Warstwa aplikacji

część 2: inne zastosowania

Sieci komputerowe

Wykład 9

Marcin Bieńkowski

W dzisiejszym odcinku

- DNS (*Domain Name System*).
- Poczta elektroniczna.
- Sieci peer-to-peer.
- NAT (*Network Address Translation*) a warstwa aplikacji.

DNS

Nazwy domen

Nazwy domen:

- `ii.uni.wroc.pl` → `156.17.4.11`
- `z-Wroclaw.poznan-gw8.rtr.pionier.gov.pl` → `212.191.225.170`

Po co?

- Łatwiejsze do zapamiętania dla ludzi.
- Możliwość zmiany adresu IP serwera (np. przenosiny do innego ISP) bez powiadamiania klientów o zmianie.

Jeden serwer?

Przechowywanie przypisań "domena → adres IP" na jednym serwerze.

- Rozwiązanie stosowane w początkach Internetu.
- Możliwe ręczne zarządzanie:
  - Aktualizacje przez maile do administratora serwisu.
  - Zamiast odpytywania bazy można ją pobrać i zapisać w lokalnym pliku `/etc/hosts`.
- **Problemy z koordynacją i skalowalnością.**

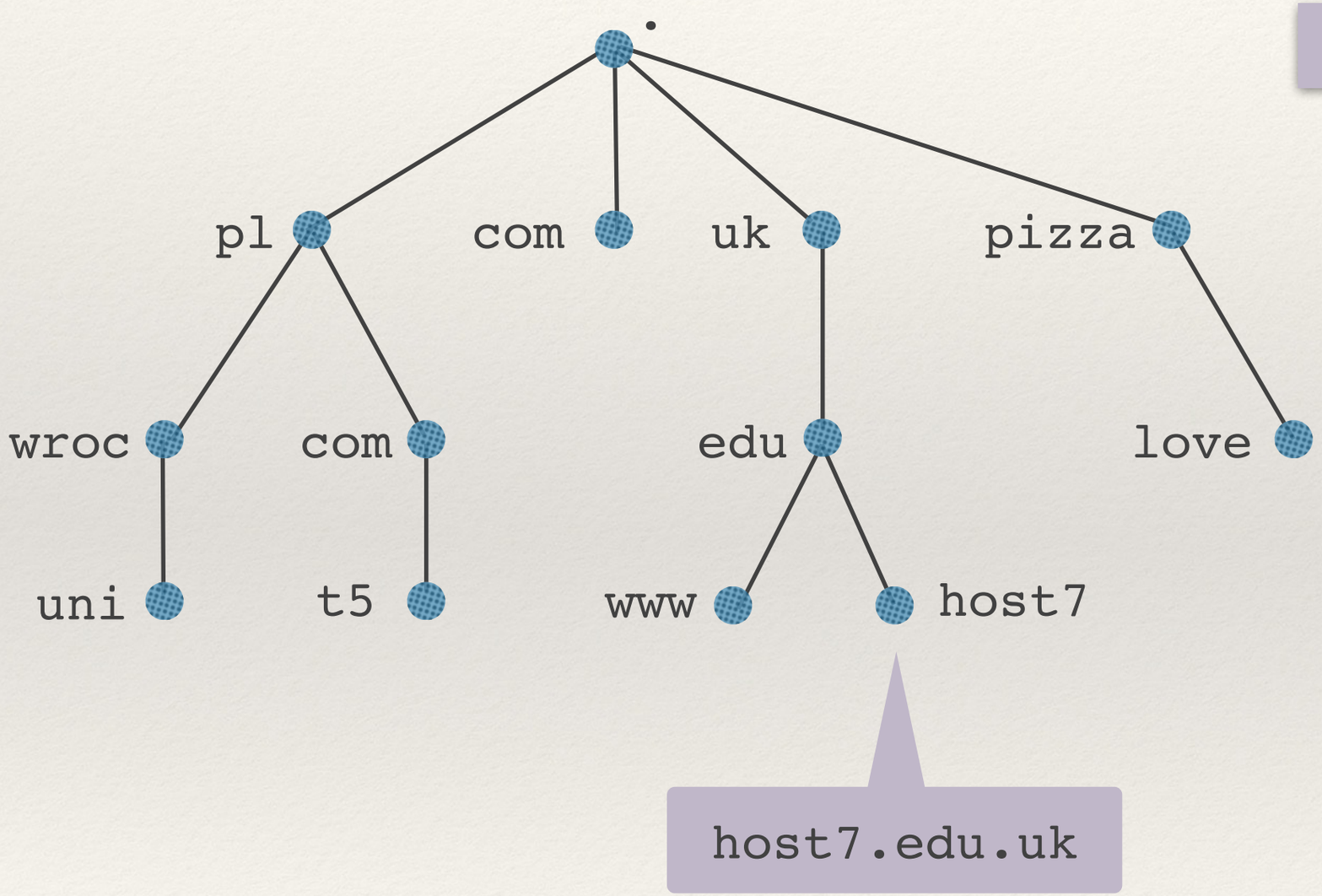
Przechowywanie rekordów

Jak przechowywać dużą bazę danych?

- Rekordy w postaci “domena → wartość”.
  - “Wartość” = adres IP lub inny tekstowy opis.
- Liczba rekordów: ~300 mln domen.
  - Nie licząc poddomen, np. prefiksów “www” lub “mail”.
- Ciągłe aktualizacje.
- Potrzebna odporność na awarie pojedynczych serwerów.

- **Ułatwienie: nazwy mają hierarchię.**

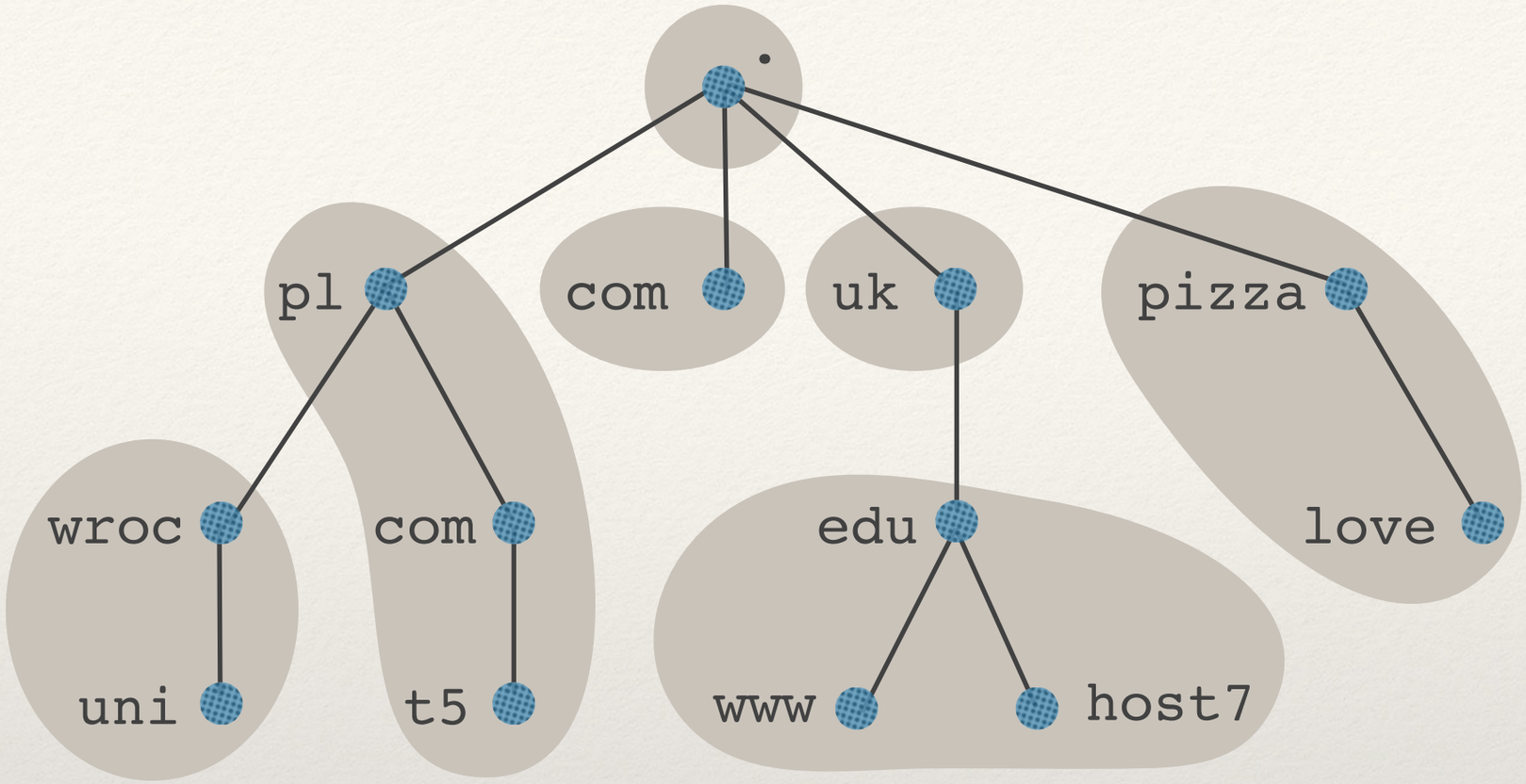
Hierarchia nazw domen



Korzeń oznaczamy kropką.

TLD (top level domains).

Rozproszone zarządzanie: strefy



Strefa:

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

Serwery główne

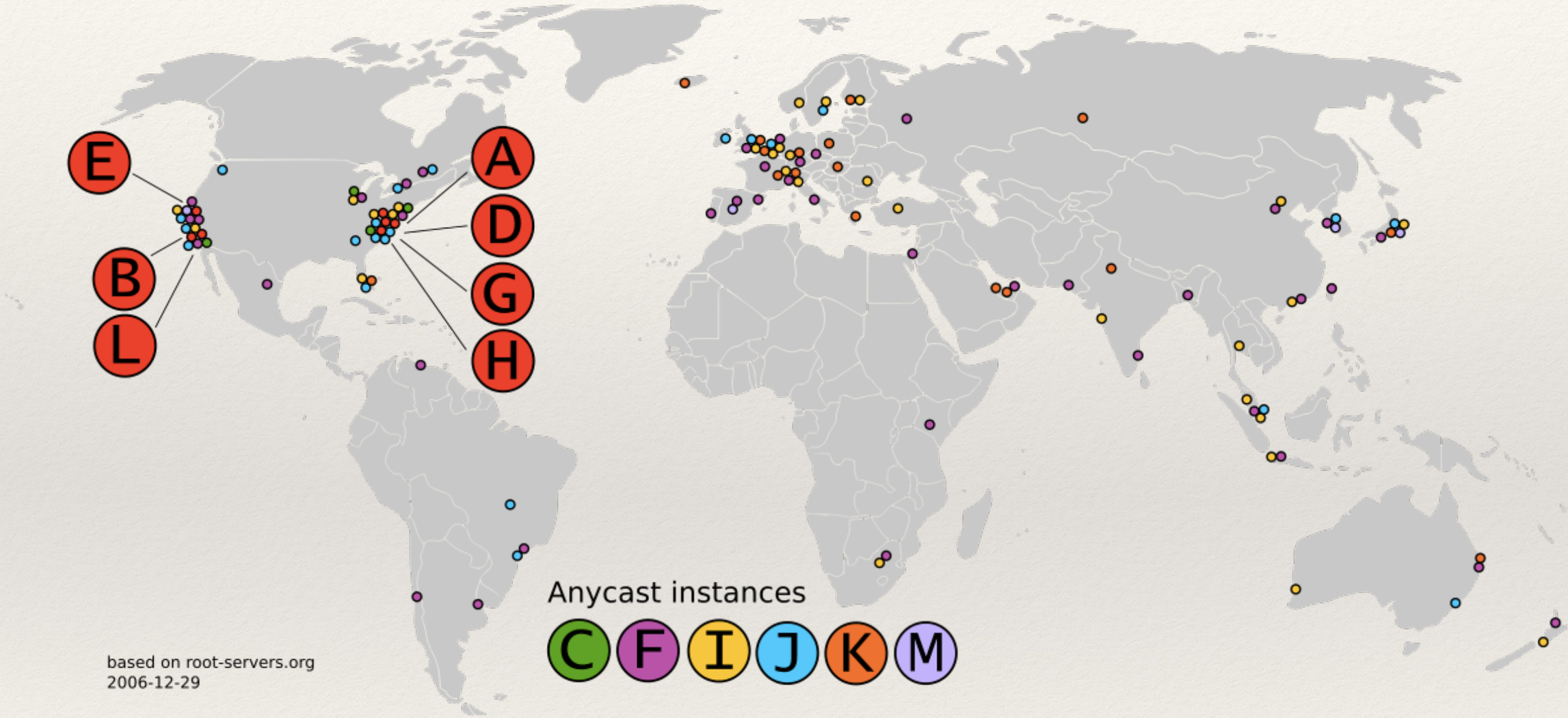
Istnieje ok. 400 serwerów głównych dla strefy „.“

```
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
…
```

Powyższa informacja umieszczana w systemowych plikach konfiguracyjnych.

Serwery główne: tylko 13 adresów IP

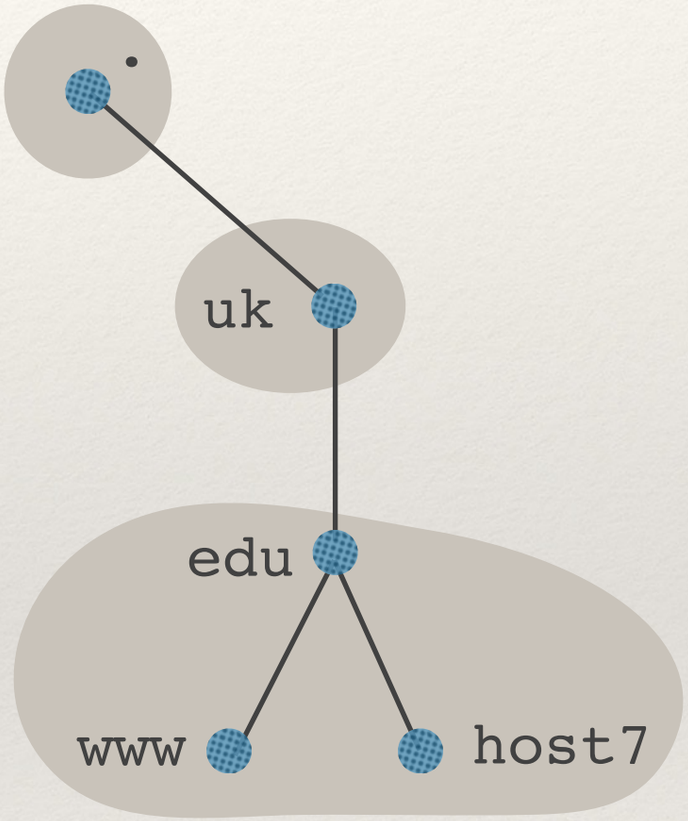
Adresy anycast = wiele serwerów ma ten sam adres IP.



