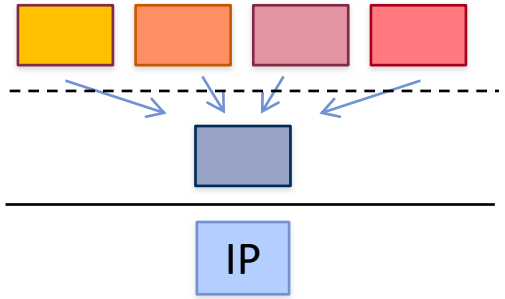
Rodina protokolů TCP/IP
verze 3

Téma 9: Transportní protokoly

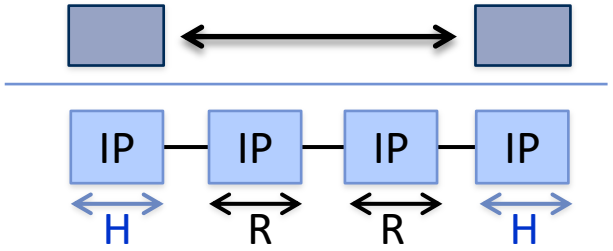
Jiří Peterka

úkoly transportní vrstvy (obecně)

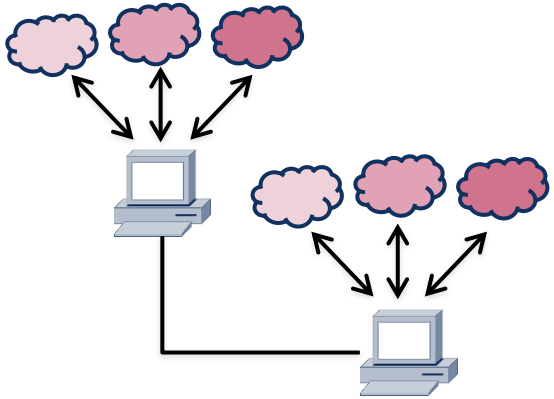
- přizpůsobuje požadavky vyšších vrstev možnostem nižších vrstev
  - mohou se týkat:
    - spojovaného/nespojovaného způsobu přenosu,
    - spolehlivosti/nespolehlivosti přenosu,
    - podpory QoS,
    - ……



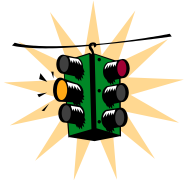
- zajišťuje end-to-end komunikaci
  - vzájemnou komunikaci koncových uzlů
    - není implementováno v mezilehlých uzlech
      - ve směrovačích



- rozlišuje různé příjemce a odesilatele v rámci jednotlivých uzlů
  - provádí multiplexing a demultiplexing
    - pomocí přechodových bodů SAP nebo pomocí portů



- může řešit i další úkoly
  - jako je řízení toku, předcházení zahlcení, …….

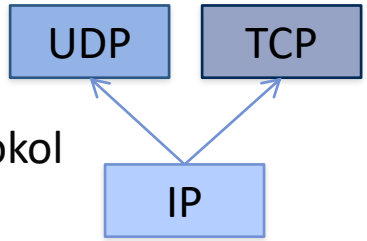


transportní vrstva TCP/IP

- „staví“ na jednotné síťové vrstvě
  - protokol IP: nespojovaný, nespolehlivý, best effort (žádná podpora QoS)
- vyšším vrstvám nabízí 2 varianty „přizpůsobení“
  - 2 varianty transportních služeb

a) minimální změna

- **transportní protokol UDP**
  - nespojovaný a nespolehlivý
    - stejně jako protokol IP
  - je to velmi jednoduchý protokol
    - stejně jako protokol IP
  - funguje stylem best effort, bez QoS
    - stejně jako protokol IP
  - nezajišťuje řízení toku ani nepředchází zahlcení
    - stejně jako protokol IP
  - přenáší data po blocích (datagramech)
    - stejně jako protokol IP



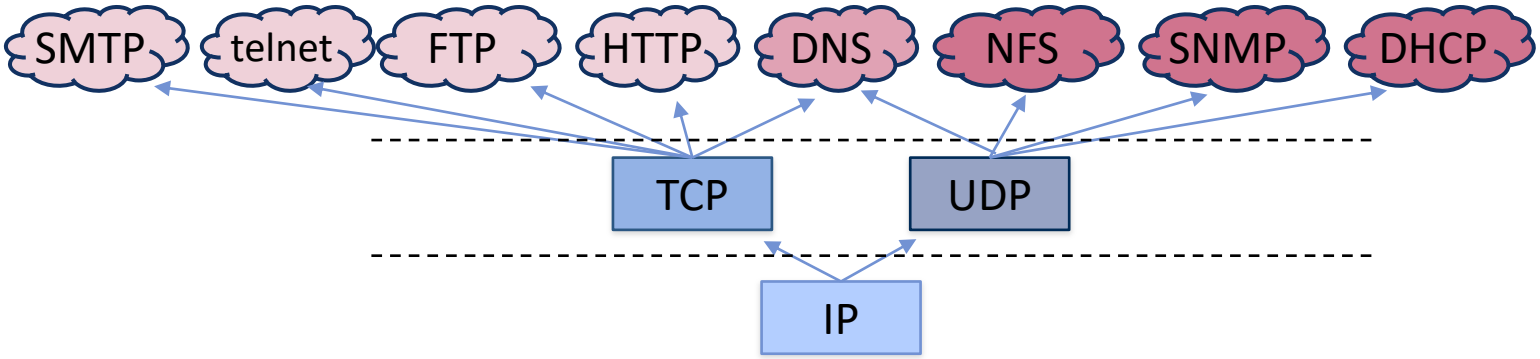
b) změnit všechno

- **transportní protokol TCP**
  - spojovaný a spolehlivý
    - na rozdíl od protokolu IP
  - velmi složitý a komplexní protokol
    - na rozdíl od protokolu IP
  - funguje stylem best effort, bez QoS
    - stejně jako protokol IP
  - zajišťuje řízení toku a předchází zahlcení
    - na rozdíl od protokolu IP
  - přenáší data jako proud bytů
    - na rozdíl od protokolu IP

- zajišťuje multiplex a demultiplex (rozlišování odesilatelů a příjemců)

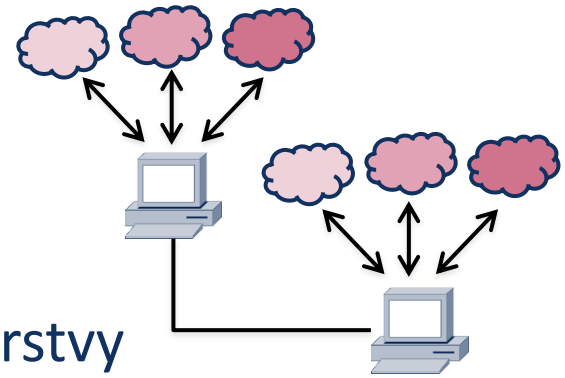
aplikace a transportní vrstva

- některé aplikace preferují spolehlivý přenos
  - bylo by ale neefektivní, aby si jej zajišťovala každá aplikace sama a znovu
    - vhodnější je implementovat spolehlivost společně (v transportním protokolu)
  - typicky:
    - aplikace, které přenáší soubory nebo „větší data“ během krátké doby
      - například: el. pošta (protokol SMTP), přenos souborů (FTP), web (HTTP), DNS (pro zónové transfery), telnet …
      - používají transportní protokol TCP
- jiné preferují nespolehlivý přenos
  - raději oželí část svých dat
    - než aby připustily nepravidelnost v doručování
  - typicky:
    - aplikace, které přenáší multimediální data, nebo „malá data“ více rozložená v čase
      - například: sdílení souborů (NFS), správa sítě (SNMP), konfigurace (BOOTP, DHCP), směrování (RIP), DNS (pro dotazy) ….
      - používají transportní protokol UDP

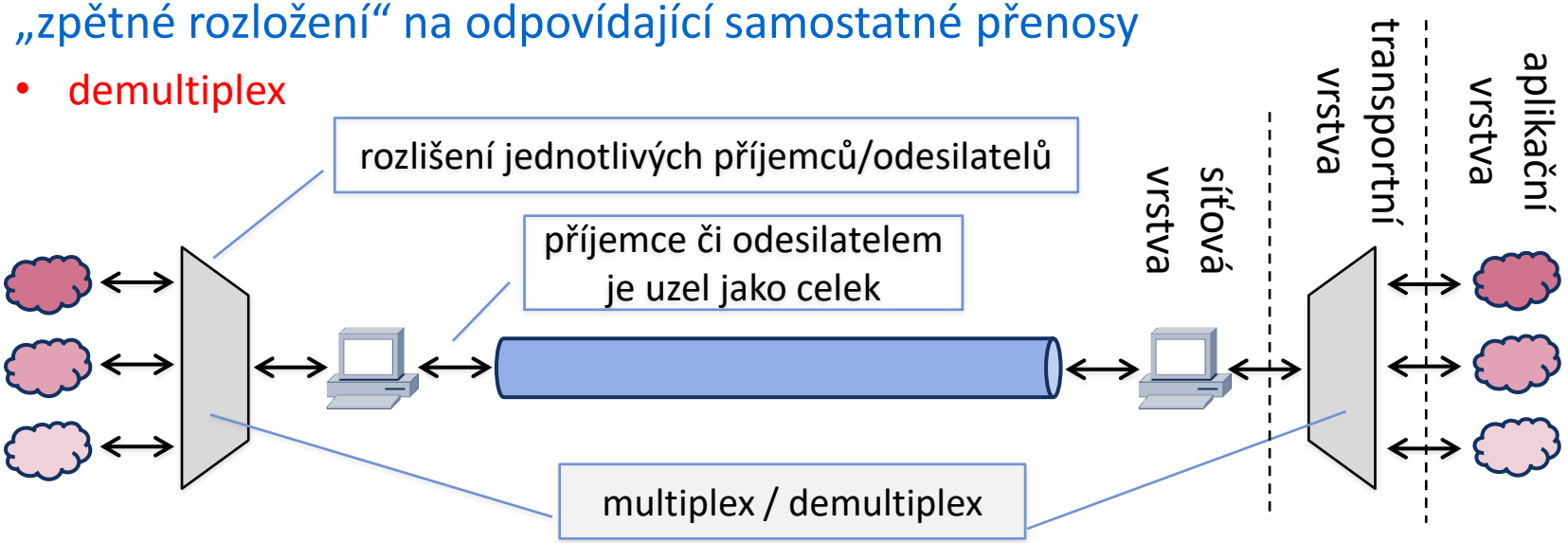


multiplex a demultiplex

- připomenutí:
  - na síťové vrstvě (i na vrstvě síťového rozhraní) se adresují uzly jako celky
    - příjemcem či odesilatelem je uzel jako celek
  - na aplikační vrstvě existuje více různých entit, které mohou vystupovat v roli odesilatelů či příjemců dat
    - a je třeba je rozlišit



- rozlišení je nutné provést na úrovni transportní vrstvy
  - „sloučení“ několika samostatných přenosů do jedné společné přenosové cesty
    - multiplex
  - „zpětné rozložení“ na odpovídající samostatné přenosy
    - demultiplex

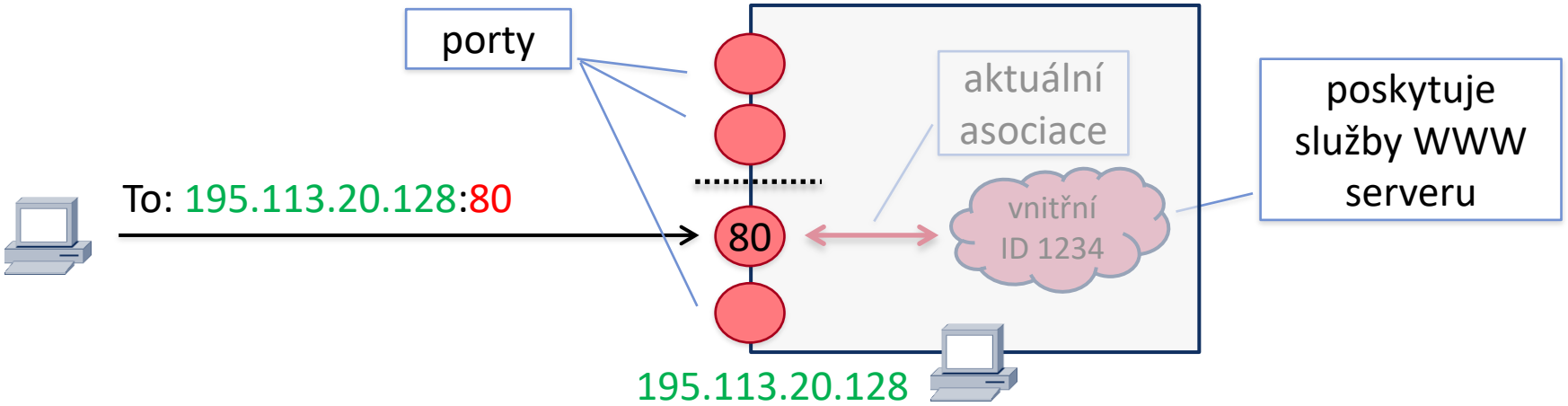


adresování na transportní vrstvě

- adresy na transportní vrstvě mohou být relativní
  - protože jsou vždy vztažené k danému uzlu
    - konkrétní entitu identifikuje dvojice **<IP adresa> : <transportní adresa>**
  - transportní adresy musí být „viditelné zvenku“
    - protože se s nimi musí pracovat i mimo daný uzel
      - odesilatel potřebuje poslat svá data konkrétní entitě (např. procesu) na cílovém uzlu
- jaký druh adres volit pro transportní vrstvu?
  - musí být všude stejné (nesmí záviset na konkrétní platformě daného uzlu)
    - nemohou to být žádné „implementačně závislé“ adresy / identifikátory
      - které by existovaly jen na některých systémových platformách, a na jiných nikoli
        - například systémové identifikátory procesů
    - musí být statické
      - nesmí se měnit v čase a být závislé na okolnostech, které „zvenku nejsou vidět“
        - například na tom, v jakém pořadí „nabíhají“ jednotlivé procesy a jaké mají přiděleny místní identifikátory
- řešení: transportní adresy budou abstraktní
  - přiřazení konkrétních entit k abstraktním adresám „nemusí být vidět“ z vně uzlu

