



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



Lekce 7: Techniky přenosu dat

Jiří Peterka, 2009

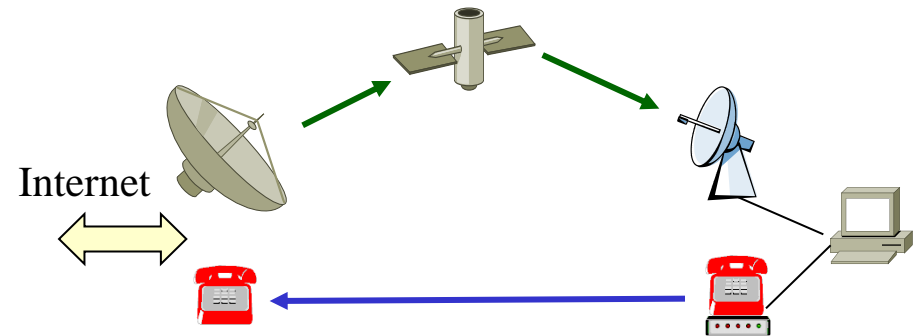
co jsou "techniky přenosu dat"?

- obecně:
 - všechno, co se týká samotného přenosu dat
 - způsoby, postupy,
- patří sem:
 - **paketový přenos**
 - přenos dat na principu přepojování paketů – bylo dříve
 - první přednáška, jako jedno ze základních "paradigmat"
 - podobně: přenos buněk, přenos na principu přepojování okruhů
 - **spolehlivý a nespolehlivý přenos**
 - **spojovaný a nespojovaný přenos**
 - **přenos "best effort" a s QoS**
 - vše bylo dříve (1. přednáška)
- dále také:
 - **simplexní, duplexní a poloduplexní přenos**
 - jak je to s přenosem v různých směrech
 - **asynchronní, arytmičkový a synchronní přenos**
 - jak je to se vzájemnou synchronizací příjemce a odesilatele
 - **izochronní přenos**
 - je přenos v reálném čase?
 - **zajištění transparency dat**
 - kdy jsou přenášená data příkazy a kdy "čistá data"
 - **framing (zajištění synchronizace na úrovni rámců, paketů, buněk, ...)**
 - jak správně rozpoznávat celé rámce, pakety ...
 - **detekce chyb**
 - **zajištění spolehlivosti přenosu**
 - **řízení toku**

simplex, duplex, poloduplex

- týká se možnosti přenosu v obou směrech
- (plně) **duplexní** přenos:
 - je možný v obou směrech, a to současně
- **poloduplexní** přenos:
 - je možný v obou směrech, ale nikoli současně
- **simplexní** přenos:
 - je možný jen v jednom směru
 - příklady:
 - optické vlákno bez WDM
 - digitální TV vysílání systémem DVB-T
 - obecně R a TV vysílání, jednosměrné satelitní přenosy
- týká se komunikace obecně:
 - nejde jen o to, co umožňuje přenosové médium
 - jde také o způsob využití
 - nad plně duplexní přenosovou cestou lze komunikovat např. jen poloduplexně
 - stylem: otázka - odpověď

- další varianty:
- **semiduplex (dusimplex)**: když je přenos každým z obou směrů realizován jinak
 - na jiných frekvencích, jinou cestou, jinou technologií
 - příklady:
 - jednosměrné satelitní připojení k Internetu
 - technologie DirecTV, zpětný kanál realizován "pozemní" cestou

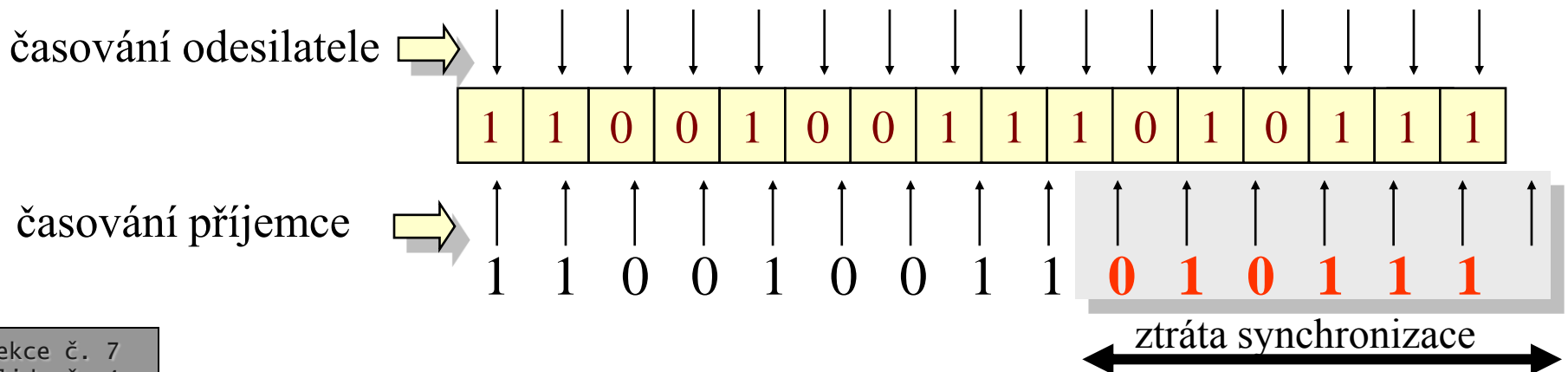


- **asymetrický** přenos:
 - když jsou (maximální, nominální) rychlosti v obou směrech různé
 - příklad:
 - ADSL (Asymmetric DSL),
 - poměr rychlostí daný technologií je cca 1:10
 - poměr rychlostí v rámci komerčních nabídek je (dnes, obvykle) 1:4

problematika synchronizace

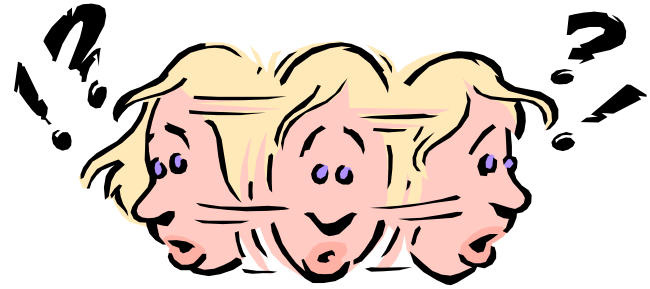
- každý bit je přenášán v rámci určitého bitového intervalu
 - tj. přenos bitu není "okamžitý", ale trvá určitou dobu (bitový interval)
 - přenášená data reprezentuje stav signálu během bitového intervalu
- příjemce vyhodnocuje stav přenášeného signálu "někde" v rámci bitového intervalu
 - rozhodující je okamžik vyhodnocení signálu
 - na základě vyhodnocení okamžitého stavu signálu pak usuzuje na to, jaká data jsou přenášena

- problém synchronizace:
 - příjemce se musí "strefit" do správného bitového intervalu
 - jinak přijme nesmyslná data
- zjednodušení:
 - odesílatel i příjemce odměřují odesílaná data podle vlastních hodin.
 - *tyto hodinky musí "tikat" dostatečně souběžně (synchronně)*
 - musí být tzv. v synchronizaci

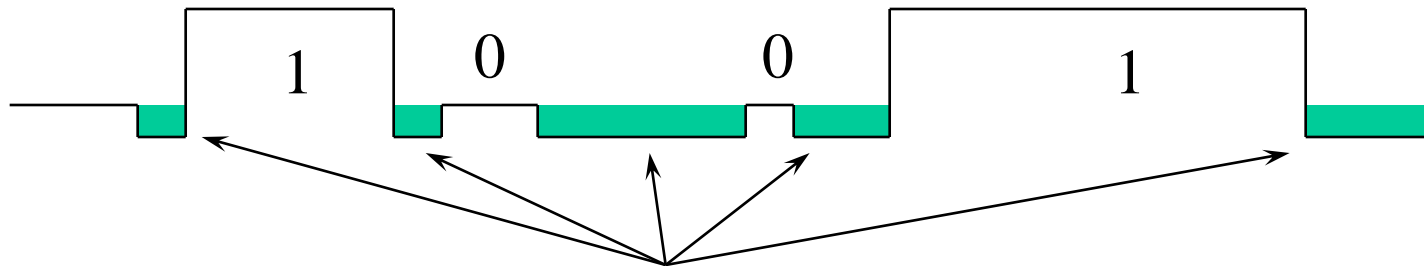


asynchronní přenos

- jeden z možných způsobů zajištění synchronizace
- a-synchronní = zcela postrádá jakoukoli synchronizaci
 - bitový interval nemá konstantní délku
 - začátek i konec každého bitového intervalu musí být explicitně vyznačen
 - je k tomu potřebná alespoň tříhodnotová logika



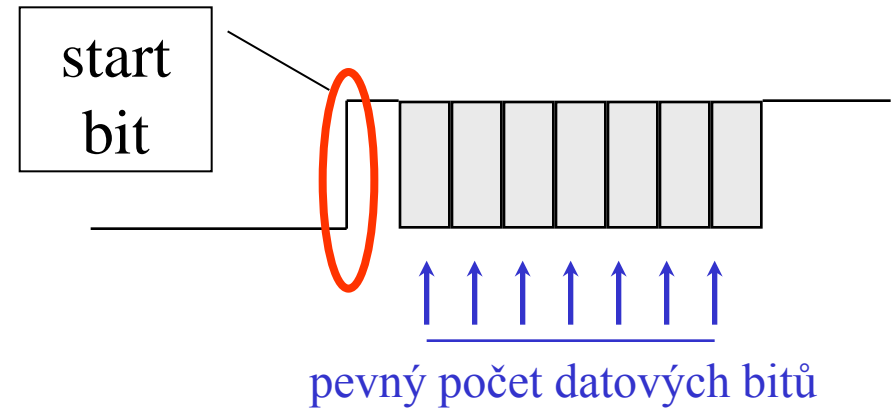
- terminologický problém:
 - když se dnes řekne "asynchronní", nemyslí se tím tato varianta !!!
 - ale to, co je správně označováno jako "arytmické" !!!
 - tato varianta se dnes prakticky nepoužívá



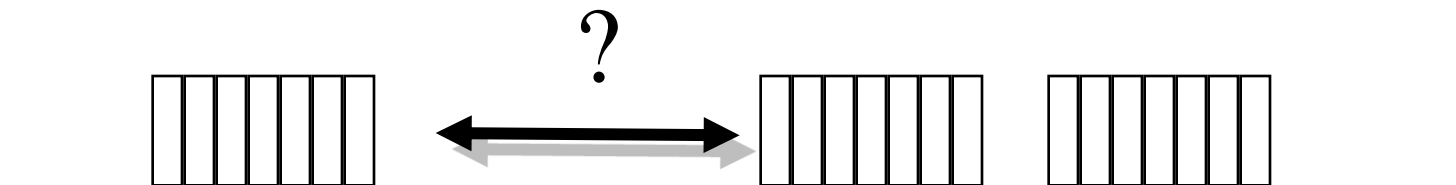
oddělovače bitových intervalů

arytmický přenos

- arytmický přenos:
 - snaží se přenášet celé skupiny bitů, tvořící tzv. znaky
 - 4 až 8 bitů (dnes spíše 8 bitů)
 - na začátku každého znaku je tzv. start-bit
 - slouží k tomu, aby si příjemce "seřídil své hodinky"
 - předpoklad:
 - po seřizení na začátku každého znaku budou hodinky příjemce "tikat" po celou dobu trvání daného znaku
 - tj. příjemce bude správně vyhodnocovat jednotlivé bity v rámci znaku



