
Routing

część 3: wewnątrz routera

Sieci komputerowe

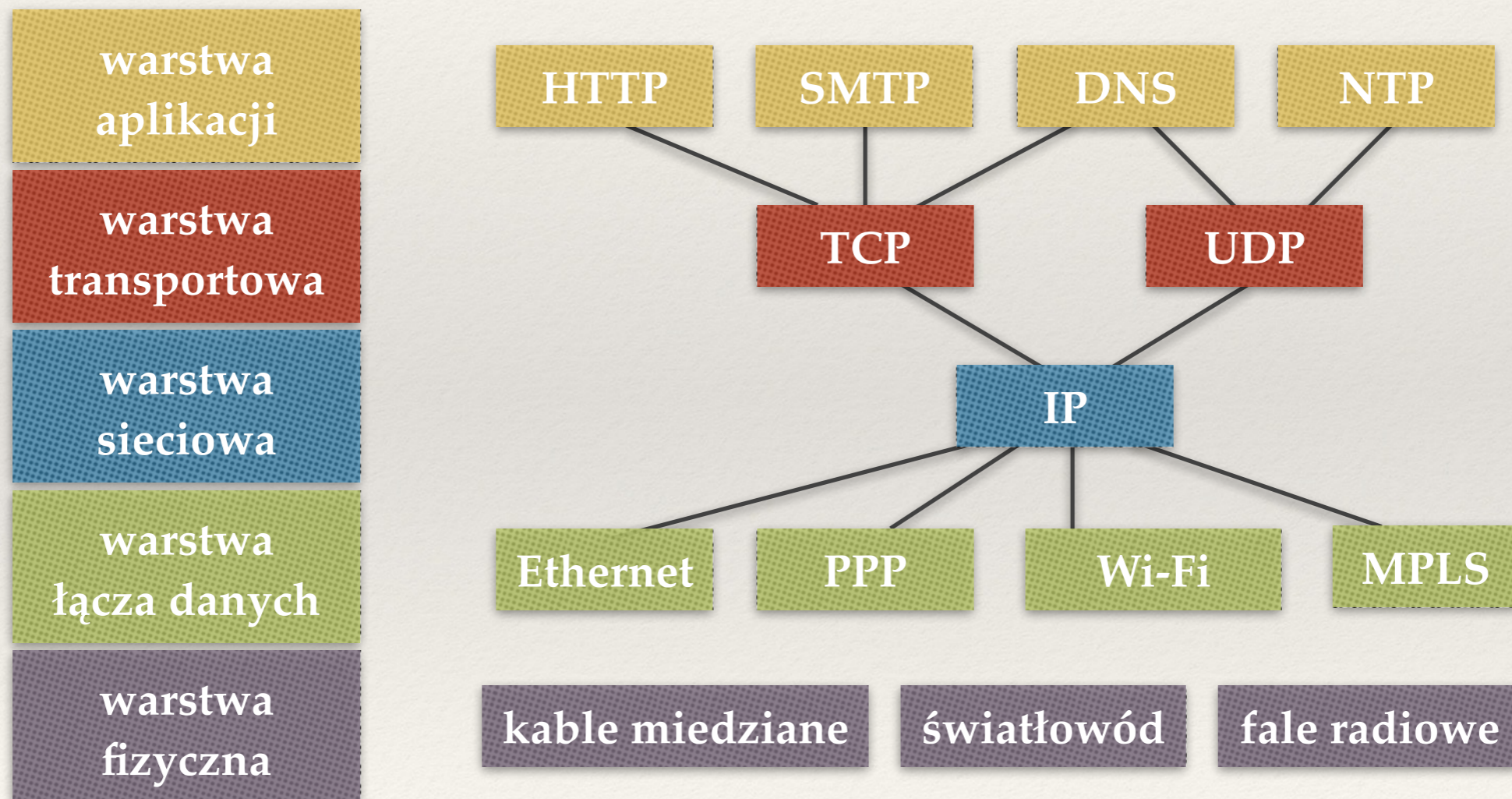
Wykład 4

Marcin Bieńkowski

Dygresja: gniazda UDP

Jedna warstwa sieci i globalne adresowanie

- ❖ Każde urządzenie w sieci posługuje się tym samym protokołem warstwy sieci. W Internecie: protokół IP.
- ❖ Każde urządzenie ma unikatowy adres. W Internecie: adresy IP.



Gniazda

Interfejs programistyczny do nadawania i odbierania pakietów.

- ❖ Umożliwiają podawanie danych do umieszczenia w datagramach UDP lub segmentach TCP.



dostęp do niektórych pól za pomocą funkcji gniazd

- ❖ Gniazda surowe: umożliwiają podawanie danych do umieszczenia bezpośrednio w danych pakietu IP.



Nagłówek pakietu IP

0	7	8	15	16	23	24	31
wersja	IHL	typ usługi	całkowita długość pakietu				
pola związane z fragmentacją pakietu							
TTL		protokół		suma kontrolna nagłówka IP			
źródłowy adres IP							
docelowy adres IP							

- ❖ Protokół = co jest przechowywane w danych pakietu (np. 1 = ICMP, 6 = TCP, 17 = UDP).

Nagłówek UDP (User Data Protocol)

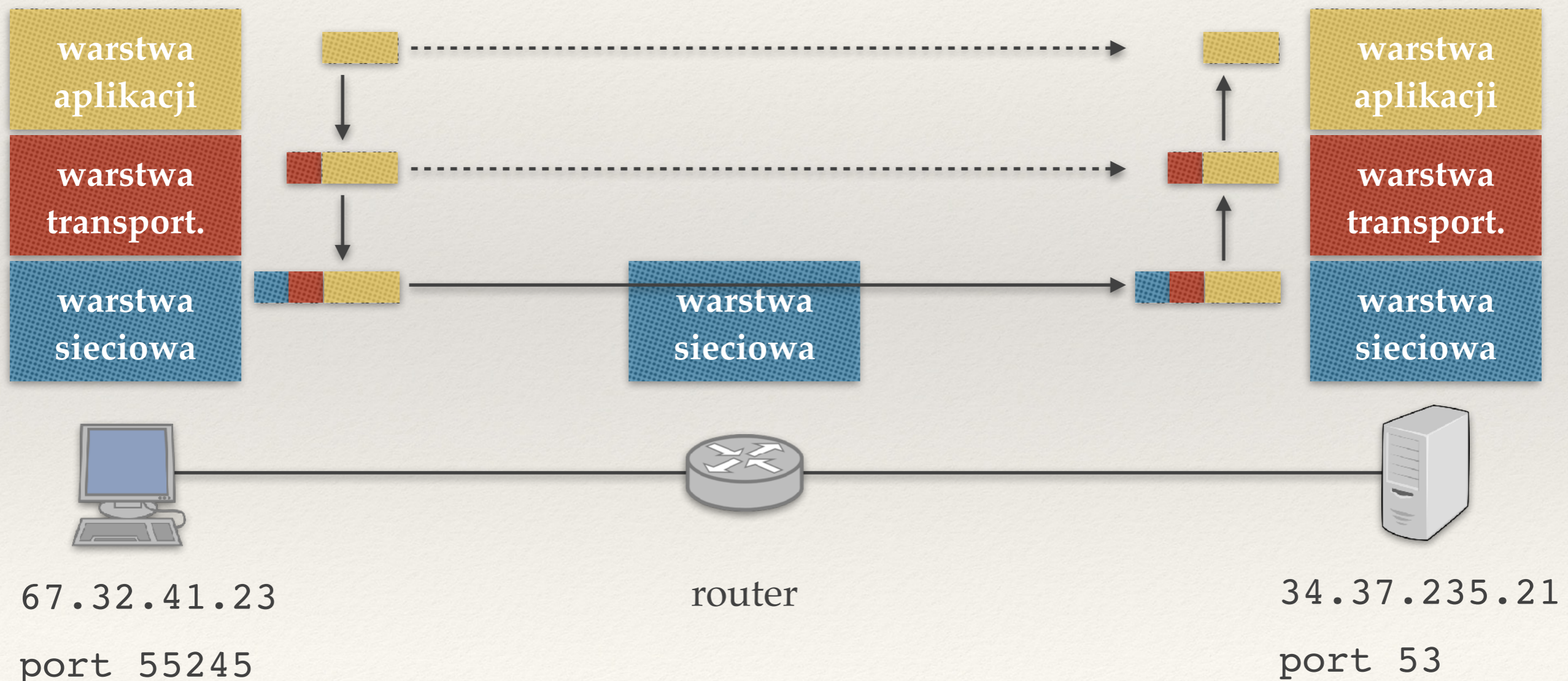
0	7 8	15 16	23 24	31
port źródłowy		port docelowy		
długość		suma kontrolna		

- ❖ Port:
 - ♦ liczba 16-bitowa;
 - ♦ identyfikuje aplikację wewnątrz danego komputera.
- ❖ Warstwa sieciowa zapewnia dostarczanie pakietów pomiędzy komputerami, warstwa transportowa pomiędzy aplikacjami.

