

# Státnicový okruh 4: Informační technologie

26. dubna 2022

## **Obsah**

# 1

John von Neumannova a harvardská architektura počítače, princip jeho činnosti. Binární logika, logické operace a funkce, logické obvody. Reprezentace čísel a znaků v paměti počítače. Osobní počítač (PC), základní deska, chipset a sběrnice (interní, externí). Procesor (CPU), vykonávání instrukcí, podprogramy a zásobník, přerušení. Paměti počítače (RAM, cache, disk, diskové pole). Přídavné karty PC, datové mechaniky a média (CD, DVD, paměťové karty), periferie.

## 1.1 John von Neumannova a harvardská architektura počítače, princip jeho činnosti

### 1.1.1 John von Neumannova architektura

#### von Neumannova koncepce počítače (1)

- John von Neumann, ~1946, Princeton Institute for Advanced Studies
- = **řízení počítače programem uloženým v paměti**

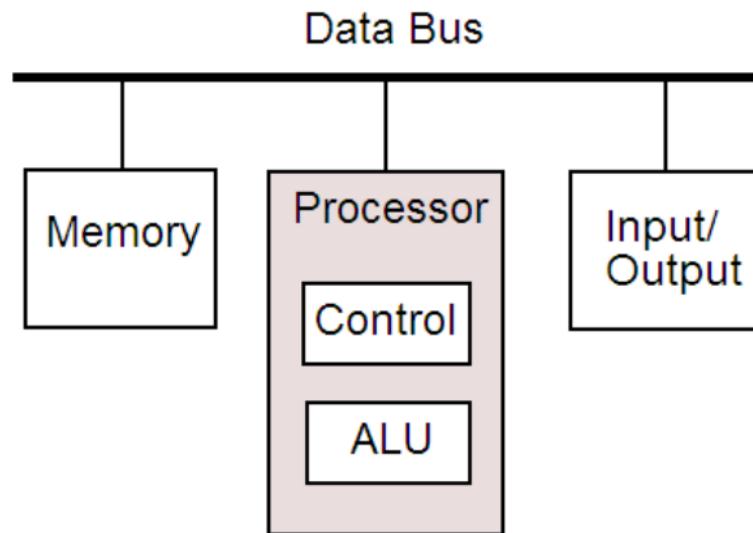
Architektura:

- procesor (CPU): **řadič** (CU) + **aritmeticko-logická jednotka** (ALU)
- **operační paměť**: lineárně organizovaná, rozdělená na stejně velké buňky, přístup pomocí adres
- **vstupně/výstupní (I/O) zařízení**
- propojené **sběrnicí** = soustava vodičů

Obrázek: Schéma architektury von-Neumannovy koncepce počítače

- prototypový počítač: IAS (1952)

## von Neumannova koncepce počítače (1)



## von Neumannova koncepce počítače (2)



- **program** = předpis pro řešení úlohy = posloupnost elementárních kroků, tzv. instrukcí
- **instrukce** = interpretovaná **binární data** se speciálním významem
- (proměnná) **data a program** načtené do **jedné společné operační paměti**
- činnost počítače řídí řadič: s využitím ALU zpracovává instrukce programu nad daty čtenými z paměti nebo vstupního zařízení, výsledná data se zapisují do paměti nebo výstupního zařízení
- **instrukce programu vykonávány sekvenčně**, výjimku tvoří instrukce skoků
- ALU: základní početní operace (sčítání, násobení, logické, bitové posuvy)
- **von Neumann bottleneck**: rychlosť zpracování instrukcí vs. rychlosť komunikace s pamětí a I/O zařízeními → cache = vyrovnávací paměť

## von Neumannova koncepce počítače (3)



Koncepce, až na drobné odlišnosti, používaná dodnes:

- rozšíření o koncepci přerušení od I/O a dalších zařízení – umožňuje efektivně zpracovávat více programů "zároveň" i na jednom CPU (multitasking)
- více než jeden procesor (řadič, ALU), zpracovávání více programů (skutečně) zároveň
- postupné načítání programu do paměti podle potřeby
- více typů a druhů sběrnic (paměťová, I/O)
- integrace některých I/O zařízení do CPU (řadiče sběrnic, grafické, síťové)

### 1.1.2 Harvardská architektura

#### Harvardská koncepce počítače

- podle počítače MARK I (program na děrné páscce, data na elektromechanických deskách)

Architektura podobná von Neumannově, až na:

- dvě **oddělené paměti pro program a pro data**
  - paměť programu často jen pro čtení
  - **paralelní přístup do paměti**
- 
- **modifikovanou** ji interně používají moderní CPU (instrukční a datová cache)
  - DSP procesory v audio/video technice, jednoúčelové (programovatelné) mikrokontroléry (Atmel AVR), kalkulátory

## 1.2 Binární logika, logické operace a funkce, logické obvody

### 1.2.1 Binární logika

#### Binární logika (1)



Základní operace v počítači = logické operace

- formální základ = **výroková logika** – zkoumá pravdivostní hodnotu výroků (pravda/nepravda, spojky/operátory “neplatí, že” → operace negace  $\neg$ , “a současně platí” → konjunkce  $\wedge$ , “nebo platí” → disjunkce  $\vee$ , “jestliže platí, pak platí” → implikace  $\Rightarrow$  aj.)
- výroky = **logické výrazy** vyhodnocované na hodnoty pravda/nepravda, 1/0
- matematický aparát pro práci s log. výrazy: **Booleova algebra (binární, dvoustavová, logika)**
- fyzická realizace – **logické elektronické obvody** – základ digitálních zařízení
- binární logika: univerzální, teoreticky zvládnutá, efektivně realizovatelná logickými el. obvody

#### Binární logika (2)

##### Logická proměnná $x$

- veličina nabývající dvou možných diskrétních logických hodnot: 0 (nepravda) a I (pravda)
- definice:  $x = I$  jestliže  $x \neq 0$  a  $x = 0$  jestliže  $x \neq I$

##### Logická funkce $f(x_1, \dots, x_n)$

- funkce  $n$  logických proměnných  $x_1, \dots, x_n$  nabývající dvou možných diskrétních hodnot 0 (nepravda) a I (pravda)
- logická proměnná = logická funkce identity proměnné, skládání funkcí
- základní = **logické operace**

##### Booleova algebra (binární logika)

- algebra logických proměnných a logických funkcí
- dvouhodnotová algebra, algebra dvou stavů
- relace rovnosti:  $f = g$ , právě když  $(f = I \wedge g = I) \vee (f = 0 \wedge g = 0)$

## 1.2.2 Logické operace

### Logické operace (1)

3 základní:

#### Negace (inverze)

- pravdivá, když operand nepravdivý, jinak nepravdivá

$x$	$\bar{x}$
0	I
I	0

- operátory:  $\bar{x}$ , NOT  $x$ ,  $\neg x$  (výrokově negace, algebraicky negace),  $\bar{X}$  (množinově doplněk)

### Logické operace (2)

#### Logický součin (konjunkce)

- pravdivá, když oba operandy pravdivé, jinak nepravdivá

$x$	$y$	$x \cdot y$
0	0	0
0	I	0
I	0	0
I	I	I

- operátory:  $x \cdot y / xy$  (prázdný),  $x$  AND  $y$ ,  $x \wedge y$  (výrokově konjunkce, algebraicky průsek),  $X \cap Y$  (množinově průnik)

## Logické operace (3)



### Logický součet (disjunkce)

- nepravdivá, když oba operandy nepravdivé, jinak pravdivá

$x$	$y$	$x + y$
0	0	0
0	I	I
I	0	I
I	I	I

- operátory:  $x + y$ ,  $x \text{ OR } y$ ,  $x \vee y$  (výrokově disjunkce, algebraicky spojení),  $X \cup Y$  (množinově sjednocení)

## Logické operace (4)



### Logický výraz

- = korektně vytvořená posloupnost (symbolů) logických proměnných a funkcí (operátorů) spolu se závorkami
- priority sestupně: negace, log. součin, log. součet
- např.  $x \cdot \bar{y} + f(x, z) = (x \cdot \bar{y}) + f(x, z)$
- = zápis logické funkce

### Logické rovnice

- ekvivalentní úpravy: negace obou stran, logický součin/součet obou stran se stejným výrazem, ..., log. funkce obou stran se stejnými ostatními operandy funkce
- NEekvivalentní úpravy: "krácení" obou stran o stejný (pod)výraz, např.  $x + y = x + z$  není ekvivalentní s  $y = z$

## Logické operace (5)

### Axiomy (Booleovy algebry)

- **komutativita:**

$$x \cdot y = y \cdot x \quad x + y = y + x$$

- **distributivita:**

$$x \cdot (y + z) = x \cdot y + x \cdot z \quad x + y \cdot z = (x + y) \cdot (x + z)$$

- **identita (existence neutrální hodnoty):**

$$\mathbf{I} \cdot x = x \quad \mathbf{0} + x = x$$

- **komplementárnost:**

$$x \cdot \bar{x} = \mathbf{0} \quad x + \bar{x} = \mathbf{1}$$

## Logické operace (5)

### Vlastnosti základních logických operací

- **nula a jednička:**

$$0 \cdot x = 0 \quad I + x = I$$

- **idempotence:**

$$x \cdot x = x \quad x + x = x$$

- **asociativita:**

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z \quad x + (y + z) = (x + y) + z$$

- **involuce (dvojí negace):**

$$\bar{\bar{x}} = x$$

- **De Morganovy zákony:**

$$\overline{x \cdot y} = \bar{x} + \bar{y} \quad \overline{x + y} = \bar{x} \cdot \bar{y}$$

- **absorpce:**

$$x \cdot (x + y) = x \quad x + x \cdot y = x$$

- a další



## Logické operace (6)

### Vlastnosti základních logických operací – použití

- důkazy: s využitím axiomů a již dokázaných vlastností, rozbořem případů (dosazením všech možných kombinací hodnot **0** a **I** za proměnné)
- ekvivalentní úpravy (pro sjednodušování) logických výrazů
- ...

## Logické operace (7)

Další operace

### Implikace

- nepravdivá, když první operand pravdivý a druhý nepravdivý, jinak pravdivá

x	y	$x \rightarrow y$
0	0	I
0	I	I
I	0	0
I	I	I

- operátory:  $x \rightarrow y$ ,  $x \rightarrow y$  (výrokově i algebraicky implikace),  $X \subseteq Y$  (množinově podmnožina)

## Logické operace (8)

1

### Ekvivalence

- pravdivá, když operandy mají stejnou hodnotu, jinak nepravdivá

x	y	$x \equiv y$
0	0	I
0	I	0
I	0	0
I	I	I

- operátory:  $x \equiv y$ ,  $x \text{ XNOR } y$ ,  $x \equiv y$  (výrokově i algebraicky ekvivalence),  $X \equiv Y$  (množinově ekvivalence nebo rovnost)

## Logické operace (7)

1

### Další operace

### Implikace

- nepravdivá, když první operand pravdivý a druhý nepravdivý, jinak pravdivá

x	y	$x \rightarrow y$
0	0	I
0	I	I
I	0	0
I	I	I

- operátory:  $x \rightarrow y$ ,  $x \rightarrow y$  (výrokově i algebraicky implikace),  $X \subseteq Y$  (množinově podmnožina)

## Logické operace (8)

1

### Ekvivalence

- pravdivá, když operandy mají stejnou hodnotu, jinak nepravdivá

x	y	$x \equiv y$
0	0	I
0	I	0
I	0	0
I	I	I

- operátory:  $x \equiv y$ ,  $x \text{ XNOR } y$ ,  $x \equiv y$  (výrokově i algebraicky ekvivalence),  $X \equiv Y$  (množinově ekvivalence nebo rovnost)

## Logické operace (9)

### Nonekvivalence (negace ekvivalence, aritmetický součet modulo 2)

- pravdivá, když operandy mají různou hodnotu, jinak nepravdivá

$x$	$y$	$x \oplus y$
0	0	0
0	I	I
I	0	I
I	I	0

- operátory:  $x \oplus y$ ,  $x$  XOR  $y$ ,  $x \not\equiv y$  (výrokově i algebraicky negace ekvivalence),  $X \not\equiv Y$  (množinově negace ekvivalence)

## Logické operace (10)

### Shefferova funkce (negace logického součinu)

- nepravdivá, když oba operandy pravdivé, jinak pravdivá

$x$	$y$	$x \uparrow y$
0	0	I
0	I	I
I	0	I
I	I	0

- operátory:  $x \uparrow y$ ,  $x$  NAND  $y$

## Logické operace (11)

### Piercova funkce (negace logického součtu)

- pravdivá, když oba operandy nepravdivé, jinak nepravdivá

$x$	$y$	$x \downarrow y$
0	0	I
0	I	0
I	0	0
I	I	0

- operátory:  $x \downarrow y$ ,  $x$  NOR  $y$

### 1.2.3 Logické funkce

#### Logické funkce (1)



- zadání **pravdivostní tabulkou**:

- úplně – funkční hodnota  $f(x_i)$  definována pro všechny  $2^n$  možných přiřazení hodnot proměnným  $x_i, 0 \leq i < n$
- neúplně – funkční hodnota pro některá přiřazení není definována (např. log. obvod realizující funkci ji neimplementuje)

- **základní tvary** (výrazu):

- **součinový (úplná konjunktivní normální forma, ÚKNF)** – log. součin log. součtů všech proměnných nebo jejich negací (úplných elementárních disjunkcí, ÚED)

$$(X_0 + \dots + X_{n-1}) \cdot \dots \cdot (X_0 + \dots + X_{n-1}) \quad X_i = x_i \text{ nebo } \overline{x_i}$$

- **součtový (úplná disjunktivní normální forma, ÚDNF)** – log. součet log. součinů všech proměnných nebo jejich negací (úplných elementárních konjunkcí, ÚEK)

$$(X_0 \cdot \dots \cdot X_{n-1}) + \dots + (X_0 \cdot \dots \cdot X_{n-1}) \quad X_i = x_i \text{ nebo } \overline{x_i}$$

#### Logické funkce (2)



##### Převod log. funkce $f(x_i)$ na základní tvar (normální formu)

- ekvivalentními úpravami a doplněním chybějících proměnných nebo jejich negací

- **tabulkovou metodou**:

- 1 pro řádky s  $f(x_i) = 0(\mathbf{I})$  sestroj log. součet (součin) všech  $x_i$  pro  $x_i = 0(\mathbf{I})$  nebo  $\overline{x_i}$  pro  $x_i = \mathbf{I}(0)$
- 2 výsledná ÚKNF (ÚDNF) je log. součinem (součtem) těchto log. součtů (součinů)

$x$	$y$	$z$	$f(x, y, z)$	ÚED	ÚEK
0	0	0	0	$x + y + z$	
0	0	1	0	$x + y + \overline{z}$	
0	1	0	0	$x + \overline{y} + z$	
0	1	1	1		$\overline{x} \cdot y \cdot z$
1	0	0	0	$\overline{x} + y + z$	
1	0	1	1		$x \cdot \overline{y} \cdot z$
1	1	0	1		$x \cdot y \cdot \overline{z}$
1	1	1	1		$x \cdot y \cdot z$

$$\text{ÚKNF}(f(x, y, z)): (x + y + z) \cdot (x + y + \overline{z}) \cdot (x + \overline{y} + z) \cdot (\overline{x} + y + z)$$

$$\text{ÚDNF}(f(x, y, z)): \overline{x} \cdot y \cdot z + x \cdot \overline{y} \cdot z + x \cdot y \cdot \overline{z} + x \cdot y \cdot z$$

## Logické funkce (3)



### Věta (O počtu log. funkcí)

Existuje právě  $2^{(2^n)}$  logických funkcí s  $n$  proměnnými.

### Funkce $f^1$ jedné proměnné

$x$	$f_0$	$f_1$	$f_2$	$f_3$
<b>0</b>	0	$x$	$\bar{x}$	<b>I</b>
<b>0</b>	0	0	<b>I</b>	<b>I</b>
<b>I</b>	0	<b>I</b>	0	<b>I</b>

### Funkce $f^2$ dvou proměnných

$x$	$y$	$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$	$f_8$	$f_9$	$f_{10}$	$f_{11}$	$f_{12}$	$f_{13}$	$f_{14}$	$f_{15}$
0	0	0	0	0	0	0	0	0	0	I	I	I	I	I	I	I	I
0	I	0	0	0	0	I	I	I	I	0	0	0	0	I	I	I	I
I	0	0	0	I	I	0	0	I	I	0	0	I	I	0	0	I	I
I	I	0	I	0	I	0	I	0	I	0	I	0	I	0	I	0	I

## Logické funkce (4)



### Funkce více než dvou proměnných

pro  $n = 3$ :

$$f(x, y, z) = x \cdot f(\mathbf{I}, y, z) + \bar{x} \cdot f(\mathbf{0}, y, z)$$

a podobně pro  $n > 3$

### Věta (O reprezentaci log. funkci)

Jakoukoliv logickou funkci libovolného počtu proměnných lze vyjádřit pomocí logických funkcí dvou proměnných.



## Logické funkce (5)

### Zjednodušení výrazu logické funkce

- = optimalizace za účelem dosažení co nejmenšího počtu operátorů (v kompromisu s min. počtem typů operátorů)
- důvod: méně (typů) log. obvodů realizujících funkci (menší, levnější, nižší spotřeba, ...)

### Algebraická minimalizace

$$\begin{aligned}f &= \bar{x} \cdot y \cdot z + x \cdot \bar{y} \cdot z + x \cdot y \cdot \bar{z} + x \cdot y \cdot z \\&\quad // dvakrát přičteme x \cdot y \cdot z \text{ (idempotence)} \\f &= (\bar{x} \cdot y \cdot z + x \cdot y \cdot z) + (x \cdot \bar{y} \cdot z + x \cdot y \cdot z) + (x \cdot y \cdot \bar{z} + x \cdot y \cdot z) \\&\quad // distributivita \\f &= y \cdot z \cdot (\bar{x} + x) + x \cdot z \cdot (\bar{y} + y) + x \cdot y \cdot (\bar{z} + z) // komplementárnost \\f &= x \cdot y + y \cdot z + x \cdot z\end{aligned}$$

- pro složitější výrazy náročná



## Logické funkce (5)

### Zjednodušení výrazu logické funkce

#### Karnaughova metoda (Veitch diagram)

- nahrazení algebraických ekvivalentních úprav geometrickými postupy
- nalezení minimálního výrazu

- 1 k výrazu v základním součtovém tvaru se sestaví tzv. **Karnaughova mapa** = tabulka vyplněná I v buňkách reprezentující log. součiny, součiny reprezentované sousedními buňkami se liší v 1 proměnné
- 2 hledání smyček (minterm) v mapě splňujících jisté podmínky (min. počet, max. obdélníková oblast vyplněná I, počet políček mocnina 2, mohou se překrývat, pokrytí všech I)
- 3 smyčky po vyloučení komplementárních proměnných a jejich negací reprezentují log. součiny výsledného součtového tvaru



## Logické funkce (6)

Zjednodušení výrazu logické funkce

**Karnaughova metoda (Veitch diagram)**

$$f = \bar{x} \cdot y \cdot z + x \cdot \bar{y} \cdot z + x \cdot y \cdot \bar{z} + x \cdot y \cdot z$$

	$\bar{x} \cdot \bar{y}$	$\bar{x} \cdot y$	$x \cdot y$	$x \cdot \bar{y}$
$\bar{z}$			I	
$z$		I	I	I

Obrázek: Karnaughova mapa

$$f = x \cdot y + y \cdot z + x \cdot z$$

- výpočetně náročná (hledání smyček)

Další algoritmické metody: tabulační (Quine-McCluskey), branch-and-bound (Petrick), Espresso logic minimizer aj.

## Logické funkce (7)

**Úplný systém logických funkcí**

- = množina log. funkcí, pomocí kterých je možné vyjádřit jakoukoliv log. funkci (libovolného počtu proměnných)
- množina log. funkcí dvou proměnných (Věta o reprezentaci log. funkcí)
  - (1) negace  $\bar{x}$ , log. součin  $x \cdot y$  a log. součet  $x + y$
  - (2) negace  $\bar{x}$  a implikace  $x \rightarrow y$
  - a další

**Minimální úplný systém logických funkcí**

- = úplný systém, ze kterého nelze žádnou funkci vyjmout tak, aby zůstal úplný
- (1) NENÍ:  $x \cdot y = \bar{\bar{x}} + \bar{\bar{y}}$ ,  $x + y = \bar{\bar{x}} \cdot \bar{\bar{y}}$  (De Morganovy zákony)
- (2) je
- (3) negace  $\bar{x}$  a log. součin  $x \cdot y$
- (4) negace  $\bar{x}$  a log. součet  $x + y$
- a další



## Logické funkce (8)

### Minimální úplný systém logických funkcí

Jediná funkce:

- **Shefferova**  $\uparrow$  (negace log. součinu)
- **Piercova**  $\downarrow$  (negace log. součtu)
- důkaz: vyjádření např. negace a log. součinu (součtu)

### Vyjádření logické funkce pomocí Shefferovy nebo Piercovy funkce

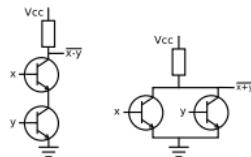
- 1 vyjádření funkce v základním součtovém tvaru
- 2 zjednodušení výrazu funkce, např. pomocí Karnaughovy metody
- 3 aplikace De Morganových zákonů pro převedení výrazu do tvaru, který obsahuje pouze Shefferovy nebo pouze Piercovy funkce

## 1.2.4 Logické obvody



### Fyzická realizace logických funkcí (1)

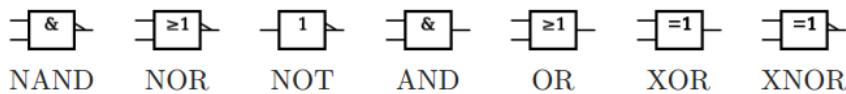
- dříve pomocí **spínacích relé a elektronek**
- dnes pomocí **tranzistorů v integrovaných obvodech**



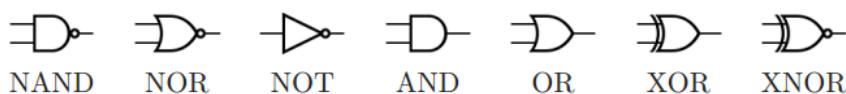
Obrázek: Realizace log. operací NAND a NOR

- realizace log. operací pomocí integrovaných obvodů – **logických členů, hradel**
  - vstupy = reprezentované log. proměnné
  - výstup = výsledek realizované log. operace
  - stavy (signály) na vstupech/výstupu = log. (binární) hodnoty 0/I = míra informace s jednotkou 1 bit
- symbolické značky log. členů ve schématech zapojení **logických obvodů** realizujících lib. log. funkci

### Fyzická realizace logických funkcí (2)



Obrázek: Symbolické značky logických členů (podle normy IEC)

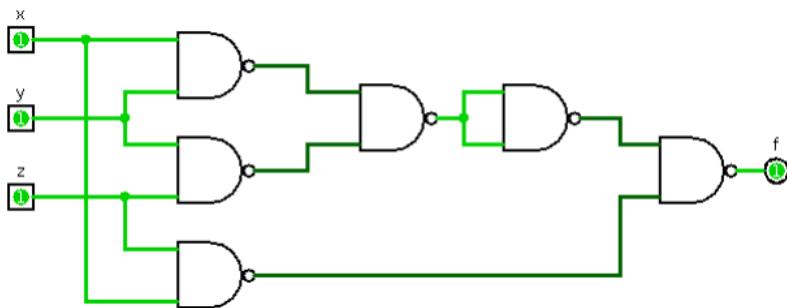


Obrázek: Symbolické značky logických členů (tradiční, ANSI)

## Fyzická realizace logických funkcí (3)



$$f = \overline{\overline{x} \cdot \overline{y} \cdot \overline{y} \cdot \overline{z} \cdot \overline{x} \cdot \overline{y} \cdot \overline{y} \cdot \overline{z} \cdot \overline{x} \cdot \overline{z}}$$



Obrázek: Schéma zapojení log. obvodu realizujícího log. funkci  $f$  pomocí log. členů realizujících log. operaci NAND

## Logické obvody



- jeden výstup: realizace jedné log. funkce
- více výstupů: realizace více log. funkcí zároveň → realizace **vícebitové log. funkce  $^n f$**
- $n$ -tice vstupů: reprezentace **vícebitových (n-bitových) log. proměnných  $^n x$  = vícebitový log. obvod**
- **kombinační**: stavy na výstupech obvodu (tj. funkční hodnota) závisí pouze na okamžitých stavech na vstupech (tj. hodnotách proměnných)
- **sekvenční**: stavy na výstupech obvodu (tj. funkční hodnota) závisí nejen na okamžitých stavech na vstupech (tj. hodnotách proměnných), ale také na přechozích stavech na vstupech

## Kombinační logické obvody (1)



- stavy na výstupech obvodu (tj. funkční hodnota) závisí pouze na okamžitých stavech na vstupech (tj. hodnotách proměnných)
- jedné kombinaci stavů na vstupech odpovídá jediná kombinace stavů na výstupech

## Kombinační logické obvody (2)



### Komparátor

- provádí srovnání hodnot dvou log. proměnných  $A$  a  $B$  na vstupu
- tři výstupy udávající pravdivost vztahů:  $A < B$ ,  $A > B$  a  $A = B$ , realizace tříbitové log. funkce  $Y_< = Y(A < B)$ ,  $Y_> = Y(A > B)$ ,  $Y_= = Y(A = B)$
- jednabitový:

$$\begin{aligned} Y_< &= \overline{A} \cdot B & Y_> &= A \cdot \overline{B} & Y_= &= A \cdot B + \overline{A} \cdot \overline{B} \\ Y_< &= \overline{\overline{\overline{A}} \cdot B} & Y_> &= \overline{\overline{A} \cdot \overline{B}} & Y_= &= \overline{\overline{A} \cdot B} \cdot \overline{A \cdot \overline{B}} \end{aligned}$$

## Kombinační logické obvody (2)



### Komparátor

- provádí srovnání hodnot dvou log. proměnných  $A$  a  $B$  na vstupu
- tři výstupy udávající pravdivost vztahů:  $A < B$ ,  $A > B$  a  $A = B$ , realizace tříbitové log. funkce  $Y_< = Y(A < B)$ ,  $Y_> = Y(A > B)$ ,  $Y_= = Y(A = B)$
- jednabitový:

$$\begin{aligned} Y_< &= \overline{A} \cdot B & Y_> &= A \cdot \overline{B} & Y_= &= A \cdot B + \overline{A} \cdot \overline{B} \\ Y_< &= \overline{\overline{\overline{A}} \cdot B} & Y_> &= \overline{\overline{A} \cdot \overline{B}} & Y_= &= \overline{\overline{A} \cdot B} \cdot \overline{A \cdot \overline{B}} \end{aligned}$$

## Kombinační logické obvody (4)



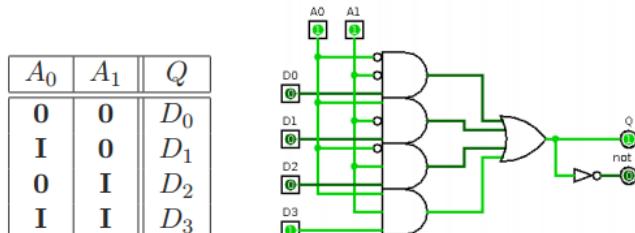
### Multiplexor

- přepíná na výstup  $Q$  log. hodnotu na jednom z  $2^n$  datových vstupů  $D_i$  vybraném na základě  $n$ -bitové hodnoty na adresním vstupu  $A$
- kromě výstupu  $Q$  navíc ještě negovaný (invertovaný) výstup  $\overline{Q}$
- např. čtyřvstupý (4 datové vstupy, dvoubitový adresní vstup) realizuje log. funkci

$$Q = \overline{A_0} \cdot \overline{A_1} \cdot D_0 + A_0 \cdot \overline{A_1} \cdot D_1 + \overline{A_0} \cdot A_1 \cdot D_2 + A_0 \cdot A_1 \cdot D_3$$

## Kombinační logické obvody (5)

### Multiplexor



Obrázek: Pravdivostní tabulka a schéma zapojení čtyřstupného multiplexoru

- použití: multiplexování datových vstupů na základě adresy

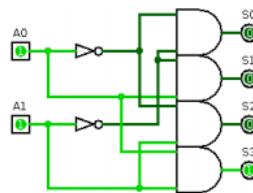
## Kombinační logické obvody (6)



### Binární dekodér

- nastaví (na I) jeden z  $2^n$  výstupů  $S_i$  odpovídající  $n$ -bitové hodnotě na adresním vstupu  $A$

$A_0$	$A_1$	$S_0$	$S_1$	$S_2$	$S_3$
0	0	I	0	0	0
I	0	0	I	0	0
0	I	0	0	I	0
I	I	0	0	0	I



Obrázek: Pravdivostní tabulka a schéma zapojení bin. dekodéru se čtyřmi výstupy

- použití: dekodér adresy pro výběr místa v paměti

## Kombinační logické obvody (7)



### Binární sčítáčka

- čísla ve dvojkové soustavě = binárně reprezentovaná
- platí stejná pravidla aritmetiky jako v desítkové soustavě, např. (+ je zde aritmetické sčítání!):

$$0 + 0 = 0 \quad 0 + I = I \quad I + I = IO$$

- sčítáčka sečte binární hodnoty v každém řádu dvou  $n$ -bitových proměnných  $A$  a  $B$  podle pravidel aritmetiky pro sčítání, tj. s přenosem hodnoty do vyššího řádu
- realizuje log. funkce součtu  $S_i$  v řádu  $0 \leq i < n$  a přenosu  $r_i$  z řádu  $i$  do vyššího řádu:

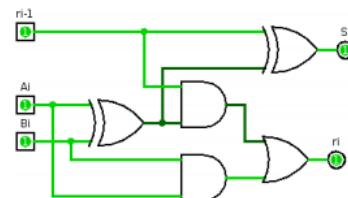
$$S_i = A_i \oplus B_i \oplus r_{i-1} \quad r_i = A_i \cdot B_i + (A_i \oplus B_i) \cdot r_{i-1}, \quad r_{-1} = 0$$

## Kombinační logické obvody (8)



### Binární sčítáčka

$A_i$	$B_i$	$r_{i-1}$	$S_i$	$r_i$
0	0	0	0	0
0	0	I	I	0
0	I	0	I	0
0	I	I	0	I
I	0	0	I	0
I	0	I	0	I
I	I	0	0	I
I	I	I	I	I



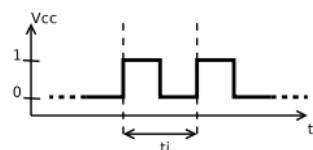
Obrázek: Pravdivostní tabulka a schéma zapojení jednobitové sčítáčky (pro řad  $i$ )

- vícebitová: zřetězené zapojení jednobitových pro každý řad vícebitových proměnných od nejméně významného po nejvýznamější s přenosem do vyššího řádu
- použití: (aritmetické) sčítání binárně reprezentovaných 8-, 16-, 32-, atd. bitových čísel

## Sekvenční logické obvody (1)



- stavy na výstupech obvodu (tj. funkční hodnota) závisí nejen na okamžitých stavech na vstupech (tj. hodnotách proměnných), ale také na přechozích stavech na vstupech
- předchozí stavy na vstupech zachyceny **vnitřním stavem obvodu**
- nutné identifikovat a synchronizovat stavy obvodu v čase
- čas: periodický impulsní signál = "hodiny" (clock), diskrétně určující okamžiky synchronizace obvodu, generovaný krystalem o dané frekvenci



Obrázek: Časový signál "hodin" (clock)

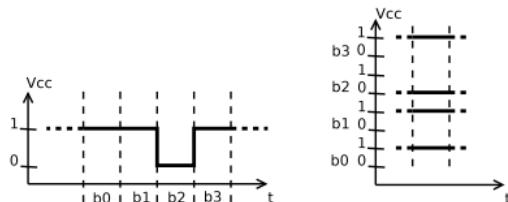
- zpětné vazby z (některých) výstupů na (některé) vstupy

## Sekvenční logické obvody (2)



Přenos dat (hodnot vícebitových log. proměnných):

- **sériový**: bity (hodnoty 0/I) přenášeny postupně v čase za sebou po jednom datovém vodiči
- **paralelní**: bity přenášeny zároveň v čase po více datových vodičích
- úlohy transformace mezi sériovým a paralelním přenosem



Obrázek: Sériový a paralelní přenos dat

## Sekvenční logické obvody (3)



### Klopné obvody

- nejjednodušší sekvenční obvody

druhy:

- **astabilní**: nemají žádný stabilní stav, periodicky (např. podle hodinových impulsů) překlápi výstupy z jednoho stavu do druhého; použití jako generátory impulsů
- **monostabilní**: jeden stabilní stav na výstupech, po vhodném řídícím signálu je po definovanou dobou ve stabilním stavu; použití k vytváření impulsů dané délky
- **bistabilní**: oba stavy na výstupech stabilní, zůstává v jednom stabilním stavu dokud není vhodným řídícím signálem překlopen do druhého; použití pro realizaci **paměti**

Řízení:

- **asynchronně** signály (0 nebo I) na datových vstupech
- **synchronně** hodinovým signálem
- **hadinou** signálu: horní (I) nebo dolní (0)
- **hranami** signálu: nástupní ( $0 \rightarrow I$  u horní hladiny) nebo sestupní ( $0 \rightarrow I$  u dolní hladiny)

## Sekvenční logické obvody (4)



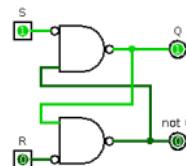
### Klopny obvod (typu) RS

- nejjednodušší bistabilní, základ ostatních
- **jednobitový paměťový člen**
- asynchronní vstupy  $R$  (Reset) pro nulování log. hodnoty na výstupu  $Q$  (v čase  $i$ ) a  $S$  (Set) pro nastavení hodnoty
- kromě výstupu  $Q$  navíc ještě negovaný (invertovaný) výstup  $\bar{Q}$

## Sekvenční logické obvody (5)

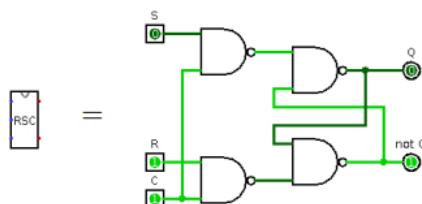
### Klopný obvod (typu) RS

$R$	$S$	$Q_i$	$\bar{Q}_i$
0	0	$Q_{i-1}$	$\bar{Q}_{i-1}$
0	1	1	0
1	0	0	1
1	1	N/A	N/A



Obrázek: Pravdivostní tabulka a schéma zapojení klopného obvodu RS

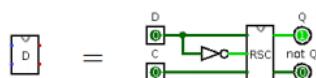
- varianta se synchronizačním vstupem  $C$  s hodinovým signálem



## Sekvenční logické obvody (6)



### Klopný obvod (typu) D



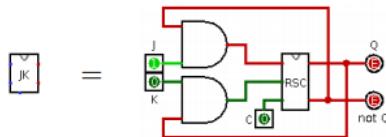
Obrázek: Schéma zapojení klopného obvodu D

- realizace pomocí klopného obvodu RS, navíc mohou být vstupy  $R$  a  $S$
- typ Latch: asynchronní řízení stavu vstupu  $D$  hladinou signálu na vstupu  $C$
- typ D: synchronní (flip-flop) řízení stavu vstupu  $D$  nástupní hranou hodinového signálu na vstupu  $C$
- typ Master-Slave: dvoufázový (master, slave), synchronní řízení stavu vstupu  $D$  nástupní i sestupní hranou hodinového signálu na vstupu  $C$ , rozšíření = **klopný obvod (typu) JK**

## Sekvenční logické obvody (7)



### Klopný obvod (typu) D



Obrázek: Schéma zapojení klopného obvodu JK

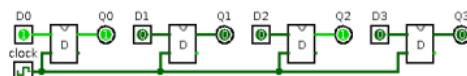
- implementace ve formě integrovaných obvodů, např. MH 7472, MH 7474, MH 7475

## Sekvenční logické obvody (8)



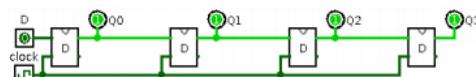
Obvody v počítačích:

- **parallelní registr (strádač):** vícebitová paměť pro hodnotu dodanou paralelně na více vstupů, paralelní zapojení klopných obvodů D



Obrázek: Schéma zapojení čtyřbitového paralelního registru

- **sériový (posuvný) registr:** vícebitová paměť pro hodnotu dodanou sériově na vstupu, sériové zapojení klopných obvodů D, použití pro transformaci sériových dat na paralelní



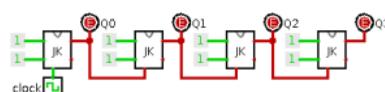
Obrázek: Schéma zapojení čtyřbitového sériového registru



## Sekvenční logické obvody (9)

Obvody v počítačích:

- **čítač:** paměť počtu impulsů na hodinovém vstupu, binárně reprezentovaný počet na vícebitovém výstupu, zřetězené zapojení klopných obvodů JK



Obrázek: Schéma zapojení čtyřbitového čítače

- **sériová sčítačka:** (aritmetické) sčítání log. hodnot dodávaných na vstupy v sériovém tvaru po jednotlivých řádech

## 1.3 Reprezentace čísel a znaků v paměti počítače

### Kódování dat



- data v počítači: celá čísla, čísla s řádovou čárkou (necelá), znaky různých abeced (pro písmena, cifry, symboly atd.) – **alfanumerické znaky**, speciální a řídící znaky
- binární reprezentace = **kódování dat do posloupnosti binárních hodnot**
  - kód (kódování) = zobrazení čísel a znaků na binární hodnoty, pomocí **kódových schémat a tabulek**
  - kód (kódové slovo) = binární hodnota, obecně posloupnost kódových znaků
  - dekódování = převod kódového slova na původní číslo nebo znak
  - různé kódy pro uložení dat, zpracování dat, zabezpečení (uložení, přenosu) dat proti chybám atd.
  - kódující a dekódující log. obvody s pamětí = kodéry, dekodéry

### 1.3.1 Reprezentace čísel

#### Celá čísla



= interval **(min. nekladné, max. nezáporné)** – hranice závisí na (konečném) počtu  $n$  bitů pro reprezentaci a použitém kódu

Nezáporná čísla:

#### Vážený poziční kód

= zápis čísla ve dvojkové poziční číselné soustavě

- např.  $123 = (123)_{10} = [IIIIOII]_2$
- $\langle 0, 2^n - 1 \rangle$

#### Dvojkově desítkový kód (BCD, Binary Coded Decimal)

= zápis desítkových číslic čísla (zapsaného v desítkové soustavě) ve dvojkové soustavě s pevným počtem 4 dvojkových číslic pro každou desítkovou číslici

- např.  $123 = [000I00I000II]_{BCD}$
- $\langle 0, 10^{n/4} - 1 \rangle$ , pro  $n = 4^k$
- neefektivní, složitější log. obvody, snadno dekódovatelný pro člověka, použití pro zobrazení čísel

## Celá čísla



Nezáporná i záporná čísla:

### Přímý kód

- = znaménkový bit (**0** pro nezáporná, **I** pro záporná čísla) + (vážený poziční) kód pro absolutní hodnotu čísla – tzv. sign-magnitude
  - např.  $-123 = [\text{IIIIIOII}]_{S2}$
  - $\langle -2^{n-1} - 1, 2^{n-1} - 1 \rangle$
  - neefektivní (nevyužitý 1 kód), nevhodný pro aritmetiku (testování znaménka, různé postupy sčítání a odčítání)

### Aditivní kód

- = vážený poziční kód pro (nezáporné) číslo rovno součtu kódovaného čísla a **zvolené konstanty**
  - konstanta obvykle  $2^{n-1}$
  - např.  $123 = [\text{IIIIIOII}]_{A(128)}, -123 = [\text{IOI}]_{A(128)}$
  - $\langle -2^{n-1}, 2^{n-1} - 1 \rangle$
  - jinak reprezentovaná nezáporná čísla, složitější násobení, použití pro reprezentaci exponentu u reprezentace čísel s řádovou čárkou

## Celá čísla



### Inverzní (jedničkový doplňkový) kód

- = pro nezáporná čísla vážený poziční kód, pro záporná log. negace všech bitů váženého pozičního kódu absolutní hodnoty, 1. bit má význam znaménka
  - např.  $-123 = [\text{I...0000I00}]_I$
  - $\langle -2^{n-1} - 1, 2^{n-1} - 1 \rangle$
  - neefektivní (nevyužitý 1 kód), nevhodný pro aritmetiku (různé postupy sčítání a odčítání)

### (Dvojkový) doplňkový kód

- = pro nezáporná čísla vážený poziční kód, pro záporná log. negace všech bitů váženého pozičního kódu absolutní hodnoty **zmenšené o 1** (ekv. log. negace všech bitů váženého pozičního kódu absolutní hodnoty s **binárním přičtením I**), 1. bit má význam znaménka
  - např.  $-123 = [\text{I...0000I0I}]_{2'}$
  - $\langle -2^{n-1}, 2^{n-1} - 1 \rangle$
  - efektivní, vhodný pro aritmetiku (odčítání pomocí sčítání se záporným číslem)

## Čísla s řádovou čárkou



= podmnožina racionálních čísel – přesnost omezena na počet platných číslic, z důvodu konečné bitové reprezentace

### Fixní řádová čárka

- = pevně zvolený max. počet  $n$  platných číslic pro necelou část čísla (část za čárkou)
- číslo  $x$  v číselné soustavě o základu  $B$  reprezentováno jako zlomek  $\frac{x \cdot B^n}{B^n}$
- uložena pouze celočíselná část  $x \cdot B^n \Rightarrow$  přibližná reprezentace
- přesnost (rozlišení čísel)  $B^{-n}$ , "přesnost na  $n$  platných číslic za čárkou"
- $\Rightarrow$  celočíselná aritmetika (se zachováním přesnosti)

## Čísla s řádovou čárkou



### Fixní řádová čárka

Reprezentace necelé části čísla:

- necelá část  $F$  čísla jako součet (případně nekonečné) mocninné řady o základu  $B$ :

$$F = a_{-1} \cdot B^{-1} + a_{-2} \cdot B^{-2} + \dots$$

$$(0,625)_{10} = 6 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3} =$$

$$(0,101)_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$$

- tatáž necelá část čísla může být v poziční soustavě o jednom základu vyjádřena konečnou řadou, zatímco v soustavě o jiném základu nekonečnou řadou, např.

$$(0,4)_{10} = 4 \cdot 10^{-1} =$$

$$(0,0110011\dots)_2 = 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 0 \cdot 2^{-5} + 1 \cdot 2^{-6} + 1 \cdot 2^{-7} + \dots$$

- získání zápisu necelé části čísla v dané číselné soustavě a naopak: podobné postupy jako pro celá čísla, jen místo dělení je násobení a naopak

## Čísla s řádovou čárkou



### Fixní řádová čárka

Získání (případně nekonečného) zápisu  $(S_{-1}S_{-2}\dots)_B$  necelé části  $F$  čísla (dané hodnoty) postupným násobením:

```
a_{-1} = 0  
i = -1  
while F > 0 do  
    F = F * B  
    a_i = F mod B  
    F = F - a_i  
    i = i - 1
```

## Čísla s řádovou čárkou



### Fixní řádová čárka

Získání (případně přibližné hodnoty) necelé části  $F$  čísla z jejího (konečného) zápisu  $(S_{-1}S_{-2}\dots S_{-n+1}S_{-n})_B$  postupným dělením:

```
F = a_{-n}
for i = -n + 1 to -1 do
    F = F//B + a_i
F = F//B
```

- // označuje dělení s řádovou čárkou
- převod zápisu necelé části čísla v soustavě o základu  $B^k$  na zápis v soustavě o základu  $B$  (a naopak) stejný jako u celých čísel

## Čísla s řádovou čárkou



### Fixní řádová čárka

Binární reprezentace:

- = BCD nebo **doplňkový kód celočíselné části čísla vynásobeného  $B^n$**  (ekv. doplňkový kód zřetězení vážených pozičních kódů celé a necelé části čísla)
- např. pro doplňkový kód  $-5, 25 = [1\dots 01011]_{2^4}$  (přesnost na 2 platné číslice za čárkou)
- interval čísel, hranice závisí na počtu  $t = m + n$  bitů pro reprezentaci a použitém kódu pro celou a necelou část čísla
- např. pro doplňkový kód:  $\langle -2^{m-1}, 2^{m-1} - 2^{-n} \rangle$
- různé formáty binární reprezentace, např. **Q<sub>m.n</sub>** (Texas Instruments), **fxm.t**
- použití u zařízení bez jednotky pro výpočty s plovoucí řádovou čárkou, při vyžadování konstantní přesnosti nebo kvůli rychlejší celočíselné aritmetice

## Čísla s řádovou čárkou



### Plovoucí řádová čárka

- = **pohyblivá pozice čárky mezi platnými číslicemi celé a necelé části čísla**  $\sim$  počítačová realizace vědecké notace čísla
- číslo  $x$  reprezentováno v **semilogaritmickém tvaru** o základu  $b$ :  $x = s \cdot b^e$ 
  - (pro  $x \neq 0$ )  $-b < s < 0$  nebo  $0 < s < b$ , tj.  $s, e$  takové, že před čárkou je pouze první nenulová číslice  $s$
  - používaný desítkový ( $b = 10$ ) a dvojkový ( $b = 2$ ) základ
  - např.  $123,456 = 1,23456 \cdot 10^2 = 1,929 \cdot 2^6$ ,  $-0,123 = -1,23 \cdot 10^{-1} = -1.968 \cdot 2^{-4}$
- uloženy **znaménko** do 1 bitu, **exponent**  $e$  (včetně znaménka) do  $m$  bitů a **normovaný tvar**  $s$  absolutní hodnoty čísla do  $n$  bitů (significand, "mantissa")
  - exponent v aditivním kódu (s konstantou rovnou  $2^{m-1} - 1$ ) – udává rozsah reprezentace,  $\langle -b^k, b^{k+1} \rangle$ , kde  $b^k = 2^{m-1} - 1$
  - normovaný tvar absolutní hodnoty čísla v kódu pro fixní řádovou čárku (u základu 2 se číslice 1 před čárkou neukládá) – udává přesnost reprezentace  $b^{-n}$
- $\Rightarrow$  přibližná reprezentace

## Čísla s řádovou čárkou



### Plovoucí řádová čárka

Různé formáty s různou přesností (standard **IEEE 754**):

- základ  $b = 2$  i  $b = 10$ , vážený poziční kód pro normovaný tvar
- **single (float, 32 bitů)** – 8 bitů pro exponent, 23 bitů pro normovaný tvar, rozsah  $\sim \langle -10^{38}, 10^{38} \rangle$ , asi 7 platných desítkových číslic

$$\begin{aligned} 123.456 &= [0100001011101101100101111001]_2 \\ -0.123 &= [1011110111110111110011101101101]_2 \end{aligned}$$

- **double (64 bitů)** – 11 bitů pro exponent, 52 bitů pro normovaný tvar, rozsah  $\sim \langle -10^{308}, 10^{308} \rangle$ , asi 16 platných desítkových číslic
- další: **half** (16 bitů, 5 pro exponent), **extended** (long double, 80 bitů, 15 pro exponent), **quad** (128 bitů, 15 pro exponent)
- **speciální "čísla"**:  $-\infty, +\infty$  (exponent samé I, normovaný tvar nulový), ***NaN*** (Not a Number, exponent samé I),  $-0 \neq 0$  (exponent i normovaný tvar nulové)

## Čísla s řádovou čárkou



### Plovoucí řádová čárka

- **aritmetika s plovoucí řádovou čárkou**
  - použité zaokrouhlovací algoritmy a výjimky (pro nedefinované operace)
  - měřítko výkonnosti počítačů (ve vědeckých výpočtech), jednotka **FLOPS** (FLoating point Operations Per Second)
- mnohem širší množina čísel než u fixní řádové čárky na úkor nižší přesnosti



## Text

- = posloupnost tisknutelných znaků = znaků různých abeced (pro písmena, cifry, symboly atd.) – **alfanumerické znaky**
- + speciální a (netisknutelné) řídící znaky – jen některé se zahrnují do **plain textu**

### ASCII (American Standard Code for Information Interchange, 1963)

- standarní kódová tabulka pro kódování znaků **anglické abecedy, cifer, symbolů** (matematických aj.), speciálních (mezera, interpunkce, atd.) a **řídících znaků** (původně pro ovládání dálnopisu, odřádkování, návrat vozíku, tabulátory, backspace aj.)
- každý znak kódován původně do **7 bitů** = 128 znaků
- přidán nejvyšší 8. bit, tj. tabulka rozšířena o dalších 128 znaků: některé **znaky národních abeced**, další speciální znaky (**grafické**, jednotky aj.)

Obrázek: ASCII tabulka

## Text

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	\$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	DA	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	DB	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	DC	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	DD	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	DE	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	DF	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

## Text

### ASCII

- několik rozšíření pro různé národní abecedy – různé kódové tabulky rozšířené ASCII, např. **ISO 8859-1**, CP437 (IBM PC, OS MS DOS)
- pro znaky české abecedy (východoevropské/středoevropské jazyky):
  - **ISO 8859-2 (ISO Latin 2)**: standard ISO, používaný v UNIXových operačních systémech (OS)
  - **Windows 1250 (CP1250)**: kód firmy Microsoft, používaný v OS MS Windows, od ISO 8859-2 se liší např. ve znacích š, t, ž
  - **Mac CE**: kód firmy Apple, používaný v Apple MAC OS
  - CP852 (PC Latin 2): kód firmy IBM, používaný v OS MS DOS
  - další (česko-slovenské): kód Kamenických (další používané v OS MS DOS), KOI8-ČS (kód v rámci RVHP) a další
- **ASCII art** – výtvarné umění kresby obrázků pomocí znaků ASCII v neproporcionálním fontu, např. emotikony ("smajlíky"), použití u textových negrafických systémů

Obrázek: ASCII art



## Text

### EBCDIC (1964)

- kódování firmy IBM podle kódu pro děrné štítky
- základní osmibitový, rozšířený 16-bitový, různé pro různé národní abecedy
- **nespojitý pro znaky latinky**, dnes nepoužívaný

## Text



### Unicode (1987–1991)

- rozšíření ASCII nestačí a jsou ad-hoc (např. problematické pro východoasijské, arabské, hebrejské aj. znaky)
- = původně 16-bitová tabulka znaků **UCS-2 (Universal Character Set)**
- později oddělení množiny znaků a kódů pro ně (do tzv. kódových bodů a do binární reprezentace)
- = standard **ISO 10646** (definice UCS-4, 31-bitová) + algoritmy pro texty zprava doleva a oboustranné texty
- UCS = **otevřená množina pojmenovaných znaků všech abeced** a kombinovaných znaků (např. diakritických), v současnosti (2012) více než 110 000 znaků (poslední verze 6.2 z roku 2012), znaky jen přidávány, prostor pro více než milion znaků
- znakové sady = kódování podmnožiny znaků do kódových bodů (nezáporných celých čísel,  $U+hexčíslo$ ), např. původní ASCII a její rozšíření, BMP (Basic Multilingual Plane) = prvních 65534 znaků UCS

## Text



### Unicode

#### Způsob kódovaní (UTF, UCS Transformation Format)

- = kódování kódových bodů do binární reprezentace
- pro jednoznačné kódování celé tabulky Unicode bylo potřeba 21 bitů (hodnoty  $0_{16}$  až  $10FFFF_{16}$ )
- **UTF-8:** do posloupnosti 1 až 6 bytů, kompatibilní s ASCII (7bitové, přímo) a ISO 8859-1 (prvních 128 dvoubajtových), nezávislý na "endián"itě systémů, všeobecně používané (zejména v UNIXových OS a na Internetu a WWW), RFC 3629
  - znaky  $U + 0$  až  $U + 7F$  do 1 bytu  $0_{16}$  až  $7F_{16}$  (přímo)
  - další jako posloupnosti bytů, kde každý má nejvyšší bit roven 1, 1. byte  $C0_{16}$  až  $FD_{16}$  určuje, kolik bytů posloupnost má (počtem nejvyšších jedničkových bitů následovaných 0), 5 bitů pro kód znaku, další byty  $80_{16}$  až  $BF_{16}$ , 6 bitů pro kód znaku, big-endian
  - BMP jen 1 až 3 byty, české 1 nebo 2 byty (diakriticke)
  - byty  $FE_{16}, FF_{16}$  nepoužity

**Unicode****Způsob kódovaní (UTF, UCS Transformation Format)**

Tabulka: Kódování UTF-8

$U + 00000000 - U + 0000007F$	0xxxxxx
$U + 00000080 - U + 000007FF$	110xxxxx 10xxxxxx
$U + 00000800 - U + 0000FFFF$	1110xxxx 10xxxxxx 10xxxxxx
$U + 00010000 - U + 001FFFFF$	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
$U + 00200000 - U + 03FFFFFF$	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
$U + 04000000 - U + 7FFFFFFF$	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

- např. "Příliš" =  $[50C599C3AD6C69C5A1]_{16}$  ("ř" =  $U + 159$ , "í" =  $U + ED$ , "š" =  $U + 161$ )

**Unicode****Způsob kódovaní (UTF, UCS Transformation Format)**

- UTF-16:** do posloupnosti 1 až 2 slov (2 byte), používané zejména v OS MS Windows a prog. jazyku Java, dříve UCS-2 (pevně 16 bitů)
  - znaky  $U + 0$  až  $U + FFFF$  do 2 bytů přímo
  - další znaky do 4 bytů, 1.  $D8_{16}$  až  $DB_{16}$ , 3.  $DC_{16}$  až  $DF_{16}$ , 2 bity pro kód znaku

Tabulka: Kódování UTF-16

$U + 000000 -- U + 00FFFF$	xxxxxxxx xxxx xxxx
$U + 010000 -- U + 10FFFF$	110110xx xxxx xxxx 110111xx xxxx xxxx

- např. "Příliš" =  $[0050015900ED006C00690161]_{16}$
- BOM (Byte-Order Mark, UTF signatura)** = znak  $U + FEFF$  ("nedělitelná mezera nulové šířky") – k rozlišení pořadí ukládání bytů (little/big-endian) v UTF-16 a odlišení UTF-16 od UTF-8, v UTF-16 byty  $FE_{16}FF_{16}$  pro big-endian a  $FF_{16}FE_{16}$  pro little-endian, v UTF-8 tyto byty neplatné, kód znaku jsou byty  $EF_{16}BB_{16}BF_{16}$  (ve standardu explicitně povolené, ale nedoporučované, ale OS MS Windows používají k označení UTF-8)

## Text



### Unicode

#### Způsob kódovaní (UTF, UCS Transformation Format)

- další: UTF-32/UCS-4 (pevně do 4 byte, příliš nepoužívané), **UTF-7** (do posloupnosti 7-bitových ASCII znaků, pro e-mail), aj.

## Text



### Kód pro nový řádek

- různý v různých operačních systémech
- **LF (Line Feed, odřádkování, A<sub>16</sub>)**: v UNIXových OS
- **CR (Carriage Return, návrat vozíku, D<sub>16</sub>) + LF**: v OS MS DOS a Windows
- **CR**: v OS od firmy Apple

### Escape sekvence

- = posloupnosti znaku **ESC (Escape, 1B<sub>16</sub>)** následovaného jedním nebo více znaky z ASCII
- rozšíření ASCII se speciálním významem sekvencí – pozice kurzoru, barva nebo font textu na obrazovce znakového terminálu, přepnutí módu zařízení aj.

## 1.4 Osobní počítač (PC), základní deska, chipset a sběrnice (interní, externí)

### 1.4.1 Osobní počítač (PC)

#### Osobní počítač



##### Personal Computer, PC

- příbuznost a (částečná nebo úplná) kompatibilita s počítači **IBM PC** (1981), procesory **Intel x86** kompatibilní
  - XT (eXTended, 1983) – 8-bitový, procesor Intel 8088, 4,77 MHz, 16–256 kB RAM, operační systém MS DOS
  - AT (Advanced Technology, od 1984) – 16/32/64-bitové, procesory Intel od 80286, (původně) až 16 MB RAM
  - ztráta vlivu na standardizaci po PS/2 (1987), ve prospěch firem Intel (hardware) a Microsoft (software)
- = základní koncepce technického provedení počítače
- dodnes převládající koncepce mikropočítačů (otevřená politika IBM vs. uzavřená jiných firem)

#### Počítačová sestava



**Počítač, počítačová sestava** = „stavebnice“ počítače s modulární architekturou ⇒ variabilita, rozšiřitelnost, vyváženosť, ...

- **hardware** = technické vybavení počítače, fyzické součásti („železo“), elektronická digitální a elektromechanická zařízení
- **software** = programové vybavení počítače, operační systém a aplikace, **firmware** = programy vestavěné do hardware (např. BIOS)
- neustálý vývoj, posouvání hranic možností, specializace

## Součásti počítače



### Skříň (case, chassis)

- provedení (form factor): klasické desktop, (mini/midi/big)tower, rackové, další laptop/notebook/netbook, tablet, atd.
- korespondence s rozměry **základní desky** (otvory pro zdroj, konektory, lišty pro přídavné karty)
- = kovová (plechová) konstrukce s plastovými kryty
- **šachty** pro mechaniky výměnných médií a karet (5, 25'', 3, 5'', PCMCIA) aj., tlačítka pro zapnutí, popř. reset, signalizace a indikátory (LED aktivit, detekce otevření), konektory pro USB, audio aj.
- výrobci: Antec, AOpen, ASUS, Chiftec, Cooler Master, DFI, Ever Case, Foxconn a další

## Součásti počítače



### Zdroj napájení (power supply) nebo Baterie

- zajišťuje el. napájení stejnosměrným proudem všech (vnitřních) součástí počítače
- konektory **PC Main** (P1, 20/24 pinů, do základní desky), **ATX12V** (P4, 4/8 pinů, do základní desky, pro procesor), **4- a 15-pinové** (pro disková zařízení PATA, SATA), **6/8-pinové** (pro PCI-Express), a **IEC C14** pro elektrickou šňůru
- typicky  $+3.3, \pm 5, \pm 12 V$ , 250–1000 W pro osobní počítače, 25–100 W pro přenosné s baterií, více zdrojů (redundantních) pro servery a pracovní stanice, efektivita od 75 do 90 %
- korespondence s rozměry základní desky (konektory) a skříně, AT nebo **ATX**
- výrobci: Antec, Enermax, Foxconn a další

## Součásti počítače



**Vnitřní součásti:** základní deska, procesor, operační pamět, rozšiřující karty (grafická, zvuková, síťová aj.), pevné disky, mechaniky výměnných médií (CD/DVD, floppy, aj.), zdroj, ventilátory

**Vnější součásti:** displej, klávesnice, myš, touchpad

**Periferie:** disková zařízení (pro pevné disky i výměnná média), síťová zařízení (přepínače, směrovače, přístupové body, modemy aj.), multimediální zařízení (reproduktoři, mikrofon webová kamera, antény), tiskárna, plotter, skenner, tablet, trackball, joystick a další

- některé vnější součásti a periferie mohou být součástí skříně (notebook, tablet, Mac), např. displej, klávesnice, touchpad, síťová a multimediální zařízení aj.