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

Rozszyfrowywanie nazw (resolving)

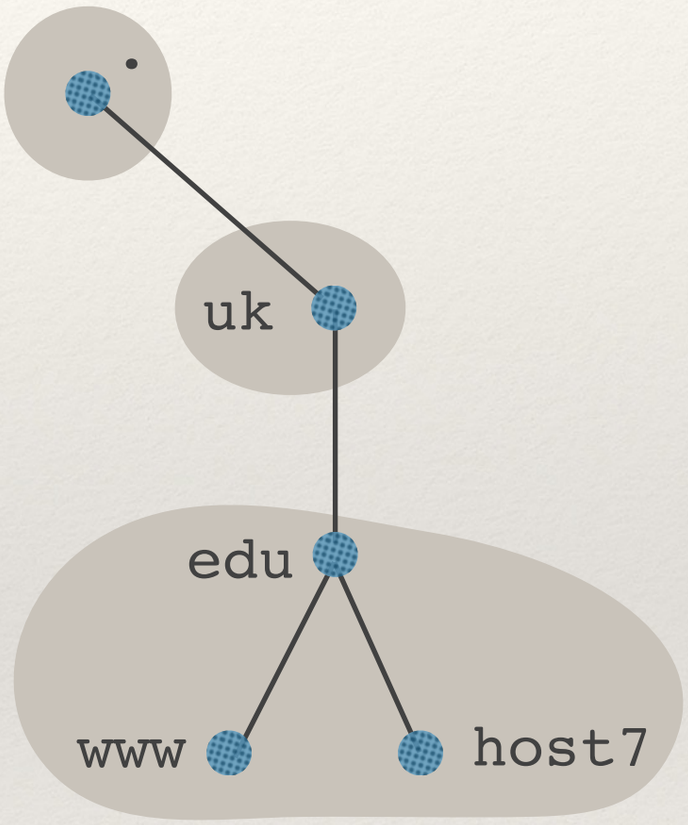
Chcemy poznać adres IP dla `host7.edu.uk`.



Rozszyfrowywanie nazw (resolving)

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

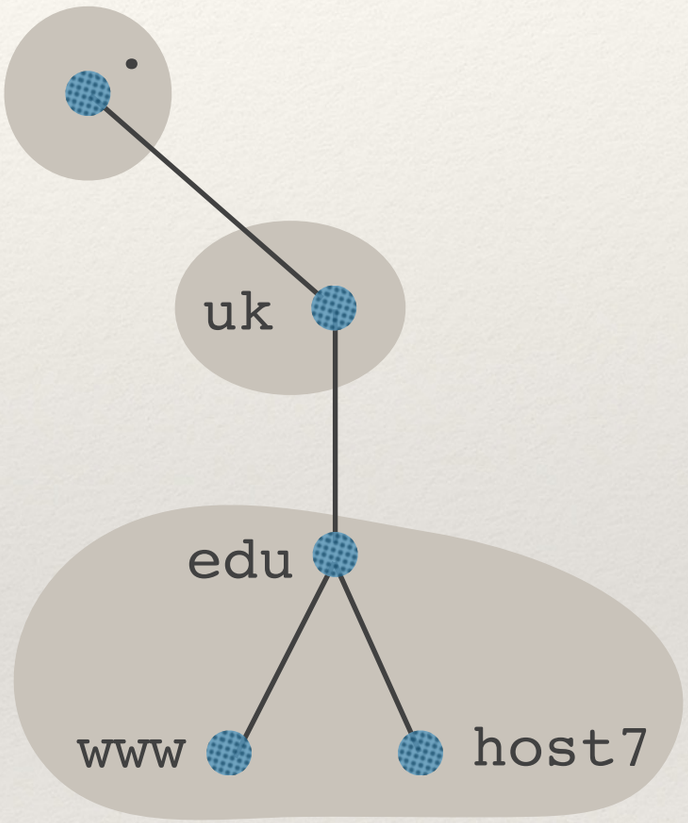
- Pytamy jeden z serwerów nazw dla "`.`",
  np. `E.ROOT-SERVERS.NET` o adresie
  `192.203.230.10`.



Rozszyfrowywanie nazw (resolving)

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

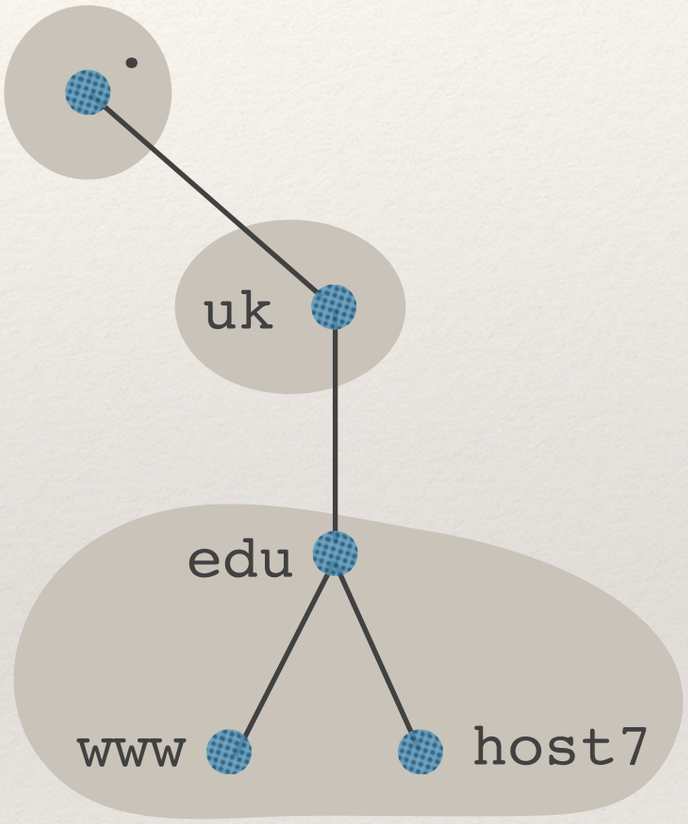
- Pytamy jeden z serwerów nazw dla "`.`", np. `E.ROOT-SERVERS.NET` o adresie `192.203.230.10`.
- Serwer nie wie, ale mówi, że serwerem nazw dla uk jest `foo.uk` o adresie `1.2.3.4`.



Rozszyfrowywanie nazw (resolving)

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

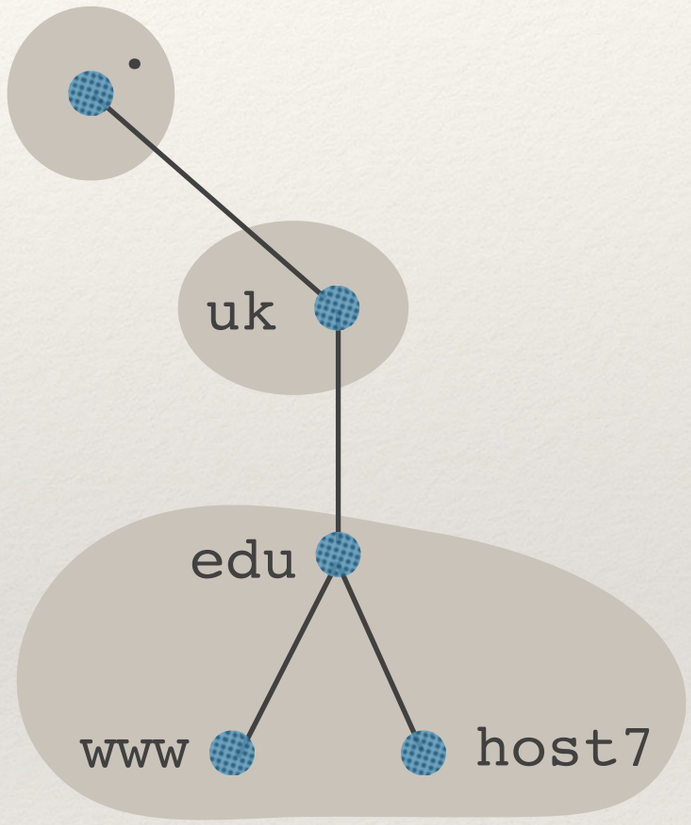
- Pytamy jeden z serwerów nazw dla "`.`", np. `E.ROOT-SERVERS.NET` o adresie `192.203.230.10`.
- Serwer nie wie, ale mówi, że serwerem nazw dla `uk` jest `foo.uk` o adresie `1.2.3.4`.
- Pytamy `foo.uk`.



Rozszyfrowywanie nazw (resolving)

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

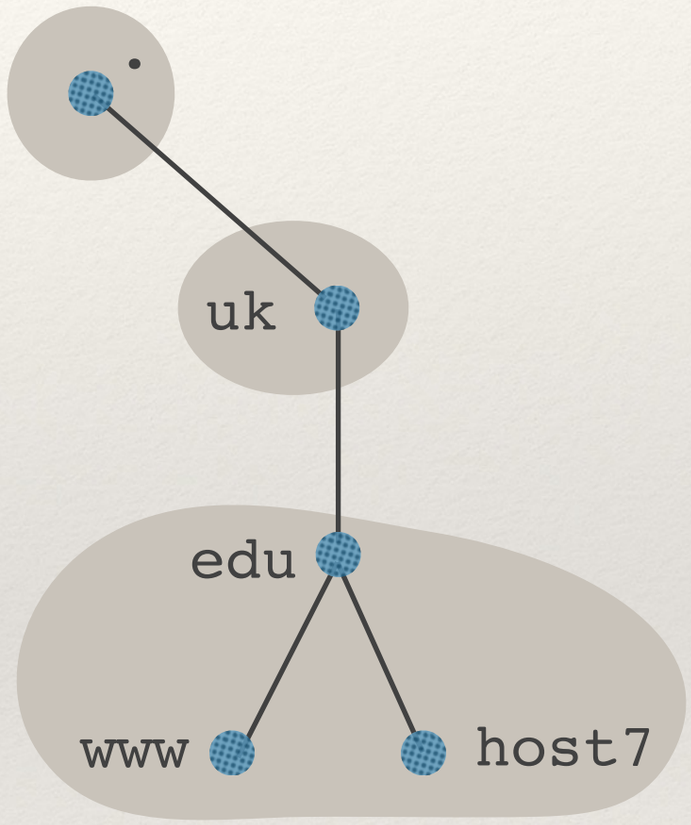
- Pytamy jeden z serwerów nazw dla "`.`", np. `E.ROOT-SERVERS.NET` o adresie `192.203.230.10`.
- Serwer nie wie, ale mówi, że serwerem nazw dla `uk` jest `foo.uk` o adresie `1.2.3.4`.
- Pytamy `foo.uk`.
- Serwer nie wie, ale mówi, że serwerem nazw dla `edu.uk` jest `foo.bar.uk` o adresie `5.6.7.8`.



Rozszyfrowywanie nazw (resolving)

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

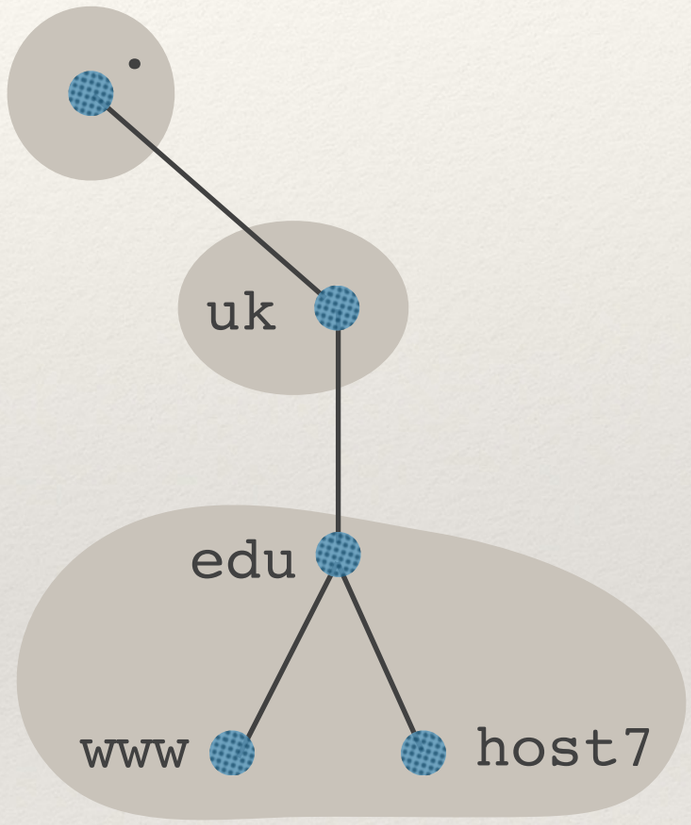
- Pytamy jeden z serwerów nazw dla "`.`", np. `E.ROOT-SERVERS.NET` o adresie `192.203.230.10`.
- Serwer nie wie, ale mówi, że serwerem nazw dla `uk` jest `foo.uk` o adresie `1.2.3.4`.
- Pytamy `foo.uk`.
- Serwer nie wie, ale mówi, że serwerem nazw dla `edu.uk` jest `foo.bar.uk` o adresie `5.6.7.8`.
-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



Rozszyfrowywanie nazw (resolving)

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

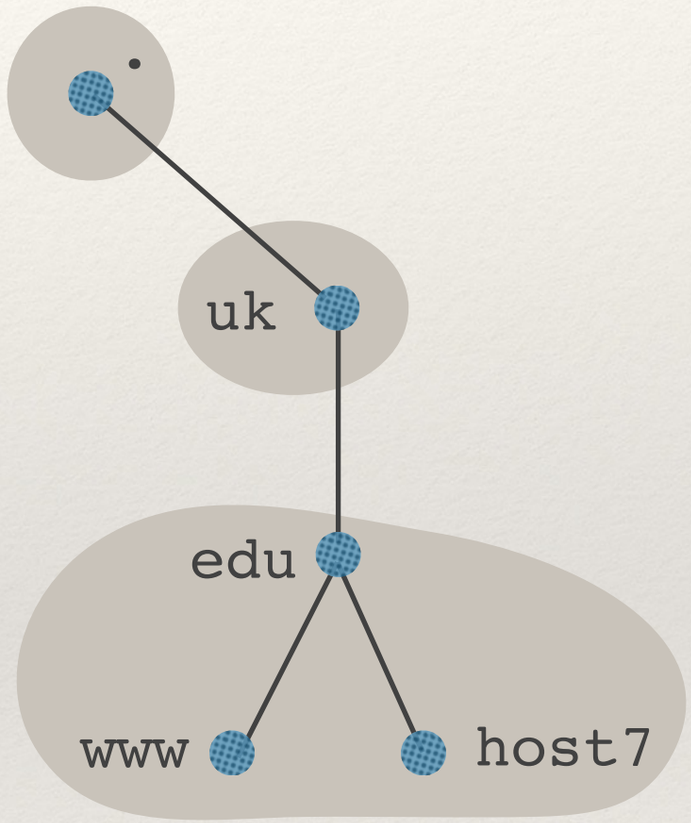
- Pytamy jeden z serwerów nazw dla "`.`",
  np. `E.ROOT-SERVERS.NET` o adresie
  `192.203.230.10`.
- Serwer nie wie, ale mówi, że serwerem nazw dla
  `uk` jest `foo.uk` o adresie `1.2.3.4`.
- Pytamy `foo.uk`.
- Serwer nie wie, ale mówi, że serwerem nazw dla
  `edu.uk` jest `foo.bar.uk` o adresie `5.6.7.8`.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

  - Resolver może być też serwerem DNS (odpowiedzialnym za jakąś strefę).

Rekordy A i AAAA

Rekord DNS = (typ, nazwa domeny, wartość).