Enkapsulacja i dekapulsacja



przesyłany pakiet



Gniazdo UDP

- ❖ Identyfikuje jeden koniec komunikacji UDP.
- ❖ Opisywane przez parę (lokalny adres IP, lokalny port).
- ❖ Związane z konkretnym procesem.

Tworzenie gniazda

```
#include <arpa/inet.h>  
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

Wiązanie gniazda z adresem i portem

Jak w przypadku gniazda surowego, ale wypełniamy też port.

```
struct sockaddr_in server_address = {0};
server_address.sin_family          = AF_INET;
server_address.sin_port           = htons(32345);
server_address.sin_addr.s_addr    = htonl(INADDR_ANY);

bind (
    sockfd,
    (struct sockaddr*)&server_address,
    sizeof(server_address)
);
```

Wiązanie gniazda z adresem i portem

Jak w przypadku gniazda surowego, ale wypełniamy też port.

```
struct sockaddr_in server_address = {0};
server_address.sin_family          = AF_INET;
server_address.sin_port           = htons(32345);
server_address.sin_addr.s_addr    = htonl(INADDR_ANY);

bind (
    sockfd,
    (struct sockaddr*)&server_address,
    sizeof(server_address)
);
```

demonstracja

Odbieranie pakietu z gniazda

Jak w gniazdach surowych.

```
struct sockaddr_in  sender;  
socklen_t          sender_len = sizeof(sender);  
uint8_t            buffer[IP_MAXPACKET+1];  
  
ssize_t packet_len = recvfrom(  
    sockfd,  
    buffer, ← pakiet jako ciąg bajtów  
    IP_MAXPACKET,  
    0,  
    (struct sockaddr*)&sender,  
    &sender_len } adres nadawcy  
);
```

Wysyłanie pakietu przez gniazdo

Jak w gniazdach surowych, ale adres odbiorcy zawiera również port.

```
char* reply = "Thank you!";  
size_t reply_len = strlen(reply);
```

```
ssize_t bytes_sent = sendto(  
    sockfd,  
    reply, } ← dowolny ciąg bajtów,  
    reply_len, } niekoniecznie napis  
    0,  
    (struct sockaddr*)&recipient, } adres odbiorcy  
    sizeof(recipient)  
);
```

Zamykanie gniazda

Zwalnia zasoby związane z gniazdem.

```
close(sockfd);
```

Kod serwera UDP (przetwarzanie pakietów)

```
bind(sockfd, (struct sockaddr*)&server_address, sizeof(server_address));

for (;;) {
    struct sockaddr_in sender;
    socklen_t sender_len = sizeof(sender);
    uint8_t buffer[IP_MAXPACKET+1];
    ssize_t datagram_len = recvfrom(sockfd, buffer, IP_MAXPACKET, 0,
                                     (struct sockaddr*)&sender, &sender_len);

    char sender_ip_str[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &(sender.sin_addr), sender_ip_str, sizeof(sender_ip_str));
    printf("Received UDP packet from IP address: %s, port: %u\n",
          sender_ip_str, ntohs(sender.sin_port));

    buffer[datagram_len] = 0;
    printf("%zd-byte message: +%s+\n", datagram_len, buffer);
    char* reply = "Thank you!";
    size_t reply_len = strlen(reply);
    sendto(sockfd, reply, reply_len, 0, (struct sockaddr*)&sender, sender_len);
}

close(sockfd);
```

Kod serwera UDP (przetwarzanie pakietów)

```
bind(sockfd, (struct sockaddr*)&server_address, sizeof(server_address));

for (;;) {
    struct sockaddr_in sender;
    socklen_t sender_len = sizeof(sender);
    uint8_t buffer[IP_MAXPACKET+1];
    ssize_t datagram_len = recvfrom(sockfd, buffer, IP_MAXPACKET, 0,
                                     (struct sockaddr*)&sender, &sender_len);

    char sender_ip_str[INET_ADDRSTRLEN];
    inet_ntop(AF_INET, &(sender.sin_addr), sender_ip_str, sizeof(sender_ip_str));
    printf("Received UDP packet from IP address: %s, port: %u\n",
           sender_ip_str, ntohs(sender.sin_port));

    buffer[datagram_len] = 0;
    printf("%zd-byte message: +%s+\n", datagram_len, buffer);
    char* reply = "Thank you!";
    size_t reply_len = strlen(reply);
    sendto(sockfd, reply, reply_len, 0, (struct sockaddr*)&sender, sender_len);
}

close(sockfd);
```

udp_server.c
kod (z obsługą błędów) na stronie wykładu

demonstracja

Wiązanie z portem

- ❖ Serwer związuje się z danym portem funkcją `bind()`.
 - ◆ Do związania z portem ≤ 1024 potrzebne uprawnienia administratora.
- ❖ Wysyłka przez gniazdo bez portu \rightarrow gniazdo dostanie port wybrany przez jądro.
 - ◆ Port tymczasowy (zazwyczaj ≥ 32768).
 - ◆ Tak działa większość klientów (np. program `nc`).

Kod klienta UDP

```
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in server_addr = {0};
server_address.sin_family      = AF_INET;
server_address.sin_port       = htons(32345);
inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);
const char* message = "Hello server!";
sendto(sockfd, message, strlen(message), 0,
        (struct sockaddr*) &server_addr, sizeof(server_addr));

close(sockfd);
```

Kod klienta UDP

```
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

struct sockaddr_in server_addr = {0};
server_address.sin_family      = AF_INET;
server_address.sin_port        = htons(32345);
inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);
const char* message = "Hello server!";
sendto(sockfd, message, strlen(message), 0,
        (struct sockaddr*) &server_addr, sizeof(server_addr));

