

---

# Warstwa aplikacji

## część 2: inne zastosowania

Sieci komputerowe

Wykład 10

---

*Marcin Bieńkowski*



# W dzisiejszym odcinku

---

- ❖ DNS
- ❖ Poczta elektroniczna.
- ❖ Sieci peer-to-peer.
- ❖ NAT (*network address translation*) a warstwa aplikacji.



---

# 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.



- ❖ 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

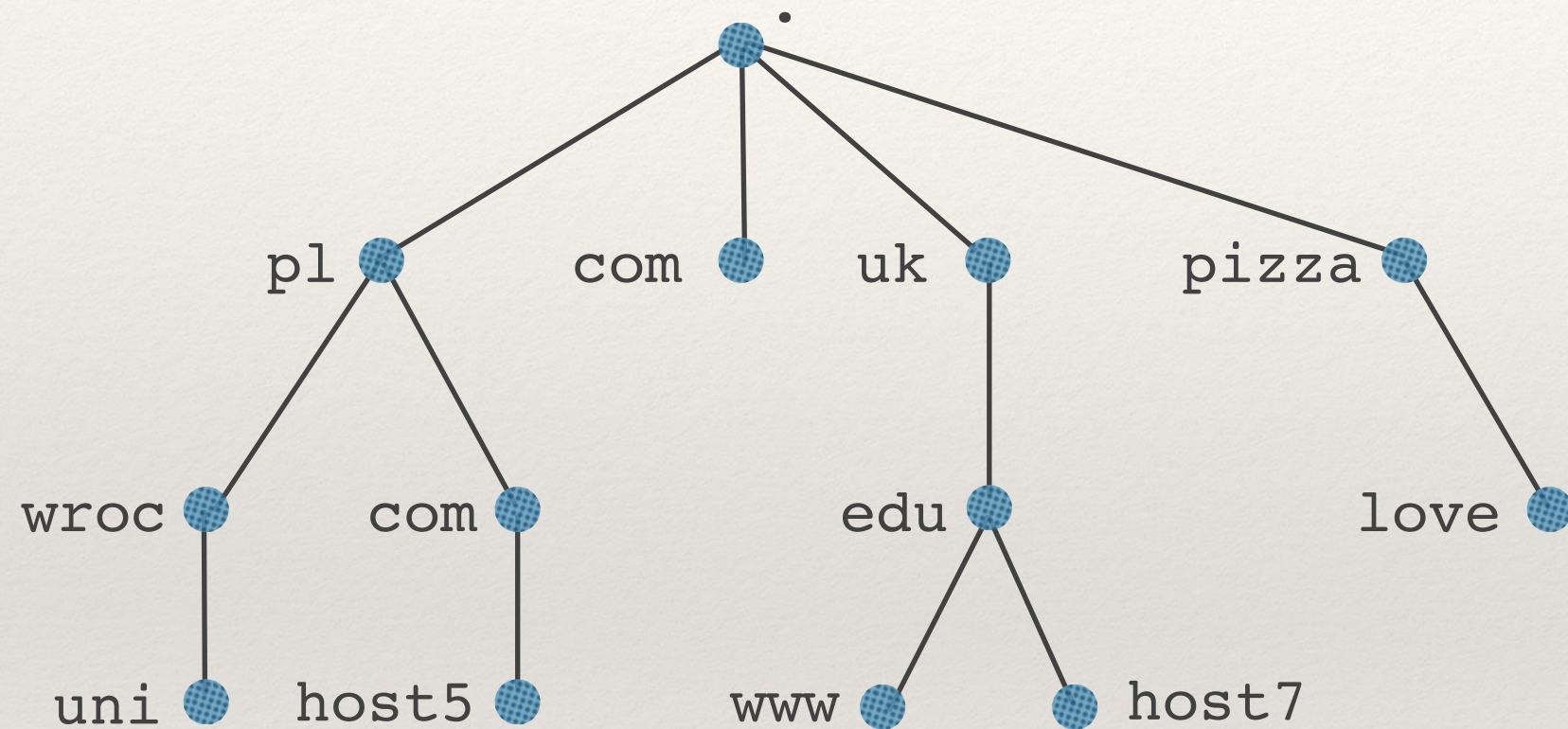
---

- ❖ 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

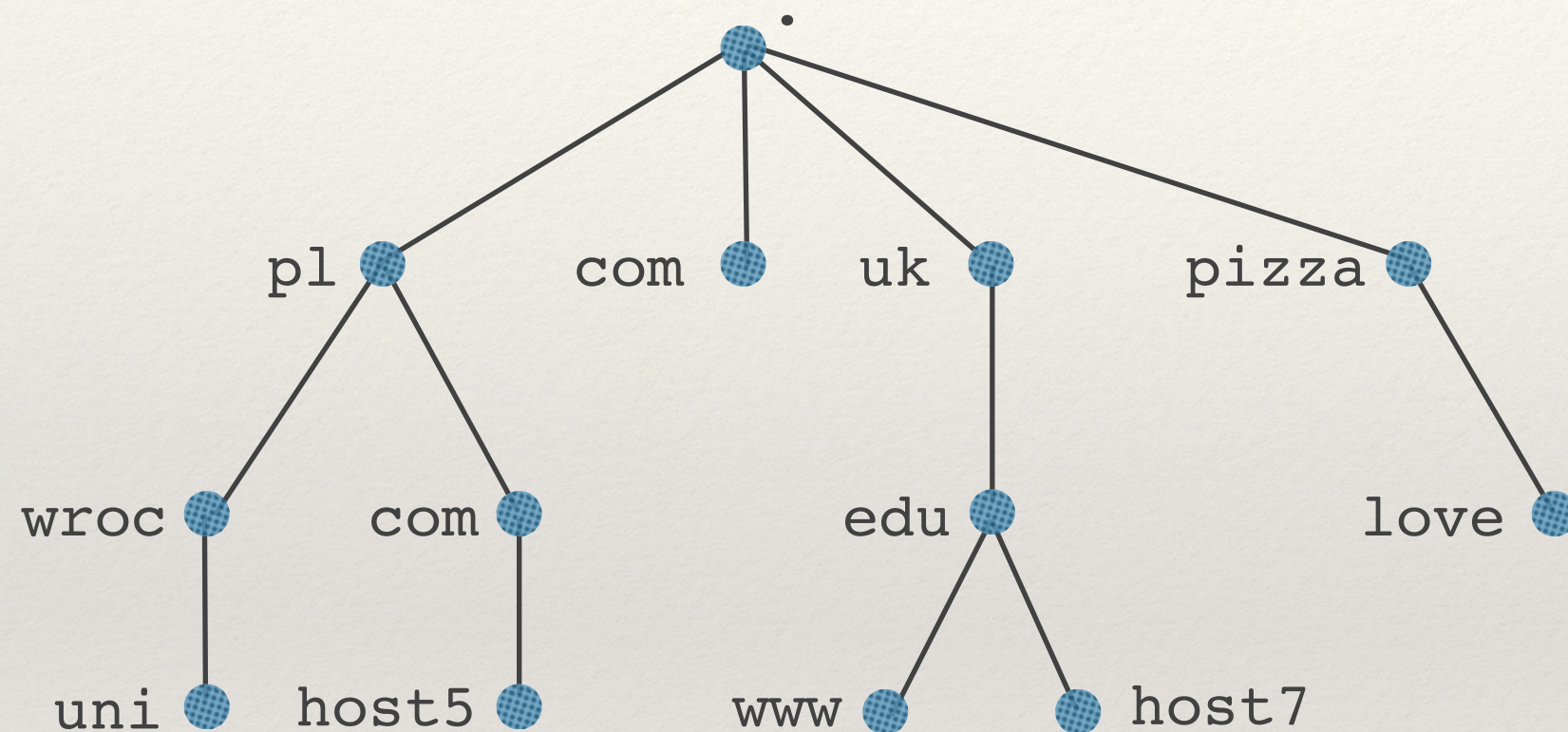
---





# Hierarchia nazw domen

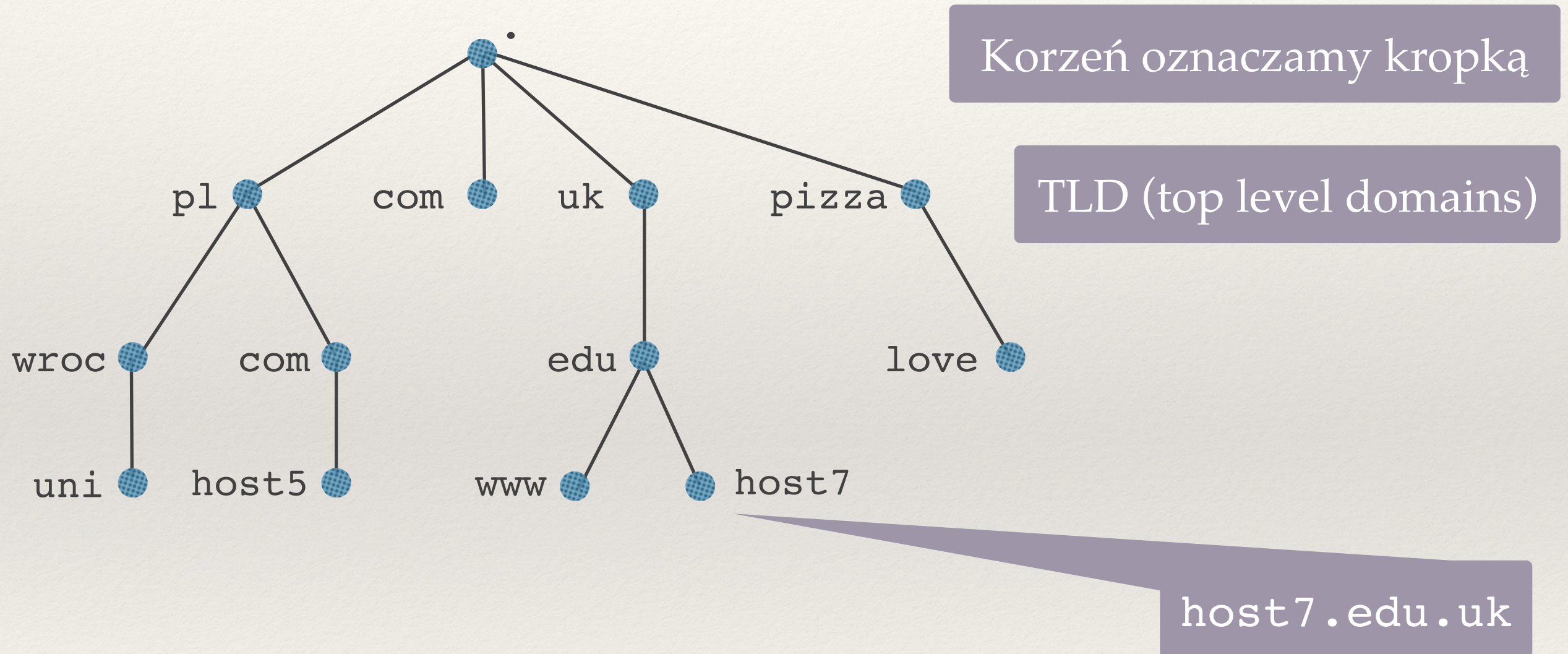
---



host7.edu.uk

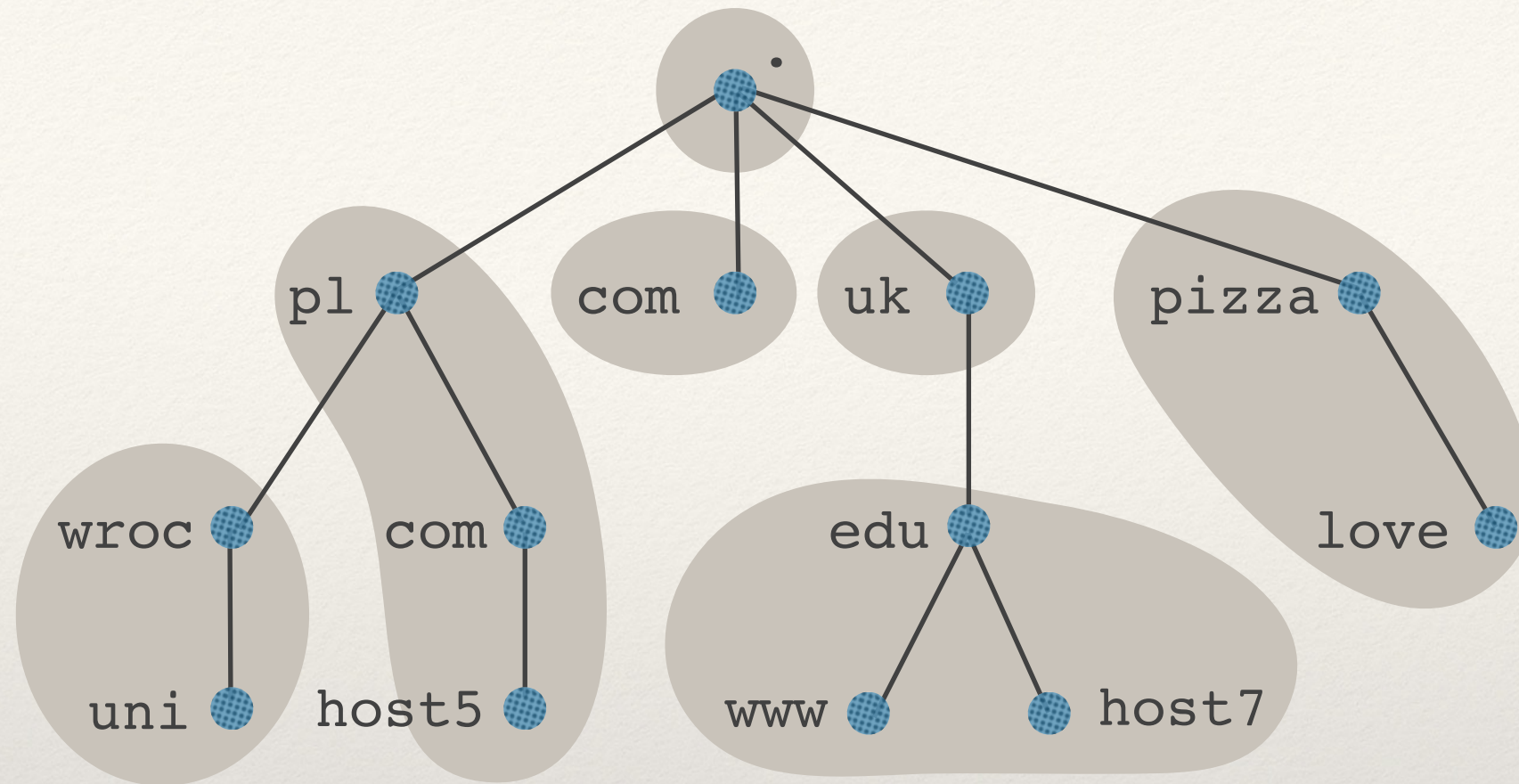


# Hierarchia nazw domen





# 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

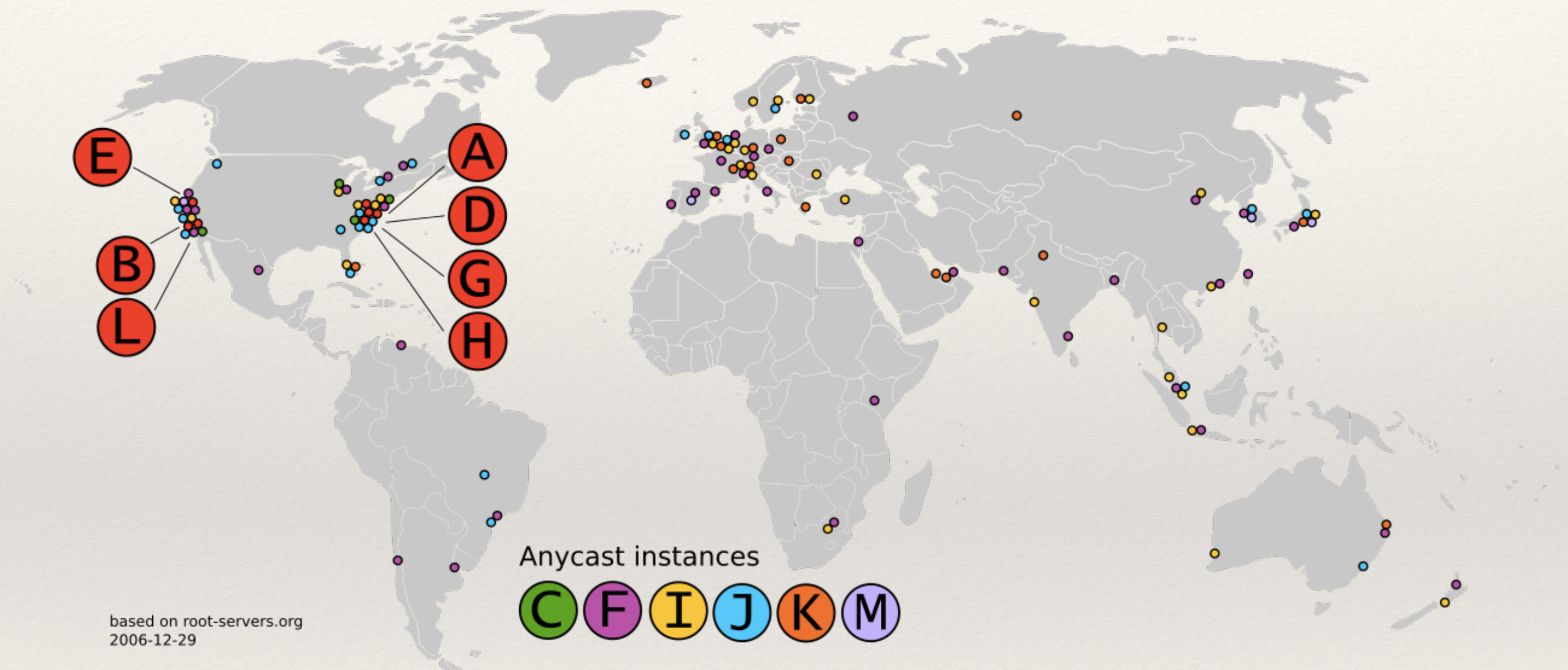
D.ROOT-SERVERS.NET = 128.8.10.90

...

Informacja wpisywana ręcznie  
(w standardowych plikach konfiguracyjnych).



# Serwery główne (2)



Obrazek ze strony [https://en.wikipedia.org/wiki/Root\\_name\\_server](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

---

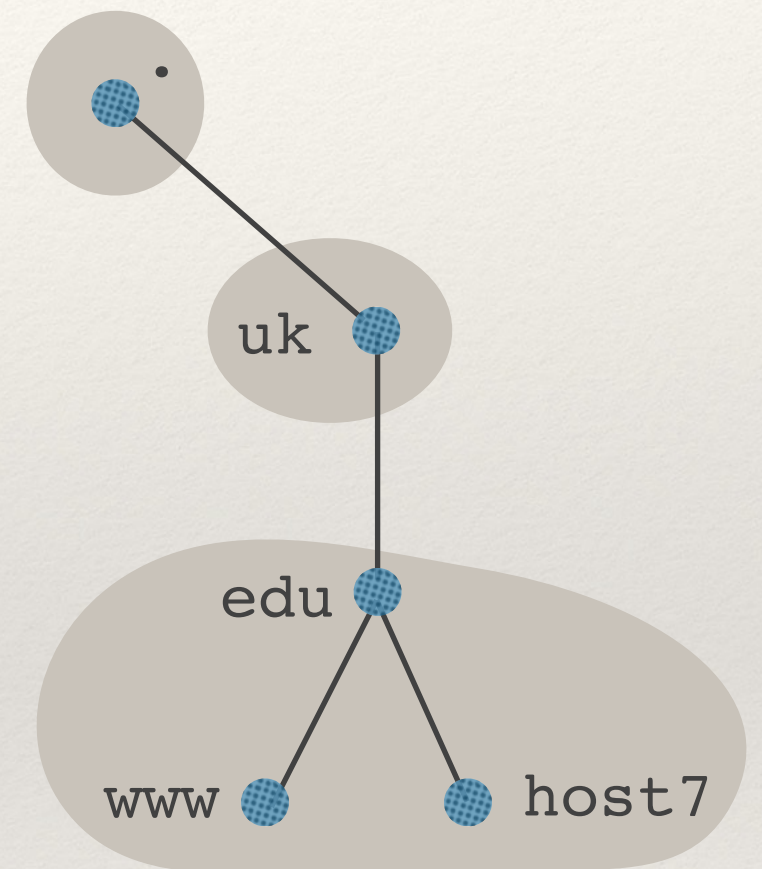