#### 1.4.2 Základní deska

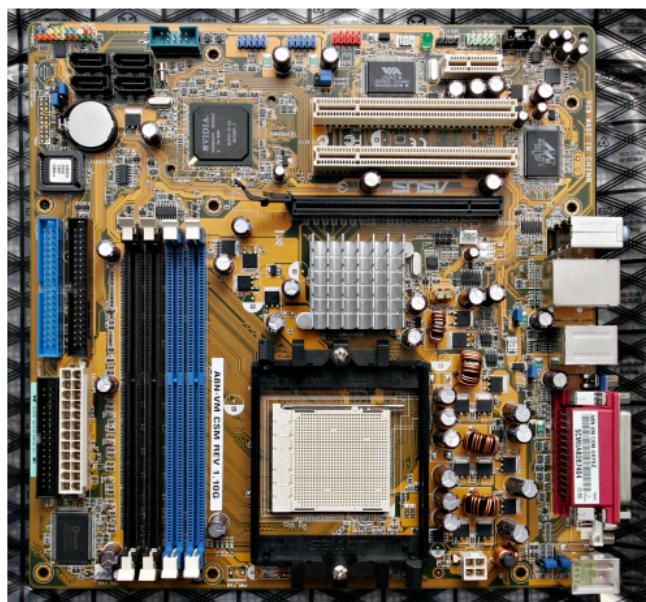
##### Základní deska (mainboard, motherboard)



Obrázek: Základní deska

- = základní součást, ke které (na kterou) se připojují další zařízení, které propojuje: procesor(y), paměti, přídavné karty, disková zařízení, periferie a další
- = vícevrstvý (obvykle) obdélníkový plošný spoj s obvody propojujícími zařízení pomocí vnitřních **sběrnic**

##### Základní deska (mainboard, motherboard)



## Základní deska (mainboard, motherboard)



- formáty (form factor):
  - PC/XT (IBM) – první pro osobní počítače
  - **AT** (IBM) – 305 × 350 mm, varianta Baby
  - **ATX** (Intel, 1995) – 244 × 305 mm, nejpoužívanější, de facto standard, varianty micro (244 × 244 mm), Extended, Flex, Ultra
  - **BTX** (Intel) – 266 × 325 mm, lepší chlazení a napájení než ATX, varianty micro, Extended aj., neujal se
  - **ETX** – 95 × 114 mm, v embedded počítačích
  - odpovídající skříň, různé konektory pro napájení od zdroje, různé rozmístění konektorů pro periferie
- výrobci: Aopen, ASRock, ASUS, Biostar, EPoX, Foxconn, Gigabyte Technology, Intel, Jetway, Micro-Star, Palit, Soyo, VIA a další

## Základní deska (mainboard, motherboard)



### Sběrnice (bus)

= paralelní nebo sériová soustava vodičů propojujících zařízení pro komunikaci a přenos dat (řízeným **protokolem**)

- parametry:
  - **šířka přenosu (bit)** – určuje, kolik bitů lze najednou přenést
  - **frekvence (MHz)** – frekvence hodinového signálu
  - **rychlosť/propustnosť (MB/s)** – určuje množství dat přenesených za jednotku času, frekvence × šířka (v bytech)
- vnitřní: na základní desce (součást jejích obvodů), vesměs paralelní
- vnější (rozhraní): k diskovým zařízením a periferiím, kombinované nebo sériové
- **synchronní** – zařízení synchronizována, většina vnitřních
- **multimaster** – může být řízena více zařízeními než jedním (typicky procesorem), tzv. **busmastering**

## Základní deska (mainboard, motherboard)



### Sběrnice (bus) – části:

- **adresová** – výběr adresy v paměti nebo zařízení na sběrnici, šířka 3 až 64 bitů – určuje, s jak velkou pamětí nebo s kolika zařízeními lze (přímo) pracovat
- **datová** – přenos dat po sběrnici, šířka 1 až 128 bitů, udává "bitovost" sběrnice
- **řídící** – řízení zařízení na sběrnici pomocí řídících a stavových informací, šířka 1 až 8 bitů – určuje počet řídících signálů a stavů

## Základní deska (mainboard, motherboard)

1

### Vnitřní sběrnice

#### procesorová, systémová (CPU, front side bus)

- připojená zařízení: procesor(y), paměti (cache, operační), **severní můstek čipsetu** (příp. řadič kanálů (periferní procesory, u mainframe počítačů))
- 8- až 64-bitová, šířka datové části (většinou) koresponduje s adresní
- frekvence (FSB): 66, 100, 133, 266, 400, 533, 1066, ..., 1600 MHz
- **patice (socket)** = konektor pro procesor
- **sloty** = konektory pro operační paměti, příp. procesor

## Základní deska (mainboard, motherboard)



### Vnitřní sběrnice

#### rozšiřující, lokální (expansion, local bus)

- určuje standard pro připojená zařízení – **přídavné karty**
- **sloty** pro karty: grafické, zvukové, síťové, multimediální, diskové řadiče, pro periferie aj.
- integrované karty – součástí základní desky, dnes běžně zvuková, síťová, diskové řadiče, někdy i grafická (tzv. all-in-one)
- **ISA (Industry Standard Architecture)** – nejstarší pro IBM PC, původně pro procesor Intel 80286, 8/16-bitová, frekvence 4,77/8,33 MHz, **manuální konfigurace karet** pomocí tzv. **jumperů** (propojka vodičů) nebo v BIOSu, dnes se u osobních počítačů téměř nevyskytuje, přetravá v průmyslových počítačích
- **MCA (Microchannel Architecture)** – od IBM pro procesory Intel 80386 jako náhrada za ISA, 16/32-bitová, frekvence 10–25 MHz, umožňuje busmastering, nekompatibilní s ISA, nerozšířila se mimo IBM (PS/2)

## Základní deska (mainboard, motherboard)



### Vnitřní sběrnice

#### rozšiřující, lokální (expansion, local bus)

- **EISA (Extended ISA)** – zpětně kompatibilní s ISA, pro procesory Intel 80386, 8/16/32-bitová, frekvence 8,33 MHz umožňuje busmastering, dnes nepoužívaná
- **VLB (VESA Local Bus)** – pro procesory Intel 80486, 32-bitová, závislá na ISA, počet slotů klesá s frekvencí 25–50 MHz (např. 3 při 33 MHz, 1 při 40 MHz), dnes nepoužívaná
- **PCI (Peripheral Component Interconnect)**
  - od Intelu pro procesory Intel Pentium, umožňuje busmastering
  - umožňuje zařízením **přímý přístup do operační paměti**, použití i v jiných počítačích než IBM PC kompatibilních
  - 64-bitová (Pentium), 32-bitové přenosy (pro procesory Intel 80486)
  - frekvence 33, 66, 100, 133 MHz
- **Plug & Play** (PnP, 1992, Intel, Microsoft, Compaq) – standard pro **automatickou konfiguraci karet**, typu, parametrů a bezkonfliktní přiřazení zdrojů (viz BIOS)
- univerzální pro všechny typy karet, dnes postupně nahrazována PCI Express



## Základní deska (mainboard, motherboard)

### Vnitřní sběrnice

#### rozšiřující, lokální (expansion, local bus)

##### ■ AGP (Accelerated Graphic Port)

- od Intelu pro procesory Intel Pentium II
- propojení **grafických karet** přímo s procesorem a operační pamětí (podobně jako procesorová sběrnice)
- 32-bitová, frekvence základní 66 MHz = AGP 1×, pak AGP 2×, 4×, 8× – různý počet bitů za takt
- dnes nahrazena PCI Express

##### ■ PCI Express (PCI-E)

- nástupce PCI (a AGP), nezpětně kompatibilní! (existuje zpětně kompatibilní rozšíření **PCI-X**)
- 1–32-bitová, 1,25 GHz
- 1× (různé karty), 16× (grafické karty), verze 3.0
- další: průmyslové (VME), ACR, AMR, CNR – pro připojení modemových a zvukových karet, dnes nepoužívané



## Základní deska (mainboard, motherboard)

### Vnější sběrnice a rozhraní

Na ploše základní desky:

- **rozhraní/sběrnice pevných disků a mechanik výměnných médií** (CD/DVD, pam. karty aj.): IDE/PATA, SATA, SCSI, Fibre Channel
- **patice pro cache paměti**, BIOS
- rozšiřující **konektory**: pro sběrnice USB, FireWire, zvukové konektory
- konektory (piny) pro další zařízení: napájení, aktivní chladiče (ventilátory), tlačítka, signalizace a indikátory, reproduktor ve skříni aj.

Konektory na (zadním) panelu základní desky:

- **integrovaných karet**: zvukové (stereo jack, optické), síťové (RJ-45), grafické (VGA = D-SUB, DVI, HDMI, DisplayPort)
- **vnějších sběrnic** USB, FireWire, eSATA, I<sup>2</sup>C, pro periferie
- **vstupně/výstupních periferií** (čip **Super I/O**): klávesnice, myš (PS/2), paralelního (Centronics, LPT), sériového (RS 232, COM), MIDI, infra rozhraní (porty)
- **pro přidavné karty**: PCMCIA, ExpressCard aj.



## Základní deska (mainboard, motherboard)

### **BIOS, Basic Input Output System**

- = program poskytující **základní nízkoúrovňové služby**: start počítače, vstupní (obsluha klávesnice, myši), výstupní (text, grafika), dále např. datum a čas, správa napájení, síťové
  - rozhraní mezi hardwarem a operačním systémem
  - umožňuje **základní konfiguraci hardwaru počítače** (tzv. **SETUP**): zapnutí/vypnutí zařízení, základní nastavení zařízení (parametrů, přiřazení zdrojů – přerušení, DMA kanály, vstupní/výstupní adresy, přiřazený paměťový rozsah aj.), zařízení pro zavedení operačního systému
- = firmware uložený v paměti ROM (Flash EEPROM) na základní desce
  - konfigurační data v paměti CMOS RAM zálohované baterií
  - výrobci: Award, Phoenix, Ami



## Základní deska (mainboard, motherboard)

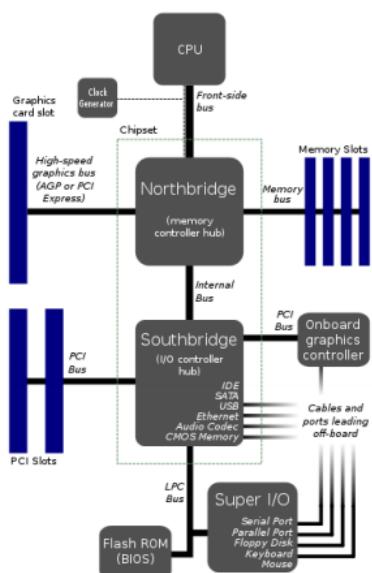
- nejvíce ovlivňuje kvalitu počítačové sestavy, zvláště vzhledem k dalšímu rozšiřování a modernizaci
- **parametry**: typ patice/slotu pro procesor(y) a použitelné procesory, chipset, počet a typy slotů pro paměti a přídavné karty, integrované karty, rozhraní a konektory pro periferie, rychlosti sběrnic

### 1.4.3 Chipset a sběrnice (interní, externí)

#### Čipová sada (chipset)

= integrované obvody (s pasivním chladičem – žebrovaná měď) na základní desce pro řízení pamětí a sběrnic, propojení procesoru, sběrnic a připojení dalších zařízení

- konstruované pro konkrétní typy a počty/množství procesorů a pamětí
- **severní můstek, systémový řadič (north bridge, memory controller hub)**
  - propojuje procesorovou sběrnici (procesor, paměti) s vnitřními sběrnicemi (AGP, PCI Express) a **jižním můstkem** (můstky, interní sběrnice)
  - obsahuje, pokud je, **integrovanou grafickou kartu**
  - dříve (dnes součást procesoru) obsahoval např. **řadič operační paměti** a **řadič cache paměti** (= vyrovnávací paměti mezi různě rychlými zařízeními na frekvenci rychlejšího, např. procesorem a operační pamětí)



## Čipová sada (chipset)

### ■ jižní můstek, vstupně/výstupní řadič (south bridge, I/O controller hub)

- propojuje severní můstek a vnitřní sběrnice (PCI, PCI Express) s vnějšími, příp. se sběrnicí ISA
- obsahuje např. **řadič diskových zařízení a polí, řadič DMA** (Direct Memory Access, umožnění přímého přístupu zařízení do operační paměti), **řídící obvody vnějších sběrnic a rozhraní, připojení čipu BIOSu** (sběrnice LPC k čipu Super I/O), další **integrované karty** (zvuková, síťová aj.)
- výrobci: Intel, AMD, NVidia, VIA Technologies, SiS a další

## Rozhraní/sběrnice pro disková zařízení



- = rozhraní/sběrnice pro komunikaci řadiče diskových zařízení a diskových paměťových (úložných) zařízení

### IDE/ATA/ATAPI

#### ■ IDE (Integrated Drive Electronics) / ATA 1 (Advanced Technology Attachment)

- řídící jednotka (řadič) integrován do diskového zařízení – pevného disku
- max. 2 disky, jeden v režimu **master** nebo single, druhý v režimu **slave**, nastavení jumpery na disku
- 40-pinový konektor na základní desce nebo přídavné kartě (i zvukové, včetně konektoru pro floppy disk a paralelního a sériových portů), 40-žilový kabel pro 1 nebo 2 disky
- adresace dat metodou **CHS (Cylinder/Head/Sector)** – max. 1024/256/64, max. kapacita disku 512 MB, po přemapování cylindrů na hlavy až 8 GB (viz pevný disk dále)
- přenosové rychlosti 3,3–8,3 MB/s (**PIO** 0–2, prakticky do 2), 2,1–8,3 MB/s (**DMA** 0–2), 4,2 MB/s (**MDMA** 0)

## Rozhraní/sběrnice pro disková zařízení



### IDE/ATA/ATAPI

#### ■ EIDE (Enhanced IDE) / ATA 2–7

- zpětně kompatibilní s IDE, až 2 rozhraní IDE = až 4 zařízení, autodetect zařízení i jejich režimů (**cable select**)
- řadič na základní desce nebo přídavné kartě, blokový přenos
- adresace metodou **LBA (Logical Block Addressing)** – logické lineární adresování sektoru pomocí 28-bitové adresy, max. kapacita zařízení 128 GB
- **SMART (Self Monitoring Analysis and Reporting Technology)** – monitorování chyb čtení/zápisu, teploty aj., ATA 3
- **ATA/ATAPI 4 (ATA Packet Interface)** – a.k.a. ATA 33, příkazy SCSI (viz dále) přes ATA, standard pro disková zařízení (mechaniky) s výměnnými médiemi (CD/DVD, ZIP), 80-žilový kabel (stínění), CRC
- 48-bitové LBA, max. 128 PB – ATAPI 6 a.k.a. ATA 100
- přenosové rychlosti ATA 2, 3: 3,4–16,7 MB/s (PIO 3,4, prakticky 6), 13,3–16,7 (MDMA 1,2), ATA 4: 16,7–33 MB/s (**Ultra DMA** 0–2/33), ATAPI 5: 44,4–66,7 MB/s (UDMA 3,4/66), ATAPI 6: 100 MB/s (UDMA 5/100) a ATAPI 7: 133 MB/s (UDMA 6/133)

## Rozhraní/sběrnice pro disková zařízení



### IDE/ATA/ATAPI

#### ■ **SATA (Serial ATA)**

- sériová sběrnice (až 3 GHz), 7-pinový konektor, 7-žilový kabel pro každý disk
- řadič na základní desce: režimy PATA, RAID, AHCI
- **NCQ (Native Command Queuing)** – optimalizace pořadí operací čtení/zápisu, SATA 2
- připojení/odpojení zařízení za chodu (hot plug/swap)
- přenosová rychlosť 150 MB/s (SATA 1), 300 MB/s (SATA 2), 600 MB/s (SATA 3)
- **AHCI (Advanced Host Controller Interface)** – univerzální rozhraní pro detekci, konfiguraci a komunikaci se SATA řadičem
- vnější **eSATA** konektor
- předchozí ATA označováno **PATA (Parallel ATA)**

## Rozhraní/sběrnice pro disková zařízení



### SCSI (Small Computer System Interface)

- **host adapter** – řadič provozu na datovém kabelu (sběrnici), přídavná karta nebo externí připojené přes paralelní port, 50/68/80-pinový konektor (paralelní SCSI)
- max. 8 zařízení (včetně adaptérů, SCSI-1), identifikace číslem ID, sběrnice ukončena terminátorem (na posledním zařízení), 16/32 zařízení (SCSI-2/Fast,Wide, SCSI-3/Ultra2,3)
- paralelní (SPI) i sériové (SSA, FC-AL, **SAS**) sběrnice, logické lineární adresování sektoru
- přenosová rychlosť do 5 MB/s (SCSI-1), 20/40/80/160 MB/s (SCSI-2/3,Fast/Wide/Utra2/3), 320 MB/s (Ultra320 SCSI), 640 MB/s (Ultra640 SCSI)
- vnitřní (pevné disky, mechaniky pro výmenná média) i vnější (tiskárny, skenery) zařízení – konektor pro další zařízení
- i mimo PC, např. Macintosh, Sun, SGI a další
- **iSCSI** – výměna dat síťovým protokolem (nad TCP/IP)

## Rozhraní/sběrnice pro disková zařízení

Další (vnější):

- **Fibre Channel** – optická počítačová síť, jednotky GB/s
- USB, FireWire (IEEE 1394), Thunderbolt – viz dále

## Vnější rozhraní/sběrnice



- pro připojení **vnějších zařízení**: diskových, (mechanik) výmenných médií, rozšiřujících karet v podobě vnějších zařízení (zvukových, síťových, televizních, multimediálních aj.)
- pro připojení (vstupně/výstupních, input/output, I/O) **periferií**: klávesnice, myš, MIDI, tiskárny, skenery atd.
- I/O karta do slotu ISA (dříve) nebo PCI/PCI Express, konektory/porty (příp. na ploše základní desky) vyvedené ze skříně, čip **Super I/O**



## Vnější rozhraní/sběrnice

### Paralelní rozhraní/port (Centronics, LPT)

- = sběrnice pro (synchronní) poloduplexní **paralelní** přenos dat po **8 bitech**
  - 0 až 0,4 V = log. 0, 2,4 až 5 V = log. I (TTL)
  - signály **DATA 0–7**, BUSY, ACK, STROBE (synchronizace) aj., možnost tzv. přeslechů – ovlivňování signálů mezi vodiči
  - rychlosť 12 000 kb/s
  - IEEE 1284 – vstup i výstup (dříve jen výstup), režimy (módy) přenosu dat (nastavení v Setupu BIOSu): **Compatible (Centronics)**, Nibble, Byte, **EPP (Enhanced Parallel Port)**, **ECP (Extended Capabilities Port)**
  - použití: tiskárna (původně), skener (starší), ZIP mechaniky, SCSI zařízení, propojení dvou počítačů
  - 1 port (LPT, PRN), konektory Centronics 36 pinů (dříve), **Cannon 25** pinů



## Vnější rozhraní/sběrnice

### Sériové rozhraní/port (V.24, RS-232(C), COM)

- = sběrnice pro (asynchronní) duplexní **sériový** přenos dat (po dvou datových vodičích) mezi dvěma zařízeními
  - 3 až 15 V = log. 0, -15 až -3 V = log. I, v klidu log. I, **start bit** a **stop bity** log. 0, 5–8 datových bitů, paritní bit
  - signály datové **TxD**, **RxD**, řídící RTS, CTS, DSR, DTR aj., bit zaslán až po potvrzení předchozího
  - rychlosť až 115 200 b/s (Bd, Baud Rate)
  - použití: myš (dříve), modem, propojení dvou počítačů (tzv. null modem), I/O a ovládání různých elektronických zařízení počítačem („bastlení“)
  - 1 nebo 2 porty (COM1,2), konektor **Cannon 9/25** pinů, kabel max. 25 m
  - specializované pro průmysl: RS-422, RS-485



## Vnější rozhraní/sběrnice

### USB (Universal Serial Bus)

- 1995, Intel, Compaq, Hewlet-Packard, Microsoft, NEC, Lucent, Philips
- = **univerzální sériová sběrnice** pro připojení různých zařízení s (později i vysokorychlostním, full) duplexním přenosem dat (po dvou datových vodičích)
- USB 1.0 (1995), **USB 1.1** (1998) – rychlosť 1,5 (Low-speed) a 12 Mb/s (Full-speed), **USB 2.0** (2000) – až 480 Mb/s (Hi-speed, efektivně 280 MB/s), zpětně kompatibilní, **USB 3.0** (2007) – až 5 Gb/s (Super-speed, efektivně 3,2 Gb/s), USB 3.1 (2013) – až 10 Gb/s (efektivně 7,2 Gb/s), zpětně kompatibilní
- řadič USB (**USB Host Controller**) na základní desce – režie protokolu 10-15 %
- **stromová topologie**: (příp. virtualní) kořenový rozbočovač/HUB, typicky součást řadiče, zařízení se připojují k HUBu, max. 127 zařízení
- třídy zařízení: **mass storage** (uložení dat), **HID** (periferie), video, audio, image, printer, wireless aj.



## Vnější rozhraní/sběrnice

### USB (Universal Serial Bus)

- **Master/Slave protokol** – Master (= řadič, v případě počítače, jinak např. mobilní telefon, dynamicky – „On-The-Go“, OTG) vytváří virtuální komunikační kanály (pipes, data v 1ms rámcích), přenosy kontrolní (připojení a konfigurace zařízení), sekvenční (přenos více dat jedním směrem), přerušení (potvrzení přenosu, menší přenosy), isochronní (pravidelné menší přenosy v reálném čase, např. zvuk), kontrola CRC
- vlastnosti: **Plug & Play**, připojení/odpojení zařízení **za chodu (hotplug)**, **napájení** zařízení (a nabíjení) 5 V, 0,75-4,5 W (nabíjení až 7,5 W, žluté „sleep-and-charge“ porty), dle specifikace „Power Delivery“ (2012) až 20 V/100 W!!! („PD-aware“ kabel, micro 60 W)
- použití: vnější součásti a periferie počítače všeho druhu, ostatní digitální elektronika (fotoaparáty, videokamery, mobilní telefony atd.)
- vlastní konektory typu **A, B, mini-A/B a micro-A/B**, kabel max. 5 m
- **Wireless USB** – 3.1–10 GHz, 110–480 Mb/s na 10–3 metry, herní ovladače, tiskárny, skenery, přenosné přehrávače a disky aj.



## Vnější rozhraní/sběrnice

### FireWire (IEEE 1394, i.Link)

- 1995, Apple
- = sériová sběrnice pro vysokorychlostní a real-time plně duplexní přenos dat
- **FireWire 400 (IEEE 1394a)** – rychlosť 400 Mb/s, **FireWire 800 (IEEE 1394b)** – 800 Mb/s (metalický) nebo 3,2 Gb/s (optický konektor), FireWire S800T (IEEE 1394c) – 800 Mb/s přes RJ-45 (Ethernet) konektor, prakticky vyšší než u USB
- řadič (na základní desce) využívá DMA přenosy, až 63 zařízení ve stromové topologii, až 100 m, peer-to-peer komunikace
- vlastnosti: **Plug & Play, napájení** zařízení až 60 W (12 V, max. 30 V)
- použití: videokamery, externí pevné disky, (jednoduché) síťové propojení počítačů
- 3 typy konektorů



## Vnější rozhraní/sběrnice

### Thunderbolt

- 2009, Intel (2011, Apple)
- = sériová sběrnice pro plně duplexní (HD, 4K) video a vysokorychlostní přenos dat
- **kombinace PCI Express a DisplayPortu** – nízkoúrovňové přímé připojení zařízení (!!), přímé připojení displejů
- rychlosť 2x10 Gb/s (verze 1), 20 Gb/s (2)
- max. 6 zařízení v různých topologiích
- původně optický („Light Peak“), dnes metalický kabel s až 10 W napájením, konektor dříve kombinovaný USB, dnes **mDP** (mini DisplayPort)
- použití: video zařízení (displeje), disková zařízení

## 1.5 Procesor (CPU), vykonávání instrukcí, podprogramy a zásobník, přerušení

### 1.5.1 Procesor

#### Procesor (central processing unit, CPU)



Obrázek: Procesor

- = centrální prvek počítače vykonávající sekvenčně (až na skoky) instrukce programu uloženého v operační paměti
- = **mikroprocesor** = integrovaný obvod/čip (velmi vysokého stupně integrace) v patici (socket) nebo slotu na základní desce
  - pasivní (dříve) a aktivní chlazení (ventilátor)
  - "mozek počítače"

#### Procesor (central processing unit, CPU)



- části:
  - **řadič (řídící jednotka, central unit, CU)** – zpracovává instrukce programu nad daty čtenými z registrů, paměti nebo vstupního zařízení, výsledky zapisuje do registrů, paměti nebo výstupního zařízení
  - **aritmeticko-logická jednotka (ALU)** – realizuje aritmetické a logické instrukce, celočíselná a v plovoucí řádové čárce
  - **registry** – paměťové buňky přímo v procesoru, nejrychleji dostupná paměť procesoru
  - **vyrovnávací paměti cache (L1, L2, L3), řadič cache**
  - **sběrnicová a stránkovací jednotka, řadič operační paměti, ...**
  - dnes navenek von Neumannova koncepce, vnitřně harwardská
- frekvence:
  - **vnitřní (taktovací)** – nominální frekvence procesoru, 1 MHz až několik (3) GHz, odvozena od vnější frekvence pomocí **násobitele** nastavovaného jumpery (dříve) nebo v BIOSu, násobky 0,5, rozsah 1,5 až 15 (u nových procesorů), typicky od 3 do 7,5
  - **vnější** – frekvence procesorové sběrnice na základní desce, určená severním můstkom chipsetu

## Procesor (central processing unit, CPU)



- **patice (socket):** vývody procesoru ve formě pinů (dříve) nebo plošek (dnes), téměř výhradní, např. DIP (40 pinů) pro Intel 8086/8, PLCC (68 pinů) pro Intel/AMD 80186-386, Socket 1–3 (169–238 pinů) pro Intel/AMD 80486, Intel Pentium OverDrive, Socket 4,5,7 (273–321 pinů) pro Intel Pentium (MMX), AMD K5, K6, Socket 370 (370 pinů) pro Intel Celeron, Pentium III, **Socket 478** pro Intel Celeron, Pentium 4, Socket A/462 pro AMD Athlon (XP), Duron, **Socket 754** pro AMD Athlon 64, **Socket 940** pro AMD Opteron, **Socket 775** pro Intel Pentium 4/D, Celeron, Core 2, **Socket AM2/+** pro AMD Athlon 64, Opteron, Sempron, Phenom, **Socket 1155/1366/2011** pro Intel Core i3/5/7, **Socket FM1/2** pro AMD A4/6/8/10 a další
- **slot:** podobně jako přídavné karty do rozšiřujících sběrnic, vyjímečně, např. Slot 1 (242 pinů) pro Intel Celeron, Pentium Pro, II a III, Slot 2 (330 pinů) pro Intel II Xeon, Slot A (242 pinů) pro AMD Athlon K7
- **výrobci (pro osobní počítače):** Intel, AMD, ARM Ltd., VIA Technologies, IBM, Transmeta, Texas Instruments a další specializované

## Procesor (central processing unit, CPU)



### Instrukční sada

- = množina všech instrukcí procesoru, pevně zabudována (dnes upravitelná/rozšiřitelná, instrukce složeny z tzv. mikroinstrukcí)
- **CISC (Complete Instruction Set Computer)** – tzv. úplná instrukční sada, všechny možné déle trvající instrukce, u osobních počítačů navenek procesoru, např. Intel, AMD
- **RISC (Reduced ISC)** – redukovaná instrukční sada, jen několik základních jednoduchých rychlých instrukcí, ostatní složitější jsou složeny ze základních, např. IBM Power PC, u osobních počítačů interně u novějších procesorů Intel, AMD
- instrukce přesunu (mezi registry, operační pamětí), aritmetické, logické (log. operace, posuvy, rotace), skoku, vstupně/výstupní (pro práci s periferiemi), ostatní (řídící aj.), a další

## Procesor (central processing unit, CPU)



### Registry

- = paměťové buňky přímo v procesoru, pro potřeby vykonávání instrukcí
- velikost podle datové části procesorové sběrnice, 8, 16, 32, 64 bitů
- **univerzální (datové)** – pro operandy, mezivýsledky a výsledky instrukcí, např. EAX (RAX), EBX (RBX) atd.
- se stanoveným **významem** – pro řízení vykonávání programu, např. EIP, ESP, EFlags, pro implicitní operandy a výsledky, např. ESI, EDI, pro řízení procesoru, např. CRx aj.
- **matematického koprocesoru** pro operace v plovoucí řádové čárce (**Floating Point Unit, FPU**), např. FPx

## Procesor (central processing unit, CPU)



### Procesory Intel

- vedoucí výrobce procesorů pro osobní počítače, od 1972 i další (IBM, AMD, ARM) – viz literatura
- **4004** (1971) – první, 4-bitový, 108 kHz, 2 300 tranzistorů
- 8008 (1972), 8080 (1974), 8088 (1979) – 8-bitové, 2–5 MHz, 6–29 tis. tranzistorů
- **8086** (1978) – 1. 16-bitový
- **80286** (1982) – 16-bitový, 24-bitová adresová, až 12 MHz, 130 tis. tranzistorů
  - **reálný režim** – po inicializaci procesoru, podle 8086
  - **chráněný režim** – zapnutí instrukcí z reálného (bez možnosti zpět), **stránkování paměti** (stránky = kusy fyzické paměti) a **virtuální paměť**, adresace až 16 MB fyzické operační paměti a 1 GB virtuální, 4 úrovně ochrany programu (Ring 0 až 3)

## Procesor (central processing unit, CPU)



### Procesory Intel

- **80386** (1985) – 1. 32-bitový, vnější 16 MHz, 280 tis. tranzistorů, verze SX (do základních desek pro 16-bitový 80286), DX, **segmentace paměti** (segmenty = oblasti virtuální paměti programu s různými právy), 32-64 kB **L2 cache** na základní desce
- 8087, 80287, 80387SX, 80487SX – **matematické koprocessory**, na základní desce vedle procesoru
- **80486** (1989) – vnější 25 MHz, 1,2 mil. tranzistorů, verze SX (vylepšený 80386, 8 kB **L1 cache**), DX, DX/2 (dvojnásobná vnitřní frekvence), DX/4 (trojnásobná), **integrovaný matematický koprosesor**, **pipelining** = více rozpracovaných instrukcí zároveň
- **Pentium** (1993) – 64-bitový (vnitřně 32!), 32-bitová adresová, vnější 60 MHz, 3.1 mil. tranzistorů, 16 kB L1 (8 kB pro instrukce, 8 kB pro data, **harwardská koncepce**), rysy RISC instrukční sady, **superskalární architektura** = více (2) proudů vykonávání instrukcí, umožňuje provádět více (2) instrukcí současně, 2 ALU, "předvídaní" cílové adresy instrukcí podmíněných skoků, klony AMD K5, Cyrix M1

## Procesor (central processing unit, CPU)



### Procesory Intel

- **Pentium MMX** (MultiMedia eXtension), MMX2 – až 200 MHz, 57 a 70 (vektorových) instrukcí pro zpracování multimediálních dat (s opakujícími se smyčkami, paralelní vykonávání, s plovoucí řádovou čárkou, využití registrů FPU), architektura SIMD (Single Instruction Multiple Data, paralelní zpracování dat – vektorů dat), data zpracovávána po 64 bitech
- **Pentium Pro** (P6, 1995) – 36-bitová adresní část sběrnice, druhý čip v pouzdře pro 256 kB až 1 MB L2 cache, RISC jádro, 5 paralelních jednotek (2 ALU, 2 sběrnicové, 1 FPU)
- **Pentium II** (1997) – od 233 MHz, vnější 100 MHz, 7,5 mil. tranzistorů, nové pouzdro (S.E.C.) do slotu Slot 1, verze Xeon pro servery a pracovní stanice (vyšší výkon), Mobile (M) pro notebooky (nižší spotřeba)
- **Celeron** – vnější 66 MHz, bez L2 cache → pomalý, od verze 300A 128 kB L2 cache, Slot 1, pouzdro (PPGA) do patice Socket 370 (existuje redukce na Slot 1)

## Procesor (central processing unit, CPU)



### Procesory Intel

- **Pentium III** (1999) – od 400 MHz, vnější až 133 MHz, 9,5 mil. tranzistorů, dvě výrobní technologie (0,25 a 0,18  $\mu$ m – vyšší výkon, nižší spotřeba, 1,6 V místo 2 V), integrovaná 256kB L2 cache na čipu, 70 nových instrukcí SSE (Streaming SIMD Extensions) pro 3D
- **Pentium 4** (2000) – od 1,3 GHz, vnější 400 a 533 MHz (technologie DualBus – dvojice paměťových karet), 42 mil. tranzistorů, plošky, nové jádro, architektura NetBurst – vyšší frekvence, ale i spotřeba (potřeba zdroje ATX-P4 s přídavným konektorem), další cache (např. Execution Trace Cache pro dekódované makroinstrukce), dalších 144 instrukcí SSE2 pro plovoucí řádovou čárku, verze HT (HyperThreading) – zdvojené registry, simulace dvou procesorů
- **Itanium** (2001) – 1. plně 64-bitový, instrukční sada IA-64 (Itanium), pro servery a pracovní stanice
- **Core** (2006) – vícejádrové, SSE 3/4, VTx/d, sdílená až 2/6MB L2 cache, verze Solo, Duo (32-bitové), Core 2 (64-bitové), Duo, Quad (2× cache), Extreme, i3/5/7 (až 12 MB L3 cache, různé mikroarchitektury)

## 1.5.2 Vykonávání instrukcí

### Správa procesoru



#### Vykonávání instrukcí

- program = sekvence (binárních) kódů instrukcí, registrů procesoru a dat (čísla, texty, hodnoty adres do operační paměti a vstupně/výstupních zařízení)
- stejná (RISC) nebo proměnná (CISC) délka kódů instrukcí – 1 až 4? byty
- **operandy** = parametry instrukcí, registry a data, specifický počet (obvykle 0 až 2), přípustné kombinace pro každou instrukci
- výsledek instrukce často ukládán do prvního operandu
- vykonávání instrukce
  - trvá určitý počet **taktů/tiků** (na vnitřní frekvenci procesoru), jednotky až stovky
  - až 7 fází: např. načtení, dekódování, načtení operandů, provedení, uložení výsledku
  - **pipelining** – částečně paralelní provádění instrukcí (různých fází), nelze vždy, např. kvůli instrukcím podmíněných skoků (= implementace podmínek a cyklů v programu)

### Správa procesoru



#### Vykonávání instrukcí

- vykonávání instrukce
  - **superskalární architektura** – více duplikovaných částí procesoru, např. ALU, částečně paralelní provádění instrukcí (i stejných fází), použití i na podmíněné skoky (vykonávání obou větví, po skoku i bez skoku, současně, u neplatné se pak ukončí – tzv. speculative execution, nebo předvídaní podmínky skoku/správné větve – tzv. branch prediction) → instrukce může ve výsledku v průměru trvat pod 1 takt
  - **vícejádrové procesory a více procesorů** – (plně) paralelní provádění instrukcí, symetrické a nesymetrické architektury (hardware i OS)
- sekvenční pořadí vykonávání instrukcí tak, jak jsou v programu (viz von Neumannova architektura)
  - registr **EIP** – adresa následující instrukce, automatické zvětšování
  - **výjimky** = jiné změny EIP: instrukce skoků (na adresu), volání podprogramů (funkcí, procedur, metod objektů apod.) a obsluh přerušení + návrat na místo volání/přerušení

### 1.5.3 Zásobník

#### Zásobník

- procesor má vyčleněný úsek paměti pro zásobník (LIFO)  $\Rightarrow$  mezivýpočty, návratové adresy, lokální proměnné, ...
- vyšší prog. jazyky obvykle neumožňují přímou manipulaci se zásobníkem (přesto má zásadní úlohu)
- procesory i386 mají jeden zásobník, který roste shora dolů
- registr ESP ukazuje na vrchol zásobníku (`mov eax, [esp]` načte hodnotu na vrcholu zásobníku)
- uložení/odebrání hodnot pomocí operací:

```
PUSH r/m/i          ; ; sub esp, 4  
                      ; ; mov [esp], op1
```

```
POP r/m            ; ; mov op, [esp]  
                      ; ; add esp, 4
```

- registr ESP musí vždy obsahovat číslo, které je násobek čtyř

#### 1.5.4 Podprogramy

### Volání podprogramů/funkcí

- k volání podprogramu se používá operace CALL r/m/i  $\Rightarrow$  uloží na zásobník hodnotu registru IP a provede skok  
`push eip ; ; tato operace neexistuje  
jmp <addr>`
- k návratu z funkce se používá operace RET  $\Rightarrow$  odebere hodnotu ze zásobníku a provede skok na adresu danou touto hodnotou
- použití zásobníku umožňuje rekurzi

### Volání funkcí

- předání parametrů
- vytvoření lokálních proměnných
- provedení funkce
- odstranění informací ze zásobníku
- návrat z funkce, předání výsledku

## Konvence volání funkcí (1/2)

- způsob, jakým jsou předávány argumenty funkčím, jsou jen konvence (specifické pro překladač, i když často součástí specifikace ABI OS)
- předávání pomocí registrů (dohodnou se urč. registry), příp. zbývající argumenty se uloží na zásobník
- předávání argumentů čistě přes zásobník
- kdo odstraní předané argumenty ze zásobníku? (volaná funkce nebo volající?)
- Konvence C (cdecl)
  - argumenty jsou předané čistě přes zásobník
  - zprava doleva
  - argumenty ze zásobníku odstraňuje volající
  - umožňuje funkce s proměnlivým počtem parametrů
- Konvence Pascal (pascal)
  - argumenty jsou předané čistě přes zásobník
  - zleva doprava
  - argumenty ze zásobníku odstraňuje volaný
  - neumožňuje funkce s proměnlivým počtem parametrů

## Konvence volání funkcí (2/2)

- Konvence fastcall (fastcall, msfastcall)
  - první dva parametry jsou předány pomocí ECX, EDX
  - zbylé argumenty jsou na zásobníku zprava doleva
  - argumenty ze zásobníku odstraňuje volaný
  - mírně komplikuje funkce s proměnlivým počtem parametrů
  - pod tímto jménem mohou existovat různé konvence
- návratová hodnota se na i386 obvykle předává pomocí registru EAX, příp. EDX:EAX

### Rámec funkce (stack frame)

- při volání funkcí se na zásobníku vytváří tzv. rámec (stack frame)
- obsahuje předané argumenty, adresu návratu, příp. lokální proměnné
- k přístupu k tomuto rámcovi se používá registr EBP

## Volání funkce s koncem cdecl (1/4)

### Volání funkce

- ① na zásobník jsou uloženy parametry funkce zprava doleva (push <arg>)
- ② zavolá se funkce (call <adresa>), na zásobník se uloží adresa návratu
- ③ funkce uloží obsah registru EBP na zásobník (adresa předchozího rámce)
- ④ funkce uloží do registru EBP obsah ESP (začátek nového rámce)
- ⑤ vytvoří se na zásobníku místo pro lokální proměnné
- ⑥ na zásobník se uloží registry, které se budou měnit (push <reg>)

### Návrat z funkce

- ① obnovíme hodnoty registrů (které byly umístěny na zásobník pop <reg>)
- ② odstraníme lokální proměnné (lze k tomu použít obsah EBP)
- ③ obnovíme hodnotu EBP
- ④ provedeme návrat z funkce ret
- ⑤ odstraníme argumenty ze zásobníku (lze použít přičtení k ESP) [≡](#)

## Obsah zásobníku

```
...  
argument n  
...  
EBP + 12 --> argument 2  
EBP + 8 --> argument 1  
                návratová hodnota  
                původní EBP  
EBP - 4 --> první lokální proměnná  
EBP - 8 --> druhá lokální proměnná  
...  
ESP --> poslední lokální proměnná
```

## Volání funkce s konvencí cdecl (3/4)

### Volání funkce

```
push arg2      ;; druhý argument
push arg1      ;; první argument
call func
add esp, 8     ;; odstrani oba argumenty ze zasobniku
```

### Tělo funkce

```
push ebp
mov ebp, esp
sub esp, n      ;; vytvorí místo pro lokální promené
push ...
...
pop ...
mov esp, ebp    ;; odstrani lokální promené
pop ebp
ret
```



## Volání funkce s konvencí cdecl (4/4)

- první argument leží na adrese [ebp + 8], druhý na [ebp + 12], atd.
- první lokální proměnná na [ebp - 4], druhá na [ebp - 8], atd.

### Uchovávání registrů

- uchovávání všech použitých registrů na začátku každé funkce nemusí být efektivní
- používá se konvence, kdy se registry dělí na
  - *callee-saved* – o uchování hodnot se stará volaný (EBX, ESI, EDI)
  - *caller-saved* – o uchování hodnot se stará volající (EAX, ECX, EDX)
- po návratu z funkce mohou registry EAX, ECX a EDX obsahovat cokoliv

## 1.5.5 Přerušení

### Správa procesoru



#### Přerušení (Interrupt)

- původně pro řešení komunikace (rychlého) procesoru s (pomalými) I/O zařízeními, běžícími samostatně:
  - dříve procesor: vyslání požadavku na zařízení, **aktivní čekání na vyřízení** (= smyčka testující stav oznamující vyřízení), pokračování ve výpočtu
  - vyslání požadavku, pokračování ve výpočtu zatímco zařízení zpracovává požadavek, oznámení vyřízení požadavku = **přerušení procesoru**
  - např. procesor vyšle požadavek čtení sektoru z disku (dá požadavek s číslem sektoru na sběrnici) a pokračuje ve výpočtu, disk najde sektor, načež do své cache a vyvolá přerušení, procesor vyšle požadavek zaslání dat, disk pošle, procesor uloží do operační paměti, požadavek na další data atd.
- = pozastavení vykonávání programu, vykonání programu tzv. **(rutiny) obsluhy přerušení** implementované OS (např. ovladači zařízení), pokračování vykonávání programu
- během vykonávání obsluhy přerušení další přerušení zakázána nebo systém **priorit přerušení**

### Správa procesoru



#### Přerušení (Interrupt)

- **hardwareová**: přídavné karty (dříve), disková zařízení (dříve), vstupně/výstupní zařízení, HW časovač aj., 256 přerušení u Intel 80x86
- **softwarová** – vyvolána OS pro vlastní potřeby fungování nebo programy při volání služeb OS (tzv. **systémová volání**)

#### DMA (Direct Memory Access)

- = způsob přenosu dat mezi zařízením a operační pamětí přímo, bez řízení procesorem, pro větší množství dat, např. pro disková zařízení
- procesor pouze naprogramuje **řadič DMA** a vyšle první požadavek, zbytek řeší řadič

#### Mapování paměti

- ... zařízení do operační paměti, např. přídavné karty
- **přímý přístup** do paměti zařízení skrze přístup do operační paměti

## 1.6 Paměti počítače (RAM, cache, disk, diskové pole)

### 1.6.1 Operační paměť (RAM)

#### Paměti (memory)



= paměťové zařízení pro ukládání (binárních) dat

■ parametry:

- **kapacita, přenosová rychlosť, přístupová doba** (doba od požadavku do vydání dat), spolehlivost (doba mezi poruchami)
- energetická závislost (neudrží data bez el. napájení, volatilní), způsob přístupu (přímý, sekvenční od začátku paměti), druh (statické/dynamické – obsah potřeba periodicky obnovovat) a další
- cena za bit – v závislosti zejm. na kapacitě (nepřímo) a přenosové rychlosti (přímo)

#### Paměti (memory)



- **vnitřní** – pro krátkodobé ukládání kódu a dat spuštěných programů, např. registry procesoru, operační a vyrovnávací paměť, přídavné karty, a trvalé uložení firmware a základních konfiguračních dat počítače, např. BIOS, přídavné karty
  - menší kapacity (do desítek GB), vyšší přenosové rychlosti (až desítky TB/s), přístupová doba do desítek ns, spolehlivé, energ. závislé i nezávislé, přímý přístup, statické i dynamické
- **vnější** – pro dlouhodobé ukládání programů a (jiných) dat, např. pevné disky, výměnná média (CD/DVD, paměťové karty, aj.) a jiná disková zařízení
  - větší kapacity (až stovky TB), nižší přenosové rychlosti (do jednotek GB/s), přístupová doba ms (přímý přístup) až min. (sekvenční přístup), méně spolehlivé, energ. nezávislé, statické

#### Paměti (memory)



#### Vnitřní paměti

- zapojeny jako matice **paměťových buněk** s kapacitou 1 bit
- po bytech adresovány hodnotou **adresy** na adresní části sběrnice, dekódované (binárním) dekodérem, který vybere adresový vodič (nastaví log. I)
- na výstupu datové části sběrnice zesilovače
- typy:
  - **pouze pro čtení (ROM, Read Only Memory)** – energ. nezávislé, použití např. firmware (BIOS, karty)
  - **i pro zápis** – např. **RAM (Random Access Memory, s náhodným přístupem)**, energ. závislé, rychlejší než ROM, použití všude jinde než ROM

## Paměti (memory)

### Statická RAM (SRAM)

- rychlé (přístupová doba jednotky ns, rychlosť až desítky TB/s), ale složité → použití: cache paměti
- realizace buňky: **bistabilní klopný obvod**, např. v technologii MOS – 2 datové vodiče: *Data* pro zápis, *Data* pro čtení (negace uložené hodnoty)

Obrázek: Realizace buňky paměti SRAM v technologii MOS



## Paměti (memory)

### Dynamická RAM (DRAM)

- uchování log. hodnoty pomocí el. náboje – **kondenzátoru**, který se samovolně i čtením vybíjí ⇒ potřeba periodicky obnovovat (čipsetem) = **refresh**
- kondenzátor + refresh = větší přístupová doba (jednotky až desítky ns), menší rychlosť (až desítky GB/s), ale jednoduché → použití: operační paměti

Obrázek: Realizace buňky paměti DRAM v technologii TTL



## Paměti (memory)

### Dynamická RAM (DRAM)

- typy:
  - **FPM (Fast Page Mode)** – využívá se toho, že data jsou v souvislé oblasti paměti a přístup je pomocí stránkování paměti, 1. byte plným počtem taktů, následující menším, např. 5-3-3-3 = **časování paměti**, moduly DIP a SIPP
  - **EDO (Extended Data Output)** – částečné překrývání operací přístupu do paměti, časování např. 5-2-2-2, moduly SIMM
  - **SDRAM (Synchronous DRAM)** – synchronní s frekvencí procesorové sběrnice základní desky, časování typicky 5-1-1-1, moduly DIMM
  - **DDR SDRAM (Double Data Rate SDRAM)** – 2 datové přenosy v 1. cyklu (na vzestupné i sestupné hraně taktu), např. DDR 200–400 (100–200 MHz, PC1600–3200), DDR2 400–1066 (200–533 MHz, PC2-3200–8500), DDR3 800–2133 (400–1066 MHz, PC3-6400–17000)
  - **RDRAM (Rambus DRAM)** – speciální paměťová sběrnice (rychlejší – 800 MHz, užší), podpůrné čipy pro komunikaci s řadičem paměti v modulu RIMM, dnes nepoužívané

## Paměti (memory)



### Dynamická RAM (DRAM)

- moduly:
  - **DIP, SIPP** – do procesorů Intel 80286, v patici na základní desce, FPM
  - **SIMM (Single Inline Memory Module)** – čipy na podélné destičce plošného spoje = karta do slotu/**banky** na základní desce, první 30 pinů, 8/16-bitové, 256 kB až 4 MB, FPM, pro Intel 80286 až 80486, další 72 pinů, 32-bitové, 4–64 MB, EDO
  - **DIMM (Dual IMM)** – 168/184/240 pinů, 64-bitové, 16 MB až 4 GB, SO-DIMM a MICRODIMM pro notebooky (72/144/200/204 a 144/172/214 pinů), (DDR) SDRAM
  - **RIMM (Rambus IMM)** – 184/232/326 pinů, 16–64-bitové, 64 MB až 1 GB, SO-RIMM (160 pinů), RDRAM

Obrázek: Moduly a sloty pro operační paměti

- výrobci: Transcend, Corsair Memory, Kingmax, Kingston Technology, Samsung Electronics a další

### CMOS RAM (Complementary Metal Oxide Silicon)

- nepatrna klidová spotřeba, zálohované baterií na základní desce
- použití: konfigurační data BIOSu

#### 1.6.2 Cache

## Paměti (memory)



### Paměť cache

- = **vyrovnávací paměť** mezi různě rychlými zařízeními (na frekvenci rychlejšího), např. procesorem a operační pamětí, pamětí a diskovým zařízením apod.
- rychlejší zařízení používá paměť cache, do které se načtou data z pomalejšího zařízení, celá oblast/blok obsahující požadovaná data
- v případě operační paměti využití toho, že programy používají (souvislé) oblasti paměti, pro instrukce i data
- **cache hit** = požadovaná data jsou v cache
- **cache miss** = požadovaná data nejsou v cache a musí se načíst z pomalejšího zařízení
- organizace do bloků, při přeplnění vyřazní bloků **algoritmem LRU (Least Recently Used)** – nejdéle nepoužívaný

## Paměti (memory)



### Paměť cache

- = **asociativní paměť** = k datům se přistupuje na základě klíče = celá nebo část adresy
  - tabulky se sloupci klíč (tag), data a další informace (platnost dat, pro LRU, synchronizační u více cache pamětí aj.)
  - plně asociativní – klíč je celá adresa, jedna tabulka ⇒ velký klíč, hodně porovnání (bin. komparátor) → nepoužívají se
  - $n$ -cestně asociativní – klíč je část adresy, zbytek tzv. třída určující řádek v  $n$  tabulkách (bin. dekodér) ⇒ menší klíč, méně ( $n$ ) porovnání → nejpoužívanější, např.  $n = 4$
  - přímo mapovaná – 1-cestně asociativní, pomalejší, moc se nepoužívá

Obrázek: Schéma cache paměti

## Paměti (memory)



### Paměť cache

- zápis dat přes cache do pomalejšího zařízení:
  - **write-through** – ihned při zápisu do cache
  - **write-back** – až později, např. při přeplnění cache, vyšší výkon

### L1 (primární, first level cache)

- mezi procesorem a procesorovou sběrnicí, dnes zvlášť instrukční a datová (Harvardská koncepce)
- 8–32 kB, součást procesoru, od procesoru Intel 80486 (4-cestně asociativní)

### L2 (sekundární, second level cache)

- mezi procesorovou sběrnicí a operační pamětí, pro urychlení práce s pamětí
- 32 kB až 6/1 MB (externí/interní), na základní desce (dříve) i jako součást procesoru (pouzdro, čip), od procesoru Intel 80386
- dnes i L3 cache – až 12 MB, součást procesoru, od Intel Core

### 1.6.3 Disk

#### Pevný disk (hard disk drive, HDD)



Obrázek: Pevný disk

- = vnější paměťové zařízení pro dlouhodobé ukládání dat
- **magnetický způsob zápisu/čtení dat:** zmagnetování povrchové vrstvy (aktivní plochy) nemagnetického kotouče (disku) pomocí čtecí/záznamové hlavy – změna magnetického toku = **impuls**, čtení zmagnetováním hlavy
- součásti:
  - rotující soustředěné **keramické kotouče** nad sebou – 1–5, rotují stejnou rychlostí (5,4, 7,2, 10, 15 tis. otáček/min), 2 aktivní plochy (horní a dolní strana kotouče), velikosti 3,5", 2,5" (pro přenosné počítače)
  - čtecí/záznamové **hlavy** vystavované nad akt. plochami – 2× počet kotoučů, vystavování pomocí krokových motorků (dřívě) nebo vystavovacích cívek, vzdálenost v jednotkách  $\mu\text{m}$  od kotouče udržována aerodynamickým vztlakem
  - elektronika – pro ovládání vystavování hlav a rotace kotoučů

#### Pevný disk (hard disk drive, HDD)



##### ■ **geometrie:**

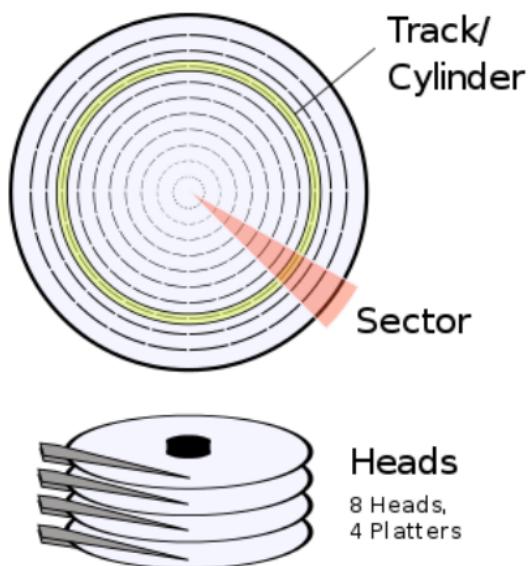
- **stopy** = soustředné kružnice na akt. ploše, číslování od 0 od vnějšího okraje
- **sektory** = části stopy, číslování od 1, konstantní datová velikost typicky 512 B nebo 4 kB = jednotka uložení dat, stejný počet pro všechny stopy nebo stejná velikost, tzv. zonální zápis (zone bit recording, ZBR, na vnějších stopách více sektorů)
- **cylindry** = množiny stop na akt. plochách nad sebou
- vytvoření tzv. *formátováním disku* – před prvním použitím

Obrázek: Ilustrace geometrie pevného disku

##### ■ čtení/zápis – přímý přístup:

- 1 vystavení hlav nad stopy / do cylindru – data zapisována po cylindrech, kvůli paralelnímu čtení/zápisu všemi hlavami
- 2 pootočení kotouče tak, aby byly sektory pod hlavami – sektory logicky za sebou ne fyzicky za sebou, ale prokládaně každý  $n$ -tý (**faktor prokládání**  $1 : n$ ), kvůli vyšší rychlosti rotaci disku než čtení/zápisu
- 3 vlastní čtení/zápis

## Pevný disk (hard disk drive, HDD)



## Pevný disk (hard disk drive, HDD)



- kapacita – desítky MB až jednotky TB, dána hustotou záznamu a způsobem **kódování binárních dat**:
  - impuls (**I**) = log. **I?** – ne, delší posloupnot **0** = delší místo bez impulsů (**N**)  $\Rightarrow$  ztráta synchronizace čtených dat a řadiče + kolik **0??**
  - maximalizace využití kapacity – krátká místa bez impulsů vs. málo impulsů (málo časté změny mag. toku)
  - **modulace FM** (**0** = IN, **I** = II, příliš impulsů, nepoužívá se), **MFM** (**0** = IN, jestliže předchozí je 0, jinak NN, **I** = NI, max 3 N za sebou, o 20 % úspornější než FM, dříve a u disket, viz dále), **2,7 RLL** (kódování dvojic až čtveřic bitů, 2–7 N za sebou, o 50 % úspornější než MFM, starší disky), dnes **A/ERLL**
- přenosová rychlosť – do stovek MB/s
- přístupová doba – jednotky ms (po roztočení), doba vystavení hlav (**seek time**) + doba pootočení kotoučů (**rotary latency period**)
- **parkování hlav** = umístění hlav nad/za (nejčastěji) nejvnitřnější stopu plochy při zastavování rotace kotoučů a vymizení aerodynamického vztluaku, softwarově (dřívě) nebo automaticky
- všechna rozhraní pro disková zařízení

#### 1.6.4 Disková pole

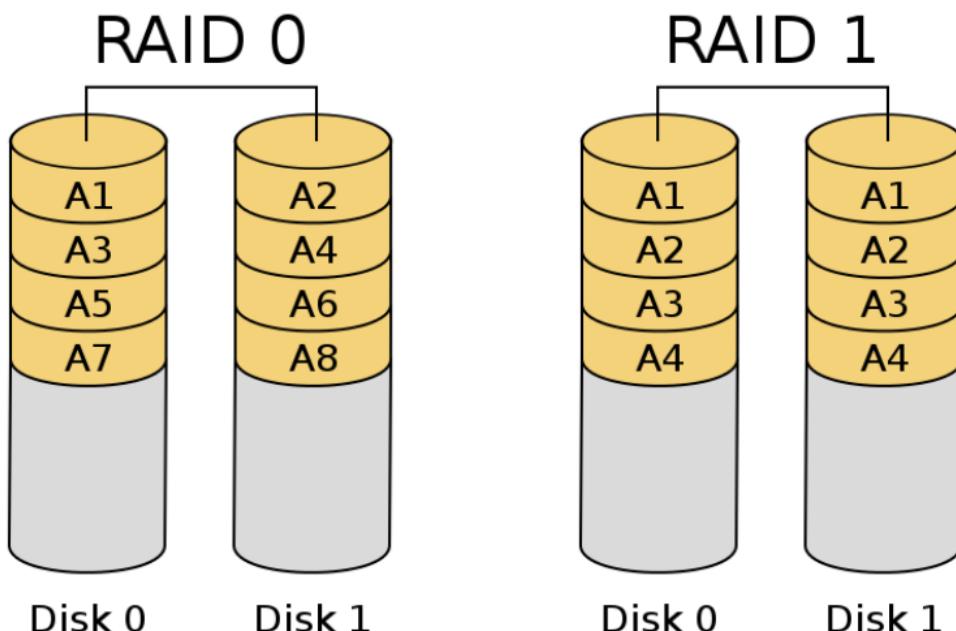
##### Diskové pole (disk array)



- = soustava pevných disků (tvářících se jako jeden) pro **redundantní ukládání dat za účelem zvýšení spolehlivosti uložení dat (chyby, odolnost)** nebo výkonu práce s daty
- ~ **RAID (Redundant Array of Independent/Inexpensive Disks)**, 1988 – diskový řadič, hardwarový v podobě přídavné karty (příp. integrované, s cache pamětí) nebo i softwarový
- **RAID 0** – zřetězení min. 2 disků do jednoho celku (JBOD) nebo prokládané uložení bloků dat na disky (**stripping**) – zvýšení výkonu (o cca 50 %) paralelním čtením z více disků, ale ne spolehlivosti
- **RAID 1 (zrcadlení)** – kopie dat na 2 disky – zvýšení spolehlivosti opravením z druhého disku, i výkonu čtením z disků zároveň, pomalejší zápis, poloviční kapacita, varianty 0+1, 1+0, 1+0+0
- **RAID 2** – složitější RAID 3, bitové prokládání s Hammingovým samoopravným kódem na dalších discích, pomalé
- **RAID 3** – min. 3 disky, bitové prokládání s **paritou (XOR)** na samostatném disku, chyba lib. jednoho disku opravena z ostatních a paritního, paritní disk úzké místo (nejvyužívanější), varianty 0+3, 3+0

---

##### Diskové pole (disk array)



## Diskové pole (disk array)



- **RAID 4** – jako RAID 3, ale blokové prokládání, parita po blocích
- **RAID 5** – jako RAID 3, ale paritní data střídavě na všech discích, chyba lib. jednoho disku opravena z ostatních, pomalejší zápis, varianty 5+0, 5+1, 5+3
- **RAID 6** – jako RAID 5, ale dvoje různá paritní data, opravení chyby až dvou disků, varianta 6+0
- **RAID 7** – odvozené od RAID 3 a 4, vyrovnávací paměť, vlastní sběrnice, opravena chyba i více disků (i sudý počet chyb)

## 1.7 Přídavné karty PC, datové mechaniky a média (CD, DVD, paměťové karty), periferie.

### 1.7.1 Přídavné karty

#### Přídavné karty



- = obdélníkové plošné spoje s konektorem pro zasunutí do slotu na základní desce
- konektory (na kraji karty) pro připojení displejů a periferií, vyvedené ven ze zadní části skříně
- konektory (na kartě) pro připojení jiných vnitřních součástí počítače: jiné karty, disková zařízení (pevné disky, mechaniky výměnných médií), zdroj
- grafická, zvuková, síťová, multimediální, diskové řadiče, pro periferie a další
- **integrované na základní desce** – součást čipsetu, dnes běžně zvuková, síťová, diskové řadiče, někdy i grafická (tzv. all-in-one)
- **integrované do procesoru** – v pouzdře s CPU, dnes grafická, někdy i síťová
- **pro vnější sběrnice** – USB, PCMCIA, ExpressCard aj.

#### Grafická karta (graphic card)



Obrázek: Grafická karta

- ~ grafický adaptér, videokarta, grafický akcelerátor (dříve zvlášť)
- = zařízení zprostředkovávající obrazový výstup počítače na displeji
- součásti (čipy na kartě):
  - **grafické procesory (GPU)** – vytváří obraz ze vstupních dat v grafické paměti, implementuje grafické operace (vykreslení graf. tvaru, vyplnění oblasti barvou, texturou, 3D grafika – **OpenGL**, stínování atd.), dekódování videoformátů (MPEG)
  - **grafická paměť** – pro uložení obrazových a dalších dat pro tvorbu obrazu, může být mapovaná do operační paměti, propojená s graf. procesorem 32–512-bitovou sběrnicí, EDO, VRAM (Video RAM), SGRAM (Synchronous Graphic RAM), SD/DDR RAM, **GDDR** (Graphics DDR)
  - převodník obrazových dat na výstupní videosignál – např. (RAM)DAC (Digital Analog Convertor)
  - další – např. Video BIOS

## Grafická karta (graphic card)

- slot/sběrnice dnes PCI Express 16×, popř. AGP
- konektory (ven) pro displeje: VGA = D-SUB, S-Video (analogové), DVI, HDMI, DisplayPort (digitální)
- konektory (na kartě): pro jiné (grafické, multimediální) karty, zdroj aj.
- výrobci: AMD/ATI, Nvidia, Intel, Matrox, VIA Technologies/S3, SiS, a další

### Režimy zobrazení

- **textový** – zobrazení (tisknutelných) znaků textu do 2D tabulky, typicky alfanumerických a speciálních (např. interpunkce, grafických) – předdefinované v BIOSu na kartě, obecně libovolných definovaných pomocí znakové 2D matic (mřížky) bodů
- **grafický** – zobrazení libovolného obrazu do 2D mřížky (rastru) obrazových bodů, tzv. **pixelů**

## Grafická karta (graphic card)

### Režimy zobrazení

- parametry režimu:
  - **rozlišení** – počet znaků/pixelů na řádku výstupu (horizontální) a ve sloupci výstupu (vertikální), např. pro textové  $80 \times 25$  (výchozí),  $80 \times 50$ ,  $40 \times 25$ , ..., pro grafické  $320 \times 200$ ,  $640 \times 480$  (VGA),  $800 \times 600$ ,  $1024 \times 768$ ,  $1280 \times 1024$ ,  $1280 \times 800$ ,  $1600 \times 1200$ ,  $1920 \times 1200$ , ... (VESA), typicky poměr stran  $4 : 3$ ,  $16 : 9 / 10$
  - **obnovovací frekvence (refresh frequency/rate)** – frekvence překreslování snímků za jednotku času, 50–160 Hz, (nepřímo) závisí na rozlišení
  - **barevná hloubka (počet barev)** – počet barev, které je možné celkem (ne nutně zároveň) zobrazit, např. pro textové 2 (monochromatický, "černobílý"), 16, pro grafické 16, 256, 64k (high color), 16,7 mil. (true color)
  - další – např. rozměr znakové matic (např.  $8/9 \times 14$ ), podporované barevné modely (např. Red-Green-Blue, Cyan-Magenta-Yellow-blacK)

## Grafická karta (graphic card)



Parametry karty:

- množina režimů zobrazení – většinou se udává maximální grafický, v závislosti na
- frekvence a počet jader grafického procesoru, velikost grafické paměti 256 kB až 1 GB
- parametry výstupů – zejm. analogového (obnovovací frekvence)
- parametry OpenGL – např. počty vestavěných programů na úpravu zobrazované scény (tzv. **shader**, vertex, geometry, pixel aj.)

