
Transport

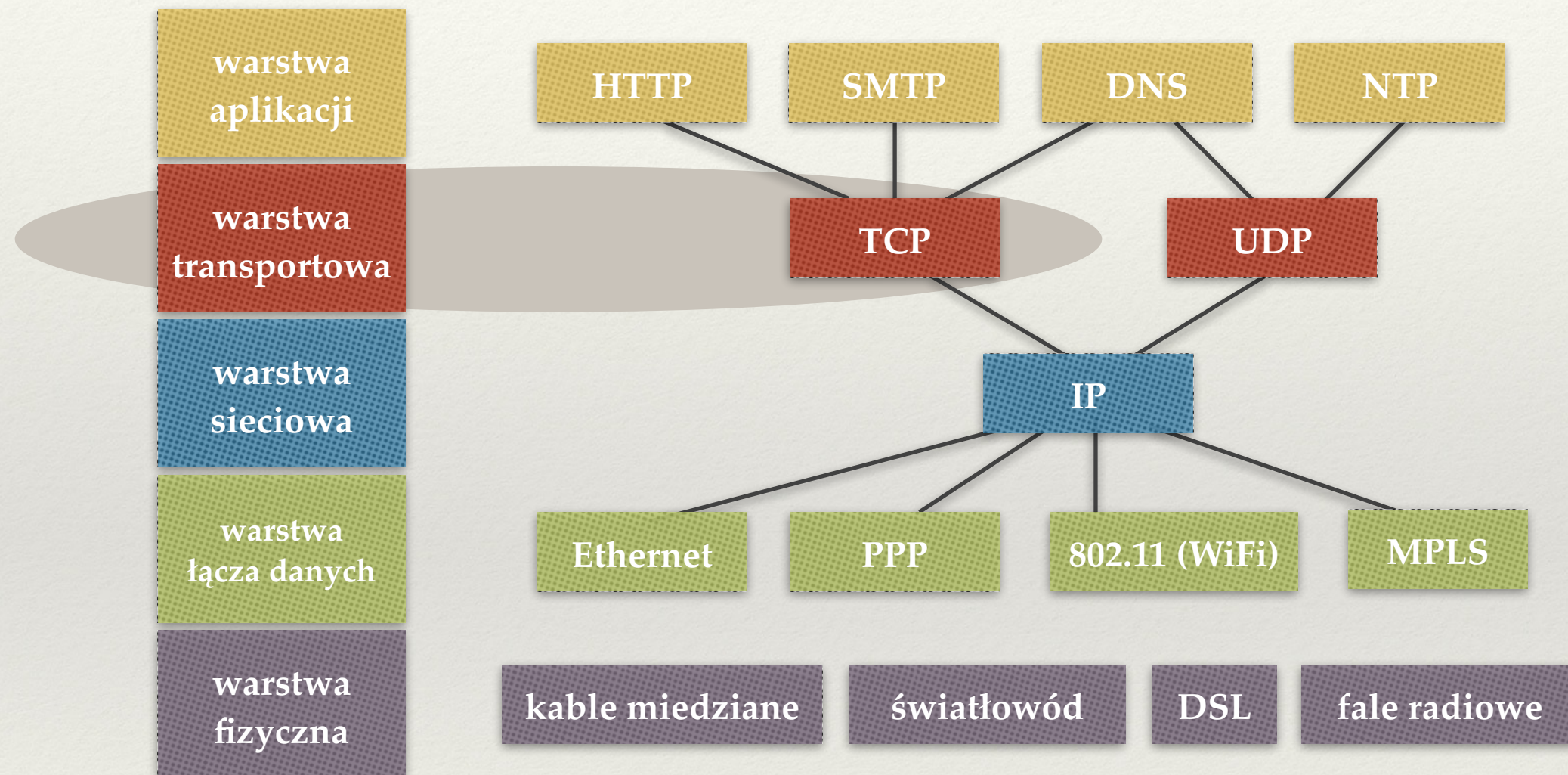
część 1: podstawy

Sieci komputerowe

Wykład 6

Marcin Bieńkowski

Protokoły w Internecie



Internetowy model warstwowy (1)



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

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

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

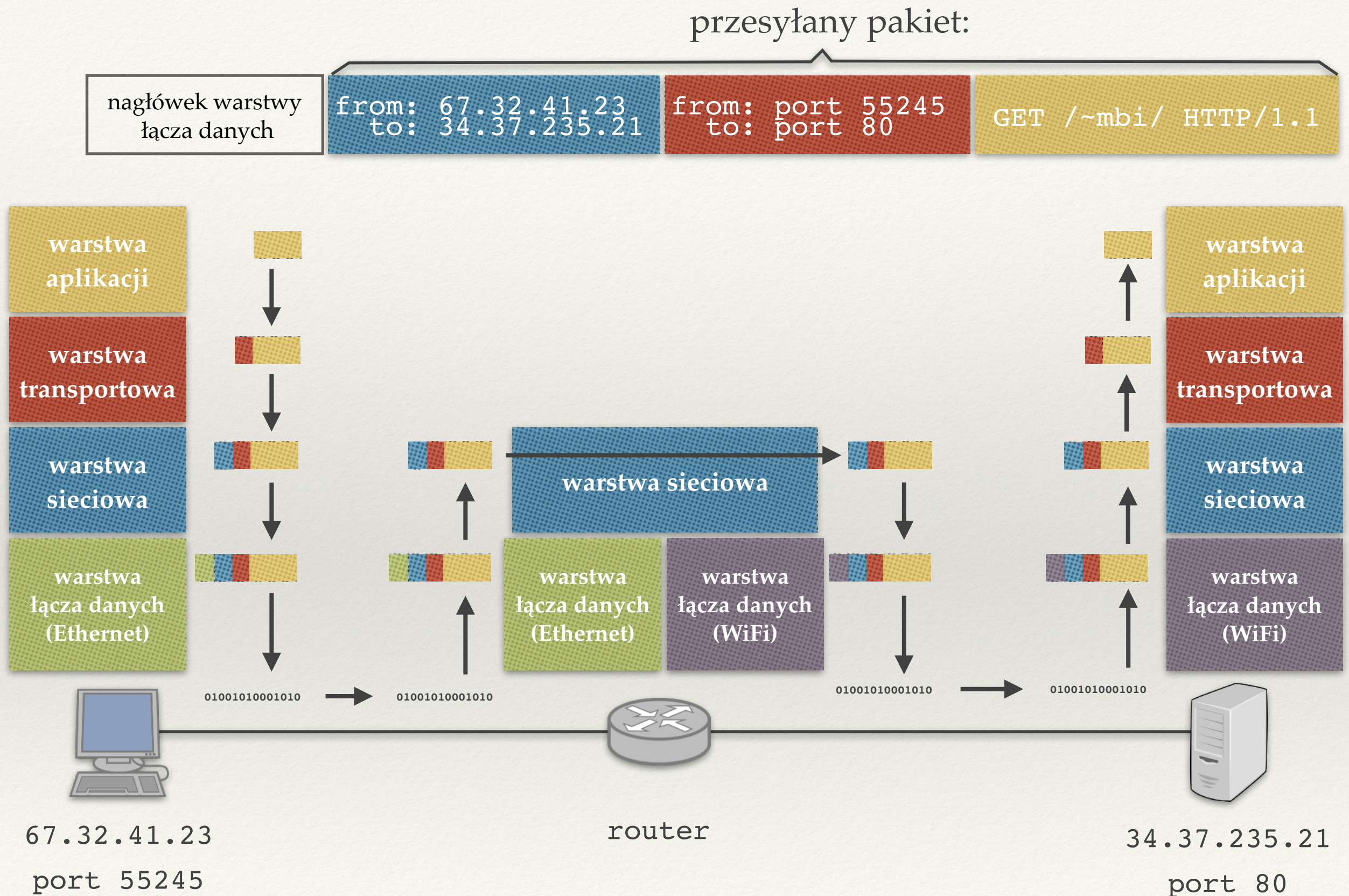
❖ Port:

- ♦ liczba 16-bitowa;
- ♦ identyfikuje aplikację wewnątrz danego komputera
→ multipleksowanie wielu strumieni danych w jednym ciągu pakietów.

❖ W nagłówku warstwy transportowej znajduje się m.in.:

- ♦ port źródłowy;
- ♦ port docelowy.

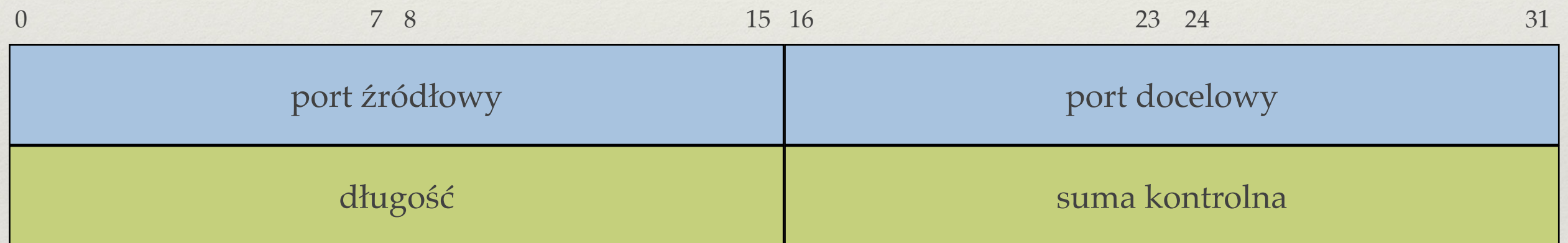
Internetowy model warstwowy (2)



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 te negatywne wydarzenia są częste?
- ❖ Lokalnie wysyłane pakiety nie będą przecież uszkodzane, gubione, duplikowane i będą przychodzić w tej samej kolejności?

demonstracja

[kod klienta wysyłającego n datagramów](#)

Gwarancje UDP (2)

- ❖ Czy te negatywne wydarzenia są częste?
- ❖ Lokalnie wysyłane pakiety nie będą przecież uszkodzane, 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 (gdy użytkownik sam dzieli na części, np. UDP)
segmenty (gdy warstwa dzieli na części, np. TCP)

warstwa
sieciowa

pakiety

warstwa
łącza danych

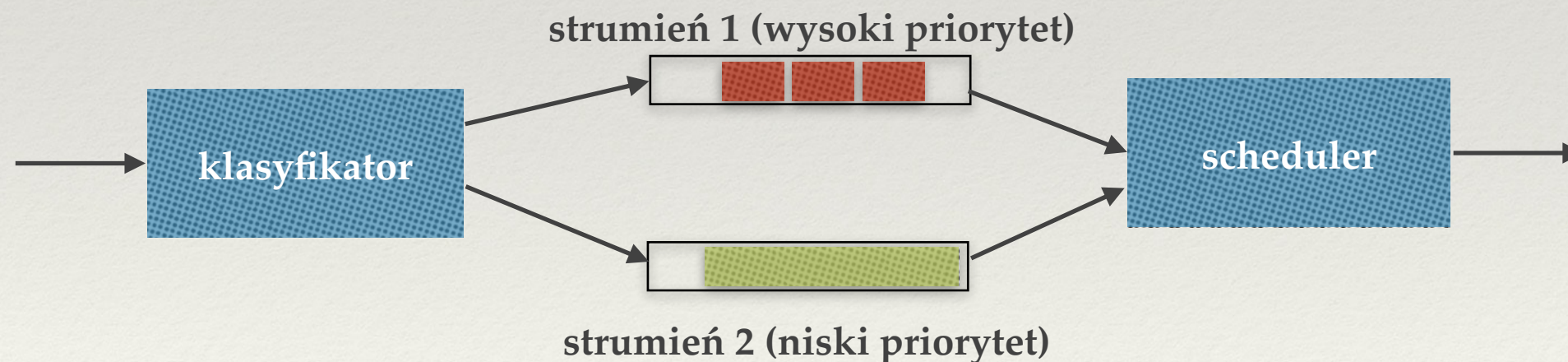
ramki

Niekonsekwentnie używane nazwy.

- ❖ Powszechnie stosowane „datagramy IP”.
- ❖ „Segment TCP” często oznacza same dane TCP (bez nagłówka TCP), np. w definicji MSS (maximum segment size).

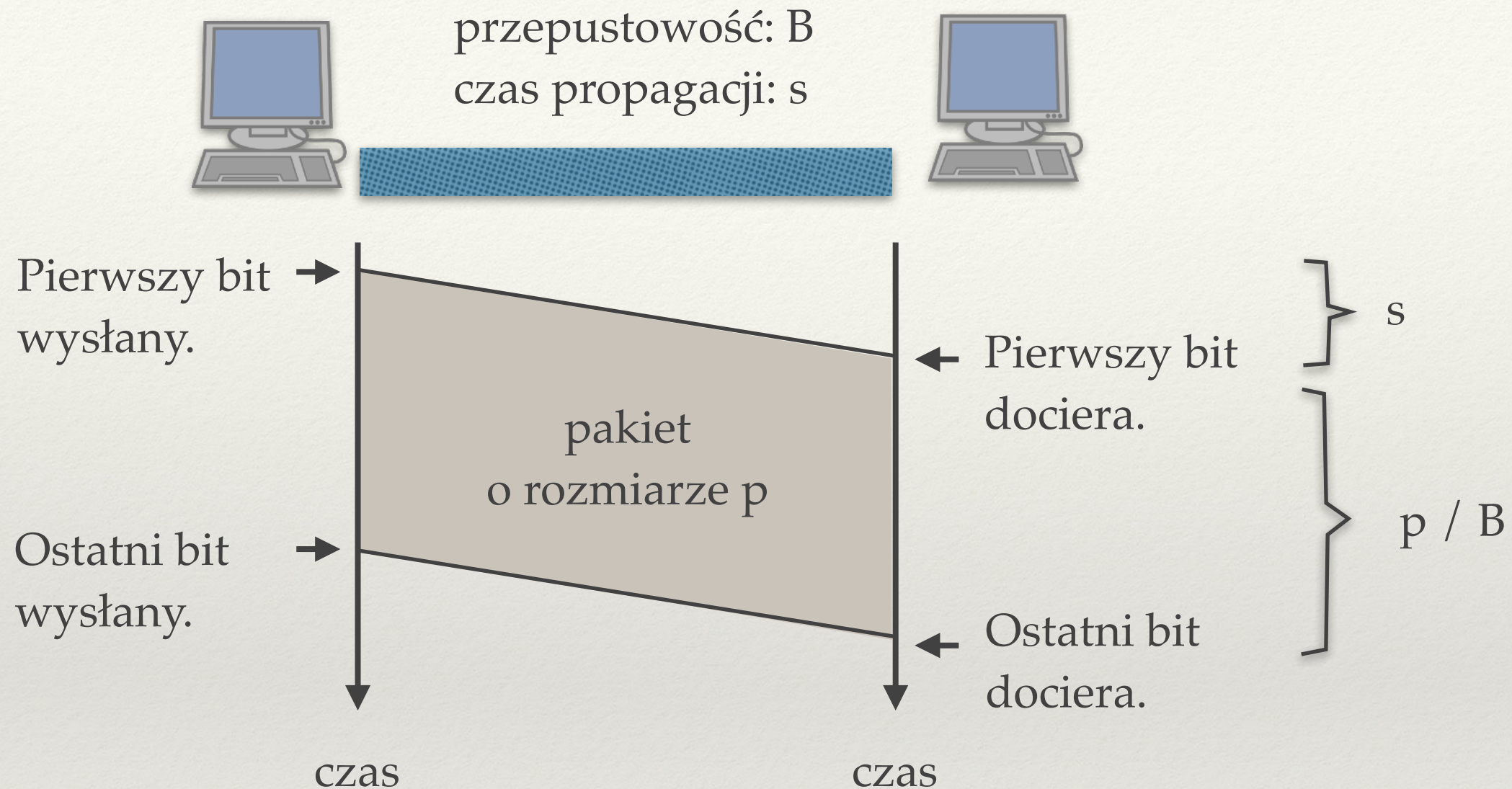
Dlaczego segmentacja jest potrzebna?

