Transport

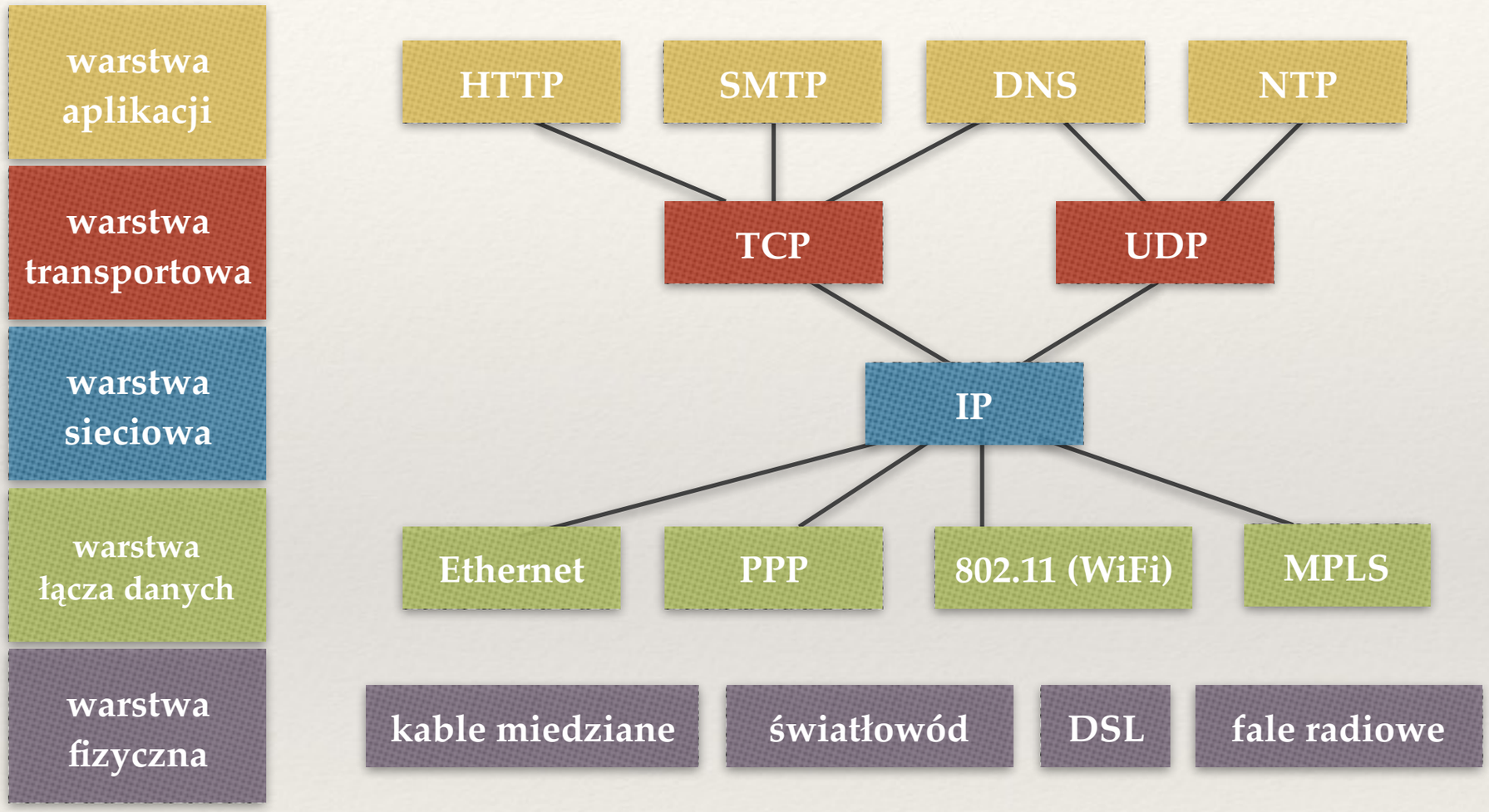
część 1: niezawodny transport

Sieci komputerowe

Wykład 6

Marcin Bieńkowski

Protokoły w Internecie



Internetowy model warstwowy (1)

warstwa transportowa

zapewnia **globalne** dostarczanie danych pomiędzy **aplikacjami**

warstwa sieciowa

zapewnia **globalne** dostarczanie danych pomiędzy **komputerami**

warstwa łącza danych

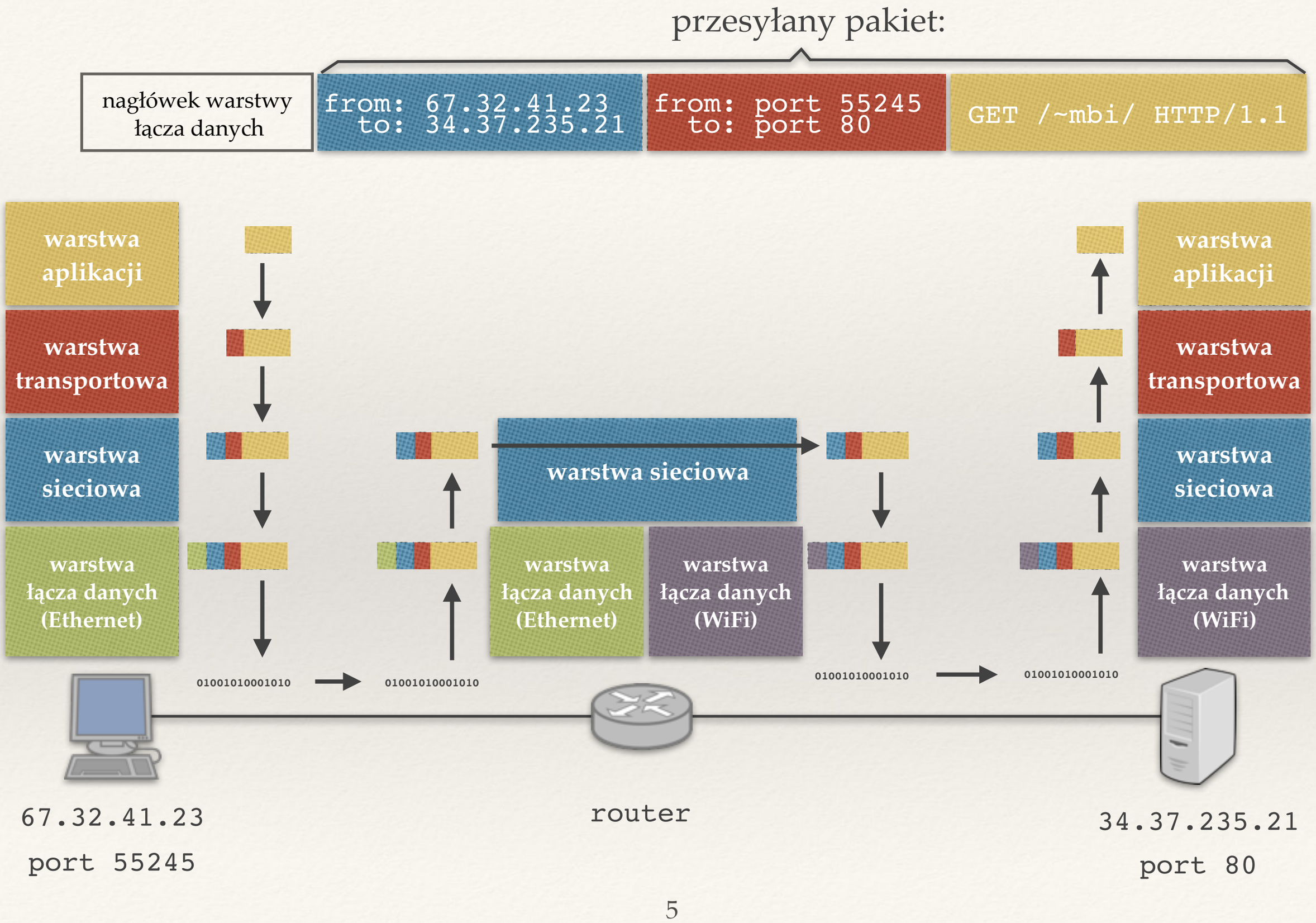
zapewnia **lokalne** dostarczanie danych pomiędzy **komputerami**

Porty

- **Port:**
  - liczba 16-bitowa;
  - identyfikuje aplikację wewnątrz danego komputera
    → umożliwiają multipleksowanie wielu strumieni danych

- **W nagłówku warstwy transportowej znajduje się m.in.:**
  - port źródłowy;
  - port docelowy.

