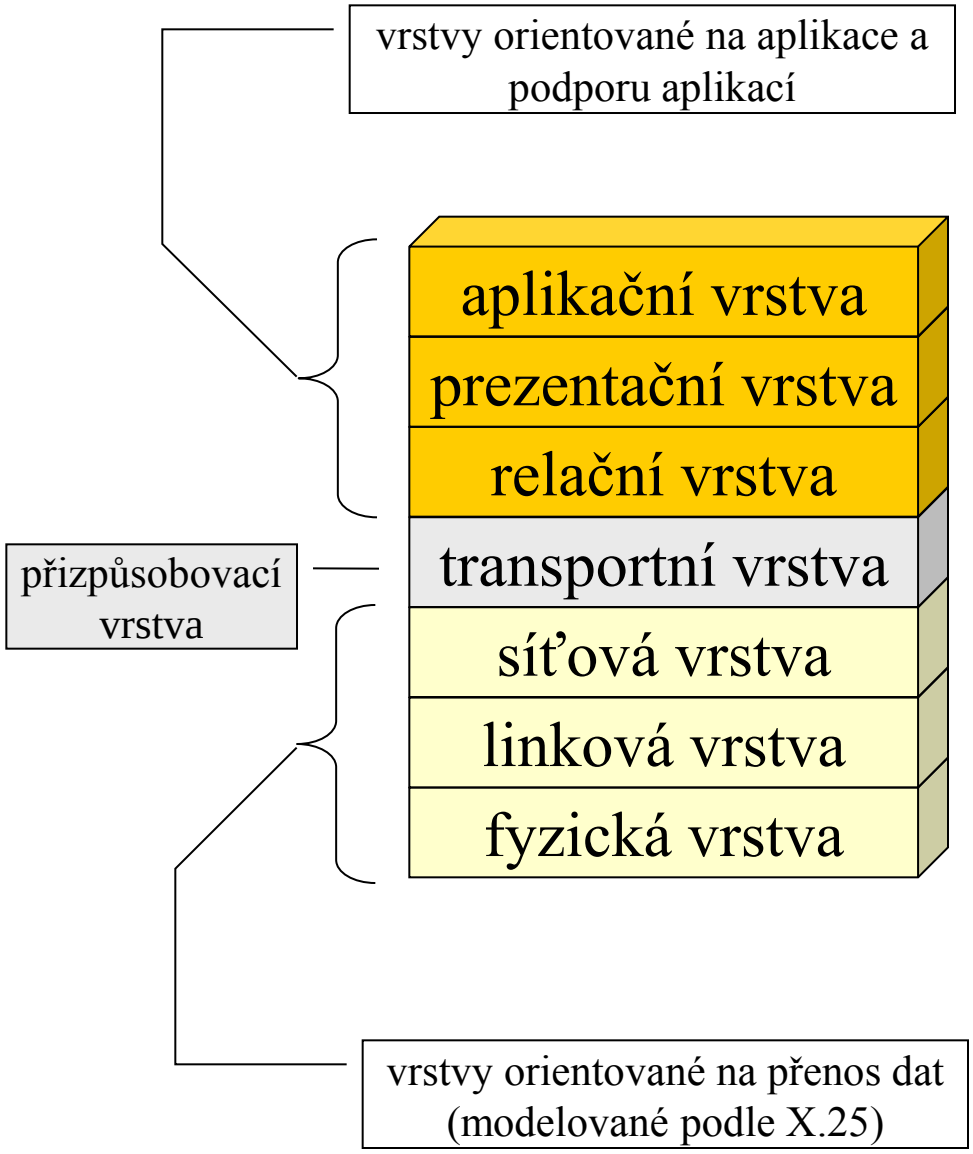
# Počítačové sítě, v. 3.6


Katedra softwarového inženýrství,
Matematicko-fyzikální fakulta,
Univerzita Karlova, Praha


# Lekce 9: Transportní vrstva

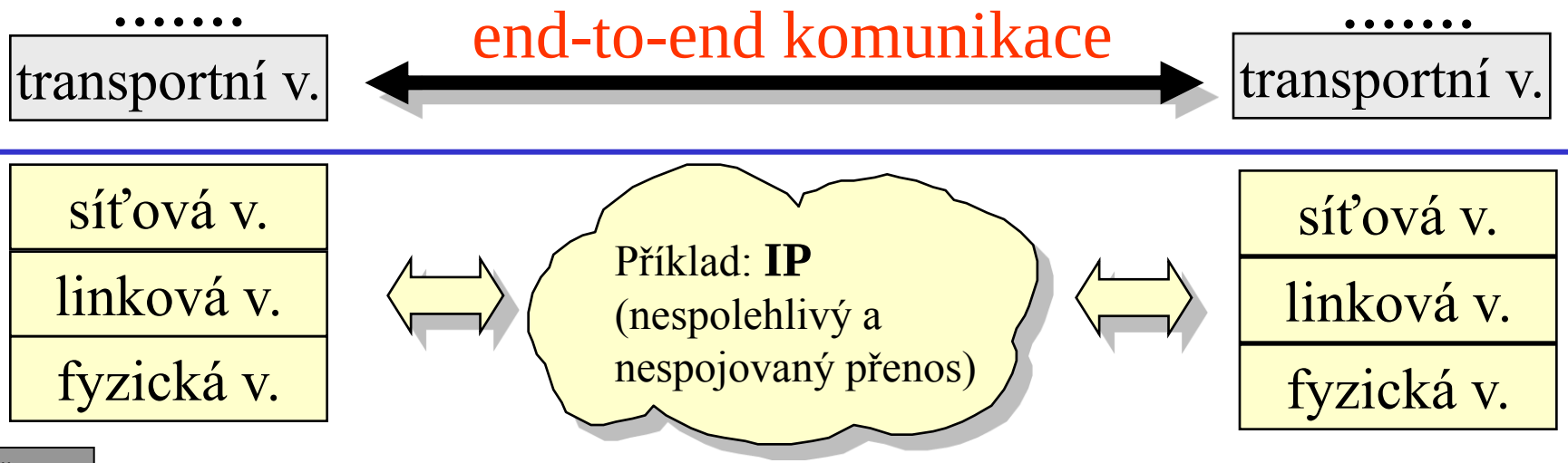
# hlavní úkoly transportní vrstvy

- zajišťovat end-to-end komunikaci
- rozlišovat mezi více odesilateli a příjemci v rámci daného uzlu
  - provádět multiplex a demultiplex
- vyrovnávat rozdíly mezi schopnostmi nižších vrstev a požadavky vyšších vrstev
  - měnit spojovaný/nespojovaný charakter
    - v případě spojované komunikace zajišťovat korektní navazování/ukončování spojení atd.
  - zajišťovat spolehlivost
    - je-li požadována
  - zajišťovat kvalitu služeb
    - je-li požadována
- předcházet/řídit zahlcení a řídit tok



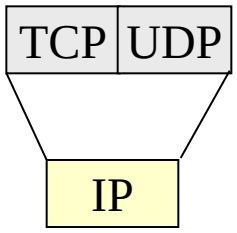
# proč je nutná přizpůsobovací vrstva?

- proč se raději nižší vrstvy nepřizpůsobí požadavkům vyšších vrstev?
  - např. pokud jde o spolehlivost, spojovaný/nespojovaný charakter, kvalitu služeb
- protože to nejde !!!!
  - požadavky různých entit vyšších vrstev (aplikací) mohou být výrazně odlišné,
    - nelze vyhovět všem současně
  - nižší vrstvy (po síťovou včetně) mohou patřit někomu jinému, než uživatelům
    - např. provozovateli veřejné datové sítě
- transportní vrstva není přítomna v přepojovacích uzlech
  - směrovačích, ústřednách apod.
- transportní vrstva je implementována až v koncových uzlech
  - je tudíž plně v moci koncových uživatelů
  - lze snáze měnit (zvyšovat) její schopnosti

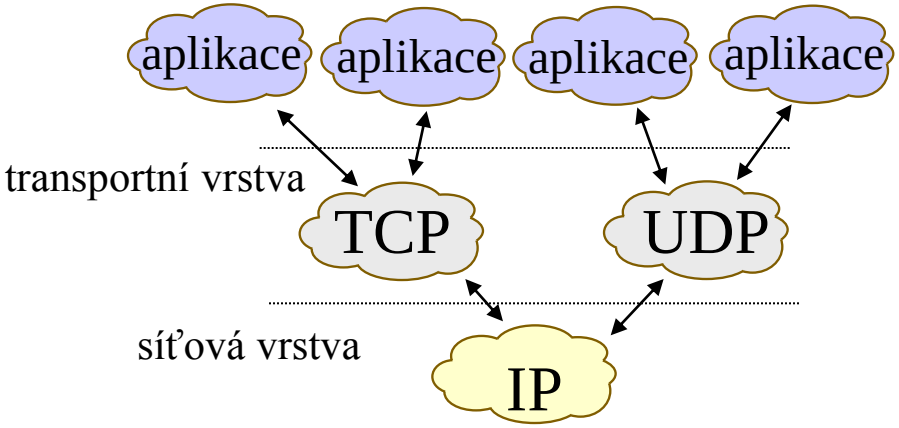


# koncepce transportní vrstvy v TCP/IP

- přenosová část (do síťové vrstvy) funguje „minimalisticky“
  - je zaměřena na rychlost a robustnost, funguje nespolehlivě a nespojovaně
- transportní vrstva je od toho, aby měnila charakter služeb síťové vrstvy
  - ale jen tehdy, když si to vyšší vrstvy přejí !!!!
    - viz: ..... nelze vyhovět všem .....

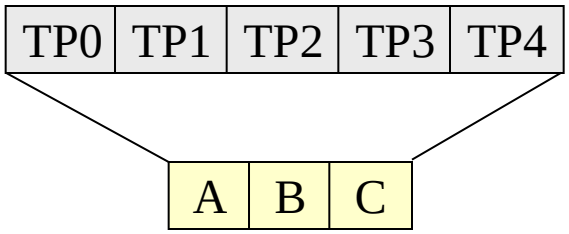



- transportní vrstva TCP/IP obsahuje dva alternativní transportní protokoly:
  - TCP (Transmission Control Protocol), mění charakter přenosových služeb
    - zajišťuje spolehlivost a funguje spojovaně
  - UDP (User Datagram Protocol), nemění charakter přenosových služeb
    - funguje nespolehlivě a nespojovaně
- vyšší vrstvy si mohou samy vybrat, který transportní protokol využijí



# koncepce transportní vrstvy v RM ISO/OSI

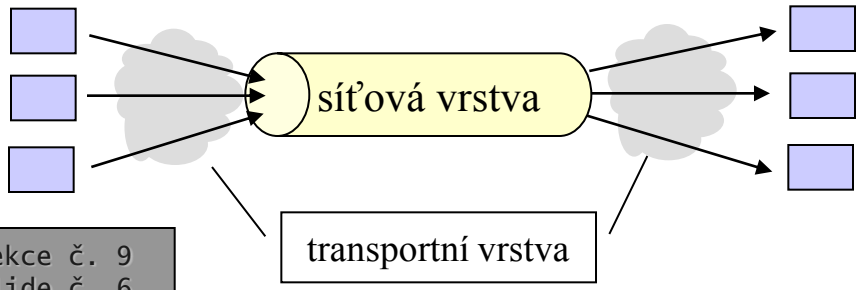
- přenosová část sítě (do síťové vrstvy) funguje spíše „maximalisticky“
  - spojovaně
  - spolehlivě
- počítá s existencí 3 různě kvalitních služeb na úrovní síťové vrstvy:
  - kategorie A: žádné ztráty paketů, žádné výpadky spojení
    - např. lokální sítě
  - kategorie B: žádné ztráty, občas výpadky spojení
    - veřejné datové sítě
  - kategorie C: občas ztráty, občas výpadky spojení
    - rozlehlé sítě



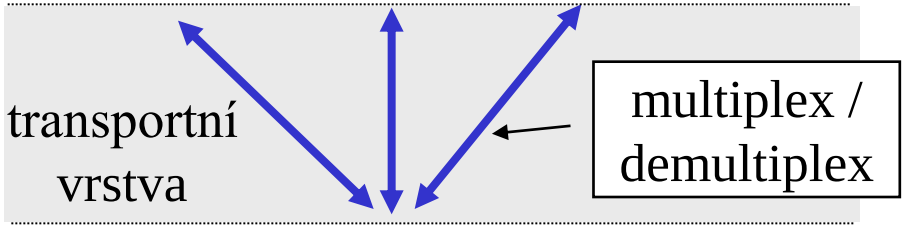

- transportní vrstva je od toho, aby „*pokračovala ve stejném duchu*“
  - teprve dodatečně se do ISO/OSI prosadily i nespolehlivé a nespojované služby
    - a na ně pak reflektuje i koncepce transportní vrstvy
- definuje 5 tříd transportních protokolů
  - **TP0:**
    - jednoduchý obal nad síťovými službami kategorie A
  - **TP1:**
    - nad B, řeší výpadky
  - **TP2:**
    - nad A, dokáže využít jedno síťové spojení pro více transportních
  - **TP3:**
    - nad B, více transportních spojení nad jedním síťovým
  - **TP4**:
    - transportní služba nad síťovými službami kategorie C (nespolehlivými, s výpadky)
      - zajišťuje spolehlivost