- ❖ **Adres anycast** = wiele serwerów ma ten sam adres IP
- ❖ Rozpowszechniany za pomocą standardowych protokołów routingu → routery poznają trasę do najbliższego serwera z danym adresem.
- ❖ Problem: wszystkie pakiety z danej komunikacji powinny być do jednego serwera.
  - ♦ najbliższy serwer może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



# Rozszyfrowywanie nazw (resolving)

---

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



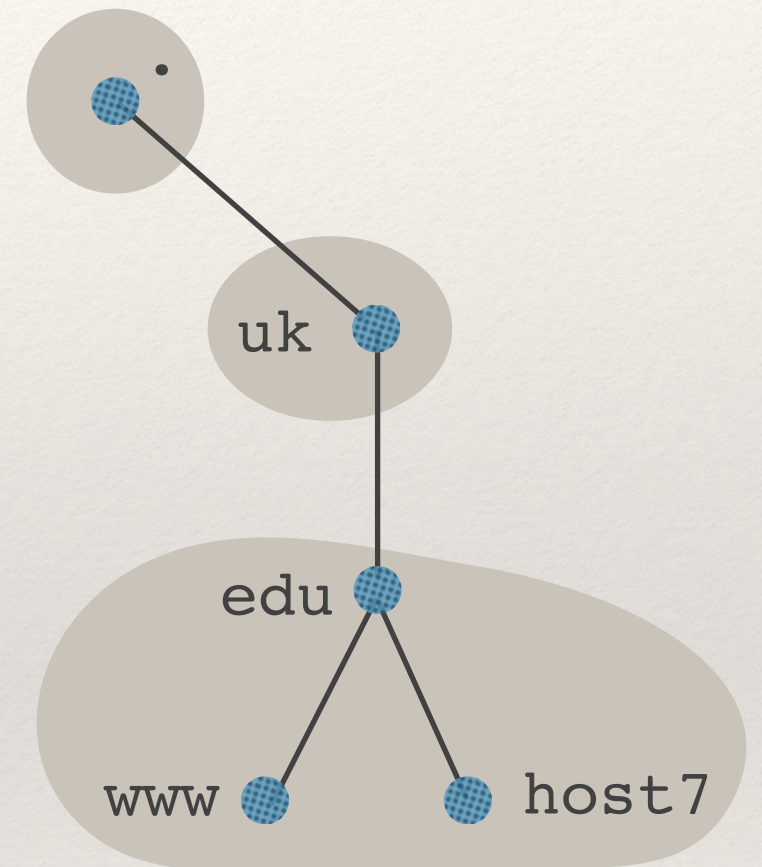


# Rozszyfrowywanie nazw (resolving)

---

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.

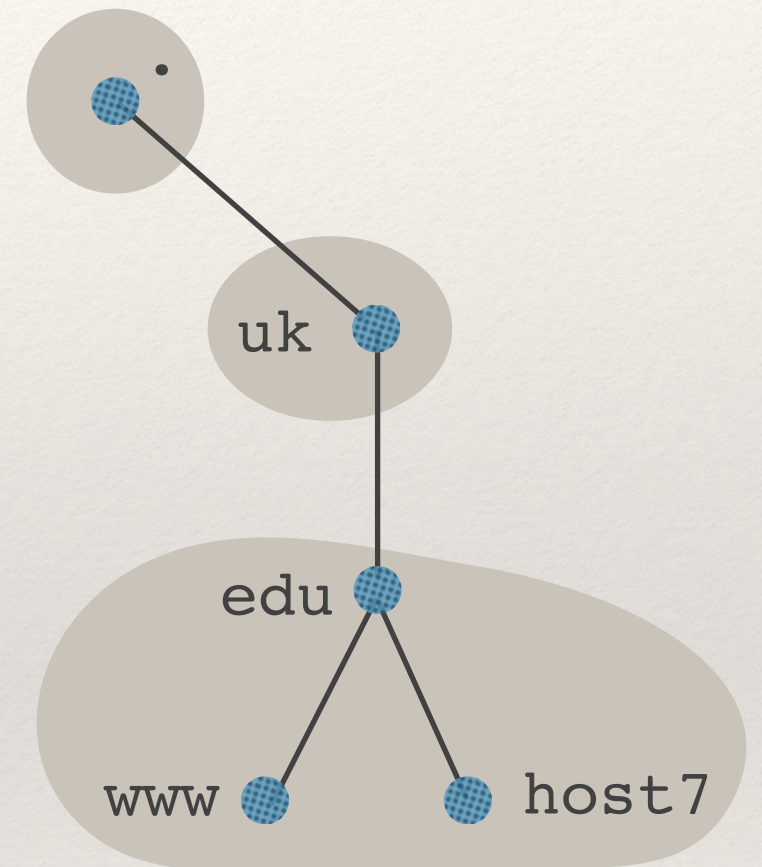




# Rozszyfrowywanie nazw (resolving)

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.

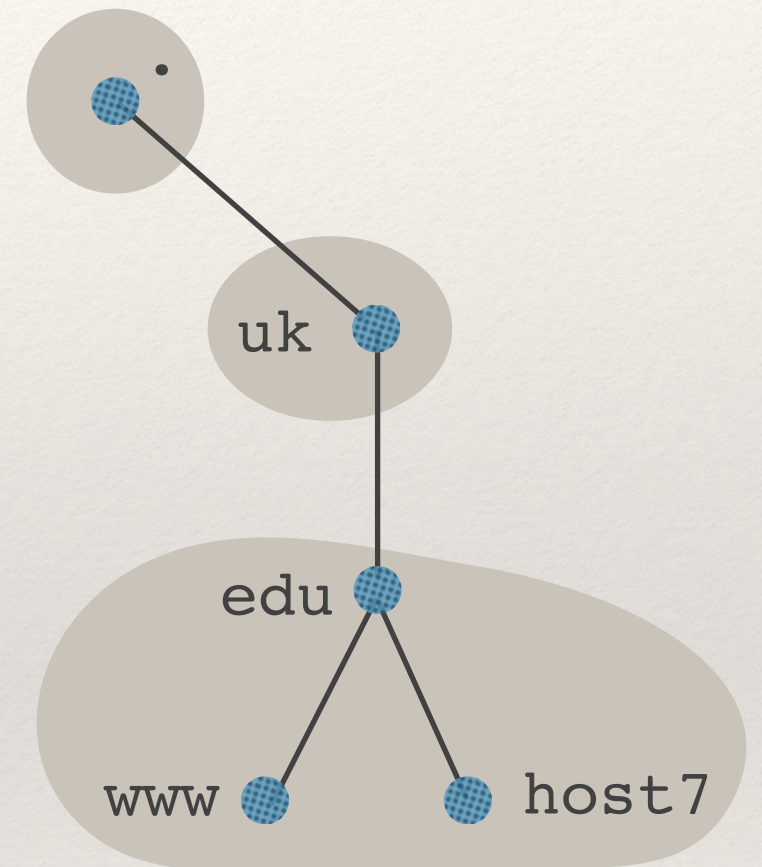




# Rozszyfrowywanie nazw (resolving)

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.

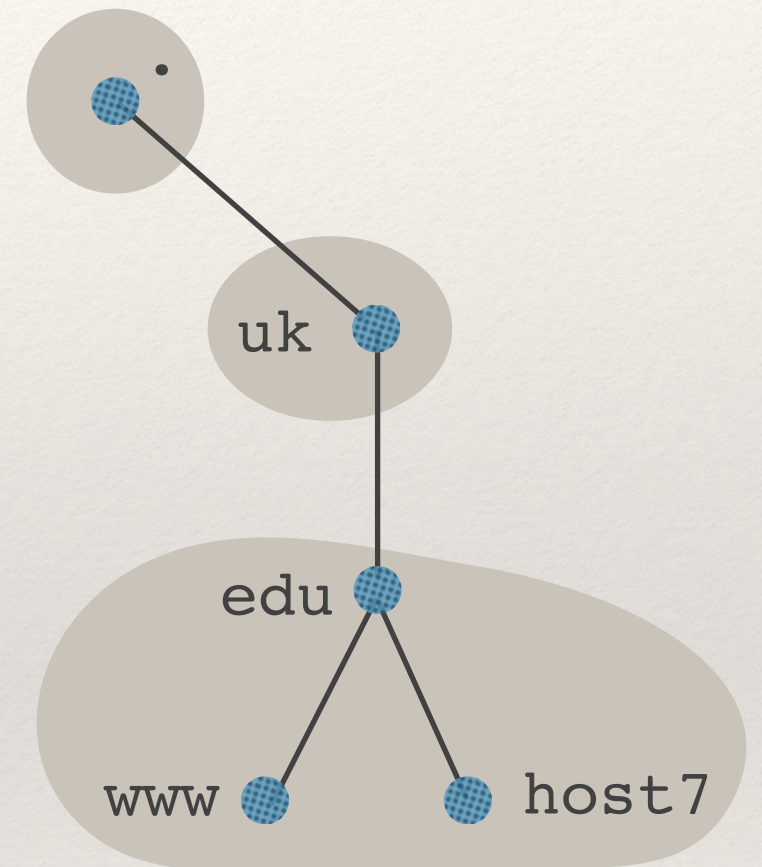




# Rozszyfrowywanie nazw (resolving)

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.

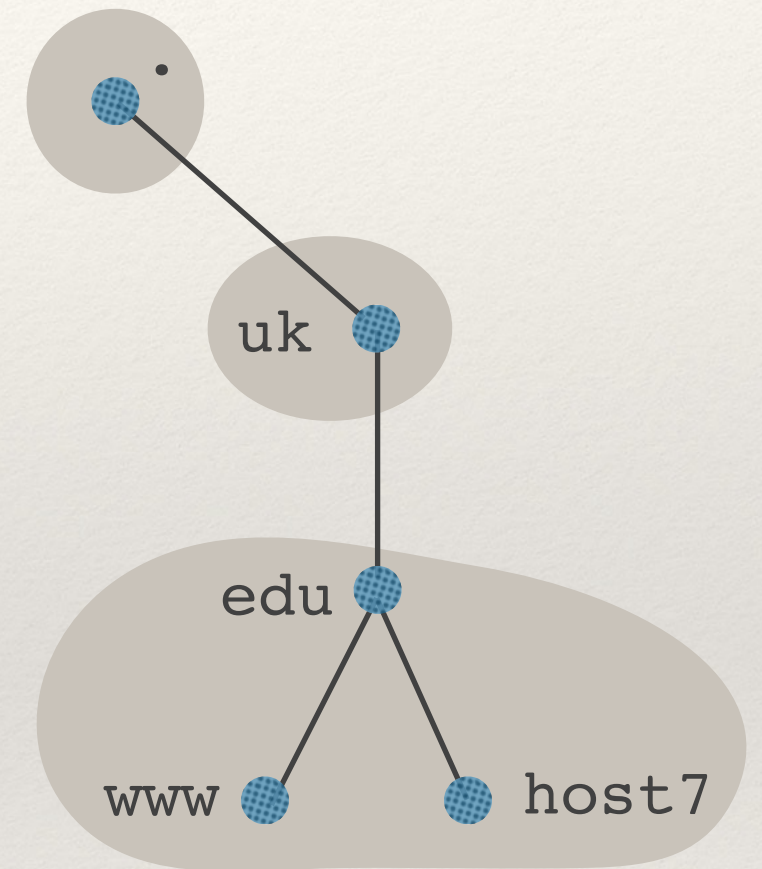




# Rozszyfrowywanie nazw (resolving)

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.

