



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



# Rodina protokolů TCP/IP, verze 2.7

## Část 8: TELNET, FTP a NFS

*Jiří Peterka, 2011*

# připomenutí: aplikace v TCP/IP

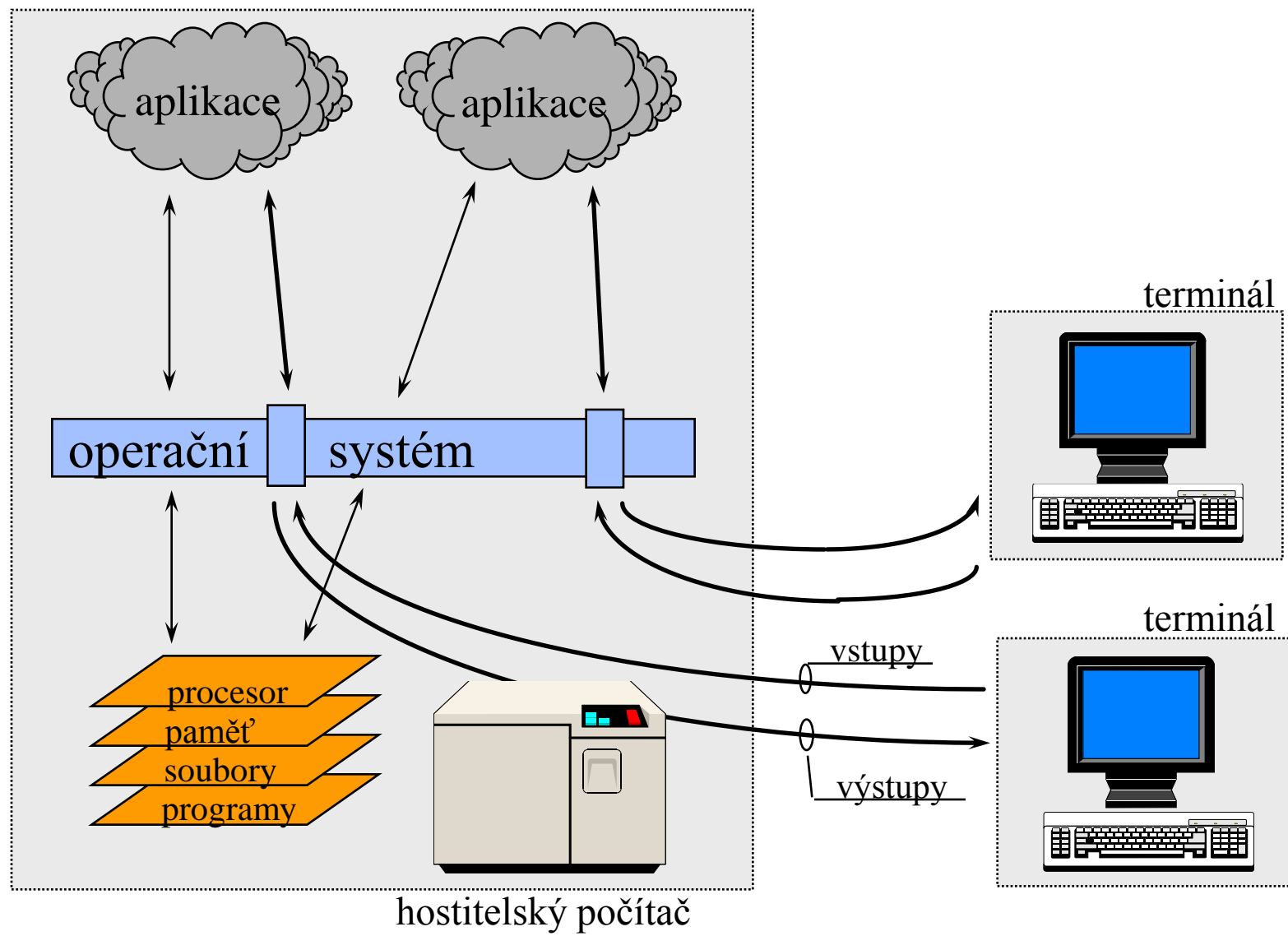
- jsou vesměs postaveny na architektuře klient/server
- součástí aplikační vrstvy jsou takové části aplikací, které jsou nutné pro interoperabilitu a "základní fungování" konkrétní služby
  - obdobně jako v RM ISO/OSI
  - např. transport zpráv a souborů
- uživatelské rozhraní již není součástí aplikační vrstvy
  - není svázáno standardy
- historický vývoj:
  - zpočátku se používaly především tyto aplikace:
    - TELNET (vzdálené přihlašování)
    - FTP (přenos souborů)
    - el. pošta (SMTP, FRC 822)
  - později také:
    - NFS (sdílení souborů)
    - Gopher
    - ....
  - dnes převládá (kromě pošty):
    - **WWW** (jako samostatná aplikace i jako platforma pro provozování dalších aplikací, např. vyhledávacích služeb, adresářů, ...)

trend k "platformizaci" – původně samostatné služby se stávají nadstavbou nad WWW

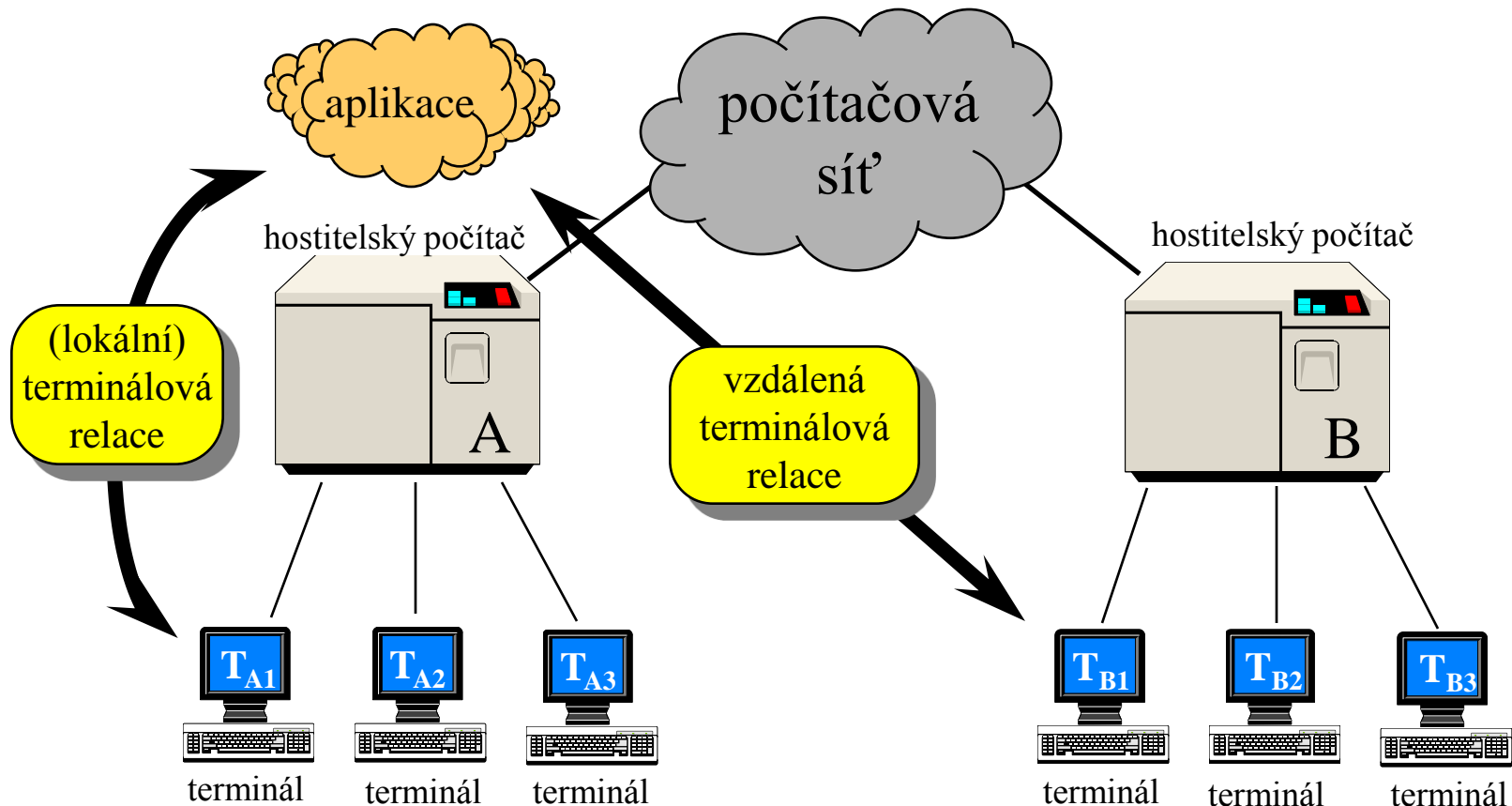
# vzdálené přihlašování (remote login)

- cílem je možnost "používat aplikace na dálku"
  - uživatel je na místním počítači, ale aplikace běží na vzdáleném počítači
- týká se aplikací fungujících na bázi výpočetního modelu "host/terminál"
  - typicky: starších aplikací fungujících ve znakovém režimu
  - aplikací které ani nemusí počítat s tím že fungují v prostředí sítě
- podmínka:
  - aplikace musí podporovat tzv. terminálové relace
    - musí své výstupy posílat "skrz OS" na terminál a své vstupy přijímat "skrz OS" z terminálu
      - takto se děje např. v prostředí UNIXu
    - nesmí "jít napevno" do videoRAM a klávesnicového bufferu
      - jako v MS DOS
  - také operační systém musí podporovat terminálové relace

# představa výpočetního modelu host/terminál



# představa terminálové relace



# vzdálené přihlašování v TCP/IP

- TELNET

- “hlavní” prostředek pro realizaci vzdáleného přihlašování v rámci TCP/IP, snaží se být maximálně univerzální

- snaží se nevázat na konkrétní systémové prostředí (nebýt závislý na operačním systému)
- podporuje terminálové relace mezi různými platformami
  - na straně uživatele počítá i s terminálovou emulací
- ... nabízí jen jednoduché služby, a nikoli např. možnost automatického přihlašování apod.