# další úkol transportní vrstvy

- síťová vrstva:
  - dívá se na každý uzel jako celek
    - síťové adresy (např. IP adresy) označují uzly jako takové
      - přesněji: jejich rozhraní
    - nerozlišují konkrétní entity v rámci uzlů
      - zejména na aplikační úrovni
- transportní vrstva:
  - má za úkol rozlišovat různé entity (procesy, úlohy, démony, …) na úrovni vyšších vrstev
  - musí provádět tzv. **multiplex**
    - "sběr" dat od více entit vyšších vrstev a jejich další přenos
  - a tzv. **demultiplex**
    - rozdělování přijatých dat mezi různé entity vyšších vrstev



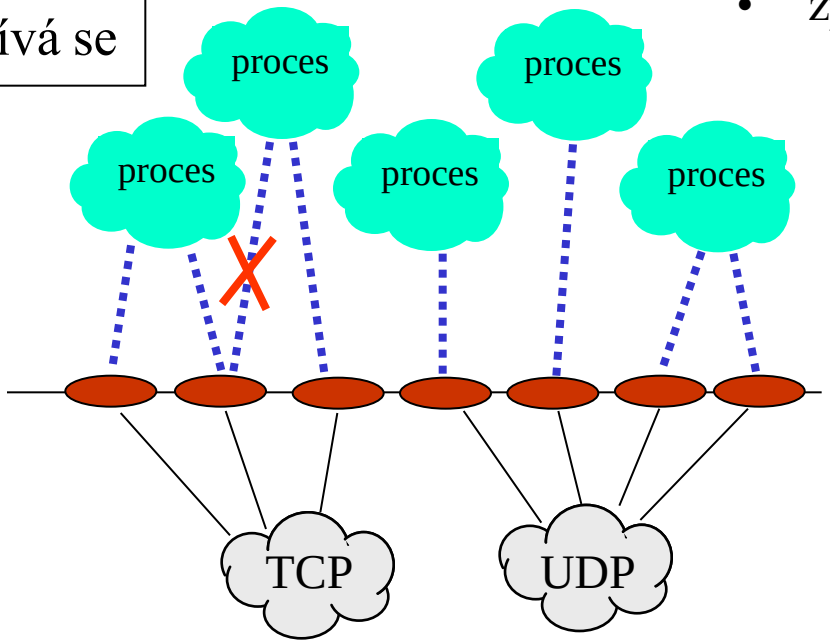





# jak identifikovat odesilatele/příjemce? (pro potřeby multiplexu a demultiplexu)

- idea:
  - dávat entitám (procesům …) identifikátory, a ty pak zahrnout do transportních adres
    - tj. adresovat přímo jednotlivé procesy
  - problém:
    - procesy vznikají dynamicky, a dynamicky také zanikají
      - jak jim přidělovat identifikátory
      - jak oznamovat okolním uzlům, zda ten který proces právě běží (existuje) či nikoli

nepoužívá se



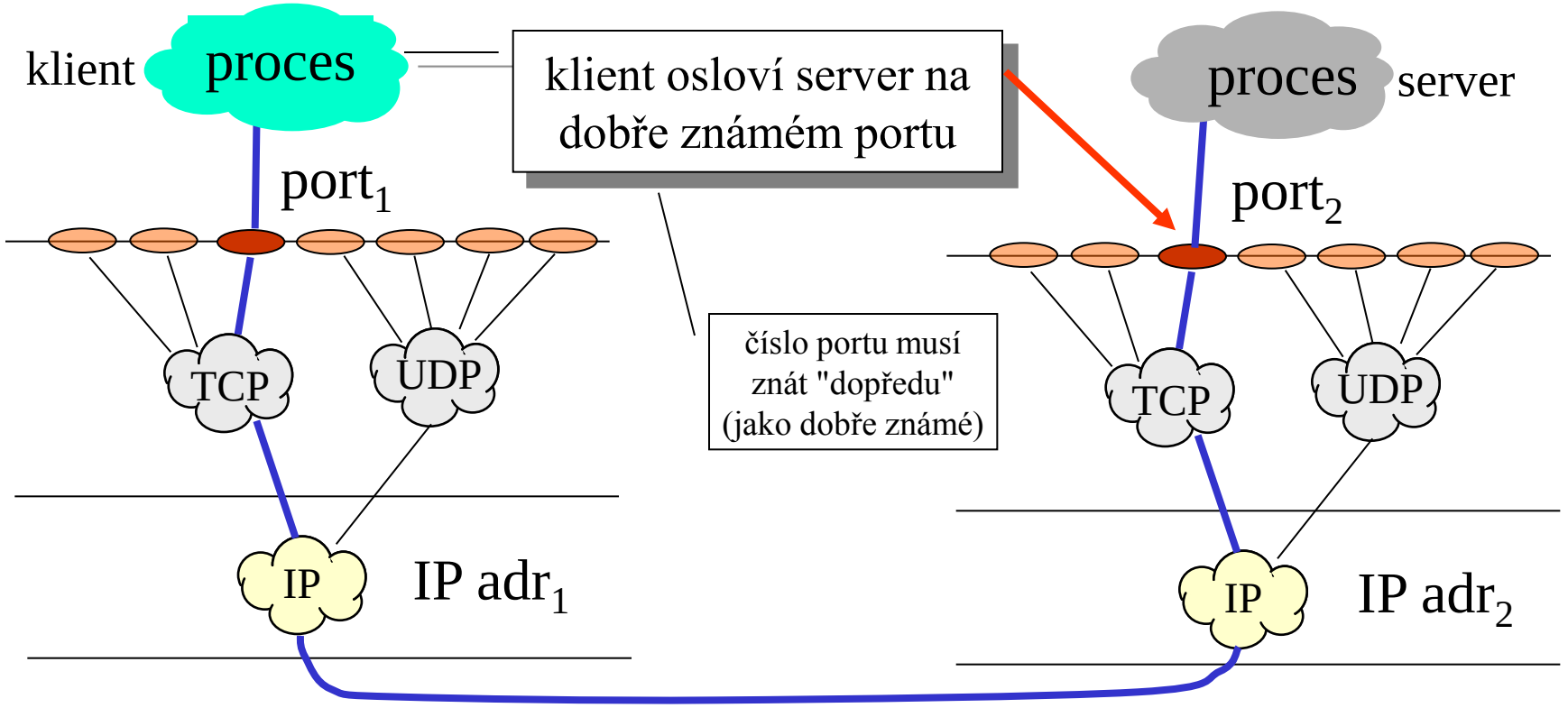

- idea:
  - neidentifikovat přímo entity vyšších vrstev, ale
  - přechodové body mezi transportní a vyšší vrstvou
- terminologie:
  - ISO/OSI: přechodový bod = bod SAP (Service Access Point)
  - TCP/IP: přechodový bod = port
- způsob fungování:
  - přechodové body (porty) mají statický charakter
    - a tudíž se snáze identifikují
    - v TCP/IP: pořadovými čísly (čísly portů)
  - přechodové body (porty) existují nezávisle na entitách vyšších vrstev
    - entity se dynamicky asociují (přidružují, "bindují") k přechodovým bodům
    - jedna entita může být asociována s více porty
    - s jedním portem NESMÍ být asociováno více entit

# porty v TCP/IP

- porty jsou identifikovány (celými, kladnými) čísly
  - číslo portu představuje relativní adresu v rámci uzlu
- význam portu:
  - je to adresa, na které je poskytována určitá služba
    - "druhé straně" může být jedno, který konkrétní proces službu poskytuje, důležitá je služba jako taková
- otázka:
  - *jak se "druhá strana" dozví o tom, na jaké adrese (čísle portu) je poskytována služba, kterou požaduje? Na který port se má obrátit?*
    - *např. na kterém portu poskytuje WWW server své stránky?*
- řešení:
  - význam některých portů je fixován
    - je apriorně dán (přidělen), přiděluje IANA (ICANN)
    - je to všeobecně známo
      - dříve se zveřejňovalo v RFC (naposledy RFC 1700)
      - dnes pouze on-line, na adrese **http://www.iana.org/numbers.html**
  - jde o tzv. dobře známé porty (well-known ports)
    - jsou to porty 0-1023
    - **smysl: na těchto portech jsou poskytovány obvyklé služby**
  - existují též tzv. registrované porty (Registered)
    - porty 1024-49151
    - IANA nepřiděluje, pouze registruje jejich použití
  - ostatní porty (Dynamic, Private)
    - 49152 - 65535
    - jsou používány volně, nejsou ani registrovány

| Port # | Popis |
|---|---|
| 21 | FTP |
| 23 | Telnet |
| 25 | SMTP |
| 69 | TFTP |
| 70 | Gopher |
| 80 | HTTP |
| 88 | Kerberos |
| 110 | POP3 |
| 119 | NNTP |
| 143 | IMAP |
| 161 | SNMP |

# představa (aplikačního) spojení

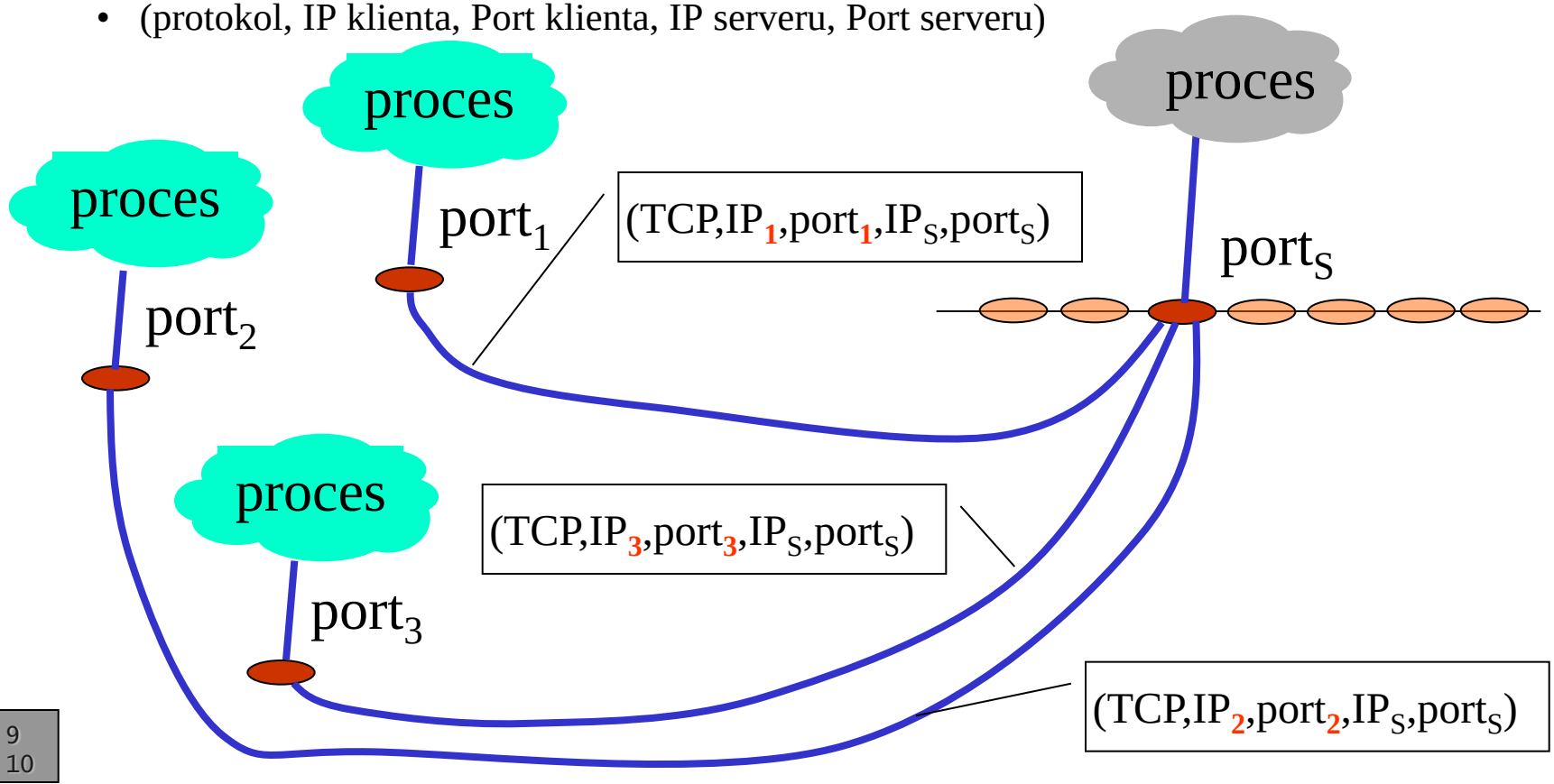



- port je "logickým zakončením" spojení (entita/proces je "fyzickým zakončením")
- (aplikační) spojení je jednoznačně určeno pěticí

(transportní protokol, IP $adr_1$, $port_1$, IP $adr_2$, $port_2$)