close(sockfd);
```

udp_client.c

kod (z obsługą błędów) na stronie wykładu

demonstracja

Wysyłanie pakietu UDP na adres rozgłoszeniowy

Trzeba włączyć odpowiednią opcję gniazda.

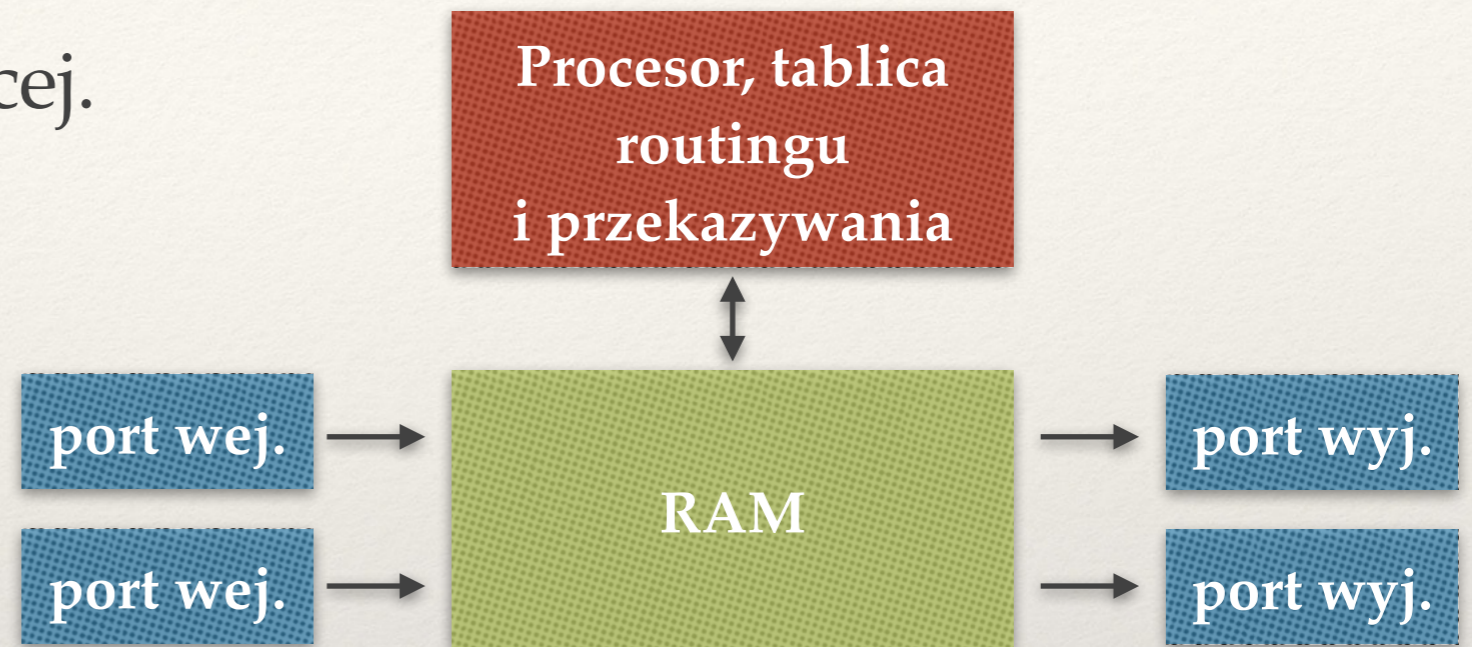
```
int so_value = 1;
setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST,
           (void*)&so_value, sizeof(so_value));
```

Wewnątrz routera

Przełączanie pakietów: przez RAM

Wczesne generacje routerów (jak PC):

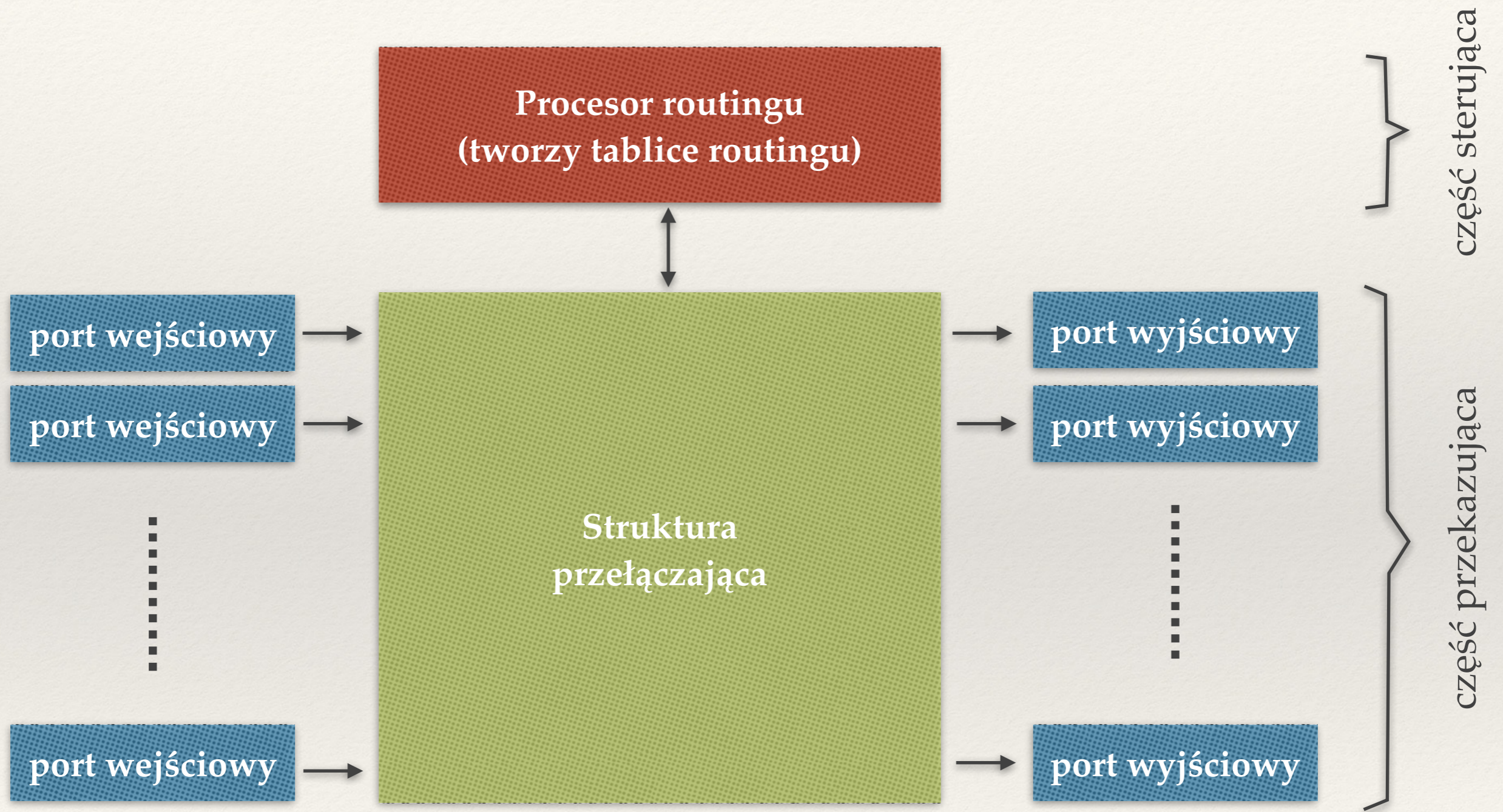
- ❖ Brak struktury przełączającej.
- ❖ Tablica przekazywania w części sterującej.



Działanie:

- ❖ Port wejściowy odbiera pakiet i zgłasza przerwanie.
- ❖ Procesor kopiuje pakiet do RAM.
- ❖ Wolny port wyjściowy zgłasza przerwanie.
- ❖ Procesor kopiuje pakiet z RAM.

Budowa współczesnego routera (1)



Budowa współczesnego routera (2)

❖ **Procesor:**

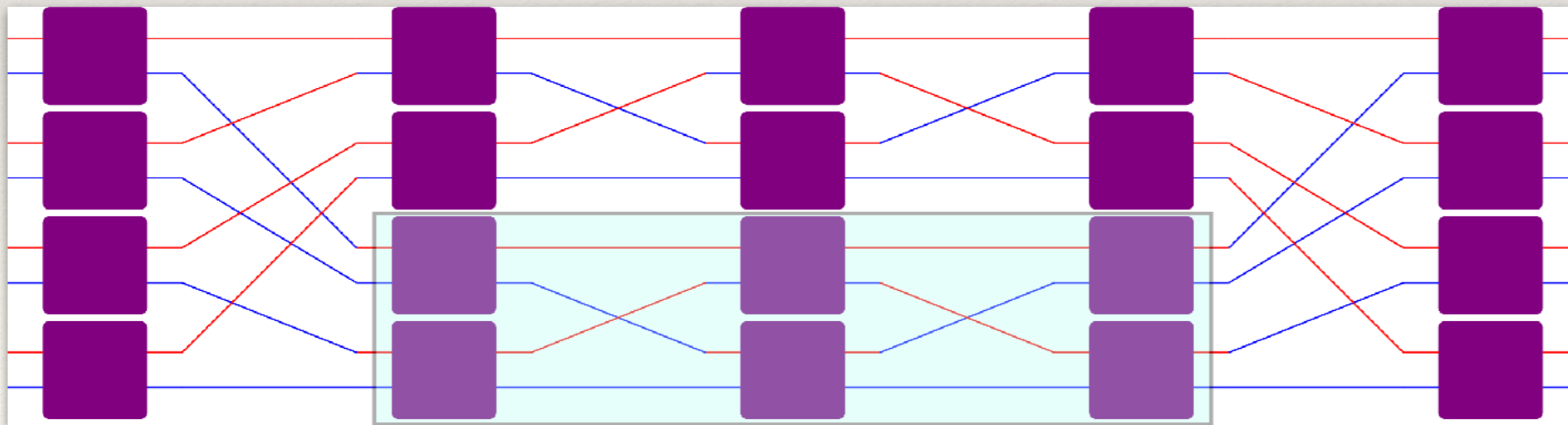
- ◆ Otrzymuje niektóre pakiety (RIP, OSPF, BGP).
- ◆ Tworzy tablice przekazywania i wysyła je do portów wejściowych.

❖ **Port wejściowy:**

- ◆ Odbiera pakiet z łącza.
- ◆ Uaktualnia nagłówki IP (TTL, suma kontrolna).
- ◆ Sprawdza, do którego portu wyjściowego go przesłać.

Struktura przełączająca

- ❖ Chcemy przekazywać pakiety z prędkością łącza.
- ❖ Dla n portów wejściowych o prędkości $R \rightarrow$ chcemy przepustowość $n \times R$ (typowe wartości to 10 Gbit/s - 1 Tbit/s).
- ❖ Stosowane sieci połączeń:
 - ✦ każdy z każdym: $O(n^2)$ połączeń (niepraktyczne dla dużych n);
 - ✦ sieci Beneša i pochodne: $O(n \log n)$ połączeń (potrafią bezkolizyjnie przesłać dowolną permutację).



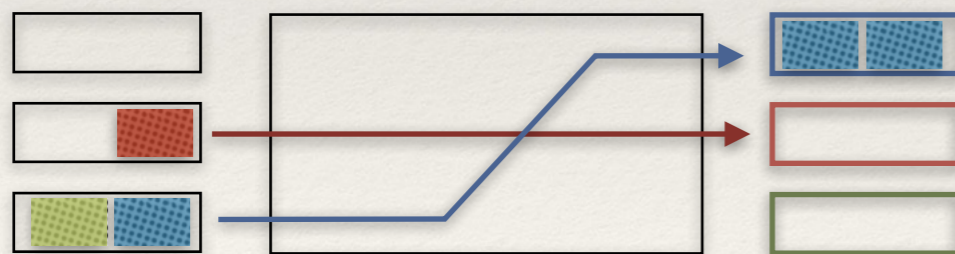
Bufory z kolejkami pakietów

❖ Przy portach wyjściowych.

- ❖ Zapobiegają utracie pakietów przy czasowym zwiększeniu liczby pakietów.

❖ Przy portach wejściowych.

- ❖ Jeśli przepustowość struktury przełączającej jest za mała.
- ❖ Pakiety kierowane do zajętych łączy wyjściowych są blokowane.
- ❖ Problem blokowania przodu kolejki.



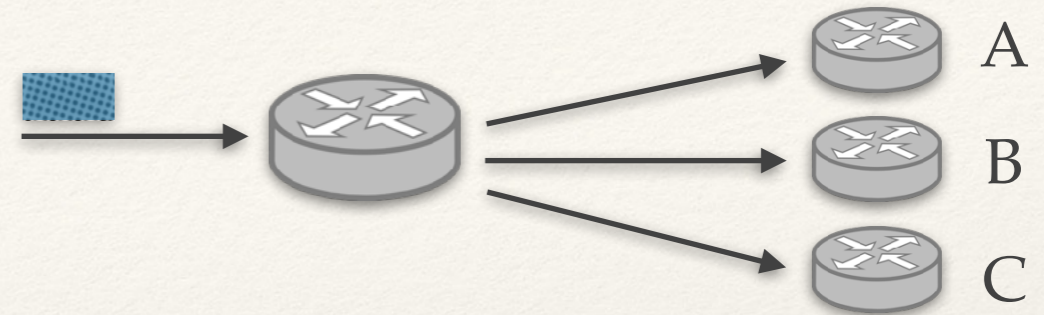
Niebieski pakiet musi czekać i blokuje wysłanie pakietu zielonego.

Rozwiązywane przez wirtualne kolejki pakietów: jedna kolejka dla każdego portu wyjściowego.

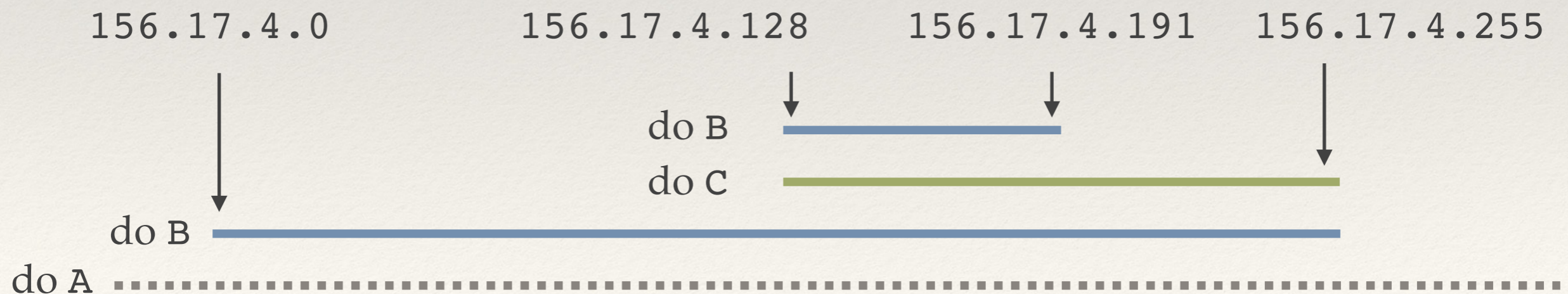
Porty wejściowe

Reguła najdłuższego pasującego prefiksu

