



IMD0043

CAMADA DE APLICAÇÃO DNS E HTTP

The diagram illustrates the OSI model as a stack of five horizontal layers. From top to bottom, the colors of the layers are green, yellow, light blue, pink, and orange. Each layer contains its name in bold, uppercase letters. The top layer is labeled 'CAMADA DE APLICAÇÃO' in black text. The second layer is labeled 'CAMADA DE TRANSPORTE' in grey text. The third layer is labeled 'CAMADA DE REDE' in grey text. The fourth layer is labeled 'CAMADA DE ENLACE' in grey text. The bottom layer is labeled 'CAMADA FÍSICA' in grey text.

CAMADA DE APLICAÇÃO

CAMADA DE TRANSPORTE

CAMADA DE REDE

CAMADA DE ENLACE

CAMADA FÍSICA

O QUE É DNS?

- ▶ Usuário tem o nome da entidade que quer acessar
 - ▶ Ex: www.imd.ufrn.br
- ▶ Porém, a Internet roteia e encaminha requisições **baseado em endereços IP**
 - ▶ Precisamos converter um nome em um endereço IP
- ▶ *Domain Name System (DNS)*
 - ▶ Provê mapeamento de nome em endereço IP
 - ▶ Usuário pergunta ao DNS: qual o endereço IP para www.imd.ufrn.br
 - ▶ DNS responde: 177.20.147.222

REQUISITOS

Endereços podem mudar

- ▶ Mover www.imd.ufrn.br para 186.192.81.5
- ▶ Usuários/aplicações não devem ser afetadas!

Um nome pode ser mapeado para múltiplos endereços IP

- ▶ www.globo.com para múltiplas instâncias de um website
- ▶ Permite **balanceamento de carga** ou então para **reduzir latência**

Podemos ter múltiplos nomes para o mesmo endereço

- ▶ Ex: www.imd.ufrn.br e imd.ufrn.br devem mapear para o mesmo endereço IP

OBJETIVOS E ABORDAGEM

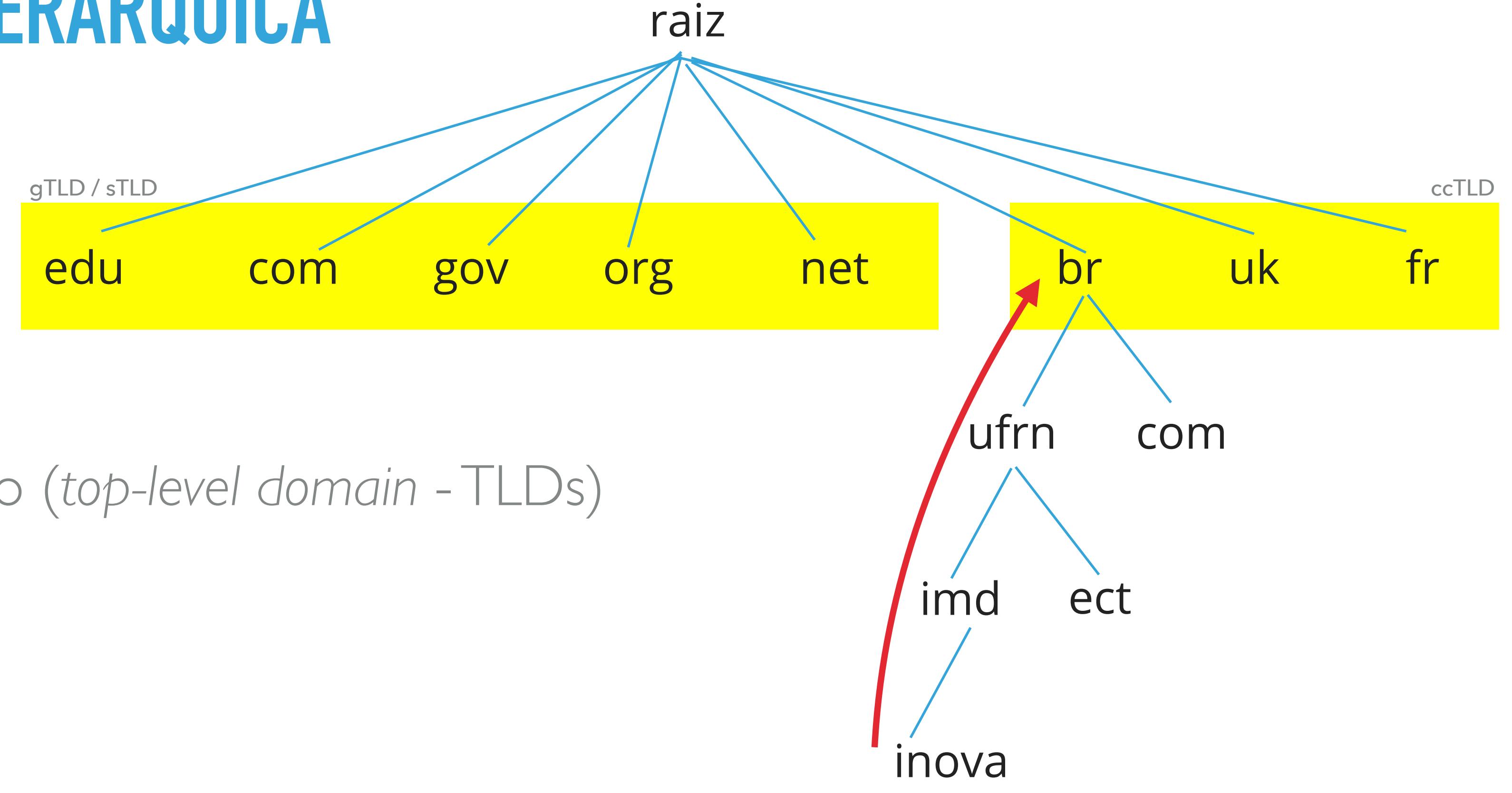
Objetivos

- ▶ Atender os requisitos (os descritos no slide anterior)
- ▶ Escalável
- ▶ Fácil gerenciamento
- ▶ Disponibilidade e consistência
- ▶ Buscas/consultas rápidas

Abordagem

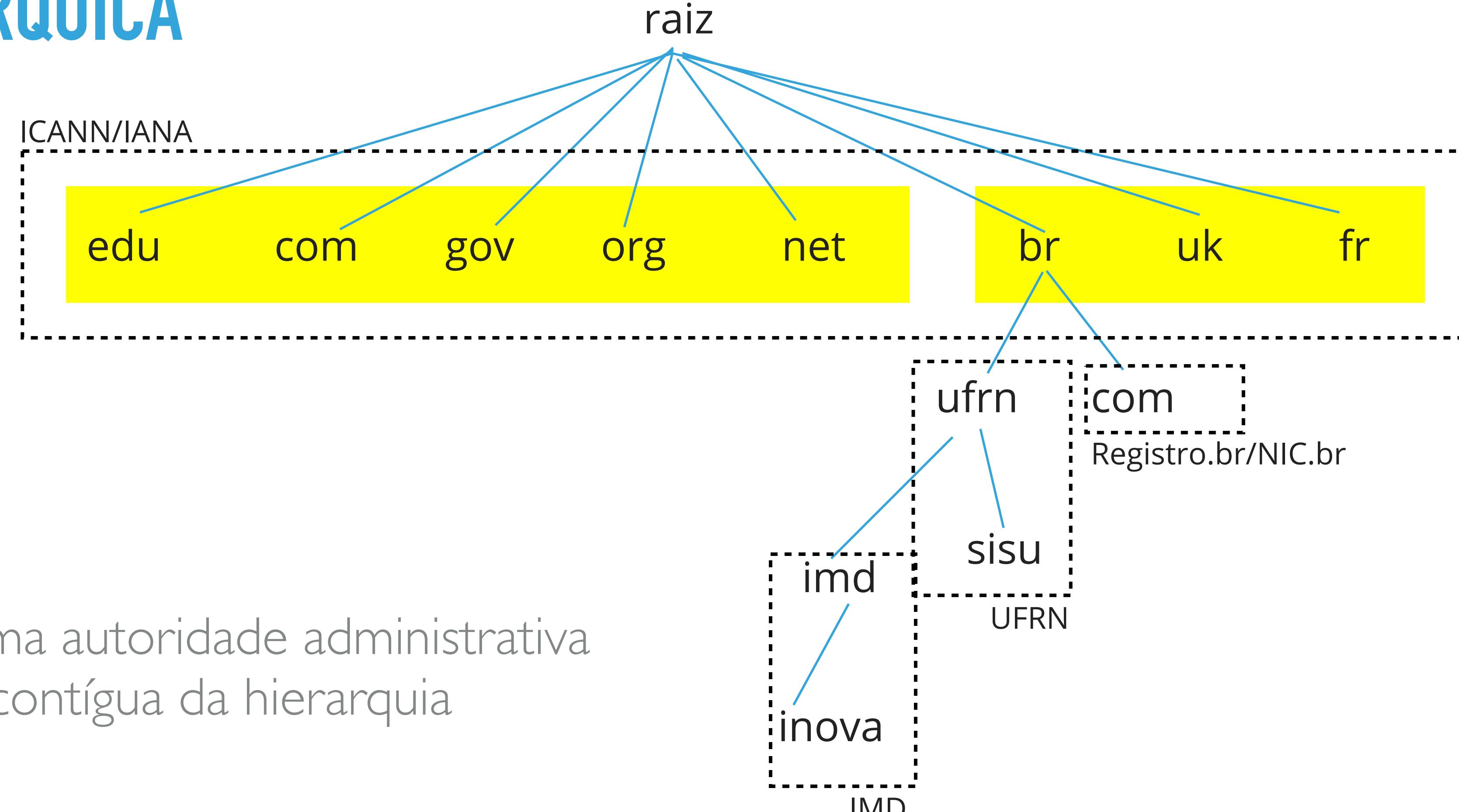
- ▶ Organização hierárquica dos nomes: estrutura hierárquica dos nomes
- ▶ Administração hierárquica: hierarquia da autoridade sobre os nomes
- ▶ Infraestrutura hierárquica: hierarquia dos servidores DNS