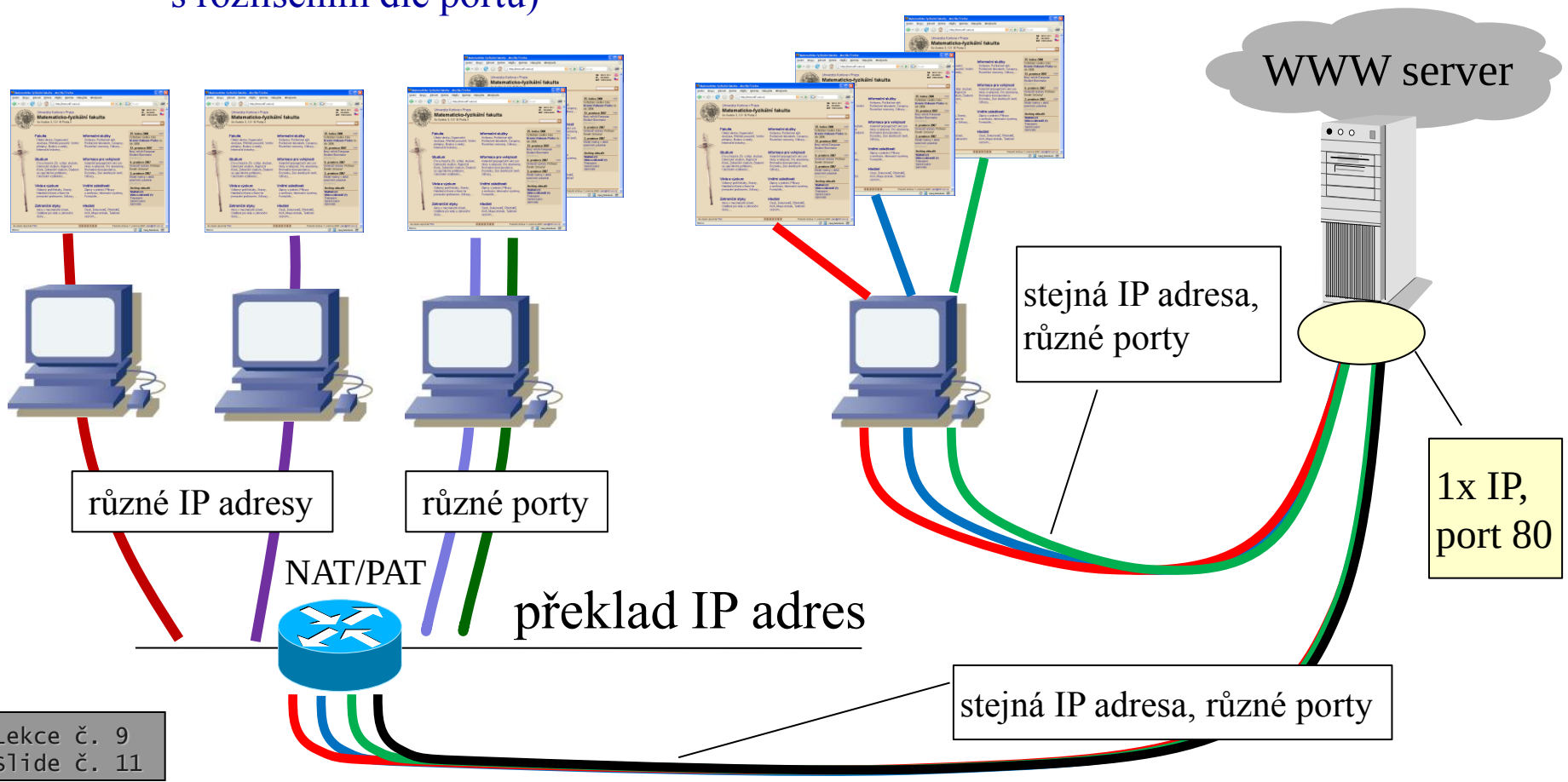
# představa (aplikačního) spojení

- jedna entita (proces, … ) může komunikovat s více entitami na jiných uzlech přes stejný port
  - jeden server může přijímat požadavky od různých klientů na stejném portu
  - více klientů může navázat spojení se stejným serverem na stejném portu
  - rozlišení umožňuje adresa a port klienta, v rámci pětice
    - (protokol, IP klienta, Port klienta, IP serveru, Port serveru)



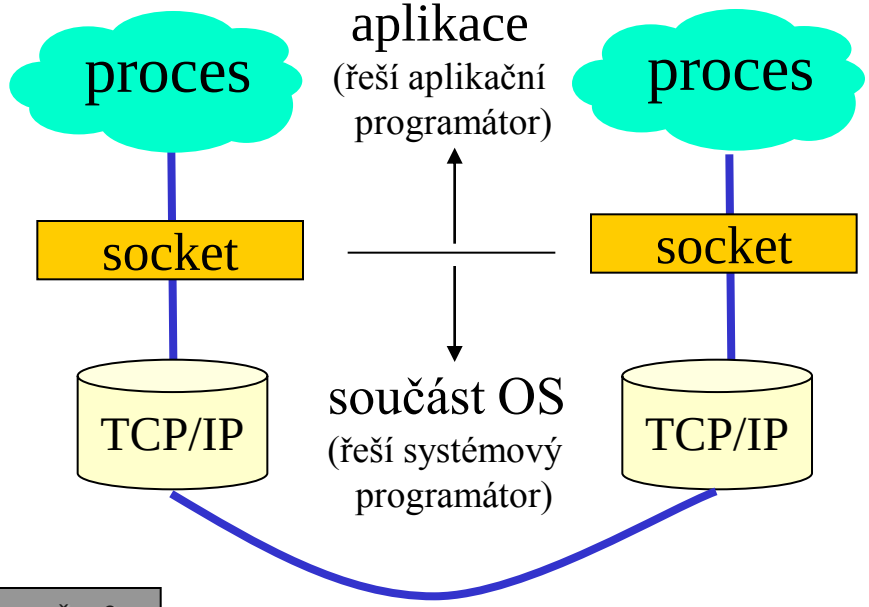
# představa (aplikačního) spojení

- jedna entita (proces, … ) může komunikovat s více entitami na stejném uzlu, i na různých uzlech za jedním NAT/PAT uzlem
  - více entit na stejném uzlu: když si uživatel spustí více instancí browseru (například)
  - více entit za uzlem NAT/PAT (který zajišťuje překlad více různých adres do jedné, s rozlišením dle portů)



# porty vs. sockety

- porty jsou logickou záležitostí
  - na všech platformách jsou stejné
    - identifikované svými čísly
  - jejich konkrétní implementace je závislá na platformě
- aplikace (entity aplikační vrstvy) obvykle pracují s porty skrze API
  - API může být součástí operačního systému, nebo může mít formu knihoven linkovaných k aplikaci




- podstatný je také "styl" práce s porty
  - dnes převažuje "styl" (paradigma) zavedený v BSD Unixu (verze 4.2), založený na tzv. socketech
- socket vznikl jako abstrakce souboru v BSD Unixu
  - pro potřeby práce se soubory (a také pro vstupu a výstupy)
  - pracuje se s ním style "open-read-write-close"
- sockety byly použity i pro potřeby síťování
  - byly rozšířeny o další možnosti/operace
- "socketové API"
  - takové API, které procesům vytváří iluzi že pracují se sockety
    - např. rozhraní WINSOCK
- socket si lze představit jako analogii brány
  - vedoucí k síťovým službám

socket =

# práce se sockety

- sockety existují nezávisle na portech
- vznik socketu:
  - voláním funkce pro vytvoření nového socketu: SOCKET (…)
    - parametry:
      - rodina protokolů (TCP/IP)
      - typ služby (STREAM, DATAGRAM, RAW)
      - protocol (TCP, UDP)
  - nově vytvořený socket není asociován (sdružen) s žádným portem
    - **vzniká "sám o sobě"**
- asociování socketu s konkrétním portem
  - voláním funkce: BIND (…)
- po skončení práce se socketem je nutné jej zavřít/zrušit
  - CLOSE(...)

- se sockety lze provádět další "primitivní operace"
  - SENDTO(socket,data,…,adresa ….)
    - pošle data zadanému příjemci
    - určeno pro nespojovaný způsob komunikace, bez navazování spojení
  - RECVFROM(socket,…,adresa, …)
    - přijme data nespojovaným způsobem

příklad: činnost serveru při nespojované komunikaci:

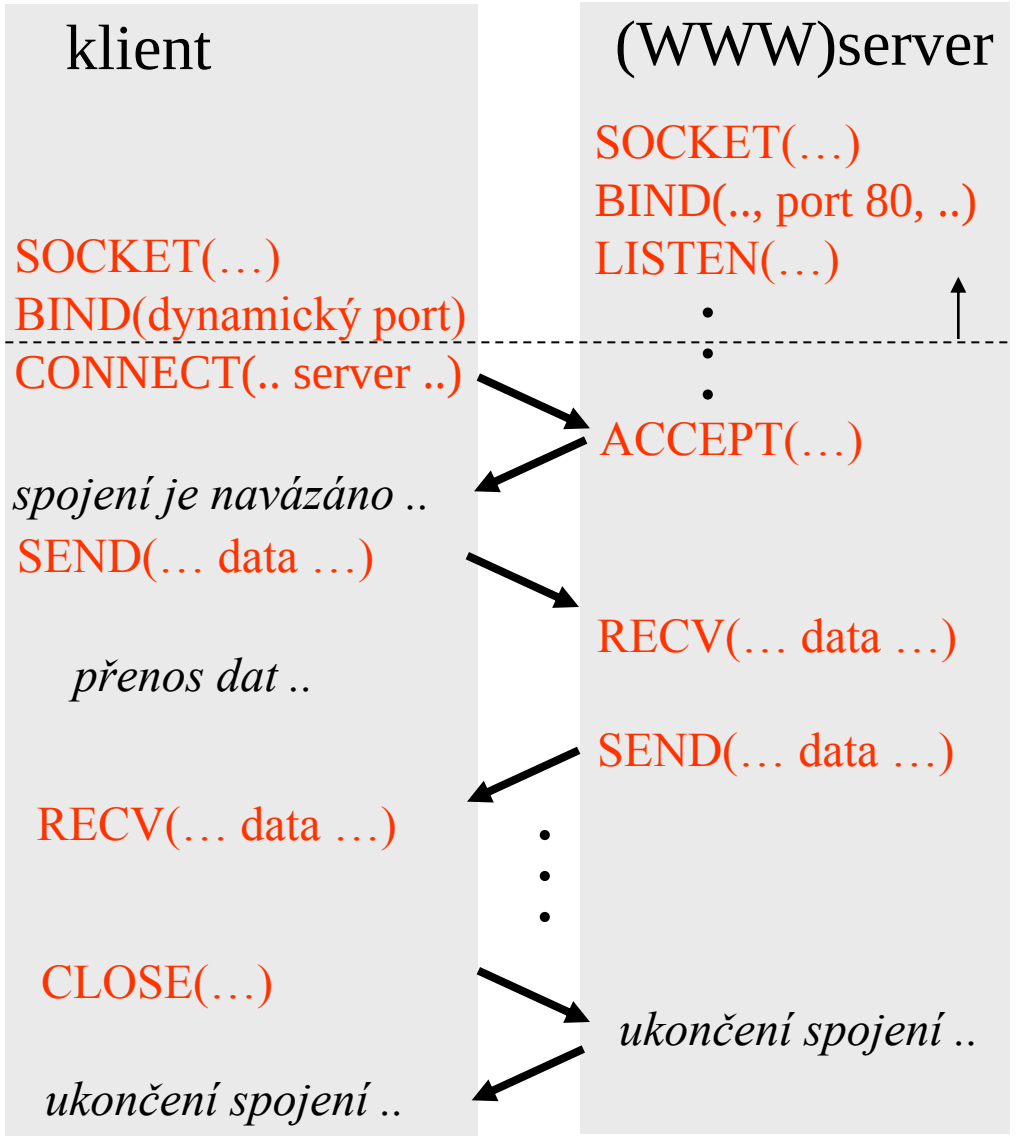
```
newsock=SOCKET(…);
BIND(newsock,číslo portu);

repeat
    RECVFROM(newsock, adresa..);
    zpracování požadavku z "adresa"
until ….
```