prefiks CIDR	akcja
0.0.0.0/0	do routera A
156.17.4.0/24	do routera B
156.17.4.128/25	do routera C
156.17.4.128/26	do routera B



- ❖ **LPM** (*longest prefix match*): z pasujących reguł wybierana jest najdłuższa.



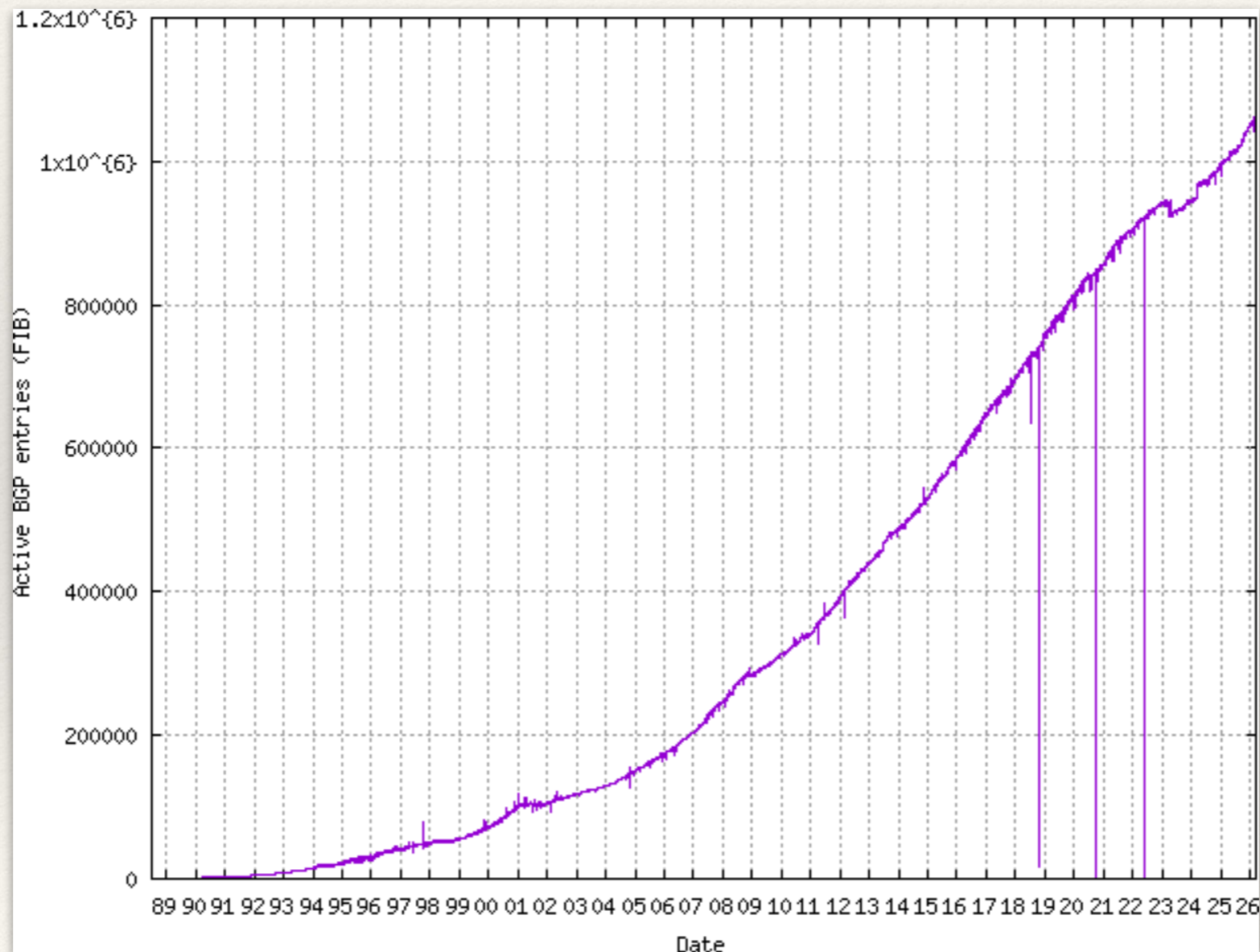
Struktura danych dla LPM

- ❖ Musi obsługiwać:
 - ♦ **lookup (adres)** — dziesiątki milionów razy / sek.
 - ♦ **insert (prefix) / delete (prefix)** — setki razy / sek.

- ❖ Notacja:
 - ♦ n - liczba prefiksów w tablicy;
 - ♦ w - rozmiar adresu (adres mieści się w słowie maszyny).

Ile prefiksów w routerach

- ❖ ~ 1060 tys na początku 2026.
- ❖ Mogłoby być ~600 tys, gdyby każdy ISP agregował rozgłaszane prefiksy CIDR.



Obrazek ze strony <https://www.cidr-report.org/as2.0/>

Implementacja LPM (1)

Lista prefiksów:

- ❖ pamięć: $O(n)$,
- ❖ lookup: $O(n)$,
- ❖ insert: $O(1)$, delete: $O(n)$.

Implementacja LPM (2)

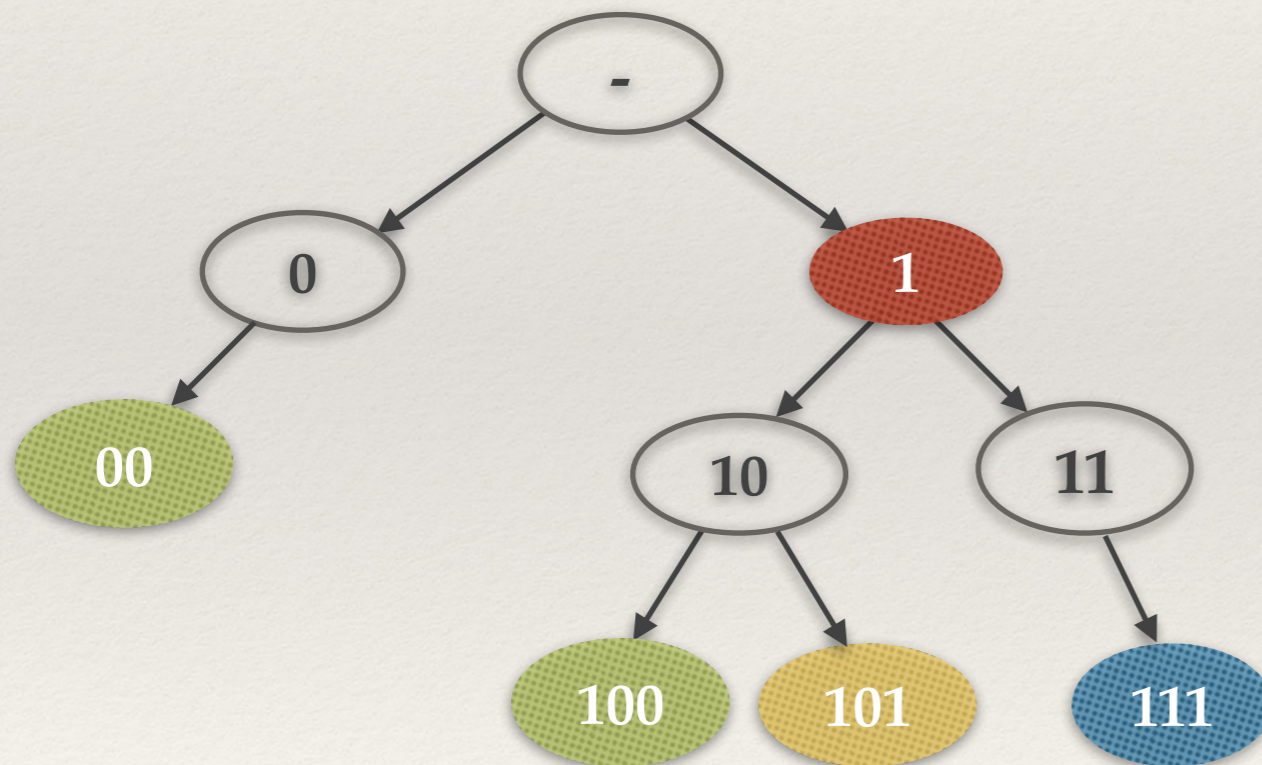
Tablice haszujące (starsze systemy uniksowe):

- ❖ $w+1$ tablic (dla każdej możliwej długości prefiksu; w czasach klas adresów IP wystarczało 5 tablic);
- ❖ pamięć: $O(n)$,
- ❖ lookup: $O(w)$ (oczekiwany),
- ❖ insert, delete: $O(1)$ (oczekiwany).

Implementacja LPM (3)

Drzewa trie (nowsze systemy uniksowe, routery sprzętowe):

- ❖ pamięć: $O(n \cdot w)$,
- ❖ lookup: $O(w)$,
- ❖ insert, delete: $O(w)$.



Przechodzimy drzewo w dół i zwracamy ostatnią pasującą regułę:

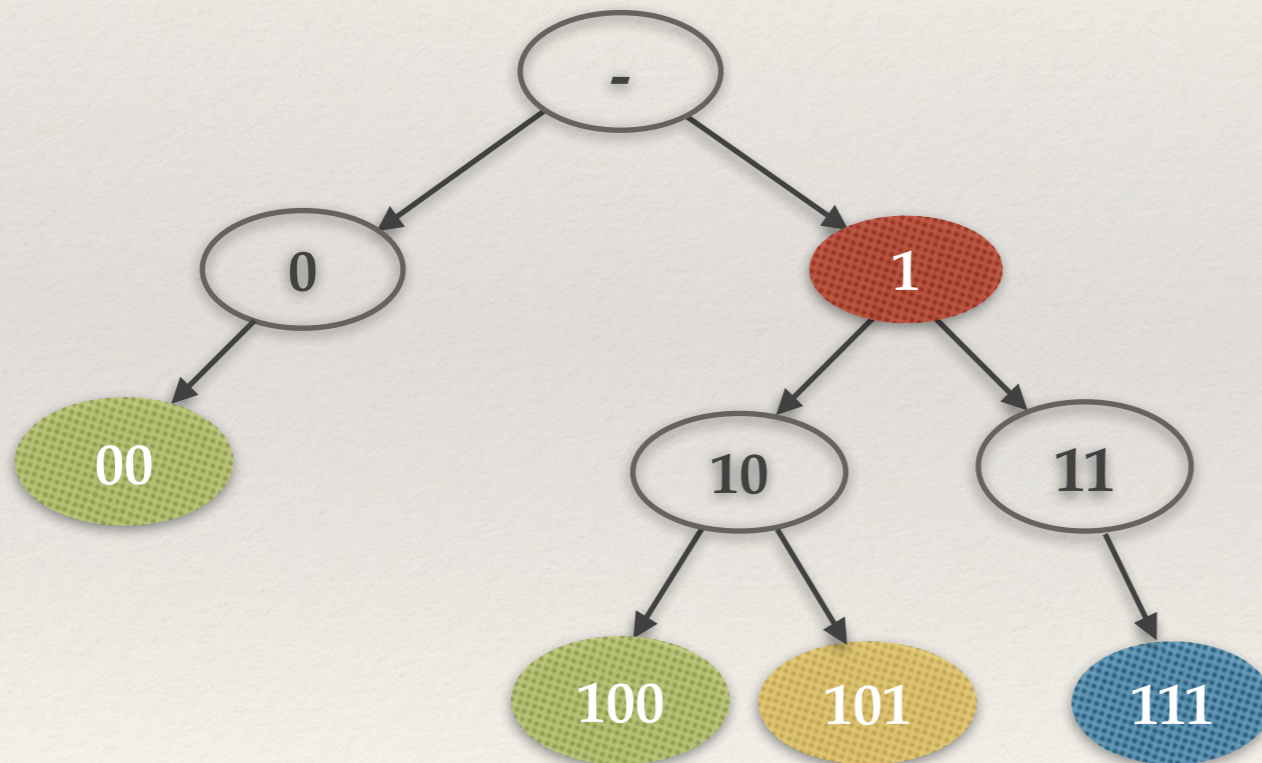
- ❖ dla adresu 10000... → port zielony;
- ❖ dla adresu 11000... → port czerwony.

Implementacja LPM (3)

Drzewa trie (nowsze systemy uniksowe, routery sprzętowe):

- ❖ pamięć: $O(n \cdot w)$,
- ❖ lookup: $O(w)$,
- ❖ insert, delete: $O(w)$.

$O(n)$ z kompresją ścieżek bez rozgałęzień



Przechodzimy drzewo w dół i zwracamy ostatnią pasującą regułę:

- ❖ dla adresu 10000... → port zielony;
- ❖ dla adresu 11000... → port czerwony.

Implementacja LPM (4)

Trie ze dodatkowymi krawędziami skrcającymi:

- ❖ lookup: $O(\log w)$,
- ❖ insert, delete: $O(n)$.
- ❖ Można zrobić operacje insert/delete w $O(\log w)$, ale stała w notacji O jest niepraktyczna.

Implementacja LPM (5)

- ❖ **TCAM (nowsze routery sprzętowe):**
 - ♦ TCAM = *ternary content addressable memory*.
