Final State

Thach Pham

May 2022

Contents

1 Subject Information and Communication Technology

1.1 Logical Circuits

1.1.1 Boolean algebra, Boolean functions and combination circuits

Boolean algebra is binary algebra (m and only two constants 0 and 1) and uses three basic operations:

- Negation - NOT
- Logical product - AND
- Logical - OR

With Boolean algebra we can minimize logical functions according to given rules and rules:

- Commutative: $a + b = b + a$
- Associative: $a + (b + c) = (a + b) + c$
- Distributable: $a(b + c) = (ab) + (ac)$
- Indepotent:
  - $a + a = a$
  - $a \times a = a$
  - $a + 1 = 1$
  - $a \times 1 = a$
  - $a + 0 = a$
  - $a \times 0 = 0$

There are several methods for evaluating the truth values of logical functions:

- Equation / Function Prescription
  - $P = a \times b \times c$
  - P becomes log 1 when all variables (a, b, c) are in log 1
- Truth table
- Karnaugh map
  - Graphical table of truth table, where each line corresponds to a field (2N field, where N is the number of variables)

1.1.2 Integer representation and corresponding arithmetic (binary complement, offset binary code, BCD code).

- Integers (Z) consist of positive and negative numbers, including 0.
- So the problem is to express both positive and negative numbers using bits → the bit with the most weight (most left) indicates whether it is a positive number (0) or a negative number (1).
- The disadvantage is that we have two states even for zero, negative and positive zero.
- Complicates arithmetic operations → different algorithms are needed for positive and negative numbers
- This method is called Direct Code

Binary complement:

- The binary complement code is the most common way to represent signed integers in a computer.
- A way of encoding integers so that the ALU uses the same arithmetic algorithms for positive/negative numbers, but reads them differently.
- Eliminates the problem of positive and negative zero.
- Subtraction is implemented as: the first number + the two's complement of the second number
- When using a two's complement, the opposite number is obtained by negating all bits of the number and adding one to the result.
- Example:
  - 0111 or number 7
  - negate the bits: 1000
  - Add 1: 1001 and we have -7

Offset binary code:

- we add to the number some known constant that we set.
- Usually this is a constant in the middle of the interval. For example, for 8-bit numbers, which can take 256 values, we choose the value 128 (10000000).
- To the number we then want to represent we add the constant we set.
- Positive numbers then have 0 at the beginning and negative numbers have 1. (the opposite of the direct code!, it follows from the calculation)

- The disadvantage of this notation is that positive numbers are different from the unsigned representation of numbers.

BCD code

- method of encoding integers using only decimal digits (0-9) in one nibble (4bits)
- there are $2^4(16)$ nibble combinations, and there are only ten decimal digits, six combinations are unused $\rightarrow$ wasteful
- Example: the number 29 is encoded using BCD as "0010 1001", each four bits corresponds to one decimal digit.
- BCD is very common in electronic systems where a numeric value is to be displayed, especially in systems consisting solely of digital logic, and not containing a microprocessor.

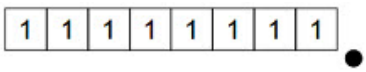
1.1.3 Fixed-point number representation, fixed-point arithmetic

Table 1: Fixed line number

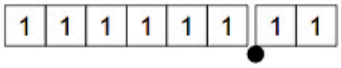
Bit Position	8	7	6	5	4	3	2	1
Bit weight	2^4	2^3	2^2	2^1	2^0	2^1	2^2	2^3
Decimal Bit weight	16	8	4	2	1	0.5	0.25	0.125

Fixed point arithmetic:

- without decimal place (8 bits)
  - primary code: 0 to 255
  - additional code: -128 to 127



- with decimal (8 bits)
  - Primary code: 0 to 63.75



- the numbers are evenly distributed on the numerical axis.

1.1.4 Floating-point representation (IEEE 754-2008, binary and decimal basis), floating-point arithmetic

- To display large or decimal numbers.

- $X = SM \times B^E (-5.97x10^47)$
  - S is a sign
  - M is mantissa (cast after decimal point)
  - B is the foundation
  - E is an exponent
- Can be displayed in any precision
  - Simple (32bit)
  - Double (64bit)
  - Extensions (80bit)
- The IEEE 754-2008 standard has expanded these accuracy with half-time and four-time accuracy

1.1.5 Character encoding, ASCII, Unicode

ASCII

- A table that defines English alphabet characters and other characters
- 7-bit, ie 2^7 (128) characters.
- As needed, expand to 8 bits (256 characters) for nations other than the Amish. The first 128 bits are identical to ASCII and the rest is added extra. For Czech Windows-1250 or ISO 8859-2
- Tl, dr created a mess and chaos, therefore Unicode

Unicode

- Unified table of all alphabets of different languages
- Currently over 100,000 characters, the possibility of up to 1,114,112 characters in the future.
- Versions UTF-8, UTF-16, UTF-32, according to the number of bits used to encode the character (however, this is the minimum size, some characters can be encoded with a higher number of bits.)
- Advantage: contains all characters
- Disadvantage: larger size and complexity of coding

1.1.6 Finite-state automaton (finite state machine), Moore and Mealy automaton

Final numbering machine

- Not DFA → UTI
- FSM → Finite State Machine
- DSM → Deterministic 'Status' Automat

Moore

- The output is displayed only after the transition, ie in the next state.
- The output is therefore determined only by the current state

Mealy

- Automat, whose output is determined by the current status and current input.
- The output is not in the state but at the transition to the next state
- This often requires a status meter

1.2 Telecommunication networks

1.1.7 LAN and WAN networks (Ethernet, ATM, Frame Relay)

LAN:

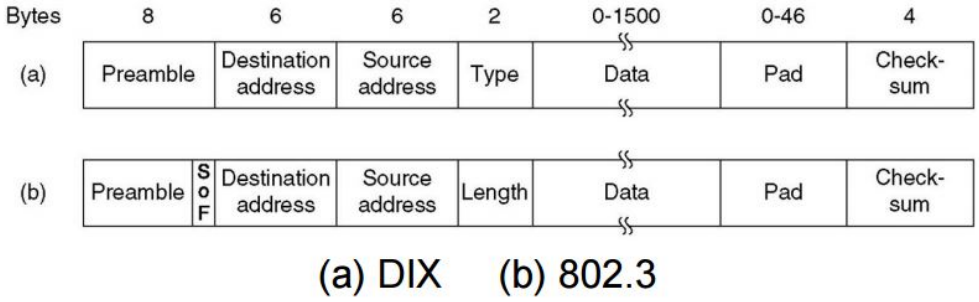
- Networks, as a rule, in only a smaller place (one building).
- Share local resources (printers, data, applications).
- From the point of view of the OSI RM, LAN technology uses only the physical and connective layer.
- The most widespread technology is Ethernet

Ethernet:
Identical to "APPS/POS"

- Summary of technologies for LAN networks.
- Standard IEEE 802.3
- Naming by format: $\{Mbps|GbpsG\}\{Base|Broad\}\{seg_{len}/100[m]|-medium\}$.*Např.* : $10Base-T$.
- Coax, TP, optics.
- For Half-duplex CSMA/CD (media access):

- Listens to see if the media is free.
- If it is free, it starts transmitting and listens for any signal from another station. If, yes, a collision has occurred (multiple stations can detect free media and start broadcasting). Stops the transmission and sends a jam signal.
- The station waits a random amount of time before trying to re-transmit. If several consecutive collisions occur, the time increases exponentially.