- podporuje pouze znakové uživatelské rozhraní

- rlogin

- vzniknul v prostředí BSD Unixu
- je vázán na prostředí Unixu (BSD Unixu), a podporuje tzv. “trusted hosts”

- dokáže si některé informace “vytáhnout” ze svého okolí (např. jméno uživatele a jeho heslo) ....
- ... a pak je využít pro automatické přihlášení uživatele na vzdáleném počítači

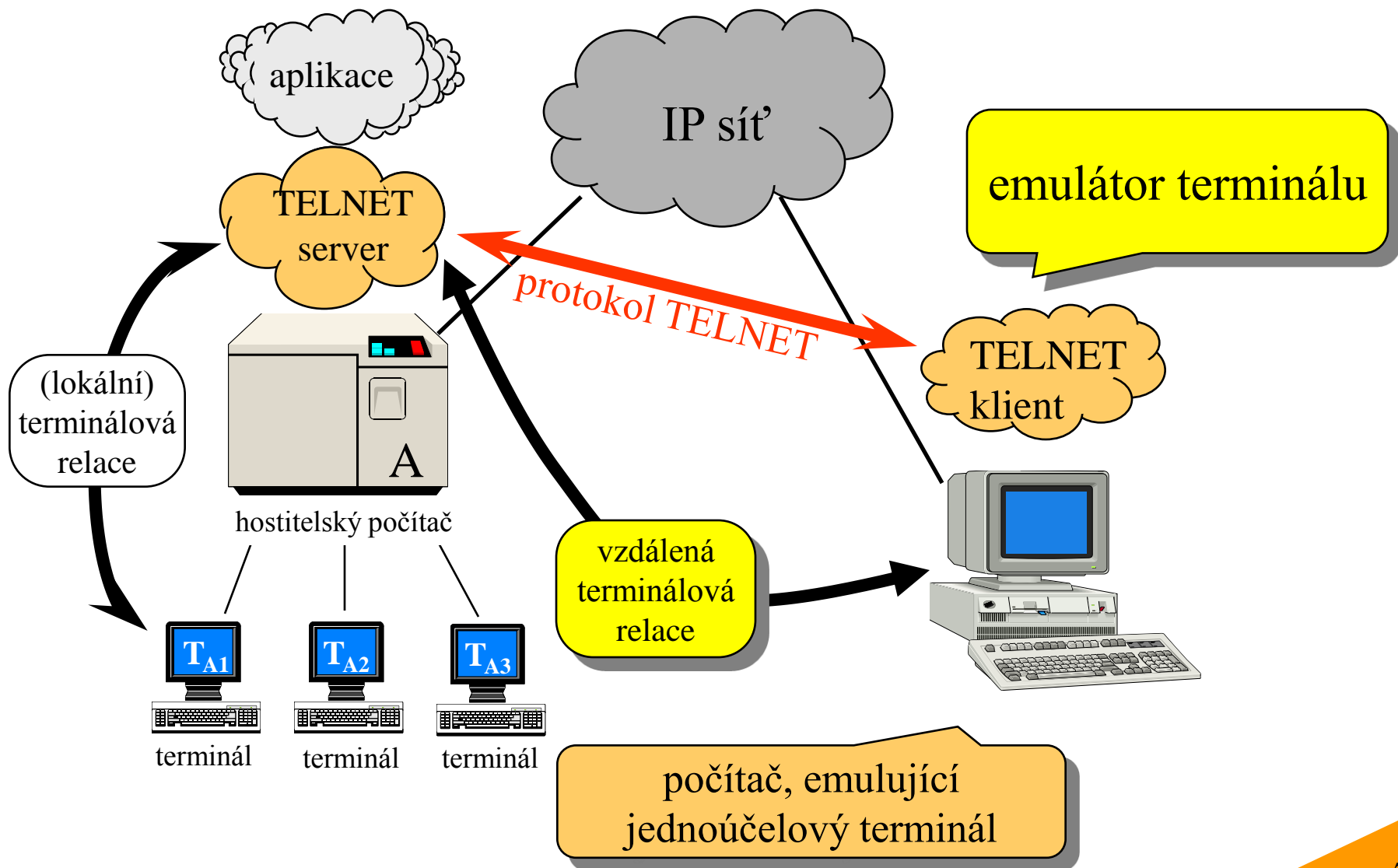
- ICA (WinFrame, MS Terminal Server)

- nové řešení, podporuje grafiku

- částečně také:

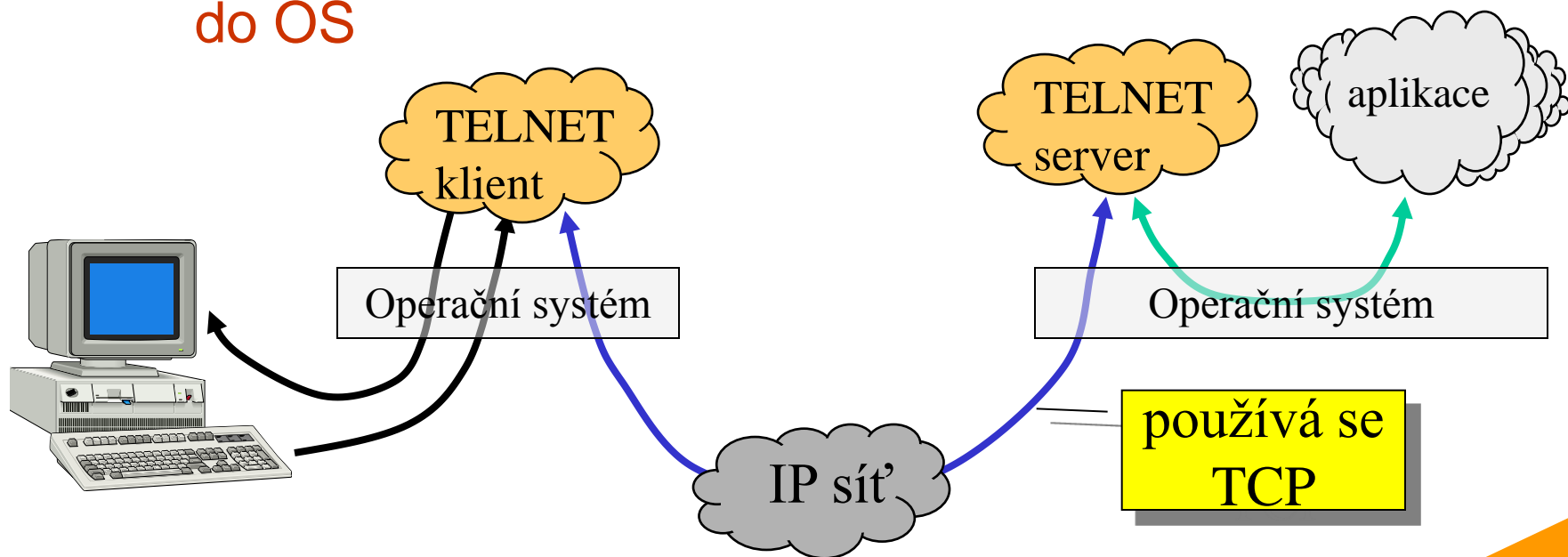
- systém X Window

# terminálová emulace vs. TELNET



# TELNET - principy

- TELNET server je realizován na aplikační úrovni
  - jako systémová úloha - démon, a nikoli “uvnitř” OS
  - výhoda: snazší modifikace
  - nevýhoda: ne každý OS tomu vychází vstříc
  - nevýhoda: je to méně efektivní než přímé zabudování do OS

