

---

# Kontrola przeciążenia

---

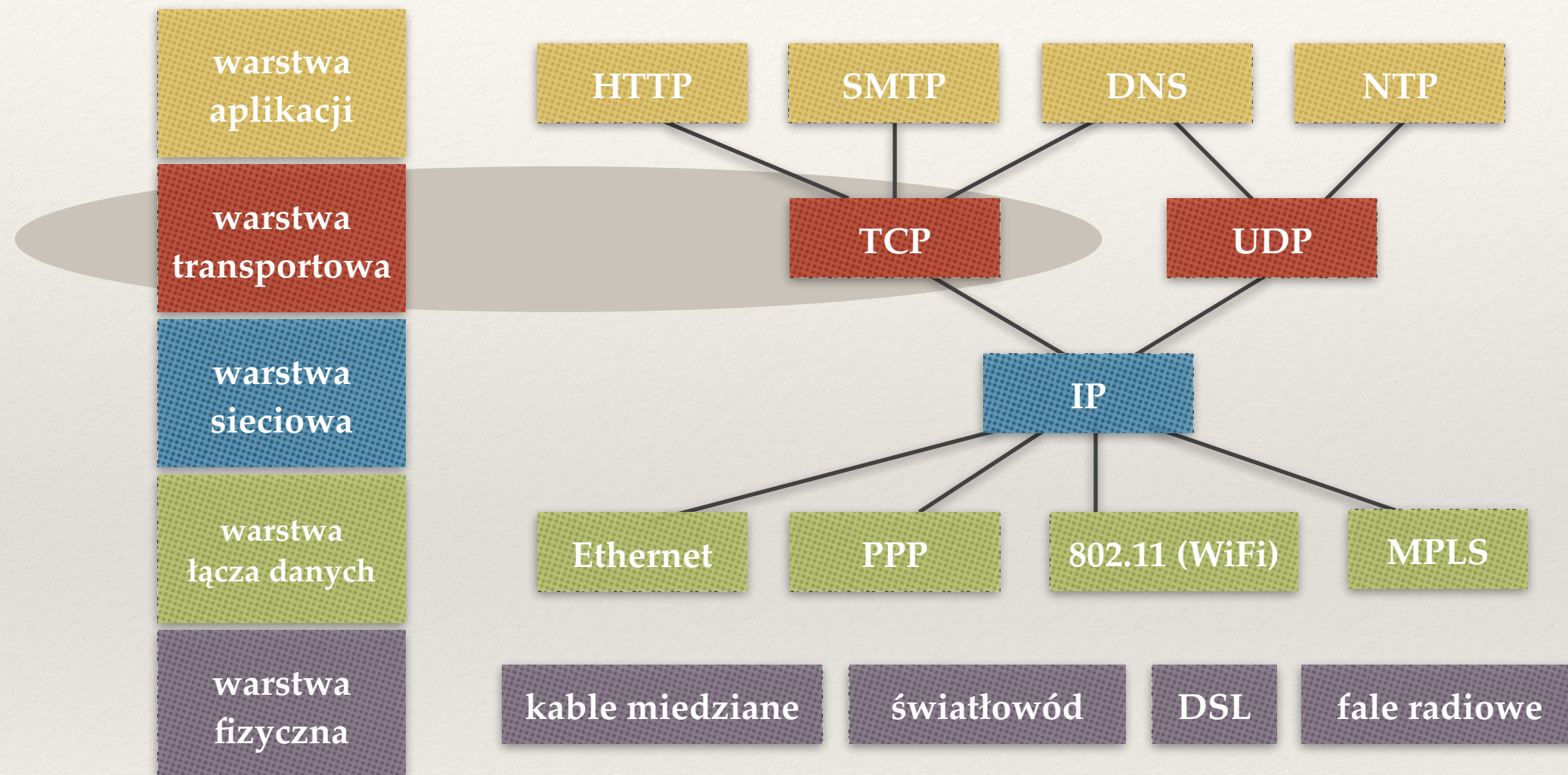
Sieci komputerowe

Wykład 13

*Marcin Bieńkowski*



# Protokoły w Internecie





---

# Podsumowanie mechanizmów warstwy transportowej

---



# Warstwa sieci = zawodna usługa przesyłania pakietów

---

- ❖ Tylko zasada dołożenia wszelkich starań (*best effort*).
  
- ❖ **Pakiety mogą zostać:**
  - ♦ uszkodzone,
  - ♦ zgubione,
  - ♦ opóźnione,
  - ♦ zamienione (kolejność),
  - ♦ zduplikowane (przez wyższe lub niższe warstwy).



# Podstawowe mechanizmy w warstwie transportowej

---

- ❖ **Segmentacja:** dzielimy przesyłany strumień danych na kawałki; dla uproszczenia będziemy wszystko liczyć w segmentach.
- ❖ **Potwierdzenia (ACK):** małe pakiety kontrolne potwierdzające otrzymanie danego segmentu.
- ❖ **Timeout (przekroczenie czasu oczekiwania):** jeśli nie otrzymamy potwierdzenia przez pewien czas (typowy dla łącza, np.  $2 * RTT$ ).
- ❖ **Retransmisje:** ponowne wysłanie danego segmentu w przypadku przekroczenia czasu oczekiwania (RTO).



# Co umiemy już zapewniać?

---

## ❖ **Niezawodny transport**

- ♦ Mechanizmy ARQ (Automatic Repeat reQuest) = wysyłanie do skutku

## ❖ **Kontrola przepływu**

- ♦ Nadawca powinien dostosowywać prędkość transmisji do szybkości z jaką odbiorca może przetwarzać dane.

## ❖ **Jak?**

- ♦ Najczęściej: okno przesuwne + potwierdzenia skumulowane.



# Przypomnienie: przesuwne okno nadawcy

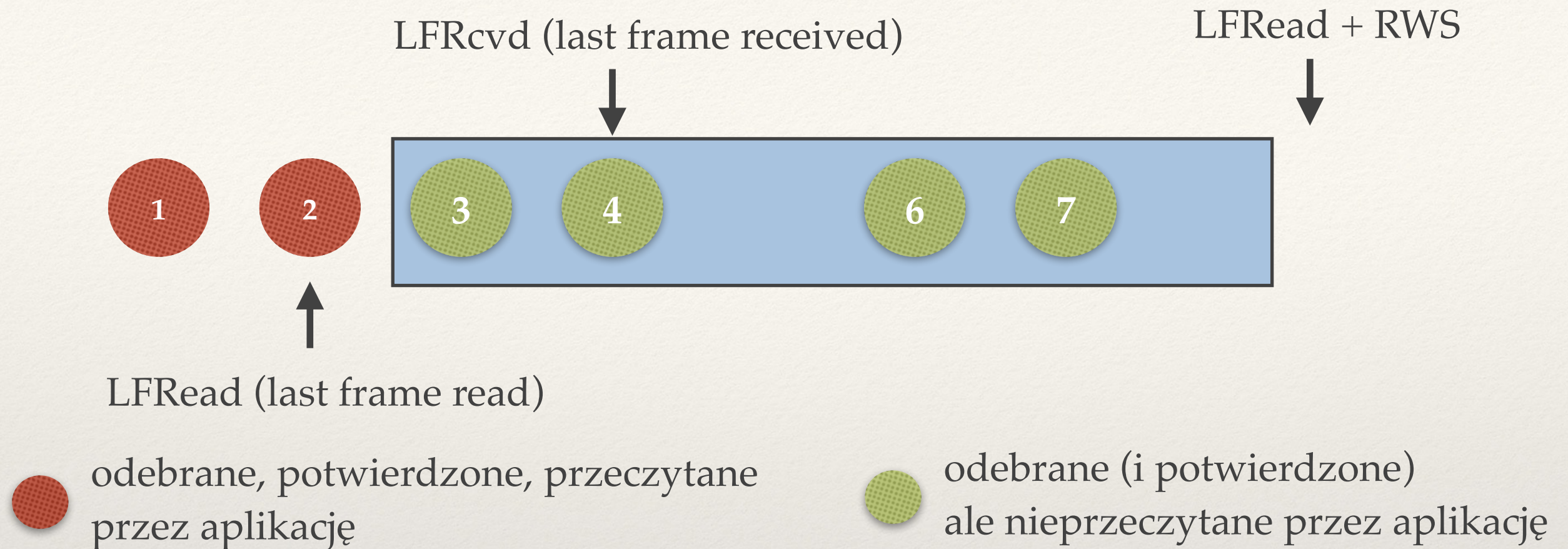


## Akcje:

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



# Przypomnienie: okno odbiorcy, potwierdzanie skumulowane



## ❖ Wysyłanie ACK:

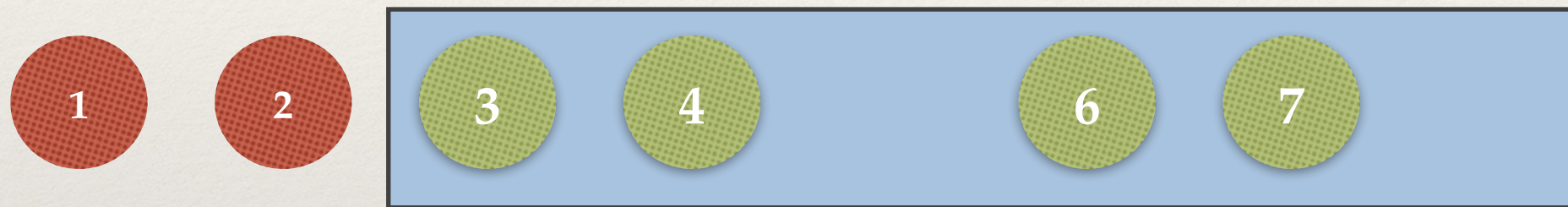
- ❖ Wysyłamy tylko jeśli otrzymamy segment  $S \leq \text{LFRRead} + \text{RWS}$ .
- ❖ W razie potrzeby aktualizujemy LFRcvd (przesuwamy okno w prawo) a następnie wysyłamy **ACK dla LFRcvd**.



# Przypomnienie: 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:

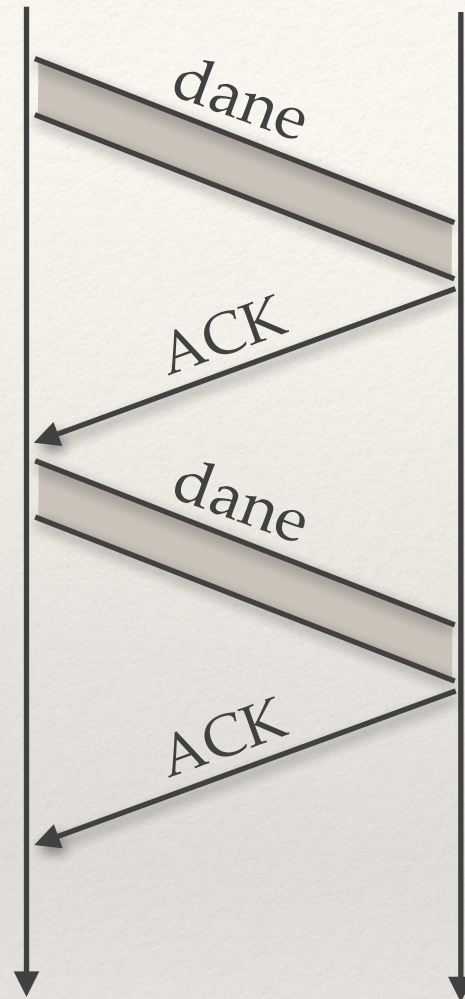
- ❖ Zmienia SWS (rozmiar swojego okna) na rozmiar oferowanego okna.
- ❖ W efekcie zazwyczaj nie wysyła danych, na które odbiorca nie ma miejsca.



