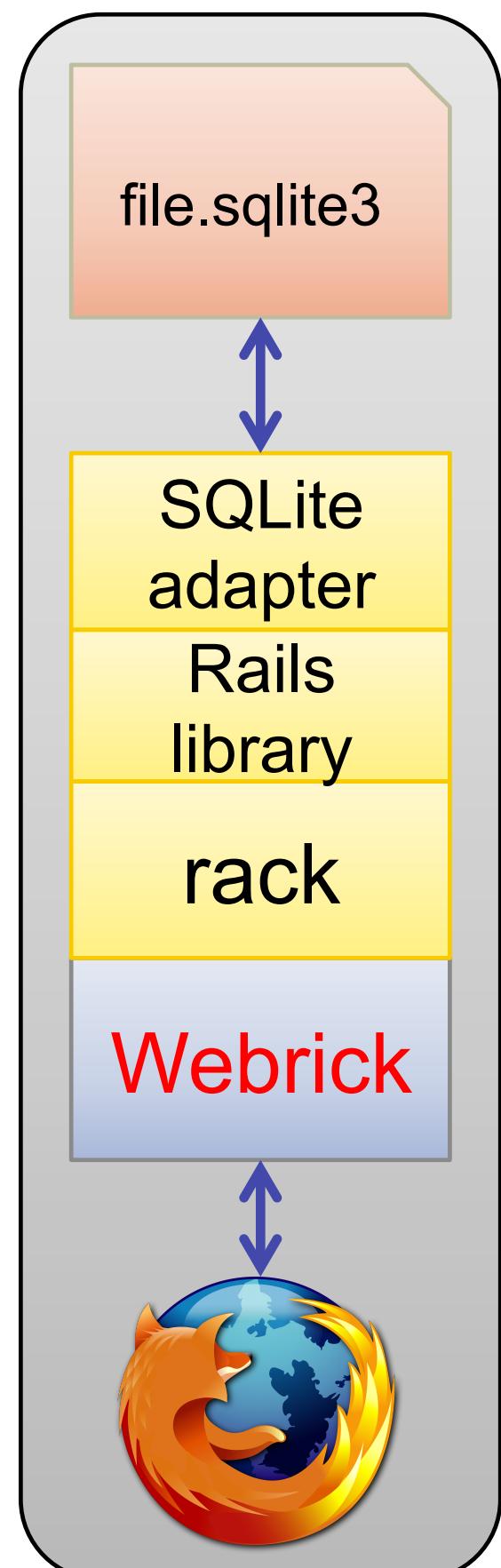


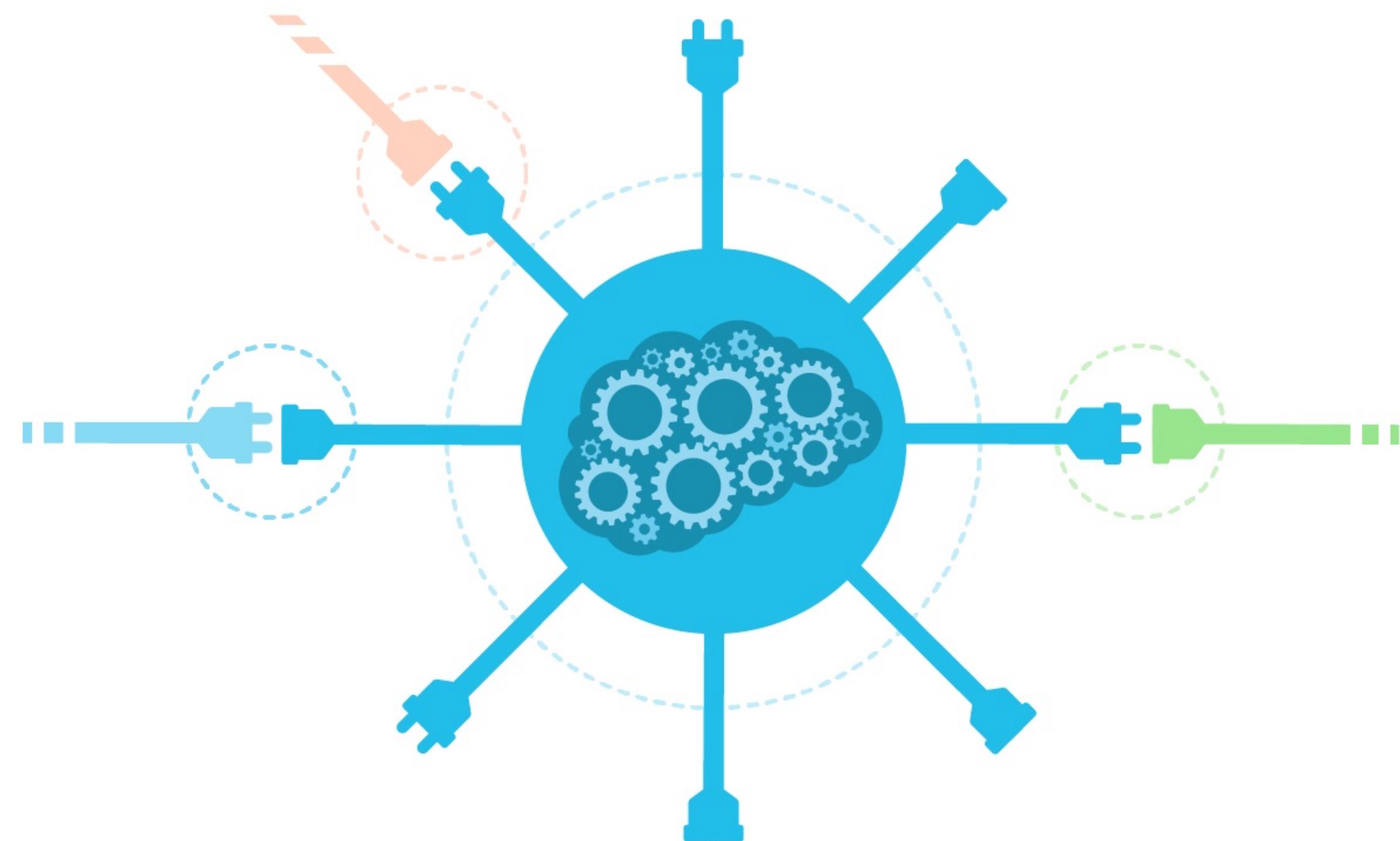
Sharding vs. Replication

- Particionamento dos dados através dos “shards”
 - + Escala
 - Ruim quando operações acessam >1 tabelas
 - Ex: user profile
- Replicação em todos os lugares
 - + Consultas multi-table rápidas
 - Difícil de escalar: escritas devem ser propagadas (inconsistência temporária)
 - Ex.: Posts no mural facebook /"likes"



Ambiente de Desenvolvimento vs. Implantação em Média-Escala



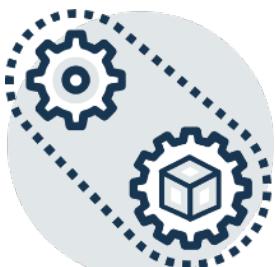


API's SaaS



Application Programming Interface (API) é

- ... Um “contrato” documentado entre o chamador (deseja usar um serviço fornecido) e o chamado



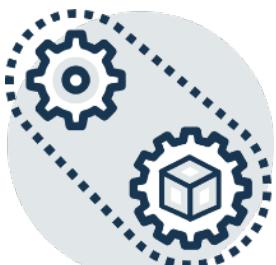
O que uma API deve especificar?

1. Como o chamador identifica e localiza a função a ser chamada?
2. Como passar argumentos obrigatórios e opcionais?
3. Como o chamador recebe o valor de retorno?
4. O que acontece em caso de erro no chamador?



Identificar o chamador?

- SaaS: endpoint URI recebe solicitação HTTP
 - rota consiste em URI base, método, caminho, argumentos opcionais e obrigatórios
 - operação geralmente primeiro componente de caminho do URI
- por exemplo, docs em developers.themoviedb.org
 - `GET /search/movie?params`
 - `GET /movie/:movie_id`



Passagem de parâmetro?

- For GET, embed in URI query string