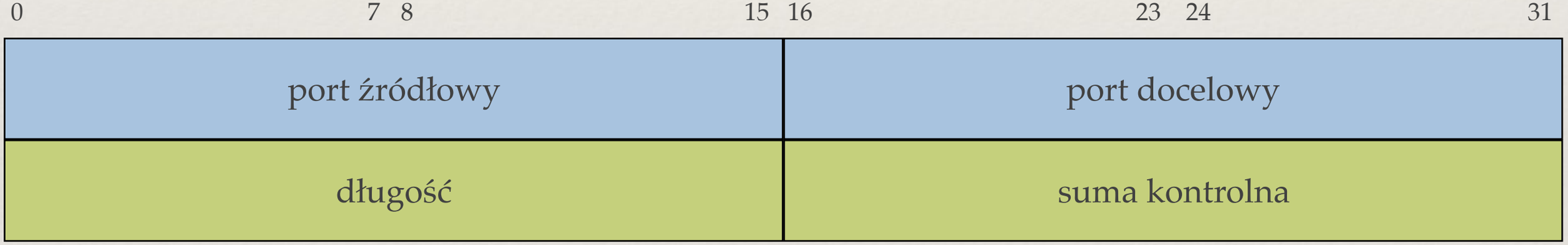
Internetowy model warstwowy (2)



Zawodny transport

UDP

- Najprostszy protokół warstwy transportowej
- Nagłówek UDP:



Gwarancje UDP (1)

- **Takie same jak IP, czyli żadne.**
  - Tylko zasada dołożenia wszelkich starań (*best effort*)

- **Pakiety mogą zostać:**
  - uszkodzone,
  - zgubione,
  - opóźnione,
  - zamienione (kolejność),
  - zduplikowane (przez wyższe lub niższe warstwy).

Gwarancje UDP (2)

- Czy to wszystko co może stać się z datagramami UDP?
- Lokalnie wysyłane pakiety nie będą przecież uszkadzane, gubione, duplikowane i będą przychodzić w tej samej kolejności?

demonstracja

kod klienta wysyłającego n datagramów

- **Kontrola przepływu** = nadawca powinien dostosowywać prędkość transmisji do szybkości odbiorcy.

Gdzie wykorzystujemy

Zawodny transport:

- Przesyłane są małe ilości danych (np. DNS, DHCP).
- Proste, ograniczone obliczeniowo urządzenia
  (np. TFTP wykorzystywany do aktualizacji firmware).
- Konieczna jest szybka reakcja (gry).
- Utrata pojedynczych datagramów nieistotna (transmisje multimedialne).
- Chcemy pełnej kontroli nad przesyłanymi danymi (NFS).

Niezawodny transport:

- Przesyłane są duże ilości danych (np. HTTP, FTP, sieci P2P).
- Istotne potwierdzanie danych: praca zdalna (np. SSH).

Niezawodny transport: segmentacja

Segmentacja

Niezawodne przesyłanie ciągu bajtów:

- Zadanie warstwy transportowej.
- Wymaga dzielenia ciągu bajtów na **segmenty**.
  - W UDP użytkownik musi to robić sam.
- **Dlaczego nie przesyłamy wszystkiego w jednym segmencie?**

Słowo o nazewnictwie

warstwa transportowa	**datagramy** (w przypadku gdy użytkownik sam dzieli na części, np. UDP) **segmenty** (w przypadku gdy warstwa dzieli na części, np. TCP)
warstwa sieciowa	**pakiety**
warstwa łącza danych	**ramki**

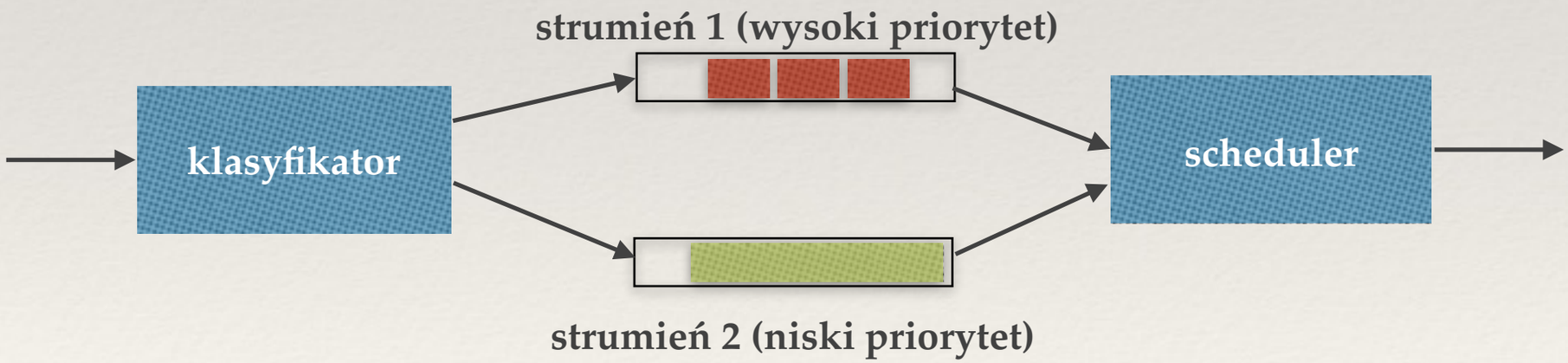
Niekonsekwentnie używane nazwy.

- Powszechnie stosowane „datagramy IP“.
- Czasem „segment TCP“ to same dane TCP (bez nagłówka TCP), np. w definicji MSS (maximum segment size).

Dlaczego segmentacja jest potrzebna?

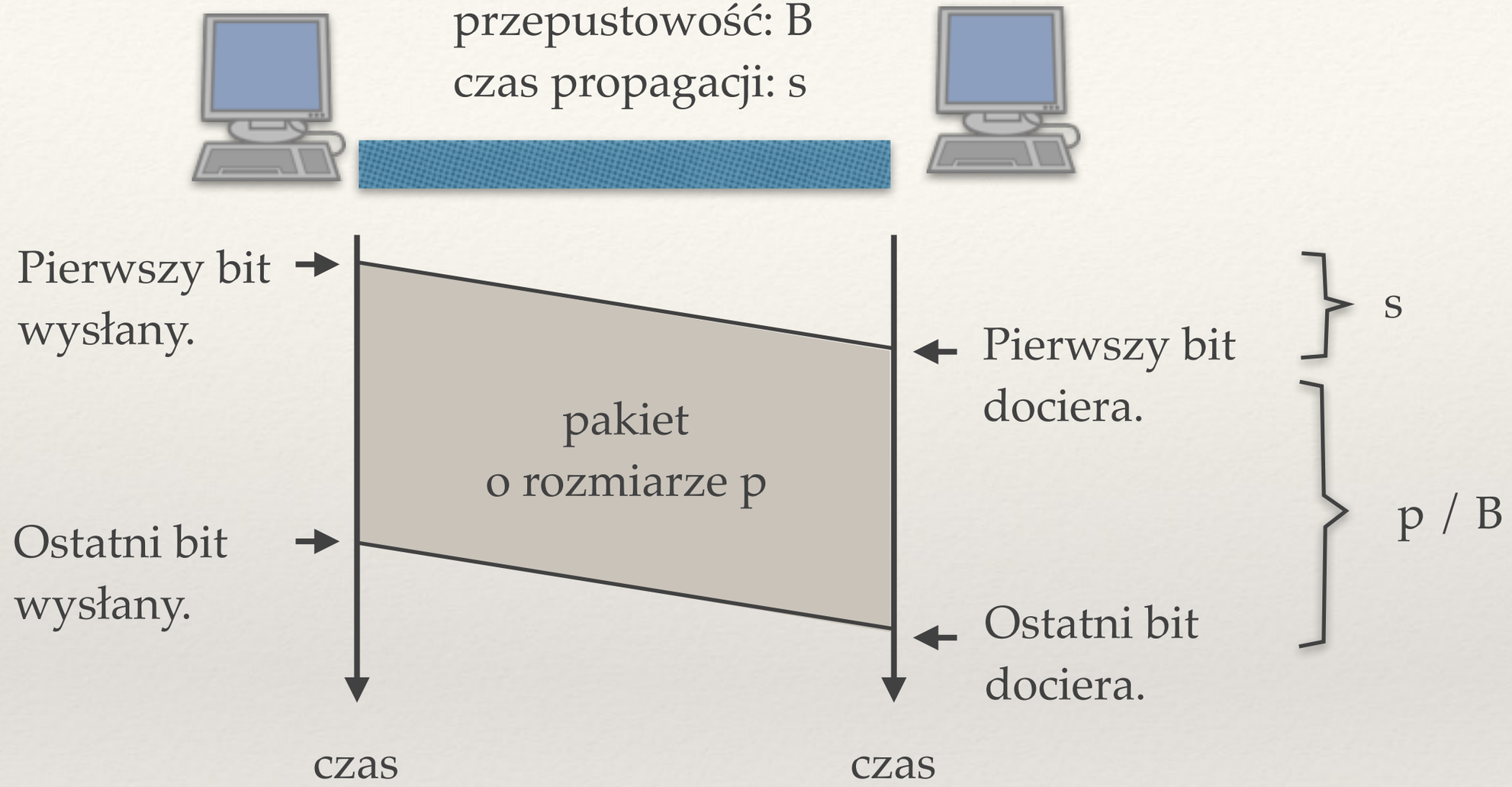
- **Skąd biorą się ograniczenia na rozmiar segmentu?**
  - MSS = MTU - IP header - TCP header

- **Dlaczego nie zwiększymy MTU (i rozmiaru pakietu IP)?**
  - Większa szansa kolizji (sieci bezprzewodowe).
  - Duże pakiety: problem w szeregowaniu w kolejce wyjściowej routera (małe pakiety mają duże opóźnienie).



