Warstwa aplikacji

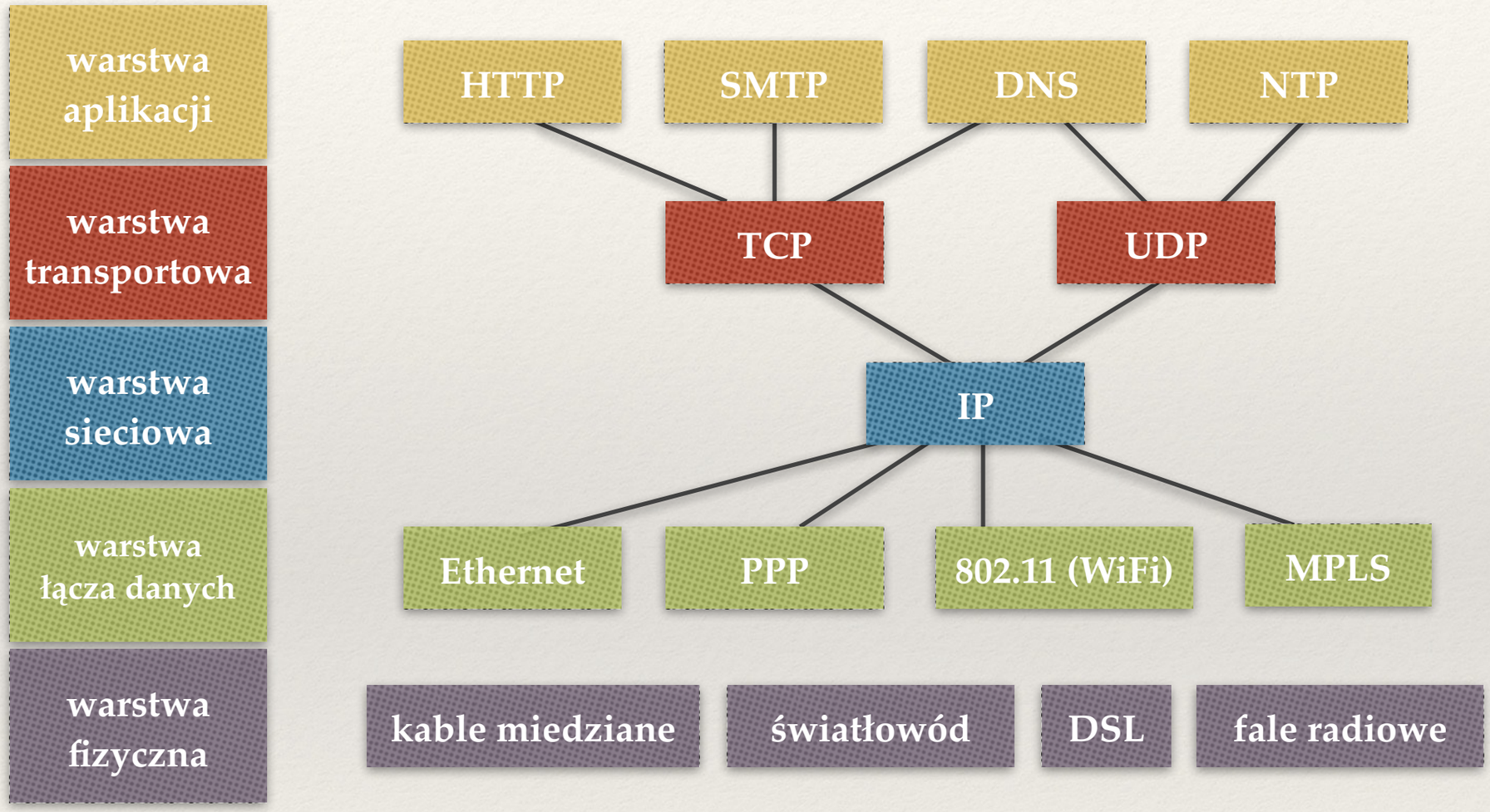
część 1

Sieci komputerowe

Wykład 9

Marcin Bieńkowski

Protokoły w Internecie



Dwa popularne zastosowania

- **DNS (*Domain Name System*)**
  - Zamienia nazwy symboliczne na adresy IP i z powrotem.
- **HTTP (*Hypertext Transfer Protocol*)**
  - Przesyłanie danych w architekturze klient-serwer.

DNS

Nazwy symboliczne a adresy IP

- Większości ludzi łatwiej zapamiętać jest nazwę symboliczną
  - `www.ii.uni.wroc.pl` → `156.17.4.11`
  - `atm-wro-pb1-wro-br1.devs.futuro.pl` → `62.233.154.25`.
- Nazwa może pozostać taka sama pomimo przeniesienia serwisu (np. strony WWW) pod inny adres IP.

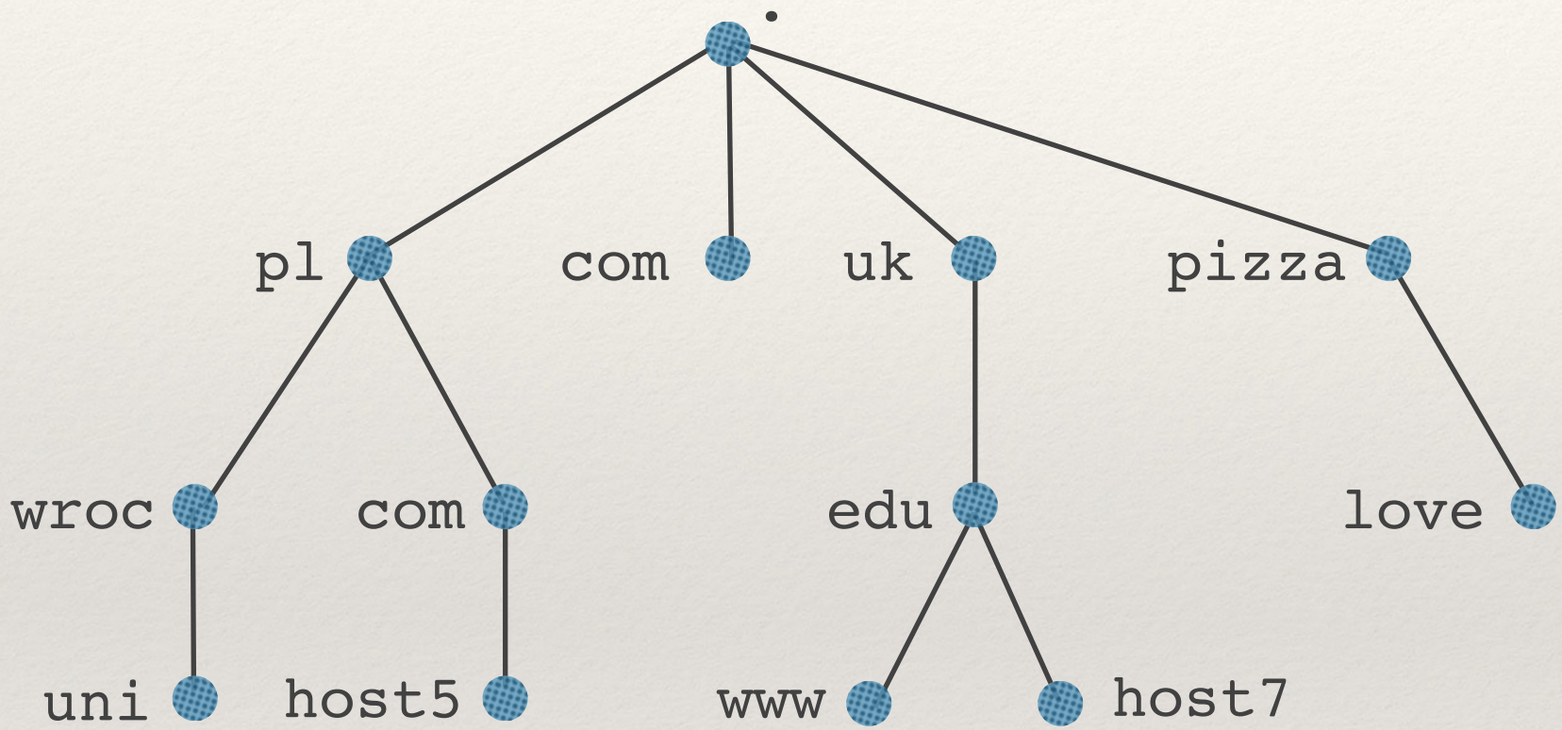
/etc/hosts

- Można takie odwzorowanie zapisać lokalnie (plik `/etc/hosts`).
- W początkach Internetu:
  - Pojedynczy i centralnie przechowywany plik HOSTS.TXT.
  - Każdy mógł go pobrać i zapisać do pliku `/etc/hosts`.
  - Aktualizacje HOSTS.TXT przez email do administratora.
  - Problemy z koordynacją, aktualizacją, dostępem, skalowalnością.