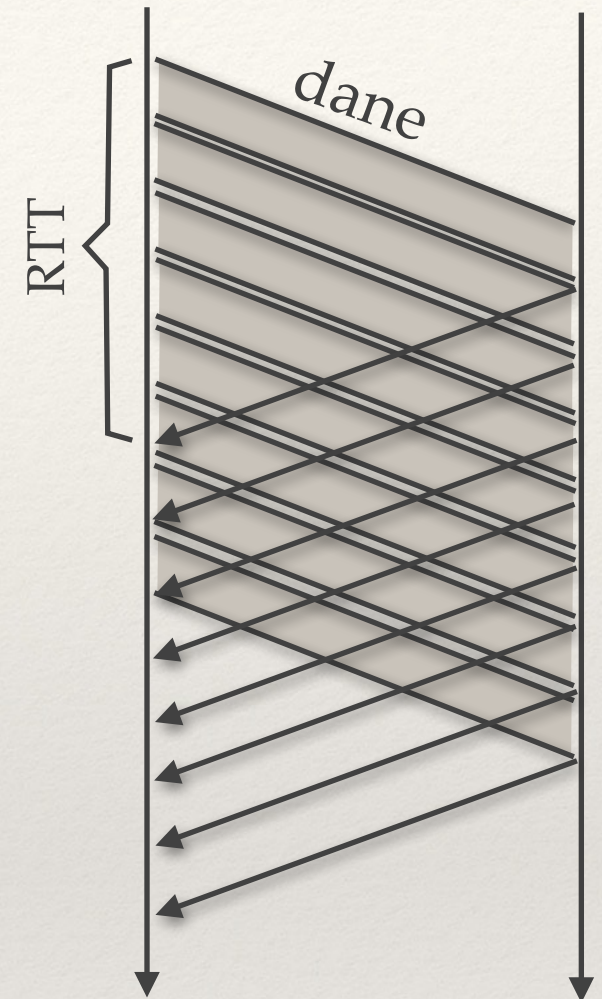


# SWS a prędkość transmisji

SWS = 1:



większe SWS:



- ❖ Dane wysyłane są ze średnią prędkością  $SWS / RTT$ .
- ❖ Okno mniejsze od  $BDP = przepustowość * RTT$   
→ nadawca nie jest w stanie wykorzystać całego łącza.



---

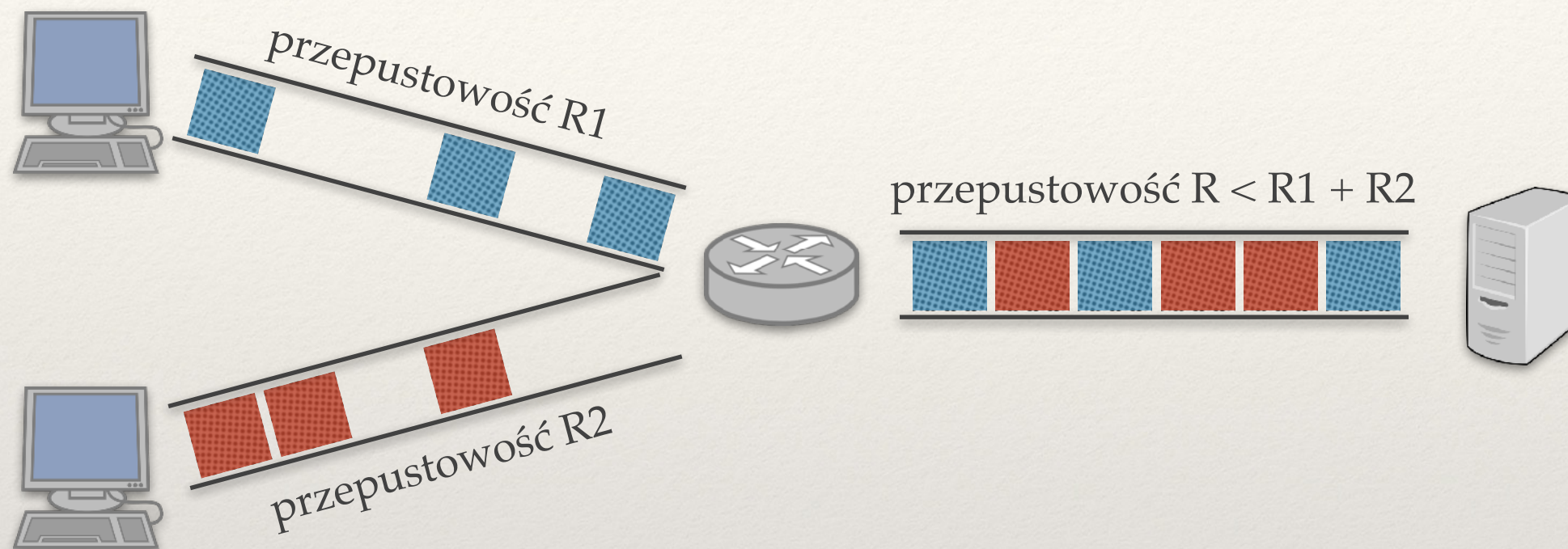
# Problem przeciążenia

---



# Statystyczny multipleksing

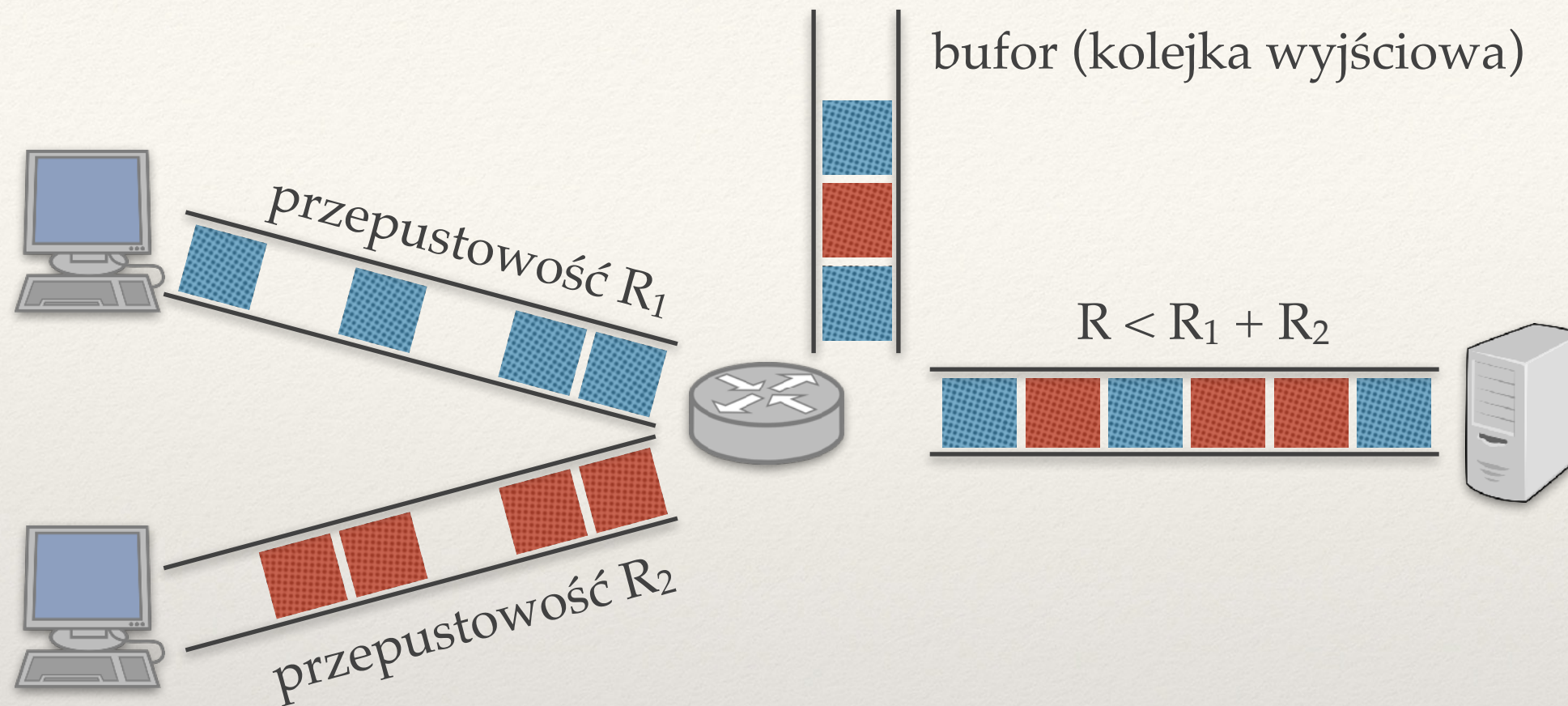
Różne strumienie danych przesyłane tym samym łączem.



Założenie: różne komputery wykorzystują łącze w innych momentach  
→ lepsze wykorzystanie łącza.



# Bufory



Bufory przy łączach wyjściowych:

- ❖ Pomagają przy przejściowym nadmiarze pakietów.
- ❖ Jeśli bufor się przepełni (**przeciążenie**) → pakiety są odrzucane.
- ❖ **Dlaczego nie zrobić większych buforów?**



# Zwiększyć bufor?

---

- ❖ Rozmiar bufora na trasie nie wpływa na przepustowość.
- ♦ Przepustowość na trasie wynika z najwolniejszego łącza.



# Zwiększyć bufor?

---

- ❖ Rozmiar bufora na trasie nie wpływa na przepustowość.
  - ♦ Przepustowość na trasie wynika z najwolniejszego łącza.
- ❖ Jeśli daną trasą będziemy cały czas przesyłać więcej niż jej przepustowość, to bufor będzie całkowicie wypełniony.
  - ♦ większe bufor → większe kolejki
  - ♦ większe kolejki → większe opóźnienie!



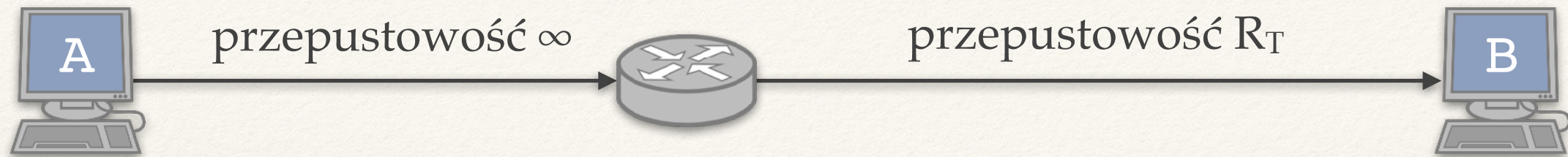
# Zwiększyć bufor?