 - ♦ Przechowuje pary $(p, m) = (\text{prefix}, \text{maska})$.
 - ♦ Dla adresu w można **równoległe** znaleźć wszystkie pary takie, że $w \& m = p \& m$, tj. wszystkie pasujące prefiksy.
 - ♦ Sprzętowo wybieramy najdłuższy z nich.

Porty wyjściowe

MTU (maximum transmission unit)

Własność warstwy łącza danych i fizycznej łącza wyjściowego.

- ❖ Maksymalny rozmiar pakietu IP (wraz z nagłówkiem).
- ❖ MTU Ethernetu: 1500 bajtów.
- ❖ (Teoretyczne) MTU sieci bezprzewodowej Wi-Fi: 7981 bajtów.

Fragmentacja

Jeśli rozmiar pakietu \geq MTU łącza wyjściowego \rightarrow pakiet jest dzielony na fragmenty.

- ❖ Dzielenie może nastąpić na dowolnym routerze na trasie.
- ❖ Łączenie fragmentów dopiero na komputerze docelowym.

Fragmentacja a nagłówek IP

0	7	8	15	16	23	24	31
wersja	IHL	typ usługi	całkowita długość pakietu				
identyfikator przy fragmentacji			0	D F	M F	offset fragmentu	
TTL	protokół		suma kontrolna nagłówka IP				
źródłowy adres IP							
docelowy adres IP							

- ❖ Fragmenty dostają identyczny identyfikator.
- ❖ MF = czy jest więcej fragmentów?
- ❖ Offset = numer pierwszego bajtu w oryginalnym pakiecie.

Fragmentacja jest nieefektywna

- ❖ Dodatkowa praca dla routerów.
- ❖ Dodatkowy narzut, przykładowo:
 - ♦ Wysyłamy 140 000 bajtów.
 - ♦ MTU pierwszego łącza = 1400.
 - ♦ MTU drugiego łącza = 1250.
 - ♦ Bez fragmentacji: $140\ 000 / 1250 = 112$ pakietów.
 - ♦ Z fragmentacją: wysyłamy $140\ 000 / 1400 = 100$ pakietów, ale każdy dzielony później na dwa.

Fragmentacja jest nieefektywna

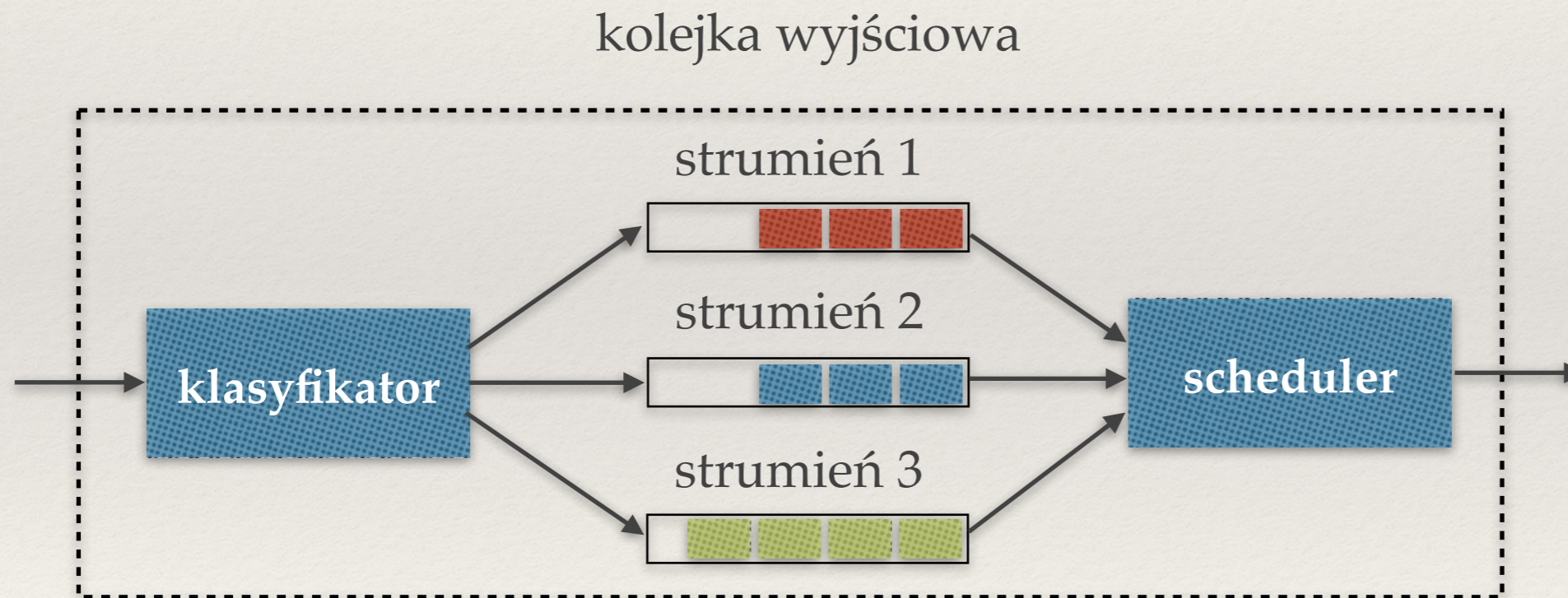
- ❖ Dodatkowa praca dla routerów.
- ❖ Dodatkowy narzut, przykładowo:
 - ♦ Wysyłamy 140 000 bajtów.
 - ♦ MTU pierwszego łącza = 1400.
 - ♦ MTU drugiego łącza = 1250.
 - ♦ Bez fragmentacji: $140\ 000 / 1250 = 112$ pakietów.
 - ♦ Z fragmentacją: wysyłamy $140\ 000 / 1400 = 100$ pakietów, ale każdy dzielony później na dwa.
- ❖ **Jak poznać najmniejsze MTU na trasie?**

Wykrywanie minimalnego MTU na ścieżce

- ❖ Ustaw bit DF (*don't fragment*) w nagłówku IP.
- ❖ Jeśli konieczna fragmentacja na routerze:
 - ♦ pakiet wyrzucony;
 - ♦ router odsyła komunikat ICMP (*destination unreachable, can't fragment*) z rozmiarem MTU kolejnego łącza.
- ❖ Zmniejsz odpowiednio rozmiar pakietu i ponów wysyłanie.