- There are multiple versions of the frames. The most common is DIX / Ethernet II.



(a) DIX (b) 802.3

- Basic comparison of some Ethernet variants:

- 10Base5 = Thick Ethernet - 10mm coax, up to 500m long segments
- 10Base2 = Thin Ethernet - 5mm coax, up to 185 long segments
- 10Base-T - Star topology, UTP3 cables, range 100m from the hub
- Fast Ethernet - similar to 10Base-T, 100Base-TX (UTP5), 100Base-FX (single / multi mode optical fiber)
- Gigabit Ethernet - 1000Base-T, 1000Base-SX
- 10 Gigabit Ethernet - pouze full duplex(no CSMA/CD), UTP6; 10GBaseT

WAN:

- Larger networks - usually it is a connection of several LAN networks using telephone lines
- The most well-known WAN is, of course, the Internet
- Frame Relay
  - Mostly permanent point-to-point virtual connections (Permanent Virtual Circuit).
  - The possibility of Switched Virtual Circuits (SVC), which is not used much
  - Ignores checking and acknowledging, leaving it to higher layers → higher speed
  - Supports QOS
  - Packet transmission is provided by the **LAPLAP-F** protocol
  - The so-called **DLCI** address is used for addressing
  - It is not used much today

- ATM
  - Uses the **virtual path method**.
  - Provides QoS for voice and video transmission
  - Data is transmitted using ATM cells as opposed to frames
  - It uses the SDH / SONET transport network
  - The protocol model corresponds to the first 4 layers of OSI

1.2.2 Transport networks (SDH, DWDM, MPLS)

Used to create a point-to-point type

SDH

- Effort to improve / speed up PDH → synchronous transmission via optics
- Defined on the OSI physical layer
- Uses time division multiplexing
- In the US, it's SONET
- It transmits data and voice via virtual containers, into which various things can be put, such as an ATM cell
- The main device of the network is a multiplexer - terminal and add / drop.
- Terminal clusters channels into 1, and add / drop adds and removes channel data to the aggregated stream.
- Uses optical regenerators - the signal is restored by converting to electrical signals and back to optical.

DWDM

- Unlike SDH, wavelength multiplexing is used here
- It uses many different wavelengths in the 1550 nm range
- It is based on WDM by compressing wavelengths (reducing spacings
- Each wavelength = $10Gb/s$ channel, experimentally up to $100gb/s$
- Similar devices as SDH, terminal and add / drop multiplexer + optical cross-connector
- Unlike SDH, it uses fiber amplifiers, which, unlike regenerators, do not require conversion to electrical signals, but work directly with the optical signal.

MPLS

- Integration of IP and virtual channels

- The main device of LSR - Label Switch Router, which plays the role of a classic router and virtual channel switcher.
- The protocol that provides this is LDP, which creates a virtual LSP path from one LER to another. LER (Label switch Edge Router) is an LSR at the boundaries of the MPLS network.

1.2.3 Internet, Secure Transport Services (VPN, IPsec, SSL)

The basis of the Internet are two services - **Web service + Internet transport network**

The Internet is the largest network in the world, it does not work according to uniform rules, but offers the same services to all users. It is actually a network of networks, and each of these sub-networks is managed by a different operator - **ISP - Internet Service Provider**.

The Internet classification corresponds to the classification of ISP providers. There are two types of classification:

- Backbone ISP - international operators owning networks covering a specific country, continent, or the entire world
- Regional ISPs - provide services within a specific region
- Local ISP - a small area, such as a city

and

- Tier 1 - corresponds to the backbone
- Tier 2 - provides services to END customers throughout the country or continent
- Tier 3 - corresponds to the regional
- Tier 4 - corresponds to local

You can also classify multiple types of providers, such as Internet Content Provider (ICP), Content Distribution Provider (CDP), Hosting provider, and Application Service Provider (ASP).

Safe transport services

Enable secure data transmission over the public network - the Internet.

- IPSec
  - Works on the Network layer
  - Provides data communication in IP networks
  - Data Authentication, Integrity and Confidentiality ¡- Uses encryption
- SSL/TLS
  - Works on the OSI RM presentation layer - secures application layer application / protocol data.

  - Very universal protocols - must be usable for a wide range of applications

- VPN
  - It seeks to achieve the characteristics of real private networks
  - The network itself is not physically separated from other networks
  - Multiple types of classifications
    * Customer vs Provider VPN (CPVPN vs PPVPN) - the client or provider takes care of the construction and management
    * Customer edge based VPN vs. Provider edge based - ie the device providing the VPN is on the side of the customer or provider
    * Data Encryption vs. Traffic Department - IPsec VPN, SSL VPN vs. MPLS VPN, ATM VPN = Secure Encrypted Channels vs. Persistent Virtual Channels
  - Two types of technology
    * Data encryption - secure channel with IPsec, SSL
    * Traffic department - two end devices have their own virtual channel, no need to encrypt it.

1.2.4 Signalling in telecommunications networks

Signaling is used for the construction and disruption of connections between exchanges and when using network services.