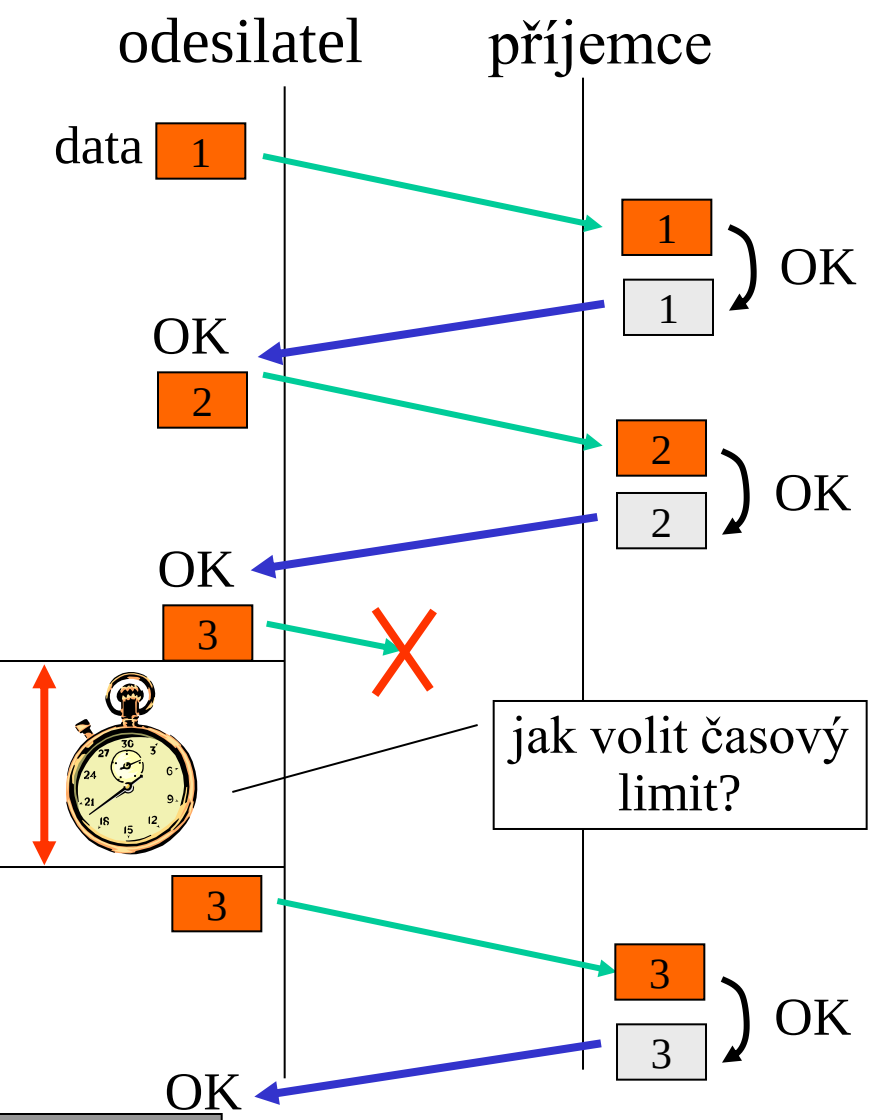
# práce se sockety – spojovaná komunikace

- primitivní operace, určené pro spojovanou komunikaci
  - LISTEN(socket, …)
    - server chce přijímat požadavky z určitého socketu – počáteční akce, čekání na žádost o navázání spojení
  - ACCEPT(socket, ….)
    - přijetí požadavku na navázání spojení (na straně serveru)
  - CONNECT(socket,adresa_serveru …)
    - požadavek (klienta) na navázání spojení (ze zadaného socketu) se serverem na zadané adrese (IP adresa, port)
  - SEND(socket,data, ….)
    - pošle data skrz navázané spojení
  - RECV(socket,buffer,délka,flags)
    - pro příjem ze zadaného socketu při navázaném spojení

představa průběhu spojované komunikace



# zajištění spolehlivosti



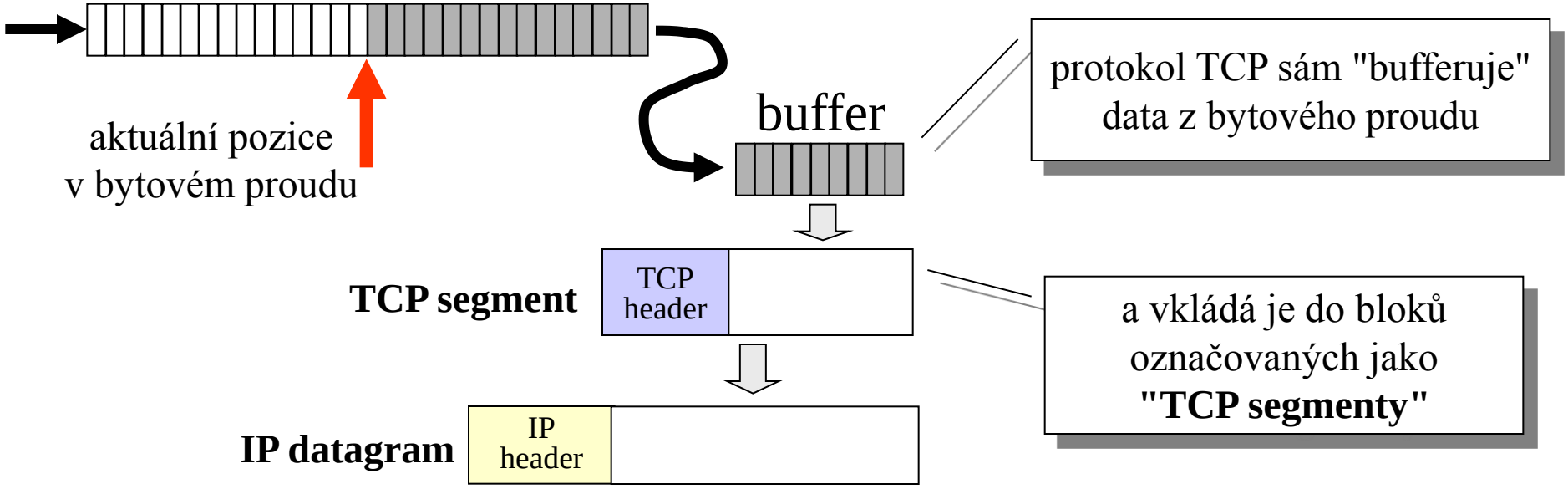

- transportní protokol může zajišťovat spolehlivost přenosu
  - pokud je to po něm požadováno
  - TCP/IP: po TCP se požaduje, po UDP nikoli
- způsob realizace:
  - přes potvrzování, využívá se zpětná vazba
  - v úvahu připadá jednotlivé i kontinuální potvrzování
- v lokálních sítích:
  - přenosové zpoždění je malé, lze použít jednotlivé potvrzování (stop&wait)
- v rozlehlých sítích:
  - je přenosové zpoždění velké, je nutné použít kontinuální potvrzování
- TCP:
  - používá kontinuální potvrzování
    - se selektivním opakováním !!!
  - příjemce generuje kladná potvrzení

# volba timeoutu (TCP)

- otázka:
  - jak volit dobu, po kterou má odesilatel čekat na potvrzení?
    - velikost timeoutu
  - když bude příliš velká
    - bude čekat zbytečně – špatné
  - když bude příliš malá
    - "zpanikaří" předčasně, začne podnikat nápravné akce zbytečně
  - velikost timeoutu výrazně ovlivňuje efektivnost protokolu, který zajišťuje spolehlivý přenos
- záměry protokolu TCP:
  - snaha přizpůsobit se různým podmínkám
    - rozlehlé sítě a lokální sítě, různě dlouhá doba obrátky (RTT, Round Trip Time)
  - snaha přizpůsobit se měnícím se podmínkách
    - doba obrátky se může dynamicky měnit, podle zátěže sítě atd.
  - dělá to dobře = je efektivní v LAN i WAN
    - ale je kvůli tomu i značně komplikovaný
- princip:
  - TCP průběžně monitoruje chování sítě, a podle něj mění délku časového intervalu, po který čeká na potvrzení
- ve skutečnosti:
  - TCP monitoruje "dobu obrátky"
    - nepozná přenosové zpoždění, ale sleduje za jak dlouho dostává odpovědi
  - TCP vyhodnocuje:
    - vážený průměr dob obrátky (RTT)
    - rozptyl dob obrátky
  - "čekací dobu" TCP vypočítává jako funkci váženého průměru a rozptylu
- výsledný efekt:
  - "čekací doba" vychází "těsně nad" střední dobou obrátky
    - je-li doba obrátky konstantní, čekací doba se jí více přibližuje
    - jakmile se doba obrátky začíná měnit, čekací doba se zvětšuje
  - dobře to reaguje na:
    - prodlužování doby obrátky při "dávkách paketů"
    - zkrácení doby obrátky po odeslání dávky paketů

potvrzování je nesamostatné, vkládá se do paketů cestujících opačným směrem (tzv. piggybacking)

# bytový proud v TCP



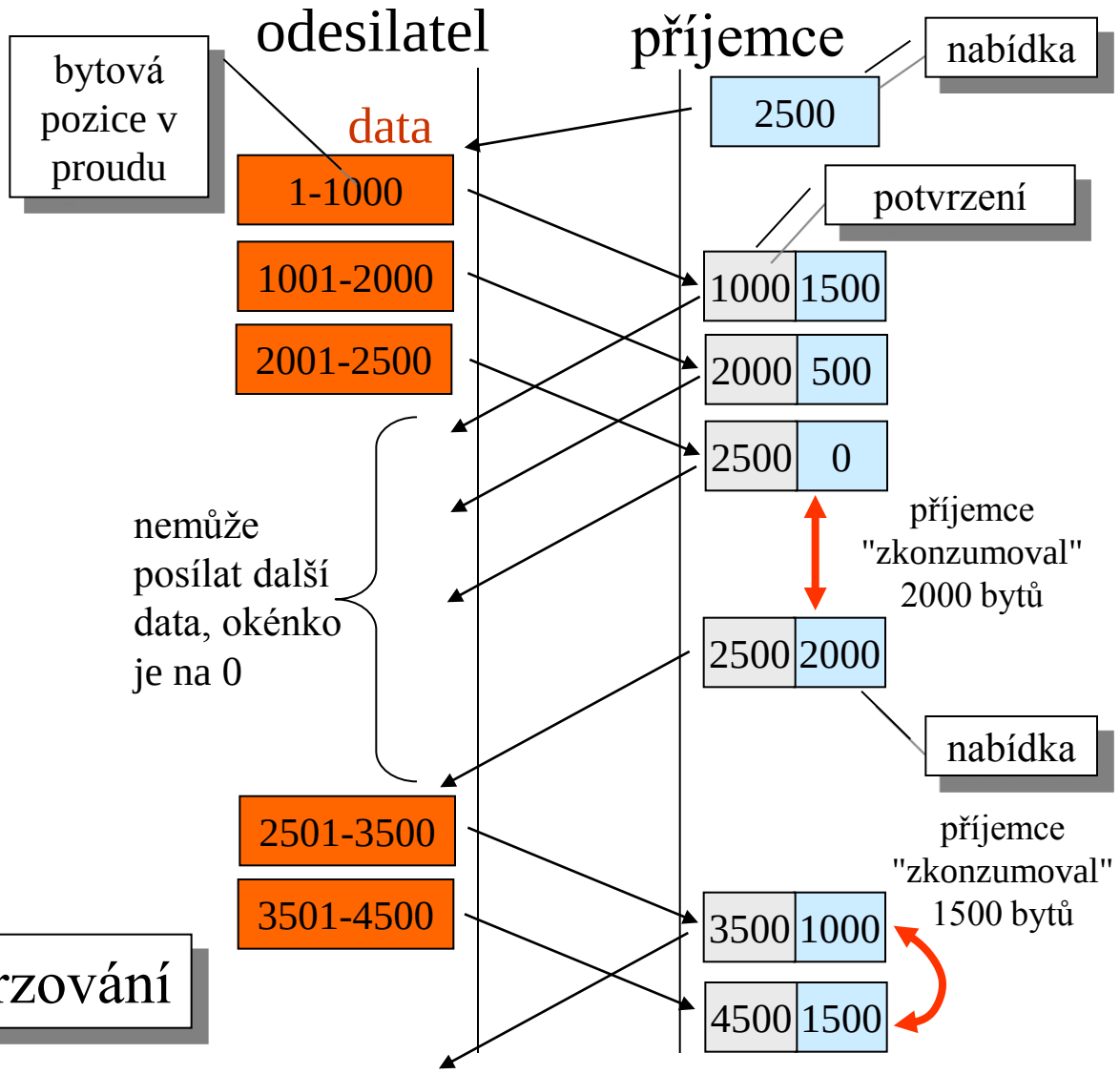

- protokol TCP přijímá/vydává data po jednotlivých bytech (pracuje s bytovým proudem, byte stream)
  - ve skutečnosti data bufferuje (ukládá do bufferu, jehož velikost volí podle parametru MTU)
- obsah bufferu je odesílán až po jeho naplnění
  - aplikace má možnost vyžádat okamžité odeslání obsahu bufferu (operace PUSH)

protokol UDP přijímá data po celých blocích, nikoli po bytech!!!