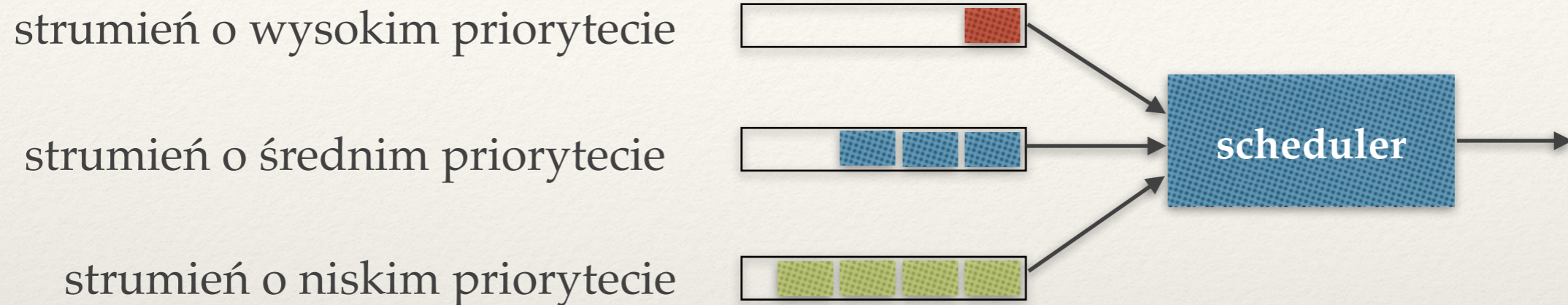
Co się dzieje w kolejce wyjściowej?

- ❖ **Kolejka FIFO:** pakiety wysyłane w takiej kolejności jak nadeszły.
- ❖ **Szeregowanie pakietów:** Przypisujemy pakiety do strumieni (na podstawie adresu i portu źródłowego + docelowego). Pakiety szeregowane w zależności od strumienia.

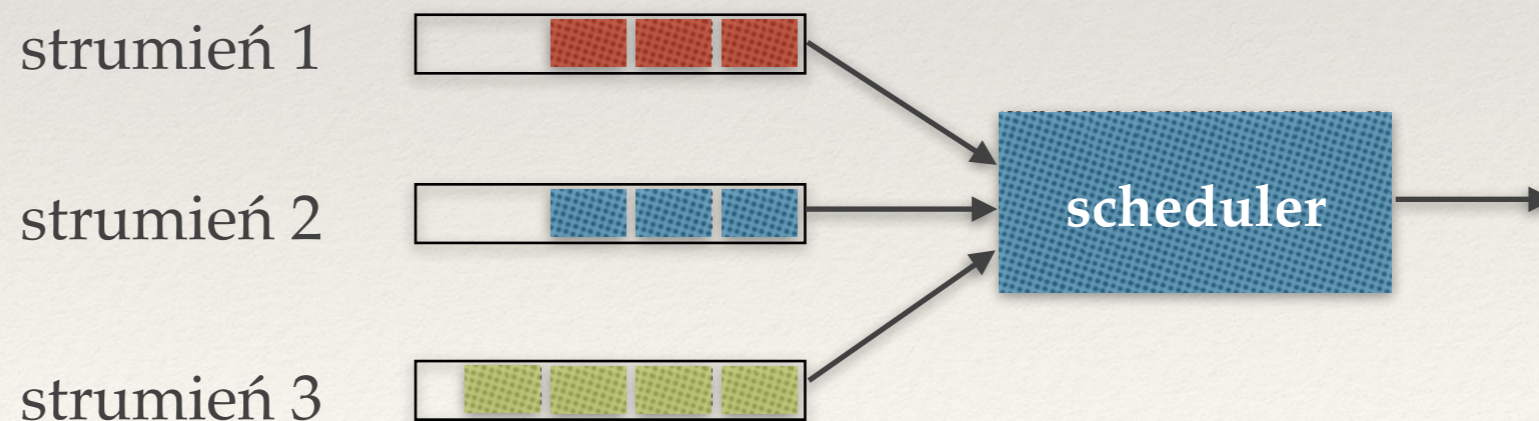


Szeregowanie pakietów w buforze

❖ Względem priorytetów strumieni



❖ Cykliczne (*round-robin*): po tyle samo pakietów z każdego strumienia.



Anycast

Rodzaje adresów IP

- ❖ Adresy zwykłe (**unicast**).
- ❖ Adresy **broadcast**: pakiet wysłany na taki adres dociera do wszystkich z danej sieci.
- ❖ Adresy **multicast**: pakiet wysłany na taki adres dociera do zdefiniowanej grupy.
- ❖ Adresy **anycast**: wiele serwerów ma taki sam adres IP.

Adresy anycast

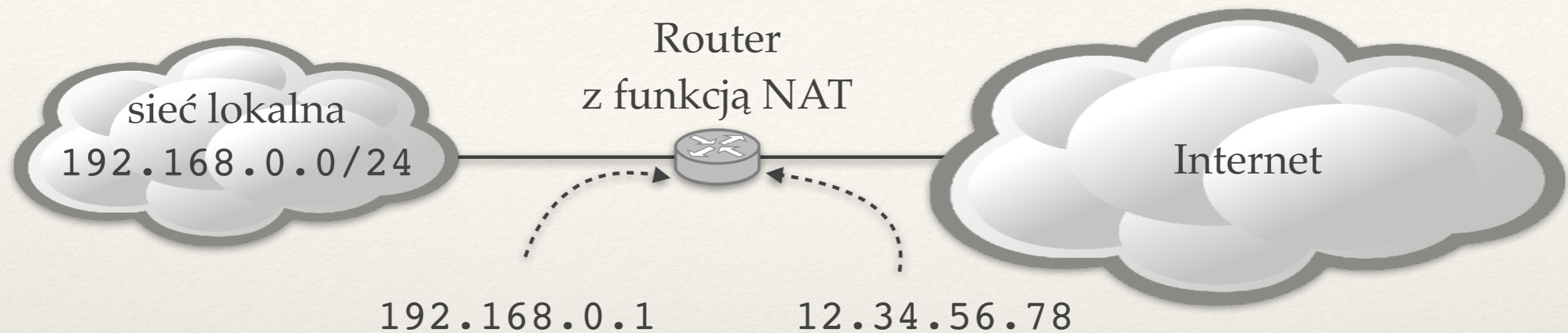
- ❖ Obsługa przez standardowe protokoły routingu dynamicznego → routery poznają trasę do **najbliższego** serwera z danym adresem.
- ❖ Potencjalny problem: wszystkie pakiety z jednego strumienia danych powinny być do jednego serwera.
 - ◆ To nie problem w przypadku protokołu DNS, bo pytanie i odpowiedź mieszczą się w jednym pakiecie UDP.
 - ◆ Istnieje ~1000 serwerów o adresach 8.8.8.8 i 1.1.1.1.
 - ◆ W praktyce problem występuje rzadko. → Anycast stosowany też np. w HTTP.

NAT

Coraz większe zapotrzebowanie na adresy IP

- ❖ Adresy IPv4 wyczerpują się.
- ❖ Wdrożenie IPv6 wciąż trwa.
- ❖ Adresy IP są dość kosztowne → pojedyncze IP dla całych firm.

NAT

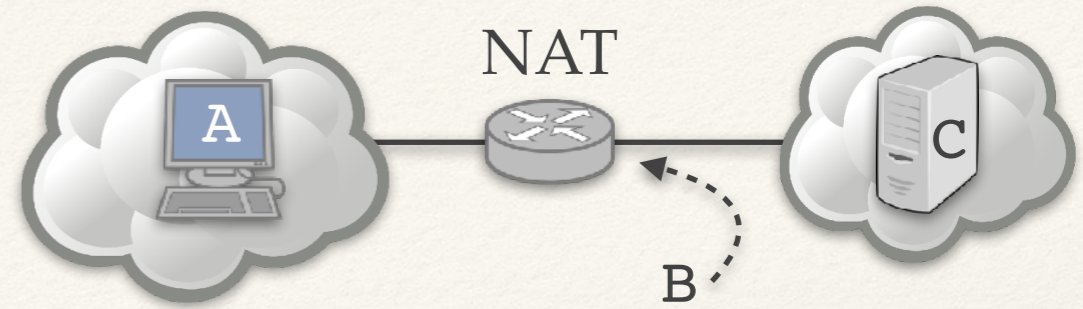


- ❖ Z reszty Internetu cała sieć lokalna wygląda tak samo, jak pojedynczy komputer z adresem 12.34.56.78.
- ❖ Nie można (w normalny sposób) dostać się z Internetu do komputerów z LAN.

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 odpowiedź do (B, P_B) to jej adres i port docelowy zostanie podmieniony na (A, P_A).

Adresy prywatne

Adresy przeznaczone do sieci lokalnych.

- ❖ Pakiety z takimi adresami nie są przekazywane przez routery.
- ❖ W różnych sieciach mogą być te same adresy.
- ❖ Pule adresów:
 - ◆ $10.0.0.0/8$ (jedna sieć klasy A);
 - ◆ $172.16.0.0/12$ (16 sieci klasy B);
 - ◆ $192.168.0.0/16$ (256 sieci klasy C).
 - ◆ $fd00::/8$ (dla IPv6).

Zalety i wady NAT