  - Główna przyczyna: **mniejsze opóźnienie przy długich ścieżkach**.

Opóźnienie pakietu na łączu (przypomnienie)

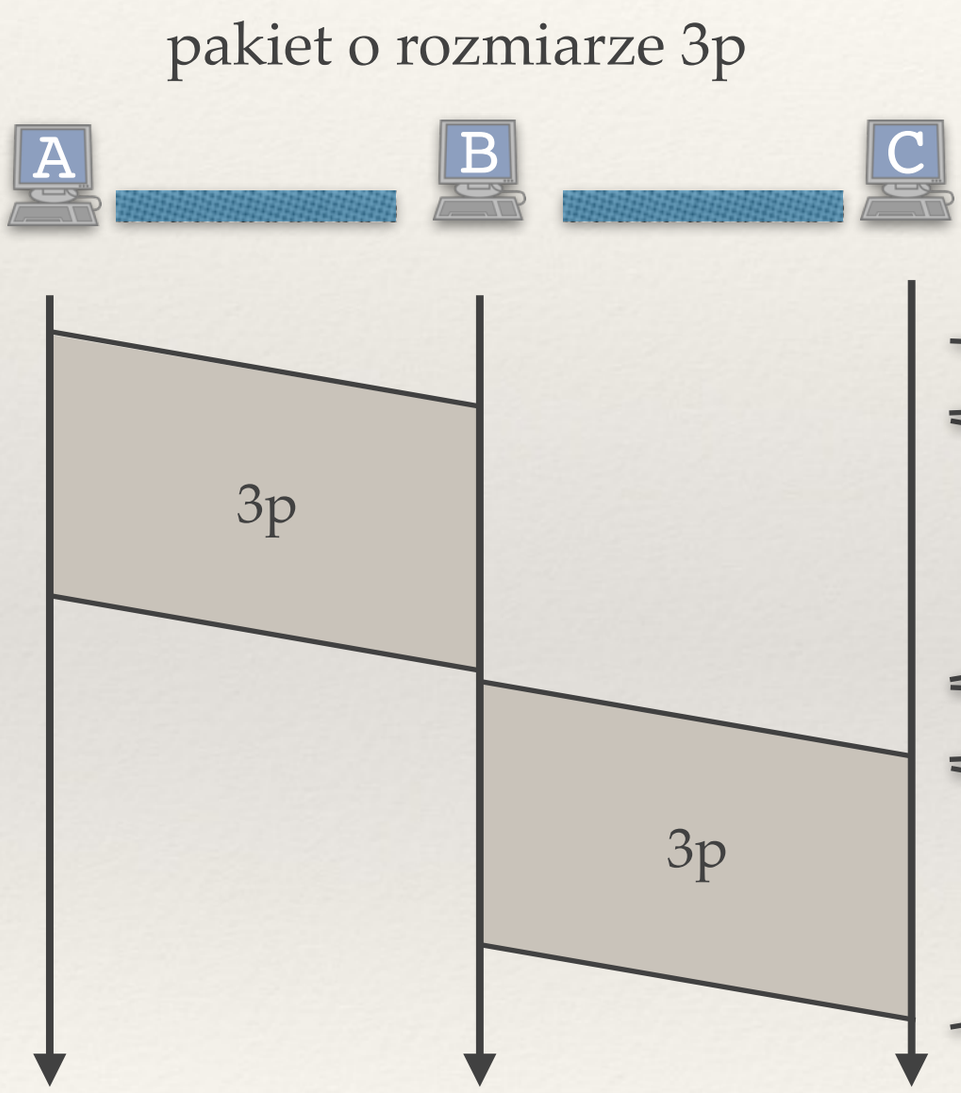


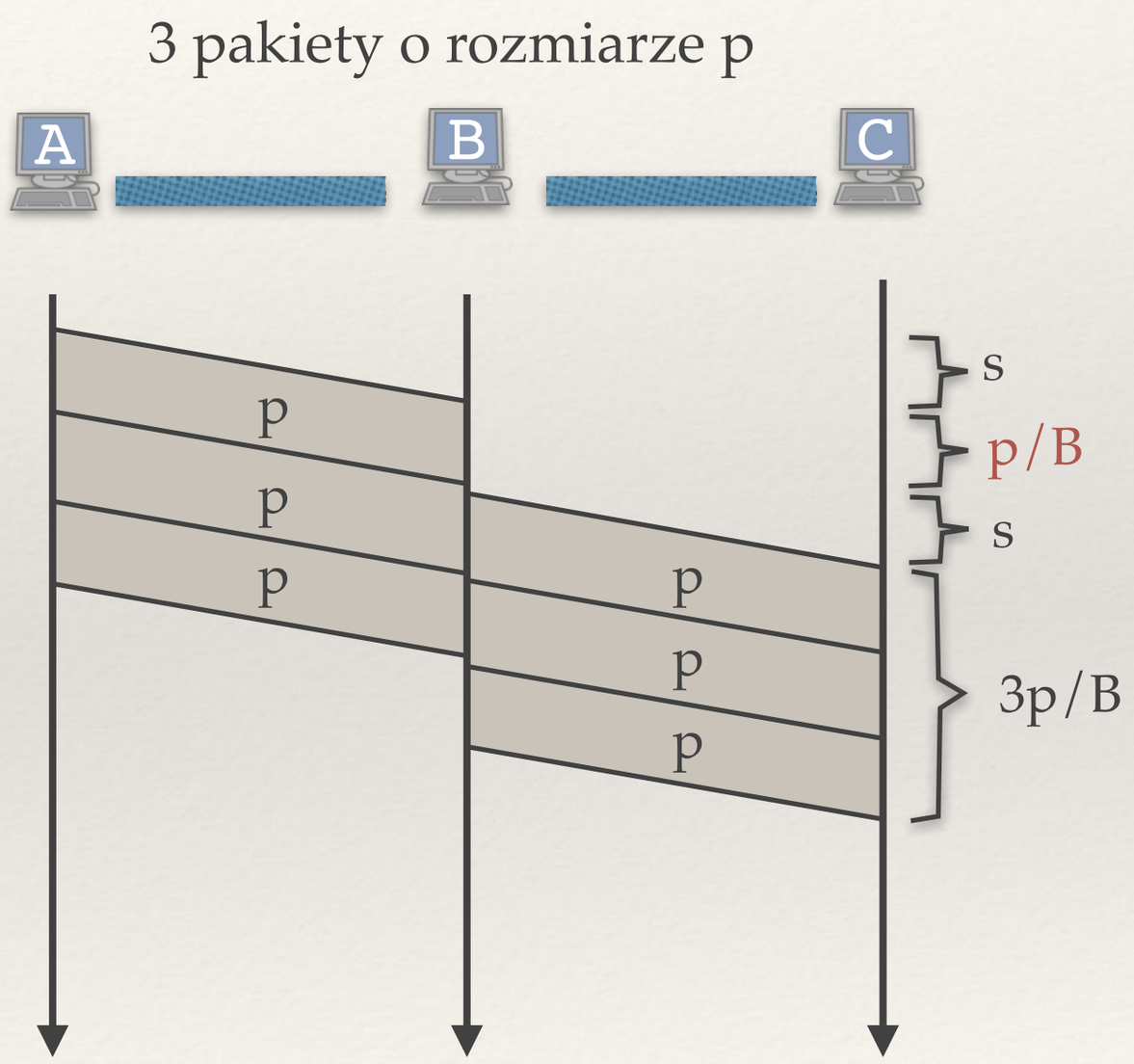
Opóźnienie na pojedynczym łączu = s + p / B.