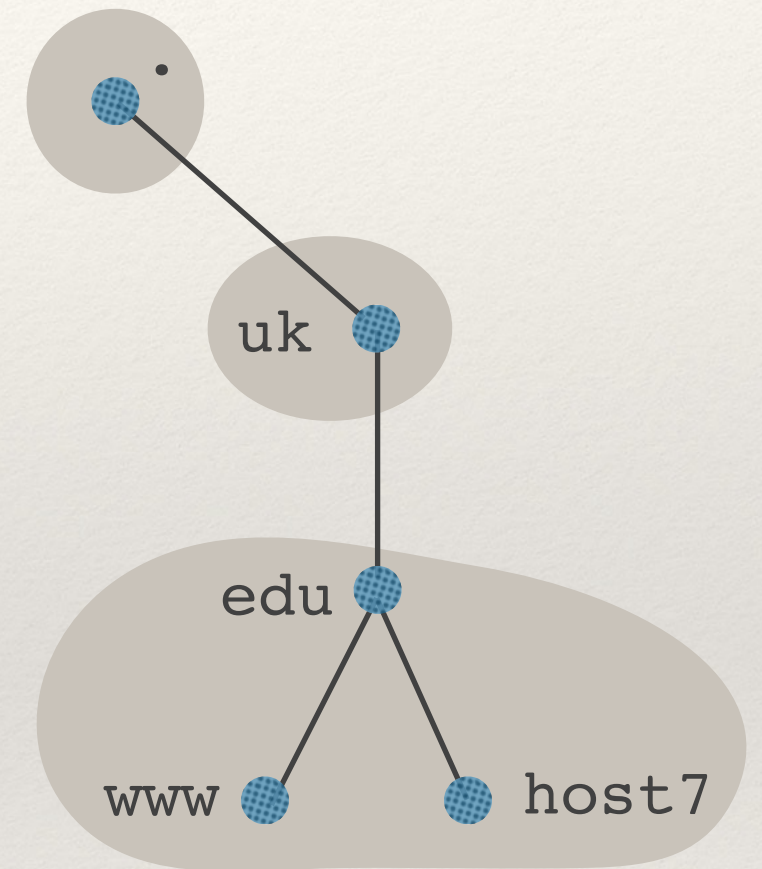




# Rozszyfrowywanie nazw (resolving)

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.

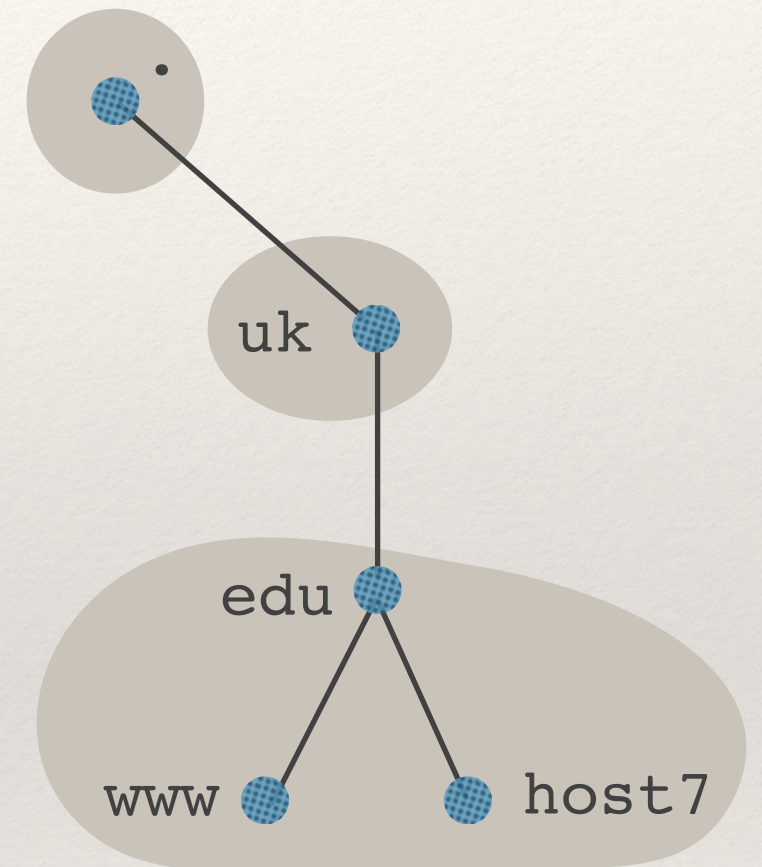




# Rozszyfrowywanie nazw (resolving)

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)

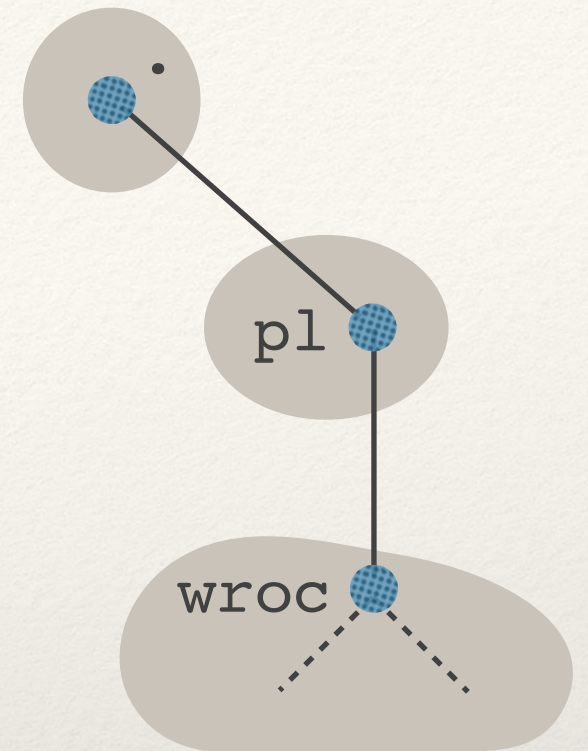


# Rekordy NS

---

## Typ NS (*nameserver*)

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





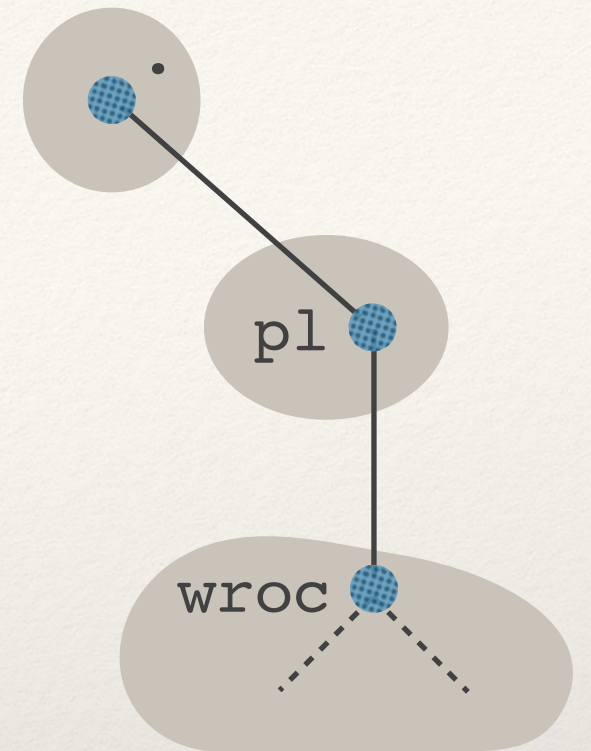
# Rekordy NS

---

## 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`?





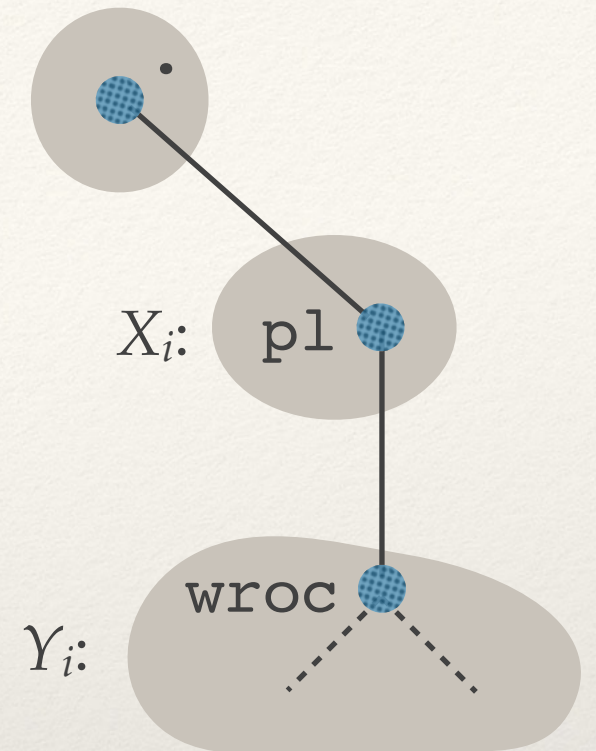
# Rekordy NS

## Typ NS (*nameserver*)

- ❖ nazwa = nazwa strefy (`wroc.p1`)
- ❖ wartość = nazwa serwera DNS obsługującego daną strefę (`sun2.pwr.wroc.p1`)

## Kto powinien pamiętać rekord NS dla strefy `wroc.p1`?

- ❖ Niech  $X_i$  = serwery nazw dla strefy `p1`
- ❖ Niech  $Y_i$  = serwery nazw dla strefy `wroc.p1` (między innymi `sun2.pwr.wroc.p1`)





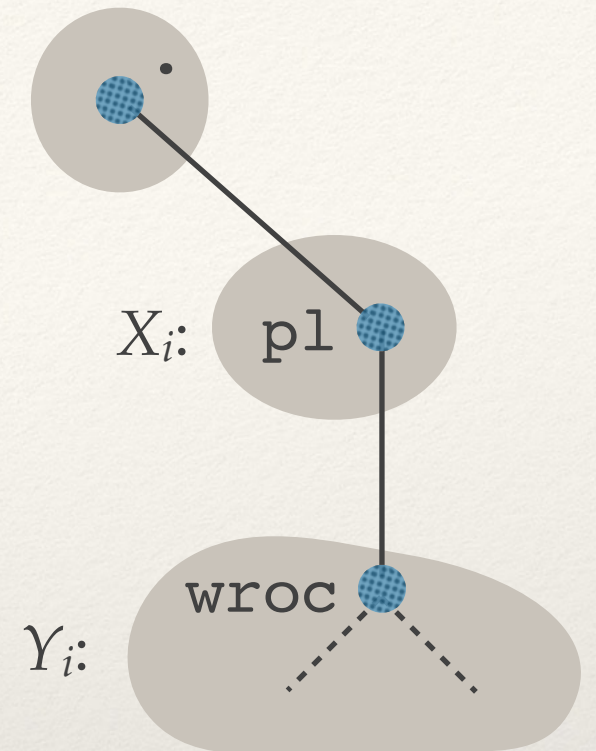
# Rekordy NS

## 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*)

- ❖ nazwa = nazwa domeny (`gmail.com`)
- ❖ wartość = nazwa serwera obsługującego pocztę (`gmail-smtp-in.1.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;  
};
```

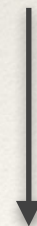
← zazwyczaj AF\_INET albo AF\_INET6

← 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,
```

```
    };
```

Brak obsługi błędów,  
plików nagłówkowych, etc.

chcemy tylko takie  
informacje

```
    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);
```

```
    }
```

```
}
```

demonstracja

kod programu na stronie wykładu



---

# Poczta elektroniczna

---



# Protokół SMTP

---

- ❖ Protokół przekazywania poczty.
- ❖ Protokół tekstowy, serwer nasłuchuje na porcie 25.



# Wysyłanie bezpośrednie (1)

---

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

- ❖ Łączymy się z adresem IP serwera odpowiedzialnego za odbieranie i przechowywanie pocztę dla domeny `xyx.com`.
- ❖ Rekord MX (*mail exchange*) w DNS a rekord A:
  - ♦ A, `ii.uni.wroc.pl` → **156.17.4.11**
  - ♦ MX, `ii.uni.wroc.pl` → `aspmx.l.google.com`
  - ♦ A, `aspmx.l.google.com` → **66.102.1.26**
  - ♦ Wysyłając pocztę do `abc@ii.uni.wroc.pl` łączymy się z **66.102.1.26** (nie z **156.17.4.11**).



# Wysyłanie bezpośrednio (2)

---

Chcemy wysłać pocztę do adresu **abc@xyz.com**.

nadawca maila

MX dla domeny xyz.com



klient SMTP

serwer SMTP



# Wysyłanie pośrednie

---

Chcemy wysłać pocztę do adresu **abc@xyz.com**.

## Krok 1.

nadawca maila

SMTP relay / smarthost



klient SMTP

serwer SMTP

## Krok 2.

SMTP relay / smarthost

MX dla domeny xyz.com



klient SMTP

serwer SMTP

(Kroków może być więcej).



# Wysyłanie pośrednie

Chcemy wysłać pocztę do adresu **abc@xyz.com**.

## Krok 1.

nadawca maila

SMTP relay / smarthost

= to co program pocztowy klienta ma ustawione jako „serwer SMTP”



klient SMTP

serwer SMTP

## Krok 2.

SMTP relay / smarthost

MX dla domeny xyz.com



klient SMTP

serwer SMTP

(Kroków może być więcej).



# Przekazywanie pośrednie

---

- ❖ Zazwyczaj wymaga autoryzacji nadawcy u SMTP relay
  - ✦ Różne mechanizmy autoryzacji są elementem protokołu SMTP.
  - ✦ Zabezpieczenie przed rozsyłaniem niechcianej poczty (spamu).
  - ✦ Czasem autoryzacja na podstawie adresu IP klienta.



# Przykładowy email (otrzymany)

**Delivered-To:** marcin.bienkowski@cs.uni.wroc.pl

**Received:** by 10.64.232.142 with SMTP id to14csp146725iec;

Sat, 23 Apr 2016 08:41:37 -0700 (PDT)

**Received:** from aisd.ii.uni.wroc.pl (156.17.4.30)

by mx.google.com with ESMTP id 1199si7484936lf1.24.2016.04.23.08.41.36

for <marcin.bienkowski@cs.uni.wroc.pl>;

Sat, 23 Apr 2016 08:41:36 -0700 (PDT)

**Received:** by aisd.ii.uni.wroc.pl (Postfix, from userid 1000) id E6BCD5F84D;

Sat, 23 Apr 2016 17:41:35 +0200 (CEST)

Date: Sat, 23 Apr 2016 17:41:35 +0200

From: mbi <mbi@ii.uni.wroc.pl>

To: marcin.bienkowski@cs.uni.wroc.pl

Subject: Testowy email

Message-ID: <20160423154135.GA11834@aisd.ii.uni.wroc.pl>

MIME-Version: 1.0

Content-Type: text/plain; charset=utf-8

Content-Disposition: inline

Content-Transfer-Encoding: 8bit

User-Agent: Mutt/1.5.23 (2014-03-12)

Jakaś treść maila.