- TCP potřebuje označovat jednotlivé byty v rámci proudu
  - jelikož nepracuje s bloky
  - potřebuje to například pro potvrzování
    - aby vyjádřil "kam až" se data přenesla
  - používá k tomu (32-bitovou) pozici v bytovém proudu
    - začíná od náhodně zvoleného čísla

# buffery a řízení toku (TCP)

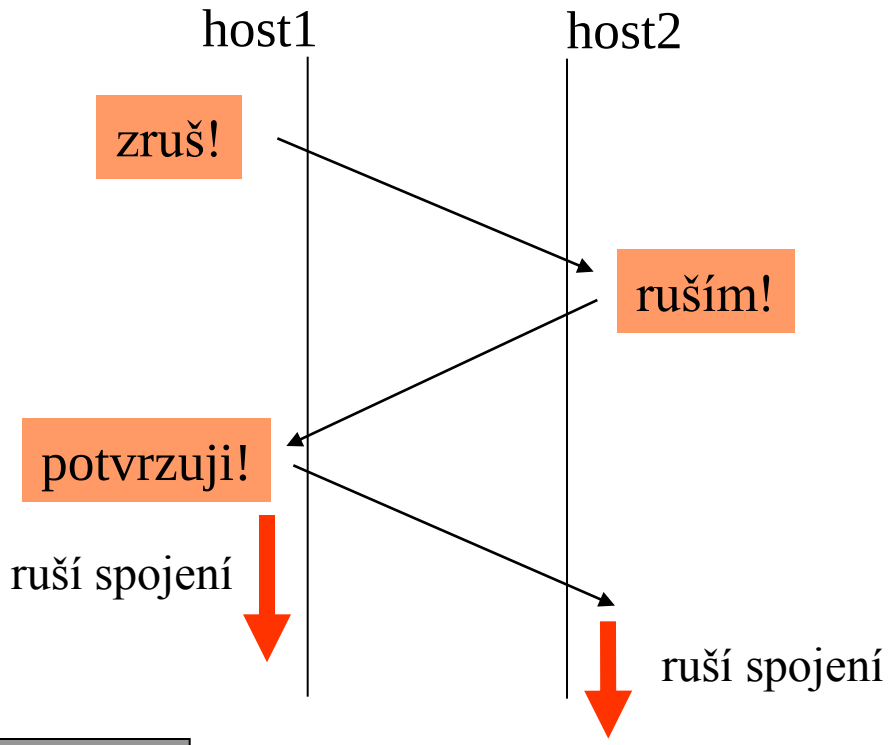
- TCP se snaží řídit tok dat
  - aby odesilatel nezahlcoval příjemce a kvůli tomu nedocházelo ke ztrátám dat
- podstata řešení:
  - používá se metoda okénka
    - okénko udává velikost volných bufferů na straně příjemce
    - odesilatel může posílat data do "zaplnění" okénka
    - příjemce spolu s každým potvrzením posílá také svou "nabídku" (údaj o velikosti okénka, window advertisement)
      - tím říká, kolik dat je ještě schopen přijmout (navíc k právě potvrzovaným)



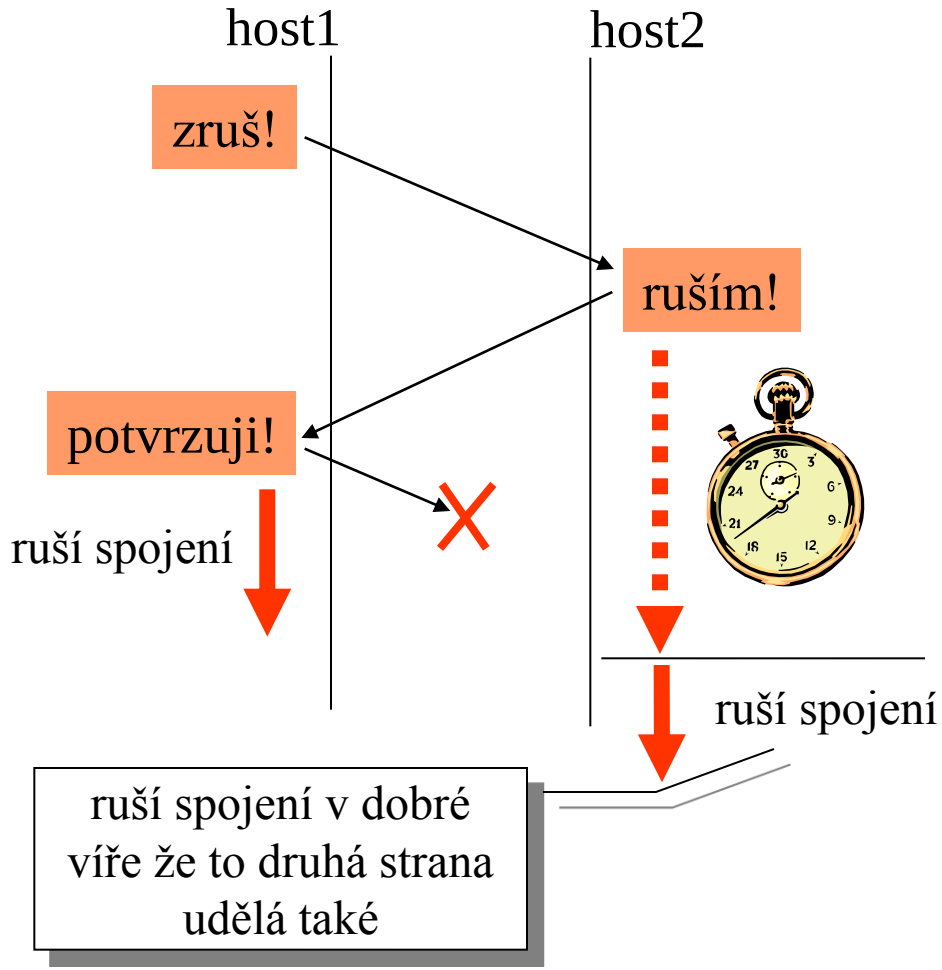

používá kontinuální potvrzování

# navazování a rušení spojení

- korektní navázání i rušení spojení je hodně náročné
  - žádosti (i potvrzení) se mohou různě ztrácet, bez náhrady
  - mohou vznikat duplicitní žádosti i potvrzení
- TCP to řeší pomocí 3-fázového handshake
  - který ošetřuje většinu nestandardních situací

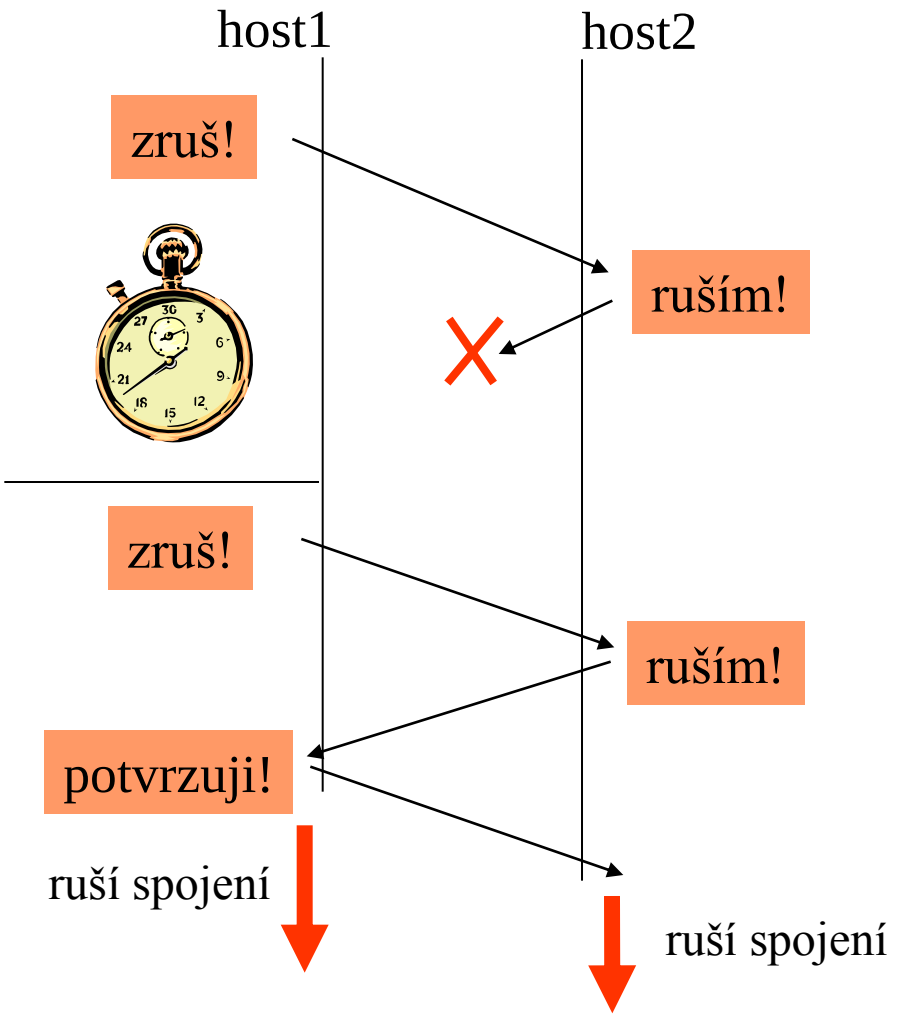


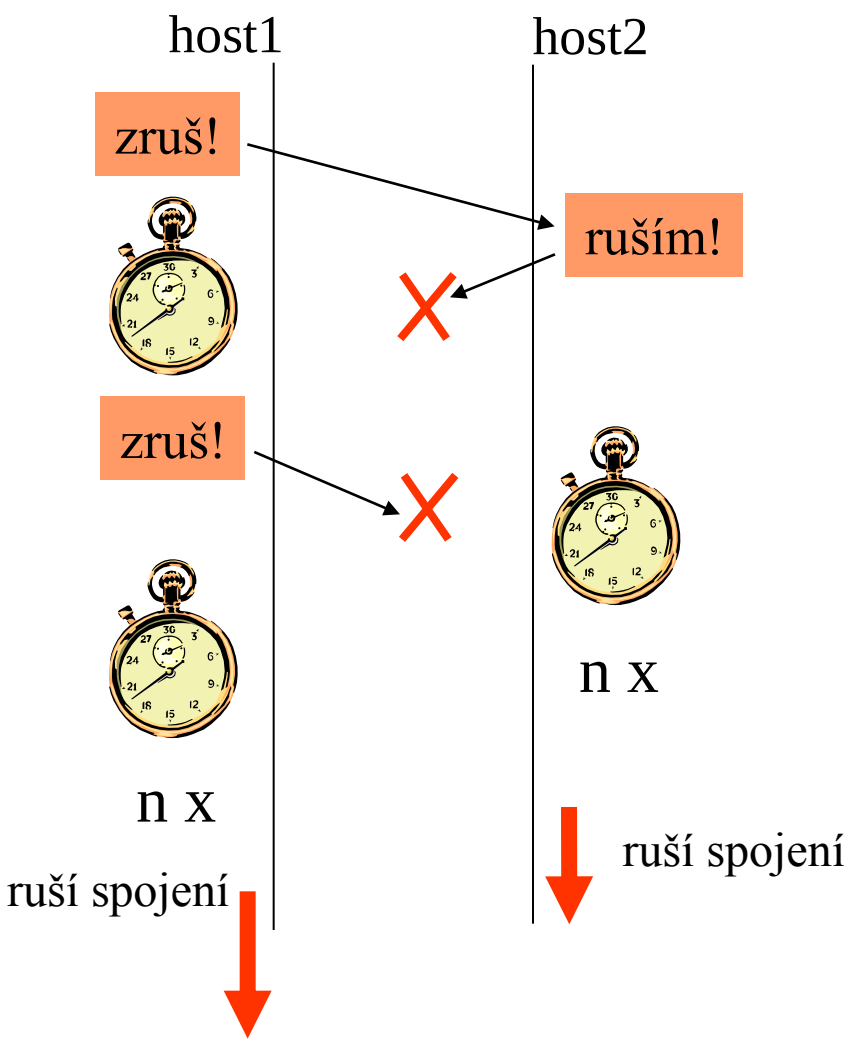

ideální případ




ztratilo se jedno z potvrzení

# navazování a rušení spojení



# ochrana před zahlcením

- připomenutí:
  - řízení toku (flow control) se týká jedné komunikující dvojice (odesilatel - příjemce)
  - ochrana před zahlcením (congestion control) se týká sítě jako takové
    - "součtu" datových toků, které se v určitém místě schází
  - lze řešit i na nižších vrstvách
    - linkové, síťové
    - nebo na transportní vrstvě
    - i na aplikační
- pozorování:
  - většina ztrát přenášených dat jde spíše na vrub zahlcení než chybám HW
  - transportní protokoly mohou nevhodným chováním důsledky ještě zhoršovat
    - tím že se snaží odesílat další data
- přístup TCP
  - každou ztrátu dat chápe jako důsledek zahlcení
    - a nasazuje protiopatření (congestion control)
  - po ztrátě paketu jej pošle znovu, ale neposílá další a čeká na potvrzení
    - tj. **přechází z kontinuálního potvrzování na jednotlivé !!**
      - nevysílá tolik dat, kolik mu umožňuje okénko!!
  - přijde-li potvrzení včas, odešle dvojnásobek dat a čeká na potvrzení
    - odešle dva pakety
  - takto postupuje dokud nenarazí na omezení dané aktuální velikostí okénka
    - postupně se tak vrací na kontinuální potvrzování