- CCS7
  - Common Channel Signaling System No. 7
  - The connection is built on the basis of the called party's number
  - High reliability, international standard, automation
  - A separate channel is used containing several useful channels, which are intended exclusively for communication between users
  - The channel is connected between units called SP = Signalization point
  - STP (transfer point) are signaling transit points that only forward signals to another STP or end SP
  - SCPs (control points) provide the control and, for example, the databases needed for routing
  - Signaling route = two opposite channels, all routes between two SP = signaling connection
  - Traffic can be partially combined (use of STP) or combined (signaling and call route are identical)

- H.323

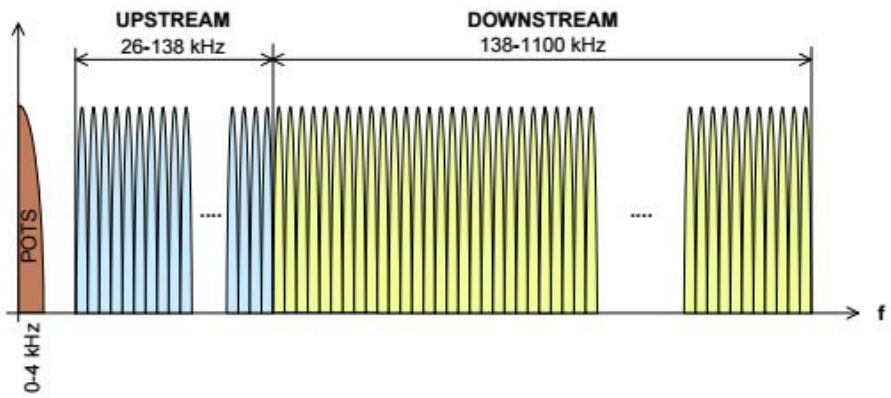
  - Not only voice services, but also video or video conferencing
  - It covers a large number of sub-protocols:
    * Signaling and secure transmission protocols
    * Protocols for real-time multimedia data transmission
    * Voice and video processing protocols
  - Architecture divided into Administrative domains - collection of components managed by the so-called Gatekeeper
  - Very robust and difficult protocol to implement

- SIP
  - Designed for easy implementation and extensibility. It works on the application layer.
  - It is used to set up, modify and terminate voice over Internet-VoIP connections
  - In connection with it, RTP (Real-Time Protocol, ensures the transmission of voice in packets over IP) and SDP (Session Description Protocol, eg the choice of codecs for the call) are also used.
  - Two basic elements:
    * User Agents - endpoints
    * SIP Proxy - connection request routing
  - Addressing using the SIP address in the sip: user @ host format
  - Communication consists of messages transmitted in UDP datagrams of the request and response type
  - Applications - INVITE, ACK, BYE, CANCEL, REGISTER
  - Answers - 1xx - 6xx

1.2.5 Access networks (xDSL, DOCSIS, FTTx)

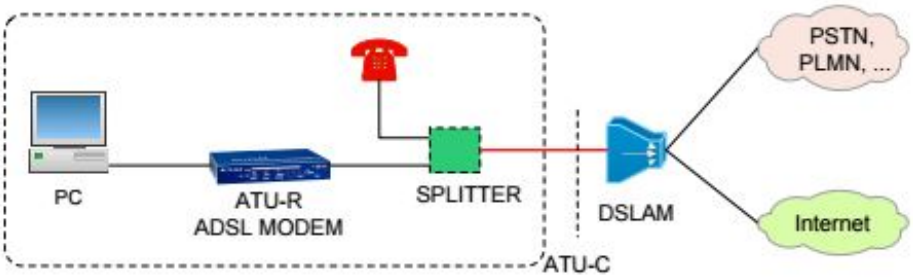
- xDSL
  - ADSL
    * Data transmission method over already existing copper symmetric pairs
    * ATU-R = subscriber-side modem
    * ATU-C = modem on the control panel side - mostly part of DSLAM
    * Splitter = frequency filter to separate telephone connections from data transmission
    * Fullduplex using FDM / Echo Cancellation
    * Band 26-1100 kHz - division into 256 sub channels using QAM
    * The number of bits in the sub-channel is not fixed at 2-12 bits, it depends, among other things, on the line properties

* 9Mbit / s DOWN, 1Mbit / s UP



Obrázek 14.2: Obsazení spektra systémem ADSL

- ADSL2
  - * Introduction of flexible structure (length) frames
  - * 12UP 3.5Down
- ADSL2+
  - * Extension up to 2.208 MHz, 511 subcarriers
  - * Up to 24Mbit / s DOWN
- VDSL
  - * Can not use EC, use FDD
  - * Expansion up to 8.5MHz at the cost of reducing the range somewhere to 1.6km
  - * 52Mbit / s DOWN, 6.4Mbit / s UP
- VDSL2+
  - * Doubling the pitch to 8.6KHz
  - * Bandwidth up to 30MHz
  - * Theoretically up to 100Mbit / s



Obrázek 14.1: Architektura systému ADSL

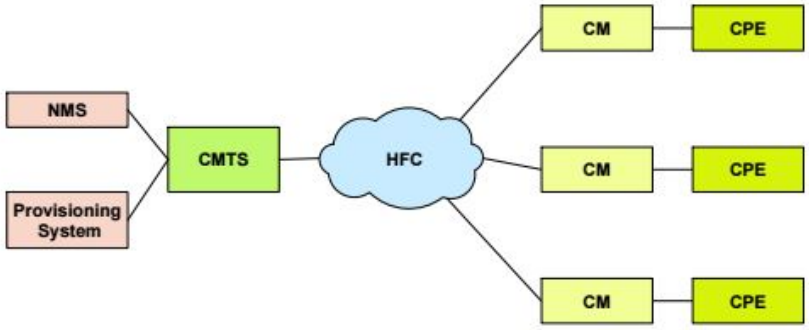
Tabulka 14.1: Počtech subkanálů a šířka pásma pro technologie xDSL

Typ přípojky	ADSL2	ADSL2+	VDSL2	VDSL2	VDSL2
Rozteč subkanálů [Hz]	4312,5	4312,5	4312,5	4312,5	8625
Počet subkanálů	256	512	1972	4096	3479
Šířka pásma [MHz]	1,1	2,2	8,5	17,7	30

- DOCSIS

  Use of cable television networks (COAX or hybrid fiber coax) for two-way data transmission

  - CMTS = core technology at the provider → signal modulation and connection control
  - CM = Cable modem - modem on the customer's side
  - This technology is defined on the first two layers of OSI RM
  - On the provider side, the necessary optimization of television programs (regrouping) and freeing up the necessary band for DOCSIS



Obrázek 15.1: Architektura systému DOCSIS

- PON

  - Passive optical networks
  - OLT - interface between provider and access network
  - ODN - Distribution network - optical network transmitting data, there is also a passive hub that "copies" signals up to 128 new streams
  - ONU - on the customer's side, it terminates the optical network and the metallic line continues
  - FTTx
    - FTTC - Curb - Distribution box from which metal cables lead to individual buildings
    - FTTB - Building - distribution box at the building level - metal cables in the apartment building
    - FTTO - Office

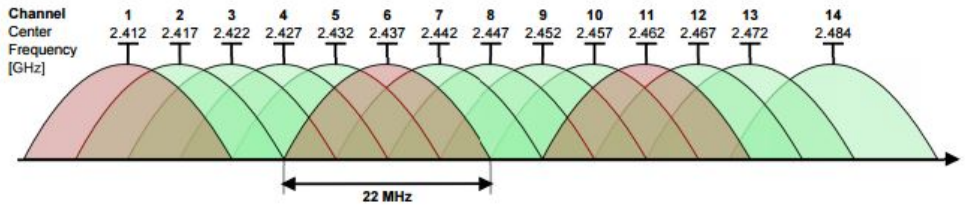
    * FTTH - Home
  - GPON - uses custom frames
  - EPON - uses Ethernet frames

1.2.6 Wireless access networks (WiFi, WIMAX, Bluetooth, Zigbee)

- WiFi
  - 802.11 standard; definition of the two lower layers of OSI
  - Utilization of the given band by the Spread Spectrum method
    * FHSS (frequency hopping) - pseudorandom jumps in frequencies, resistant to interference
    * DSSS (direct sequence) - pseudo-random mask (code) known by the transmitter and receiver spreads each bit
    * OFDM (orthogonal fr. Division multiplexing) - serial stream converted to more slower parallel streams + FDM. The frequencies are more condensed because they are orthogonal to each other and do not interfere.