Typ A

- wartość = adres IPv4 (`172.66.147.243`)

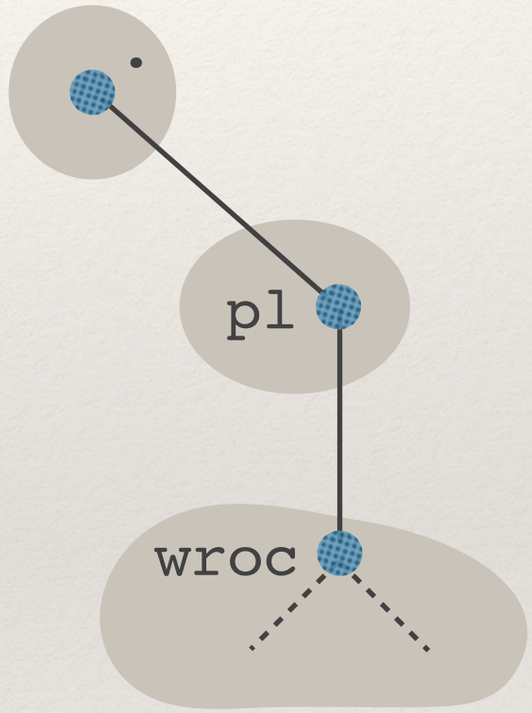
Typ AAAA

- wartość = adres IPv6 (`2606:4700:10::6814:179a`)

Rekordy NS

Typ NS (*nameserver*)

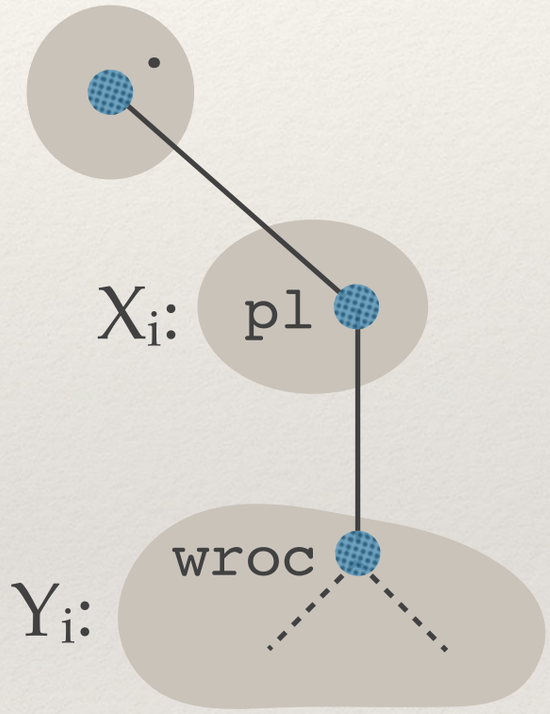
- Nazwa = nazwa strefy (`wroc.pl`).
- Wartość = nazwa serwera DNS obsługującego daną strefę (`sun2.pwr.wroc.pl`).



Gdzie przechowywać rekordy NS?

Rekord (NS, `wroc.pl` → `sun2.pwr.wroc.pl`)

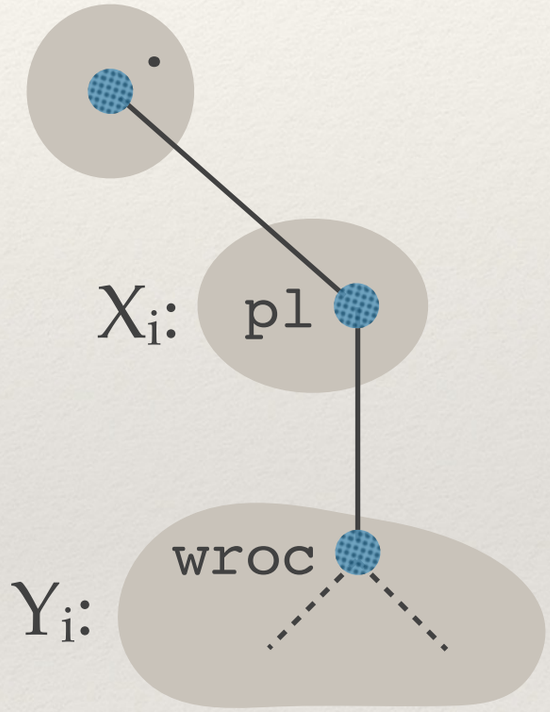
- Zapisany na serwerach nazw Y_i dla strefy `wroc.pl` (m.in. `sun2.pwr.wroc.pl`).
- Zapisany na serwerach nazw X_i dla strefy `pl` (tzw. „**delegacje**“). **Potrzebne podczas przechodzenia drzewa DNS.**



Gdzie przechowywać rekordy NS?

Rekord (NS, `wroc.pl` → `sun2.pwr.wroc.pl`)

- Zapisany na serwerach nazw Y_i dla strefy `wroc.pl` (m.in. `sun2.pwr.wroc.pl`).
- Zapisany na serwerach nazw X_i dla strefy `pl` (tzw. „**delegacje**“). **Potrzebne podczas przechodzenia drzewa DNS.**
- Serwery X_i zazwyczaj mają również odpowiednie rekordy typu A, np. `sun2.pwr.wroc.pl` → `156.17.5.2`.



Dodatkowe rekordy DNS

Typ CNAME (*canonical name*)

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

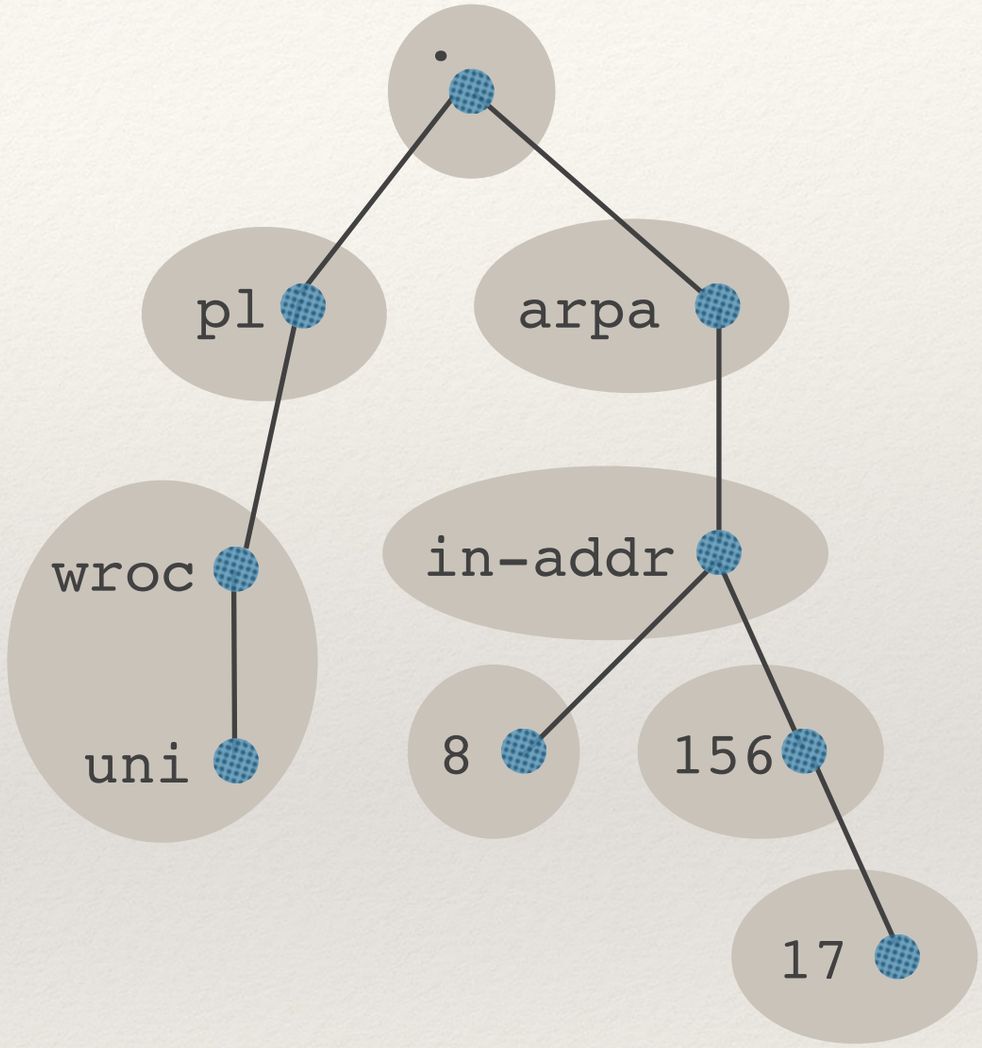
Typ MX (*mail exchanger*)

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

Domena odwrotna

Typ PTR:

- nazwa = “odwrócony” adres IP (`11.4.17.156.in-addr.arpa`)
- wartość = „główna“ nazwa domeny (`ii.uni.wroc.pl`)



Pamięć podręczna DNS

Pamięć podręczna.

- Resolver DNS zapisuje poznany fragment drzewa (w tym zwrócony wynik) w swojej pamięci podręcznej.

Rekordy DNS mają czas życia (TTL).

- Po tym czasie wyrzucane z pamięci podręcznej.
- 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.
  - Niezauważalne 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.example.com`, `ftp.example.com`, `mail.example.com`).

Poczta elektroniczna

Gdzie przechowywać maile?

Maile dla użytkownika `user@uwr.edu.pl`:

- **Wpis DNS typu A:**
  - A, `uwr.edu.pl` → `156.17.87.85`
  - Tam nie będzie maili.

- **Wpis DNS typu MX (*mail exchange*):**
  - MX, `uwr.edu.pl` → `uwr-edu-pl.mail.protection.outlook.com`
  - A, `uwr-edu-pl.mail.protection.outlook.com` → `52.101.73.4`
  - Maile użytkownika `user@uwr.edu.pl` będą przechowywane na serwerze `52.101.73.4`.

Wysyłanie poczty

Chcemy wysłać pocztę do adresu `user@xyx.com`.

SMTP = protokół przekazywania poczty

- Wykorzystuje protokół TCP (port 25 lub 587).
- Przykładowa komunikacja (C = klient, S = serwer):

```
C: MAIL FROM:<sender@example.org>
S: 250 2.1.0 Ok
C: RCPT TO:<recipient@example.com>
S: 250 2.1.5 Ok
C: DATA
C: From: "Sender Name" <sender@example.org>
C: To: "Recipient Name" <recipient@example.com>
C: Subject: Example SMTP Communication
C:
C: Hello,
C: ...
C: Marcin
C: .
S: 250 2.0.0 Ok: queued as 12345
```

treść maila

Przykładowy email

```
Date: Fri, 17 Apr 2026 17:41:35 +0200
From: mbi <mbi@ii.uni.wroc.pl>
To: marcin.bienkowski@cs.uni.wroc.pl
Subject: Testowy email
Message-ID: <E1wFJAR-00000005H6i-1xrTii.uni.wroc.pl>
MIME-Version: 1.0
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: 8bit
User-Agent: Mutt/2.2.13 (2024-03-09)

Jakaś treść maila.
M.
```

Inne częste pola nagłówka (ustawiane przez klienta)

- `Cc:`
- `Bcc:` („ślepa kopia“)
- `In-Reply-To:` (ID maila, na którego odpowiadamy)
- `References:`

Typ zawartości

Pole `Content-Type:` w nagłówku.

- Określa, czym jest treść maila (w standardzie MIME), np:
  - czysty tekst (`text/plain`),
  - HTML (`text/html`).

- Definiuje kodowanie znaków:

```
Content-Type: text/plain; charset=utf-8
Content-Transfer-Encoding: 8bit
```

Załączniki

```
Content-Type: multipart/mixed; boundary="--UNIKATOWY-CIĄG-ZNAKÓW"
Content-Transfer-Encoding: 8bit

--UNIKATOWY-CIĄG-ZNAKÓW
Content-Type: text/plain; charset=utf-8
Content-Disposition: inline
Content-Transfer-Encoding: 8bit

Wiadomość testowa
M.
--UNIKATOWY-CIĄG-ZNAKÓW
Content-Type: image/jpeg
Content-Disposition: attachment;
                     filename="obrazek.jpg"
Content-Transfer-Encoding: base64

ZAWARTOŚĆ-PLIKU-ZAKODOWANA-W-BASE64.
--UNIKATOWY-CIĄG-ZNAKÓW
```

treść tekstowego maila w UTF-8

załącznik obrazek.jpg

HTTP vs SMTP

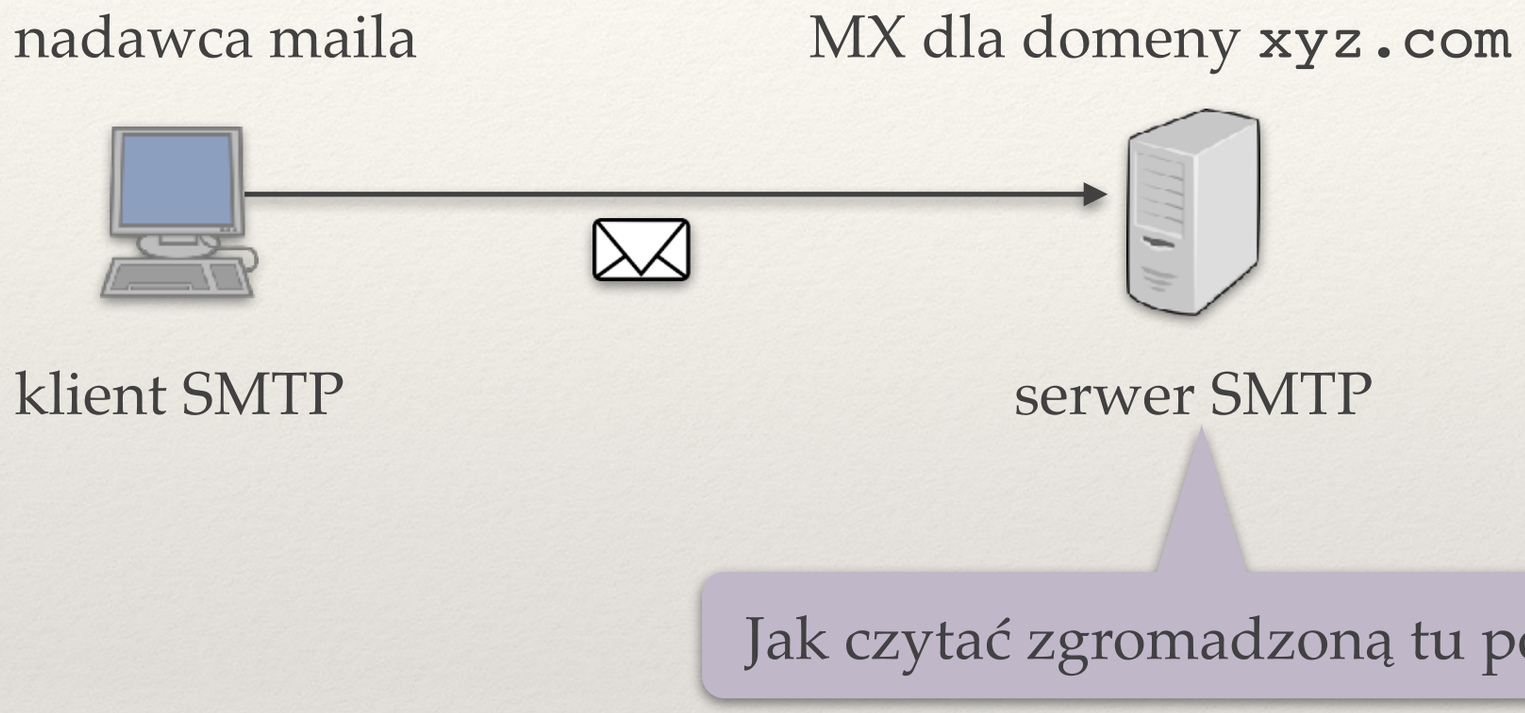
	HTTP	SMTP
co robią	Przesyłają pliki	
kto wysyła	serwer	klient
pojedyncza komunikacja	para zapytanie-odpowiedź, bezstanowy	bardziej interaktywny, stanowy
	jeden plik	wiele plików (mail z wieloma załącznikami)

Co dalej?