ESTRUTURA DE NOMES HIERÁRQUICA



- ▶ Servidores de domínio de topo (*top-level domain - TLDs*) estão no topo
- ▶ Domínios são sub-árvores
 - ▶ Ex: ufrn.br, imd.ufrn.br
- ▶ Nome é o no sentido folha para raiz
 - ▶ inova.imd.ufrn.br

ADMINISTRAÇÃO HIERÁRQUICA



- ▶ Uma zona corresponde a uma autoridade administrativa responsável por uma parte contígua da hierarquia
- ▶ UFRN controla *.ufrn.br
- ▶ IMD controla *.imd.ufrn.br

INFRAESTRUTURA HIERÁRQUICA

- ▶ Topo da hierarquia: raiz (root)
 - ▶ Críticos: primeiro passo para a tradução de nomes
- ▶ Próximo nível: servidores TLDs
 - ▶ .com, .edu, .br
- ▶ Níveis mais baixos: servidores DNS autoritativos
 - ▶ Realizam o mapeamento

DISPONIBILIDADE POR DOMÍNIO

- ▶ Servidores DNS são **replicados**
 - ▶ Servidores de nomes primário e secundário são requeridos
 - ▶ O serviço de nomes estará disponível se pelo menos uma instância está operacional
 - ▶ Consultas podem ser balanceadas entre as réplicas

QUEM SABE O QUE?

- ▶ Todo servidor DNS sabe o endereço de um **servidor raiz** (*root name server*)
- ▶ Servidores raiz sabem o endereço de todos os servidores TLDs
- ▶ Todo nó sabe o endereço de todos seus filhos
- ▶ Um servidor DNS **autoritativo** armazena mapeamentos nome-endereço (**registro de recursos**) para todos os nomes do domínio que ele tem autoridade
- ▶ Portanto, cada servidor:
 - ▶ Armazena somente um sub-conjunto do banco de dados global DNS (escalável!)
 - ▶ Pode descobrir servidores para qualquer parte da hierarquia

BENEFÍCIOS DESSA ABORDAGEM

- ▶ Escalável nos nomes, atualizações, consultas, usuários, ...
- ▶ Alta disponibilidade: domínios replicam independentemente
- ▶ Extensível: podemos adicionar TLDs atualizando somente os servidores raiz
- ▶ Administração autônoma
 - ▶ Cada domínio gerencia seus próprios nomes e servidores
 - ▶ Pode delegar em níveis mais baixos
 - ▶ Garante unicidade e consistência das bases de nomes

REGISTROS DNS

- ▶ Servidores DNS armazenam **registros de recursos** (RRs)
 - ▶ RR é (nome, valor, tipo, TTL)

Tipo = A (Address) - (AAAA para IPv6)

- ▶ nome = nome do host
- ▶ valor = endereço IPv4

Tipo = NS (Name Server)

- ▶ nome = nome do domínio
- ▶ valor = nome do servidor DNS para o domínio

REGISTROS DNS

Tipo = MX (*Mail eXchanger*)

- ▶ nome = domínio do e-mail
- ▶ valor = nome do(s) servidor(es) de e-mail

Tipo = CNAME (*Canonical NAME*)

- ▶ nome = apelido (*alias*)
- ▶ valor = nome canônico

Tipo = PTR (*PoinTeR*)

- ▶ nome = IP reverso
- ▶ valor = nome do host correspondente

ADICIONANDO REGISTROS DE RECURSOS NO DNS

- ▶ Exemplo: criamos a empresa Fulano
- ▶ Recebemos um bloco de endereços IP do nosso provedor de serviços
 - ▶ Ex: 212.44.9.128/25
- ▶ Registrarmos fulano.com.br (no Registro.br)
 - ▶ Provemos ao Registro.br os nomes e IPs do seu(s) servidor(es) de nomes autoritativos
 - ▶ Registro.br insere RRs para o servidor .com.br:
 - ▶ (fulano.com.br, dns1.fulano.com.br, NS)
 - ▶ (dns1.fulano.com.br, 212.44.9.129, A)
 - ▶ Armazena RRs no seu servidor dns1.fulano.com.br
 - ▶ Ex: registro tipo A para www.fulano.com.br
 - ▶ Ex: registro tipo MX para fulano.com.br

SERVidores DNS RAIZ

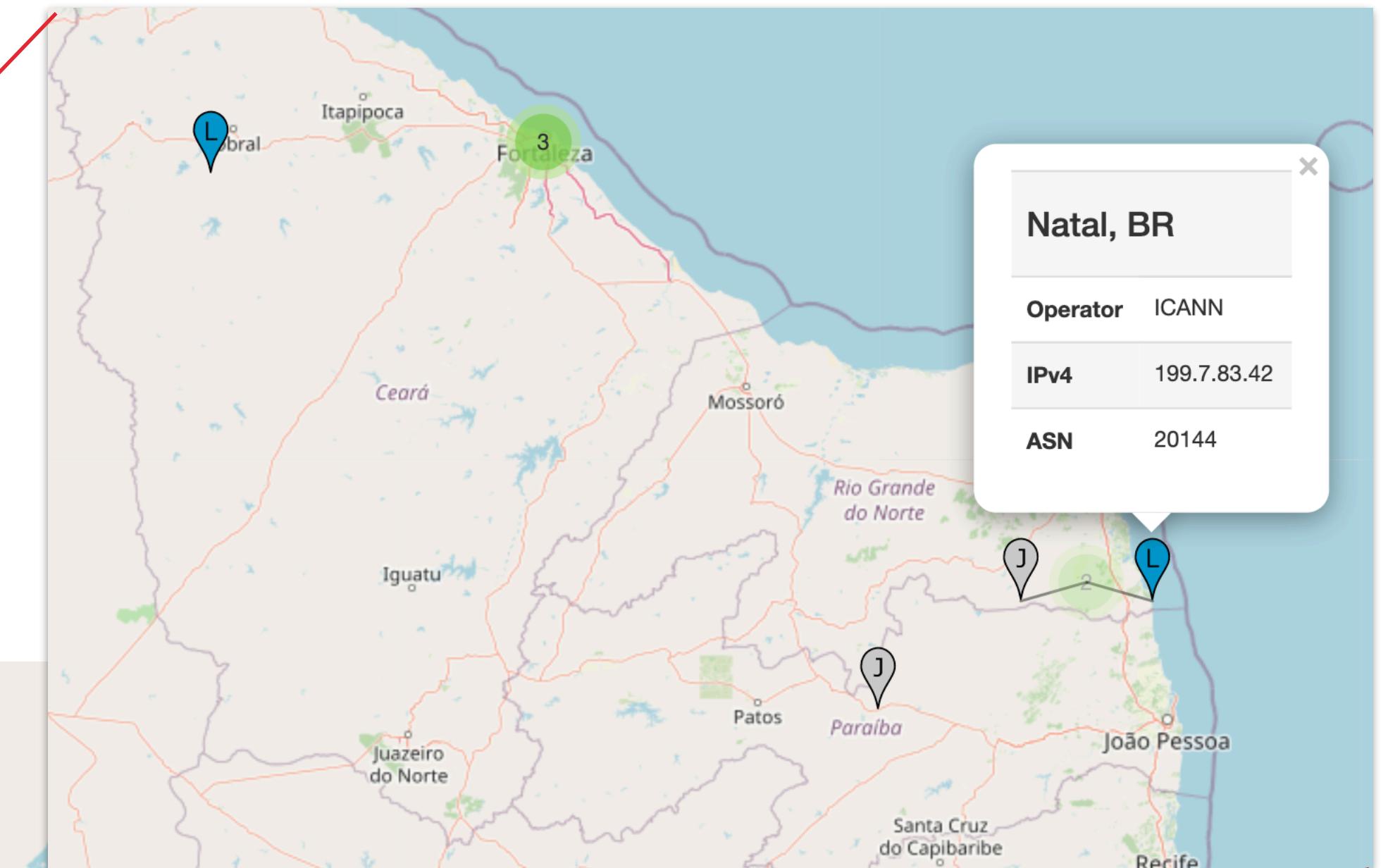
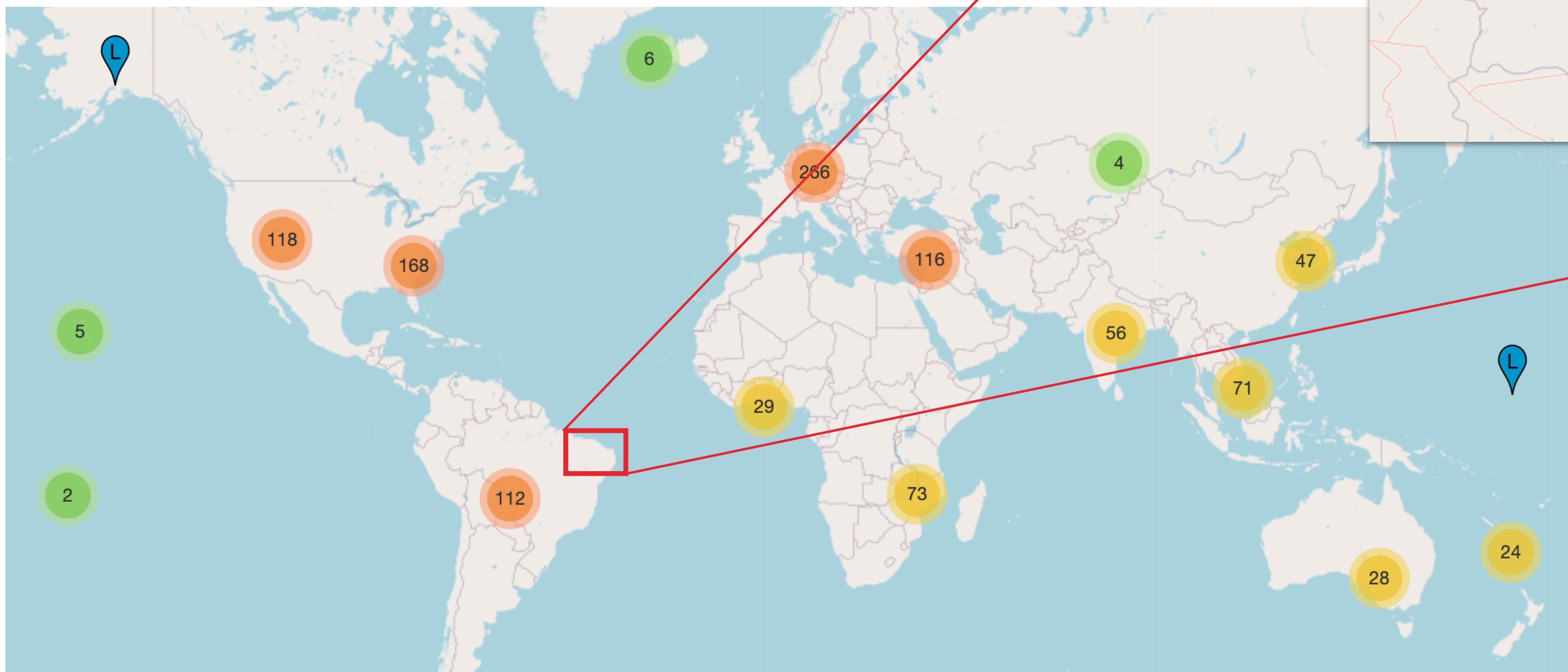
- ▶ 13 servidores raiz (root servers)
- ▶ Rotulados de A até M
- ▶ <https://root-servers.org>
- ▶ [a-m].root- servers.net
- ▶ Como escalar?