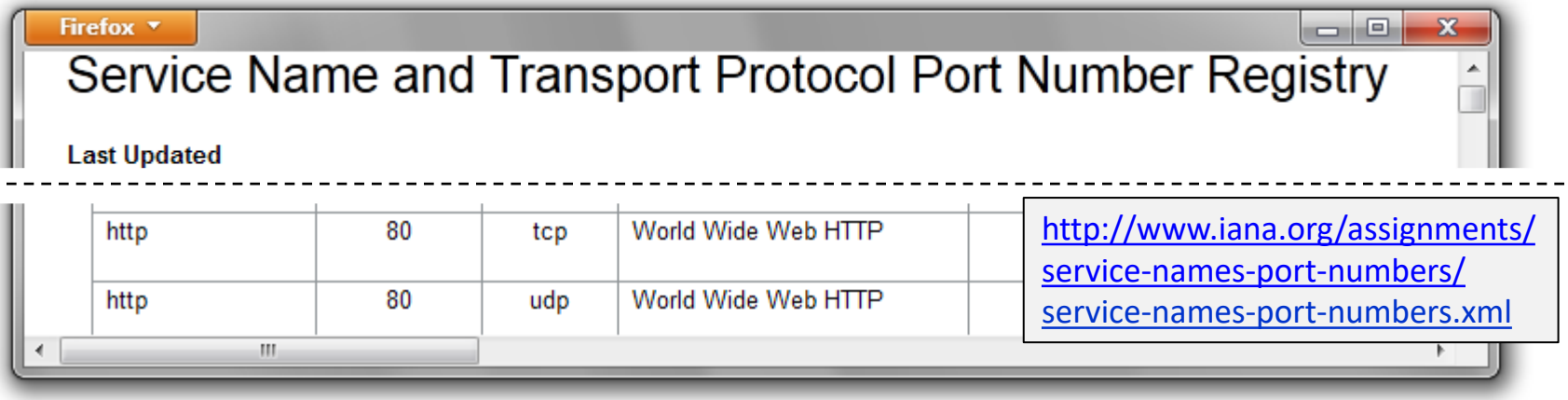
koncept portů

- abstraktními transportními adresami v TCP/IP jsou tzv. porty
  - jde o čísla v rozsahu 16 bitů: s hodnotami od 0 do 65 535
- porty jsou všude stejné
  - a konkrétní entity (např. procesy) se k nich dynamicky „asociují“ (angl. bind)
- princip fungování portů
  - odesilatel posílá svá data *„na konkrétní port na konkrétním uzlu“*
    - adresuje ji dvojici <IP adresa> : <port>
      - například: 195.113.20.128:80
  - příjemcem je *„ta entita, která je právě asociována s příslušným portem“*
    - například: s portem č.80 je asociována entita, poskytující služby WWW serveru



dobře známé porty

- odesilatel (kdokoli „zvenku“)
  - nepotřebuje vědět, která konkrétní entita je právě asociována s daným portem
  - potřebuje vědět, co tato entita dělá a jaké služby poskytuje
- důsledek
  - s některými porty je dopředu (a priori) spojena konkrétní funkčnost
    - příklad: na portu č. 80 jsou poskytovány služby WWW serveru
  - musí být založeno na konvenci
    - kterou někdo spravuje a udržuje – zde organizace IANA
  - jde konkrétně o konvenci o tzv. dobře známých portech (well-known ports)
    - konkrétně jde o porty 0 až 1023
      - dříve byla zveřejňována formou RFC dokumentu, dnes je publikována on-line



dobře známé a registrované porty

- dobře známé porty (0 až 1023)
  - konvence zajišťuje unikátnost účelu
    - stejný port slouží jen jednomu účelu
      - je pro daný účel vyhrazen
        - typicky: „pro systémové věci“
    - neměl by se používat pro jiné účely
- registrované porty (1024 až 49151)
  - konvence zajišťuje unikátnost účelu
    - každý port je (za)registrován jen pro jeden účel
    - ale může se používat i pro jiné účely
      - i pro „uživatelské věci“
        - proto též tzv. uživatelské porty
- dynamické porty (49152 až 65535)
  - žádná konvence o jejich využití
  - mohou být využity pro jakékoli účely
    - bez potřeby/možnosti registrace

UDP

Port #	Popis
21	FTP
23	Telnet
25	SMTP
69	TFTP
70	Gopher
80	HTTP
88	Kerberos
110	POP3
119	NNTP
143	IMAP
161	SNMP
443	HTTPS
993	IMAPS
995	POP3S

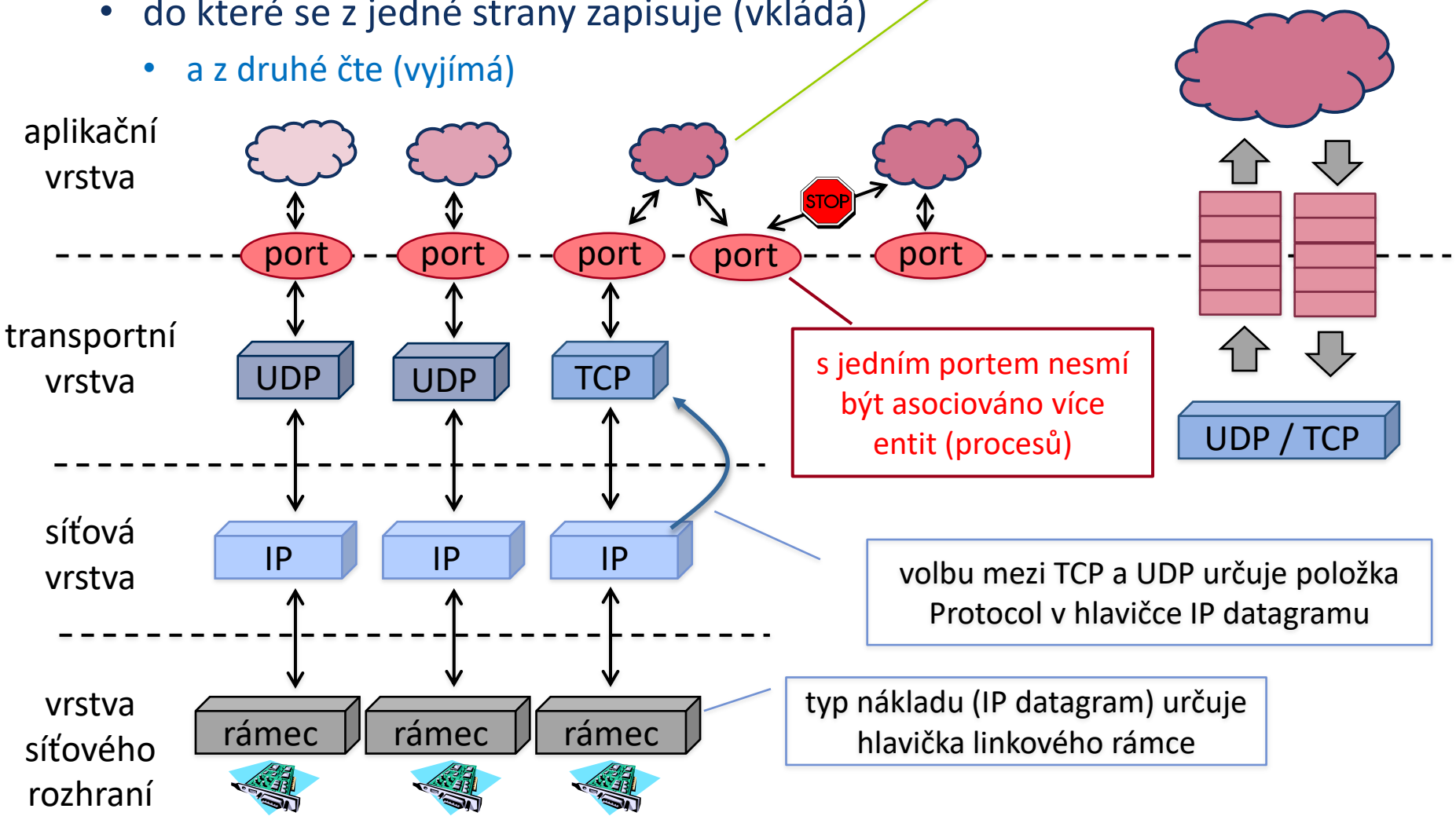
TCP

Port #	Popis
21	FTP
23	Telnet
25	SMTP
69	TFTP
70	Gopher
80	HTTP
88	Kerberos
110	POP3
119	NNTP
143	IMAP
161	SNMP
443	HTTPS
993	IMAPS
995	POP3S

je-li to možné, je konvence stejná pro UDP i TCP !!!

představa portů

- porty si lze představit jako přechodové body mezi transportní vrstvou a aplikační vrstvou
  - jako datovou strukturu charakteru fronty
    - do které se z jedné strany zapisuje (vkládá)
      - a z druhé čte (vyjímá)

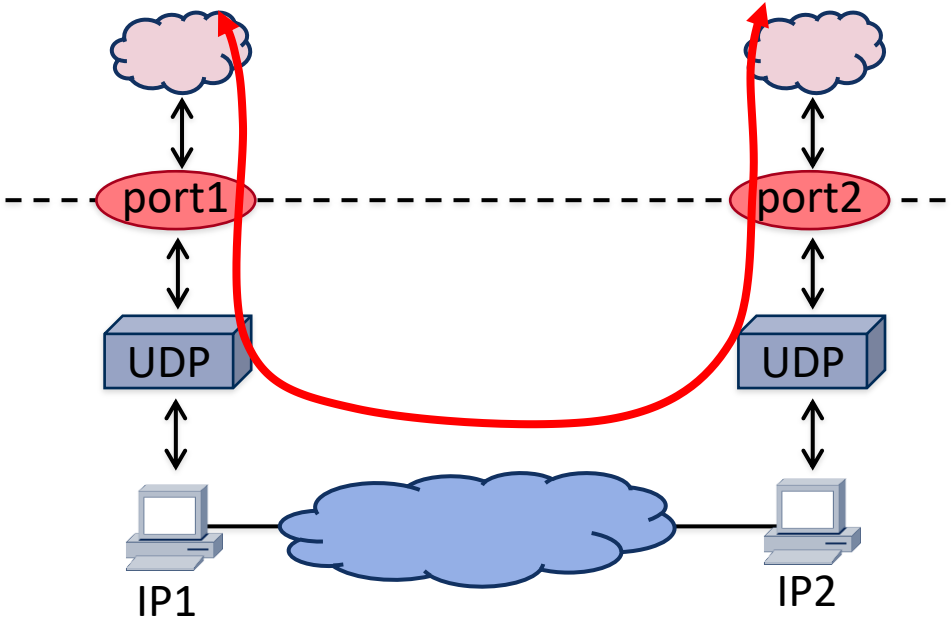


identifikace aplikačních spojení

- jak jednoznačně definovat spojení/přenos mezi dvěma různými entitami (na úrovni aplikační vrstvy)?
  - čtveřice <IP1,port1,IP2,port2> nestačí
    - protože může být využit jak protokol UDP, tak TCP
      - a mohou to být na sobě nezávislé přenosy

proto je třeba přidat ještě údaj o transportním protokolu

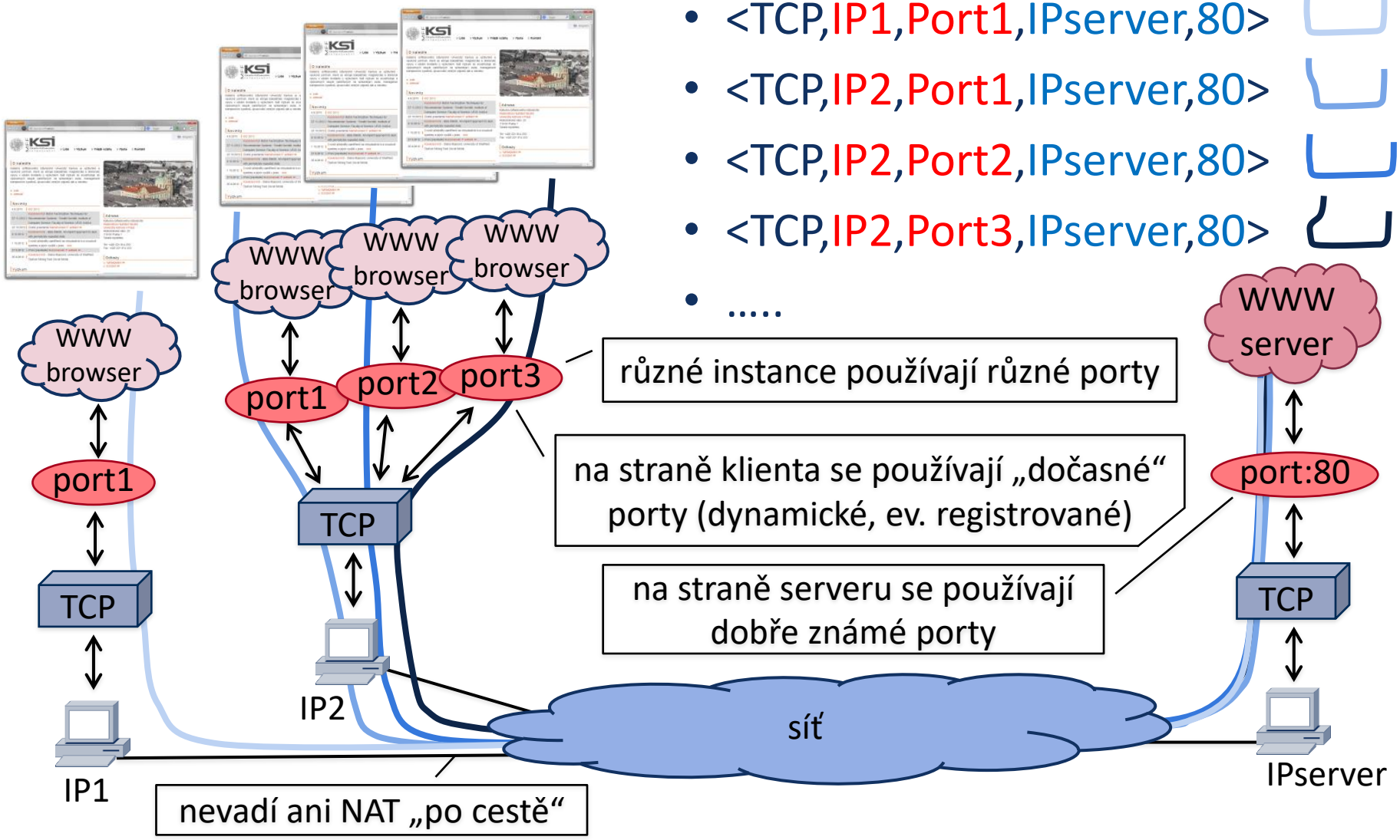
- spojení (u TCP) či přenos (u UDP) jednoznačně identifikuje až pětice:
  - **<transportní protokol,IP1,port1,IP2,port2>**



- díky tomu lze jednoznačně identifikovat (rozlišit mezi sebou) nejrůznější kombinace spojení
  - kdy více spojení „vede“ ke stejné entitě na stejném uzlu
    - např. „vede“ k jednomu serveru
  - kdy spojení „začínají“ na více různých entitách na stejném uzlu
    - např. spojení z více záložek jedné instance browseru

aplikační spojení

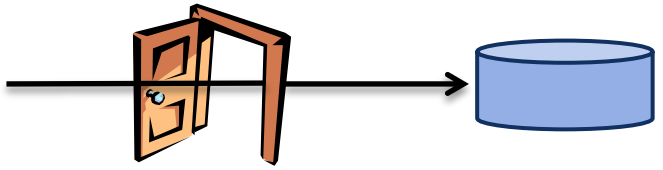
- příklad: 1x WWW server (port 80), více klientů (instancí/záložek)
  - server dokáže rozlišit požadavky různých klientů (instancí/záložek) podle pětice
    - <TCP,IP1,Port1,IPserver,80>
    - <TCP,IP2,Port1,IPserver,80>
    - <TCP,IP2,Port2,IPserver,80>
    - <TCP,IP2,Port3,IPserver,80>
    - .....