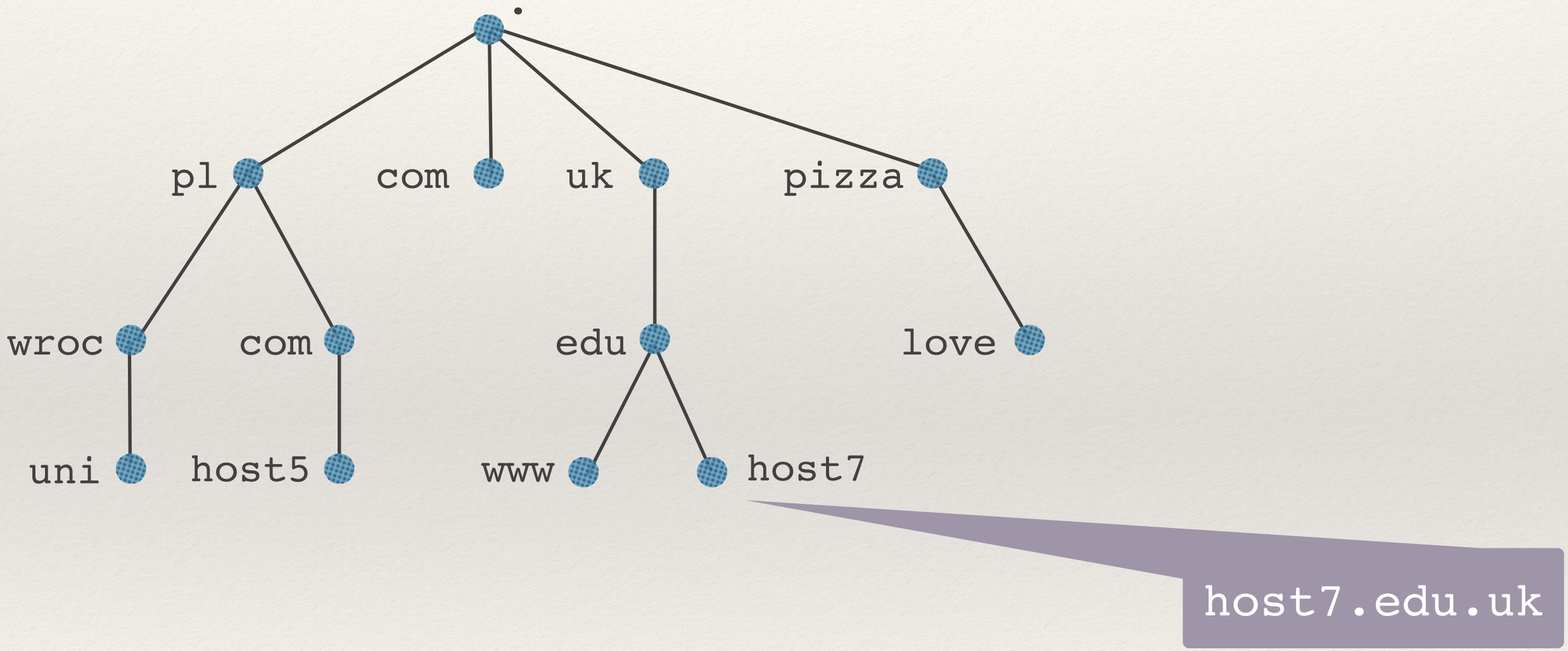
Cele DNS

- Przekształcanie nazw na adresy (lub ogólnie: na inne informacje).
- Obsługiwanie dużej liczby rekordów (ok. 300 mln nazw, nie licząc poddomen).
- Rozproszone zarządzanie.
- Odporne na błędy pojedynczych serwerów.

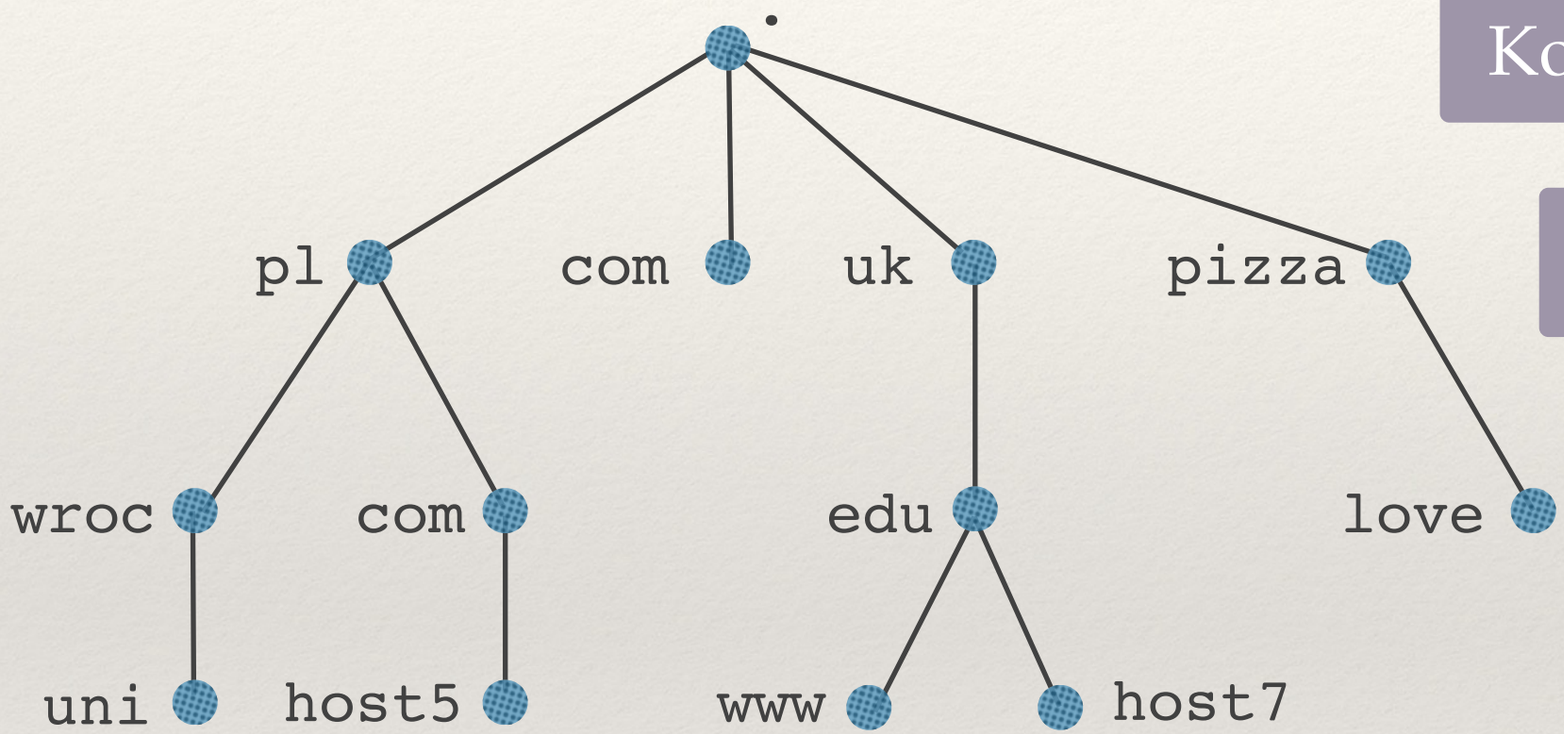
Hierarchia nazw domen



Hierarchia nazw domen



Hierarchia nazw domen

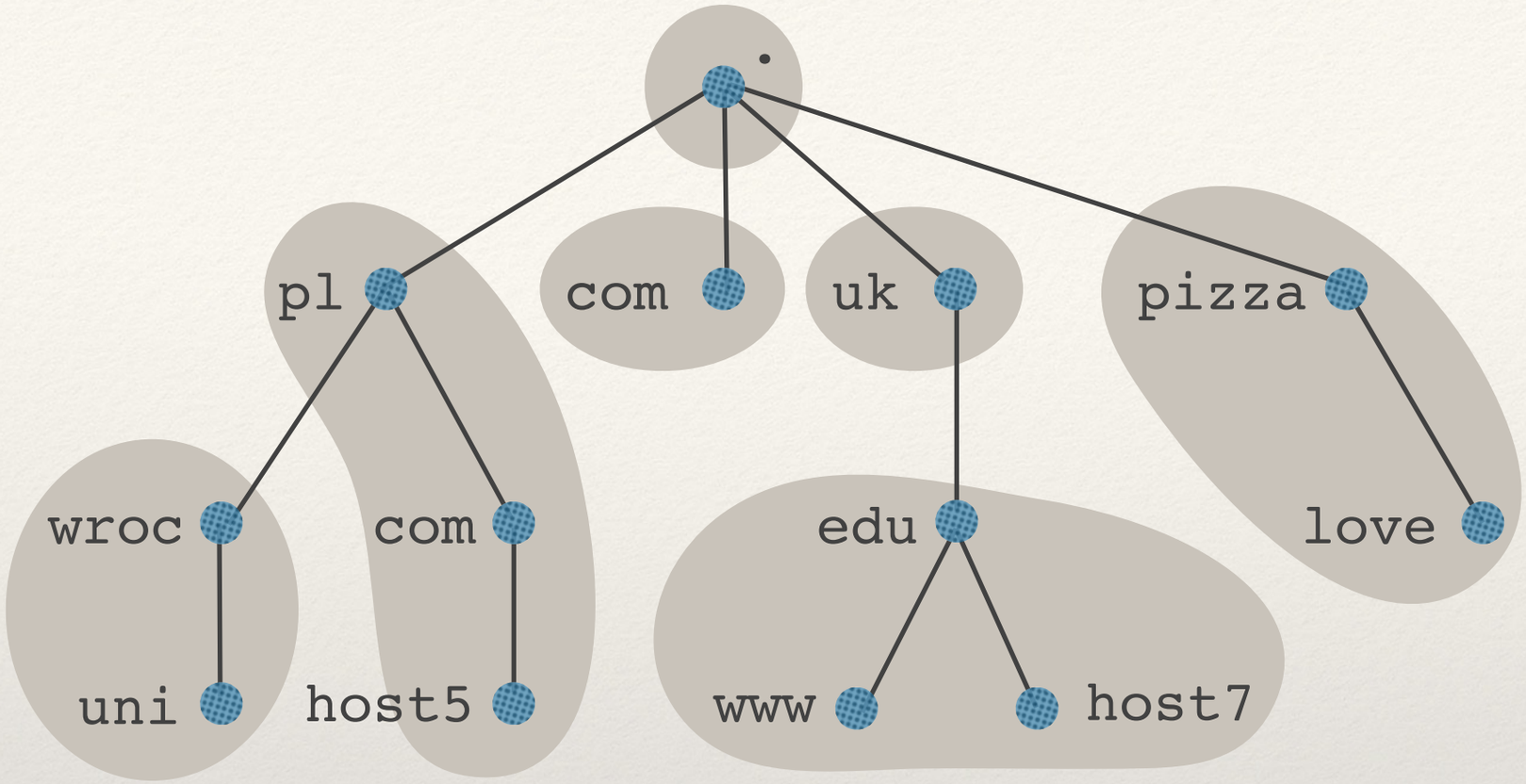


Korzeń oznaczamy kropką

TLD (top level domains)