Zalety:

- ❖ Rozwiązuje problem braku adresów IP.
- ❖ Można zmienić adresy IP wewnątrz sieci bez powiadamiania Internetu.
- ❖ Można zmienić ISP pozostawiając adresowanie IP wewnątrz sieci.

Wady:

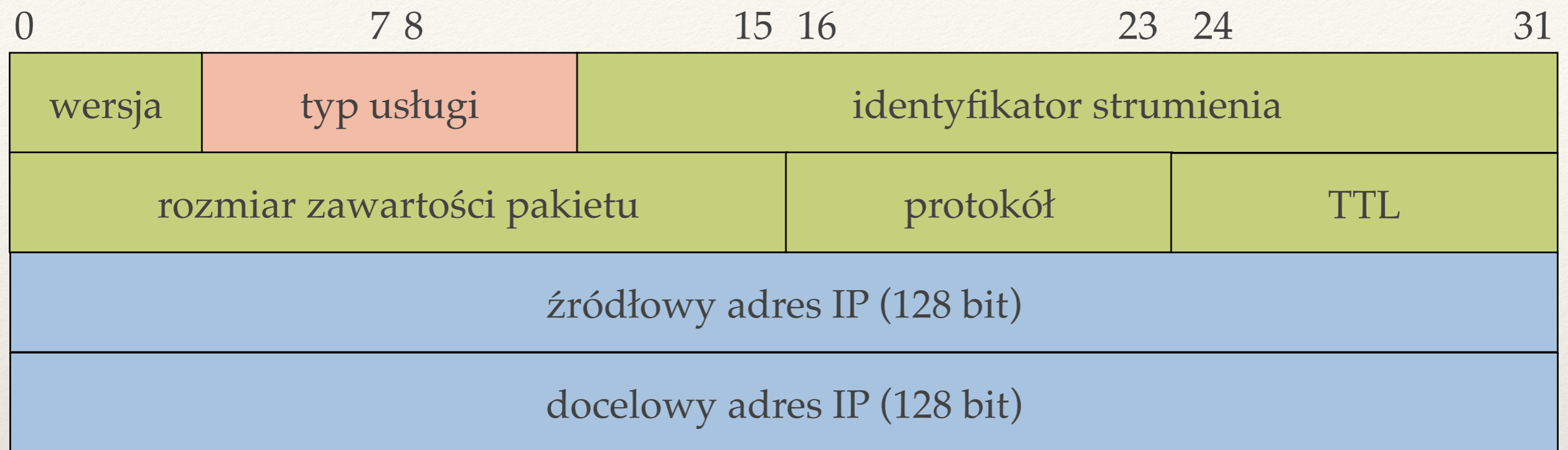
- ❖ Nieosiągalność komputerów z Internetu (aplikacje P2P).
- ❖ Psucie modelu warstwowego (router modyfikuje treść pakietu).

IPv6

Dlaczego nowa wersja IP?

- ❖ Adresy IPv4 wyczerpują się (IANA oddała ostatnią pulę regionalnym rejestratorom w 2011 r.).
- ❖ W 1994 rozpoczęto pracę nad nową wersją (IPv6).
- ❖ 128-bitowe adresy.

Nagłówek IPv6



Mniejszy koszt dla routerów:

- ❖ nagłówki stałej długości;
- ❖ brak fragmentacji;
- ❖ brak sumy kontrolnej;
- ❖ etykieta strumienia (nie trzeba patrzeć na porty).

Adresy IPv6

❖ Notacja:

- ♦ 8 bloków po 4 cyfry szesnastkowe, rozdzielonych przez dwukropek.
- ♦ Przykładowo $A = 2001:0db8:0000:0000:0000:0000:1428:0000$.
- ♦ Localhost = $0000:0000:0000:0000:0000:0000:0000:0001/128$.

❖ Uproszczenia zapisu:

- ♦ Można opuszczać wiodące zera w każdym bloku (do niepustego ciągu).
- ♦ **Jeden** ciąg zerowych bloków zer można zastąpić przez $::$
- ♦ Przykłady:
 - ♦ $A = 2001:db8::1428:0$,
 - ♦ localhost = $::1/128$,
 - ♦ Pula adresów prywatnych dla IPv6: $fd00::/8$.

Sieci IPv6

- ❖ Typowy prefix dla sieci lokalnej w IPv6 to /64.
- ❖ Prywatni odbiorcy zazwyczaj dostają prefix /56 lub /60.
- ❖ Organizacje zazwyczaj dostają prefix /48.
- ❖ Brak adresu rozgłoszeniowego (broadcast).
- ❖ Zdefiniowane wiele specjalnych adresów multicastowych, np:
 - ♦ `ff02::1` - wszystkie adresy w sieci lokalnej (jak broadcast);
 - ♦ `ff02::2` - wszystkie routery w sieci lokalnej.

- ❖ Oferuje funkcje analogiczne do ICMPv4.
- ❖ Może zastępować DHCP.
- ❖ Zastępuje ARP.
- ❖ ARP i DHCP poznamy na kolejnych wykładach.

Migracja do IPv6

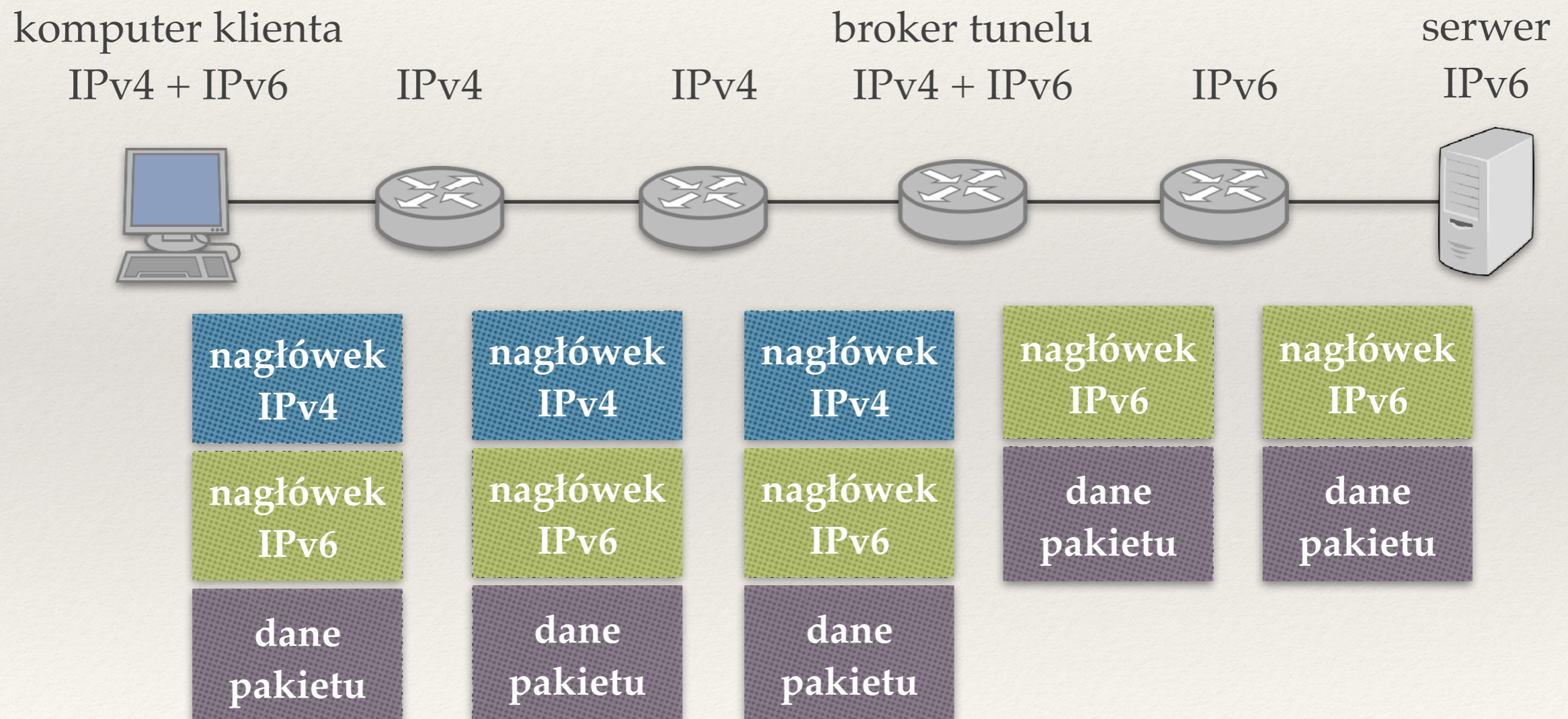
- ❖ Większość dużych serwisów (Google, Facebook, ...) ma swoje wersje IPv6.
 - ✦ Osobne serwery lub serwery z podwójnym stosem (potrafią interpretować pakiety IPv4 i IPv6).
- ❖ Duża część routerów w rdzeniu Internetu potrafi przesyłać pakiety IPv6.

Migracja do IPv6

- ❖ Większość dużych serwisów (Google, Facebook, ...) ma swoje wersje IPv6.