# QoS - zajištění kvality služeb

- pozorování:
  - různé druhy přenosů mají různé nároky
  - ale standardní způsob fungování přenosových sítí (best effort) měří všem stejně!!
- řešení:
  - žádné / "hrubou silou"
    - zvýší se přenosová i další kapacita, tak aby k problémům nedocházelo tak často
  - podpora kvality služeb (QoS, Quality of Service)
    - s různými druhy přenosů bude "nakládáno různě"
- možnosti implementace QoS:
  - přímo na úrovni síťové vrstvy
    - kde jinak vzniká "best effort"
    - např. protokol IP má v hlavičce položku (ToS) pro vyjádření požadavků paketu na QoS – ale standardně se ignoruje
  - "předpoklady" na síťové vrstvě, hlavní část řešení na transportní vrstvě
    - příklad: na úrovni síťové vrstvy se rezervují určité kapacity, které se pak využívají na úrovni transportní vrstvy
  - řešení na transportní nebo aplikační vrstvě
    - bez "předpokladů" na síťové vrstvě
- možné přístupy a techniky
  - traffic conditioning,
    - úpravy datového provozu tak, aby "lépe prošel"
  - "Integrated Services"
    - zásadnější změny v přenosové části sítě, tak aby bylo možné rezervovat potřebné zdroje a dát "pevné záruky"
  - "Differentiated Services"
    - menší zásahy do přenosové části sítě, snaha diferencovat provoz a poskytnout alespoň "záruku rozdílu"

# QoS – požadavky aplikací

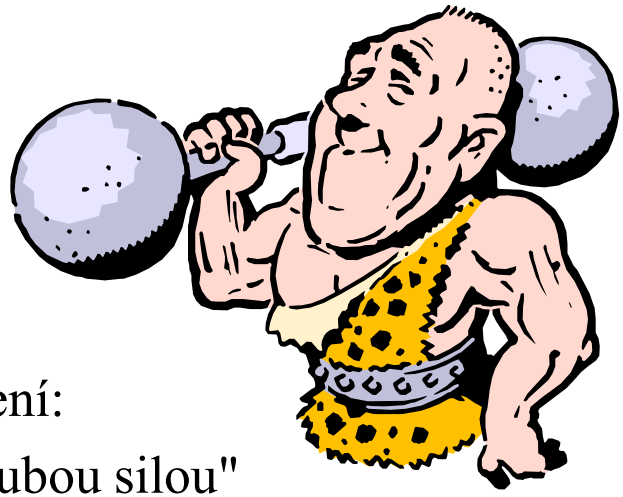
- problém multimediálních aplikací:
  - potřebují dostávat svá data
    - **včas** (s malým přenosovým zpožděním - latence)
    - **pravidelně** (s rovnoměrnými odstupy mezi jednotlivými částmi - jitter)
  - příklad: přenos hlasu (latence)
    - business kvalita: max zpoždění 150 milisekund
    - při 250 msec. zpoždění znatelná a vadí
    - nad 500 msec. nepoužitelné pro přenos hlasu
  - příklad: přenos obrazu (jitter)
    - nepravidelnost lze přirovnat k nerovnoměrné rychlosti posunu filmového pásu při promítání
      - když jsou nerovnoměrnosti moc velké, nelze se na to dívat

| | Požadavek na | | | |
|---|---|---|---|---|
| | **spolehlivost** | **nízké zpoždění (latence)** | **pravidelnost (nízký jitter)** | **přenosovou kapacitu** |
| **email** | Max. | Min. | Min. | Min. |
| **přenos souborů** | Max. | Min. | Min. | Medium |
| **www** | Max. | Medium | Min. | Medium |
| **remote login** | Max. | Medium | Medium | Min. |
| **Audio on demand** | Min. | Min. | Max. | Medium |
| **Video on demand** | Min. | Min. | Max. | Max. |
| **IP telefonie** | Min. | Max. | Max. | Min. |
| **Videokonference** | Min. | Max. | Max. | Max. |

- spolehlivost:
  - řada multimediálních aplikací na spolehlivosti netrvá
    - a dávají přednost nízké latenci a pravidelnosti doručování
  - například srozumitelnosti hlasu (zásadněji) nevadí ani ztráta či poškození 20% dat

# transportní vrstva a QoS

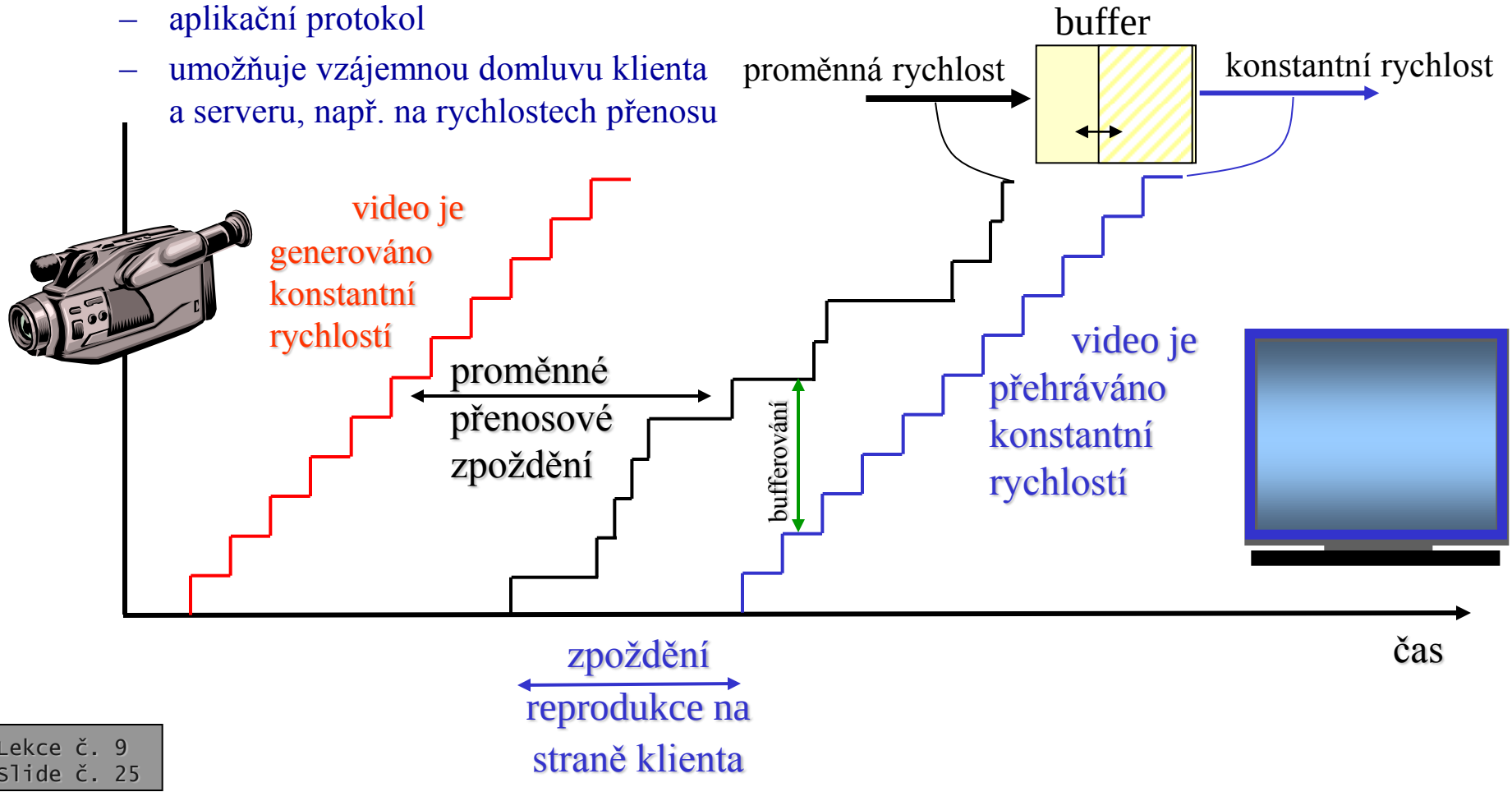
- standardní způsob řešení transportní vrstvy QoS (Quality of Service) nepodporuje
  - síťová vrstva "měří všem přenosům stejně"
    - pokud funguje na paketovém principu a na bázi "best effort", jako IP v rámci TCP/IP sítích
  - ani TCP ani UDP (v rámci transportní vrstvy TCP/IP) nevychází vstříc potřebám QoS
    - negarantují maximální zpoždění ani pravidelnost v doručování
- díky tomu je přenosová infrastruktura dnešního Internetu tak laciná, dostupná a rychlá
  - problém je ale v tom, že nevychází vstříc multimediálním přenosům, které mají své specifické požadavky na QoS



- možné řešení:
- přístup "hrubou silou"
  - zvyšuje se disponibilní přenosová kapacita
    - hlavně na úrovni páteřních sítí
  - problém se tím neřeší
    - pouze se statisticky snižuje četnost jeho výskytu
  - dnes je to nejschůdnější cesta
    - relativně laciná
    - ostatní jsou komplikovanější

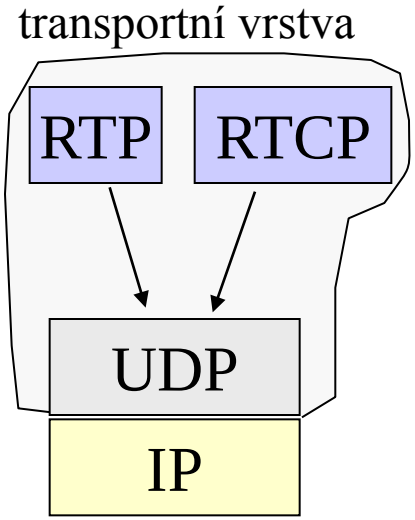
# "client buffering"
# QoS na aplikační úrovni - princip

- u jednosměrných (neinteraktivních) multimédií – například u videa – lze vyrovnávat "jitter" až u klienta vhodným bufferováním
  - u interaktivních přenosů lze využít také, ale celkové zpoždění nesmí být příliš velké!!
- RTSP – Real Time Streaming Protocol
  - aplikační protokol
  - umožňuje vzájemnou domluvu klienta a serveru, např. na rychlostech přenosu



# RTP/RTCP

- "čistě transportní" podpora QoS
  - standardizovaný způsob "balení" multimediálních dat do přenášených paketů, s podporou jejich multimediálního charakteru
  - ale bez vlivu na způsob jejich přenosu
    - ten je stále best effort!!!
- RTP (Real Time Protocol)
  - "balí" jednotlivé části multimediálních dat do vlastních bloků (paketů)
    - a ty vkládá do UDP paketů
  - připojuje informace
    - o typu multimediálního obsahu
      - Payload type 0: PCM, 64 kbps
      - Payload type 3, GSM, 13 kbps
      - Payload type 26, Motion JPEG
      - Payload type 33, MPEG2 video
      - ….




    - o pořadí paketu
      - jednotlivé pakety čísluje, usnadňuje detekci ztracených paketů
    - o čase vzniku dat (timestamp)
      - říká kdy přesně data vznikla
      - tím usnadňuje jejich bufferování na straně klienta
    - o konkrétním streamu (proudu)
      - v rámci jednoho RTP přenosu může být přenášeno více samostatných proudů (streamů)
  - podporuje multicast
- RTCP (Real Time Control Protocol)
  - zprostředkovává vzájemné informování zdroje a příjemců
    - např. o procentu ztracených paketů, o jejich zpoždění, o schopnostech příjemce apod.
    - přenáší popis RTP streamu, ….