host7.edu.uk

Rozproszone zarządzanie: strefy



Strefa

- Spójny fragment drzewa
- Najmniejsza jednostka administracyjną DNS, odrębnie zarządzana.
- Właściciel strefy = serwer(y) DNS (zazwyczaj 2-5), wie wszystko o nazwach domen w strefie.

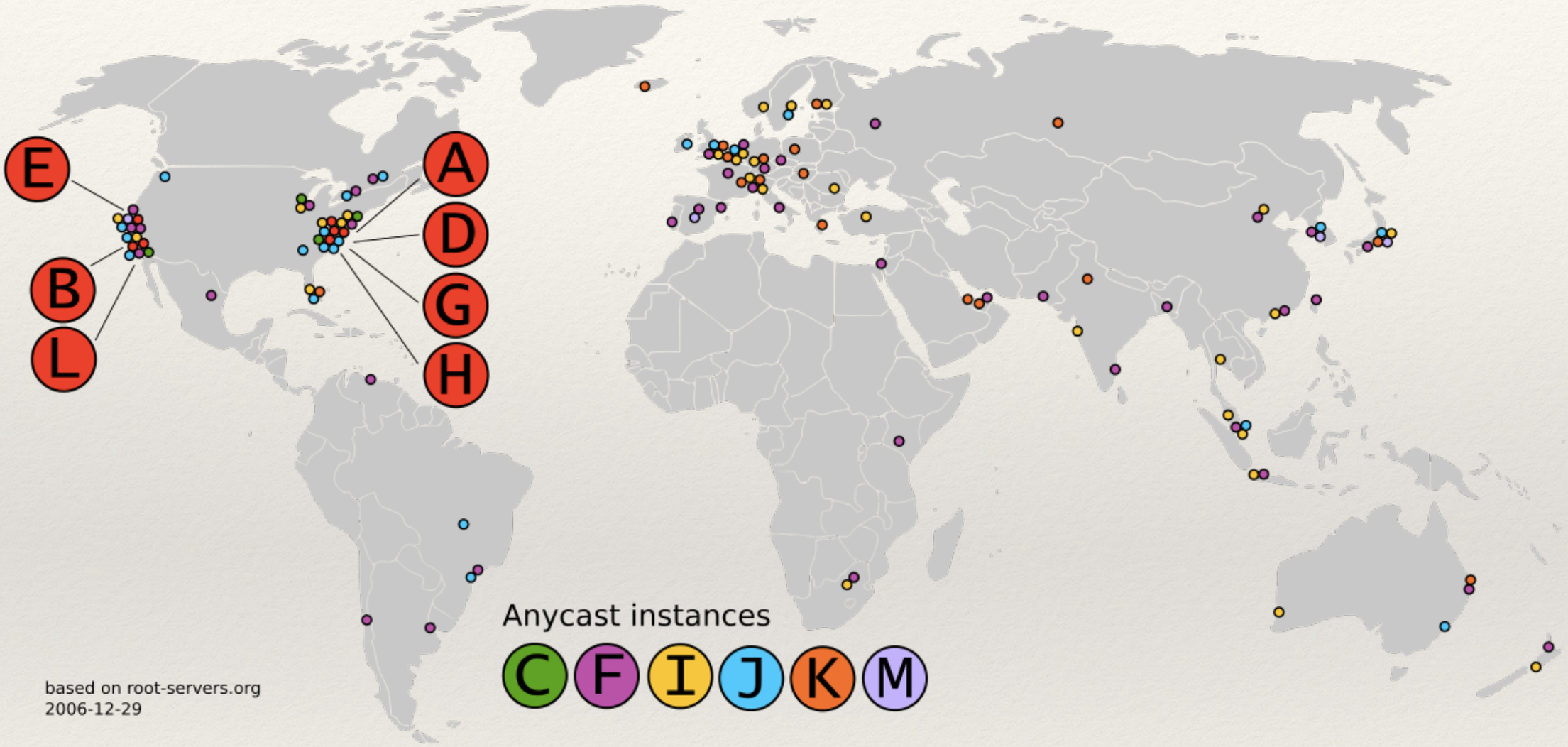
Serwery główne (1)

13 serwerów *głównych* dla strefy „.“

```
A.ROOT-SERVERS.NET =  198.41.0.4

B.ROOT-SERVERS.NET = 192.228.79.201

C.ROOT-SERVERS.NET = 192.33.4.12

D.ROOT-SERVERS.NET = 128.8.10.90

...
```

Informacja wpisywana ręcznie
(w standardowych plikach konfiguracyjnych).

Serwery główne (2)



Obrazek ze strony https://en.wikipedia.org/wiki/Root_name_server

Anycast

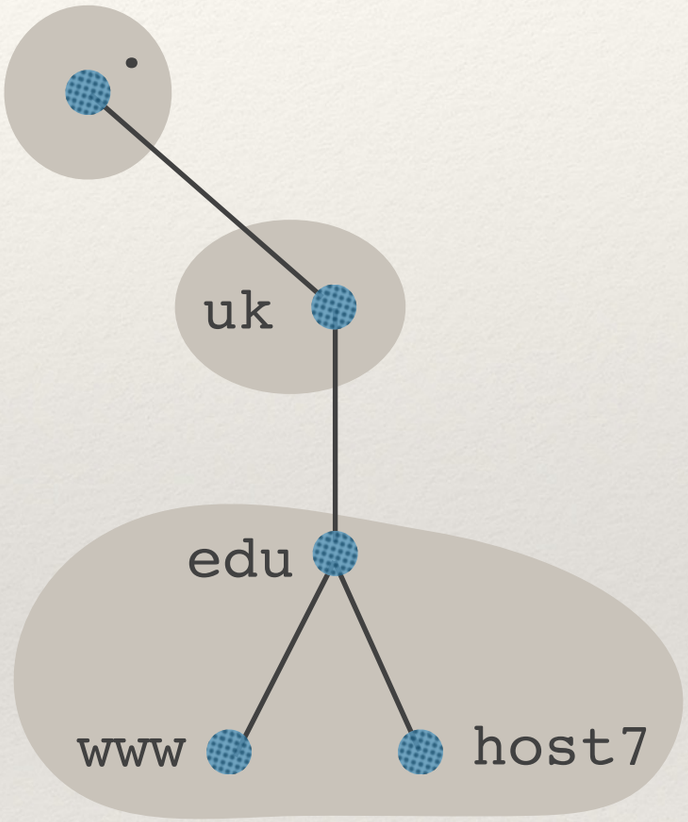
- **Adres anycast** = wiele serwerów ma ten sam adres IP
- Rozpowszechniany za pomocą standardowych protokołów routingu → routery poznają trasę do najbliższego serwera z danym adresem.

Anycast

- **Adres anycast** = wiele serwerów ma ten sam adres IP
- Rozpowszechniany za pomocą standardowych protokołów routingu → routery poznają trasę do najbliższego serwera z danym adresem.
- Problem: wszystkie pakiety z danej komunikacji powinny być do jednego serwera.
  - najbliższy serwer może zmienić się w trakcie
  - DNS nie ma problemu: komunikacja = jeden pakiet z zapytaniem

Rozszyfrowywanie nazw (resolving)

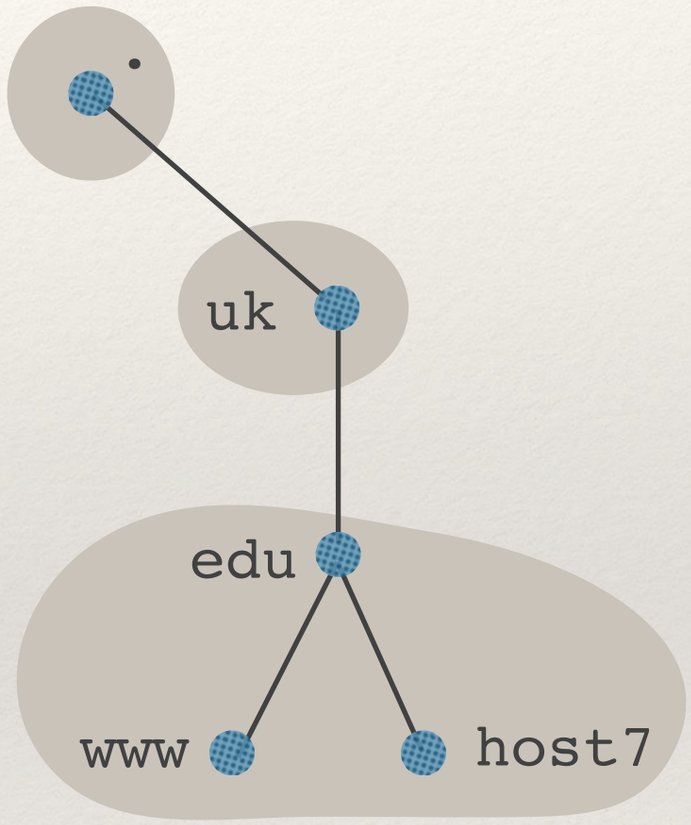
Chcemy poznać adres IP dla `host7.edu.uk`.