Chcemy wysłać pocztę do adresu user@xyx.com.

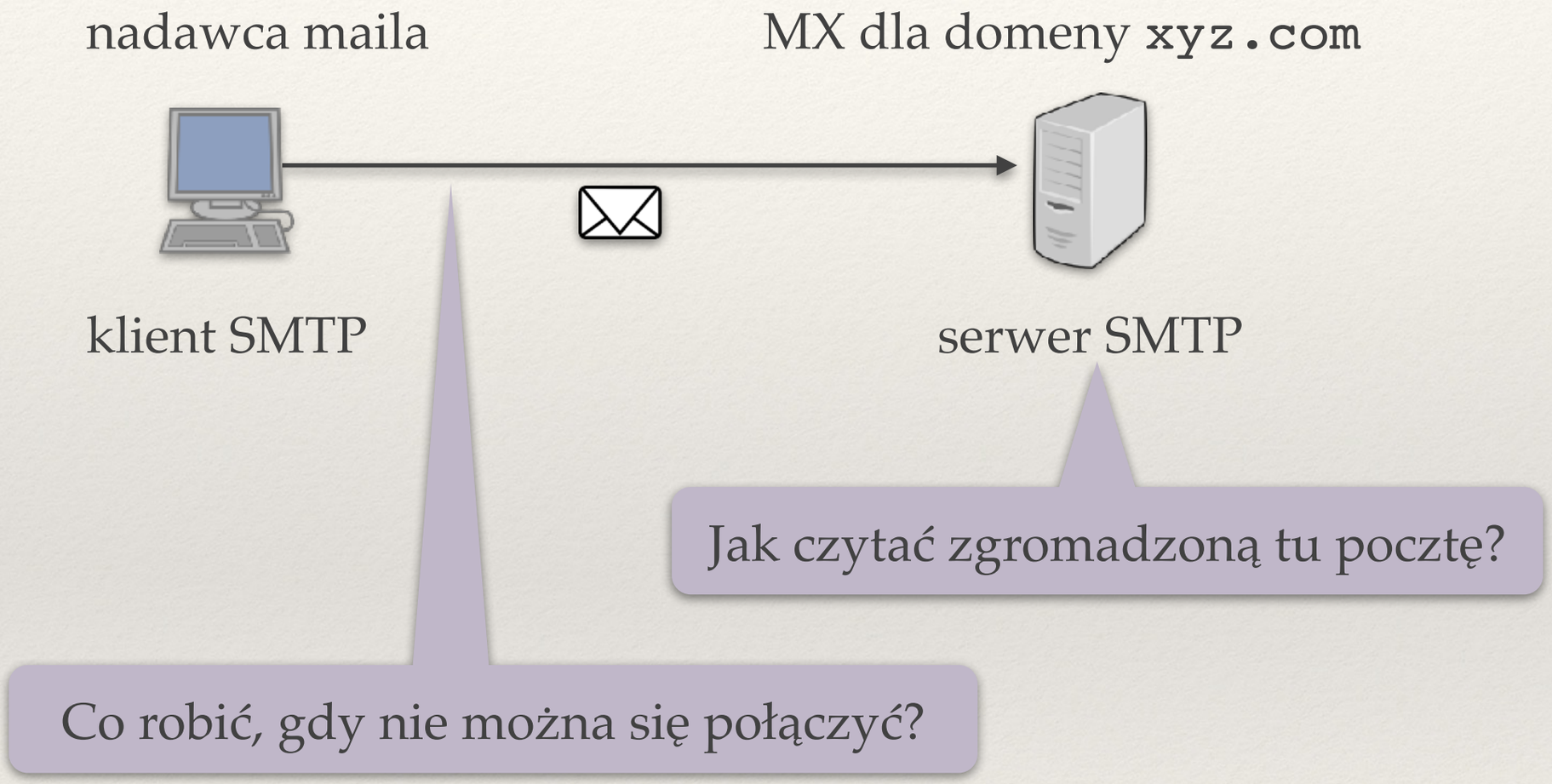
Co dalej?

Chcemy wysłać pocztę do adresu user@xyx.com.

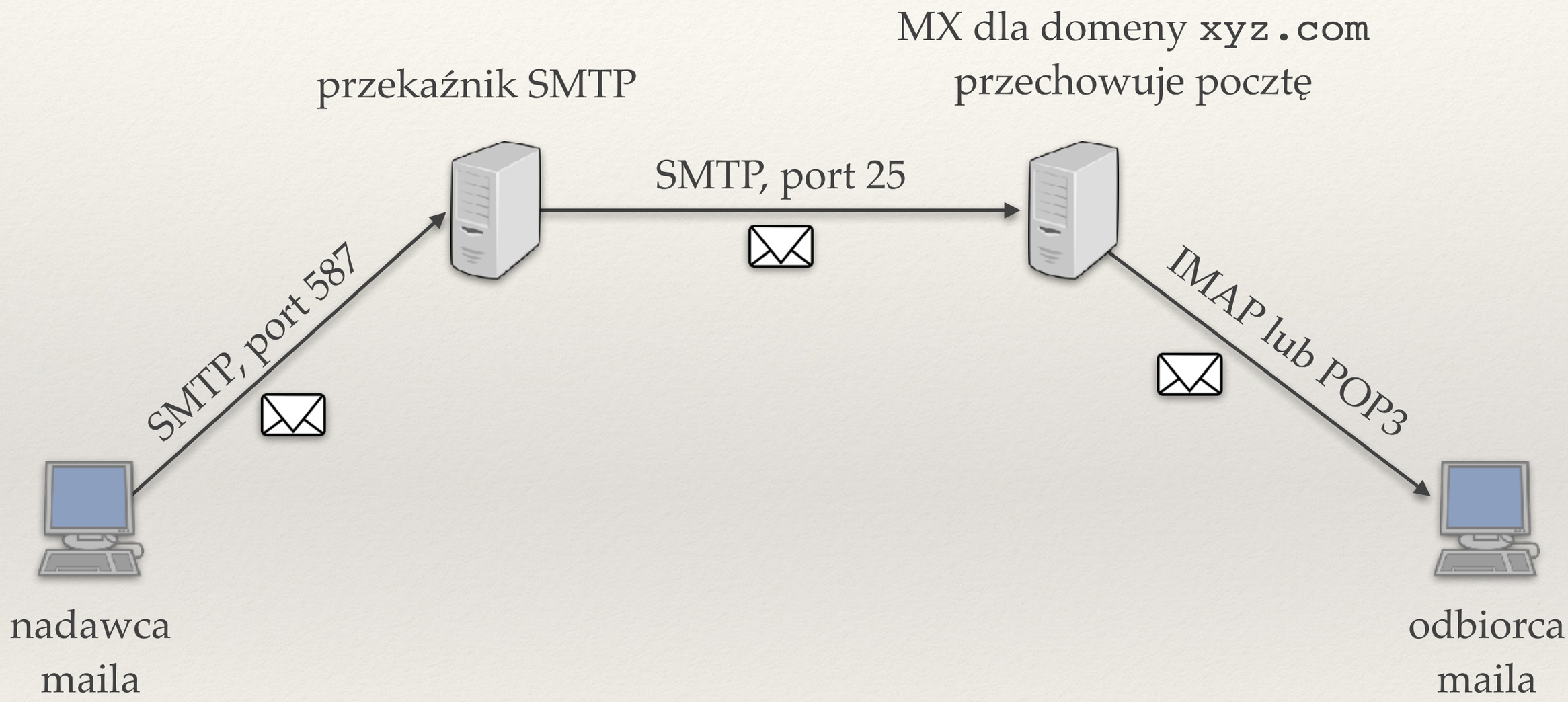


Co dalej?

Chcemy wysłać pocztę do adresu user@xyx.com.

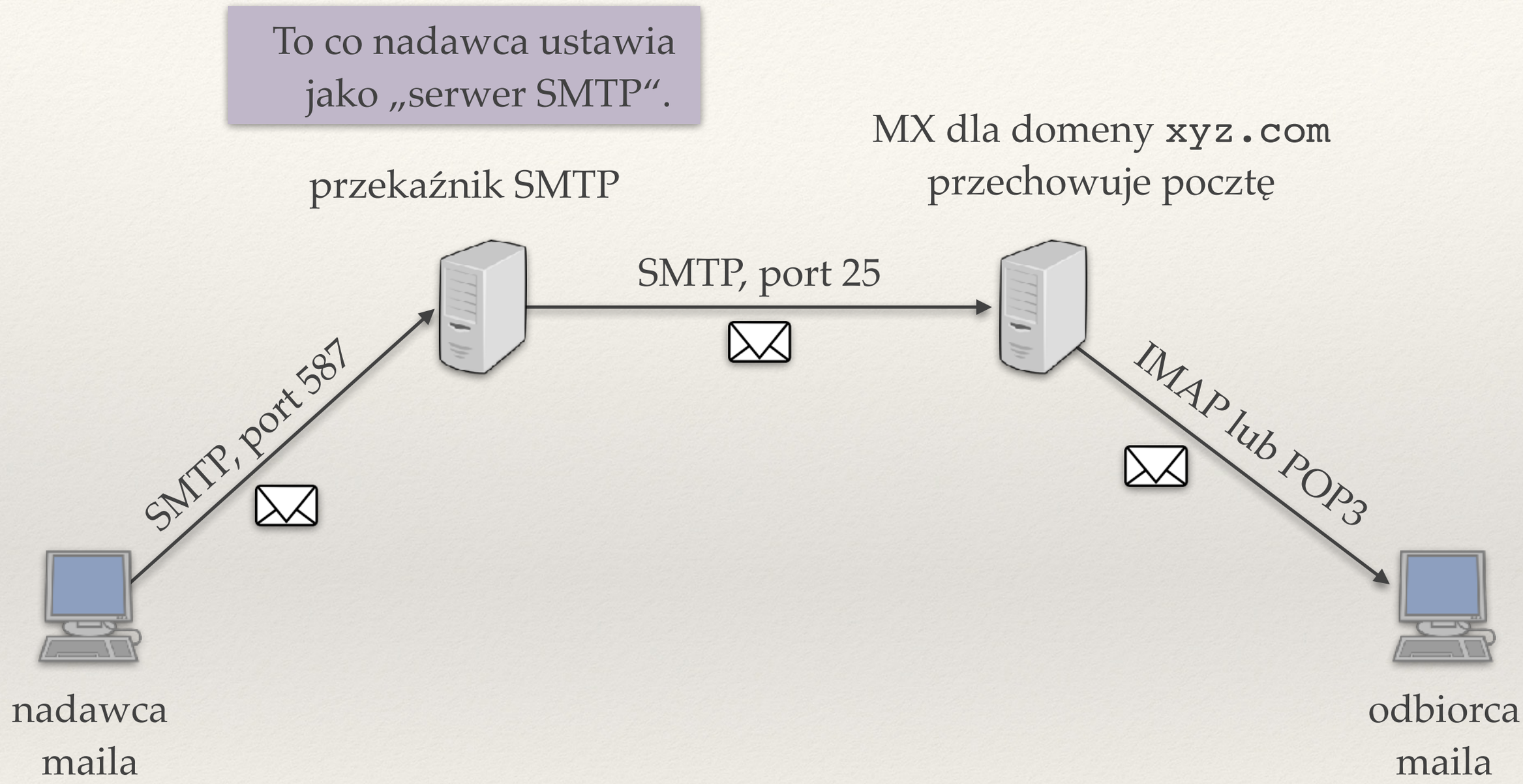


Wysyłanie i odbieranie poczty



Wysyłanie i odbieranie poczty

To co nadawca ustawia jako „serwer SMTP“.



Wysyłanie i odbieranie poczty

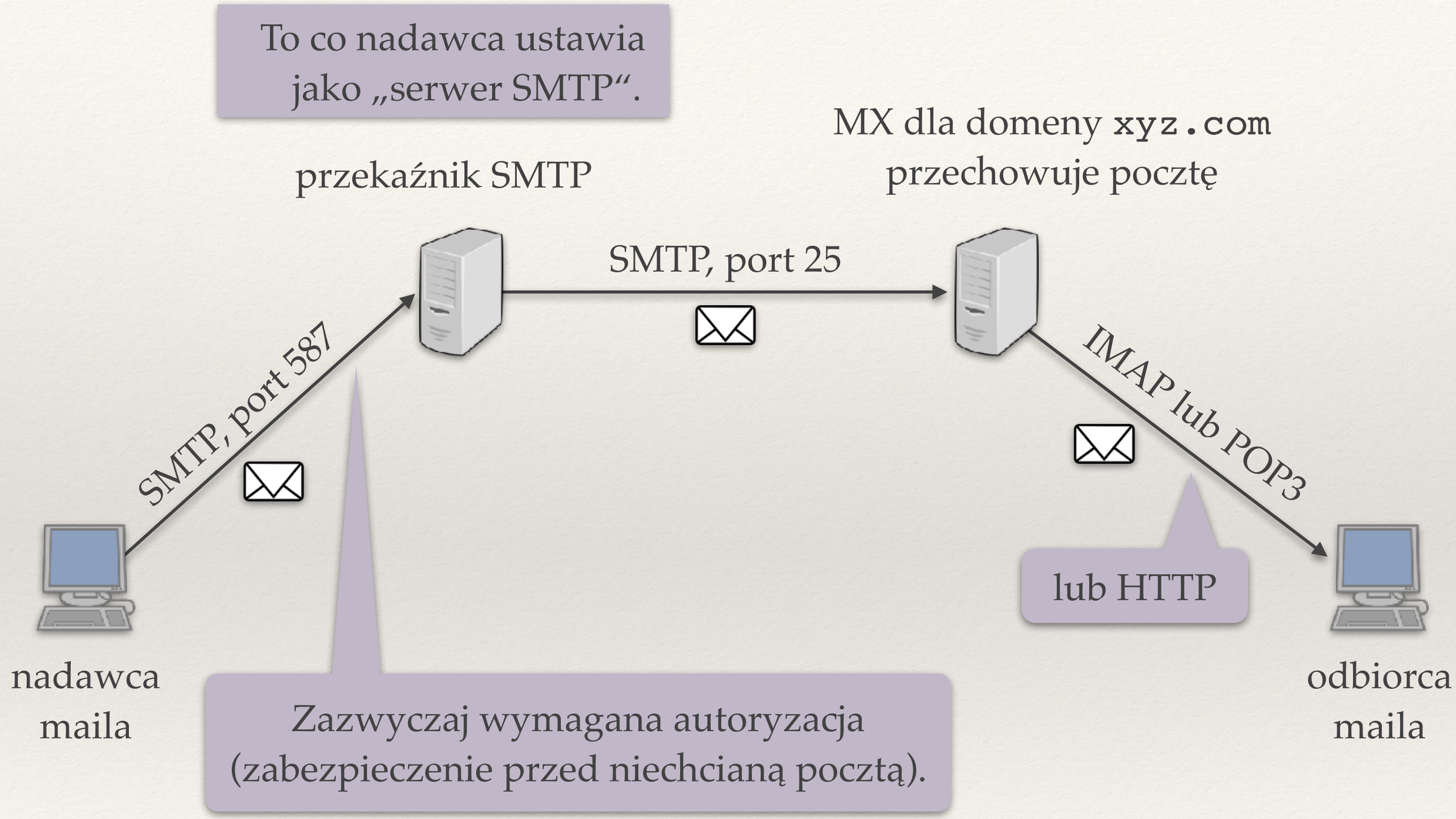
To co nadawca ustawia jako „serwer SMTP“.

przekaźnik SMTP

MX dla domeny `xyz.com` przechowuje pocztę

Zazwyczaj wymagana autoryzacja (zabezpieczenie przed niechcianą pocztą).