Tabulka 18.1: Přehled standardů IEEE 802.11

Standard	Rok vydání	Pásmo [GHz]	Maximální rychlost [Mbit/s]	Fyzická vrstva
IEEE 802.11	1997	2,4	2	DSSS a FHSS
IEEE 802.11a	1999	5	54	OFDM
IEEE 802.11b	1999	2,4	11	DSSS
IEEE 802.11g	2003	2,4	54	OFDM
IEEE 802.11n	2009	2,4 nebo 5	600	MIMO OFDM
IEEE 802.11ac	2013	5	1000	MIMO OFDM
IEEE 802.11ad	2014	2,4 , 5 a 60	7000	OFDM



- WIMAX
  - Standard 802.16; definition of the two lower layers of OSI
  - It is a standard for wireless data distribution focused on outdoor networks, ie in addition to Wi-Fi understood as a standard for indoor networks.
  - Uses OFDM
- Bluetooth

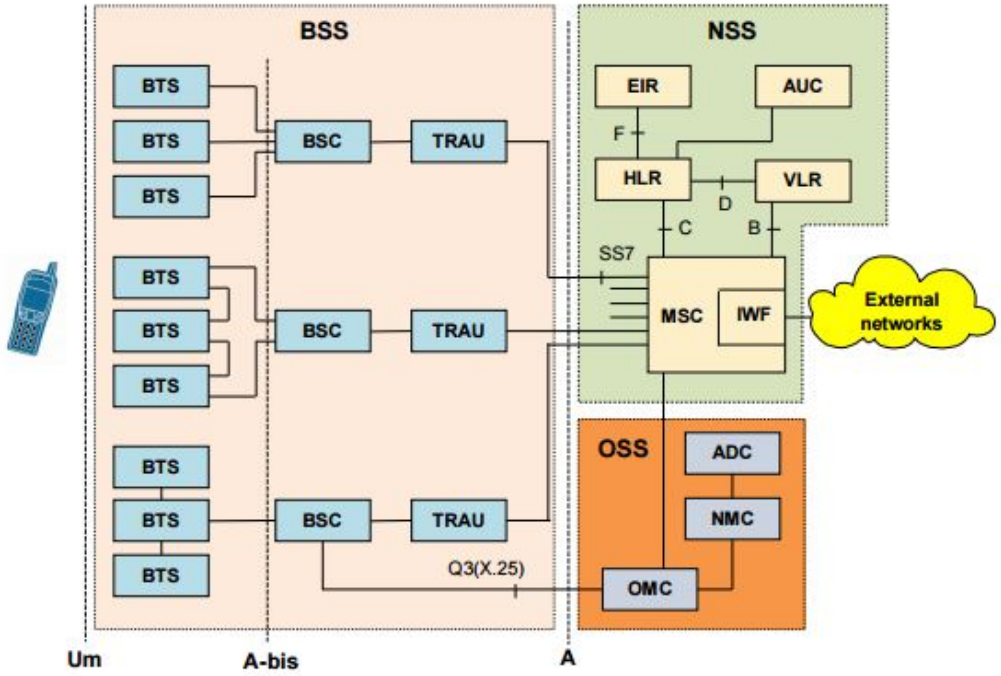
  - Works with some wifi devices on the 2.4Ghz channel
  - It uses FHSS for jumps
  - Sorting according to power into 3 classes (1 - 100mW, 2 - 2.5mW and 1mW with a range of 100, 10 and 1 meter)
  - Srting according to the standard on BT1.2, 2.0, 3.0 and today 4.0
  - Unlike wifi and others, it is defined on all 7 layers of OSI RM
  - Topology:
    * Bluetooth supports both point-to-point and multipoint communication. If several stations are connected to an ad-hoc network (so-called piconet), one radio station acts as a master and can simultaneously serve up to 7 slave devices. All devices in the picosite are synchronized with the beat of the control station and with the frequency hopping method. The specification allows the simultaneous use of up to 10 picosets on an area with a diameter of 10 meters and further grouping these picosets into so-called "scatternets" or "spread" networks.

- Zigbee
  - 802.15.4
  - Similar to Bluetooth, however, the emphasis is not on data speed and volume, but on reliability and ease of implementation
  - Application - PC peripherals, building automation, remote control, industrial automation

1.2.7 Mobile radio networks (1^{st} to 4^{th} generation)

- Generation 1
  - Purely analog devices using FDMA
  - Representatives are NMT (Nordic Mobile Telephony) and AMPS (Advanced Mobile Phone System)
  - A lot of standards, a lot of applicants and small capacity options $\rightarrow$ expansion to 2G
- Generation 2
  - GSM in Europe and CDMA One in America
  - Unclosed system - access to other networks
  - 3 main components
    * BSS - Base Station Subsystem - provides and manages transmission paths between mobile stations and the NSS
    * NSS - Network and Switching Subsystem - the main part of the network, mimics the functions of a traditional telephone exchange

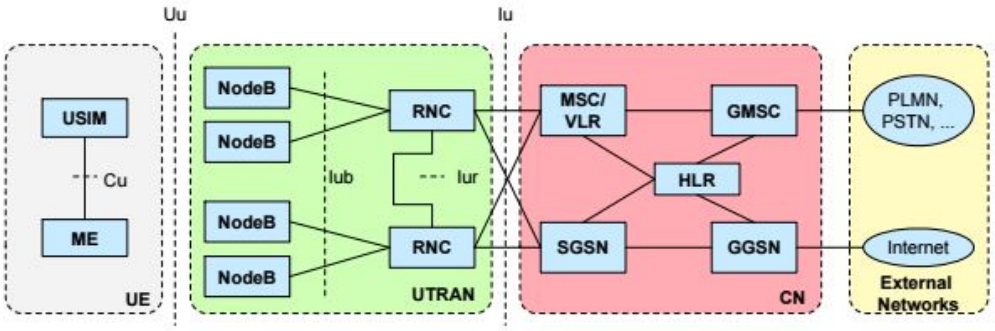
    - OSS - Operational Support Subsystem - administrative functions used by the operator's employees
  - Change the approach to a combination of TDMA and FDMA
  - Transition from analog to digital devices
  - New services such as messaging and email
  - Reduction of cells → reduction of hanging power, better durability and increased safety
  - There are two intermediate stages referred to as 2.5G and 2.75G, respectively. GPRS and EDGE = internet connection and later increase internet speed using 8-PSK