- For POST, can post as part of form body
 - GET /search/movie?query=Coco

Método

URI base, versão da API

• GET https://api.themoviedb.org/3/search/movie?query=Coco

operação

parâmetros

• GET https://api.themoviedb.org/3/search/movie?
query=Coco&api_key=...

Obtendo resultados no retorno: JSON (JavaScript Object Notation)

- JSON object = hash não ordenado em { }
- Chaves são strings; valores podem ser string, number, [Array], ou {hash aninhado}
- Par chave/valor separado por vírgulas; espaço em branco é irrelevante

```
{ "title": "Coco",
  "rating": "G",
  "genres": [{ "id": 16, "name": "Animation" },
              { "id": 10751, "name": "Family" }],
  "vote_average": 8.1
}
```

E agora?

- API docs: `GET /movie/:movie_id`
retorna detalhes de um filme específico
- De onde vem o valor de `movie_id`?
 - A. Você procura nos documentos da API
 - B. De uma chamada anterior para `/movie/search`
 - C. É apenas o título do filme (ou uma substring do título do filme)



40



E agora?

- API docs: `GET /movie/:movie_id`
retorna detalhes de um filme específico
 - Para obter um URI totalmente qualificado para esta chamada,
precisamos adicionar todos os seguintes itens, exceto:
- A. `api_key=...` parâmetro na URI
 - B. Parâmetros adicionais da consulta (argumentos) na URI
 - C. URI base

`GET https://api.themoviedb.org/3/movie/354912?api_key= ...`