Rozszyfrowywanie nazw (resolving)

Chcemy poznać adres IP dla `host7.edu.uk`.

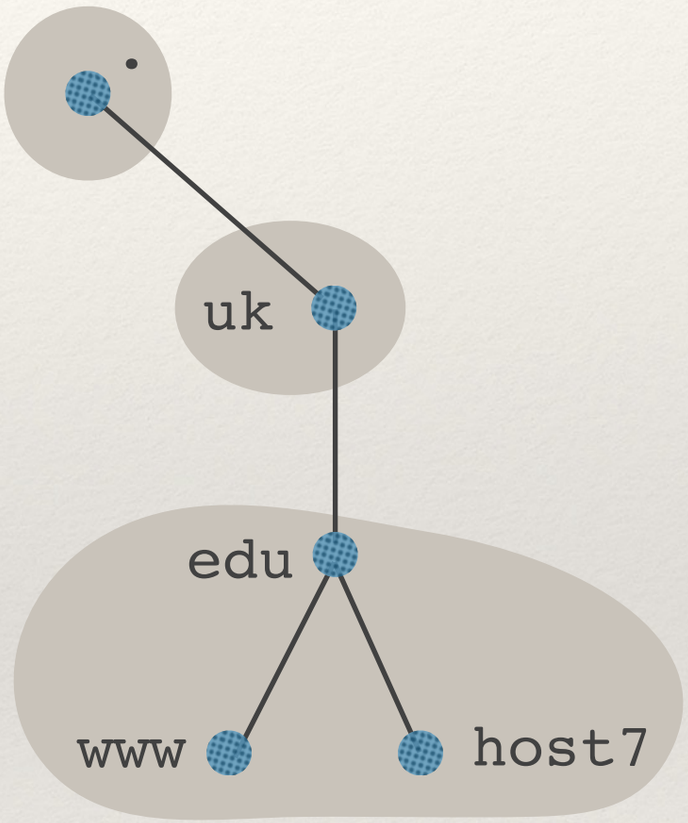
- Pytamy jeden z serwerów nazw dla "`.`",
  np. `E.ROOT-SERVERS.NET` o adresie
  `192.203.230.10`.



Rozszyfrowywanie nazw (resolving)

Chcemy poznać adres IP dla `host7.edu.uk`.

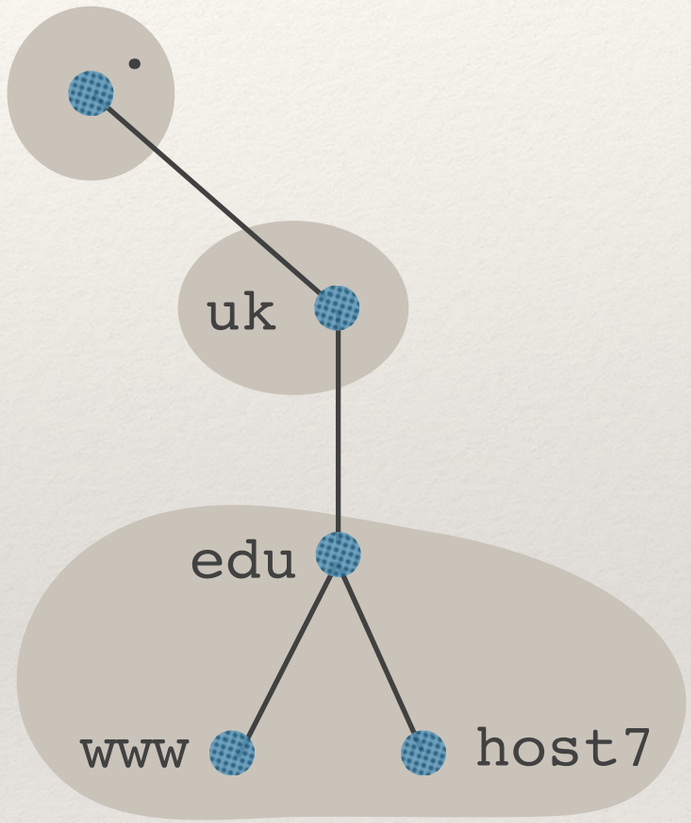
- Pytamy jeden z serwerów nazw dla "`.`", np. `E.ROOT-SERVERS.NET` o adresie `192.203.230.10`.
- Serwer nie zna, ale mówi, że serwerem nazw dla `uk` jest `foo.uk` o adresie `1.2.3.4`.



Rozszyfrowywanie nazw (resolving)

Chcemy poznać adres IP dla `host7.edu.uk`.

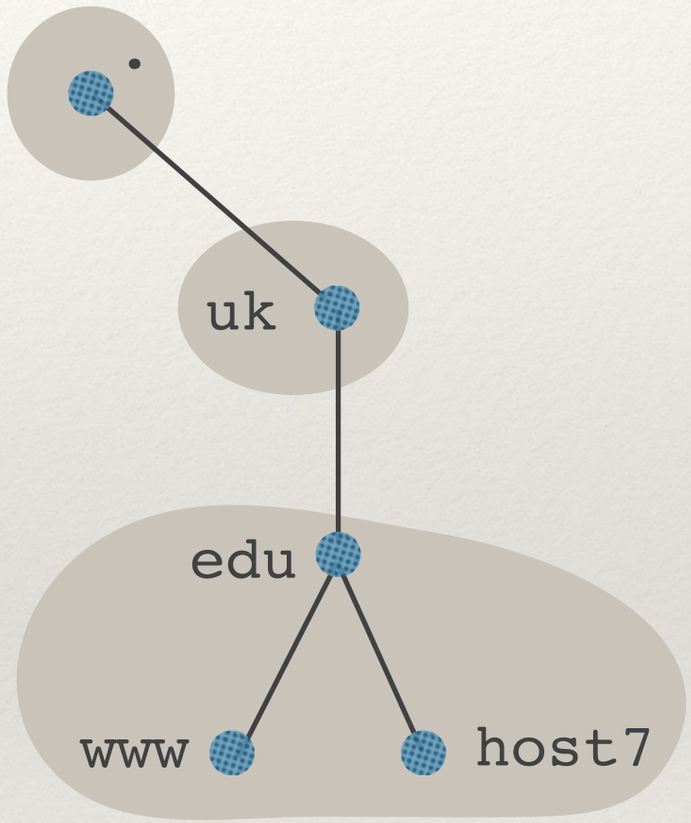
- Pytamy jeden z serwerów nazw dla "`.`", np. `E.ROOT-SERVERS.NET` o adresie `192.203.230.10`.
- Serwer nie zna, ale mówi, że serwerem nazw dla `uk` jest `foo.uk` o adresie `1.2.3.4`.
- Pytamy `foo.uk`.



Rozszyfrowywanie nazw (resolving)

Chcemy poznać adres IP dla `host7.edu.uk`.

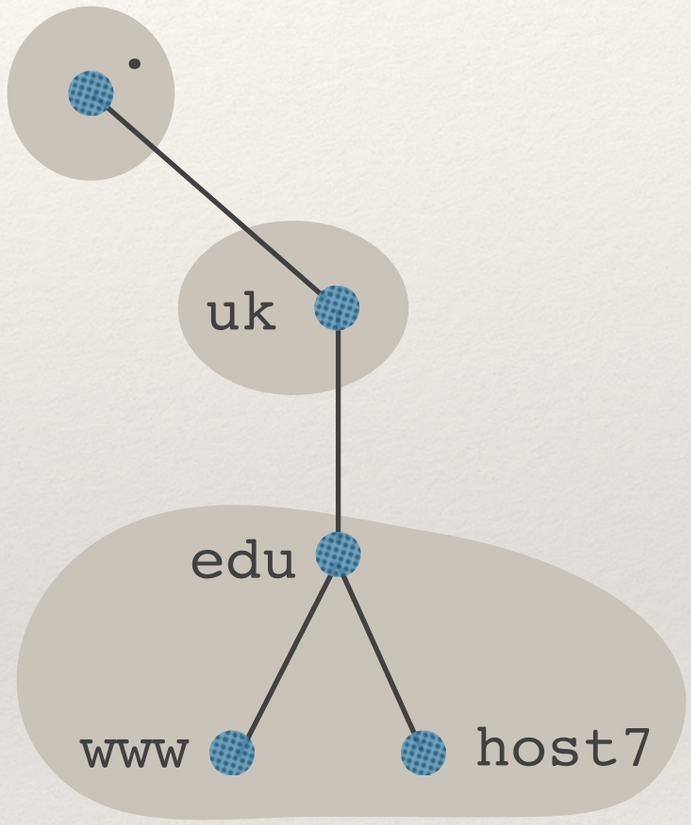
- Pytamy jeden z serwerów nazw dla "`.`", np. `E.ROOT-SERVERS.NET` o adresie `192.203.230.10`.
- Serwer nie zna, ale mówi, że serwerem nazw dla `uk` jest `foo.uk` o adresie `1.2.3.4`.
- Pytamy `foo.uk`.
- Serwer nie zna, ale mówi, że serwerem nazw dla `edu.uk` jest `foo.bar.uk` o adresie `5.6.7.8`.