- Ignorujemy czas kolejkowania pakietu w buforze.

Opóźnienie pakietu na ścieżce

- Dla dowolnego łącza: przepustowość = B, czas propagacji = s.
- Im dłuższa ścieżka tym większy efekt.





Niezawodny transport: ARQ

ARQ = Automatic Repeat reQuest

- **Zajmiemy się niezawodną transmisją jednokierunkową.**
  - Od nadawcy do odbiorcy.
  - W drugą stronę identyczny mechanizm.

- **ARQ**
  - Wysyłanie „do skutku“ (do otrzymania potwierdzenia od odbiorcy).
  - Odbiorca wysyła informacje zwrotne (otrzymałem pakiet / zwolnij / przyspiesz / …)
  - Niezawodny transport na bazie zawodnej usługi przesyłania pakietów.

- **Założymy, że strumień bajtów jest już posegmentowany.**

Dostępne podstawowe mechanizmy

- **Sumy kontrolne:** możemy wykrywać, czy pakiet został uszkodzony.

- **Potwierdzenia (ACK):** małe pakiety kontrolne potwierdzające otrzymanie danego segmentu.

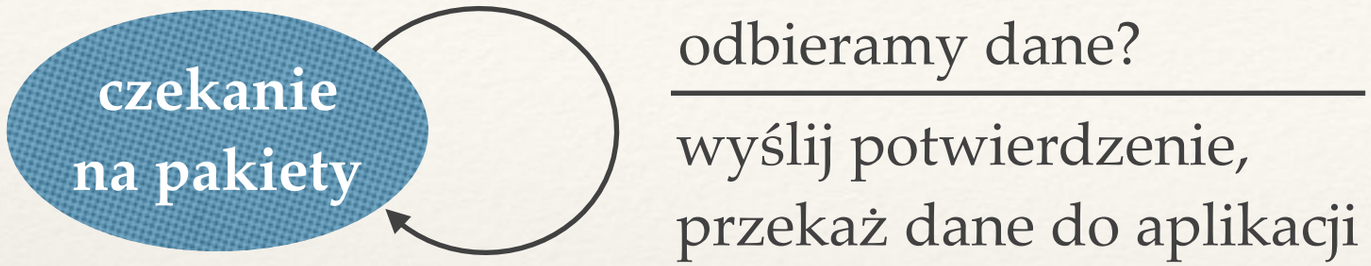
- **Timeout (przekroczenie czasu oczekiwania):** jeśli nie otrzymamy potwierdzenia przez pewien czas (typowy dla łącza, np. RTT, jak go mierzyć?)

- **Retransmisje:** ponowne wysłanie danego segmentu w przypadku przekroczenia czasu oczekiwania.

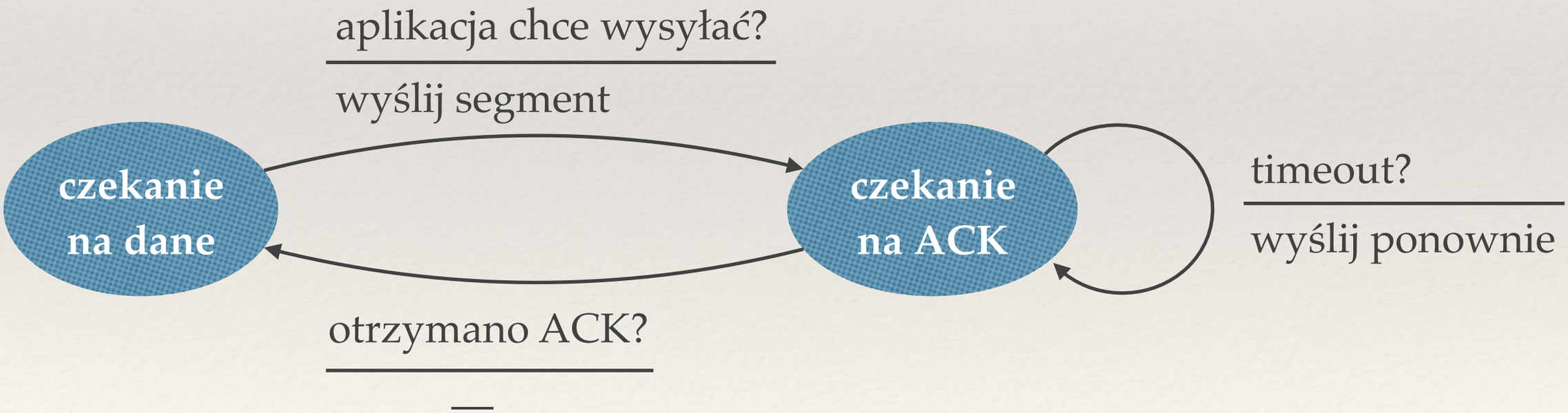
ARQ: Stop-and-Wait

Protokół Stop-and-Wait

Algorytm odbiorcy:

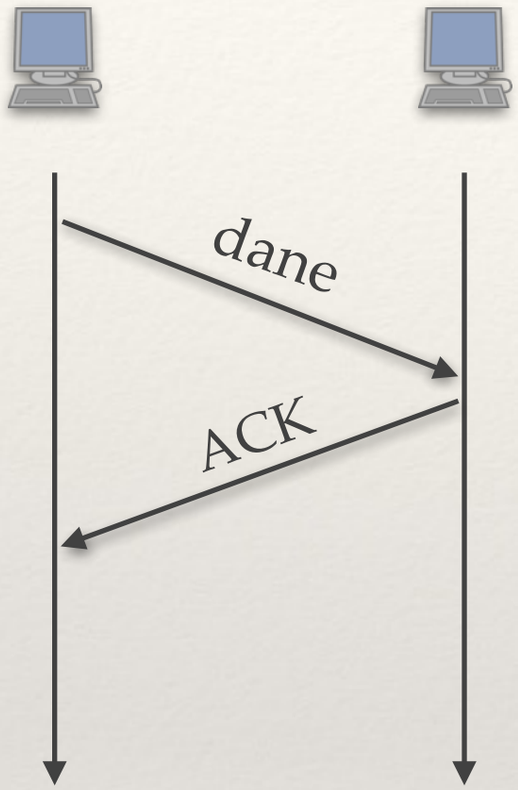


Algorytm nadawcy:

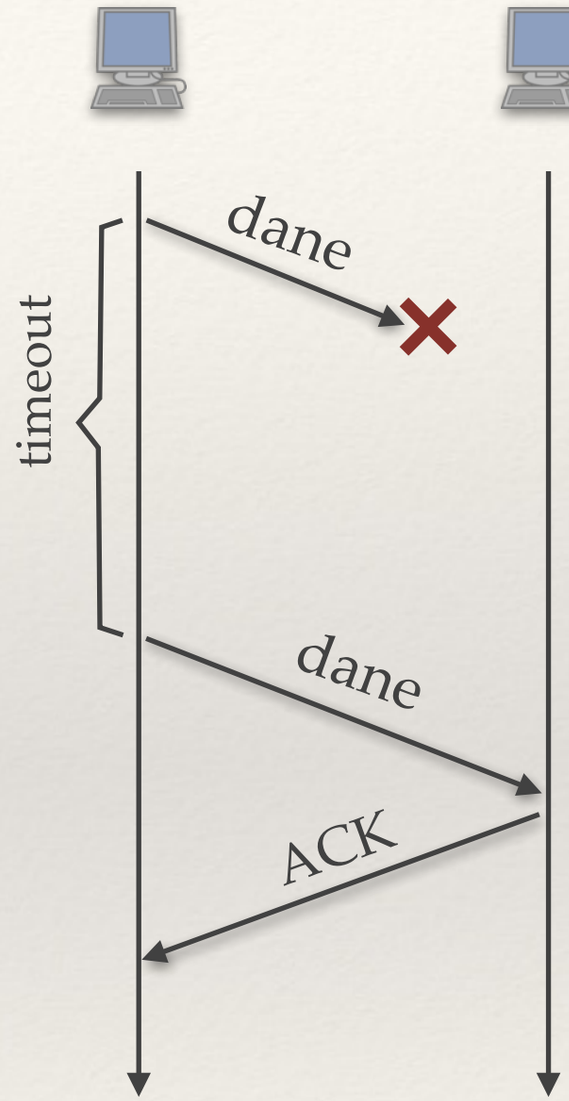


Stop-and-wait: typowe wykonania (1)