Typy grafických karet:

- **MDA (Monochrome Display Adapter)** – IBM, 1981, 1. pro PC, jen textový režim  $80 \times 25$  s ASCII znaky, digitální výstup
- **CGA (Color Graphics Adapter)** – IBM, barevné textové  $\times 16$  i grafické režimy  $320 \times 200 \times 4/16$ ,  $640 \times 200 \times 2$
- **EGA (Enhanced Graphics Adapter)** – IBM, 1984, 1. všeobecně využitelná, až  $640 \times 350 \times 16/64$ , později výkonnější PGA (Professional GA)

## Grafická karta (graphic card)



Typy grafických karet:

- **VGA (Video Graphics Array)** – IBM, 1987, původně pro IBM PS/2, rychle de facto standard, až  $640 \times 480 \times 16$  (standardní VGA), analogový výstup (RGB), varianta XGA (eXtended Graphics Adapter, až  $1024 \times 768 \times 256$ ), **VGA BIOS** – součást BIOSu
- **SVGA (Super VGA)** – 1. s grafickým procesorem (grafický akcelerátor), standard **VESA** (Video Electronics Standard Association) pro rozlišení nad  $640 \times 480 \times 16$ , rozšíření BIOSu **VBE**
  - dnes více grafických procesorů **GPU** (až stovky), složitá 3D grafika (operace stínování, textury, mapování, buffering, filtrování, vyhlazování, renderování atd. = výpočet zobrazované scény, jednotky pixel a vrchol (vertex)), obecné paralelní výpočty (**GPGPU** – CUDA, OpenCL)
  - integrace do pouzdra s CPU, CPU + GPU = **APU** (Accelerated PU)

## Zvuková karta (sound card)



Obrázek: Zvuková karta a PC speaker

- = zařízení zprostředkovávající zvukový výstup počítače z reproduktorů a zpracování zvuku na vstupu
- **PC speaker** – malý reproduktor ve skříni připojený k základní desce, typicky pro jednoduché zvuky (např. varovné pípání při chybě), ale i primitivní hudbu
- součásti (čipy na kartě): **AD-DA převodníky** (pro digitalizaci analogového signálu a obráceně), **FM a wave table syntezátory**, paměti pro vzorky tónů nástrojů pro wave table syntézu, **efektový procesor** (pro úpravy zvuku v reálném čase, vytváření efektů, např. prostorového zvuku apod.) a další
- slot/sběrnice dnes PCI, popř. PCI Express, USB

## Zvuková karta (sound card)



- konektory (ven, barevně odlišené): reproduktorský výstup (dříve, zesílený pro pasivní reproduktory), linkový vstup a výstup, mikrofonní vstup, výstupy pro středový, LFE/subwoofer, zadní, boční aj. reproduktory – analogové stereo jack 3,5 mm, S/PDIF digitální linkový výstup (optický), rozhraní MIDI pro elektronické hudební nástroje nebo joystick (**MIDI/Gameport**) – 15-pinový konektor DA15
- konektory (na kartě): CD-Audio pro propojení s optickou mechanikou (pro přehrávání Audio CD), IDE/PATA (dříve), patice pro paměťové moduly pro vzorky tónů nástrojů pro wave table syntézu
- výrobci: Creative Technology, C-Media, VIA Technologies a další

## Zvuková karta (sound card)



### Digitální záznam analogového signálu zvuku:

- analogový signál – spojité reprezentované vlnění např. pomocí úrovně napětí
- digitalizace v AD převodníku pomocí **vzorkování (sampling)** = v periodických časových intervalech zaznamenána okamžitá úroveň signálu (vzorek) – PCM (Pulse Code Modulation)
- **Nyquist-Shanonova věta:** signál spojitý v čase je plně určen posloupností vzorků zaznamenaných ve stejných periodických intervalech, je-li jejich frekvence větší než dvojnásobek nejvyšší frekvence signálu
- parametry: **vzorkovací frekvence** (11,025, 22,05, 44,1, 48, 96 kHz), počet rozlišitelných úrovní signálu (8, 16, 24 bitů)
- kvality: telefonní (11 kHz, 8bit, mono), rádiová (22 kHz, 8bit, mono), CD (44 kHz, 16bit, stereo)
- ztrátové komprese – např. MPEG (MP3,4), OGG Vorbis aj.

## Zvuková karta (sound card)



### MIDI (Musical Instrument Digital Interface)

- = standard pro (digitální) komunikaci, kontrolu a synchronizaci elektronických hudebních nástrojů (včetně počítače)
- ne zvukový signál, ale informace o druhu nástroje, výšce, délce, intenzitě tónu, tempu tónů atd.
- nástroj, např. zvuková karta, musí vytvářet zvukový signál (tóny nástroje):
  - **FM syntéza (frequency modulation)** – složení sinusových vlnění plus aplikování efektů, emulace tónů nástroje, realizovaná FM syntezátorem (čip OPL 2, 3, 4)
  - **wave table syntéza** – úprava digitalizovaných vzorků tónů skutečných nástrojů, uložených v paměti (ROM, RAM)
  - parametry: použitý FM syntezátor, velikost paměti pro vzorky, nahrané vzorky

## Síťová karta (network card)



Obrázek: Síťová karta

- = zařízení připojující počítač do (lokální) **počítačové sítě**
- slot/sběrnice dnes PCI, PCI Express, USB, ExpressCard
- konektory (ven): různé pro různá přenosová média (kably nebo rádiové vlny skrze antény) používané v dané síti (**Ethernet**, **Wi-Fi**, optické, ...), např. BNC, **RJ-45**, Wi-Fi, optické aj.
- parametry: typ a rychlosť sítě, (konfigurační) parametry sítě, hardwarová podpora zpracování síťových dat, probuzení počítače ze sítě (wake on LAN) a Boot ROM
- **Boot ROM** – paměť (EEPROM, Flash) obsahující program pro zavedení operačního systému ze sítě
- výrobci: 3Com, Cisco, Edimax, Intel, Linksys, Atheros a další

## Další karty



### Rádiové a televizní karty (radio and TV card)

- = zařízení pro příjem, popř. záznam, rádiového a televizního signálu
- televizní většinou obsahuje i rádiovou, podpora teletextu
- součásti: AD převodník (u analogových pro digitalizaci analogového signálu), **dekodér**, **enkování** pro záznam
- slot/sběrnice dnes PCI, USB
- konektory (ven): pro antény (koaxiální), S-Video, kompozitní audio/video, komponentní video, HDMI, dálkové ovládání aj.
- konektory (na kartě): pro propojení s grafickou kartou (dříve, dnes pomocí sběrnice)
- parametry: typ – analogové (FM rádio), digitální (pozemní, kabelové, satelitní), analogové TV normy (PAL, SECAM), digitální multiplexy, aj.

## Další karty

- multimedialní karty pro **zpracování videa** v reálném čase: střihové, enkódovací, atd.
- **řadiče diskových zařízení a polí (RAID), SSD disky**
- **modemové** – pro připojení k počítačové sítí skrze telefonní síť, slot AMR/CNR (dříve)
- „sběrnicové“ (**adaptéry**) – poskytující další sloty vnitřních (např. PCI, PCI Express), i konektory/porty vnějších (např. USB, FireWire, SATA, sériová, paralelní) sběrnic
- ... a další
- slot/sběrnice dnes PCI, PCI Express, PCMCIA, ExpressCard

### 1.7.2 Optické disky

## Optické disky + mechaniky



Obrázek: Optický disk a mechanika

- = výměnné paměťové médium pro dlouhodobé ukládání dat
- **optický způsob zápisu/čtení dat**: vytvoření prohlubní (tvz. **pitů**, 100 nm hluboké, 500 nm široké, 850–3000 nm dlouhé) v povrchové vrstvě záznamové plochy disku lisováním nebo vypálením laserem, čtení snímáním laseru různě (pit, bez pitu) odraženého od odrazové vrstvy záznamové plochy
- = v mechanice rotující **polykarbonátový kotouč** (záznamová plocha, odrazová vrstva kov) – rotace 500 (vnitřní) až 200 (vnější okraj) otáček/min, 1 nebo více vrstev pro záznam (oboustranné a vícevrstvé kotouče – polopruhledné vrstvy), velikosti **12 a 8 cm** (plus CD seříznutá do tvaru vizitky), tloušťka 1,2 mm

## Optické disky + mechaniky

- geometrie:
  - stopy = **spirály** od vnitřního okraje kotouče za sebou v záznamové ploše, číslování od 1
  - sektory (tzv. velké rámce) = části stopy, konstantní datová velikost typicky 2048 B (CD/DVD, Mode 1, 2352 Audio) = jednotka uložení dat, rozděleny na tzv. malé rámce (98 pro CD)
- čtení/zápis – přímý/postupný přístup:
  - **lisování** (zápis) – vyražení pitů celého disku podle šablony
  - **vypalování** – laserem ze záznamové hlavy (stovky °C), postupný zápis stop tzv. multisession
- **kódování binárních dat**: střídání pitů a ploch bez pitu = log. I, 1 B dat do 14 bitů včetně samoopravných kódů (Reed-Solomonovy)
- přístupová doba cca 100 ms (po roztočení)
- mechanika – čtecí/zapisovací hlava + elektronika, elektronicky ovládané vysouvání disku, rotace s konstantní úhlovou (CAV) nebo obvodovou (CLV) rychlostí, rychlosti uváděné pro zápis/přepis/čtení
- rozhraní EIDE/ATAPI, SATA, SCSI, USB

## Optické disky + mechaniky

### CD (Compact Disc)

- 1979, Sony, Philips
- rozestup stop  $1,6 \mu\text{m}$ , laser o vlnové délce 785 nm
- kapacity: 184–210 MB = 21-24 min. (8 cm), 550 MB = 63 min. (starší), **656 MB = 74 min.** zvuku, **702 MB = 80 min.** aj.
- přenosová rychlosť – udávána jako násobek přenosové rychlosti **150 kB/s** pro Audio CD, 1 – 52x
- formáty (standardy označovány jako **barevné knihy**):
  - **Audio CD** (červená) – záznam navzorkovaného zvuku v CD kvalitě (CDDA), typicky hudby
  - **CD-ROM** (žlutá) – po zápisu lisováním pouze pro čtení
  - **CD-R** (Recordable, oranžová) – po zápisu pouze pro čtení, **CD-RW** (ReWritable) – přepisovatelné jako celek nebo tzv. paketový zápis, cca 1000 přepisů, vrstva v amorfni nebo krystalické struktuře
  - **Video CD** (VCD, bílá) – spec. adresářová struktura (a záznam dat) pro video ve formátu MPEG-1 (rozlišení  $352 \times 288$  PAL/SECAM) se zvukem ve formátu MP2
  - další: SACD (šarlatová), PhotoCD (béžová), CD-I (Interactive, zelená), Enhanced CD/CD Plus/CD-G (modrá)

## Optické disky + mechaniky

### DVD (Digital Versatile/Video Disc)

- 1996/7, DVD Fórum (DVD-R(W)), DVD Alliance (DVD+R(W)) – do určité míry kompatibilní, nástupce CD
- jednostranné (DVD-5,9), **oboustranné** (DVD-10,14,18) a **dvouvrstvé** disky (na jedné straně DVD-9,14, na obou stranách DVD-18)
- rozestup stop  $0,74 \mu m$ , laser o vlnové délce 660 nm
- kapacity: **4,7/4,4 GB/GiB** (DVD-5, DVD-RW/RAM), **8,5/7,9 GB/GiB** (DVD-9)
- přenosová rychlosť – udávána jako násobek **1385 kB/s**, 1 – 24×
- typy:
  - **DVD-Video** – spec. adresářová struktura pro video ve formátu MPEG-2 (rozlišení  $720 \times 576$  PAL/SECAM) se zvukem ve formátu MP3 nebo Dolby Digital (AC3), 5.1, **interaktivita** (DVD menu, zvukové stopy, kapitoly, pohledy, titulky), **šifrování CSS** a regiony, analogové kódování **Macrovision**
  - **DVD-Audio** – spec. adresářová struktura pro zvuk v CD a lepší kvalitě (AC3, prostorové, DTS, vzorkování až 192 kHz), podobné SACD
  - **DVD-Data** – lib. data

## Optické disky + mechaniky

### DVD (Digital Versatile/Video Disc)

- média: **DVD-ROM**, **DVD-R/RW** (kompatibilní s DVD-ROM), **DVD+R/RW**, **DVD+R DL (Dual Layer)**, **DVD-RAM** (přímý zápis podobně jako např. pevný disk, desítky až stovky tis. přepisů, verze v pouzdře)
- **EcoDisc** – poloviční toušťka, nekompatibilní se štěrbinovými mechanikami
- mechaniky čtou/zapisují i CD

### 1.7.3 Paměťové karty

#### Paměťové karty

= integrované obvody na plošném spoji na plastové destičce

- kapacity do desítek GB
- nejdříve rozhraní PCMCIA, pak **vlastní různá rozhraní/konektory**
- druhy: **SmartMedia** (SM, první), **CompactFlash** (CF, I, II), **Memory Stick** (MS, Sony, Duo), **Secure Digital** (SD, mini, micro), **Multimedia Card** (MMC), **XD Picture Card** (Olympus) a další
- použití i mimo počítače: PDA, fotoaparáty, kamery, mobilní telefony, audio přehrávače apod.

#### 1.7.4 Periferie

### Displej (display, monitor)



- = výstupní zobrazovací zařízení pro zobrazení výstupu počítače od grafické (popř. televizní, multimedialní) karty
- zobrazení obrazových bodů od grafické karty (textového nebo grafického režimu) na obrazovce

**CRT (Cathode Ray Tube)** – “klasický monitor”

Obrázek: CRT monitor

- 1897, K. F. Braun
- obrazové body zobrazovány (s tzv. dosvitem) po řádcích z levého horního do pravého dolního, pak zatemnění, opakování **obnovovací frekvencí (refresh frequency)**

### Displej (display, monitor)



**CRT (Cathode Ray Tube)** – “klasický monitor”

- obrazovka na bázi **katodové trubice (elektronky)**: elektronové svazky emitované katodovou trubicí (1 pro monochromatickou, 3 pro barevnou – složky RGB) zaostřovány a vychylovány pomocí zaostřovacích a vychylovacích elektromagnetických cívek, urychlovány a modulovány pomocí mřížek (anody, jako stěna celé baňky, napětí v řádu kV) a filtrovány **maskou** (u barevných, mřížka pro zaostření svazku, stejně nabité elektrony se odpuzují = rozostření) na vrstvy luminoforů na zadní straně fosforeskujícího **stínítka** měnících energii elektronů na barvu složky RGB, celé ve vakuové baňce
- typy masek: delta, stérbinová (in-line), trinitron

Obrázek: Ilustrace principu fungování CRT obrazovky

- rušení mag. polem, elektromagnetické záření

## Displej (display, monitor)

### LCD (Liquid Crystal Display) – "displej"

- plochá tenká obrazovka na bázi **tekutých krystalů**: buňky/pixely ( $5\text{--}7 \mu\text{m}$ ) s látkou s tekutou krystalickou strukturou otáčející bez el. pole polarizační rovinu procházejícího světla až o  $90^\circ$  (tzv. twisted nematic, **TN effect**), před a za krystaly **elektrody** (přední určující zobrazené tvary, nebo tvořící mřížku, el. pole mezi nimi) a **polarizační filtry** vzájemně otočené o  $90^\circ$ , vše průhledné, vzadu odrazivá plocha nebo zdroj světla; trojice buněk/subpixelů pro barevné složky RGB s barevným filtrem

## Displej (display, monitor)

### LCD (Liquid Crystal Display) – "displej"

- obrazový bod  $\sim$  buňka, **nativní rozlišení** – max., určeno počtem elektrod v mřížce (řádky, sloupce), ostatní (menší) rozlišení emulovány (pixel více body/buňkami), subpixel rendering při znalosti organizace subpixelů
- nižší obnovovací frekvence než u CRT (ale "nebliká"), potřeba látna s přesnou dobou (setrvačností) otočení polarizační roviny světla (jinak blikání nebo "duchové" při rychlých změnách obrazu), malý kontrast a jas (u **pasivních displejů, STN**), malý pozorovací úhel  $\rightarrow$  kalkulačky, hodinky apod.
- vylepšení: více buněk pro obrazový bod, **podsvícení** – kvůli absorpci světla (buňkami a barevnými filtry) vzadu elektroluminiscenční fólie nebo fluorescenční zářivky (zbarvující světlo), **aktivní displej** – krystaly s nízkou setrvačností v buňce zálohované kondenzátorem (s tranzistorem, podobně jako RAM paměti, **TFT**, Thin Film Tranzistor), typy TN, IPS, MVA, PVA, S-PVA, S-IPS aj.
- použití: zejména přenosné počítače, tablety, mobilní telefony a jiná zařízení, dnes i periferní displej místo CRT

## Displej (display, monitor)

Další:

- **plazmový displej** – jako LCD, buňka s el. nabitým ionizovaným inertním plynem a luminofory, místo polarizačních filtrů jen sklo, elektrody v dielektriku, velký jas a kontrast, obnovovací frekvence jako u CRT
- **\*LED displeje** – (organické, OLED) elektroluminiscenční diody
- **video/datový projektor** – projekce obrazu, CRT, LCD, DLP, LED aj.
- ...

## Displej (display, monitor)

Konektory vstupů od grafické karty:

- oddělené **analogové RGB** + synchronizační – dříve, BNC konektory
- **VGA = D-SUB** – analogový (+ synchronizační), 15 pinů DE-15, typy VGA15, Mini-VGA, rozlišení do  $2048 \times 1536$  (QXGA), kanál **DDC(2)** pro specifikaci displeje (formát dat VESA EDID)
- **DVI (Digital Visual Interface)** – kombinovaný (nekomprimovaný) digitální a analogový, 3 typy: **DVI-D** (jen digitální), **DVI-A** (jen analogový), **DVI-I** (obojí), rozšíření **M1-DA** (obojí + USB), varianty mini/micro-DVI, DDC(2), druhý spoj (link) pro obraz s vysokým rozlišením (více než  $1920 \times 1200$ , WUXGA), i pro A-V techniku
- **HDMI (High-Definition Multimedia Interface)** – (nekomprimovaný) digitální, obraz, **zvuk** (až 8 kanálů) a signály dálkového ovládání, popř. i síť (Ethernet), volitelně šifrování HDCP, rozlišení do  $2560 \times 1600$  (WQXGA, single link, standard speed kabel),  $3840 \times 2400$  (WUXGA, dual link, hight speed kabel), typy: **A**, **B** (zpětně kompatibilní s DVI-D single, dual link) a **C** (mini, redukce na A), **D** (micro), i pro A-V techniku, zpoplatněný



## Displej (display, monitor)

Parametry:

- technické: typ (CRT, LDC), konektory vstupů, spotřeba (LCD až  $3\times$  méně než CRT), rozměry, váha, atd.
- velikost/**úhlopříčka** – 10" až 40"
- **rozlišení** textových a grafických režimů – grafické přes  $1920 \times 1200$  (FullHD) neomezeně, poměr stran 4 : 3, 16 : 9/10
- **obnovovací frekvence** – 50–160 Hz (CRT, optimální 85–100), 50–70 Hz (LCD, nepodstatné), udává graf. karta – při vyšší než podporované displejem více obrazů přes sebe (dříve) nebo černá obrazovka s hláškou "frequency out of range"
- **doba odezvy** – čas mezi rozsvícením a zhasnutím bodu/buňky, jednotky ms (LCD), pod ms (CRT)
- **pozorovací úhel** – max. úhel mezi směry, ze kterých je obraz pozorovatelný, změna barvy, jasu, kontrastu atd., širší u CRT než LCD
- max. **jas, kontrast** (CRT až 20 000:1, LCD do 2 000:1), podporované barvy atd. (lepší u CRT než LCD, "černější černá")
- ergonomické normy – MPR (dříve), dnes TCO
- další: OSD (On Screen Display) menu, vestavěné nebo přídavné reproduktory a webová kamera, USB HUB aj.

## Klávesnice (keyboard)



Obrázek: Klávesnice

- = (základní) vstupní zařízení pro ovládání počítače zadáváním znaků textu a příkazů
- odvozena od klávesnice psacího stroje nebo dálnopisu
- **abecední + ovládací klávesy:** pro znaky anglické/národní abecedy, číslic, interpunkce a některých symbolů + Enter/Return, (horizontální) tabulátor, Backspace, Escape, přepínače (Shift, Ctrl, Alt/Meta, CapsLock, „Win“, „Menu“ aj.)
- **funkční klávesy:** F1 až F12, význam závisí na programu
- **kurzorové a numerické klávesy:** šipky, Insert, Delete, Home, End, Page Up/Down a pro znaky číslic, symbolů desetinné tečky/čárky a aritmetických operací (+, -, ×, /), Enter/Return, přepínač NumLock
- **další klávesy:** PrintScreen/SysRq, ScrollLock, Pause/Break
- **speciální klávesy** (a klávesnice): multimedialní (hlasitost), pro aplikace webový prohlížeč, poštovní klient, kancelářské a další

## Klávesnice (keyboard)



- **rozložení kláves:** národnostní – americké (US/QWERTY), české (QWERTY, QWERTZ), aj., specializované (programátorské, účetní, ergonomické atd.), norma ISO/IEC 9995
- **XT** – vestavěný řadič, 83 kláves (abecední + ovládací, funkční F1–F10, kurzorové a numerické), pro IBM PC XT, nekompatibilní s AT
- **AT** – řadič (Super I/O) na základní desce, 101 a více kláves (104, 105)
- **ergonomická** – ergonomické rozložení kláves
- stisk kláves – mikrospínač (dříve), kapacitní snímač, vyhodnocení mikroprocesorem v klávesnici a odeslání kódu
- konektor: **DIN-5** (dříve), **PS/2** (Mini-DIN, fialový), USB (i redukce na PS/2), USB zdířky (obsahuje USB HUB), speciální (servery), bezdrátové (Bluetooth)

## Myš (mouse)

- = polohovací vstupní zařízení pro udávání pozice kurzoru na výstupu počítače a ovládání počítače (pomocí kurzoru a příkazů)
- 1963, D. Engelbart, Stanford, první 1982 Apple
- **(opto)mechanická** – pohyb snímán odvalováním dvou koleček (dříve) nebo pogumované **kuličky** ("kuličková" myš, 1972, Xerox) se dvěma dotýkajícími se válečky převedeným na rotaci dvou kotoučů (pro dva směry) s vodivou vrstvou pro spínače (dříve) nebo otvory pro dvojice fotodiod
- **optická** (1980) – pohyb snímán odrazem vyslaného **infračerveného paprsku** od desky, vysílače LED nebo laser, snímače fotodiody nebo CCD prvek (jako v dig. fotoaparátu), první potřebovaly speciální **podložku**, vyhodnocení posunu nasmínamých odrazů (v reálném čase, stovky pixelů<sup>2</sup> a snímků/s)

## Myš (mouse)

- **tlačítka**: levé, pravé, prostřední (dříve), kolečko/-a (1990, pro skrolování, stisk místo prostředního), boční, horní aj.
- konektor: sériového rozhraní (dříve), PS/2 (zelený), USB (i redukce na PS/2), bezdrátové (Bluetooth)
- alternativy:
  - **touchpad/trackpad** – destička měřící změny el. kapacity pohybem prstu po ní, tlačítka před a tzv. **tap** = „poklepání“, skrolování na pravém a dolním okraji, u přenosných počítačů
  - **trackpoint** – nakláněcí tyčinka v klávesnici ("malý joystick"), tlačítka před klávesnicí, u přenosných počítačů IBM/Lenovo (HP)
  - **trackball** – rotace kuličkou ("převrácená myš"), u přenosných počítačů (dříve), použití v grafice
  - **tablet** – citlivá deska (příp. včetně displeje) + pero nebo tzv. **puk** (jako myš se zaměřovacím křížem), pro kreslení, použití v grafice

## 2

Operační systém, architektura, poskytovaná rozhraní. Správa procesoru: procesy a vlákna, plánování jejich běhu, komunikace a synchronizace. Problém uváznutí, jeho detekce a metody předcházení. Správa operační paměti: segmentace, stránkování, virtuální paměť. Správa diskového prostoru: oddíly, souborové systémy, zajištění konzistence dat.

### 2.1 Operační systém, architektura, poskytovaná rozhraní

#### 2.1.1 Operační systém

##### **Operační systém (Operating System, OS)**

- = základní softwarové vybavení počítače – rozhraní mezi uživatelem a počítačem
- umožňuje programům (aplikacím) běh na/v počítači pomocí programového rozhraní (API) a uživateli práci s počítačem pomocí svého uživatelského rozhraní (UI) a programů
- cíl: snadné a efektivní využití počítače (pro uživatele i aplikace)
  - víceméně protichůdné požadavky – dříve důraz na efektivitu (a vůbec možnost), nyní spíše snadnost
  - kompromis, závisí na způsobu využití a typu počítače → různé OS
- poskytuje **abstrakci** (funkcí) hardware počítače, odstínuje uživatele a aplikace od hardware, např. čtení souboru:
  - aplikace: otevření souboru zadaného (úplným) jménem a získání „objektu“ souboru, čtení souboru pomocí „objektu“ po bytech, zavření souboru
  - OS: nalezení režijních informací o souboru na základě jména a vrácení „objektu“ souboru, nalezení (čísel) sektorů disku s daty souboru a čtení sektorů, zrušení „objektu“ souboru

## Úvod



##### **Operační systém (Operating System, OS)**

- abstrakce hardwarových zdrojů počítače: procesoru, operační paměti a vstupně/výstupních zařízení (viz von Neumannova koncepce počítače)
  - **dvě rozhraní**: pro komunikaci s hardware a pro umožnění využití hardware aplikacemi (API) a uživatelem (UI) skrze OS
  - **API (Application Programming Interface)** – zpřístupňuje služby OS programům, řešeno tzv. systémovými službami, dnes i virtualizované
  - **UI (User Interface)** – zpřístupňuje služby OS a programů uživatelům, textové příkazové (command line interface, CLI), grafické (GUI), virtualizované, mnohdy vícenásobné
- zajišťuje (bezpečnou a efektivní) **správu systémových (hardwareových) zdrojů** počítače
  - sdíleny běžícími programy v (krátkých) časových úsecích: procesor
  - rozdělovány mezi programy: paměť
  - dočasně programům přidělovány podle jejich potřeby: klávesnice, myš, I/O zařízení
  - virtualizovány pro transparentní sdílení programy: disková zařízení, grafický výstup, zvukový a síťový vstup/výstup aj., I/O zařízení

## Operační systém (Operating System, OS)

### ■ části:

- **jádro (kernel)** – vlastní OS, monolitické (jeden program s veškerou funkcionalitou, příp. i s ovladači hardware) nebo tzv. mikrojádro (jen správa procesoru, paměti a komunikace mezi ostatními částmi realizujícími ostatní funkcionalitu a ovladače hardware), dnes i hybridní
- **základní obslužné programy** – pro práci s OS a zdroji počítače, např. nástroje pro ovládání OS a práci s programy (aplikacemi) – **shell**, administrátorské a diagnostické nástroje pro práci s hardware a poč. sítí, základní nástroje pro manipulaci s daty aj.
- **uživatelské rozhraní (UI)** – součást jádra OS, shellu nebo programy – záleží na použití a typu OS, neinteraktivní (úložové/dávkové), interaktivní – textové s interpretorem příkazů (shell) nebo grafické s (typicky) okenním systémem

## Operační systém (Operating System, OS)

### ■ typy:

- různé v závislosti na způsobu využití a typu počítače
  - **univerzální** – pro desktopové a přenosné počítače typu **PC**, servery, mainframe apod.
  - **embedded** – specializované pro embedded zařízení, i upravené univerzální pro přenosné účelové počítače (např. Linux/Android, Apple iOS, MS Windows)
  - **reálného času** – zaručení vyřízení požadavku/odpovědi v pevně daném čase, např. VxWorks, QNX, upravené univerzální (např. RTLinux, MS Windows RT) i HW řešení, např. pro řízení strojů
  - **distribuované** – pro běh současně na více počítačích, simulace např. jedné společné paměti, pro počítačové **klastry (cluster)** = počítače propojené do sítě s možností běhu (typicky výpočetních) programů současně na všech
  - další ...
- dnes nejvíce používané: na desktopových PC MS Windows, Mac OS X, GNU/Linux, na síťových serverech unixové (GNU/Linux, BSD), MS Windows, na účelových počítačích různé (Linux/Android, Apple iOS, MS Windows), na embedded zařízeních různé (Linux, MS Windows)

## Historie OS



- před 50. léty neinteraktivní ovládání počítače, bez OS, max. 1 program
- **multiprogramování** – od 50. let, více programů (dávek), dávkové OS ⇒ potřeba přidělování paměti programům = správa operační paměti
- **jednoúložové (single task)** – max. 1 spuštěný program (hlavní úloha), po dokončení nebo pozastavení další (vedlejší/doplňkové)
- **víceúložové (multi task)** – od 60. let, více běžících úloh střídajících se při čekání na hardware (disk, periferie, obecně I/O) ⇒ potřeba plánování úloh = správa procesoru, samostatná činnost procesoru a ostatních zařízení → koncept přerušení
- **sdílení času (time-sharing)** – od 70. let, úlohám přidělován procesor na krátká časová kvanta → iluze současného běhu úloh ⇒ potřeba (hardwarový) časovač ... dnešní OS
- **víceuživatelské (multi user)** – více uživatelů současně ⇒ potřeba virtualizace uživ. rozhraní

## Historie OS



- od 50. let pro mainframe počítače, např. **OS/360** (pro IBM System/360, správa hardware, 50. a 60. léta), SCOPE (CDC), MCP (Burroughs, virtuální paměť, 60. léta), GECOS (GE, úrovně oprávnění programů a uživatelů), **Multics** (MIT/GE/Bell Labs, víceuživatelský), TOPS (DEC, 70. léta), **Unix**, VMS (DEC VAX), SunOS/Solaris (Sun), z/OS (IBM), **Linux**
- od 70. let pro mikropočítače, první minimalistické v ROM (tzv. monitor), diskové jednoúlohové např. CP/M (Digital Research), **MS DOS** (Microsoft, IBM PC, PC DOS, Free DOS), od 80. let víceúlohové pro PC např. **MS Windows** (Microsoft), NeXTSTEP (NeXT), **Mac OS X** (Apple, 90. léta), **Linux**, se zvyšováním výkonu i OS pro mainframe (Unix, Solaris)

## Příklady OS (pro PC)



### Unix

- 1965 Multics, konec 60. let, 1971, Bellovy laboratoře, verze System V, BSD (Berkeley System Distribution), AIX (IBM), HP-UX (HP), SunOS/Solaris (Sun) aj., volně použitelné i proprietární
- víceúlohový, víceuživatelský
- architektura: jádro + shell + programy (i implementující textové i grafické UI)
- pro různé procesory a počítače
- inspirující a ovlivňující vývoj dalších OS

**Další:** Minix (výukový), OS/2 (IBM, ukončený), Hurd (GNU), Plan 9 (Bell Labs, experimentální), Android (Linux, Google), iOS (Apple), Chrome OS (Linux + webový prohlížeč Google Chrome), Firefox OS (Linux, Mozilla), RTOS1 (real-time) aj.

## Příklady OS (pro PC)



### DOS

- 1981, Microsoft, IBM, Digital Research, různé proprietární verze
- jednoúlohový, jednouživatelský
- architektura: jádro + shell + programy (i implementující textové UI)
- pro procesory Intel 80x86

### MS Windows

- 1993 (NT), Microsoft (od 1985 (1983?) pouze jako víceúlohová GUI nadstavba nad DOS), proprietární
- víceúlohový, jednouživatelský (řada 9x, dříve), víceuživatelský (řada NT, dnešní)
- architektura (NT): jádro + subsystémy emulující API jiných OS (DOS, starší MS Windows, částečně unixové, OS/2) + shell (implementující GUI) + programy
- pro procesory Intel 80x86, ARM, Alpha (dříve)

## Příklady OS (pro PC)



### Mac OS X

- 1999 (Server, 2001 desktop), Apple (od 1984 Mac OS), pro počítače Apple Macintosh (Mac), proprietární
- víceúlohový, víceuživatelský, unixový
- architektura: jádro XNU (BSD Unix/Mach) + shell (API) + GUI + programy
- pro procesory Intel 80x86, ARM, IBM PowerPC (dříve)

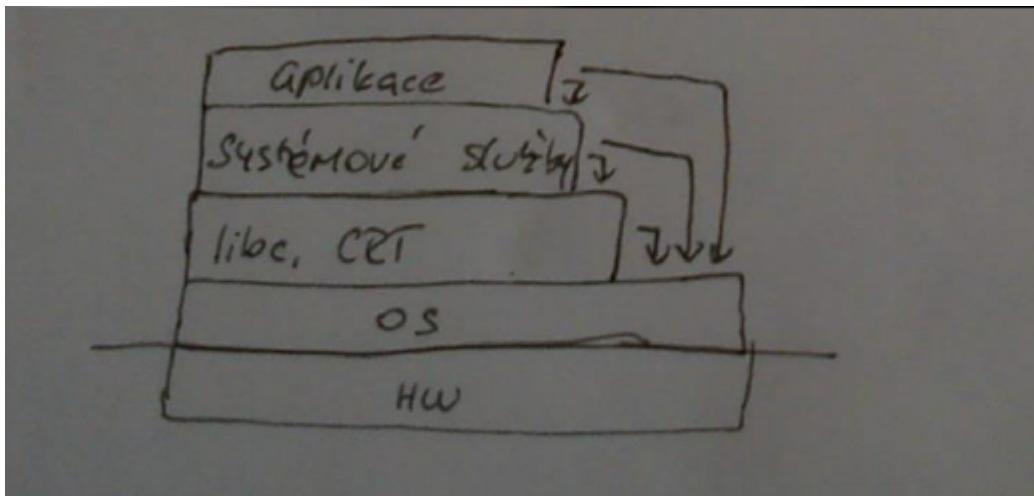
### GNU/Linux

- 1991 (1983 GNU), Linus Torvalds, svobodný (free) software (licence GNU GPL) ve formě distribucí („balení“ s dalšími obslužnými programy ve formě balíčků, i tzv. „živé“)
- víceúlohový, víceuživatelský, unixový
- architektura: jádro Linux + shell (GNU aj.) + programy (i implementující textové i grafické UI)
- pro mnoho procesorů, počítačů (i mimo PC) a jiných zařízení (elektronika)

## 2.2 Správa procesoru: procesy a vlákna, plánování jejich běhu, komunikace a synchronizace

### 2.2.1 Procesy

- Neformálně: proces = běžící program (vykonává činnost)
- Proces charakterizuje:
  - Kód programu

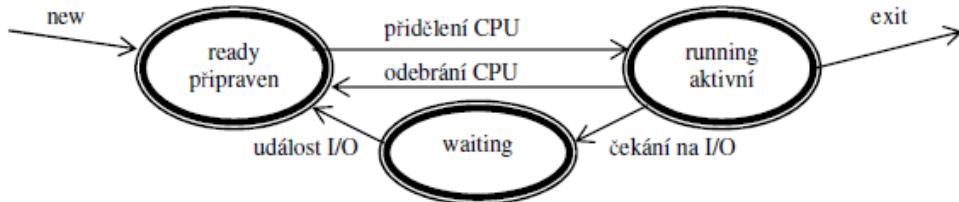


Obrázek 1: vrstvy HW/SW

- Paměťový prostor
- Data – statická a dynamická (halda – jsou tam uložená data, které se alokují při běhu procesu, typicky objekty alokované malloc nebo volané přes new)
- Zásobník
- Registry
- Oddělení jednotlivých úloh (abstrakce)
- Multiprogramování: (zdánlivě) souběžný běh více procesů
  - kooperativní - proces si sám odebere procesor
  - preemtivní - OS sám určuje, kdy odebere procesu procesor (velikost časového kvanta)
- Komunikace mezi procesy, sdílení zdrojů - synchronizace
- Informace o procesu má OS uloženy v Process Control Block (PCB)
  - identifikace procesu
  - informace o stavu
  - adresu instrukce, kterou bude program pokračovat
  - stav registrů
  - informace k plánování procesů
  - informace o přidělené paměti
  - informace o používaných I/O zařízeních, otevřených souborech, atd.
- potřeba plánování

### OBECNÝ ŽIVOTNÍ CYKLUS PROCESU

- Nový (new) – proces byl vytvořen
- Připravený (ready) – proces čeká, až mu bude přidělen CPU
- Běžící (running) – CPU byl přidělen a právě provádí činnost
- Čekající (waiting/blocked) – proces čeká na vnější událost (např. na vyřízení I/O požadavku, synchronizační primitiva)
- Ukončený (terminated) – proces skončil svou činnost (dobrovolně x nedobrovolně)



Obrázek 2: stavy procesu

### 2.2.2 Plánování procesů

- Potřeba efektivně plánovat procesorový čas
- Časové kvantum: maximální čas přidělený procesu
- Samotné přepnutí procesu má režii (uložení kontextu, vyprázdnění cache) -> latence
- Symmetric Multi-Processing - přibývá problém, jak vybrat procesor
  - oddělené plánování pro každý procesor
  - maska affinity (definuje procesor, na kterém může proces běžet)

#### TYPY PLÁNOVÁNÍ:

- Dlouhodobé – rozhoduje, zda bude přijat k běhu (změna stavu z NEW na READY)
- Střednědobé – načtení/odložení procesu do sekundární paměti
- Krátkodobé – rozhoduje dostupné procesy ke spuštění na CPU

#### RŮZNÉ TYPY ÚLOH/SYSTÉMŮ:

- Iterativní
- Dávkové zpracování
- Pracují v reálném čase

#### OBECNÉ POŽADAVKY NA PLÁNOVÁNÍ PROCESŮ:

- Spravedlnost – každému procesu by v rozumné době měl být přidělen CPU
- Vyházenost – celý systém běží
- Efektivita – maximální využití CPU
- Maximalizace odvedené práce (throughput)
- Minimalizace doby odevzdy
- Minimalizace doby průchodu systémem (turnaround) – od spuštění do konce procesu, aby byl co nejmenší časový úsek

#### ALGORITMY PRO PLÁNOVÁNÍ PROCESŮ

- FIRST-COME-FIRST-SERVED

- První proces získává procesor
- Jednoduchý, neefektivní
- Nepreemptivní

- SHORTEST JOB FIRST

- Vybere se takový proces, který poběží nejkratší dobu
- Je potřeba znát (odhadnout) čas, který proces potřebuje
- Nepreemptivní
- SHORTEST REMAINING TIME NEXT
  - Pokud aktivní proces potřebuje k dokončení činnosti méně času než aktuální, je spuštěn
  - Preemptivní
- ROUND ROBIN (CHODÍ PEŠEK OKOLO)
  - Každý proces má pevně stanovené kvantum
  - Připravené procesy jsou zařazeny ve frontě a postupně dostávají CPU
  - Nevýhoda: přiděluje stejný čas jak systémovým, tak uživatelským procesům
- VÍCEÚROŇOVÁ FRONTA
  - Každý proces má definovanou prioritu
  - Statické x dynamické nastavení priority (např. vyšší priorita pro I/O)
  - Systém eviduje pro každou frontu (čekající procesy)
  - Riziko vyhladovění procesů s nízkou prioritou (Rozšíření: nastavení různých velikostí kvant pro jednotlivé priority)
  - Používán Windows NT, staršími Linuxovými jádry
- SHORTEST PROCESS NEXT - Používá se odhad, podle předchozí aktivity procesu
- GUARANTEED SCHEDULING
  - Máme-li n procesů, každý proces má získat 1/n CPU
  - Volí se proces s nejmenším poměrem
  - Používají novější Linuxové jádra
- LOTTERY SCHEDULING - Proces dostane příděl "losů"
- FAIR-SHARE SCHEDULING - Plánování podle skupin procesů (např. podle uživatelů)

### 2.2.3 Vlákna

- OS umožňují rozdělit procesy na více vláken (je základní entitou)
- Každé vlákno má svůj zásobník + registry (přepnutí vláken v rámci procesu je rychlejší)
- vlákna v rámci procesu sdílí paměťový prostor, data a prostředky -> nové problémy se synchronizací

#### IMPLEMENTACE VLÁKEN:

- Jako knihovna v uživatelském prostoru (první Linuxové systémy)
- Součást jádra operačního systému (Windows a další Linuxové systémy)
- Kombinované řešení

#### PROCESY A VLÁKNA V UNIXECH

- Původně základní entitou byl proces
- Procesy tvoří hierarchii rodič-potomek (na vrcholu je proces init)
- Vytvoření procesu pomocí volání fork(), který vytvoří klon procesu

- Přepsání procesu pomocí exec
- Komunikace mezi procesy: zasílání signálů, roury, ...
- Možnost nastavit nice ness procesu (prioritu od -20 do 20)
- Vlákna do Unixu přidána až později (implementace se v rámci různých OS liší)
- V Linuxu (pthreads) vnitřně implementovány stejně jako procesy (task), ale sdílí paměťový prostor

### **PROCESY A VLÁKNA VE WINDOWS:**

- Windows NT navrženy s vlákny jako základní entitou pro běh aplikace
- Proces sdružuje vlákna, plánování se účastní vlákna
- Sofistikovaný systém priorit při plánování vláken
- Priorita vláken je odvozena od třídy priority procesu
- Proměnlivá velikost kvant
- Priority boost - dochází k dočasněmu navýšení priority
  - po dokončení I/O operace
  - po dokončení čekání na semafor
  - vlákno již dlouho neběželo

#### **2.2.4 Komunikace**

- IPC – Inter-process communication
- Procesy oddělené, potřeba kooperace
- Základní kategorie
  - Sdílená paměť
    - \* Procesy sdílejí kousek paměti - nutná spolupráce se správou paměti
    - \* Čtení i zápis, náhodný přístup
    - \* Deklarace, že paměť je sdílená + namapování do adresního prostoru
    - \* SIGNÁLY - Mechanizmus podobný přerušení (asynchronní volání)
    - \* ROURY - Mechanizmus umožňující jednosměrnou komunikaci mezi procesy
  - Zasílání zpráv
    - \* obecný mechanizmus komunikace mezi procesy
    - \* vhodný pro počítače se sdílenou pamětí i pro distribuované systémy
    - \* lze jeho pomocí implementovat vzájemné vyloučení (mutual exclusion)
  - Synchronizace
  - Vzdálené volání procedur

#### **2.2.5 Synchronizace**

- procesy a vlákna přistupují ke sdíleným zdrojům (paměť, souborový systém)
- příklad: současné zvýšení hodnoty proměnné o 1 (díky preemtivnímu plánování tohle opravdu může nastat)
  1. A: načte hodnotu proměnné X z paměti do registru ( $X = 1$ )
  2. A: zvýšení hodnotu v registru o jednu
  3. B: načte hodnotu proměnné X z paměti do registru ( $X = 1$ )

- 4. A: uloží hodnotu zpět do paměti ( $X = 2$ )
- 5. B: zvýší hodnotu v registru o jedna
- 6. B: uloží hodnotu zpět do paměti ( $X = 2$ )
- řešení - atomické operace nebo synchronizace
- je potřeba zajistit, aby v průběhu manipulace s určitými zdroji nemohl manipulovat někdo jiný (vzájemné vyloučení)

## ATOMICKÝ PŘÍSTUP DO PAMĚTI

- Obecné přístupy do paměti nemusí být atomické (záležitosti CPU, překladače)
- Lze vynutit určité chování -> klíčové slovo volatile – často záleží na překladači
- Memory barriers umožňují vynutit si synchronizaci (záležitost CPU)

## ATOMICKÉ OPERACE:

- Test-and-Set (TAS): nastav proměnnou a vrát její původní hodnotu
- Compare-and-Swap (CAS): ověří, jestli se daná hodnota rovná požadované a pokud ano, přiřadí ji novou (CMWXCHG)
- Fetch-and-Add: vrátí hodnotu místa v paměti a zvýší jeho hodnotu o jedna (XADD)
- Load-link/Store-Conditional (LL/SS): načte hodnotu a pokud během čtení nebyla změněna, uloží do ní novou hodnotu

## SYNCHRONIZAČNÍ PRIMITIVA

- slouží k řízení přístupu ke sdíleným zdrojům
- Petersonův algoritmus - umožňuje implementovat vzájemné vyloučení dvou procesů pomocí běžných operací

## KRITICKÁ SEKCE

- Obecně třeba zajistit, aby se sdílenými zdroji pracoval jen jeden proces
- Část kódu, kdy program pracuje se sdílenými zdroji (např. pamětí)
- Pokud jeden proces je v kritické sekci, další proces nesmí vstoupit do své kritické sekce
- Každý proces před vstupem žádá o povolení vstoupit do kritické sekce
- Požadavky na kritickou sekci:
  - Vzájemné vyloučení – maximálně jeden proces je v daný okamžik v KS
  - Absence zbytečného čekání – není-li žádný proces v kritické sekci a proces do ní chce vstoupit, není mu bráněno
  - Zaručený vstup – proces snažící se vstoupit do KS, do ní v konečném čase vstoupí
- Řešení:
  - Zablokování přerušení (použitelné v rámci jádra OS); více CPU -> neefektivní
  - Aktivní čekání - Spinlocks – testujeme pořád dokola jednu proměnnou

Př. 1) ŠPATNÉ - Vstup do kritické sekce a její uzamknutí  
není provedeno atomicky

```
while (lock); // čekej
lock = 1;
// kritická sekce
lock = 0;
```

Př. 2)

Uvažujeme následující atomickou operaci

```
bool test_and_set(bool * target) {
bool rv = *target;
*target = true;
return rv;
}
```

A kód:

```
while (test_and_set(&lock) == true);
// kritická sekce
lock = false;
```

Př. 3)

Uvažujeme následující atomickou  
operaci, která prohodí dvě hodnoty

```
void swap(bool *a, bool *b) {
bool tmp = *a;
*a = *b;
*b = tmp;
}
```

A kód

```
key = true;
while(key == true)
swap(&lock, &key);
// kritická sekce
lock = false;
```

## SEMAFOR

- Chráněná proměnná obsahující počítadlo s nezápornými celými čísly
- Operace P (proberem – zkusit): pokud je hodnota čísla nenulová sníží hodnotu o jedna, jinak čeká, až bude hodnota zvýšena (operace někdy označena i jako wait)

```
void P (Semaphore s) {
while (s <= 0) { }
s--;
}
```

- Operace V (verhogen – zvýšit): zvýší hodnotu o jedna (operace označování jako signal, post)

```
void V (Semaphore s) {
s++;
}
```

- operace P a V se provádí atomicky - operace jsou na začátku a na konci zamykané pomocí spinlocku, protože je kód relativně krátký, tak se nečeká moc dlouho na čekání

- Binární semafor – může nabývat hodnot 0, 1 (mutex, implementace kritické sekce)
- Obecný semafor – slouží k řízení přístupu ke zdrojům, kterých je končené množství
- Implementace s pomocí aktivního čekání nebo OS (pasivní čekání)

```

struct sem {
    int value;
    struct process * list;
};

void P (struct sem * s) {
    s -> value--;
    if (s -> value < 0) {
        // přidej proces s->list;
        block(); // uspi aktuální proces
    }
}

void V(struct sem * s) {
    s-> value++;
    if (s -> value <= 0) {
        // odeber proces P z s s-> list
        wakeup(P);
    }
}

```

- Operace musí být provedeny atomicky

#### DALŠÍ SYNCHRONIZAČNÍ NÁSTROJE:

- Bariéry - Synchronizačních metod vyžadujících, aby se proces zastavil v daném bodě, dokud všechny procesy nedosáhnout daného bodu
- Read-Write zámky
  - Vhodné pro situace, které čtou i zapisují do sdíleného prostředku
  - Čtecí a zapisovací režim zámku
  - Vhodný pokud jde rozlišit čtenáře a písáře (písářů je více)
- Monitor
  - Modul nebo objekt
  - V jeden okamžik může kteroukoliv metodu používat pouze jeden proces/vlákno
  - Nutná podpora programovacího jazyka
  - Java (synchronized), .NET (lock)

### 2.3 Problém uváznutí, jeho detekce a metody předcházení

#### 2.3.1 Deadlock

- Uváznutí – systém se dostal do stavu, kdy nemůže dál pokračovat
- U množiny procesů došlo k uváznutí (deadlocku), pokud každý proces z této množiny čeká na událost, kterou pouze proces z této množiny může vyvolat.

#### UŽÍVÁNÍ PROSTŘEDKŮ:

- Request – požadavek na prostředek, není-li k dispozici, proces čeká

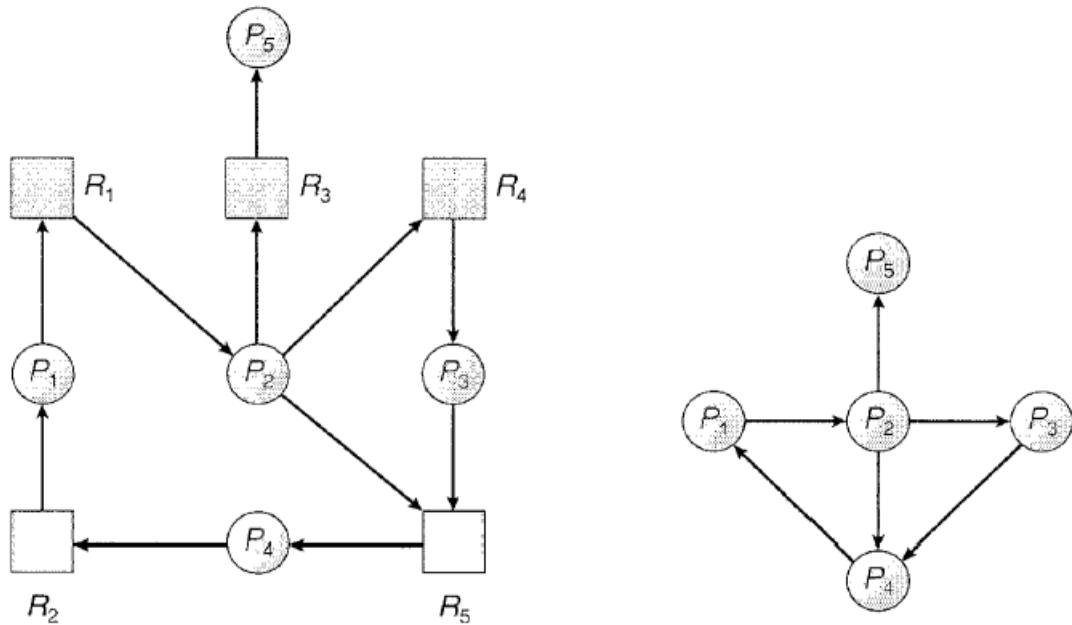
- Use – vlákno s prostředkem pracuje
- Release – uvolnění prostředku pro další použití

#### PODMÍNKY VZNIKU:

- Mutual exclusion – alespoň jeden prostředek je výlučně užíván jedním procesem
- Hold and wait – proces vlastní alespoň jeden prostředek a čeká na další
- No preemption – prostředek nelze násilně odebrat
- Circular wait – cyklické čekání (proces A vlastní prostředek 1, chce prostředek 2, který drží B a současně žádá o 1)

#### 2.3.2 Řešení deadlocku

- IGNORACE - "neřešení", v praxi často používané tzv. pštrosí algoritmus – strčí se hlava do píska a dělá se, že žádný problém není
- DETEKCE
  - Pokud vznikne deadlock, je detekován a některý proces odstraněn
  - K detekci se používá alokační graf prostředků a graf čekání
  - Alokační graf:
    - \* Orientovaný graf
    - \* Dva typy uzlů – prostředek, proces
    - \* Hrana proces-prostředek – proces čeká na prostředek
    - \* Hrana prostředek-proces – prostředek je vlastněn procesem
  - Graf čekání vznikne vynecháním uzlů prostředků a přidáním hran  $P_n \rightarrow P_m$  pokud existovaly hrany  $P_n \rightarrow R$  a  $R \rightarrow P_m$ , kde  $P_n$  a  $P_m$  jsou procesy a  $R$  je prostředek
  - Ukázka alokačního grafu a grafu čekání:



Obrázek 3: alokační graf a graf čekání

- Deadlock vznikne, pokud je v grafu čekání na cyklus

- ZAMEZENÍ VZNIKU (PREVENCE)

- Snažíme se zajistit, že některá z podmínek není splněna
- Zamezení výlučnému vlastnění prostředků (často nelze z povahy zařízení)
- Zamezení držení a čekání
  - \* Proces zažádá o všechny prostředky hned na začátku
  - \* Problém s odhadem
  - \* Plýtvání a hladovění
  - \* Množství prostředků nemusí být známé předem
  - \* Jde použít i v průběhu procesu (ale proces se musí vzdát všech prostředků)
- Zavedení možnosti odejmout prostředek – vhodné tam, kde nelze odejmout prostředky tak, aby nešlo poznat, že byly odebrány
- Zamezení cyklickému čekání – zavedení globálního číslování prostředků a možnost žádat prostředky jen v daném pořadí

- VYHÝBÁNÍ SE UVÁZNUTÍ

- Procesy žádají prostředky libovolně
- Systém se snaží vyhovět těm požadavkům, které nemohou vést k uváznutí
- Je potřeba znát předem, kolik prostředků bude vyžádáno
- Tomu je přizpůsobeno plánování procesů
- Bezpečný stav – existuje pořadí procesů, ve kterém jejich požadavky budou vyřízeny bez vzniku deadlocku
- Systém, který není v bezpečném stavu, nemusí být v deadlocku
- Systém odmítne přidělení prostředků, pokud by to znamenalo přechod do bezpečného stavu (proces musí čekat)
- ALGORITMUS NA BÁZI ALOKAČNÍHO GRAFU

- \* Vhodný, pokud existuje jen jedna instance každého prostředku
- \* Do alokačního grafu přidáme hrany (proces-prostředek) označující potenciální žádosti procesu a prostředky
- \* Žádosti a prostředek se vyhoví pouze tehdy, pokud konverze hrany na hranu typu (prostředek je vlastněn-procesem) nepovede ke vzniku cyklu

- BANKÉŘŮV ALGORITMUS

- \* Vhodný tam, kde je větší počet prostředků daného typu
- \* Na začátku každý proces oznámí, kolik prostředků jakého typu bude maximálně potřebovat
- \* Při žádosti o prostředky systém ověří, jestli se nedostane do nebezpečného stavu
- \* Pokud nelze vyhovět, je proces pozdržen
- \* Porovnávají se volné prostředky, s aktuálně přidělenými a maximálními
- \* Uvažujme m prostředků a n procesů
- \* Matice m x n
  - Max – počet prostředků, které bude každý proces žádat
  - Assigned – počet přiřazených prostředků jednotlivým procesům
  - Needed – počet prostředků, které bude proces ještě potřebovat (evidentně needed = max – assigned)
- \* Vektory velikosti m
  - E – počet existujících prostředků
  - P – počet aktuálně držených prostředků
  - A – počet zdrojů

	Assigned					Needed			
	K	L	M	N		K	L	M	N
A	3	0	1	1		1	1	0	0
B	0	1	0	0		0	1	1	2
C	1	1	1	0		3	1	0	0
D	1	1	0	1		0	0	1	0
E	0	0	0	0		2	1	1	0

$E = \langle 6, 3, 4, 2 \rangle$

$P = \langle 5, 3, 2, 2 \rangle$

$A = \langle 1, 0, 2, 0 \rangle$

- Podmínu splňuje proces D → odebrán a  $A \leftarrow \langle 2, 1, 2, 1 \rangle$
- Podmínu splňuje proces A → odebrán a  $A \leftarrow \langle 5, 1, 3, 2 \rangle$
- Podmínu splňuje proces B → odebrán a  $A \leftarrow \langle 5, 2, 3, 2 \rangle$
- Podmínu splňuje proces C → odebrán a  $A \leftarrow \langle 6, 3, 4, 2 \rangle$
- Podmínu splňuje proces E → odebrán a  $A \leftarrow \langle 6, 3, 4, 2 \rangle$

Obrázek 4: bankéřův algoritmus

1. Najdi řádek i v needed takový, že  $\text{needed}[i] \leq A$ , pokud takový není, systém není v bezpečném stavu
2. Předpokládej, že proces skončil a uvolnil své zdroje;  $A \leftarrow A + \text{assigned}[i]$  a odstraň řádný i ze všech matic
3. Opakuj body 1 a 2 dokud nejsou odstraněny všechny procesy nebo není jasné, že systém není v bezpečném stavu

## 2.4 Správa operační paměti: segmentace, stránkování, virtuální paměť.

### 2.4.1 Stránkování

- adresní (logický) prostor procesu je rozdelen na menší úseky - stránky (pages)
- fyzická paměť je rozdělena na úseky stejně délky - rámce (frame, page frames)
- provádí se mapování logický adres  $\rightarrow$  na fyzické
- procesy už nemusí být umístěny v souvislých blocích paměti
- výpočet fyzické adresy musí být implementovaný HW (efektivita) - pokud by implementoval OS, nemohl by ho používat
- CPU si udržuje stránkovací tabulku
- logická adresa má dvě části: pd, kde p je číslo stránky a d je offset
- fyzická adresa vznikne jako fd, kde f je číslo rámce v tabulce stránek příslušného stránce p

### ADRESÁŘ STRÁNEK

- velikost stránek je většinou mocnina 2 (typicky 4 KB)

- jednoduchý přesun na disk
- uvažujeme-li velikost stránky 4 KB a velikost adresního prostoru 32 bitů -> velikost adresní tabulky je 1 milion záznamů
- nepraktické udržovat tak velkou tabulkou (obzvlášť pro každý proces)
- používá se víceúrovňová tabulka
- část logické adresy udává tabulkou, další část index v tabulce a další část offset
- např. pro 4 KB stránky může být toto rozložení 10-10-12 bit.
- tzn. systém k adresaci používá 1024 tabulek po 1024 záznamech
- značnou výhodou: nevyužité tabulky mohou být prázdné -> nemusí se evidovat
- v praxi se používají i tří a čtyřúrovňové tabulky

### **Translation Lookaside Buffer (TLB)**

- cache procesoru obsahující hodně používané části stránkovacích tabulek
- pro danou stránku uchovává adresu rámce z důvodů efektivity
- pokud je adresa v cache (cache hit) je hodnota vrácena velice rychle (cca 1 hodinový cyklus)
- pokud hodnota není v cache (cache miss) načtení adresy trvá delší dobu (10 - 100 cykl.), typicky miss rate <1% ! průměrný přístup k zjištění rámce je 1.5 hodinového cyklu
- princip lokality - je vhodné programovat tak, aby data ke kterému se přistupuje byly na jednom místě (lokální proměnné na zásobníku, ne globální, které jsou rozesety po paměti programu)

## **VLASTNOSTI STRÁNEK**

- Rezervovaná stránka
  - existují v adresním prostoru, ale nezapisovalo se do ní
  - každá stránka je nejdřív rezervovaná
  - vhodná pro velké pole, ke kterým se přistupuje postupně
- Komitovaná stránka (Committed)
  - stránka má rámec v primární nebo sekundární paměti
  - musí řešit jádro
  - paměť je často současně komitovaná i rezervovaná
- dirty bit - 0 pokud má stránka přesnou kopii v sekundární paměti; 1 nastaveno při změně (nutná podpora HW)

## **VÝMĚNA STRÁNEK**

- page fault
- pokud není stránka v primární paměti, načte stránku do ní
- není-li volný rámec v primární paměti, je potřeba odsunout nějakou stránku do sekundární paměti
  - získáme volný rámec v sekundární paměti (pokud není volný rámec v sekundární paměti, najde se takový, který má kopii v primární paměti, nastaví se dirty bit a daný rámec se použije)
  - vybere se "oběť"- stránka v primární paměti, která bude uvolněna
  - pokud má stránka nastavený dirty bit, překopíruje se obsah rámce do sekundární paměti
  - načte se do primární paměti stránka ze sekundární

- zopakuje instrukci, která vyvolala page fault
- některé stránky je možné zamknout, aby nebyly odswapovány (nutné pro jádro, rámce sdílené s HW)

## VÝBĚR OBĚTI

- hledáme stránku, která nebude v budoucnu použitá (případně v co nejvzdálenější budoucnosti)
- FIFO
  - velice jednoduchý algoritmus
  - stačí udržovat frontu stránek
  - při načtení nové stránky je stránka zařazena na konec fronty
  - pokud je potřeba uvolnit stránku bere se první z fronty
  - nevýhoda - odstraní i často užívané stránky
- Least Frequently Used (LFU)
  - málo používané stránky -> nebudou potřeba
  - problém se stránkami, které byly nějaký čas intenzivně využívány (např. inicializace)
- Most Frequently Used (MFU)
  - práva načtené stránky mají malý počet přístupů
- Least Recently Used (LRU)
  - jako oběť je zvolena stránka, která nebyla nejdýl používaná
  - je potřeba evidovat, kdy bylo ke stránce naposledy přistoupeno
  - počítadlo v procesoru, inkrementované při každém přístupu a ukládané do tabulky stránek
- LRU (přibližná varianta)
  - každá stránka má přístupový bit (reference bit) nastavený na 1, pokud se ke stránce přistupovalo
  - na počátku se nastaví reference bit na 0
  - v případě hledání oběti je možné určit, které stránky se nepoužívaly

## TRASHING

- systém je ve stavu, kdy odvádí spoustu práce, ale bez rozumného efektu
- ideální je, aby měl proces tolik rámců, kolik potřebuje
- řešení:
  - Pracovní množina rámců(working-set)
    - \* má-li proces tolik rámců, kolik jich v nedávné době (lokalitě) použil -> OK
    - \* má-li jich více -> neefektivní využití
    - \* má-li jich méně ! hrozí trashing a je lepší celý proces odsunout z primární paměti
  - Frekvence výpadků stránek
    - \* sledujeme, jak často dochází u procesu k výpadku stránky
    - \* pokud proces je mimo tyto meze -> přidat/ubrat rámce

#### **2.4.2 Segmentace**

- paměť je rozdělena na několik segmentů (například kód, data, zásobník)
- speciální případ přidělování souvislých bloků paměti
- stránkování je obvykle pro programátora nerozeznatelné
- naproti tomu segmentace umožňuje rozdělit program do logických celků (kód, data)
- při použití segmentace a stránkování programy nepracují přímo s lineární adresou
- používají adresu ve tvaru segment + offset a ta se až převádí na fyzickou adresu pomocí stránkování
- ochrana paměti přes začátek a délku segmentu + oprávnění

#### **i386: SEGMENTACE**

- pro každý segment lze nastavit oprávnění (ochrana paměti)
- segmenty jsou popsány pomocí deskriptorů 8 B záznam  
  - báze
  - limit (velikost segmentu)
  - požadované oprávnění (ring 0-4)
- deskriptory segmentů uloženy v:
  - Global Descriptor Table (GDT) - sdílená všemi procesy
  - Local Descriptor Table (LDT) - každý proces má vlastní
- každá může mít až 8192 záznamů
- přístupné přes registry GDTR, LDTR
- první záznam v GDT "null"deskriptor

#### **i386: PŘEKLAD ADRES I. (SEGMENTACE)**

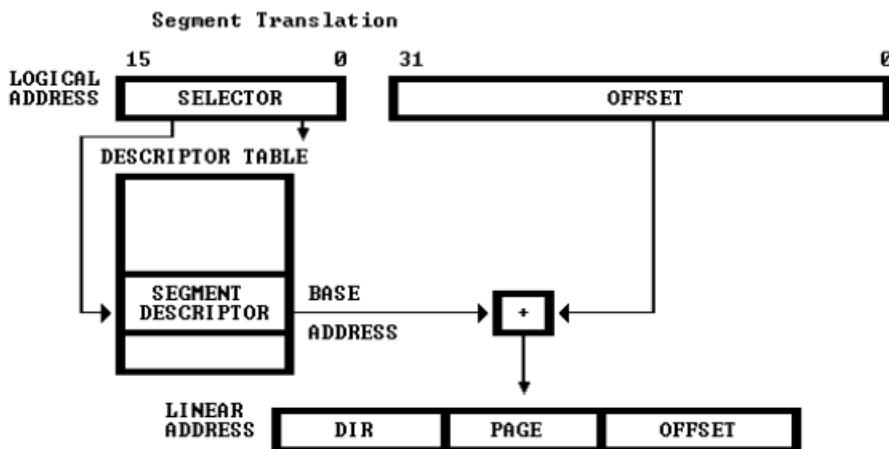
- logická adresa -> lineární adresa (segmentace)
- ověří se oprávnění a limit (přístup za hranici segmentu) -> neoprávněný přístup
- báze segmentu je načtena s offsetem -> lineární
- lineární adresa -> fyzická adresa
- standardní stránka/rámec: 4 KB

#### **i386: PŘEKLAD ADRES II. (STRÁNKOVÁNÍ)**

#### **2.4.3 Virtuální paměť**

##### **MOTIVACE**

- paměť RAM je relativně drahá -> nemusí vždy dostačovat
- aktuálně používaná data (například instrukce) musí byt v RAM, nepoužívaná data nemusí (velké programy -> nepoužívané funkce)
- je vhodné rozšířit primární paměť (RAM) o sekundární (např.: HDD)
- zvětšením dostupné paměti je možné zjednodušit vývoj aplikací (není potřeba se omezovat v množství použité paměti)



Obrázek 5: segmentace

- sekundární paměť bývá řádově pomalejší
- k efektivní implementaci je potřeba spolupráce HW (MMU) a OS
- z pohledu aplikace musí být přístup k paměti transparentní
- virtuální paměť (VM) je součástí soudobých OS (swapování)
  - Windows NT - stránkovací soubor (pagefile.sys)
  - Linux - swap partition (ale může být i soubor)
- bezpečnost dat v sekundární paměti? (např.: po vypnutí počítače, řeší se např. kódováním)

## VIRTUÁLNÍ PAMĚŤ

- způsob správy operační paměti počítače
- umožňuje předložit běžícímu procesu adresní prostor paměti, který je uspořádán jinak nebo je dokonce větší, než je fyzicky připojená operační paměť RAM
- virtuální adresy - pracují s nimi strojové instrukce, resp. běžící proces
- fyzické adresy - odkazují na konkrétní adresové buňky paměti RAM
- převod mezi virtuální a fyzickou adresou je zajišťován samotným procesorem
- virtuální paměť je implementována pomocí stránkování paměti spolu se stránkováním na disk, které rozšiřuje operační paměť o prostor na pevném disku

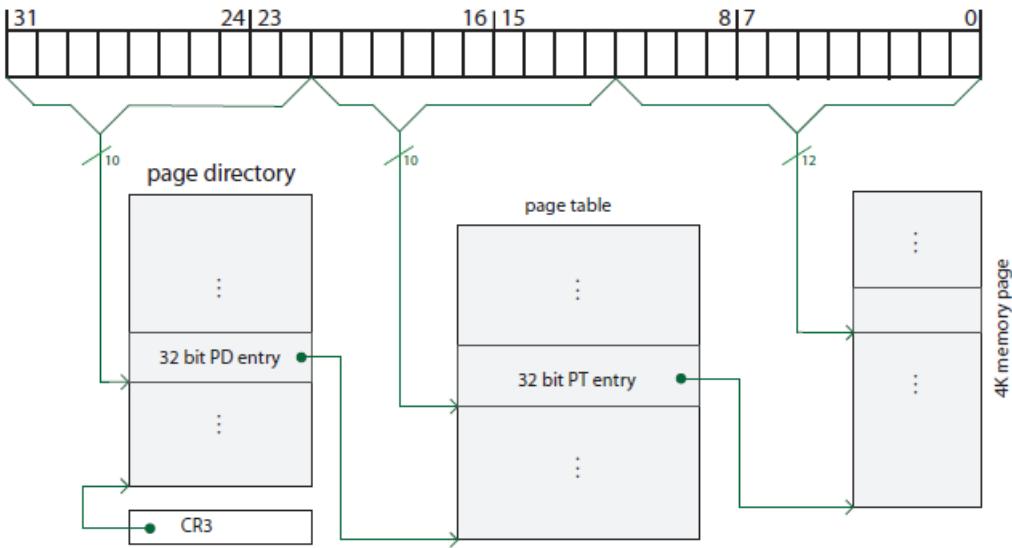
## 2.5 Správa diskového prostoru: oddíly, souborové systémy, zajištění konzistence dat

### 2.5.1 Souborové systémy

- způsob organizace dat ve formě souborů (a většinou i adresářů) tak, aby k nim bylo možné snadno přistupovat
- souborové systémy jsou uloženy na vhodném typu elektronické paměti, která je umístěna přímo v počítači (pevný disk nebo CD,...)