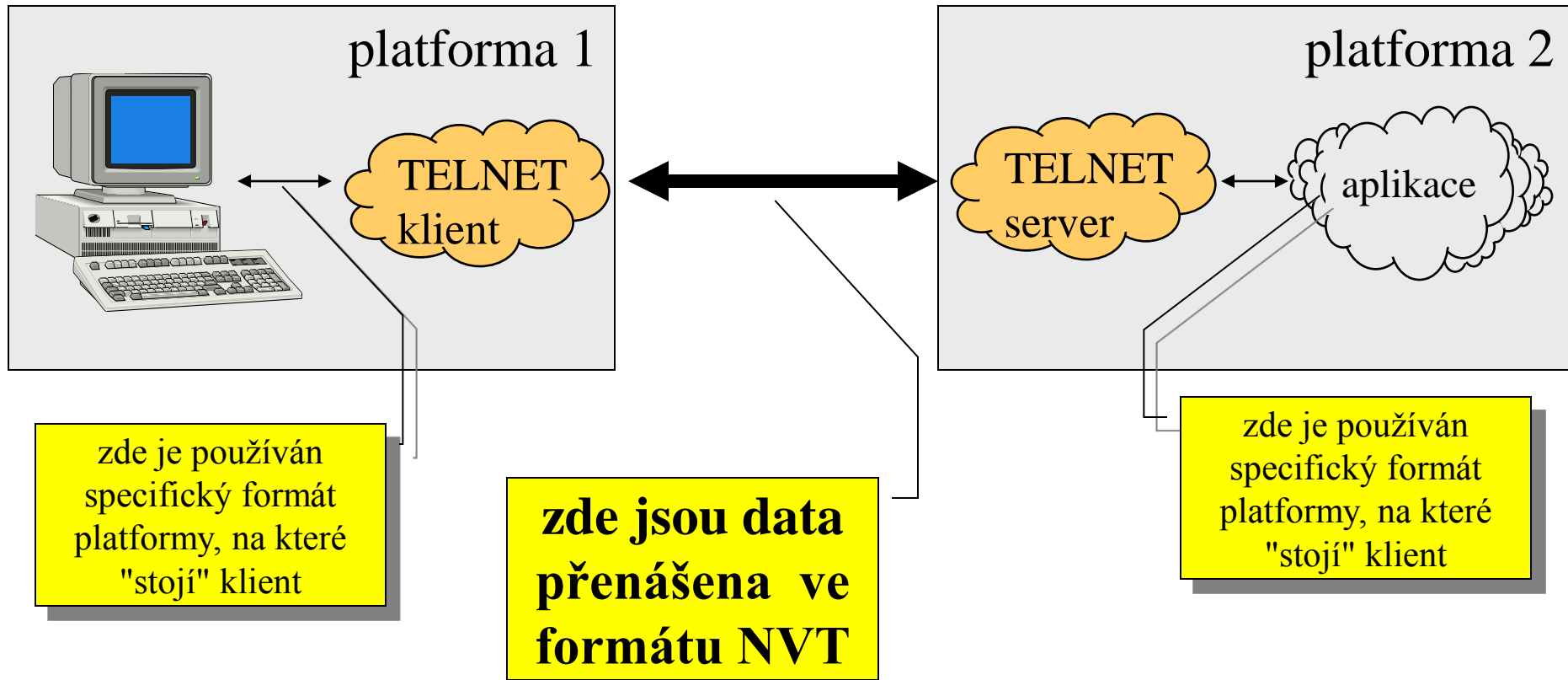




# TELNET – základní problém

- **“každý terminál je jiný”**
- liší se:
  - v rozsahu schopností
  - v parametrech
  - v režimu fungování
    - znakový
    - celoobrazovkový
    - formulářový
- s odlišnostmi terminálů se lze vyrovnat přizpůsobením stylem “každý s každým”
  - ... nebo přes společný mezistupeň, s přizpůsobením typu “každý s jedním” a “jeden s každým”
    - v případě TELNETu jde hlavně o tvar, v jakém se mají data přenášet TCP/IP sítí
- TELNET zavádí jeden, pevně daný mezistupeň: **virtuální terminál NVT** (Network Virtual Terminal)
  - NVT má vždy stejné vlastnosti
  - reálné terminály se mu přizpůsobují (jsou mapovány z/do NVT)
  - NVT především definuje formát skutečně přenášených dat

# TELNET - představa



# vlastnosti NVT

- NVT je “společným jmenovatelem” ...
  - ... povinným minimem, které musí “umět” všechny reálné terminály
  - NVT odpovídá jednoduchému řádkovému terminálu
- TELNET obsahuje mechanismy, pomocí kterých se klient se serverem mohou dohodnout “na lepším”
  - jde o tzv. **options**
- NVT odpovídá jednoduchému znakovému terminálu, tj.:
  - data jsou členěna na řádky
  - data jsou přenášena po znacích
  - komunikuje se pouze polo-duplexně
  - používá se lokální echo
- původní představa:
  - **NVT odpovídá klávesnici a tiskárně**

# vlastnosti NVT

- pro přenos využívá spolehlivé přenosové služby TCP (ale jen poloduplexním způsobem)
  - přenos je znakově orientovaný
  - a vždy 8-bitový (tj. jednotlivé znaky se přenáší vždy v 8 bitových bytech)
  - ... ale standardně se předpokládá přenos 7-bitových ASCII znaků
    - je povinnost zobrazovat 95 tisknutelných ASCII znaků (s kódy 32 až 127)
    - ... a povinnost interpretovat znaky NULL, CR a LF
    - dále je přesně definován význam znaků BEL, BS, HT, VT, FF (ale jejich interpretace není povinná)
- ostatní znaky nemají mít žádný efekt
- požaduje se, aby klávesnice byla schopna generovat všech 128 ASCII znaků
- znak, kterým je zakončena řádka (resp. znaky), musí TELNET klient nahradit dvojicí znaků CR a LF
  - klient server naopak nahradí CR a LF takovým znakem (znaky), které na jeho straně představují zakončení řádky
- přenos 8-bitových znaků je jedním z volitelných režimů (options)

# TELNET - řídicí příkazy

- NVT počítá i s implementací řídicích mechanismů
  - např. pro přerušování právě probíhajícího programu, pomocí CTRL+C)
- ...a k tomu potřebuje přenášet řídicí příkazy, povely atd.
  - řídicí příkazy mají zásadně znakovou povahu (tvoří je řídicí znaky)
- pro zajištění transparency se používá prefixace (uvození) speciálním znakem IAC (Interpret As Command), s kódem 255
  - případný výskyt znaku IAC v datech se řeší jeho zdvojením.
- řídicí příkazy jsou trojího druhu:
  - editační příkazy:
    - umožňují mazat znak a řádku
  - řídicí příkazy:
    - umožňují přerušit probíhající proces, zastavit jeho výstup ...
  - “dohadovací” příkazy:
    - umožňují oběma stranám dohodnout se na případných rozšířeních oproti NVT

# TELNET - rozšíření

- umožňují oběma stranám dohodnout se na jiném, než co vyplývá z pevně daných vlastností NVT, např.:
  - zda budou komunikovat v plném duplexu,
  - že přenášená data budou odesílána po řádcích s možností jejich lokální editace
  - že budou používat vzdálené echo
  - jakou velikost displeje budou používat
  - jaký způsob budou řízení toku využívat
  - .....
- zásada: použití rozšíření je dobrovolné
  - kterákoli strana má právo navrhnout použití konkrétního rozšíření (tj. jak klient, tak i server)
  - druhá strana má právo použití rozšíření odmítnout
- k “dohadování” slouží samostatné příkazy:
  - **WILL** ("*já chci používat rozšíření ....*")
    - odpověď: DO vs. DON'T
  - **DO** ("*chci abys ty používal rozšíření ....*")
    - odpověď: WILL vs. WON'T

# TELNET - rozšíření

- přesná specifikace jednotlivých rozšíření (číselný kód, sémantika) není apriorně uzavřena (omezena)
  - nová rozšíření mohou vznikat postupně (a být definována prostřednictvím dokumentů RFC)
  - existuje dokonce i mechanismus pro “rozšiřování rozšíření”
- další příklad rozšíření: výměna informací o konkrétním typu terminálu
  - umožňuje aplikacím lépe přizpůsobit svůj výstup možnostem terminálu (např. přímým nastavováním kurzoru na zadanou pozici apod.)
- pro jednotlivá rozšíření může dojít k podrobnějším “licitacím”
  - TELNET jejich průběh nespecifikuje
  - ... je definován až samotným rozšířením (a může tedy být pro něj specifický)
  - TELNET definuje pouze způsob zajištění transparence dat při další “licitaci”

# vzdálené přihlašování vs. práce se soubory

---

- cíl:
  - *“chci pracovat se souborem, který se nachází na jiném počítači”*
  - přesněji: chci zpracovávat obsah vzdáleného souboru, pomocí aplikace, která běží na “mém” uzlu
- !!!... účelem není dostat se do role vzdáleného terminálu jiného uzlu ... !!!
  - pak by byl soubor zpracováván aplikací běžící na vzdáleném počítači