Obrázek 26.1: Architektura systému GSM

- Generation 3
  - = UMTS
  - The main difference is that instead of the GSM RAN / GERAN radio access network, the so-called UTRAN is now used.
  - Oriented to broadband data transmission
  - Fully digital, a significant increase in data rates
  - CDMA approach method, resp. WCDMA, which is spread CDMA
  - Architecture of 3 main elements
    - UE - user equipment = SIM card
    - UTRAN - radio accessible part

∗ CN - Core network = core (mobile exchange + databases and registers)



Obrázek 27.1: Detailní architektura systému UMTS

- Generation 4
  - LTE = long term evolution
  - Formally, LTE still belongs to 3 and up to 4G belongs to LTE Advanced
  - The whole network is data oriented, the voice is also represented as data and not as sound.
  - OFDM instead of WCDMA

1.3 Introduction to Theoretical Computer Science

1.3.1 Sets, relations, functions.

Sets

- A collection of objects (elements) understood as a whole/ A collection of mutually distinguishable objects - elements.
- Definition by enumeration of x characteristic property - $M = \{1, 2, 3\}, M = \{x \in R; x < 4\}$
- Order does not matter, duplicates are not possible $\{1, 2, 2\} = \{2, 1\}, \{1, 2\} = \{2, 1\}$
- $Size = |X|; X = \{1, 2, 2\}; |X| = 2$
  - Infinite x Finite
  - $Emptyset = \varnothing, |\varnothing| = 0! = \varnothing, |\varnothing| = 1$
- We denote the element x belonging to the set M by $x \in M$, we read the element x belongs to the set M
- Subset P of set M = all elements of set P are contained in set M (denoted by $\subseteq$)
  - $\forall A : \varnothing \subseteq A$
  - $M \subseteq M$
  - $P = \{1\}, M = \{1, 2\} \rightarrow$ P $\subseteq M$

- Operations on sets:
  - Intersection $\Rightarrow M \cap N =$ will contain elements that occur in both M and N
  - Unification $\Rightarrow M \cup N =$ will contain elements that occur in M or in N (i.e. even those that are in both)
  - Difference $\Rightarrow M - N =$ will contain elements that are in M but not in N
  - Complement $\Rightarrow M' or \bar{M} =$all elements from the universe that M does not contain
  - Cartesian product $\Rightarrow M \times N =$ will contain ordered pairs - the first term from M the second from N
  - Potential set 2M -

Relationships

- An arbitrary relationship between a group of elements of one or more sets
- Classification by arity - unary, binary, ternary,... n-ary - set of ordered tuples
- Binary session = set of ordered pairs, e.g. $R = \{(a, a), (b, b), (c, c)\}$
- For a binary session it is possible to define properties such as reflexivity, symmetry, antisymmetry, transitivity
- A mapping from a set X to a set Y is a relation $R \subseteq X \times Y$, where for every element $x \in X$ there is exactly one element $y \in Y$ such that $(x, y) \in R$.
- Binary replace
  - Reflexivity = each element of the set is in relation with itself. Examples of a reflexive relation are identity, the relation "to be less than or equal to", the relation of divisibility, etc.
  - Ireflexivity = no element of the set is in relation to itself. An example of an ireflexive relation is the relation "to be less than"
  - Symmetry = if element a is in a relation with b, then b is in a relation with a; $(a, b) \rightarrow (b, a)$. An example of a symmetric relation is an identity, inequality, or property of the type "having the same sign"
  - Antisymmetry = if both (a,b) and (b,a) then it must hold that a = b. An example of an antisymmetric relation is the "be less than or equal to" relation.
  - Asymmetry = strong antisymmetry = if element a is related to b, then b is not related to a; $(a, b) \rightarrow !(b, a)$. Examples of asymmetric relations are "not equal to", "being less than" or "being greater than" relations.
  - Transitivity = if there is (a, b) and (b, c) then i (a,c) exists. An example of a transitive relation is again a relation "be less or equal", an identity, or relation "to be related" to a set of people.

- Completeness - For each pair, there exists either (a, b) or (b, a) (or both). An example of a full relation is, for example, a relation $\leq$ to an arbitrary number set

Functions

- A function is a name for a mapping from some set M to a set of numbers (usually real or complex), or to a vector space
- Alt.: A function on the set $M \subset R$ is a prescription that assigns exactly one R to each number in the set D.
- The set M is called the Definition Domain of the function and R is called the Value Domain of the function.
- For functions we can decide on many properties (more a matter of mathematics but whatever)
  - Monotonicity of the function - determines whether the function is decreasing or increasing in a given interval
  - Even function - symmetric along the Y axis $\rightarrow f(x) = -f(x)$
  - Odd function - symmetric along X and Y axis $\rightarrow -f(x) = f(-x)$
  - Simple function - the values for the different x's in the definitional domain are always different; the function is always increasing or decreasing throughout the definitional domain
  - limitations
  - global/global minima/maxima

1.3.2 Propositional logic, first-order predicate logic

Propositional logic

- Statement - "Jana got wet" = atomic statement $\rightarrow$ cannot be decomposed
- Compound statement - "If it rained, Jana got wet" - decomposable into the atomic statements "It rained" and "Jana got wet"
- Logical connectives - connect atomic statements into compound = $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- Formulas = atomic or compound statements; Truthfulness of formulas - evaluated binary (0 x 1) (T x F)
- Truthfulness rating
  - Tautology - true in all evaluations
  - Satisfiable - there is at least one true evaluation
  - Contradiction - there is not even one true evaluation (= not satisfiable)

  - Equivalence = two formulas are equivalent if they have the same truth value at the same evaluation "v"

- Logical inference - if the premises are true, then the conclusion is true - verification by the resolution method