M.

pola ustawiane  
przez odbiorcę

pola ustawiane  
przez serwery  
pośredniczące

pola ustawiane przez nadawcę



# Pola nagłówka ustawiane przez klienta

---

- ❖ From:
- ❖ To:
- ❖ Subject:
- ❖ Cc:
- ❖ Bcc: („ślepa kopia“)
- ❖ Message-ID: (unikatowy identyfikator wiadomości)
- ❖ Date: (data wysłania)
- ❖ In-Reply-To: (ID maila, na którego odpowiadamy)
- ❖ References:



# Typ zawartości

---

Pole `Content-Type`: nagłówek określa:

- ❖ czym jest treść maila (w standardzie MIME)

- ♦ czysty tekst (`text/plain`)

- ♦ HTML (`text/html`)

- ❖ kodowanie znaków

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

`Content-Transfer-Encoding: 8bit`



# Załączniki pocztowe

**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.

treść tekstowego maila w UTF-8

--UNIKATOWY-CIĄG-ZNAKÓW

**Content-Type: image/jpeg**

Content-Disposition: attachment; filename="obrazek.jpg"

Content-Transfer-Encoding: base64

ZAWARTOŚĆ-PLIKU-ZAKODOWANA-W-BASE64.

załącznik obrazek.jpg

--UNIKATOWY-CIĄG-ZNAKÓW



# Dostarczanie poczty do użytkownika

---

- ❖ Protokół POP3.
- ❖ Protokół IMAP.
- ❖ Klienci pocztowe jako aplikacje WWW.



# Spam: niechciane wiadomości pocztowe

---

Sposoby wykrywania i usuwania spamu:

- ❖ metody statystyczne, uczenie maszynowe
- ❖ greylisting
- ❖ SPF
- ❖ ...



**Wolne wysyłanie spamu jest nieopłacalne.**

- ❖ Pomysł: każemy klientowi SMTP wysłać wiadomość ponownie za jakiś czas:

```
-> MAIL FROM: <nadawca@jakasdomena.pl>
```

```
<- 250 2.1.0 Sender ok
```

```
-> RCPT TO: <odbiorca@innadomena.pl>
```

```
<- 451 4.7.1 Please try again later
```

- ❖ Ustawiamy okno czasowe (np. „nie wcześniej niż za 10 min. i nie później niż po godzinie“).
  - ✦ Jeśli klient ponowi w danym oknie czasowym, to akceptujemy jego email.
  - ✦ Problemy z poprawnym ustawieniem okna.
  - ✦ Dostarczanie poczty przestaje być szybkim procesem.
- ❖ Stosowany wariant: zamiast odrzucać, odbieraj wolniej z okna TCP.



# Spam: SPF (Sender Policy Framework)

---

- ❖ Rekord SPF (o typie TXT) w DNS dla danej domeny:
  - ♦ TXT, `ii.uni.wroc.pl` → „`v=spf1 ip4:156.17.4.0/24  
mx:ii.uni.wroc.pl  
mx:gmail.com  
mx:google.com  
-all`”
- ❖ Definiuje jakie komputery są uprawnione do wysyłania poczty z polem `From:` równym `adres@ii.uni.wroc.pl`.
  - ♦ komputery z adresów `156.17.4.0/24`.
  - ♦ serwery SMTP obsługujące pocztę dla domen `ii.uni.wroc.pl`, `gmail.com` i `google.com`.
- ❖ Rekord sprawdzany przez odbiorcę.



---

# Peer-to-peer

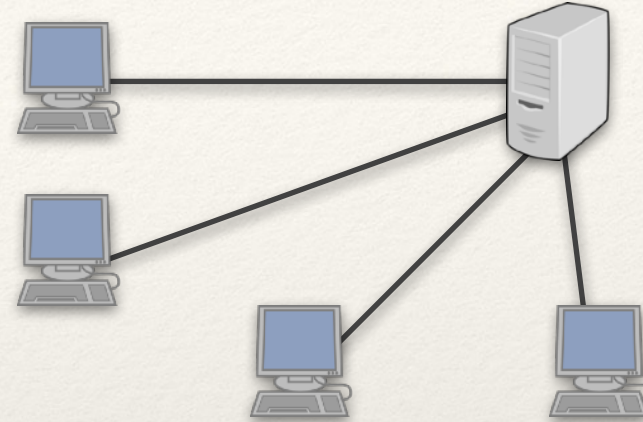
---



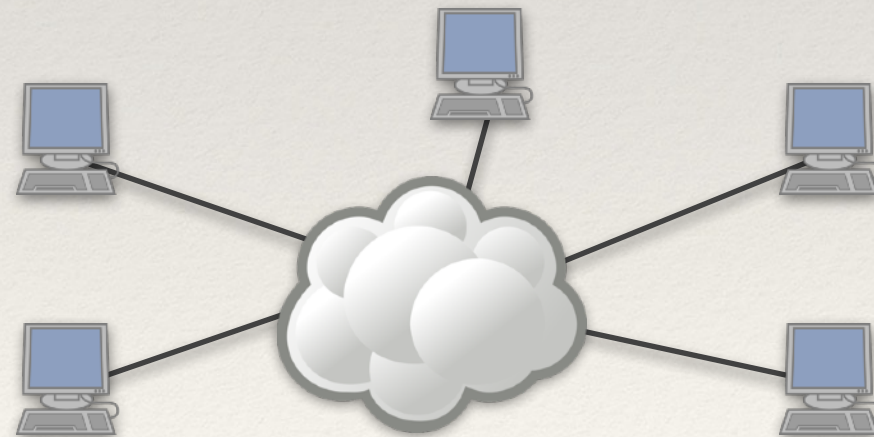
# Klient-serwer a peer-to-peer

---

- ❖ Do tej pory mówiliśmy o **architekturze klient-serwer** (HTTP, DNS, poczta elektroniczna,...)



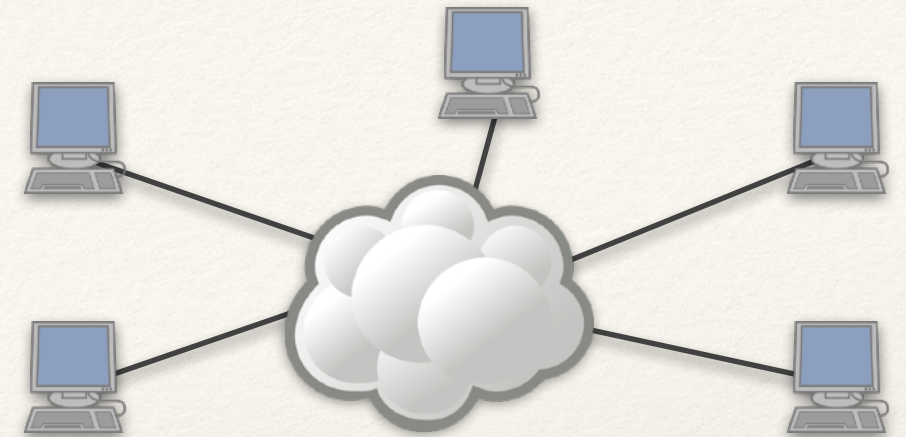
- ❖ Część rozwiązań pracuje w **architekturze peer-to-peer** (BitTorrent, Skype, ...)





# Peer-to-peer

---



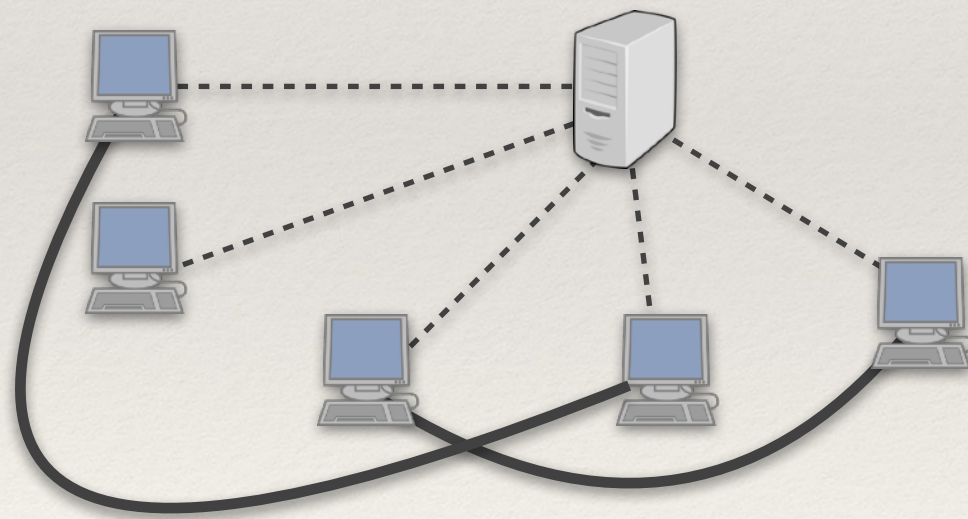
## Architektura P2P:

- ❖ 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 (Skype)
  - ♦ Pomagają w podłączeniu (punkt pierwszego kontaktu + później)



..... komunikaty kontrolne  
— właściwe dane



# Warstwa aplikacji a warstwa transportowa

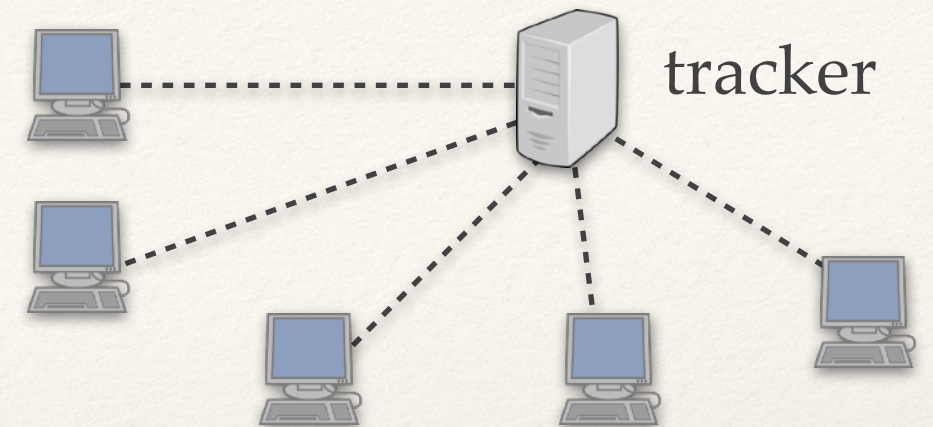
---

- ❖ Peer-to-peer = określenie logiki warstwy aplikacji.
- ❖ Dwa punkty sieci peer-to-peer chcą wymieniać dane.
  - ✦ Wykorzystują w tym celu warstwę transportową (TCP lub UDP).
  - ✦ Z punktu widzenia TCP lub UDP jeden z nich (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.

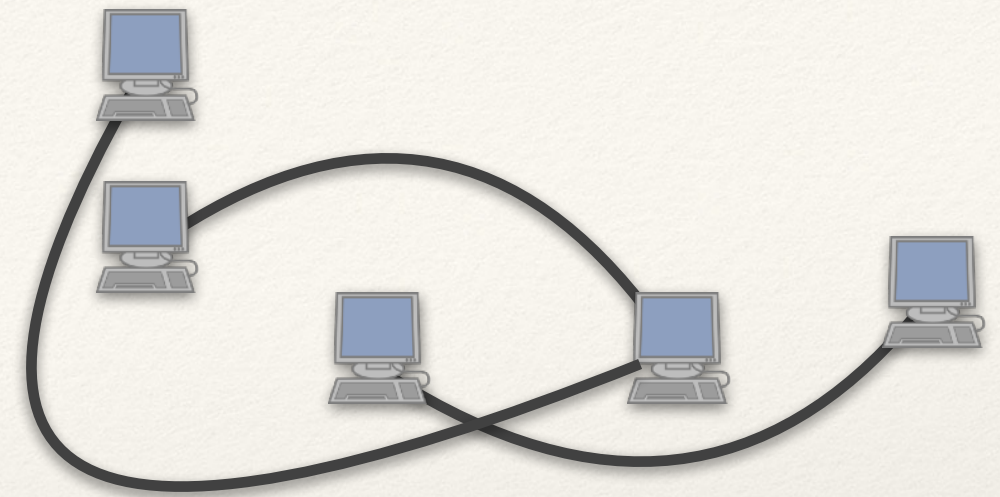


# BitTorrent: przesyłanie pliku (1)

---

## Plik dzielony na kawałki

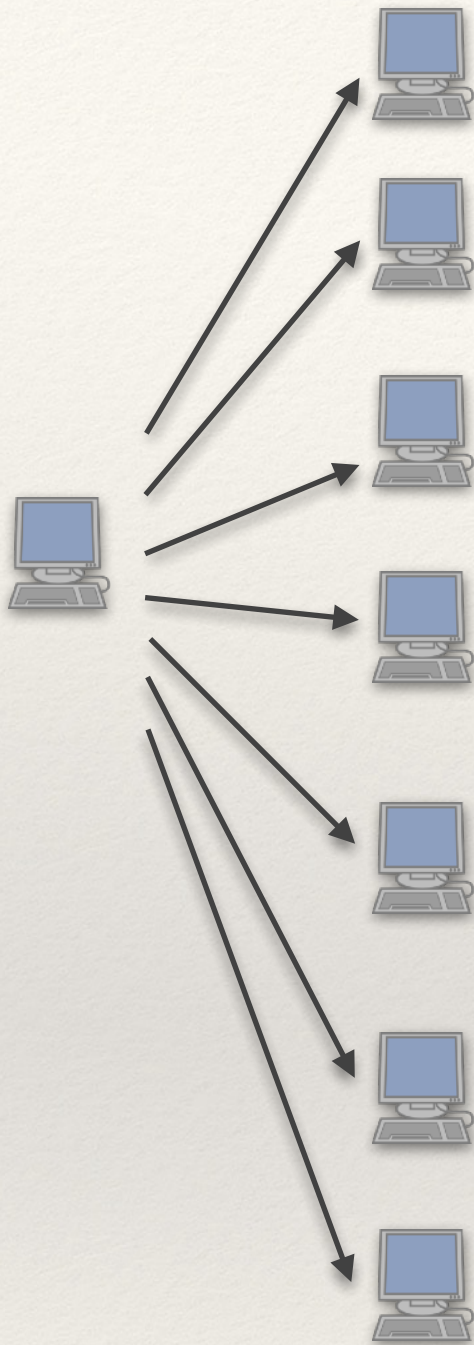
- ❖ Każdy kawałek pobierany niezależnie
- ❖ Rozmiar kawałka ok. 256 KB - 16 MB
  - ♦ Duży → żeby okno TCP miało czas urosnąć.
  - ♦ Mały → żeby plik miał wiele kawałków (urównoleglenie).
- ❖ *Seeder* = ma wszystkie kawałki.
- ❖ *Leecher* = ma niektóre kawałki.