ANYCAST

- ▶ Roteamento encontra o menor caminho até o destino
- ▶ Se várias localidades recebem o mesmo endereço:
 - ▶ Rede vai entregar o pacote para a localização mais próxima com aquele endereço
 - ▶ Isso é chamado *anycast*
 - ▶ Não requer modificação do roteamento para isso...

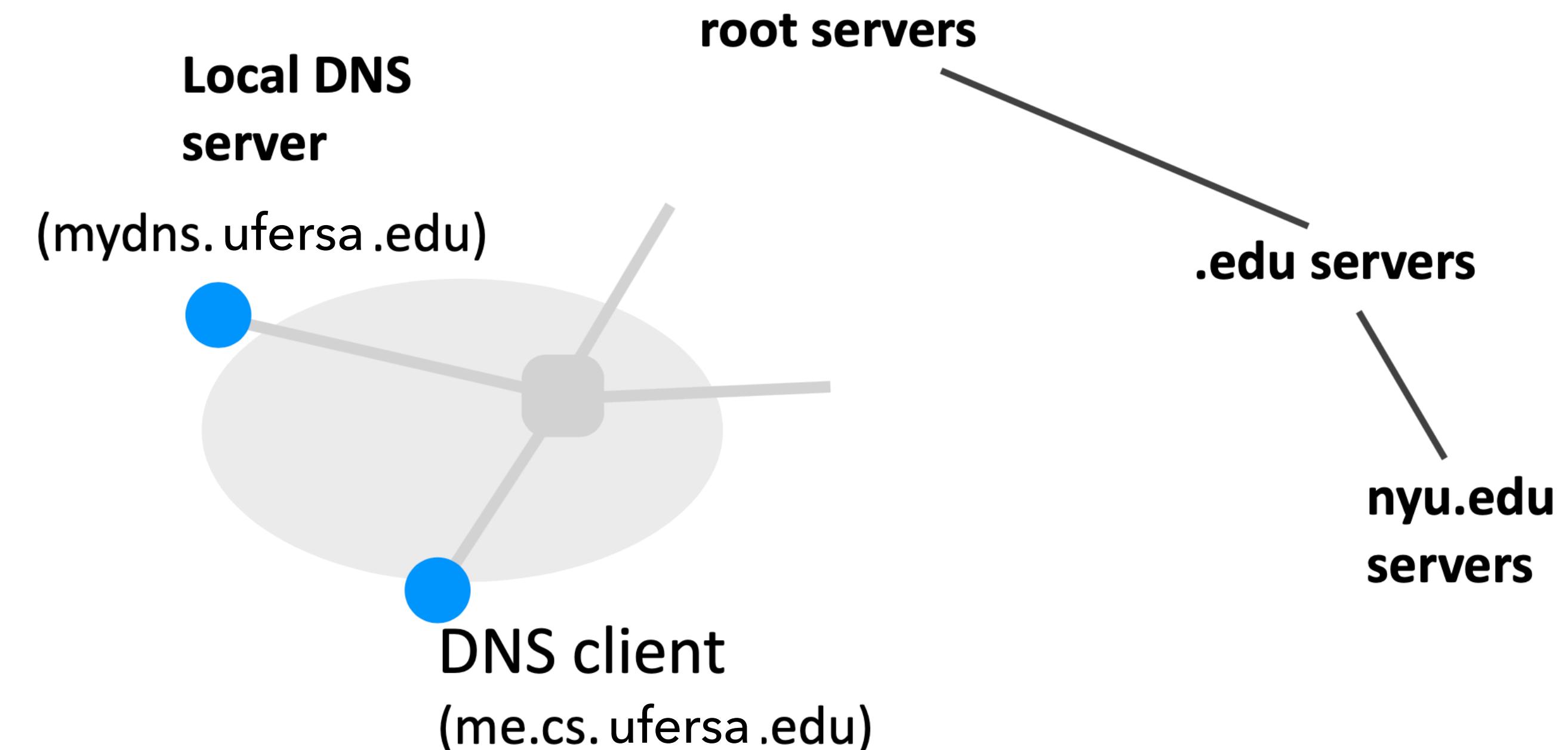
ANYCAST



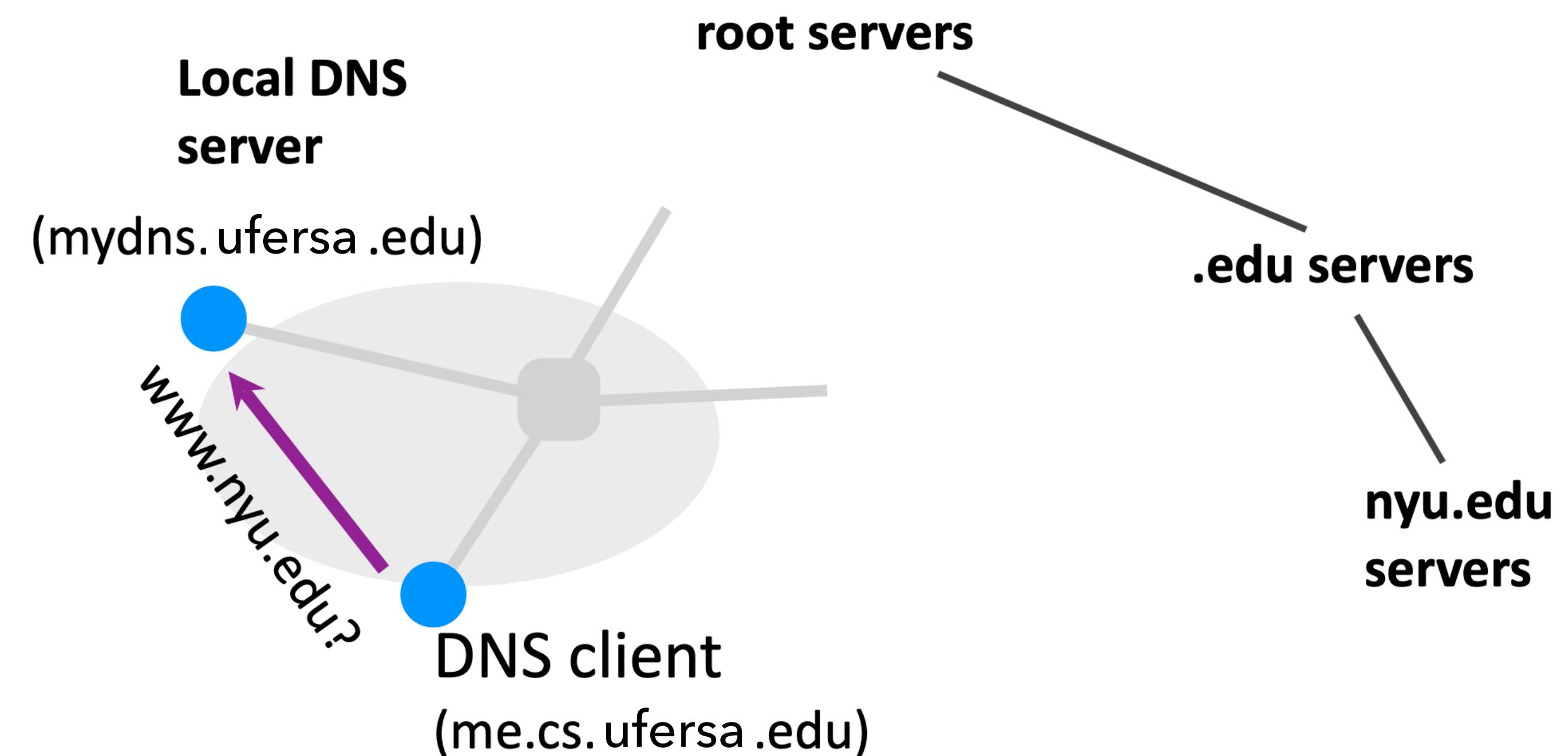
USANDO DNS

- ▶ Dois componentes
 - ▶ Servidores DNS locais
 - ▶ *Resolver* nos hosts
- ▶ Servidor DNS local (*default name server*)
 - ▶ Normalmente próximo ao dispositivo final que o utiliza
 - ▶ Hosts configuram manualmente (`/etc/resolv.conf`) ou aprendem através do DHCP
- ▶ Aplicação cliente
 - ▶ Obtém o nome (através de, por exemplo, a URL)
 - ▶ Utilizam *resolver* para “resolver” o nome