# přenos vs. sdílení souborů

- přenos souborů:
  - jde o **netransparentní řešení**
    - uživatel/aplikace si **uvědomuje**, že soubor se nachází na vzdáleném počítači
    - uživatel/aplikace **musí** explicitně **znát umístění** souboru na vzdáleném počítači
    - uživatel/aplikace **musí explicitně podnikat** určité kroky ke zpřístupnění souboru
  - typicky: vzdálený soubor se celý přenesse na "místní" počítač a zde se zpracuje
- označuje se jako "file transfer"
  - v rámci TCP/IP řeší protokol **FTP (File Transfer Protocol)**
- sdílení souborů:
  - jde o **transparentní řešení**
    - uživatel/aplikace si **neuvědomuje**, že soubor se nachází na vzdáleném počítači
    - uživatel/aplikace **nemusí** explicitně **znát umístění** souboru na vzdáleném počítači
    - uživatel/aplikace **nemusí podnikat** žádné explicitní kroky ke zpřístupnění souboru
  - typicky: vzdálený soubor se "chová" ("tváří") jako místní soubor, a také se s ním jako s místním pracuje
- označuje se jako "file sharing"
  - v rámci TCP/IP řeší protokol **NFS (Network File System)**

# problémy přenosu a sdílení souborů

- protokoly pro přenos a sdílení se musí vyrovnat s mnoha úskalími, typu:
  - rozdílnost v pohledu na soubory, jejich jména, přípony  
....
  - vlastnictví souborů, jejich atributy
  - .....
- řešení je snazší u netransparentního přístupu (file transfer)
  - kde lze požadovat po uživateli, aby rozdílnosti vyřešil vlastním rozhodnutím
    - například aby zadal atributy místního souboru, do kterého má být přenesen (zkopírován) obsah vzdáleného souboru
- problém je i se zajištěním vícenásobného přístupu k souborům
  - lze použít “obecné” techniky typu:
    - uzamykání celých souborů či jejich částí
    - replikace
    - ponechat vše na uživateli
    - .....

# protokol FTP (File Transfer Protocol)

- je starší než rodina protokolů TCP/IP
  - pochází již z roku 1971 (vznikl nad protokolem NCP)
  - teprve později “portován” nad přenosové protokoly TCP/IP
  - .... přesněji nad protokol TCP
- protokol FTP vzniknul v době, kdy se operační systémy a platformy lišily více než dnes
  - mj. ve velikosti slov, znázornění znaků, např.:
  - někdo umisťoval čtyři 9-bitové znaky do jednoho 36-bitového slova
  - jiný OS umístil do 36-bitového slova pět 7-bitových znaků
  - .....
- dnes ale většina těchto "dávných" vlastností není podporována
  - protože mezitím došlo ke značnému sjednocení
- jediný významnější pozůstatek:
  - snaha konvertovat text při jeho přenosu mezi různými platformami
    - například kódování, konce řádek, ...
- FTP pracuje ve dvou režimech:
  - **textovém** – provádí konverze
    - implicitně
  - **binárním** – konverze neprovádí

musí se mu říci, co bude přenášet  
(zvolit jeden z režimů)

# FTP – textové přenosy

- FTP zavádí jednotný formát dat pro potřeby přenosu
  - veškeré konverze z/do tohoto formátu ponechává na koncových uzlech
  - umožňuje ale oběma stranám dohodnout se v konkrétním případě na jiném formátu (kvůli větší efektivnosti přenosu)
- FTP přenáší data zásadně jako 8-bitové byty
- pro text používá FTP stejný formát, jako protokol TELNET:
  - jednotlivé znaky přenášeny v 8 bitech
  - konec řádky = CR+LF
  - kódování ASCII
- alternativní možností je použití kódu EBCDIC, použití nestandardní velikosti bytu atd.

# FTP – představa a přenos souborů

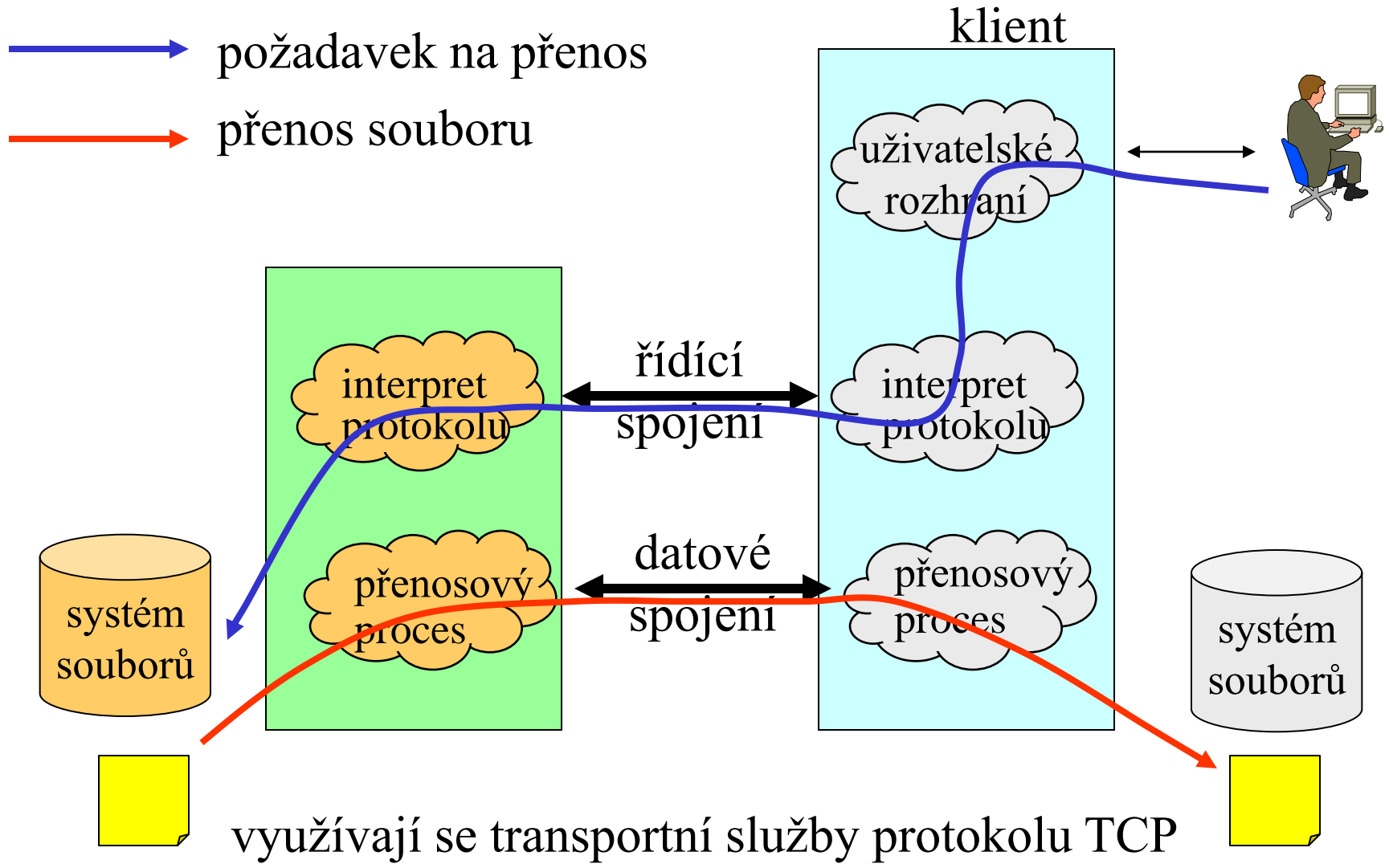
- FTP implicitně chápe soubor jako dále nestrukturovaný (bez vnitřní struktury) - označováno jako **file structure**
  - alternativně je schopen se na něj dívat jako na posloupnost stejně velkých záznamů (records) - **record structure**
  - nebo jako na množinu stránek (které mohou tvořit nespojitý soubor) - **page structure**
- implicitně je obsah souboru přenášen jako spojitý proud dat (tzv. **stream mode**)
  - alternativou je blokový režim (**block mode**), při kterém je možné vkládat mezi bloky “zarážky”, a po ev. výpadku spojení se k nim vracet
  - další alternativou je zhuštěný režim (**compressed mode**), kdy je používána jednoduchá metoda komprese (eliminující opakující se znaky)

dnes se již nepoužívá

# FTP - implementace

- vychází z modelu klient/server
  - klient je typicky aplikačním programem
  - server obvykle systémovým procesem (démonem, rezidentním programem apod.)
  - návrh protokolu je uzpůsoben možnosti úsporné implementace (takové, která si nárokuje většinu systémových zdrojů až v okamžiku jejich skutečné potřeby)
- zajištění potřebných funkcí v rámci FTP je rozděleno mezi dva subjekty:
  - interpret protokolu (**PI, Protocol Interpreter**)
  - přenosový proces (**DTP, Data Transfer Process**)
- interpret protokolu existuje trvale,
  - přenosový proces vzniká až na základě konkrétního požadavku
- používají se dvě různá spojení:
  - **řídící** (pro přenos příkazů)
  - **datové** (pro přenos souborů)