- ❖ Skąd biorą się ograniczenia na rozmiar segmentu?
 - ♦ $MSS = MTU - \text{rozmiar nagłówka IP} - \text{rozmiar nagłówka TCP}$
- ❖ Dlaczego nie zwiększymy MTU (i rozmiaru pakietu IP)?
 - ♦ Większa szansa na zakłócenia (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)



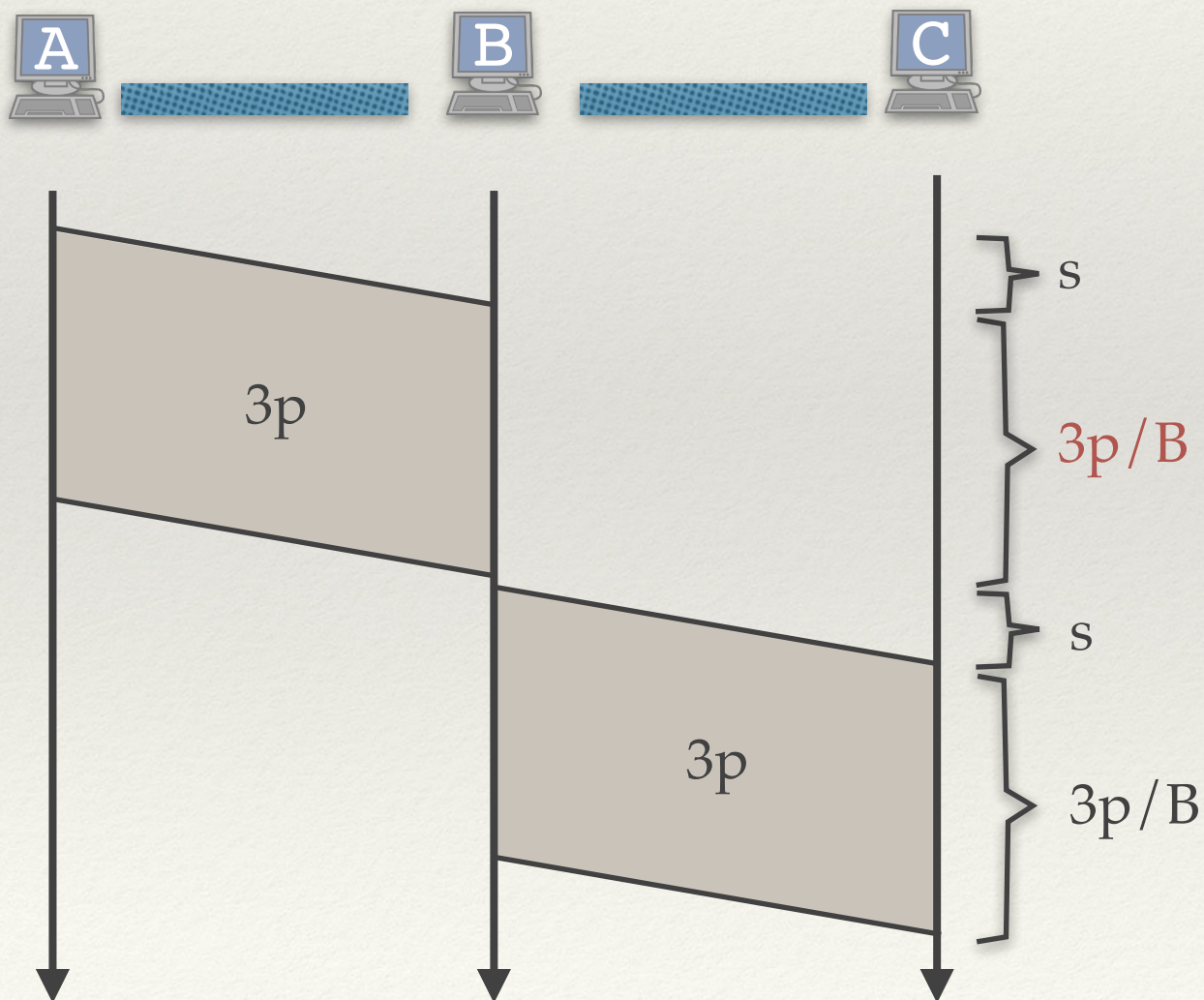
Opóźnienie na pojedynczym łączu = $s + p / B$.

- ❖ Ignorujemy czas kolejowania pakietu w buforze.

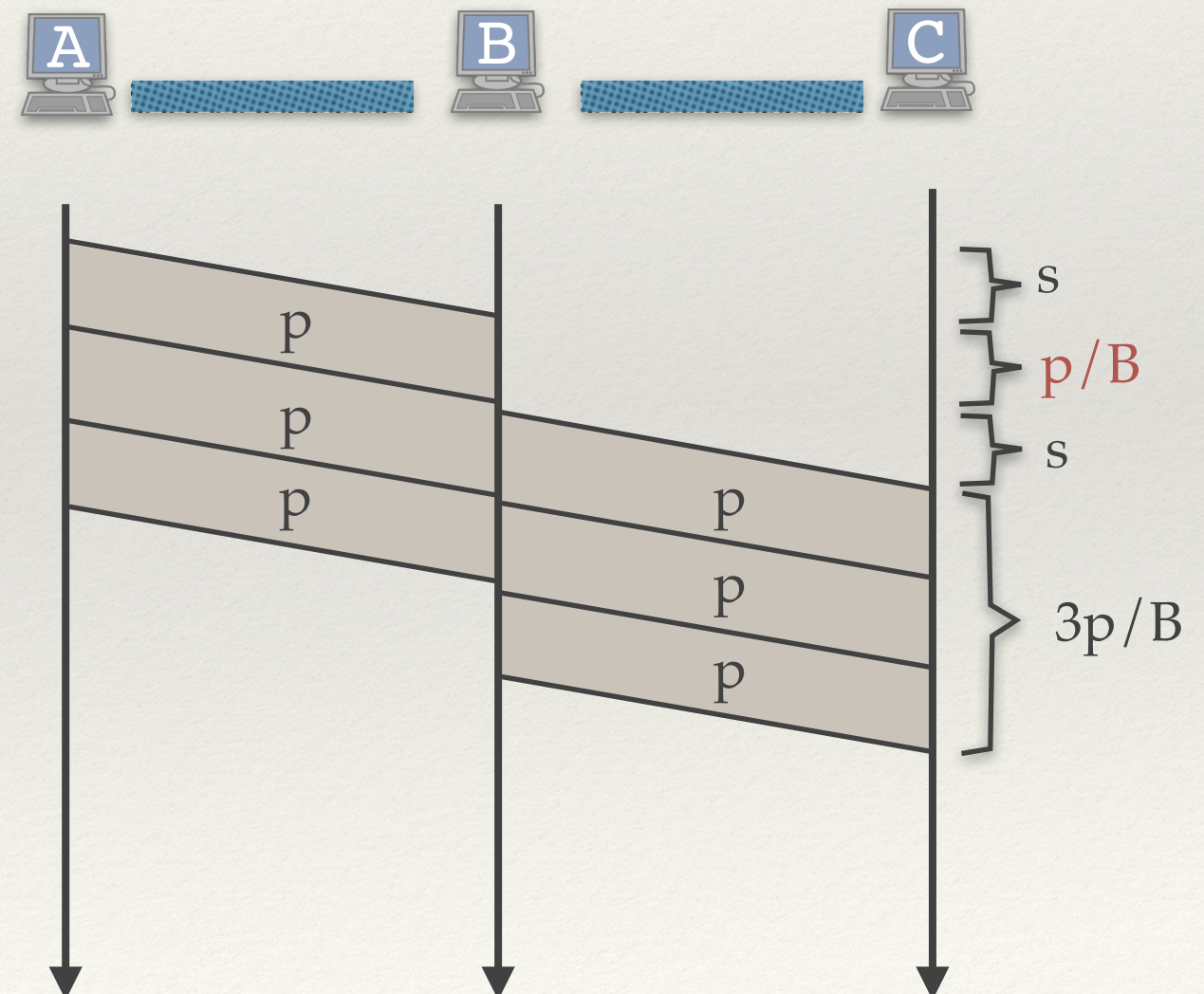
Opóźnienie pakietu na ścieżce

- ❖ Dla łącz poniżej: przepustowość = B , czas propagacji = s .
- ❖ Im dłuższa ścieżka, tym większy efekt.
- ❖ Pomijamy tu narzut związany z rozmiarem nagłówka!

pakiet o rozmiarze $3p$



3 pakiety o rozmiarze p



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.

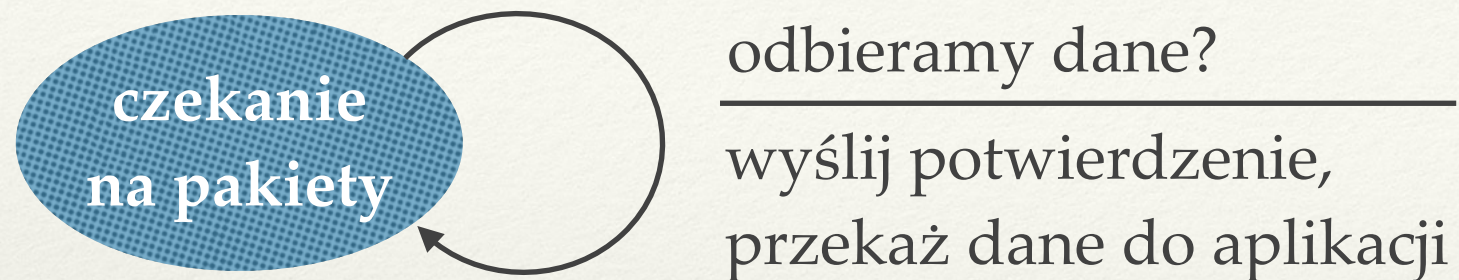
Dostępne podstawowe mechanizmy

- ❖ **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.

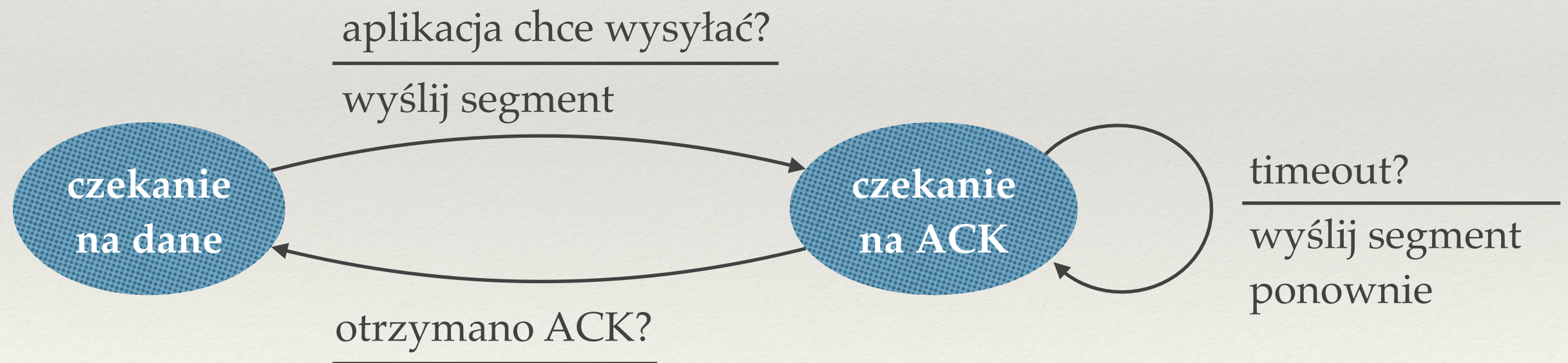
ARQ: Stop-and-Wait

Protokół Stop-and-Wait

Algorytm odbiorcy:

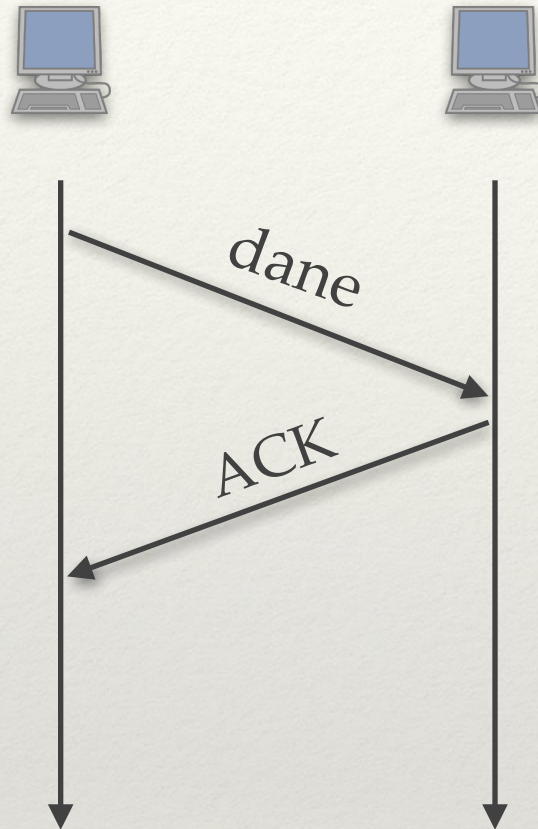


Algorytm nadawcy:



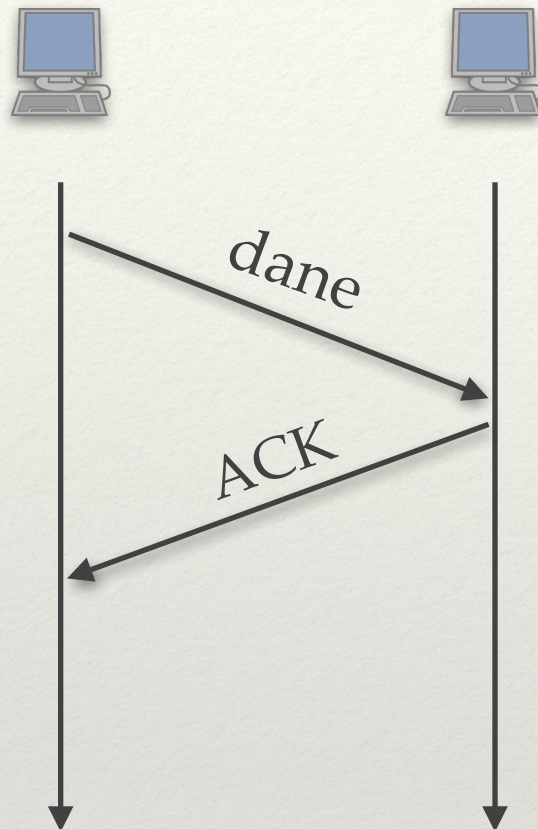
Stop-and-wait: typowe wykonania

1. bez utraty pakietów

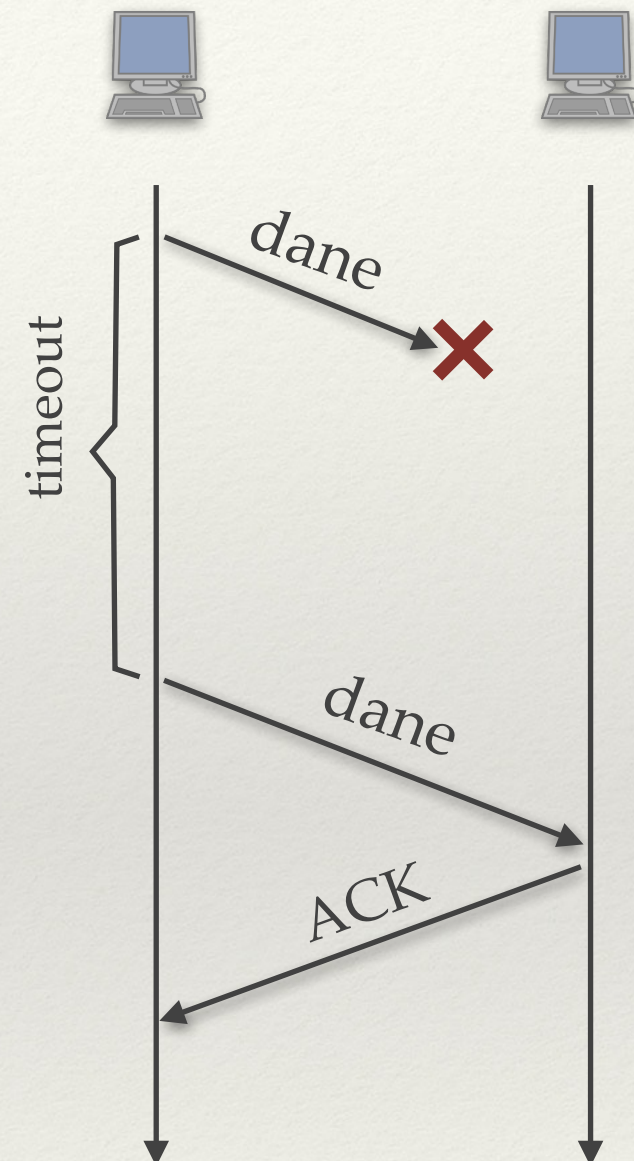


Stop-and-wait: typowe wykonania

1. bez utraty pakietów

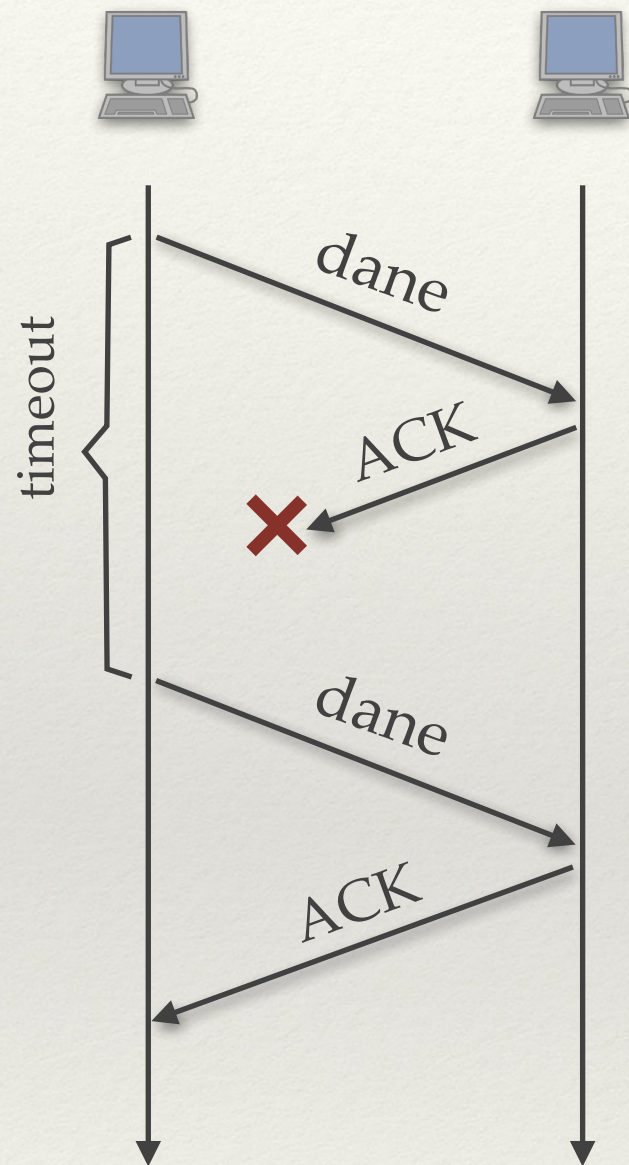


2. utrata danych



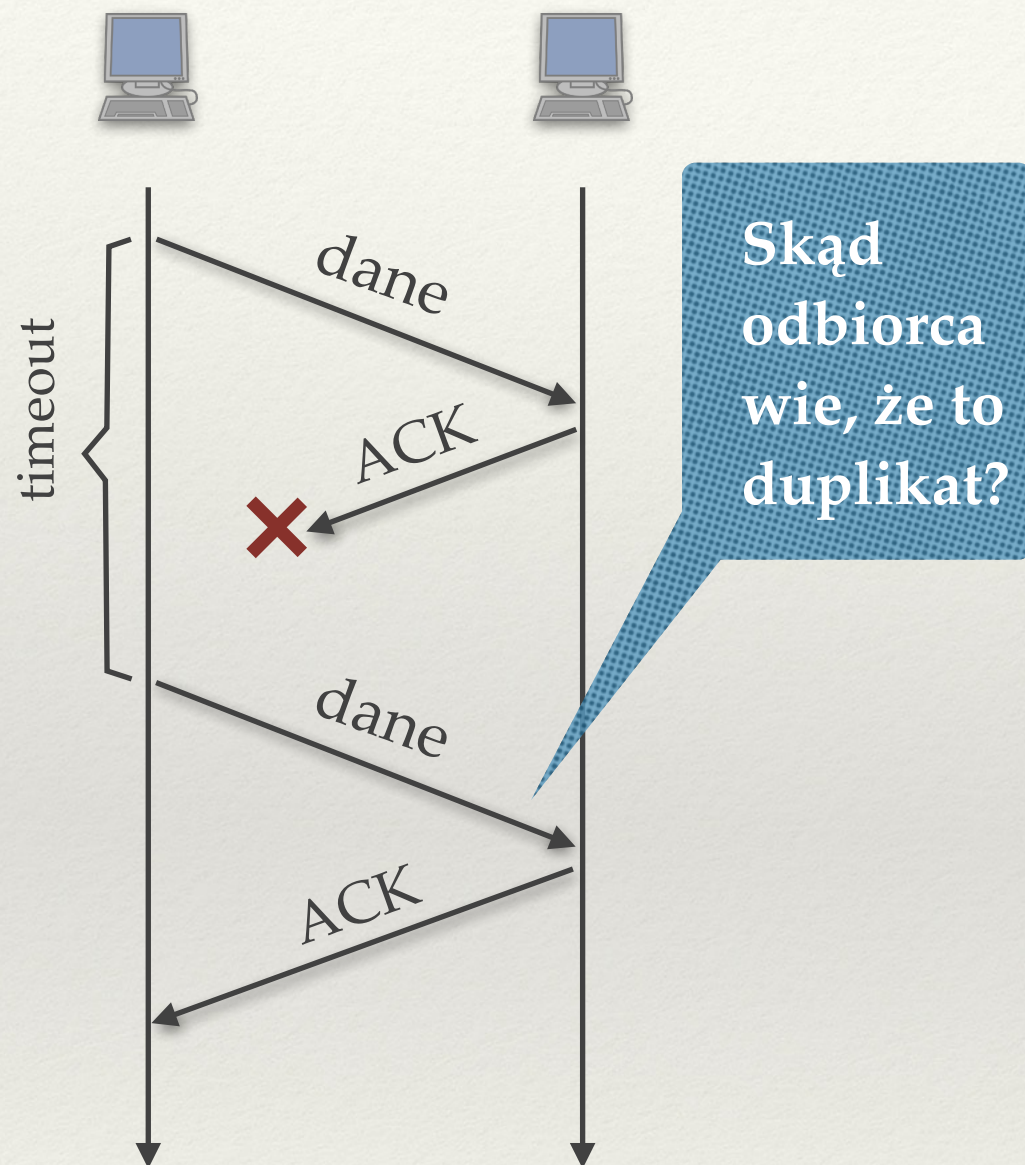
Stop-and-wait: możliwe wykonania

3. utrata ACK



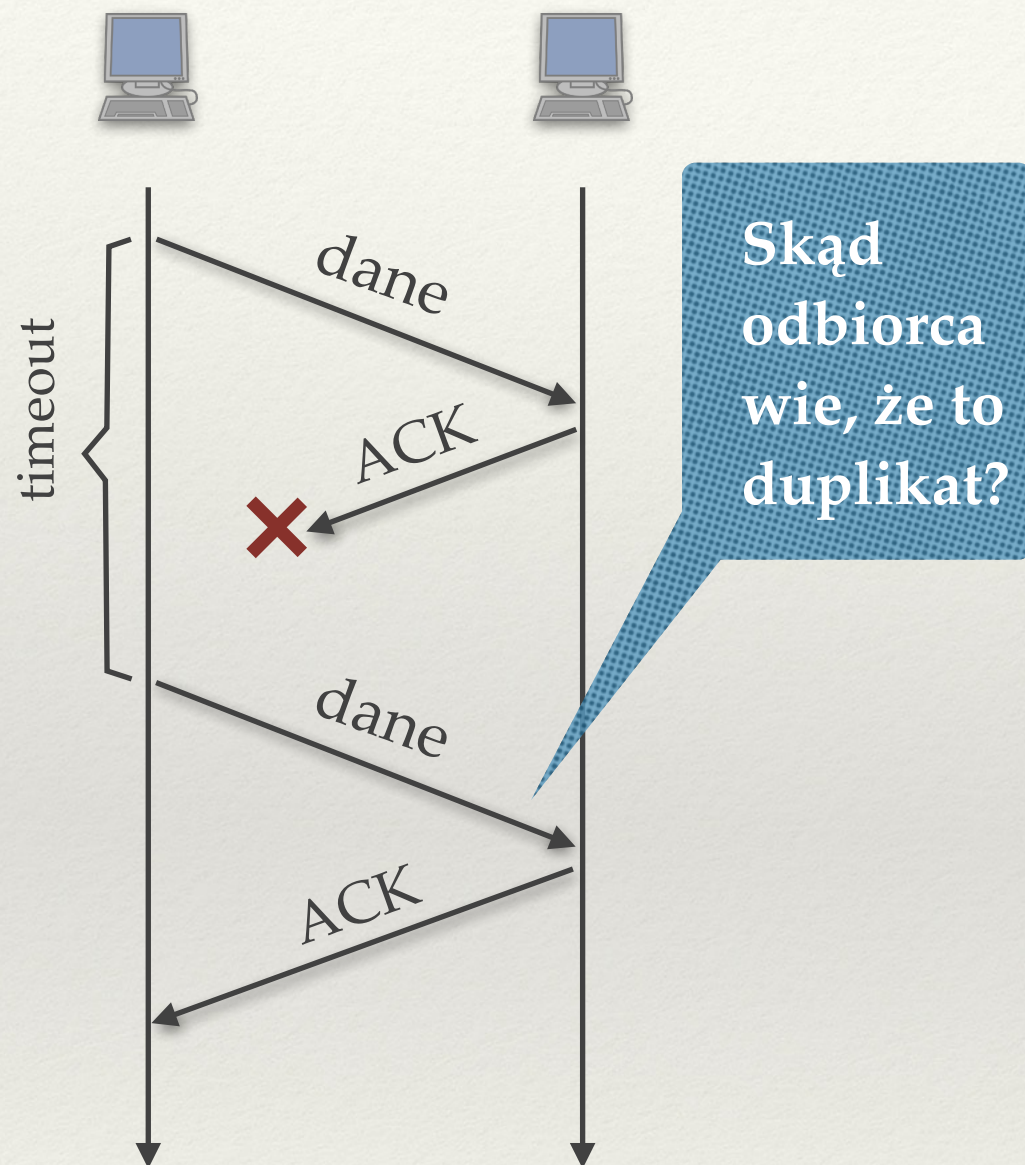
Stop-and-wait: możliwe wykonania

3. utrata ACK

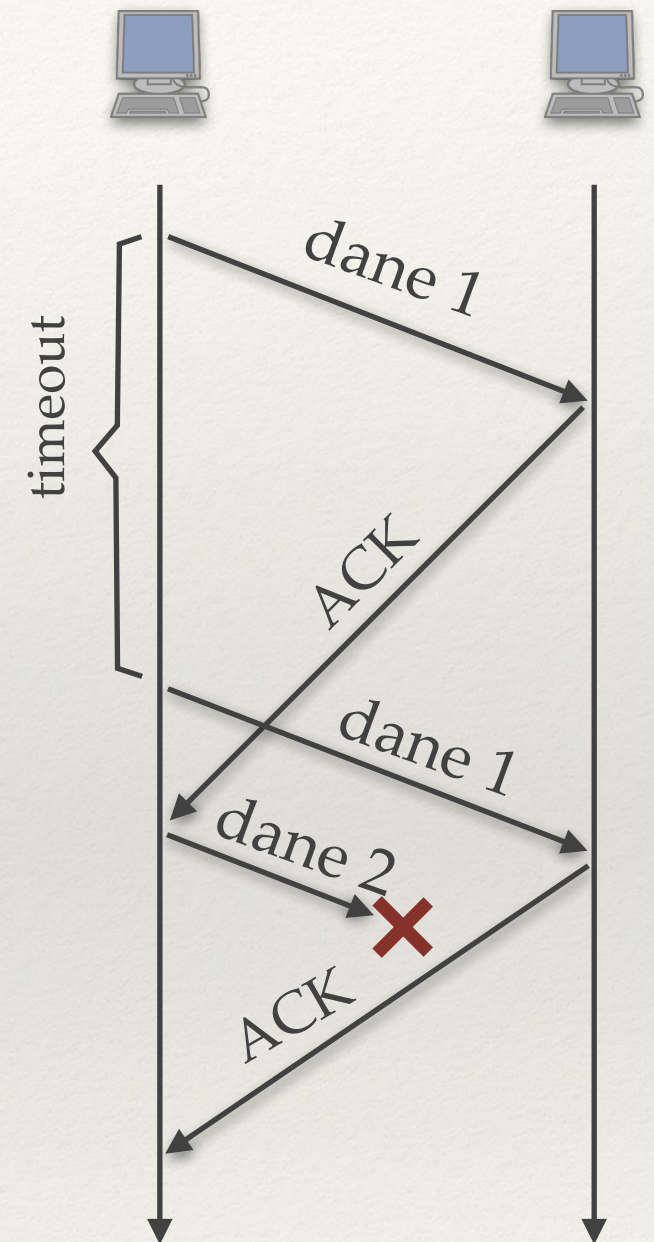


Stop-and-wait: możliwe wykonania

3. utrata ACK

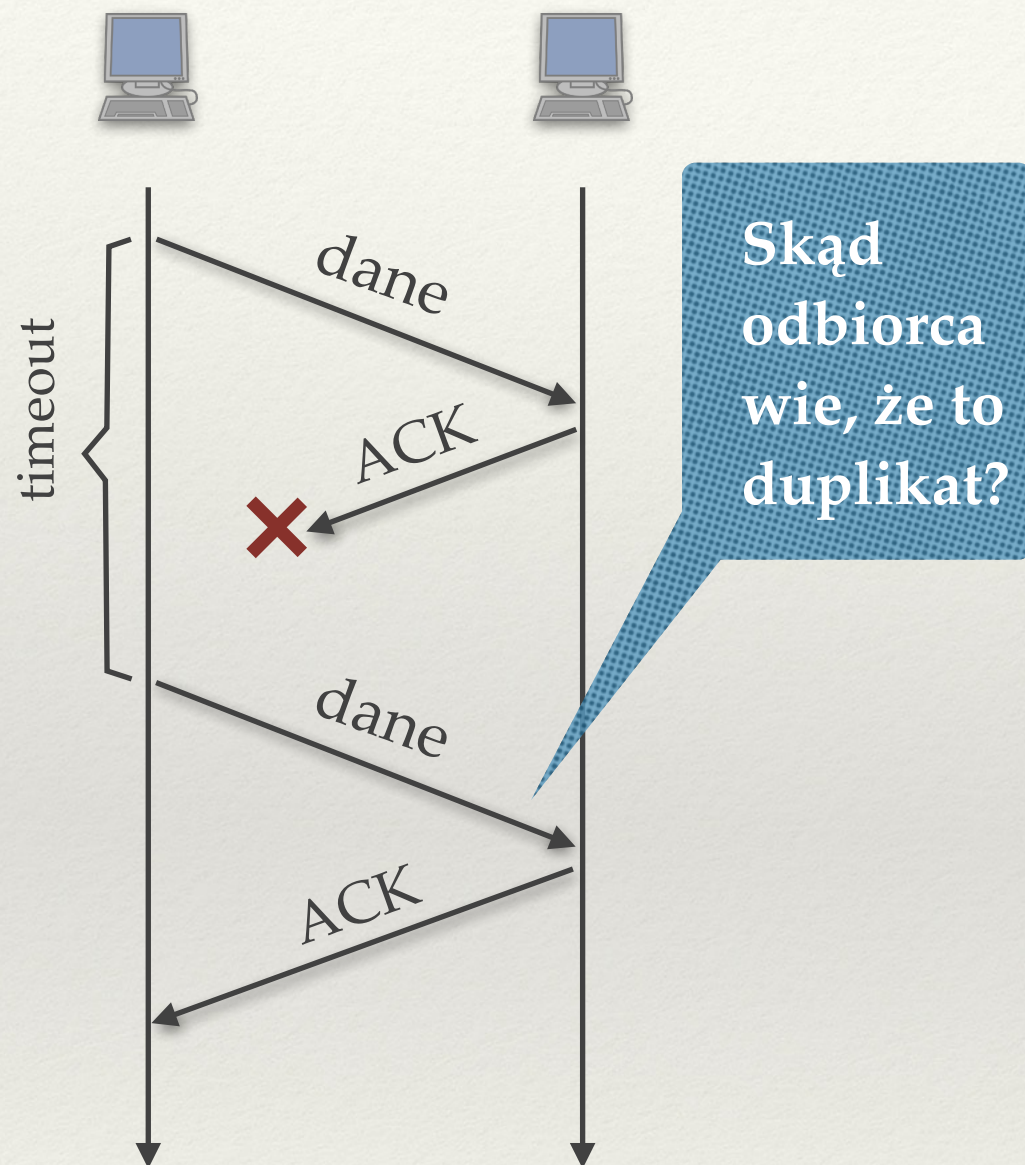


4. opóźnienie ACK + utrata danych

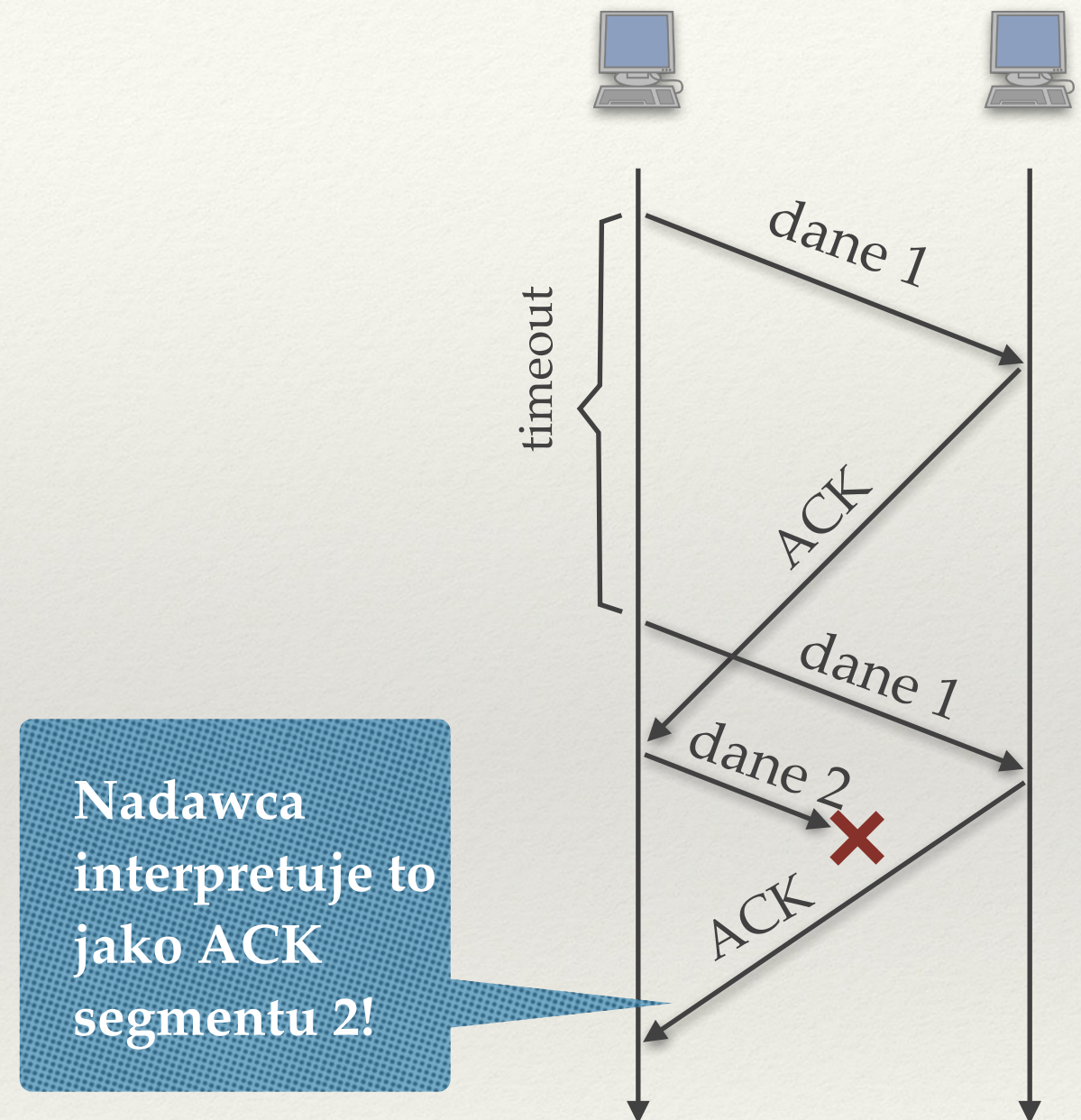


Stop-and-wait: możliwe wykonania

3. utrata ACK