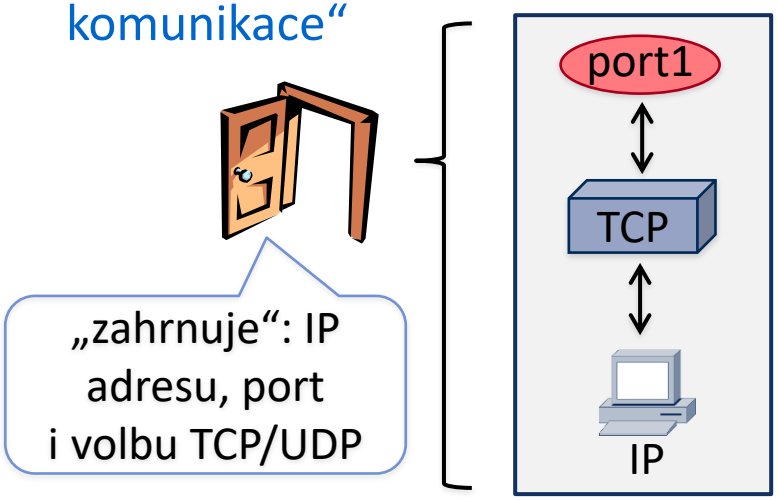
porty vs. sockety

- porty jsou abstrakcí
  - jsou všude (na všech platformách) stejné
    - jsou identifikovány čísly
  - všude (na každé platformě) musí být nějak implementovány
    - a tato implementace již může být různá
- nejčastější formou implementace jsou tzv. sockety
  - socket je abstrakce souboru, poprvé vytvořená v BSD Unixu
    - se souborem se pracuje stylem Open – Read/Write – Close
      - soubor se nejprve musí otevřít (open)
      - pak se z něj dá číst nebo do něj zapisovat (read, write)
      - na konci se zase musí zavřít (close)

- socket si ho představit jako bránu
  - která může „vést“ k souboru



  - nebo být „koncovým bodem komunikace“



  - se sockety se pracuje skrze „socketové API“
    - je k dispozici na většině systémových platforem, včetně MS Windows
      - rozhraní/API Winsock

princip práce se sockety

- v rámci API jsou definovány základní operace se sockety, např.
  - **SOCKET (domain, type, protocol)** /* vytvoření nového socketu */
    - sockety mají různé typy
      - stream socket: přenáší data jako proud dat (bez jakéhokoli logického členění)
        - obousměrně, spolehlivě, se zachováním pořadí a eliminací duplicit
      - datagram socket: přenáší data po blocích
        - obousměrně, bez zajištění spolehlivosti, bez zachování pořadí a eliminace duplicit
      - raw socket: pro přímý přístup k protokolům nižších vrstev
        - pro systémové účely, obvykle přenáší data po blocích
    - sockety mohou fungovat v různých „doménách“, např.
      - Unix (File) domain: pro „vnitřní“ účely, přístup k souborům
      - Internet domain: pro „komunikační účely“ pomocí protokolů TCP/IP
        - včetně rozlišení mezi IPv4 a IPv6
    - a pracují s různými protokoly (v Internet domain):
      - TCP, UDP, SCTP, DCCP
  - **BIND (…., číslo portu)** /* vytvořený socket je asociován se zadaným portem */
    - na všech síťových rozhraních, které uzel má

nově vytvořený socket ještě není asociován s žádným portem

nespojovaný způsob komunikace (UDP)

- není navazováno spojení, vždy je třeba explicitně specifikovat cílovou adresu&port
  - **SENDTO (socket, data, …, adresa,….)** /*odeslání dat nespojovaným způsobem */
    - skrze socket pošle data na zadanou adresu (IP adresa, port)
  - **RECVFROM (socket, data, .., adresa, ..)** /* příjem dat nespojovaným způsobem */
    - skrz socket přijme data ze zadané adresy /* data a adresa jsou výstupní parametry */

„UDP socket“

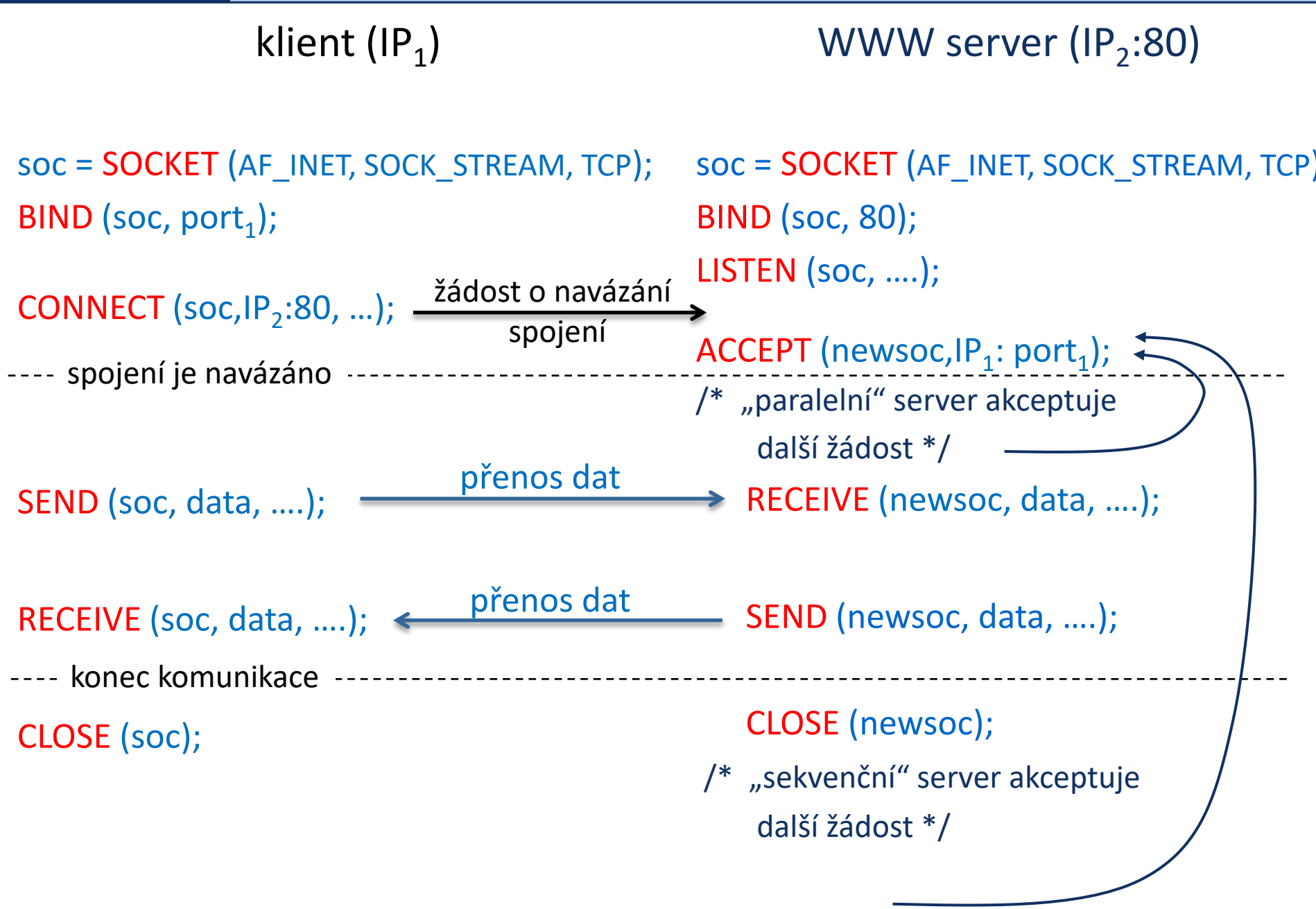
odesilatel (IP_1):	příjemce (IP_2):
soc = SOCKET (AF_INET, SOCK_DGRAM,UDP);	soc = SOCKET (AF_INET, SOCK_DGRAM,UDP);
/* vytvoření socketu */	/* vytvoření socketu */
BIND (soc, $port_1$);	BIND (soc, $port_2$);
/* asociace s $port_1$ */	/* asociace s $port_2$ */
SENDTO (soc, data, …, IP_2:$port_2$, …);	
	RECEIVEFROM (soc, data, …, adresa, …);
	SENDTO (soc, data, …, IP_1:$port_1$, …);
RECEIVEFROM (soc, data, … adresa, …);	
CLOSE (soc);	CLOSE (soc);
/* zrušení socketu */	/* zrušení socketu */

spojovaný způsob komunikace (TCP)

- adresa protistrany se zadává jen při navazování spojení
  - **CONNECT (socket, adresa, …)** /* pro toho, kdo navazuje spojení - klienta */
    - požadavek na navázání spojení s protistranou na zadané adrese (IP, port)
  - **LISTEN (socket, …)** /* pro toho, kdo čeká na výzvu k navázání spojení – server */
    - uvedení socketu do „stavu poslouchání“
      - nikoli samotné čekání na příchod žádosti
  - **ACCEPT (socket, adresa, ….)** /* přijetí požadavku na navázání spojení */
    - kladná odpověď na žádost o navázání spojení, vyslání potvrzení o navázání
      - pokud žádná žádost dosud nepřišla, ACCEPT na ni čeká
    - je vytvořen nový socket a spojení je navázáno s ním
      - server může požadavky v rámci spojení vyřizovat sekvenčně nebo paralelně
  - **SEND (socket, data, ….)**
    - pošle data skrz spojení, navázané se socketem
  - **RECV (socket, data, ….)**
    - přijme data skrze spojení, navázané se socketem
  - **CLOSE (socket)**
    - v případě TCP ukončí spojení a uvolní zdroje přidělené socketu (paměť atd.)

v případě UDP jen uvolní zdroje

spojovaný způsob komunikace (TCP)

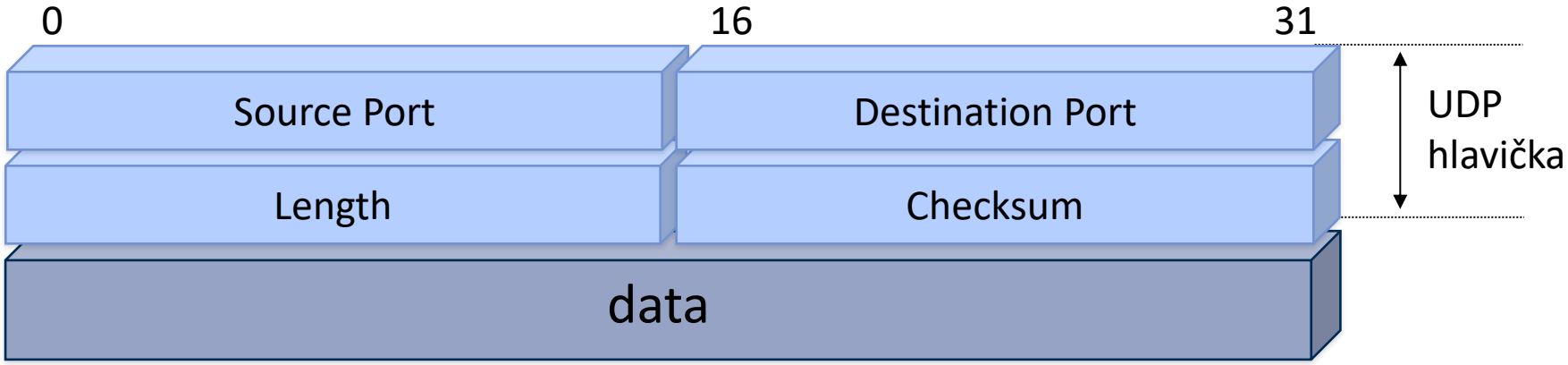


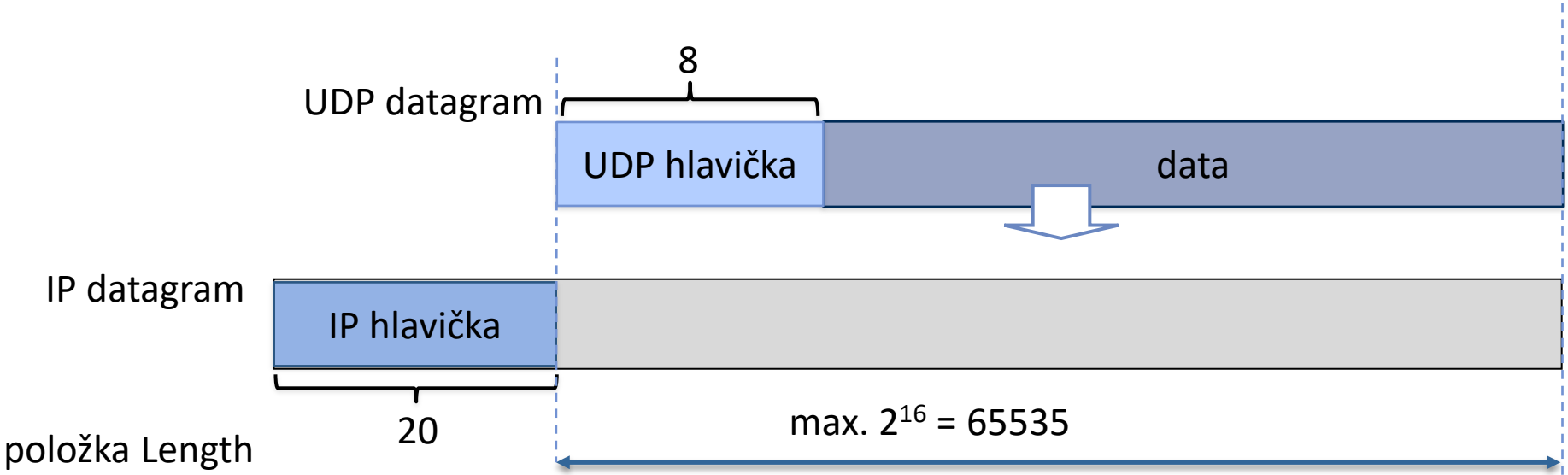
UDP (User Datagram Protocol)