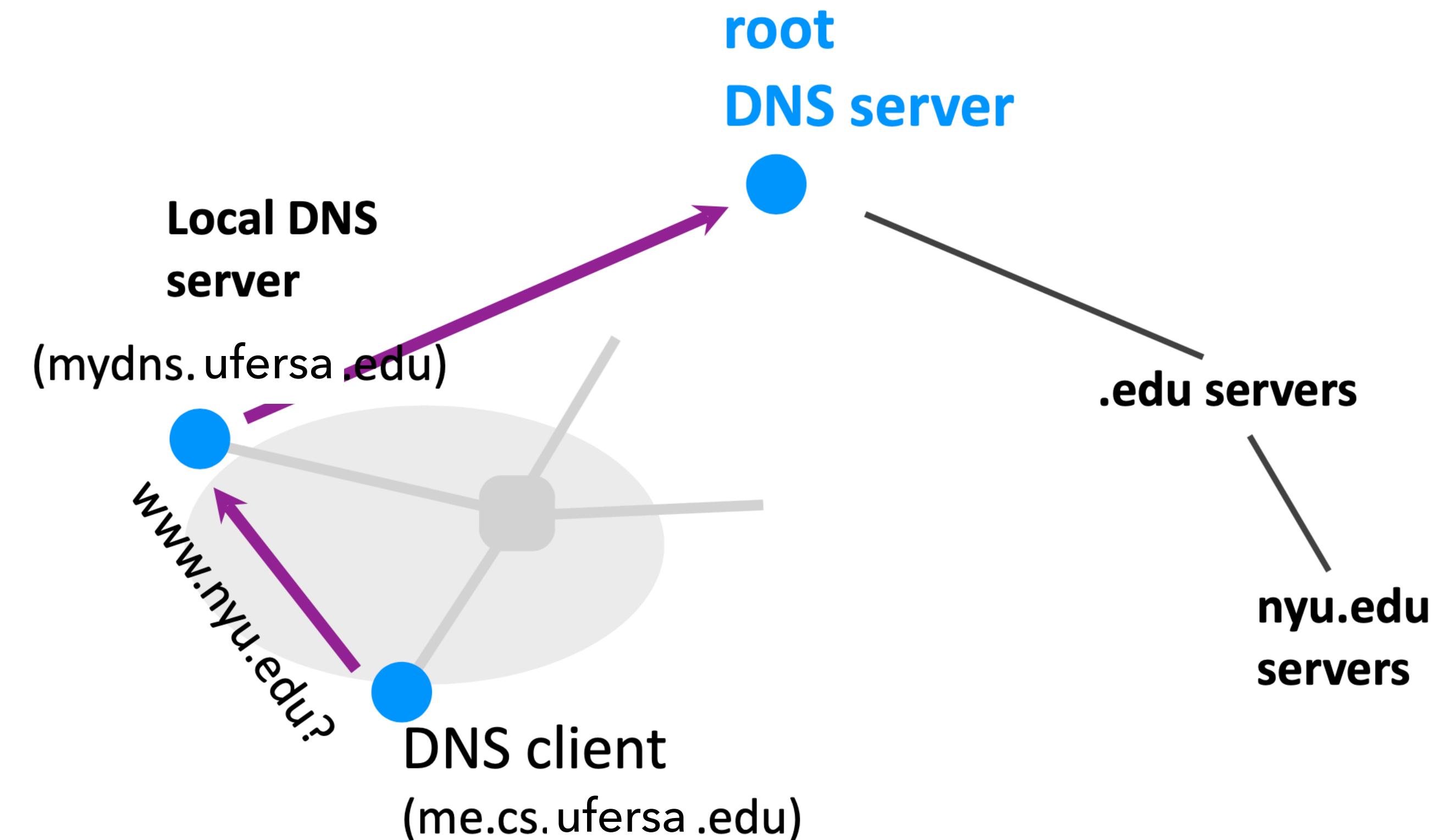
USANDO DNS



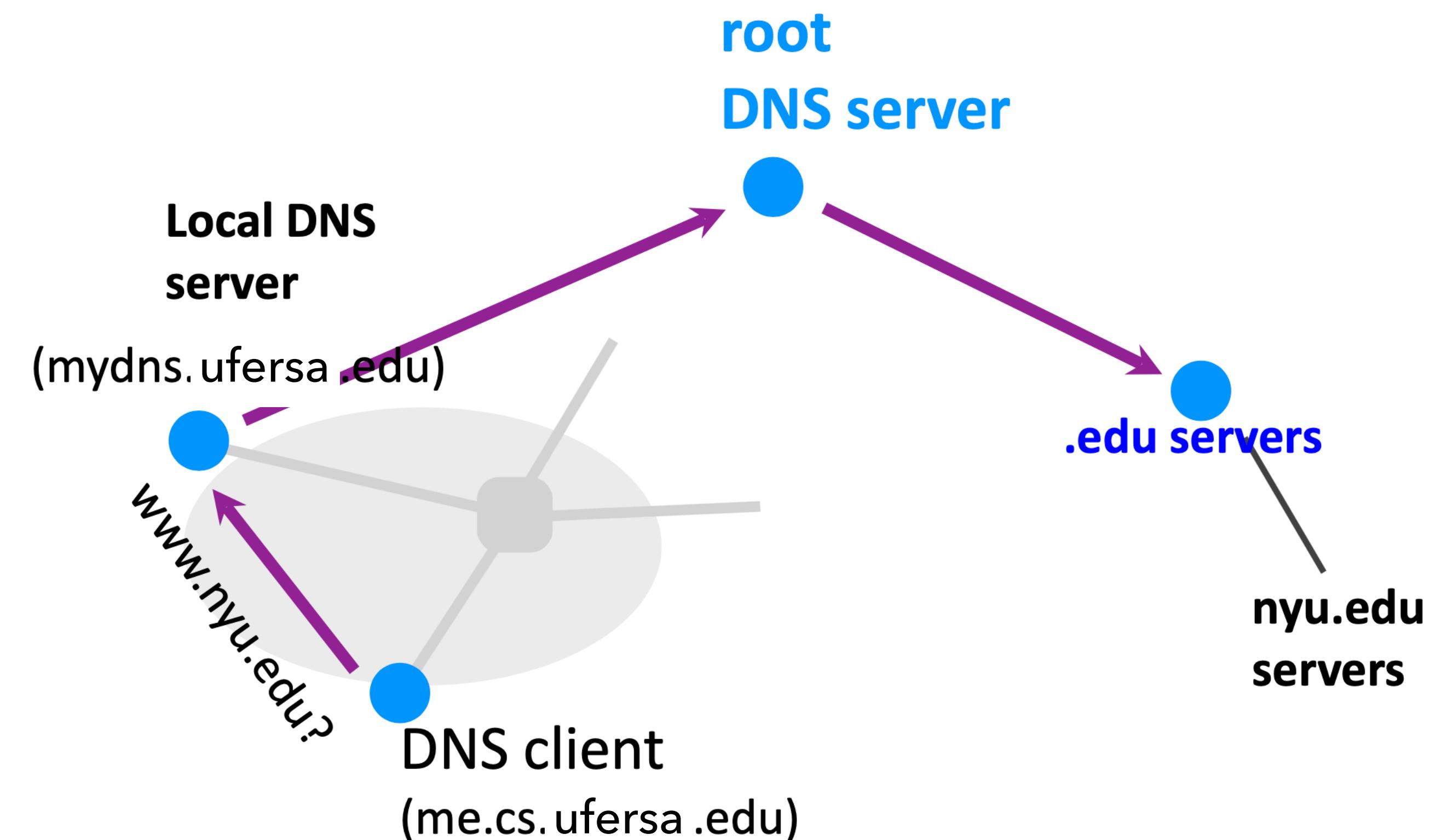
USANDO DNS



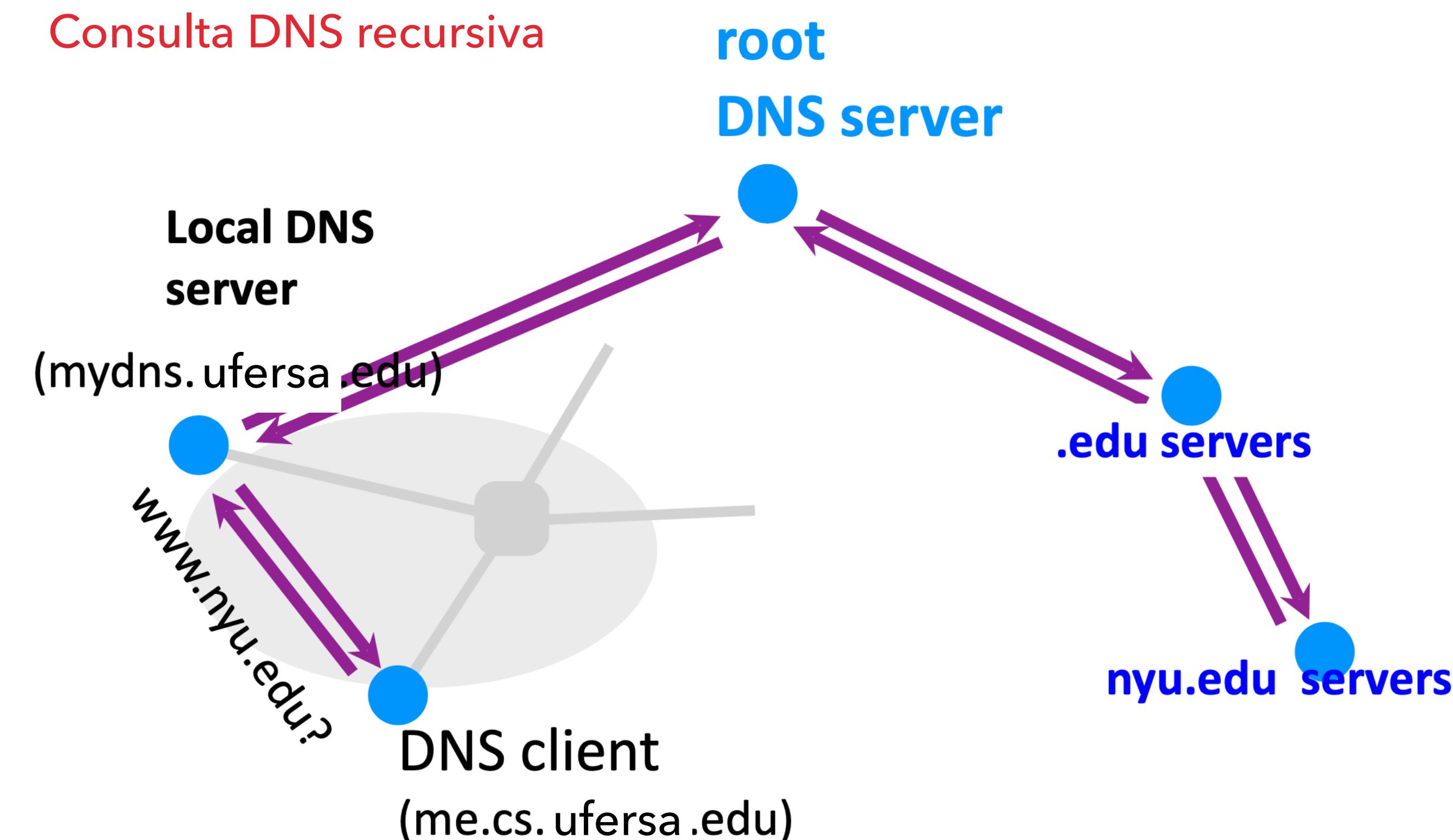
USANDO DNS



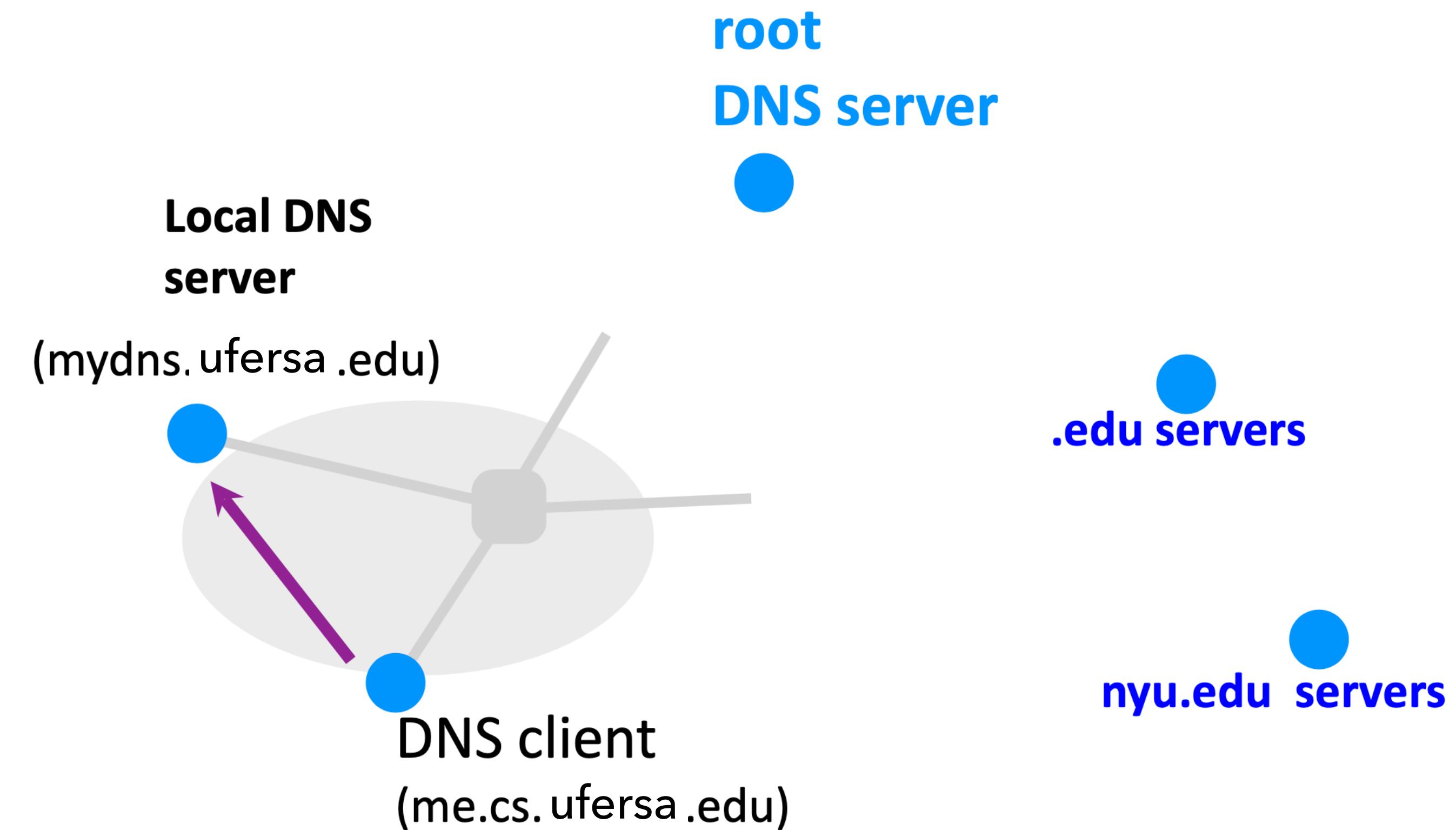
USANDO DNS



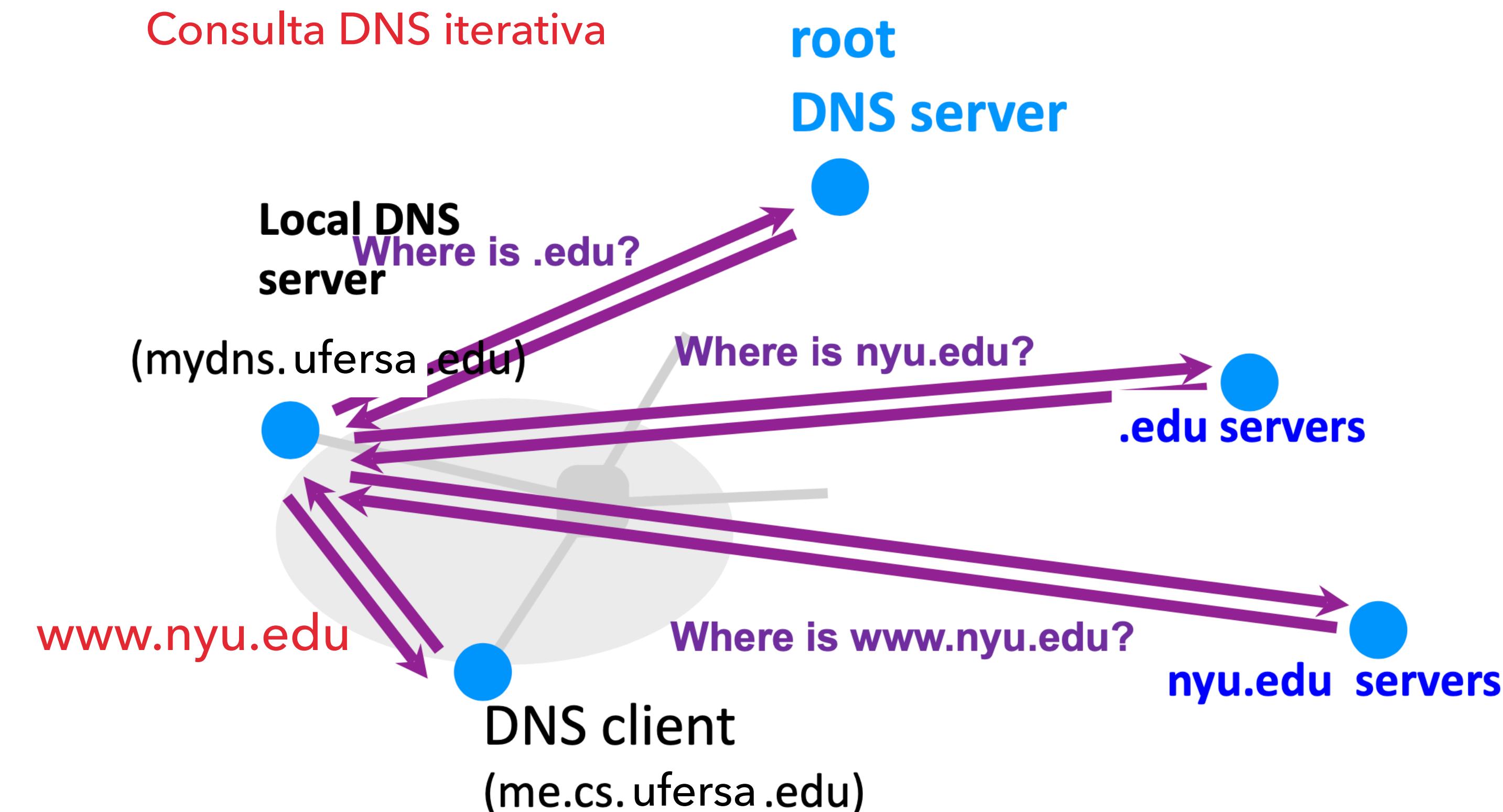
USANDO DNS



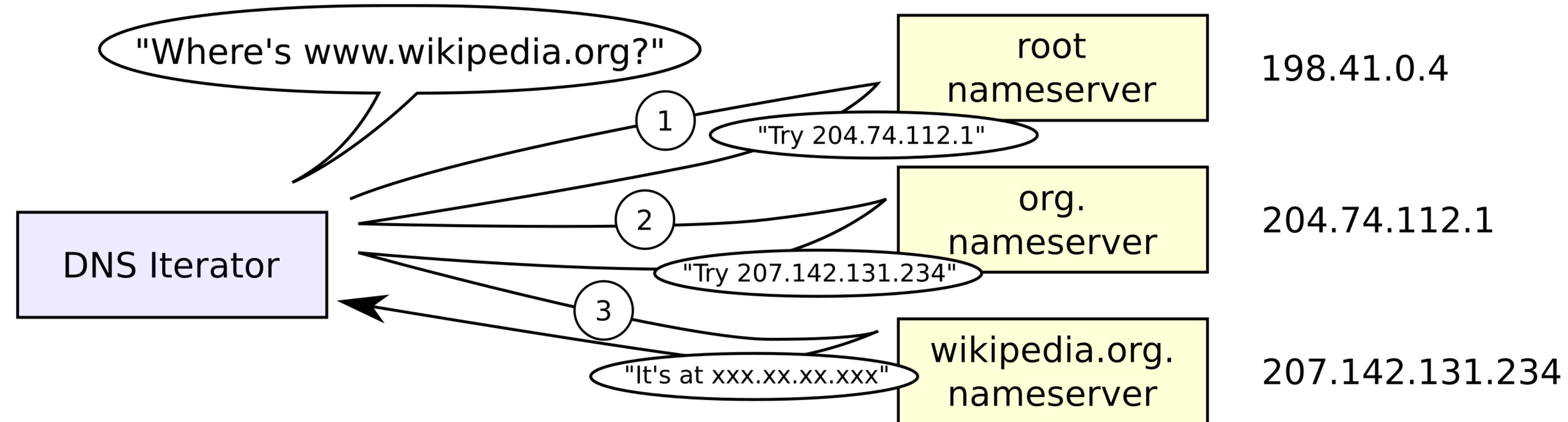
USANDO DNS



USANDO DNS



CONSULTA DNS ITERATIVA (NÃO RECURSIVA)



PROTOCOLO DNS

- ▶ Protocolo no formato **requisição-resposta**
- ▶ Interação cliente-servidor na porta UDP 53
- ▶ Resolução quase sempre é iterativa
 - ▶ Recursiva tem alguns problemas, principalmente na segurança

OBJETIVOS

Objetivos

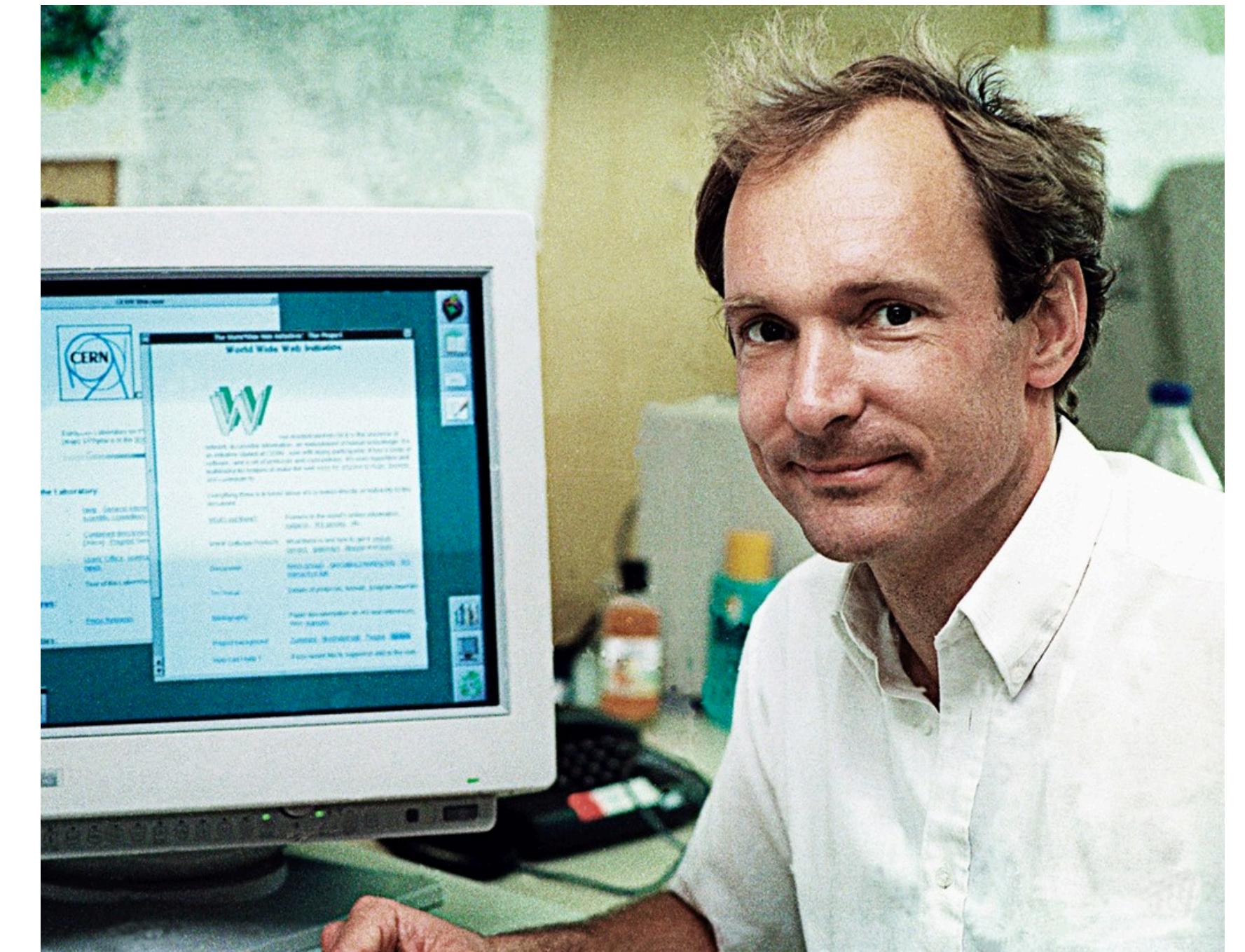
- ▶ Requisitos **OK**
- ▶ Escalável **OK**
- ▶ Fácil gerenciamento **OK**
- ▶ Disponibilidade e consistência **OK**
- ▶ Buscas/consultas rápidas ?