---

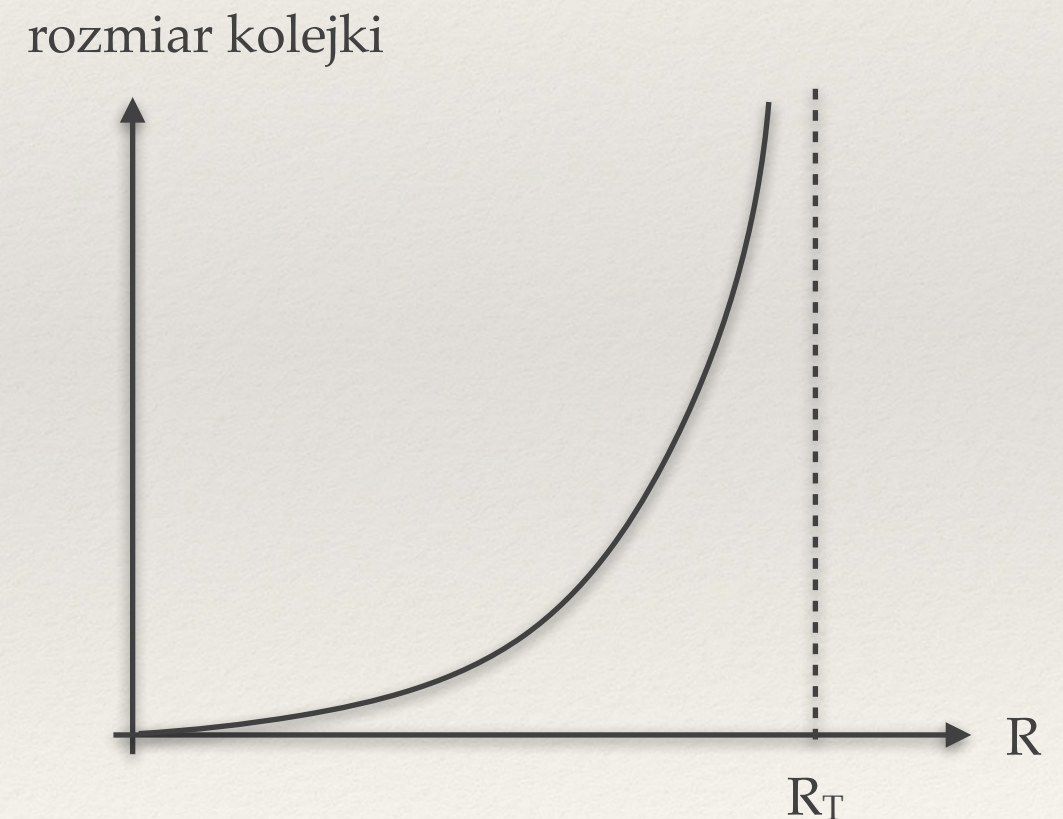
- ❖ Rozmiar bufora na trasie nie wpływa na przepustowość.
  - ♦ Przepustowość na trasie wynika z najwolniejszego łącza.
- ❖ Jeśli daną trasą będziemy cały czas przesyłać więcej niż jej przepustowość, to bufory będą całkowicie wypełnione.
  - ♦ większe bufory → większe kolejki
  - ♦ większe kolejki → większe opóźnienie!
- ❖ Przy dużych buforach pakiety są tak opóźniane, że TCP zaczyna wysyłać je ponownie → jeszcze bardziej zwiększa przeciążenie.



# Rozmiar kolejki

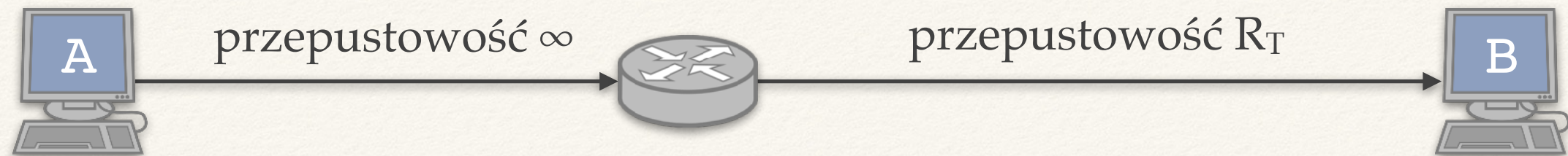


- ❖ Załóżmy, że pakiety wysyłane są od A do B losowo, ze **średnią** prędkością  $R$ .
- ❖ Matematyczna teoria kolejek → wykres rozmiaru kolejki.
- ❖ Opóźnienie jest liniową funkcją rozmiaru kolejki.

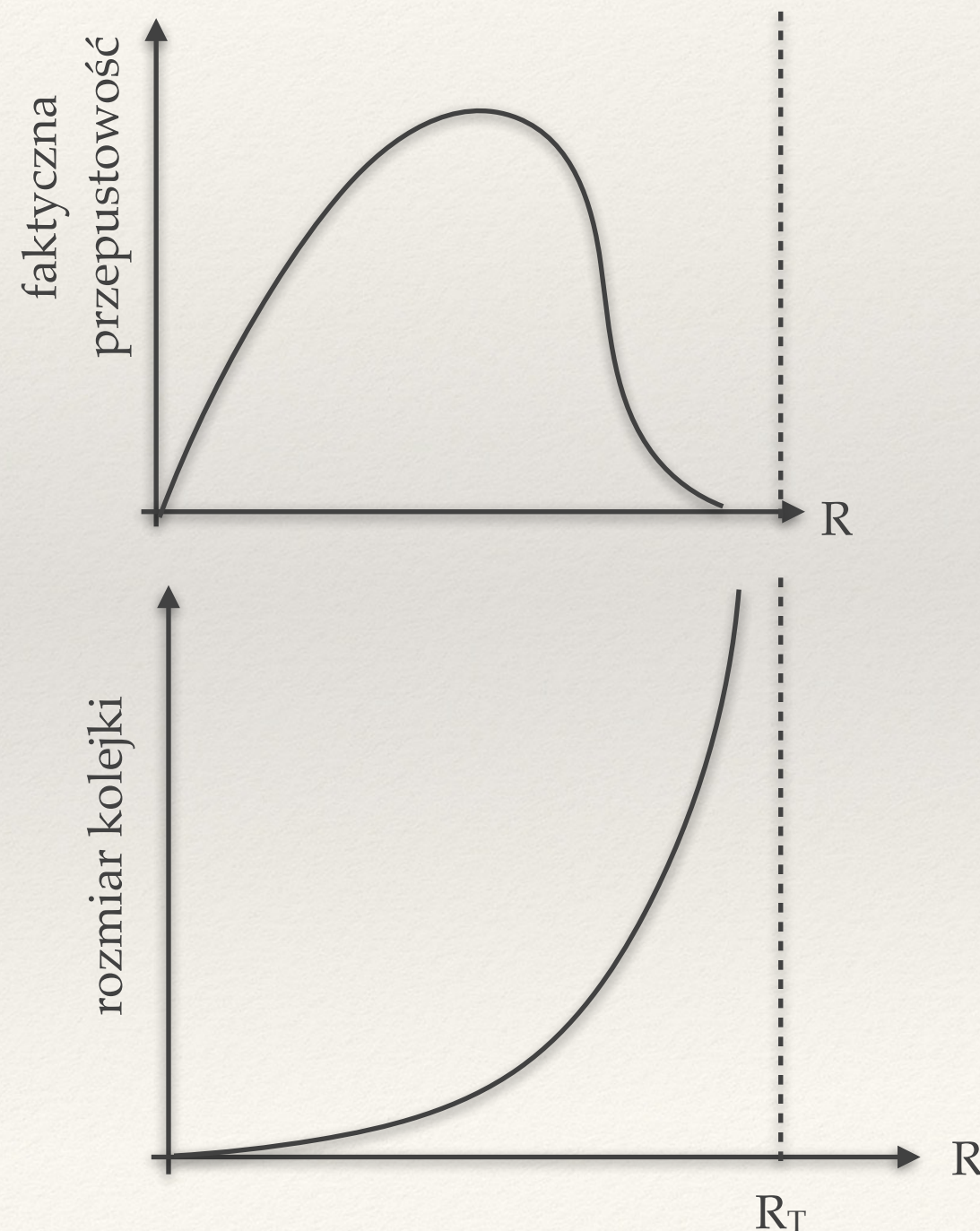




# Opóźnienie i faktyczna przepustowość (1)

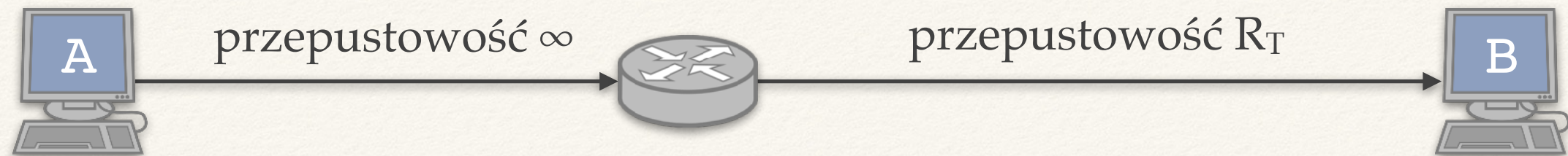


- ❖ Rosnące kolejki:
  - ♦ bardzo duże opóźnienia,
  - ♦ bardzo małą faktyczną przepustowość (dużo duplikatów).
- ❖ Potrzebujemy sposobu na spowolnienie strumienia danych.

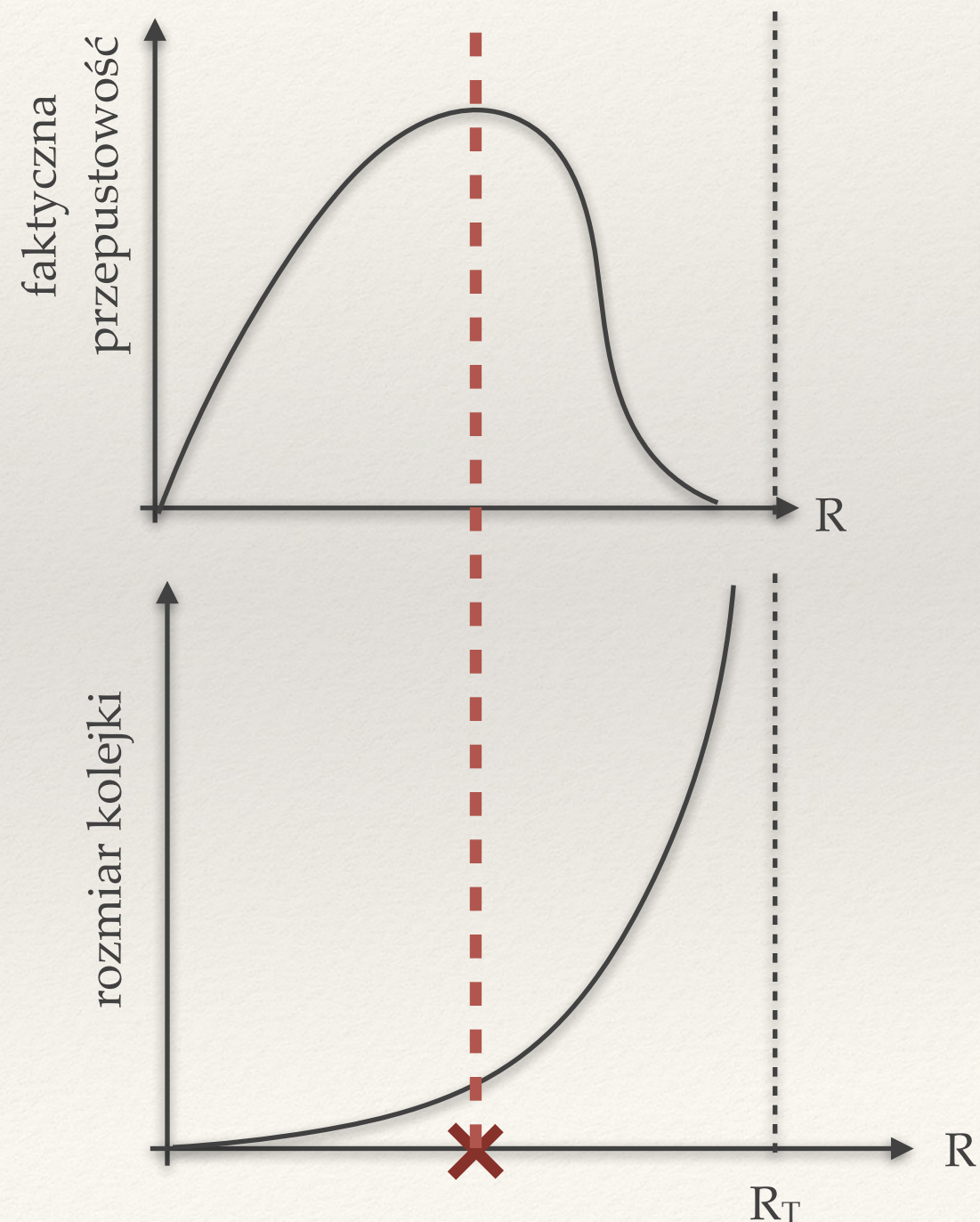




# Opóźnienie i faktyczna przepustowość (1)



- ❖ Chcemy mieć mechanizm, który będzie utrzymywać obciążenie w okolicach optymalnego punktu.