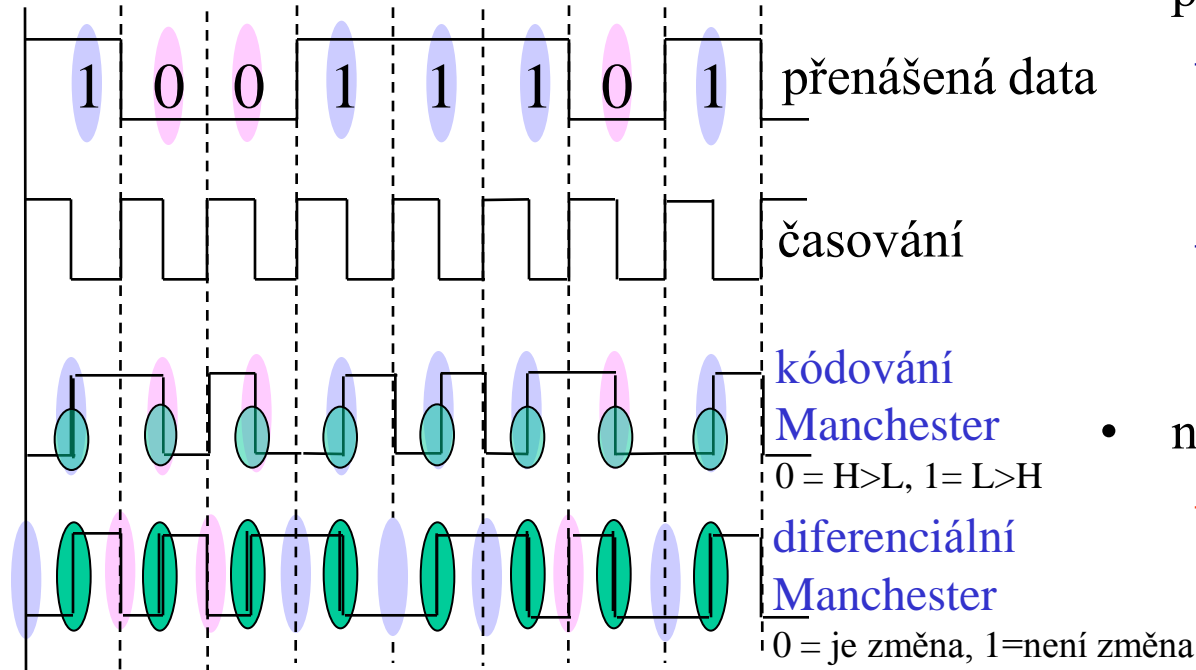
- časové prodlevy mezi jednotlivými znaky mohou být různě velké !!!
 - proto a-rytmický přenos: chybí mu rytmus přenosu jednotlivých znaků
 - během prodlevy mezi znaky se hodinky příjemce mohou libovolně "rozejít"
 - na začátku dalšího znaku budou znovu "zasynchronizovány" pomocí start bitu





synchronní přenos

- synchronní přenos:
 - synchronizace je udržována trvale
 - přenáší se celé souvislé bloky dat
 - libovolně velké!!
 - synchronizace se udržuje po celou dobu přenosu souvislého bloku
 - někdy se udržuje i mezi bloky
 - jindy se mezi bloky neudržuje
 - hodinky odesílatele i příjemce se se mezi bloky mohou "rozejít", a na začátku nového bloku se zase "zasynchronizovat"
- výhody:
 - synchronní přenos je obecně rychlejší než asynchronní a arytmičtější
 - používá se na vyšších rychlostech
- způsob zajištění trvalé synchronizace:
 - bloky jsou libovolně dlouhé = již není možné se spoléhat na to, že hodinky příjemce "vydrží"
 - a "nerozejdou se" během přenosu bloku
- synchronizaci je třeba udržovat průběžně
 - průběžně seřizovat hodinky příjemce během celého přenosu
 - možnosti:
 - skrze samostatný "synchronizační" (časovací) signál
 - přenáší "tikání" hodin odesílatele
 - příliš nákladné, samostatný signál není k dispozici
 - skrze redundantní kódování
 - zahrnutí časového signálu do kódování jednotlivých bitů
 - příklad: kódování Manchester
 - synchronizací z dat

příklad – časování spolu s daty

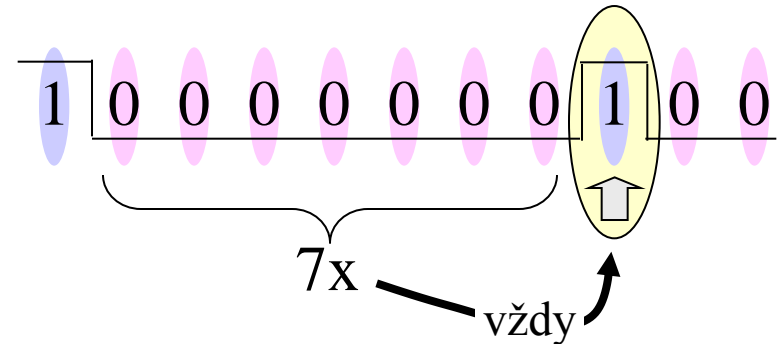


- představa:
 - časování se smíchá s daty
 - sloučí (sečte) se datový a časovací signál
 - příjemce využije "časovací část" pro průběžné seřizování svých hodin
- nevýhoda:
 - modulační rychlost (i šířka pásma) je 2x vyšší než přenosová rychlost !!!
 - na 1 bit jsou 2 změny signálu

- příklad: kódování Manchester (např. u Ethernetu)
 - uprostřed každého bitového intervalu je vždy hrana, která "nese" data (svou polaritou)
 - současně tato hrana může sloužit i pro potřeby synchronizace 
 - vyskytuje se vždy, bez ohledu na hodnotu dat
- příklad: kódování "diferenciální Manchester" (např. u Token Ringu)
 - uprostřed každého bitového intervalu je hrana, slouží pouze potřebám časování 
 - data "nese" hrana/absence hrany na začátku bitového intervalu
 - data nesdílí stejnou hranu s časováním !!!

příklad: synchronizace z dat

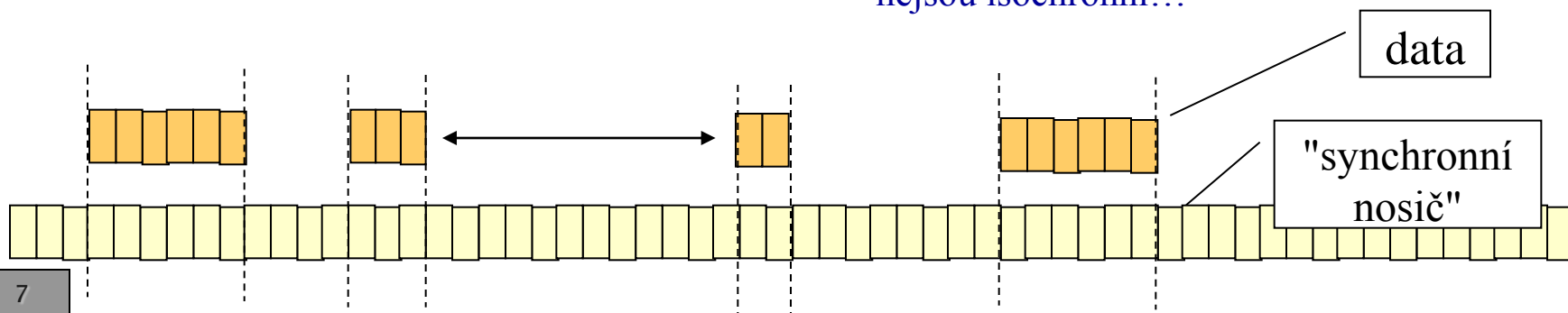
- myšlenka:
 - přenášený signál nebude obsahovat žádné časování
 - příjemce si průběžně seřizuje hodinky podle datových bitů
 - v okamžiku výskytu hrany která signalizuje bit
- problém:
 - mohou se vyskytnout dlouhé posloupnosti bitů, které negenerují žádnou změnu přenášeného signálu
 - hodinky příjemce by mohly ztratit synchronizaci
- řešení:
 - technika **bit-stuffing** (vkládání bitů)
 - pokud by se vyskytla příliš dlouhá sekvence bitů, která by mohla způsobit ztrátu synchronizace, odesílatel vloží do odesílaných dat vhodný bit, který vyvolá hranu
 - a příjemce ji zase odstraní
 - používá se i jinde
 - pro tzv. framing, u bitově orientovaných protokolů



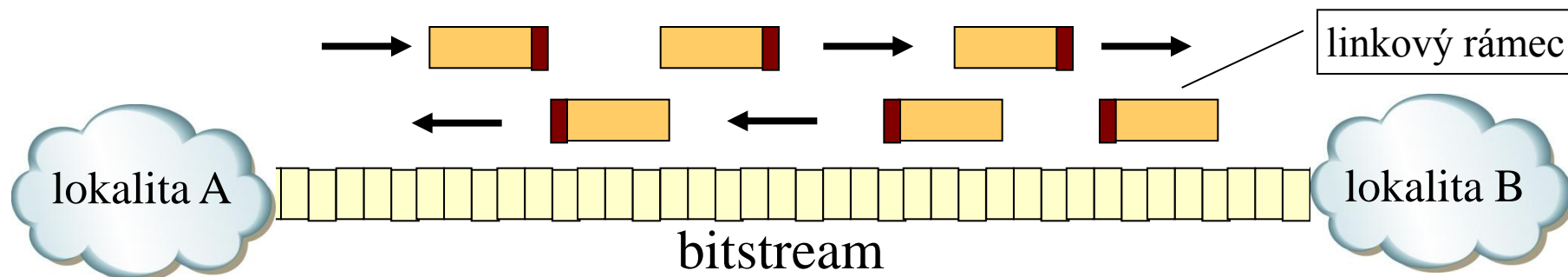
- příklad (techniky bit-stuffing):
 - je známo, že hodinky příjemce "vydrží" 7 bitových intervalů bez synchronizace
 - při 8 a více by se již rozešly
 - řešení:
 - na straně odesílatele: za každou šestou (po sobě jdoucí) nulu zařadí jeden jedničkový bit
 - na straně příjemce: po každých 7 souvislých nulách smaže následující jedničku
 - "spotřeba"
 - zvyšuje se tím počet přenesených bitů,
 - již to není 1 bit = 1 změna přenášeného signálu
 - ale je to velmi blízko (limitně rovno)

isochronní přenos

- isochronní:
 - = "probíhající ve stejném čase"
 - vhodné (nutné) pro multimediální přenosy
 - obraz, zvuk
 - může být určité přenosové zpoždění
 - např. až 500 ms
 - ale je požadována vysoká pravidelnost !!
 - přenosové zpoždění je konstantní a nemění se !!!
- důsledky "isochronnosti":
 - data mají zaručeno, za jak dlouho se dostanou ke svému cíli
 - nemusí to být "ihned", ale je to pravidelně
- představa:
 - jdou to asynchronní data, vkládaná do synchronního přenosového mechanismu
 - například do časových slotů, event. přímo do bitových intervalů
 - podstatné je:
 - mezi jednotlivými "částmi" (asynchronních) dat jsou vždy celistvé násobky prázdných slotů intervalů)
- příklady:
 - přepojování okruhů je (může být) isochronní
 - časový multiplex (TDM) zachovává isochronní charakter
 - statistický multiplex a přepojování paketů nejsou isochronní!!!



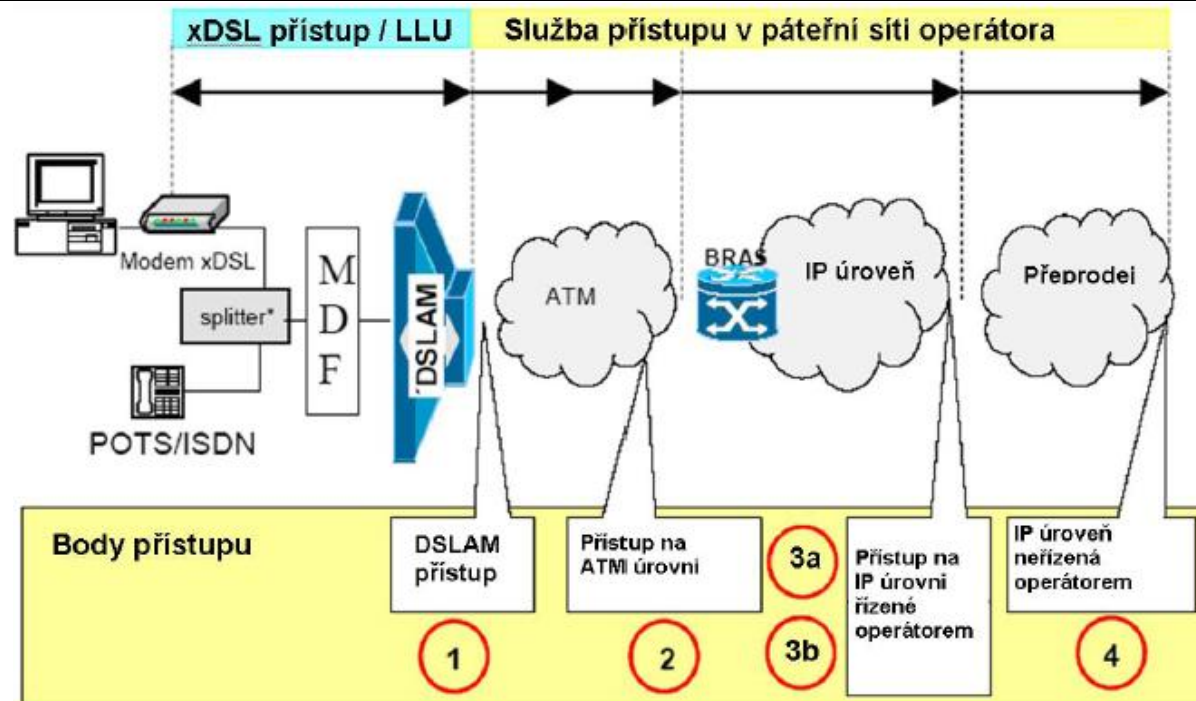
bitstream (bitový proud)