Wysyłanie i odbieranie poczty



Serwery (i przekaźniki) modyfikują maila

```
Delivered-To: marcin.bienkowski@cs.uni.wroc.pl
Received: by 10.64.232.142 with SMTP id …;
        Fri, 17 Apr 2026 08:41:37 -0700 (PDT)
Received: from aisd.ii.uni.wroc.pl (156.17.4.30)
         by mx.google.com with ESMTP id …
         for <marcin.bienkowski@cs.uni.wroc.pl>;
         Fri, 17 Apr 2026 08:41:36 -0700 (PDT)
Received: by aisd.ii.uni.wroc.pl (Postfix) id …;
         Fri, 17 Apr 2026 17:41:35 +0200 (CEST)
Date: Fri, 17 Apr 2026 17:41:35 +0200
From: mbi <mbi@ii.uni.wroc.pl>
…
Jakaś treść maila.
M.
```

pola ustawiane przez odbiorcę

pola ustawiane przez serwery pośredniczące

pola ustawiane przez nadawcę

Spam: niechciane wiadomości pocztowe

Sposoby wykrywania i usuwania spamu:

- Metody statystyczne, uczenie maszynowe.
- Blokowanie zakresów adresów IP.
- Spowalnianie połączeń:
  - SMTP 451 “Please try again later”.
  - Wolne odbieranie z okna TCP.
- Zabezpieczenia przed podszywaniem: SPF, podpisy cyfrowe, …

SPF (Sender Policy Framework)

Rekord SPF (o typie TXT) w DNS.

- Przykładowo:

```
TXT, cs.uni.wroc.pl „v=spf1 ip4:156.17.4.0/24
                    mx:cs.uni.wroc.pl mx:gmail.com -all”
```

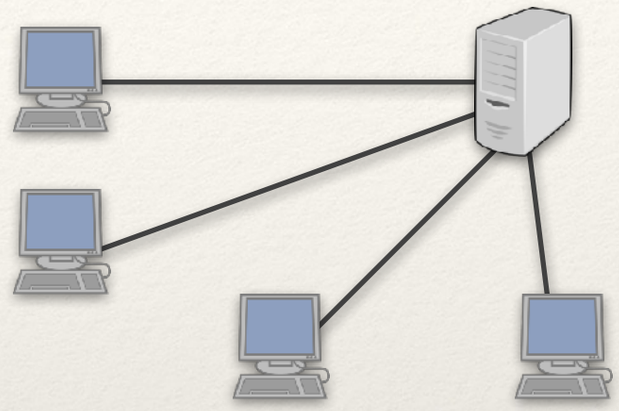
- Definiuje, jakie komputery są uprawnione do wysyłania poczty z polem `From: .*@cs.uni.wroc.pl`:
  - komputery o adresach z `156.17.4.0/24`.
  - serwery SMTP obsługujące pocztę dla domen `cs.uni.wroc.pl` i `gmail.com` .

- Rekord sprawdzany przez odbiorcę.

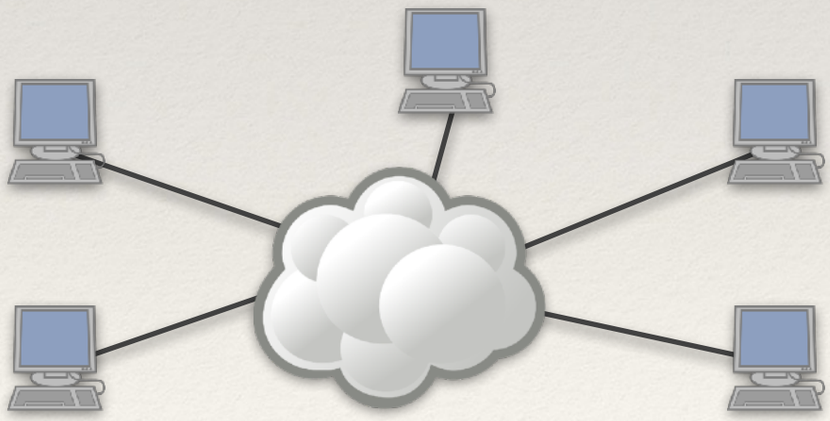
Peer-to-peer

Klient-serwer a peer-to-peer

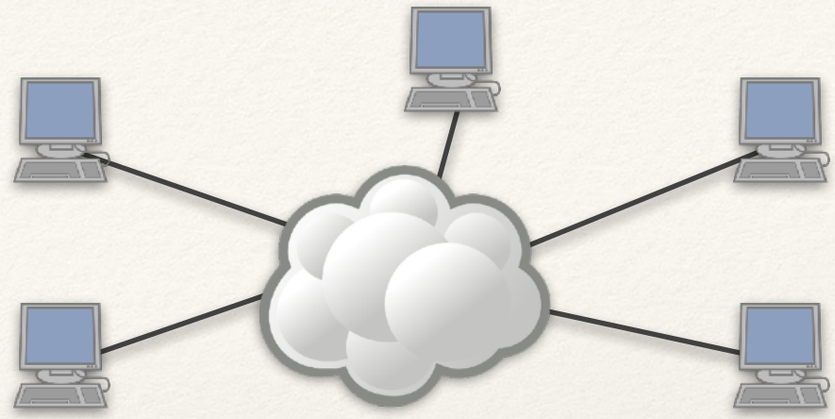
- **Architektura klient-serwer** (HTTP, DNS, SMTP, IMAP, …)



- **Architektura peer-to-peer** (BitTorrent, połączenia video 1:1, …)



Peer-to-peer

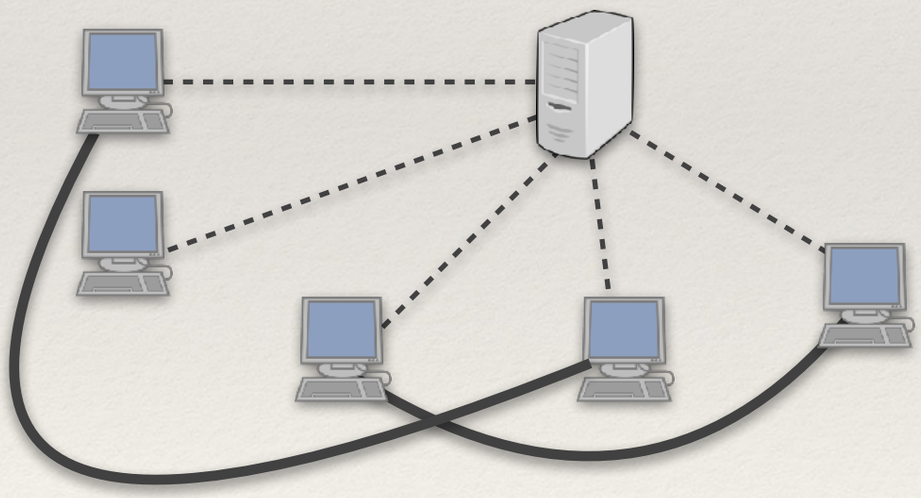


Architektura P2P:

- Wszystkie komputery są jednocześnie klientami i serwerami.
- Każdy komputer może nawiązywać połączenia z innymi.
- Brak centralnego miejsca z danymi.
  - Lepsza skalowalność i niezawodność.
  - Autonomia ale trudniejsze zagwarantowanie współpracy całości.

„Niepełne” peer-to-peer

- W większości architektur peer-to-peer istnieją wyróżnione komputery.
  - Przechowują np. bazę użytkowników.
  - Pomagają w podłączeniu (punkt pierwszego kontaktu + później).



komunikaty kontrolne

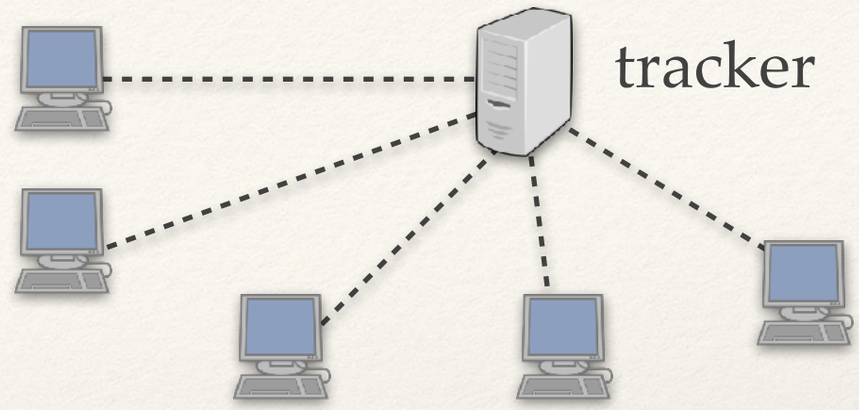
właściwe dane

Warstwa aplikacji a warstwa transportowa

Peer-to-peer to określenie logiki warstwy aplikacji.

- Do wymiany danych wykorzystywana warstwa transportowa (TCP lub UDP).
- Z punktu widzenia TCP lub UDP jeden z członków sieci P2P (serwer) oczekuje na połączenie, a drugi (klient) łączy się z nim.

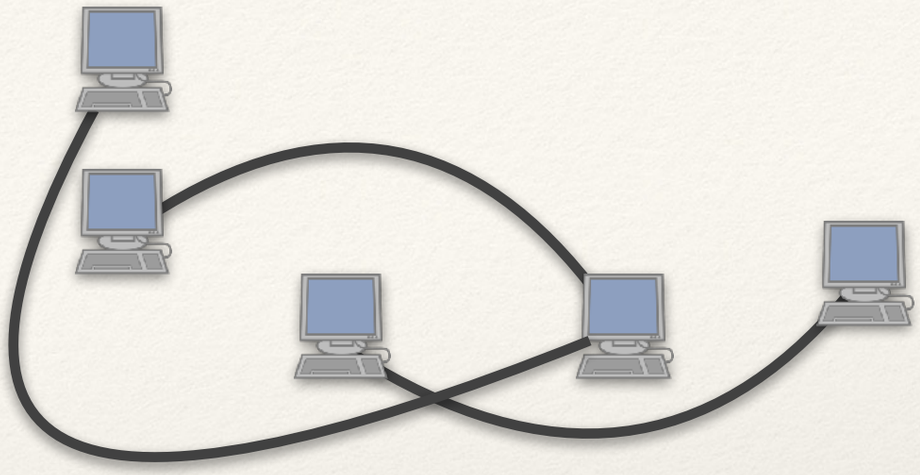
Przykład: BitTorrent



Podłączanie się do sieci.

- Łączymy się z trackerem.
- Tracker zna adresy członków sieci i udostępnia adresy niektórych (50-100).
- Po jakimś czasie możemy prosić o kolejne adresy.

BitTorrent: przesyłanie pliku (1)

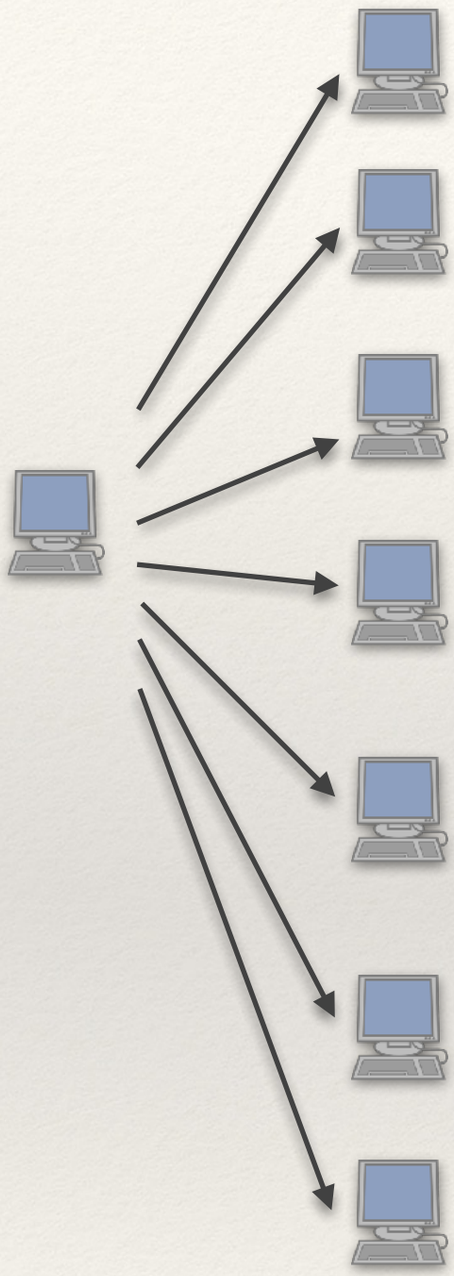


Plik dzielony na fragmenty.

- Każdy fragment pobierany niezależnie.
- Rozmiar fragmentu = ok. 256 KB - 16 MB.
  - Duży → żeby okno TCP miało czas urosnąć.
  - Mały → żeby plik miał wiele fragmentów (urównoleglenie).

- **Seeder** = ma wszystkie fragmenty.
- **Leecher** = ma niektóre fragmenty.

Jak rozpowszechnić jeden fragment? (1)



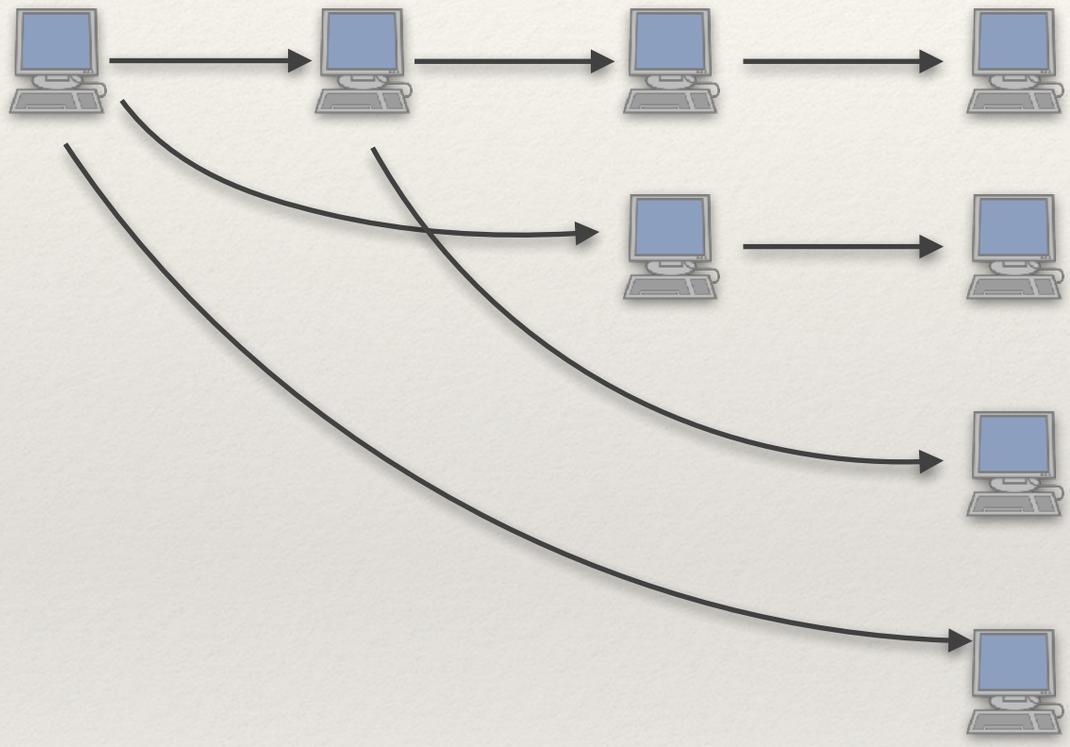
1. Minimalne opóźnienie, maksymalne obciążenie pojedynczego członka sieci.