# Cele kontroli przeciążenia

---

- ❖ **Wysokie wykorzystanie łączy.**
  - ♦ Szybkie przesyłanie danych.
- ❖ **Sprawiedliwy podział łączy (*fairness*).**
  - ♦ Co to znaczy?
- ❖ **Dodatkowe cele.**
  - ♦ Rozproszony algorytm.
  - ♦ Szybko reaguje na zmieniające się warunki.



---

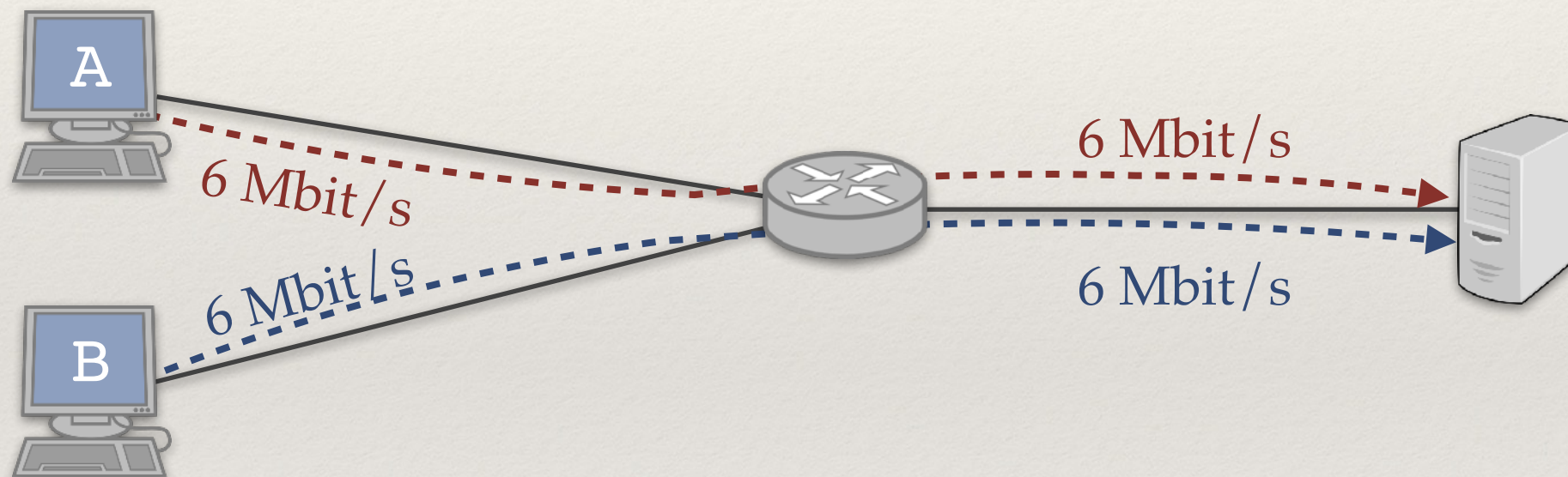
# Sprawiedliwy podział łączy

---



# Sprawiedliwy podział łącza: przykład 1

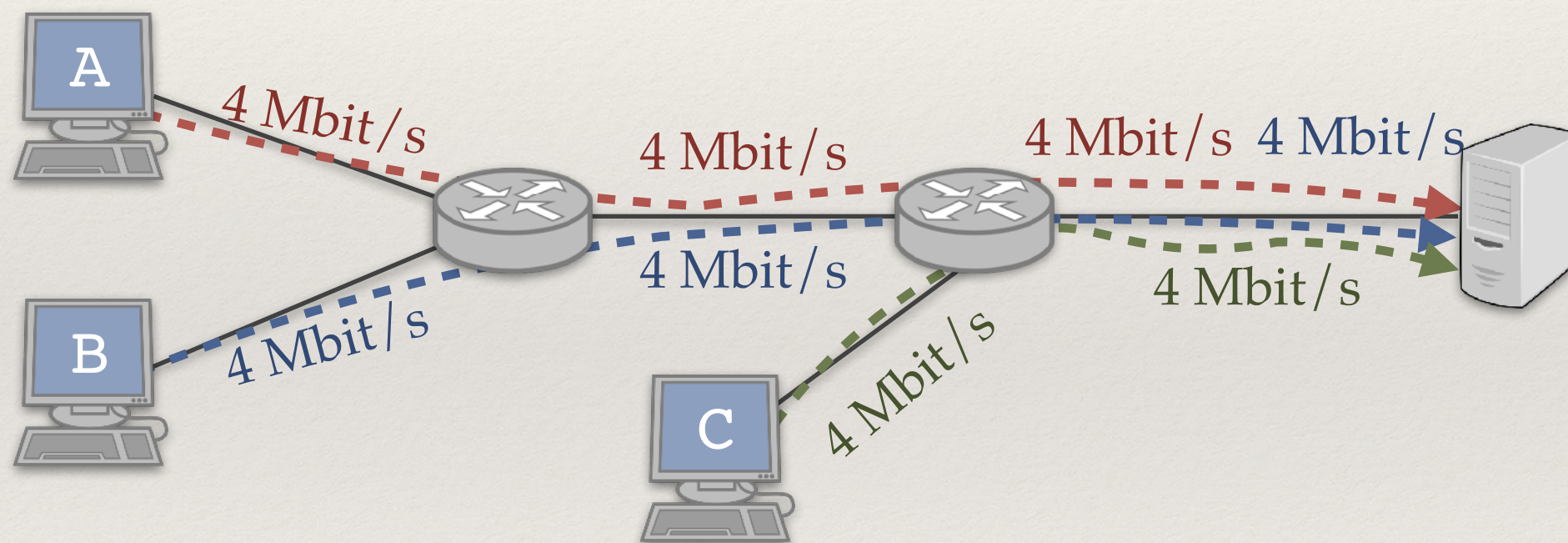
- ❖ Każde łącze ma przepustowość 12 Mbit/s.
- ❖ Każdy komputer chce wysłać do serwera jak najszybciej.





# Sprawiedliwy podział łącza: przykład 2

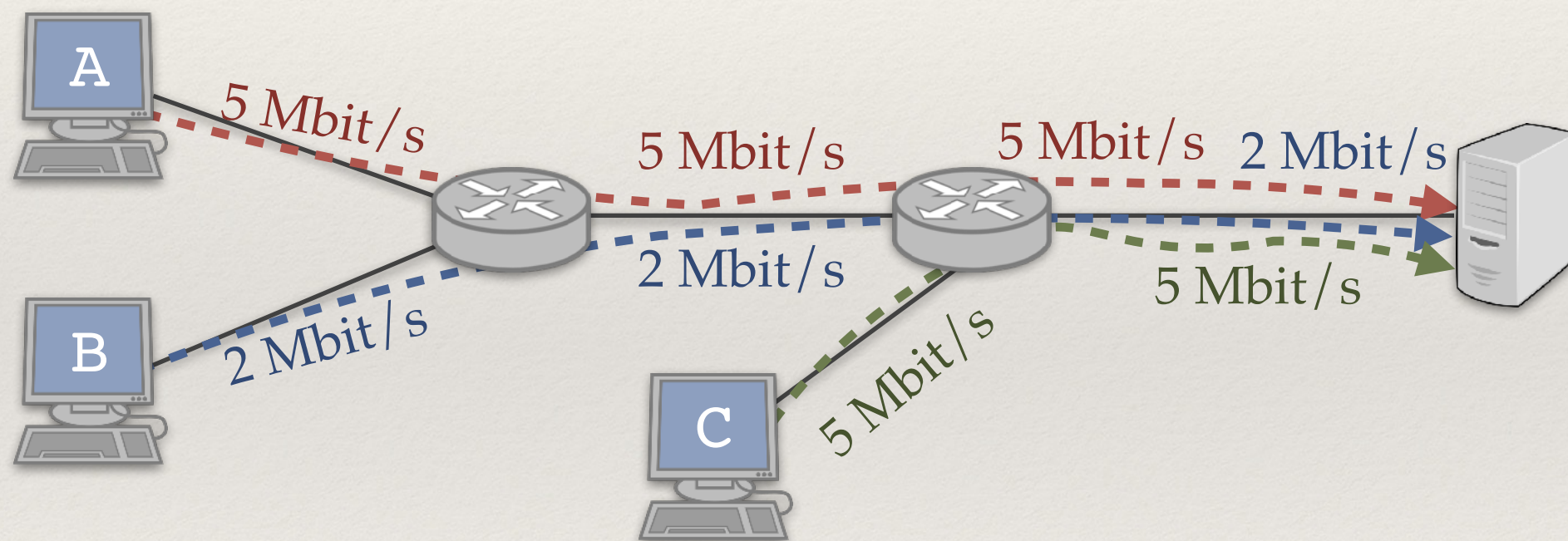
- ❖ Każde łącze ma przepustowość 12 Mbit/s.
- ❖ Każdy komputer chce wysłać do serwera jak najszybciej.





# Sprawiedliwy podział łącza: przykład 3

- ❖ Każde łącze ma przepustowość 12 Mbit/s, poza łączem między B a routerem, które ma przepustowość 2 Mbit/s.
- ❖ Każdy komputer chce wysyłać do serwera jak najszybciej.

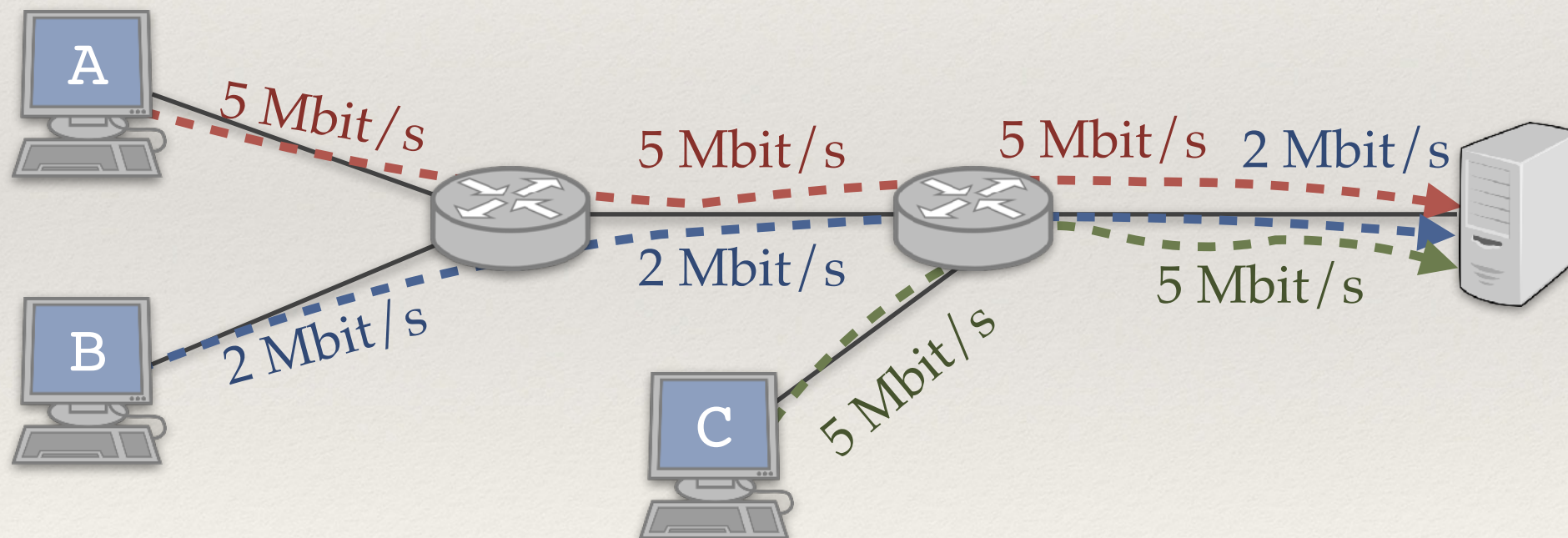


- ❖ Czy to przypisanie jest „sprawiedliwe”, czy też powinniśmy dać B proporcjonalnie mniej łącza?