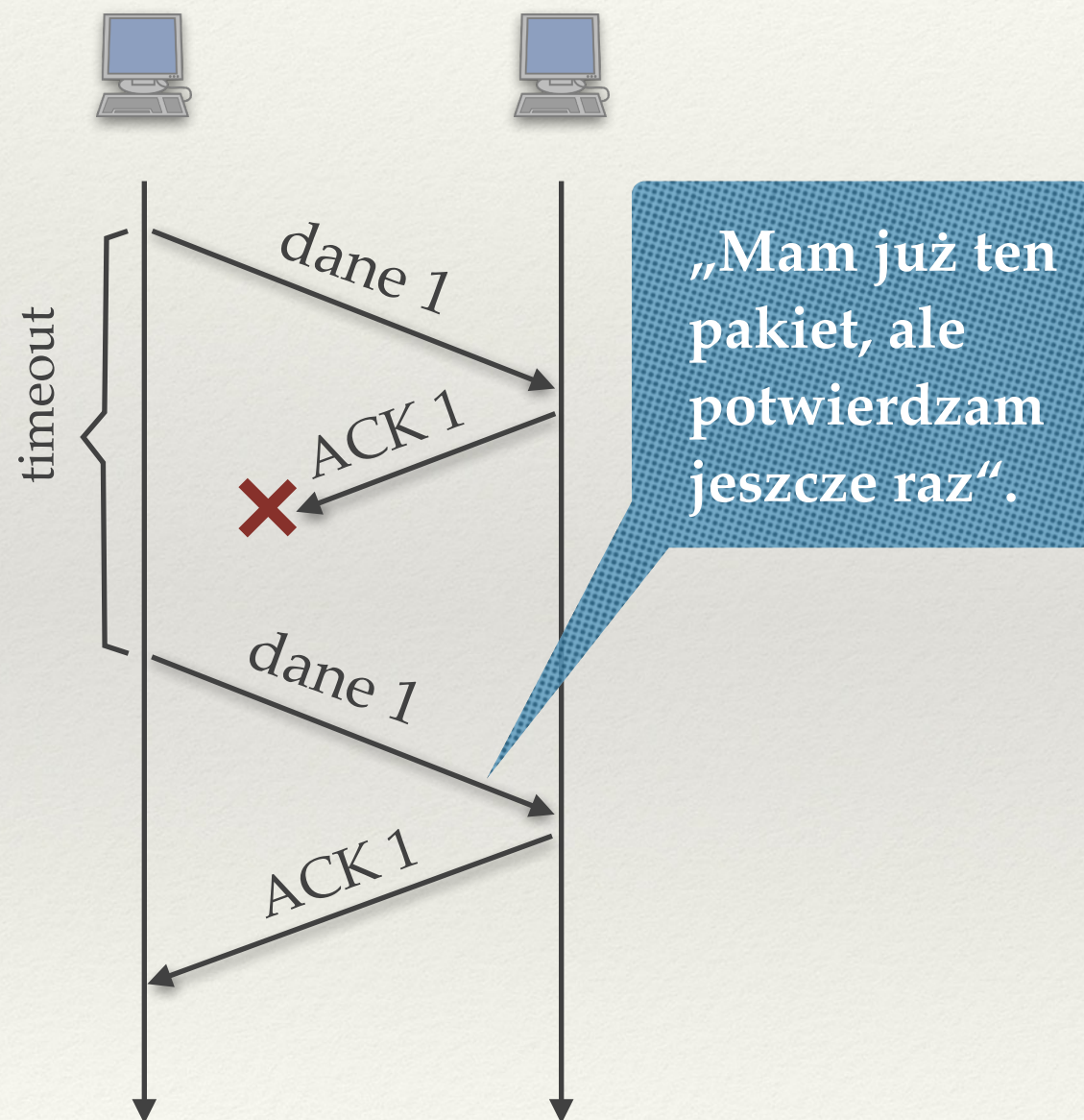


4. opóźnienie ACK + utrata danych



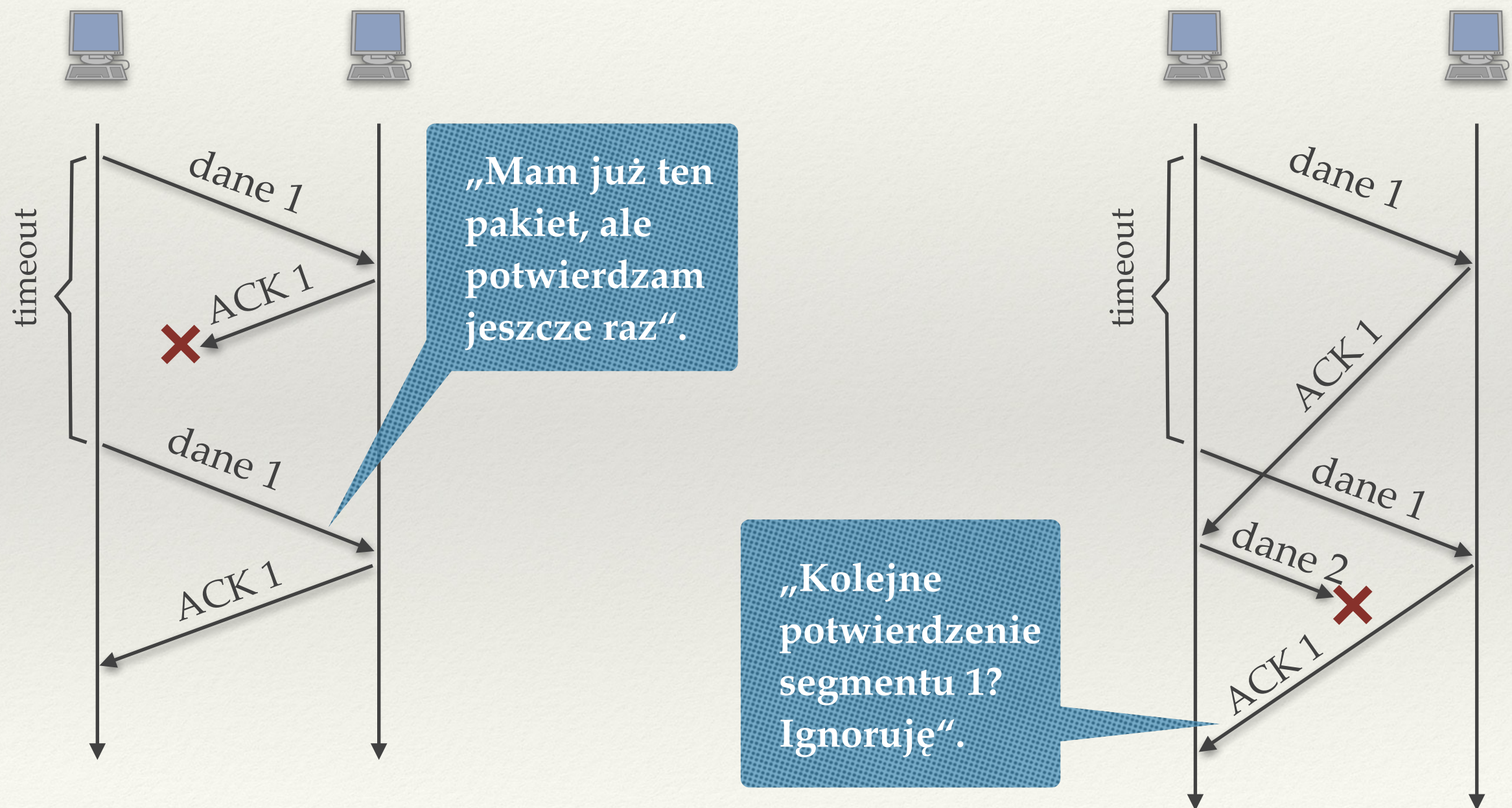
Numery sekwencyjne

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



Numery sekwencyjne

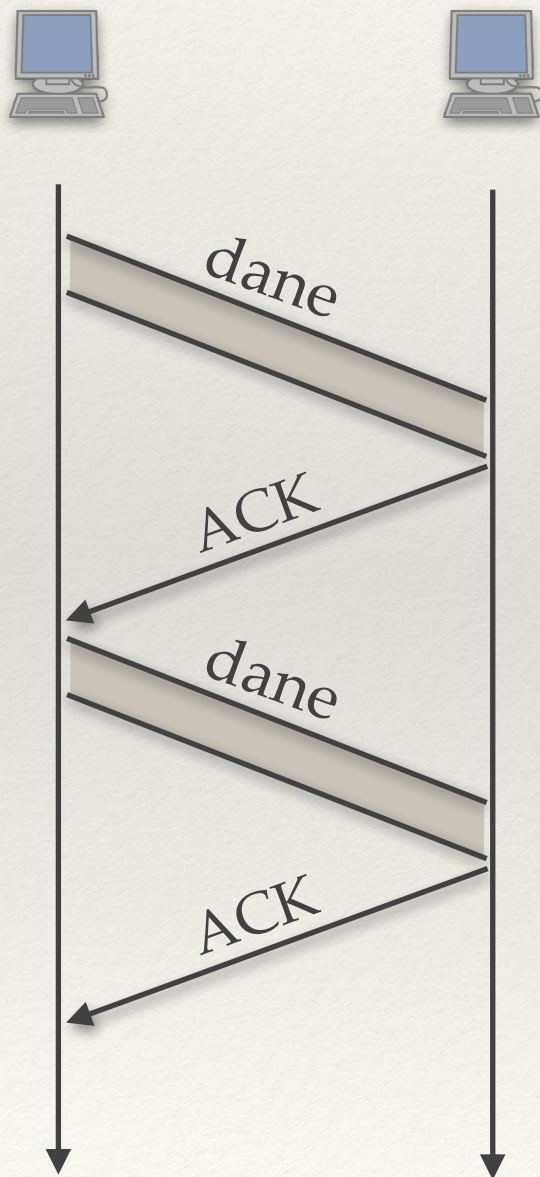
- ❖ Każdy segment jest numerowany.
- ❖ ACK zawiera numer potwierdzanego segmentu.



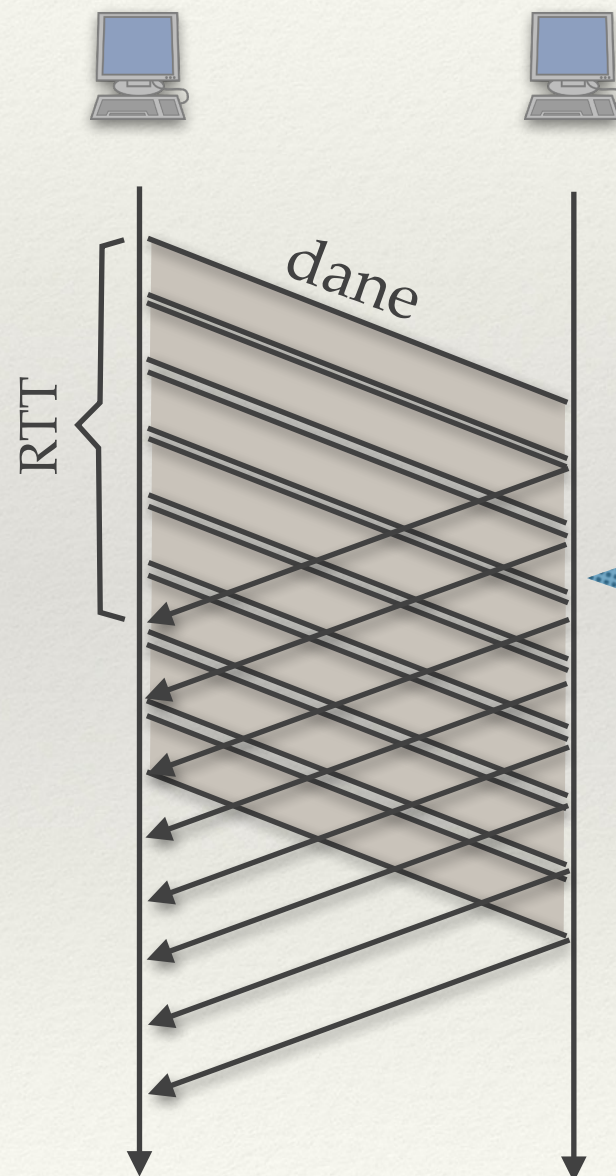
Stop-and-wait + numery sekwencyjne

- ❖ 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 BDP danych.
(BDP = *bandwidth-delay product*)

ARQ: Okno przesuwne

Przesuwne okno nadawcy (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 (okno nadawcy):



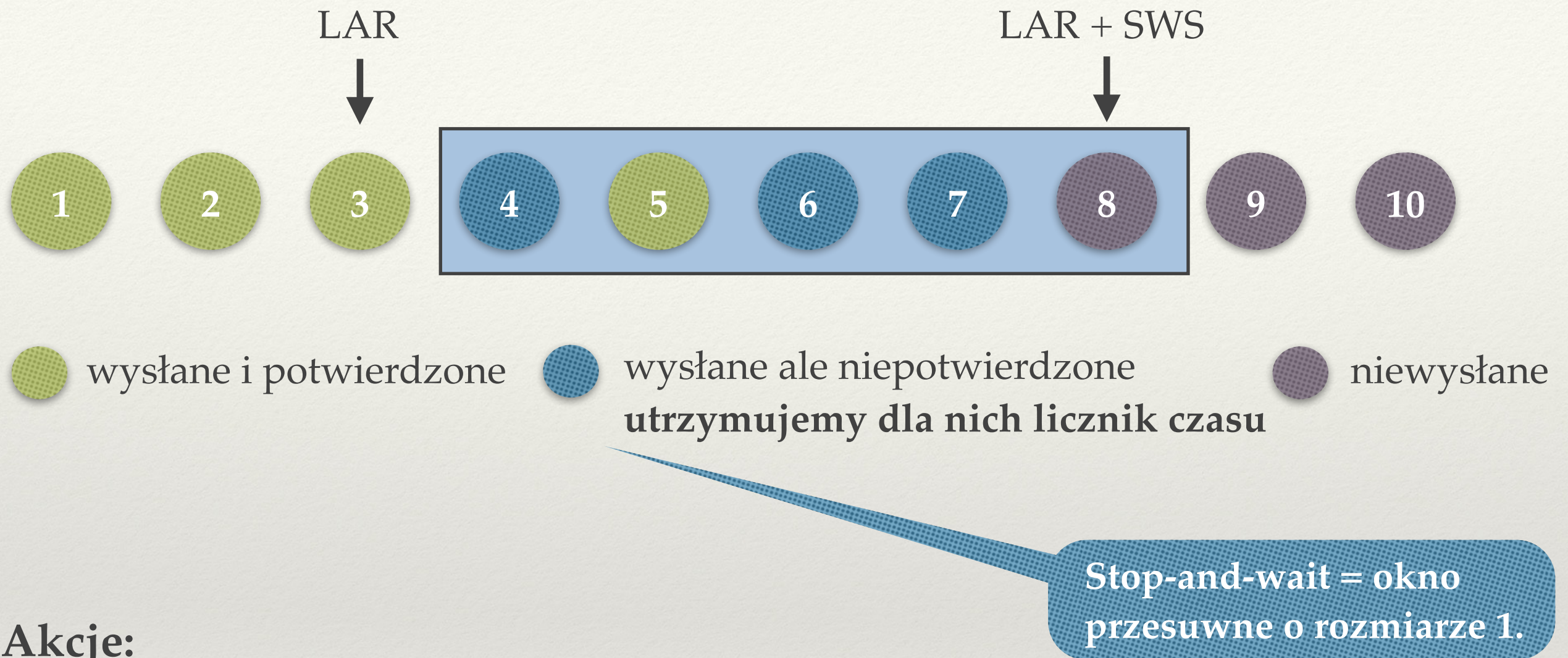
Przesuwne okno nadawcy (2)



Akcje:

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

Przesuwne okno nadawcy (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 typowe mechanizmy dla odbiorcy:

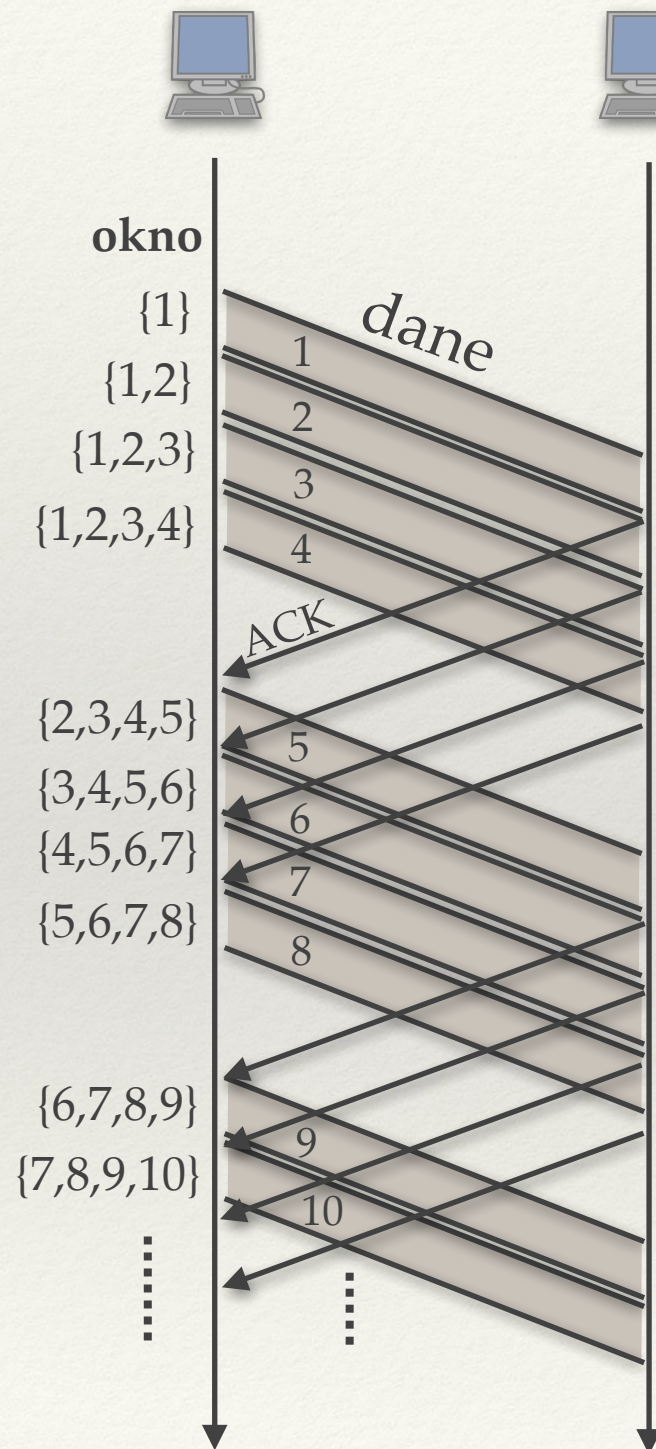
- ❖ Go-Back-N
- ❖ Potwierdzenie selektywne
- ❖ Potwierdzenie skumulowane

Potwierdzanie Go-Back-N

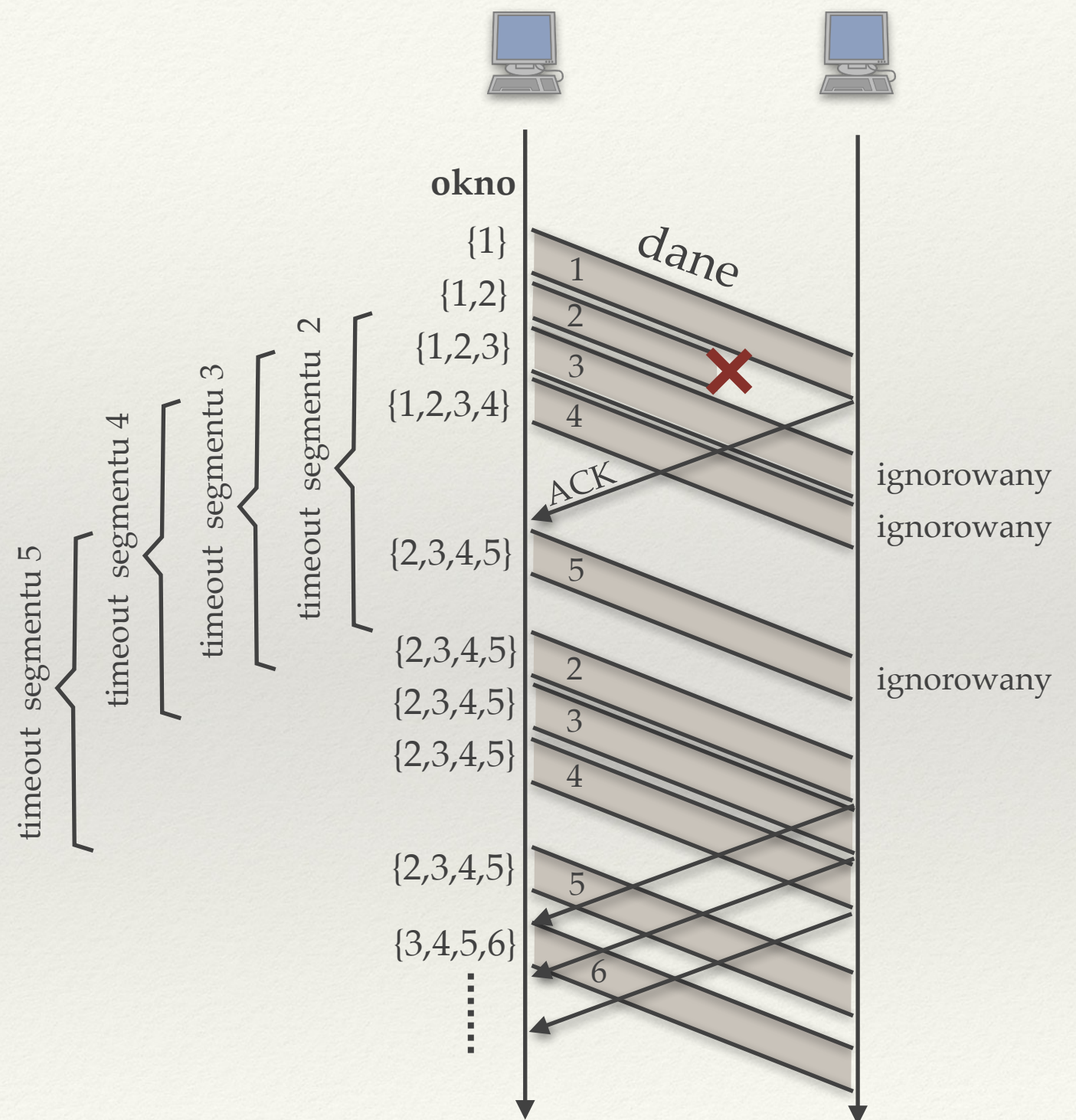
- ❖ Załóżmy, że odbiorca dostał już segmenty do P włącznie.
- ❖ Czy wysłać ACK dla otrzymanego segmentu S ?
 - ♦ Tak dla $S = P+1$.
 - ♦ Tak dla $S \leq P$ (ponowne potwierdzenie, prawdopodobnie poprzedni ACK zaginął).
 - ♦ Nie dla $S > P+1$.
 - Prosta implementacja.
 - Jeśli warstwa transportowa odbiorcy sama przetwarzałaaby segmenty (byłaby aplikacją), to nie potrzebowalibyśmy bufora odbiorcy.