Rozszyfrowywanie nazw (resolving)

Chcemy poznać adres IP dla `host7.edu.uk`.

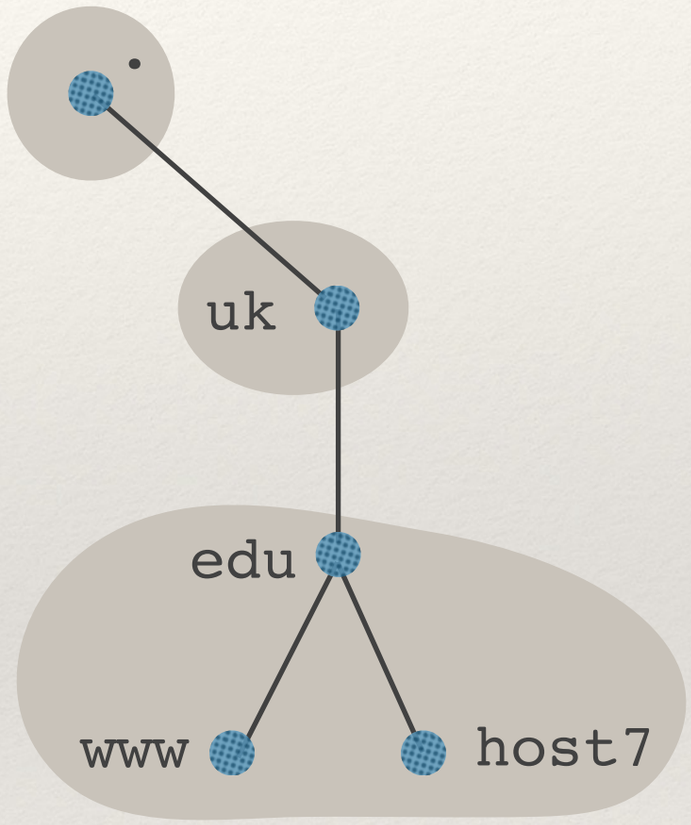
- Pytamy jeden z serwerów nazw dla "`.`", np. `E.ROOT-SERVERS.NET` o adresie `192.203.230.10`.
- Serwer nie zna, ale mówi, że serwerem nazw dla `uk` jest `foo.uk` o adresie `1.2.3.4`.
- Pytamy `foo.uk`.
- Serwer nie zna, ale mówi, że serwerem nazw dla `edu.uk` jest `foo.bar.uk` o adresie `5.6.7.8`.
- Pytamy `foo.bar.uk`.



Rozszyfrowywanie nazw (resolving)

Chcemy poznać adres IP dla `host7.edu.uk`.

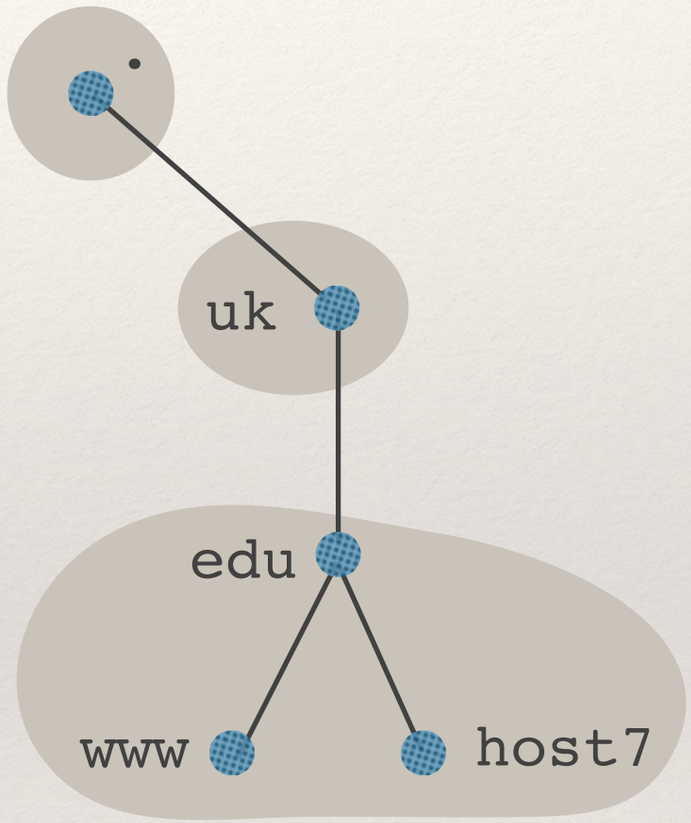
- Pytamy jeden z serwerów nazw dla "`.`",
  np. `E.ROOT-SERVERS.NET` o adresie
  `192.203.230.10`.
- Serwer nie zna, ale mówi, że serwerem nazw dla
  `uk` jest `foo.uk` o adresie `1.2.3.4`.
- Pytamy `foo.uk`.
- Serwer nie zna, ale mówi, że serwerem nazw dla
  `edu.uk` jest `foo.bar.uk` o adresie `5.6.7.8`.
- Pytamy `foo.bar.uk`.
- Serwer `foo.bar.uk` odpowiada adresem IP,
  bo jest serwerem nazw dla strefy zawierającej
  `host7.edu.uk`.



Rozszyfrowywanie nazw (resolving)

Chcemy poznać adres IP dla `host7.edu.uk`.

- Pytamy jeden z serwerów nazw dla "`.`",
  np. `E.ROOT-SERVERS.NET` o adresie
  `192.203.230.10`.
- Serwer nie zna, ale mówi, że serwerem nazw dla
  `uk` jest `foo.uk` o adresie `1.2.3.4`.
- Pytamy `foo.uk`.
- Serwer nie zna, ale mówi, że serwerem nazw dla
  `edu.uk` jest `foo.bar.uk` o adresie `5.6.7.8`.
- Pytamy `foo.bar.uk`.
- Serwer `foo.bar.uk` odpowiada adresem IP,
  bo jest serwerem nazw dla strefy zawierającej
  `host7.edu.uk`.



demonstracja

Rozszyfrowywanie iteracyjne i rekurencyjne

- **Rozszyfrowywanie iteracyjne** = klient przechodzi drzewo DNS zaczynając od korzenia (jak na poprzednim slajdzie).

- **Rozszyfrowywanie rekurencyjne** = pytamy resolver DNS, a on w naszym imieniu wykonuje odpytywanie.

- **Resolver DNS** = to co wpisujemy w polu „serwer DNS“ w konfiguracji sieci naszego komputera.
  - Dla poprawy wydajności, zapisuje zwracane wyniki w pamięci podręcznej.
  - Może być też serwerem DNS (odpowiedzialnym za jakąś strefę).

Rekordy A i AAAA

Rekord DNS = (typ, nazwa, wartość)

Typ A (*address*)

- nazwa = nazwa domeny (`google.pl`)
- wartość = adres IPv4 (`216.58.209.67`)

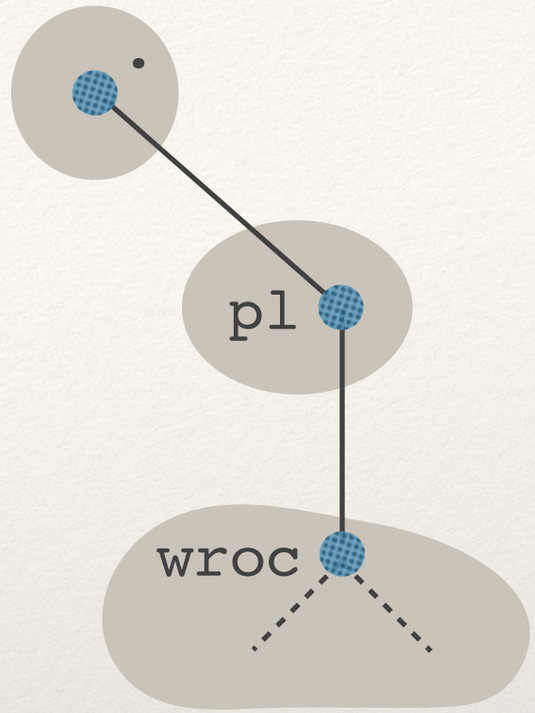
Typ AAAA

- nazwa = nazwa domeny (`google.pl`)
- wartość = adres IPv6 (`2a00:1450:401b:801::2003`)

Rekordy NS

Typ NS (*nameserver*)