1. bez utraty pakietów

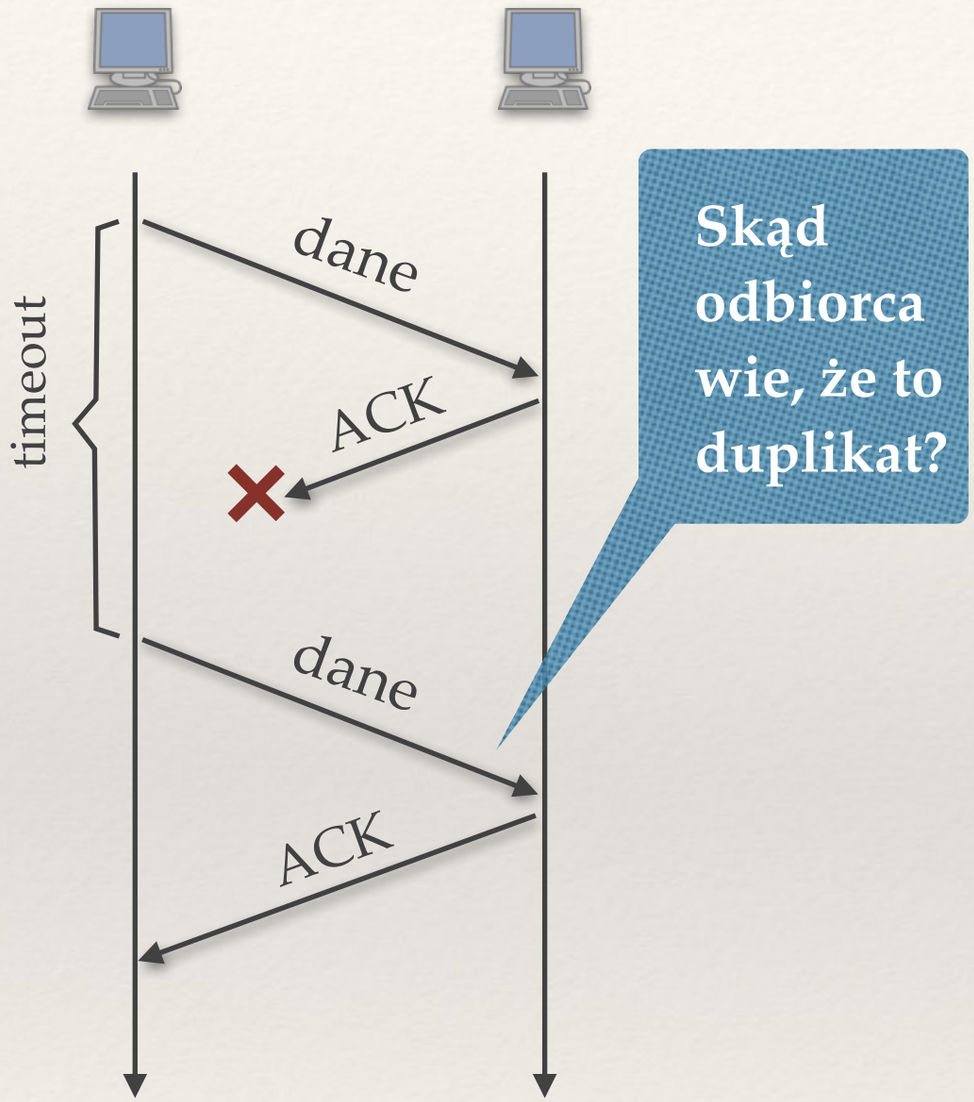


2. utrata danych

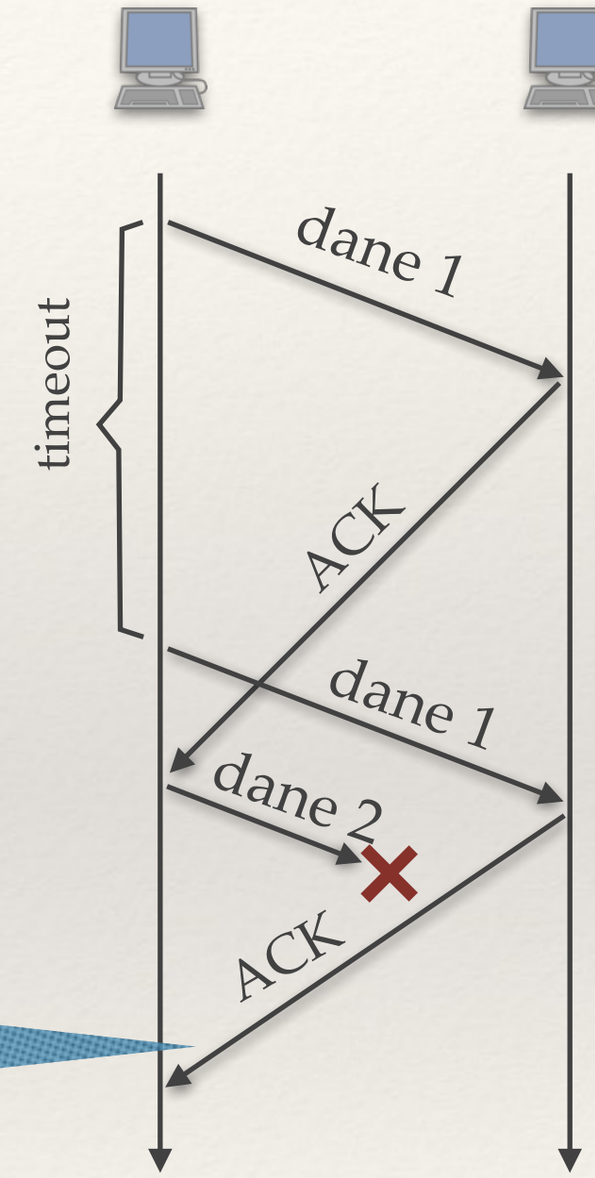


Stop-and-wait: typowe wykonania (2)