(Jak w modelu klient-server).

Jak rozpowszechnić jeden fragment? (2)

2. Minimalne obciążenie pojedynczego członka sieci, duże opóźnienie.

Jak rozpowszechnić jeden fragment? (3)



3. Rozwiązanie pośrednie:
   - Logarytmiczna głębokość.
   - Logarytmiczne obciążenie pojedynczych wierzchołków.
   - Duża odporność na opuszczanie sieci przez komputery.

BitTorrent: przesyłanie pliku (2)

- Seeder udostępnia fragmenty innym członkom sieci.

- Leecher ma listę P klientów, którym udostępnia fragmenty.
  - To klienci, którzy wysyłają mu swoje fragmenty najszybciej.
  - Eksploracja: czasem daje fragment losowemu członkowi sieci. (Może prześle posiadany fragment odpowiednio szybko?)
  - Jeśli klient mówi, że jest nowy, to dostaje fragment za darmo.

- Klienci zazwyczaj chcą fragmenty występujące najrzadziej w sieci.

BitTorrent: przesyłanie pliku (2)

- Seeder udostępnia fragmenty innym członkom sieci.

- Leecher ma listę P klientów, którym udostępnia fragmenty.
  - To klienci, którzy wysyłają mu swoje fragmenty najszybciej.
  - Eksploracja: czasem daje fragment losowemu członkowi sieci. (Może prześle posiadany fragment odpowiednio szybko?)
  - Jeśli klient mówi, że jest nowy, to dostaje fragment za darmo.

animacja

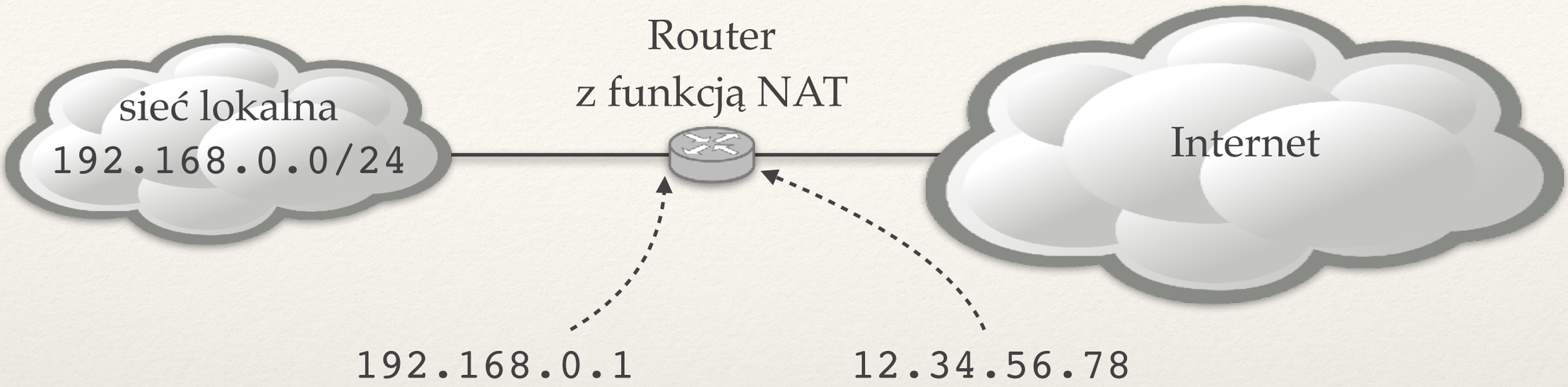
- Klienci zazwyczaj chcą fragmenty występujące najrzadziej w sieci.

BitTorrent: metadane

- Z plikiem X związany jest plik `.torrent`, umieszczany na WWW.
- Zawiera IP trackera.
- Zawiera funkcje skrótu dla wszystkich fragmentów
  → umożliwia sprawdzenie, czy pobraliśmy dobry fragment.

NAT a warstwa aplikacji

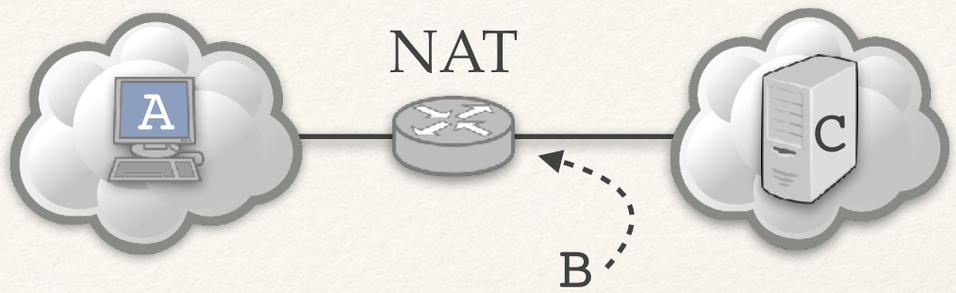
NAT



- Bardzo powszechne rozwiązanie.
- Z reszty Internetu cała sieć lokalna wygląda tak samo, jak pojedynczy komputer z adresem `12.34.56.78`.

Co robi router z funkcją NAT?

- **Pakiet**
  - Z adresu i portu (A, P_A).
  - Do adresu i portu (C, P_C).
  - NAT na podstawie krotki (A, P_A, C, P_C) wybiera port P_B.
  - Adres i port źródłowy pakietu podmienione na (B, P_B).

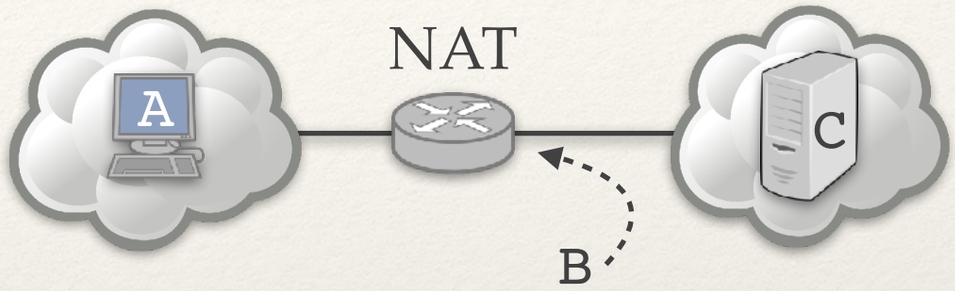


- **Tablica NAT:**
  - Przechowuje przez pewien czas przypisanie (A, P_A, C, P_C) → P_B.
  - **Dla kolejnych podobnych pakietów przypisanie będzie takie samo.**
  - Jeśli przychodzi odpowiedź do (B, P_B) to jej adres i port docelowy zostanie podmieniony na (A, P_A).

NAT a P2P

Kiedyś:

- Komunikacja zawsze w modelu klient-serwer.
- Serwery nie są za routerami z NAT.
- Klienci mogą być za routerami z NAT.

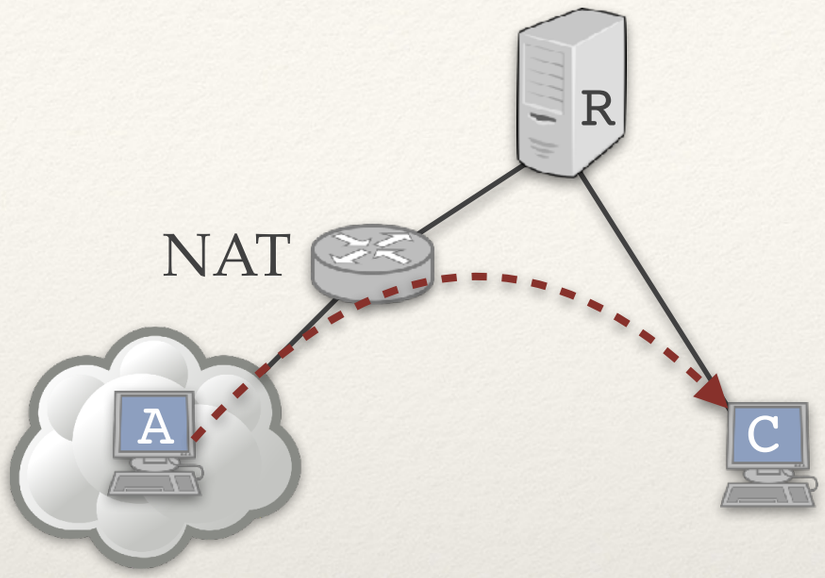


- Pierwszy pakiet (np. segment TCP SYN) od klienta do serwera tworzy przypisanie $(A, P_A, C, P_C) \rightarrow P_B$. Pakiety z odpowiedziami serwera mogą wracać do klienta.

Obecnie:

- Chcemy przesyłać dane w modelu peer-to-peer.
- Obie strony komunikacji są często za NAT.
- Brak naturalnej możliwości zainicjowania połączenia.

Odwrócone połączenie



- C chce się połączyć z A, ale A jest za NAT.
- Jeśli oba utrzymują kontakt z R, to C może poprosić (przez R) komputer A o zainicjowanie połączenia (do C).

Odwrócone połączenie w FTP

FTP: protokół przesyłania plików.

- A łączy się z portem 21 serwera C (połączenie na komunikaty kontrolne).

- A wysyła polecenie „chcę pobrać plik i słucham na porcie X“
  - C łączy się z portem X klienta A i wysyła plik (odrębne połączenie TCP).
  - Połączenie odrzucane przez NAT.

Odwrócone połączenie w FTP

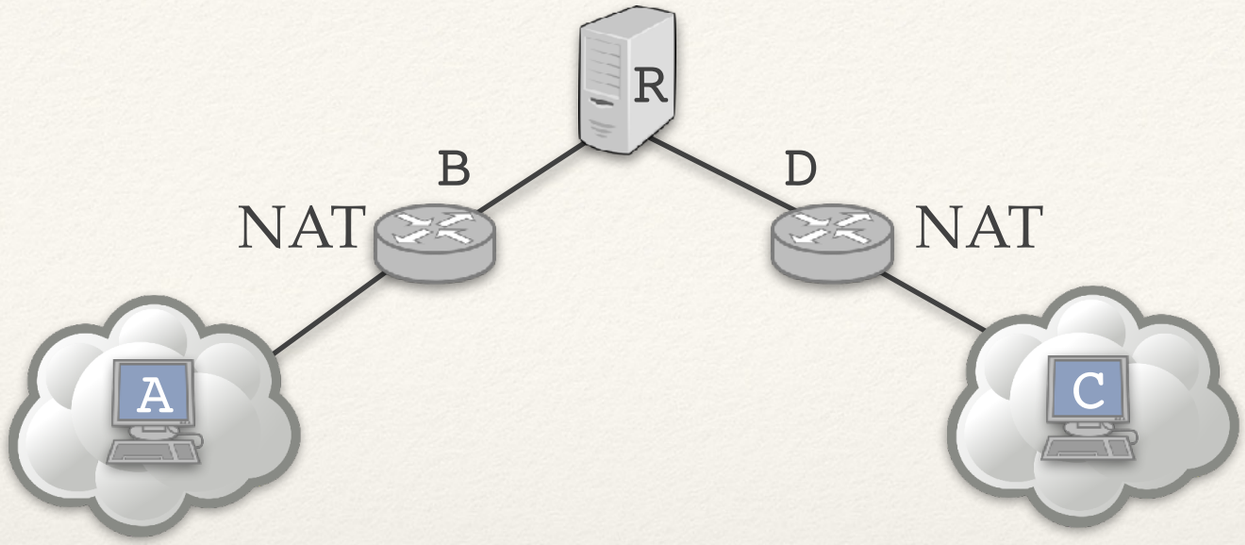
FTP: protokół przesyłania plików.

- A łączy się z portem 21 serwera C (połączenie na komunikaty kontrolne).

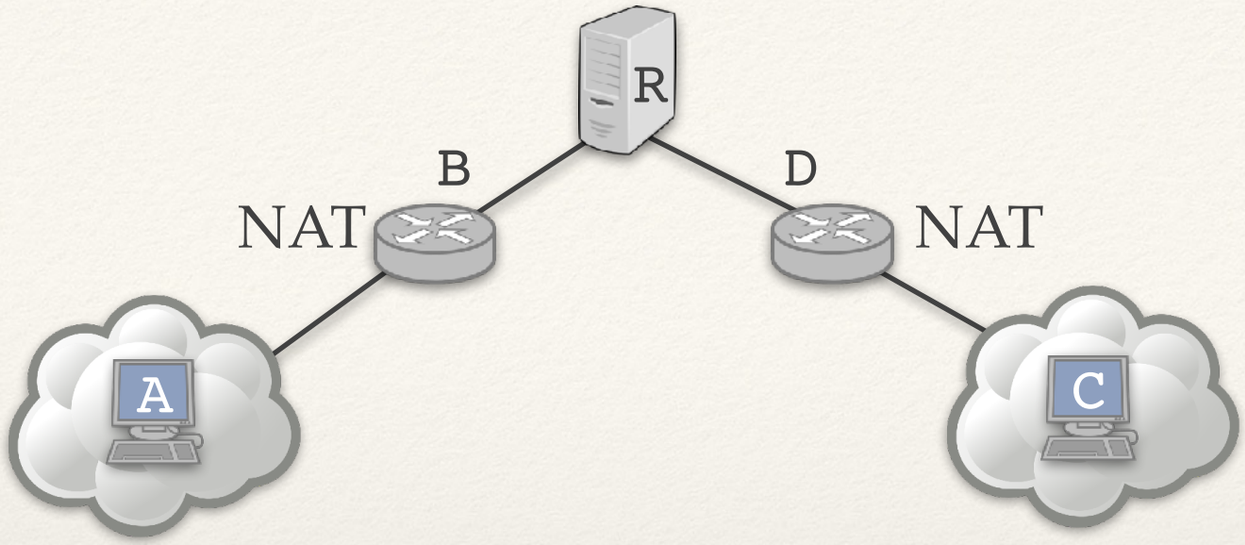
- A wysyła polecenie „chcę pobrać plik i słucham na porcie X“
  - C łączy się z portem X klienta A i wysyła plik (odrębne połączenie TCP).
  - Połączenie odrzucane przez NAT.

- **Tryb pasywny FTP:** A wysyła polecenie „chce pobrać plik w trybie pasywnym“.
  - C zaczyna słuchać na porcie Y i wysyła komunikat „słucham na porcie Y“.
  - A łączy się z portem Y serwera C i pobiera plik.