Potwierdzenie Go-Back-N: przykłady

SWS = 4, pakiety nie giną

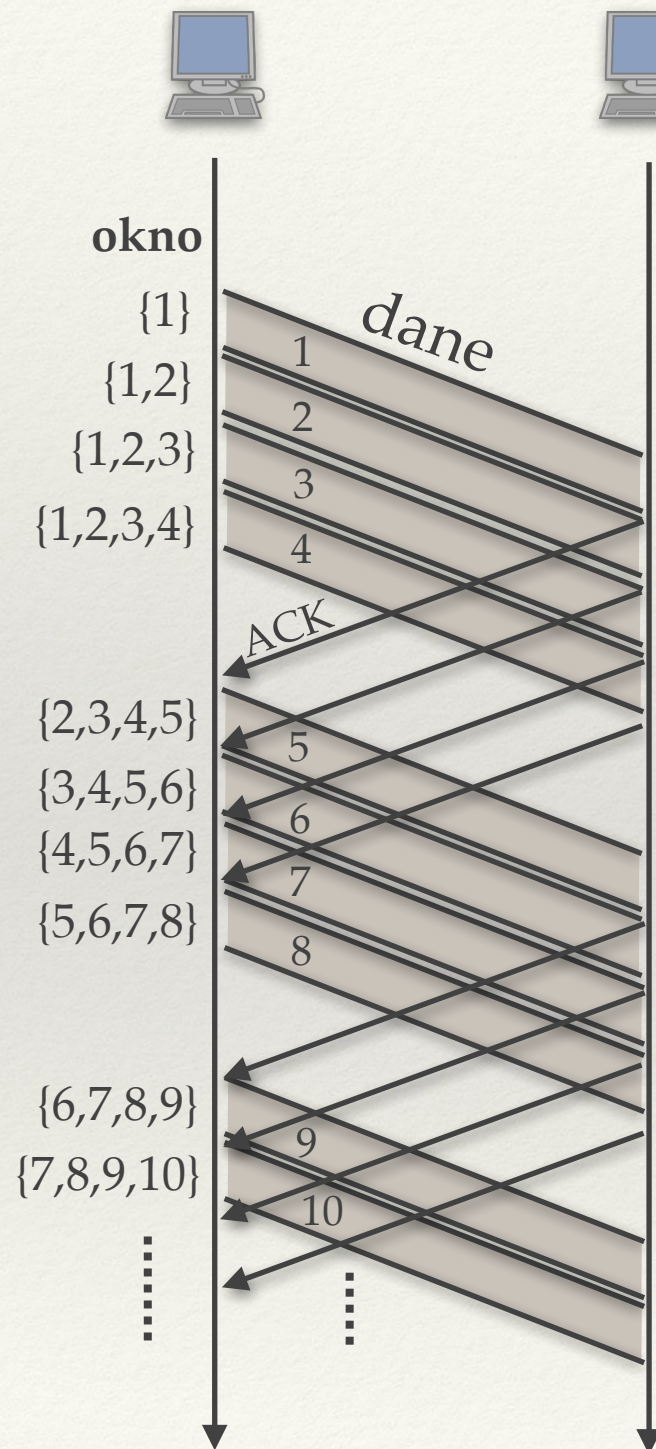


SWS = 4, pakiet ginie

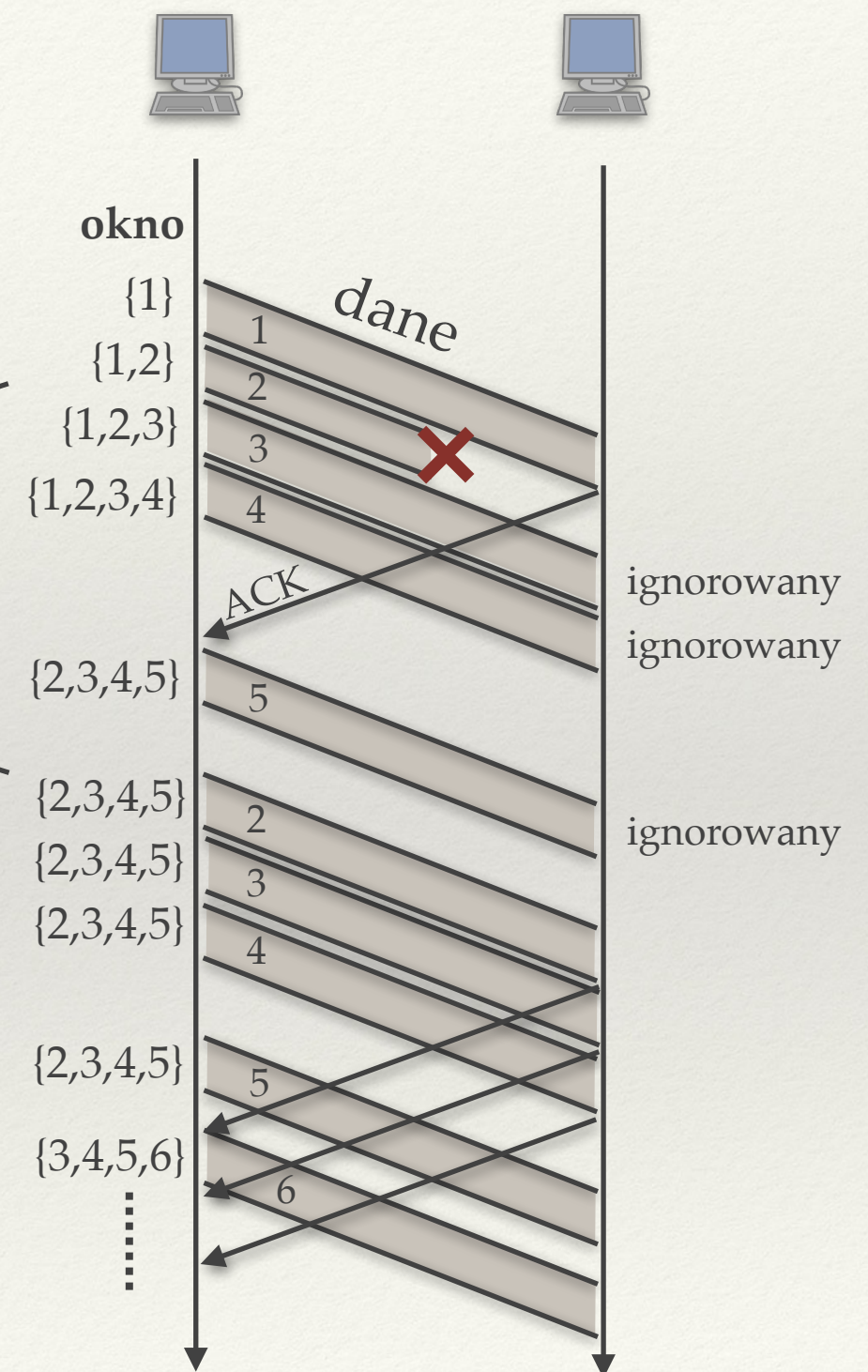


Potwierdzenie Go-Back-N: przykłady

SWS = 4, pakiety nie giną



SWS = 4, pakiet ginie

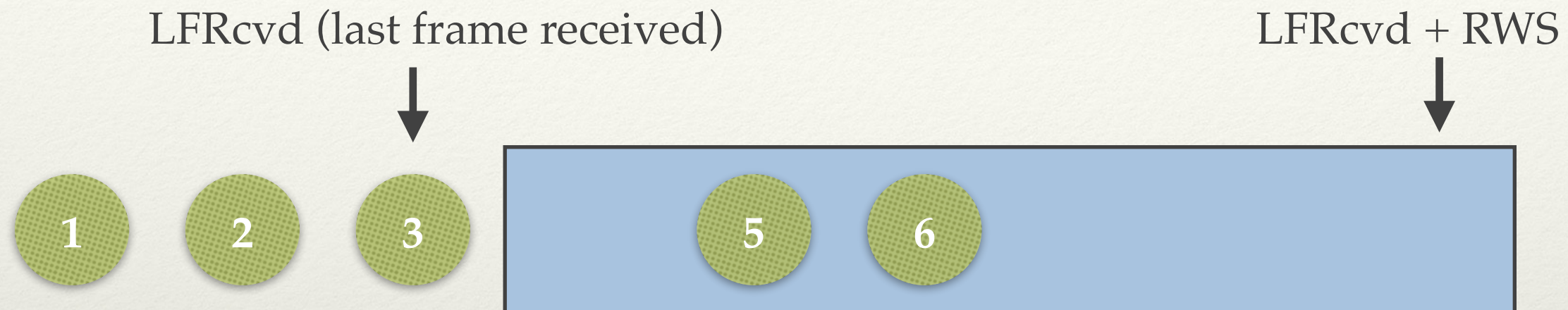


Zaginięcie jednego segmentu
→ konieczność ponownego
wysłania całego okna.

Potwierdzanie selektywne: okno odbiorcy (1)

Odbierany ciąg segmentów:

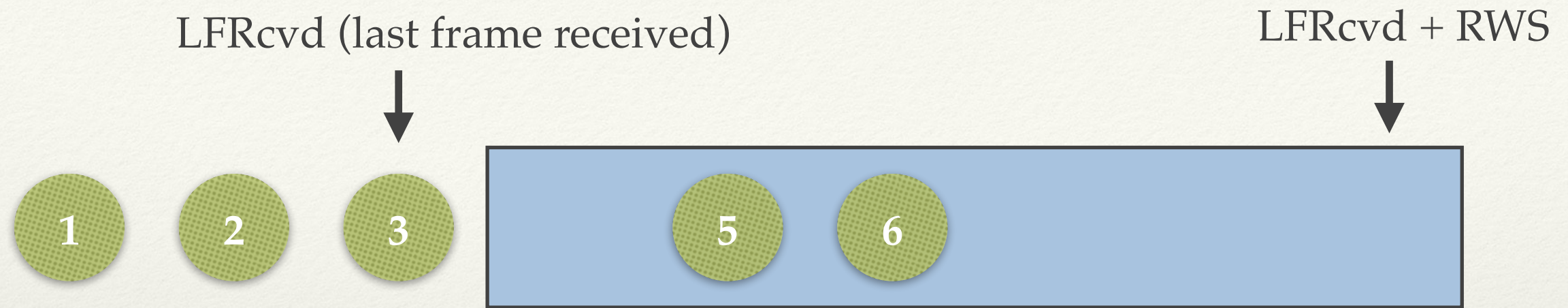
● odebrane (i potwierdzone)



Okno odbiorcy:

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

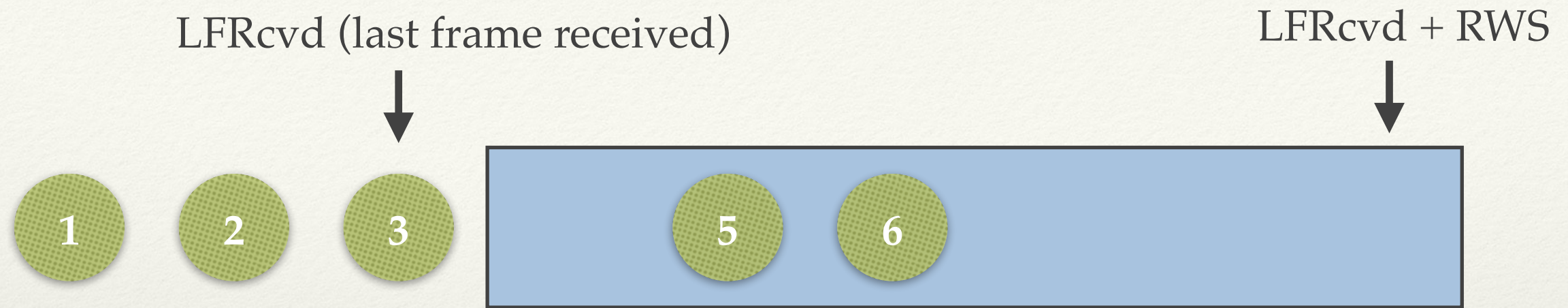
Potwierdzanie selektywne: okno odbiorcy (2)



❖ Otrzymujemy segment S

- ❖ $\text{LFRcvd} < S \leq \text{LFRcvd} + \text{RWS} \rightarrow$ zapisz segment w buforze odbiorcy.
- ❖ $S \leq \text{LFRcvd} + \text{RWS} \rightarrow$ odeślij ACK.
- ❖ $S > \text{LFRcvd} + \text{RWS} \rightarrow$ ignoruj segment.
- ❖ $S = \text{LFRcvd} + 1 \rightarrow$ aktualizuj LFRcvd (przesuń okno).