# Jak przesłać jeden kawałek, żeby wszyscy go mieli? (1)

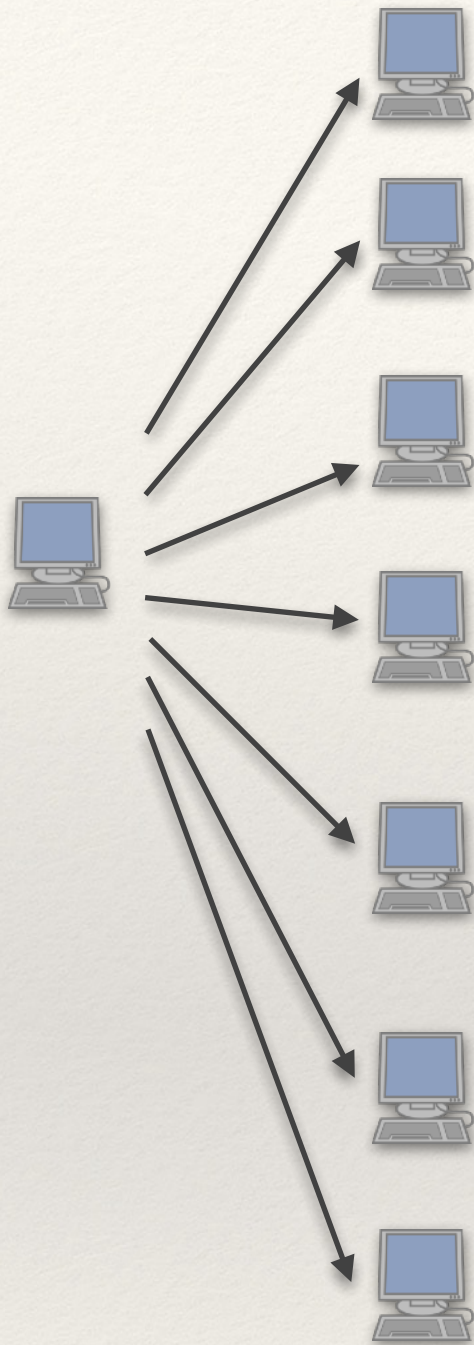
---



1. Minimalne opóźnienie, maksymalne obciążenie pojedynczego członka sieci  
(jak w modelu klient-server)



# Jak przesłać jeden kawałek, żeby wszyscy go mieli? (1)



1. Minimalne opóźnienie, maksymalne obciążenie pojedynczego członka sieci  
(jak w modelu klient-server)

2. Minimalne obciążenie pojedynczego członka sieci, duże opóźnienie

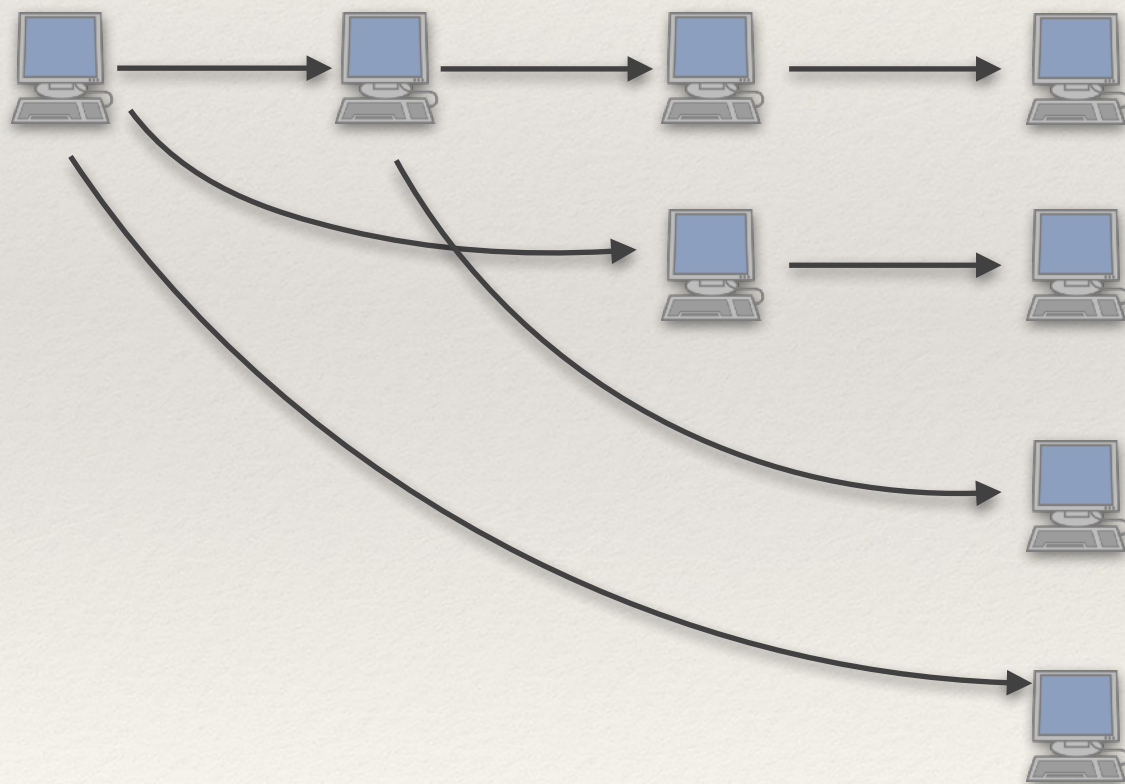




# Jak przesłać jeden kawałek, żeby wszyscy go mieli? (2)

## 3. Rozwiązanie pośrednie:

- ❖ logarytmiczna głębokość
- ❖ logarytmiczne obciążenie pojedynczych wierzchołków
- ❖ duża odporność na opuszczanie sieci przez komputery





# BitTorrent: przesyłanie pliku (2)

---

- ❖ **Typowo:** każdy klient ma mnóstwo chętnych na posiadane kawałki.
  - ♦ Klienci zazwyczaj chcą kawałki, które występują najrzadziej w sieci.
- ❖ **Seeder** wybiera kandydatów wśród chętnych po kolei (co najwyżej  $Q$  naraz).
- ❖ **Leecher** ma listę  $P$  klientów, którym udostępnia.
  - ♦ Tylko pod warunkiem, że *dostał od nich coś w zamian*.
  - ♦ Na liście jest  $P$  takich klientów, którzy najszybciej wysyłają mu swoje kawałki.
  - ♦ Eksploracja: czasem dajemy kawałek losowemu członkowi sieci
    - może odeśle nam jakiś kawałek odpowiednio szybko?
  - ♦ Nowi klienci: jeśli klient mówi, że jest nowy, to dostaje kawałek za darmo.



# BitTorrent: metadane

---

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



---

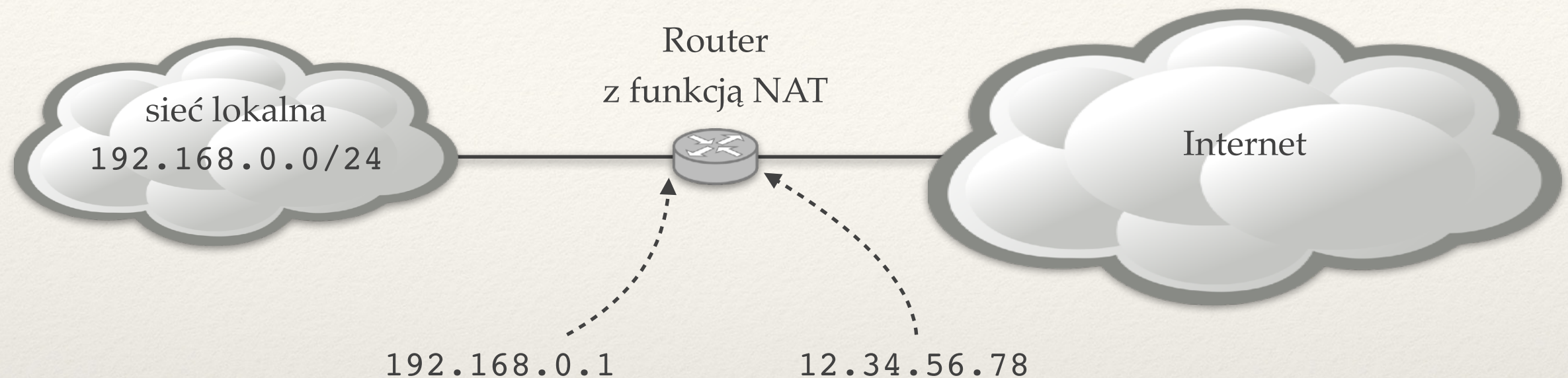
# NAT vs. warstwa aplikacji

---



# NAT

---



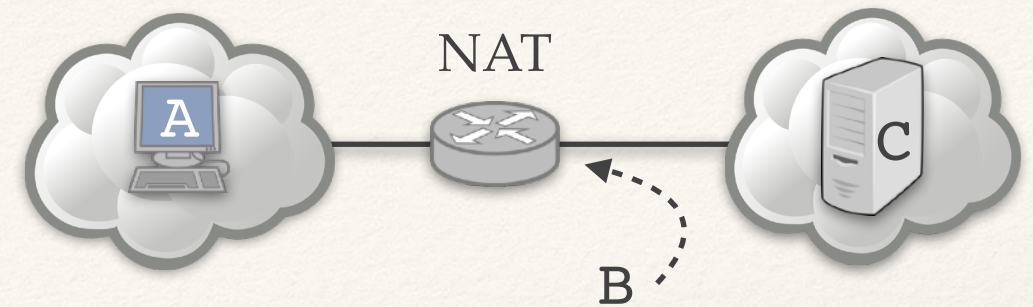
- ❖ Bardzo powszechne rozwiązanie.
- ❖ Z reszty Internetu cała sieć lokalna wygląda tak samo, jak pojedynczy komputer z adresem 12.34.56.78.



# 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,  $P_C$ )  $\rightarrow$   $P_B$ .
- ✦ **Dla kolejnych podobnych pakietów przypisanie będzie takie samo.**
- ✦ Jeśli przychodzi pakiet **z Internetu** do (B,  $P_B$ ) to jego adres i port docelowy zostanie podmieniony na (A,  $P_A$ ).

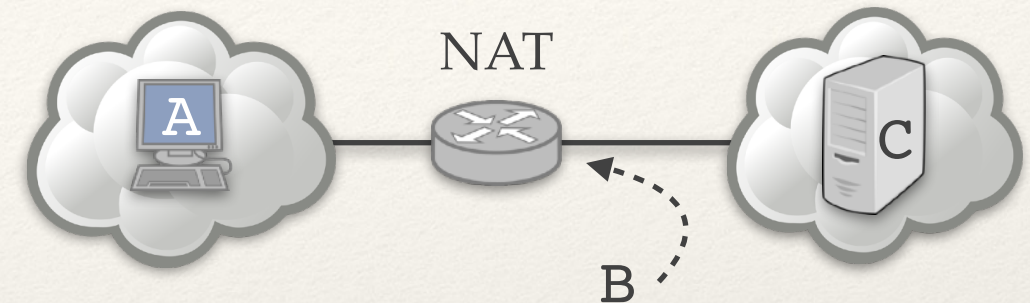


# NAT a P2P

---

## Kiedyś:

- ❖ Komunikacja zawsze w modelu klient-serwer.
- ❖ Serwery nie są za routerami z NAT.
- ❖ Klienci mogą być za routerami z NAT
- ❖ Początkowa transmisja (np. TCP SYN) od klienta do serwera tworzy przypisanie  $(A, P_A, C, P_C) \rightarrow P_B$ , dzięki któremu pakiety z odpowiedziami serwera mogą wracać do klienta.



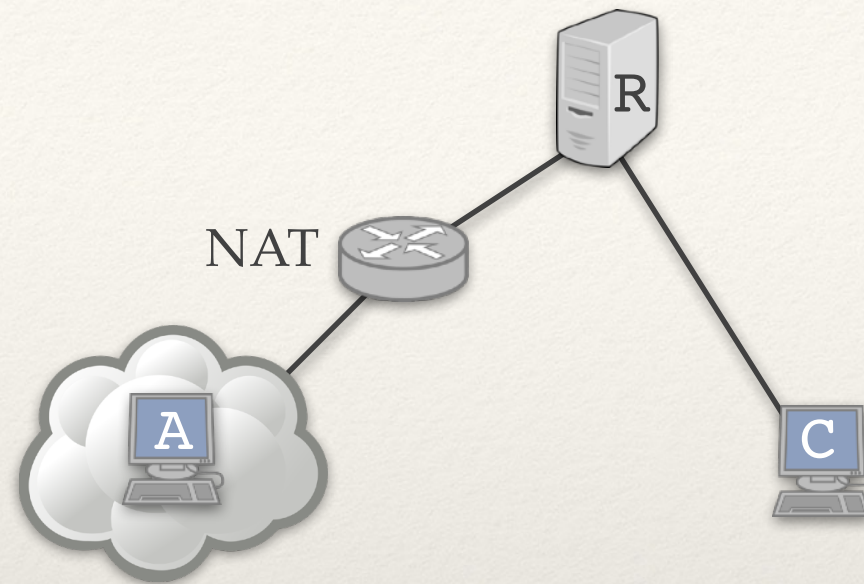
## Obecnie:

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



# Odwrócone połączenie

---

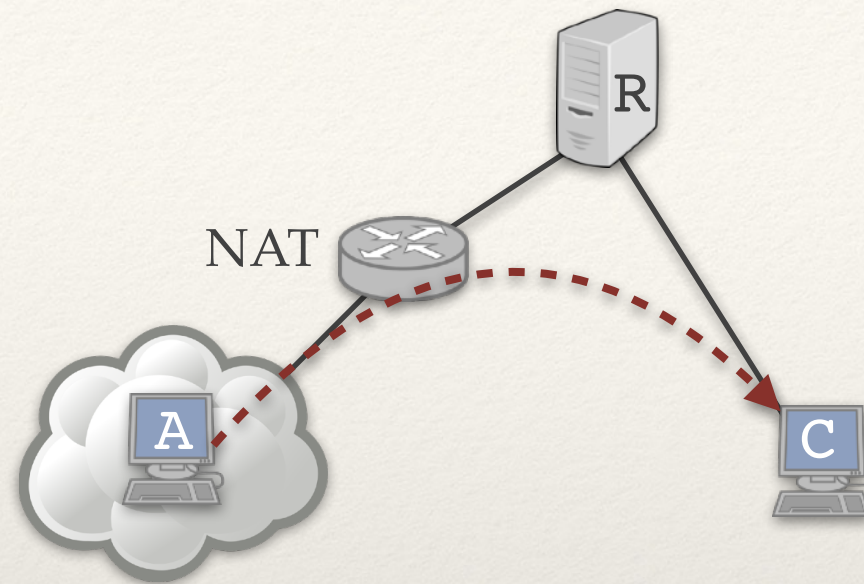


- ❖ C chce nawiązać połączenie z A, ale A jest za NAT.
- ❖ Jeśli oba utrzymują kontakt z R, to C może poprosić (przez R) komputer A o nawiązanie bezpośredniego połączenia z C.
- ❖ Stosowane np. w Skype.



# Odwrócone połączenie

---

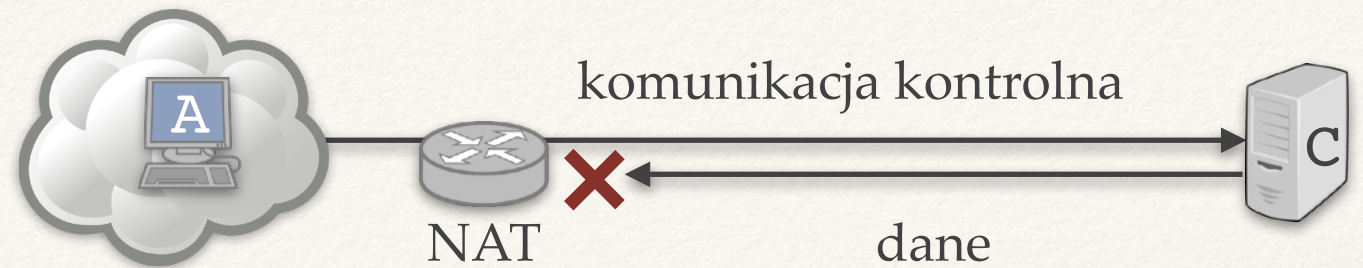


- ❖ C chce nawiązać połączenie z A, ale A jest za NAT.
- ❖ Jeśli oba utrzymują kontakt z R, to C może poprosić (przez R) komputer A o nawiązanie bezpośredniego połączenia z C.
- ❖ Stosowane np. w Skype.