## MOTIVACE

Linear address:



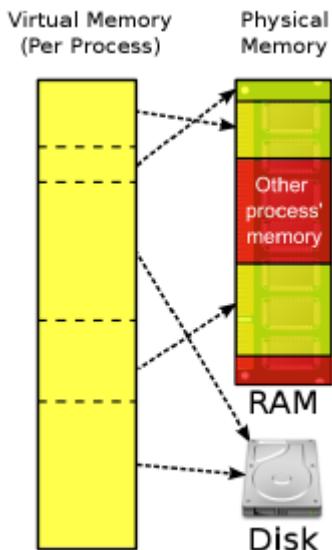
Obrázek 6: strankování

- potřeba uchovávat větší množství dat (primární paměť nemusí dostačovat)
- data musí být perzistentní (musí přežít ukončení procesu)
- k souborům musí být umožněn současný přístup
- řešení v podobě ukládání dat na vnější paměť (např.: disk)
- data ukládána do souborů tvořících souborový systém (File System/FS)
- soubor jako proud bytů (doprovázen doplňujícími informacemi)
- souborový systém jako abstrakce (odstínění od implementačních detailů)
- zajímavý problém: pojmenování objektů (souborů) a jejich organizace

## OPERACE SE SOUBORY

- create - vytvoření souboru
- write/append - zápis do souboru (na konec, popř. přepis); souvislý blok vs. postupný zápis
- read - čtení ze souboru (do přichystaného bufferu)
- seek - změna pozice
- erase - odstranění souboru (uvolnění místa); link a unlink
- truncate - zkrátil daný soubor na požadovanou velikost
- open - otevře soubor, aby s ním šlo manipulovat přes popisovač (file descriptor, file handle)
- close - uzavře soubor
- "soubory" nemusí mít jméno
- jeden soubor může být otevřen vícekrát (vice ukazatelů na pozici v souboru)

## ORGANIZACE SOUBORŮ



Obrázek 7: virtuální paměť

- soubory jsou rozlišované podle názvů (často specifikované pro daný OS nebo FS)
- rozlišování velkých a malých písmen (Unix vs. MS-DOS a Windows)
- MS-DOS: požadavek na jméno souboru ve tvaru 8+3: jméno + přípona (zůstalo zachováno ve FAT)
- typicky se soubory organizují do adresáře (složek)
- každý adresář může obsahovat běžné (popř. speciální) soubory i další adresáře -> stromová struktura
- k přístupu k souboru se používají:
  - absolutní cesty: /foo/bar/baz.dat
  - relativní cesty: foo/bar.dat -> každý proces má aktuální adresář
- operace s adresáři: Create, Delete, OpenDir, CloseDir, ReadDir, Rename
- struktura nemusí být hierarchická -> obecný graf (acyklický, cyklický)
  - hardlink - ukazatel na soubor (jeho tělo/obsah) (výhoda - transparentnost, nevýhoda - odkaz pouze v jednom FS)
  - symlink - soubor jako ukazatel na jiný soubor (specifikovaný cestou) (lze používat odkaz na jiný FS, ale nejsou plně transparentní - může vzniknout odkaz na neexistující soubor)

## 2.5.2 Oddíly

### DĚLENÍ DISKU

- každý fyzický disk se skládá z jedné nebo více logických částí (partition, oddíl); popsané pomocí partition table daného disku
- v každém partition může existovat souborový systém (označovaný jako svazek)
- v Unixech je každý svazek připojen (mounted) jako adresář (samostatný svazek pro /, /home, /usr)
- ve Windows jednotlivé svazky označeny (a:, b:, c:, . . . ); ale funguje i mountování (preferovaný jeden svazek pro vše)

- Virtual File System (VFS)
  - využití abstrakce =) umožňující kombinovat různé FS do jednoho VFS
  - možnost připojit běžný soubor jako svazek (i svazek jako soubor)
  - síťové disky (NFS, CIFS)

## **STRUKTURA SOUBORŮ**

- často je soubor chápán jako proud bytu
- sekvenční vs. náhodný přístup
- společně s daty jsou k souboru připojena metadata (atributy)
  - vlastník souboru
  - přístupová práva
  - velikost souboru
  - příznaky (skrytý, archivace, spustitelný, systémový)

### **2.5.3 Souborové systémy - implementace**

#### **OČEKÁVÁME:**

- (budeme předpokládat souborové systémy pro práci s disky s rotujícími částmi)
- schopnost pracovat se soubory a disky adekvátní velikosti
- efektivní práce s místem (evidence volného místa, nízká fragmentace)
- rychlý přístup k datům
- eliminace roztroušených dat na disku
- odolnost proti poškození při pádu systému (výpadku napájení) -> rychlé zotavení
- snapshoty - obraz disku v jednom okamžiku ke kterému jsme schopni se vrátit
- komprese dat
- možnost zvětšovat/zmenšovat FS za běhu
- kontrolní součty
- defragmentace za běhu
- správa oprávnění

## **STRUKTURA DISKU**

- pro jednoduchost předpokládáme, že struktura disku je lineární
- MBR - master boot record: informace o rozdělení disku na svazky + zavaděč
- sektor disku - obvykle velikost 512 B
- pracuje se s většími bloky 1 - 32 kB (často 4 kB nebo 8 kB)
- jednotlivé svazky obsahují souborový systém (vlastní organizace dat)

## **FAT**

- souborový systém pro MS-DOS (přežil až do Windows ME)
- jednoduchý design

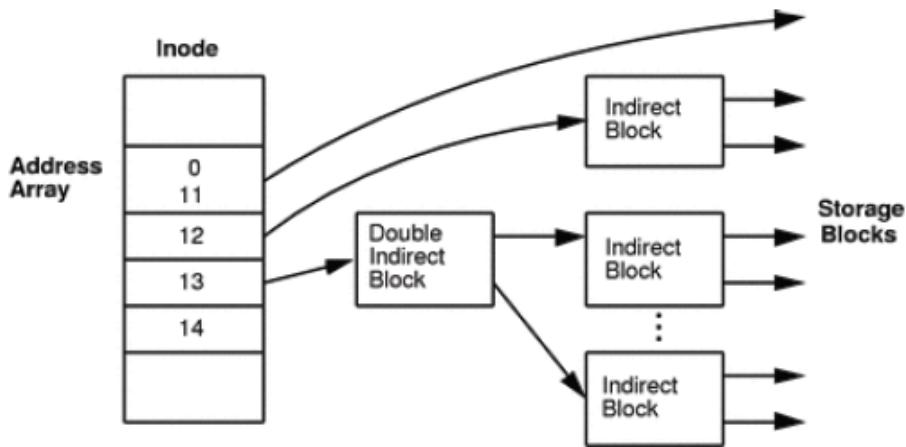
- soubory se jmény ve tvaru 8+3, nepodporuje oprávnění
- nemá metody proti poškození dat
- disk rozdelený na bloky (clustery)
- soubory popsány pomocí File Allocation Table (FAT) - spojový seznam
- disk (oddíl) rozdelen na 4 části:
  - bootsector (rezervovaná oblast) + informace o svazku
  - 2x FAT
  - kořenový adresář
  - data
- adresáře jako soubory, kořenový svazek je vytvořen hned na začátku
- původní FAT nepodporoval adresáře
- FAT12, 16, 32: podle velikosti clusteru (max. kapacity - 32 MB, 2 GB, 8 TB)
- Virtual FAT
  - s Windows 95
  - podpora dlouhých jmen (LFN)
  - až 256 znaků
  - soubor má dvě jména - dlouhé a ve tvaru 8+3
- exFAT
  - určen pro flash paměti
  - podpora větších disků (512 TB, 64 ZB)
  - podpora v novějších Windows (původně Windows CE 6)

## **UFS: UNIX FILE SYSTEM**

- v různých variantách přítomných v unixových OS - BSD, Solaris, System V, Linux (ext[2,3,4])
- disk se skládá:
  - bootblock - místo pro zavaděč OS
  - superblock - informace o souborovém systému
  - místo pro inody
  - místo pro data

## **INODY**

- struktura popisující soubor
- informace o souboru
  - \* typ souboru, vlastníka (UID, GID), oprávnění (rwx)
  - \* časy (vytvoření, přístup)
  - \* počet ukazatelů, počet otevřených popisovačů
- informace o uložení dat
- struktura inody umožňuje mít řídké soubory
- adresář je soubor obsahující sekvenci dvojic (jméno souboru, číslo inody)
- k evidenci volného místa a inod se používají bitmapy



Obrázek 8: inode

## FS V LINUXU

- Linux nemá jeden hlavní FS
- nejčastěji se používá: Ext2/3/4
- název souboru může mít až 256 znaků
- vychází z UFS
- ext2: maximální velikost souboru 16 GB - 2 TB, disku: 2 TB - 16 TB
- ext3: přidává žurnál (3. úrovně - journal, ordered, unordered), binárně kompatibilní s ext2
- ext4: maximální velikost souboru 16 GB - 2 TB, disku 1 EB; optimalizace alokací

## NTFS

- hlavní souborový systém Windows NT
- kořeny v OS/2 a jeho HPFS (vyvíjen od roku 1993)
- velikost clusteru podle velikosti svazku (512 B - 4 KB) <- max. velikost disku 256 TB, max. velikost souboru 16 TB
- oproti FAT (souborovému systému W9x) ochrana před poškozením + práva
- žurnálování a transakce
- dlouhé názvy (255 znaků) + unicode
- podpora standartu POSIX; hardlinky, symlinky
- adresáře
  - opět technicky soubory
  - jména v B+ stromech
  - části metadat jsou součástí adresáře, část metadat je součástí souborů

## • STRUKTURA DISKU

- na začátku disku: boot sector
- 12% MFT (Maste File Table), 88% data souborů

- MFT je soubor popisující všechny soubory na FS (MFT je taky soubor)
- MFT se skládá ze záznamů o velikosti 1 KB
- každý soubor je popsán tímto záznamem
- informace o souborech včetně jména, časů, atd. uloženy jako záznam v MFT jako dvojice atribut-hodnota
- tělo souboru je taky atribut (uniformní přístup, možnost uložit malé soubory přímo do MFT
- v případě potřeby může jeden soubor zabrat více záznamů v MFT
- případně lze použít místo mimo MFT (rezidentní a nerezidentní atributy)

#### 2.5.4 Zajištění konzistence dat

### TRANSAKČNÍ PAMĚŤ

- rozdělení programu na části, které jsou provedeny "atomicky" (ACI)

```
void transfer(Account a1, Account a2, int amount) {
    atomic {
        a1.balance += amount;
        a2.balance -= amount;
    }
}
```

- změny se neprovádí přímo, ale ukládají se do logu, v případě "commitu" se ověří, jestli došlo ke kolizi
- zjednodušení vývoje - na úkor režie
- omezení: transakci musí být možné provést vícekrát

```
void transfer(Account a1, Account a2, int amount) {
    atomic {
        a1.balance += amount;
        a2.balance -= amount;
        launchTheMissiles();
    }
}
```

- podpora jako knihovny /rozšíření Javy/C# /C++

### ŽURNÁLOVÁNÍ

- data se zapisují v transakcích (přesun FS z jednoho konzistentního stavu do druhého)
- nejdříve se transakce zapíše do žurnálu (logu)
- po zapsání do žurnálu je záznam označen speciální značkou a data se můžou zapsat na disk
- po zapsání na disk je zápis z žurnálu odstraněn
- při připojení FS se kontroluje stavu žurnálu
  - zápis záznamu do žurnálu nebyl dokončen (transakce se nepovede)
  - případně, transakce se provede podle informací ze žurnálu
- často se žurnálují jen metada - v případu výpadku můžeme ztratit data, ale není narušena bezpečnost a není potřeba obnovovat strukturu FS
- žurnál je cyklický; při zaplnění se zapíší/uvolní ty na začátku

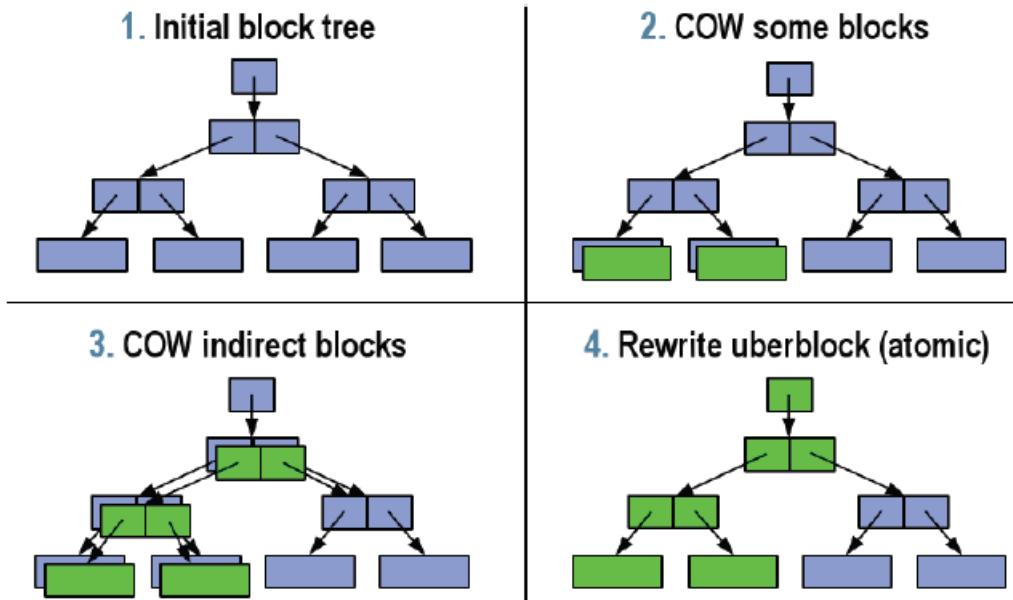
- pro transakce je potřeba atomických zápisů na disk
- cache a buffery komplikují implementaci atomických zápisů na disk

### **Copy-on-Write (CoW)**

- může být užitečné sdílet paměť (komunikace, úspora místa)
- dva stejné programy/knihovny v paměti
- stránky více procesů jsou navázány na jeden rámec
- daná stránka má nastavený příznak CoW
- dojde-li k pokusu o zápis, vznikne výjimka a procesu je vytvořena kopie rámce
- bude-li na rámci s příznakem CoW odkazovat jenom jedna stránka, příznak se odstraní
- fork() - vytvoří identickou kopii procesu; data jsou sice izolovaná, ale je možné sdílet kód
- úspora místa; úspora strojového času (není potřeba vytvářet kopie stránek/rámců); nízká penalizace pokud stránky přepíšeme
- virtuální paměť umožňuje další optimalizace (mapování souborů do paměti, atd.)

### **ZAJIŠTĚNÍ KONZISTENCE DAT**

- používání blokového zařízení: RAID (Redundant Array of Independent Disk)
  - RAID-0 (stripping) - nepomůže konzistenci
  - RAID-1 (mirroring)
  - RAID-2 Hammingův kód - dělí data po bitech, disk pro paritu
  - RAID-3 dělí data po bytech, XOR, disk pro paritu
  - RAID-4 podobné jako RAID-3, používá paritní bloky
  - RAID-5 podobné jako RAID-4, paritní bloky jsou distribuovány
  - RAID-6 podobné jako RAID-5, Reed-Solomon kód, dva paritní bloky
- u dat jsou evidovány kontrolní součty (ochrana proti tichému poškození (chyba HW i SW))
- konzistence založena na metodě Copy-on-Write
- používaná data nikdy nejsou přepsána (nejdřív jsou zapsána data a pak jsou (automaticky) změněna metadata)
- infrastruktura pro vytváření snapshotů/klonů souborového systému
- výhodně sloučovat operace do transakcí



1. alokujeme bloky pro data
2. pro kazda data vytvorime novy blok
3. vytvorime metadata, ktere predstavujuji bloky dat
4. po zapsani provedeme prohozeni těch částí, které jsou měněné

Obrázek 9: zachování konzistence

### 3

Klasifikace (LAN/MAN/WAN) a služby počítačových sítí. Síťová architektura TCP/IP a referenční model ISO OSI. Strukturovaná kabeláž, přepínaný Ethernet a WLAN. Protokol IP, adresa a maska, směrování, IP multicast. Protokoly TCP a UDP, správa spojení a řízení toku dat. Systém DNS, překlad jména (na IP adresu, reverzní), protokol. Služby WWW, elektronické pošty, přenosu souborů a vzdáleného přihlášení.

**POZOR!** packet = jakákoli naformátovaná zpráva jako packet vs. datagram = packet nespolehlivého přenosu dat.

#### 3.1 Klasifikace (LAN/MAN/WAN)

Klasifikace sítí podle různých kritérií: rozlehlosť, rychlosť prenosu (klasické, vysokorychlosťné), forma aplikácie, delenie podľa postavenia uzlov (peer-to-peer, klient-server), podľa druhu prenášenia signálu (analog vs. digital) aj.

##### 3.1.1 Delenie podľa rozlehlosťí

- **PAN (Personal Area Network)**: Síť s najmenšou rozlehlosťou. Propojenie mobilov, PDA, atď. Požiadavky na odolnosť voči rušeniu, nízká spotreba energie, snadná implementácia. Prenosová rychlosť není prioritou (zpravidla len niekoľko Mb/s). Dosah pouze niekoľko metrov. Typické technológie Bluetooth, IrDA, WiFi.
- **LAN (Local Area Network)**: Síť propojujúca koncové uzly typu počítač, tiskárna, server. V súkromnej správe, dosah jednotky km (v rámci budovy/komplexu budov). Prenosové rychlosť 10Mb/s až 1Gb/s. Súčasné využitie prenosového média. Ethernet, WiFi.

- **MAN (Metropolitan Area Network)**: Propojení několika LAN (účelem přenosové sítě, charakterem lokální). V rámci města (desítky km). Přenosové rychlosti jak vyšší (několik Gb/s), tak i nižší (<1Gb/s) ve srovnání s LAN.
- **WAN (Wide Area Network)**: Rozsáhlé sítě spojující LAN/MAN (páteřní sítě, telekomunikační - broadband). Velké vzdálenosti (prakticky neomezené). Mohou být soukromé i veřejné. Vysoké přenosové rychlosti (až stovky Gb/s). Příklad GPRS, xDSL, aj. Pronájem kapacity sítě = vyhrazené nesdílené využití přenosového média.

### 3.1.2 Dělení podle topologie

Topologie počítačové sítě říká, jak jsou vlastně prvky v této síti uspořádány.

**Kruhová topologie (RING)** Každý počítač je propojen přímo s předchozím a následujícím počítačem v kruhu. V LAN je používána velmi málo, používá se v průmyslových sítích nebo sítích MAN. Výhody:

- lehce rozšiřitelná struktura
- malý počet spojů
- snadné vysílání, zprávy v kruhu od stanice ke stanici

Nevýhody:

- výpadek libovolné stanice zapříčiní výpadek celé sítě, úplný výpadek sítě při přerušení kabelu v libovolném místě
- poměrně veliké nebezpečí odposlechu síťové komunikace, která prochází přes spojovací počítače

Problém výpadku se řešil tzv. dvojitým kruhem, ve kterém byly stanice propojeny dvěma kruhy, každý v opačné směru.

**Sběrnicová topologie (BUS)** Tato topologie patří k nejstarším, všechny stanice jsou připojeny na pasivní společné médium, které sdílejí. Dnes už se tato topologie příliš nepoužívá, ale na začátku devadesátých let byla dominantní. Tím společným médiem byl koaxiální kabel, pomocí kterého se jednotlivé počítače připojily do sítě.

Výhody:

- nezávislost stanic na výpadku libovolné jiné stanice
- levné náklady takového řešení
- neexistence aktivních prvků
- snadné všeobecné vysílání

Nevýhody:

- úplný výpadek sítě při přerušení kabelu v libovolném místě
- nutnost vyřešení přístupu stanic k médiu (kdo bude vysílat)

Výhody a nevýhody jsou relativní a poplatné době. To, že se v dobách používání této topologie v sítích LAN, považovala absence aktivních prvků za výhodu, bychom v dnešní době řadili spíše k nevýhodě.

**Hvězdicová topologie** Tato topologie je dnes jednoznačně nejpoužívanější topologií v sítích LAN. Myšlenka spočívá v tom, že existuje centrální prvek, který spojuje všechny prvky. Dříve tím centrálním prvkem býval počítač, dnes je aktivní prvek (HUB nebo SWITCH).

Výhody:

- lehce rozšiřitelná struktura
- výpadek libovolné stanice neznamená výpadek celé sítě
- větší možnosti zabezpečení, při použití aktivních prvků typu SWITCH je většina síťové komunikace skryta před ostatními účastníky sítě

Nevýhody:

- nutnost použití hubu nebo switche
- vyžaduje veliké množství kabelů a je tak náročná na montáž

**Páteřní topologie** Páteřní topologií rozumíme situaci, kdy pomocí určité topologie propojujeme celé síť LAN. Páteřní topologie může být zapojena jako sběrnice, hvězda i kruh, často se používá zapojení typu kruh. Jejím základem je vytvoření nezávislé hlavní části, která propojuje důležité celky. Na ni se naopak připojují různé subsítě nebo segmenty. V případě výpadku libovolného segmentu zůstává provoz na páteři neohrožen. Páteř může mít vyšší přenosovou rychlosť.

### 3.1.3 Služby počítačových sítí

- připojení k síti
- vzdálený přístup, sdílení výpočetních prostředků a přenos dat (sdílené databáze, peer-to-peer síť, sdílené soubory)
- sdílení technických prostředků (tiskárny, faxy, disky, apod.)
- adresářové služby (jednotný přístup do informačního systému a k informacím z centrální databáze, např. LDAP, Active Directory)
- elektronická pošta a výměna dokumentů (objednávky, faktury, atd.)
- online komunikace/multimedia (např. IRC, VoIP, straming, hry) - vysoké nároky na síť
- informační služby, internetové aplikace (WWW, business a desktopové aplikace)
- monitorování a vzdálená administrace sítě

Komunikace uzlů a propojovacích prvků sítě na různých úrovních:

- nižší - přenos bloků dat, většinou nespolehlivý (bez potvrzení a opakování přenosu), nespojová komunikace
  - **unicast**: dvoubodová, základní
  - **multicast**: bod-skupina, např. straming, virtuální síť
  - **broadcast**: bod-všichni, např. konfigurace a zapojení do sítě
- vyšší - komunikace aplikací, většinou spolehlivá (s potvrzením doručení, popř. opakování přenosu), spojově orientovaná (vytvořeno spojení mezi aplikacemi)
  - **peer-to-peer**: zpravidla rovnocenná výměna dat
  - **klient-server**: hierarchická, forma požadavek-odpověď

Typy koncových uzlů:

- **pracovní stanice** (work station, klient): převážně využívá služeb sítě

- **tenký klient**: znakový/grafický HW terminál, pouze zprostředkování vstupu a výstupu pro vzdálený server, nemůže pracovat samostatně
- **tlustý klient**: osobní počítač - klientské části síťových služeb i lokální úlohy, může (do určité míry) fungovat samostatně
- **server**: převážně poskytuje služby v síti souborový (FTP, NFS, SMB), databázový/adresářový (SŘBD, LDAP), poštovní (IMAP, POP3, SMTP), terminálový (telnet, SSH), informační/WWW (HTTP), komunikační (IM, VoIP), tiskový, aj.

## 3.2 Síťová architektura TCP/IP a referenční model ISO OSI.

Snaha o vytvoření univerzálního konceptu sítě (topologie, formy a pravidlakomunikace, poskytování služeb atd.). Požadavky: decentralizace služeb, rozumná adresace uzlů, data zasílána v nezávislých blocích, směrování bloků, zabezpečení, kontrola a řízení přenosu aj. Dříve proprietární uzavřená řešení, následně standartizace s koncepcí komunikace nezávislé na implementaci.

### Komunikace ve vrstvách

- definované službami poskytované vyšším vrstvám a využívající služby nižších vrstev, implementace skryté okolním vrstvám
- samostatné vrstvy s funkcemi podobnými v rámci vrstvy a odlišnými v různých vrstvách, nezávislé na implementaci

Komunikace mezi vrstvami pomocí **mezivrstvových protokolů** - na každé komunikující straně zvlášť; skrze **programová rozhraní**; prostřednictvím přístupových bodů; využívající tzv. služební primitiva.

### Obecná služební primitiva

- žádost o službu (request)
- oznamení poskytovatele o přijetí žádosti (indication) - nepovinné
- odezva poskytovatele (response), příp. vytvoření spojení
- potvrzení odezvy žadatelem (confirmation) - nepovinné

Komunikace ve stejných vrstvách mezi entitami (zařízeními) pomocí **vrstvových protokolů**.

### 3.2.1 Protokol

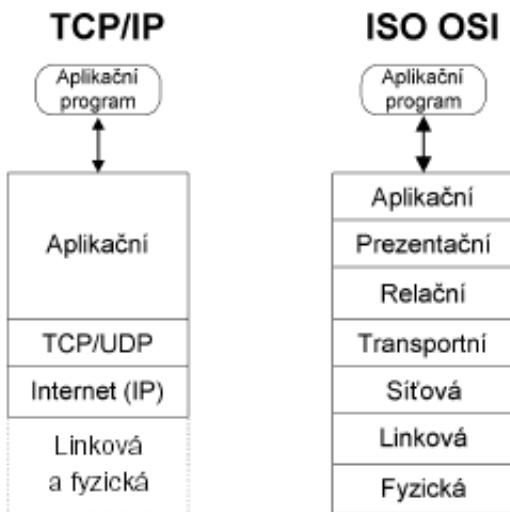
= souhrn pravidel (norem a doporučení) a procedur pro komunikaci (výměnu dat). Obsahuje syntaktická a sémantická pravidla výměny protokolových datových jednotek.

**Protokolové datové jednotky** = režijní informace a data (rámce, pakety, segmenty). Komunikace zprostředkována sousední nižší vrstvou. Na straně odesílatele **zapouzdřování** od nejvyšší po nejnižší vrstvu. Na straně příjemce **rozbalování** dat v opačném směru.

**Síťová (protokolová) architektura** = definice vrstev, služeb funkcí, protokolů a forem komunikace. Normalizované (OSI, TCP/IP) a firemní proprietární (Novell NetWare, SMB, Apple Appletalk aj.)

**Abstraktní referenční síťový model** Abstrakce konkrétních síťových architektur - nemusí podporovat všechny funkce modelu (např. průmyslové sítě nepodporují směrování, propojení pomocí mostů a bran).

### 3.2.2 Referenční model ISO/OSI



Propojení otevřených systémů = zařízení podporující příslušné normy. Deinovány koncové uzly (koncová datová zařízení) a mezipodlaží (propojovací prvky).

Každá ze sedmi vrstev vykonává skupinu jasně definovaných funkcí potřebných pro komunikaci. Pro svou činnost využívá služeb své sousední nižší vrstvy. Své služby pak poskytuje sousední vyšší vrstvě. Podle referenčního modelu není dovoleno vynechávat vrstvy, ale některá vrstva nemusí být aktivní. Takové vrstvy se říkají nulová, nebo transparentní.

Na počátku vznikne požadavek některého procesu v aplikační vrstvě. Příslušný podsystém požádá o vytvoření spojení prezentační vrstvy. V rámci aplikační vrstvy je komunikace s protějším systémem řízena aplikačním protokolem. Podsystémy v prezentační vrstvě se dorozumívají prezentačním protokolem. Takto se postupuje stále níže až k fyzické vrstvě, kde se použije pro spojení přenosové prostředí. Současně se při přechodu z vyšší vrstvy k nižší přidávají k uživatelským (aplikativním) datům záhlaví jednotlivých vrstev. Tak dochází k postupnému zapouzdřování původní informace. U příjemce se postupně zpracovávají řídící informace jednotlivých vrstev a vykonávají jejich funkce.

**Fyzická vrstva** Specifikuje fyzickou komunikaci. Aktivuje, udržuje a deaktivuje fyzické spoje mezi koncovými systémy. Definuje všechny elektrické a fyzikální vlastnosti zařízení (tvary konektorů; typy médií = kroucená dvojlinka, optické vlákno, mikrovlny; přenosové rychlosti). Obsahuje rozložení pinů, napěťové úrovně a specifikuje vlastnosti kabelů.

Funkce poskytované fyzickou vrstvou:

- Navazování a ukončování spojení s komunikačním médiem.
- Spolupráce na efektivním rozložení všech zdrojů mezi všechny uživatele.
- Modulace neboli konverze digitálních dat na signály používané přenosovým médiem (a zpět) (A/D, D/A převodníky).

HW zařízení: HUB, opakovač.

**Linková vrstva** Poskytuje spojení mezi dvěma sousedními systémy. Uspořádává data z fyzické vrstvy do logických celků = **rámce (frames)**: záhlaví s linkovou adresou příjemce a odesílatele (např. MAC) + data + zápatí s kontrolním součtem (CRC) celého rámce. Stará se o nastavení parametrů přenosu linky, oznamuje neoprávněné chyby. Formátuje fyzické rámce, opatřuje je fyzickou adresou a poskytuje synchronizaci pro fyzickou vrstvu. Příkladem je MAC u Ethernetu. Poskytuje propojení pouze mezi místně připojenými zařízeními a tak vytváří doménu na druhé vrstvě pro směrové a všeobecné vysílání. HW zařízení: most, přepínač, síťová karta, přístupový bod.

**Síťová vrstva** Stará se o směrování v síti a síťové adresování. Poskytuje spojení mezi systémy, které spolu přímo nesousedí. Poskytuje funkce k zajištění přenosu dat různé délky od zdroje k příjemci skrze jednu případně několik vzájemně propojených sítí při zachování kvality služby, kterou požaduje přenosová vrstva. Síťová vrstva poskytuje směrovací funkce a také reportuje o problémech při doručování dat. Jednotkou přenosu je **síťový packet**: záhlaví se síťovou adresou příjemce a odesílatele (např. IP) + data + zápatí jen výjimečně.

Funkce poskytované síťovou vrstvou:

- abstrakce různých linkových technologií
- správa linkových spojení, multiplexování síťových spojení do linkových (více datových toků kombinováno do jednoho)
- formátování dat do packetů
- směrování packetů
- zjišťování a oprava chyb
- vytváření podsítí

HW zařízení: směrovač, brána.

**Transportní vrstva** Zajišťuje přenos dat mezi koncovými uzly. Jejím účelem je poskytnout takovou kvalitu přenosu, jakou požadují vyšší vrstvy. Vrstva nabízí spojové (TCP) a nespojové orientované (UDP) protokoly. Jednotkou přenosu je **transportní packet**: záhlaví s transportní adresou příjemce a odesílatele + data.

**TCP**: Protokol garantuje spolehlivé doručování a doručování ve správném pořadí. TCP také umožňuje rozšiřovat a rozdělovat data pro více aplikací (například webový server a emailový server) běžících na stejném počítači. TCP využívá mnoho populárních aplikačních protokolů a aplikací na internetu, včetně WWW, e-mailu a SSH.

**UDP**: Na rozdíl od protokolu TCP nezaručuje, zda se přenášený datagram neztratí, zda se nezmění pořadí doručených datagramů, nebo zda některý datagram nebude doručen vícekrát. UDP je vhodný pro nasazení, které vyžaduje jednoduchost nebo pro aplikace pracující systémem otázka-odpověď (např. DNS, sdílení souborů v LAN).

Funkce poskytované transportní vrstvou:

- adresování (transportní na síťové)
- správa síťových spojení
- multiplexování a větvění
- rozdělení dat na datagramy, segmentace, formátování
- řízení proudu dat (správné pořadí datagramů), optimalizace služeb
- koncová detekce a oprava chyb

Umožňuje **duplexní přenos** (= přenos oběma směry).

**Relační vrstva** Smyslem vrstvy je organizovat a synchronizovat dialog mezi spolupracujícími relačními vrstvami obou systémů a řídit výměnu dat mezi nimi (např. sdílení síťového disku). Umožňuje vytvoření a ukončení relačního spojení, synchronizaci a obnovení spojení, oznamování výjimečných stavů. Jednotka přenosu je **relační packet**: poze data.

Funkce poskytované relační vrstvou:

- organizace a synchronizace dialogu výměny dat (pomocí kontrolních bodů)
- zobrazení relačních spojení do transportních
- správa transportních spojení

**Prezentační vrstva** Funkcí vrstvy je transformovat data do tvaru, který používají aplikace (šifrování, konvertování, komprimace). Formát dat (datové struktury) se může lišit na obou komunikujících systémech, navíc dochází k transformaci pro účel přenosu dat nižšími vrstvami. Vrstva se zabývá jen strukturou dat, ale ne jejich významem, který je znám jen vrstvě aplikační.

Funkce poskytované prezentační vrstvou:

- transformace a výběr reprezentace dat
- formátování, komprese, zabezpečení (šifrování), integrita dat
- transparentní přenos zpráv (nezná jejich význam)

**Aplikační vrstva** Účelem vrstvy je poskytnout aplikacím přístup ke komunikačnímu systému a umožnit tak jejich spolupráci.

Funkce poskytované aplikační vrstvou:

- zprostředkování funkcionality sítě
- přenos zpráv, určení kvality, synchronizace
- identifikace, stanovení pověření
- dohoda o ochraně, o opravách chyb

Protokoly: SMTP, SSH, Telnet, POP3, DHCP, FTP, DNS aj.

**Funkce společné více vrstvám** Komunikace **se spojením** má 3 fáze: navázání spojení, přenos dat, ukončení spojení. Dohoda na parametrech, použití potvrzování přijetí/nepřijetí (spolehlivost), stejné pořadí dat na vstupu i výstupu.

Komunikace **bez spojení**: při každém přenosu všechny parametry, nezávislý přenos datových jednotek, různé pořadí na vstupu a výstupu, spolehlivost i nespolehlivost.

Dále se může mezi těmito typy konvertovat. Transportní služby musí být se spojením.

### 3.2.3 TCP/IP

použití v síti Internet, nejpoužívanější síťová architektura. Síť tvořena směrovači, specializovanými bránami (bezpečnostní, aplikační, telekomunikační), lokálními sítěmi a koncovými zařízeními. Vlastní protokoly.

**Vrstva síťového rozhraní** Nejnižší vrstva umožňuje přístup k fyzickému přenosovému médiu. Je specifická pro každou síť v závislosti na její implementaci.

**Síťová vrstva** Vrstva zajišťuje především síťovou adresaci, směrování a předávání datagramů.

**Transportní vrstva** je implementována až v koncových zařízeních (počítačích) a umožňuje proto přizpůsobit chování sítě potřebám aplikace. Poskytuje transportní služby kontrolovaným spojením spolehlivým protokolem TCP nebo nekontrolovaným spojením nespolehlivým protokolem UDP.

**Aplikační vrstva** Vrstva aplikací. To jsou programy (procesy), které využívají přenosu dat po síti ke konkrétním službám pro uživatele.

Aplikační protokoly používají vždy jednu ze dvou základních služeb transportní vrstvy: TCP nebo UDP, případně obě dvě (např. DNS). Pro rozlišení aplikačních protokolů se používají tzv. porty, což jsou domluvená číselná označení aplikací. Každé síťové spojení aplikace je jednoznačně určeno číslem portu a transportním protokolem (a samozřejmě adresou počítače).

### 3.2.4 Bezpečnost

Obecné metody ochrany:

- omezování přenosu dat a přístupu k síti: blokace, filtrace
- autorizace přístupu: jméno a heslo, vícefaktorové, speciální protokoly
- zabezpečení kanálu: šifrování, výměna klíčů
- autenticita zpráv: digitální podpis, certifikáty

**TCP/IP** původně žádné zabezpečení, ponecháno na aplikaci.

Útoky:

- falešná adresace (spoofing)
- analýza hesla, trojský kůň
- odposlech
- odmítnutí služby (DoS, zahlcení, vyčerpání zdrojů)
- zneužití chyb aplikací
- ...

Ochrana:

- firewall (oddělení vnitřní sítě od vnější)
- překlad adres (NAT) - vlastní skrytá adresace
- aplikační brány (proxy)
- autentizace komunikujících stran
- zabezpečení komunikace (šifrování)
- opatření proti zahlcení
- ...

## 3.3 Strukturovaná kabeláž, přepínací ethernet a WLAN

### 3.3.1 Strukturovaná kabeláž

Obecný pojem pro metalické a optické prvky, které umožňují propojení.

**Telekomunikační přípojky** Slouží pro připojení koncových uživatelských zařízení - např. stolní počítač, notebook, analogový nebo ISDN telefon, VoIP telefon či síťová tiskárna. Telekomunikační zásuvky bývají umístěny přímo v pracovních prostorách (např. kancelářích) každé budovy, a to buď přímo ve zdi, v parapetních žlabech, případně podlahových systémech tak, aby byly lehce dostupné.

**Patch panely** Na rozdíl od běžně dostupných zásuvek jsou patch panely umístěny v rozvaděčích v telekomunikační místnosti a nejsou tedy pro běžné uživatele přístupné. Patch panely slouží správci sítě k připojení jednotlivých uživatelů do aktivních zařízení jako jsou switche nebo telefonní ústředny. Pro připojení vodičů do zárezových konektorů se používá narážecí nástroj.

**Horizontální kabely, Patch kabely** Měděné kabely obsahující čtyři kroucené páry, které vzájemně propojují telekomunikační zásuvky a patch panely.

**Koaxiální kabel** se dnes už nepoužívá. Vnější (stínění) a vnitřní ( jádro) vodič. BNC konektor. Maximálně 500m.

**Kroucená dvojlinka** je tvořena páry vodičů, které jsou po své délce pravidelným způsobem zkrouceny a následně jsou do sebe zakrouceny i samy výsledné páry. Oba vodiče jsou v rovnocenné pozici (i v tom smyslu, že žádný z nich není spojován se zemí či s kostrou), a proto kroucená dvojlinka patří mezi tzv. symetrická vedení. Maximálně 100m. Nestíněná (UTP) vs. stíněná (STP). Konektor RJ-45.

Jednou ze základních odlišností kroucené dvojlinky od koaxiálního kabelu je skutečnost, že na kroucené dvojlince není možné dělat odbočky. Kroucená dvojlinka je proto použitelná jen pro vytváření dvoubodových spojů. Nemožnost vytvářet odbočky pak ale nutně znamená, že prostřednictvím kroucené dvojlinky nelze vytvořit sběrnicovou topologii sítě.

**Optická vlákna** Vlákna imunní vůči elektromagnetickému rušení. Dvě vrstvy skla (obal a jádro). Vícevidové vlákno - odraz od rozhraní skel. Jednovidové vlákno - buzení laserem. Různé optické konektory (FC, LC, ST) nutné navařit. Vlákno simplexní, pro duplexní přenos dvojice vláken.

### 3.3.2 WLAN

= bezdrátové lokální sítě. Důvodem pro WLAN je rozšířitelnost, nižší náklady, mobilita, snadná použitelnost, roaming (vysílače si klienta předávají). Připojení od poskytovatele i v rámci LAN. Norma IEEE 802.11.

#### Konfigurace

- peer-to-peer/ad-hoc: přímá komunikace mezi stanicemi (do 10ti stanic)
- AP (access point): propojení WLAN a LAN, bezpečnostní prvky (autorizace, filtrace, šifrování, atd.)
- s více AP (roaming): AP propojeny pevnou sítí, klient se připojuje k zařízení s nejsilnějším signálem
- point-to-point: připojení pomocí dvou AP

Přenosové medium jsou radiové vlny (2,4 GHz - 802.11b/g/n nebo 5GHz - 802.11a/n) Není třeba licence, vzájemné rušení.

**antény:** omezení výkonu normou ČTÚ na 100 mW

#### Bezpečnost

- obtížná ochrana proti odposlechu na fyzické vrstvě
- SSID: označení AP (název sítě), AP jej nemusí vysílat
- WEP: Autentizace stanic vůči AP (40 bitů heslo společně s MAC adresou), lze v krátkém čase prolomit
- WPA, WPA2: silná šifra AES, autentizace (heslo, EAP)

### 3.3.3 přepínaný Ethernet

Norma IEEE 802.1: Propojení sítí na úrovni MAC, tvorva VLAN. Propojení LAN pomocí přepínačů a mostů, propojení WAN pomocí směrovačů.

**podvrstva LLC** je podvrstvou linkové vrstvy. Funguje jako rozhraní mezi síťovou vrstvou a podvrstvou Media Access Control (MAC). V dnešních protokolech linkové vrstvy plní často jen funkci multiplexování. LLC hlavička říká linkové vrstvě, co má provést s paketem, když je přijat datový rámec.

**Ethernet** Rámce se šíří segmentem po sdíleném mediu nezávisle na sobě, stanice (sítové rozhraní) "vidí" všechny, ale přijímá jen ty adresované jí nebo všeobecné. **Promiskuitní režim** (přijímá všechny rámce). Uzly rovnocenné, jen jeden v daný čas využívá sdílené medium pro vysílání rámciů (= half duplex). 10Gbit režim už bez sdíleného media (= full duplex).

**Protokol CSMA/CD** Kolizní přístup ke sdílenému médiu: Stanice naslouchá (Carrier) a vysílá, až nevysílá nikdo jiný. Když takto začne vysílat více stanic najednou, dojde ke kolizi. První stanice detekující kolizi vyšle signal **JAM** a všechny stanice se na náhodný čas odmlčí (čas odvozený od MAC adresy).

**Přepínaný Ethernet** využívá místo opakovače most. Normy IEEE 802.1d a 802.1q.

**Přepínač (Switch)** je multiportový most, který zpracovává příchozí rámce na svých rozhraních paralelně a vytváří souběžné virtuální linkové segmenty (dvojbodové plně duplexní spoje) propojující odesílatele s adresátem. Virtuální linkové segmenty jsou bezkolizní, kolize nastávají pouze u různých odesílatele se shodným adresátem. Pro přepínání používá **přepínací matici**. Dokáže spojit síť s různými rychlostmi (má vyrovňávací paměť, store-and-forward) a může hned po načtení hlavičky rámce načítat další (cut-through).

**Ethernet II** předepsaný pro lokální síť připojené přímo do internetu.

**MAC adresy:**

- globální (první tři bajty udávají výrobce), "zadrátované" v trvalé paměti sítové karty
- skupinová (nejnižší bit prvního bytu = 1)
- všeobecná (samé 1)

### 3.4 Protokol IP, adresa a maska, směrování, IP multicast

#### 3.4.1 Protokol IP

Poskytuje nespolehlivou nespojovou službu (nevytváří spojení, nepotvrzuje příjem packetů). Spojuje lokální síť do globální síť Internet. Tvořen několika dílčími protokoly: vlastní protokol IP, ICMP (diagnostika a signifikace mimořádných stavů), IGMP (skupinové adresování), ARP a RARP (zjištění linkové adresy k IP adrese a opačně). Sítové rozhraní uzlu má alespoň jednu IP adresu.

IP protokol byl původně vyvinut pro potřeby komunikace v Internetu. IP adresa musí být v dané síti jednoznačná (jedno rozhraní může mít více IP adres, ale stejná IP adresa nemůže být na více rozhraních), avšak lze používat NAT a privátní IP adresy.

**IP packet** záhlaví 20B povinných položek + volitelné položky, data, maximální délka 64kB

**Fragmentace** Linkové rámce mají omezenou velikost (max. jednotky kB). Maximální velikost = **MTU (Maximum Transfer Unit)**. IP packet může mít až 64kB -> fragmentace. Pokud je zakázána (DF), je packet zahoden (odesílateli signalizováno pomocí ICMP). Zvyšuje nároky režie, OS se snaží vytvářet packety menší než je MTU, aby k fragmentaci nedocházelo.

Fragmentace = dělení IP packetu na fragmenty o celkové délce menší, než MTU. **Freament** = samostatný IP packet se stejnou hlavičkou, jako původní. Skládání fragmentů do původního packetu provádí pouze příjemce. Lze fragmentovat fragmenty.

**Složení hlavičky packetu:** byty označující verzi (IPv4 vs IPv6), typ služby = ToS (nepoužíváno, v současnosti QoS), délka packetu, tag pomáhající k rekonstrukci packetu z více fragmentů, příznak DF (zakázání fragmentace) a MF (indikace dalších fragmentů), TTL (Time To Live - přes kolik routerů může projít, než bude zničen), byty označující protokol (ICMP, UDP, TCP, ...), kontrolní součet, zdrojová a cílová IP.

**doba života TTL** zamezení nekonečného "toulání" packetu. Každý směrovač snižuje alespoň o 1 (a musí tedy přepsat kontrolní součet v záhlaví). Při 0 se packet zahodí a odesílateli je to signalizováno pomocí ICMP.

### 3.4.2 IP adresa

Každé síťové rozhraní může mít jednu či více IP adres (jednozačných). Přidělení IP adresy staticky síťovému rozhraní pomocí ipconfig (MS Windows) nebo ifconfig/ip (UNIX, GNU/Linux).

**IPv4** Číslo o délce 4B. Notace v desítkovém tvaru oddělené tečkou po každém bytu (např. 127.0.0.1).

**Struktura adresy** Adresa se dělí na: adresu sítě, adresu podsítě a adresu počítače. Adresu sítě pro danou koncovou síť přiděluje poskytovatel připojení (oficiálně ji přiděluje lokální registrátor, ale tím bývá právě poskytovatel). Je třeba o ni požádat prostřednictvím standardních formulářů. Strukturu lokální části adresy – zda bude rozdělena na podsítě a jaká její část bude případně věnována adrese podsítě a jaká adrese počítače – určuje správce dotyčné sítě. Ten také přiděluje adresy.

**Třídy sítě** Historická záležitost, třídy A-E (rozdelení podle toho jaká část IP adresy určuje síť a jaká část počítač)

třída	1. bajt	minimum	maximum	maska podsítě
A	0–127	0.0.0.0	127.255.255.255	255.0.0.0
B	128–191	128.0.0.0	191.255.255.255	255.255.0.0
C	192–223	192.0.0.0	223.255.255.255	255.255.255.0
D	224–239	224.0.0.0	239.255.255.255	255.255.255.255
E	240–255	240.0.0.0	255.255.255.255	—

Například síť třídy A je taková síť, kde první číslo čtyřčíselné IP adresy označuje síť a zbylá tři čísla označují adresu hostitele. Třída B používá první dvě pro označení síťové adresy a zbývající dvě pro hostitele a síť třídy C používá první tři čísla pro označení sítě a poslední pro označení hostitele.

Postupem času se však i toto rozdelení ukazovalo jako velice nepružné a s rostoucím nedostatkem adres se hledaly způsoby na vylepšení původního systému.

**Síťová maska** Masku sítě zapsanou v binárním tvaru má zleva samé jedničky až do místa, kde končí číslo sítě a na místě části pro číslo síťového rozhraní jsou samé nuly. Masku sítě je v IPv4 zapisována stejně jako IP adresa – čtyřmi desítkovými čísly oddělenými tečkami, z nichž každé odpovídá jednomu oktetu v 32bitové adrese. Někdy se udává maska sítě zkrácenou formou zápisu, kterou označujeme CIDR (Classless Inter-Domain Routing, beztřídní mezidoménové směrování), ve kterém je možno v IP adrese hranici mezi číslem sítě a číslem počítače umisťovat libovolně (mezi dva libovolné byty). Adresa síťového rozhraní je pak zapisována pomocí IP adresy a délky prefixu (místo prefixu, který určuje délku čísla sítě v bitech lze použít i desítkový zápis masky sítě) (192.168.24.0/21 nebo 192.168.24.0/255.255.248.0). Protože má maska sítě na místě čísla sítě samé jedničky, můžeme z libovolné IP adresy určit číslo sítě pomocí logického součinu IP adresy a masky sítě. Pokud známe číslo sítě a masku, můžeme spočítat rozsah IP adres, které se v této podsítí nacházejí a tím i počet IP adres, které je možné použít pro síťová rozhraní v této podsítí.

Speciální adresy:

- celá adresa samé 0: tento uzel (loopback bez přidělené adresy)
- uzel 0: adresa sítě
- síť 0: uzel v síti (nepoužívá se)
- uzel samé 1: všešměrová adresa sítě (network broadcast)
- celá adresa samé 1: všešměrová adresa lokální sítě (local broadcast), nesměruje se
- 127.cokoliv: smyčka (loopback), většinou 127.0.0.1, odesílaný packet "ihned přijde"

**IPv6** adresa má délku 128 bitů. Zpisuje se jako osm skupin po čtyřech hexadecimálních číslicích (např. 2001:0718:1c01:0016:0214:22ff:fec9:0ca5). Úvodní nuly v každé skupině lze ze zápisu vynechat. Pokud adresa obsahuje několik po sobě jdoucích nulových skupin, lze místo nich zapsat jen "::". Tato zkratka smí být v adrese pouze jedna (např. loopback je ::1).

Tři typy adres:

- Individuální (unicast) která identifikují právě jedno síťové rozhraní.
- Skupinové (multicast) označují skupinu síťových rozhraní, jejímž členům se mají data dopravit. Skupinově adresovaný datagram se doručuje všem členům skupiny.
- Výběrové (anycast) označují také skupinu síťových rozhraní, data se však doručují jen jejímu nejbližšímu členovi.

IPv6 neobsahuje všeobecné (broadcast) adresy. Byly nahrazeny obecnějším modelem skupinových adres a pro potřeby doručení dat všem zařízením připojeným k určité síti slouží speciální skupinové adresy (např. ff02::1 označuje všechny uzly na dané lince). Zavádí se také koncepce dosahu (scope) adres. Adresa je jednoznačná vždy jen v rámci svého dosahu. Nejčastější dosah je pochopitelně globální, kdy adresa je jednoznačná v celém Internetu. Kromě toho se často používá dosah linkový, definující jednoznačnou adresu v rámci jedné linky (lokální sítě, např. Ethernetu).

**DHCP** Používá se pro automatickou konfiguraci počítačů připojených do počítačové sítě. DHCP server přiděluje počítačům pomocí DHCP protokolu zejména IP adresu, masku sítě, implicitní bránu a adresu DNS serveru. Platnost přidělených údajů je omezená, proto je na počítači spuštěn DHCP klient, který jejich platnost prodlužuje.

**Směrování** Chce-li počítač vyslat IP datagram, musí nejprve zjistit, do které sítě zadaná IP adresa náleží. Tuto činnost označujeme jako směrování IP datagramů. **Směrování (routing)** = odeslání packetů na další směrovač nebo cílový uzel (next hop). **Předávání (forwarding)** = předávání packetu v rámci směrovače mezi jeho síťovými rozhraními. Packety mohou být **filtrovány** nastavením filtračních pravidel v OS, na základě IP záhlaví, TCP/UDP záhlaví nebo aplikáčního protokolu.

Směrování probíhá podle směrovací tabulky, ve které je na každém řádku definována jedna síť (pomocí čísla sítě a masky). Tabulku vytváří administrátor **ručně** nebo vzniká **dynamicky** pomocí směrovacích protokolů. Tabulka je setříděna podle jednotlivých masek sítí tak, že záznamy s nejdélší maskou (tj. nejkonkrétnější) jsou ve směrovací tabulce umístěny nejdříve, což umožňuje ve směrovací tabulce vytvářet výjimky (z větší sítě vymout jako speciální případ menší podsítě). Zjišťování příslušnosti cílové IP adresy k síti (definované v řádku směrovací tabulky) probíhá tak, že se postupně pro každý řádek ve směrovací tabulce provede logický součin cílové IP adresy s maskou definované sítě a porovná se výsledek s definovaným číslem sítě. Pokud dojde ke shodě, patří IP adresa do sítě definované na řádku, kde ke shodě došlo.

**IP multicast** je metoda přeposílání IP datagramů z jednoho zdroje skupině více koncových stanic. Místo odesílání jednotlivých datagramů ke každému cíli je odeslán jedený datagram. K identifikaci jednotlivých multicastových skupin se používají IP adresy třídy D. Aby cílová stanice mohla přijímat multicastová data, musí být přihlášena alespoň do jedné skupiny a samozřejmě router sítě musí multicasting podporovat, udržovat tedy v sobě tabulkou skupin, které má odchytávat. Router, který přijímá informace z multicastových skupin, se od uzel snaží zjistit, které skupiny mají být vysílány uzlům do bezprostředně připojené sítě. Tuto službu nám zajišťuje IGMP protokol, díky kterému se uzly mohou přidávat do skupin.

**IGMP protokol** dynamicky registruje jednotlivé hostitele, patřící do skupiny adres D. Hostitel identifikuje členství ve skupině odesláním zpráv protokolu IGMP a data zasílá vždy všem členům skupiny. Směrovače používající protokol IGMP pravidelně naslouchají zprávám protokolu IGMP a systematicky odesílají dotazy s cílem zjistit, které skupiny jsou v síti LAN aktivní.

Každá IP adresa se v síti musí překládat na MAC adresu tedy i multicastová, právě díky této adrese se přenáší multicastová data v lokální síti. Zde však nastává problém v možné nejednoznačnosti, protože 48 bitová MAC adresa musí mít pro multicast prefix 01:00:5e následovaný nulovým bitem a zbylých 23 bitů MAC adresy je vyplněno posledními 23 bity IP adresy. Více stejně končícím IP adresám je tak přiřazena

stejná multicastová MAC adresa. Tento problém se někdy nazývá 32-to-1 overlapping, protože právě 32 multicastových IP adres je mapováno na jedinou multicastovou MAC adresu.

### 3.5 Protokoly TCP a UDP

#### Opakování:

- **spojová služba** - mezi aplikacemi navázáno spojení s plně duplexní výměnou dat, spolehlivá služba (ztracená a poškozená data znova vyžádána), integrita dat zajištěna kontrolním součtem, zpracovává souvislý prud/tok (uspořádaných) dat od vyšší vrstvy (stream)
- **nespojová služba** - nenavazuje spojení, nespolehlivá služba (nezaručuje pořadí doručených dat, znovuzaslání poškozených/ztracených), zpracovává nesouvislé části dat od vyšší vrstvy

**Port** = identifikátor aplikace (aplikace jich může používat více), transportní adresa, je to číslo o délce 2B (0 až 65535).

Rozdělení na:

- privilegované - může je používat pouze privilegovaná aplikace, rozsah 0 - 1023
- neprivilegované - může je použít kdokoliv, pokud je volný

Pro běžné služby jsou známá "standarní" čísla portů.

#### 3.5.1 Transmission Control Protocol (TCP)

TCP je spojově orientovaný protokol pro přenos toku bajtů na transportní vrstvě se spolehlivým doručováním. TCP využívá mnoho populárních aplikačních protokolů a aplikací na internetu, včetně WWW, e-mailu a SSH.

**TCP segment** Aplikace posílá proud (stream) bajtů TCP protokolu k doručení sítí, TCP rozděluje proud bajtů do přiměřeně velkých segmentů. (Velikost segmentů je určena parametrem **MTU (maximum transmission unit)** linkové vrstvy sítě, ke které je počítáč připojen.) TCP pak předá takto vzniklé pakety IP protokolu k přepravě internetem do TCP modulu na druhé straně TCP spojení. TCP ověří, že se pakety neztratily tím, že každému paketu přidělí **pořadové číslo**, které se také použije k ověření, že data byla přijata ve správném pořadí. TCP modul na straně příjemce posílá zpět potvrzení pro pakety které byly úspěšně přijaty. Pokud by se odesílateli potvrzení nevrátilo do rozumné doby (dané obousměrným zpožděním, anglicky round-trip time, **RTT**), vypršel by odesíatelův časovač a (pravděpodobně ztracená) data by vyslal znovu. TCP protokol ověřuje, zda přenesená data nebyla poškozena šumem tím, že před odesláním spočte **kontrolní součet**, uloží jej do odesílaného paketu a příjemce kontrolní součet vypočte znovu a ověří, že se shodují.

Příznaky:

- CWR, ECN - pro oznamení zahlcení sítě
- URG - segment nese urgentní data, která mají být zpracována přednostně
- ACK - signalizace správného pořadového čísla přijímaného bytu
- RST - odmítnutí navazovaného TCP spojení
- SYN - nová sekvence číselování odesílaných bytů, pořadové číslo odesílaného bytu je číslo 1. bytu toku dat, nastaven u 1. segmentu při navazování spojení
- FIN - ukončení odesílání dat, ukončení spojení pro daný směr přenosu dat

**Délka okna** je počet bytů, které je příjemce schopen přijmout - předcházení zahlcení sítě.

Protože TCP je spojovaná transportní služba, musí se před odesláním dat navázat spojení mezi klientem a serverem. K tomu slouží trojcestný handshaking (three-way handshake). Obě strany otevřou port (pomocí socketu), klient v **aktivním režimu**, server **pasivní režim** (očekávání spojení). V průběhu navazování spojení se obě strany dohodnou na číslu sekvence a potvrzovacím čísle. Pro navázání spojení se odesílají datagramy s nastavenými příznaky SYN a ACK.

### Navazování spojení:

- Klient odešle na server datagram s nastaveným příznakem SYN a náhodně vygenerovaným číslem sekvence ISN (x), potvrzovací číslo=0, maximální délka přijímaných segmentů (MSS).
- Server odešle klientovi datagram s nastavenými příznaky SYN a ACK, potvrzovací číslo=x+1, číslo sekvence ISN je náhodně vygenerované (y), navrhnutá MSS směrem od serveru.
- Klient odešle datagram s nastaveným příznakem ACK, číslo sekvence=x+1, číslo odpovědi=y+1.

Navrhované MSS je menší, než MTU, aby se zamezilo IP fragmentaci.

Obě strany si pamatují číslo sekvence své i protistrany. Používají se totiž i pro další komunikaci a určují pořadí paketů. Když úspěšně proběhne trojcestný handshaking, je spojení navázáno a zůstane tak až do ukončení spojení. To se může zneužít na SYN flood útok.

**Ukončování spojení:** probíhá podobně jako jeho navázání. Používá se k tomu příznaků FIN a ACK:

- Klient odešle datagram s nastaveným příznakem FIN
- Server odpoví datagramem s nastaveným příznakem ACK
- Server odešle datagram s nastaveným příznakem FIN
- Klient odpoví s nastaveným příznakem ACK

Tepřve po těchto čtyřech krocích je spojení ukončeno.