3. utrata ACK

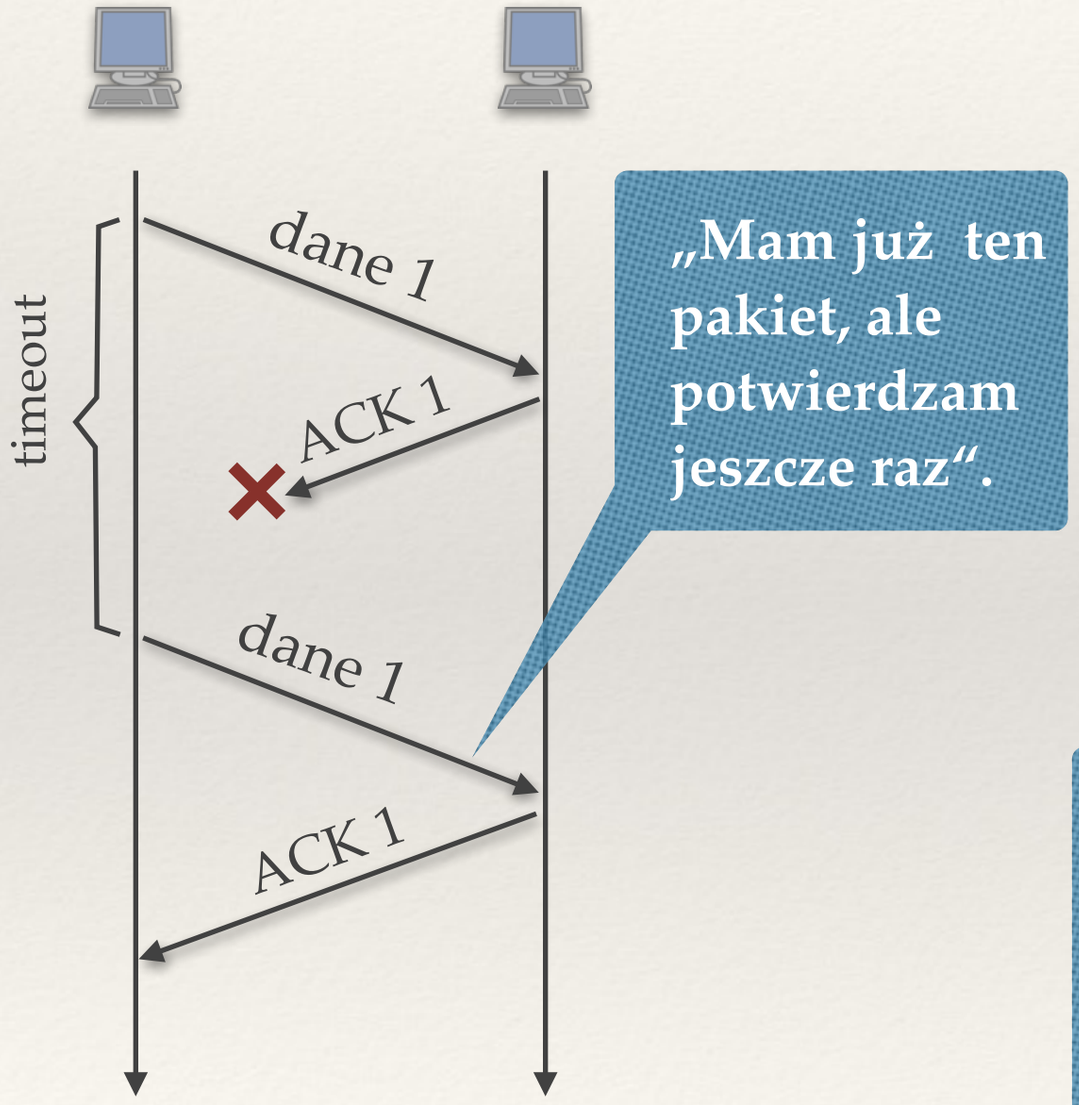


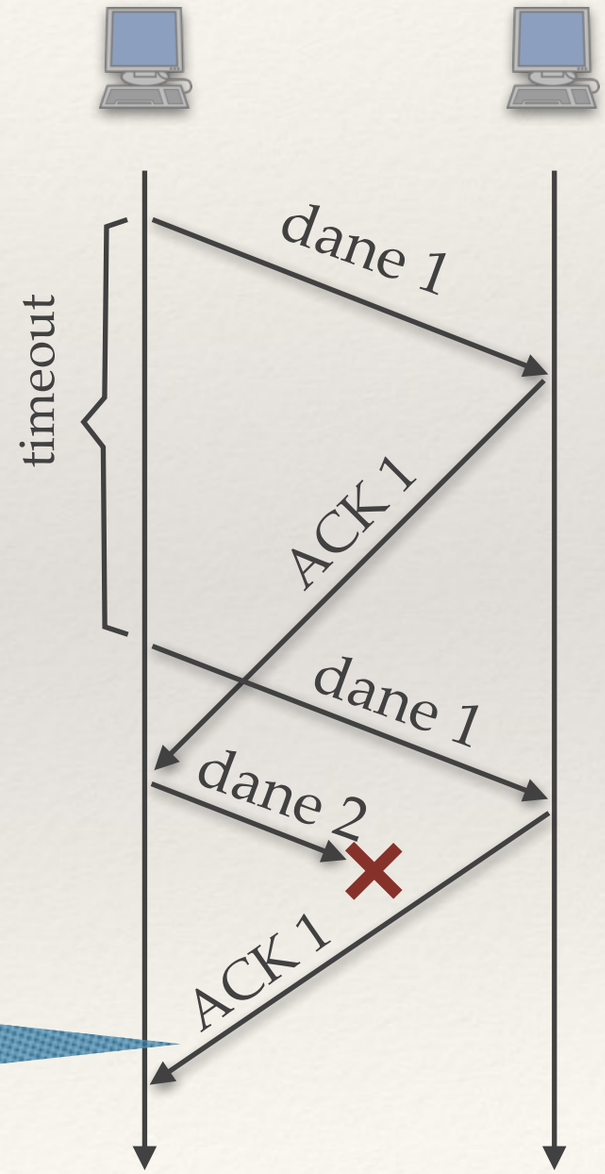
4. opóźnienie ACK + utrata danych



Numery sekwencyjne

- Każdy segment jest numerowany.
- ACK zawiera numer potwierdzanego segmentu.

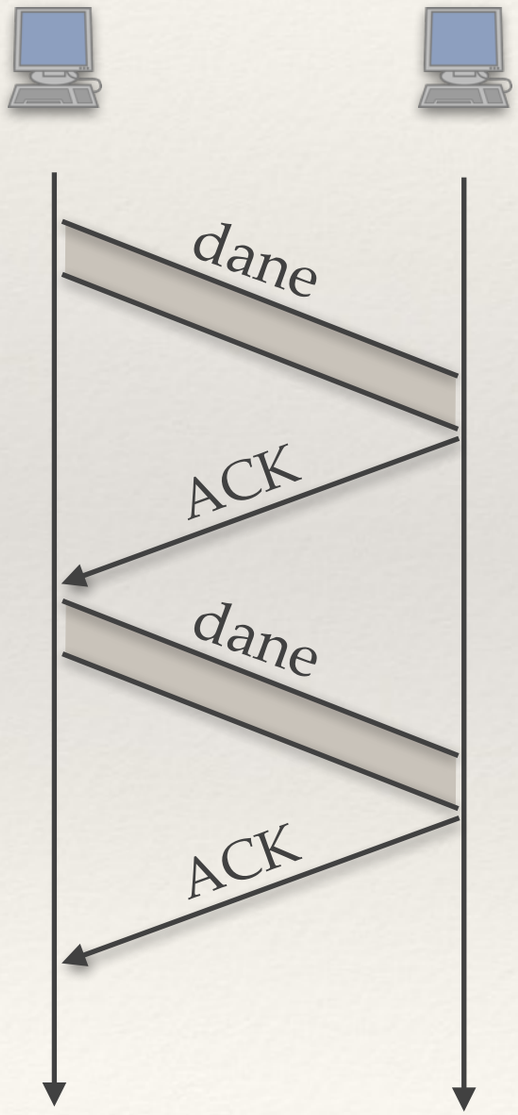




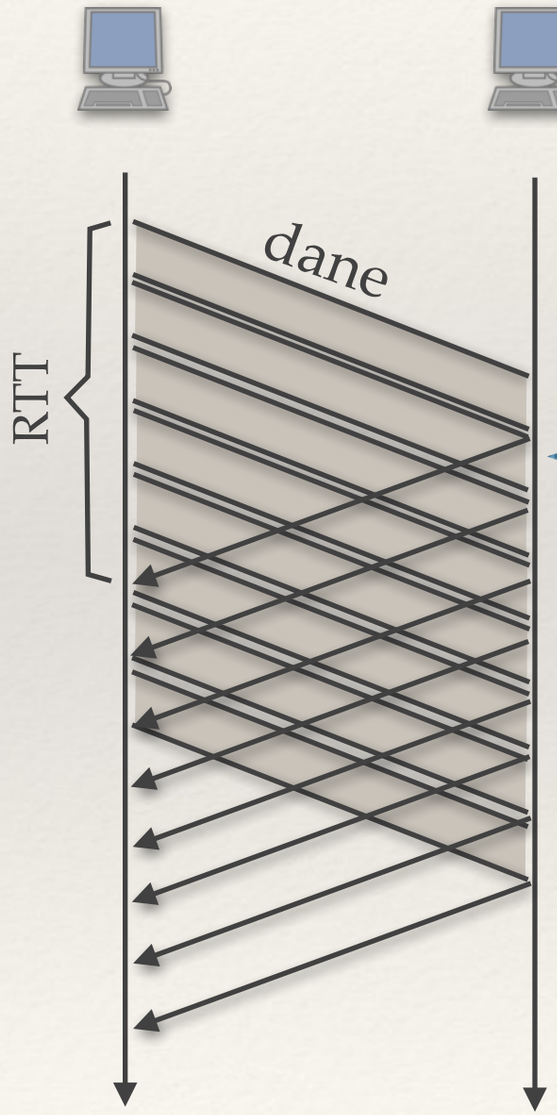
Stop-and-wait + numery sekwencyjne

- Działa, ale przy długich łączach o dużej przepustowości wykorzystuje ułamek ich możliwości!

stop-and-wait



lepiej: (jak?)