- tzv. bitstream (bitový proud) je telekomunikační služba
 - synchronní přenos bitů mezi dvěma lokalitami
 - má konstantní přenosovou rychlost
 - a konstantní přenosové zpoždění
- lze jej chápat jako službu fyzické vrstvy
 - službu charakteru odešli/přijmi bit
 - se synchronním způsobem fungování
- bitstream (bitový proud) je vhodným "podložím" pro přenosové služby vyšších úrovní
 - nad bitovým proudem lze realizovat přenos (linkových) rámečů
- nad bitovým proudem lze realizovat různé druhy přenosů:
 - paketový / best effort
 - isochronní
 - s QoS

bitstream na různých úrovních

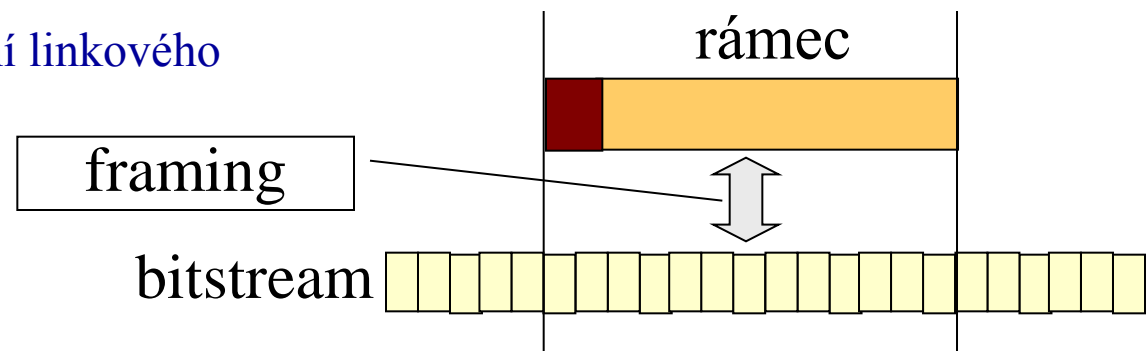
- v poslední době se hovoří o různých úrovních bitstreamu
 - na úrovni DSLAM
 - tradiční pojetí
 - "to, co vychází z DSLAM-u"
 - na úrovni ATM
 - přesněji: před agregačním směrovačem, ještě lze považovat za bitstream
 - na úrovni IP
 - nemá základní charakteristiky bitstreamu
 - není isochronní
 - jde o "packet stream"
 - data již jsou "zabalena" do paketů, podléhají agregaci kterou nelze měnit,



- v ČR nabízen jen "bitstream na úrovni IP",
 - ostatní varianty nikoli
 - byť EU po nás požadovala jejich zavedení
- názor: "IP bitstream" není žádným bitstreamem
 - jde spíše o uměle zavedený pojem, který má zastříť absenci skutečného bitstreamu
- oficiální překlad: **datový tok**

framing, aneb: synchronizace na úrovni

- synchronizace **na úrovni bitů**
 - jde o správné rozpoznání jednotlivých bitů (bitových intervalů)
 - to, co jsme až dosud popisovali
- synchronizace **na úrovni znaků**
 - jde o správné rozpoznání celých znaků (u znakově orientovaných přenosů)
 - při asynchronním (arytmickém) přenosu je to dáno start bity
 - při synchronním přenosu je nutné „odpočítávat“ bity
- synchronizace **na úrovni rámců**
 - alias tzv. **framing**
 - jde o správné rozpoznání linkového rámce
 - začátku, konce atd.
- příklady:
 - **znakově orientované linkové protokoly:**
 - přenáší data členěná na znaky
 - pro vyznačení začátku/konce používají speciální znaky ASCII sady
 - **bitově orientované linkové protokoly:**
 - přenáší data jako posloupnosti bitů
 - nečlení je na znaky
 - pro vyznačení začátku/konce využívají speciálních bitových posloupností
 - tzv. křídlových značek

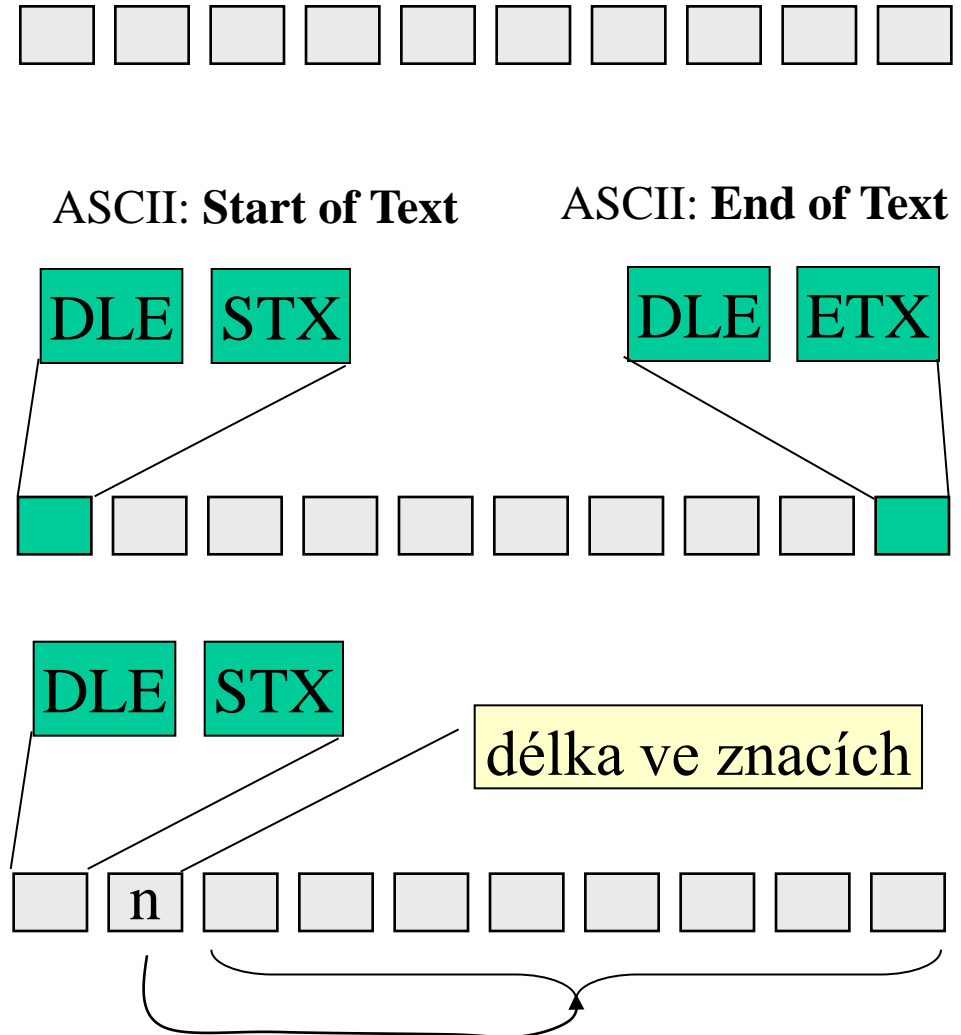


zajištění transparence dat

- související problém:
 - jak vždy spolehlivě poznat, která data jsou:
 - **řídící** (hlavičky, patičky, příkazy atd.) a mají být interpretována
 - **"užitečná data"** a nemají být nijak interpretována
- možné základní přístupy:
 - samostatné přenosové kanály pro řídicí příkazy a pro data
 - někdy je možné, někdy ne
 - sloučení příkazů a dat do jednoho přenosového kanálu
 - častější
 - nutné mít schopnost rozpoznat, kdy se jedná o "užitečná data" a kdy o příkazy
- příklady řešení (se společným přenosovým kanálem):
 - prefixace speciálním ESCAPE znakem
 - před každý znak, který má mít význam řídicího znaku, se umístí speciální "escape" znak
 - např. znak DLE (Data Link Escape) ze sady ASCII
 - případný výskyt speciálního escape znaku v "užitečných datech" se řeší jeho zdvojením
 - příjemce musí druhý výskyt odstranit
 - tzv. **character stuffing**
 - prefixace speciální bitovou posloupností (tzv. křídlovou značkou)
 - používá se u bitově orientovaných protokolů, pro vyznačení začátku (a event. i konce rámce)
 - případný výskyt speciální bitové posloupnosti v užitečných datech se řeší pomocí **bit-stuffingu**

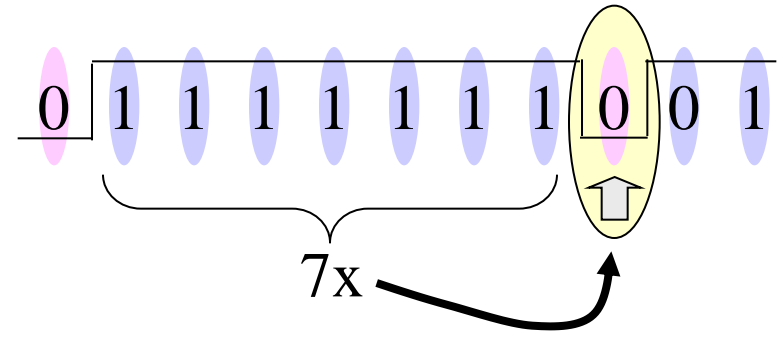
znakově orientovaný přenos

- přenášená data jsou chápána jako posloupnost znaků
 - každý o stejném počtu bitů
- jak rozpoznat začátek a konec?
 - na začátek rámce dát speciální „uzavírací“ znak, a na konec „ukončovací“ znak
 - prefixovaný pomocí znaku DLE
 - na začátek rámce dát speciální "uzavírací" znak, a za něj údaj o délce rámce
 - z roku 1964
- příklad:
 - linkový protokol IBM BiSync

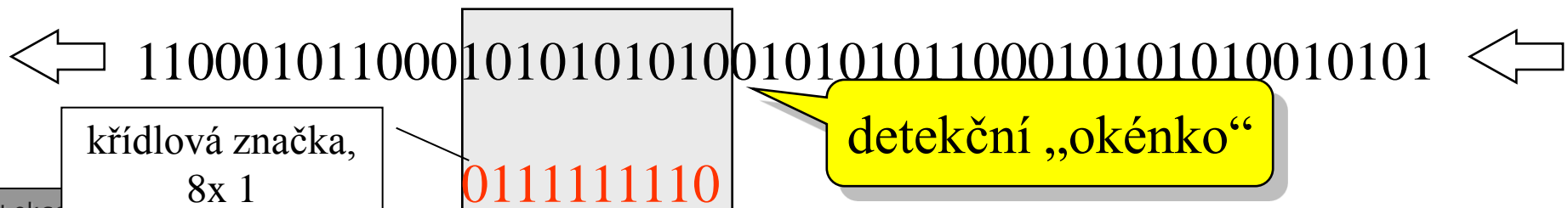


bitově orientovaný přenos

- přenášený text je chápán jako posloupnost bitů
 - tj. přenášená data nejsou členěna na znaky
- představa:
 - v přenášeném řetězci bitů se hledá vzorek (posloupnost, značka), indikující začátek (konec)
 - tzv. křídlová značka
 - výskyt křídlové značky představuje začátek rámce
 - konec může být také označen křídlovou značkou, nebo určen údajem o délce (za úvodní křídlovou značkou)



- zajištění transparency dat:
 - aby se křídlová značka nevyskytla "v datech"
- řeší se pomocí techniky bit-stuffing
 - příklad:
 - tvoří-li křídlovou značku 8 po sobě jdoucích jedniček, pak
 - odesílatel za každých 7 po sobě jdoucích jedniček přidá 0