Normal forms - a specific form of a formula that is equivalent to the original

- DNF - Disjunctive Normal Form - elementary conjunctions connected by disjunctions, e.g. $(p \wedge q) \vee (\neg p \wedge r) \vee (p)$
- CNF/KNF - Conjunctive normal form - elementary disjunctions connected by conjunctions, e.g. $(p \vee \neg q) \wedge (\neg p \vee r) \wedge (p)$
- Full K/D normal form - in each part (conjunct/disjunct) all literals occur exactly once, e.g. $(p \vee q \vee r) \wedge (\neg p \vee \neg q \vee r) = UKNF$

first-order predicate logic

- Introduction of variables
- Universe - The set of elements from which we can select into variables
- Valuation - Assignment of universe elements to variables
- Predicates - Returns true/false
  - Arity - null, unary - n-ary, describe properties or relations of 0 - n variables
- Quantifiers - $\forall x$ = for all x holds, $\exists x$ = there is at least one x, cannot be applied to constants
- Predicate logic formulas
  - Atomic formulas - can contain constants (zero functions), functions, predicates and equations
  - Otherwise may contain quantifiers, logical connectives
- Terms - expressions composed of variables, constants and functions representing elements of the universe
- auxiliary symbols = "(" + ")"

1.3.3 Regular languages, finite automata

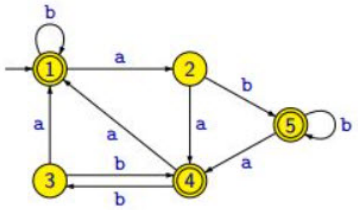
Regular languages

- The simplest formal language. A formal language is a set of words over some alphabet.
- Alphabet = non-empty finite set of symbols/characters

- Word = finite sequence of characters from a particular alphabet
- Language = the set of (some) words formed by symbols from a given alphabet
- Word length = number of symbols, empty word = word of length 0
- Word concatenation = OST - RAVA = OSTRAVA
- Operations on languages = union, intersection, complement, difference, concatenation, iteration
- Language iteration = concatenation of any number (0-n) of words from a language
- Regular language = empty language + language containing just one character from the alphabet + languages resulting from the operations described above on these languages
- Mirror image of word w = w^R is word w written "backwards" AHOJ → JOHA

finite automata

- Computational model generating formal languages
- Describes a very simple computer that can be in one of several states, which it transitions between based on the symbols it reads from input
- Finite automaton = deterministic = there is always a clear initial state, it is always clear what state it transitions to for a given input, there are always clear receiving states.
- Formally, a finite automaton is defined as an ordered five $(Q, \sum, \delta, q0, F)$ (names may vary)
  - Q = non-empty set of states
  - $\sum$ = non-empty set of input symbols (alphabet)
  - δ = state transition rules (for each state, n rules (n = number of symbols in the alphabet))
  - q0 = initial state (just one)
  - F = set of receiving states



1.3.4 Algorithms and algorithmic problems, models of computation

Algorithms

Processes input and generates output

- Control flow - sequence of execution of individual steps, including branching and loops
- Two types of instructions - instructions working with data, instructions affecting the control flow(branching, cycles, returns,...)
- Algorithm is executed by a machine - e.g. real machine, virtual machine, mathematical model
- The machine works in steps, in each step the configuration changes somehow = description of the overall machine(memory, state)
- Algorithm computation - starts in initial configuration, each step goes to a different configuration, computation ends in final configuration
- Invariant - a condition that must always be met, at a specific point in the algorithm
  - e.g. the invariant for cycle i=0;while(i < 10){i++} is that 0 >= i <= 10. Since the last step in the loop will be that i is set to 10 and whenever it is off, it won't fall into the loop at all.
  - For each control state (vertex of the control flow graph), an invariant can be determined
- Correctness - No unauthorized operation, the calculation is final, the output corresponds to the input.

Models of computation

- idealised computer models used to calculate complexity
- Define the set of allowed operations
- Turing machine
  - Church-Turing thesis = to every algorithm there is a Turing machine
  - Respectively, if a problem cannot be described by a Turing machine, the problem is not algorithmically decidable
  - Formally, a Turing machine is a six
    - A set of internal states
    - The final alphabet of symbols on the tape
    - Finite set of input symbols (subset of the alphabet) initial state
    - Transition function right or left
    - Set of end states

- RAM machine
  - In terms of computing power equivalent to a Turing machine
- Lambda calculus
- Final machine

1.3.5 Algorithmically undecidable problems

- There is no algorithm for the P problem. This issue is explored by the field of science - Theory of Excellence. Any algorithm that is decidable can be translated into the Turing machine language. Examples of such problems are eg. Stopping the problem

1.3.6 Computational complexity of algorithms, asymptotic notation.

- For algorithms, we usually deal with two types of complexity
  - Time complexity - how the calculation time depends on the amount of input data
  - Memory complexity - how the amount of memory used depends on the amount of input data
- Input size - value indicating the input size - number of elements in the sequence, number of characters in the string, etc. The exact calculation time can be given as number of steps or number of seconds
- As a rule, we deal with the worst case, but sometimes it is useful to deal with the average case
- For a function expressing time complexity, we usually just need an estimate - we neglect the minor terms $an^2 + b \Rightarrow n^2$

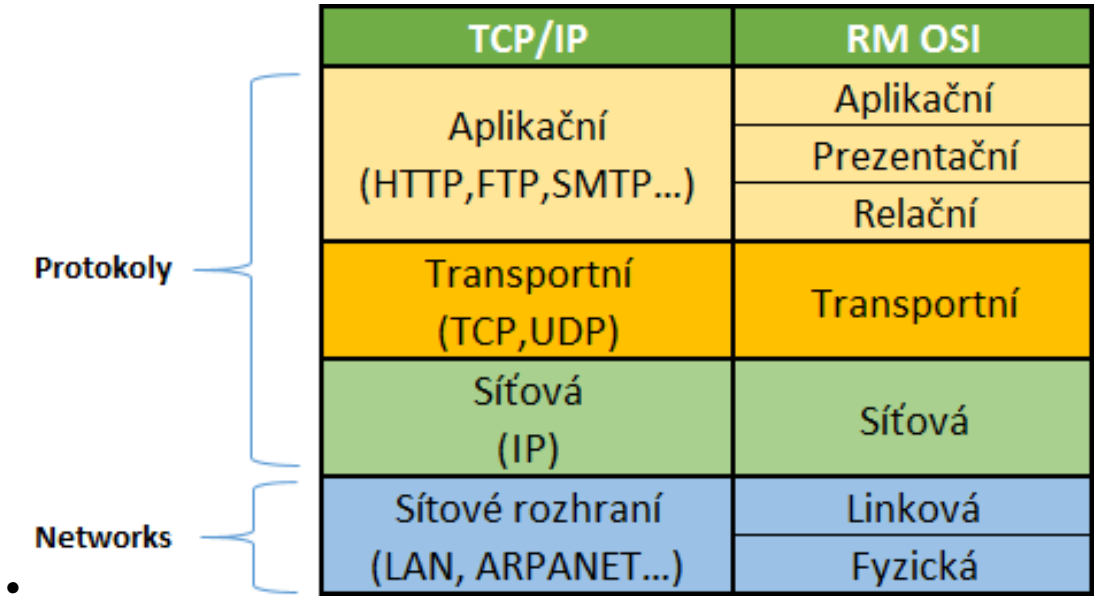
Asymptotic notation

- They describe how fast the function grows
- O(f) - the set of all functions that grow at most as fast as f
- $\Omega(f)$ - the set of all functions that grow at least as fast as f
- $\Theta(f)$ - the set of all functions that grow as fast as f

1.4 Computer architectures, Computer networks

1.4.1 The TCP / IP protocol family and its relationship to the ISO - OSI reference model. Network address translation - NAT, IPv6 - specifics of the new version of the protocol.