# Max-Min fairness

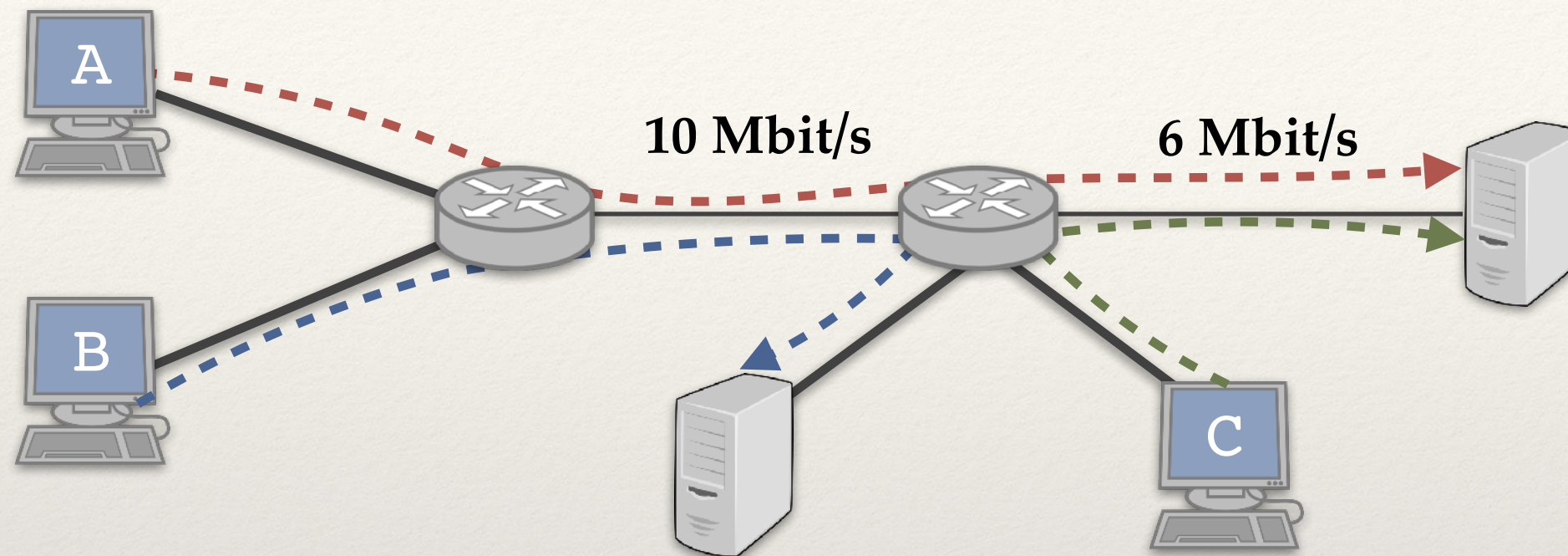
- ❖ „Maksymalizuj minimalną przepustowość”
- ❖ Równoważnie: przypisanie jest *max-min fair*, jeśli nie można zwiększyć szybkości żadnego ze strumieni bez spowolnienia innego strumienia, który jest wolniejszy lub tak samo szybki.





# Sprawiedliwy podział łącza vs. przepustowość

Nieoznaczone łącza mają nieskończoną przepustowość.



	„sprawiedliwe“	„niesprawiedliwe“
<b>A</b>	3 Mbit/s	1 Mbit/s
<b>B</b>	7 Mbit/s	9 Mbit/s
<b>C</b>	3 Mbit/s	5 Mbit/s
suma	13 Mbit/s	15 Mbit/s



---

AIMD

---



# Kontrola przeciążenia w warstwie transportowej

---

- ❖ Algorytm dla nadawcy.
- ❖ Wykorzystuje istniejące mechanizmy (okno przesuwne, modyfikuje rozmiar okna).
- ❖ Reaguje na obserwowane zdarzenia (utrata pakietów).
- ❖ Szacuje, ile wysłanych pakietów może bezproblemowo być „w trasie” do celu (wysłanych i jeszcze niepotwierdzonych).



# Kontrola przepływu vs. kontrola przeciążenia

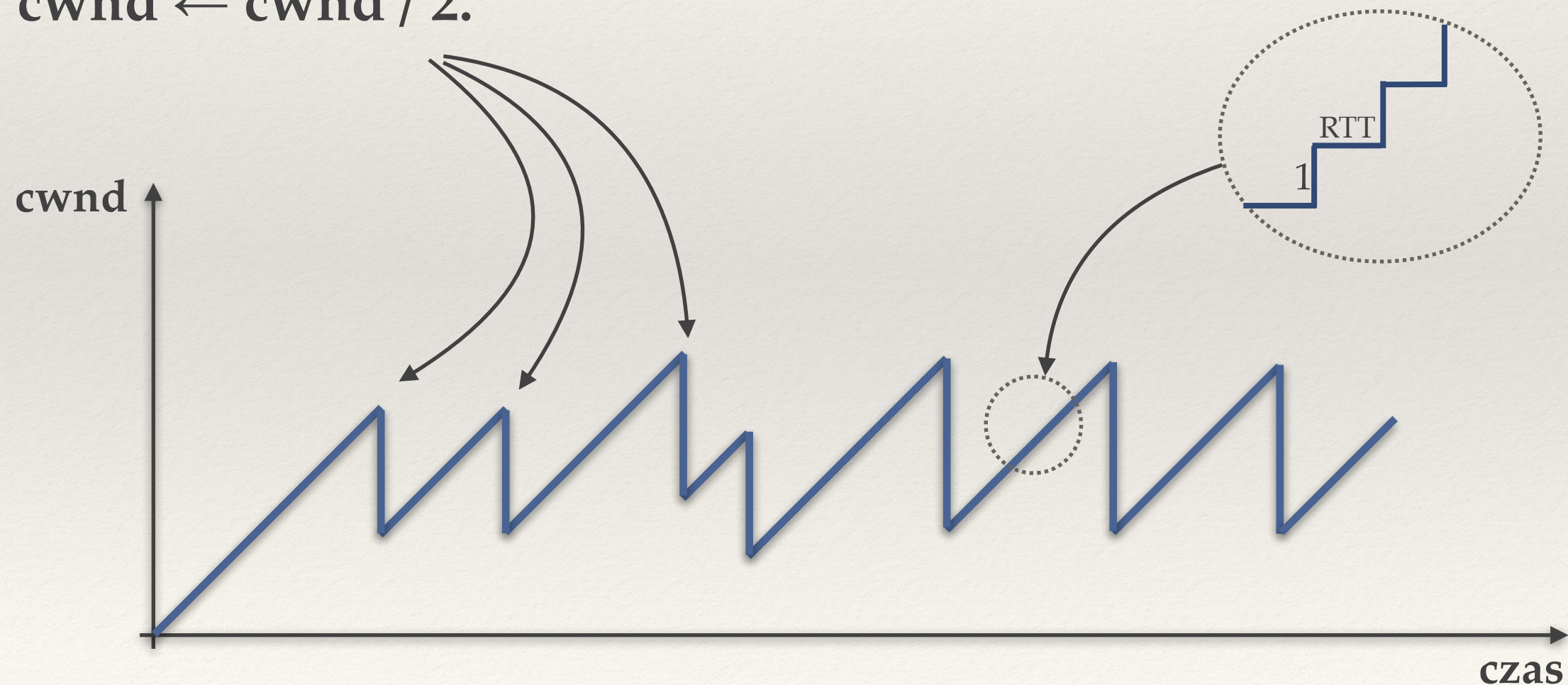
---

- ❖ Kontrola przepływu = nie chcemy zalać odbiorcy danymi.
  - ♦  $SWS = \text{oferowane okno}$ .
- ❖ Kontrola przeciążenia = nie chcemy zalać sieci danymi.
  - ♦ Parametr  $cwnd$  (*congestion window*) obliczany przez nadawcę.
  - ♦  $SWS = \min \{ \text{oferowane okno}, cwnd \}$ .
- ❖ Będziemy zakładać, że oferowane okno  $= \infty$ .



# AIMD (Additive Increase, Multiplicative Decrease)

- ❖ Pakiet wysłany poprawnie (otrzymaliśmy ACK):  
 $\text{cwnd} \leftarrow \text{cwnd} + 1 / \text{cwnd}.$   
(W ciągu RTT wysyłane cwnd segmentów, więc cwnd zwiększa się o 1).
- ❖ Pakiet zgubiony lub opóźniony (ACK nie dociera przed RTO)  
 $\text{cwnd} \leftarrow \text{cwnd} / 2.$





## Inny sposób patrzenia:

- ❖ AIMD nie kontroluje szybkości wysyłania.
- ❖ AIMD kontroluje liczbę pakietów (danego strumienia), która jednocześnie może być w sieci.

animacje



## Inny sposób patrzenia:

- ❖ AIMD nie kontroluje szybkości wysyłania.
- ❖ AIMD kontroluje liczbę pakietów (danego strumienia), która jednocześnie może być w sieci.

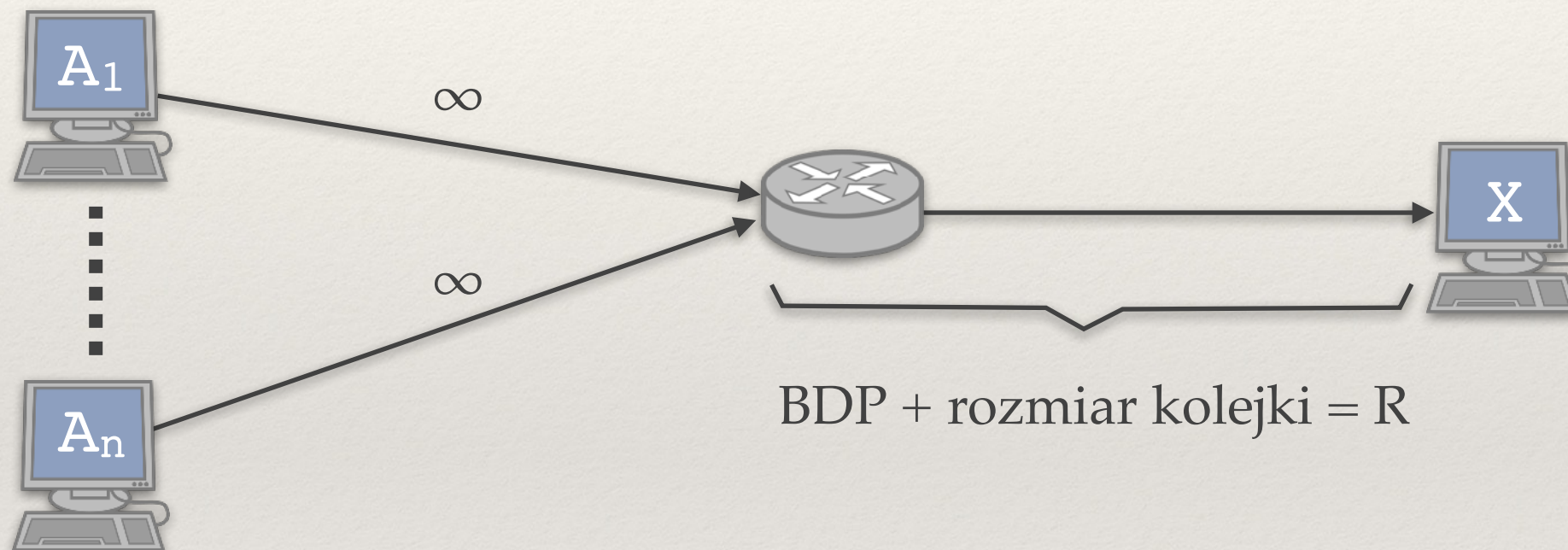
animacje

- ❖ Przy odpowiednio dużych buforach, najbardziej krytyczne łącze jest cały czas wykorzystane w 100%.