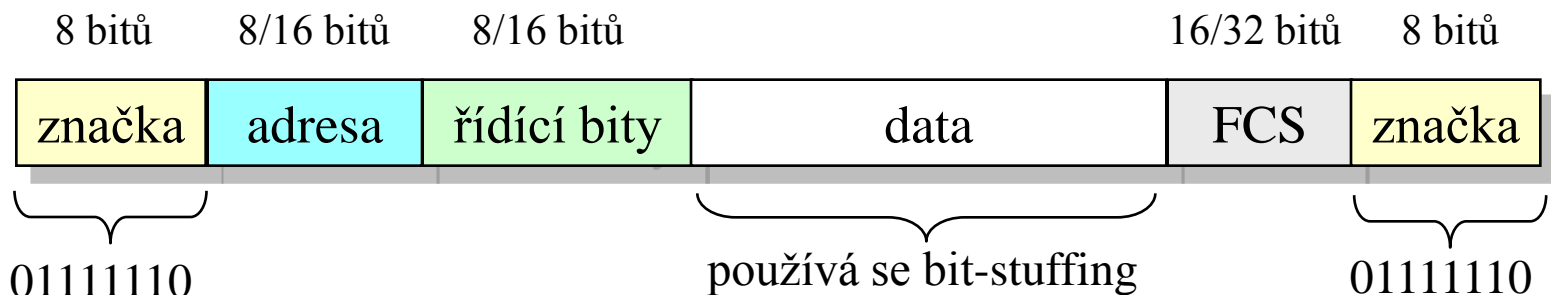


bitově vs. znakově orientované protokoly

- dnes se úrovní linkové vrstvy používají téměř výlučně bitově orientované protokoly
 - kvůli nižší režii na zajištění transparence dat
 - jsou novější
- příklady bitově orientovaných protokolů:
 - **SDLC**
 - vyvinula firma IBM v roce 1975
 - první bitově orientovaný protokol
 - **HDLC**
 - vyvinula ISO v roce 1979 podle SDLC
 - základ všech dnešních bitově orientovaných protokolů
 - funguje poloduplexně nebo duplexně
 - lze použít na dvoubodových i vícebodových spojích

- **LAP (Link Access Protocol)**
 - vyvinula ITU-T od roku 1981, podle HDLC
 - má několik verzí:
 - LAPB
 - » pro B kanály ISDN
 - LAPD
 - » pro D-kanál ISDN
 - LAPM
 - » pro modemy
- **Ethernet**
 - rámce Ethernetu jsou také bitově-orientované
 - značka se říká "preamble" (preamble)
-

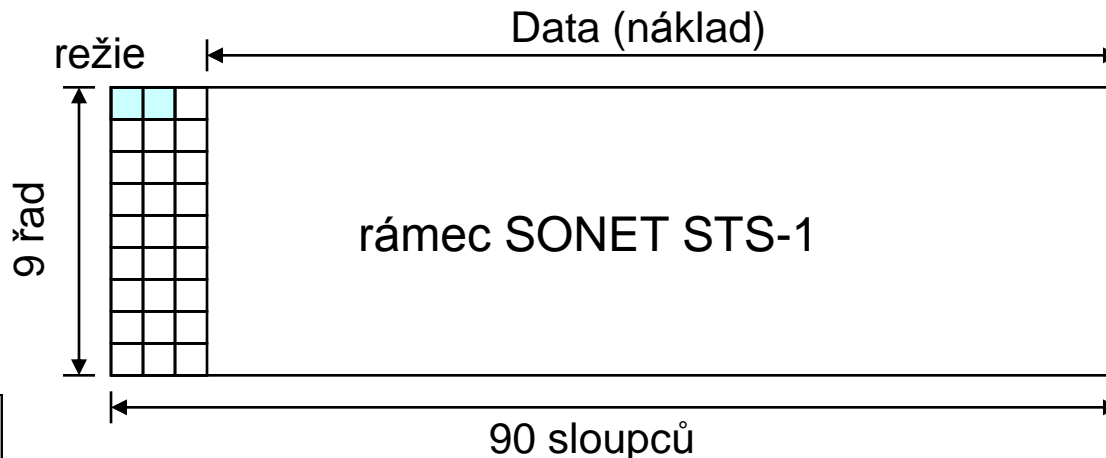
formát rámce
HDLC



rámce s pevnou velikostí

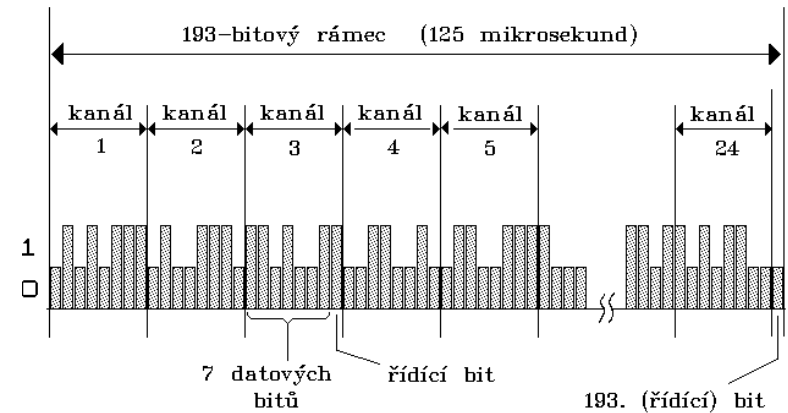
- v telekomunikacích se často používají rámce pevné velikosti
 - nejvíce v rámci digitálních hierarchií
 - PDH, SDH, SONET
 - začátek bloku (rámce) obsahuje speciální bitovou sekvenci
 - údaj o délce/konci není nutný
 - předpokládá se pevná velikost bloku !!!
 - nepoužívá se bit stuffing !!!
 - příjemce zná velikost bloku a další bitovou sekvenci hledá až po "uplynutí" bloku

- příklad: rámec SONET
 - Synchronous Optical Networking
 - rámec má pevnou velikost 810 bytů (90 "sloupců" x 9 "řádek")



- příklad: rámec T1
 - 23x 8 bitů

- pro 23+1 hlasových kanálů, každá z nich potřebuje přenést 8 bitů 8000x za sekundu
- rámec T1 se musí opakovat 8000x za sekundu



- příklad: buňka ATM
 - 5 bytů hlavička
 - 48 bytů náklad

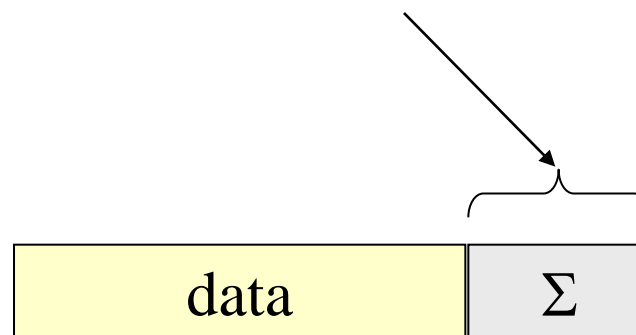
zajištění spolehlivosti

- může být realizováno na kterékoli vrstvě
 - kromě fyzické
- TCP/IP:
 - řeší se až na transportní vrstvě
 - protokol TCP
- RM ISO/OSI:
 - očekává se od všech vrstev, počínaje linkovou vrstvou
- princip a způsob realizace je v zásadě stejný na všech vrstvách
- podmínkou je schopnost detekce chyb
 - schopnost rozpoznat, že došlo k nějaké chybě při přenosu
 - musí být použit vhodný detekční mechanismus
- co dělat, když se zjistí chyba při přenosu?
 - nespolehlivý přenos: nic
- spolehlivý přenos:
 - postarat se o nápravu
- možnosti:
 - použití samoopravných kódů
 - např. Hammingovy kódy
 - problémem je velká míra redundance, která zvyšuje objem přenášených dat
 - současně nízká účinnost
 - používá se jen výjimečně
 - pomocí potvrzování
 - příjemce si nechá znovu zaslat poškozená data
 - podmínkou je existence zpětné vazby / zpětného kanálu
 - alespoň poloviční duplex, aby příjemce mohl kontaktovat odesílatele

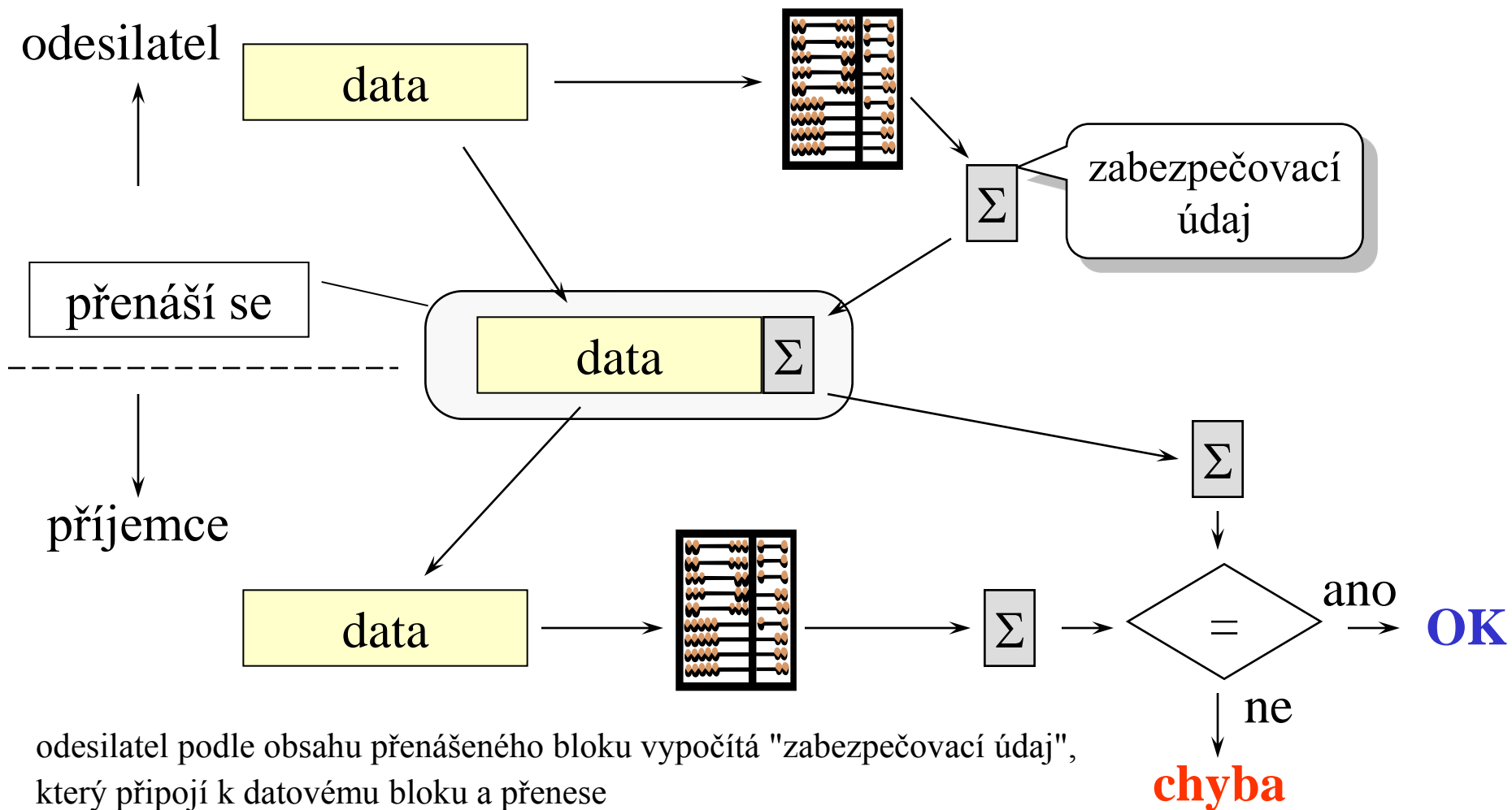
používá se

jak řešit detekci?