 - ✦ Osobne serwery lub serwery z podwójnym stosem (potrafią interpretować pakiety IPv4 i IPv6).
- ❖ Duża część routerów w rdzeniu Internetu potrafi przesyłać pakiety IPv6.
- ❖ **Co zrobić, jeśli router naszego ISP nie ma adresu IPv6?**

Mechanizmy migracji: tunelowanie 6in4

- ❖ Pakiety IPv6 przesyłane jako dane pakietów IPv4.
- ❖ Pomiedzy komputerem a brokerem tworzony jest logiczny kanał (1 hop z punktu widzenia IPv6).



Lektura dodatkowa

- ❖ Kurose & Ross: rozdział 4.
- ❖ Tanenbaum: rozdział 5.
- ❖ Stevens: rozdział 8.

- ❖ Dokumentacja IPv6 i UDP:
 - ◆ <https://web.archive.org/web/20211130063606/http://networksorcery.com/enp/Protocol/ipv6.htm>
 - ◆ <https://web.archive.org/web/20211206051320/http://networksorcery.com/enp/Protocol/udp.htm>

- ❖ Raporty CIDR: <https://www.cidr-report.org/as2.0/>
- ◆ Beej's Guide to Network Programming: <http://beej.us/guide/bgnet/>

Zagadnienia

- ❖ Co to są prywatne adresy IP? Jakie pule adresów są zarezerwowane na takie adresy?
- ❖ Co robi funkcja `bind()`?
- ❖ Czym różnią się porty o numerach mniejszych niż 1024 od innych?
- ❖ Jakie są zadania procesora routingu, portu wejściowego, portu wyjściowego i struktury przełączającej?
- ❖ Czym się różni przełączanie pakietów w routerze za pomocą RAM od przełączania za pomocą struktury przełączającej?
- ❖ Jakie są pożądane cechy struktury przełączającej w routerze?
- ❖ Gdzie w routerze stosuje się buforowanie? Po co?
- ❖ Po co w portach wyjściowych klasyfikuje się pakiety?
- ❖ Co to jest blokowanie początku kolejki? Gdzie występuje? Jak się go rozwiązuje?
- ❖ Rozwiń skrót LPM.
- ❖ Jakie znasz struktury danych implementujące LPM? Porównaj je.
- ❖ Co to jest pamięć TCAM? Jak można ją zastosować do implementacji LPM?
- ❖ Na czym polega fragmentacja IP? Gdzie się ją stosuje i dlaczego? Gdzie łączy się fragmenty?
- ❖ Co to jest MTU? Na czym polega technika wykrywania wartości MTU dla ścieżki?
- ❖ Jak działa szeregowanie pakietów w buforze wyjściowym routera?
- ❖ Na czym polega NAT i po co się go stosuje? Jakie są jego zalety i wady?
- ❖ Jaki stan musi przechowywać router z funkcją NAT?
- ❖ Jakie są różnice pomiędzy nagłówkami IPv4 i IPv6?
- ❖ Zapisz adres IPv6 `0321:0000:0000:0123:0000:0000:0000:0001` w najkrótszej możliwej postaci.
- ❖ Co to jest tunelowanie 6in4?