# AIMD vs. sprawiedliwy podział i efektywność

Własność AIMD:  $A_1 \dots A_n$  rozpoczynają transmisje do  $X$  w dowolnych momentach  $\rightarrow$  ich rozmiary okien zbiegną do  $R/n$ .

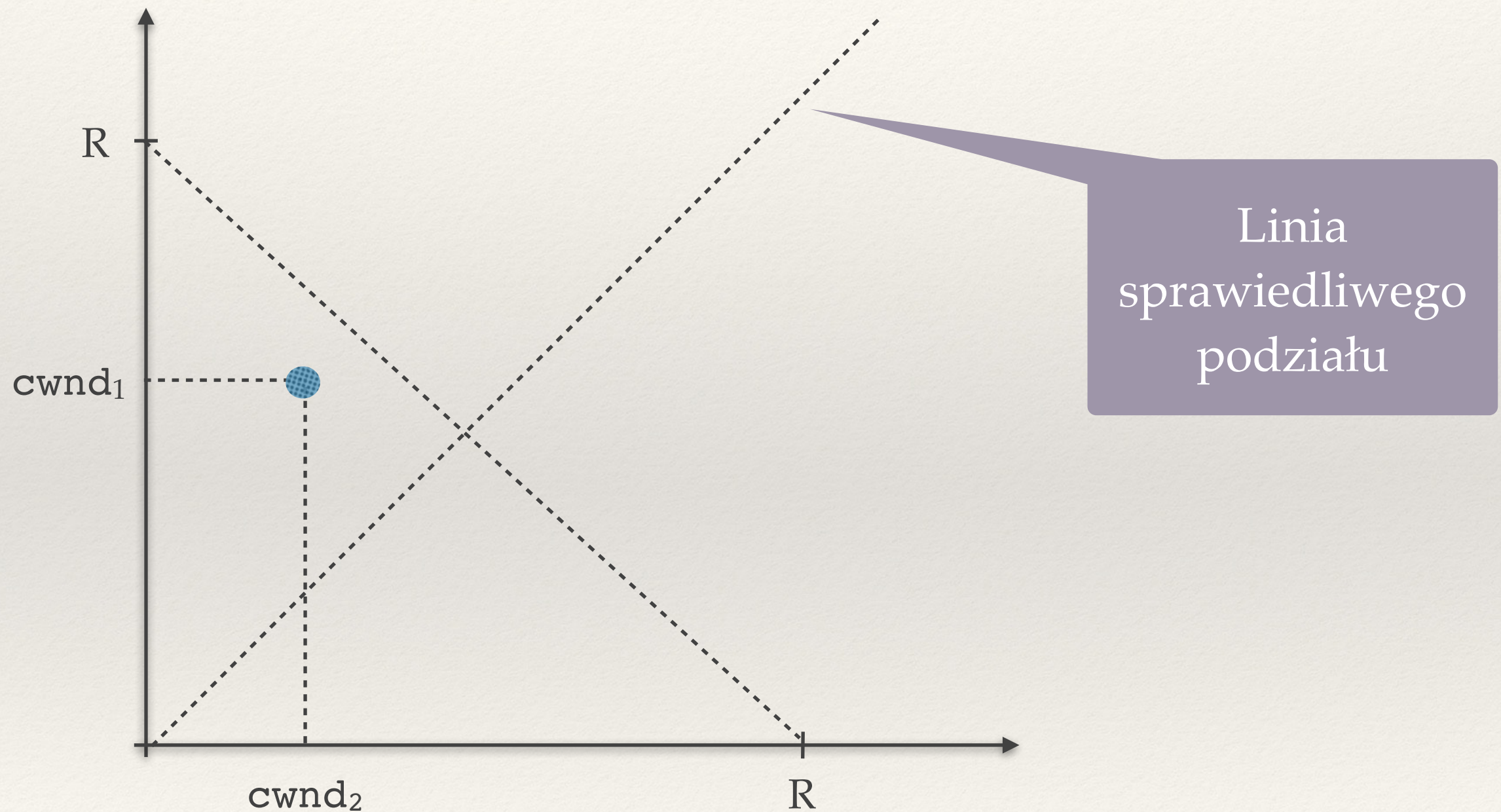


Pokażemy to dla  $n = 2$  na obrazku.



# Sprawiedliwość podziału i efektywność (1)

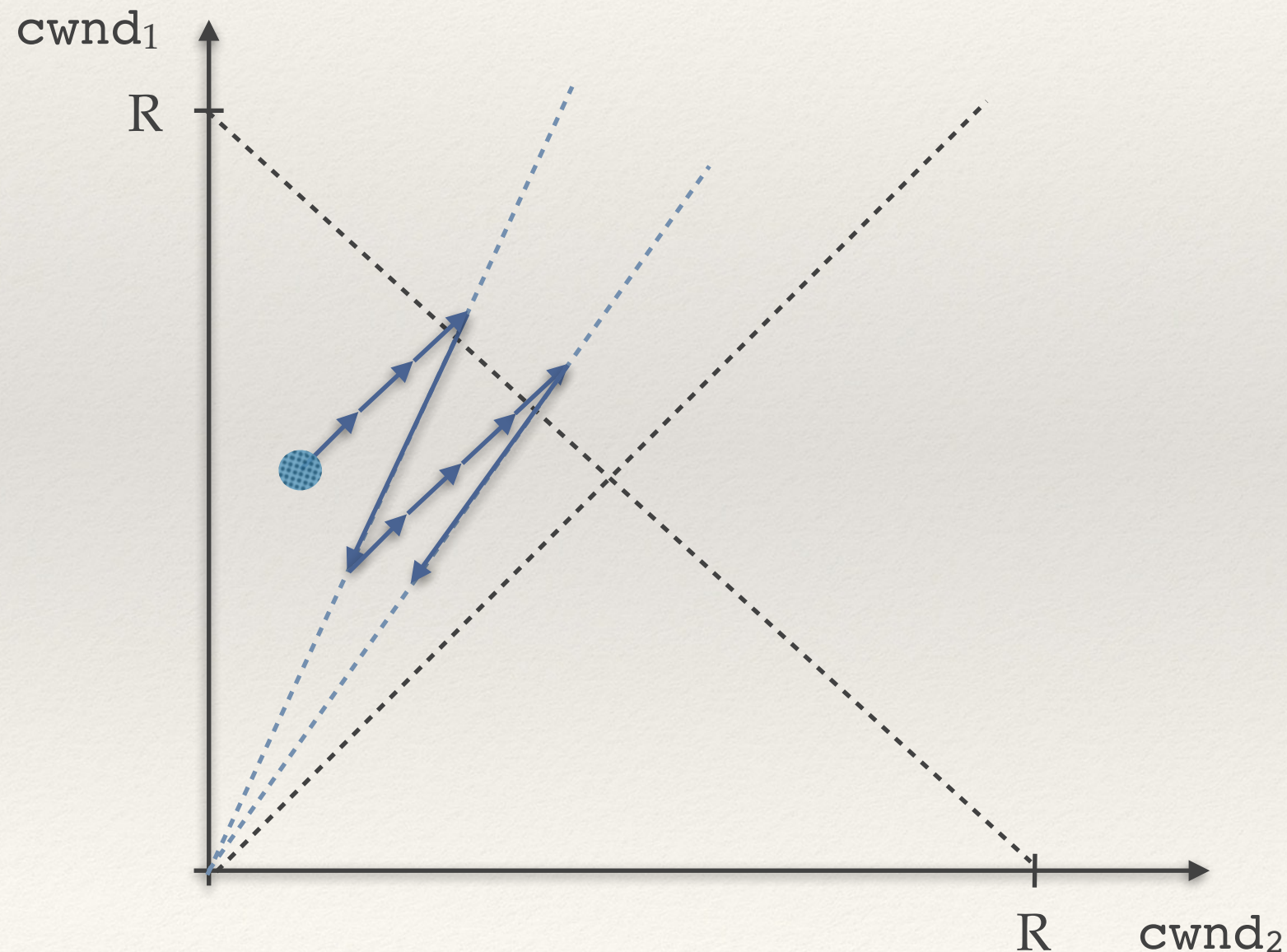
- ❖ Pakiety gubią się wtedy i tylko wtedy, jeśli  $cwnd_1 + cwnd_2 > R$





# Sprawiedliwość podziału i efektywność (2)

- ❖ Rozmiary  $cwnd_1$  i  $cwnd_2$  jednocześnie rosną o 1 lub maleją dwukrotnie
- ❖ Docelowo mamy sprawiedliwy podział łącza i  $cwnd_1 + cwnd_2$  oscylujące w przedziale  $[R/2, R]$ .





---

# Kontrola przeciążenia w TCP

---



# Kontrola przeciążenia w TCP

---

## ❖ AIMD w TCP:

- ♦ **ACK pakietu** (z potwierdzenia skumulowanego) → zwiększamy  $\text{cwnd}$  o  $\text{MSS} * \text{MSS} / \text{cwnd}$ .
  - Co RTT wysyłane jest  $\text{cwnd} / \text{MSS}$  segmentów: wszystkie będą potwierdzone → zwiększymy  $\text{cwnd}$  o  $\text{MSS}$ .
- ♦ **Pakiet zaginął** (przekroczony timeout albo otrzymaliśmy podwójne potwierdzenie) → zmniejszamy  $\text{cwnd}$  dwukrotnie.



# Kontrola przeciążenia w TCP

---

## ❖ AIMD w TCP:

- ♦ **ACK pakietu** (z potwierdzenia skumulowanego) → zwiększamy cwnd o  $MSS * MSS / cwnd$ .
  - Co RTT wysyłane jest cwnd / MSS segmentów: wszystkie będą potwierdzone → zwiększymy cwnd o MSS.
- ♦ **Pakiet zaginął** (przekroczony timeout albo otrzymaliśmy podwójne potwierdzenie) → zmniejszamy cwnd dwukrotnie.

Nie do końca prawda.

To jest tylko faza „unikania przeciążenia”.

Dodatkowo TCP wprowadza fazę „wolnego startu”



# Wolny start w TCP

---

## ❖ Faza wolnego startu:

- ♦ Zaczynamy od  $cwnd = MSS$ .
- ♦ Po każdym ACK zwiększamy  $cwnd$  o  $MSS$ .
  - $\rightarrow$  Co RTT  $cwnd$  zwiększa się dwukrotnie.
- ♦ Faza trwa do utraty pierwszego pakietu.

## ❖ Strata pakietu w dowolnej fazie:

- ♦  $ssthresh \leftarrow cwnd / 2$ .
- ♦ przechodzimy do fazy wolnego startu i zostajemy w niej aż  $cwnd > ssthresh$ .



# Wolny start w TCP

## ❖ Faza wolnego startu:

- ♦ Zaczynamy od  $cwnd = MSS$ .
- ♦ Po każdym ACK zwiększamy  $cwnd$  o  $MSS$ .
  - $\rightarrow$  Co RTT  $cwnd$  zwiększa się dwukrotnie.
- ♦ Faza trwa do utraty pierwszego pakietu.