Nadawca powinien móc nadawać bez czekania na ACK:

- **co najmniej przez RTT;**
- **równoważnie: nadać co najmniej 2 × BDP danych.**

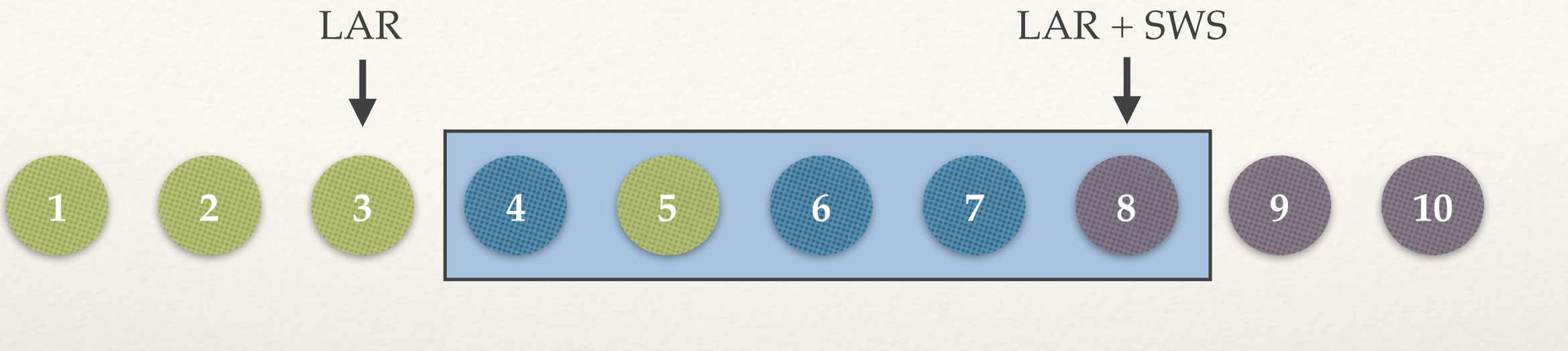
ARQ: Okno przesuwne

Przesuwne okno (1)

- SWS (*sender window size*) = maksymalna liczba wysłanych i niepotwierdzonych segmentów.
- Segmenty od 1 do LAR są potwierdzone, a LAR+1 nie.
- Nadawca może wysyłać tylko segmenty leżące w oknie.

Przesyłany ciąg segmentów:

Przesuwne okno (2)



Akcje:

- Otrzymanie ACK → sprawdzamy, czy możemy przesunąć okno.
- Przesunięcie okna → wysyłamy dodatkowe segmenty.
- Timeout dla (niepotwierdzonego) segmentu → wysyłamy go ponownie.

Mechanizmy potwierdzania

Nadawca: mechanizm przesuwnego okna.

Trzy mechanizmy dla odbiorcy:

- Go-Back-N
- Potwierdzanie selektywne
- Potwierdzanie skumulowane

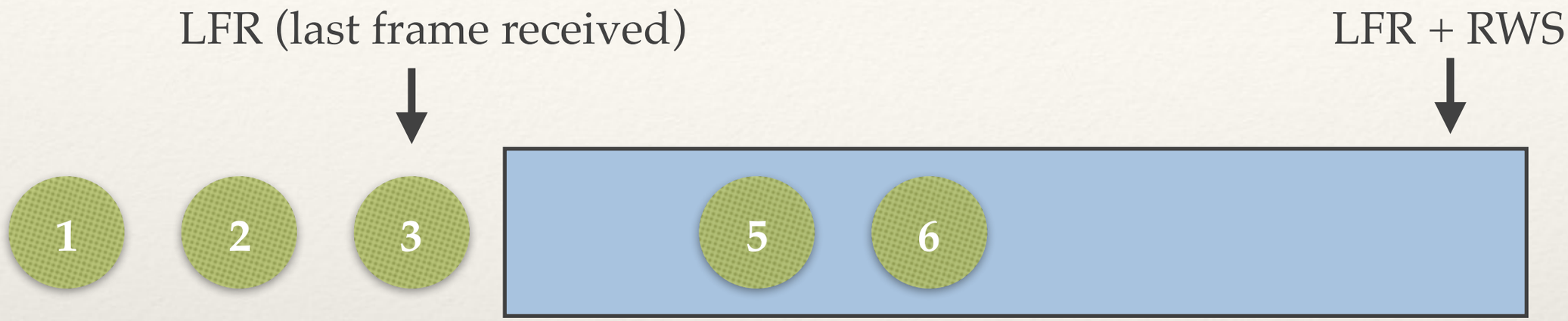
Potwierdzanie Go-Back-N

- Załóżmy, że odbiorca dostał już segmenty do P włącznie.
- Wyśle ACK dla otrzymanego segmentu S jeśli $S \leq P + 1$.
  - Dlaczego potwierdzanie już potwierdzonych segmentów jest ważne?
- Odbiorca może przekazywać dane od razu warstwie wyższej (brak bufora odbiorcy).

Potwierdzanie selektywne (1)

Odbierany ciąg segmentów:

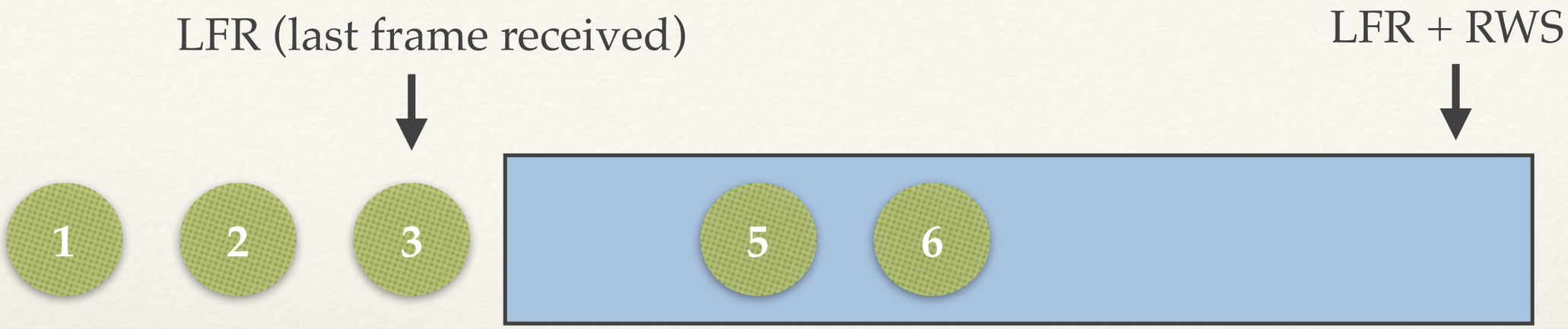
odebrane (i potwierdzone)



Okno odbiorcy:

- Rozmiar RWS (*receiver window size)*.
- Segmenty od 1 do LFR (*last frame received*) są otrzymane, a segment LFR+1 nie.

Potwierdzanie selektywne (2)



- **Otrzymujemy segment S**
  - LFR < S ≤ LFR + RWS → zapisz segment w buforze odbiorczym.
  - S ≤ LFR + RWS → odeślij ACK.
  - S > LFR + RWS → ignoruj segment.
  - S = LFR + 1 → aktualizuj LFR (przesuń okno).

- **Dla RWS = 1, potwierdzanie selektywne = Go-Back-N.**

Potwierdzanie skumulowane

- **Poza ACK wszystko jak przy potwierdzaniu selektywnym.**

- **Wysyłanie ACK:**
  - Również tylko jeśli otrzymamy segment S ≤ LFR + RWS
  - Ale niekoniecznie wysyłamy ACK dla S!
  - W razie potrzeby aktualizujemy LFR (przesuwamy okno w prawo) a następnie wysyłamy ACK dla LFR.

demonstracja

Kontrola przepływu

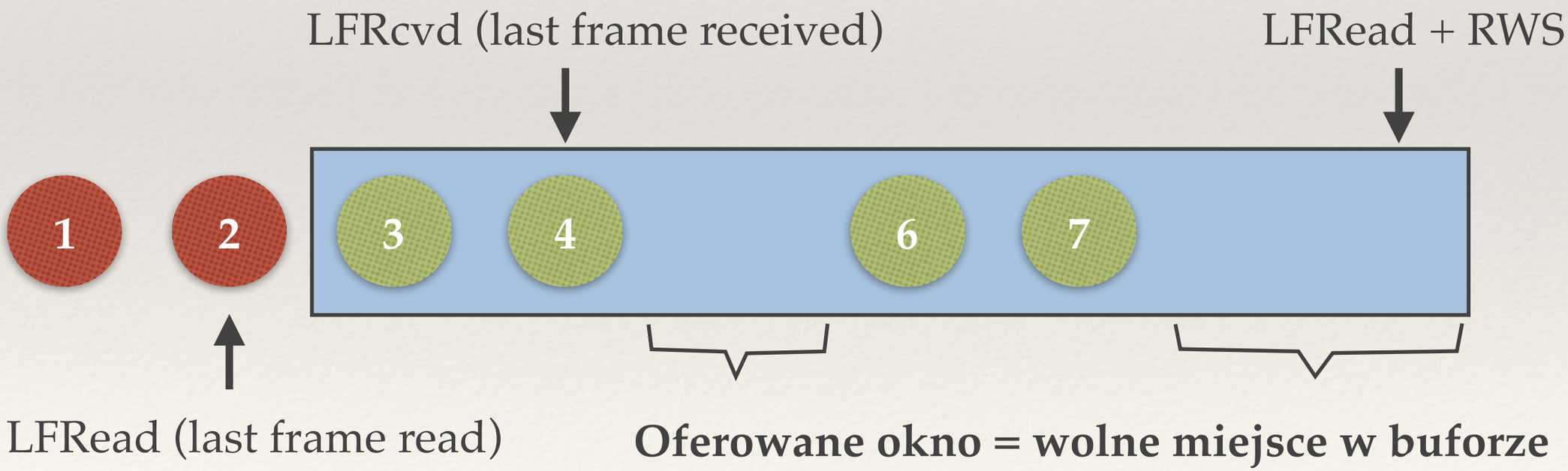
Kontrola przepływu (1)

Co to znaczy „przekazujemy dane do warstwy aplikacji“?