# Odwrócone połączenie w protokole FTP

**FTP: protokół przesyłania plików.**

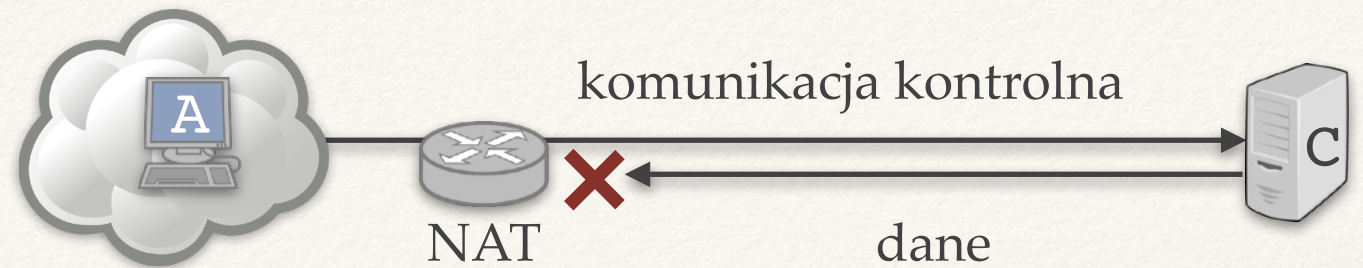


- ❖ Na początku klient A łączy się z serwerem C na porcie 21 (połączenie na komunikaty kontrolne).
- ❖ A wysyła polecenie „chcę pobrać plik i słucham na porcie X”
  - ♦ C łączy się z portem X klienta A i wysyła plik (odrębne połączenie TCP).
  - ♦ Połączenie odrzucane przez NAT.



# Odwrócone połączenie w protokole FTP

**FTP: protokół przesyłania plików.**

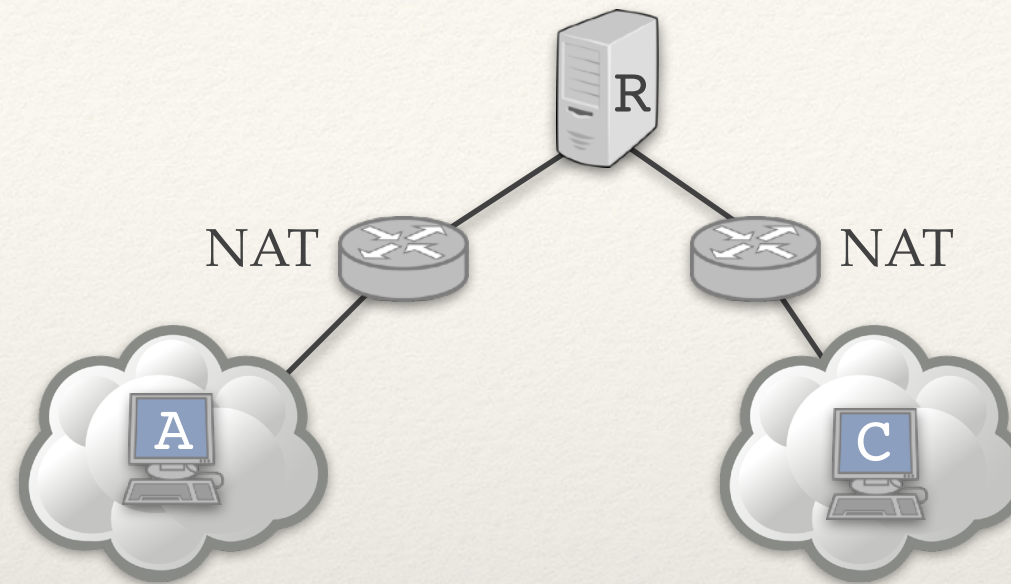


- ❖ Na początku klient A łączy się z serwerem C na porcie 21 (połączenie na komunikaty kontrolne).
- ❖ A wysyła polecenie „chcę pobrać plik i słucham na porcie X”
  - ✦ C łączy się z portem X klienta A i wysyła plik (odrębne połączenie TCP).
  - ✦ Połączenie odrzucane przez NAT.
- ❖ Tryb pasywny FTP: 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.



# Przełączniki

---

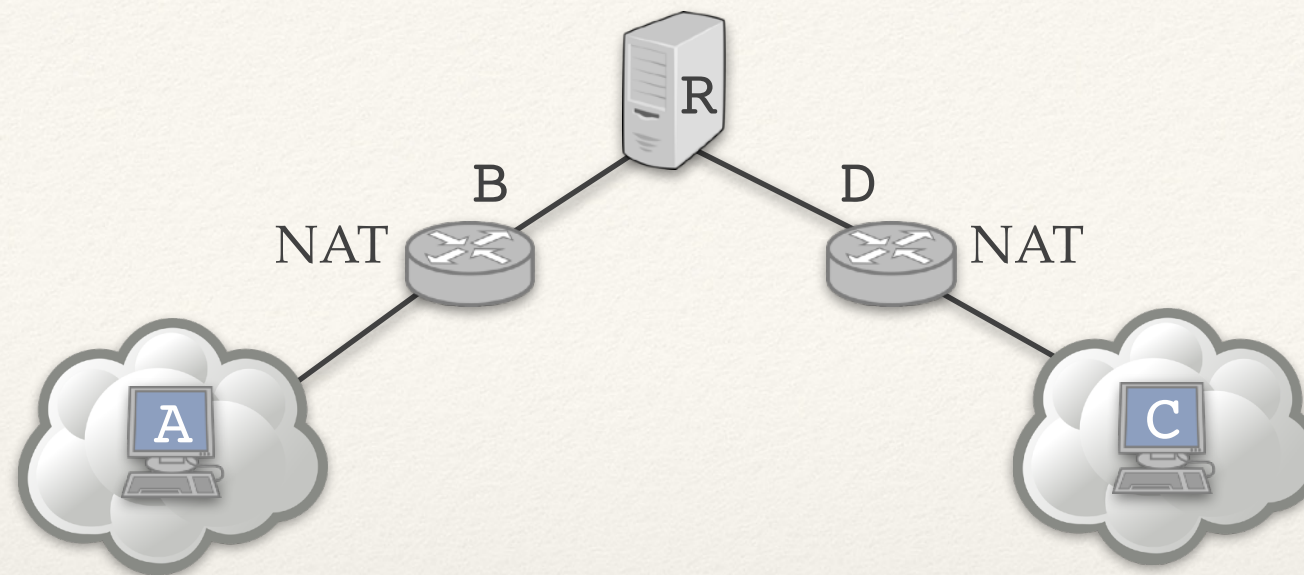


- ❖ Jeśli A i C utrzymują kontakt z R, to oba mogą nawiązać połączenie z R i R może przekazywać między nimi dane.
- ❖ Stosowane np. w Skype (jeśli wszystko inne zawiedzie).



# Przechodzenie przez NAT (1)

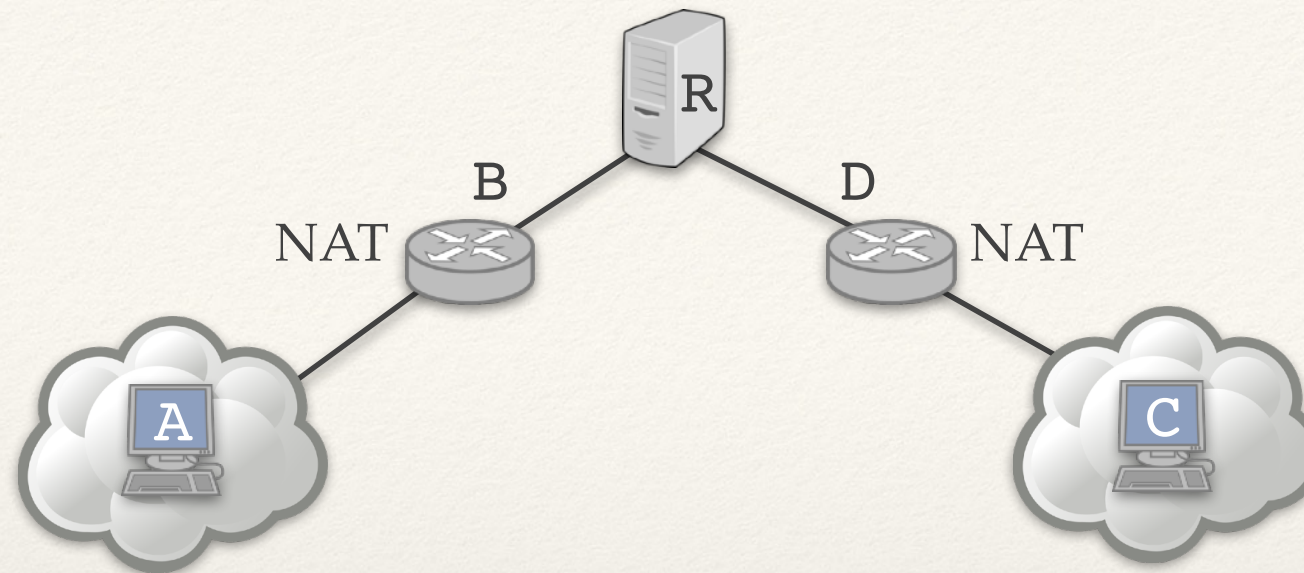
---





# Przechodzenie przez NAT (1)

---

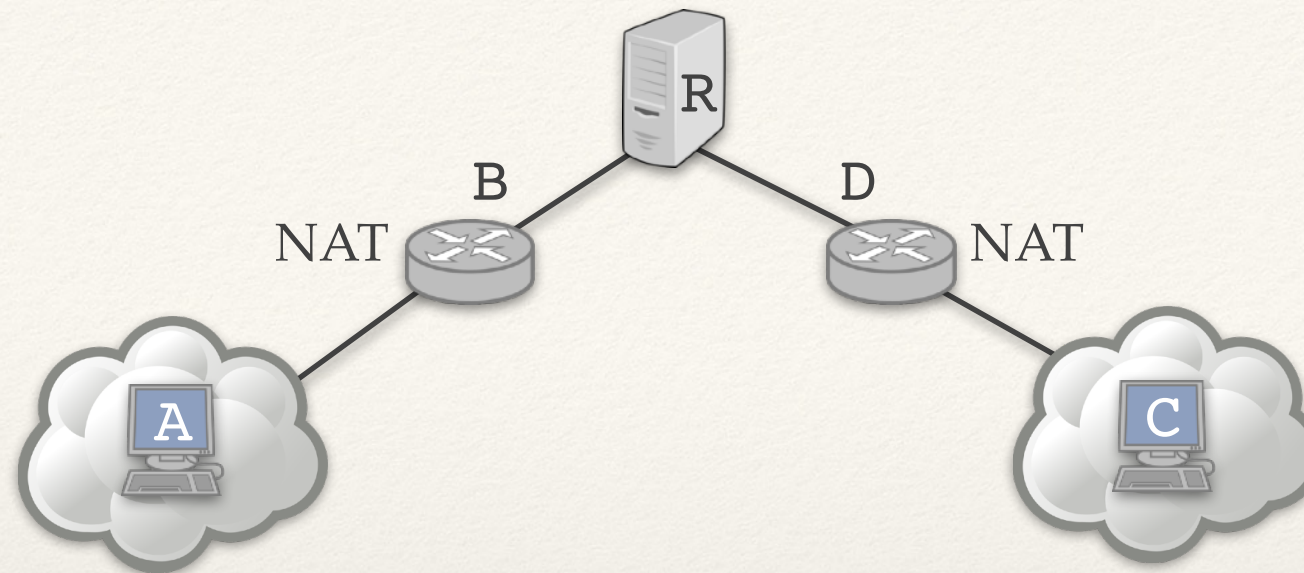


- ❖ A wysyła z portu  $P_A$  pakiet do R o treści „ $(A, P_A)$ ”.
- ❖ Na routerze NAT zostaje utworzone przypisanie  $(A, P_A) \rightarrow (B, P_B)$ .
- ❖ R widzi pakiet o treści „ $(A, P_A)$ ” od  $(B, P_B)$ , tj. poznaje przypisanie wygenerowane przez B.



# Przechodzenie przez NAT (1)

---



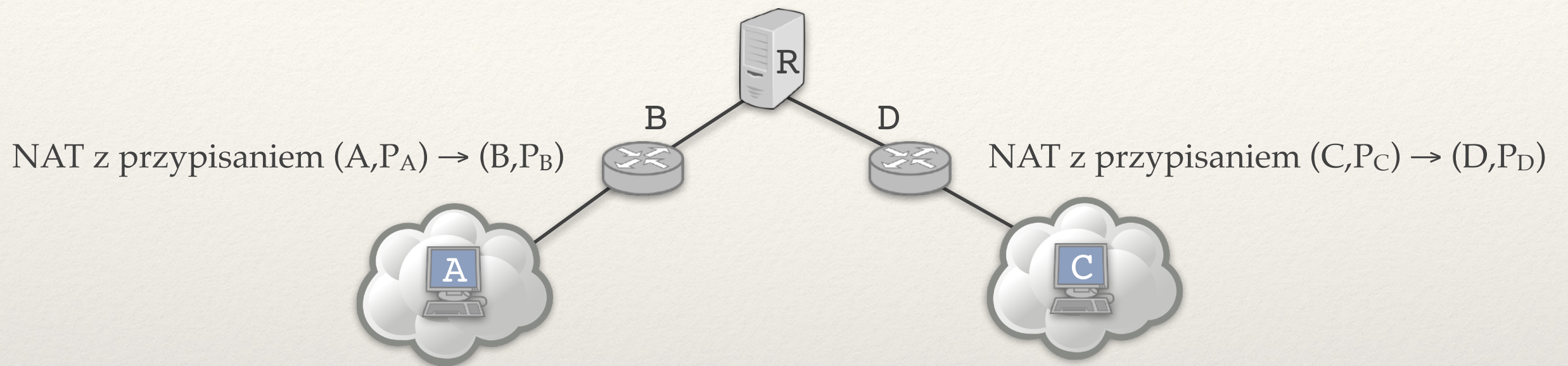
- ❖ A wysyła z portu  $P_A$  pakiet do R o treści „ $(A, P_A)$ ”.
- ❖ Na routerze NAT zostaje utworzone przypisanie  $(A, P_A) \rightarrow (B, P_B)$ .
- ❖ R widzi pakiet o treści „ $(A, P_A)$ ” od  $(B, P_B)$ , tj. poznaje przypisanie wygenerowane przez B.
- ❖ W taki sam sposób R poznaje przypisanie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



# Przechodzenie przez NAT (2)

---

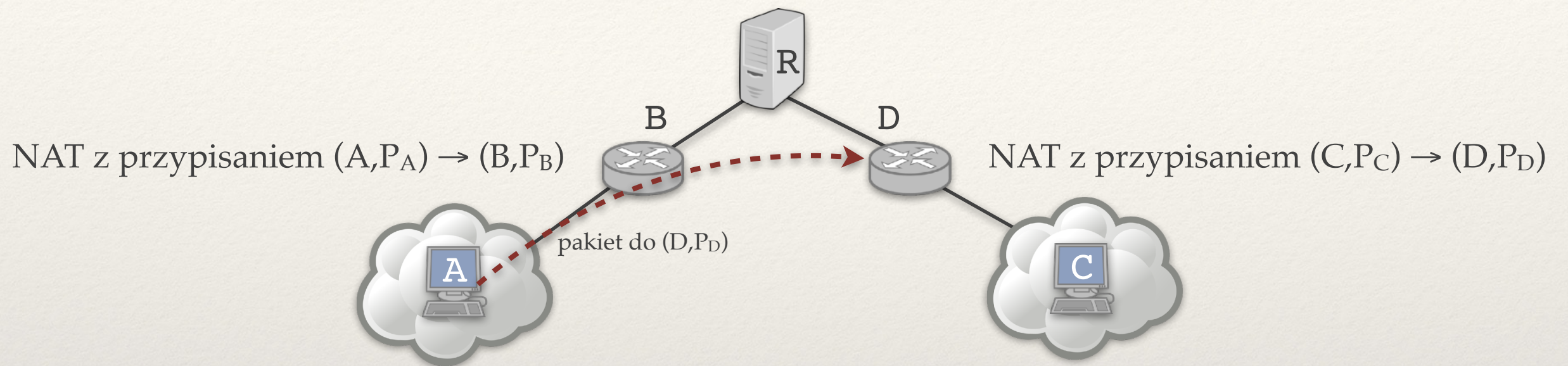
**Fakt:** Jeśli R wyśle dane do  $(D, P_D)$  to zostaną przesłane do  $(C, P_C)$ .