# implementace protokolu FTP



# datové a řídicí spojení

- oddělení datového a řídicího spojení je výhodné:
  - kvůli zajištění transparence
  - kvůli možnosti přerušit probíhající přenos
  - kvůli možnosti signalizovat konec souboru
    - uzavření datového spojení signalizuje konec souboru
    - lze přenášet soubory které během přenosu narůstají
- definice FTP (RFC) požaduje aby datové spojení bylo 1 pro všechny přenášené soubory
  - v praxi se pro každý přenášený soubor používá 1 (samostatné, nové) datové spojení
- řídicí spojení "přežívá" po celou dobu relace, datová spojení se mění
- řídicí spojení iniciuje (navazuje) klient
  - ze svého (dynamicky přiděleného) portu na port 21
    - ruší se až explicitním příkazem
- datové spojení iniciuje (navazuje) server
  - ze svého portu 20 na port klienta, ze kterého bylo navázáno řídicí spojení
  - **passive-mode**: datové spojení nenavazuje server, ale klient
    - kvůli firewallům, které neakceptují žádosti o otevření spojení vedoucí dovnitř na "náhodný" port



# uživatelé a FTP

- FTP je “user aware” -
  - uvědomuje si existenci uživatelů
- server při přístupu k místním souborům vždy vystupuje jménem konkrétního uživatele
  - FTP potřebuje mechanismy pro přihlášení uživatele a jeho autentikaci (ověření identity)
- uživatelé se v rámci "FTP relace" musí identifikovat
  - musí se přihlásit místním uživatelským jménem a prokázat platným heslem
- anonymní FTP
  - konvence: má-li být něco veřejně přístupné, uživatelé se hlásí jako "anonymous"
    - a heslo není významné, obvykle se požaduje emailová adresa kvůli statistikám

# řídící jazyk FTP

---

- FTP definuje vlastní řídící jazyk
  - příkazy řídícího jazyka jsou přenášeny řídícím spojením
- řídící příkazy mají textovou povahu, a jsou přenášeny ve stejném tvaru, jakou předpokládá protokol TELNET
  - resp. pro jejich přenos mohou být využívány již existující implementace TELNETu
- příkazy lze rozdělit na:
  - **řízení přístupu** (access control commands) - např. pro zadání uživatelského jména a hesla
  - **nastavení parametrů** přístupu (transfer parameter commands) - např. pro změnu implicitních čísel portů, pro nastavení režimu přenosu apod.
  - **výkonné příkazy** (FTP service commands) - pro vlastní přenos souborů, rušení, přejmenovávání atd., pro přechody mezi adresáři apod.

# odpovědi na příkazy FTP

- každý příkaz vyvolá alespoň jednu odpověď
- odpovědi mají číselný charakter (s textovým komentářem)
- odpovědi tvoří trojmístné číslo:
  - první číslice vyjadřuje celkový charakter odpovědi
  - druhá číslice upřesňuje odpověď
  - třetí ještě blíže specifikuje
- hierarchický charakter odpovědí vychází vstříc různé inteligenci procesů, které je vyhodnocují
  - “hloupý” klient či server se může spokojit jen s první číslicí
  - “chytrý” klient (server) využije všechny číslice

1xx	<b>předběžná kladná odpověď</b> (akce byla zahájena, budou ještě další odpovědi)
2xx	<b>kladná odpověď</b>
3xx	<b>prozatímní odpověď</b> (jsou nutné další příkazy)
4xx	<b>dočasná záporná odpověď</b> (nepodařilo se, ale je vhodné opakovat)
5xx	<b>trvalá záporná odpověď</b> (nepodařilo se a nemá smysl opakovat)

# příklad

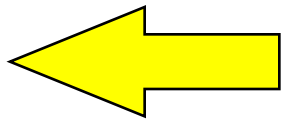
klient

server

- **RETR** <soubor>

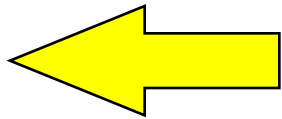


začátek



průběh činnosti

konec



- .... požadavek

- **160** ASCII retrieve of  
<soubor> started

- .. probíhá přenos

- **226** Transfer completed,  
123456 bytes transferred

# řídící vs. uživatelský jazyk

- pevně definován a závazný je řídící jazyk
  - příkazy řídícího jazyka mohou být vysílány “ručně”, pomocí TELNETu (na porty FTP)
- v praxi je na straně klienta vždy implementováno nějaké uživatelské rozhraní (řádkové, grafické, .....)
  - toto rozhraní může nabízet uživateli v podstatě jakýkoli "uživatelský" (řídící) jazyk
    - a překládat jej do řídícího jazyka

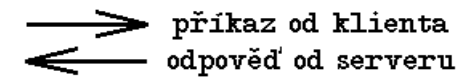
řídící jazyk	uživatelský jazyk
RETR	GET
STORE	PUT
LIST	DIR
CWD	CD

typické pro  
řádkové klienty

# příklad

Uživatel. jazyk

Řídící jazyk



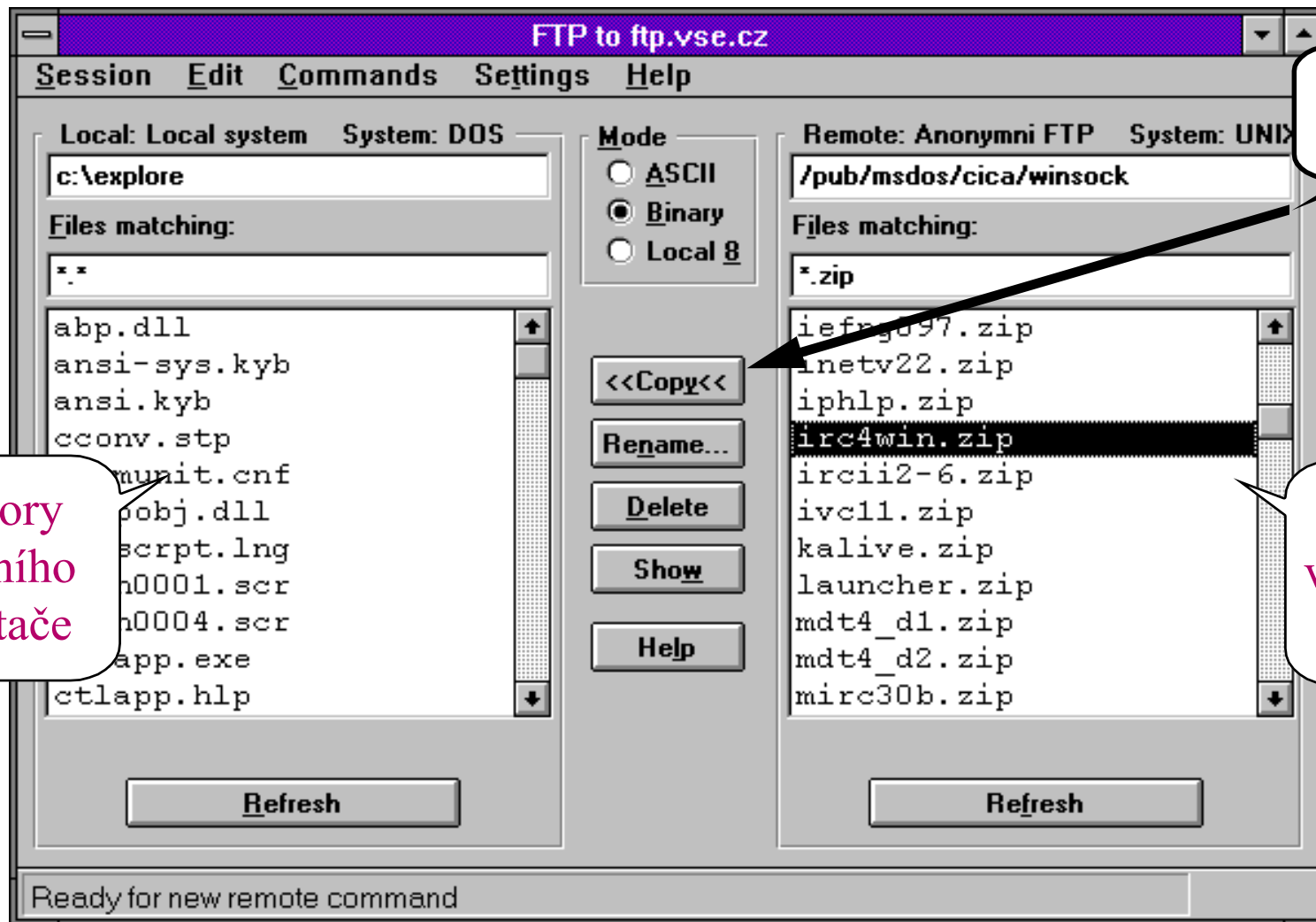
```
ftp <adresa serveru>          --- je zřizováno řídicí spojení ---
                                ← 220 <identifikační text, kterým se hlásí server>
name: anonymous                → USER anonymous
                                ← 331 Anonymous user ok, send
                                real ident as password
password:.....                → PASS <zadané heslo>
                                ← 230 User anonymous logged in
                                at <údaj o čase připojení>
ftp> get help.doc              → RETR help.doc
                                ← 150 ASCII retrieve of help.doc
                                started
                                --- probíhá přenos souboru ---
                                ← 226 Transfer Completed, 123456
                                bytes transferred in 4.5e+02
                                seconds (0.27 Kbytes/s)
ftp> quit                       → QUIT
                                ← 221 QUIT command received, Goodbye
                                --- řídicí spojení je zrušeno ---
```