- možnosti detekce chyb:
 - parita (příčná a podélná)
 - má nejmenší účinnost
 - kontrolní součty
 - lepší účinnost
 - cyklické redundantní kódy (CRC)
 - zdaleka nejlepší účinnost
- druhy chyb:
 - pozměněná data
 - některé bity jsou změněny
 - shluky chyb
 - celé větší skupiny bitů/bytů jsou změněny nebo ztraceny
 - výpadky dat
 - například ztráta celého rámce
- u synchronních protokolů:
 - stačí detekovat chybu/bezchybnost na úrovni celých rámců/paketů
 - kvůli možnost vyžádat si opakované vyslání
 - to se dělá pro celé bloky
 - informaci o chybě v určitém bytu by nebylo možné využít
 - stejně by se znovu přenesl celý rámec/paket
- obecná představa:
 - k přenášeným datům se připojí "zabezpečovací údaj"



obecná představa



- odesilatel podle obsahu přenášeného bloku vypočítá "zabezpečovací údaj", který připojí k datovému bloku a přenesení
- příjemce znovu vypočítá "zabezpečovací údaj" (stejným postupem) a porovná jej s přijatým zabezpečovacím údajem

parita

- **paritní bit**

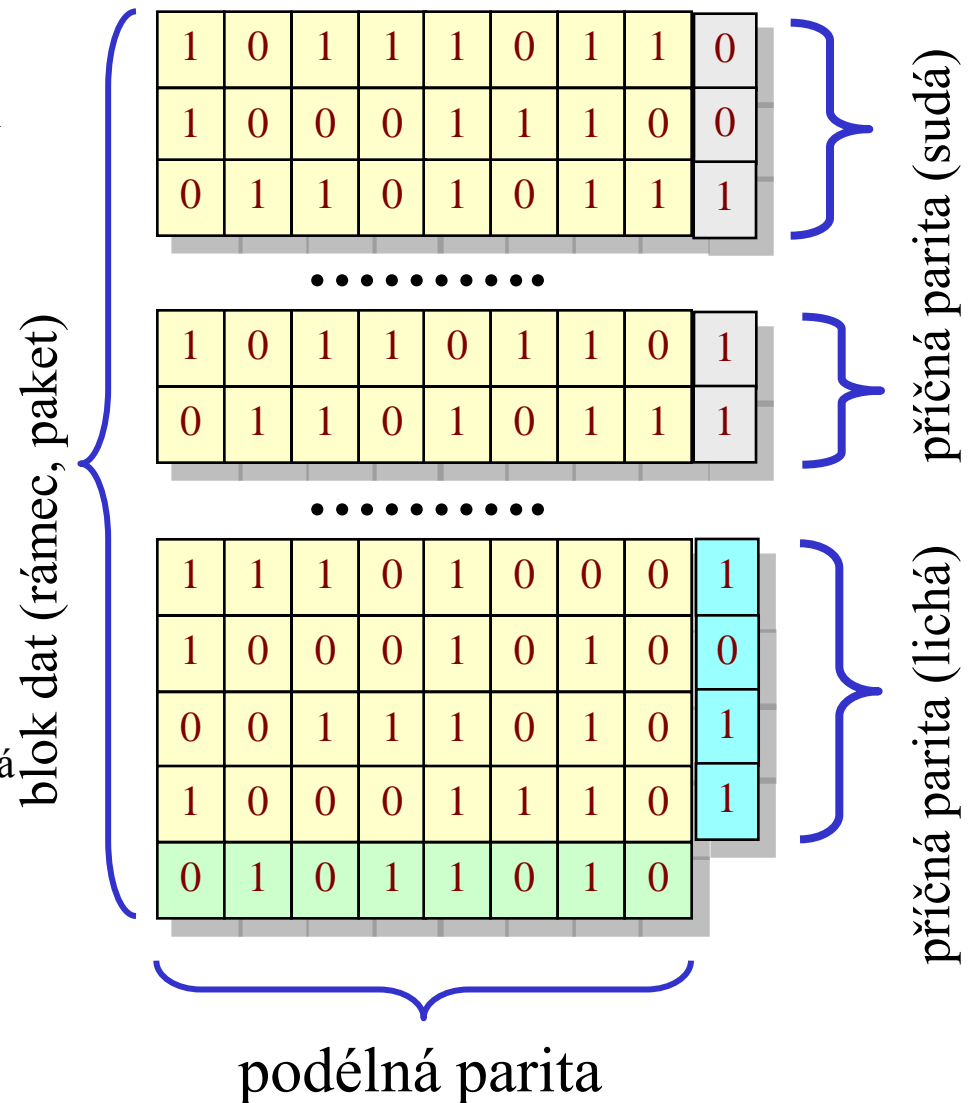
- bit přidán navíc k datovým bitům
- **sudá parita:**
 - paritní bit je nastaven tak, aby celkový počet 1 byl sudý
- **lichá parita:**
 - ... aby byl lichý
- **jedničková parita:**
 - paritní bit pevně nastaven na 1
 - nemá zabezpečující efekt
- **nulová parita**
 - ... nastaven na 0

- **příčná parita:**

- **po jednotlivých bytech/slovech**
 - informace o tom, který byte (slovo) je poškozen, je nadbytečná
 - stejně se znovu posílá celý blok (rámeček, paket)

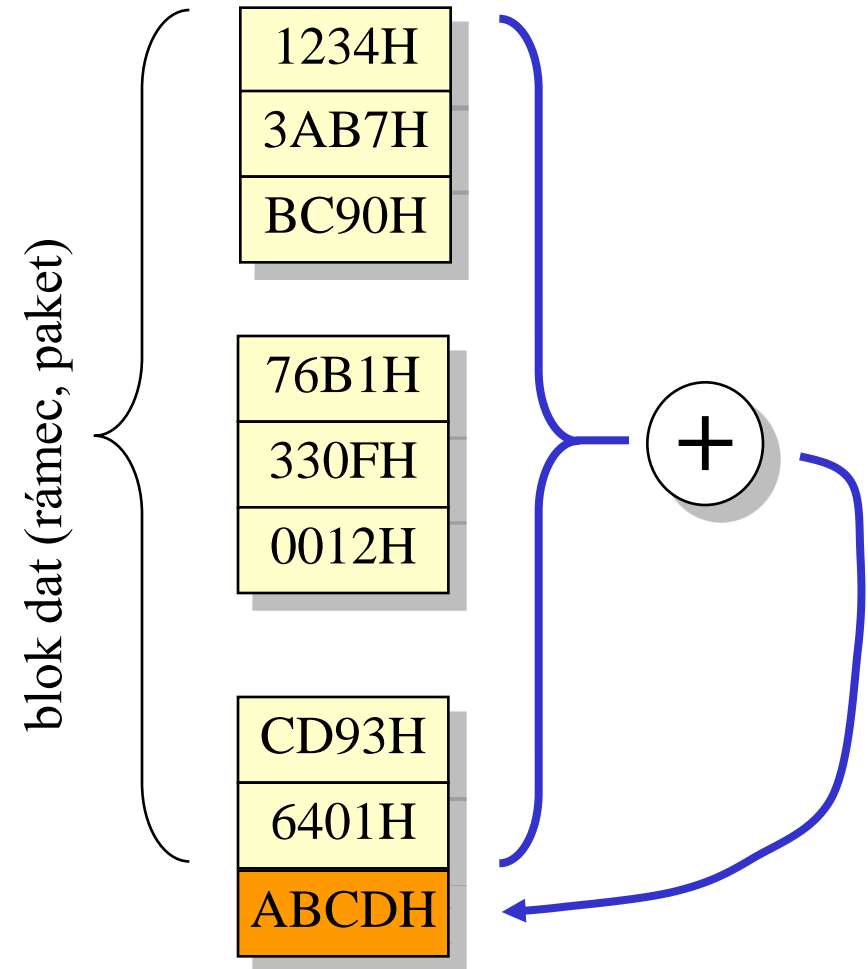
- **podélná parita:**

- parita ze všech stejnohlých bitů všech bytů/slov



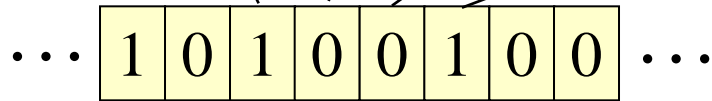
kontrolní součet

- jednotlivé byty/slova/dvojslova tvořící přenášený blok se interpretují jako čísla a sečtou se
- výsledný součet se použije jako zabezpečovací údaj
 - obvykle se použije jen část součtu, například nižší byte či nižší slovo
- alternativa:
 - místo součtu se počítá XOR jednotlivých bitů
- účinnější než parita
 - ale stále je "míra zabezpečení" příliš nízká



CRC – Cyclic Redundancy Check

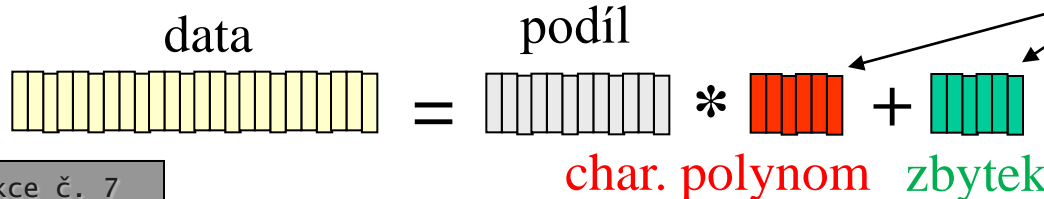
- posloupnost bitů, tvořící blok dat, je interpretována jako polynom
 - polynom nad tělesem charakteristiky 2, kde jednotlivé bity jsou jeho koeficienty
 - $\dots + 1.x^{14} + 0.x^{13} + 0.x^{12} + 1.x^{11} + \dots$
- tento polynom je vydělen jiným polynomem (tzv. charakteristickým polynomem)
 - např. CRC-16: $x^{16} + x^{15} + x^2 + 1$
- výsledkem je podíl a zbytek



- schopnosti detekce jsou "vynikající":
 - všechny shluky chyb s lichým počtem bitů
 - všechny shluky chyb do velikosti n bitů
 - kde n je stupeň charakteristického polynomu
 - všechny shluky chyb velikosti $> n+1$ s pravděpodobností 99.99999998%
 - CRC-32

– v roli zabezpečení se použije zbytek po dělení charakteristickým polynomem

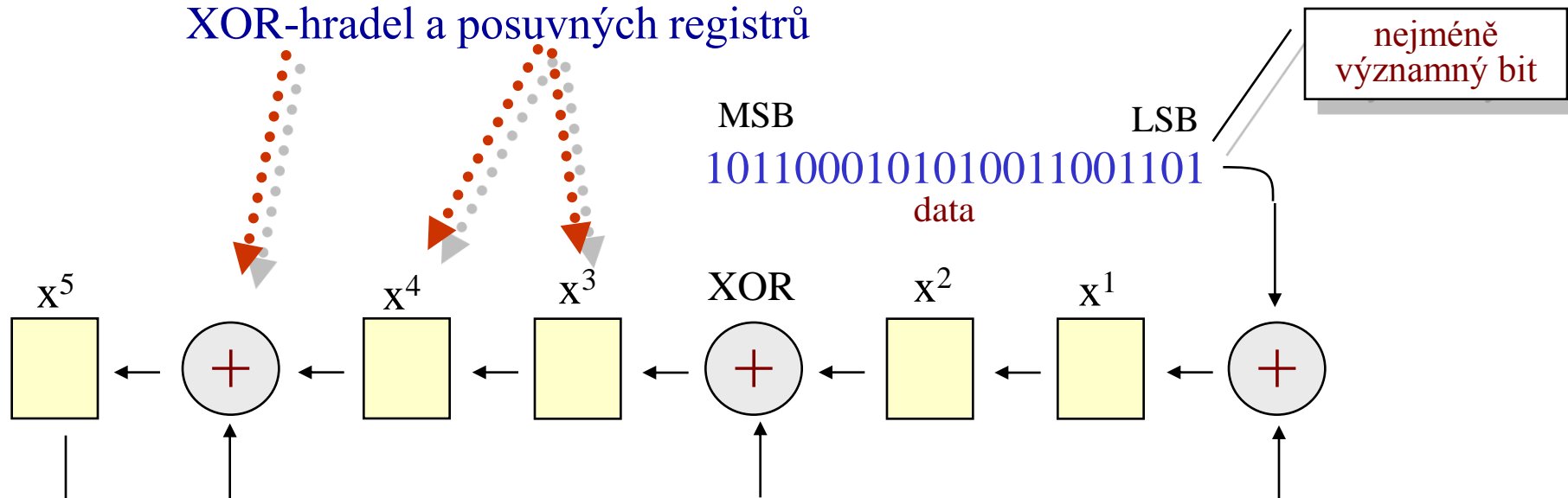
- chápaný již jako posloupnost bitů



používá se CRC v rozsahu 16 bitů nebo 32 bitů

výpočet CRC

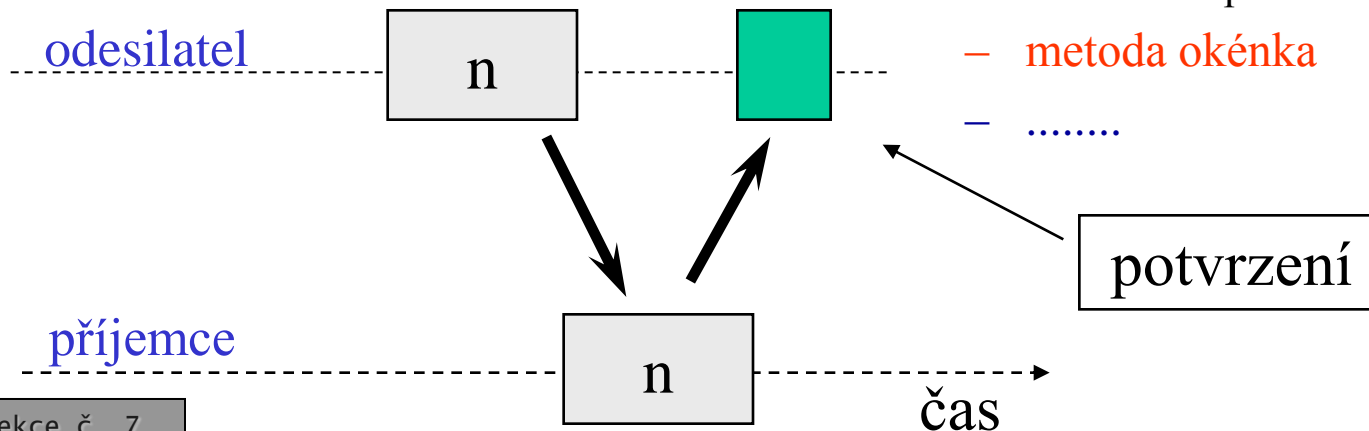
- spolehlivost CRC kódů se opírá o silné teoretické výsledky z algebry
- samotný výpočet CRC-kódu (zbytku po dělení) je velmi jednoduchý
 - a může být snadno implementován v HW, pomocí XOR-hradel a posuvných registrů