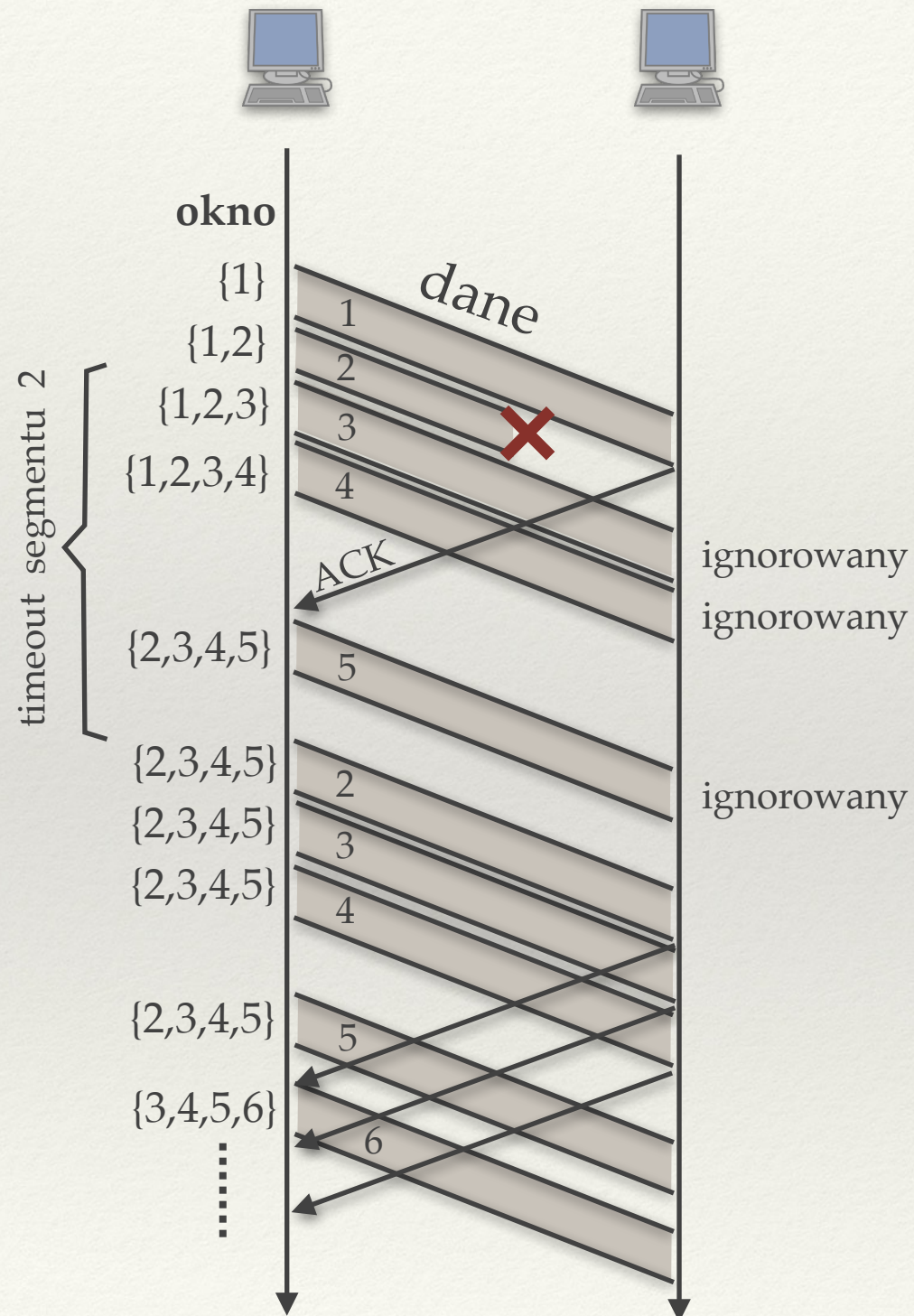
Potwierdzanie selektywne: okno odbiorcy (2)



- ❖ **Otrzymujemy segment S**
 - ♦ $LFRcvd < S \leq LFRcvd + RWS \rightarrow$ zapisz segment w buforze odbiorcy.
 - ♦ $S \leq LFRcvd + RWS \rightarrow$ odeślij ACK.
 - ♦ $S > LFRcvd + RWS \rightarrow$ ignoruj segment.
 - ♦ $S = LFRcvd + 1 \rightarrow$ aktualizuj LFRcvd (przesuń okno).
- ❖ **Dla $RWS = 1$, potwierdzanie selektywne = Go-Back-N.**