CACHING DNS

- ▶ Como a cache do DNS funciona
 - ▶ Servidores DNS respondem a consultas
 - ▶ Respostas incluem um campo TTL (*time to live*)
 - ▶ Servidores deleta aquela entrada da cache quando o TTL expira
- ▶ Por que caching é efetivo?
 - ▶ Servidores TLDs raramente mudam
 - ▶ Site populares = servidores DNS locais normalmente tem na cache

A WEB: O PRECURSOR

- ▶ Físico tentando resolver um problema real
 - ▶ Acesso distribuído a documentos
- ▶ **World Wide Web (WWW)**: um banco distribuído de "páginas linkadas" através do *Hypertext Transport Protocol (HTTP)*
 - ▶ Primeira implementação - 1990
 - ▶ Tim Berners-Lee no CERN
 - ▶ HTTP/0.9 - 1991
 - ▶ HTTP/1.0 - 1992
 - ▶ HTTP/1.1 - 1996
 - ▶ HTTP/2 - 2015
 - ▶ HTTP/3 (*draft*)



COMPONENTES DA WEB

Infraestrutura

- ▶ Clientes
- ▶ Servidores
- ▶ Proxies

Conteúdo

- ▶ Objetos individuais (arquivos, etc.)
- ▶ Web sites (coleção de objetos)

Implementação

- ▶ URL: conteúdo nominal
- ▶ HTTP: protocolo para troca de conteúdo

SINTAXE DA URL

protocolo://hostname[:porta]/path/recurso

protocolo	http, ftp, https, smtp, rtsp, etc.
hostname	nome de domínio, endereço IP
porta	Normalmente o padrão para o protocolo usado ex: http: 80, https: 443
path	Hierárquico, normalmente reflete o sistema de arquivos
recurso	Identifica o recurso desejado

HTTP

- ▶ Arquitetura **cliente-servidor**
 - ▶ Servidor está "sempre operacional" e "bem conhecido"
 - ▶ Cliente inicia o contato com o servidor
- ▶ Protocolo síncrono baseado em **requisição/resposta**
 - ▶ Executado por cima do protocolo de transporte TCP, porta 80
- ▶ *Stateless*
- ▶ Formato ASCII