- To warstwa aplikacji pobiera dane wywołaniem `read()`.

odebrane, potwierdzone, przeczytane przez aplikację

odebrane (i potwierdzone) ale nieprzeczytane przez aplikację



Kontrola przepływu (2)

Oferowane okno = wolne miejsce w buforze

- Wysyłane nadawcy (razem z ACK).
- Nadawca zmienia SWS (rozmiar swojego okna) na tę wartość:

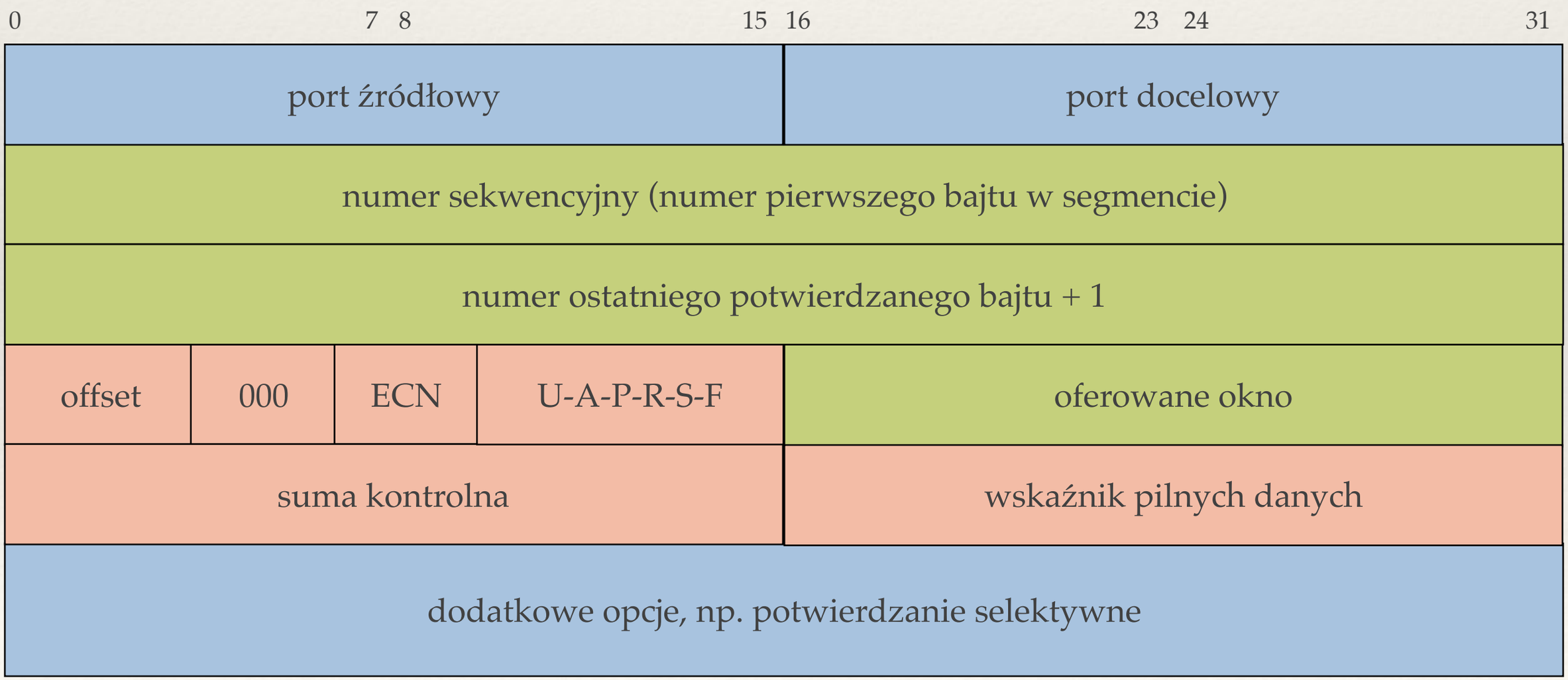
- Nadawca nie będzie bezsensownie wysyłał danych, na które odbiorca nie ma miejsca.

Niezawodny transport: TCP

TCP

- **Numeruje bajty, a nie segmenty.**
- **Potwierdzanie skumulowane:**
  - Uwaga: ACK n = oznacza „mam wszystko do bajtu n-1 włącznie“.
  - ACK wysyłany w pakiecie razem z danymi w drugą stronę.

demonstracja



Numerowanie bajtów: przykładowe komplikacje

- Jeśli oferowaliśmy okno = 0 i aplikacja zwolniła miejsce?
  → Wyślij osobno rozmiar okna (bez ACK).

- Jeśli nie mamy danych do wysłania w drugą stronę?
  → Opóźnione wysyłanie ACK.

- Jeśli oferowane okno jest mniejsze niż MSS?
  → Czekamy.

- Jeśli aplikacja generuje dane mniejsze niż MSS
  → Wyślij dopiero jeśli kiedy wszystkie poprzednie dane zostaną potwierdzone (algorytm Nagle'a).
  → Co z interaktywnymi programami (praca zdalna)?

Ustawianie timeoutu dla segmentu

- **Obliczamy średnie RTT ważone wykładniczo czasem**
  - avg-RTT = α × avg-RTT + (1 - α) × zmierzone-RTT
  - RTT segmentów stałe → avg-RTT zbiega do tej wartości.
  - W podobny sposób mierzymy wariancję RTT (var-RTT).

- **RTO (*retransmission timeout*) = 2 × avg-RTT + 4 × var-RTT**

Gdzie implementować?

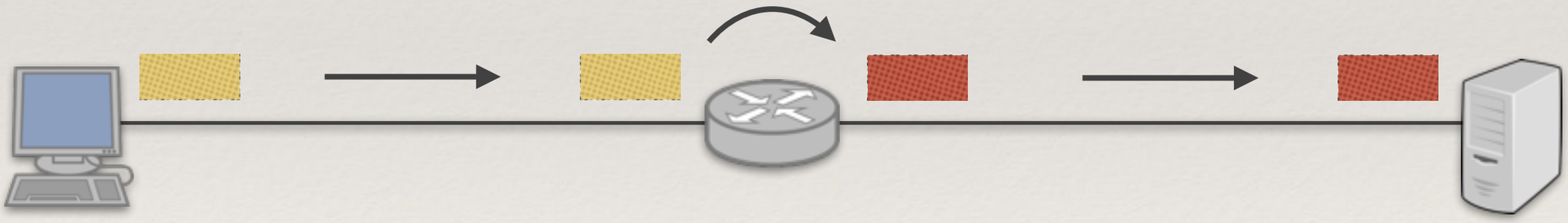
Gdzie implementować warstwę transportową?

Warstwa sieciowa:

- Implementowana na routerach i urządzeniach końcowych.

Warstwa transportowa:

- Wczesna filozofia: niezawodność dostarczania zapewniać na każdym łączu.
- Problem: błąd może nie być związany z łączem!



- Wniosek: niezawodność dostarczania i tak musi być kontrolowana na urządzeniach końcowych.

Zasada end-to-end (1)

Słaba wersja

- Niezawodne przesyłanie danych musi być implementowane na urządzeniach końcowych, ale *warstwy niższe mogą w tym pomagać.*

Silna wersja

- Niezawodne przesyłanie danych musi być implementowane na urządzeniach końcowych, *warstwy niższe nie powinny się tym w ogóle zajmować.*

Zasada end-to-end (2)

Która wersja zasady? Spór filozoficzny na przykładzie TCP + WiFi.

- TCP działa dobrze tylko jeśli łącza są w miarę niezawodne.
- Łącza bezprzewodowe tracą średnio 20-80% pakietów
- Potwierdzanie i retransmisja na poziomie warstwy łącza danych.

- Łamiemy model warstwowy i silną wersję zasady.
  - Krótkoterminowe duże korzyści.
  - Ale być może trudności we wprowadzeniu innej wersji warstwy transportowej.

Lektura dodatkowa

- Kurose & Ross: rozdział 3.
- Tanenbaum: rozdział 6.
- Dokumentacja online:
  - `http://www.networksorcery.com/enp/protocol/tcp.htm`