(charakteristický polynom je $x^5 + x^4 + x^2 + 1$)

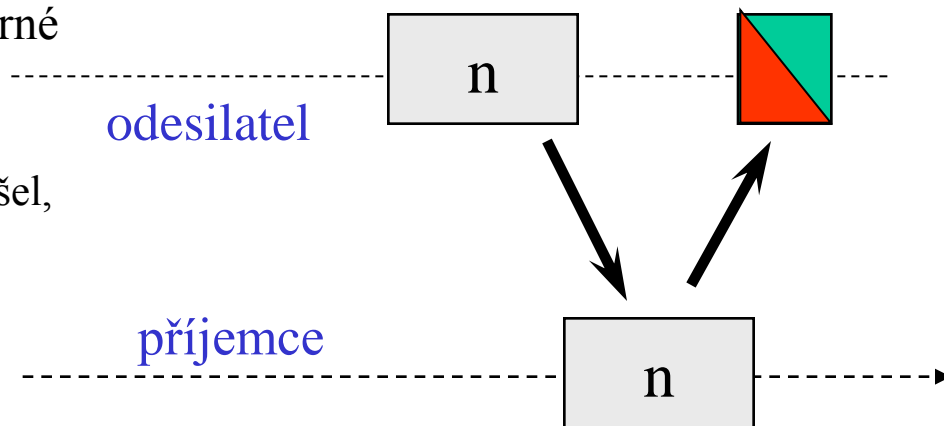
potvrzování (acknowledgement)

- jde o obecnější mechanismus, který slouží (může sloužit) více účelům současně:
 - **zajištění spolehlivosti**
 - umožňuje, aby si příjemce vyžádal opakované zaslání poškozeného rámce
 - **řízení toku**
 - aby příjemce mohl regulovat tempo, jakým mu odesílatel posílá data
- existuje více možných způsobů jak realizovat potvrzování ("stupňů volnosti"):
 - **kladné a záporné potvrzování**
 - potvrzují se správně resp. chybně přijaté bloky
 - **jednotlivé a kontinuální potvrzování**
 - podle toho, zda odesílatel vždy čeká na potvrzení nebo odesílá "do foroty"
 - **samostatné a nesamostatné potvrzování**
 - zda potvrzení cestuje jako samostatný rámec/paket, nebo je vnořeno do datového rámce/paketu
 - **metoda okénka**
 -

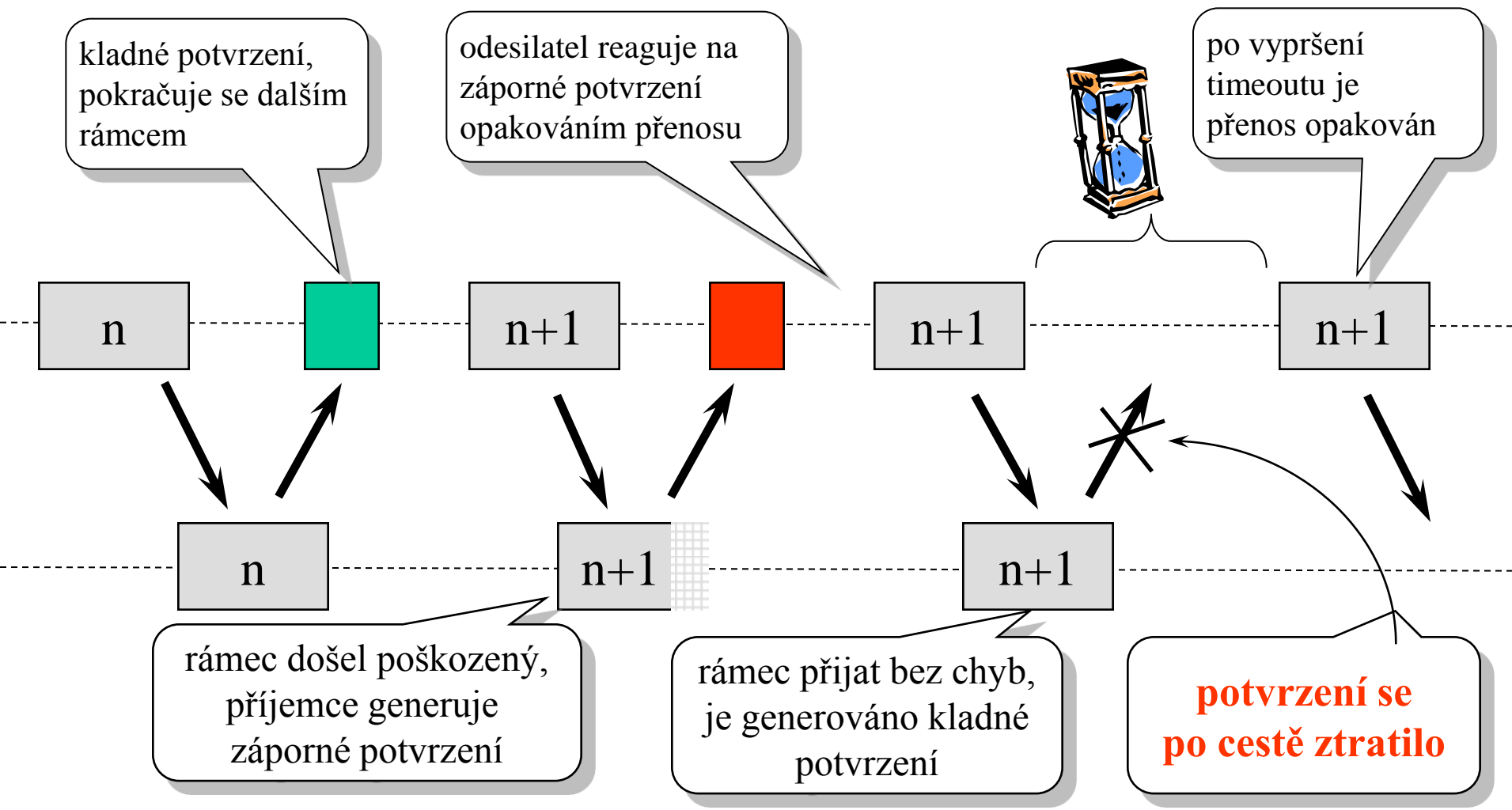


Stop&Wait ARQ

- jde o samostatné jednotlivé potvrzování
 - **samostatné** = potvrzení je přenášeno jako samostatný (řídící) blok
 - **jednotlivé** = potvrzován je každý jednotlivý rámeček/paket
 - **kladně:** že došel v pořádku
 - **záporně:** že došel, ale nebyl v pořádku
 - **timeout:** když potvrzení nepřijde do předem stanoveného intervalu (interpretuje se stejně jako záporné potvrzení)
 - možné příčiny:
 - » rámeček/paket vůbec nedošel, příjemce neví že by měl něco potvrdit
 - » ztratilo se potvrzení
- průběh:
 - odesílatel odešle datový rámeček a čeká na jeho potvrzení (kladné či záporné)
 - tj. další rámeček ještě neodesílá
 - příjemce odešle potvrzení
 - kladné nebo záporné
 - podle druhu potvrzení odesílatel buď opakuje přenos téhož rámečku, nebo vyšle další rámeček
 - nebo čeká na vypršení timeoutu, které interpretuje stejně jako záporné potvrzení

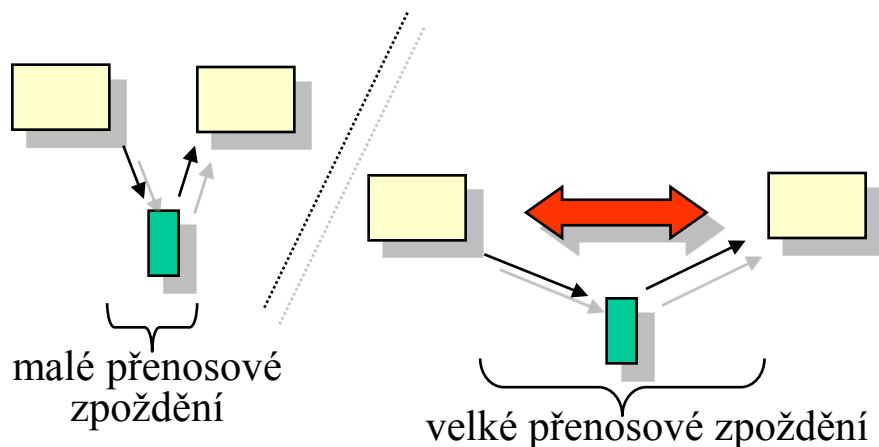


příklad (stop & wait ARQ)

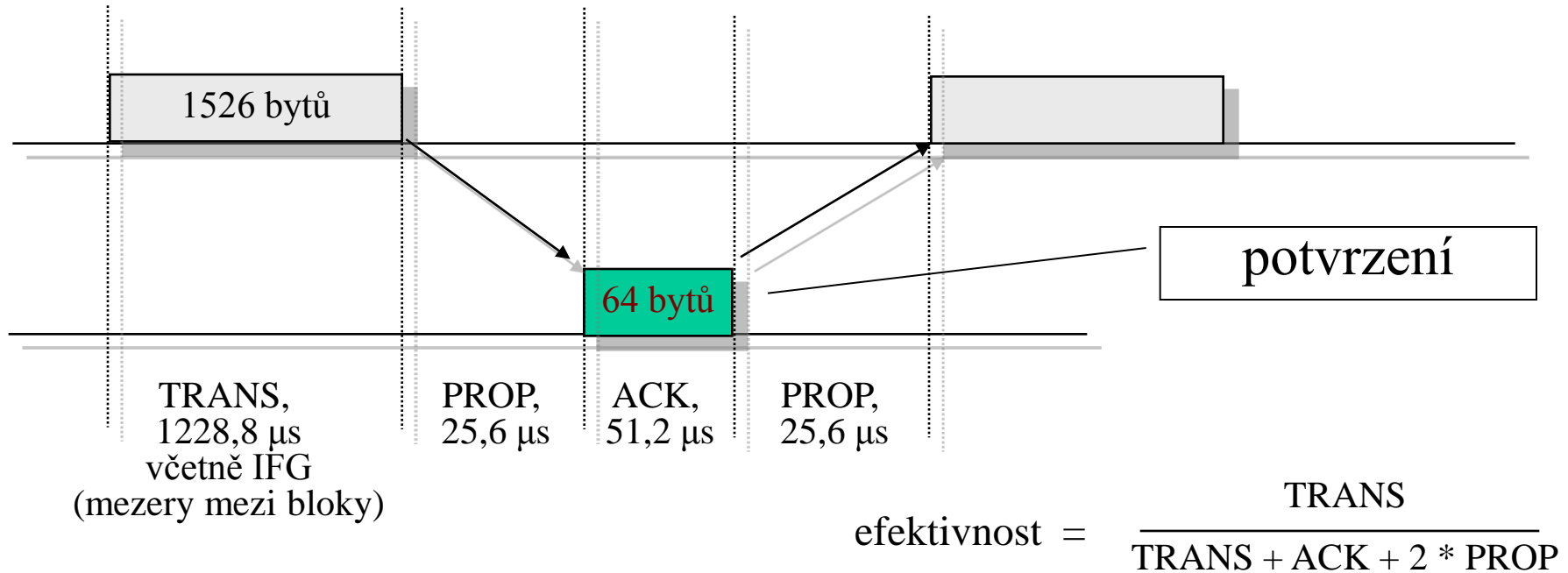


vlastnosti „stop&wait“

- jednoduchá a přímočará implementace
- charakter přenosu vychází ryze poloduplexní
 - nevyužívá se případné (plné) duplexnosti přenosové cesty
- používá se např. v protokolech IPX/SPX firmy Novell
- má smysl v sítích LAN,
 - kde je přenosové zpoždění únosně malé
- ale nikoli v sítích WAN
 - kde je zpoždění velké
- při větším přenosovém zpoždění se tento způsob potvrzování stává velmi neefektivní
 - dochází k velkým prodlevám mezi přenosy jednotlivých bloků
 - novellské protokoly IPX/SPX nejsou vhodné pro nasazení v rozlehlých sítích!!!
 - řešení s IPX/SPX
 - náhrada protokoly TCP/IP
 - speciální úprava, která změní jednotlivé potvrzování na kontinuální