REQUISIÇÃO HTTP

- ▶ **Linha de requisição:** método, recurso e versão do protocolo
- ▶ **Cabeçalho da requisição:** provê informações ou modifica requisição
- ▶ **Corpo:** dados opcionais (ex: para no método POST enviar dados ao servidor)

```
GET / HTTP/1.1
Host: imd.ufrn.br
Connection: keep-alive
User-Agent: Chrome/86.0.4240.198
Accept-Language: pt
(linha em branco)
```

RESPOSTA HTTP

- ▶ **Linha de status:** versão do protocolo, código de *status*, mensagem de *status*
- ▶ **Cabeçalho da resposta:** provê informações ou modifica requisição
- ▶ **Corpo:** dados opcionais

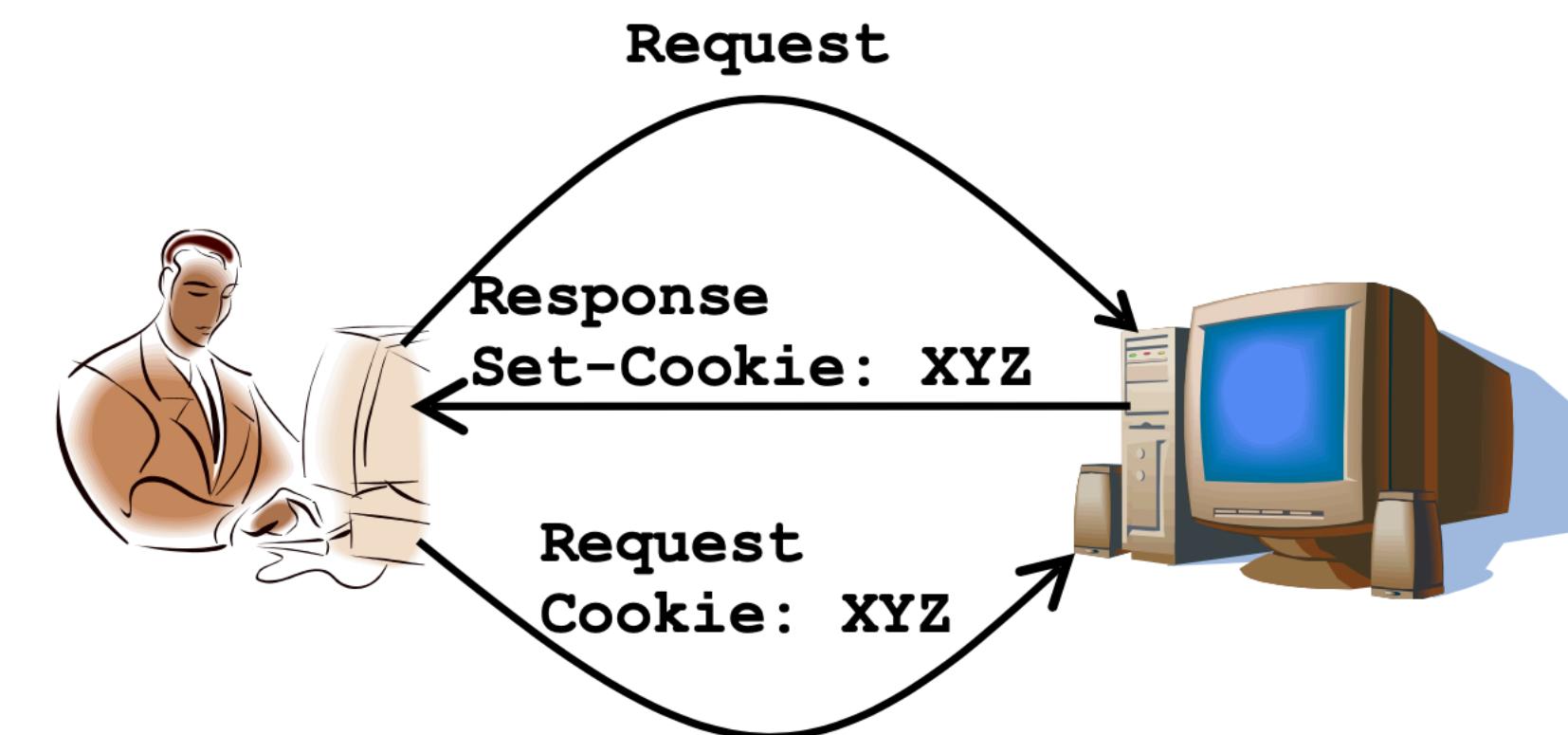
```
HTTP/1.1 200 OK
Date: Tue, 08 Dec 2020 12:07:13 GMT
Server: Apache/1.3.0 (Unix)
Content-Type: text/html
Content-Language: pt-BR
Last-Modified: Tue, 08 Dec 2020
Connection: close
(linha em branco)
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="pt">
<head>
    <meta charset="UTF-8">
    <link rel="icon" href="assets/images/imd-footer.png">
    <script async src="/portal/js/is.js"></script>
```

HTTP É STATELESS

- ▶ Cada requisição-resposta é tratado independentemente
 - ▶ Não é exigido que servidores retenham estado para o HTTP
 - ▶ A aplicação pode ter muitos estados, mas não o HTTP
- ▶ **Vantagens:** melhoria na escalabilidade do lado do servidor
 - ▶ Tratamento de falhas é mais simples
 - ▶ Pode manipular uma taxa maior de requisições
 - ▶ Ordem das requisições não importa (para o HTTP)
- ▶ **Desvantagens:** Algumas aplicações necessitam estados persistentes
 - ▶ Para por exemplo, identificar um usuário ou persistir informação temporária
 - ▶ ex: carrinhos de compras, perfil de usuário, etc.

COOKIES PARA MANTER ESTADO EM PROTOCOLO STATELESS

- ▶ Cliente mantém o estado
 - ▶ O cliente armazena um pequeno estado em nome do servidor
 - ▶ O cliente envia o estado em requisições futuras a esse mesmo servidor
- ▶ Pode prover autenticação



DESEMPENHO DO HTTP

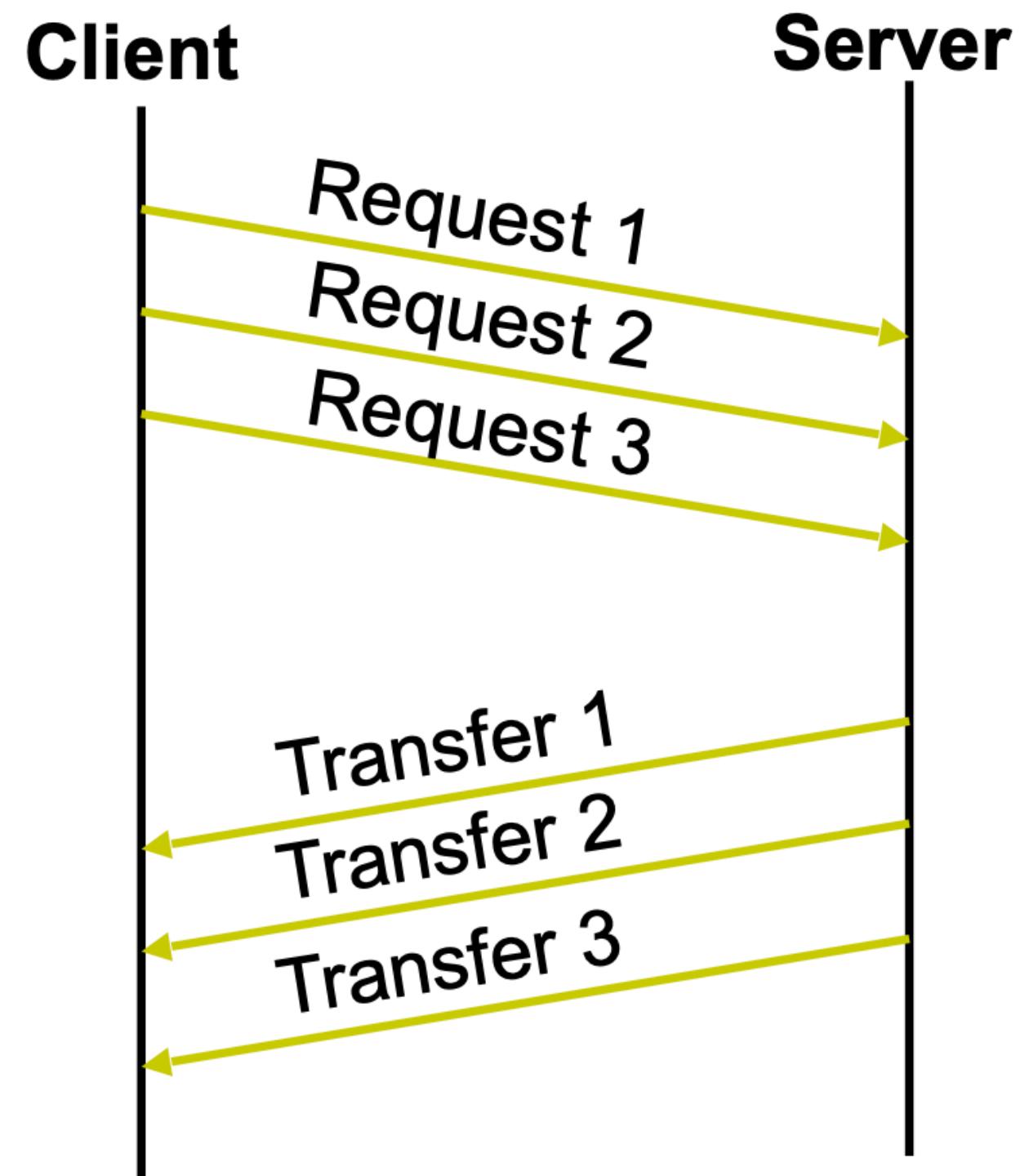
- ▶ Maioria das páginas tem múltiplos objetos
 - ▶ ex: arquivo HTML com várias imagens/scripts/css/etc.
- ▶ Como você requisita esses objetos?
 - ▶ Um de cada vez?
- ▶ Uma nova conexão para cada objeto!