**Odmítnutí spojení:** Pokud cílový port na straně příjemce není otevřen (např. segmenty jsou zahazovány firewallem nebo neběží aplikace serveru), klient, bez odpovědi serveru, po čase opakuje požadavek na navázání spojení do vypršení celkového času nebo počtu pokusů = časová prodleva. Odmítnutí nastává kdykoliv po zaslání segmentu s příznakem RST. Například u neúspěšného navázání šifrovaného spojení SSL/TLS.

### 3.5.2 Řízení toku dat

#### Ztráta segmentu

- Odesílatel: Má definovaný časový interval pro příjem potvrzovacího segmentu od příjemce (retransmission timeout). Při ztrátě nebo poškození segmentu po vypršení intervalu nebo příjmu tří opakováných stejných potvrzení od příjemce **opakuje odesílání segmentu**. Hodnota intervalu se dynamicky mění podle stavu sítě na základě předpokládané doby odezvy (vypočítané z RTT).
- Příjemce: Má definovaný časový interval pro příjem následujícího segmentu v toku dat (dle pořadových čísel). Při neobdržení následujícího segmentu po vypršení časového intervalu nebo obdržení segmentu s dalšími daty mimo pořadí **opakuje potvrzení přijetí** předchozích dat. Ukládá si data obdržené mimo pořadí do bufferu a po obdržení chybějícího segmentu **potvrdí příjem všech dat**.

**Zpoždění odpovědi** je výhodné u interaktivních (konzolových) aplikací (Telnet, SSH, příkazový kanál FTP) vyměňujících malé segmenty. Ilustrační situace: Uživatel stiskne klávesu, klient odešle znak serveru (v segmentu IP packetu v linkovém rámci), server potvrdí příjem, zpracuje znak a odešle ho klientovi ke zobrazení, klient potvrdí příjem a zobrazí -> velká režie.

Zpoždění je tedy potvrzení příjmu dat se zpožděním a ne hned, kdy se mohou data k odeslání nahromadit. **Delayed ACK** je odesílání v intervalech (např. 200 ms (< jak 500 ms)). **Nagleův algoritmus**: odeslání dat až po obdržení dalších dat od druhé strany nebo až objem dat k odeslání překročí MSS. Lze vypnout pomocí síťového API OS příznakem TCP\_NODELAY.

**Posuvné okno (Sliding windows)** Využití při odesílání většího množství dat, zamezení zahlcení sítě. Segmenty se posílají bez potvrzování každého až do počtu odeslaných bytů rovno délce posuvného okna. Délka okna udává, kolik bytů je příjemce schopen přijmout či zpracovat. Při navazování spojení příjemce navrhne délku posuvného okna spolu s MSS (typicky 6-8x MSS) a pak ji může v potvrzovacích segmentech měnit nebo vynulovat (zakázat odesílateli odesílat další data, když "nestihá"). Potvrzováním příjmu dat se okno posouvá a mění velikost = řízení toku dat.

**Zahlcení sítě** Posuvné okno udává množství dat akceptované příjemcem. Pokud je příliš velké a síť na straně příjemce plně využita nebo pomalá, odesílatel může síť zahltit a ta začne data zahazovat. Řešením je **okno zahlcení** na straně odesílatele, které udává množství nepotvrzených dat, které je možné odeslat, aniž by došlo k zahlcení sítě (cíl je samozřejmě největší možné). Odesílatel pak odesílá data do menší z velikosti posuvného okna a okna zahlcení.

Dvě fáze určení velikosti okna zahlcení:

- **Pomalý start:** Od navázání spojení se velikost okna zahlcení s každým potvrzovacím segmentem zdvojnásobuje až do ztráty segmentu nebo dosáhnutí velikosti posuvného okna nebo parametru SSTHRESH (hranice pravděpodobnosti zahlcení sítě - typicky 64kB). Po třech stejných potvrzenech předchozího segmentu se okno zahlcení zmenší na polovinu a stejně nastaví SSTHRESH (ta však musí být minimálně 2xMSS). Po neobdržení potvrzení se okno zahlcení nastaví na MSS, SSTHRESH se nastaví na 2xMSS a začne se znova.
- **Předcházení/vyhýbání se zahlcení:** Následuje po pomalém startu, kdy se s každým potvrzením navýšuje. Algoritmy vyhýbání se zahlcení: Reno, New Reno, BIC, CUBIC, aj. SACK = selective ACK (selektivní potvrzování) je potvrzování segmentů i mimo pořadí pomocí volitelných položek záhlaví.

Pro každé spojení uživatel udržuje velikost MSS, posuvného okna, okna zahlcení a parametru SSTHRESH. SSTRESH se ukládá do směrovací tabulky a používá se jako výchozí hodnota pro nová spojení v témže směru.

Při ztrátě segmentu:

- po třetím stejném potvrzení se nastaví SSTHRESH na polovinu aktuálního okna zahlcení (minimálně 2xMSS), zopakuje se segment, nastaví se okno o něco výšší, než SSTHRESH a při opakování potvrzeních se zvyšuje o MSS
- po potvrzení ztraceného segmentu se nastaví okno na původní SSTHRESH a opět probíhá pomalé zvětšování
- po neobdržení potvrzení znova pomalý start (okno = MSS, SSTRESH = 2xMSS)

### 3.5.3 User Datagram Protocol (UDP)

Poskytuje nespojovou nespolehlivou službu: nezaručuje se doručení ani znovuzasílání ztracených nebo poškozených dat. Vyšší výkon a rychlosť než u TCP za cenu nespolehlivosti. Využití u streamování multimediálního obsahu.

Vlastní číslování portů (nezávisle na TCP). Velikost datagramu menší, než MTU aby se zamezilo fragmentaci. Oproti TCP může mít příjemcem skupina uzlů (vše směrová či multicast=skupinová). Záhlaví obsahuje informaci o délce dat a kontrolní součet. Protože UDP je bezestavový a odesílatel nemusí vyžadovat odpověď, zdrojový port je volitelný. Pokud není použit, zdrojový port by měl být nastaven na nulu.

Kvůli chybějícím zárukám se UDP aplikace musí smířit s nějakými ztrátami, chybami nebo duplikacemi. Některé aplikace mohou podle potřeby přidávat jednoduchý mechanismus spolehlivosti do aplikacní vrstvy. Aplikace používající UDP nejčastěji opravný mechanismus nepotřebují, a dokonce jím mohou být zdržovány. Pokud aplikace vyžaduje vysoký stupeň spolehlivosti, může se místo něj použít TCP nebo opravné kódy.

Protože UDP postrádá mechanismus předcházení a regulace zahlcení sítě, je nutné nadbytečné UDP datagramy na routerech zahazovat. Ačkoliv je celkové množství UDP provozu na typické síti jen v řádu procent, je UDP používán řadou klíčových služeb včetně DNS, SNMP, DHCP a RIP (směrovací protokol).

## Rozdíl mezi TCP a UDP

- TCP je spojově orientovaný protokol což znamená, že k navázání "end-to-end" komunikace potřebuje, aby proběhl mezi klientem a serverem tzv. "handshaking". Poté, co bylo spojení navázáno, data mohou být posílaná oběma směry.

Charakteristické vlastnosti TCP:

- spolehlivost – TCP používá potvrzování o přijetí, opětovné posílání a překročení časového limitu. Pokud se jakákoli data ztratí po cestě, server si je opětovně vyžádá. U TCP nejsou žádná ztracená data, jen pokud několikrát po sobě vyprší časový limit, tak je celé spojení ukončeno.
  - zachování pořadí – Pokud pakety dorazí ve špatném pořadí, TCP vrstva příjemce se postará o to, aby se některá data pozdržela a finálně je předala správně seřazená.
  - vyšší režie – TCP protokol potřebuje např. tři pakety pro otevření spojení, umožňuje to však zaručit spolehlivost celého spojení.
- UDP je jednodušší protokol založený na odesílání nezávislých zpráv.

Charakteristika:

- bez záruky – Protokol neumožňuje ověřit, jestli data došla zamýšlenému příjemci. Datagram se může po cestě ztratit. UDP nemá žádné potvrzování, přeposílání ani časové limity. V případě potřeby musí uvedené problémy řešit vyšší vrstva.
- nezachovává pořadí – Při odeslání dvou zpráv jednomu příjemci nelze předvídat, v jakém pořadí budou doručeny.
- jednoduchost – Nižší režie než u TCP (není zde řazení, žádné sledování spojení atd.).

### 3.5.4 Bezpečnost TCP a UDP

**TCP** Útoky: převzetí spojení (connection hijacking, autentizováno a dále nezabezpečeno), odepření služby (Denial of Service - vyčerpání zdrojů systému pro spojení), port scanning (zjišťování otevřených portů), aj.

Řešení: šifrování, vytvoření tunelů na jiných portech, omezení počtu spojení za daný čas, sledování skenování portů, aj.

**UDP** na směrovačích povolený porty 53/udp a 53/tcp pro DNS, vyplnění kontrolního součtu je nepovinné, jinak lze přepočítat.

**firewall** = filtrace packetů a segmentů/datagramů na základě TCP/UDP záhlaví.

**překlad adres (NAT)** překlad IP adresy z vnitřní sítě na IP adresu hraničního směrovače vnější sítě.  
**maškaráda** = překlad adresy a zdrojového portu spojení/přenosu na zdrojový port nového spojení/-přenosu ze směrovače. Zasahuje i do aplikacní vrstvy.

## 3.6 Systém DNS, překlad jména, protokol

### 3.6.1 Domain Name System (DNS)

hlavním úkolem a přičinou vzniku jsou vzájemné převody doménových jmen a IP adres uzlů sítě. Později ale přibral další funkce (např. pro elektronickou poštu či IP telefonii) a slouží dnes de facto jako distribuovaná databáze síťových informací. Protokol používá porty TCP/53 i UDP/53. Servery DNS jsou organizovány hierarchicky, stejně jako jsou hierarchicky tvořeny názvy domén. Systém DNS umožňuje efektivně udržovat decentralizované databáze doménových jmen a jejich překlad na IP adresy. Stejně tak zajišťuje zpětný překlad IP adresy na doménové jméno.

Protože je používání názvů pro člověka daleko příjemnější než používání číselných adres, vznikla už v dobách ARPANETu potřeba takový převod realizovat. Původně byl na všechny počítače distribuován jediný soubor (v Unixu /etc/hosts). Tato koncepce přestala velmi rychle vyhovovat potřebám a především nárokům na rychlou aktualizaci. Přesto se tento soubor dodnes používá, v závislosti na konfiguraci

systému je možné jej použít buď prioritně před dotazem na DNS nebo v případě, že DNS neodpovídá. Je možné jej také použít k uložení vlastních přezdívek pro často navštěvované servery, případně také pro blokování reklam a podobně.

Prostor doménových jmen tvoří strom s jedním kořenem. Každý uzel tohoto stromu obsahuje informace o části jména (doméně), které je mu přiděleno a odkazy na své podřízené domény. Kořenem stromu je tzv. kořenová doména, která se zapisuje jako samotná tečka. Pod ní se v hierarchii nachází tzv. domény nejvyšší úrovně (Top-Level Domain, TLD). Ty jsou buď tematické (com pro komerci, edu pro vzdělávací instituce atd.) nebo státní (cz pro Českou republiku, sk pro Slovensko, jo pro Jordánsko atd.).

Strom lze administrativně rozdělit do zón, které spravují jednotliví správci (organizace nebo i soukromé osoby), přičemž taková zóna obsahuje autoritativní informace o spravovaných doménách. Tyto informace jsou poskytovány autoritativním DNS serverem.

Výhoda tohoto uspořádání spočívá v možnosti zónu rozdělit a správu její části svěřit někomu dalšímu. Nově vzniklá zóna se tak stane autoritativní pro přidělený jmenný prostor. Právě možnost delegování pravomocí a distribuovaná správa tvoří klíčové vlastnosti DNS a jsou velmi podstatné pro jeho úspěch. Ve vyšších patrech doménové hierarchie platí, že zóna typicky obsahuje jednu doménu. Koncové zóny přidělené organizacím připojeným k Internetu pak někdy obsahují několik domén – například doména kdesi.cz a její poddomény výroba.kdesi.cz, marketing.kdesi.cz a obchod.kdesi.cz mohou být obsaženy v jedné zóně a obhospodařovány stejným serverem.

**Složení doménového jména** Celé jméno se skládá z několika částí oddělených tečkami. Na jeho konci se nachází domény nejobecnější, směrem doleva se postupně konkretizuje.

- část nejvíce vpravo je doména nejvyšší úrovně, např. wikipedia.org má TLD org.
- jednotlivé části (subdomény) mohou mít až 63 znaků a skládat se mohou až do celkové délky doménového jména 255 znaků. Doména může mít až 127 úrovní.

**DNS servery** DNS server může hrát vůči doméně (přesněji zóně, ale ve většině případů jsou tyto pojmy zaměnitelné) jednu ze dvou rolí:

- **Autoritativní server** je ten, na němž jsou trvale uloženy záznamy k dané doméně/zóně. Autoritativních serverů je obvykle více (minimálně dva - primární a sekundární, ale běžně i více) a až na velmi speciální případy se na všech udržují totožné záznamy, tzn. každou změnu v záznamech je potřeba propagovat na všechny autoritativní servery. Autoritativní DNS servery jsou obvykle provozovány registrátorem domény nebo poskytovatelem webhostingu.
- **Rekurzivní (caching only) server** je server, na který se se svými dotazy obracejí klientská zařízení (počítač, mobil aj.). Server pro ně příslušný záznam získá rekurzivními dotazy u autoritativních DNS serverů a po stanovenou dobu (definovanou pomocí parametru TTL) je má uloženy v cache, aby mohl odpovídat klientům rychleji a šetřil zatížení serverů autoritativních. Na tomto serveru nejsou žádné zóny uloženy trvale. Informaci o DNS serverech na dané síti klient zjišťuje nejčastěji přes protokol DHCP, na IPv6 přes NDP (DHCPv6 tuto informaci neposkytuje).

#### typy jmenných serverů:

- **primární** - jediný hlavní autoritativní server pro doménu/zónu, poskytuje autoritativní odpověď pro autoritativní záznamy ze své zóny a neautoritativní odpověď pro záznamy z cache
- **sekundární** - autoritativní pro svoji doménu/zónu, pravidelně kopíruje záznamy zóny (zone transfer) z primárního serveru, poskytuje stejně odpovědi jako primární server
- **caching only** - neautoritativní server pro doménu nebo zónu, poskytuje pouze neautoritativní odpovědi
- **kořenový** - primární server pro kořenovou doménu/zónu, je jich víc
- **forwarder** - server provádějící překlad pro jiný server (v roli klienta)

**Root servery** Kořenové jmenné servery (root name servers) představují zásadní část technické infrastruktury Internetu, na které závisí spolehlivost, správnost a bezpečnost operací na internetu. Tyto servery poskytují kořenový zónový soubor (root zone file) ostatním DNS serverům. Jsou součástí DNS, celosvětově distribuované databáze, která slouží k překladu unikátních doménových jmen na ostatní identifikátory. Kořenový zónový soubor popisuje, kde se nacházejí autoritativní servery pro domény nejvyšší úrovně. Tento kořenový zónový soubor je relativně velmi malý a často se nemění – operátoři root serverů ho pouze zpřístupňují, samotný soubor je vytvářen a měněn organizací IANA. Pojem root server je všeobecně používán pro 13 kořenových jmenných serverů. Root servery se nacházejí ve 34 zemích světa, na více než 80 místech. Root servery jsou spravovány organizacemi, které vybírá IANA.

### 3.6.2 Překlad jména

Každý koncový počítač má ve své konfiguraci síťových parametrů obsaženu i adresu lokálního DNS serveru, na nějž se má obracet s dotazy. V operačních systémech odvozených od Unixu je obsažena v souboru /etc/resolv.conf, v MS Windows ji najdete ve vlastnostech protokolu TCP/IP (případně můžete z příkazového řádku v XP zadat textový příkaz ipconfig /all). Adresu lokálního serveru počítač typicky obdrží prostřednictvím DHCP.

Pokud počítač hledá určitou informaci v DNS (např. IP adresu k danému jménu), obrátí se s dotazem na tento lokální server. Každý DNS server má ve své konfiguraci uvedeny IP adresy kořenových serverů (autoritativních serverů pro kořenovou doménu). Obrátí se tedy s dotazem na některý z nich.

Kořenové servery mají autoritativní informace o kořenové doméně. Konkrétně znají všechny existující domény nejvyšší úrovně a jejich autoritativní servery. Dotaz je tedy následně směrován na některý z autoritativních serverů domény nejvyšší úrovně, v níž se nachází cílové jméno. Ten je opět schopen poskytnout informace o své doméně a posunout řešení o jedno patro dolů v doménovém stromě. Tímto způsobem řešení postupuje po jednotlivých patrech doménové hierarchie směrem k cíli, až se dostane k serveru autoritativnímu pro hledané jméno, který pošle definitivní odpověď.

Získávání informací z takového systému probíhá rekurzí. Resolver (program zajišťující překlad) postupuje od kořene postupně stromem směrem dolů dokud nenaleze autoritativní záznam o hledané doméně. Jednotlivé DNS servery jej postupně odkazují na autoritativní DNS pro jednotlivé části jména.

Může se ale stát, že některý z oslovených serverů má hledanou informaci ve své vyrovnavací paměti, protože odpovídající dotaz nedávno řešil. V takovém případě poskytne neautoritativní odpověď z vyrovnavací paměti a další dotazování odpadá. Ve vyrovnavací paměti mohou být i mezivýsledky - například lokální DNS server v ní skoro jistě bude mít informaci o autoritativních serverech pro doménu org, protože v ní pravděpodobně hledá každou chvíli. Lokální server by se s dotazem rovnou obrátil na některý z autoritativních serverů domény org.

**Dva odlišné druhy chování** Při **rekurzivním řešení dotazu** se server chopí vyřízení dotazu, najde odpověď a pošle ji tazateli. Rekurzivní přístup server zatěžuje (musí sledovat postup řešení, ukládat si mezivýsledky apod.), ale projde jím odpověď a tu si může uložit do vyrovnavací paměti. Typicky se tak chovají lokální servery, aby si plnily vyrovnavací paměti a mohly dalším tazatelům poskytovat odpovědi rovnou. Při **nerekurzivním řešení dotazu** server dotaz neřeší, pouze poskytne tazateli adresy dalších serverů, jichž se má ptát dál. Takto se chovají servery ve vyšších patrech doménové hierarchie (kořenové a autoritativní servery TLD), které by rekurzivní chování kapacitně nezvládaly.

### 3.6.3 Reverzní dotazy

Nejběžnějším úkolem DNS je poskytnout informace (nejčastěji IP adresu) pro zadané doménové jméno. Dovede ale i opak – sdělit jméno, pod kterým je daná IP adresa zaregistrována. Při vkládání dat pro zpětné dotazy bylo ale třeba vyřešit problém s opačným uspořádáním IP adresy a doménového jména. Zatímco IP adresa má na začátku obecné informace (adresu síťe), které se směrem doprava zpřesňují až k adrese počítače, doménové jméno má pořadí přesně opačné. Instituce připojená k Internetu typicky má přidělen začátek svých IP adres a konec svých doménových jmen.

Tento nesoulad řeší DNS tak, že při reverzních dotazech obrací pořadí bajtů v adrese. K obrácené adrese pak připojí doménu in-addr.arpa a výsledné „jméno“ pak vyhledává standardním postupem. Hledá-li například jméno k IP adrese 145.97.39.155, vytvoří dotaz na 155.39.97.145.in-addr.arpa. Obrácení IP

adresy umožňuje delegovat správu reverzních domén odpovídajících sítím a podsítím správcům dotyčných sítí a podsítí. V příkladu použitou síť 145.97.0.0/16 spravuje nizozemský SURFnet a ten má také ve správě jí odpovídající doménu 97.145.in-addr.arpa. Kdykoli zavede do sítě nový počítač, může zároveň upravit data v reverzní doméně, aby odpovídala skutečné situaci.

Je dobré mít na paměti, že na data z reverzních domén nelze zcela spoléhat. Do reverzní domény se v principu dají zapsat téměř libovolná jména. Nikdo například nemůže zabránit SURFnetu, aby o počítači 145.97.1.1 prohlásil v reverzní zóně, že se jedná třeba o www.seznam.cz. Pokud na tom záleží, je záhadno si poskytnutou informaci ověřit normálním dotazem (zde nalézt IP adresu k www.seznam.cz a porovnat ji s 145.97.1.1). Jestliže odpověď na něj bude původní IP adresa, jsou data důvěryhodná – správce klasické i reverzní domény tvrdí totéž. Pokud se liší, znamená to, že data v reverzní doméně jsou nekorektní.

**DNS v praxi** Téměř každý DNS server funguje zároveň jako DNS cache. Při opakovaných dotazech pak nedochází k rekurzivnímu prohledávání stromu, ale odpověď je získána lokálně. V DNS záznamech je totiž uložena i informace jak dlouho lze záznam používat (TTL) a lze také zjistit, zda byl záznam změněn. Po vypršení platnosti je záznam z DNS cache odstraněn. Problém s uložením záznamů v cache nastává v případě změny záznamu. Pokud administrátor nastaví TTL na 6 hodin, a poté provede změnu záznamu, nastane situace, že někteří uživatelé sítě dostanou informaci již novou a někteří ještě starou. Tato situace bude trvat právě oněch 6 hodin, v závislosti na nastavení ostatních serverů a také v závislosti na době která uplynula od jejich posledního dotazu. Je proto nutné zvolit správný poměr mezi rychlosťí šíření změn a ušetřeným výkonem a přenosovým pásmem DNS serveru. Pokud se změny provádí často, je vhodné zvolit kratší TTL v řádu jednotek hodin, pokud se změny téměř neprovádějí, může být TTL ve dnech.

Obsah zóny (domény či několika domén) je uložen v tzv. zónovém souboru. Skládá se z jednotlivých záznamů (přesný název zní zdrojové záznamy, resource records, RR) obsahujících dílčí informace. Formát textového zápisu zónového souboru se liší v závislosti na použitém serveru, zde použijeme nejrozšířenější BIND. Záznamy v něm mají tvar 'jméno životnost třída typ parametry'.

**jméno** je doménové jméno, pro něž záznam vytváříte. Zpravidla patří do aktuálně definované domény, pak se příše jen samotné jméno bez tečky na konci a bude k němu doplněna aktuální doména. Pokud jméno ukončíte tečkou, nic se k němu nedoplňuje a bere se jako kompletní. Nemusí být uvedeno, pak se přebírá z předchozího záznamu.

**životnost** určuje dobu platnosti záznamu v sekundách. Většinou se neuvádí a záznamům se ponechává implicitní životnost. Umožňuje vám však udělat výjimku.

**třída** určuje rodinu protokolů, k níž se záznam vztahuje. DNS lze teoreticky používat i pro jiné síťové architektury, v praxi však třída vždy bývá IN, což znamená Internet.

**typ** určuje typ definovaného záznamu (viz níže).

**parametry** se vztahují k typu záznamu, poskytují mu potřebná data. Obsahem parametrů často bývají doménová jména. Je třeba zdůraznit, že v parametrech se smí vyskytovat jen skutečná jména, nikoli přezdívky zavedené pomocí CNAME.

Zónový soubor vždy musí začínat záznamem typu SOA.

### Typy záznamů:

- **A** (address record) obsahuje IPv4 adresu přiřazenou danému jménu, například když jménu cosi.kdesi.cz náleží IP adresa 1.2.3.4, bude zónový soubor pro doménu kdesi.cz obsahovat záznam **cosi IN A 1.2.3.4**
- **AAAA** (IPv6 address record) obsahuje IPv6 adresu. Zmíněnému stroji bychom IPv6 adresu 2001:718:1c01:1:02e0:7dff:fe96:daa8 přiřadili záznamem **cosi IN AAAA 2001:718:1c01:1:02e0:7dff:fe96:daa8**
- **CNAME** (canonical name record) je alias - jiné jméno pro jméno již zavedené. Typicky se používá pro servery známých služeb, jako je například WWW. Jeho definice pomocí přezdívky umožňuje jej později snadno přestěhovat na jiný počítač. Pokud náš cosi.kdesi.cz má sloužit zároveň jako www.kdesi.cz, vložíme do zónového souboru **www IN CNAME cosi**
- **NS** (name server record) ohlašuje jméno autoritativního DNS serveru pro danou doménu. Bude-li mít doména kdesi.cz poddoménu obchod.kdesi.cz, jejímiž servery budou ns.kdesi.cz (primární) a

ns.jinde.cz (sekundární), bude zónový soubor pro kdesi.cz obsahovat **obchod IN NS ns a IN NS ns.jinde.cz**.

- **PTR** je speciální typ záznamu pro reverzní zóny. Obsahuje na pravé straně jméno počítače přidělené adrese na straně levé (adresa je transformována na doménu výše popsaným postupem). Držme se našeho příkladu pro záznam typu A - v souladu s ním by zónový soubor pro doménu 3.2.1.in-addr.arpa měl obsahovat (zónový soubor definuje reverzní doménu, proto je třeba psát na pravé straně kompletní jméno s tečkou, jinak by za ně připojil reverzní doménu) **4 IN PTR cosi.kdesi.cz**.
- **SOA** (start of authority record) je zahajující záznam zónového souboru. Obsahuje jméno primárního serveru, adresu elektronické pošty jejího správce (zavináč je v ní ale nahrazen tečkou) a následující údaje:
  - Serial — sériové číslo, které je třeba zvětšit s každou změnou v záznamu. Podle něj sekundární server pozná, že v doméně došlo ke změně. Pokud jej zapomenete zvětšit, rozejde se obsah sekundárních serverů s primárním, což rozhodně není dobré. Pro přehlednost často ve formátu YYYYMMDDHH.
  - Refresh — jak často se má sekundární server dotazovat na novou verzi zóny (v sekundách).
  - Retry — v jakých intervalech má sekundární server opakovat své pokusy, pokud se mu nedaří spojit s primárním.
  - Expire — čas po kterém označí sekundární servery své záznamy za neaktuální, pokud se jim nedaří kontaktovat primární server.
  - TTL — implicitní doba platnosti záznamů.

Časové údaje jsou v sekundách. Novější implementace umožní pro vyšší pohodlí používat k číslům přípony 'M', 'H', 'D' a 'W' (minuta, hodina, den, týden) – například 8h znamená 8 hodin, čili totéž co hodnota 28800.

#### 3.6.4 Protokol DNS

Aplikační protokol pracující na bázi dotaz-odpověď poskytující službu typu klient-server (klient pošle dotaz, server odpoví). Základní operace **DNS Query** pro získání informací z DNS databáze na serveru. Používá TCP i UDP, pro oba port 53 (stejný pro dotaz i odpověď). Pro běžné dotazy typu překlad adres používá UDP (kvůli režii TCP), odpověď případně zkrácena na 512B kvůli fragmentaci. Kompletní odpověď nebo zone transfer pomocí TCP. Protokol není zcela spolehlivý (kvůli využívání UDP). Pro různé DNS operace různé DNS packety (neobsahují kontrolní součet - přenechán na UDP datagram).

**DNS Query** = základní operace protokolu DNS: dotaz (klienta) a odpověď (serveru) s informacemi (záznamy) podle požadavků v dotazu nebo negativní (záznam podle požadavků neexistuje). Stejný formát DNS packetu pro žádost i odpověď. Packet obsahuje záhlaví (povinné), dotazy, odpovědi, autoritativní jmenné servery a doplňující informace.

- **záhlaví (HEADER)** v dotazu i odpovědi
  - ID - identifikátor stejný v dotazu i odpovědi pro spárování
  - QR - 0 pro dotaz, 1 pro odpověď
  - Opcode - typ dotazu
  - AA - 1 pro autoritativní odpověď
  - TC - pro zkrácenou odpověď (na 512B)
  - RD, RA - možnosti rekurzivního překladu
  - AD - zabezpečená odpověď
  - Rcode - kód odpovědi (0 bez chyby, 1 chyba formátu dotazu, 2 neschopnost odpovědi, 3 negativní odpověď, 5 odmítnutí odpovědi, atd.)

- **dotazy (QUESTION)** - většinou jeden záznam, v dotazu i odpovědi (zopakován)
- **ANSWER** - odpověď s požadovanými záznamy
- **AUTHORITY** - seznam autoritativních serverů
- **ADDITIONAL**

**Komprese DNS packetu** další výskytu (části) doménového jména v packetu jsou nahrazeny odkazem na první výskyt.

**Inverzní dotaz** (Opcode = 1): jako reverzní, ale pro odpověď se místo záznamu typu PTR používají záznamy typu A, nemusí být servery podporovány.

### 3.6.5 Zabezpečení DNS

**DNSsec** = zabezpečení záznamů na jmenných serverech a v DNS packetech, použití el. podpisu (soukromým klíčem subdomény/zóny podepsány všechny její záznamy), RRSIG pro ověření integrity DNS packetu. Ověření el. podpisů (veřejný klíč DNSKEY, podepsaný soukromým klíčem nadřízené domény). Možnost uložit certifikáty pro aplikace jako záznam CERT. **Nevýhody:** Je třeba podepsat každý packet.

**TSIG (Transaction Signatures)** = autorizace komunikace DNS serverů. MD5 hash přenášených záznamů a sdíleného tajemství v záznamu typu TSIG. Použití u DNS Update - může jen autorizovaný server.

**Registrace domény** Internet Corporation for Assigned Names and Numbers (ICANN) je organizace, která má na starost přidělování a správu doménových jmen a IP adres. Zastřešuje také další regionální organizace, které působí na jednotlivých kontinentech. Každý stát má potom určeného správce zóny, který se stará o příslušnou TLD. Správce buď může domény registrovat sám nebo prostřednictvím tzv. doménových registrátorů, kteří mají náležitá oprávnění. ICANN zveřejňuje kompletní seznam všech TLD správců. Informace o vlastnících domén jsou udržovány v online databázi, která je dostupná přes službu WHOIS. Doménové registry obsahují informace o více než 240 národních a generických doménách (ccTLD, gTLD). Např. v ČR se o doménu .cz stará CZ.NIC z.s.p.o. **Doména musí být volná.**

## 3.7 Služby WWW, elektronické pošty, přenosu souborů a vzdáleného přihlášení

### 3.7.1 DHCP

DHCP protokol umožňuje prostřednictvím DHCP serveru nastavovat stanicím v počítačové síti sadu parametrů nutných pro komunikaci pomocí IP protokolu (tj. využívat rodinu protokolů TCP/IP). Umožňuje předávat i doplňující a uživatelsky definované parametry.[1] Významným způsobem tak zjednodušuje a centralizuje správu počítačové sítě (například při přidávání nových stanic, hromadné změně parametrů[2] nebo pro skrytí technických detailů před uživateli). DHCP servery mohou být sdruženy do skupin, aby bylo přidělování adres odolné vůči výpadkům. Pokud klient některým parametry nerozumí, ignoruje je.

Typicky se pomocí DHCP nastavují tyto parametry:

- IP adresa
- maska sítě
- implicitní brána (default gateway)
- DNS server
- další údaje např. pro NTP, WINS,...

**Princip činnosti** Klienti žádají server o IP adresu, ten u každého klienta eviduje půjčenou IP adresu a čas, do kdy ji klient smí používat (doba zapůjčení, anglicky lease time). Poté co vyprší, smí server adresu přidělovat jiným klientům.

Klient komunikuje na UDP portu 68, server naslouchá na UDP portu 67. Po připojení do sítě klient vysíle broadcastem DHCPDISCOVER paket. Na ten odpoví DHCP server paketem DHCPOFFER s nabídkou IP adresy. Klient si z (teoreticky několika) nabídek vybere jednu IP adresu a o tu požádá paketem DHCPREQUEST. Server mu ji vzápětí potvrdí odpověď DHCPACK. Jakmile klient obdrží DHCPACK, může už IP adresu a zbylá nastavení používat. Klient musí před uplynutím doby zapůjčení z DHCPACK obnovit svou IP adresu. Pokud lhůta uplyne aniž by dostal nové potvrzení, klient musí IP adresu přestat používat.

Protokol definuje roli i tzv. DHCP relay agenta. Používá se v situaci, kdy existují dvě nebo více sítí oddělené směrovačem a jen jedna síť obsahuje DHCP server. V takovém případě správce na směrovači zapne relay agenta a nastaví jej tak, aby všechny (broadcast) DHCP dotazy ze sítí bez DHCP serveru přesídlil DHCP serveru. Agent k přesílanému dotazu přidá číslo sítě a masku sítě, na které klienta zaslechl, aby DHCP server poznal, ze kterého adresního rozsahu má klientovi adresu přiřadit.

IP adresa může být stanici přidělena několika způsoby:

- Ruční nastavení - nevyužívá se DHCP serveru a konfigurace se zapisuje ručně do stanic
- Statická alokace - DHCP server obsahuje seznam MAC adres a k nim příslušných IP adres. Pokud je žádající stanice v seznamu, dostane vždy přidělenou stejnou pevně definovanou IP adresu
- Dynamická alokace - Správce sítě na DHCP serveru vymezí rozsah adres, které budou přidělovány stanicím, které nejsou registrovány. Časové omezení pronájmu IP adresy dovoluje DHCP serveru již nepoužívané adresy přidělovat jiným stanicím. Registrace dříve pronajatých IP adres umožňuje DHCP serveru při příštém pronájmu přidělit stejnou IP adresu.

### 3.7.2 Elektornická pošta

je způsob odesílání, doručování a přijímání zpráv přes elektronické komunikační systémy. Termín e-mail se používá jak pro internetový systém elektronické pošty založený na protokolu SMTP (Simple Mail Transfer Protocol), tak i pro intranetové systémy, které dovolují posílat si vzájemně zprávy uživatelům uvnitř jedné společnosti nebo organizace (tyto systémy často používají nestandardní protokoly, mívají ovšem bránu, která jim dovoluje posílat a přijímat e-maily z internetu). K širokému rozšíření e-mailu přispěl zejména internet. Pro e-mail je definován přenos 7bitové ASCII informace. Přesto je většina e-mailových přenosů 8bitových, kde ale nelze zaručit bezproblémovost. Z toho důvodu byla elektronická pošta rozšířena o standard MIME, aby bylo umožněno kódování vkládaných HTML a binárních příloh, obrázků, zvuků a videí.

**Komunikační protokoly** Mezi počítači na internetu se vyměňují zprávy pomocí Simple Mail Transfer Protocol a softwaru typu MTA jako např. Sendmail.

Uživatelé mívají na svém počítači nainstalován program, který se nazývá e-mailový klient. Ten stahuje zprávy z poštovního serveru použitím protokolů POP nebo IMAP, avšak v prostředí velkých společností se stále vyskytuje použití některého komerčního protokolu jako např. Lotus Notes nebo Microsoft Exchange Server.

Je možné ukládat e-maily buď na straně serveru, nebo na straně klienta. Standardní formáty pro mailové schránky jsou např. Maildir a mbox. Několik e-mailových klientů používá vlastní formát a na konverzi mezi těmito formáty je potřebný speciální program.

Některí uživatelé nepoužívají e-mailového klienta, ale přistupují ke zprávám umístěným na poštovním serveru přes webové rozhraní. Tento postup se často používá zejména u freemailových (bezplatných) služeb.

**Doručování** Při posílání pošty přes internet má být zaručen spolehlivý přenos zprávy i v případě dočasného výpadku cílového serveru. Zpráva se obvykle píše v prostředí programu typu e-mailového klienta nebo v obdobném formuláři webového rozhraní. Klient pomocí Simple Mail Transfer Protocol (SMTP) pošle zprávu programu Mail Transfer Agent (MTA), například smtp.a.org, který může běžet buď na samostatném smtp poštovním serveru, nebo i přímo na počítači odesílatele. Program MTA zjistí

z uvedených cílových adres název domény (část adresy za zavináčem) a tyto domény vyhledá v Domain Name System (DNS), aby zjistil mail exchange servery přijímající poštu pro danou doménu. DNS server domény b.org, tedy ns.b.org, odpoví MX záznamem, kde uvede mail exchange server pro danou doménu. MTA server (např. smtp.a.org) odešeď zprávu na mail exchange server (např. mx.b.org) pomocí protokolu SMTP. Domény obvykle mají záložní (backup) mail exchange server, takže můžou pokračovat v přijímaní pošty, i když je právě nedostupný hlavní mail exchange server. Když není možné zprávu doručit, MTA příjemce o tom musí odeslat zpět odesílateli zprávu (bounce message), ve které ukazuje na problém. Mail exchange server zprávu doručí do schránky adresáta. Ze schránky adresáta si zprávu stáhne pomocí protokolu POP (POP3) nebo IMAP nebo ji adresátovi umožní prohlédnout poštovní klient příjemce nebo webová služba. Bývalo zvykem, že kterýkoliv MTA přijímal zprávy pro kteréhokoli uživatele na internetu a udělal, co se dalo, aby zprávu doručil. Takové MTA se nazývají open mail relays. To bylo důležité v začátcích internetu, když byla síťová spojení nespolehlivá a nepermanentní. Když MTA nemohl doručit zprávu do cíle, mohl ji alespoň poslat agentovi bližšímu k cíli. Ten by měl větší šanci ji doručit. Ukázalo se však, že tento mechanizmus byl zneužitelný lidmi posílající nevyžádanou hromadnou poštu (spam), a proto velmi málo ze současných MTA jsou open mail relays (tj. přijímají poštu pro známé uživatele resp. domény). Prakticky všechny open relays jsou rychle odhalené a zneužité spamery, kteří neustále prohledávají (skenují) IP rozsahy celého internetu.

**E-mailová adresa** Každý uživatel musí mít pro příjem zpráv svoji e-mailovou adresu, která identifikuje jeho elektronickou poštovní schránku. Ta je fyzicky umístěna na nějakém internetovém serveru, populární jsou zejména servery, které nabízejí e-mailovou schránku zdarma a s webovým rozhraním (např. GMail, Seznam.cz).

**Hlavička e-mailové zprávy** obvykle obsahuje alespoň 4 pole:

- Od (From): e-mailová adresa (popř. i jméno) odesílatele zprávy (zpravidla vyplňuje program automaticky)
- Komu (To): e-mailová adresa (popř. i jméno) příjemce zprávy, adresátů může být více současně (vyplňuje odesílatel)
- Předmět (Subject): stručný popis obsahu zprávy (vyplňuje odesílatel, nepovinně)
- Datum (Date): místní datum a čas odeslání zprávy (vyplňuje program automaticky)

Informace v hlavičce na počítači příjemce je podobná záhlaví na konvenčním dopisu – skutečná informace o tom, komu byla zpráva adresována, je odstraněná mailovým serverem potom, co je přiřazená správné mailové schránce. Pole „Od“ nemusí obsahovat adresu skutečného odesílatele. Je velmi jednoduché to zfalšovat a zpráva potom vypadá, že přišla z uvedené adresy. Je možné e-mail digitálně podepsat, aby bylo jisté, od koho zpráva pochází.

Další běžné součásti hlavičky:

- Kopie (Cc): carbon copy – kopie (carbon proto, že psací stroje používají „kopírák“ (carbon paper) k vytvoření kopíí dopisů) (vyplňuje odesílatel, nepovinná položka)
- Slepá čili skrytá kopie (Bcc): blind carbon copy – neviditelná kopie (adresát bude vidět osoby uvedené v poli „Komu“ a „Cc“, ale ne adresy v „Bcc“) (vyplňuje odesílatel, nepovinná položka)
- Received: přijato – trasové informace vytvořené servery, kterými zpráva prošla (automatické zápis y serverů)
- Content-type: druh obsahu – informace o tom, jak má být zpráva zobrazena, obvykle MIME typ (automatický zápis)

**MIME**, plným názvem Multipurpose Internet Mail Extensions („Víceúčelová rozšíření internetové pošty“), je internetový standard, který umožňuje pomocí elektronické pošty zasílat zprávy obsahující text s diakritikou, lze k ní přiložit přílohu v nejrůznějších formátech, umožňuje funkci digitálního podpisu apod.

**SMTP** je internetový protokol určený pro přenos zpráv elektronické pošty (e-mailů) mezi přepravci elektronické pošty (MTA). Protokol zajišťuje doručení pošty pomocí přímého spojení mezi odesílatelem a adresátem; zpráva je doručena do tzv. poštovní schránky adresáta, ke které potom může uživatel kdykoli (off-line) přistupovat (vybírat zprávy) pomocí protokolů POP3 nebo IMAP. Jedná se o jednu z nejstarších aplikací. SMTP funguje nad protokolem TCP, používá port TCP/25.

**POP3** je internetový protokol, který se používá pro stahování emailových zpráv ze vzdáleného serveru na klienta. Jedná se o aplikační protokol pracující přes TCP/IP připojení. V současné době používají téměř všichni uživatelé elektronické pošty pro stahování emailů programy využívající POP3 nebo IMAP. Ze vzdáleného serveru se stáhnou všechny zprávy, třeba i ty, které uživatel číst nechce, nebo spam (pokud ho již nefiltruje poštovní server). Většina POP3 serverů sice umožňuje stáhnout i pouze hlavičky zpráv (a následně vybrat zprávy, které se stáhnou celé), ale podpora v klientech vesměs chybí. Tuto nevýhodu může odstranit protokol IMAP, který pracuje se zprávami přímo na serveru. Pro odesílání zpráv se používá protokol SMTP, nezávisle na použitém protokolu pro příjem pošty.

Protokol POP3 má pro své účely vyhrazen TCP port 110. Komunikace probíhá na principu výměny zpráv mezi klientem a serverem. Příkaz vždy začíná na začátku řádky, v základní implementaci POP3 mají příkazy 3 nebo 4 znaky. Příkazy nerozlišují velká a malá písmena. Za příkazem můžou následovat další argumenty, oddělené mezerami. Řádky jsou oddělovány pomocí CRLF. Každá odpověď od serveru musí začínat indikaci stavu operace - buď +OK, nebo -ERR. Následovat může textový řetězec s popsaným důvodem stavu. POP3 implementace jsou často poměrně komunikativní a dají se užívat i „ručně“.

**IMAP** je internetový protokol pro vzdálený přístup k e-mailové schránce prostřednictvím e-mailového klienta. IMAP nabízí oproti jednodušší alternativě POP3 pokročilé možnosti vzdálené správy (práce se složkami a přesouvání zpráv mezi nimi, prohledávání na straně serveru a podobně) a práci v tzv. on-line i off-line režimu. Protokol IMAP umožňuje trvalé (tzv. on-line) připojení k e-mailové schránce. Díky tomu je možné s celou poštovní schránkou plně pracovat z libovolného místa. Všechny zprávy a složky jsou uloženy na poštovním serveru a na počítač se stahují jen nezbytné informace, takže při zobrazení složky se stáhnou jen záhlaví zpráv a jejich obsah až v případě, že zprávu chce uživatel přečíst. U jednotlivých zpráv se uchovává jejich stav (nepřečtená, odpovězená, důležitá), uživatel může zprávy přesouvat mezi složkami, složky vytvářet, mazat, prohledávat na straně serveru apod. Protokol umožňuje současné připojení více klientů zároveň.

Oproti protokolu POP3 je IMAP4 velmi komplikovaný protokol. Jeho implementace je značně složitější a tedy i náchylnější k chybám než implementace POP3. Navzdory tomu IMAP používá mnoho e-mailových serverů a klientů jako jejich standardní přístupovou metodu.

**Poštovní klient** je program, který zajišťuje odesílání zpráv a vybírání schránek. Příkladem je např. Microsoft Outlook, Mozilla Thunderbird, Opera, Mutt, Pine a další. Je to v podstatě specializovaný editor, který umí kromě vytvoření zprávy také manipulovat se schránkami, odeslat zprávu nejbližšímu MTA a převzít zprávu ze serveru prostřednictvím POP3 nebo IMAP. Vlastním doručováním zprávy po sítí až k adresátovi se klient nezabývá. Součástí klienta bývá také více či méně složitý adresář, který pomáhá uživateli udržet přehled o adresách.

**Poštovní server** (MTA) běží obvykle jako démon či Služba Windows a naslouchá na portu TCP/25. K tomuto portu se může připojit (navázat TCP spojení) buď poštovní klient, nebo jiný server, který předá zprávu k doručení. MTA zkонтroluje, zda je zpráva určena pro systém, na kterém běží. Pokud ano, předá ji programu MDA (lokální doručení). Pokud je zpráva určena jinému počítači, naváže spojení s příslušným serverem a zprávu mu předá. Při vyhledávání vzdáleného serveru, kterému má předat zprávu, musí MTA spolupracovat se systémem DNS. Od serveru DNS si vyžádá tzv. MX záznam pro cílovou doménu, který obsahuje IP adresu počítače, který se stará o doručení pošty v této doméně. Pokud DNS tento záznam neobsahuje, pokusí se poštovní server doručit zprávu přímo na počítač uvedený v adrese za zavináčem. Poštovní server obsahuje v konfiguraci řadu parametrů, pomocí kterých můžeme mimo jiné nastavit, pro které domény MTA přijímá zprávy. Stejně tak je možné určit, od koho bude nebo nebude zprávy přijímat, což je velmi důležité z hlediska bezpečnosti a ochrany proti spamu.

### 3.7.3 HTTP

HTTP (Hypertext Transfer Protocol) je internetový protokol určený pro výměnu hypertextových dokumentů ve formátu HTML. Používá obvykle port TCP/80. Tento protokol je spolu s elektronickou poštou tím nejvíce používaným a zasloužil se o obrovský rozmach internetu. V současné době je používán i pro přenos dalších informací. Pomocí rozšíření MIME umí přenášet jakýkoli soubor (podobně jako e-mail), používá se společně s formátem XML pro tzv. webové služby (spouštění vzdálených aplikací) a pomocí aplikačních bran zpřístupňuje i další protokoly, jako je např. FTP nebo SMTP. HTTP používá jako některé další aplikace tzv. jednotný lokátor prostředků (URL, Uniform Resource Locator), který specifikuje jednoznačné umístění nějakého zdroje v Internetu. Samotný protokol HTTP neumožňuje šifrování ani zabezpečení integrity dat. Pro zabezpečení HTTP se často používá TLS spojení nad TCP. Toto použití je označováno jako HTTPS.

Protokol funguje způsobem dotaz-odpověď. Uživatel (pomocí programu, obvykle internetového prohlížeče) pošle serveru dotaz ve formě čistého textu, obsahujícího označení požadovaného dokumentu, informace o schopnostech prohlížeče apod. Server poté odpoví pomocí několika řádků textu popisujících výsledek dotazu (zda se dokument podařilo najít, jakého typu dokument je atd.), za kterými následují data samotného požadovaného dokumentu. Pokud uživatel bude mít po chvíli další dotaz na stejný server (např. proto, že uživatel v dokumentu kliknul na hypertextový odkaz), bude se jednat o další, nezávislý dotaz a odpověď. Z hlediska serveru nelze poznat, jestli tento druhý dotaz jakkoli souvisí s předchozím. Kvůli této vlastnosti se protokolu HTTP říká bezestavový protokol – protokol neumí uchovávat stav komunikace, dotazy spolu nemají souvislost. Tato vlastnost je nepřijemná pro implementaci složitějších procesů přes HTTP (např. internetový obchod potřebuje uchovávat informaci o identitě zákazníka, o obsahu jeho „nákupního košíku“ apod.). K tomuto účelu byl protokol HTTP rozšířen o tzv. HTTP cookies, které umožňují serveru uchovávat si informace o stavu spojení na počítači uživatele.

**Dotazovací metody** HTTP definuje několik metod, které se mají provést nad uvedeným objektem (dokumentem). <metoda> <objekt> HTTP/<verze>

- GET: Požadavek na uvedený objekt se zasláním případných dat (proměnné prohlížeče, session id, ...). Výchozí metoda při požadavku na zobrazení hypertextových stránek, RSS feedů aj. Celkově nejpoužívanější.
- HEAD: To samé jako metoda GET, ale už nepředává data. Poskytne pouze metadata o požadovaném cíli (velikost, typ, datum změny, ...).
- POST: Odesílá uživatelská data na server. Používá se například při odesílání formuláře na webu. S předaným objektem se pak zachází podobně jako při metodě GET. Data může odesílat i metoda GET, metoda POST se ale používá pro příliš velká data (víc než 512 bajtů, což je velikost požadavku GET) nebo pokud není vhodné přenášená data zobrazit jako součást URL (data předávaná metodou POST jsou obsažena v HTTP požadavku).
- PUT: Nahraje data na server. Objekt je jméno vytvářeného souboru. Používá se velmi zřídka, pro nahrávání dat na server se běžně používá FTP nebo SCP/SSH.
- DELETE: Smaže uvedený objekt ze serveru. Jsou na to potřeba jistá oprávnění stejně jako u metody PUT.
- TRACE: Odešle kopii obdrženého požadavku zpět odesílateli, takže klient může zjistit, co na požadavku mění nebo přidávají servery, kterými požadavek prochází.
- OPTIONS: Dotaz na server jaké podporuje metody.
- CONNECT: Spojí se s uvedeným objektem přes uvedený port. Používá se při průchodu skrze proxy pro ustanovení kanálu SSL.

**Bezpečné metody** Některé metody (například, HEAD, GET, OPTIONS a TRACE) jsou definovány jako bezpečné, což znamená, že jsou určeny pouze k získávání informací a neměly by změnit stav serveru. Tvorba libovolných GET požadavků bez ohledu na souvislost aplikace může být proto považována za bezpečnou.

Naproti tomu metody jako POST, PUT a DELETE jsou určeny pro akce, které mohou způsobovat nežádoucí účinky třeba na serveru. Tyto metody jsou proto obvykle používány v souladu webových robotů nebo webového prohledávače, některé, které neodpovídají, obvykle podávají žádosti bez ohledu na kontext nebo důsledky.

Kromě toho jsou metody, jako TRACE, TRACK a DEBUG považovány za potenciálně "nebezpečné" některými profesionály v oblasti bezpečnosti, protože mohou být použity útočníky k získání informací, nebo k obejítí bezpečnostní kontroly při útocích.

**URI (Uniform Resource Identifier)** je textový řetězec s definovanou strukturou, který slouží k přesné specifikaci zdroje informací (ve smyslu dokument nebo služba).

URI je nejobecnější z několika příbuzných typů identifikátorů. URI může popisovat zdroj jak čistě z hlediska jeho identity (a neurčovat, kde je možno zdroj získat), tak čistě z hlediska toho, jak je možno zdroj nalézt (a nepopisovat jeho identitu), tak i obojí současně – přesnou identitu zdroje i jak je možno ho dosáhnout. Oproti URI popisuje URL primárně způsob, jakým se lze ke zdroji dostat, naopak URN specifikuje zdroj jako takový a nesnaží se o návod k jeho dosažení. Hranice mezi těmito typy je však mírně mlhavá a zejména místo URL se často uvádí obecnější termín URI.

Jelikož je URI velmi obecný koncept, jeho základní formát je velmi volný: v principu se jedná o název takzvaného schématu, následovaný dvojtečkou a dále prakticky libovolným řetězcem, jehož význam a formátování už závisí právě na použitém schématu. Standard URI specifikuje pouze základní syntax, která popisuje, které znaky je dovoleno v URI použít apod. URI tedy má následující tvar: **schéma:hierarchická část?dotaz#fragment**, kde části ?dotaz a #fragment jsou nepovinné.

**schéma** musí začínat písmenem a obsahovat jen písmena, číslice a znaky plus (+), mínus (-) a tečku (.). Určuje, o jaký druh URI se jedná a jaký význam a syntaxe platí pro zbytek URI. U schématu nezáleží na velikosti písmen, ale za základní se považuje podoba se všemi písmeny malými.

**hierarchická část** obsahuje identifikátor zdroje v rámci nějaké hierarchické struktury. Standard URI dovoluje, aby tato část byla formátována prakticky libovolně, ale předepisuje také několik předdefinovaných syntaxí užitečných pro obvyklé situace. Jednou z nich je formát, kde po dvojtečce oddělující název schématu následují dvě lomítka (//), po kterých následuje označení tzv. autority, které je tvořeno jménem či IP adresou počítače, před kterým smí být informace o uživateli oddělená zavináčem (@), za ním smí být číslo portu oddělené dvojtečkou. Za označením autority následuje cesta: posloupnost segmentů oddelených lomítky (/) – značení obdobné adresářům, ale nemusí se jednat přímo o ně, ale obecně o jakýkoli hierarchický systém.

**dotaz** popisuje nehierarchickou část identifikátoru, která slouží k bližšímu určení požadovaného zdroje. Tato část nemá žádnou standardizovanou syntaxi, ovšem v praxi se velmi často používá posloupnost dvojic klíč=hodnota oddělená ampersandy (blah=foo&bar=baz). **fragment** nepřímo popisuje sekundární zdroj na základě primárního zdroje určeného předešlými částmi URI. Může popisovat nějakou konkrétní část (např. kapitolu knihy) tohoto zdroje, nějakýjinou reprezentaci příslušného zdroje apod.

**Zabezpečení HTTP** Existují dvě metody zabezpečeného http připojení: HTTPS URI a nadstavba HTTP 1.1. Druhou metodu ovšem zatím prohlížeče moc nepodporují, takže HTTPS se k vytvoření zabezpečené komunikace používá nejčastěji.

**HTTPS URI** Je syntakticky identické jako http, pouze přidává signalizaci prohlížeči, aby použil šifrovací metodu SSL/TLS k přenosu dat. SSL je vhodné pro HTTP, protože dokáže poskytnout ochranu přenosu, i když je pouze jedna strana komunikace ověřená. Typicky je ověřen pouze server (např. uživatel potvrdí certifikát). Aby pomocí HTTPS bylo možné rozlišovat virtuální servery, existuje rozšíření SNI (metoda, jak obejít nedostatek v protokolu SSL v2, který neumožňuje při přístupu přes HTTPS vytvářet virtuální webové servery – tj. více různých doménových jmen umístěných na jedné IP adrese (využívá se při webhostingu)). Pomocí SNI může klient před výměnou šifrovacích klíčů nejprve sdělit jméno požadovaného webového serveru a server tak může vybrat příslušný šifrovací klíč požadovaného virtuálního serveru.)

**HTTPS** Protokol HTTPS využívá asymetrické šifrování. Obě strany si před zahájením komunikace vygenerují pár klíčů (privátní a veřejný). Při zahájení komunikace si vymění veřejné klíče, které by obě strany měly ověřit pomocí jiného komunikačního kanálu. Ověření může proběhnout kontrolou výtahu (otisk, minatura, hash) veřejného klíče u protistrany například pomocí telefonu nebo lze použít princip přenosu důvěry, kdy nám protistrana předá veřejný klíč, který je digitálně podepsaný (nejlépe certifikační autoritou, které důvěřujeme a jejíž veřejný klíč máme v důvěryhodném úložišti, např. THAWTE, VeriSign, RapidSSL, GeoTrust, ...). Digitální certifikáty jsou základním kamenem zabezpečení poskytovaného protokolu SSL/TLS.

Jedním z (menších) limitů je mírné zpomalení oproti protokolu HTTP. Toto zpomalení je způsobeno právě výše zmíněným šifrováním daným informací.

**Proxy server** funguje jako prostředník mezi klientem a cílovým počítačem (serverem), překládá klientské požadavky a vůči cílovému počítači vystupuje sám jako klient. Přijatou odpověď následně odesílá zpět na klienta. Může se jednat jak o specializovaný hardware, tak o software provozovaný na běžném počítači. Proxy server odděluje lokální počítačovou síť (intranet) od Internetu.

**Aplikační proxy** je server speciálně určený pro určitý protokol nebo aplikaci. Proxy server může analyzovat obsah komunikace, případně ji pozměňovat (např. odstraňování reklam z http požadavků, blokování webových stránek podle obsahu a podobně) nebo ukládat požadavky do vyrovnávací paměti (cache), ze které mohou být při opakovaném požadavku odpovědi poskytnuty rychleji.

**HTTP session** Session v protokolu HTTP dává webovému serveru možnost uložit si libovolné (většinou však ne příliš obsáhlé) informace o uživatelích, kteří k němu přistupují, a to o každém zvlášť. Protokol HTTP ze svého principu (a způsobu komunikace stylem požadavek - odpověď) postrádá kontext o jednotlivých klientech, a právě session ho webovým aplikacím dokáže dát.

Session je v HTTP předáváno dvěma nejrozšířenějšími způsoby: v URL cílové stránky – jako její proměnná nebo jako HTTP cookie

To, jakým způsobem se session bude přenášet a pod jakým názvem, nastavuje webový server. Nevýhodou přenosu přes cookies je to, že některé archaické internetové prohlížeče je nepodporují (těch je ovšem mizivé procento) a ty soudobé dovolují cookies vypnout (opět, podíl uživatelů s vypnutými cookies je zanedbatelný). Jinou potenciální nevýhodou může být to, že pokud by potenciální útočník získal přístup k adresáři na serveru, do kterého se cookies ukládají, dostal by se i k session. Naopak, session v URL danou adresu znepřehledňují, činí ji příliš dlouhou, nezapamatovatelnou a působí proti principům SEO (mohou dokonce „rozmělňovat“ odkazovatelnost toho kterého odkazu). Při práci s aplikací s tímto způsobem předávání se jednotlivá session navíc ukládají i do historie prohlížeče a záložek. Je-li například (proti bezpečnostním zásadám) hodnota session generovaná na základě přístupových údajů uživatele, může se k nim útočník dostat snadněji než v případě ukládání do cookies. Některé webové aplikace (např. phpMyAdmin) proto při přihlašování vyžadují možnost ukládat cookies.

Vyplývá z příkladem je použití HTTP cookie k uložení jednoznačného identifikátoru (pojmenovaného SESSIONID, SESSID, SID apod., jehož hodnota je při startu session náhodně vygenerována). Podle taktu uloženého HTTP cookie pak server ve své paměti najde potřebné informace, například o přihlášeném uživateli, jeho úrovni přístupu a podobně. Podstatné je, že samotná data se již nepřenášejí (ani v URL, ani v samotném požadavku). Pokud se klient může připojit k libovolnému serveru z clusteru, je třeba mezi jednotlivými servery informace o sessions buď sdílet, nebo zajistit, že se stejný klient vždy připojí ke stejnemu uzlu. V opačném případě by se klient mohl spojit se serverem, který o zahájené session neví, a tak přijít o přihlášení, stav nákupního košíku a podobně.

Z hlediska skriptovacích jazyků pro programování intranetových/internetových aplikací, session představuje množinu proměnných (někde přístupnou přes sadu funkcí, jinde přes globální proměnnou), které dovolují uchovávat hodnoty, které jim byly nastaveny, po dobu připojení (tj. se znova učtením stránky se neztratí). Například, v jazycích ASP a ASP.NET jsou přístupné automaticky jako jednotlivé prvky pole Session("klíč"), v PHP pak pod tzv. superglobálním polem \$SESSION["klíč"]. V obou příkladech lze do session ukládat „jednoduché“ typy (celá čísla, racionalní čísla, řetězce, null) i z nich složená pole (teoreticky libovolné složitosti). Hodnoty se ukládají tzv. serializované (převedené na řetězec podle formátu, ze kterého jej lze zpětně načíst (přibližně jako třeba JSON)).

Zabezpečení session funguje na tom principu, že přístup k proměnným konkrétního uživatele je možný přes jeho identifikátor v podobě dostatečně dlouhého, náhodně generovaného, a tedy neuhádnutelného

klíče. Druhé specifikum je to, že session má poměrně krátkou dobu platnosti (implicitně bývá 24 minut, není-li nastaveno jinak).

**HTTP cookie** Cookies běžně slouží k rozlišování jednotlivých uživatelů, ukládají se do nich uživatelské předvolby apod.

Funkce cookies je definována v RFC 2965 pomocí HTTP hlaviček Set-Cookie (nebo její novější varianty Set-Cookie2) a Cookie. Hlavička Set-Cookie je poslána v odpovědi serveru a obsahuje:

- název (identifikátor) cookie
- data cookie (omezená prohlížečem, vyžadována je podpora alespoň pro 4096 byte),
- dobu platnosti cookie (resp. čas, kdy hodnota vyprší),
- doména, pro kterou cookie platí,
- adresář na serveru, pro který cookie platí.

Pokud má prohlížeč alespoň jednu cookie pro daný server (a daný adresář na něm), posílá s každým dotazem danému serveru i hlavičku Cookie, která obsahuje stejná data, která server původně poslal. Cookie lze nastavovat také na straně serveru (podle možností skriptovacího jazyka) – nastavení hodnoty cookie představuje v podstatě přidání HTTP hlavičky do odpovědi serveru (z toho důvodu musí nastavení proběhnout před vypsání čehokoli na výstup).

Cookies neznamenají žádné nebezpečí pro počítač jako takový. Přesto cookies mohou být nebezpečné pro ochranu soukromí. Navštívený web si totiž může ukládat do cookies jakékoli informace, které o návštěvníkovi zjistí a může tak postupně zjišťovat zájmy konkrétního návštěvníka. Které stránky navštěvuje, jaké informace vyhledává, jak často daný web navštěvuje apod. Těchto informací se dá posléze i bez vědomí návštěvníka využívat pro cílenou reklamu, statistické vyhodnocování chování návštěvníků, apod. Tyto informace však lze získávat i bez cookies, proto toto jejich využití nemůže být považováno za zvlášť nebezpečné. Cookies lze zneužít zejména tehdy, pokud získá útočník přístup k počítači uživatele, neboť cookies na počítači nejsou nijak chráněny. Pak lze předstírat např. cizí identitu.

### 3.7.4 FTP - přenos dat

FTP (File Transfer Protocol) je protokol pro přenos souborů mezi počítači pomocí počítačové sítě. Využívá protokol TCP z rodiny TCP/IP a může být používán nezávisle na použitém operačním systému (je platformně nezávislý). Jeho podpora je součástí webových prohlížečů nebo specializovaných programů (tzv. FTP klientů). FTP je jeden z nejstarších protokolů, využívá porty TCP/21 a TCP/20. Port 21 slouží k řízení a jsou jím také přenášeny příkazy FTP. Port 20 slouží k vlastnímu přenosu dat, který je 8bitový. Přenos může být binární nebo ascii (textový). Při textovém přenosu dochází ke konverzi koncu řádků – CR/LF (DOS, Microsoft Windows) nebo jen LF (unixové systémy), pokud jsou koncové systémy rozdílné. Při binárním přenosu není do dat nijak zasahováno. Protokol je interaktivní a umožňuje řízení přístupu (přihlašování login/heslo), specifikaci formátu přenášeného souboru (znakově – binárně), výpis vzdáleného adresáře atd. V současné době už není považován za bezpečný a z tohoto důvodu pro něj byla definována některá rozšíření. V protokolu je použit model klient-server. FTP server poskytuje data pro ostatní počítače. Klient se k serveru připojí a může provádět různé operace (výpis adresáře, změna adresáře, přenos dat atd.). Operace jsou řízeny sadou příkazů, které jsou definovány v rámci FTP protokolu, proto kdokoliv může vytvořit klienta pro jakékoli prostředí nebo operační systém. Existuje mnoho programů pro FTP servry i klienty a mnoho je jich volně dostupných.