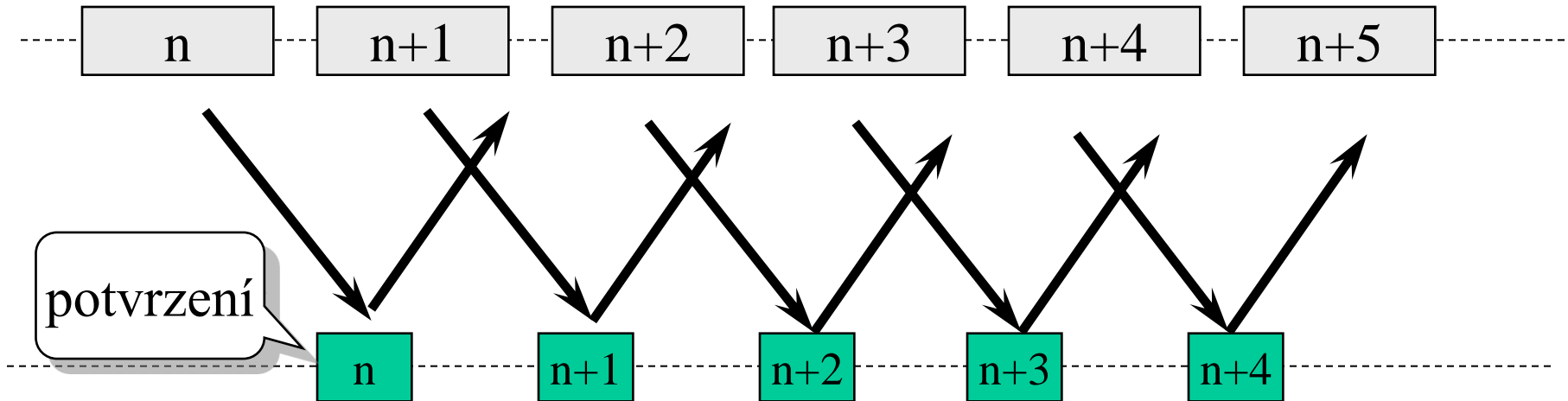
příklad: Ethernet



- pro 10 Mbps Ethernet v prostředí LAN, s RTT (Round Trip Time) 52,1 μs
 - vychází efektivnost cca 92%
- v prostředí WAN se RTT pohybuje v řádu milisekund!
 - např. při RTT = 100 ms (PROP=50 000 μs) klesla na pouhé 2,3%

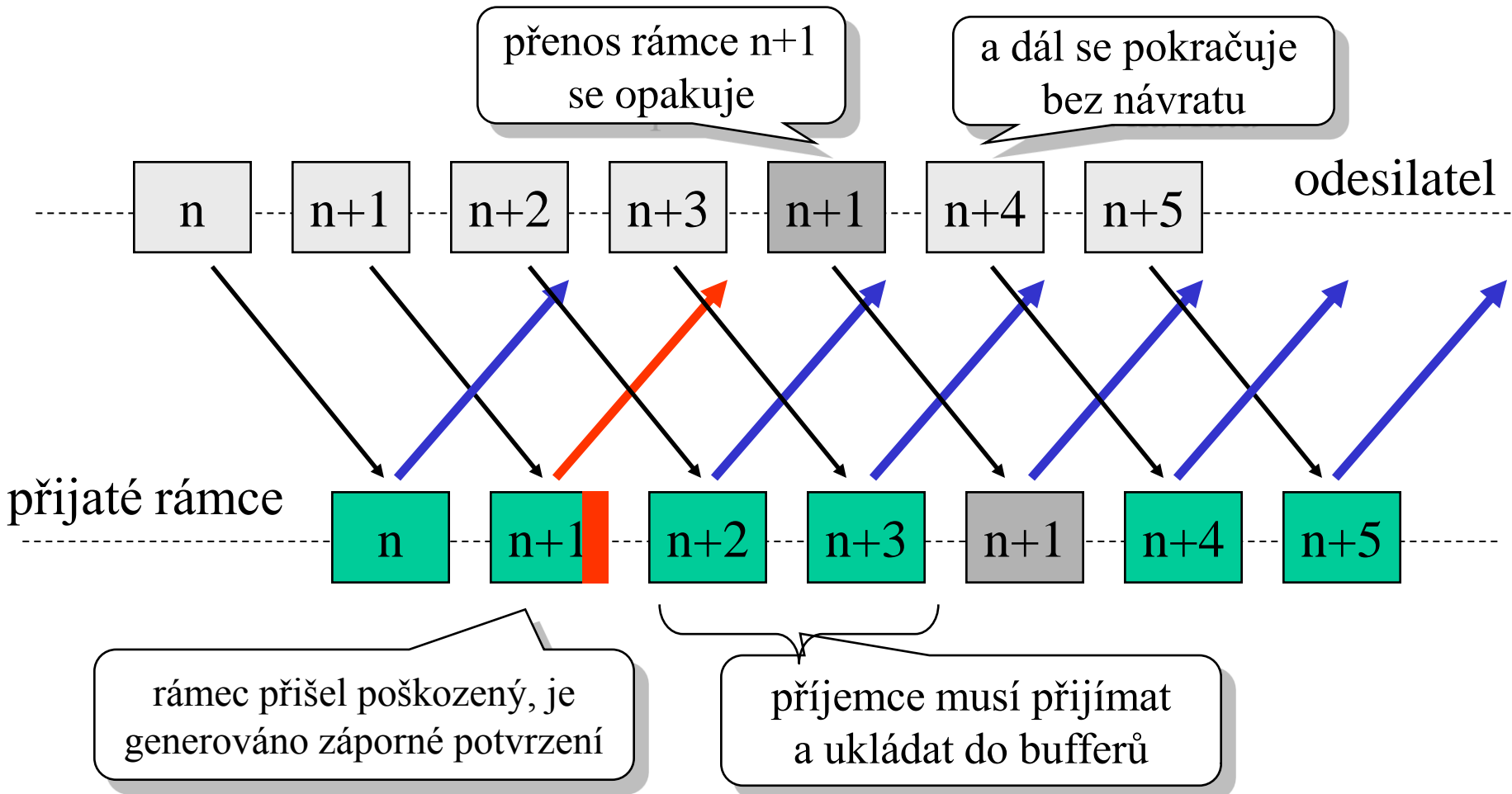
kontinuální potvrzování continuous ARQ

- idea: odesílatel bude vysílat datové rámce „dopředu“, a příslušná potvrzení bude přijímat průběžně, s určitým zpožděním



- otázka: jak se má odesílatel zachovat, když dostane záporné potvrzení?
 - a už mezitím odeslal několik dalších rámců
- varianta „**selektivní opakování**“:
 - odesílatel znovu vyšle jen ten rámeček, který se poškodil
 - další rámce, které se mohly úspěšně přenést, se nevysílají znovu (šetří to přenosovou kapacitu)
 - příjemce musí úspěšně přijaté rámce ukládat do bufferů (je to náročné na jeho hospodaření s pamětí)

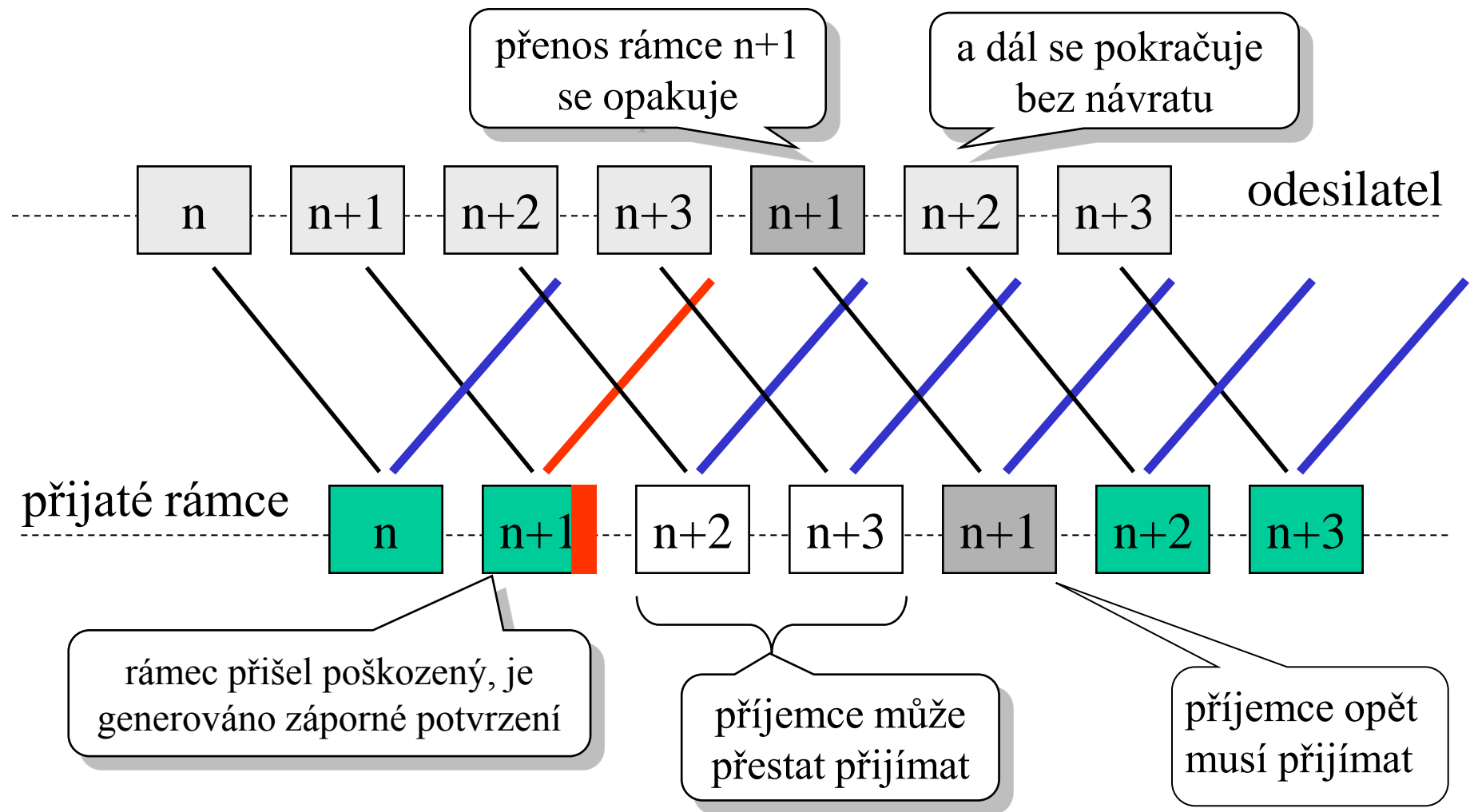
kontinuální potvrzování, varianta se selektivním opakováním



varianta: návrat zpět

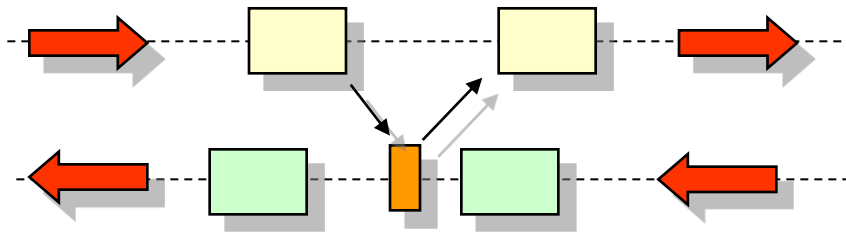
- alternativa k selektivnímu opakování
- řeší situaci kdy záporné potvrzení (informace o poškození určitého rámce) přijde se zpožděním
 - v době kdy již byly odeslány nějaké další rámce
 - řeší se „zahozením“ již odeslaných rámců
- odesílatel znovu vyšle poškozený rámec
- **a po něm postupně vysílá následující rámce**
 - které již mohly být jednou odeslány
- nevýhody:
 - „plýtvání“ přenosovou kapacitou
- výhody:
 - příjemci stačí čekat na nový přenos poškozeného rámce, a pak pokračovat v příjmu dalších rámců
- obecné vlastnosti kontinuálního potvrzování:
 - dokáže lépe „snášet“ větší přenosové zpoždění
 - hodí se i do prostředí WAN, kde přenosové zpoždění je velké
 - používá se např. v protokolech TCP/IP
 - potvrzování se používá v protokolu TCP, který zajišťuje spolehlivý přenos
 - TCP se snaží průběžně adaptovat na podmínky přenosu
 - dynamicky si upravuje různé časové limity a další parametry, aby se choval optimálně
- další otázky:
 - kolik rámců si odesílatel může dovolit vyslat „dopředu“
 - ještě než dostane jejich potvrzení?
 - odpověď záleží na konkrétní velikosti přenosového zpoždění, reakční době protistrany, velikosti rámců,

kontinuální potvrzování, varianta s návratem

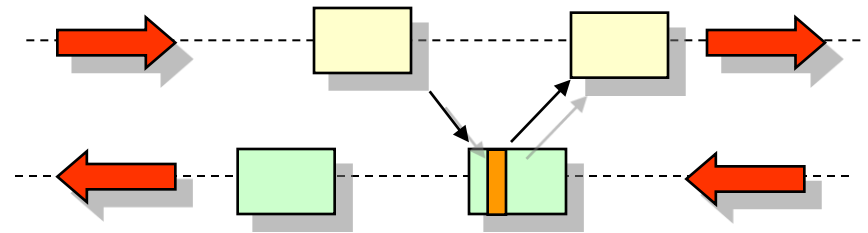


samostatné a nesamostatné potvrzování

- **samostatné** = potvrzení je přenášeno jako samostatný rámec speciálního typu
 - je to spojeno s relativně velkou reží
 - samotné potvrzení je hodně malé, „obal“ je pak velký