TCP / IP and its relation to the ISO - OSI reference model



- An application protocol, FTP, HTTP, SMTP, DNS ... runs on the application layer.
  - Then either TCP or UDP on the transport layer.
  - TCP takes data from the application layer, divides it into numbers, numbers it and sorts it. It checks and requires confirmation of data receipt in case of resending, etc.
  - UDP works similarly, but coughs up control. Fire-and-forget. The loss of one packet is calm aka without one soldier the war will be. Video transmission e.g.
- The network layer then adds a header to the segment and creates an IP datagram. - therefore performs addressing and routing.

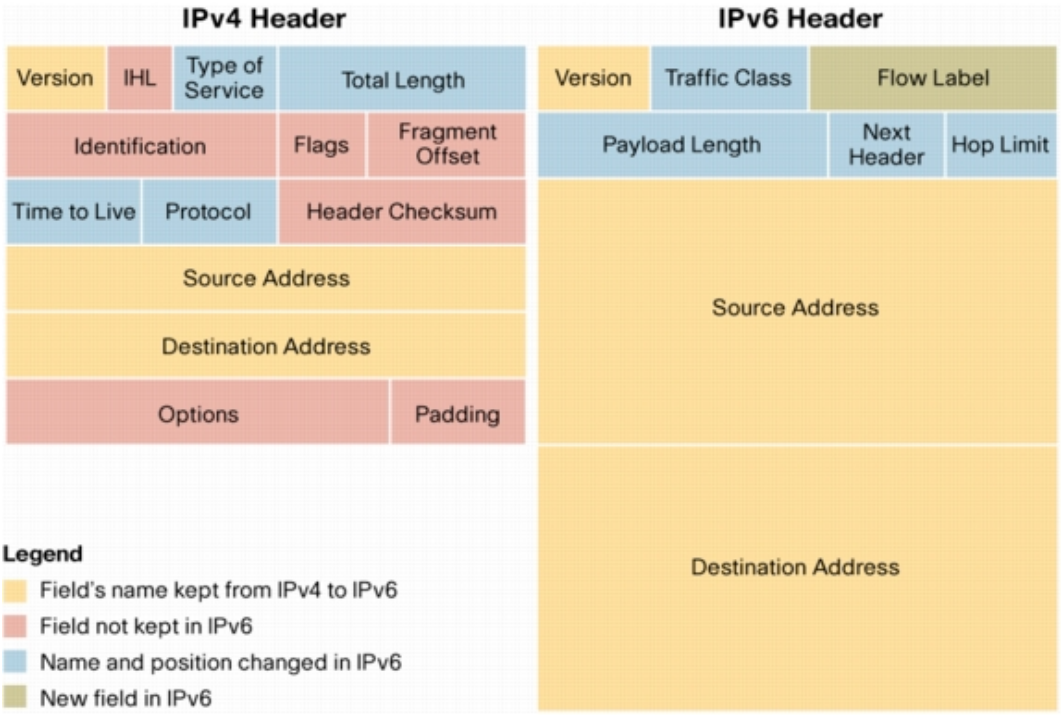
Network Address Translation - NAT

- This is the translation of addresses from the internal network to the addresses of the external network (eg the Internet)
- We don't have to have a 1: 1 address on any subnet.
- The downside is that it limits communication with the Internet. Or if we have asymmetric routing (the internal network is connected to the external network via more than 1 router).
- It stores these data / overlays in the NAT table.
- Static vs Dynamic NAT → manually reconfigured or use dynamic pool.

IPv6 - specifics of the new version of the protocol

- The main change is the much larger address space. (IPv4 2^{32} vs. IPv6 2^{128}).
- For each of the 7 billion people on earth, there are theoretically 5×10^{28} addresses.

- The need to use NAT, which was introduced due to the exhaustion of the IPv4 address space, is eliminated.
- These are 128-bit hexadecimal addresses.
- FEDC:0A98:7654:1230:0000:0000:7546:3210 - where leading zeros may be omitted
  → FEDC:A98:7654:1230:::7546:3210
- SLAAC (Stateless Address Autoconfig) - the router has a prefix and the MAC address of the device is added to it.
- Some protocols are already implemented as a component (IPSec), with IPv4 being optional but almost always implemented.
- Does not know broadcast, only multicast to all addresses (...)



1.4.2 Active elements of computer networks and their functions - hub, switch, router.

Hub (HUB)

- The simplest active element
- It acts like a repeater - whatever comes to a port is repeated for all the protests in it.
- Originally, repeaters were used for old networks with a very long coax, which at its length was already losing signal, so there was a repeater.

Switch (SWITCH)

- He replaced the hub - he almost knows what to send and doesn't copy it - he sends it to the given interface / port.
- When a frame arrives at a previously unknown address (MAC), it sends it to everyone and one of the stations responds, thus adding it to the table to which port / interface it should route next.

Router (ROUTER)

- The most advanced active element
- L3 device → has no idea about IP addresses.
- It can perform routing according to algorithms and thus choose the best path for the packet.
- Implements address translation (NAT)
- Packet filtering
- Nowadays, L3 switches that have the same functionality are often used.

1.4.3 Internet services and their protocols: electronic mail (SMTP, POP, IMAP), WWW, SSH and Telnet. DNS system.

SMTP

- Simple Mail Transfer Protocol
- Used only for sending emails
- It uses TCP over port 25

POP3

- Used only for downloading mail
- TCP/110
- It downloads the email and disconnects, then browsing is "offline". Modern POP3 only downloads email headers and content after clicking.

IMAP

- Protocol for remote access to the e-mail box.
- Unlike POP3, it still requires a connection.
- Downloads only headers. Content after clicking.

- Allows multiple clients to be connected to a single mailbox + maintains email status information (if read, etc.) + synchronizes it between clients. (Good at service board need?)