Tři způsoby přenosu:

- Stream mode: spojitý proud dat, nejrychlejší, nejmenší nároky na klienta i server
- Block mode: data rozdělena do bloků, po přerušení lze navázat a pokračovat, cenou je vyšší režie
- Compressed mode: jednoduchá komprese – nahrazuje opakující se hodnoty

FTP používá obvyklý koncept uživatelů, hesel a přístupových práv (hesla posílá otevřeně!)

- anonymní přístup  
uživatel: ftp nebo anonymous  
heslo: e-mailová adresa  
nejčastěji pro distribuci volného software
- autentizovaný přístup  
uživatelské jméno a heslo definuje správce serveru, např. pro správu WWW stránek

**Bezpečnostní problémy** Při běžném připojování pomocí protokolu FTP jsou přihlašovací údaje (jméno a heslo) přenášeny v textové podobě a je technicky možné je odchytit. Následně mohou být data zcizena nebo upravena.

**SFTP** Protokol SFTP sám o sobě nezajišťuje ani autentizaci ani zabezpečení přenášených dat a obvykle k zajištění těchto služeb využívá protokol SSH-2.

**FTPS** je rozšíření protokolu FTP, která zajišťují zabezpečený přenos citlivých informací. AUTH TLS definuje příkaz, který určuje, že bude použito TLS zabezpečení přenosu. Klient se připojuje na port 21, zahajuje nešifrovanou komunikaci a žádá o aktivaci TLS před tím, než budou posílána citlivá data.

**TFTP** TFTP je určen pro přenos souborů v případech, kdy je běžný protokol FTP nevhodný pro svou komplikovanost. Typickým případem je bootování bezdiskových počítačů ze sítě, kdy se celý přenosový protokol musí vejít do omezeného množství paměti, která je k dispozici na bezdiskovém stroji. **Funguje na portu UDP.**

**Příkazy** Příkazy řídícího jazyka FTP lze rozdělit do tří skupin:

- Access Control Commands (příkazy řízení přístupu)
  - USER – zadání uživatelského jména
  - PASS – zadání uživatelského hesla
  - ACCT – zadání uživatelského účtu (téměř se nepoužívá)
  - CWD – změna aktuálního adresáře
  - CDUP – změna aktuálního adresáře na nadřazený adresář
  - QUIT – ukončení spojení
- Transfer Parameter Commands (příkazy nastavující parametry přenosu)
  - PORT – specifikuje počítač a port pro datové spojení. Klient pošle tento příkaz a bude na daném portu čekat na datové spojení.
  - PASV – žádá server o pasivní mód, tzn. že server bude poslouchat a klient bude iniciovat datové spojení.
  - TYPE – určuje typ reprezentace dat, např. text nebo binární
- FTP Service Commands (obsluhující příkazy)
  - RETR – slouží k přenosu souboru ze serveru
    - \* GET – přenos jednoho souboru ze serveru (příkaz klientského software ftp)
    - \* MGET – přenos více souborů ze serveru (příkaz klientského software ftp)
  - STOR – přenos souboru na server
    - \* PUT – přenos jednoho souboru na server (příkaz klientského software ftp)
    - \* MPUT – přenos více souborů na server (příkaz klientského software ftp)
  - RNFR, RNTO – přejmenování souboru
  - DELE – smazání souboru

- MKD – vytvoření nového adresáře
- RMD – smazání adresáře (adresář musí být prázdný)
- ABOR – zrušení předchozího příkazu
- PWD – zjištění aktuálního pracovního adresáře
- LIST – získání seznamu souborů. K získání tohoto seznamu se musí otevřít datové spojení. Pokud parametr tohoto příkazu je adresář, získá se výpis tohoto adresáře, pokud je to soubor, získají se informace o tomto souboru a pokud příkaz nemá parametr, je vrácen výpis aktuálního adresáře. Výpis příkazu list však závisí na systému a je určen především pro člověka.
- SYST – slouží k zjištění typu systému, na kterém běží FTP server.

Příkazy řídícího jazyka se přenášejí po řídícím spojení od FTP klienta na server.

**Aktivní režim** Na portu TCP/20 jsou přenášena data (data connection). V aktivním režimu navazuje připojení pro přenos dat server, klient naslouchá. Problém zpravidla nastává v případě, kdy se klient připojuje z privátní sítě a jeho IP adresa je překládána (NAT).

**Pasivní režim** V pasivním režimu navazuje data connection klient, kterému při sestavování připojení poslal server svou IP adresu a TCP port, na kterém naslouchá.

**FTP server, port forward (PF) a pasivní připojení** Pokud je připojení k FTP serveru realizováno prostřednictvím PF (nejčastěji se jedná o router s NAT a PF), tak router musí mít následující vlastnost - čte datovou část paketů FTP připojení, zjistí na jakém portu server naslouchá pro navázání data connection klientem a tento port začne forwardovat směrem k serveru. Po ukončení relace je ukončen i popsaný PF. Routery mají zpravidla tuto vlastnost již vestavěnou.

**SCP** slouží k bezpečnému přenosu souborů mezi dvěma počítači propojenými počítačovou sítí pomocí protokolu Secure Shell (SSH). SCP má omezené možnosti, a proto je nahrazován komplexnějším protokolem SFTP.

### 3.7.5 vzdálené přihlášení - Telnet

Protokol telnet pracuje na aplikační vrstvě používaný TCP/IP. Používá se v Internetu pro realizaci spojení typu klient-server protokolem TCP, přičemž přenáší osmibitové znaky oběma směry (duplexní spojení). Serverová část standardně naslouchá na portu číslo 23. Součástí protokolu je vyjednání nastavení určitých voleb důležitých pro vzájemnou komunikaci.

**Program Telnet** Telnet je též označení pro program, který realizoval komunikaci mezi dvěma počítači pomocí telnet protokolu. Program je dodnes součástí Microsoft Windows a unixových systémů. Dříve se používal (spolu s protokolem telnet) pro připojení ke vzdálenému počítači prostřednictvím počítačové sítě jako emulace terminálu, která poskytovala možnost práce uživatele na vzdáleném počítači v příkazovém rádku. Telnet tak byl nástupcem terminálů, ze kterých se uživatelé připojovali ke vzdálenému počítači pomocí sériové linky. Hlavní nevýhodou telnetu je **absence šifrování přenášených dat**, a proto dnes uživatelé místo telnetu používají protokol SSH.

Tři základní služby:

- Síťový virtuální terminál (NVT – Network Virtual Terminal), který poskytuje standardní rozhraní.
- Vyjednávání klienta/serveru o nastavení určitých voleb
- Symetrické zobrazení terminálu a procesů.

Síťový virtuální terminál zajišťuje průhlednost všech operací vůči uživateli. Nejsou zde rozdíly mezi jednotlivými komunikujícími zařízeními. Virtuální terminál poskytuje obecnou sadu příkazů pro všechny typy zařízení. Pro přenos využívá spolehlivé přenosové služby TCP, ale jen poloduplexním způsobem.

Definovaný formát NVT používá sedmibitový kód ASCII pro znaky a zobrazení. Díky tomu může Telnet operovat na různých operačních systémech. Klávesnice NVT generuje všech 128 kódů ASCII pomocí kláves, klávesových kombinací.

Přestože obě komunikující strany předpokládají, že je protější strana vybavena NVT, proběhne nejdříve výměna údajů, ve které se obě strany dohodnou na určitých parametrech a volbách komunikace. Pokud jedna strana neumí použít danou volbu, požadavek zamítne. Strany však musí dodržet minimální standard NVT.

Existují čtyři možné požadavky:

- WILL - odesílatel chce danou volbu zapnout
- DO - odesílatel chce, aby příjemce danou volbu zapnul
- WONT - odesílatel chce danou volbu vypnout
- DONT - odesílatel chce, aby příjemce danou volbu vypnul

### 3.7.6 vzdálené přihlášení - SSH

Označení pro program a zároveň pro zabezpečený komunikační protokol v počítačových sítích, které používají TCP/IP. SSH byl navržen jako náhrada za telnet a další nezabezpečené vzdálené shelly (rlogin, rsh apod.), které posílají heslo v nezabezpečené formě a umožňují tak jeho odposlechnutí při přenosu pomocí počítačové sítě. Šifrování přenášených dat, které SSH poskytuje, slouží k zabezpečení dat při přenosu přes nedůvěryhodnou síť.

SSH umožňuje bezpečnou komunikaci mezi dvěma počítači, která se využívá pro zprostředkování přístupu k příkazovému řádku, kopírování souborů a též jakýkoliv obecný přenos dat (s využitím síťového tunelování). Zabezpečuje autentizaci obou účastníků komunikace, transparentní šifrování přenášených dat, zajištění jejich integrity a volitelnou bezeztrátovou kompresi. Server standardně naslouchá na portu TCP/22.

Protokol SSH-2 má dobře navrženou vnitřní architekturu rozdelenou na oddělené vrstvy. Otevřená architektura nabízí významnou flexibilitu umožňující použití SSH nejen pro zabezpečený shell. Funkce transportní vrstvy samotné je srovnatelná s TLS. Vrstva autentizace uživatele je navržena pro snadné rozšíření vlastními autentizačními metodami. Vrstva spojení nabízí použití více podružných relací přenášených jedním SSH spojením.

**Transportní vrstva** zajišťuje počáteční výměnu klíčů, serverovou autentizaci, kompresi a ověření integrity. Poskytuje vysší vrstvě prostředí pro posílání a přijímání nešifrovaných paketů dat. Transportní vrstva také zajišťuje opětovnou výměnu klíčů – obvykle po 1 GB přenesených dat nebo po uplynutí 1 hodiny, podle toho, co nastane dříve.

**Vrstva autentizace uživatele** zajišťuje autentizaci klientů, která může být provedena mnoha způsoby. Samotná autentizace je řízena SSH klientem, server pouze reaguje na autorizační požadavky od SSH klienta.

- password - metoda pro přímou výměnu hesla, zahrnující možnost změny hesla. Tato metoda není implementována ve všech programech.
- publickey - metoda přihlášení pomocí veřejného klíče, obvykle podporuje alespoň RSA nebo RSA klíče. Tato metoda může zabránit útokům hrubou silou (když je vypnuta metoda "password").
- keyboard-interactive - univerzální metoda, při které server pošle klientovi jednu nebo více výzv, na které uživatel odpovídá pomocí klávesnice. Nejčastěji se používá pro jednorázovou autentizaci jako je S/Key nebo SecurID.
- GSSAPI - metody poskytují rozšířitelné rozhraní pro autentizaci pomocí externích mechanizmů jako je Kerberos 5 nebo NTLM, poskytujících možnost centrální autentizace pro přihlášení pomocí SSH relace. Tyto metody jsou obvykle používány v komerčních SSH implementacích.

**Vrstva spojení** definuje koncept kanálů, požadavků kanálů a globálních požadavků skrze které jsou poskytovány SSH služby. Jedno SSH spojení může hostovat více kanálů zároveň, kdy každý může přenášet data v obou směrech. Požadavky kanálů jsou použity pro přenos mimopásmových dat, jako jsou např. změna velikosti terminálového okna nebo návratový kód procesu na straně serveru. SSH klient si může pomocí globálního požadavku vyžádat forwardování (tunelování) portu na straně serveru.

**Autentizace pomocí veřejného klíče** Pro autentizaci uživatele je možné v SSH použít veřejný klíč. Nejprve je vygenerován pár šifrovacích klíčů – privátní (soukromý) klíč a veřejný klíč. Privátní je bezpečně uložen u uživatele a je chráněn heslovou frází. Veřejný klíč je uložen na cílový server (typicky do domácího adresáře uživatele, v unixových systémech do souboru `/.ssh/authorized_keys`). Při pokusu o přihlášení server veřejným klíčem, který má k dispozici, zašifruje blok náhodných dat (tzv. výzvu typu challenge-response), kterou nelze snadno odvodit nebo uhádnout a pošle ji klientovi. Klient výzvu pomocí privátního klíče dešifruje a dešifrovanou ji pošle zpět serveru. Pokud je výzva správně rozšifrována, má tím server ověřeno, že klient má k dispozici privátní klíč, který odpovídá veřejnému klíči, který má server dispozici, a tudíž může přístup schválit (autorizovat). Pokud výzva nebude správně rozšifrována, bude přístup pro klienta zamítnut. Z toho plyne, že privátní klíč neopustí klientův počítač, tudíž se nemůže stát, že by ho někdo odcizil při přenosu po síti, a přesto může dojít k ověření autenticity klienta a umožnění přístupu.

## Příklady použití z wiki

- **běžné přihlášení**

Uživatel Bob se přihlašuje na alicePC pod jejím jménem v rámci lokální sítě, nebo má-li alicePC veřejnou IP adresu (zda na alicePC běží ssh démon/server lze ověřit výpisem `# netstat -nltp`):  
Bob: \$ ssh alice@alicePC.cz

- **port forwarding**

V případě, kdy alicePC ani bobPC veřejnou IP adresu nemá, je potřeba využít zařízení, které jí má - walterPC. Oba se tedy přihlásí na Walterův stroj a Bob se z něj dostane na alicePC. Neboť běžný uživatel nemůže manipulovat s porty nižšími než 1024 a z bezpečnostních důvodů root přihlášení pomocí ssh nemívá povolenou, je třeba, aby ssh server Alice naslouchal např. na portu 6262. Do souboru `/etc/ssh/sshd_config` stačí přidat řádek `Port 6262` (po změně nastavení je třeba službu restartovat). Nyní:

Alice: \$ ssh walter@walterPC.cz -p 20085 -R 6262:localhost:6262  
Bob: \$ ssh walter@walterPC.cz -p 20085  
Bob již z walterPC: \$ ssh alice@localhost -p 6262

Relační databázové systémy: relační model dat, základní pojmy (atributy, domény, nalice, relační schémata, relace). Operace relační algebry: množinové operace, projekce, selekce, dělení, spojení a jeho typy. Vzájemné 2 vztahy relačních operací. Jazyk SQL: tabulky, pohledy, indexy, jejich vytváření a použití. Jazyk SQL: projekce, selekce a spojení tabulek. Referenční integrita v relačním modelu dat. Spolupráce SQL s jinými jazyky, procedury a triggery. Prostředky pro administraci relačního databázového systému. Systém řízení báze dat: služby, architektura, transakční zpracování dat, uzamykací protokoly

## 4 Relační databázové systémy

### Co je databázový systém

**databáze** (angl.: *data base*):

- *kolekce perzistentních dat* používaných aplikacemi nějakého subjektu
- **perzistence** = data přetrvávají výpočetní proces, který je vytvořil
- příklad: subjekt = univerzita, aplikace = studijní agenda, výzkumná agenda

**databázový systém, angl.: database system**

Systém pro organizaci, definici, manipulaci a dotazování nad perzistentními daty, který lze popsat jako množinu algoritmů pracujících s daty v určeném tvaru.

**často chápán dvojím způsobem:**

- ① jako teorie, tj. **formální model** (přesně definovaný a který lze zkoumat)
- ② jako konkrétní **softwarová implementace** vycházející z teorie (viz bod ①)

pro naše účely: **data = informace** (nerozlišujeme význam)

### Co je model dat

**(formální) model dat, angl.: data model**

Množina abstraktních a soběstačných formálních definic datových struktur a operací s daty (případně dalších operací, omezení a podobně), které dohromady tvoří formální výpočetní model, se kterým mohou uživatelé interagovat.

**poznámky:**

- existuje několik různých formálních modelů dat (viz přehled dále)
- stále je potřeba rozlišovat formální model × jeho implementace

**model dat (určitého subjektu)**

Přesněji: **model databáze** – návrh nebo implementace organizace dat určitého subjektu (např. návrh organizace dat „studijní agendy“ subjektu „univerzita“).

## Formální model × dotazovací jazyk

### kategorie jazyků souvisejících s formálními modely

- **dotazovací jazyk** (angl.: *query language*, zkráceně QL) = jazyk pro vyjadřování dotazů (angl.: *queries*) pro získávání dat z databáze
  - **jazyk pro definici dat** (angl.: *data definition language*, zkráceně DDL) = jazyk pro popis typu a struktury dat, která budou v databází uložena
  - **jazyk pro modifikaci dat** (angl.: *data modification language*, zkráceně DML) = jazyk pro vkládání, aktualizaci a mazání dat v databázi
- 
- jazyky používají (typicky) uživatelé různých rolí:
    - **administrátor databáze** – jazyk pro definici dat
    - **uživatel databáze** – dotazovací jazyk, jazyk pro modifikaci dat
  - k jednomu formálnímu modelu typicky existuje *víc jazyků* dané kategorie (!!) (např. pro relační model dat jsou jazyky SQL, QUEL, Tutorial D, ...)

## Přehled paradigm: Relační model

### charakteristika:

- jeden typ databázových objektů: **relace (nad relačními schématy)** je
  - matematický pojem *n*-ární relace = formální protějšek pojmu „datová tabulka“
  - formalizuje **základní data, výsledky dotazů i vztahy mezi daty**
- Codd, E. F.: A relational model of data for large shared data banks  
*Communications of the ACM* 13: 6 (1970)

### vlastnosti:

- + dobrý teoretický model, který lze navíc efektivně implementovat
- + od počátku formalizuje i související fenomény (závislosti v datech, normalizace)
- + **logická nezávislost dat** – fyzická a logická vrstva je oddělena
- + model je **referenčně transparentní**
- + k dispozici hodně kvalitativně různých SŘBD cílených na různou klientelu
- čistý relační model žádný (komerčně nasaditelný) SŘBD neimplementuje

## Příklad („Datové tabulky“ v relačním modelu dat)

jmeno	id	rodne-cislo
Adams	12345	571224/4023
Black	33355	840525/6670
Chang	66066	891117/1024

stuID	rok	predmet	typ
12345	2013	KMI/DATA1	A
12345	2013	KMI/FJ	B
33355	2012	KMI/DATA1	A
33355	2013	KMI/DATA2	B
33355	2013	KMI/PP1	A
66066	2012	KMI/DATA1	C

stuID	rok	predmet	vysl	datum
12345	2013	KMI/DATA1	95%	18/01/13
12345	2013	KMI/FJ	FAIL	25/06/13
12345	2013	KMI/FJ	35%	27/06/13
33355	2012	KMI/DATA1	FAIL	18/01/13
66066	2012	KMI/DATA1	FAIL	19/01/13
66066	2012	KMI/DATA1	85%	06/02/13

## Přehled paradigm: Objektové × relační databáze

### charakteristika:

- snaha zkombinovat relační model a objektové paradigma
- mnoho koncepcně různých (správných i nesprávných) přístupů

### dva hlavní typy přístupů:

- ➊ **nesprávný přístup:** perverzní rozšíření relačního modelu o reference
  - relační model de facto degeneruje na síťový model
  - všechny výhody relačního modelu jsou ztraceny
- ➋ **správný přístup:** přijmeme fakt, že *relační typy = třídy a hodnoty = objekty*
  - + zavádí subtypování do relačního modelu (koerce obecně ne)
  - + je referenčně transparentní, objekty lze pouze konstruovat, ne mutovat
  - + zachovává flexibilitu relačního dotazování (pokud je model dobře implementovaný)

 Date C. J., Darwen H.: *Foundation for Object/Relational Databases*  
Addison-Wesley Professional 1998, ISBN 978-0201309782

#### 4.0.1 Relační model

## Relační model dat (C. J. Date & H. Darwen)

### relační model dat, angl.: *relational model of data*

Relační model se skládá z pěti komponent:

- ① kolekce **skalárních typů** zahrnující typ „pravdivostní hodnota“ (angl.: *boolean*);
- ② systém pro **generování relačních typů**;
- ③ prostředky pro definici **relačních proměnných** daných relačních typů;
- ④ operátor pro **relační přiřazení**, tj. přiřazení hodnot relačním proměnným;
- ⑤ kolekce generických **relačních operací** pro vyjadřování relací z jiných relací.

### poznámky o rolích komponent:

- kolekce skalárních typů a relačních operací jsou „otevřené“ (ne pevně dané)
- relace = hodnoty; relační proměnné = jména (nabývající hodnot)
- prostředky pro definici dat (①–③), modifikaci dat (④–⑤) a dotazování (③, ⑤)

## Struktury relačního modelu dat (neformálně)

*Neformálně lze chápát (instanci) relační databáze jako kolekci pojmenovaných datových tabulek spolu s množinou integritních omezení.*

### tabulky mají dvě části:

- ① **záhlaví** (*metadata tabulky*), definuje:
  - jména sloupců
  - typy hodnot přípustných ve sloupcích
- ② **tělo** (*data tabulky*)
  - skládá se z řádků ( $n$ -tic hodnot)
  - průsečíky řádků a sloupců obsahují hodnoty

jmeno	id	rodne-cislo
Adams	12345	571224/4023
Black	33355	840525/6670
Chang	66066	891117/1024

### poznámky:

- ne každá tabulka splňující předchozí je „uspokojivou strukturou“ v RM
- cílem je používat tabulky, které jsou formalizovatelné jako  $n$ -ární relace

## Požadavky na tabulky

### **motto:**

*A table is normalized—equivalently, it is in the first normal form, 1NF—if and only if it is a direct and faithful representation of some relation. — C. J. Date*

### **první normální forma, angl.: first normal form**

Tabulka je v první normální formě (1NF), pokud splňuje následující podmínky:

- ① neuvažuje se **žádné uspořádání řádků** shora-dolů,
- ② neuvažuje se **žádné uspořádání sloupců** zleva-doprava,
- ③ v tabulce se nevyskytují **žádné duplicitní řádky**,
- ④ v každém vnitřním poli tabulky je **právě jedna hodnota daného typu**,
- ⑤ všechny **atributy jsou regulární**.

**poznámka:** regularita = řádky (skrytě) neobsahují žádnou extra informaci, která je dostupná (pouze) prostřednictvím speciálních funkcí

### 4.0.2 Základní pojmy

## Základní pojmy RM: Atribut

základní pojem použitý pro formalizaci „záhlaví tabulky“:

### **atribut, angl.: attribute**

Atributy jsou symbolická jména. Množinu všech atributů, o které předpokládáme, že je nekonečná a spočetná, označujeme  $Y$ .

### **poznámky:**

- značení:
  - ①  $y, y', y_1, y_2, \dots$  (označení obecných úvahách o RM) nebo
  - ② name, id, foo, ... (konkrétní atributy v příkladech)
- *intuitivní interpretace:* atribut je jméno, které může být použito pro „označení sloupce“ datové tabulky
- atribut je čistě *syntaktický pojem* (sám o sobě nemá žádnou hodnotu)

## Základní pojmy RM: Typ (doména)

### **motivace:**

*RM je silně typovaný, každý atribut (relace, ...) má přiřazen svůj typ tak, aby bylo možné provádět operace pouze s hodnotami povolených typů.*

### **typ, angl.: type**

Typ je pojmenovaná (nejvýš spočetná) množina elementů (hodnot).

### **poznámky:**

- ekvivalentní název pro typ je *doména* (historický starší, Codd)
- alternativní chápání pojmu typ/doména (někdy užitečné, my nevyužijem):
  - **typ** je *symbolické jméno* pro množinu hodnot (syntaktický pojem)
  - **doména** je *interpretace typu* – množina konkrétních hodnot (sémantický pojem)
- **rozlišitelnost hodnot**: hodnoty daného typu lze porovnávat pomocí „=“
  - porovnávat hodnoty různých typů nelze (!!)

## Základní pojmy RM: Relační schéma

formalizace pojmu „záhlaví tabulky“:

### **relační schéma, angl.: relation scheme**

Konečná množina  $R = \{\langle y_1, \tau_1 \rangle, \dots, \langle y_n, \tau_n \rangle\}$ , kde  $y_1, \dots, y_n$  jsou vzájemně různé atributy z  $Y$  a  $\tau_1, \dots, \tau_n$  jsou typy, se nazývá relační schéma.

### **poznámky:**

- značení:  $R, R', R_1, R_2, \dots, S, S', S_1, S_2, \dots$
- relační schéma může být prázdné, to jest  $R = \emptyset$  (existuje právě jedno)
- stejně jako hodnoty atributů, i relace mají svůj typ (relační schéma)

### **úmluva o zjednodušeném značení relačních schémat**

Pokud typy (domény) atributů vyplývají jasně z kontextu, pak relační schémata ztotožňujeme s konečnými podmnožinami  $R \subseteq Y$ ; typ (doménu)  $y \in Y$  značíme  $D_y$ .

## Intermezzo: Opakování užitých pojmu

- **množina** – chápeme naivně, budeme uvažovat nejvýš spočetné
- **uspořádaná dvojice** – značíme  $\langle a, b \rangle$ , pozor:  $\langle 10, 20 \rangle \neq \langle 20, 10 \rangle$
- **(naivní) kartézský součin** množin  $A$  a  $B$  je množina  $A \times B$  všech uspořádaných dvojic prvků jejichž první složky jsou prvky z  $A$  a druhé složky jsou prvky z  $B$ , to jest  $A \times B = \{\langle a, b \rangle \mid a \in A \text{ a } b \in B\}$
- **relace mezi množinami**  $A$  a  $B$  je libovolná podmnožina  $A \times B$
- **zobrazení** je množina  $f \subseteq A \times B$  taková, že pro každý prvek  $a \in A$  existuje právě jedno  $b \in B$  tak, že  $\langle a, b \rangle \in f$  (což označujeme  $f(a) = b$ )

**poznámky:**

- fakt, že  $f \subseteq A \times B$  je zobrazení píšeme  $f: A \rightarrow B$
- $f \subseteq A \times B$  je zobrazení p. k. následující podmínky jsou (obě) splněny:
  - ① pro každé  $a \in A$  existuje  $b \in B$  tak, že platí  $\langle a, b \rangle \in f$ ;
  - ② pro každé  $a \in A$  a  $b_1, b_2 \in B$  platí: pokud  $\langle a, b_1 \rangle \in f$  a  $\langle a, b_2 \rangle \in f$ , pak  $b_1 = b_2$ .

## Intermezzo: Obecný kartézský součin

Relace v RM jsou formalizovány jako konečné podmnožiny obecných kartézských součinů, ve kterých jsou  $n$ -tice hodnot formalizované zobrazeními.

### Kartézský součin, angl.: *Cartesian product*

Mějme systém množin  $A_i$ , které jsou indexovány prvky z množiny  $I$  (tzv. **indexy**).

**Kartézský součin množin**  $A_i$  ( $i \in I$ ) je množina  $\prod_{i \in I} A_i$  všech zobrazení  $f: I \rightarrow \bigcup_{i \in I} A_i$  takových, že  $f(i) \in A_i$  platí pro každý index  $i \in I$ .

Každé zobrazení  $f \in \prod_{i \in I} A_i$  se nazývá  **$n$ -tice** (angl.: *tuple*).

**poznámka:**

- pro  $f \in \prod_{i \in I} A_i$  se  $f(i) \in A_i$  nazývá **hodnota  $i$  v  $n$ -tici  $f$**
- český pojem  $n$ -tice je matoucí, protože  $n$  nemá vztah k  $I$  ani k  $A_i$  (!!)
- pokud je  $I = \{1, \dots, n\}$ , pak lze psát  $\prod_{i=1}^n A_i$
- důležité: indexy z množiny  $I$  nemají žádné „pořadí“ (!!)

## Základní pojmy RM: Relace nad relačním schématem

Definice (relace nad relačním schématem, angl.: *relation*)

Mějme relační schéma  $R \subseteq Y$  a nechť  $D_y$  ( $y \in Y$ ) označují domény atributů  $y \in R$ .

**Relace  $\mathcal{D}$**  nad relačním schématem  $R$  je libovolná konečná podmnožina  $\prod_{y \in R} D_y$ .

Číslo  $|\mathcal{D}|$  se nazývá **velikost relace  $\mathcal{D}$** , číslo  $|R|$  se nazývá **stupeň relace  $\mathcal{D}$** .

**značení:**

- relace označujeme  $\mathcal{D}, \mathcal{D}', \mathcal{D}_1, \mathcal{D}_2, \dots$

**poznámky:**

- $\prod_{y \in R} D_y$  je *kartézský součin domén* (indexy = atributy z  $R$ )
- $r \in \prod_{y \in R} D_y$  je *n-tice*, to jest  $r(y)$  je prvek z  $D_y$
- každá *n-tice*  $r \in \mathcal{D}$  reprezentuje jeden „řádek“ v tabulce odpovídající  $\mathcal{D}$
- zřejmě: tabulka je 1NF p. k. reprezentuje relaci na relačním schématu
- *nulární relace* (stupně 0), *unární relace* (stupně 1), *binární relace* (stupně 2), ...

## Zajímavé případy relací

**singeton** – jednoprvková unární relace (to jest  $|\mathcal{D}| = 1, |R| = 1$ )

$$\mathcal{D} = \{\{\langle y, d \rangle\}\}$$

<i>y</i>
<i>d</i>

kde  $R = \{y\}$  a  $d \in D_y$

**prázdná relace** – relace velikosti 0 (to jest  $|\mathcal{D}| = 0$ )

$$\mathcal{D} = \emptyset$$

<i>y</i> <sub>1</sub>	<i>y</i> <sub>2</sub>	...	<i>y</i> <sub><i>n</i></sub>
-----------------------	-----------------------	-----	------------------------------

kde  $R = \{y_1, y_2, \dots, y_n\}$

**relace nad prázdným schématem** – relace stupně 0 (to jest  $R = \emptyset$ );

- ①  $\mathcal{D}_{\top} = \{\emptyset\}$  (TABLE\_DEE, relační reprezentace *logické pravdy*)
- ②  $\mathcal{D}_{\perp} = \emptyset$  (TABLE\_DUM, relační reprezentace *logické nepravdy*)

**značení:** názvy TABLE\_DEE a TABLE\_DUM zavedl C. J. Date, viz

 Carroll L.: *Through the Looking-Glass, and What Alice Found There*, Macmillan 1871

## Skalární, $n$ -ticové a relační typy

tři kategorie typů v relačním modelu:

① **relační typy** (angl.: *relation types*)

- jsou definované relačními schématy
- hodnoty jsou relace nad relačními schématy

②  **$n$ -ticové typy** (angl.: *tuple types*)

- jsou definované relačními schématy (případně pouze syntakticky odlišené)
- hodnoty jsou  $n$ -tice nad relačními schématy

③ **skalární typy** (angl.: *scalar types*)

- všechny ostatní typy a jejich hodnoty

**poznámka:**

Tradiční chápání „skalárních“ a „neskalárních“ typů a hodnot (hodnoty skalárního typu nemají pro uživatele žádné „viditelné složky“) je analogicky vágní jako pseudodefinice 1NF.

## Základní dotazovací systémy:

① **relační algebra:**

- specifikuje množinu **operací s relacemi**
- dotazy = výrazy skládající se ze složených relačních operací
- interpretace dotazů: postupné vyhodnocení operací
- elementární operace s relacemi (SŘBD může dobře optimalizovat)

② **relační kalkuly:**

- několik typů: **doménový relační kalkul,  $n$ -ticový relační kalkul**
- dotazy = formule predikátové logiky (s volnými proměnnými)
- interpretace dotazů: ohodnocování formulí ve struktuře (instanci databáze)
- ryze deklarativní, vychází z něj řada jazyků (QUEL, do jisté míry SQL)

**poznámky:**

- moderní překladače dotazů (obvykle) vytvářejí *plány* používající operace, které jsou blízko operacím relační algebry
- známý mýtus: relační algebra „není deklarativní“ (nedává smysl)

#### 4.0.3 Operace relační algebry

### Typy relačních operací

#### **motto:**

*Množina relačních operací přímo ovlivňuje, jak silný (expresivní) bude dotazovací jazyk, který je na ní založen.*

#### **dělení operací relační algebry:**

- základní (minimální množina operací) / odvozené
- podle počtu operandů (operace s jednou, dvěma, třemi, ... relacemi)
- podle významu (množinové operace, protějšky kvantifikátorů, ... )  
⋮

#### **rozumná množina operací:**

*Množina operací, která zaručuje, že relační algebra je stejně silná jako (doménově nezávislý) doménový relační kalkul.*

### Množinové operace: Průnik relací

#### Definice (průnik relací, angl.: *intersection*)

Pro dvě relace  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na relačním schématu  $R \subseteq Y$  zavádíme

$$\mathcal{D}_1 \cap \mathcal{D}_2 = \{r \in \prod_{y \in R} D_y \mid r \in \mathcal{D}_1 \text{ a zároveň } r \in \mathcal{D}_2\}.$$

Relace  $\mathcal{D}_1 \cap \mathcal{D}_2$  na schématu  $R$  se nazývá **průnik relací**  $\mathcal{D}_1$  a  $\mathcal{D}_2$ .

#### **Tutorial D:**

$\langle \text{relační-výraz}_1 \rangle \text{ INTERSECT } \langle \text{relační-výraz}_2 \rangle$   
 $\text{INTERSECT } \{\langle \text{relační-výraz}_1 \rangle, \langle \text{relační-výraz}_2 \rangle, \dots\}$

#### **SQL:**

```
SELECT * FROM <jméno1>
    INTERSECT
SELECT * FROM <jméno2>
```

## Množinové operace: Sjednocení relací

Definice (sjednocení relací, angl.: *union*)

Pro dvě relace  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na relačním schématu  $R \subseteq Y$  zavádíme

$$\mathcal{D}_1 \cup \mathcal{D}_2 = \{r \in \prod_{y \in R} D_y \mid r \in \mathcal{D}_1 \text{ a/nebo } r \in \mathcal{D}_2\}.$$

Relace  $\mathcal{D}_1 \cup \mathcal{D}_2$  na schématu  $R$  se nazývá **sjednocení relací**  $\mathcal{D}_1$  a  $\mathcal{D}_2$ .

**Tutorial D:**

$\langle \text{relační-výraz}_1 \rangle \text{ UNION } \langle \text{relační-výraz}_2 \rangle$   
 $\text{UNION } \{\langle \text{relační-výraz}_1 \rangle, \langle \text{relační-výraz}_2 \rangle, \dots\}$

**SQL:**

```
SELECT * FROM <jméno1>
    UNION
SELECT * FROM <jméno2>
```

### Věta (Základní vlastnosti $\cap$ a $\cup$ )

Operace  $\cap$  a  $\cup$  s relacemi na schématu  $R$  mají následující vlastnosti:

- ①  $\cap$  a  $\cup$  jsou asociativní, komutativní, idempotentní a isotonné;
- ②  $\emptyset_R$  (prázdná relace na  $R$ ) je neutrální prvek operace  $\cup$  a anihilátor operace  $\cap$ ;
- ③  $\cap$  a  $\cup$  jsou vzájemně distributivní, to jest  
 $\mathcal{D}_1 \cup (\mathcal{D}_2 \cap \mathcal{D}_3) = (\mathcal{D}_1 \cup \mathcal{D}_2) \cap (\mathcal{D}_1 \cup \mathcal{D}_3)$ ,  
 $\mathcal{D}_1 \cap (\mathcal{D}_2 \cup \mathcal{D}_3) = (\mathcal{D}_1 \cap \mathcal{D}_2) \cup (\mathcal{D}_1 \cap \mathcal{D}_3)$ ;
- ④  $\cap$  a  $\cup$  jsou vzájemně absorbují, to jest  
 $\mathcal{D}_1 = \mathcal{D}_1 \cap (\mathcal{D}_1 \cup \mathcal{D}_2)$ ,  
 $\mathcal{D}_1 = \mathcal{D}_1 \cup (\mathcal{D}_1 \cap \mathcal{D}_2)$ .

**Důkaz.**

Tvrzení ①–④ plynou z vlastností pravdivostních funkcí logických spojek „konjunkce“ a „disjunkce“. Například pro ④: Pokud  $r \in \mathcal{D}_1$ , pak i  $r \in \mathcal{D}_1 \cup \mathcal{D}_2$  a proto  $\mathcal{D}_1 \subseteq \mathcal{D}_1 \cap (\mathcal{D}_1 \cup \mathcal{D}_2)$ . Opačně, pokud  $r \in \mathcal{D}_1 \cap (\mathcal{D}_1 \cup \mathcal{D}_2)$ , pak zřejmě  $r \in \mathcal{D}_1$ .  $\square$

## Množinové operace: Rozdíl relací

Definice (rozdíl relací, angl.: *difference/minus*)

Pro dvě relace  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na relačním schématu  $R \subseteq Y$  zavádíme

$$\mathcal{D}_1 \setminus \mathcal{D}_2 = \{r \in \prod_{y \in R} D_y \mid r \in \mathcal{D}_1 \text{ a zároveň } r \notin \mathcal{D}_2\}.$$

Relace  $\mathcal{D}_1 \setminus \mathcal{D}_2$  na schématu  $R$  se nazývá **rozdíl relací**  $\mathcal{D}_1$  a  $\mathcal{D}_2$ .

### Tutorial D:

$\langle \text{relační-výraz}_1 \rangle \text{ MINUS } \langle \text{relační-výraz}_2 \rangle$

### SQL:

```
SELECT * FROM <jméno1>
EXCEPT
SELECT * FROM <jméno2>
```

## Věta (Základní vlastnosti $\cap$ , $\cup$ a $\setminus$ )

Operace  $\cap$ ,  $\cup$ ,  $\setminus$  s relacemi na schématu  $R$  mají následující vlastnosti:

- ①  $\setminus$  je isotonní v prvním argumentu a antitonní v druhém argumentu;
- ②  $\cup$  a  $\setminus$  jsou adjungované operace:  $\mathcal{D}_1 \setminus \mathcal{D}_2 \subseteq \mathcal{D}_3$  právě tehdy, když  $\mathcal{D}_1 \subseteq \mathcal{D}_2 \cup \mathcal{D}_3$ ;
- ③  $\mathcal{D}_1 \subseteq \mathcal{D}_2$  právě tehdy, když  $\mathcal{D}_1 \setminus \mathcal{D}_2 = \emptyset_R$ ;
- ④  $(\mathcal{D}_1 \setminus \mathcal{D}_2) \cap (\mathcal{D}_2 \setminus \mathcal{D}_1) = \emptyset_R$ ;
- ⑤  $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}_1 \cup (\mathcal{D}_2 \setminus \mathcal{D}_1)$ ;
- ⑥  $\mathcal{D}_1 \cap \mathcal{D}_2 = \mathcal{D}_2 \setminus (\mathcal{D}_2 \setminus \mathcal{D}_1)$ ;
- ⑦ pokud  $\mathcal{D}_1 \subseteq \mathcal{D}_2$ , pak  $\mathcal{D}_1 = \mathcal{D}_2 \setminus (\mathcal{D}_2 \setminus \mathcal{D}_1)$ ;
- ⑧  $\setminus$  je distributivní (zleva/zprava) vzhledem k  $\cup$  a  $\cap$  následovně:  
 $(\mathcal{D}_1 \cup \mathcal{D}_2) \setminus \mathcal{D}_3 = (\mathcal{D}_1 \setminus \mathcal{D}_3) \cup (\mathcal{D}_2 \setminus \mathcal{D}_3)$ ,  
 $(\mathcal{D}_1 \cap \mathcal{D}_2) \setminus \mathcal{D}_3 = (\mathcal{D}_1 \setminus \mathcal{D}_3) \cap (\mathcal{D}_2 \setminus \mathcal{D}_3)$ ,  
 $\mathcal{D}_1 \setminus (\mathcal{D}_2 \cap \mathcal{D}_3) = (\mathcal{D}_1 \setminus \mathcal{D}_2) \cup (\mathcal{D}_1 \setminus \mathcal{D}_3)$ ,  
 $\mathcal{D}_1 \setminus (\mathcal{D}_2 \cup \mathcal{D}_3) = (\mathcal{D}_1 \setminus \mathcal{D}_2) \cap (\mathcal{D}_1 \setminus \mathcal{D}_3)$ .

## Odvozené množinové operace

**otázky související s množinovými operacemi:**

- Které operace vzít jako základní a které jako odvozené?
- Jak zavést obecný koncept (booleovské) operace s relacemi?

### Definice (obecná booleovská operace, angl.: *Boolean operation*)

Mějme relace  $\mathcal{D}_1, \dots, \mathcal{D}_n$  a  $\mathcal{D}$  nad relačním schématem  $R$  takové, že platí  $\mathcal{D}_i \subseteq \mathcal{D}$  pro každé  $i = 1, \dots, n$ . Dál uvažujme výrokovou formuli  $\varphi$ , která obsahuje nejvýš výrokové symboly  $p_1, \dots, p_n$ . Pak  $\text{Bool}(\mathcal{D}_1, \dots, \mathcal{D}_n, \mathcal{D}, \varphi)$  je definovaná

$$\text{Bool}(\mathcal{D}_1, \dots, \mathcal{D}_n, \mathcal{D}, \varphi) = \{r \in \mathcal{D} \mid e_r(\varphi) = 1\},$$

kde  $e_r$  je ohodnocení výrokových symbolů (jednoznačně rozšířené na všechny výrokové formule), splňující

$$e_r(p_i) = \begin{cases} 1, & \text{pokud } r \in \mathcal{D}_i, \\ 0, & \text{jinak.} \end{cases}$$

## Intermezzo: Operace s $n$ -ticemi

### sjednocení (zřetězení) $n$ -tic, angl.: *concatenation/union*

Mějme  $n$ -tice  $r \in \prod_{y \in R} D_y$  a  $s \in \prod_{y \in S} D_y$  takové, že  $r(y) = s(y)$  pro každý atribut  $y \in R \cap S$ . Zobrazení  $r \cup s$  (zkráceně  $rs$ ) nazveme sjednocení (zřetězení)  $n$ -tic  $r$  a  $s$ .

### projekce (zúžení) $n$ -tice, angl.: *projection*

Mějme  $n$ -tici  $r \in \prod_{y \in R} D_y$  a pak  $S \subseteq R$  definujeme  $r(S) \in \prod_{y \in S} D_y$  tak, že  $(r(S))(y) = r(y)$  pro každý  $y \in S$ . Zobrazení  $r(S)$  se nazývá projekce  $r$  na  $S$ .

**poznámky:**

- sjednocení je: komutativní ( $rs = sr$ ), asociativní ( $r(st) = (rs)t$ ), idempotentní ( $rr = r$ ), neutrální vzhledem k  $\emptyset$  ( $r\emptyset = \emptyset r = r$ )
- sjednocení  $n$ -tic  $r \cup s$  je množinově-teoretické sjednocení zobrazení, odtud:

$$(rs)(y) = \begin{cases} r(y), & \text{pokud } y \in R, \\ s(y), & \text{jinak.} \end{cases}$$

# Projekce

## Definice (projekce, angl.: *projection*)

Mějme relaci  $\mathcal{D}$  na schématu  $R$ . Pri libovolné  $S \subseteq R$  položíme:

$$\pi_S(\mathcal{D}) = \{s \in \prod_{y \in S} D_y \mid \text{existuje } t \in \prod_{y \in R \setminus S} D_y \text{ tak, že } st \in \mathcal{D}\}.$$

Relace  $\pi_S(\mathcal{D})$  se nazývá **projekce  $\mathcal{D}$  na schéma  $S$** .

### Tutorial D:

$\langle \text{relační-výraz} \rangle \{ \langle \text{atribut}_1 \rangle, \dots, \langle \text{atribut}_n \rangle \}$

$\langle \text{relační-výraz} \rangle \{ \text{ALL BUT } \langle \text{atribut}_1 \rangle, \dots, \langle \text{atribut}_n \rangle \}$

### SQL:

`SELECT DISTINCT  $\langle \text{atribut}_1 \rangle, \dots, \langle \text{atribut}_n \rangle$  FROM  $\langle \text{jméno} \rangle$`

## Věta (Základní vlastnosti projekce)

Pro relaci  $\mathcal{D}$  na schématu  $R$  platí:

- ①  $\pi_R(\mathcal{D}) = \mathcal{D}$ ;
- ②  $\pi_\emptyset(\mathcal{D}) = \begin{cases} \emptyset, & \text{pokud } \mathcal{D} = \emptyset_R, \\ \{\emptyset\}, & \text{jinak;} \end{cases}$ ;
- ③  $\pi_S(\mathcal{D}) = \{r(S) \mid r \in \mathcal{D}\}$ ;
- ④  $\pi_{S_1}(\pi_{S_2}(\mathcal{D})) = \pi_{S_1}(\mathcal{D})$  pro každé  $S_1 \subseteq S_2 \subseteq R$ ;
- ⑤  $\pi_S(\mathcal{D}_1 \cup \mathcal{D}_2) = \pi_S(\mathcal{D}_1) \cup \pi_S(\mathcal{D}_2)$ .

### Důkaz.

① je zřejmá; ② plyne z toho, jak vypadají relace nad prázdným schématem; ③ plyne z toho, že pokud  $s \in \prod_{y \in S} D_y$ , pak existuje  $t \in \prod_{y \in R \setminus S} D_y$  tak, že  $st \in \mathcal{D}$  p. k.  
 $s = r(S)$  pro nějakou  $r \in \mathcal{D}$ ; pro ④ je  $\pi_{S_1}(\pi_{S_2}(\mathcal{D})) = \{s_2(S_1) \mid s_2 \in \pi_{S_2}(\mathcal{D})\} = \{r(S_2)(S_1) \mid r \in \mathcal{D}\} = \{r(S_1) \mid r \in \mathcal{D}\}$ ; ⑤  $\pi_S(\mathcal{D}_1 \cup \mathcal{D}_2) = \{r(S) \mid r \in \mathcal{D}_1 \cup \mathcal{D}_2\} = \{r(S) \mid r \in \mathcal{D}_1\} \cup \{r(S) \mid r \in \mathcal{D}_2\} = \pi_S(\mathcal{D}_1) \cup \pi_S(\mathcal{D}_2)$ . □

## Restrikce

### Definice (restrikce, angl.: *restriction*)

Mějme relaci  $\mathcal{D}$  na schématu  $R$  a nechť  $\theta$  je skalární výraz typu „pravdivostní hodnota“, který může obsahovat jména atributů z  $R$ . Řekneme, že  $r \in \mathcal{D}$  **splňuje** (podmínu danou výrazem)  $\theta$ , pokud má  $\theta$  hodnotu „pravda“ za předpokladu, že jsme nahradili jména atributů v  $\theta$  jejich hodnotami z  $r$ . Položíme

$$\sigma_\theta(\mathcal{D}) = \{r \in \mathcal{D} \mid r \text{ splňuje } \theta\}$$

Relace  $\sigma_\theta(\mathcal{D})$  se nazývá **restrikce  $\mathcal{D}$**  splňující  $\theta$ .

**poznámky:**

- **restrikce na rovnost** – restrikce tvaru  $\sigma_{y=d}(\mathcal{D})$
- terminologie: *restrikce* = *selekce* (nezaměňovat se **SELECT** z SQL, !!)
- restrikce (zmenšení velikosti relace)  $\times$  projekce (zmenšení stupně relace)

## Restrikce v Tutorial D a SQL

**Tutorial D:**

$\langle \text{relační-výraz} \rangle \text{ WHERE } \langle \text{podmínka} \rangle$

**SQL:**

`SELECT * FROM jméno WHERE podmínka`

### Poznámka o výrazech v restrikcích

Výraz  $\langle \text{podmínka} \rangle$  chápeme jako obecný výraz, který lze formulovat v daném dotazovacím jazyku. Pro zjednodušení dalších úvah budeme předpokládat, že  $\langle \text{podmínka} \rangle$  je výraz, který se chová z hlediska své interpretace *funkcionálně* (nemá vedlejší efekty); lze jej chápat jako zobrazení  $\theta: \prod_{y \in R} D_y \rightarrow \{0, 1\}$ , kde  $\theta(r) = 1$  znamená, že  $r$  splňuje  $\theta$ .

**důsledek:** pokud je  $\prod_{y \in R} D_y$  konečná,  $\sigma_\theta(\mathcal{D})$  lze chápat jako průnik relací (!!)

## Věta (Základní vlastnosti restrikce)

Pro relaci  $\mathcal{D}$  na schématu  $R$  platí:

- ①  $\sigma_{\theta_1}(\sigma_{\theta_2}(\mathcal{D})) = \sigma_{\theta_2}(\sigma_{\theta_1}(\mathcal{D}))$
- ②  $\sigma_\theta(\sigma_\theta(\mathcal{D})) = \sigma_\theta(\mathcal{D})$

### Důkaz.

Bod ① plyne z komutativity konjunkce, konkrétně  $r \in \sigma_{\theta_1}(\sigma_{\theta_2}(\mathcal{D}))$  p. k.  $r$  splňuje  $\theta_1$  a náleží do  $\sigma_{\theta_2}(\mathcal{D})$  což platí p. k.  $r$  splňuje  $\theta_1$  a  $r$  splňuje  $\theta_2$  a náleží do  $\mathcal{D}$ , což je p. k.  $r$  splňuje  $\theta_2$  a  $r$  splňuje  $\theta_1$  a náleží do  $\mathcal{D}$  (viz poznámku o funkcionálním charakteru podmínek), to jest  $r \in \sigma_{\theta_2}(\sigma_{\theta_1}(\mathcal{D}))$ . Bod ② plyne analogicky z idempotence konjunkce (opět důležitý předpoklad z předchází poznámky).  $\square$

### značení:

- pro podmínky  $\theta_1, \dots, \theta_n$  píšeme  $\sigma_{\theta_1, \dots, \theta_n}(\mathcal{D})$  místo  $\sigma_{\theta_1}(\sigma_{\theta_2}(\dots(\sigma_{\theta_n}(\mathcal{D}))\dots))$
- alternativní značení:  $\sigma_{\theta_1 \wedge \dots \wedge \theta_n}(\mathcal{D})$  ( $\wedge$  je symbol pro konjunkci)

## Spojitelné $n$ -tice a spojení $n$ -tic

rozšíření pojmu (PŘEDNÁŠKA 3):

**spojitelnost, spojení  $n$ -tic, angl.: joinable tuples, tuple join**

Mějme  $n$ -tice  $r \in \prod_{y \in R} D_y$  a  $s \in \prod_{y \in S} D_y$ . Pokud  $r(y) = s(y)$  pro každý atribut  $y \in R \cap S$ , pak řekneme, že  $r$  a  $s$  jsou spojitelné. Pokud jsou  $r$  a  $s$  spojitelné, pak zobrazení  $r \cup s$  (zkráceně  $rs$ ) nazveme spojení  $n$ -tic  $r$  a  $s$ .

### poznámky:

- spojení je: komutativní ( $rs = sr$ ), asociativní ( $r(st) = (rs)t$ ), idempotentní ( $rr = r$ ), neutrální vzhledem k  $\emptyset$  ( $r\emptyset = \emptyset r = r$ )
- důsledek: lze rozšířit na libovolné množství  $n$ -tic
- spojení  $n$ -tic  $r \cup s$  je množinově-teoretické sjednocení zobrazení, odtud:

$$(rs)(y) = \begin{cases} r(y), & \text{pokud } y \in R, \\ s(y), & \text{jinak.} \end{cases}$$

## Přirozené spojení

**neformálně:**

Spojení relací  $\mathcal{D}_1$  a  $\mathcal{D}_2$  je relace obsahující spojení  $r_1 r_2$  všech spojitelných  $n$ -tic  $r_1 \in \mathcal{D}_1$  a  $r_2 \in \mathcal{D}_2$  (a je to nejmenší relace této vlastnosti).

**Definice (přirozené spojení, angl.: natural join)**

Mějme relace  $\mathcal{D}_1$  nad schématem  $R \cup S$  a  $\mathcal{D}_2$  nad schématem  $S \cup T$  tak, že  $R \cap T = \emptyset$ . Relace  $\mathcal{D}_1 \bowtie \mathcal{D}_2$  nad  $R \cup S \cup T$  definovaná

$$\mathcal{D}_1 \bowtie \mathcal{D}_2 = \{rst \in \prod_{y \in R \cup S \cup T} D_y \mid rs \in \mathcal{D}_1, st \in \mathcal{D}_2 \text{ a } s \in \prod_{y \in S} D_y\}$$

se nazývá (**přirozené**) spojení relací  $\mathcal{D}_1$  a  $\mathcal{D}_2$ .

**poznámky:**

- $(R \cup S) \cap (S \cup T) = S \cup (R \cap T) = S \cup \emptyset = S$   
( $S$  je množina všech společných atributů relací  $\mathcal{D}_1, \mathcal{D}_2$ )
- $\mathcal{D}_1 \bowtie \mathcal{D}_2$  obsahuje všechny atributy z obou tabulek

**Příklad (Příklady přirozených spojení)**

<table border="1"> <thead> <tr><th>FOO</th><th>BAR</th><th>BAZ</th></tr> </thead> <tbody> <tr><td>444</td><td>ghi</td><td>103</td></tr> <tr><td>555</td><td>def</td><td>102</td></tr> <tr><td>555</td><td>ghi</td><td>103</td></tr> <tr><td>666</td><td>abc</td><td>101</td></tr> </tbody> </table>	FOO	BAR	BAZ	444	ghi	103	555	def	102	555	ghi	103	666	abc	101	$\bowtie$	<table border="1"> <thead> <tr><th>BAR</th><th>BAZ</th><th>QUX</th></tr> </thead> <tbody> <tr><td>abc</td><td>111</td><td>zzz</td></tr> <tr><td>def</td><td>102</td><td>www</td></tr> <tr><td>def</td><td>102</td><td>yyy</td></tr> <tr><td>ghX</td><td>103</td><td>xxx</td></tr> <tr><td>ghi</td><td>103</td><td>ttt</td></tr> <tr><td>ghi</td><td>103</td><td>uuu</td></tr> <tr><td>ghi</td><td>103</td><td>vvv</td></tr> </tbody> </table>	BAR	BAZ	QUX	abc	111	zzz	def	102	www	def	102	yyy	ghX	103	xxx	ghi	103	ttt	ghi	103	uuu	ghi	103	vvv	$=$	<table border="1"> <thead> <tr><th>FOO</th><th>BAR</th><th>BAZ</th><th>QUX</th></tr> </thead> <tbody> <tr><td>444</td><td>ghi</td><td>103</td><td>ttt</td></tr> <tr><td>444</td><td>ghi</td><td>103</td><td>uuu</td></tr> <tr><td>444</td><td>ghi</td><td>103</td><td>vvv</td></tr> <tr><td>555</td><td>def</td><td>102</td><td>www</td></tr> <tr><td>555</td><td>def</td><td>102</td><td>yyy</td></tr> <tr><td>555</td><td>ghi</td><td>103</td><td>ttt</td></tr> <tr><td>555</td><td>ghi</td><td>103</td><td>uuu</td></tr> <tr><td>555</td><td>ghi</td><td>103</td><td>vvv</td></tr> </tbody> </table>	FOO	BAR	BAZ	QUX	444	ghi	103	ttt	444	ghi	103	uuu	444	ghi	103	vvv	555	def	102	www	555	def	102	yyy	555	ghi	103	ttt	555	ghi	103	uuu	555	ghi	103	vvv
FOO	BAR	BAZ																																																																													
444	ghi	103																																																																													
555	def	102																																																																													
555	ghi	103																																																																													
666	abc	101																																																																													
BAR	BAZ	QUX																																																																													
abc	111	zzz																																																																													
def	102	www																																																																													
def	102	yyy																																																																													
ghX	103	xxx																																																																													
ghi	103	ttt																																																																													
ghi	103	uuu																																																																													
ghi	103	vvv																																																																													
FOO	BAR	BAZ	QUX																																																																												
444	ghi	103	ttt																																																																												
444	ghi	103	uuu																																																																												
444	ghi	103	vvv																																																																												
555	def	102	www																																																																												
555	def	102	yyy																																																																												
555	ghi	103	ttt																																																																												
555	ghi	103	uuu																																																																												
555	ghi	103	vvv																																																																												
<table border="1"> <thead> <tr><th>FOO</th><th>BAR</th><th>BAZ</th></tr> </thead> <tbody> <tr><td>555</td><td>def</td><td>102</td></tr> <tr><td>555</td><td>ghi</td><td>103</td></tr> <tr><td>666</td><td>abc</td><td>101</td></tr> </tbody> </table>	FOO	BAR	BAZ	555	def	102	555	ghi	103	666	abc	101	$\bowtie$	<table border="1"> <thead> <tr><th>BAR</th><th>BAZ</th><th>QUX</th></tr> </thead> <tbody> <tr><td>AAA</td><td>101</td><td>AAA</td></tr> <tr><td>def</td><td>102</td><td>BBB</td></tr> <tr><td>ghi</td><td>333</td><td>CCC</td></tr> </tbody> </table>	BAR	BAZ	QUX	AAA	101	AAA	def	102	BBB	ghi	333	CCC	$=$	<table border="1"> <thead> <tr><th>FOO</th><th>BAR</th><th>BAZ</th><th>QUX</th></tr> </thead> <tbody> <tr><td>555</td><td>def</td><td>102</td><td>BBB</td></tr> </tbody> </table>	FOO	BAR	BAZ	QUX	555	def	102	BBB																																											
FOO	BAR	BAZ																																																																													
555	def	102																																																																													
555	ghi	103																																																																													
666	abc	101																																																																													
BAR	BAZ	QUX																																																																													
AAA	101	AAA																																																																													
def	102	BBB																																																																													
ghi	333	CCC																																																																													
FOO	BAR	BAZ	QUX																																																																												
555	def	102	BBB																																																																												
<table border="1"> <thead> <tr><th>FOO</th><th>BAR</th><th>BAZ</th></tr> </thead> <tbody> <tr><td>333</td><td>jkl</td><td>103</td></tr> <tr><td>444</td><td>ghi</td><td>102</td></tr> <tr><td>555</td><td>def</td><td>101</td></tr> <tr><td>666</td><td>abc</td><td>101</td></tr> </tbody> </table>	FOO	BAR	BAZ	333	jkl	103	444	ghi	102	555	def	101	666	abc	101	$\bowtie$	<table border="1"> <thead> <tr><th>BAZ</th></tr> <tr><td>101</td></tr> <tr><td>103</td></tr> </thead></table>	BAZ	101	103	$=$	<table border="1"> <thead> <tr><th>FOO</th><th>BAR</th><th>BAZ</th></tr> </thead> <tbody> <tr><td>333</td><td>jkl</td><td>103</td></tr> <tr><td>555</td><td>def</td><td>101</td></tr> <tr><td>666</td><td>abc</td><td>101</td></tr> </tbody> </table>	FOO	BAR	BAZ	333	jkl	103	555	def	101	666	abc	101																																													
FOO	BAR	BAZ																																																																													
333	jkl	103																																																																													
444	ghi	102																																																																													
555	def	101																																																																													
666	abc	101																																																																													
BAZ																																																																															
101																																																																															
103																																																																															
FOO	BAR	BAZ																																																																													
333	jkl	103																																																																													
555	def	101																																																																													
666	abc	101																																																																													

## Přirozené spojení v Tutorial D a SQL

### Tutorial D:

```
 $\langle \text{relační-výraz}_1 \rangle \text{ JOIN } \langle \text{relační-výraz}_2 \rangle$   
 $\text{JOIN } \{\langle \text{relační-výraz}_1 \rangle, \langle \text{relační-výraz}_2 \rangle, \dots\}$ 
```

### SQL:

```
SELECT * FROM  $\langle jméno_1 \rangle$  NATURAL JOIN  $\langle jméno_2 \rangle$   
SELECT  $\langle jméno_1 \rangle.*$ ,  $\langle jméno_2 \rangle.\langle T\text{-atribut}_1 \rangle, \dots, \langle jméno_2 \rangle.\langle T\text{-atribut}_m \rangle$   
FROM  $\langle jméno_1 \rangle$ ,  $\langle jméno_2 \rangle$   
WHERE  $\langle jméno_1 \rangle.\langle S\text{-atribut}_1 \rangle = \langle jméno_2 \rangle.\langle S\text{-atribut}_1 \rangle$   
AND ... AND  
 $\langle jméno_1 \rangle.\langle S\text{-atribut}_n \rangle = \langle jméno_2 \rangle.\langle S\text{-atribut}_n \rangle$ , kde  
 $\langle T\text{-atribut}_i \rangle$  jsou všechny atributy z  $\langle jméno_2 \rangle$ , které nejsou v  $\langle jméno_1 \rangle$   
 $\langle S\text{-atribut}_j \rangle$  jsou všechny atributy společné pro  $\langle jméno_1 \rangle$  a  $\langle jméno_2 \rangle$ 
```

## Speciální případy spojení: Průnik

### Věta (Vztah spojení a průniku)

Nechť  $\mathcal{D}_1$  a  $\mathcal{D}_2$  jsou relace nad stejným schématem, pak  $\mathcal{D}_1 \bowtie \mathcal{D}_2 = \mathcal{D}_1 \cap \mathcal{D}_2$ .

### Důkaz.

Jelikož mají  $\mathcal{D}_1$  a  $\mathcal{D}_2$  stejná schémata, pak  $R = T = \emptyset$  a dostáváme:

$$\begin{aligned}\mathcal{D}_1 \bowtie \mathcal{D}_2 &= \{rst \in \prod_{y \in R \cup S \cup T} D_y \mid rs \in \mathcal{D}_1, st \in \mathcal{D}_2 \text{ a } s \in \prod_{y \in S} D_y\} \\ &= \{\emptyset s \emptyset \in \prod_{y \in \emptyset \cup S \cup \emptyset} D_y \mid \emptyset s \in \mathcal{D}_1, s \emptyset \in \mathcal{D}_2 \text{ a } s \in \prod_{y \in S} D_y\} \\ &= \{s \in \prod_{y \in S} D_y \mid s \in \mathcal{D}_1 \text{ a } s \in \mathcal{D}_2\} = \mathcal{D}_1 \cap \mathcal{D}_2.\end{aligned}$$

□

důsledek (ekvivalence výrazů) za předpokladu stejných schémat argumentů:

- $\text{SELECT * FROM foo INTERSECT SELECT * FROM bar} \equiv \text{SELECT * FROM foo NATURAL JOIN bar}$
- $\text{foo JOIN bar} \equiv \text{foo INTERSECT bar}$

## Speciální případy spojení: Restrikce na rovnost

Věta (Vztah spojení a restrikce na rovnost)

Pokud je  $\mathcal{D}$  relace na schématu  $R$ ,  $y \in R$  a  $d \in D_y$ , pak  $\sigma_{y=d}(\mathcal{D}) = \mathcal{D} \bowtie \{\{\langle y, d \rangle\}\}$ .