- je maximálně jednoduchou nadstavbou nad protokolem IP
  - nemění základní vlastnosti protokolu IP
  - navíc poskytuje jen multiplexing/demultiplexing
  - má kontrolní součet, který pokrývá hlavičku i data
    - kontrolní součet IP datagramu pokrývá pouze hlavičku
    - kontrolní součet UDP datagramu lze vypnout
- protokol UDP používají takové aplikace, které potřebují co nejrychlejší a nejefektivnější komunikaci
  - UDP není zatížen velkou režií
    - jako protokol TCP
- vlastnosti UDP:
  - poskytuje nespolehlivé přenosové služby
  - funguje nespojovaně
  - vytváří iluzi blokového přenosu
    - přenáší UDP Datagramy
    - velikost bloku (datagramu):
      - taková, aby se vešla do IP datagramu (2^{16} – 20 - 8)
        - ale mělo by se předcházet fragmentaci – dbát na MTU
      - v praxi se posílají spíše malé bloky, např. do 512 bytů
  - může být použit pro rozesílání
    - broadcast i multicast
      - u spojovaného protokolu to nejde
  - komunikace je bezestavová
    - u TCP je stavová

formát UDP datagramu

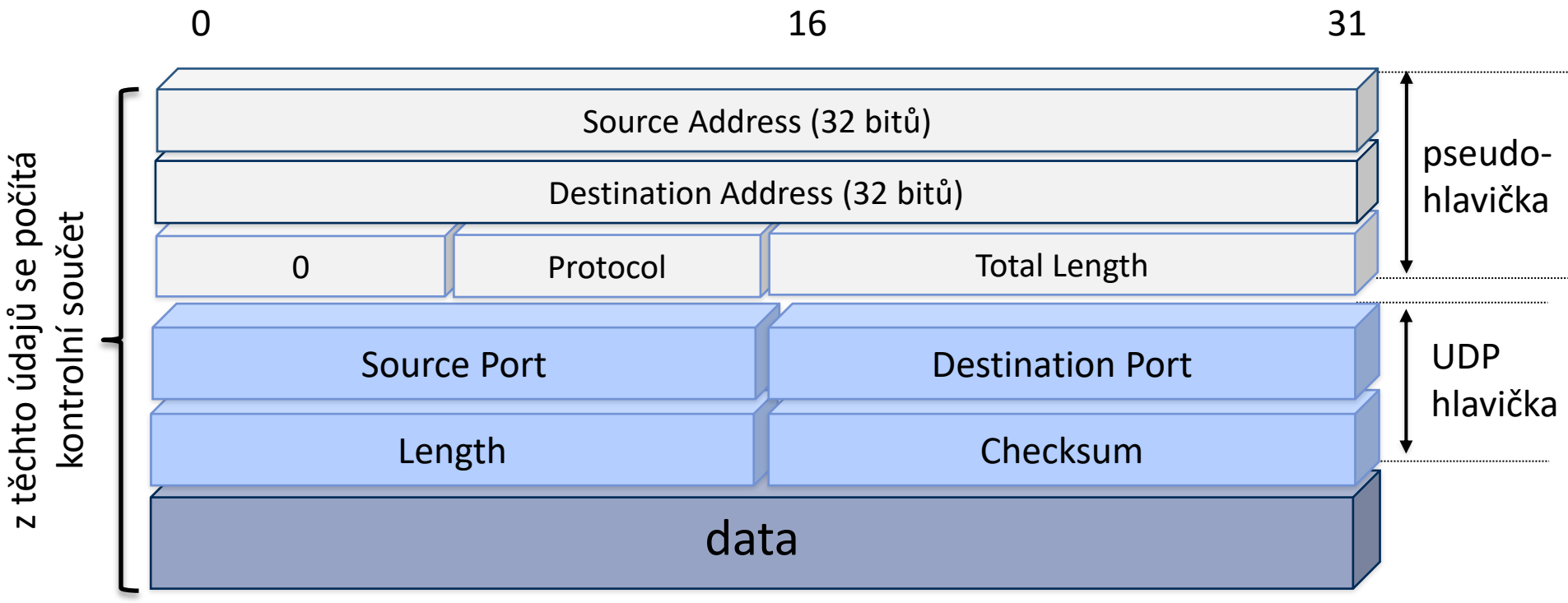
- je velmi jednoduchý
  - celý datagram má proměnnou velikost: max 2^{16} bytů
  - hlavička má pevnou velikost: 8 bytů
    - obsahuje volitelný kontrolní součet (Checksum)





pseudohlavička UDP datagramu

- (volitelný) kontrolní součet se počítá z celého UDP datagramu, doplněného o pseudohlavičku
  - ale tato pseudohlavička se nepřenáší
    - existuje jen pro potřebu výpočtu kontrolního součtu
- kontrolní součet se počítá v jedničkovém doplňku (one's complement)
  - nulový kontrolní součet = samé jedničky (tzv. záporná nula)
  - žádný kontrolní součet = samé nuly (tzv. kladná nula)



smysl pseudohlavičky: příklad

- smyslem pseudohlavičky je ochrana proti nesprávně doručeným datagramům

stejnou ochranu používá i protokol TCP

- příklad: odesilatel odesílá UDP datagram D na adresu IP_1
  - po cestě, v důsledku nějaké chyby (či: útoku), dojde k přepsání cílové adresy (IP_1) v příslušném IP datagramu na jinou hodnotu (IP_2)
    - a tak je celý datagram doručen na nesprávný cílový uzel (s IP_2 místo IP_1).
  - bez zahrnutí pseudohlavičky by (nesprávný) cílový uzel neměl šanci poznat, že není zamýšleným příjemcem
  - díky pseudohlavičce to pozná, skrze nesprávný kontrolní součet
    - ten byl u odesilatele vypočítán ještě se správnou cílovou adresou IP_1, ale u příjemce je počítán s nesprávnou cílovou adresou IP_2, a tak se budou obě hodnoty lišit
    - UDP datagram s nesprávným kontrolním součtem je zahozen
      - a není generována ICMP zpráva o jeho zahození

TCP segmenty pracují se stejnou pseudohlavičkou

- důsledek:
  - mechanismus NAT musí přepočítávat kontrolní součet i v rámci TCP segmentů a UDP datagramů (v jejich pseudohlavičkách) !!

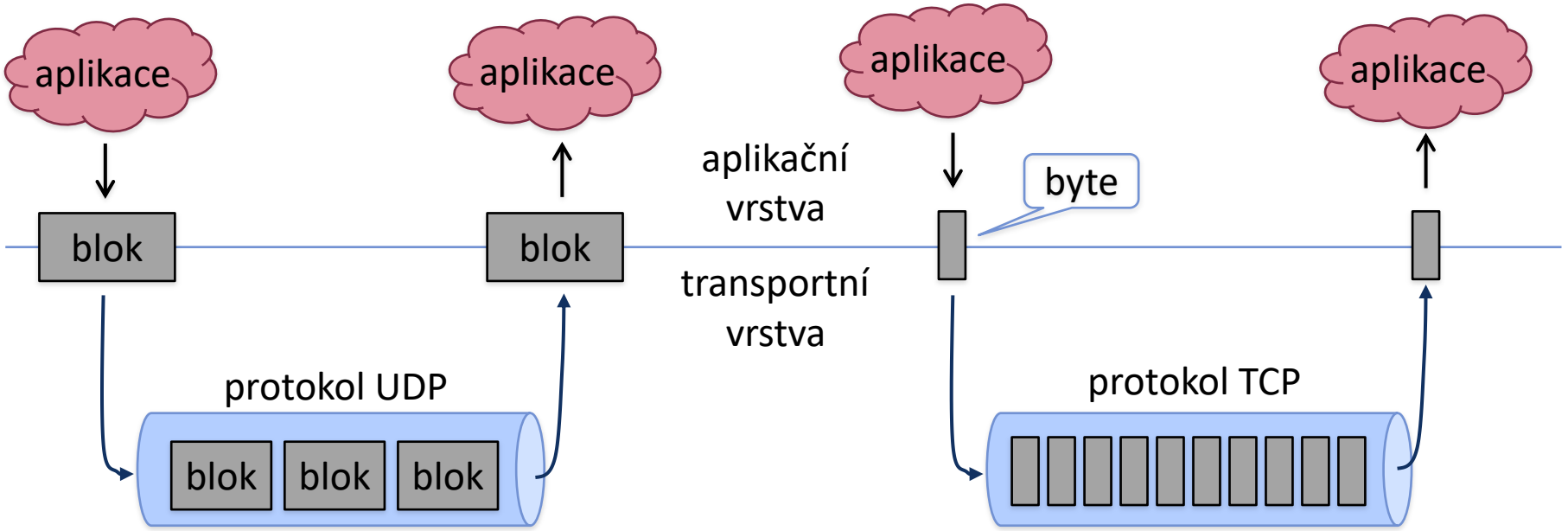
TCP: Transmission Control Protocol

- je velmi úspěšný a adaptivní
  - dobře řeší poměrně složitý problém
    - funguje efektivně i v sítích, které se významně liší svými vlastnostmi
      - např. přenosovým zpožděním
- vlastnosti poskytovaných služeb
  - spojovaný charakter
    - práce stylem: navaž spojení, posílej/přijímej, ukonči spojení
  - "plná" spolehlivost
    - protokol ošetřuje chyby při přenosech, duplicity, ztráty, garantuje pořadí doručování dat
      - využívá kontinuální potvrzování
  - dvoubodové spojení
    - vždy jen mezi jedním příjemcem a jedním odesilatelem
      - nelze využít pro broadcast či multicast

- TCP zajišťuje:
  - řízení toku
    - přizpůsobuje se schopnostem příjemce
  - ochranu před zahlcením
    - každou ztrátu dat interpretuje jako zahlcení a reaguje změnou potvrzování
  - "stream interface„ (bytové rozhraní)
    - vůči vyšším vrstvám vytváří iluzi bytové roury, přijímá i vydává data po bytech, nikoli po blocích
  - korektní navazování a ukončování spojení
    - zajišťuje, že obě strany souhlasí s navázáním spojení a že nedojde k deadlocku ani "ztrátám" pokusů o navázání spojení
    - zajistí, že před ukončením spojení jsou přenesena všechna odeslaná data

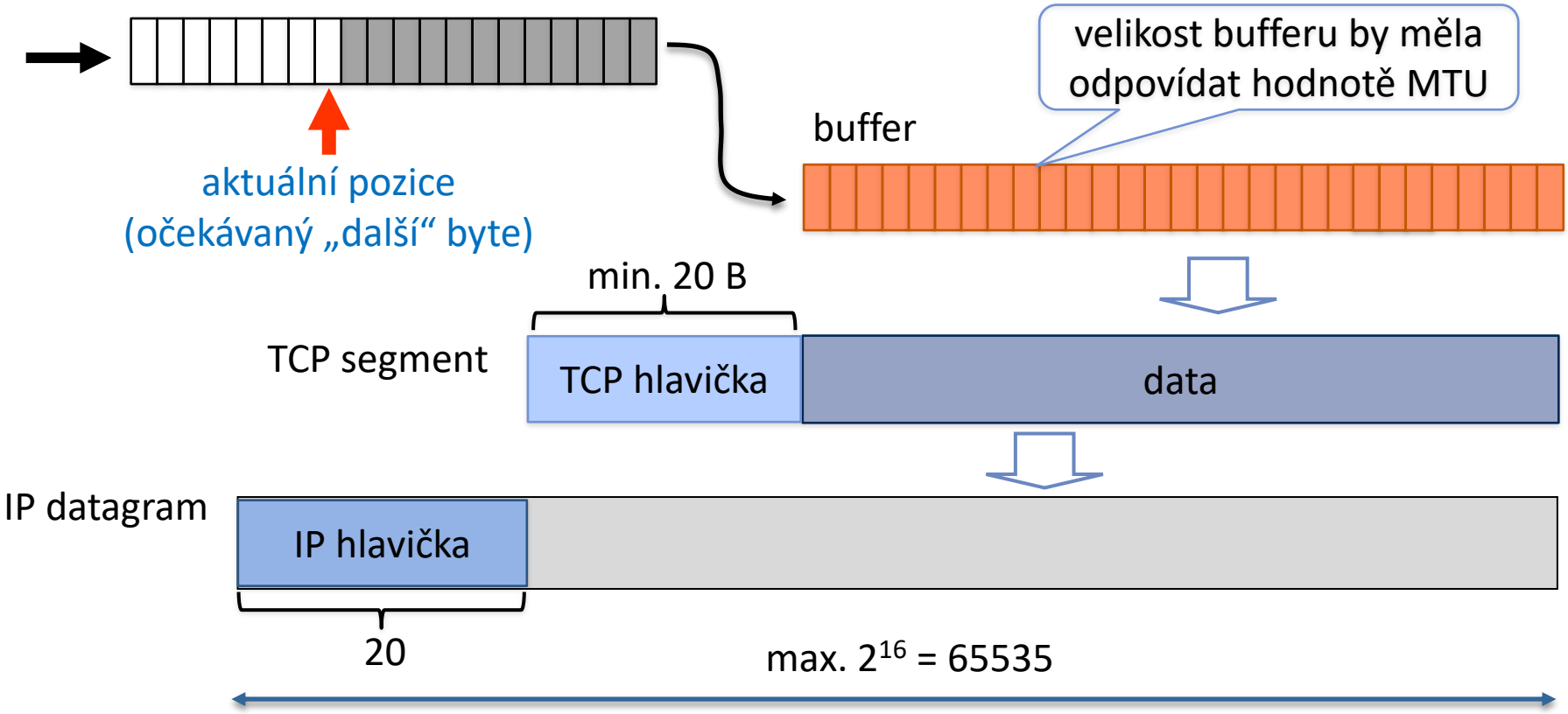
srovnání UDP a TCP

- UDP přenáší celé bloky dat
  - vytváří „blokové rozhraní“ vůči vyšším vrstvám
    - od entit aplikační vrstvy dostává k přenosu celé bloky dat
      - data, „již naporcovaná“ na bloky
    - tyto bloky vkládá do UDP datagramů
- TCP přenáší data jako proud
  - vytváří „bytové rozhraní“
    - od entit aplikační vrstvy dostává data k přenosu po jednotlivých bytech
      - vytváří jim iluzi „bytové roury“
        - ve skutečnosti také přenáší data po blocích: tzv. TCP segmentech