# QoS: Integrated services

- snaha:
  - garantovat každému přesně to, co potřebuje
- základní princip:
  - při navazování spojení žadatel specifikuje, co bude potřebovat
    - jakou kapacitu, rychlost, zpoždění atd.
  - síť posoudí, zda to dokáže zajistit a garantovat
  - pokud ano:
    - spojení je navázáno
  - pokud ne:
    - žádost o navázání spojení je odmítnuta
- jak lze realizovat?
  - je nutné k tomu vyhradit (rezervovat) potřebný objem zdrojů
    - včetně přenosové kapacity a výpočetní kapacity v přepojovacích uzlech
  - nelze řešit výhradně na úrovni transportní vrstvy !!!!
    - je nutná určitá spolupráce již na úrovni vrstvy síťové
    - nestačí to implementovat v koncových uzlech, **musí být změněny i vnitřní a páteřní části sítě**

- princip realizace:
  - žadatel o navázání spojení uvede své R-spec
    - (Requirements), co bude od sítě požadovat
  - žadatel uvede své T-spec
    - jak bude vypadat datový provoz (Traffic), který bude generovat
  - musí existovat mechanismus (protokol), který R-spec i T-spec předá jednotlivým směrovačům v síti a "sjedná s nimi" jejich akceptování/odmítnutí
    - takovým protokolem je **RSVP**
      - **ReSerVation Protocol**
    - směrovače pak příslušné zdroje vyhradí pro právě vznikající virtuální okruh, po kterém pak přenos probíhá

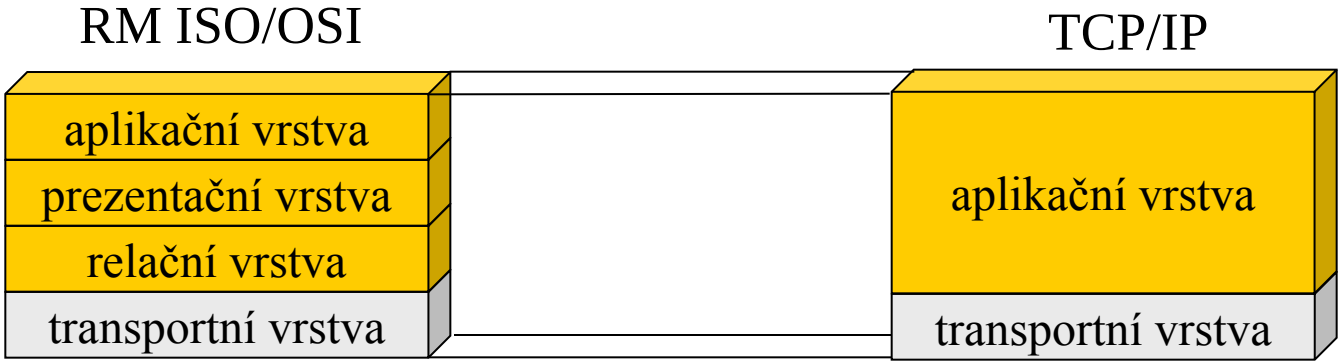
v zásadě je to návrat k principu přepojování okruhů
se všemi problémy – například neefektivní využití vyhrazené kapacity

# QoS: Differentiated Services

- myšlenka:
  - nesnažit se o "absolutní" naplnění požadavků (jako Integrated Services), ale nabídnout alespoň "relativní" (rozdílové, differentiated) služby
    - jeden druh provozu je upřednostňován na úkor jiného
- princip realizace:
  - zavede se několik tříd provozu
    - a každá třída bude mít jinou přednost/prioritu při přenosu a zpracování
  - každý přenášený paket či každé spojení se může "přihlásit" k určité třídě (prioritě)
    - a podle toho je s ním nakládáno
  - musí být podporováno v celé síti, již na úrovni síťové vrstvy
    - na rozdíl od "integrovaných služeb" mohou být jednotlivé třídy nastaveny dopředu a pevně
- praktické využití:
  - jednotlivé pakety deklarují svou příslušnost k určité třídě provozu skrze vhodnou "nálepku"
    - prefix
    - nastavení údaje ve své hlavičce
  - v IPv4: využívá se byte ToS (Type of Service)
- příklady (TCP/IP):
  - Expedited Forwarding
    - 2 třídy: Expedited a Regular
    - pakety v třídě Expedited mají absolutní přednost při přenosu před pakety z třídy regular
  - Assured Forwarding
    - 4 třídy podle priorit pro přenos
    - k tomu ještě 3 úrovně priorit pro zahazování paketů (při přetížení)
    - celkem 12 kombinací (tříd)

# podpora aplikací

- pozorování:
  - kromě přenosu dat potřebují nejrůznější aplikace další funkce/činnosti/aktivity/….
  - například:
    - minimalizaci efektu výpadků spojení (transportních), například při přenosu větších souborů
    - řízení dialogů (vzájemné komunikace aplikací)
    - podporu transakcí
    - zajištění bezpečnosti
    - konverze pro potřeby přenosu a pro stejnou interpretaci dat
    - …….
- otázka: mají být tyto činnosti zajištěny centrálně (společně), nebo si je má každá aplikace zajišťovat sama?
- RM ISO/OSI: mají být zajištěny centrálně
  - proto v RM ISO/OSI existuje samostatná relační a prezentační vrstva, která je zajišťuje
    - RM ISO/OSI má 7 vrstev
- TCP/IP: nechť si je každá aplikace zajistí sama
  - proto v TCP/IP neexistuje samostatná relační a prezentační vrstva
    - TCP/IP má jen 4 vrstvy



# relační vrstva

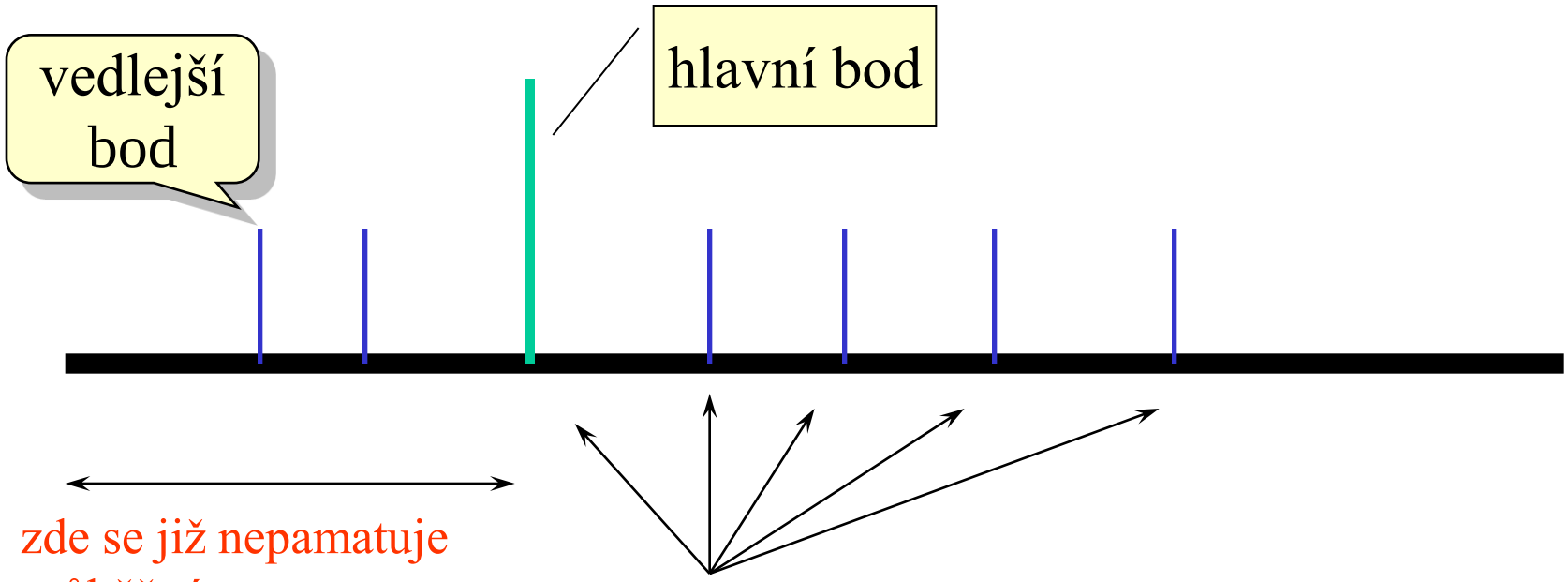
relační vrstva je nejvíce kritizovanou vrstvou ISO/OSI
- kvůli tomu, že toho má nejméně na práci

- úkoly relační vrstvy:
  - vést relace a řídit dialog
  - zajišťovat synchronizaci
  - podporovat transakce
  - zajišťovat bezpečnost
  - ……
- co jsou relace?
- analogie s telefonním hovorem:
  - vytočení tel. čísla a navázání spojení (telefonní hovor) odpovídá transportnímu spojení
  - dialog vedený po telefonu odpovídá relaci
    - dialog lze vést i jiným způsobem, než jen po telefonu (např. písemně, vysílačkami, ....)
    - jeden telefonní hovor (transportní spojení) může být postupně využit pro více různých dialogů (relací)
    - jeden dialog (relace) může pokračovat i přes více po sobě jdoucích tel. hovorů (transportních spojení

  - obecně (vedení relací):
    - jednu relaci lze vést prostřednictvím více (transportních) spojení
    - prostřednictvím jednoho transportního spojení lze vést více relací (dialogů)
  - relace (dialogy) mohou být:
    - plně duplexní
    - poloduplexní – je třeba vhodně řídit
  - ochrana přes zablokováním vzájemné komunikace
    - **starvation** (vyhladovění): obě strany čekají na reakci té druhé
    - **constipation** (zácpa): obě strany se snaží něco odeslat, a žádná nepřijímá
  - podpora transakcí
    - podpora nedělitelných skupin operací (transakcí), two-phase committ atd.

# podpora synchronizace

- synchronizace
  - pokud je přerušeno spojení (např. při přenosu souborů), je vhodné aby nebylo nutné začínat odznovu
  - relační vrstva umožňuje vkládat do přenášených dat „kontrolní body“
  - ke kontrolním bodům se lze vracet a pokračovat dál od nich
    - vedlejší bod: lze se přes něj vrátit
    - hlavní bod: nelze se přes něj vrátit zpět



# problém relační vrstvy

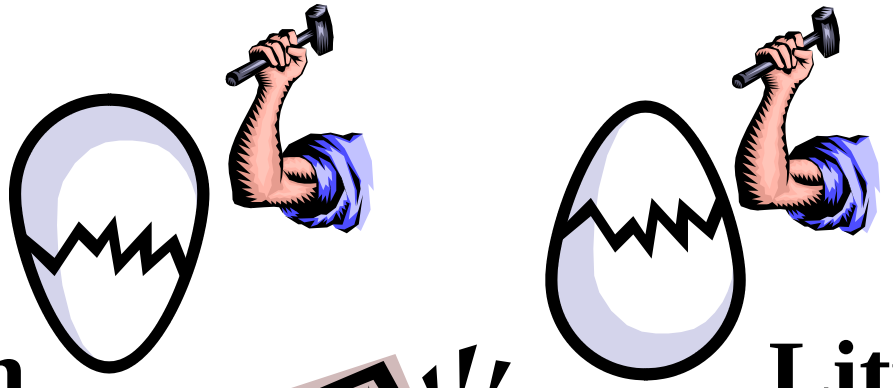
- (smysluplné) úkoly pro relační vrstvu by se našly
  - existují i protokoly, které by je umožnily realizovat
- ale tyto úkoly se řeší (protokoly se využívají) jinde
  - na úrovni aplikační vrstvy
- důvod:
  - celkový neúspěch RM ISO/OSI a přechod na model TCP/IP, který nemá samostatnou relační vrstvu
  - relační protokoly vznikají v rámci TCP/IP již pro aplikační vrstvu

- příklady relačních protokolů:
  - SIP (Session Initiation Protocol)
    - slouží (nejen) potřebám IP telefonie, ale i dalším formám komunikace
    - obdoba H.323
  - RPC (Remote Procedure Call)
    - vzdálené volání procedur
  - X-Window
    - zobrazovací služby
  - SQL (Structured Query Language)
  - ……

# prezentační vrstva

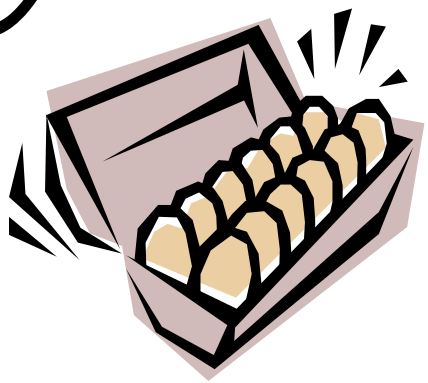
- nižší vrstvy se snaží doručit každý bit přesně tak, jak byl odeslán
- stejná posloupnost bitů může mít pro příjemce jiný význam než pro odesilatele, např. kvůli
  - kódování znaků (ASCII, EBCDIC,...)
  - formátování textů
  - formátu čísel (celá, reálná, …)
  - formátu struktur, polí
  - ukazatelům (pointerům)
- prezentační vrstva má na starosti potřebné konverze, tak aby obě strany interpretovaly přenášená data stejně!!
  - dále má na starosti převod dat z/do takového tvaru, ve kterém jsou data přenášena
- data určená k přenosu je nutné:
  - převést do takového tvaru, který je vhodný pro přenos
    - přenosový kanál je lineární (jednorozměrný), přenášená data musí být „zlinearizována“
      - např. vícerozměrná pole musí být převedena na jednorozměrná
      - pointery musí být eliminovány (nahrazeny něčím jiným)
  - dále je nutné poskytnout příjemci takovou informaci, která mu umožní správně pochopit význam dat
    - aby věděl co představují
    - aby si je uměl „poskládat“ zpět z přenosového tvaru do takového, s jakým on sám pracuje

# Little Endian vs. Big Endian



## Big Endian

(vyšší byte je na nižší adrese)



| | |
|---|---|
| 3 | |
| 2 | |
| 1 | 34H |
| 0 | 12H |

⇦ 1234H ⇨

## Little Endian

(nižší byte je na nižší adrese)

| | |
|---|---|
| 3 | |
| 2 | |
| 1 | 12H |
| 0 | 34H |

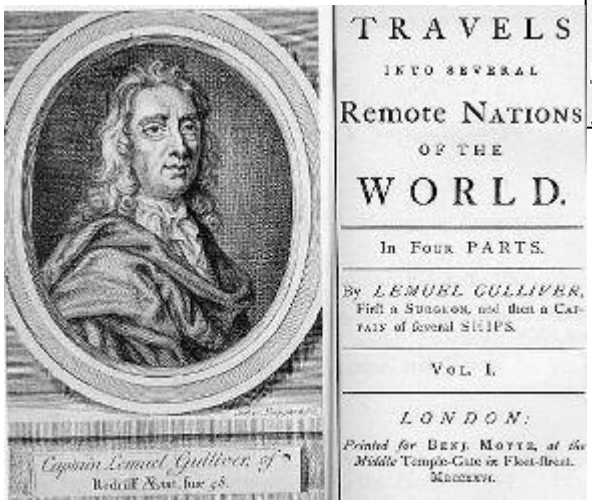
např. mikroprocesory Motorola, TCP/IP, Ethernet