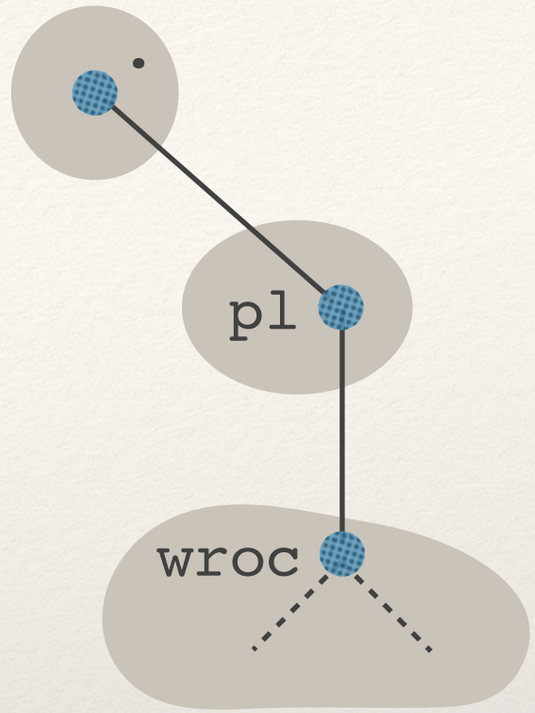
- nazwa = nazwa strefy (`wroc.pl`)
- wartość = nazwa serwera DNS obsługującego daną strefę (`sun2.pwr.wroc.pl`)



Rekordy NS

Typ NS (*nameserver*)

- nazwa = nazwa strefy (`wroc.pl`)
- wartość = nazwa serwera DNS obsługującego daną strefę (`sun2.pwr.wroc.pl`)

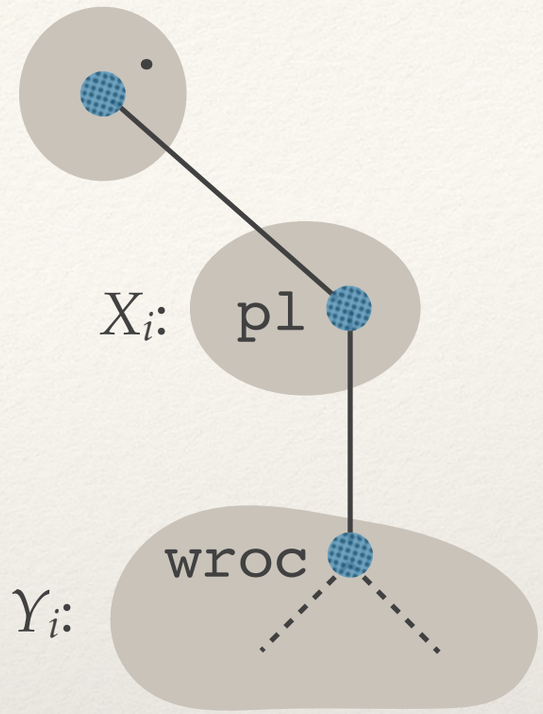


Kto powinien pamiętać rekord NS dla strefy `wroc.pl`?

Rekordy NS

Typ NS (*nameserver*)

- nazwa = nazwa strefy (`wroc.pl`)
- wartość = nazwa serwera DNS obsługującego daną strefę (`sun2.pwr.wroc.pl`)



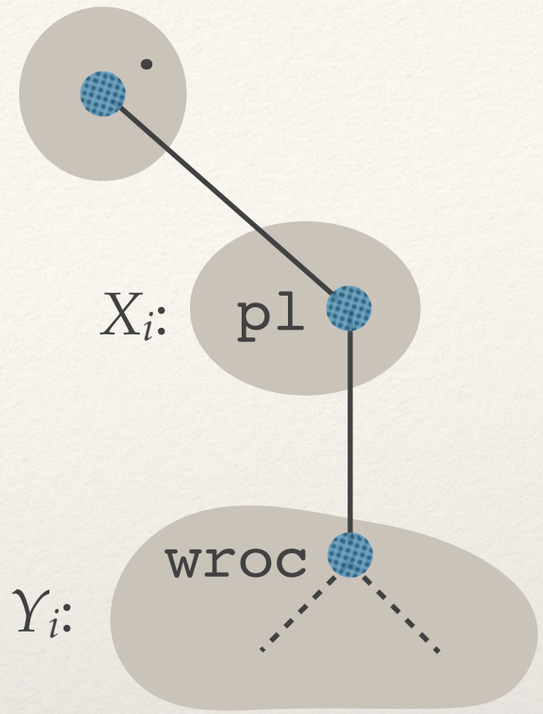
Kto powinien pamiętać rekord NS dla strefy `wroc.pl`?

- Niech X_i = serwery nazw dla strefy `pl`
- Niech Y_i = serwery nazw dla strefy `wroc.pl` (między innymi `sun2.pwr.wroc.pl`)

Rekordy NS

Typ NS (*nameserver*)

- nazwa = nazwa strefy (`wroc.pl`)
- wartość = nazwa serwera DNS obsługującego daną strefę (`sun2.pwr.wroc.pl`)



Kto powinien pamiętać rekord NS dla strefy `wroc.pl`?

- Niech X_i = serwery nazw dla strefy `pl`
- Niech Y_i = serwery nazw dla strefy `wroc.pl` (między innymi `sun2.pwr.wroc.pl`)
- Wpis „(NS, `wroc.pl` → `sun2.pwr.wroc.pl`)“ zazwyczaj przechowywany:
  - na serwerach Y_i (zbędne z punktu widzenia odpytywania),
  - na serwerach X_i („delegacje“, podczas odpytywania od góry drzewa DNS wiemy kogo pytać następnego).
  - Dodatkowo serwery X_i zazwyczaj znają również odpowiednie adresy IP: (A, `sun2.pwr.wroc.pl` → `156.17.5.2`).

Dodatkowe rekordy DNS

Typ CNAME (*canonical name*)

- nazwa = alias nazwa domeny (`www.ii.uni.wroc.pl`)
- wartość = „główna“ nazwa domeny (`swiatowit.ii.uni.wroc.pl`)

Typ MX (*mail exchanger*)

Szczegóły za tydzień.

- nazwa = nazwa domeny (`gmail.com`)
- wartość = nazwa serwera obsługującego pocztę (`gmail-smtp-in.l.google.com`)

Domena odwrotna

- **Odwrotna konwersja: adres IP → nazwa domeny.**
  - Wykorzystuje typ rekordu PTR.
  - Sztuczna domena `in-addr.arpa`, której poddomenami są klasy lub adresy IP.

- **Przykładowo:**
  - strefa `33.22.11.in-addr.arpa` zawiera informacje na temat sieci `11.22.33.0/24`
  - w szczególności zawiera wpis
    PTR `44.33.22.11.in-addr.arpa` → nazwa.domena.org

Pamięć podręczna DNS

Rekordy DNS mają czas życia (TTL)

- Po tym czasie powinny być wyrzucane z pamięci podręcznej serwerów / resolverów DNS.
- Duży TTL → zmniejsza liczbę zapytań do serwerów DNS.
- Mały TTL → szybsza propagacja zmian.

Negatywna pamięć podręczna

- Zapamiętujemy też fakt, że dana domena nie istnieje.

DNS = dodatkowa warstwa abstrakcji

- Łatwa wymienialność adresów IP przy zachowaniu nazw domen.
  - Niewidoczne dla ludzi i aplikacji.

- Wiele adresów IP dla tej samej nazwy (rekordy A).
  - Możliwość równoważenia obciążenia serwerów.
  - Możliwość zwracania „bliskiego“ serwera.

- Wiele nazw dla tego samego adresu (rekordy CNAME).
  - Wiele usług na tym samym serwerze (`www.domena.pl`, `ftp.domena.pl`, `mail.domena.pl`).

Odpytywanie DNS w programie (1)

```
int getaddrinfo(const char *domain, const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

```
struct addrinfo {
  int ai_family;
  int ai_socktype;
  …
  struct sockaddr *ai_addr;
  struct addrinfo *ai_next;
};
```

zazwyczaj `AF_INET` albo `AF_INET6`

`SOCK_STREAM`, `SOCK_DGRAM`, …

result jest listą struktur `addrinfo`

Najprostszy przypadek użycia:

```
struct addrinfo* result;
int getaddrinfo("www.example.com", NULL, NULL, &result);
```

Odpytywanie DNS w programie (2)

```
int main(int argc, char* argv[])
{
    struct addrinfo* result;
    struct addrinfo hints = {
        .ai_family = AF_INET,
        .ai_socktype = SOCK_STREAM,
    };
    getaddrinfo (argv[1], NULL, &hints, &result);

    for (struct addrinfo* r = result; r != NULL; r = r->ai_next) {
        struct sockaddr_in* addr = (struct sockaddr_in*)(r->ai_addr);
        char ip_address[20];
        inet_ntop (AF_INET, &(addr->sin_addr), ip_address,
            sizeof(ip_address));
        printf ("%s\n", ip_address);
    }
}
```

Brak obsługi błędów, plików nagłówkowych, etc.

chcemy tylko takie informacje

demonstracja

kod programu na stronie wykładu

HTTP

HTTP

- Zaprojektowany do przesyłania hipertekstu (tekst z odnośnikami).

- Obecnie: również do przesyłania przesyłania olbrzymich danych, streamingu video (Youtube, Netflix), …

- Korzysta z protokołu TCP, portu 80 (szyfrowana wersja: port 443).

URL (*Uniform Resource Locator*)

- Indentyfikuje dany zasób
- Składa się z 2 części rozdzielonych dwukropkiem:
  - schemat: (`http`, `https`, `ftp`, `mailto`, ...)
  - część zależna od rodzaju zasobu.

- Przykłady:
  - `http://www.ii.uni.wroc.pl/index.html`
  - `https://pl.wikipedia.org/wiki/URL`
  - `mailto:jan.kowalski@serwer.com`

URL dla schematu http lub https

- Po dwukropku:
  - `//`
  - nazwa serwera WWW
  - opcjonalnie `:port`
  - `/`
    - identyfikator zasobu wewnątrz serwera
      - niekoniecznie ścieżka do pliku,
      - `/` w identyfikatorze wskazuje na hierarchię.

- Przykład: `https://canvas.ii.uni.wroc.pl:443/courses/10`

Pobieranie strony WWW krok po kroku (1)

- Przeglądarka WWW dostaje URL
- URL jest rozbijany na człony (zakładamy, że schemat = `http`).
- Nawiązuje połączenie TCP z portem 80 serwera WWW.
- Wysyła żądanie HTTP:

```
GET /courses/10 HTTP/1.1
Host: canvas.ii.uni.wroc.pl
Accept: text/html;q=0.9,application/xml;q=0.8
Accept-Language: en-US,en;q=0.8,pl;q=0.6,de;q=0.4
User-Agent: Mozilla/5.0 … Chrome/49.0.2623.112
```

Pobieranie strony WWW krok po kroku (2)

- Serwer analizuje żądanie, pobiera z dysku odpowiedni plik.

- Serwer sprawdza typ MIME pliku (heurystyki na podstawie tego jak plik wygląda, rozszerzenia itp.). Przykłady:
  - `text/plain`
  - `text/html`
  - `image/jpeg`
  - `video/mpeg`
  - `application/msword` — dokument `.doc(x)`
  - `application/pdf` — dokument PDF
  - `application/octet-stream` — ciag bajtów bez interpretacji

Pobieranie strony WWW krok po kroku (3)

- Serwer wysyła odpowiedź:

```
HTTP/1.1 200 OK
Server: Apache/2.4.38 … OpenSSL/0.9.8k
Last-Modified: Wed, 29 Apr 2020 21:58:30 GMT
Content-Length: 5387
Content-Type: text/html

PLIK (w tym przypadku dokument HMTL)
```

- Serwer zamyka połączenie TCP (lub czeka na następne polecenie).
- Przeglądarka wykonuje akcję w zależności od typu MIME (pola `Content-Type`), tj. wyświetla, używa wtyczki, używa zewnętrznej aplikacji.

Zapytanie warunkowe GET

- W nagłówku podajemy:

```
If-Modified-Since: Wed, 20 Apr 2021 23:27:04 GMT
```

- Możliwe odpowiedzi:
  - `200 OK`
  - `304 Not Modified`

- Umożliwia implementację pamięci podręcznej w przeglądarce.

Odpowiedzi HTTP

Typy odpowiedzi:

- `1xx`: informacyjne
- `2xx`: sukces (`200` = OK)
- `3xx`: przekierowania
- `4xx`: błąd po stronie klienta (błędne żądanie, brak autoryzacji, zabroniony dostęp, `404` = Not Found)
- `5xx`: błąd po stronie serwera (`500` = Internal Server Error)

Hipertekst

- Wiele standardów: HTML, XHTML, XML, …
- Dokument zawiera:
  - odnośniki do innych dokumentów
  - oraz odnośniki do elementów osadzonych w dokumencie:
    - obrazki i filmy
    - skrypty w javascript
    - arkusze stylów CSS (definiują wygląd, HTML określa tylko strukturę).
    - czcionki
    - …
  - elementy osadzone są pobierane przez kolejne żądania HTTP i wyświetlane przez przeglądarkę.

Połączenia trwałe (1)

- **HTTP 1.0**
  - Każda para żądanie-odpowiedź w osobnym połączeniu TCP.
  - Nawiązywanie połączenia TCP = duży narzut czasowy.
  - Zazwyczaj przeglądarka pobiera wiele dokumentów naraz (np. strona WWW + obrazki).

- **HTTP 1.1, 2.0, …**
  - Wiele żądań i odpowiedzi w jednym połączeniu TCP.
  - Połączenie domyślnie otwarte.
  - Zamknięcie połączenia po odpowiedzi na żądanie, w którym umieścimy wiersz `Connection: close`.

Połączenia trwałe (2)

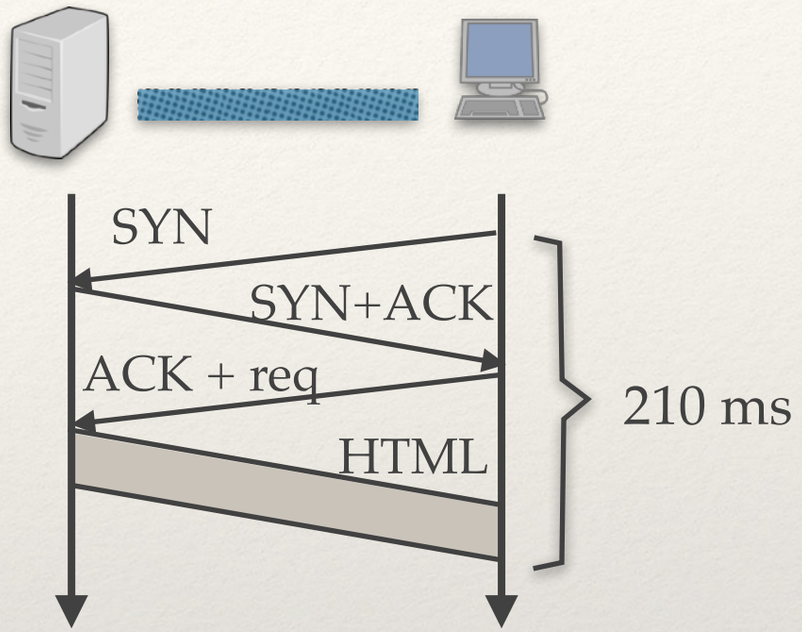
Przykład:

- Pobieranie strony HTML + 10 obrazków.
- Każdy obiekt mieści się w jednym segmencie TCP.
- Czas propagacji: 50 ms.
- Czas nadawania (pełnego) segmentu z danymi: 10 ms.
- Czas nadawania segmentu kontrolnego TCP lub segmentu z zapytaniem HTTP: 0 ms.

Połączenia trwałe (3)

HTTP/1.0 (bez połączeń trwałych).

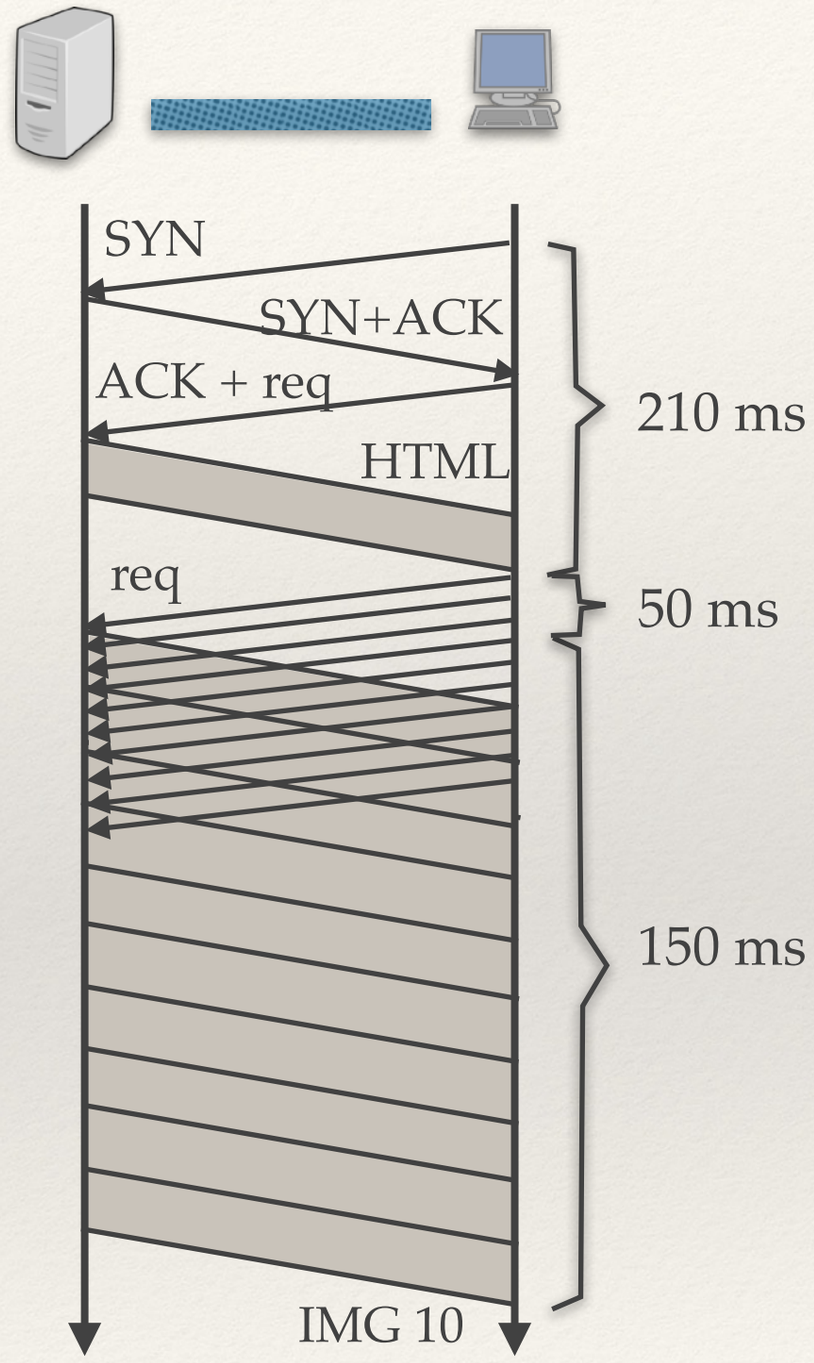
- Otrzymanie strony HTML: 210 ms.
- Pobieranie każdego z obrazków: kolejne: 210 ms.
- Usprawnienie: dwa równoległe połączenia do serwera → pobieranie 10 obrazków trwa 210 ms * (10/2) = 1050 ms.
- Całkowity czas: 210 + 1050 = 1260 ms.



Połączenia trwałe (4)

HTTP/1.1 (połączenia trwałe).

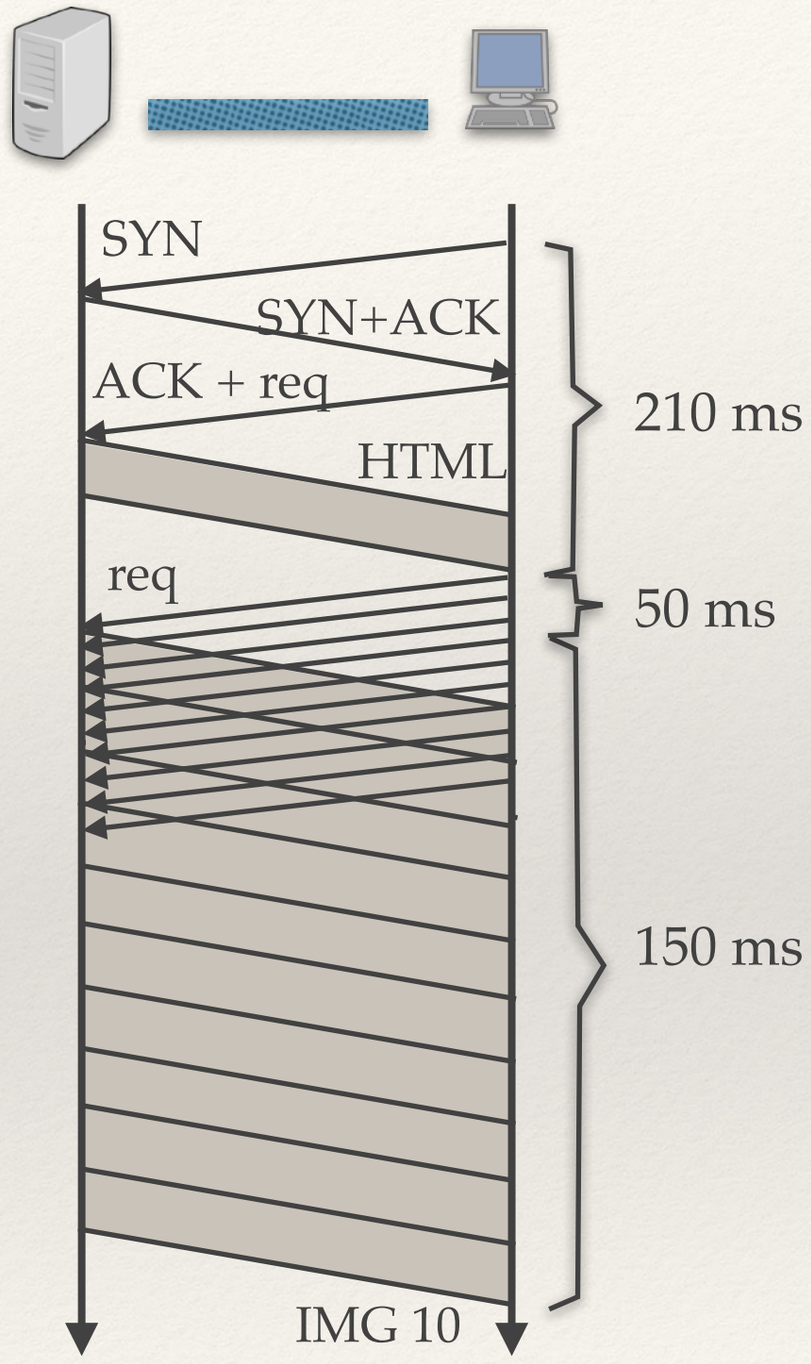
- Otrzymanie strony HTML: 210 ms.
- Wysłanie zapytania o obrazek nr 1: 50 ms.
- Wysyłanie obrazków: 50 + 10 * 10 = 150 ms.
- Całkowity czas: 210 + 200 = 410 ms.



Połączenia trwałe (4)

HTTP/1.1 (połączenia trwałe).

- Otrzymanie strony HTML: 210 ms.
- Wysłanie zapytania o obrazek nr 1: 50 ms.
- Wysyłanie obrazków: 50 + 10 * 10 = 150 ms.
- Całkowity czas: 210 + 200 = 410 ms.
- Niepotrzebne dodatkowe połączenia TCP.
- Jedno połączenie: okno TCP szybciej rośnie.



HTTP/2

- Oparty na SPDY (protokół zaproponowany przez Google).
- Binarny protokół.
- Kolejkowanie żądań (*pipelining*, obecny już w HTTP/1.1) + przesyłanie odpowiedzi w innej kolejności niż żądania.
- Server push: wysyłanie odpowiedzi na niezadane zapytania.
- Usuwanie powtarzających się nagłówków.
- Kompresja.

Dynamika po stronie klienta WWW

- Javascript: prosty obiektowy interpretowany język zintegrowany z HTML.
  - Współcześnie wzbogacany przez różne biblioteki (React, Angular, Vue, …)
- (Wycofywane) aplikacje Flash, Silverlight, aplety Javy (wykonanie realizowane przez odpowiednie wtyczki do przeglądarki).

Dynamika po stronie serwera WWW

URL może wskazywać na program generujący kod HTML.

- Popularne „frameworki“ do tworzenia aplikacji po stronie serwera: Django, Flask (Python), Spring (Java), Laravel (PHP), Node.js (Javascript), Phoenix (Erlang), Ruby on Rails (Ruby), …

- Dany program może komunikować się z serwerem WWW
  - za pośrednictwem IPC: standardy CGI / FastCGI (*Common Gateway Interface*)
  - za pomocą API: np. interfejs WSGI

- Formularze, przekazywanie parametrów (metody GET i POST).
- Cookies = utrzymywanie stanu sesji.

Formularze

- **Wysyłanie metodą GET**
  - Przeglądarka pobiera stronę `http://domena/program?par1=val1&par2=val2`
  - Serwer WWW uruchamia `program` i przekazuje mu parametry, `program` generuje odpowiedź HTML.
  - Problem: nie powinno się tak przekazywać haseł (dlaczego?)
  - Problem: ograniczenie na rozmiar przekazywanych danych.

- **Wysyłanie metodą POST**
  - Przeglądarka wysyła żądanie POST o stronę `http://domena/program`
  - W treści żądania (nie w nagłówku) znajduje się `par1=val1&par2=val2`
  - Można w ten sposób wysyłać też pliki do serwera.

HTTP jako warstwa transportowa

- Pisanie poprawnych programów korzystających z TCP jest niełatwe.
- Jak wykorzystać HTTP do przesyłania danych?
- Testowego klienta (przeglądarkę www) mamy za darmo.

- **REST**
  - Zautomatyzowany dostęp do niektórych serwisów WWW (eBay, Amazon, Twitter, Flickr, ...)
  - REST (Representational State Transfer) tworzenie usługi sieciowej wykorzystując metody (GET, PUT, POST, DELETE) protokołu HTTP.
  - REST nie jest standardem, raczej filozofią.
  - Łatwy do zautomatyzowania, czytelny dla człowieka

Lektura dodatkowa

- Kurose & Ross: rozdział 2.
- Tanenbaum: rozdział 7.
- HTTP 1.1: `https://tools.ietf.org/html/rfc2616`
- HTTP 2: `https://tools.ietf.org/html/rfc7540`

Zagadnienia

- Jaki jest cel systemu nazw DNS?
- Do czego służy plik `/etc/hosts`?
- Rozwiń skrót TLD (kontekst: DNS), podaj parę przykładów.
- Czym są strefy i delegacje DNS?
- Czym różni się rekurencyjne odpytywanie serwerów DNS od iteracyjnego?
- Jak działa odwrotny DNS? Jaki typ rekordów i jaką domenę wykorzystuje?
- Jakie znasz typy rekordów DNS? Co to jest rekord `CNAME`?
- Po co są wpisy sklejające w opisie delegacji DNS?
- Co robi funkcja `getaddrinfo`()?
- Opisz budowę adresu URL. Opisz budowę adresu URL w przypadku schematu `http`.
- W jakim celu serwer WWW ustawia typ MIME dla wysyłanej zawartości? Podaj kilka przykładów typów MIME.
- Wymień parę możliwych odpowiedzi HTTP wraz z ich znaczeniem.
- Po co w nagłówku żądania `HTTP/1.1` podaje się pole `Host`?
- Do czego służą pola `Accept, Accept-Language, User-Agent, Server, Content-Length, Content-Type` w nagłówku HTTP?
- Jak wygląda warunkowe zapytanie GET protokołu HTTP?
- Jakie znasz kody odpowiedzi protokołu HTTP?
- Na czym polegają połączenia trwałe w `HTTP/1.1`? Do czego służy opcja `Connection: close` w nagłówku HTTP?
- Do czego służą arkusze stylów CSS?
- Wymień parę możliwości uzyskiwania dynamicznych stron WWW.
- Po co stosuje się metodę POST?
- Co to jest technologia REST?