TCP segmenty a iluze bytové roury

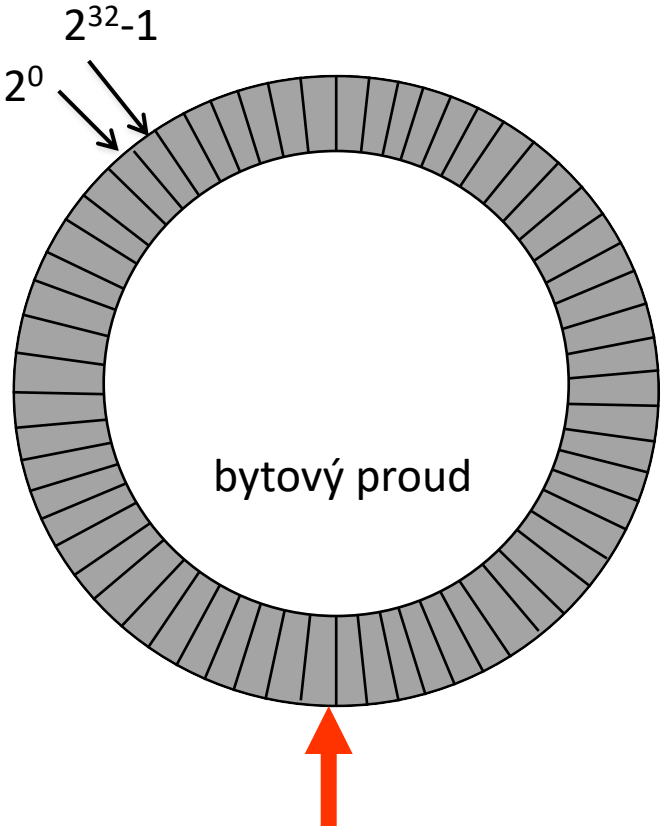
- bytové rozhraní (stream interface) protokolu TCP je pouze iluzí
  - již jen proto, že protokol TCP sám využívá služeb protokolu IP, který přenáší celé bloky dat (IP datagramy) a nikoli jednotlivé byty
- TCP ve skutečnosti ukládá jednotlivé byty do svého bufferu
  - a odesílá ho (standardně) až tehdy, když se celý naplní
    - nebo když si aplikace explicitně vyžádá odeslání (příkaz PUSH)



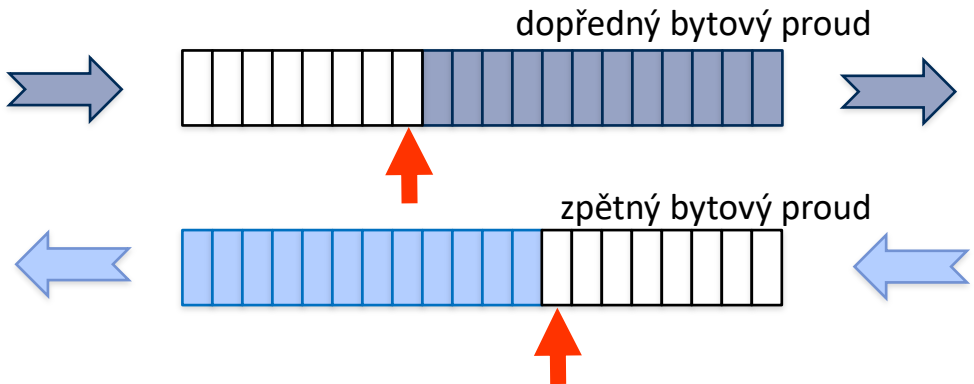
pozice v bytovém proudu

- TCP pracuje s pozicemi v bytovém proudu v rozsahu 32 bitů
  - to odpovídá představě „lineárního“ bytového proudu s pozicemi od 0 do 2^{32}-1

  - ve skutečnosti je bytový proud „zacyklen“
    - počítá se modulo 2^{32}

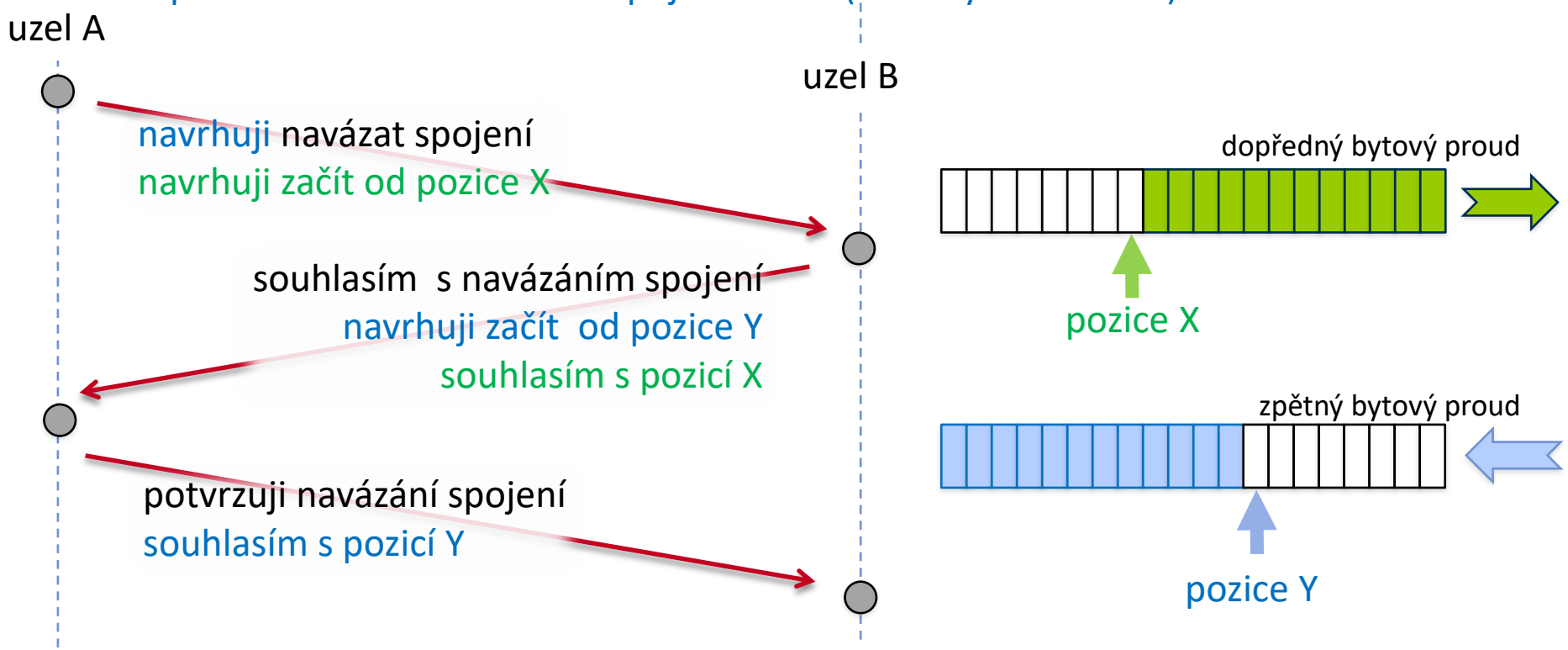


- TCP je (plně) duplexní
  - přenáší data v obou směrech
    - proto pracuje se dvěma bytovými proudy
      - a dvěma aktuálními pozicemi



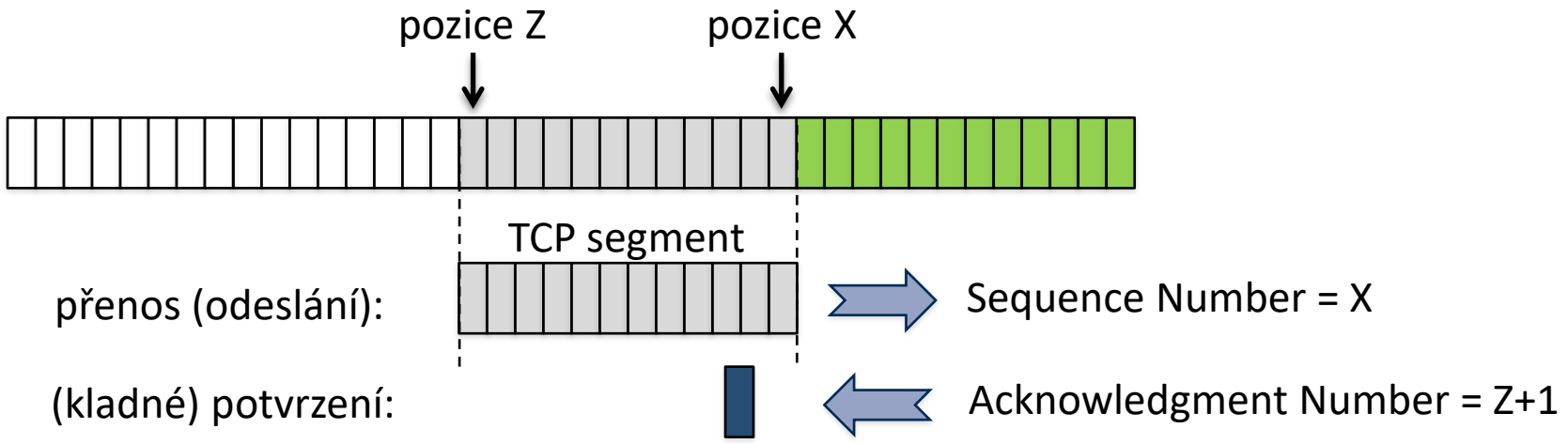
pozice v bytovém proudu

- pro bezpečnost je důležité, aby pozice v bytovém proudu nezačínaly na stejných hodnotách
- vhodné řešení: budou začínat na náhodně zvolených pozicích (v obou směrech)
  - v rámci navazování spojení se obě strany musí na těchto počátečních pozicích (ISN, Initial Sequence Number) dohodnout
    - proto musí mít navazování spojení 3 fáze (3 – way handshake)



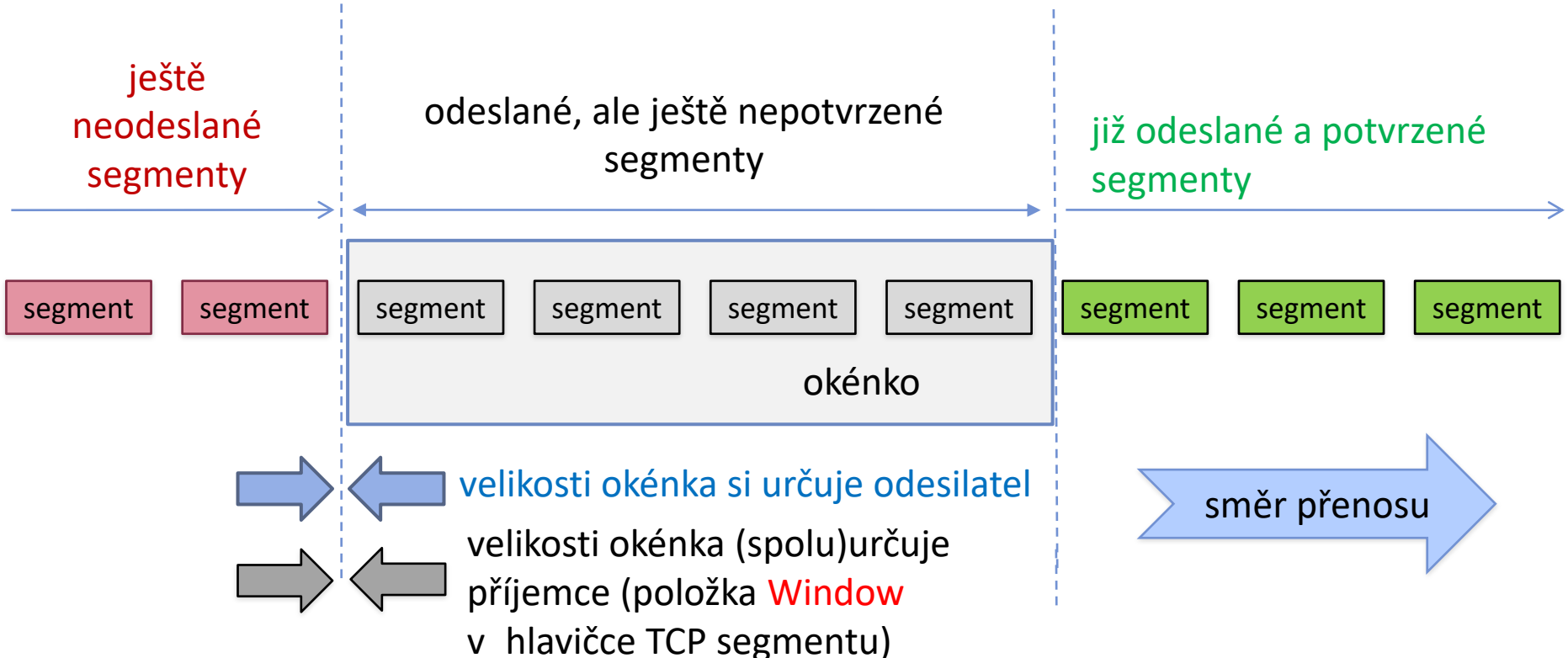
potvrzování

- TCP zajišťuje spolehlivý přenos: pomocí kontinuálního potvrzování
  - ale: nečísluje jednotlivé TCP segmenty
    - místo toho identifikuje data podle jejich pozice v bytovém proudu
- při odesílání:
  - říká: *„posílám data z proudu počínaje pozicí X“*
    - v hlavičce TCP segmentu jde o položku Sequence Number
- při potvrzování:
  - říká: *„přijal jsem v pořádku data až do pozice Z“*
    - přesněji: *„jako další očekávám data od pozice Z+1“*
      - v hlavičce TCP segmentu jde o položku Acknowledgment Number