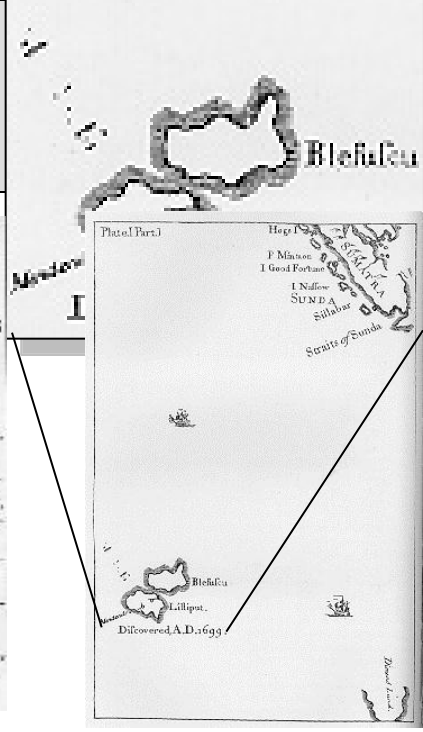
též: network byte order

např. mikroprocesory Intel

# proč Little a Big Endian?

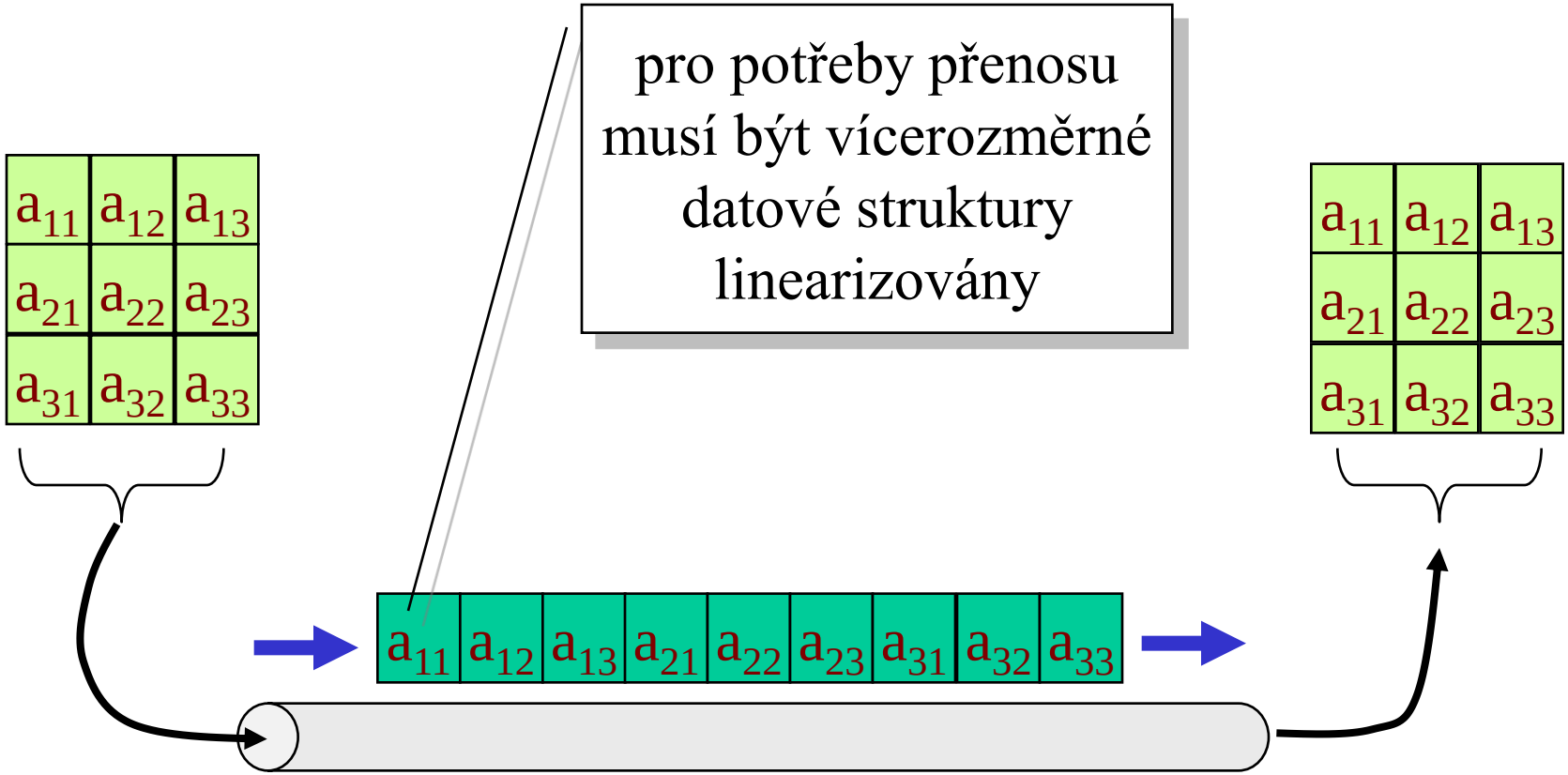
- 1. apríla 1980 napsal Dave Cohen dokument IEN 137
  - IEN (Internet Experiment Notes)
    - byly typem dokumentů, které posléze splynuly s RFC
- s názvem:
  - ON HOLY WARS AND A PLEA FOR PEACE




  - This is an attempt to stop a war. I hope it is not too late and that somehow, magically perhaps, peace will prevail again.
  - The latecomers into the arena believe that the issue is: "What is the proper byte order in messages?".
- přirovnává to k pasáži z Gulliverových cest
  - kde Jonathan Swift parodoval náboženské války mezi Anglií a Francií, mezi protestanty a katolíky
    - *Gulliver se dostal do země Lilliput, která vedla nesmiřitelnou válku s říší Blefuscu*
    - *bojovalo se o to, zda rozbíjet vajíčka "na malém konci" ("at the **L**ittle End", jak to praktikovali v říši **L**illiput), nebo "na velkém konci" ("at the **B**ig End", v říši **B**lefuscu)*

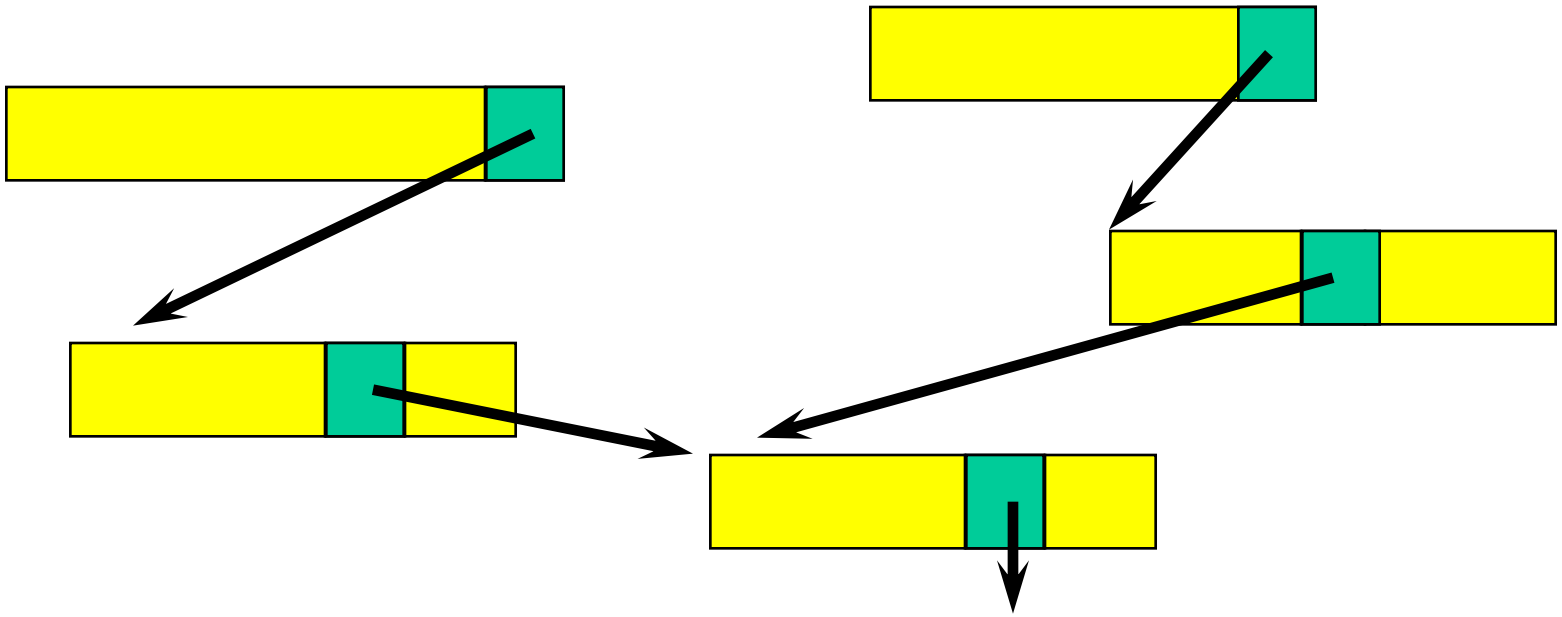
# představa: linearizace struktur




přenosový kanál je lineární

# jiný problém:

- co dělat se strukturami, které jsou provázány pointry?
  - jsou pevně vázány na adresový prostor odesilatele,
  - v adresovém prostoru příjemce nemají smysl!!!!!
- jediné možné řešení:
  - převést struktury s pointry na ekvivalentní strukturu bez pointerů, tu přenést
    - a pak případně "nějak vrátit zpět" do struktury s pointry



# možnosti řešení konverzí

- přizpůsobení stylem „každý s každým“
  - každá dvojice příjemce-odesilatel se dohodne na společném formátu
  - konverze se provádí jen 1x
    - buďto u odesilatele, nebo u příjemce
    - je to efektivnější
  - stále je nutné převádět z/do přenosové tvaru
  - v praxi je to problematické
    - ne vždy se mohou libovolné dvě strany předem dohodnout
- přes společný mezitvar
  - je definován jeden společný "mezitvar"
    - i přenosový tvar
  - konverze se provádí 2x
    - 1x u odesilatele
    - 1x u příjemce
    - vše společně s převodem do přenosového tvaru
  - je to jednodušší na vzájemnou koordinaci
    - každé straně stačí naučit se převádět z/do společného mezitvaru

# představa ISO/OSI

- otázka:
  - jak zajistit to, aby příjemce věděl jak rekonstruovat data, přijatá v přenosovém (mezi)tvaru?
    - tak, aby pro něj měla stejný význam jako pro odesilatele?
- řešení:
  - u odesilatele se nejprve (obecně) popíší data, která mají být přenesena
    - obecným způsobem, který zachycuje jejich význam
      - představa: jako kdyby šlo o deklarace v programovacím jazyku
  - vznikne "průvodka" k datům
  - data se zkonvertují a převedou do přenosového tvaru
  - k datům v přenosovém tvaru se připojí průvodka
    - a vše se přenese
  - příjemce podle průvodky převede data z přenosového tvaru do takového tvaru, který má pro něj stejný význam jako pro odesilatele
- co je nutné mít k dispozici?
  - jazyk pro "průvodku"
    - jazyk umožňující obecný popis dat
    - šlo by použít nějaký konkrétní programovací jazyk
      - a z něj by stačily jen deklarace
    - ale žádný programovací jazyk nebyl vhodný
      - standardizovaný, …
    - pro ISO/OSI byl vytvořen speciální jazyk pro abstraktní popis obecných dat
      - **ASN.1 (Abstract Syntax Notation One)**
      - je to ISO Standard X.680
      - používá se hodně i v rámci Internetu
  - pravidla pro převod z/do přenosového tvaru
    - pro ASN.1 byla vytvořena samostatná pravidla **BER (Basic Encoding Rules)**
      - říkají, jak reprezentovat jednotlivé datové položky
      - používá se tzv. TLV kódování (Type, Length, Value)

# příklad: ASN.1 a BER

instance dat

{lastname, 'smith'}
{weight, 259}

popis v ASN.1 – "průvodka"

```
lastname ::= OCTET STRING;
weight ::= INTEGER
```

kódování BER

| datový typ | kód |
|---|---|
| Boolean | 1 |
| Integer | 2 |
| Bitstring | 3 |
| Octet string | 4 |
| Null | 5 |
| Object Identifier | 6 |
| Real | 9 |