Důkaz.

Pro  $\mathcal{D}$  na  $R$ ,  $T = \emptyset$  a  $S = \{y\}$  dle definice spojení dostáváme:

$$\begin{aligned}\mathcal{D} \bowtie \{\{\langle y, d \rangle\}\} &= \{rst \in \prod_{y \in R \cup S \cup T} D_y \mid rs \in \mathcal{D}, st \in \{\{\langle y, d \rangle\}\} \text{ a } s \in \prod_{y \in S} D_y\} \\ &= \{rs \in \prod_{y \in R} D_y \mid rs \in \mathcal{D} \text{ a } s = \{\langle y, d \rangle\}\} \\ &= \{rs \in \mathcal{D} \mid s = \{\langle y, d \rangle\}\} = \{r \in \mathcal{D} \mid r(y) = d\} = \sigma_{y=d}(\mathcal{D}).\end{aligned}$$

□

**poznámky:**

- restrikci na rovnost lze vyjádřit pomocí spojení se singletonem (PŘEDNÁŠKA 2)
- $\langle \text{relační-výraz} \rangle \text{ WHERE } \langle \text{atribut} \rangle = \langle \text{hodnota} \rangle \equiv \langle \text{relační-výraz} \rangle \text{ JOIN RELATION } \{\text{TUPLE } \{\langle \text{atribut} \rangle \langle \text{hodnota} \rangle\}\}$

## Intermezzo: Přejmenování $n$ -tic

neformálně:

Přejmenováním  $n$ -tice zkonestruujeme novou  $n$ -tici, která obsahuje stejná data (stejných typů), ale může mít jiná jména atributů.

### přejmenování $n$ -tice, angl.: tuple renaming

Mějme  $n$ -tici  $r \in \prod_{y \in R} D_y$  a injektivní zobrazení  $h: R \rightarrow Y$ . Pak složené zobrazení  $\rho_h(r) = h^{-1} \circ r$  nazveme přejmenování  $n$ -tice  $r$  podle  $h$ .

**význam:**  $\rho_h(r)$  je  $n$ -tice nad schématem  $h(R)$ , kde

$$(\rho_h(r))(h(y)) = r(y), \text{ pro každé } y \in R.$$

**Tutorial D:**

$\langle n\text{-ticový-výraz} \rangle \text{ RENAME } \{\langle \text{staré-jméno}_1 \rangle \text{ AS } \langle \text{nové-jméno}_1 \rangle, \dots\}$   
 $\langle n\text{-ticový-výraz} \rangle \text{ RENAME } \{\text{PREFIX } \langle \text{starý-řetězec} \rangle \text{ AS } \langle \text{nový-řetězec} \rangle\}$

## Přejmenování

Definice (přejmenování, angl.: *renaming*)

Mějme relaci  $\mathcal{D}$  na schématu  $R$  a injektivní zobrazení  $h: R \rightarrow Y$ . Položíme:

$$\rho_h(\mathcal{D}) = \{\rho_h(r) \mid r \in \mathcal{D}\}.$$

Relace  $\rho_h(\mathcal{D})$  se nazývá **přejmenování  $\mathcal{D}$  podle  $h$** . Pro jednoduchost někdy značíme  $\rho_{y'_1 \leftarrow y_1, \dots, y'_n \leftarrow y_n}(\mathcal{D})$  pokud  $h(y_i) = y'_i$  ( $i = 1, \dots, n$ ) a  $h(y) = y$  jinak.

**Tutorial D:**

*(relační-výraz)* **RENAME** {*<staré-jméno<sub>1</sub>>* **AS** *<nové-jméno<sub>1</sub>>*, ...}  
*(relační-výraz)* **RENAME** {**PREFIX** *<starý-řetězec>* **AS** *<nový-řetězec>*}

**SQL:**

**SELECT** *<staré-jméno<sub>1</sub>>* **AS** *<nové-jméno<sub>1</sub>>*, ... **FROM** *<jméno>*  
**SELECT** *<staré-jméno<sub>1</sub>>*  $\sqcup$  *<nové-jméno<sub>1</sub>>*, ... **FROM** *<jméno>*

## Odvozené operace: Kartézský součin

Definice (kartézský součin, angl.: *cross join*)

Mějme relace  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na schématech  $R$  a  $T$  takových, že  $R \cap T = \emptyset$ . Pak  $\mathcal{D}_1 \bowtie \mathcal{D}_2$  se spojení nazývá **kartézský součin relací  $\mathcal{D}_1$  a  $\mathcal{D}_2$** .

**podle definice spojení:**

$$\begin{aligned}\mathcal{D}_1 \bowtie \mathcal{D}_2 &= \{rst \mid rs \in \mathcal{D}_1, st \in \mathcal{D}_2 \text{ a } s \in \prod_{y \in S} D_y\} \\ &= \{r\emptyset t \mid r\emptyset \in \mathcal{D}_1 \text{ a } \emptyset t \in \mathcal{D}_2\} \\ &= \{rt \mid r \in \mathcal{D}_1 \text{ a } t \in \mathcal{D}_2\}\end{aligned}$$

**poznámky:**

- speciální případ  $\bowtie$  pro  $S = \emptyset$  (disjunktní schémata)
- každá  $n$ -tice z  $\mathcal{D}_1$  je spojitelná s každou  $n$ -ticí z  $\mathcal{D}_2$
- $|\mathcal{D}_1 \bowtie \mathcal{D}_2| = |\mathcal{D}_1| \cdot |\mathcal{D}_2|$  (pro obecná spojení platí pouze  $\leq$ )

## Kartézský součin v Tutorial D a SQL

### Tutorial D:

$\langle \text{relační-výraz}_1 \rangle \text{ TIMES } \langle \text{relační-výraz}_2 \rangle$   
 $\text{TIMES } \{ \langle \text{relační-výraz}_1 \rangle, \langle \text{relační-výraz}_2 \rangle, \dots \}$

### SQL:

`SELECT * FROM jméno1, jméno2`  
`SELECT * FROM jméno1 CROSS JOIN jméno2`

### poznámky:

- **TIMES** je ekvivalentní **JOIN**, ale testuje disjunktnost schémat
- SQL povolí provést i nad schématy se společnými atributy (výsledek obsahuje více sloupců stejných jmen, lze odstranit přejmenováním)
- obecně lze přejmenováním atributů „vynutit kartézský součin“

## Kartézský součin: Tři různé pojmy

pojem „kartézský součin“ používáme ve třech různých významech:

- ① **naivní kartézský součin** (dvou nebo více množin v daném pořadí)
  - značení:  $A \times B, A_1 \times \dots \times A_n$
  - formalizace: množina všech uspořádaných dvojic ( $n$ -tic) prvků z daných množin
  - použití: pro zavedení pojmu *zobrazení*
- ② **obecný kartézský součin** (indexovaného systému množin)
  - značení:  $\prod_{i \in I} A_i$
  - formalizace: množina všech zobrazení  $f: I \rightarrow \bigcup_{i \in I} A_i$  kde  $f(i) \in A_i$  ( $i \in I$ )
  - použití: pro zavedení pojmu *relace nad relačním schématem*
- ③ **kartézský součin relací nad relačními schématy** (operace s relacemi)
  - značení:  $\mathcal{D}_1 \bowtie \mathcal{D}_2$
  - formalizace: *přirozené spojení* relací nad disjunktními schématy
  - použití: relační dotazování

**zřejmě:**  $\mathcal{D}_1 \times \mathcal{D}_2 \neq \mathcal{D}_1 \bowtie \mathcal{D}_2$  (neplést, !!)

## Speciální případy: Polospojení

Definice (polospojení, angl.: *semijoin*)

Mějme relace  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na schématech  $R_1$  a  $R_2$ . Položme:

$$\mathcal{D}_1 \ltimes \mathcal{D}_2 = \pi_{R_1}(\mathcal{D}_1 \bowtie \mathcal{D}_2).$$

Relace  $\mathcal{D}_1 \ltimes \mathcal{D}_2$  se nazývá **polospojení  $\mathcal{D}_1$  a  $\mathcal{D}_2$  (v tomto pořadí)**.

**Tutorial D:**

$\langle$ relační-výraz $_1\rangle$  MATCHING  $\langle$ relační-výraz $_2\rangle$

**SQL:**

SELECT DISTINCT  $\langle jméno_1 \rangle . *$  FROM  $\langle jméno_1 \rangle$  NATURAL JOIN  $\langle jméno_2 \rangle$

**význam:**

- $r_1 \in \mathcal{D}_1 \ltimes \mathcal{D}_2$  právě tehdy, když  $r_1 \in \mathcal{D}_1$  a  $r_1$  je spojitelná s nějakou  $r_2 \in \mathcal{D}_2$

### Příklad (Příklady polospojení)

FOO	BAR	BAZ		BAR	BAZ	QUX	
444	ghi	103		abc	111	zzz	
555	def	102		def	102	www	
555	ghi	103		def	102	yyy	
666	abc	101		ghX	103	xxx	
				ghi	103	ttt	
				ghi	103	uuu	
				ghi	103	vvv	

×

	FOO	BAR	BAZ		BAR	BAZ	QUX
	444	ghi	103		def	102	www
	555	def	102		def	102	yyy
	555	ghi	103		ghi	103	ttt
	666	abc	101		ghi	103	uuu
					ghi	103	vvv

=

BAR	BAZ	QUX		FOO	BAR	BAZ	
abc	111	zzz		444	ghi	103	
def	102	www		555	def	102	
def	102	yyy		555	ghi	103	
ghX	103	xxx		666	abc	101	
ghi	103	ttt					
ghi	103	uuu					
ghi	103	vvv					

×

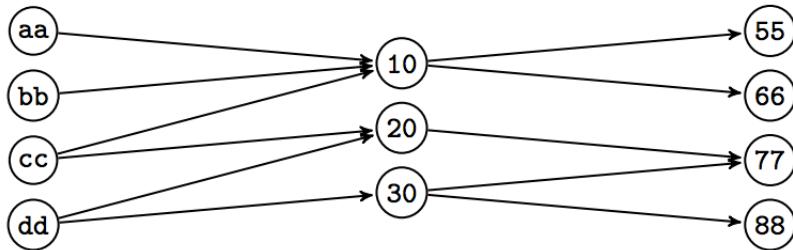
	FOO	BAR	BAZ		BAR	BAZ	QUX
	def	102	www		def	102	yyy
	def	102	yyy		ghi	103	ttt
	ghX	103	xxx		ghi	103	uuu
	ghi	103	ttt		ghi	103	vvv
	ghi	103	uuu				
	ghi	103	vvv				

=

## Intermezzo: Kompozice binárních relací

**pojem související s binárními relacemi:**

- **skládání (kompozice) binárních relací** – mějme binární relace  $R_1 \subseteq A \times B$  a  $R_2 \subseteq B \times C$  (množiny uspořádaných dvojic). Pak kompozicí  $R_1 \circ R_2$  relací  $R_1$  a  $R_2$  (v tomto pořadí) rozumíme binární relaci  $R_1 \circ R_2 \subseteq A \times C$  definovanou:  
$$R_1 \circ R_2 = \{\langle a, c \rangle \in A \times C \mid \text{existuje } b \in B \text{ tak, že } \langle a, b \rangle \in R_1 \text{ a } \langle b, c \rangle \in R_2\}.$$
- skládání hran v grafu, násobení Booleovských matic (sousednosti), ...



## Intermezzo: Kompozice $n$ -tic

**motivace:**

Lze zavést pojem „skládání relací nad relačními schématy“ tak, aby souvisel s pojmem skládání binárních relací (formalizovaných jako podmnožiny naivních kartézských součinů dvou množin)?

**složení (kompozice)  $n$ -tic, angl.: tuple composition**

Mějme  $n$ -tice  $r \in \prod_{y \in R} D_y$  a  $s \in \prod_{y \in S} D_y$ , které jsou spojitelné. Pak zobrazení  $(r \cup s)((R \setminus S) \cup (S \setminus R))$  nazveme složení (kompozice)  $n$ -tic  $r$  a  $s$ .

**Tutorial D:**

$\langle n\text{-ticový-výraz}_1 \rangle \text{ COMPOSE } \langle n\text{-ticový-výraz}_2 \rangle$

**vlastnosti:**

- skládání  $n$ -tic je komutativní (výsledek spojení a projekce  $n$ -tic)

## Odvozené operace: Kompozice

Definice (složení (kompozice), angl.: *composition*)

Mějme relace  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na schématech  $R \cup S$  a  $S \cup T$  tak, že  $R \cap T = \emptyset$ . Položme:

$$\mathcal{D}_1 \circ \mathcal{D}_2 = \pi_{R \cup T}(\mathcal{D}_1 \bowtie \mathcal{D}_2).$$

Relace  $\mathcal{D}_1 \circ \mathcal{D}_2$  na  $R \cup T$  se nazývá **složení (kompozice) relací  $\mathcal{D}_1$  a  $\mathcal{D}_2$** .

**Tutorial D:**

*(relační-výraz<sub>1</sub>) COMPOSE (relační-výraz<sub>2</sub>)*  
*COMPOSE {⟨relační-výraz<sub>1</sub>⟩, ⟨relační-výraz<sub>2</sub>⟩, ...}*

**SQL:**

```
SELECT DISTINCT ⟨jméno1⟩.⟨R-atribut1⟩, ..., ⟨jméno1⟩.⟨R-atributm⟩,  
                  ⟨jméno2⟩.⟨T-atribut1⟩, ..., ⟨jméno2⟩.⟨T-atributn⟩  
FROM ⟨jméno1⟩ NATURAL JOIN ⟨jméno2⟩
```

## Úplné spojení

**motivace:**

V některých spojeních jsou všechny  $n$ -tice z výchozích tabulek spojitelné s některými  $n$ -ticemi z ostatních tabulek – vede na pojem úplné spojení

**nespojitelná  $n$ -tice, angl.: *dangling tuple***

Mějme relace  $\mathcal{D}_1, \dots, \mathcal{D}_2$  na schématech  $R_1, \dots, R_n$ . Pak  $n$ -tice  $r_i \in \mathcal{D}_i$  se nazývá nespojitelná vzhledem k  $\bowtie_{i=1}^n \mathcal{D}_i$  pokud neexistují žádné  $r_j \in \mathcal{D}_j$  ( $j \neq i$ ) tak, že  $r_1 \cup \dots \cup r_n \in \bowtie_{i=1}^n \mathcal{D}_i$ .

**úplné spojení, angl.: *complete join***

Relace  $\mathcal{D}_1, \dots, \mathcal{D}_2$  na schématech  $R_1, \dots, R_n$  mají úplné spojení, pokud žádná  $\mathcal{D}_i$  neobsahuje nespojitelnou  $n$ -tici vzhledem k  $\bowtie_{i=1}^n \mathcal{D}_i$ .

**poznámka:**  $\mathcal{D}_1$  a  $\mathcal{D}_2$  mají úplné spojení p. k.  $\mathcal{D}_1 \ltimes \mathcal{D}_2 = \mathcal{D}_1$  a  $\mathcal{D}_2 \ltimes \mathcal{D}_1 = \mathcal{D}_2$ .

## Příklad (Úplné spojení)

relace, které nemají úplné spojení:

$\mathcal{D}_1$			$\mathcal{D}_2$			$\mathcal{D}_1 \bowtie \mathcal{D}_2$				$\mathcal{D}_1 \ltimes \mathcal{D}_2$			$\mathcal{D}_2 \ltimes \mathcal{D}_1$		
FOO	BAR	BAZ	BAR	BAZ	QUX	FOO	BAR	BAZ	QUX	FOO	BAR	BAZ	BAR	BAZ	QUX
444	ghi	103	abc	111	zzz	444	ghi	103	ttt	444	ghi	103	def	102	www
555	def	102	def	102	www	444	ghi	103	uuu	555	def	102	def	102	yyy
555	ghi	103	def	102	yyy	444	ghi	103	vvv	555	ghi	103	ghi	103	ttt
666	abc	101	ghx	103	xxx	555	def	102	www	555	def	102	ghi	103	uuu
			ghi	103	ttt	555	ghi	103	ttt	555	ghi	103	ghi	103	vvv
			ghi	103	uuu	555	ghi	103	uuu						
			ghi	103	vvv	555	ghi	103	vvv						

dle předchozího tvrzení:

- relace  $\mathcal{D}_3 = \mathcal{D}_1 \ltimes \mathcal{D}_2$  a  $\mathcal{D}_4 = \mathcal{D}_2 \ltimes \mathcal{D}_1$  mají úplné spojení  $\mathcal{D}_1 \bowtie \mathcal{D}_2 = \mathcal{D}_3 \bowtie \mathcal{D}_4$

## Bezeztrátová dekompozice relace

**motivace:**

Duální pojem k pojmu úplné spojení: Je možné vyjádřit výchozí relace pomocí spojení některých jejich projekcí?

**bezeztrátová dekompozice, angl.: nonloss decomposition**

Relace  $\mathcal{D}$  na schématu  $R_1 \cup \dots \cup R_n$  má bezeztrátovou dekompozici vzhledem k množině schémat  $\{R_1, \dots, R_n\}$  pokud  $\mathcal{D} = \bowtie_{i=1}^n \pi_{R_i}(\mathcal{D})$ .

**existence dekompozic:**

- každá  $\mathcal{D}$  na  $R$  má (jednoprvkovou) bezeztrátovou dekompozici vzhledem k  $\{R\}$  (triviální případ; hledáme dekompozice vzhledem k větším množinám schémat)

**další typy dekompozic:**

- horizontální/vertikální typy dekompozic, faktorové dekompozice, ...

## Odvozené operace: $\theta$ -spojení

### motivace:

*Chceme spojovat data ze dvou tabulek tak, aby kritérium spojitelnosti nebylo dáno pouze rovností na určitých hodnotách, ale obecnou podmínkou vztahující se na hodnoty n-tic z obou tabulek.*

### Definice ( $\theta$ -spojení, angl.: $\theta$ -join)

Mějme relace  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na schématech  $R$  a  $T$  takových, že  $R \cap T = \emptyset$  a nechť  $\theta$  je skalární výraz typu „pravdivostní hodnota“, který může obsahovat jména atributů z  $R \cup T$ . Pak  **$\theta$ -spojení  $\mathcal{D}_1$  a  $\mathcal{D}_2$  splňující  $\theta$**  je relace  $\sigma_\theta(\mathcal{D}_1 \bowtie \mathcal{D}_2)$  a označujeme ji  $\mathcal{D}_1 \bowtie_\theta \mathcal{D}_2$ .

### poznámky:

- $\bowtie_\theta$  je definováno jako *restrikce z kartézského součinu*
- podle definice  $\sigma_\theta$  a  $\bowtie$ :

$$\mathcal{D}_1 \bowtie_\theta \mathcal{D}_2 = \{rt \mid r \in \mathcal{D}_1 \text{ a } r \in \mathcal{D}_2 \text{ tak, že } rt \text{ splňuje } \theta\}.$$

## Odvozené operace: Spojení na rovnost

### motivace:

*Speciální případ  $\theta$ -spojení, ve kterém je podmínka  $\theta$  formulována jako konjunkce podmínek vyjadřující rovnost hodnot atributů stejných typů.*

### Definice (spojení na rovnost, angl.: equijoin)

Mějme relace  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na schématech  $R$  a  $T$  takových, že  $R \cap T = \emptyset$  a nechť  $\{y_1, \dots, y_n\} \subseteq R$ ,  $\{y'_1, \dots, y'_n\} \subseteq T$  tak, že atributy mají  $y_i$  a  $y'_i$  stejný typ. Pak  $\theta$ -spojení  $\mathcal{D}_1$  a  $\mathcal{D}_2$ , kde  $\theta$  je v tvaru  $y_1 = y'_1 \wedge \dots \wedge y_n = y'_n$  nazýváme **spojení  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na rovnost** (atributů  $y_1, y'_1$  až  $y_n, y'_n$ ) a označujeme jej  $\mathcal{D}_1 \bowtie_{y_1=y'_1, \dots, y_n=y'_n} \mathcal{D}_2$ .

### Tutorial D:

$(\langle \text{relační-výraz}_1 \rangle \text{ JOIN } \langle \text{relační-výraz}_2 \rangle) \text{ WHERE } \langle \text{podmínka} \rangle$

### SQL:

`SELECT * FROM jméno1, jméno2 WHERE podmínka`

## Intermezzo: Tranzitivní uzávěr

### tranzitivní uzávěr relace na množině, angl.: *transitive closure*

Mějme binární relaci  $R \subseteq A \times A$ . Pak tranzitivním uzávěrem  $R$ , rozumíme binární relací  $R^\infty \subseteq A \times A$  splňující následující podmínky:

- ①  $R \subseteq R^\infty$ ,
- ②  $R^\infty$  je tranzitivní,
- ③  $R^\infty$  je nejmenší relace splňující ① a ②.

**konstruktivně** lze  $R^\infty$  vyjádřit jako

$$R^\infty = \bigcup_{n=1}^{\infty} R^n = \bigcup_{n=1}^{\infty} \underbrace{R \circ \cdots \circ R}_{n\text{-krát}},$$

**přitom platí:**

- pokud je  $R$  konečná, pak existuje index  $m$  tak, že  $R^\infty = \bigcup_{n=1}^m R^n$
- pokud je navíc  $R$  reflexivní, pak  $R^\infty = R^m$

## Tranzitivní uzávěr relací

### Definice (tranzitivní uzávěr relace, angl.: *transitive closure*)

Mějme relaci  $\mathcal{D}$  nad dvouprvkovým schématem  $R = \{y_1, y_2\}$ . Pak relace  $\mathcal{D}^\infty$  nad relačním schématem  $R$  splňující následující podmínky:

- ①  $\mathcal{D} \subseteq \mathcal{D}^\infty$ ,
- ②  $\rho_{y \leftarrow y_1}(\mathcal{D}^\infty) \circ \rho_{y \leftarrow y_2}(\mathcal{D}^\infty) \subseteq \mathcal{D}^\infty$ ,
- ③  $\mathcal{D}^\infty$  je nejmenší relace splňující ① a ②,

se nazývá **tranzitivní uzávěr relace  $\mathcal{D}$** .

**Tutorial D:**

**TCLOSE** (*<relační-výraz>*)

**poznámky:**

- SQL nepodporuje přímo jako relační operaci (ale je definovatelné)
- důležitý aspekt: ukázat *konstruktivní předpis* pro výpočet **TCLOSE**

## Odvozená operace: Polorozdíl

Definice (polorozdíl, angl.: *semidifference*)

Mějme relace  $\mathcal{D}_1$  a  $\mathcal{D}_2$  na schématech  $R_1$  a  $R_2$ . Položme:

$$\mathcal{D}_1 \setminus \mathcal{D}_2 = \mathcal{D}_1 \setminus \pi_{R_1}(\mathcal{D}_1 \bowtie \mathcal{D}_2).$$

Relace  $\mathcal{D}_1 \setminus \mathcal{D}_2$  se nazývá **polorozdíl  $\mathcal{D}_1$  a  $\mathcal{D}_2$  (v tomto pořadí)**.

**Tutorial D** (zobecňuje MINUS):

`<relační-výraz1> NOT MATCHING <relační-výraz2>`

**SQL:**

```
SELECT * FROM <jméno1>
EXCEPT
SELECT DISTINCT <jméno1>.* FROM <jméno1> NATURAL JOIN <jméno2>
```

**význam:**  $r_1 \in \mathcal{D}_1 \setminus \mathcal{D}_2$  p. k.  $r_1 \in \mathcal{D}_1$  není spojitelná s žádnou  $n$ -ticí z  $\mathcal{D}_2$

### Příklad (Vnější spojení a polorozdíly)

$\mathcal{D}_1$			$\mathcal{D}_2$			levé				oboustranné				pravé			
FOO	BAR	BAZ	BAR	BAZ	QUX	FOO	BAR	BAZ	QUX	FOO	BAR	BAZ	QUX	FOO	BAR	BAZ	QUX
444	ghi	103	abc	111	zzz	444	ghi	103	ttt	444	ghi	103	ttt	444	ghi	103	ttt
555	def	102	def	102	www	444	ghi	103	uuu	444	ghi	103	uuu	444	ghi	103	uuu
555	ghi	103	def	102	yyy	444	ghi	103	vvv	444	ghi	103	vvv	444	ghi	103	vvv
666	abc	101	ghX	103	xxx	555	def	102	www	555	def	102	www	555	def	102	www
			ghi	103	ttt	555	def	102	yyy	555	def	102	yyy	555	def	102	yyy
			ghi	103	uuu	555	ghi	103	ttt	555	ghi	103	ttt	555	ghi	103	ttt
			ghi	103	vvv	555	ghi	103	uuu	555	ghi	103	uuu	555	ghi	103	uuu
						555	ghi	103	vvv	555	ghi	103	vvv	555	ghi	103	vvv
						666	abc	101		666	abc	101		666	abc	111	zzz
										abc	111	zzz		ghX	103	xxx	

$\mathcal{D}_1 \setminus \mathcal{D}_2 =$	FOO	BAR	BAZ
	666	abc	101

 $\mathcal{D}_2 \setminus \mathcal{D}_1 =$ 

BAR	BAZ	QUX
abc	111	zzz
ghX	103	xxx

## 5 Jazyk SQL

Jazyk SQL (Structured Query Language) je jedním z nejrozšířenějších jazyků pro Coddův relační databázový model. Jedná se o deklarativní jazyk umožňující dotazování, definici a modifikaci dat. Většina implementací jazyka obsahuje i procedurální rozšíření (např. PL/pgSQL). Jazyk SQL se poprvé objevil v roce 1974 (pod názvem SEQUEL), což jej činí jedním z nejstarších jazyků pro relační model (základy modelu položil Edgar F. Codd v letech 1970-1972). Jazyk SQL vychází ze základních teoretických dota-zovacích jazyků relačního modelu – relační algebry a n-ticového relačního kalkulu. Od relačního modelu se však jazyk SQL odchyluje a striktně řečeno jej nelze označovat jako relační. Navíc od počátku vývoje tohoto jazyka byl kladen důraz na efektivitu vyhodnocování a dobrý návrh jazyka byl opomíjen. V důsledku má jazyk komplikovanější či logicky nekonzistentní syntaxi (WHERE  $\times$  HAVING, apod.). Jazyk je standardizován, nicméně se lze setkat s nestandardními rozšířeními. Nejnovější revize standardu byla vydána v roce 2011. Jazyk je implementován v mnoha současných komerčních i open-source databázových systémech – používají jej databázové systémy Oracle Database, Microsoft SQL Server, PostgreSQL, MySQL, MariaDB, MySQL, SQLite a další.

### 5.1 Tabulky

#### Vytvoření tabulky

**Vlastnosti sloupců** Na sloupce je možné klást dodatečné podmínky, které musejí data v těchto sloupcích splňovat:

**NOT NULL** – sloupec nesmí obsahovat hodnotu NULL

**PRIMARY KEY** – daný sloupec je primárním klíčem, hodnota umožňuje jednoznačně identifikovat příslušný řádek

**UNIQUE** – daný sloupec musí obsahovat vzájemně různé hodnoty (hodnoty se nesmí opakovat)

**DEFAULT** výraz – implicitní hodnota; pokud není uvedeno, implicitní hodnota je NULL

**CHECK** podmínka – obecná podmínka, kterou musí hodnoty v daném sloupci splňovat

Některá omezení je možné definovat pro více sloupců současně:

**UNIQUE**(*sloupec<sub>i</sub>, …, sloupec<sub>k</sub>*) – kombinace hodnot z těchto sloupců musí být napříč tabulkou unikátní

**PRIMARY KEY**(*sloupec<sub>i</sub>, …, sloupec<sub>k</sub>*) – kombinace hodnot z těchto sloupců tvoří primární klíč

**CHECK** podmínka

— Zakladni syntax vytvoreni tabulky

```
CREATE TABLE nazev_tabulky (
    sloupec_1 datovy_typ_1 [vlastnosti_1] ,
    \dots
    sloupec_n datovy_typ_n [vlastnosti_n] ,
    [omezeni_pro_vice_sloupcu]);
```

— Priklady

```
CREATE TABLE zamestnanci (
    jmeno varchar(30)      PRIMARY KEY,
    vek int                 NOT NULL CHECK (vek > 0),
    plat numeric(8, 2)      NOT NULL CHECK (plat > 8500),
    funkce varchar(20));
```

```
CREATE TABLE katalog_modelu_automobilu (
    vyrobce varchar(40),
    model varchar(40),
    pocet_dveri int CHECK (pocet_dveri > 0),
    PRIMARY KEY (vyrobce, model));
```

## Vkládání do tabulky

-- Zakladni syntax

```
INSERT INTO jmeno_tabulky (sloupec_1, ..., sloupec_n)
    VALUES (hodnota_1, ..., hodnota_n);
```

-- Priklady

```
INSERT INTO zamestnanci (jmeno, vek, plat, funkce)
    VALUES ('Jan', 24, 32345, 'Designer');
```

```
INSERT INTO zamestnanci VALUES ('Tomas', 32, 49999, 'Programator');
```

```
INSERT INTO zamestnanci (jmeno, vek, plat) VALUES ('Jiri', 40, 49999);
```

```
INSERT INTO zamestnanci (jmeno, vek, plat, funkce) VALUES
    ('Pavel', 54, 82345, 'Manager'),
    ('Ota', 32, 21000, 'Udrzbar');
```

## Selekce dat z tabulky

```
SELECT * FROM zamestnanci;
```

## Změna hodnot

-- zakladni synzaxe

```
UPDATE jmeno_tabulky SET sloupec_i = vyraz_i, ..., sloupec_k = vyraz_k;
-- zmena se provede pro vsechny radky
```

```
UPDATE jmeno_tabulky SET sloupec_i = vyraz_i, ..., sloupec_k = vyraz_k
    WHERE podminka;
```

-- zmena se provede pouze u radku splnujici podminku

-- priklady

```
UPDATE zamestnanci SET plat = plat * 1.1;
```

```
UPDATE zamestnanci SET funkce = 'Senior_programator', plat = plat * 2
    WHERE funkce = 'Programator';
```

## Mazání dat

-- Zakladni syntax

```
DELETE FROM jmeno_tabulky;
-- odstrani vsechyn radky
```

```
DELETE FROM jmeno_tabulky WHERE podminka;
-- odstrani pouze redky vyhovujici podmince
```

-- Priklad

```
DELETE FROM zamestnanci WHERE vek > 28;
```

## 5.2 Pohledy

Pohled si lze představit jako virtuální tabulku (relaci), jejíž obsah tvoří výsledek dotazu, který jsme uvedli v definici pohledu. Výsledek dotazu se počítá při každém použití pohledu. Pohled nezabírá (skoro) žádné místo na disku.

### 5.2.1 Vyuziti pohledu

**Abstrakce** Pohledy mohou sloužit k pojmenování opakujícího se dotazu, který se často vyskytuje jako součást jiných dotazů.

**Zajištění kompatibility** V databázi mohlo být nutné změnit schéma některých tabulek. Nicméně se může stát, že aplikace, které tuto databázi využívají, mohou očekávat původní schéma. Aby nebylo nutné aplikace upravovat, pomocí pohledu pojmenujeme dotaz, který z nových tabulek vypočítá původní tabulku.

**Ochrana citlivých údajů** Tabulky mohou obsahovat citlivé údaje (rodná čísla, čísla platebních karet, apod.). Pokud chceme některým uživatelům umožnit přístup pouze k některým datům, vytvoříme pro ně pohled, který citlivá data obsahovat nebude a přístup k základním tabulkám těmto uživatelům zakážeme.

**Optimalizace** Pomocí speciálního typu pohledu – tzv. materializovaného pohledu – je možné provádět jistý druh optimalizací. Představme si následující situaci. Výpočet výsledku dotazu je tak časově náročný, že jej nemůžeme vyhodnocovat interaktivně, protože by uživatel musel na odpověď čekat příliš dlouho. Nicméně výsledek tohoto dotazu i tak potřebujeme mít k dispozici pro interaktivní dotazování. Povaha dotazu však umožňuje výsledek předpočítat a uložit jej (např. nepotřebujeme mít k dispozici zcela aktuální informace, ale postačují informace vypočtené z dat z předchozích dnů). K tomuto účelu lze využít materializovaný pohled. Při jeho definici se vypočítá výsledek dotazu a ten se uloží (výpočet lze odložit). Pokud je to potřeba, je možné výsledek dotazu přepočítat.

### 5.2.2 Syntaxe

```
/* Zakladni syntaxe */
CREATE VIEW nazev_pohledu AS query;

/* Odstraneni pohledu */
DROP VIEW nazev_pohledu;

/* Docasny pohled — automaticky odstranen pri ukonceni sezeni */
CREATE TEMPORARY VIEW nazev_pohledu AS query;

/* Materializovany pohled (PostgreSQL ve verzi 9.3 a novejsi) */
CREATE MATERIALIZED VIEW nazev_pohledu AS query;

/* Materializovany pohled bude vytvoren, dotaz se nebude vyhodnocovat */
CREATE MATERIALIZED VIEW nazev_pohledu AS query WITH NO DATA;

/* Aktualizace dat v materializovanem pohledu */
REFRESH MATERIALIZED VIEW nazev_pohledu;
```

### 5.2.3 Příklad

```
CREATE TABLE customers_all (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(255),
    phone VARCHAR(20),
    cardno VARCHAR(16),
    expdat DATE,
    ccv    VARCHAR(3));

INSERT INTO customers_all (name, email, phone, cardno, expdat, ccv) VALUES
    ('Timothy_H._Lambert', 'TimothyHLambert@a.example',
     '903-748-9005', '4532673594042130', '2019-11-01', '230'),
    ('Alberto_C._Whitney', 'AlbertoCWhitney@b.example',
     '541-210-6628', '4716945870605022', '2016-09-01', '142'),
    ('Lynn_B._Parent', 'LynnBParent@c.example',
     '971-217-5883', '5396092899337578', '2015-10-01', '987');
```

```

ALTER TABLE customers_all
    RENAME COLUMN expdat TO expiration_date;
ALTER TABLE customers_all
    RENAME COLUMN cardno TO card_number;
ALTER TABLE customers_all
    RENAME TO customers_all_2;

CREATE VIEW customers_all AS (
    SELECT id, name, email, phone, card_number
        AS cardno, expiration_date AS expdat, ccv
    FROM customers_all_2);

SELECT * FROM customers_all;

CREATE VIEW customers AS
    (SELECT id, name, email, phone FROM customers_all_2);

SELECT * FROM customers;

CREATE MATERIALIZED VIEW materialized_test AS
    SELECT * FROM customers_all_2;
DELETE FROM customers_all_2
    WHERE expiration_date < '2016-01-01';

SELECT * FROM materialized_test;
REFRESH MATERIALIZED VIEW materialized_test;
SELECT * FROM materialized_test;

```

### 5.3 Indexy

Index je dodatečná datová struktura (není součástí tabulky), kterou je možné využít ke zrychlení některých typů dotazů. Mezi dotazy, které je možné pomocí indexů zrychlit, patří především ty, které obsahují operace porovnání  $=$ ,  $<$ ,  $\leq$ ,  $>$ ,  $\geq$ ; množinové operace (UNION, ...) nebo třídění či odstraňování duplicit. Použití indexů není "zdarma" – indexy zabírají místo na disku a při změnách dat v tabulkách je nutné příslušné indexy aktualizovat. Operace INSERT, UPDATE a DELETE se pro tabulky s indexy částečně zpomalí (čím více indexů na dané tabulce máme, tím pomalejší tyto operace budou). Na druhou stranu operace SELECT se pro tabulky s indexy může (často velmi výrazně) zrychlit.

**Představa.** Uvažujme město, ve kterém si vedou kroniku (tabulku) o narození občanů. Kronika se skládá z jednotlivých stránek, na kterých jsou jednotlivé záznamy o narození. Při narození dítěte kronikář jednoduše zapíše jeho jméno na nejbližší volné místo v kronice (levná operace). Problém nastává v okamžiku, kdy by někdo rád zjistil, kdy se narodil konkrétní člověk. Nemá jinou možnost než celou kroniku prolistovat a projít ji záznam po záznamu (sekvenční přístup). Navíc čím více záznamů v kronice bude, tím dražší bude tuto operaci provést. Situace se zcela změní, pokud máme k dispozici abecedně seřazený rejstřík jmen (dodatečnou datovou strukturu). V takovém případě prohledáme rejstřík (rychlá operace, můžeme použít binární vyhledávání) a zjistíme, na které stránce se nachází požadovaný záznam, nalistujeme danou stránku a nalezneme záznam. Na druhou stranu, zápis do kroniky již není tak přímočarý – kronikář musí kromě zapsání nového záznamu do kroniky ještě přidat nové jméno na správné místo v rejstříku, což již může být relativně drahá operace.

### 5.3.1 Syntaxe

```
-- Vytvoreni  
CREATE [UNIQUE] INDEX jmeno_indexu  
    ON jmeno_tabulky (sloupec_1, ..., sloupec_n);
```

```
-- Smazani  
DROP INDEX jmeno_indexu;
```

## 5.4 Projekce

Výsledkem tohoto dotazu je tabulka zúžená na vybrané sloupce. Pokud chceme vybrat všechny sloupce, můžeme seznam sloupců nahradit symbolem \*.

### 5.4.1 Syntax

Výsledná tabulka může obsahovat duplicitní řádky. Duplicity řádky je možné eliminovat uvedením klíčového slova DISTINCT:

```
SELECT sloupec_1, ..., sloupec_n FROM nazev_tabulky;
```

```
SELECT DISTINCT sloupec_1, ..., sloupec_n FROM nazev_tabulky;
```

## 5.5 Příklady

```
SELECT jmeno, vek FROM zamestnanci;  
SELECT * FROM zamestnanci;
```

## 5.6 Selekcí

Výsledkem tohoto dotazu je tabulka obsahující pouze ty řádky, které vyhovují zadáné podmínce. Selekcí je možné kombinovat s projekcí.

### 5.6.1 Syntax

```
SELECT * FROM nazev_tabulky WHERE podminka;
```

### 5.6.2 Příklady

```
SELECT * FROM zamestnanci  
WHERE funkce = 'Programator';
```

```
SELECT jmeno, plat FROM zamestnanci  
WHERE plat < 20000 OR plat > 40000;
```

## 5.7 Spojení tabulek

Pokud provádíme např. spojení dvou tabulek a tyto tabulky obsahují sloupec stejného jména, nelze se již na tento sloupec odkazovat pouze jeho jménem, ale musíme specifikovat, ze které tabulky má daný sloupec být (pomocí tečkové notace ve tvaru tabulka.sloupec). V rámci dotazu se nesmí vyskytovat dvě tabulky stejného jména, proto v části FROM je možné pro potřeby daného dotazu tabulky pojmenovat:

```
SELECT * FROM tab, tab; --chyba
```

```
SELECT * FROM tab AS t1, tab AS t2;  
--v ramci dotazu se t1 a t2 chovají jako dve ruzne tabulky  
--(obe obsahuji tataz data jako tabulka tab)
```

### 5.7.1 Přirozené spojení

```
/* Prirozeno spojene */
SELECT * FROM tabulka_1 NATURAL JOIN tabulka_2;

/* Ekvivalentni vyjadreni */
-- T_atribut_i = nazvy sloupcu vyskytujicich se pouze v tabulka_2
-- S_atribut_i = nazvy sloupcu, ktere jsou spolecne pro tabulka_1 a tabulka_2
SELECT tabulka_1.* , tabulka_2.T_atribut_1 , ... , tabulka_2.T_atribut_m
FROM tabulka_1 , tabulka_2 WHERE
    tabulka_1.S_atribut_1 = tabulka_2.S_atribut_1
    AND ... AND
    tabulka_1.S_atribut_n = tabulka_2.S_atribut_n;

/* nebo tez: */
SELECT tabulka_1.* , tabulka_2.T_atribut_1 , ... , tabulka_2.T_atribut_m
FROM tabulka_1 [INNER] JOIN tabulka_2 ON
    tabulka_1.S_atribut_1 = tabulka_2.S_atribut_1
    AND ... AND
    tabulka_1.S_atribut_n = tabulka_2.S_atribut_n;
```

### 5.7.2 Přejmenování

```
SELECT stare_jmeno_1 [AS] nove_jmeno_1 , ... ,
       stare_jmeno_n [AS] nove_jmeno_n FROM tabulka;
```

### 5.7.3 Kartézský součin

```
SELECT * FROM tabulka_1 , tabulka_2;
/* ekvivalentne:
SELECT * FROM tabulka_1 CROSS JOIN tabulka_2;
```

### 5.7.4 Polospojení

```
SELECT DISTINCT rel1.* FROM rel1 NATURAL JOIN rel2;
SELECT DISTINCT rel2.* FROM rel1 NATURAL JOIN rel2;
```

### 5.7.5 Kompozice

```
SELECT foo , qux FROM rel1 NATURAL JOIN rel2;
```

## 6 Referenční integrita

Data uložená ve více tabulkách jsou často logicky provázána. Například můžeme mít tabulku odběratelé a tabulku faktury, kde každá faktura přísluší některému odběrateli. Chceme, aby pro každou fakturu odběratel vždy existoval. Je nutné zajistit, aby nebylo možné například vložit fakturu pro neexistujícího odběratele nebo smazat odběratele, ke kterému máme uložené nějaké faktury. K tomuto účelu v jazyce SQL slouží tzv. cizí klíč(e). Omezení dané cizím klíčem vyžaduje, aby hodnoty ve specifikovaném sloupci (nebo sloupcích) odpovídaly hodnotám, které se nacházejí v některém řádku jiné tabulky. Tímto způsobem je zajištěna tzv. referenční integrita mezi dvěma tabulkami (řádky z jedné tabulky se nemohou odkazovat na neexistující řádky z jiné tabulky).

## Integritní omezení

**integritní omezení, angl.: integrity constraints**

Množina integritních omezení relační databáze je konečná množina skalárních výrazů typu „pravdivostní hodnota“. Instance databáze splňuje danou množinu integritních omezení, pokud jsou všechny dané skalární výrazy *pravdivé*.

**dva základní tvary výrazů RA definujících integritní omezení**

- ①  $\mathfrak{E} = \emptyset_R$ , kde  $\mathfrak{E}$  je výraz RA typu  $R$ ;
- ②  $\mathfrak{E} \subseteq \mathfrak{F}$ , kde  $\mathfrak{E}$  a  $\mathfrak{F}$  jsou výrazy RA stejného typu.

**poznámky:**

- $\mathfrak{E} = \emptyset_R$  interpretujeme: „žádná hodnota nesmí splňovat  $\mathfrak{R}$ “
- $\mathfrak{E} \subseteq \mathfrak{F}$  interpretujeme: „každá hodnota splňující  $\mathfrak{E}$  splňuje i  $\mathfrak{F}$ “
- vzájemná převoditelnost ( $\mathfrak{E} = \emptyset_R$  p. k.  $\mathfrak{E} \subseteq \emptyset_R$ ;  $\mathfrak{E} \subseteq \mathfrak{F}$  p. k. jako  $\mathfrak{E} \setminus \mathfrak{F} = \emptyset_R$ ) důsledek adjunkce  $\setminus$  a  $\cup$  (PŘEDNÁŠKA 3)

## Obecná integritní omezení v Tutorial D a SQL

**Tutorial D:**

```
CONSTRAINT <jméno> <skalární-výraz>;
DROP CONSTRAINT <jméno>;
```

**SQL:**

```
ALTER TABLE <jméno-tabulky> ADD
    CONSTRAINT <jméno-omezení> <definice-omezení>;
ALTER TABLE <jméno-tabulky> DROP
    CONSTRAINT <jméno-omezení>;
CREATE TABLE <jméno-tabulky> (
    <definice-atributu> CONSTRAINT <jméno-omezení1> <definice-omezení1>,
    ...
    CONSTRAINT <jméno-omezení_n> <definice-omezení_n>);
```

## Příklad (SQL: Obecná integritní omezení)

```
CREATE TABLE tab (
    foo NUMERIC NOT NULL PRIMARY KEY,
    bar NUMERIC NOT NULL,
    baz VARCHAR NOT NULL);

ALTER TABLE tab ADD
    CONSTRAINT tab_nnega CHECK (foo < 0);

ALTER TABLE tab ADD
    CONSTRAINT tab_fubar CHECK (foo < bar);

INSERT INTO tab VALUES (10, 20, 'aaa');
INSERT INTO tab VALUES (-1, 20, 'bbb'); /* fail */
INSERT INTO tab VALUES (30, 20, 'ccc'); /* fail */

ALTER TABLE tab DROP CONSTRAINT tab_nnega;
ALTER TABLE tab DROP CONSTRAINT tab_fubar;
```

## Příklad (SQL: Omezení přípustných hodnot)

```
CREATE TABLE tab (
    foo NUMERIC NOT NULL PRIMARY KEY CHECK (foo >= 0),
    bar NUMERIC NOT NULL,
    CHECK (foo <= bar),
    baz VARCHAR NOT NULL);

CREATE TABLE tab (
    foo NUMERIC NOT NULL PRIMARY KEY CONSTRAINT c1 CHECK (foo >= 0),
    bar NUMERIC NOT NULL,
    CONSTRAINT c2 CHECK (foo <= bar),
    baz VARCHAR NOT NULL);
```

### poznámky k použití **CHECK**:

- na úrovni *atributu* může odkazovat pouze na hodnotu atributu (v každé  $n$ -tici)
- na úrovni *tabulky* může operovat s hodnotami všech atributů (každé z  $n$ -tic)
- nepodporuje vnořené dotazy (PostgreSQL, !!)

### klíč, angl.: key

Uvažujme relační proměnnou  $X$  typu  $R = \{y_1, \dots, y_n\}$ . Množina klíčů proměnné  $X$  je libovolná neprázdná množina  $\{K_1, \dots, K_n\}$  jejíž prvky jsou podmnožiny  $R$  a splňují podmínu, že  $K_i \not\subseteq K_j$  pro každé  $i \neq j$ .

### relační přiřazení, angl.: relational assignment

Mějme relační proměnnou  $X$  typu  $R$  a nechť  $\{K_1, \dots, K_n\}$  je množina klíčů proměnné  $X$ . Pak relaci  $\mathcal{D}$  typu  $R$  lze **přiřadit jako hodnotu** proměnné  $X$  pokud je splněna následující podmínka: Pro každé  $i = 1, \dots, n$  a libovolné  $r_1, r_2 \in \mathcal{D}$  platí:

pokud  $r_1(y) = r_2(y)$  pro každý  $y \in K_i$ , pak  $r_1 = r_2$ .

V opačném případě říkáme, že  $\mathcal{D}$  porušuje integritní omezení dané některým klíčem relační proměnné  $X$ .

## Referenční integritní omezení

### motivace:

*Hodnoty atributů jedné relace se musí nacházet jako hodnoty (jiných) atributů jiné relace.*

### referenční integritní omezení, angl.: referential integrity constraint

Mějme relační proměnné  $r$  typu  $R$  a  $s$  typu  $S$ . Referenční integritní omezení je výraz ve tvaru  $\rho_f(\pi_{R'}(r)) \subseteq \pi_{S'}(s)$ , kde  $R' \subseteq R$  a  $S' \subseteq S$ .

omezení  $\rho_f(\pi_{R'}(r)) \subseteq \pi_{S'}(s)$  je *splňeno v  $\mathcal{D}$*  pokud pro každou  $r \in r^{\mathcal{D}}$  platí, že existuje  $s \in s^{\mathcal{D}}$  tak, že  $r(f(y)) = s(y)$  pro každý atribut  $y \in S'$

### Příklad (Referenční integritní omezení)

$s$  (studenti),  $p$  (předměty),  $z$  (student má zapsaný předmět)

$\rho_{ID \leftarrow STUDENT\_ID}(\pi_{STUDENT\_ID}(z)) \subseteq \pi_{ID}(s)$

$\rho_{ID \leftarrow COURSE\_ID}(\pi_{COURSE\_ID}(z)) \subseteq \pi_{ID}(p)$

## Referenční integritní omezení v SQL

**SQL (částečně) implementuje pomocí cizích klíčů:** pro  $\rho_f(\pi_{R'}(\mathbf{r})) \subseteq \pi_{S'}(\mathbf{s})$  je

- $R'$  **cízí klíč** relační proměnné  $\mathbf{r}$ , který se odkazuje na atributy  $S'$  proměnné  $\mathbf{s}$
- nutný předpoklad:  $S'$  musí být klíč v  $\mathbf{s}$  (**PRIMARY KEY** nebo **UNIQUE**, !!)

**jednoatributový cízí klíč** (sloupcové omezení):

... **REFERENCES**  $\langle jméno-tabulky \rangle$  ( $\langle atribut \rangle$ )

**víceatributový cízí klíč** (omezení v rámci celé tabulky):

**FOREIGN KEY** ( $\langle \mathbf{r-atribut}_1 \rangle, \langle \mathbf{r-atribut}_2 \rangle, \dots$ )  
**REFERENCES**  $\langle jméno-tabulky \rangle$  ( $\langle \mathbf{s-atribut}_1 \rangle, \langle \mathbf{s-atribut}_2 \rangle, \dots$ )

**poznámka:**

- jako ostatní omezení lze použít s **CONSTRAINT** a **ALTER TABLE**

### Příklad (SQL: Použití REFERENCES)

```
CREATE TABLE student (
    id NUMERIC NOT NULL PRIMARY KEY,
    name VARCHAR NOT NULL,
    major VARCHAR NOT NULL);

CREATE TABLE course (
    id NUMERIC NOT NULL PRIMARY KEY,
    name VARCHAR NOT NULL,
    ver NUMERIC NOT NULL);

/* table with two foreign keys */
CREATE TABLE enrolled (
    student_id NUMERIC NOT NULL REFERENCES student (id),
    course_id NUMERIC NOT NULL REFERENCES course (id),
    year NUMERIC NOT NULL,
    PRIMARY KEY (student_id, course_id, year));
```

### Příklad (SQL: Použití FOREIGN KEY a REFERENCES)

```
CREATE TABLE student (
    id NUMERIC NOT NULL PRIMARY KEY,
    name VARCHAR NOT NULL,
    major VARCHAR NOT NULL);

CREATE TABLE course (
    name VARCHAR NOT NULL,
    ver NUMERIC NOT NULL,
    PRIMARY KEY (name, ver));

CREATE TABLE enrolled (
    student_id NUMERIC NOT NULL REFERENCES student (id),
    c_name VARCHAR NOT NULL,
    c_ver NUMERIC NOT NULL,
    FOREIGN KEY (c_name, c_ver) REFERENCES course (name, ver),
    year NUMERIC NOT NULL,
    PRIMARY KEY (student_id, c_name, c_ver, year));
```

## Chování cizích klíčů v SQL

**předpoklad:** je dáno  $\rho_{y \leftarrow x}(\pi_{\{x\}}(r)) \subseteq \pi_{\{y\}}(s)$

*modifikace proměnné r, které končí chybou:*

- vložení hodnoty  $x$  do  $r$ , která se nenachází mezi hodnotami  $y$  z  $s$   
příklad: vložení výsledku zkoušky pro ID, které nepatří žádnému studentovi
- jako v předchozím případě, pro UPDATE místo DELETE  
příklad: snaha modifikovat ID na hodnotu, která nepatří žádnému studentovi

*modifikace proměnné s, kdy je možné specifikovat chování:*

- pokus smazat z  $s$   $n$ -tici  $s$ , kde  $s(y)$  se stále používá v  $r$   
příklad: smazání studenta, který má stále záznamy o zkoušce
- jako v předchozím případě, pro UPDATE místo DELETE  
příklad: modifikace ID studenta, který má záznamy o zkoušce

## Možnosti reakcí na porušení omezení v SQL

**předpoklad:**

- je dáno  $\rho_{y \leftarrow x}(\pi_{\{x\}}(r)) \subseteq \pi_{\{y\}}(s)$
- snažíme se provést modifikaci s porušující toto omezení

**možné způsoby ošetření:**

- ① **NO ACTION:** **implicitní chování** – okamžité zastavení, nahlášení chyby
- ② **RESTRICT:** **nahlášení chyby** – bez možnosti odložení kontroly (viz dále)
- ③ **CASCADE:** **kaskádování** – nedojde k chybě, ale změna se propaguje do tabulek, ve kterých je hodnota přítomna jako cizí klíč:
  - při **DELETE** se smažou odpovídající  $n$ -tice
  - při **UPDATE** se adekvátně změní hodnoty
- ④ **SET NULL:** **smazání hodnoty z  $n$ -tic** – hodnoty v tabulkách s cizím klíčem budou ne definované (nebezpečné, PŘEDNÁŠKA 5)
- ⑤ **SET DEFAULT:** **nastavení na implicitní hodnotu** – pokud má atribut dánu implicitní hodnotu pomocí **DEFAULT**

## 7 Spolupráce SQL s jinými jazyky

### 7.1 Java

K provázání jazyka Java s databází využívající jazyk SQL slouží API zvané JDBC ( Java Database Connectivity). Java může spolupracovat s celou škálou databází například: MySQL, SQLite, PostgreSQL, .... Java ovšem obsahuje databázi Apache Derby již v JVM. Pro propojení Java aplikace s databází je potřeba tzv. JDBC driver který implementuje veškerou předepsanou funkcionality. Odstíňuje aplikaci od toho s jakou databází spolupracuje.

#### 7.1.1 Připojení k databázi

```
//Connection to MySQL database

Connection con = null;
String connectionURL = "jdbc:mysql://tux.inf.upol.cz:3306/vyjiro00";

try {
    con = DriverManager.getConnection(connectionURL ,
        "admin" , "Password");

    System.out.println("pripojeno");
} catch (SQLException e) {
    e.printStackTrace();
}

//Connect to embeded SQLite database
Class.forName("org.sqlite.JDBC");
connection = DriverManager.getConnection("jdbc:sqlite:blog.sqlite");
```

#### 7.1.2 Vytváření dotazů

Pro vytvoření dotazu nabízí JDBC objekt **Statement**, který ve svém konstruktoru bere sql **String** jako argument.

```

try (Statement stmt = con.createStatement()){
    stmt.addBatch("CREATE_TABLE_person(name, VARCHAR(30), PRIMARY_KEY_(name))");
    stmt.executeBatch(); //execute statement
}

```

Další možností jak vytvářet dotazy jsou tzv. `PreparedStatement`, které nám pomáhají zabránit SQL-injection. Nebo-li jinak řečeno nemůže se stát, že by nám do dotazu, kde se vkládá jméno a heslo, nějaký hacker vložil dotaz který vymaže celou tabulkou apod. Použití `PreparedStatement` je následující:

```

public Author insertAuthor(String name) throws DatabaseException {
String sql = "INSERT INTO [author] ([name]) VALUES (?)";

try (PreparedStatement stmt = connection.prepareStatement(sql)) {
    stmt.setString(1, name);
    stmt.executeUpdate();

} catch (SQLException e) {
    throw new DatabaseException("Author insert failed with message: "
        + e.getMessage());
}
}

```

## 7.2 C#

V C# databáze moc neumím omlouvám se na nekompletní vysvětlení. Čerpáno z [1].

### 7.2.1 připojení k databázi

```

SqlConnectionStringBuilder csb = new SqlConnectionStringBuilder();
csb.DataSource = @"localhost\SQLEXPRESS";
csb.InitialCatalog = "SlovnicekDB";
csb.IntegratedSecurity = true;
string prijovaciRetezec = csb.ConnectionString;

using (SqlConnection prijenci = new SqlConnection(connectionString))
{
    prijenci.Open();
    Console.WriteLine("Aplikace se uspesne prijala k databazi.");
}
Console.ReadKey();

// Connect to SQLite
SQLiteConnection.CreateFile("Database.sqlite");
con = new SQLiteConnection("Data_Source=Database.sqlite;Version=3;");

```

### 7.2.2 Vytváření dotazů

```

SqlCommand prikaz = new SqlCommand();
prikaz.Connection = prijenci;
prikaz.CommandText = "SELECT COUNT(*) FROM Word";
int pocetSlovicek = (int)prikaz.ExecuteScalar();
prijenci.Close();
Console.WriteLine("Pocet slovicek v DB je {0}", pocetSlovicek);

//C# equivalent of Java PreparedStatement
string addDriverStmt = "INSERT INTO drivers (name, adress) VALUES (@name, @adress)";
using (SQLiteCommand cmd = new SQLiteCommand(addDriverStmt, con))

```

```

{
    con . Open () ;

    cmd . Parameters . AddWithValue ( "@name" , name ) ;
    cmd . Parameters . AddWithValue ( "@adress" , adress ) ;
    cmd . ExecuteNonQuery () ;
}
con . Close () ;

```

### Příklad (PostgreSQL, příklad použití v PHP5)

```

$db = pg_connect ("host=slon.inf.upol.cz dbname=mojedb" .
                   "user=vychodil password=heslo");

$query = "SELECT id, jmeno FROM katedra WHERE zkratka = 'KI'";

$result = pg_query ($query);

while ($tuple = pg_fetch_array ($result, NULL, PGSQL_ASSOC)) {
    printf ("ID: %s, JMENO: %s\n",
            $tuple ["id"], $tuple ["jmeno"]);
}

pg_free_result ($result);

pg_close ($db);

```

### Příklad (PostgreSQL, použití v Common LISPU, balík postmodern)

```

(ql:quickload "postmodern")
(use-package :postmodern)

(with-connection '("mojedb" "vychodil" "heslo" "slon.inf.upol.cz")
  (doquery (:select 'id 'jmeno
                    :from 'katedra
                    :where (:= 'zkratka "KI"))
    (id jmeno)
    (format t "ID: ~A, JMENO: ~A~%" id jmeno)))

```

#### **poznámky:**

- <http://marijnhaverbeke.nl/postmodern/> (instalovatelné přes quicklisp)
- odstraňuje „prkenné“ psaní dotazů ve formě řetězců (makra generující dotazy)

## 8 Procedury a triggery

V jazyce PL/pgSQL je možné kromě uživatelsky definovaných funkcí programovat i tzv. triggery, což jsou funkce asociované s konkrétními tabulkami, které se spouští při operacích jako je vložení nových

řádků, apod.

**Definice triggeru.** Funkce, která slouží jako trigger, se definuje standardním způsobem, pouze nesmí mít žádné parametry a návratový typ musí být TRIGGER.

```
CREATE FUNCTION nazev_funkce() RETURNS TRIGGER AS $$  
[DECLARE  
    deklarace] —nepovinný blok deklaraci  
BEGIN  
    prikazy  
END;  
$$ LANGUAGE plpgsql;
```

Jakmile máme příslušnou funkci nadefinovánu, můžeme ji přiřadit ke konkrétní tabulce a konkrétním operacím (INSERT, UPDATE, DELETE). V okamžiku, kdy se provádí daná operace nad danou tabulkou, systém automaticky spustí naši funkci. Navíc můžeme specifikovat, jestli se má daná funkce spustit před (BEFORE) nebo po (AFTER) provedení dané operace. Dále lze určit, jestli bude daná funkce spuštěna nad celou tabulkou (FOR EACH STATEMENT) anebo pro každý vkládaný/modifikovaný/mazaný řádek zvlášť (FOR EACH ROW). Volitelně potom můžeme zadat podmínu, která musí být splněna, aby byl trigger systémem spuštěn.

#### Speciální proměnné

Při spuštění triggeru jsou v rámci funkce k dispozici následující proměnné:

- NEW (typ RECORD) – pro triggery spouštěné pro každý řádek tato proměnná obsahuje při INSERT nebo UPDATE nový řádek, který bude (byl) vložen do databáze nebo změněn. Pro celotabulkové triggery a pro DELETE je proměnná rovna NULL.
- OLD (typ RECORD) – pro triggery spouštěné pro každý řádek tato proměnná obsahuje při UPDATE nebo DELETE původní řádek, který bude (byl) změněn nebo odstraněn. Pro celotabulkové triggery a pro INSERT je proměnná rovna NULL.

```
CREATE TRIGGER nazev_triggeru { BEFORE | AFTER } { udalost [ OR ... ] }  
ON nazev_tabulky  
[ FOR [ EACH ] { ROW | STATEMENT } ]  
[ WHEN ( podminka ) ]  
EXECUTE PROCEDURE nazev_funkce ();
```

#### — Příklady

```
CREATE TRIGGER check_update  
BEFORE INSERT OR UPDATE OF balance ON accounts  
FOR EACH ROW  
EXECUTE PROCEDURE check_account_update();
```

```
CREATE TRIGGER check_update  
BEFORE UPDATE ON accounts  
FOR EACH ROW  
WHEN (OLD.balance IS DISTINCT FROM NEW.balance)  
EXECUTE PROCEDURE check_account_update();
```

```
CREATE TRIGGER log_update  
AFTER UPDATE ON accounts  
FOR EACH ROW  
WHEN (OLD.* IS DISTINCT FROM NEW.*)  
EXECUTE PROCEDURE log_account_update();
```

```
CREATE TABLE platy (  
    jmenoZamestance      TEXT,
```

```

plat          NUMERIC(8, 2),
posledniZmena  TIMESTAMP,
zmenuProvedl   TEXT
);

CREATE FUNCTION platy_stamp() RETURNS trigger AS $$ 
BEGIN
    -- Check that empname and salary are given
    IF NEW.jmenoZamestance IS NULL THEN
        RAISE EXCEPTION 'empname_cannot_be_null';
    END IF;
    IF NEW.plat IS NULL THEN
        RAISE EXCEPTION '%_cannot_have_null_salary', NEW.empname;
    END IF;
    -- Who works for us when she must pay for it?
    IF NEW.plat < 0 THEN
        RAISE EXCEPTION '%_cannot_have_a_negative_salary', NEW.empname;
    END IF;
    -- Do not insert or modify Bill's salary (silently, *evil-laugh*)
    IF NEW.jmenoZamestance = 'Bill' THEN
        RETURN NULL;
    END IF;
    -- Remember who changed the payroll when
    NEW.posledniZmena := current_timestamp;
    NEW.zmenuProvedl := current_user;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER platy_stamp BEFORE INSERT OR UPDATE ON platy
FOR EACH ROW EXECUTE PROCEDURE platy_stamp();

```

## 9 Prostředky pro administraci relačních databázových systémů

Prostředků je nepřeberné množství. Jsou bud konzolové a nebo grafické pro databázový systém MySQL je možné použít MySQLWorkbench (GUI), případně konzolovou verzi. U MySQL dále poskytuje možnost správy pomocí webového rozhraní zvané PHPMyAdmin. Pro administraci databázového systému PostgreSQL máme následující nástroje: pgadmin (GUI), psql (konzole).