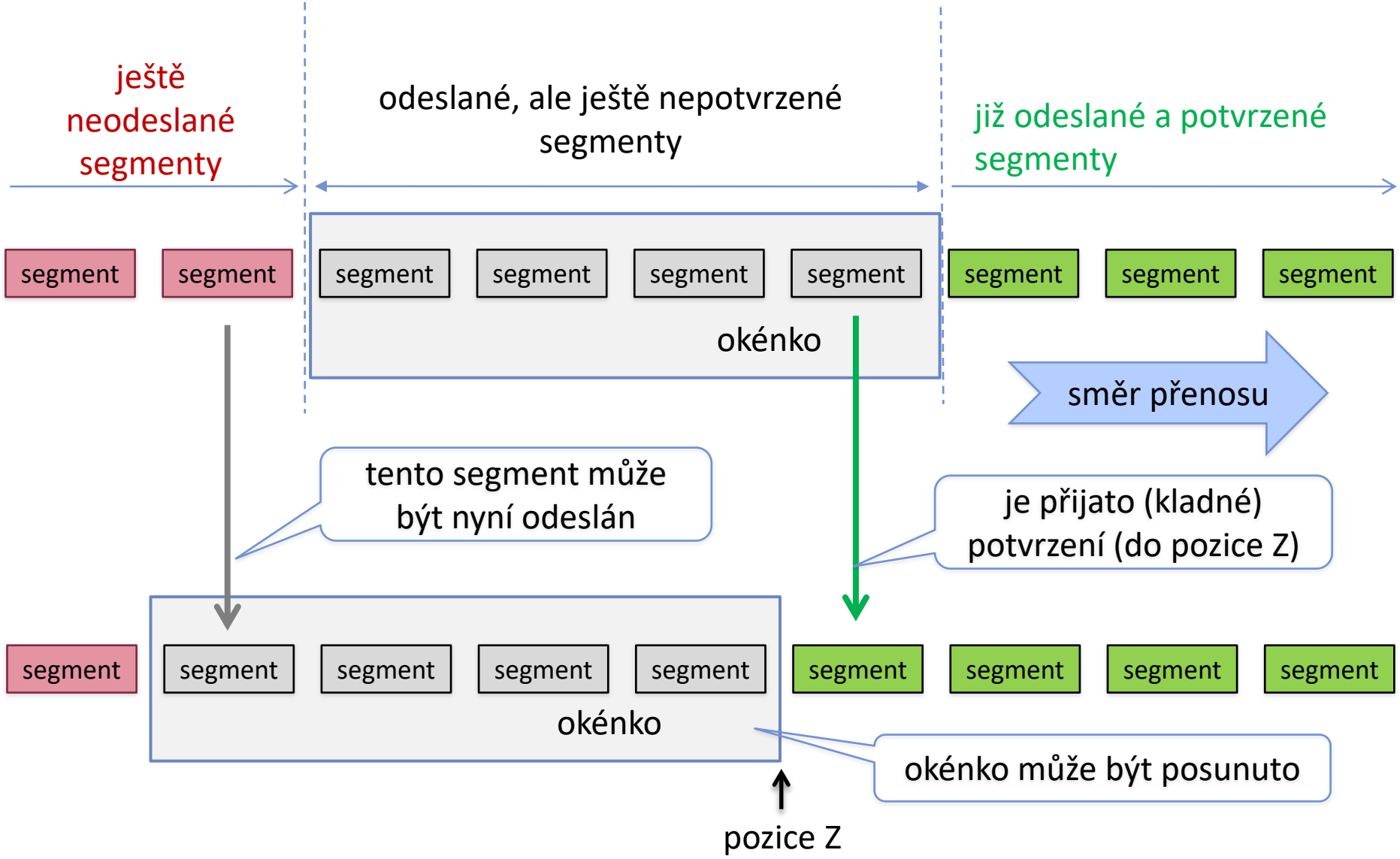
potvrzování a řízení toku

- TCP používá metodu okénka
  - jak pro (kontinuální) potvrzování, tak i pro řízení toku
- představa:
  - „okénko“ udává, kolik dat ještě může odesilatel odeslat
    - v rámci kontinuálního potvrzování: než dostane potvrzení o jejich doručení
    - v rámci řízení toku: aby nezahltil příjemce



metoda okénka

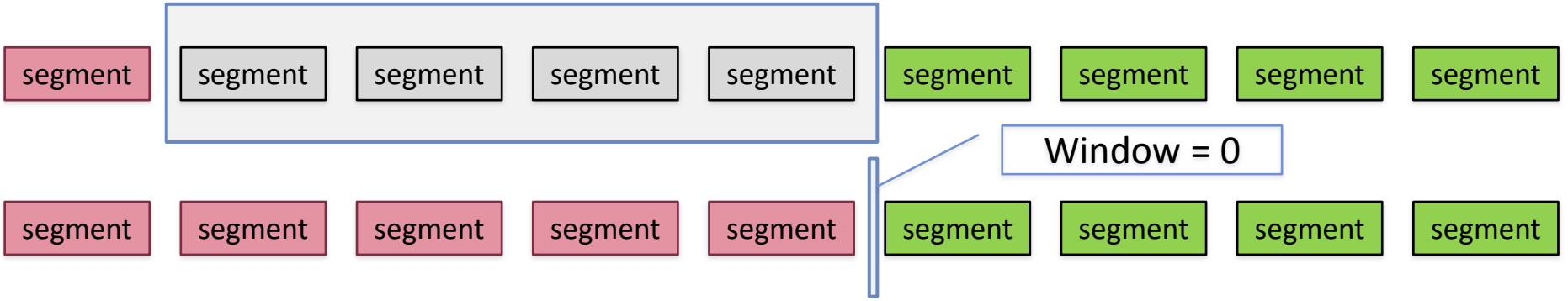
- odesilatel si průběžně „posouvá“ okénko podle toho, jak mu přichází (kladná) potvrzení o doručení



řízení toku

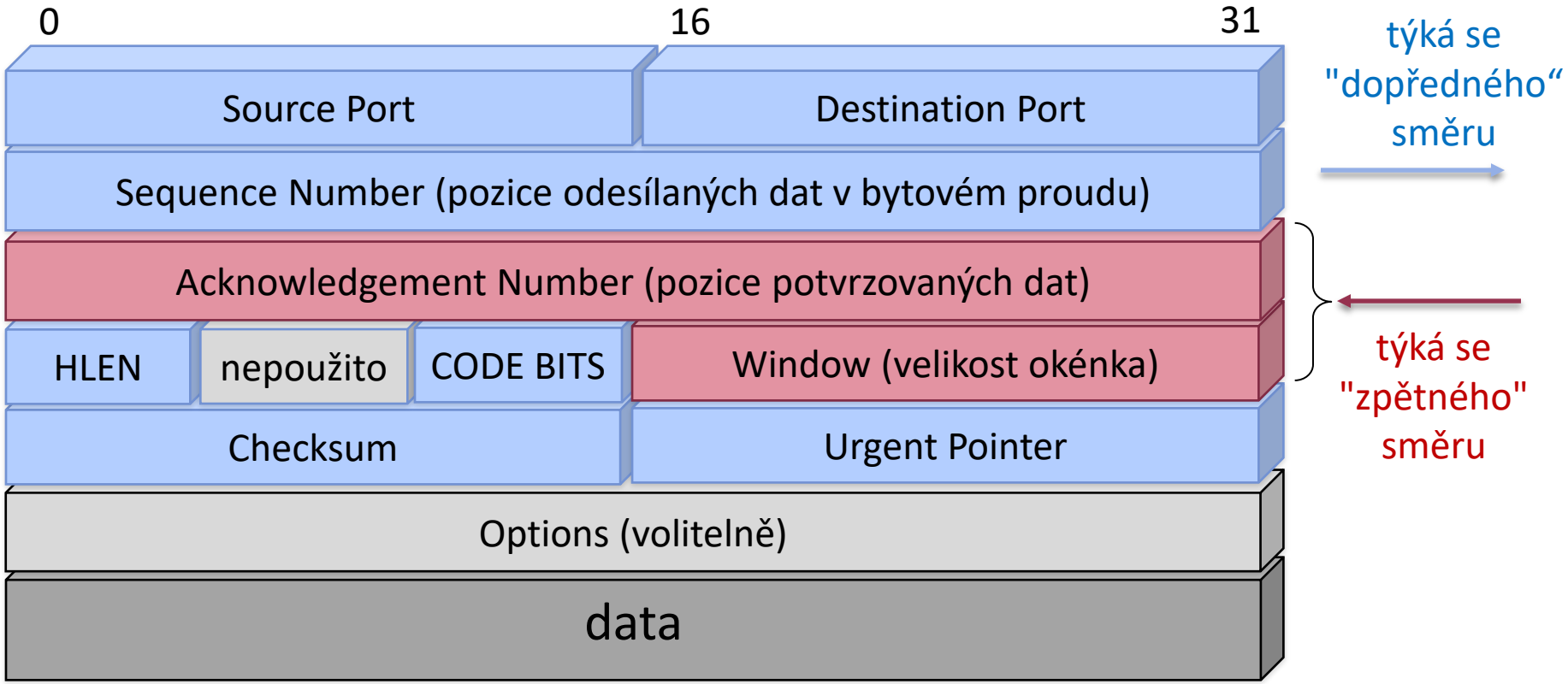
- cílem řízení toku je zabránit zahlcení příjemce
  - pokud by mu odesilatel posílal více dat, než je schopen přijmout
- v rámci protokolu TCP
  - příjemce „inzeruje“, kolik dat je ještě schopen přijmout
  - odesilatel podle toho nastavuje velikost okénka
    1. podle zpoždění, s jakým mu přichází jednotlivá potvrzení, v rámci kontinuálního potvrzování
    2. podle inzerovaného objemu dat (položka Window), který je příjemce schopen přijmout

    velikost okénka se řídí tou hodnotou, která je více omezující
  - příjemce může úplně zastavit odesílání dalších dat
    - když inzeruje svou schopnost přijmout nulový objem dat (Window = 0)

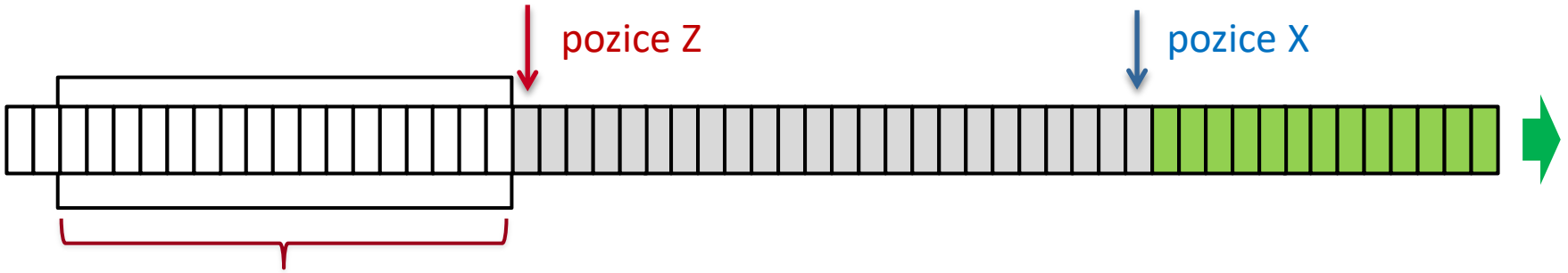


formát TCP segmentu

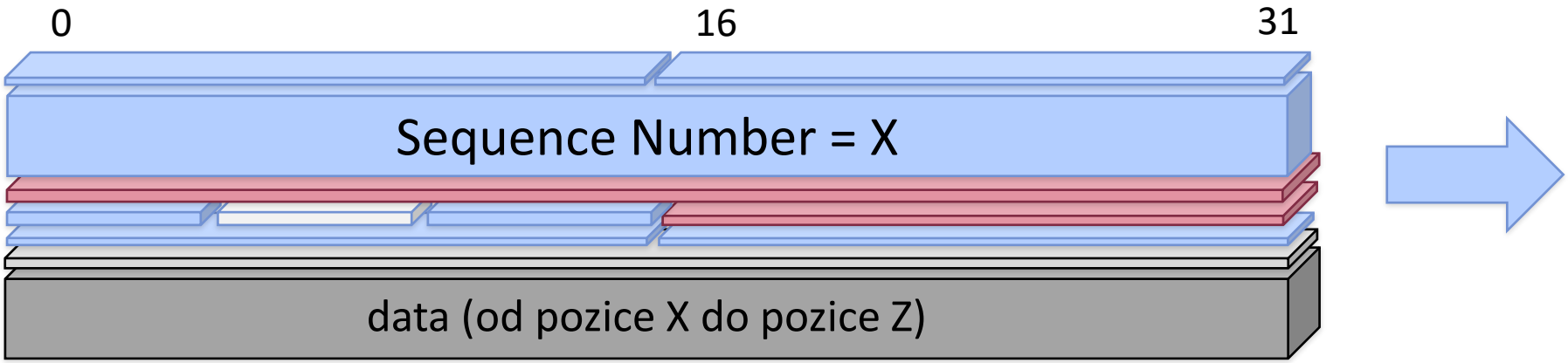
- hlavička TCP segmentu má proměnnou velikost
  - proto potřebuje explicitní údaj o své délce (HLEN: Header LENgth)
    - má 4 bity, velikost se udává v násobcích 32-bytových slov
      - někdy je tato položka označována jako Data Offset
        - „kolik zbývá do začátku užitečných dat“
- obsahuje údaje, které se týkají obou směrů přenosu