MELHORANDO O DESEMPENHO DO HTTP

- ▶ Conexões persistentes
- ▶ Mantém a conexão para múltiplas requisições
- ▶ Vantagens no desempenho:
 - ▶ Evita o overhead do estabelecimento e encerramento da conexão
- ▶ Padrão no HTTP/1.1

MELHORANDO O DESEMPENHO DO HTTP

- ▶ Requisições e respostas em *pipeline*
- ▶ Lote de requisições e respostas para reduzir o número de pacotes



MELHORANDO O DESEMPENHO DO HTTP: CACHING

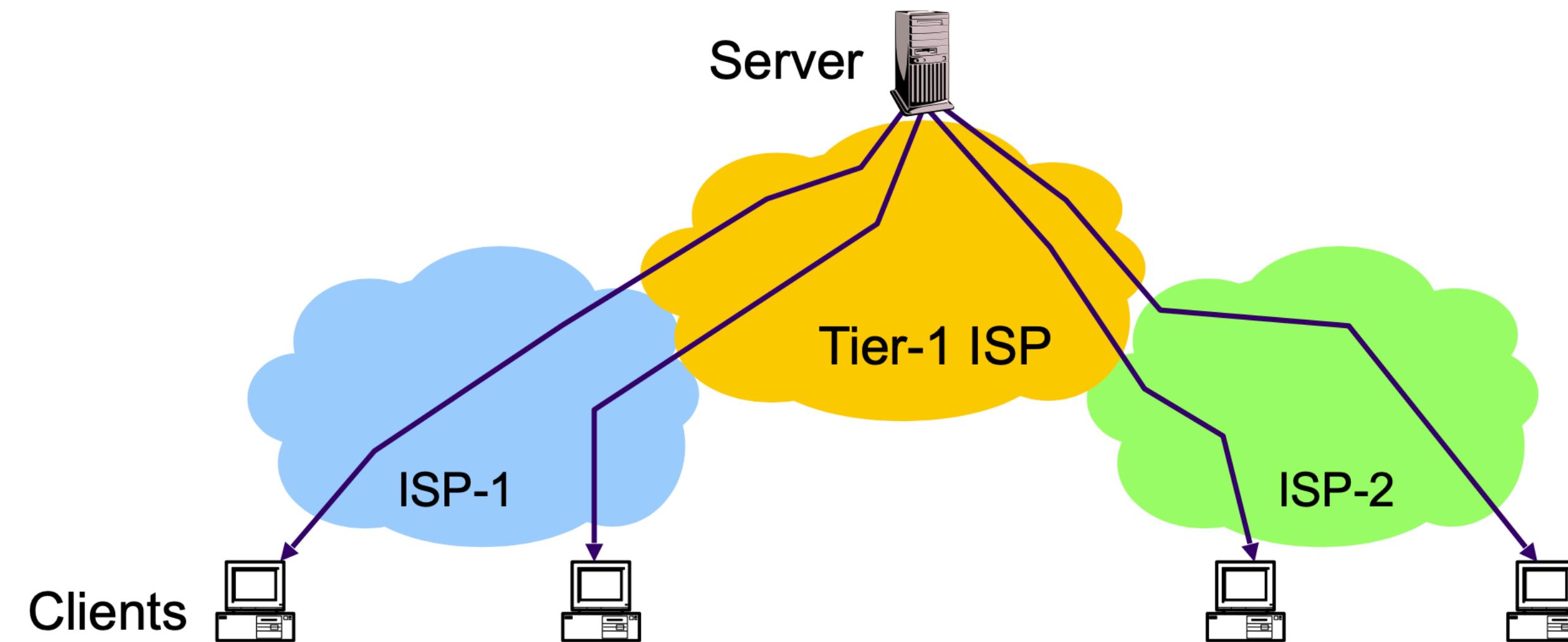
- ▶ Modificador na requisição GET
 - ▶ **If-modified-since** : retorna “*not modified*” se o recurso não foi modificado desde a data especificada
- ▶ Cabeçalho de resposta
 - ▶ **Expires** : TTL; quanto tempo é seguro cachejar o recurso
 - ▶ **No-cache** : ignora qualquer cache, sempre pega do servidor

MELHORANDO O DESEMPENHO DO HTTP: CACHING

- ▶ Cliente requisita objeto
- ▶ Se está na cache local do cliente:
 - ▶ Se está dentro do TTL, responde a cliente
 - ▶ Se não está dentro do TTL, envia *If-modified-since* para servidor
 - ▶ Se o servidor tem uma cópia atualizada, envia;
 - ▶ Se não, servidor responde que não tem mudanças
- ▶ Se não está na cache do cliente
 - ▶ Envia requisição para o servidor
- ▶ Obs: a cache pode ocorrer em vários outros níveis!

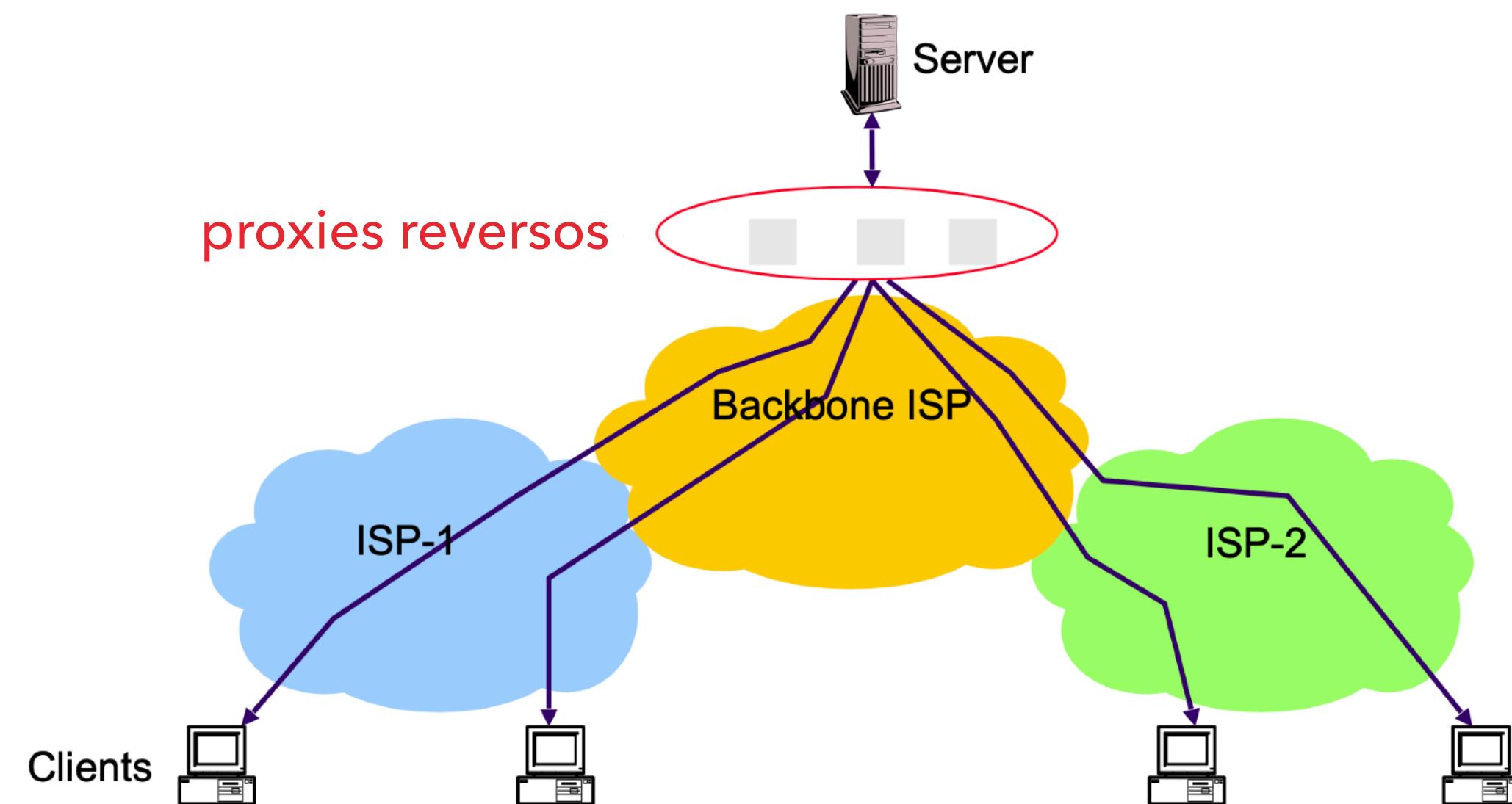
MELHORANDO O DESEMPENHO DO HTTP: ONDE FAZER CACHING?

- ▶ Muitos clientes requisitam a mesma informação
 - ▶ Geram carga desnecessária no servidor e na rede
 - ▶ Clientes experimentam latência desnecessária



CACHING COM PROXY REVERSO

- ▶ Cache de documentos perto do servidor
 - ▶ Diminui a carga do servidor
- ▶ Tipicamente feito por provedores de conteúdo



CACHING COM PROXY DIRETO

- ▶ Cache de documentos perto dos clientes
- ▶ Diminui o tráfego de rede e diminui latência
- ▶ Tipicamente feito por ISPs