# Przechodzenie przez NAT (2)

**Fakt:** Jeśli R wyśle dane do  $(D, P_D)$  to zostaną przesłane do  $(C, P_C)$ .

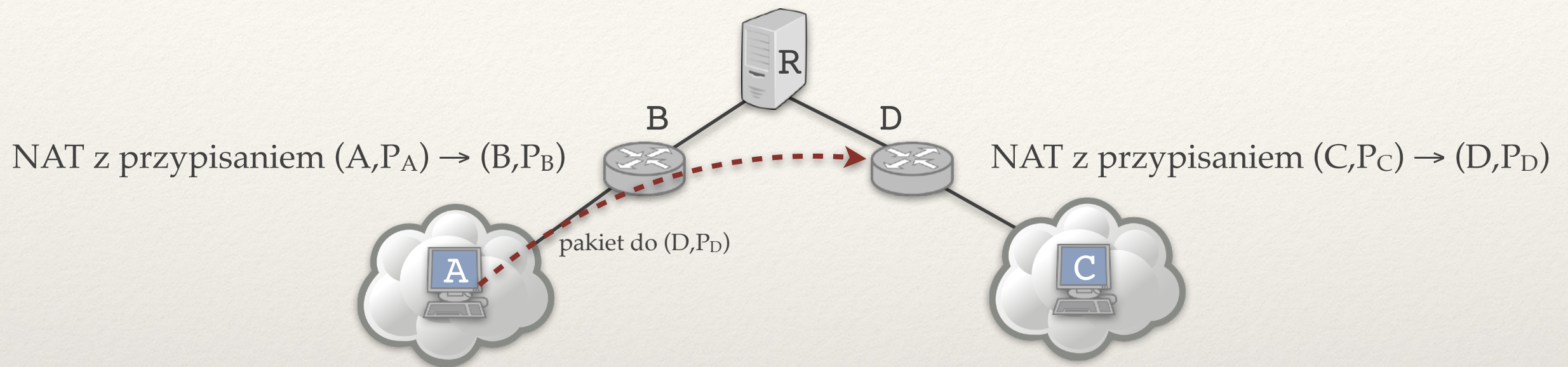


**Pytanie:** Czy jeśli A wyśle dane do  $(D, P_D)$  to zostaną one przesłane do  $(C, P_C)$ ?



# Przechodzenie przez NAT (2)

**Fakt:** Jeśli R wyśle dane do  $(D, P_D)$  to zostaną przesłane do  $(C, P_C)$ .



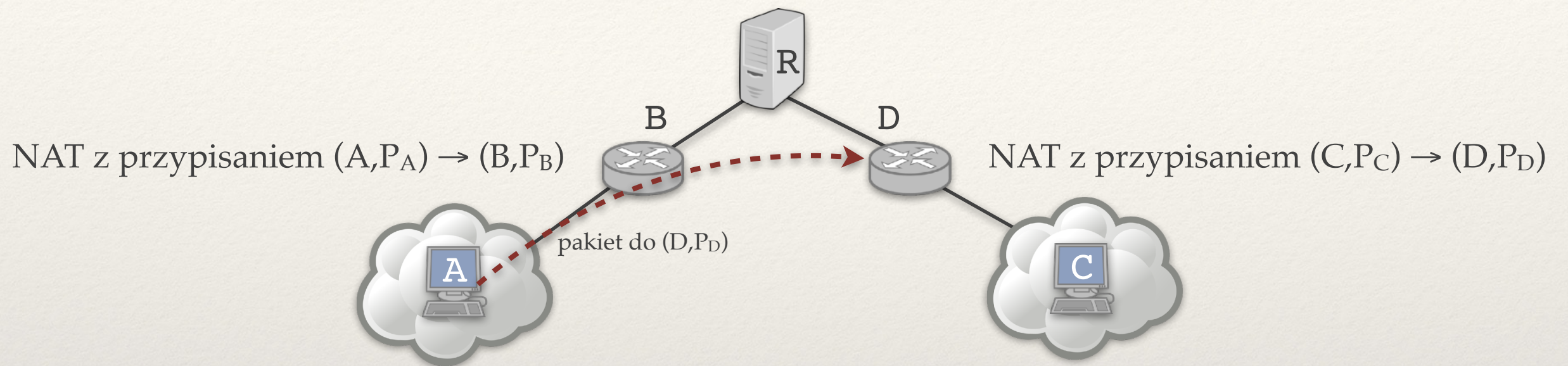
**Pytanie:** Czy jeśli A wyśle dane do  $(D, P_D)$  to zostaną one przesłane do  $(C, P_C)$ ?

❖ Niestety nie zawsze!



# Przechodzenie przez NAT (2)

**Fakt:** Jeśli R wyśle dane do  $(D, P_D)$  to zostaną przesłane do  $(C, P_C)$ .



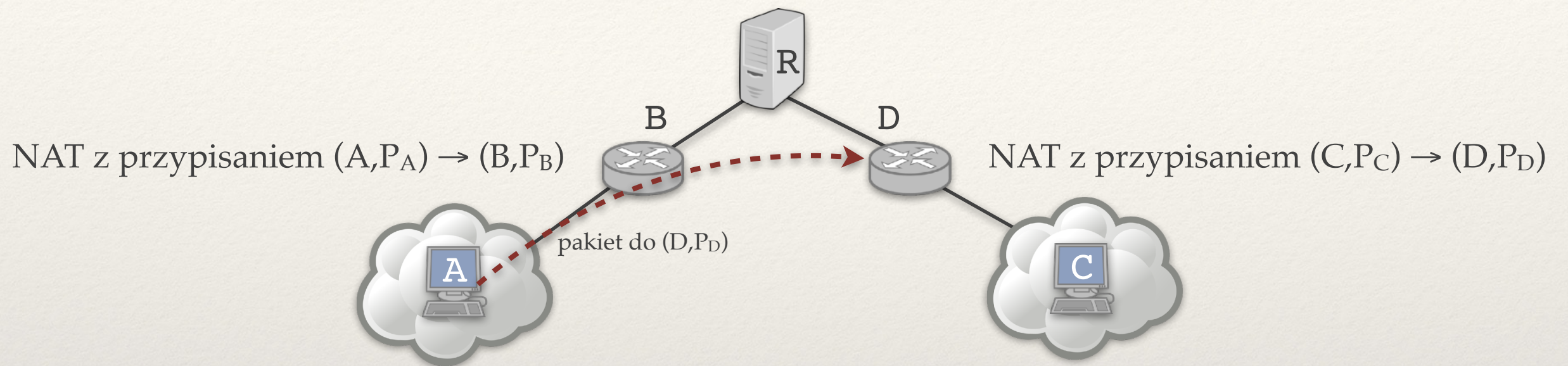
**Pytanie:** Czy jeśli A wyśle dane do  $(D, P_D)$  to zostaną one przesłane do  $(C, P_C)$ ?

- ❖ Niestety nie zawsze!
- ❖ Ale jeśli tak jest, to możliwa jest 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)

**Fakt:** Jeśli R wyśle dane do  $(D, P_D)$  to zostaną przesłane do  $(C, P_C)$ .



**Pytanie:** Czy jeśli A wyśle dane do  $(D, P_D)$  to zostaną one przesłane do  $(C, P_C)$ ?

❖ Niestety nie zawsze!

**dłaczego nie?**

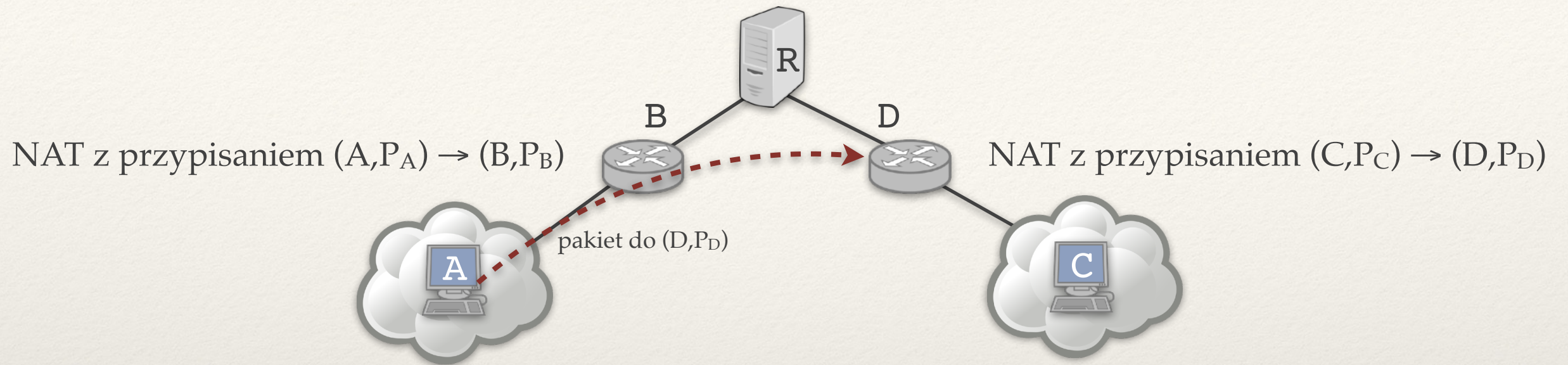
❖ Ale jeśli tak jest, to możliwa jest komunikacja:

- ❖ 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 (3)

A wysyła pakiet do  $(D, P_D)$ ...

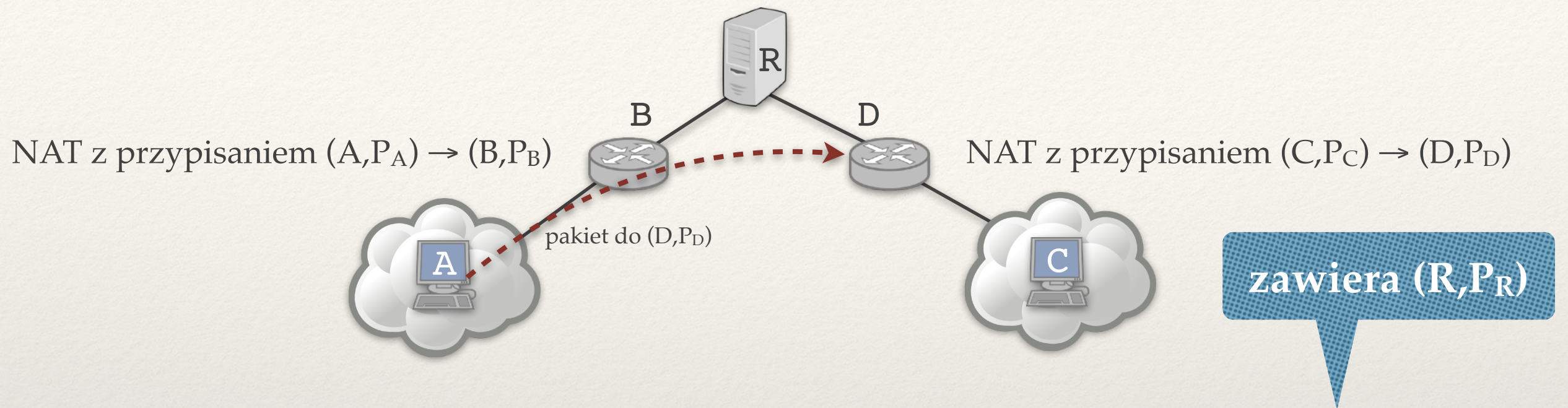


Poza przypisaniem  $(C, P_C) \rightarrow (D, P_D)$  router D pamięta **listę odbiorców** pakietów, które wychodziły przez  $(D, P_D)$ .



# Przechodzenie przez NAT (3)

A wysyła pakiet do  $(D, P_D)$ ...

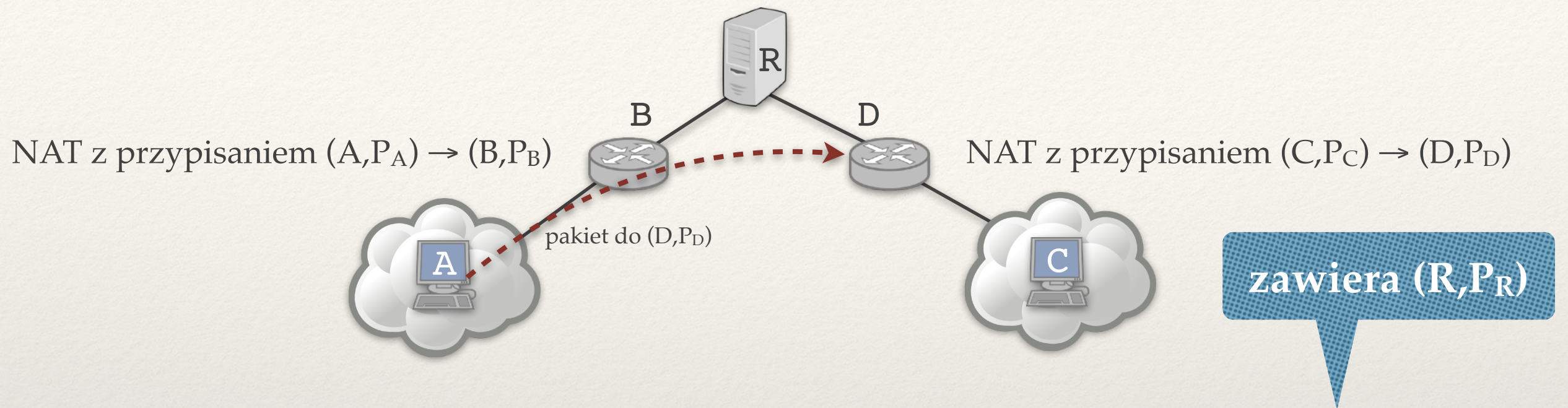


Poza przypisaniem  $(C, P_C) \rightarrow (D, P_D)$  router D pamięta **listę odbiorców** pakietów, które wychodziły przez  $(D, P_D)$ .



# Przechodzenie przez NAT (3)

A wysyła pakiet do  $(D, P_D)$ ...



Poza przypisaniem  $(C, P_C) \rightarrow (D, P_D)$  router D pamięta **listę odbiorców** pakietów, które wychodziły przez  $(D, P_D)$ .

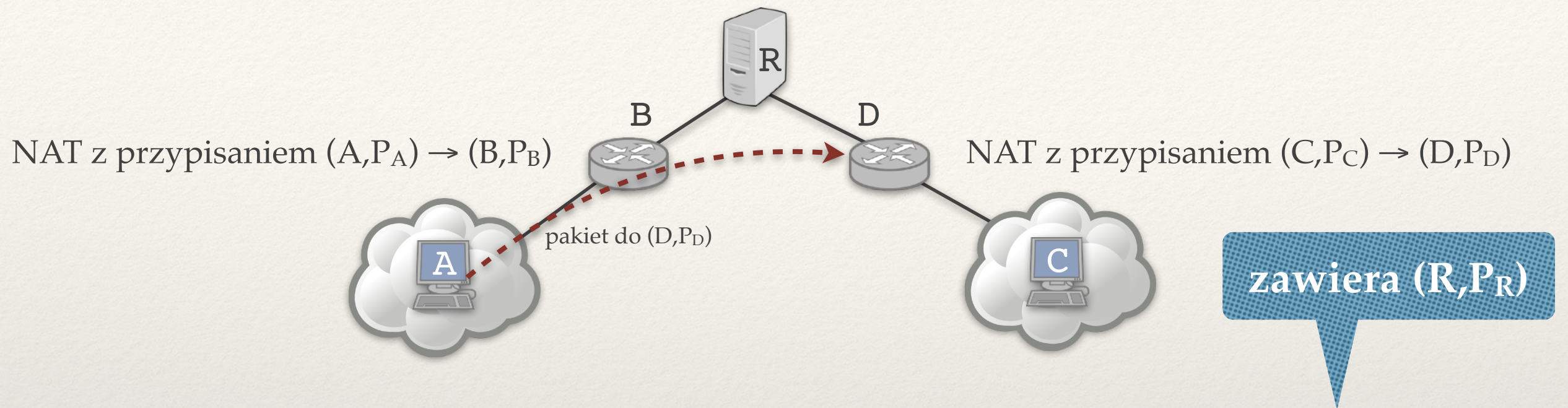
D przekaże do  $(C, P_C)$ :

- ❖ wszystkie pakiety (**pełny asymetryczny NAT**).
- ❖ pakiety tylko od IP z listy (**ograniczony as. NAT**).
- ❖ pakiety tylko od par  $(IP, port)$  z listy (**ogranicz. portowo as. NAT**).