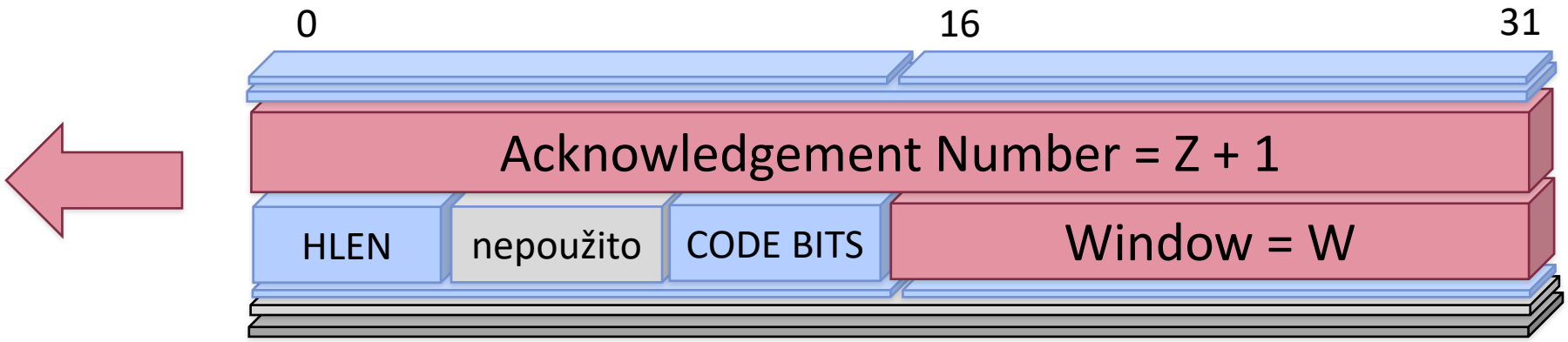


formát TCP segmentu



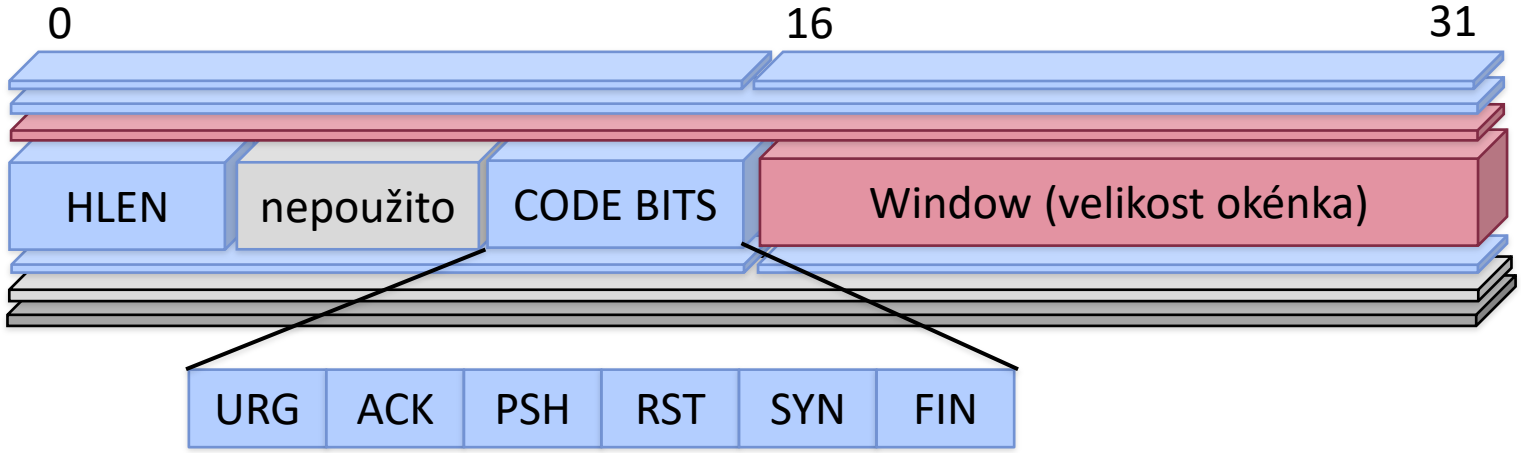
W – inzerovaná velikost okénka (kapacita pro příjem dalších dat)





formát TCP segmentu

- hlavička obsahuje 6 příznaků, význam při nastaveném příznaku:
  - URG: položka Urgent Pointer udává pozici začátku „urgentních dat“
    - ale není definováno, co to znamená
  - ACK: v položce "ACKNOWLEDGEMENT NUMBER" je platná hodnota
    - pozice dalšího očekávaného bytu
  - PSH: data byla odeslána přednostně (příkazem Push, před naplněním bufferu)
  - RST: spojení má být okamžitě zrušeno (rozvázáno, ukončeno)
  - SYN: „synchronizace“ pozic v datovém proudu (při navazování spojení)
    - v položce "SEQUENCE NUMBER" je počáteční hodnota pozice
  - FIN: povel k ukončení spojení (ale pouze v daném směru)
    - v poli "SEQUENCE NUMBER" je pořadové číslo posledního přeneseného bytu



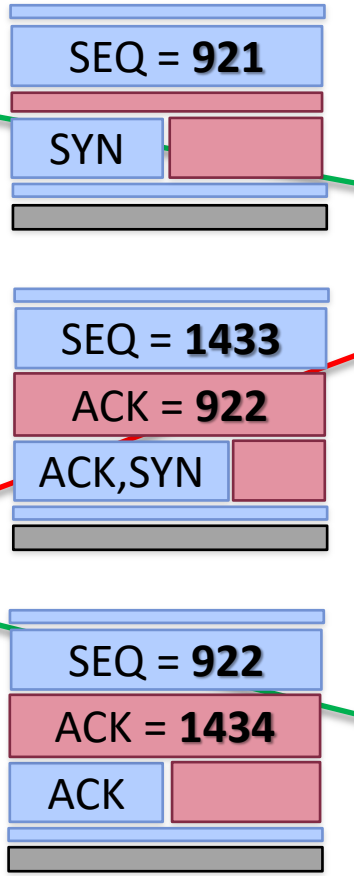
TCP segment a navazování spojení

uzel, který navazuje spojení

volí počáteční pozici v „odchozím“ bytovém proudu (zde: **921**), nastavuje příznak SYN (tzv. aktivní open)

je nutný 3-fázový dialog (3-phase handshake)

pozice v „odchozím“ bytovém proudu je domluvena, akceptuje počáteční pozici v „příchozím“ bytovém proudu (nastavuje ACK, očekává data od pozice **1434**)

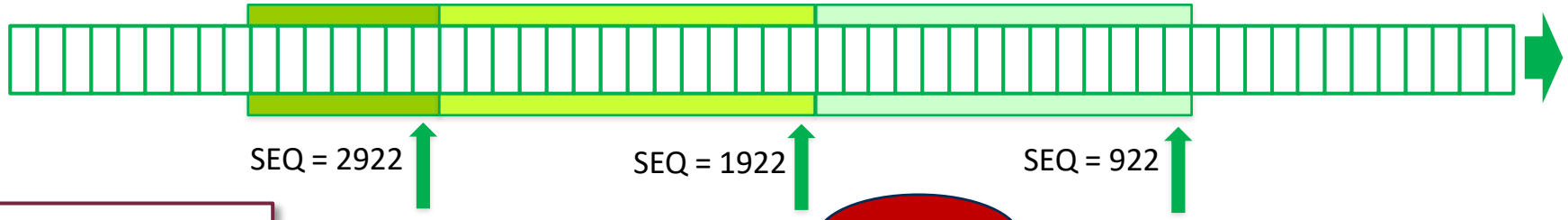


uzel, který akceptuje pojení

akceptuje počáteční pozici v „příchozím“ bytovém proudu (nastavuje ACK, očekává data od pozice **922**), volí počáteční pozici v „odchozím“ bytovém proudu (zde: **1433**), nastavuje SYN

pozice v „odchozím“ bytovém proudu je domluvena

přenos dat



postupně odešle 2500 bytů ve třech segmentech

odesilatel

příjemce

SEQ = 922
1000 bytů

SEQ = 1922
1000 bytů

SEQ = 2922
500 bytů

posuvné okénko velikosti 2500 bytů

STOP

ACK W=**2500**

je připraven přijmout **2500** bytů

ACK = **1922**
ACK W=**1500**

další data očekává od pozice **1922** (tj. potvrzuje, že přijal vše do pozice 1922-1

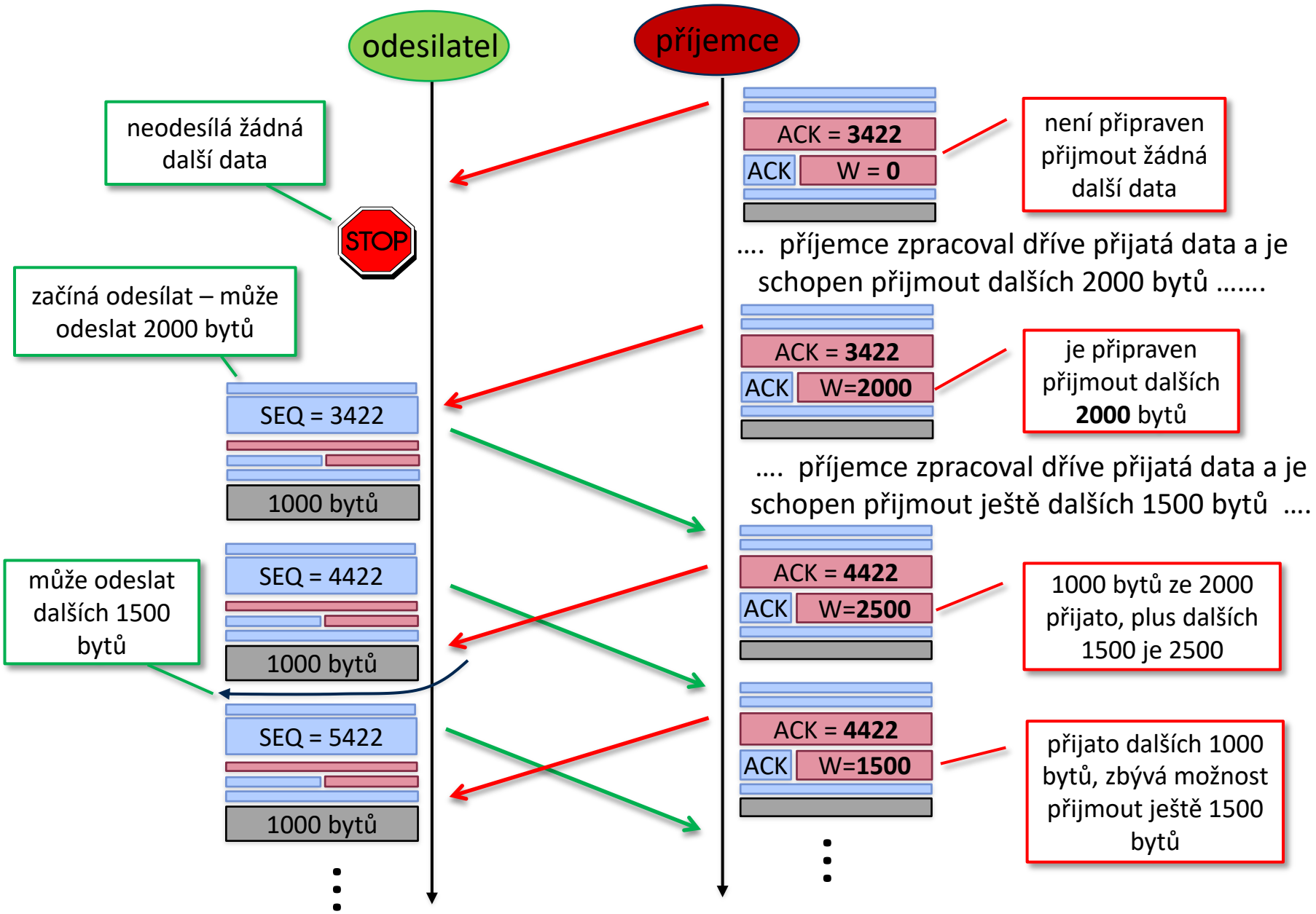
je připraven přijmout dalších **1500** bytů

ACK = **2922**
ACK W=**500**

ACK = **3422**
ACK W=**0**

není připraven přijmout žádná další data

přenos dat (pokračování)



způsob potvrzování

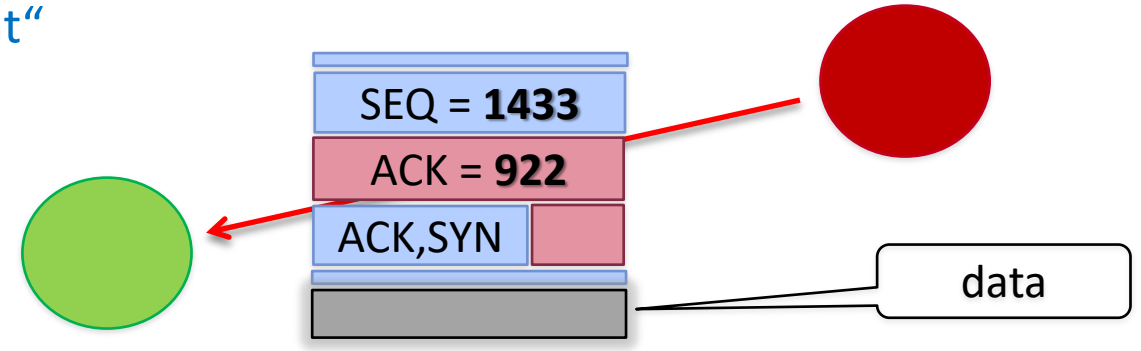
- odesilatel:
  - pro každý odeslaný TCP segment si odesilatel udržuje časovač (timer)
    - jeho počáteční hodnotu volí podle střední doby přenosového zpoždění
      - podle toho, za jakou dobu mu v průměru přichází zpět kladná potvrzení
    - pokud se časovač vynuluje a potvrzení nepřišlo, odesilatel může:
      1. znovu odeslat příslušný segment a všechny následující (do velikosti okénka)
         - což odpovídá kontinuálnímu potvrzování s návratem, nebo
      2. znovu odeslat pouze příslušný segment (který nebyl potvrzen)
         - což odpovídá selektivnímu kontinuálnímu potvrzování

- příjemce:
  - také může fungovat ve 2 režimech
    1. „in-order“, kdy přijímá pouze data „v pořadí“ a ostatní ignoruje
       - a potvrzuje stylem „přijal jsem vše do pozice X“
    2. „out-of-order“, kdy přijímá i data „mimo pořadí“ a následně se je snaží zařadit „do pořadí“
       - pak musí potvrzovat jinak než obvykle (tzv. SACK, Selective ACK), kdy potvrzuje jen určitý úsek dat
         - „od pozice X do pozice Z“

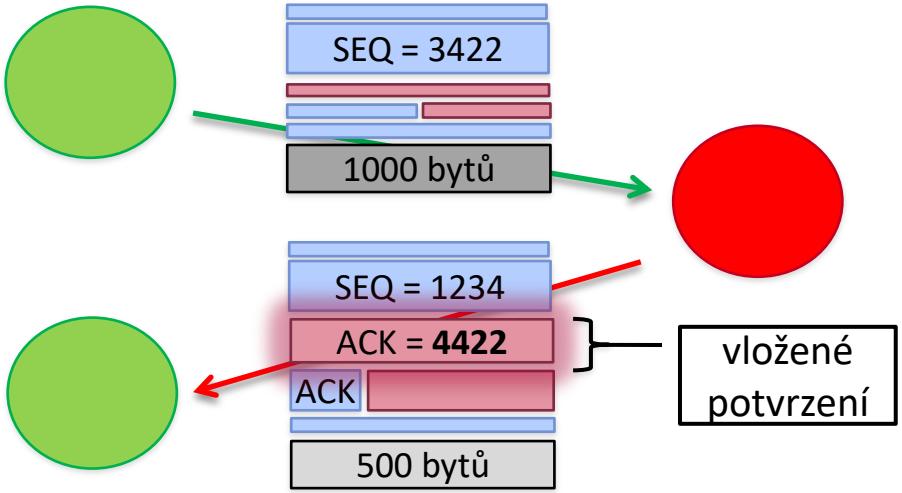
původní a implicitní (default) řešení, které je ale méně efektivní (zbytečně zatěžuje přenosové cesty)

piggybacking

- piggybacking:
  - snaha „vložit něco užitečného“ do TCP segmentu, který je přenášen „v protisměru“
- při navazování spojení:
  - pokud je nastaven příznak ACK, do TCP segmentu již může být vložena a přenesena první část „užitečných dat“



- při potvrzování:
  - potvrzení o přijetí dat lze vložit do TCP segmentu, který je přenášen „v protisměru“ s jinými „užitečnými daty“
    - pokud takový segment je přenášen
      - otázka: jak dlouho na něj čekat?