Przechodzenie przez NAT (1)

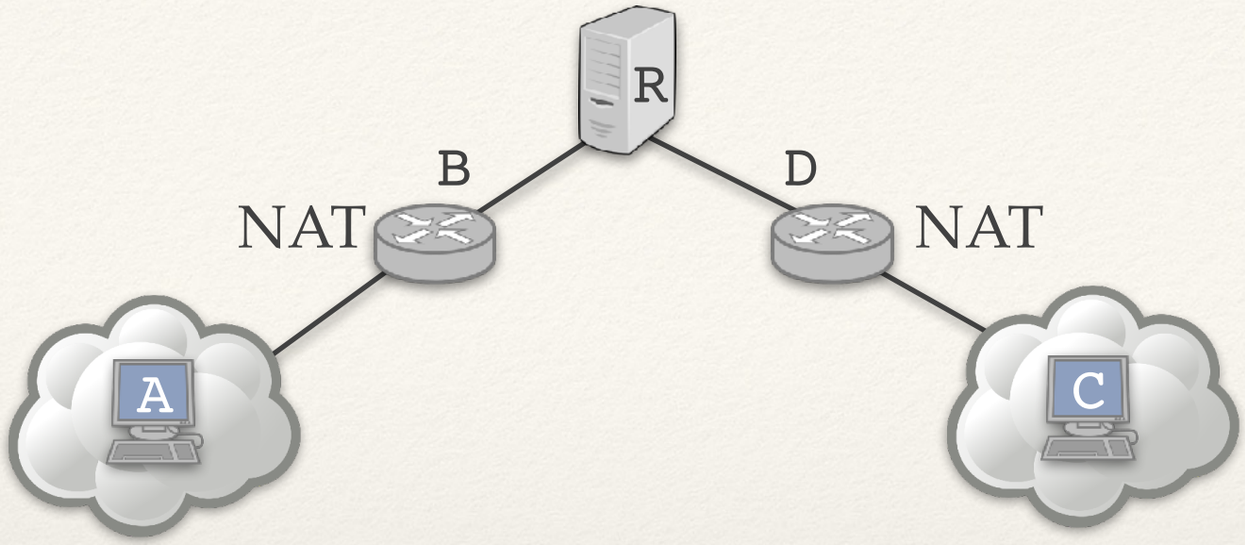


Przechodzenie przez NAT (1)



- A wysyła z portu P_A pakiet do R o treści "(A, P_A)".
- Na routerze NAT zostaje utworzone przypisanie (A, P_A) → (B, P_B).
- R widzi pakiet o treści "(A, P_A)" od (B, P_B), tj. poznaje przypisanie wygenerowane przez B.

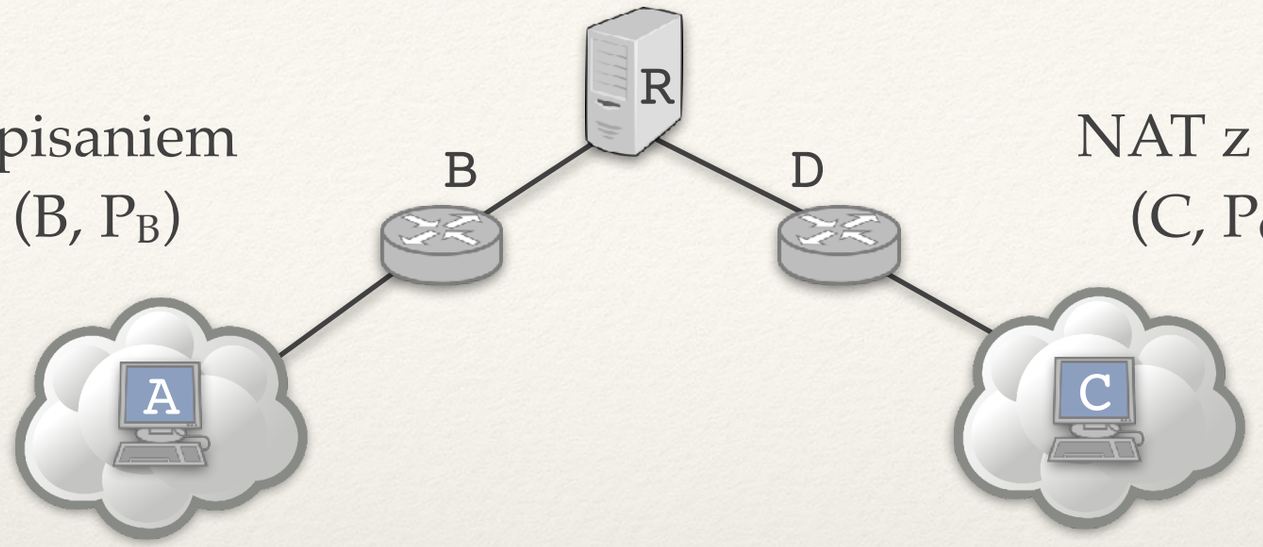
Przechodzenie przez NAT (1)



- A wysyła z portu P_A pakiet do R o treści "(A, P_A)".
- Na routerze NAT zostaje utworzone przypisanie (A, P_A) → (B, P_B).
- R widzi pakiet o treści "(A, P_A)" od (B, P_B), tj. poznaje przypisanie wygenerowane przez B.
- W taki sam sposób R poznaje przypisanie (C,P_C) → (D,P_D).
- R odsyła poznane przypisania do A i C.

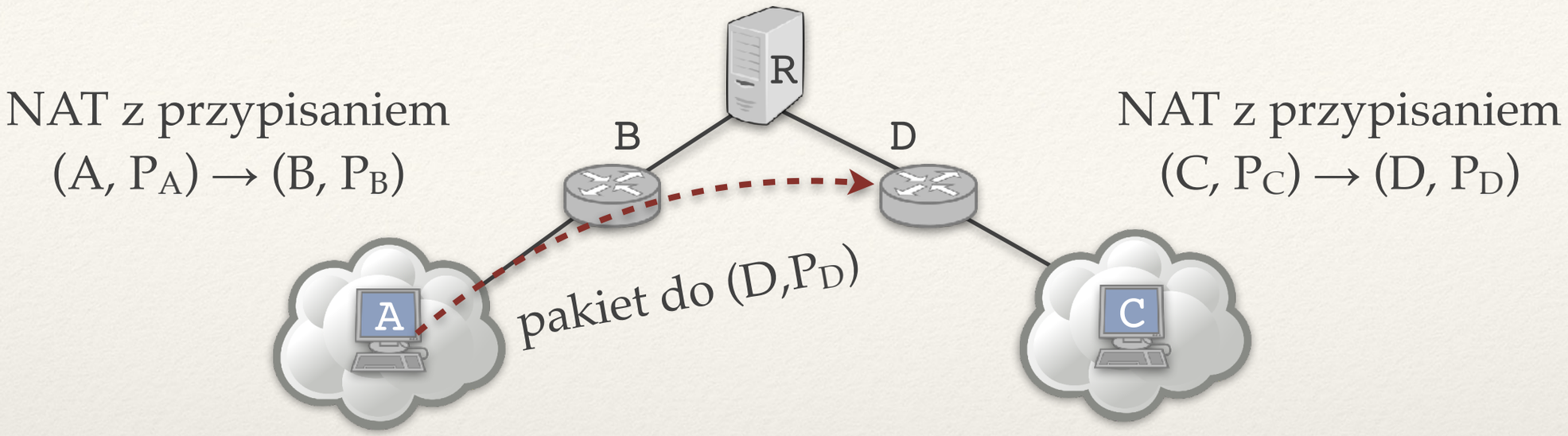
Przechodzenie przez NAT (2)

NAT z przypisaniem
$(A, P_A) \rightarrow (B, P_B)$



NAT z przypisaniem
$(C, P_C) \rightarrow (D, P_D)$

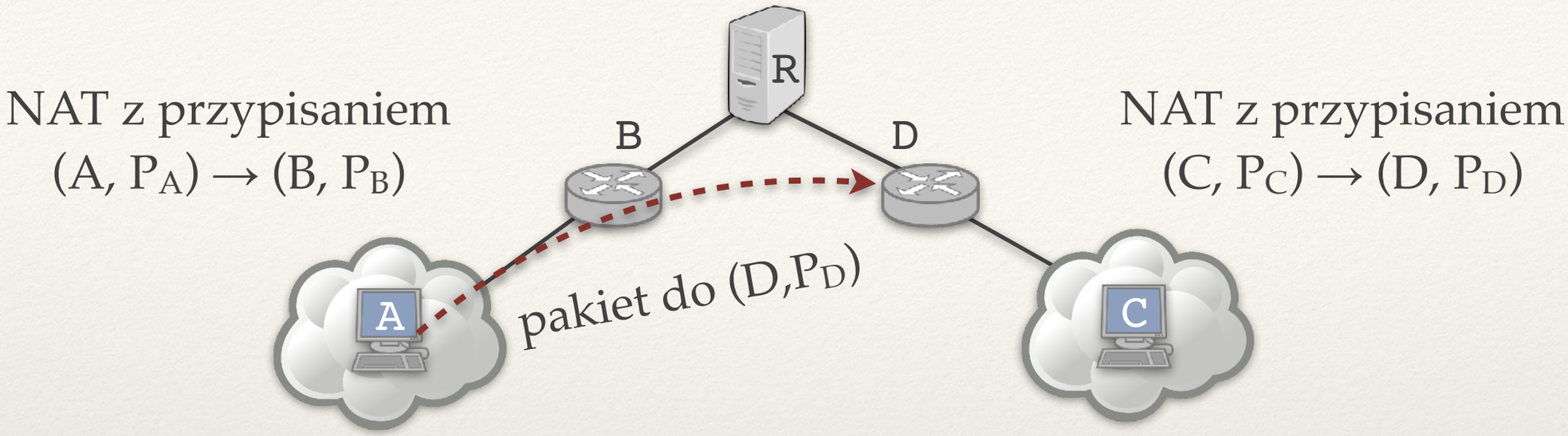
Przechodzenie przez NAT (2)



Komunikacja:

- A adresuje pakiety do (D,P_D). Przychodzą one do C jako pakiety od (B,P_B).
- C adresuje pakiety do (B,P_B). Przychodzą one do A jako pakiety od (D,P_D).

Przechodzenie przez NAT (2)



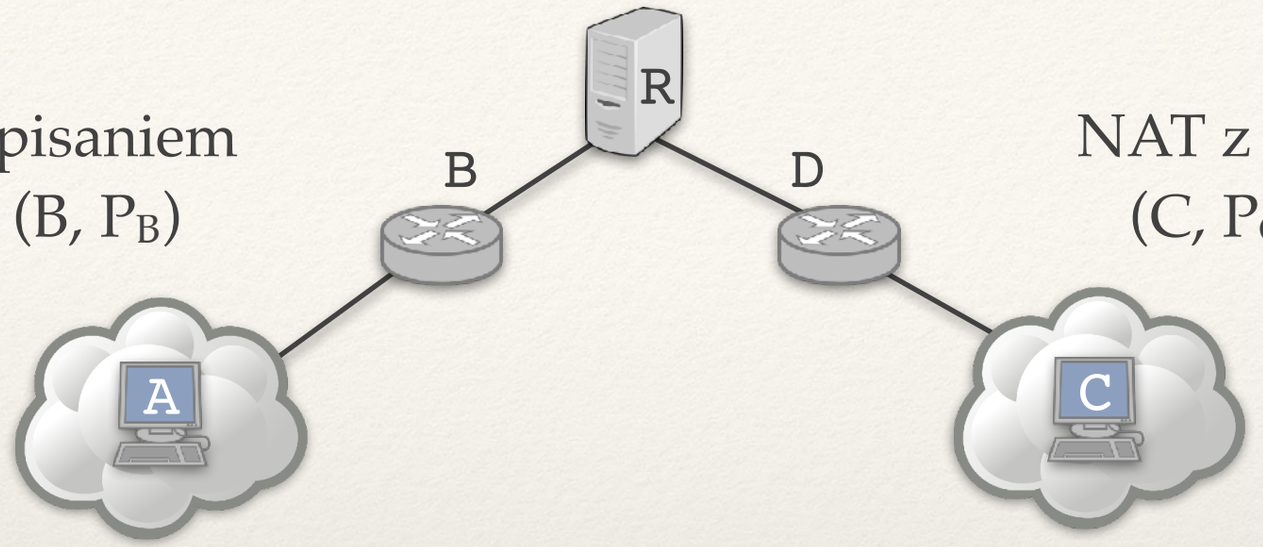
Komunikacja:

- A adresuje pakiety do (D,P_D). Przychodzą one do C jako pakiety od (B,P_B).
- C adresuje pakiety do (B,P_B). Przychodzą one do A jako pakiety od (D,P_D).

Ale nie każdy NAT przekaże pakiet od A do (D, P_D) do komputera (C, P_C)!

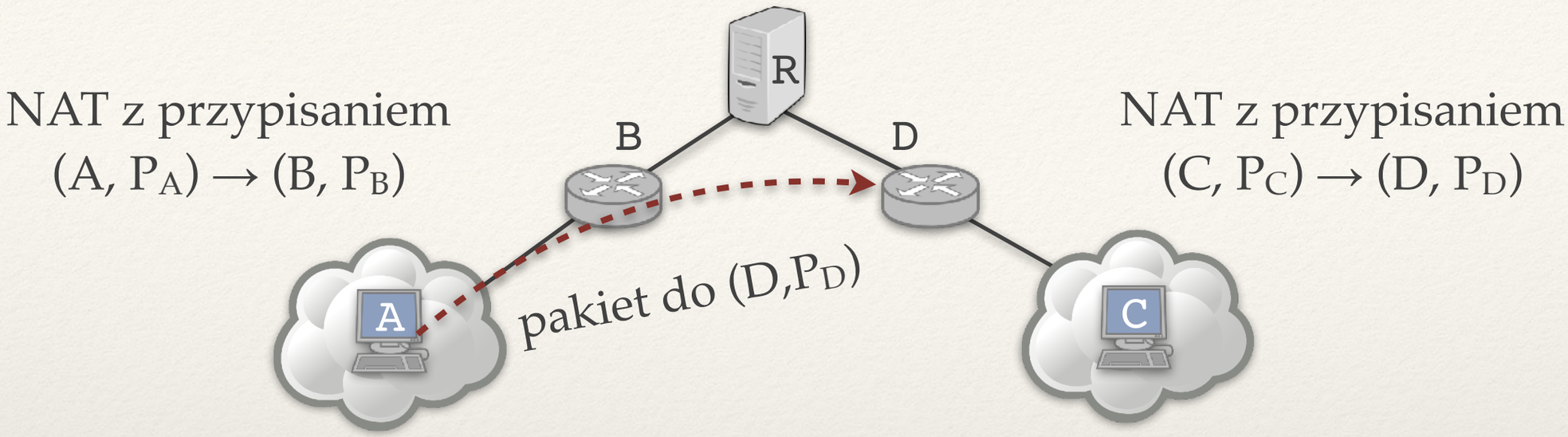
Przechodzenie przez NAT (2)

NAT z przypisaniem
$(A, P_A) \rightarrow (B, P_B)$



NAT z przypisaniem
$(C, P_C) \rightarrow (D, P_D)$

Przechodzenie przez NAT (2)

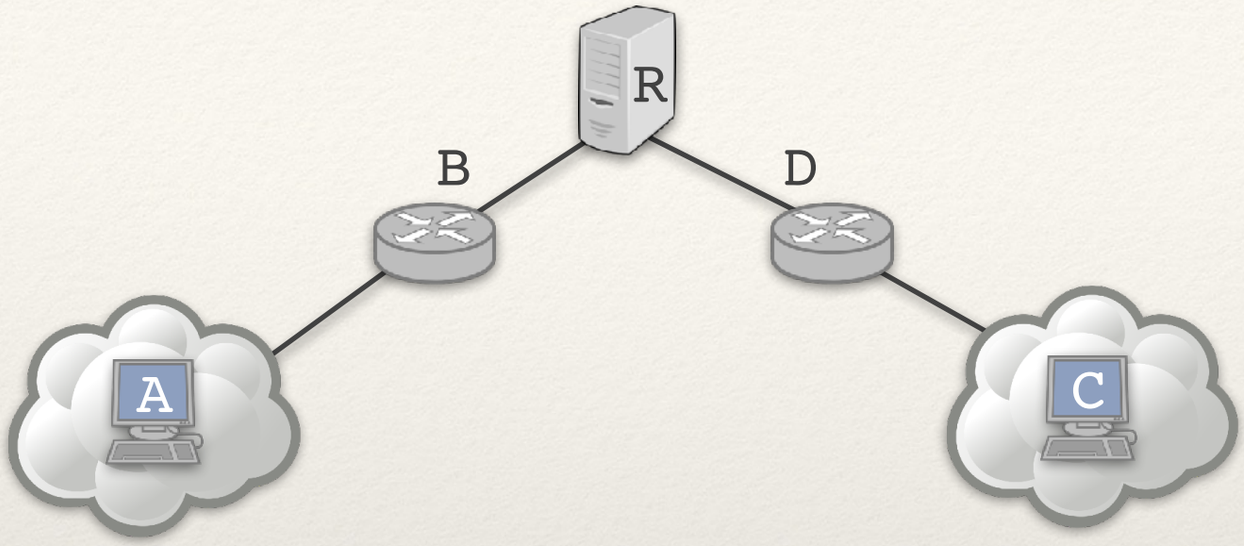


Czy D przekaże pakiety skierowane do (D, P_D) do komputera (C, P_C)?