# Przechodzenie przez NAT (3)

A wysyła pakiet do  $(D, P_D)$ ...



Poza przypisaniem  $(C, P_C) \rightarrow (D, P_D)$  router D pamięta **listę odbiorców** pakietów, które wychodziły przez  $(D, P_D)$ .

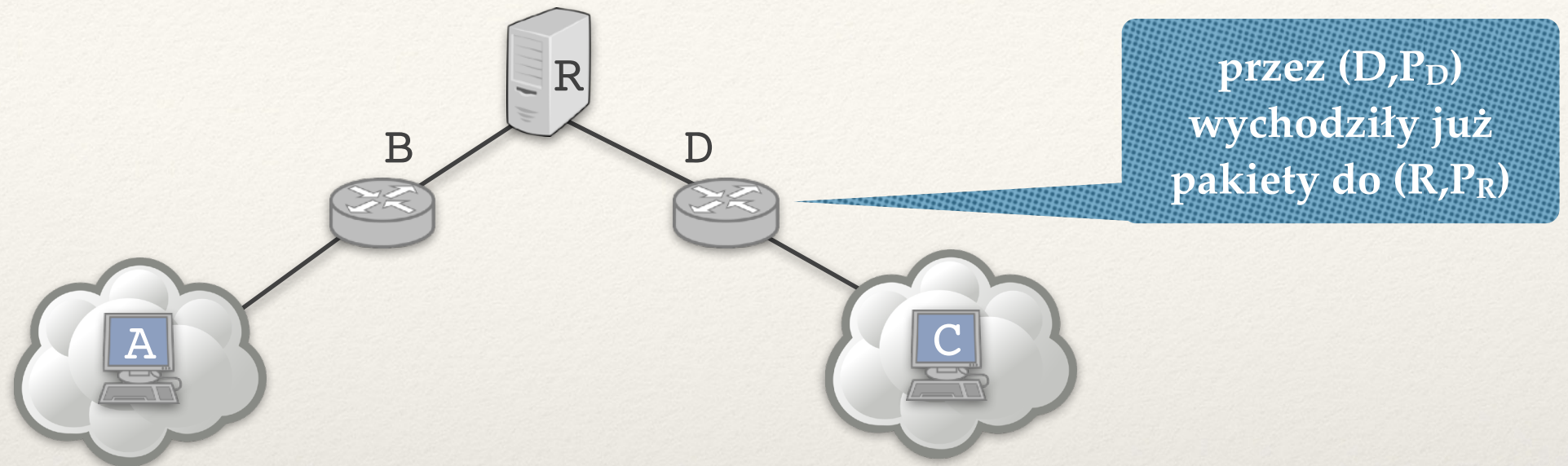
D przekaże do  $(C, P_C)$ :

- ❖ wszystkie pakiety (**pełny asymetryczny NAT**). pakiet od  $(A, P_A)$   
przejdzie przez D
- ❖ pakiety tylko od IP z listy (**ograniczony as. NAT**).
- ❖ pakiety tylko od par  $(IP, port)$  z listy (**ogranicz. portowo as. NAT**).



# Wybijanie dziur (*hole punching*)

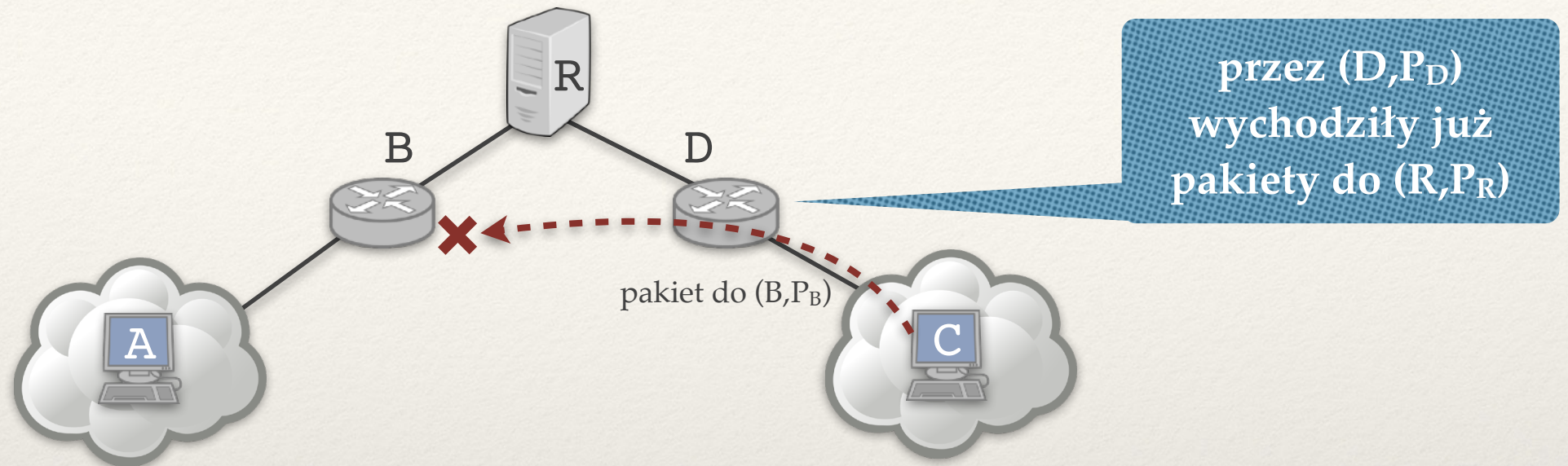
Co z asymetrycznymi ograniczonymi (portowo) NAT?





# Wybijanie dziur (*hole punching*)

Co z asymetrycznymi ograniczonymi (portowo) NAT?

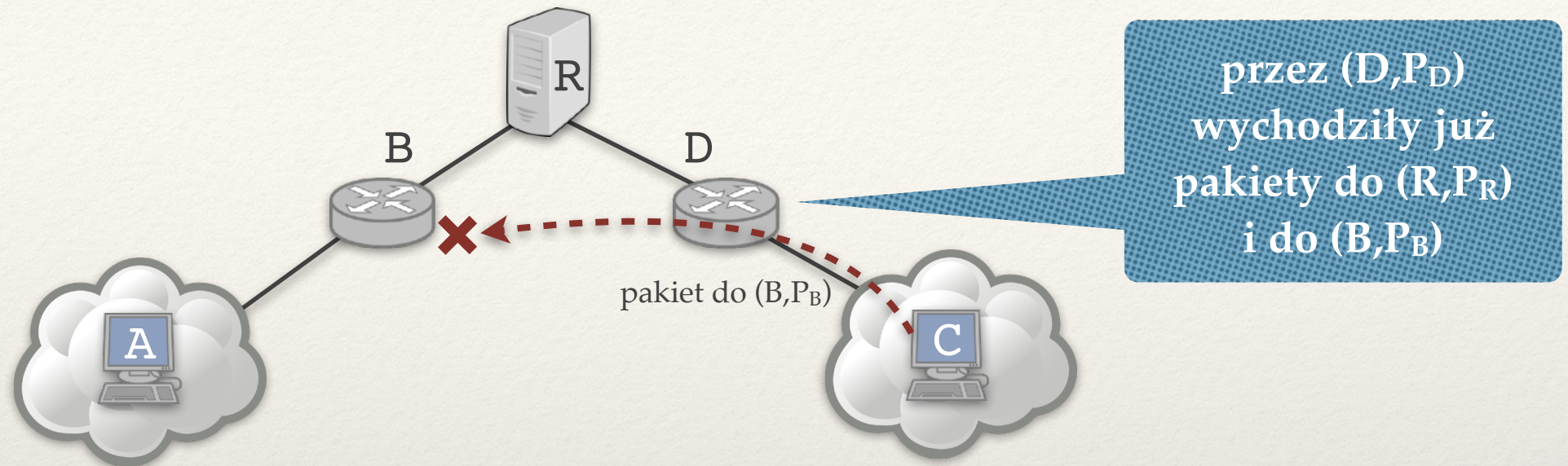


- ❖  $(C, P_C)$  wysyła pakiet do  $(B, P_B)$ . B odrzuca ten pakiet.



# Wybijanie dziur (*hole punching*)

## Co z asymetrycznymi ograniczonymi (portowo) NAT?

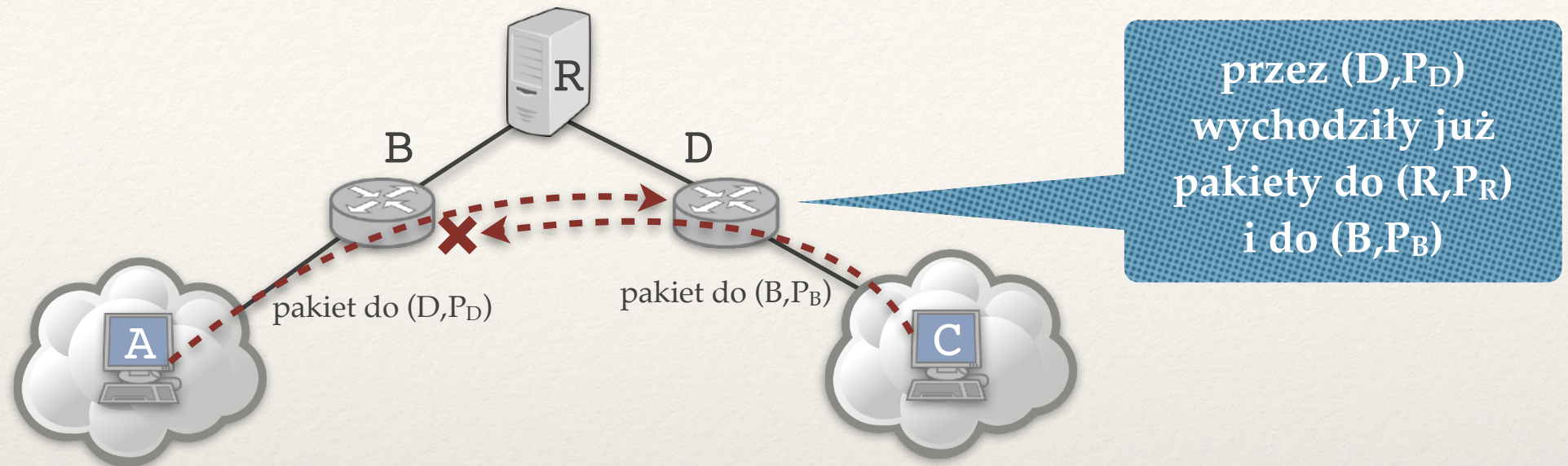


- ❖  $(C, P_C)$  wysyła pakiet do  $(B, P_B)$ . B odrzuca ten pakiet.
- ❖ Na routerze D:  $(B, P_B)$  dodany do listy odbiorców pakietów wychodzących przez  $(D, P_D)$ !



# Wybijanie dziur (*hole punching*)

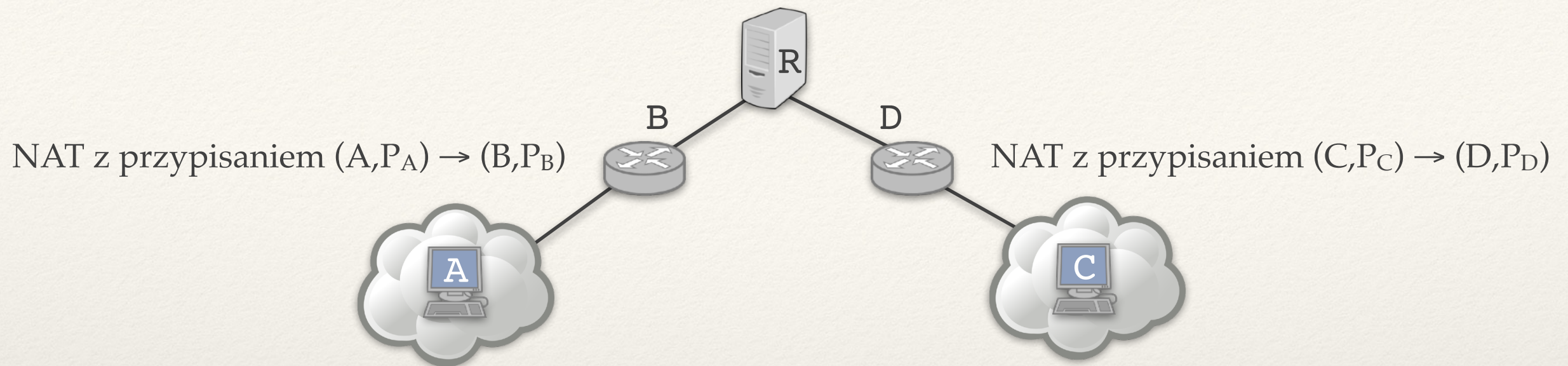
## Co z asymetrycznymi ograniczonymi (portowo) NAT?



- ❖  $(C, P_C)$  wysyła pakiet do  $(B, P_B)$ . B odrzuca ten pakiet.
- ❖ Na routerze D:  $(B, P_B)$  dodany do listy odbiorców pakietów wychodzących przez  $(D, P_D)$ !
- ❖  $(A, P_A)$  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



- ❖ Milcząco założyliśmy, że jeśli  $(A, P_A)$  wysyłało pakiet do  $(R, P_R)$  i potem do  $(D, P_D)$ , to w obu przypadkach B wybierze port  $P_B$ .
- ❖ **NAT asymetryczny:**  $P_B$  zależy tylko od adresu i portu nadawcy.
- ❖ **NAT symetryczny:**  $P_B$  zależy od adresu i portu nadawcy i odbiorcy. Wybijanie dziur nie działa, pomagają tylko przekąźniki.



# Lektura dodatkowa

---

- ❖ Kurose & Ross: rozdział 2.
- ❖ Tanenbaum: rozdział 7.



# 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()`?
- ❖ Do czego służy protokół SMTP a do czego POP3?
- ❖ Co to jest przekazywanie poczty (*relaying*)? Co to jest *smarthost*?
- ❖ 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`?
- ❖ Na czym polega greylisting?
- ❖ Na czym polega mechanizm SPF?
- ❖ Co umożliwia standard MIME?
- ❖ Co to jest spam? Jakie znasz metody walki ze spamem?
- ❖ Jaka jest rola trackera w sieci Bittorrent?
- ❖ Po co w plikach `.torrent` stosuje się funkcje skrótu?
- ❖ Jakie są różnice w postępowaniu *seeder*a i *leech*era w sieci BitTorrent?
- ❖ Na czym polegają połączenia odwrócone? Jak stosuje się je w protokole FTP?
- ❖ Opisz podobieństwa i różnice asymetrycznych (*cone*) NAT (pełnego, ograniczonego i ograniczonego portowo) i symetrycznych NAT.
- ❖ Opisz technikę wybijania dziur (*hole punching*) w NAT. Po co konieczny jest serwer pośredniczący?