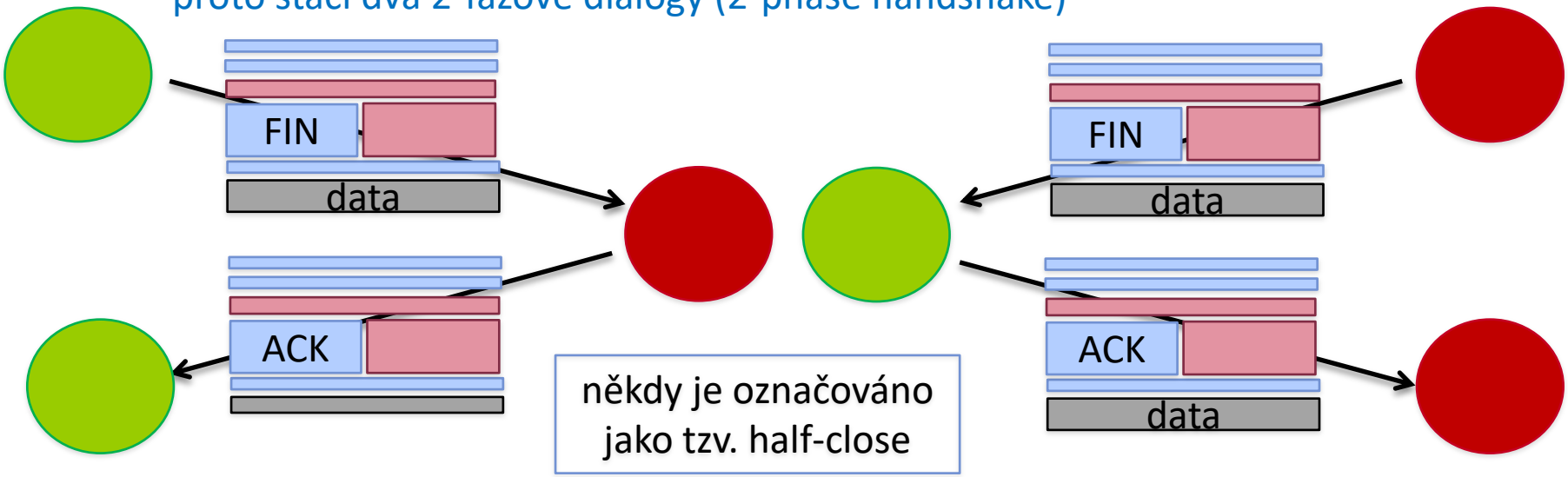


ukončování spojení

- je značně složité (ještě složitější než navazování spojení)
  - při navazování spojení: jde o 1 dvoustranný úkon
    - protože všechny aktivity probíhají bezprostředně po sobě
      - proto je zapotřebí 3-fázový dialog (3-phase handshake)
  - při ukončování spojení: jde o 2 jednostranné úkony
    - nemusí na sebe bezprostředně navazovat (ale mohou)
    - každá strana může (jednostranně) ukončit spojení
      - když v odesílaném TCP segmentu nastaví příznak FIN
    - tím říká: *„už ti nebudu dále nic posílat, ale budu stále přijímat“*
      - proto stačí dva 2-fázové dialogy (2-phase handshake)

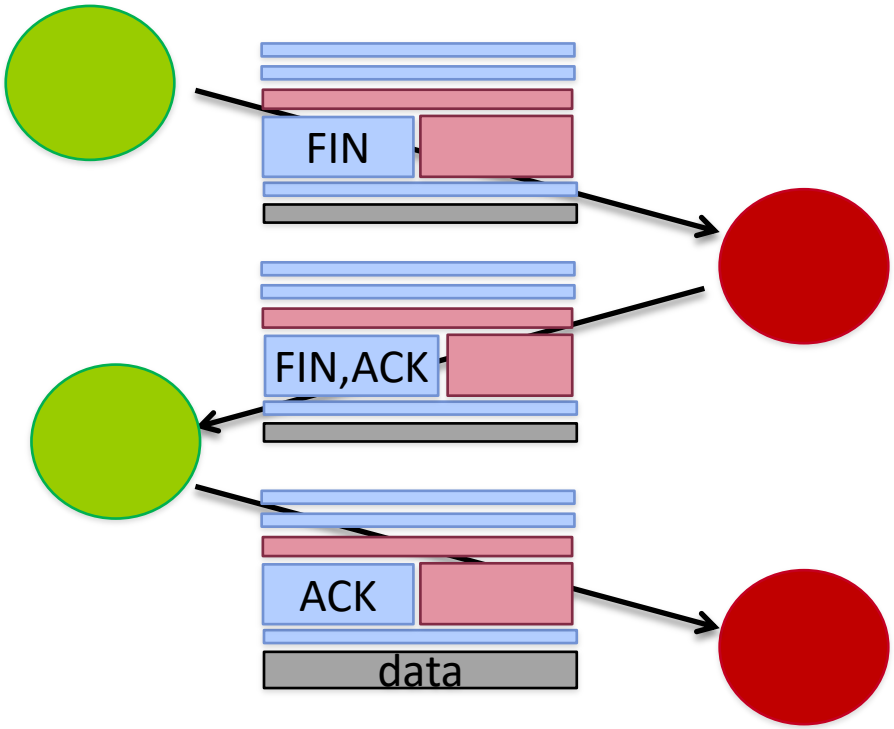
musí být ošetřena řada nestandardních situací (např. ztráta segmentu, přerušení spojení atd.)

v praxi obvykle jako první ukončuje klient



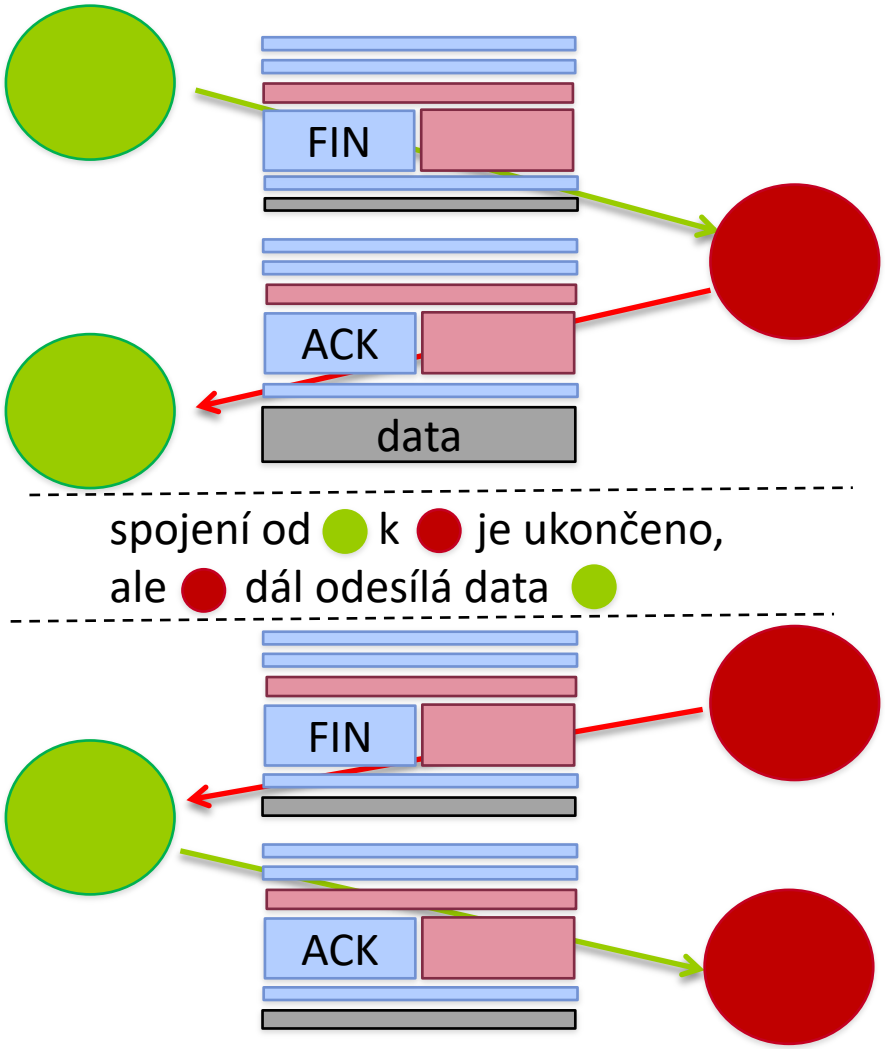
možná ukončení spojení

- nejčastěji jedna strana (klient) ukončuje spojení (odešle FIN) a druhá strana (server) reaguje okamžitě
  - pošle svůj FIN spolu s ACK



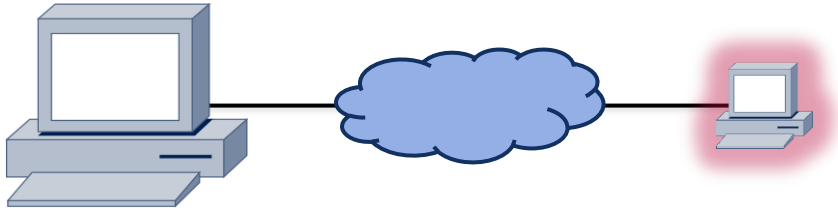
  - fakticky jde o 3-fázový handshake

- ale druhá strana může i nadále posílat data
  - a teprve pak ukončit spojení



řízení toku vs. ochrana před zahlcením

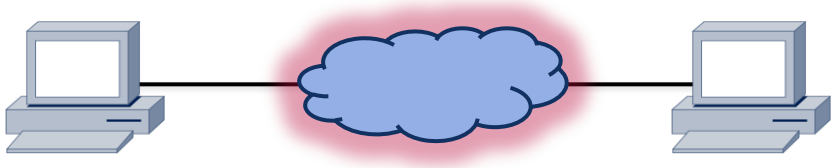
- při řízení toku je „úzkým hrdlem“ příjemce
  - zatímco přenosová síť je dostatečně dimenzovaná



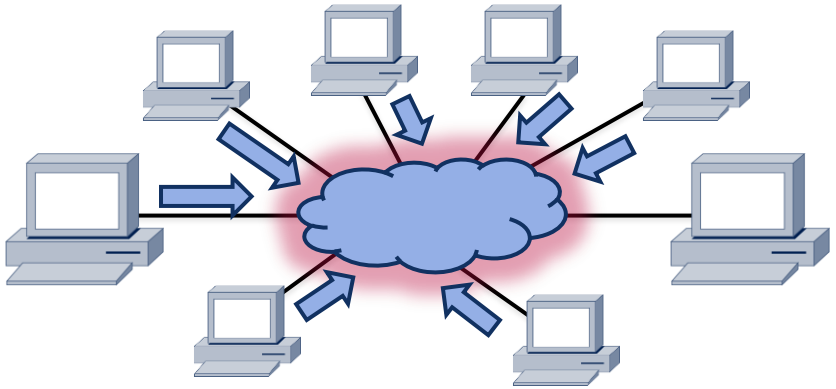
  - TCP řeší skrze metodu okénka
    - příjemce „inzeruje“, kolik dat je schopen přijmout, a tím ovlivňuje velikost okénka

- řízení toku může být realizováno i jinými prostředky
  - a na jiných úrovních
    - např. linkové, fyzické

- při zahlcení je „úzkým hrdlem“ přenosová síť
  - zatímco koncový příjemce je dostatečně dimenzovaný



  - zahlcení může být způsobeno až souběhem více přenosů



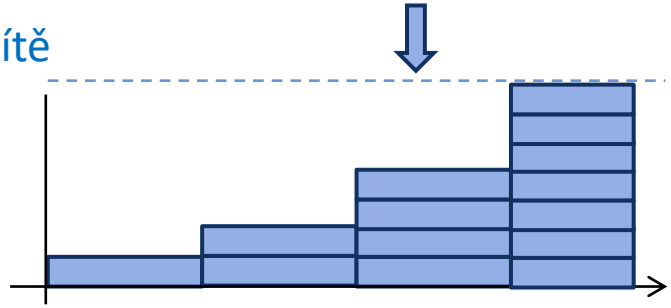
  - TCP ale nemá šanci takovýto souběh rozpoznat
    - nemá podle čeho

TCP a ochrana před zahlcením

- TCP má velmi propracované mechanismy, usilující předcházet zahlcení (congestion control)
  - z počátku: měl jich jen velmi málo
  - postupně přibývají další a další, stále propracovanější
- princip:
  - TCP nemá podle čeho poznat, zda zahlcení způsobil někdo jiný
    - proto to „bere na sebe“ – interpretuje to tak, že zahlcení způsobil on
      - a podniká nápravná opatření
- problém:
  - TCP má jen málo možností, jak poznat, že k zahlcení došlo
    - nemůže/nechce se spoléhat na ICMP Source Quench a podobné mechanismy
  - vychází primárně z absence potvrzení
    - pokud nedostane včas (kladné) potvrzení o doručení, interpretuje to jako zahlcení (přenosové sítě)
      - které způsobil on !!!

TCP slow start

- původně:
  - po úspěšném navázání spojení mohly obě strany ihned začít přenášet data s maximální intenzitou
    - co nejrychleji odeslat všechny segmenty, které se „vešly“ do aktuálního okénka
      - to velmi často způsobilo zahlcení přenosové sítě



- (později nasazené) řešení:
  - používat tzv. pomalý start (slow start)
    - nejprve je odeslán jediný TCP segment
      - vlastně začíná s jednotlivým potvrzováním (stop&wait) místo kontinuálního
    - pokud je včas a kladně potvrzen, mohou být odeslány dva TCP segmenty
    - atd., dokud není dosaženo maxima, které povoluje velikost aktuálního okénka
- další řešení (Congestion Avoidance):
  - kdykoli TCP nedostane včas (kladné) potvrzení odeslaného segmentu, chápe to jako (jím způsobené) zahlcení
    - a reaguje tak, jako kdyby „začínal od začátku“
      - tj. odešle jeden segment
      - a pak aplikuje pomalý start (slow start)