- Tak, jeśli D jest pełnym asymetrycznym NAT (*full cone* NAT).
- Tak, jeśli D jest ograniczonym asymetrycznym NAT (*restricted cone* NAT) i **D kiedyś wysyłał do nadawcy pakiety z portu P_D**.
  - Pakiety od R przyjdą.
  - Pakiety od A (widziane jako pakiety od (B, P_B)) zostaną odrzucone.

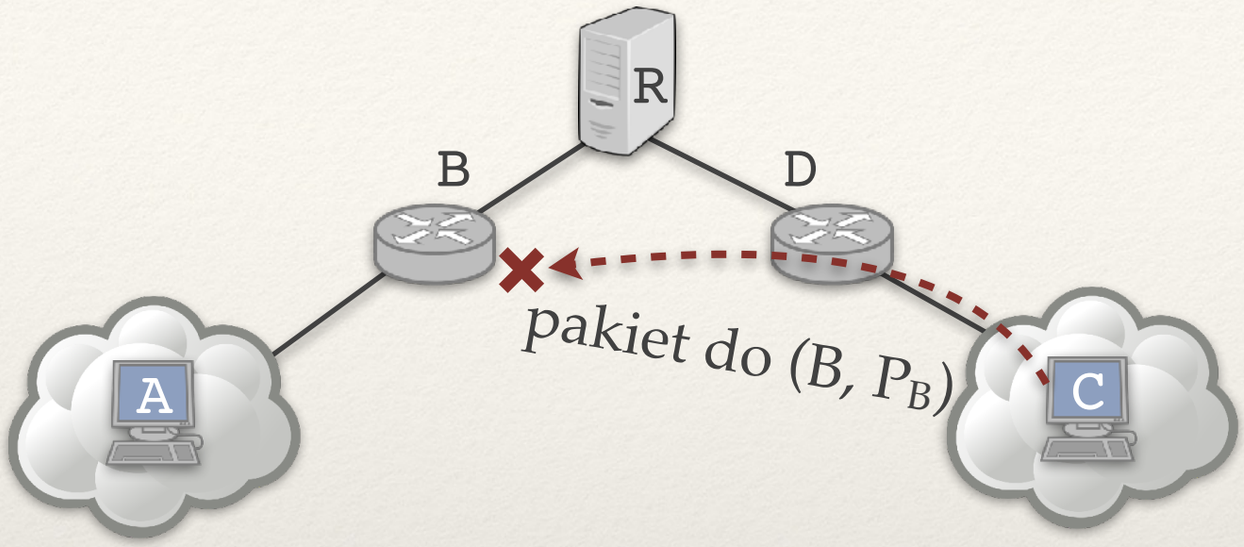
Wybijanie dziur (*hole punching*)

Jak poradzić sobie z asymetrycznymi ograniczonymi NAT?



Wybijanie dziur (*hole punching*)

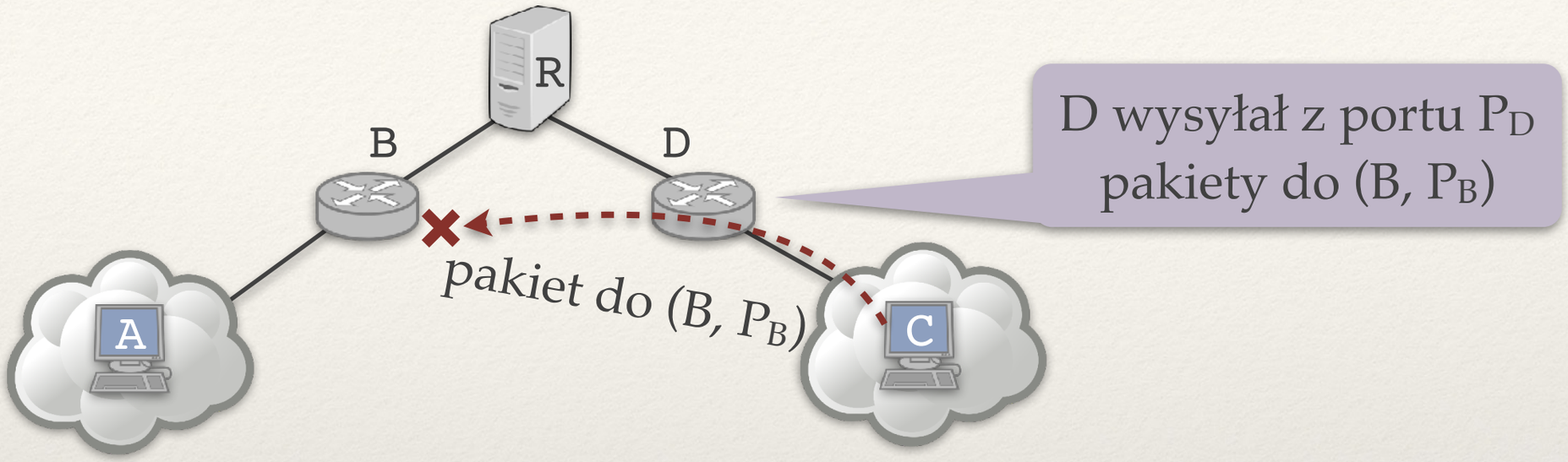
Jak poradzić sobie z asymetrycznymi ograniczonymi NAT?



- (C, P_C) wysyła pakiet do (B, P_B). B odrzuca ten pakiet.

Wybijanie dziur (*hole punching*)

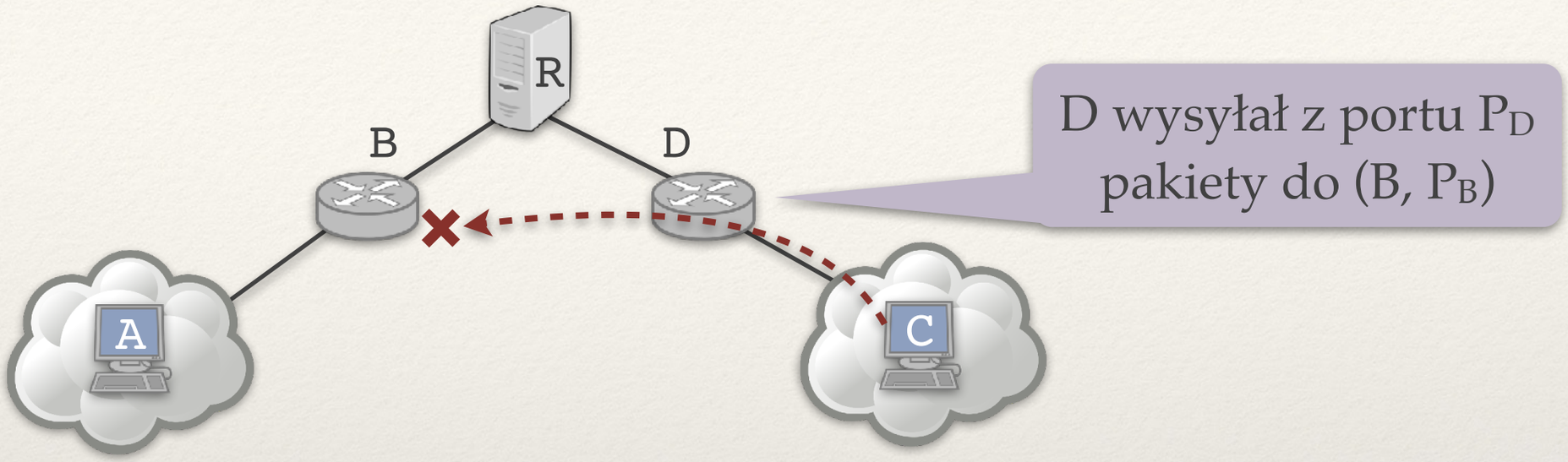
Jak poradzić sobie z asymetrycznymi ograniczonymi NAT?



- (C, P_C) wysyła pakiet do (B, P_B). B odrzuca ten pakiet.

Wybijanie dziur (*hole punching*)

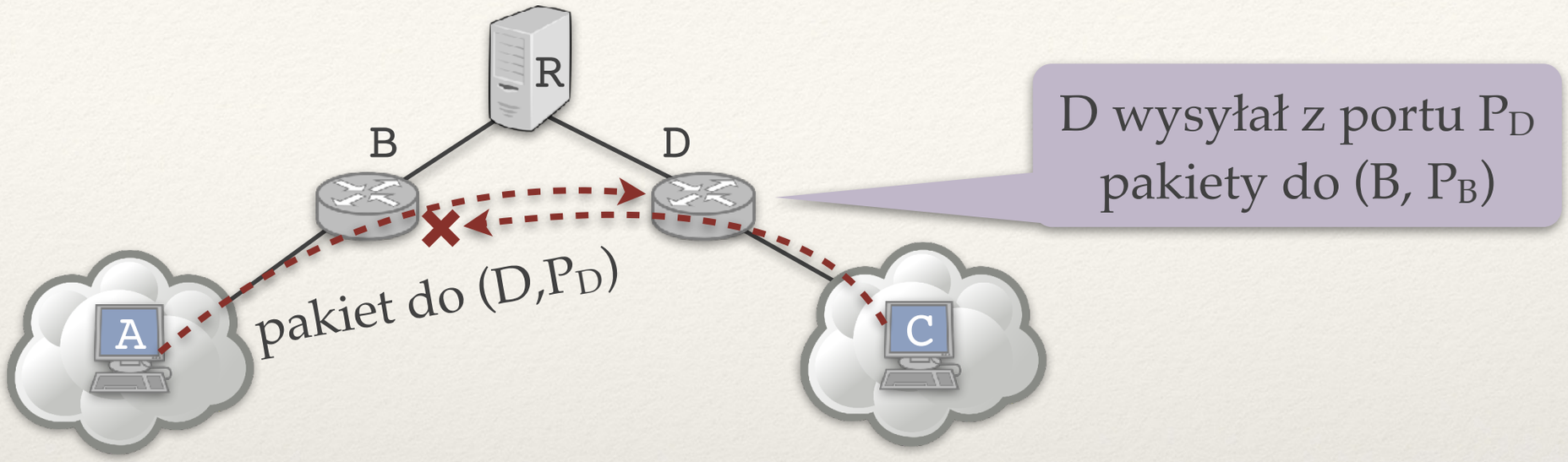
Jak poradzić sobie z asymetrycznymi ograniczonymi NAT?



- (C, P_C) wysyła pakiet do (B, P_B). B odrzuca ten pakiet.

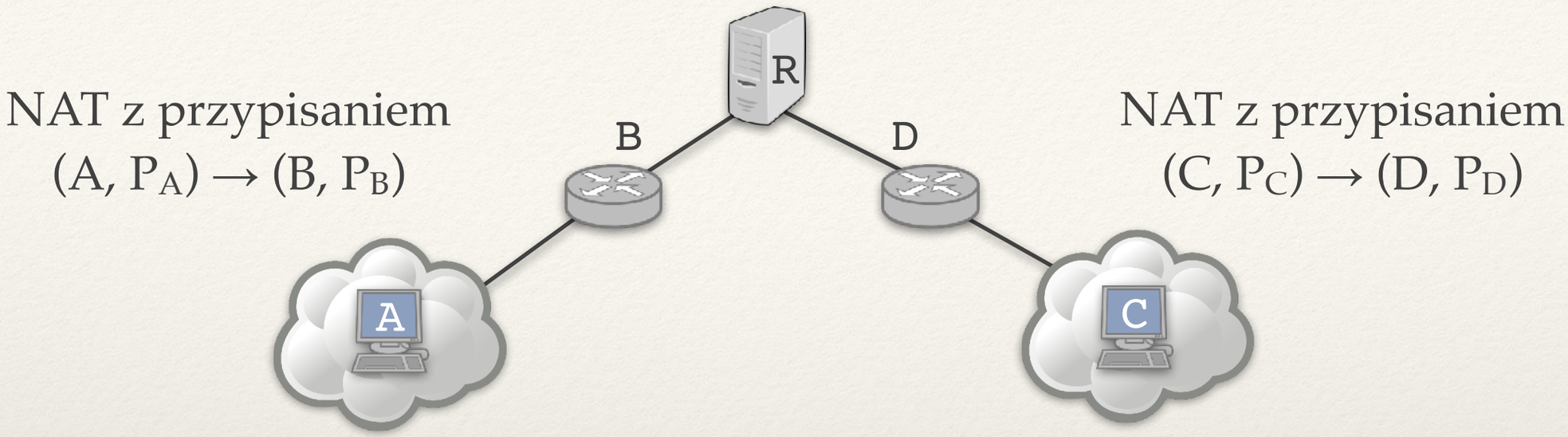
Wybijanie dziur (*hole punching*)

Jak poradzić sobie z asymetrycznymi ograniczonymi NAT?



- (C, P_C) wysyła pakiet do (B, P_B). B odrzuca ten pakiet.
- (A, P_A) wysyła pakiet do (D, P_D):
  - adres źródłowy zostaje podmieniony na (B, P_B)
  - D przepuszcza pakiet „od (B, P_B)“ do (C, P_C).

NAT symetryczny



- **NAT asymetryczny na B:**
  - Wybrany port P_B zależy tylko od nadawcy.
  - W przypadku komunikacji $(A,P_A) \rightarrow R$ oraz $(A,P_A) \rightarrow D$ wybierany jest ten sam port P_B.
  - Kluczowe założenie w mechanizmie wybijania dziur.

- **NAT symetryczny na B:** P_B zależy od adresu i portu nadawcy i odbiorcy.

Lektura dodatkowa

- Kurose & Ross: rozdział 2.
- Tanenbaum: rozdział 7.
- Zawartość strefy `.` : `https://www.internic.net/domain/root.zone`
- NAT: `https://en.wikipedia.org/wiki/Network_address_translation`

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`?
- Do czego służy protokół SMTP a do czego IMAP?
- Co to są przekaźniki SMTP (*relays*)?
- Jaki rekord DNS jest sprawdzany przed wysłaniem poczty do danej domeny?
- Wymień parę popularnych pól w nagłówku maila. Do czego służą pola `Received` i `Bcc`?
- Co umożliwia standard MIME?
- Co to jest spam? Jakie znasz metody walki ze spamem?
- Na czym polega mechanizm SPF?
- Jaka jest rola trackera w sieci Bittorrent?
- Po co w plikach `.torrent` stosuje się funkcje skrótu?
- Jakie są różnice w postępowaniu seedera i leechera w sieci BitTorrent?
- Na czym polegają połączenia odwrócone? Jak stosuje się je w protokole FTP?
- Opisz podobieństwa i różnice asymetrycznych (*cone)* NAT (pełnego i ograniczonego) i symetrycznych NAT.
- Opisz technikę wybijania dziur (*hole punching*) w NAT. Po co konieczny jest serwer pośredniczący?