# příklad (řádkový klient)

```
C:\WINAPPS\FTPW.EXE
230-
230- Welcome, archive user! This is an experimental FTP server. If
230- you have any unusual problems, please report them via e-mail to
230- ftp-admin@vse.cz
230-
230- >>> Please do not complain about limit on number of users <<<
230- >>> and other restrictions. <<<
230-
230-
230-
230-
230-
230 Guest login ok, access restrictions apply.
ftp>dir _____
Listen on port 3341
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 30
d--x--x--x  2 0      0      512 Sep 29 1994 bin
d--x--x--x  2 0      0      512 Feb  4 1993 dev
d--x--x--x  3 0      0      512 Jan 26 1993 etc
drwx-----  2 0      0     8192 Feb 21 1993 lost+found
drwxrwxr-x 23 0     106    1024 Nov 29 12:18 pub
drwxrwxr-x  2 999   106     512 Nov 29 14:56 upload
d--x--x--x  7 0      0      512 Apr 30 1996 usr
drwx--x--x  2 1002  1002    512 Oct  3 08:03 warez
226 Transfer complete.
ftp>
```

Příkaz uživatele  
pro vypsaní obsahu  
adresáře

# příklad (grafický klient)



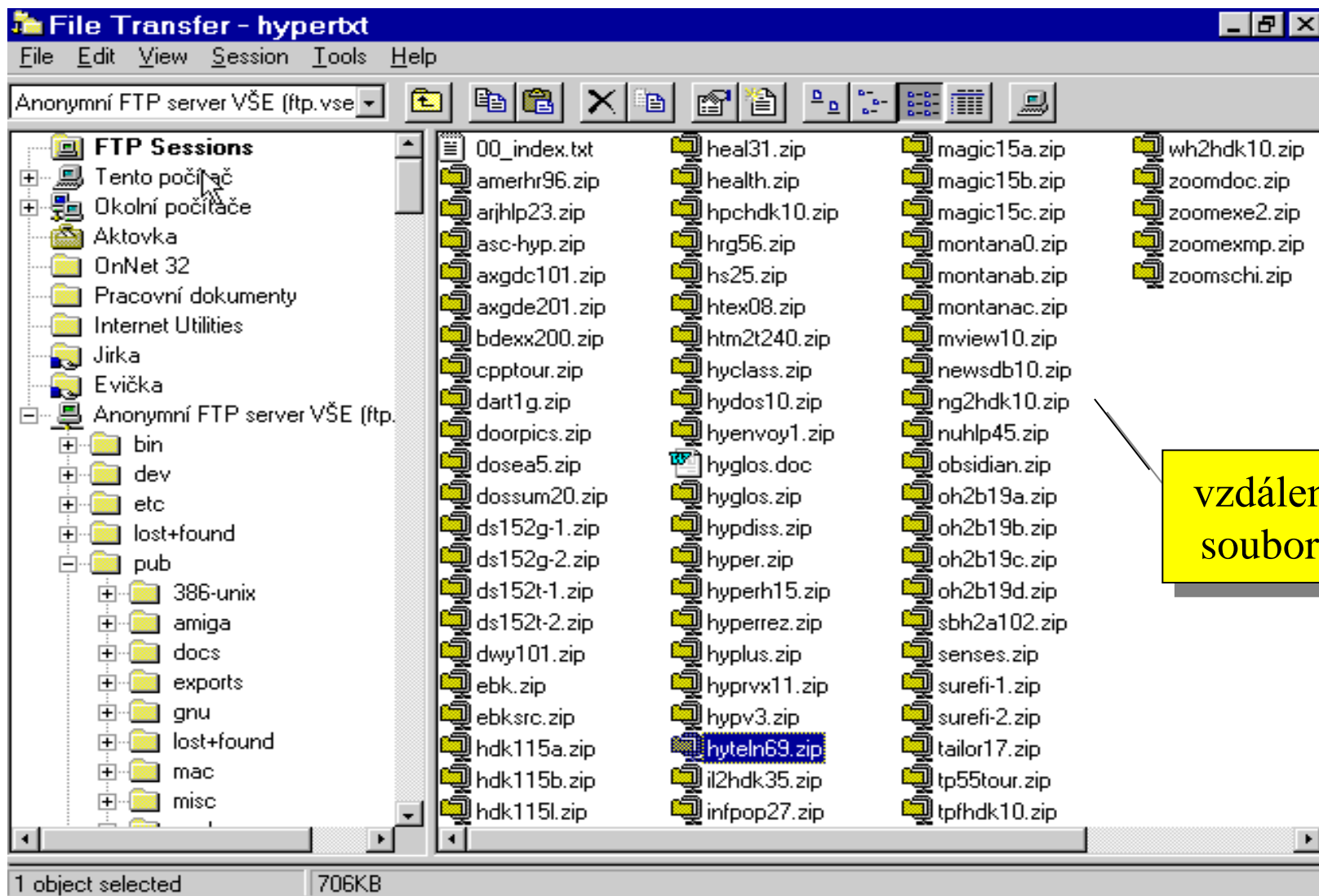
příkaz  
kopírování

soubory  
místního  
počítače

soubory  
vzdáleného  
počítače



# příklad (grafický klient)



# TFTP – Trivial FTP

- existují situace, kdy protokol FTP není nejvýhodnější:
  - např. pro tzv. bootstrap bezdiskových stanic je příliš složitý
  - pro některé jednoduché OS je problém jej implementovat
- v rámci rodiny TCP/IP existuje “ořezaná” verze FTP pod názvem **TFTP** (Trivial FTP)
  - používá se hlavně pro “natažení” tzv. boot image při startu bezdiskových stanic
- využívá přenosových služeb protokolu UDP (FTP využívá TCP)
  - TFTP si spolehlivost zajišťuje sám,
    - využívá jednotlivé potvrzování
    - přenáší data po blocích velikosti 512 bytů
- TFTP nezná pojem uživatele
  - nezajišťuje žádné přihlašování na vzdáleném počítači
    - ponechává na implementaci, jak se vyřeší přístupová práva.
    - Obvykle: pro TFTP dostupné je to, co je dostupné pro všechny uživatele
- TFTP nezajišťuje na vzdáleném počítači žádné systémové akce typu ls, cwd, rm apod.
  - nezná pojem aktuálního adresáře
- uživatel musí explicitně zadat úplnou přístupovou cestu k souboru, který má na mysli (a musí jej znát)

# Protokol NFS (Network File System)

- protokol pro transparentní sdílení souborů (file sharing)
  - v rámci TCP/IP není jediný, ale je nejrozšířenější
- má jiný původ než “klasické” protokoly TCP/IP
  - pochází od firmy Sun Microsystems (původně byl proprietárním řešením)
  - posléze byl NFS předložen IAB (IETF) ke standardizaci
  - dnes je standardem (RFC 1094) a jeho specifikace jsou public domain
- vznikl v prostředí Unixu (SunOS, na bázi BSD Unixu)
  - je ale koncipován jako otevřený (jako univerzální síťové rozšíření systémů souborů na různých platformách)
- není vázán ani na SunOS, ani na Unix jako takový
  - dnes je implementován snad na všech platformách
- připouští, aby klient i server stáli na různých platformách