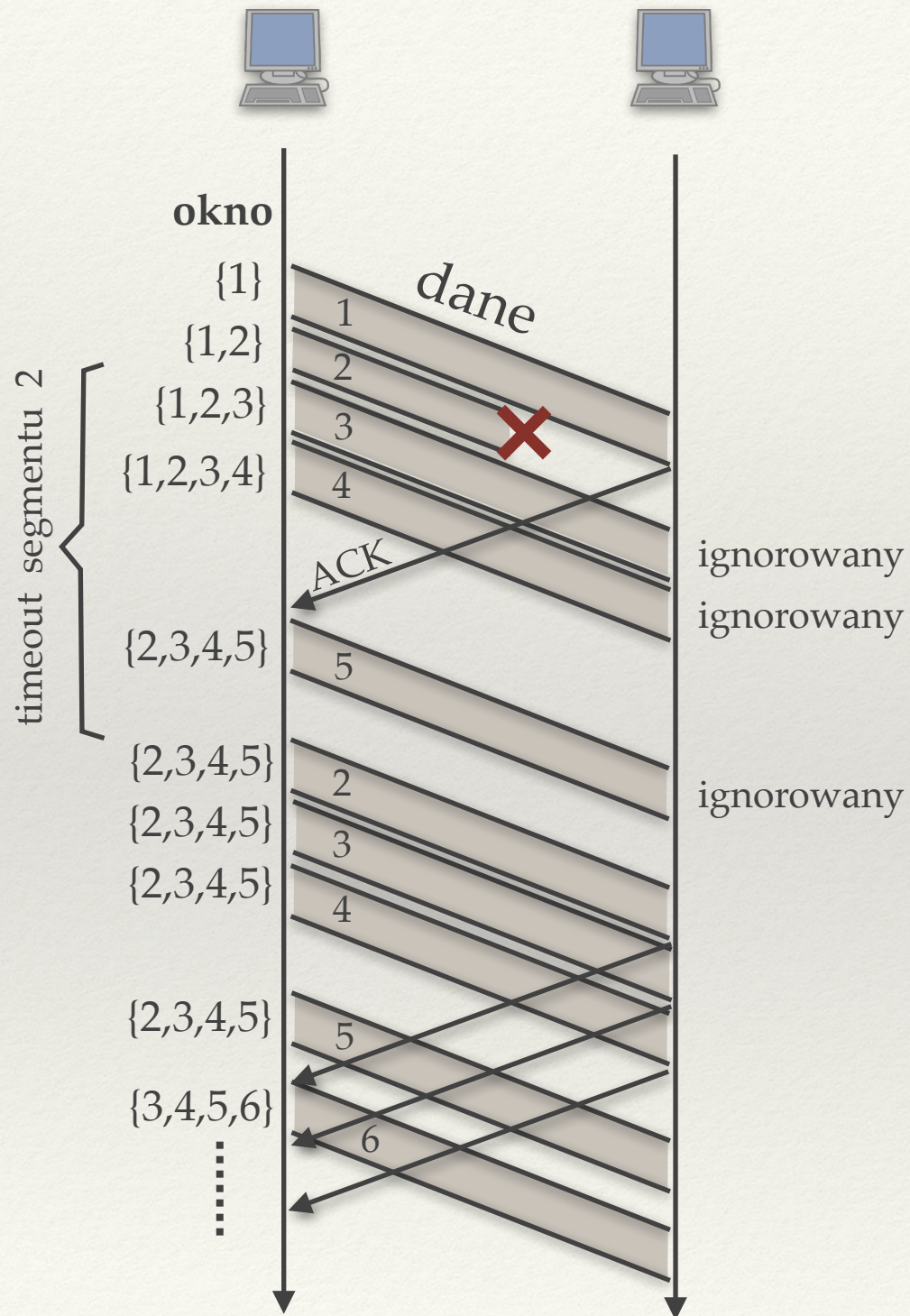
Go-Back-N vs. potwierdzenie selektywne

SWS = 4, Go-Back-N

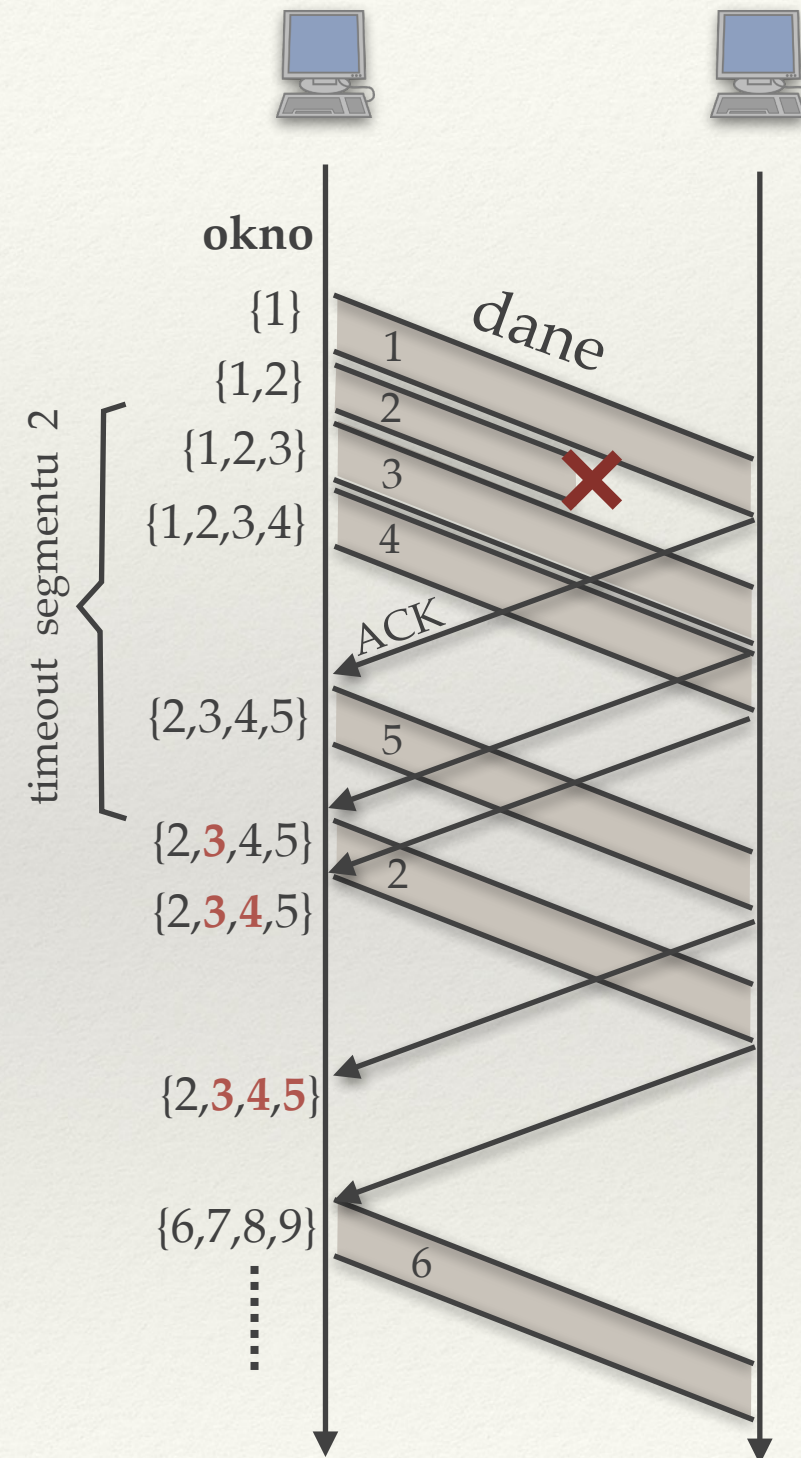


Go-Back-N vs. potwierdzenie selektywne

SWS = 4, Go-Back-N

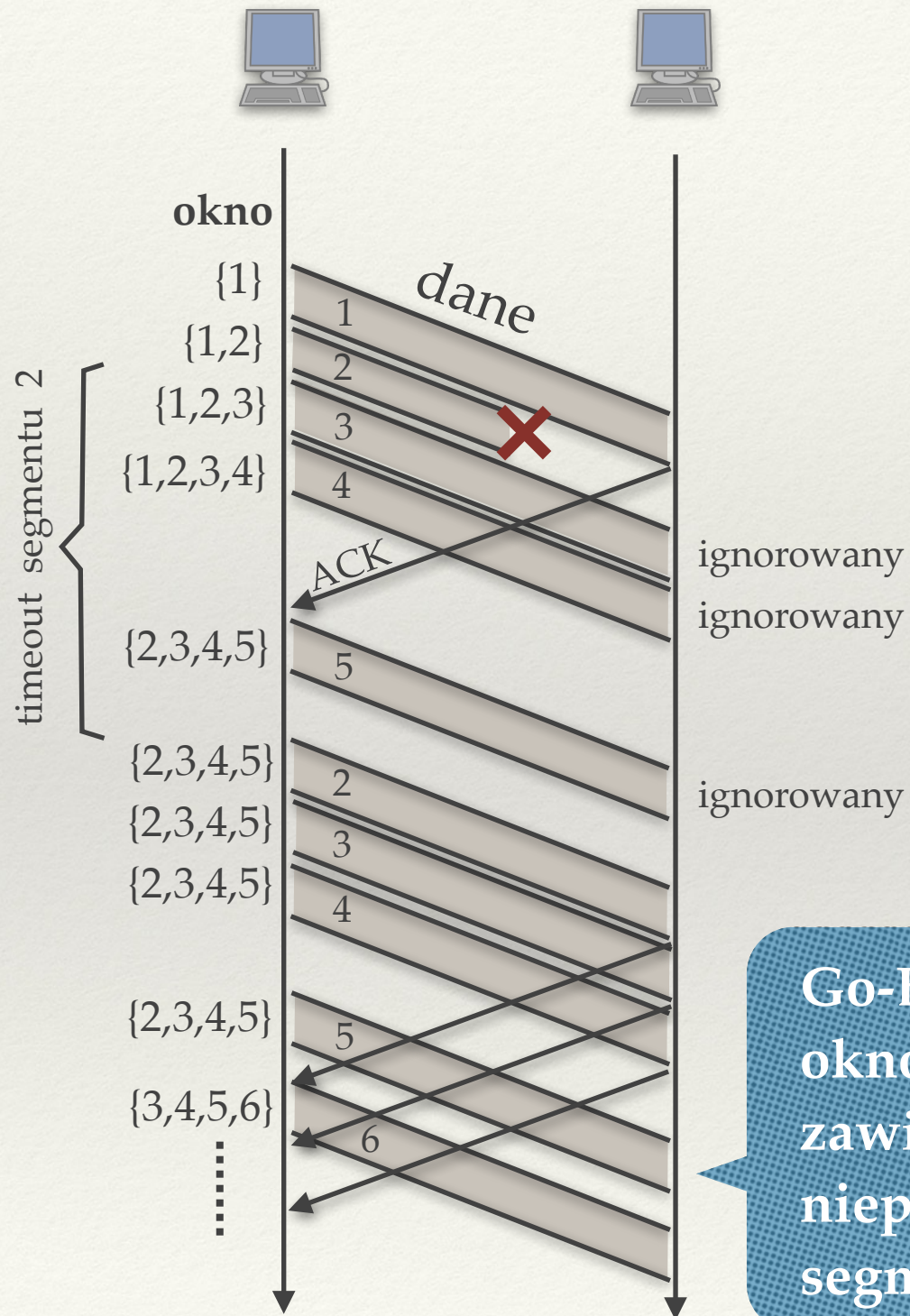


SWS = 4, potwierdzenie selektywne

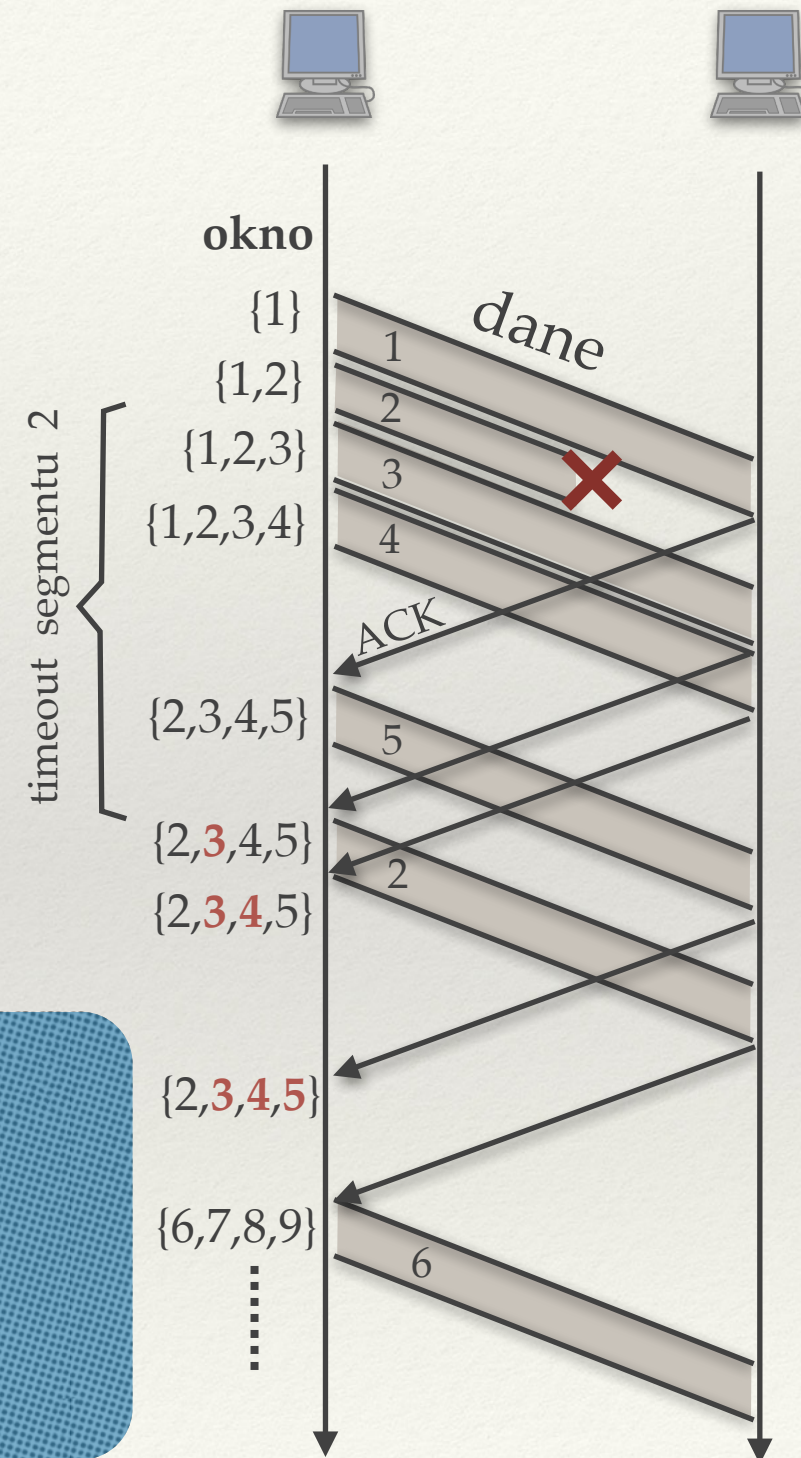


Go-Back-N vs. potwierdzenie selektywne

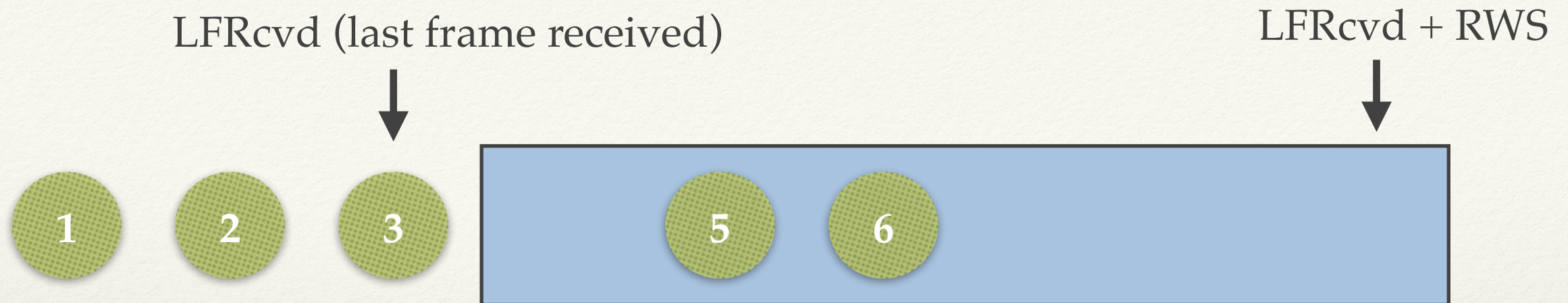
SWS = 4, Go-Back-N



SWS = 4, potwierdzenie selektywne



Potwierdzenie skumulowane



- ❖ Poza ACK wszystko jak przy potwierdzaniu selektywnym.
- ❖ Wysyłanie ACK:
 - ♦ Wysyłamy tylko jeśli otrzymamy segment $S \leq \text{LFRcvd} + \text{RWS}$ (jak poprzednio)
 - ♦ W razie potrzeby aktualizujemy LFRcvd (przesuwamy okno w prawo) a następnie wysyłamy **ACK dla LFRcvd**.

[animacja \(wymaga Flash\)](#)

Potwierdzanie skumulowane vs. selektywne

- ❖ **Selektywne.** Wiemy dokładnie które pakiety dotarły
- ❖ **Skumulowane.** Możemy wprowadzić mechanizm opóźnionych potwierdzeń:
 - ✦ Jeśli są dane do wysłania w drugą stronę, to wysyłamy ACK z tymi danymi.
 - ✦ W p.p. wymuszamy określony czas (ułamek RTT) pomiędzy kolejnymi ACK.
 - ✦ Zmniejsza liczbę potwierdzeń bez generowania timeoutów.
- ❖ **Selektywne i skumulowane.** Wnioskowanie na podstawie „dziur” w ACK:
 - ✦ Selektywne: ACK 1, ACK 2, ACK 4, ACK 5.
 - ✦ Skumulowane: ACK 1, ACK 2, ACK 2, ACK 2, ACK 2.
 - ✦ Być może warto wysłać segment 3 bez czekania na jego timeout?

Kontrola przepływu

Kontrola przepływu a warstwa aplikacji (1)

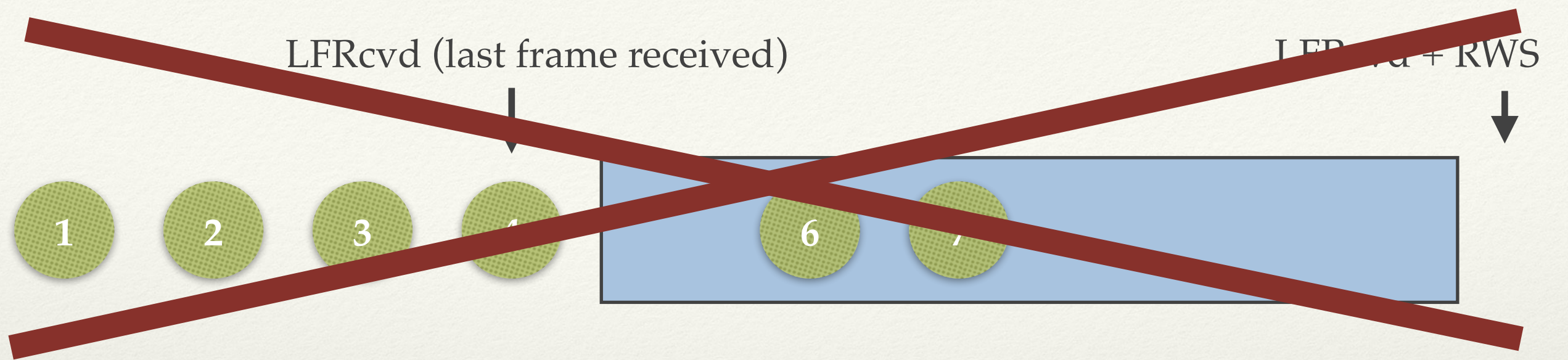
Co to znaczy „przekazujemy dane do warstwy aplikacji”?

- ❖ To warstwa aplikacji pobiera dane funkcją `read()`.
- ❖ Okno odbiorcy to bufor na odebrane pakiety, jeszcze nieprzeczytane przez aplikację.

Kontrola przepływu a warstwa aplikacji (2)

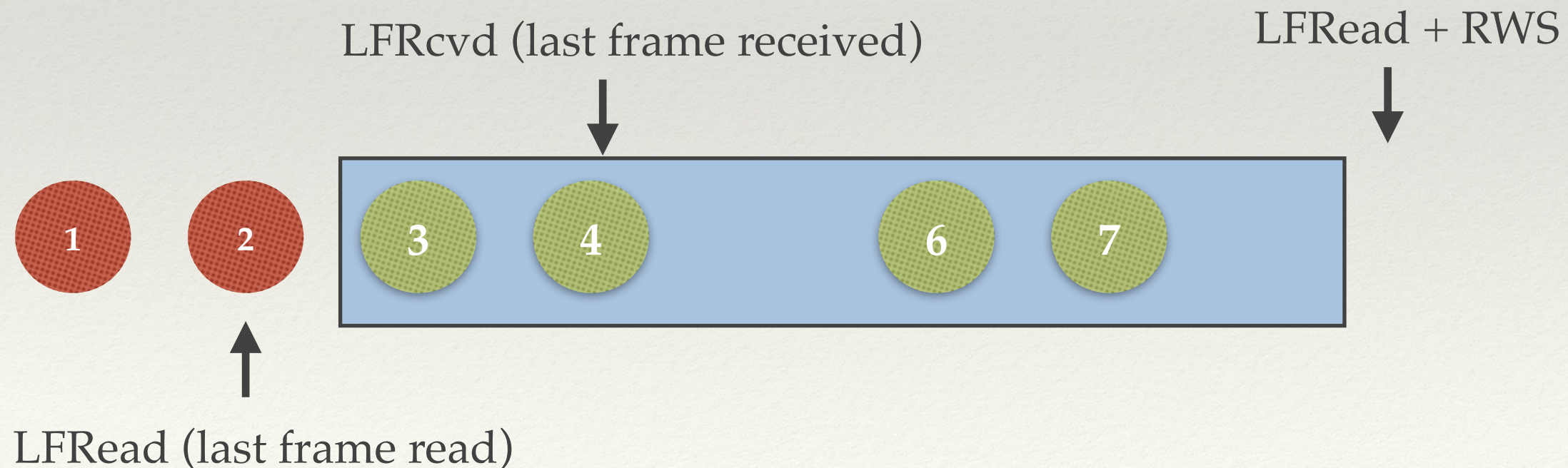


Kontrola przepływu a warstwa aplikacji (2)



● odebrane, potwierdzone, przeczytane przez aplikację

● odebrane (i potwierdzone) ale nieprzeczytane przez aplikację



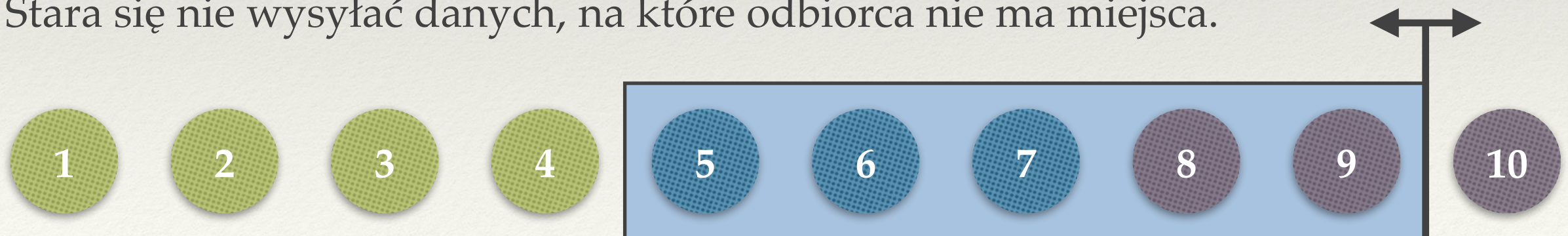
Kontrola przepływu: oferowane okno

Odbiorca:

- ❖ Oferowane okno = wolne miejsce w buforze
- ❖ Oferowane okno wysyłane nadawcy (zazwyczaj razem z ACK).
- ❖ Np.: pakiety potwierdzone, ale aplikacja wolno czyta → oferowane okno jest małe.

Nadawca:

- ❖ Zmienia SWS (rozmiar swojego okna) na rozmiar oferowanego okna.
- ❖ Stara się nie wysyłać danych, na które odbiorca nie ma miejsca.



Niezawodny transport: TCP

TCP

❖ Numeruje bajty, a nie segmenty.

demonstracja

❖ Potwierdzanie skumulowane:

- ♦ Uwaga: ACK n = oznacza „mam wszystko do bajtu $n-1$ włącznie”.
- ♦ ACK zazwyczaj wysyłany w pakiecie razem z danymi w drugą stronę.

0				7 8				15 16				23 24				31			
port źródłowy								port docelowy											
numer sekwencyjny (numer pierwszego bajtu w segmencie)																			
numer ostatniego potwierdzanego bajtu + 1																			
offset		000		ECN		U-A-P-R-S-F				oferowane okno									
suma kontrolna								wskaźnik pilnych danych											
dodatkowe opcje, np. potwierdzanie selektywne																			

Numerowanie bajtów: przykładowe komplikacje

- ❖ Oferowaliśmy okno = 0 i aplikacja zwolniła miejsce?
→ Wyślij osobno rozmiar okna (bez ACK).
- ❖ Nie mamy danych do wysłania w drugą stronę?
→ Opóźnione wysyłanie ACK.
- ❖ Oferowane okno jest mniejsze niż MSS?
→ Czekamy z nadawaniem.
- ❖ Aplikacja generuje dane mniejsze niż MSS
→ Wyślij dopiero jeśli kiedy wszystkie poprzednie dane zostaną potwierdzone (mechanizm Nagle'a).
→ Co z interaktywnymi programami (praca zdalna)?

Ustawianie timeoutu dla segmentu

- ❖ Obliczamy średnie RTT ważone wykładniczo czasem
 - ♦ $\text{avg-RTT} = \alpha \times \text{avg-RTT} + (1 - \alpha) \times \text{zmierzone-RTT}$
 - ♦ RTT segmentów stałe \rightarrow avg-RTT zbiega do tej wartości.
 - ♦ W podobny sposób mierzymy wariancję RTT (var-RTT).
- ❖ $\text{RTO (retransmission timeout)} = 2 \times \text{avg-RTT} + 4 \times \text{var-RTT}$

Gdzie implementować?

Gdzie implementować warstwę transportową?

Warstwa sieciowa:

- ❖ Implementowana na routerach i urządzeniach końcowych.

Warstwa transportowa:

- ❖ Wczesna filozofia: niezawodność dostarczania zapewniać na każdym łączy.
- ❖ Problem: błąd może nie być związany z łączyem!



- ❖ 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>