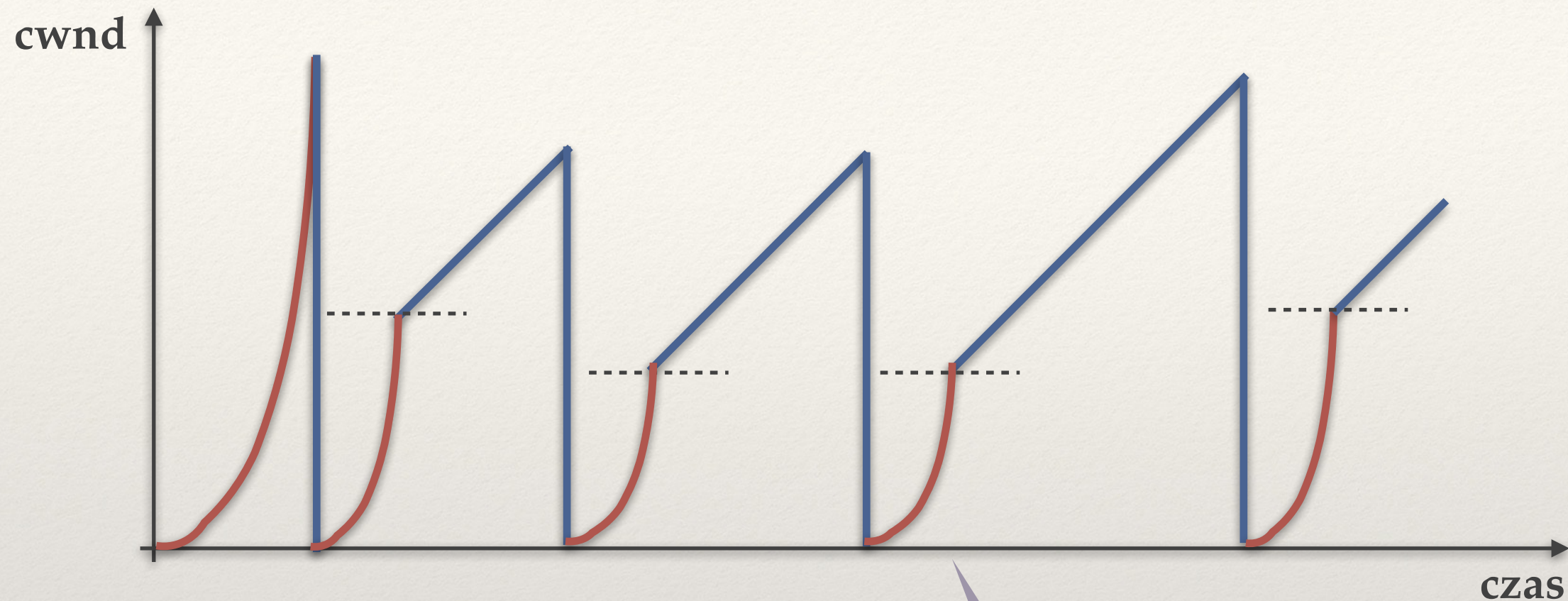
Wolny start = nie ustawiamy okna od razu na oferowane przez odbiorcę.

## ❖ Strata pakietu w dowolnej fazie:

- ♦  $ssthresh \leftarrow cwnd / 2$ .
- ♦ przechodzimy do fazy wolnego startu i zostajemy w niej aż  $cwnd > ssthresh$ .



# AIMD w TCP: przykład



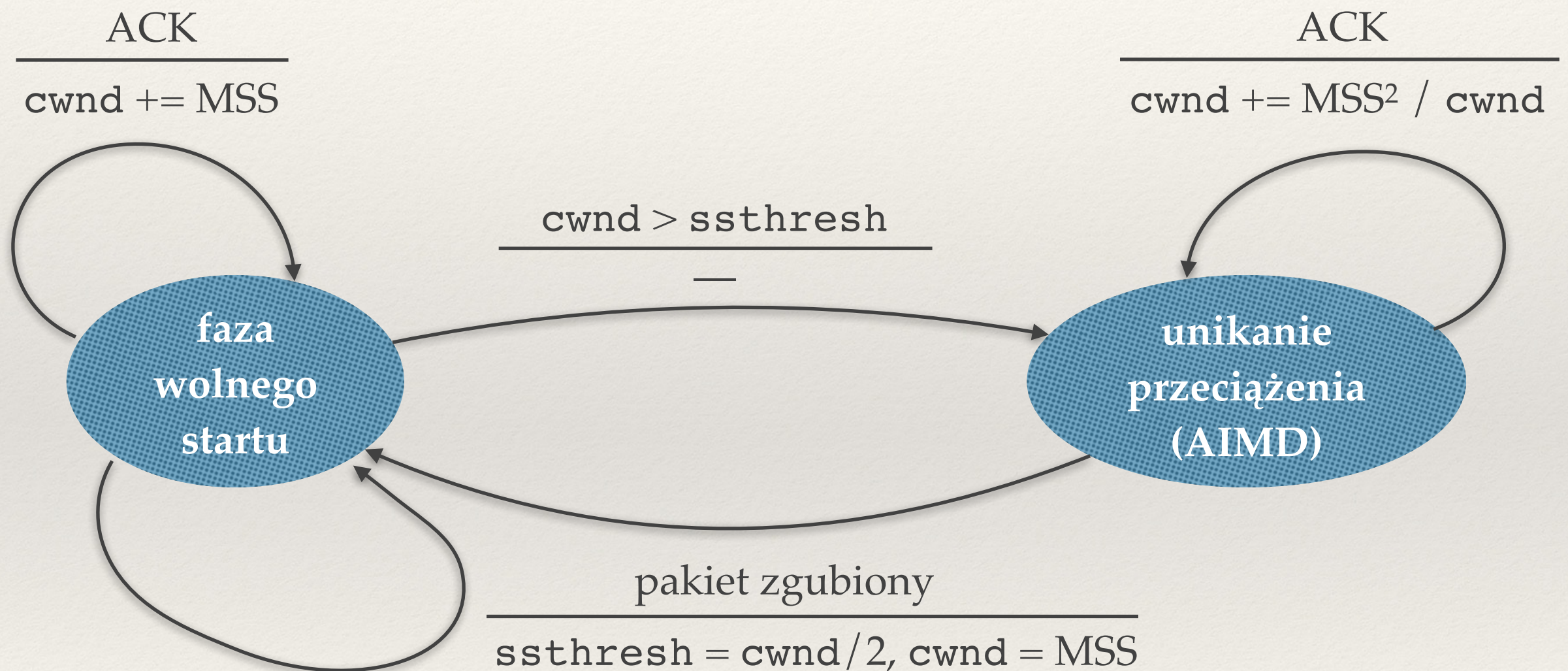
- faza wolnego startu
- faza unikania przeciążenia
- - - ssthresh

Fazy wolnego startu mają logarytmiczną długość, poza nimi TCP wykonuje AIMD.



# Fazy w TCP: diagram przejścia

Inicjalizacja: faza wolnego startu,  $ssthresh = \infty$ ,  $cwnd = MSS$ .





# Źródła informacji o utracie pakietu

---

## ❖ Timeout dla pakietu

## ❖ Wielokrotny ACK

- ♦ Przykładowo: odbiorca dostaje segmenty 1, 2, 3, 5, 6  
→ wysyła ACK 1, ACK 2, ACK 3, ACK 3, ACK 3.
- ♦ Statystycznie mniejsza szansa, że mamy do czynienia z dłuższym przeciążeniem (kolejne pakiety doszły do odbiorcy).
- ♦ **Szybka retransmisja** (wysyłamy brakujący segment bez czekania na timeout).
- ♦ **Szybkie przywracanie** (pomijamy fazę krótkiego startu):  
 $ssthresh = cwnd / 2; \quad cwnd = ssthresh.$



---

# Wspomaganie przez routery

---



## RED (Random Early Detection)

- ❖ Router na trasie wyrzuca losowe pakiety.
- ❖ Prawdopodobieństwo wyrzucenia ustalane jako rosnąca funkcja **średniej** długości kolejki.
  - ♦ Nie reaguje na krótkotrwałe zwiększenia kolejki.
- ❖ Krótsze kolejki → mniejsze opóźnienia.
- ❖ Desynchronizacja strumieni (zmniejszają prędkości w różnych momentach).



# ECN (Explicit Congestion Notification)

- ❖ Bliskie przeciążenie  
→ router ustawia bity ECN w nagłówku IP.
- ❖ Odbiorca ustawia bity ECN w nagłówku TCP ACK.
- ❖ Nadawca reaguje tak, jak na utratę pakietu.

## nagłówek IP:

wersja	IHL	typ usługi	całkowita długość pakietu
pola związane z fragmentacją pakietu			
TTL	protokół		suma kontrolna nagłówka IP
źródłowy adres IP			
docelowy adres IP			

## nagłówek TCP:

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



---

# Problemy z kontrolą przeciążenia

---



# Problemy

---

- ❖ Krótkie połączenia.
- ❖ Przepustowość proporcjonalna do  $1/\sqrt{p}$ , gdzie  $p$  jest frakcją traconych pakietów.
- ❖ Podatność na oszukiwanie.



# Krótkie połączenia

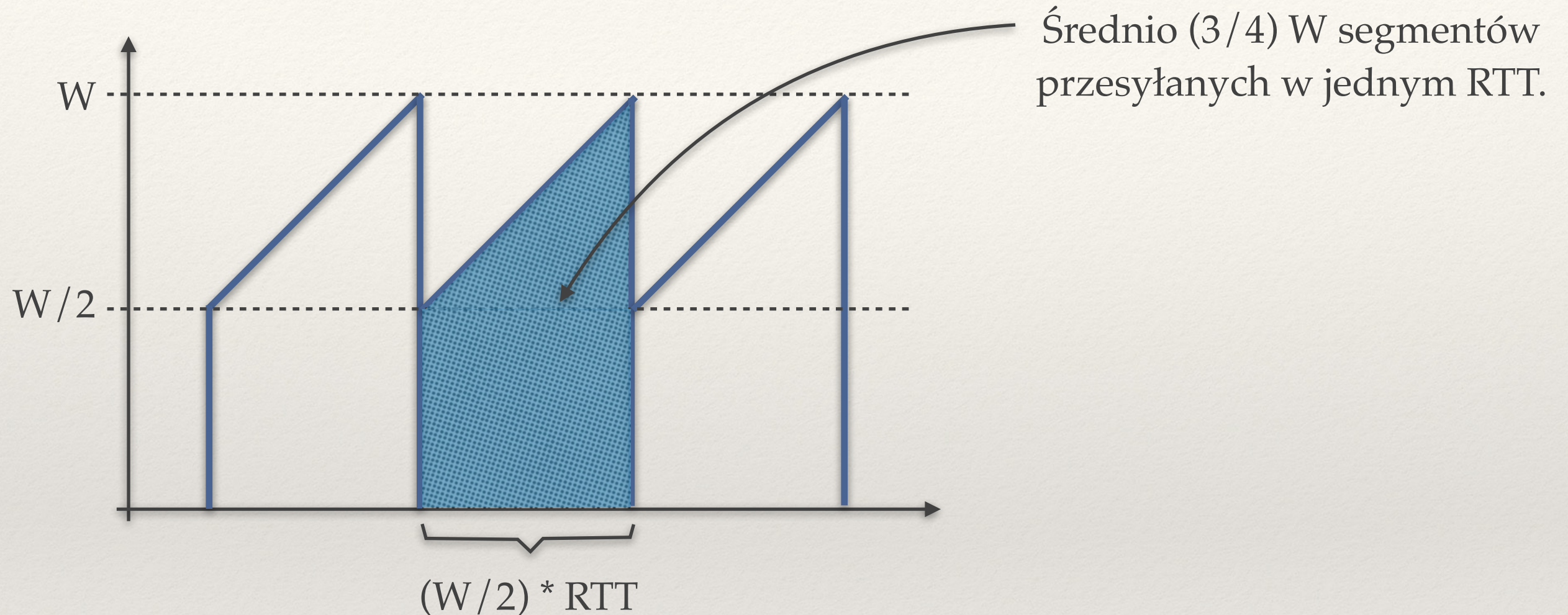
---

- ❖ Większość połączeń przesyła poniżej 100 KB danych.
- ❖ Nie wychodzą poza fazę wolnego startu.
- ❖ Remedium: trwałe połączenia, HTTP / 2, HTTP / 3.