# Protokol NFS (Network File System)

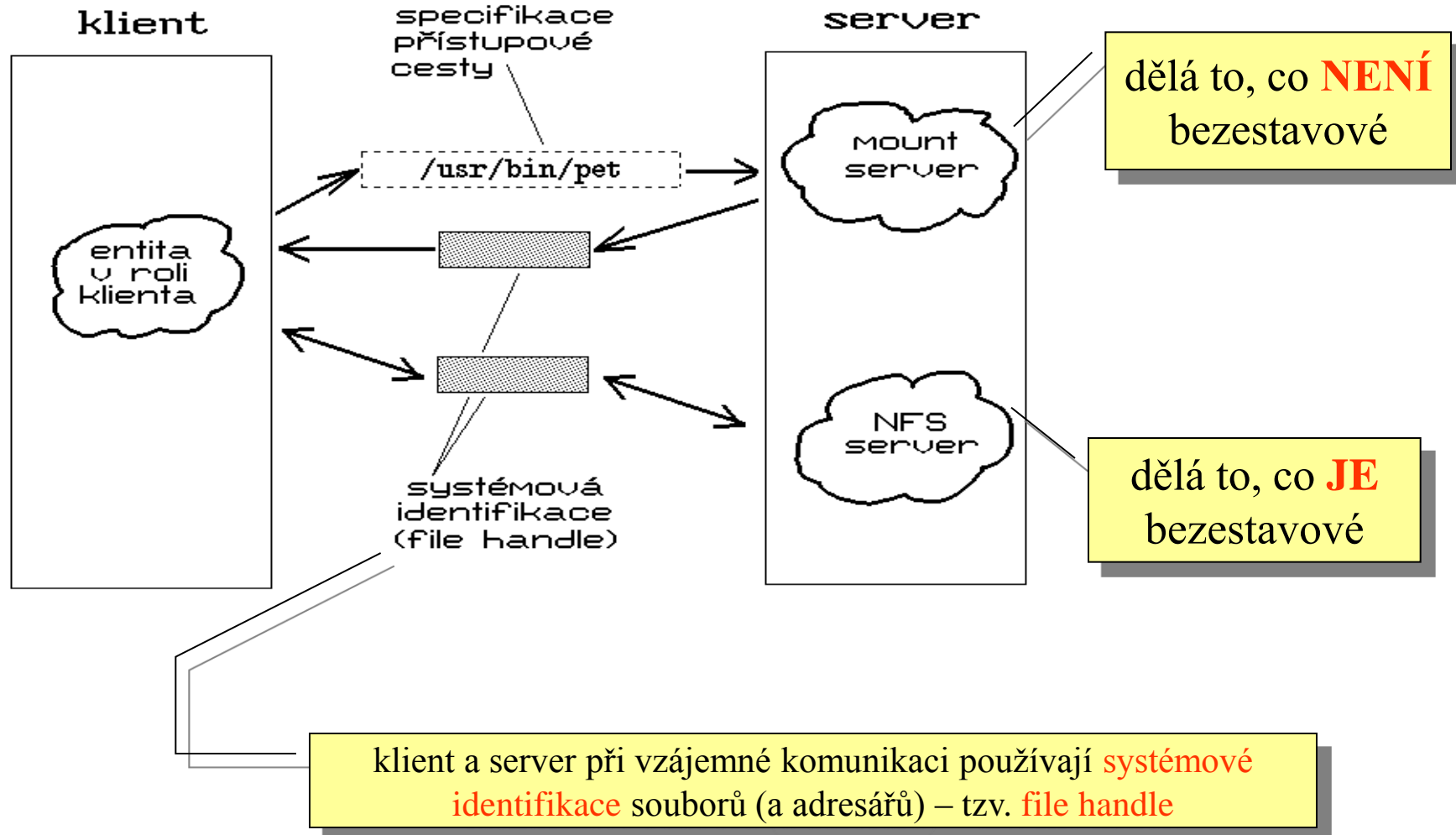
- NFS je bezstavovým protokolem
  - každý jednotlivý požadavek klienta vůči serveru je “uzavřený”, před a po provedení příkazu se server nachází ve stejném stavu
  - velmi to zjednodušuje zajištění korektnosti komunikace klienta a serveru (reakci na nestandardní situace, výpadky, ..)
- bezstavovost NFS je klíčem k **velké robustnosti NFS**
  - není nutné ošetřovat různé výpadky a singularity
- bezstavový charakter připouští pouze použití **idempotentních** operací mezi klientem a serverem (*takových, které lze vícekrát opakovat se stejným efektem*)
  - nelze např. používat příkazy typu “pošli mi další část souboru XY”
  - mohou to být pouze příkazy typu “pošli mi M bytů souboru XY, počínaje bytem N”

důvod úspěšnosti NFS

# problémy kolem NFS

- je možné realizovat všechny požadavky na přístup k souborům pomocí idempotentních operací?
  - NE, nelze např. provést OPEN (otevření), a soubor ponechat otevřený, obdobně pro CLOSE
  - NFS řeší tak, že pro potřeby vyřízení každého jednotlivého požadavku soubor nejprve otevře, a pak jej zase ihned zavře
- některé případy nelze obejít
  - jako u OPEN a CLOSE
  - např. APPEND (přidání za aktuální konec souboru)
    - **NFS řeší zákazem takovýchto operací**
- NFS si klade za cíl být použitelný na různých platformách
  - nemůže se proto vázat na konvence žádné specifické platformy
  - při odkazech na soubory nemůže používat specifikace přístupových cest typu /usr/bin/neco.txt (které jsou závislé na platformě)
    - **přístupové cesty sestavuje vždy až klient podle místních konvencí, server používá vždy jen "jednorozměrná" jména souborů a adresářů**
  - počáteční "přimontování" (mount) části adresářového stromu není bezestavové
    - **NFS řeší vyčleněním do tzv. MOUNT SERVERU**

# představa



# systemová identifikace - handle

- pro odkazy na konkrétní soubory se v rámci NFS používají tzv. systémové identifikace (file handles)
  - pro klienta je handle identifikátorem (dále nedělitelnou posloupností bitů)
  - pro server handle obsahuje tři složky, identifikující
    - systém souborů
    - soubor
    - instanci souboru
- systémová identifikace (handle) jednoznačně identifikuje buď soubor, nebo adresář
  - systémové identifikace přiděluje server, klient je pouze používá
  - systémové identifikace mají absolutní povahu (a nikoli relativní, NFS nezná pojem aktuálního adresáře)
- systémová identifikace (handle) je ukazatelem na jedno konkrétní místo v rámci systému souborů serveru
  - když klient vlastní systémovou identifikaci adresáře, může si vyžádat výpis jeho obsahu
  - součástí výpisu je seznam znakových řetězců, popisujících jednotlivé soubory a podadresáře
  - klient si může vyžádat na serveru systémovou identifikaci zadaného souboru v adresáři (či identifikaci podadresáře) ....
    - ... serveru přitom musí předat systémovou identifikaci adresáře, a znakový řetězec popisující soubor či podadresář

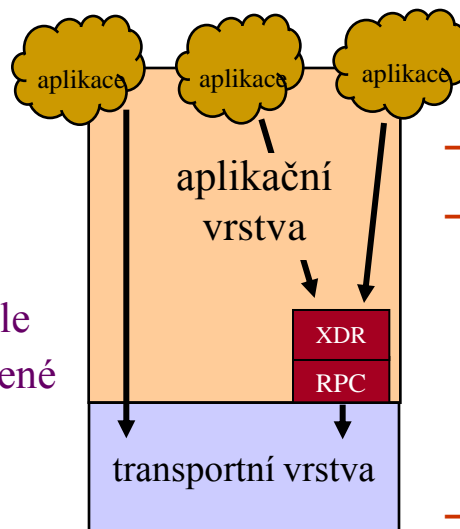
# MOUNT server

- klient musí získat alespoň jeden vstupní bod do systému souborů serveru (alespoň jednu systémovou identifikaci)
  - pak má k dispozici prostředky pro procházení celého stromu (systému souborů serveru)
- poskytnutí "prvního" handle ale není v silách NFS serveru
  - vyžaduje specifikaci přístupové cesty
- **první handle poskytuje MOUNT server !!**
- MOUNT server řeší další činnosti, které také nemůže zajišťovat NFS server
  - vede evidenci zpřístupňovaných (exportovaných) adresářových stromů
  - počáteční "přimontování" adresářového stromu
    - včetně ověření identity uživatele a jeho přístupových práv
- MOUNT server je typicky řešen na aplikační úrovni
  - nezáleží na jeho rychlosti
  - NFS server bývá součástí jádra a je optimalizován na rychlost