### 9.1 Klient psql

PsqI je konzolový klient, který slouží k administraci databázového systému PostgreSQL. Pokud chceme spravovat databázi nejjednodušší možností je se přihlásit na server, na kterém běží PostgreSQL pomocí ssh následovně: `ssh vase_prijmeni@slon.inf.upol.cz`. Potom můžeme spustit klienta zadáním příkazu `psql`. Ovládání klienta je následující:

- `\e` – spustí externí editor (vhodné pro delší SQL příkazy)
- `\i <soubor>` – provede SQL příkazy ze souboru
- `\d` – vypíše všechny tabulky, pohledy, indexy v aktuální databázi
- `\d <objekt>` – vypíše podrobné informace o objektu
- `\?` – nápověda

- \c <databáze> – připojí se k databázi
- \q – ukončí klienta

### Příklad (PostgreSQL, použití interaktivního klienta psql)

```
$ psql -h slon.inf.upol.cz slondb -U vychodil [ENTER]
psql (9.1.9)

slondb=> \connect mojedb [ENTER]

mojedb=> \dt [ENTER]
:
mojedb=> SELECT * FROM katedra WHERE zkratka = 'KI'; [ENTER]
   id   |      jmeno      | zkratka
-----+-----+-----
 1024 | katedra informatiky | KI
(1 row)

mojedb=> \quit [ENTER]
```

Dále si myslím, že není moc co popisovat každý s nás aspoň jednou nějakého klienta používal takže tady je vždycky o čem mluvit.

## 10 Řízení báze dat

### Tři vrstvy architektury databázového systému

#### fyzická vrstva, angl.: *physical level*:

- nejnižší vrstva, zabývá se fyzickým (efektivním a perzistentním) uložením dat
- zajímavá z pohledu implementace DB systému, pro uživatele (téměř) nezajímavá

#### logická vrstva, angl.: *logical level*:

- vrstva mezi fyzickou a externí vrstvou, abstrahuje od fyzického uložení dat

#### externí vrstva, angl.: *external level*:

- definuje, jakým způsobem jsou data reprezentována pro konkrétní uživatele
- umožňuje individuální pohled na databázi (poskytuje individuální služby)

#### **poznámky:**

- budeme se zabývat převážně logickou a externí vrstvou, fyzickou minimálně
- ANSI/SPARC Study Group on Database Management Systems (1975)

## Systém řízení báze dat (SŘBD)

systém řízení báze dat, angl.: *database management system (DBMS)*

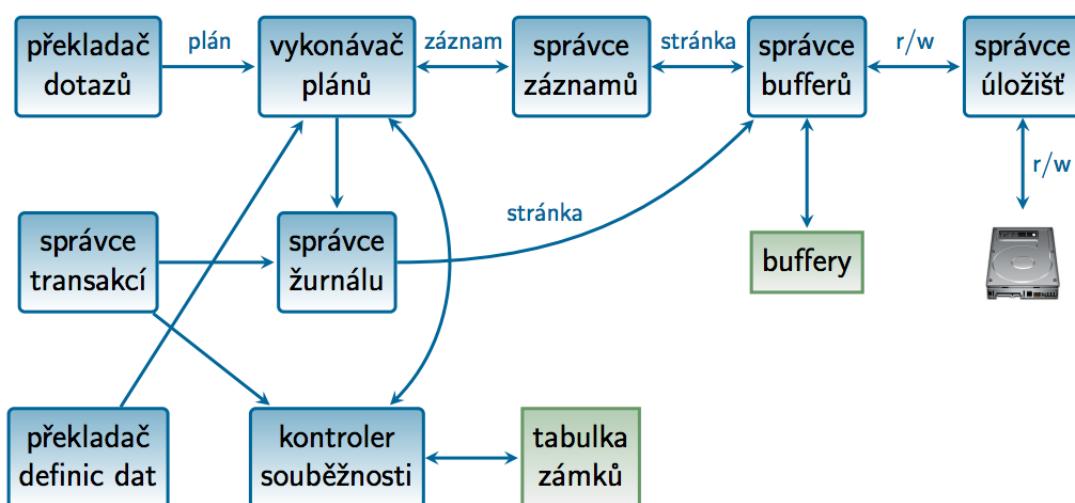
Programový celek implementující databázový systém vycházející z určitého formálního modelu dat a poskytující následující služby:

- souběžný **víceuživatelský přístup** k databázi (neblokované zpracování dotazů),
- **transakční zpracování dat** (atomicita, konzistence, izolace, trvanlivost),
- **perzistentní uložení dat** a systém **zotavení z chyb** (žurnálování dat),
- **integritní omezení** (prevence vytvoření nesmyslných nebo nekonzistentních dat),
- **bezpečnost přístupu k datům** (autorizovaný přístup, šifrování),  
⋮

### poznámky:

- složitosti implementace jsou vyspělé SŘBD srovnatelné s operačními systémy
- „malé (účelově vytvořené) SŘBD“ neposkytují všechny uvedené služby

## Typická struktura SŘBD



## Přehled modelů dat (paradigmat databázových systémů)

- **souborový model** – historicky nejstarší (cca 1955–)  
začátky: Grace M. Hopper, jazyk FLOW-MATIC (později COBOL)
- **síťový model** – Charles Bachman (1969, vývoj trval enormní dobu)  
grafový pohled na schéma databáze (model je komplikovaný a přežitý)
- **hierarchický model** – IBM (cca 1960, přežitý model ale zažívá renesanci, XML)  
lze chápat jako zjednodušení síťového modelu (grafy jsou nahrazeny stromy)
- **relační model** – Edgar F. Codd (1969, rychlý rozmach, dnes mainstream)  
model založený na pojmu  $n$ -ární relace s úzkou vazbou na predikátovou logiku
- **objektové modely** – mnoho modelů, 1989 Statice (Symbolics Inc.)  
perzistentní uložení objektů, obvykle omezené možnosti dotazování
- **relačně/objektové modely** – víc návrhů, různá úroveň, 1990–  
pokusy o doplnění objektových rysů do relačního modelu
- **další modely** – vše, co se nevešlo do předchozí klasifikace (hodně)  
modely pro key-value databáze, semistrukturovaná data, XML databáze, . . .

## Přehled paradigm: Relační model

### charakteristika:

- jeden typ databázových objektů: **relace (nad relačními schématy)** je
  - matematický pojem  $n$ -ární relace = formální protějšek pojmu „datová tabulka“
  - formalizuje **základní data, výsledky dotazů i vztahy mezi daty**
- Codd, E. F.: A relational model of data for large shared data banks  
*Communications of the ACM* 13: 6 (1970)

### vlastnosti:

- + dobrý teoretický model, který lze navíc efektivně implementovat
- + od počátku formalizuje i související fenomény (závislosti v datech, normalizace)
- + **logická nezávislost dat** – fyzická a logická vrstva je oddělena
- + model je **referenčně transparentní**
- + k dispozici hodně kvalitativně různých SŘBD cílených na různou klientelu
- čistý relační model žádný (komerčně nasaditelný) SŘBD neimplementuje

## Příklad („Datové tabulky“ v relačním modelu dat)

jmeno	id	rodne-cislo
Adams	12345	571224/4023
Black	33355	840525/6670
Chang	66066	891117/1024

stuID	rok	predmet	typ
12345	2013	KMI/DATA1	A
12345	2013	KMI/FJ	B
33355	2012	KMI/DATA1	A
33355	2013	KMI/DATA2	B
33355	2013	KMI/PP1	A
66066	2012	KMI/DATA1	C

stuID	rok	predmet	vysl	datum
12345	2013	KMI/DATA1	95%	18/01/13
12345	2013	KMI/FJ	FAIL	25/06/13
12345	2013	KMI/FJ	35%	27/06/13
33355	2012	KMI/DATA1	FAIL	18/01/13
66066	2012	KMI/DATA1	FAIL	19/01/13
66066	2012	KMI/DATA1	85%	06/02/13

## Dostupné relační SŘBD

**uzavřená řešení:** Oracle (Oracle), MS SQL Server (Microsoft), DB2 (IBM), ...

⊕ obvykle dobrá podpora, stabilita, ověřeno dlouhým provozem

⊖ cena, software je blackbox (často obří monolit)

### otevřená řešení:

- Ingres (1973, UC Berkeley, <http://www.actian.com/products/ingres>)
  - komerční podpora od Actian Corporation; podporuje SQL a QUEL
- MariaDB (1995, komunitní fork MySQL, <http://mariadb.org/>)
  - velký počet nasazení, nezávislé storage engines (fyzická vrstva DB)
- PostgreSQL (1985, <http://postgresql.org/>)
  - vyzrálý „velký“ databázový systém, dobrá programovatelnost
- SQLite (2000, <http://sqlite.org/>)
  - embedded databáze, nepotřebuje spuštěný server, nejpoužívanější SQL engine

# PostgreSQL

## historie vývoje:

- UC Berkeley (1986, M. Stonebraker) – projekt navazující na databázi Ingres
- nejprve dotazovací jazyk POSTQUEL, později SQL (Postgres95, PostgreSQL)
- současnost (9. září 2013): verze 9.3

## důležité rysy:

- + drží se standardů: implementuje ISO SQL
- + „velká databáze“ funkčně a výkonově srovnatelná s komerčními produkty
- + stabilita a spolehlivost
- + programovatelnost (PL/pgSQL, PL/Perl, PL/Python)
- + rozšiřitelnost (možnost doprogramovat SŘBD podle potřeb)
- + detailní dokumentace (<http://postgresql.org/docs/manuals/>)
- + platformová nezávislost

## 11 Transakční zpracování dat

Můžeme uvažovat následující problémy:

**1. převod peněz mezi účty** Převod peněz mezi účty potřebujeme vykonat atomicky. Potřebujeme nedělitelně odepsat danou částku z jednoho účtu a připsat ji na druhý účet. Nemůžeme dopustit situaci, kdy bychom odepsali peníze z účtu, ale např. kvůli chybě systémů již peníze nepřipsali na druhý účet. Vyžadujeme, aby byly provedeny buď všechny změny nebo žádná.

**2. rezervace místa v letadle** Databázový systém je běžně využíván mnoha uživateli najednou. Při souběžném přístupu více uživatelů požadujeme co nejfektivnější zpracování jejich požadavků při zachování konzistence dat v databázi.

Rezervace místa v letadle může probíhat (zhruba) takto:

1. `SELECT seat INTO S FROM flights WHERE ... LIMIT 1;`
2. `if found then`  
`UPDATE flights SET ... WHERE seat = ...;`  
`end if`

Může dojít k souběžnému přístupu dvou uživatelů:

U1:     1.           2.  
U2:           1.        2.  
čas: >>>>>>>>>>>>>

Potřebujeme zajistit konzistenci dat – druhý uživatel nesmí získat stejné sedadlo jako první.

## Řešení: Transakce

Transakce lze chápat jako sekvence databázových operací, které splňují ACID:

1. Atomicita – Všechny operace v rámci transakce se provedou buď všechny nebo žádná.
2. Konzistence – Před i po vykonání transakce se databáze nachází v konzistentním stavu. Musí být splněna všechna integritní omezení.
3. Izolace – Jednotlivé transakce jsou izolované. Změny prováděné v průběhu zpracovávání dané transakce nejsou viditelné v ostatních transakcích.
4. Trvalost (durability) – Po úspěšném ukončení transakce jsou data uložena a nemohou být ztracena.

## Transakce v SQL

V jazyce SQL máme pro práci s transakcemi k dispozici následující příkazy:

```
BEGIN; — spousti transakci  
... prikazy v ramci transakce ...  
COMMIT; — ulozi a ukonci transakci  
  
BEGIN;  
... prikazy v ramci transakce ...  
ROLLBACK; — vrati zmeny provedene transakci a ukonci ji
```

Příkaz ROLLBACK může vyvolat i databázový systém při kolizi dané transakce s jinou transakcí. Poznámka: Každý samostatný příkaz je automaticky vykonán v rámci vlastní transakce (např. při vykonání UPDATE nevidíme jednotlivé změny řádků, ale až celou upravenou tabulkou).

## Úrovně izolace

Požadavky ACID jsou relativně silné a jejich zajištění je časově náročné. Proto je možné tyto podmínky zeslabit pomocí tzv. úrovní izolace. Rozeznáváme 4 úrovně (od nejslabší po nejsilnější):

### 1. READ UNCOMMITTED

Transakci je umožněno číst data zapisovaná jinou transakcí, aniž by tato jiná transakce byla ukončena pomocí COMMIT. Čtení takových dat označujeme jako tzv. dirty read. Takto přečtená data mohou být nekonzistentní (čas plyne shora dolů):

Transakce 1	Transakce 2
<pre>SELECT vek FROM lide WHERE id = 5;  vek ----- 20</pre>	
	<pre>UPDATE lide SET vek = 21 WHERE id = 5;</pre>
<pre>SELECT vek FROM lide WHERE id = 5;  vek ----- 21</pre>	

V první transakci jsme zjistili, že věk člověka 5 je 21, ačkoliv v databázi takový záznam neexistuje. Tato úroveň izolace je vhodná pouze pro specifické případy (např. servisní analýza databáze), kdy nepotřebujeme přesná a konzistentní data.

## 2. READ COMMITTED

Na této úrovni izolace již nemůže dojít k dirty read. Data, která přečteme, byla vždy zapsána transakcí, která byla úspěšně ukončena pomocí COMMIT. Avšak může dojít k tzv. non-repeatable read. Pokud v rámci transakce čteme nějaká data vícekrát, může se stát, že se při opakováném čtení pokaždé nevrátí stejný výsledek. Mezi čteními dat naší transakcí totiž mohla být úspěšně ukončena souběžná transakce, která zapsala/upravila/smazala nějaká data.

Transakce 1	Transakce 2
<pre>SELECT vek FROM lide WHERE id = 5;  vek ----- 20</pre>	
	<pre>UPDATE lide SET vek = 21 WHERE id = 5; COMMIT;</pre>
<pre>SELECT vek FROM lide WHERE id = 5;  vek ----- 21</pre>	

## 3. REPEATABLE READ

Na této úrovni je zajištěno, že nedojde k non-repeatable read. Data, která jednou přečteme, se již nemohou při dalším čtení změnit. Může však nastat tzv. phantom read. Pokud v rámci transakce čteme nějaká data vícekrát, může se stát, že výsledek opakovitého čtení bude obsahovat nové řádky, které při

předchozím čtení ve výsledku nebyly. Mezi čteními dat naší transakcí totiž mohla být úspěšně ukončena souběžná transakce, která zapsala nová data.

Transakce 1	Transakce 2
<pre>SELECT jmeno FROM lide WHERE vek BETWEEN 10 AND 30;  jmeno ----- Jan Eva</pre>	
	<pre>INSERT INTO lide VALUES ( 'Oto', 24 ); COMMIT;</pre>
<pre>SELECT jmeno FROM lide WHERE vek BETWEEN 10 AND 30;  jmeno ----- Jan Eva Oto</pre>	

#### 4. SERIALIZABLE

Nemůže dojít k žádnému výše uvedenému fenoménu, vynucuje ACID.

##### Shrnutí a poznámky

Přehled možných fenoménů při jednotlivých úrovních izolace:

Úroveň izolace	Dirty read	Non-repeatable read	Phantom read
READ UNCOMMITTED	možný	možný	možný
READ COMMITTED		možný	možný
REPEATABLE READ			možný
SERIALIZABLE			

V PostgreSQL je možné nastavit úroveň izolace dané transakce pomocí `SET TRANSACTION ISOLATION LEVEL`:

## 12 Uzamykací protokoly

**Uzamykací protokol** je soustava pravidel stanovující, kdy může transakce uzamčít, resp. odemčít databázový objekt.

- existují protokoly zajišťující uspořádatelnost vzhledem ke konfliktům a případně i odstraňující nebezpečí zablokování
- **Dvoufázový uzamykací protokol (2PL)**
  1. *Fáze růstu (growing)* - transakce uzamyká podle potřeby objekty, ale žádný neodemyká. Konec této fáze se nazývá *uzamykací bod (lock point)*.
  2. *Fáze zmenšování (shrinking)* - transakce odemyká objekty, ale již nesmí žádný uzamčít.
    - zajišťuje uspořádatelnost vzhledem ke konfliktům, ale nevylučuje možnost zablokování

Modifikace:

**Striktní 2PL** - všechny výlučné zámky uvolňuje transakce až ve stavu potvrzení. Protokol zabraňuje kaskádnímu rušení transakcí (kaskádní rollback).

**Rigorózní 2PL** - všechny zámky uvolňuje transakce až ve stavu potvrzení.

Zjednodušení: lock\_S(Q), ..., upgrade(Q), ..., downgrade(Q), ... unlock(Q)

Jednoduché schéma uzamykání (často používané):

- Požaduje-li transakce operaci *read(Q,q)*, systém nejprve provede uzamykací operaci *lock\_S(Q)* a teprve pak *read(Q,q)*.
- Požaduje-li transakce operaci *write(Q,q)*, systém provede uzamykací operaci *upgrade(Q)*, resp. *lock\_X(Q)* a teprve pak *write(Q,q)*.
- Všechny zámky držené transakcí jsou uvolněny teprve poté, co transakce potvrdí nebo je zrušena.

- **Implementace uzamykání**

Správce uzamykání (lock manager) používající tabulkou zámku (hašovaná tabulka se seznamem uzamčených datových položek a čekajících transakcí + index identifikátorů transakcí).

- **Granularita uzamykání**

**Granularita uzamykání** udává, jak velká část databáze podléhá uzamykací operaci. Typické úrovně jsou řádek tabulky, blok, tabulka, databáze.

Př) Oracle: řádek, tabulka (LOCK TABLE), SQLBase: stránka, databáze

#### 9.4.4. Protokoly založené na časových razítkách

**Časovým razítkem** rozumíme časový údaj vztažený k nějaké události.

- **Podstata**

Transakcím a databázovým objektům jsou přiřazena časová razítka, která nesou informaci o čase určitých operací a potom se používají při zajištění uspořádatelnosti.

- **Protokol s uspořádáním časových razítek (timestamp-ordering)**



*Platí-li pro  $T_i$  a  $T_j$   $TS(T_i) < TS(T_j)$ , pak budou přípustné pouze plány ekvivalentní vzhledem ke konfliktům se sériovým plánem  $T_i, T_j$ .*

#### 9.4.5. Další typy protokolů

- **Protokoly založené na validaci (validation-based)**

- patří mezi optimistické techniky, vhodná pro prostředí, kde většina transakcí pouze čte

**Podstata:** dvě nebo tři fáze transakce (čtení, validace, zápis). Ve fázi validace se ověřuje, zda došlo ke konfliktu s nějakou souběžnou transakcí (použití časových razítek pro začátek fází + informace o modifikovaných datech), uspořádání podle časového razítka pro validaci.

- **Schémata s verzováním**

**Podstata:** Každá operace  $\text{write}(Q,q)$  vytváří novou verzi objektu  $Q$ . Schéma řízení přístupu musí zajistit, že při čtení transakce dostane správnou hodnotu (čtení je vždy úspěšné), zápis může vést na rollback transakce.

**Př) Oracle – kombinace s uzamykáním, použití tzv. rollback segmentů**

## 13

Funkční závislosti: definice, pravdivost v datech, modely, sémantické vyplývání, kanonické modely a jejich generátory, charakterizace sémantického vyplývání, sémantické uzávěry množin atributů, algoritmy pro jejich výpočet (Closure, LinClosure). Funkční závislosti stanovené z dat: báze, redundancy, nalezení minimální báze. Axiomatizace sémantického vyplývání funkčních závislostí:

### 13.1 Funkční závislosti

#### Syntax a sémantika funkčních závislostí

##### Definice (funkční závislost, angl.: *functional dependency*)

Nechť  $R$  je relační schéma. Pak **funkční závislost** nad schématem  $R$  je formule ve tvaru  $A \Rightarrow B$ , kde  $A, B \subseteq R$ .

##### Definice (pravdivost funkční závislosti v datech)

Nechť  $R$  je relační schéma a  $\mathcal{D}$  je relace nad schématem  $R$ . Pak funkční závislost  $A \Rightarrow B$  nad schématem  $R$  je **pravdivá** v  $\mathcal{D}$ , což označujeme  $\mathcal{D} \models A \Rightarrow B$ , pokud pro každé  $n$ -tice  $r_1, r_2 \in \mathcal{D}$  platí:

$$\text{pokud } r_1(A) = r_2(A), \text{ pak } r_1(B) = r_2(B).$$

V opačném případě říkáme, že  $A \Rightarrow B$  neplatí v  $\mathcal{D}$  a píšeme  $\mathcal{D} \not\models A \Rightarrow B$ .

**slovně:**  $\mathcal{D} \models A \Rightarrow B$  právě tehdy, když pro každé dvě  $n$ -tice  $r_1$  a  $r_2$  z  $\mathcal{D}$  platí, že pokud jsou si rovny na všech natributech z množiny  $A$ , pak jsou si rovny i na všech attributech z množiny  $B$ .

#### Triviální funkční závislosti

$A \Rightarrow B$  se nazývá **triviální**, pokud  $\mathcal{D} \models A \Rightarrow B$  pro každou  $\mathcal{D}$  nad  $R \supseteq A \cup B$

##### Věta (Charakterizace triviálních funkčních závislostí)

$A \Rightarrow B$  je triviální právě tehdy, když  $B \subseteq A$ .

##### Důkaz.

Pokud  $B \not\subseteq A$ , pak existuje  $y \in B$  tak, že  $y \notin A$ . Uvažujme  $\mathcal{D} = \{r_1, r_2\}$  nad  $A \cup B$  tak, že  $r_1(z) = 1$  pro každé  $z \in A \cup B$  a

$$r_2(z) = \begin{cases} 1, & \text{pokud } z \in A, \\ 0, & \text{jinak.} \end{cases}$$

Pak zřejmě  $\mathcal{D} \not\models A \Rightarrow B$ , protože  $r_1(A) = r_2(A)$ , ale  $r_1(y) \neq r_2(y)$  a  $y \in B$ .

Obráceně předpokládejme, že  $B \subseteq A$ . Pokud pro  $r_1, r_2 \in \mathcal{D}$  platí, že  $r_1(A) = r_2(A)$ , tím spíš  $r_1(B) = r_2(B)$  a tedy  $\mathcal{D} \models A \Rightarrow B$ . □

## Věta (O pravdivosti funkčních závislostí)

Pro každé  $A, B, C \subseteq R$  a libovolné relace  $\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2$  nad  $R$  platí:

- ① pokud  $\mathcal{D} \models A \Rightarrow B$ , pak  $\mathcal{D} \models A \Rightarrow B \cap C$ ;
- ② pokud  $\mathcal{D} \models A \Rightarrow B$ , pak  $\mathcal{D} \models A \cup C \Rightarrow B$ ;
- ③  $\mathcal{D} \models A \Rightarrow B$  právě tehdy, když  $\mathcal{D} \models A \Rightarrow B \setminus A$ ;
- ④ pokud  $\mathcal{D}_1 \subseteq \mathcal{D}_2$  a  $\mathcal{D}_2 \models A \Rightarrow B$ , pak  $\mathcal{D}_1 \models A \Rightarrow B$ .

### Důkaz.

Předpokládejme, že  $\mathcal{D} \models A \Rightarrow B$  a vezměme libovolné  $r_1, r_2 \in \mathcal{D}$  takové, že  $r_1(A) = r_2(A)$ . Dle předpokladu pak  $r_1(B) = r_2(B)$  a tím spíš  $r_1(B \cap C) = r_2(B \cap C)$ , protože  $B \cap C \subseteq B$ , to dokazuje ① a jednu stranu tvrzení ③. Analogicky ②. Pokud  $\mathcal{D} \models A \Rightarrow B \setminus A$  a platí  $r_1(A) = r_2(A)$ , pak z  $r_1(B \setminus A) = r_2(B \setminus A)$  a  $r_1(A) = r_2(A)$  dostaneme  $r_1(A \cup (B \setminus A)) = r_2(A \cup (B \setminus A))$ , to jest platí ③, protože  $B \subseteq A \cup (B \setminus A)$ . Bod ④ je přímým důsledkem toho, že pro každé  $r_1, r_2 \in \mathcal{D}_1$  platí  $r_1, r_2 \in \mathcal{D}_2$ .  $\square$

## Teorie a modely

### motivace:

Chceme se zabývat tím, které funkční závislosti vyplývají z jiných závislostí.

Primárně se zajímáme o sémantické vyplývání, pro jeho zavedení potřebujeme pojmy teorie a model.

### Definice (teorie, angl.: theory)

Množinu funkčních závislostí (nad schématem  $R$ ) nazveme **teorie** (nad  $R$ ). Pokud je  $\Gamma$  teorie a  $A \Rightarrow B \in \Gamma$ , pak říkáme, že  $A \Rightarrow B$  je **předpokladem** z  $\Gamma$ .

### Definice (model, angl.: model)

Mějme teorii  $\Gamma$ . Relace  $\mathcal{D}$  je **model**  $\Gamma$  pokud pro každou  $A \Rightarrow B \in \Gamma$  platí, že  $\mathcal{D} \models A \Rightarrow B$ . Množinu všech modelů  $\Gamma$  označujeme  $\text{Mod}(\Gamma)$ , to jest:

$$\text{Mod}(\Gamma) = \{\mathcal{D} \mid \text{pro každou } A \Rightarrow B \in \Gamma \text{ platí } \mathcal{D} \models A \Rightarrow B\}.$$

## Příklad (Teorie a modely)

uvažujme následující teorii:

$$\begin{aligned}\Gamma = \{\{&\text{QUX}\} \Rightarrow \{\text{FOO}, \text{BAR}, \text{BAZ}\}, \\ &\{\text{BAR}, \text{BAZ}\} \Rightarrow \{\text{FOO}\}, \\ &\{\text{FOO}, \text{BAZ}\} \Rightarrow \{\text{BAR}\}, \\ &\{\text{FOO}, \text{BAR}\} \Rightarrow \{\text{BAZ}\}\}\end{aligned}$$

potom pro danou  $\Gamma$  například:

FOO	BAR	BAZ	QUX
10	22	a	222
10	33	b	333
10	22	a	444
20	33	a	555

je model

FOO	BAR	BAZ	QUX
10	22	a	222
10	33	b	333
20	22	a	444
20	33	a	555

není model

FOO	BAR	BAZ	QUX
10	22	a	222
10	33	b	222
10	22	a	444
20	33	a	555

není model

## Sémantické vyplývání

Definice (sémantické vyplývání, angl.: *semantic entailment*)

Funkční závislost  $A \Rightarrow B$  **sémanticky plyne** z teorie  $\Gamma$  pokud je  $A \Rightarrow B$  pravdivá v každém modelu  $\Gamma$ , to znamená pokud  $\text{Mod}(\Gamma) \models A \Rightarrow B$ . Fakt, že  $A \Rightarrow B$  sémanticky plyne z  $\Gamma$  značíme  $\Gamma \models A \Rightarrow B$ .

**slovně:**

Funkční závislost  $A \Rightarrow B$  plyne z teorie  $\Gamma$  pokud je  $A \Rightarrow B$  pravdivá v každé relaci, ve které jsou pravdivé všechny formule z teorie  $\Gamma$ .

**speciální případ:**

- pro  $\Gamma = \emptyset$  píšeme  $\models A \Rightarrow B$  místo  $\emptyset \models A \Rightarrow B$
- význam:  $\models A \Rightarrow B$  p. k.  $A \Rightarrow B$  je pravdivá v každé relaci
- z předchozího víme:  $\models A \Rightarrow B$  p. k.  $A \Rightarrow B$  je triviální, to jest  $B \subseteq A$

## Věta (Vlastnosti teorií, modelů a sémantického vyplývání)

Pro libovolné teorie  $\Gamma, \Gamma_1, \Gamma_2$  nad  $R$  platí:

- ① pokud  $\Gamma_1 \subseteq \Gamma_2$ , pak  $\text{Mod}(\Gamma_2) \subseteq \text{Mod}(\Gamma_1)$ ;
- ②  $\text{Mod}(\emptyset)$  je množina všech relací nad  $R$ ;
- ③  $\text{Mod}(\Gamma) \neq \emptyset$  pro každou teorii  $\Gamma$ ;
- ④ pokud  $\Gamma_1 \subseteq \Gamma_2$  a  $\Gamma_1 \models A \Rightarrow B$ , pak  $\Gamma_2 \models A \Rightarrow B$ ;
- ⑤  $\Gamma \models A \Rightarrow B$  pro každou  $A \Rightarrow B \in \Gamma$ .

### Důkaz.

Pokud  $\mathcal{D} \in \text{Mod}(\Gamma_2)$ , pak pro každou  $A \Rightarrow B \in \Gamma_2$  platí, že  $\mathcal{D} \models A \Rightarrow B$ . Jelikož ale  $\Gamma_1 \subseteq \Gamma_2$ , tím spíš platí  $\mathcal{D} \models A \Rightarrow B$  pro každou  $A \Rightarrow B \in \Gamma_1$ , což ukazuje ①; ② plyne přímo z definice; ③ platí, protože  $\mathcal{D} \in \text{Mod}(\Gamma)$  pro každou  $\mathcal{D}$  splňující  $|\mathcal{D}| < 2$ ; ④ je důsledkem ①; ⑤ platí triviálně, protože  $\mathcal{D} \models A \Rightarrow B$  pro každé  $A \Rightarrow B \in \Gamma$  a  $\mathcal{D} \in \text{Mod}(\Gamma)$ .  $\square$

## Kanonické relace a modely

### Definice (kanonická relace, angl.: *canonical relation*)

Pro libovolnou  $M \subseteq R$  definujeme relaci  $\mathcal{D}_M$  nad schématem  $R$  tak, že  $\mathcal{D}_M = \{r_1, r_2\}$ , přitom  $r_1(y) = p$  pro každý  $y \in R$  a

$$r_2(y) = \begin{cases} p, & \text{pokud } y \in M, \\ q, & \text{pokud } y \notin M, \end{cases}$$

přitom  $p, q$  označují dva různé (fixní) prvky domén atributů z  $R$ . Takto zavedenou relaci  $\mathcal{D}_M$  nazveme **kanonická relace** nad schématem  $R$ .

### Definice (kanonický model, angl.: *canonical model*)

Mějme teorii  $\Gamma$ . Kanonická relace  $\mathcal{D}_M$ , která je modelem  $\Gamma$ , se nazývá **kanonický model**  $\Gamma$ . Množinu všech kanonických modelů  $\Gamma$  značíme  $\text{Mod}_C(\Gamma)$ , to jest:

$$\text{Mod}_C(\Gamma) = \{\mathcal{D}_M \mid \mathcal{D}_M \in \text{Mod}(\Gamma)\}.$$

## Věta (Pravdivost funkčních závislostí v kanonických relacích)

Pro libovolné  $A, B, M \subseteq R$  jsou následující tvrzení ekvivalentní:

- ①  $\mathcal{D}_M \models A \Rightarrow B$ ,
- ② pokud  $A \subseteq M$ , pak  $B \subseteq M$ .

### Důkaz.

Nejprve ukážeme, že pro  $\mathcal{D}_M = \{r_1, r_2\}$  a libovolnou  $C \subseteq R$  platí, že  $r_1(C) = r_2(C)$  p. k.  $C \subseteq M$ . Dle definice kanonické relace máme, že  $r_1(C) = r_2(C)$  znamená  $r_2(y) = p$  pro každý  $y \in C$ . To znamená,  $y \in M$  pro každý  $y \in C$ , to jest  $C \subseteq M$ .

S použitím tohoto faktu,  $\mathcal{D}_M \models A \Rightarrow B$  právě tehdy, když  $r_1(A) = r_2(A)$  implikuje  $r_1(B)r_2(B)$ , to platí právě tehdy, když  $A \subseteq M$  implikuje  $B \subseteq M$ . To jest body ① a ② jsou ekvivalentní.  $\square$

### poznámka:

- pozorování slouží ke zjednodušenému vyjádření  $\mathcal{D}_M \models A \Rightarrow B$

## Věta (Charakterizace $\models$ pomocí kanonických modelů)

$\Gamma \models A \Rightarrow B$  právě tehdy, když  $\text{Mod}_C(\Gamma) \models A \Rightarrow B$ .

### Příklad (Kanonické relace odpovídající $\mathcal{D}$ )

FOO	BAR	BAZ	QUX
10	22	a	222
20	33	a	555

FOO	BAR	BAZ	QUX
10	22	a	222
10	22	a	444

FOO	BAR	BAZ	QUX
10	22	a	444
10	33	b	333

FOO	BAR	BAZ	QUX
10	22	a	222
10	33	b	333
10	22	a	444
20	33	a	555

FOO	BAR	BAZ	QUX
10	22	a	222
10	33	b	333

FOO	BAR	BAZ	QUX
10	22	a	444
20	33	a	555

FOO	BAR	BAZ	QUX
10	33	b	333
20	33	a	555

## Příklad (Kanonické relace odpovídající $\mathcal{D}$ )

FOO	BAR	BAZ	QUX
0	0	1	0
1	1	1	1

FOO	BAR	BAZ	QUX
1	1	1	0
1	1	1	1

### Sémantické uzávěry množiny atributů

$$M_\Gamma = \{M \subseteq R \mid D_M \in Mod(\Gamma)\}$$

se nazývá *systém generátorů kanonických modelů*.

**Věta 1.**  $M_\Gamma$  je uzávěrový systém.

**Definice 1.** Pro teorii  $\Gamma$  a  $M \subseteq R$  definujeme  $[M]_\Gamma \subseteq R$  pedpisem :

$$[M]_\Gamma = \cap \{N \in \mathcal{M}_\Gamma \mid M \subseteq N\}$$

kde

$$\mathcal{M}_\Gamma = \{M \subseteq R \mid D_M \in Mod_c(\Gamma)\}$$

**Uzávěrový operátor  $[...]$**  musí splňovat následující 3 podmínky

1. extenzivitu:  $A \subseteq [A]_\Gamma$
2. monotonii: pokud  $A \subseteq B$  pak  $[A]_\Gamma \subseteq [B]_\Gamma$
3. idempotenci:  $[[A]_\Gamma]_\Gamma = [A]_\Gamma$

**Věta 2.** Následující tvrzení jsou ekvivalentní

$$i \quad \Gamma \models A \Rightarrow B$$

$$ii \quad D_{[A]_\Gamma} \models A \Rightarrow B$$

$$iii \quad B \subseteq [A]_\Gamma$$

**Definice 2.** Teorie  $\Gamma_1$  a  $\Gamma_2$  jsou sémanticky ekvivalentní, pišeme  $\Gamma_1 \equiv \Gamma_2$  pokud pro každou  $A \Rightarrow B$  platí  $\Gamma_1 \models A \Rightarrow B$  p.k.  $\Gamma_2 \models A \Rightarrow B$ .

**Věta 3.** Následující tvrzení jsou ekvivalentní:

$$i \quad \Gamma_1 \equiv \Gamma_2$$

$$ii \quad Mod(\Gamma_1) = Mod(\Gamma_2)$$

$$iii \quad Mod_c(\Gamma_1) = Mod_c(\Gamma_2)$$

Definujme posloupnost množin  $M_\Gamma^0, M_\Gamma^1, \dots$

$$M_\Gamma^i = \begin{cases} M & \text{pokud } i = 0 \\ M_\Gamma^{i-1} \cup \bigcup\{B \mid A \Rightarrow B \in \Gamma \text{ a } A \subseteq M_\Gamma^{i-1}\} & \text{jinak} \end{cases}$$

**Věta 4.**  $[M]_\Gamma = M_\Gamma^\infty$  pro každé  $M \subseteq R$ .

**Věta 5.** Máme teorii  $\Gamma$  a vezmeme  $\Gamma' = \{A \ni [A]_\Gamma \mid A \Rightarrow B \in \Gamma\}$ . Pak  $\Gamma \equiv \Gamma'$

## 13.2 Uzávěrové algoritmy

### 13.2.1 Closure

```
repeat
    set stable to true;
    forall the  $A \Rightarrow B \in \Gamma$  do
        if  $A \subseteq M$  then
            set  $M$  to  $M \cup B$ ;
            set stable to false;
            set  $\Gamma$  to  $\Gamma \setminus \{A \Rightarrow B\}$ ;
        end;
    end;
until stable;
return  $M$ ;
```

### 13.2.2 LinClosure

$\text{count}[A \Rightarrow B] \rightarrow \text{integer}$   
 $\text{list}[y] \rightarrow \text{seznam formulí s atributem } y \text{ na levé straně}$

### Příprava

```
vstup:  $M, \Gamma$ 
forall the  $A \Rightarrow B \in \Gamma$  do
    set  $\text{count}[A \Rightarrow B]$  to  $|A|$ 
    if  $A =$  then
        set  $M$  to  $M \cup B$ 
    else
        forall the  $y \in A$  do
            add  $A \Rightarrow B$  to  $\text{list}[y]$ 
        end
    end
end
```

### Algoritmus

```
set update to  $M$ 
while update  $\neq$  do
    choose  $y$  from update
    set update to  $update \setminus \{y\}$ 
    forall the  $A \Rightarrow B \in \text{list}[y]$  do
        set  $\text{count}[A \Rightarrow B]$  to  $\text{count}[A \Rightarrow B] - 1$ 
        if  $\text{count}[A \Rightarrow B] = 0$  then
            set new to  $B \setminus M$ 
            set  $M$  to  $M \cup new$ 
```

```

        set update to update ∪ new
    end
end
return M

```

### Příklad

$$\begin{aligned}\Gamma = \{\{a, b\}\} &\Rightarrow \{c\}, \\ \{a, c\} &\Rightarrow \{f, g\}, \\ \{e\} &\Rightarrow \{f, c\}, \\ \{c, g\} &\Rightarrow \{g, h\}, \\ \{d\} &\Rightarrow \{a, b\}, \\ \{g\} &\Rightarrow \{a, e\}, \\ \{f\} &\Rightarrow \{b\}\end{aligned}$$

Closure:

$$\begin{aligned}[\{d\}]_\Gamma &= ? \\ M &= d, a, b, c, f, g, h, e \\ \Gamma &= 5, 1, 2, 4, 6, 7, 3 \\ [\{d\}]_\Gamma &= R\end{aligned}$$

LinClosure:

count		list
1	II	a 1, 2
2	II	b 1
3	I	c 2, 4
4	II	d 5
5	I	e 3
6	I	f 7
7	I	g 4, 6
		h

$$\begin{aligned}[\{a, c\}]_\Gamma &= ? \\ M &= a, c, f, g, b, h, e \\ \text{update} &= c, f, g, b, h, e \\ y &= a, c, f, g, b \\ \text{new} &= \{f, g\}, \{b\}, \{h\}, \{e\}, \{b\}, \{} \\ [\{a, c\}]_\Gamma &= M\end{aligned}$$

### 13.3 Funkční závislosti stanovené z dat

**Definice 3.** Teorie  $\Gamma$  se nazývá báze  $D$  pokud pro každou  $A \Rightarrow B$  platí:  $\Gamma \models A \Rightarrow B$  p.k.  $D \models A \Rightarrow B$

**Věta 6.** Pokud jsou  $\Gamma_1$  a  $\Gamma_2$  báze  $D$ , pak  $\text{Mod}(\Gamma_1) = \text{Mod}(\Gamma_2)$

*Důkaz:*  $\Gamma_1 \models A \Rightarrow B$  p.k.  $D \models A \Rightarrow B$  p.k.  $\Gamma_2 \neq A \Rightarrow B$  tj. použijeme větu ??.

**Definice 4.** Pro  $D$  a  $M \subseteq R$  zavedeme:  $E_D(M) = \{(r, r') | r(M) = r'(M)\}$ .

$E_D(M)$  je binární relace na  $D$ . Je reflexivní, symetrická, tranzitivní. Pro libovolné  $M$  je  $E_D(M)$  ekvivalence na  $D$ .

**Definice 5.** Dále definujeme  $C_D(M) = \{y \in R | E_D(M) \subseteq E_D(\{y\})\}$ . Jsou-li si  $n$ -tice  $r$  a  $r'$  rovny  $M$ , pak si musí být rovny i na  $y$ .

**Věta 7.**  $C'_D$  je uzávěrový operátor v  $2^R$ .

**Věta 8.** Následující tvrzení jsou ekvivalentní:

- i  $D \models A \Rightarrow B$
- ii  $E_D(A) \subseteq E_D(B)$
- iii  $B \subseteq C_D(A)$

**Věta 9.**  $\Gamma = \{A \Rightarrow C_D(A) | A \subseteq R\}$  je báze  $D$ .

**Definice 6.**  $\Gamma$  je neredundantní báze  $D$ , pokud  $\Gamma$  je báze  $D$  a pro každou  $\Gamma' \subset \Gamma$  platí, že  $\Gamma'$  není báze  $D$ .

**Věta 10.**  $\Gamma$  je neredundantní báze p.k.  $\Gamma$  je báze  $D$  a pro každou  $A \Rightarrow B \in \Gamma$  platí, že  $\Gamma - \{A \Rightarrow B\} \not\models A \Rightarrow B$ .

**Věta 11.**  $\Gamma = \{P \Rightarrow C_D(P) | P \in P_D\}$  je neredundantní báze  $D$ .

Kde  $P_D = \{P \neq C_D(P) | \text{pro každou } Q \in P_D \text{ takovou, že } Q \subset P \text{ platí } C_D(Q) \subseteq P\}$ .

**Definice 7.** Pokud je  $\Gamma$  báze a pro libovolnou  $\Gamma'$ , která je báze  $D$  platí, že  $|\Gamma| \leq |\Gamma'|$ , pak se  $\Gamma$  nazývá minimální báze  $D$ .

**Věta 12.** Pokud pro  $P \in P_D$  a  $A \subseteq R$  platí  $P \subseteq C_D(A)$ , pak  $C_D(A) \cap P = C_D(C_D(A) \cap P)$ .

**Věta 13.** Nechť  $\Gamma'$  je báze  $D$ . Potom pro každou  $P \in P_D$  existuje  $A \Rightarrow B \in \Gamma'$  taková, že  $C_D(A) = C_D(P)$  a  $D_P \not\models A \Rightarrow B$ .

**Věta 14.** Pokud  $P_1, P_2 \in P_D$  a platí  $P_1 \not\subseteq P_2$  a  $P_2 \not\subseteq P_1$ , pak  $C_D(P_1 \cap P_2) = P_1 \cap P_2$ .

**Věta 15.**  $\Gamma = \{P \Rightarrow C_D(P) | P \in P_D\}$  je minimální báze  $D$ .

**Algoritmus pro výpočet minimální báze**

$$\begin{aligned} M_\Gamma^i & i = 0, 1, \dots \\ M_\Gamma^0 &= M \\ M_\Gamma^{i+1} &= M_\Gamma^i \cup \bigcup \{F | E \Rightarrow F \in \Gamma \text{ a } E \subset M_\Gamma^i\} \\ M_\Gamma^\infty &= \bigcup_{i=0}^{\infty} M_\Gamma^i \end{aligned}$$

**Věta 16.**  $M_\Gamma^\infty$  je uzávěrový operátor.

**Věta 17.** Nechť  $\Gamma = \{P \Rightarrow C_D(P) | P \in P_D\}$ . Pak je ekvivalentní:

- i  $M = M_\Gamma^\infty$
- ii bud'  $M = C_D(M)$  nebo  $M \in P_D$

## 14

Armstrongova pravidla, důkazy, odvozená pravidla, dokazatelnost, korektnost, úplnost. Automatické dokazování: simplifikační logika, RAP sekvence a jejich grafická reprezentace. Algoritmy pro nalezení klíčů. Normalizace: Boyce-Coddova normální forma, normalizace dekompozicí a kompozicí schémát.

### 14.1 Armstrongova pravidla, důkazy, odvozená pravidla, dokazatelnost, korektnost, úplnost

#### 14.1.1 Armstrongova pravidla

(Ax)  $\overline{A \cup B \Rightarrow A}$

(Cut)  $\frac{A \Rightarrow B, B \cup C \Rightarrow D}{A \cup C \Rightarrow D}$  pravidlo řezu (pseudotranzitivita)

### 14.1.2 Důkazy

**Definice 8.** Posloupnost formulí  $f_1, \dots, f_n$ , kde:

- $f_i \in \Gamma$
- $f_i$  je instanci ( $Ax$ )
- spadá z některých  $f_j$  a  $f_k$  kde ( $j, k < i$ ) pomocí ( $Cut$ )

je důkaz  $f_n$  z  $\Gamma$

Značení:  $\Gamma \vdash f_n$  (existuje důkaz  $f_n$  z  $\Gamma$ )

### 14.1.3 Odvozená pravidla

$$(\text{Weak}) \quad \frac{A \Rightarrow B}{A \cup C \Rightarrow B}$$

$$(\text{Ref}) \quad \overline{A \Rightarrow A}$$

$$(\text{Add}) \quad \frac{A \Rightarrow B, A \Rightarrow C}{A \Rightarrow B \cup C}$$

$$(\text{Pro}) \quad \frac{A \Rightarrow B \cup C}{A \Rightarrow B}$$

$$(\text{Tra}) \quad \frac{A \Rightarrow B, B \Rightarrow C}{A \Rightarrow C}$$

$$(\text{Acc}) \quad \frac{A \Rightarrow B \cup C, C \Rightarrow D}{A \Rightarrow B \cup C \cup D}$$

$$( ) \quad \frac{A \Rightarrow B}{A \cup C \Rightarrow B \cup C}$$

$$(\text{Sim}) \quad \frac{A \Rightarrow B, C \Rightarrow D}{A \cup (C \setminus B) \Rightarrow D}$$

### 14.1.4 Dokazatelnost

**Definice 9.** Libovolná funkční závislost je dokazatelná pokud existuje důkaz pro tuto funkční závislost.

### 14.1.5 Korektnost

**Definice 10.** Korektností se rozumí, že libovolná závislost, kterou odvodíme pomocí pravidel ( $Ax$ ) a ( $Cut$ ) a pravidel odvozených z ( $Ax$ ) a ( $Cut$ ), platí na každé relační instanci  $r$  z  $R$ , která splňuje závislosti v  $F$ .

Korektnost v podstatě říká, že: jsou-li platné předpoklady pravidla, je platný i důsledek.  
Nebo-li co z množiny  $F$  odvodíme, platí pro libovolnou instanci  $R$ .

**Definice 11.** Odvozovací pravidlo je korektní pokud platí:  $\{A_1 \Rightarrow B_1, \dots, A_n \Rightarrow B_n\} \models A \Rightarrow B$

**Definice 12.** Věta o korektnosti:

$$(\{A_1 \Rightarrow B_1, \dots, A_n \Rightarrow B_n\} \vdash A \Rightarrow B) \Rightarrow (\{A_1 \Rightarrow B_1, \dots, A_n \Rightarrow B_n\} \models A \Rightarrow B)$$

**Věta 18.** Pravidla ( $Ax$ ) a ( $Cut$ ) jsou korektní

### 14.1.6 Úplnost

**Definice 13.** Axiomatický systém je množina závislostí, které splňují pravidla bezespornosti, úplnosti a nezávislosti.

- Úplnost: lze odvodit všechny závislosti v dané oblasti

- Bezespornost: nelze z ax. systému odvodit dvě protikladné tvrzení
- Nezávislost: žádná závislost z tohoto systému není odvoditelná pomocí ostatních

**Definice 14.** *Úplností se rozumí, že opakovaným používáním pravidel některého axiomatického systému při odvozování závislostí (tak dlouho, dokud jde něco odvodit) dostáváme jako výsledek kompletní množinu všech možných závislostí.*

Nebo-li lze jimi odvodit všechny funkční závislosti platné ve všech instancích R vzhledem k danému F.

Některé axiomatické systémy

- $A_1$ : (Ax) (Cut)
- $A_2$ : (Ax) (Sim)
- $A_3$ : (Ax) (Tra) ( )
- $A_3$ : (Ref)(Acc) (Pro)

## 14.2 Automatické dokazování: simplifikační logika, RAP sekvence a jejich grafická reprezentace.

### 14.3 Algoritmy pro nalezení klíčů.

Klíče z pohledu funkčních závislostí:

**Definice 15.** *Mějme relační schéma R a teorii  $\Gamma$  nad R. Pak nadklíč schématu R vzhledem k  $\Gamma$  je libovolná  $K \subseteq R$  taková, že  $\Gamma \models K \Rightarrow R$ . Pokud je K nadklíč R vzhledem k  $\Gamma$  a žádná  $K' \subset K$  není nadklíč R vzhledem k  $\Gamma$ , pak je K klíč R vzhledem k  $\Gamma$ .*

#### Příklad (Nadklíče a klíče)

Pro  $\Gamma = \{\{A\} \Rightarrow \{B, C\}, \{B, D\} \Rightarrow \{E, F\}, \{C, E\} \Rightarrow \{G\}, \{F\} \Rightarrow \{A, B\}, \{D, G\} \Rightarrow \{A, C, H\}\}$  z předchozích příkladů jsou klíče schématu  $R = \{A, \dots, H\}$  vzhledem k  $\Gamma$  následující:

$$K_1 = \{A, D\}, \quad K_2 = \{B, D\}, \quad K_3 = \{C, D, E\}, \quad K_4 = \{D, F\}, \quad K_5 = \{D, G\},$$

protože pro  $K_1$  platí:

$$[\{A, D\}]_\Gamma = R, \quad [\{A\}]_\Gamma = \{A, B, C\} \neq R, \quad [\{D\}]_\Gamma = \{D\} \neq R,$$

to znamená:  $\Gamma \models K_1 \Rightarrow R$  a  $\Gamma \not\models M \Rightarrow R$  pro každou  $M \subset K_1$ , to jest  $K_1$  je klíč.

Analogicky se dá ukázat pro  $K_2, \dots, K_5$ . Například pro  $K_3$  je  $[K_3]_\Gamma = R$  a

$$[\{C, D\}]_\Gamma = \{C, D\} \neq R, \quad [\{C, E\}]_\Gamma = \{C, E, G\} \neq R, \quad [\{D, E\}]_\Gamma = \{D, E\} \neq R.$$

Následující jsou nadklíče, ale nejsou klíče:

$$\{A, D, E\}, \quad \{B, D, G\}, \quad \{C, D, E, F\}, \quad \{D, F, H\}, \quad \{A, \dots, H\}$$

Například  $\{A\}$ ,  $\{A, E\}$ ,  $\{E, F\}$  nejsou nadklíče, protože  $[\{A\}]_\Gamma = \{A, B, C\} \neq R$ ,  $[\{A, E\}]_\Gamma = \{A, B, C, E, G\} \neq R$  a  $[\{E, F\}]_\Gamma = \{A, B, C, E, F, G\} \neq R$ .

### Příklad (Nalezení klíče postupnou redukcí nadklíče)

Pokud je dána  $\Gamma$  nad  $R$ , (některý) klíč schématu  $R$  vzhledem k  $\Gamma$  lze nalézt tak, že vyjdeme z nadklíče  $K = R$  a postupně z něj odebíráme atributy, dokud je množina pořád nadklíč. Pokud už žádný atribut nelze odebrat, výsledná množina  $K$  je klíč.

Pro  $\Gamma = \{\{A\} \Rightarrow \{B, C\}, \{B, D\} \Rightarrow \{E, F\}, \{C, E\} \Rightarrow \{G\}, \{F\} \Rightarrow \{A, B\}, \{D, G\} \Rightarrow \{A, C, H\}\}$  z předchozích příkladů můžeme najít klíče  $K_1, K_2, K_3$  takto:

$\{A, B, C, D, E, F, G, H\}$	$\{A, B, C, D, E, F, G, H\}$	$\{A, B, C, D, E, F, G, H\}$
$\{A, B, \textcolor{blue}{C}, D, E, G, H\}$	$\{A, B, \textcolor{blue}{C}, D, E, F, H\}$	$\{A, B, C, D, E, \textcolor{blue}{G}, H\}$
$\{A, B, D, E, \textcolor{blue}{G}, H\}$	$\{A, B, D, \textcolor{blue}{E}, F, H\}$	$\{\textcolor{blue}{A}, B, C, D, E, H\}$
$\{A, \textcolor{blue}{B}, D, E, H\}$	$\{A, B, D, \textcolor{blue}{F}, H\}$	$\{B, C, D, E, \textcolor{blue}{H}\}$
$\{A, D, E, \textcolor{blue}{H}\}$	$\{A, B, D, H\}$	$\{\textcolor{blue}{B}, C, D, E\}$
$\{A, D, \textcolor{blue}{E}\}$	$\{\textcolor{blue}{A}, B, D\}$	$\{C, D, E\} = K_3$
$\{A, D\} = K_1$	$\{B, D\} = K_2$	

Analogicky pro  $K_4$  a  $K_5$ .

**pozor:** algoritmus závisí na výběru prvku z aktuálního nadklíče (!!)

## 14.4 Normalizace: Boyce-Coddova normální forma, normalizace dekompozicí a kompozicí schémat.

### 14.4.1 Boyce-Coddova normální forma

Připomenutí ostatních NF (Normálních forem)

**Definice 16.** 1NF: Relační schéma obsahuje pouze atomické atributy.

**Definice 17.** 2NF: jestliže je v první normální formě a každý neklíčový atribut je úplně závislý na každém klíči schématu  $R$ . Jinak – žádný neklíčový atribut není závislý na podmnožině žádného klíče  $R$ .

**Definice 18.** 3NF: jestliže je v první normální formě a každý neklíčový atribut je úplně závislý na každém klíči schématu  $R$ . Jinak – žádný neklíčový atribut není závislý na podmnožině žádného klíče  $R$ .

**Definice 19.** 3NF: jestliže je v první normální formě a každý neklíčový atribut je úplně závislý na každém klíči schématu  $R$ . Jinak – žádný neklíčový atribut není závislý na podmnožině žádného klíče  $R$ .

Boyce-Coddova normální forma

Neformálně: omezením třídy relací ve 3NF - vyloučením všech netriviálních funkčních závislostí mezi atributy kromě závislosti na klíči dostaneme relace v Boyce Coddově normální formě.

**Definice 20.** Mějme relační schéma  $R$  a teorii  $\Gamma$ . Pak  $R$  je v Boyce-Coddově normální formě (BCNF) vzhledem k  $\Gamma$ , pokud pro každou netriviální  $A \Rightarrow B \in \Gamma$  platí že  $\Gamma \models A \Rightarrow R$ .

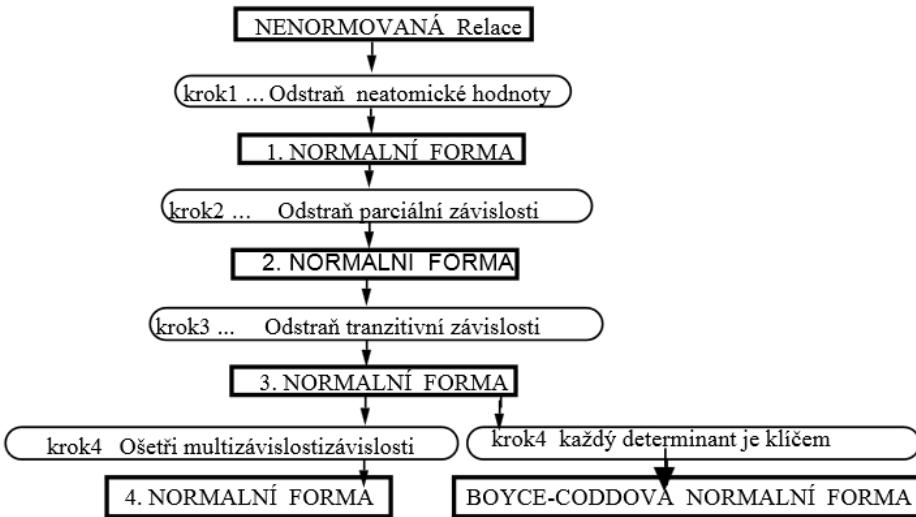
BCNF zaručuje potlačení redundancy

### 14.4.2 normalizace dekompozicí a kompozicí schémat.

Normalizace pomocí dekompozice:

Pokud není  $R$  v BCNF vzhledem ke  $\Gamma$  potom :

- vezmeme netriviální  $A \Rightarrow B \in \Gamma$  takovou, že  $\Gamma \not\models A \Rightarrow R$ .
- položme  $R_1 = A \cup B$  a  $\Gamma_1 = \{C \cap R_1 \Rightarrow D \cap R_1 | C \cap R_1 \Rightarrow D \in \Gamma\}$
- položme  $R_2 = A \cup (R \setminus B)$  a  $\Gamma_2 = \{C \cap R_2 \Rightarrow D \cap R_2 | C \cap R_2 \Rightarrow D \in \Gamma\}$
- proces se pokusíme opakovat pro dvojici  $R_1$  a  $\Gamma_1$  pokud  $R_1$  není v BCNF vzhledem k  $\Gamma_1$  a analogicky pro  $R_2$  a  $\Gamma_2$



Obr. 15 Postup normalizace relací

### Příklad (Schéma, které není v BCNF)

Uvažujme relační schéma

$$R = \{\text{SCHOOL}, \text{DEAN}, \text{DEPT}, \text{HEAD}, \text{ID}, \text{COURSE}, \text{YEAR}\}$$

a teorii  $\Gamma$  popisující závislosti mezi atributy:

$$\begin{aligned} \Gamma = \{ &\{\text{DEPT}\} \Rightarrow \{\text{HEAD}, \text{SCHOOL}, \text{DEAN}\}, \\ &\{\text{SCHOOL}\} \Rightarrow \{\text{DEAN}\}, \\ &\{\text{COURSE}, \text{YEAR}\} \Rightarrow \{\text{ID}\}, \\ &\{\text{ID}, \text{YEAR}\} \Rightarrow \{\text{DEPT}\} \}. \end{aligned}$$

Schéma  $R$  není v BCNF vzhledem k  $\Gamma$ , protože (například):

- $\{\text{DEPT}\} \Rightarrow \{\text{HEAD}, \text{SCHOOL}, \text{DEAN}\} \in \Gamma$ , ale  
 $[\{\text{DEPT}\}]_{\Gamma} = \{\text{SCHOOL}, \text{DEAN}, \text{DEPT}, \text{HEAD}\} \neq R$ , to jest  $\Gamma \not\models \{\text{DEPT}\} \Rightarrow R$ , nebo:
- $\{\text{SCHOOL}\} \Rightarrow \{\text{DEAN}\} \in \Gamma$ , ale  
 $[\{\text{SCHOOL}\}]_{\Gamma} = \{\text{SCHOOL}, \text{DEAN}\} \neq R$ , to jest  $\Gamma \not\models \{\text{SCHOOL}\} \Rightarrow R$ .

## Příklad (Normalizace schématu pomocí dekompozice)

$R = \{\text{SCHOOL}, \text{DEAN}, \text{DEPT}, \text{HEAD}, \text{ID}, \text{COURSE}, \text{YEAR}\}$

$\Gamma = \{\{\text{DEPT}\} \Rightarrow \{\text{HEAD}, \text{SCHOOL}, \text{DEAN}\}, \dots\}$  (viz předchozí příklad)

- $R_1 = \{\text{SCHOOL}, \text{DEAN}, \text{DEPT}, \text{HEAD}\}$

$\Gamma_1 = \{\{\text{DEPT}\} \Rightarrow \{\text{HEAD}, \text{SCHOOL}, \text{DEAN}\}, \{\text{SCHOOL}\} \Rightarrow \{\text{DEAN}\}\}$

- $R_{11} = \{\text{SCHOOL}, \text{DEAN}\}$

$\Gamma_{11} = \{\{\text{SCHOOL}\} \Rightarrow \{\text{DEAN}\}\}$

- $R_{12} = \{\text{SCHOOL}, \text{DEPT}, \text{HEAD}\}$

$\Gamma_{12} = \{\{\text{DEPT}\} \Rightarrow \{\text{HEAD}, \text{SCHOOL}\}, \{\text{SCHOOL}\} \Rightarrow \{\}\}$

- $R_2 = \{\text{DEPT}, \text{ID}, \text{COURSE}, \text{YEAR}\}$

$\Gamma_2 = \{\{\text{DEPT}\} \Rightarrow \{\}, \{\text{COURSE}, \text{YEAR}\} \Rightarrow \{\text{ID}\}, \{\text{ID}, \text{YEAR}\} \Rightarrow \{\text{DEPT}\}\}$

- $R_{21} = \{\text{DEPT}, \text{ID}, \text{YEAR}\}$

$\Gamma_{21} = \{\{\text{DEPT}\} \Rightarrow \{\}, \{\text{ID}, \text{YEAR}\} \Rightarrow \{\text{DEPT}\}\}$

- $R_{22} = \{\text{ID}, \text{COURSE}, \text{YEAR}\}$

$\Gamma_{22} = \{\{\text{COURSE}, \text{YEAR}\} \Rightarrow \{\text{ID}\}, \{\text{ID}, \text{YEAR}\} \Rightarrow \{\}\}$

## Příklad (Reprezentace výchozích dat v normalizované databázi)

SCHOOL	DEAN	DEPT	HEAD	ID	COURSE	YEAR
SCI	Blangis	AF	Durcet	7	QOPT1	2012
SCI	Blangis	AF	Durcet	8	LASR1	2012
SCI	Blangis	AF	Durcet	8	LASR1	2013
SCI	Blangis	CS	Curval	3	ALMA1	2012
SCI	Blangis	CS	Curval	3	ALMA1	2013
SCI	Blangis	CS	Curval	6	DATA1	2012
SCI	Blangis	CS	Curval	6	DATA1	2013
SCI	Blangis	CS	Curval	6	PAPR1	2012

=

DEPT	ID	YEAR
AF	7	2012
AF	8	2012
AF	8	2013
CS	3	2012
CS	3	2013
CS	6	2012
CS	6	2013

ID	COURSE	YEAR
3	ALMA1	2012
3	ALMA1	2013
6	DATA1	2012
6	DATA1	2013
6	PAPR1	2012
7	QOPT1	2012
8	LASR1	2012
8	LASR1	2013

SCHOOL	DEAN
SCI	Blangis

SCHOOL	DEPT	HEAD
SCI	AF	Durcet
SCI	CS	Curval

⊗

Normalizace pomocí kompozice:  
Opak dekompozice, slévání tabulek.