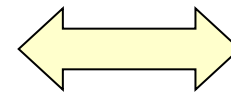
- **nesamostatné** = potvrzení je zasíláno jako součást „datových“ rámců
 - přenášeny v opačném směru než rámce, které jsou potvrzovány
 - označováno jako tzv. **piggybacking**



problematika řízení toku

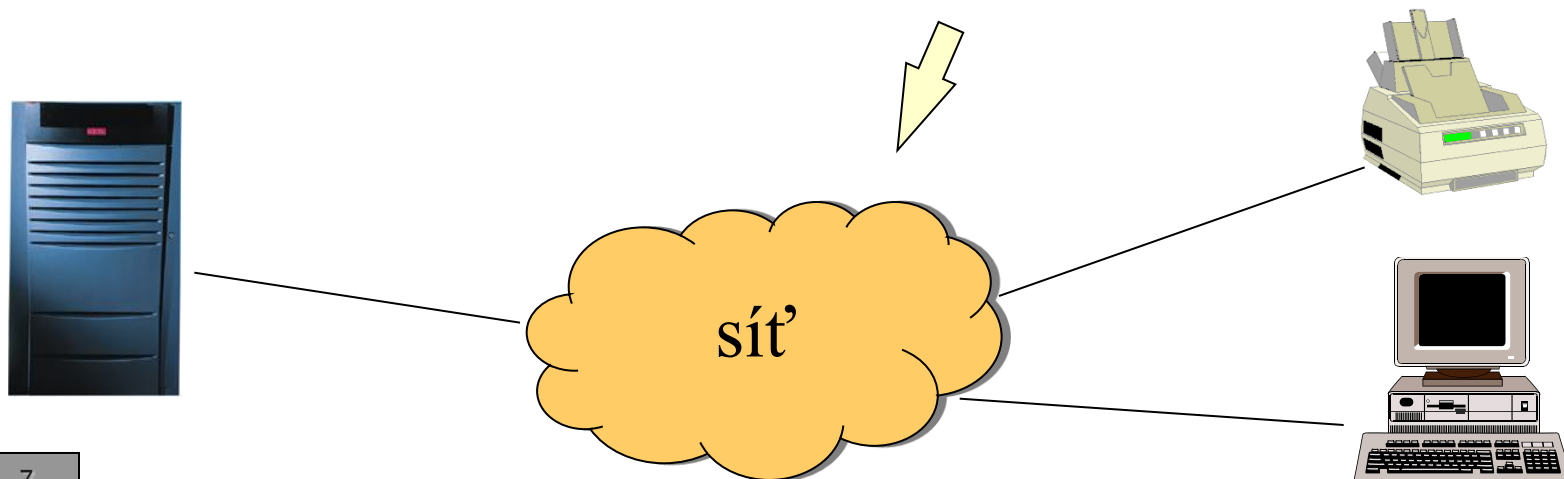
- podstata problému:
 - rychlost (výpočetní síla) dvou komunikujících stran může být i dosti výrazně odlišná
 - je nutné se vyvarovat toho, aby příjemce „nestíhal“
 - kvůli své rychlosti
 - kvůli nedostatku bufferů
 -
 - a musel kvůli tomu zahazovat správně přenesené rámce/pakety
- proto je nutné řídit tok dat mezi odesilatelem a příjemcem
 - podle možností příjemce!!!
- lze řešit na různých úrovních:
 - na úrovni jednotlivých znaků
 - tzv. **hardwarový handshake** (využívají se k tomu samostatné signály, RTS a CTS rozhraní RS-232-C)
 - tzv. **softwarový handshake** (příjemce posílá odesilateli znaky XON/XOFF, regulující tok dat)
 - na úrovni celých rámců
 - příjemce si reguluje, zda chce nebo nechce poslat další rámce

tj. příjemce by měl diktovat tempo

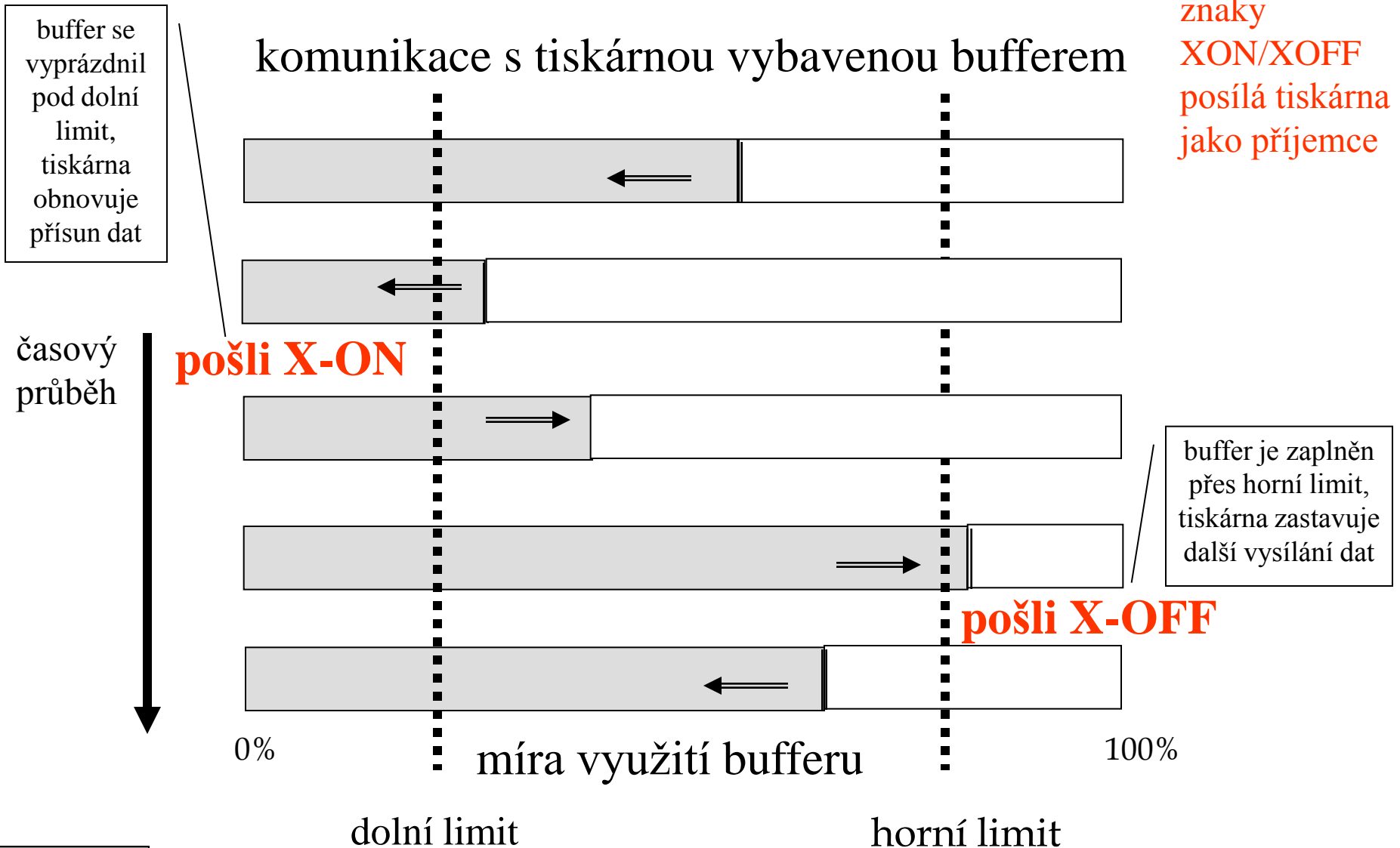


řízení toku vs. zahlcení

- řízení toku (**flow control**) se týká koncových uzlů
 - aby odesílatel nezahltl příjemce
 - přitom se předpokládá, že přenosová cesta mezi nimi není úzkým hrdlem
 - že je dostatečně dimenzovaná, že se nezahlcuje
 - obvyklé řešení:
 - příjemce diktuje tempo přenosu
- předcházení zahlcení (**congestion control**) se týká přenosové sítě
 - opatření proti tomu, aby se zahlcovala přenosová část sítě mezi komunikujícími stranami
 - jakoby: odesílatel i příjemce jsou dostatečně rychlí, potenciálním úzkým hrdlem je přenosová síť
 - možná řešení:
 - traffic shaping, traffic policing, přechod z kontinuálního na jednotlivé potvrzování, ...



příklad: řízení toku pomocí XON/XOFF



metoda okénka (sliding window)

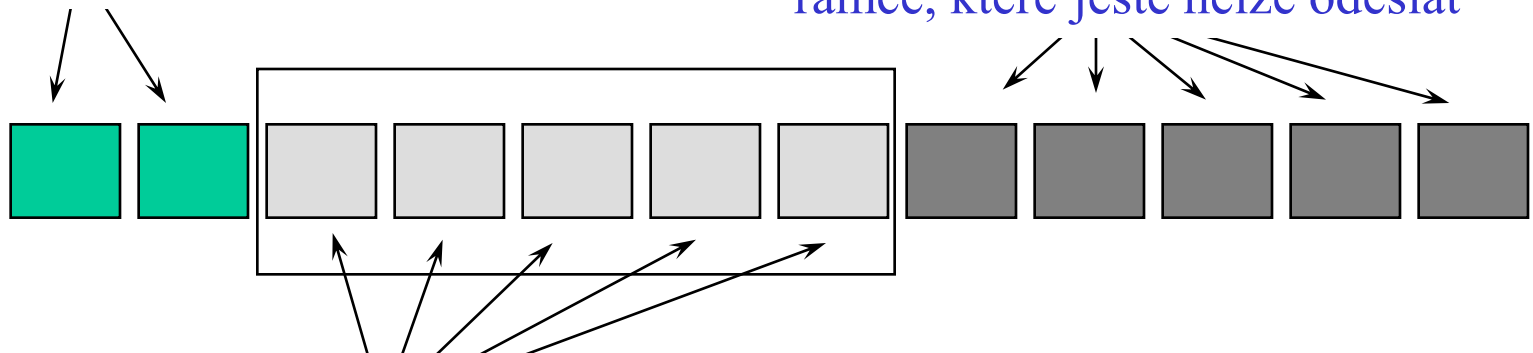
idea:

- spojit potvrzování s řízením toku na úrovni rámců
- odesílatel si udržuje vysílací „okénko“
 - velikost okénka udává, kolik rámců smí vyslat "dopředu"
 - aniž by je měl ještě potvrzené
- používá např. protokol TCP

velikost okénka určuje:

- odesílatel, dle „chování“ a vlastností sítě
 - většinou stanoví maximální velikost
- příjemce, podle svých možností (např. dostupnosti bufferů)
 - ovlivňováním velikosti okénka může příjemce efektivně regulovat tok dat směrem k sobě !!!!!
 - zmenšením okénka na nulovou velikost lze zastavit vysílání

již potvrzené rámce



rámce, které mohou být odeslány (i bez potvrzení)

poznámka

- příjemce by mohl regulovat tok dat i tím, že nebude potvrzovat přijaté rámce
 - nebude posílat žádná potvrzení
- odesílatel bude čekat na vypršení timeoutu, a pak vyšle rámeček znovu
- bude to fungovat, ale nebude to příliš „šetrné“
 - budou opakovány přenosy, které proběhly úspěšně, bude se plýtvat přenosovou kapacitou
 - hrozí nebezpečí, že při větším počtu neúspěšných pokusů (bez kladného potvrzení) to odesílatel vzdá
- v praxi se to nepoužívá