# implementace NFS

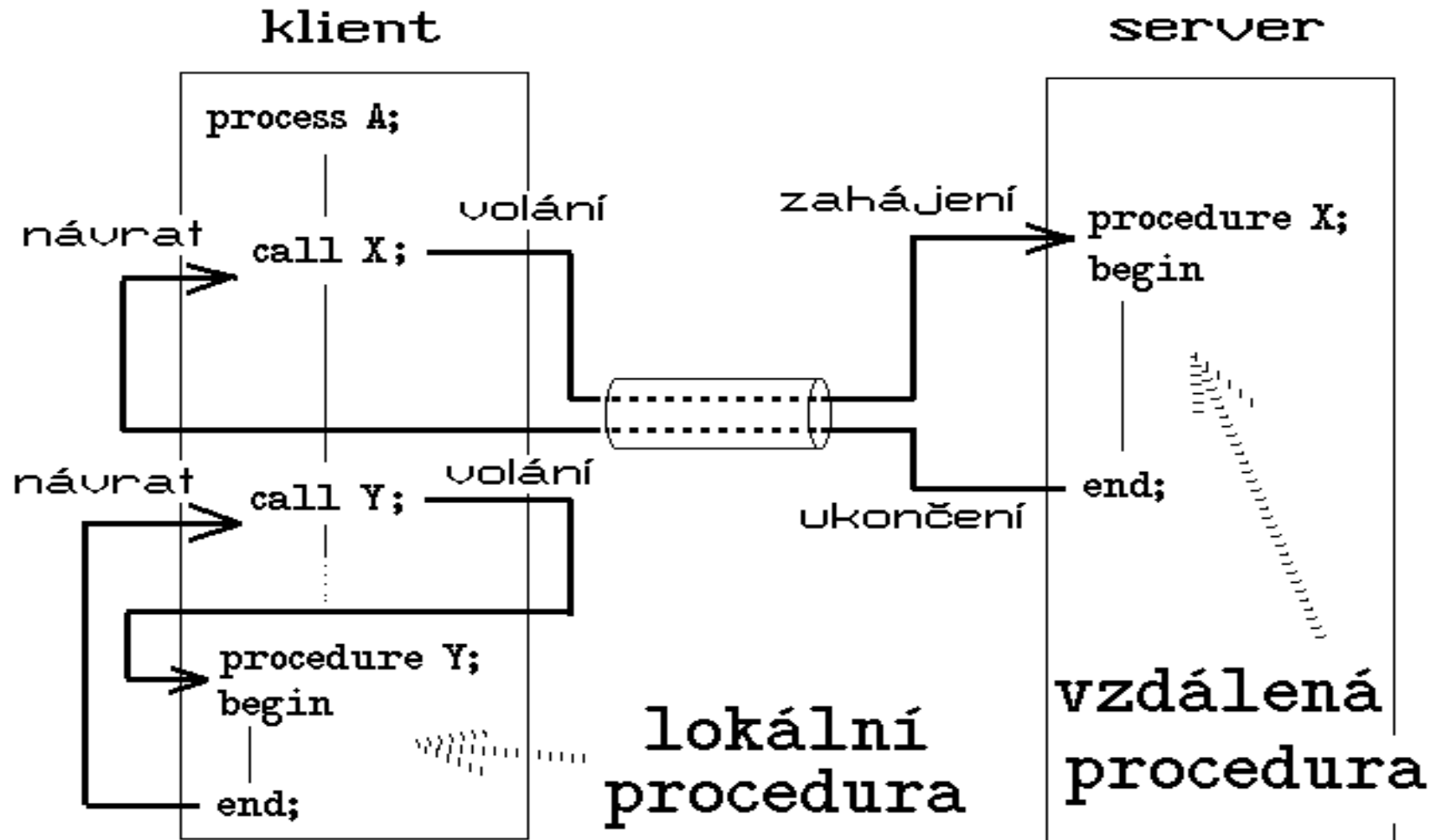
- poprvé významněji použít koncept tzv. vzdáleného volání procedur
  - prostřednictvím protokolu RPC - Remote Procedure Call)
  - jde v zásadě o změnu na úrovni programování
    - programátor nemusí řešit asynchronní komunikaci, ale pouze volá předem připravené procedury knihovního charakteru
- implementace RPC je řešena tak, že může být využita i samostatně
  - mimo implementaci NFS
- dále je využit také protokol XDR (eXternal Data Representation)
  - samostatný standard, který definuje společný přenosový mezikvar
  - je standardem v rámci TCP/IP
  - definuje jednotný způsob reprezentace přenášených dat, nezávislý na konkrétní architektuře příjemce a odesilatele
  - definuje jazyk pro nezávislý popis těchto dat
- také XDR je implementováno samostatně, jako RPC



# RPC – Remote Procedure Call

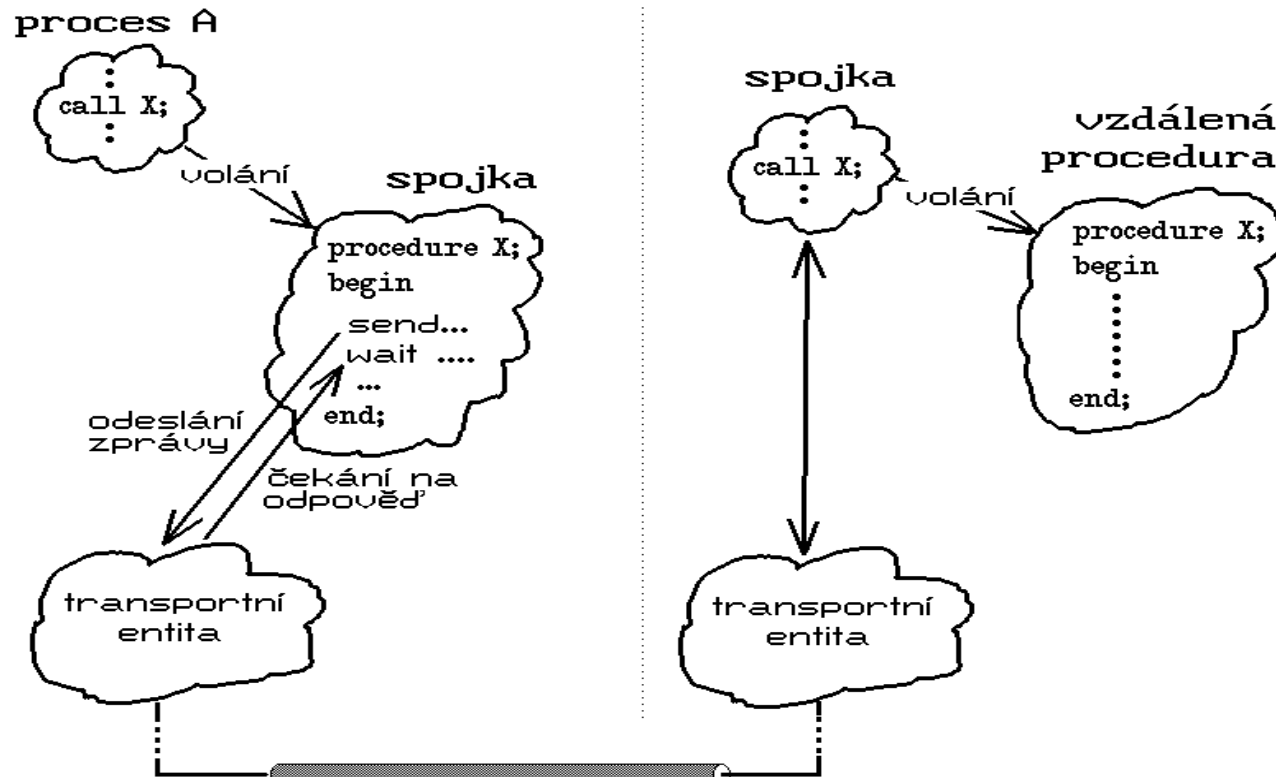
- klasické řešení (bez RPC):
  - klient zformuluje svůj požadavek, sestaví jej do tvaru zprávy a odešle
    - klient čeká (na asynchronní událost - příchod zprávy s odpovědí) - např. je suspendován
  - zpráva s odpovědí přichází, klient je aktivován
- důsledky:
  - klient si uvědomuje, že:
    - pracuje v distribuovaném síťovém prostředí
    - některé akce probíhají na vzdáleném počítači (předem neznámou rychlostí)
  - klient se musí přizpůsobovat asynchronní povaze komunikace
    - posílat zprávy, čekat na odpovědi, ...
    - **nezapadá to do zásad strukturovaného programování !!!**
- řešení s RPC:
  - RPC vytváří klientovi iluzi, že všechny akce probíhají “u něj”, a mají formu volání lokálních procedur
    - každá akce končí v okamžiku výstupu (návratu) z příslušné procedury - klient se pak nemusí explicitně zabývat čekáním
    - klient předává parametry operací jako parametry volané procedury
      - a nikoli jako data, vkládaná do zpráv dle zásad příslušného komunikačního protokolu
  - klient si nemusí uvědomovat, že pracuje v prostředí sítě
- výhody:
  - výrazné zjednodušení implementace klienta (i serveru)
    - v podstatě je pod klienta “podstrčena” další vrstva (vrstva RPC), která přijímá požadavky na volání procedur, “balí” je do zpráv a zprostředkovává jejich skutečné provedení na jiném uzlu
    - klient tak volá procedury, které jsou ve skutečnosti prováděny na vzdáleném uzlu

# RPC – představa fungování



# princip implementace RPC

- klient ve skutečnosti volá procedury (podprogramy)
  - charakteru knihoven
    - jsou součástí implementace mechanismu RPC
  - tyto procedury se označují jak **spojky** (stubs) ...
    - ... a komunikují s partnerskými spojkami (stubs) na druhé straně, které pak skutečně zajistí požadované volání



# předávání parametrů v RPC

- spojky (stubs) přebírají své vstupní parametry takovým způsobem, jaký je v daném prostředí při volání procedur obvyklý
  - a převádí do takového tvaru, který je vhodný pro jejich odeslání druhé straně
    - provádí tzv. **marshalling, serializing**
  - při příjmu výsledků analogicky převádí získané odpovědi na výstupní parametry
- pro potřeby implementace jsou všechny vzdálené procedury koncipovány tak, aby měly právě jeden parametr
  - odkaz na datovou strukturu, ve které jsou všechna vstupní data
- pro přenos se tato data převádí do jednotného přenosového formátu
  - je předem stanoven, a všechny implementace RPC mu musí rozumět

značně netriviální, viz např. převod pointerů