# Przepustowość vs. straty segmentów (1)

- ❖ Rozważmy stabilny stan i pomińmy fazy wolnego startu.

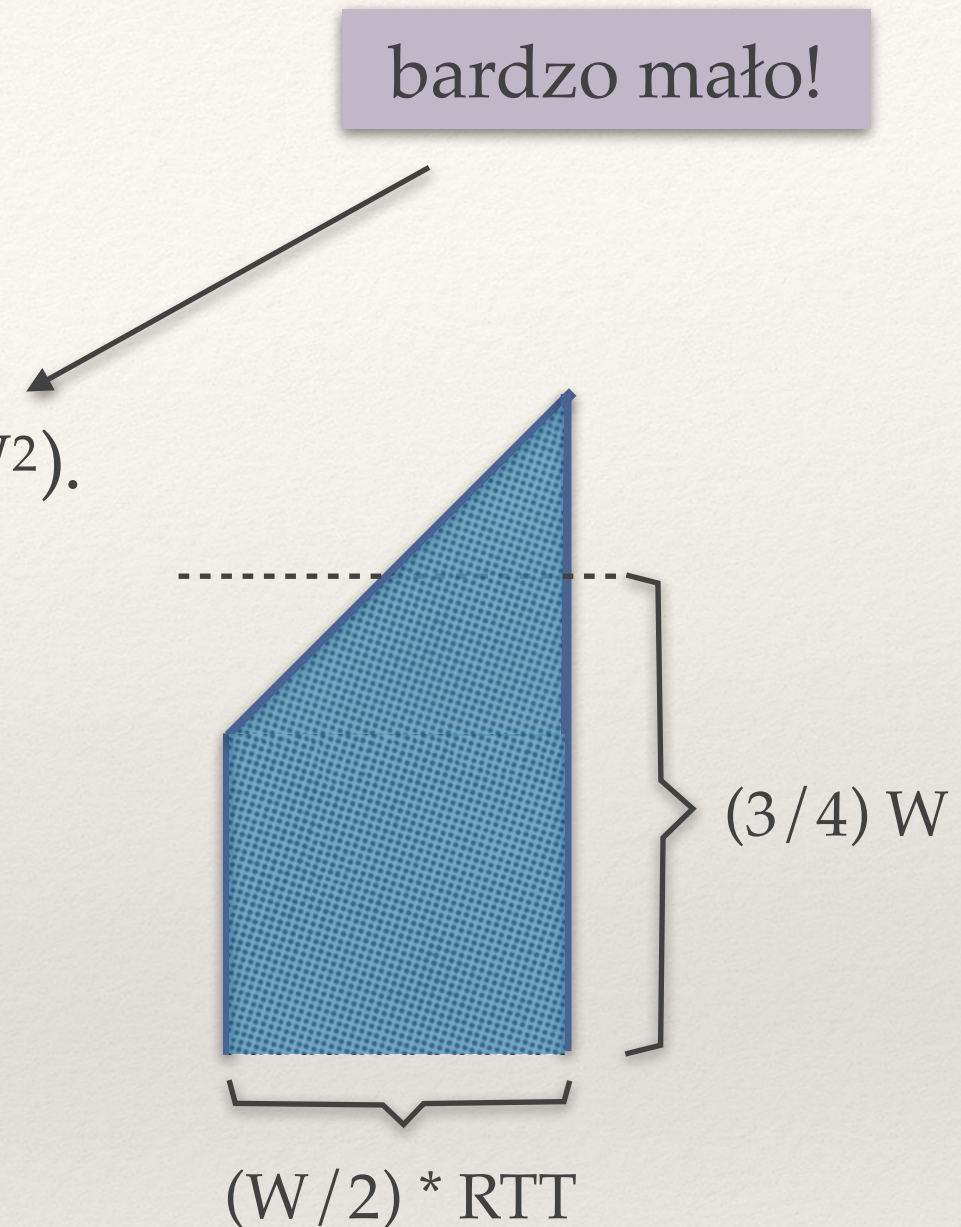


- ❖ W jednej fazie: przesyłanych  $(3/8) W^2$  segmentów, jeden gubiony.
- ❖ Średnia prędkość przesyłania:  $(3/4) W / RTT$ .



# Przepustowość vs. straty segmentów (2)

- ❖ Frakcja traconych segmentów:  $p = 8/3 (1/W^2)$ .





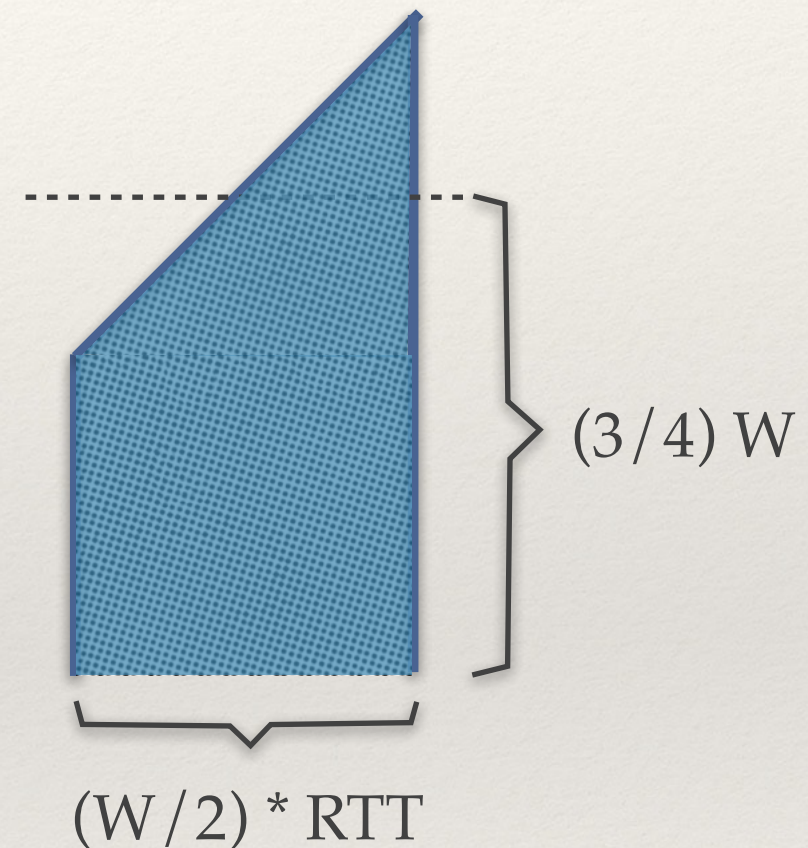
# Przepustowość vs. straty segmentów (2)

❖ Frakcja traconych segmentów:  $p = 8/3 (1/W^2)$ .

❖ Średnia prędkość przesyłania (segment / sek.)

$$T = \frac{3}{4} \cdot \frac{W}{\text{RTT}} = \sqrt{\frac{3}{2}} \cdot \frac{1}{\text{RTT} \cdot \sqrt{p}}$$

bardzo mało!





# Łączy o dużym BDP

---

- ❖ Prędkość przesyłania:  $T = \sqrt{\frac{3}{2}} \cdot \frac{1}{\text{RTT} \cdot \sqrt{p}}$  segmentów / sekundę.
- ❖ Transmisja RTT = 100 ms, MSS = 1500 bajtów, łącze 10 Gbit/s.
- ❖ Żeby osiągnąć  $T = 10 \text{ Gbit/s}$ ,  $p \approx 2 \cdot 10^{-10}$ :
  - ✦ Co najwyżej jeden tracony segment na  $2 \cdot 10^{10}$  segmentów.
  - ✦ Nie do zrealizowania w praktyce.
- ❖ Współczesne implementacje (np. CUBIC TCP) modyfikują AIMD, bardziej agresywnie zwiększając cwnd.



# TCP w WiFi

---

- ❖ Przepustowość proporcjonalna do  $1/\sqrt{p}$ , gdzie  $p$  jest frakcją traconych pakietów.
- ❖ Pakiety tracone niekoniecznie przez przeciążenie, np. w sieciach bezprzewodowych tracone przez interferencje.
- ❖ Żeby TCP działało sensownie konieczne są retransmisje dokonywane przez warstwy niższe (warstwę łącza danych).



# Transmisje o mniejszym RTT są preferowane

---

- ❖ Prędkość przesyłania:  $T = (3/4) W / \text{RTT}$  (segmentów / sek).
- ❖ Dwie transmisje korzystającego z tego samego łącza o małej przepustowości: ich okna przeciążenia ( $W$ ) są takie same.
- ❖ Prędkości transmisji proporcjonalne do  $1 / \text{RTT}$ : transmisje o mniejszym RTT przesyłane szybciej.



# Nadużycia

---

- ❖ Zaczynanie wolnego startu z  $cwnd > 1 \text{ MSS}$ .
- ❖ Szybsze zwiększanie  $cwnd \rightarrow$  niesprawiedliwy podział łącza.
- ❖ Otwieranie wielu połączeń (np. aplikacje P2P), bo każde połączenie ma takie samo  $cwnd$ .



# Lektura dodatkowa

---

- ❖ Kurose & Ross: rozdział 3.
- ❖ Tanenbaum: rozdział 6.
- ❖ TCP Congestion Control RFC: <https://tools.ietf.org/html/rfc5681>
- ❖ Bufferbloat:  
[`https://en.wikipedia.org/wiki/Bufferbloat`](https://en.wikipedia.org/wiki/Bufferbloat)
- ❖ [`https://en.wikipedia.org/wiki/TCP\_congestion\_control`](https://en.wikipedia.org/wiki/TCP_congestion_control)



# Zagadnienia

---

- ❖ Czym różni się kontrola przepływu od kontroli przeciążenia?
- ❖ Co to jest przeciążenie?
- ❖ Na czym polega mechanizm opóźnionych potwierdzeń?
- ❖ Jaka jest zależność między rozmiarem okna nadawcy a prędkością transmisji?
- ❖ Czy nieskończone bufory rozwiązałyby problem przeciążenia?
- ❖ Jak zależy średni rozmiar kolejki od średniej prędkości nadchodzenia pakietów?
- ❖ Jakie są cele kontroli przeciążenia?
- ❖ Jak można definiować sprawiedliwy podział łącza? Co to jest *max-min fairness*?
- ❖ Na jakiej podstawie zmienia się rozmiar okna przeciążenia?
- ❖ Kiedy TCP wnioskuje, że pakiet zaginął?
- ❖ Opisz algorytm ustalania rozmiaru okna przeciążenia
- ❖ Rozwiń skrót AIMD. Czego dotyczy?
- ❖ W jaki sposób AIMD gwarantuje sprawiedliwy podział łącza?
- ❖ Opisz fazy unikania przeciążenia i wolnego startu w TCP.
- ❖ Opisz mechanizm szybkiej retransmisji i szybkiego przywracania.
- ❖ Na czym polega mechanizm RED?
- ❖ Opisz działanie mechanizmu ECN (*explicit congestion notification*).
- ❖ Jaka jest relacja w AIMD między przepustowością a traconymi pakietami?
- ❖ Jakie modyfikacje wprowadza FastTCP do AIMD? Dlaczego?