WWW

- World Wide Web - a system for viewing Internet content
- The HTTP / HTTPS communication protocol is used for communication
- HTTP is used to transfer hypertext documents (HTML) - TCP / 80
- In addition, HTTPS provides secure connections using SSL - TCP / 443

Telnet

- The application layer protocol is used for client-server connections under the TCP protocol.
- Originally terminal emulation for remote access.
- The main disadvantage is that the transmission is not encrypted → this creates SSH
- It used to be OK, it didn't need much security, it was used on university networks, etc., but today in Internet times it's a mess.

SSH

- Telnet replacement
- Encrypts data
- TCP/22
- SFTP (SSH FTP)

DNS

- Translation of IP addresses into more memorable names
- Hierarchical distribution of names.
- Each node max 63 chars, full address max 256. Case insensitive, national characters possible, but not recommended.
- Top-Level Domains: generic (.edu, .com, .org, .net) nebo national (.cz, .uk)
- Primary and Secondary Name Servers (NS) - data is permanently stored in the primary, the secondary will copy it. When the update is at the primary, the admin must upgrade to sync - this is a common mistake. Both are authoritative.
- The resolver is the part of the OS on the client side that communicates with the NS.
- TCP / UDP

1.4.4 Security of computer networks with TCP / IP: attacks, packet filters, stateful firewall. Encryption and authentication, virtual private networks.

- Confidentiality - the listener does not understand the data on the channel
- Authentication - the certainty that the sender is who he claims to be
- Integrity - the certainty that the data has not been modified along the way
- Non-repudiation - the data source cannot deny their sending

Computer network attacks:

- Denial of service (DoS): An attacker's goal is to deplete the system resources or server (memory, CPU, bandwidth) and crash or change the desired behavior. A SYN or flood is a type of attack called a DoS. The attacker sends a sequence of packets with the SYN flag to the target computer, but no longer responds. Brute Force

Packet filters

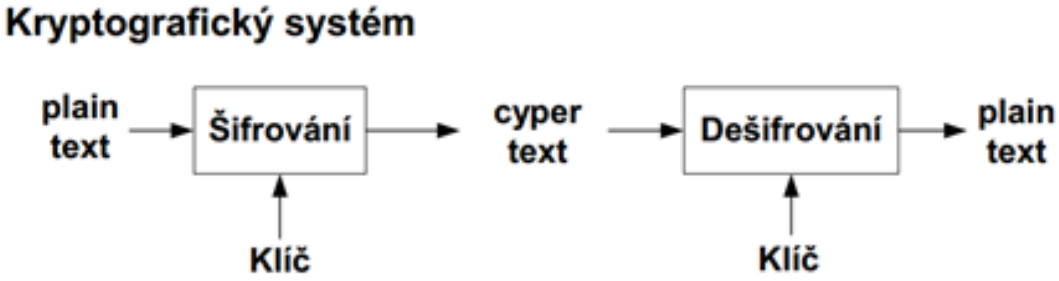
- ACL
  - Most often at the router interface, filtering according to information from the network and higher layers
  - Reflexive ACL - Automatically passes input traffic that corresponds to the allowed output traffic
- Stateful firewall
  - is an IT designation for a firewall that supports Stateful packet inspection (SPI), which means that it is able to monitor and maintain all established TCP / UDP sessions (it works on the transport layer of the ISO / OSI reference model). The state firewall is able to distinguish different packet states within individual sessions (connections) and its task is to release only those that belong to an already allowed session (others are rejected). The stateful firewall provides greater efficiency in checking individual packets because it only checks the state table for existing connections, instead of checking a set of defined complex rules for each packet. The stateful firewall has nothing to do with packet depth checking. It is able to keep important parameters of all connections in memory from start to finish. The most demanding check is performed at the time of connection setup.

Encryption and authentication

- The secret algorithm, when betrayed, is a useless implementation
- Introduce keys parameterizing the algorithm, if there are enough possible keys, the algorithm may be known

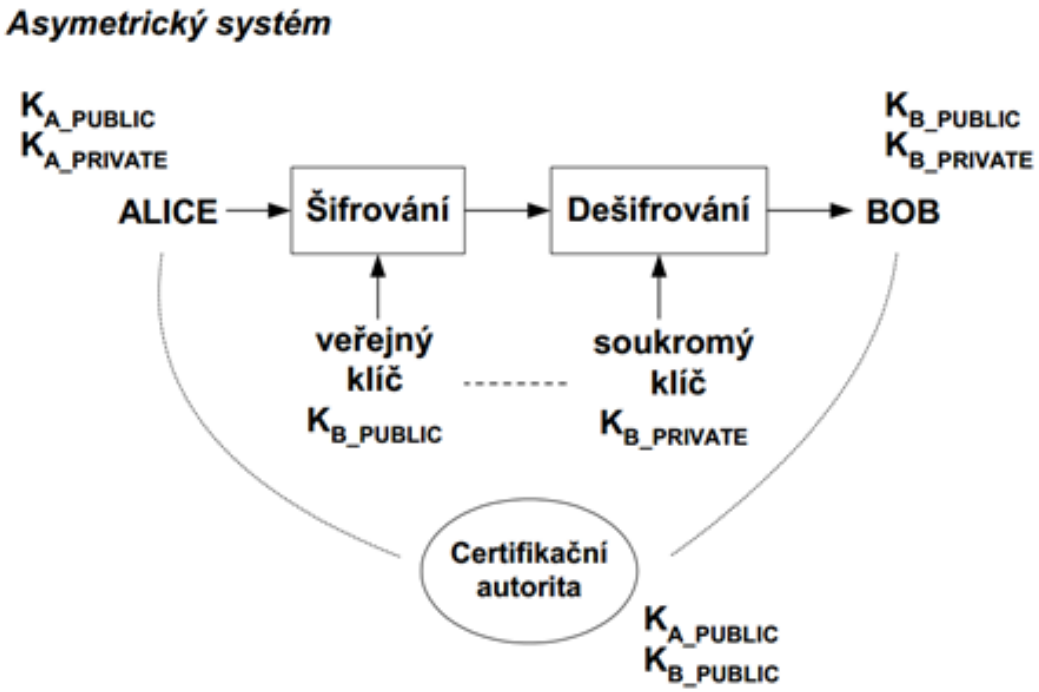
Symmetric system

- Shared key
- Implementation of efficient algorithms (speed) can be implemented in hardware
- Algorithms DES, 3DES, AES,...
- Authentication in a symmetric system - Encryption with the username key at the sender, the same decryption key at the recipient + test of meaningfulness of the name (eg connection of Hash value to the name and check calculation with comparison on the receiver)



Asymmetric system

- The keys are generated as a complementary pair - public and private key
- One key used for encryption, the other for decryption (it doesn't matter which one for what)
- Much more difficult to calculate, slower
- An asymmetric system is commonly used to pass (dynamically generated) keys to a symmetric system.



•

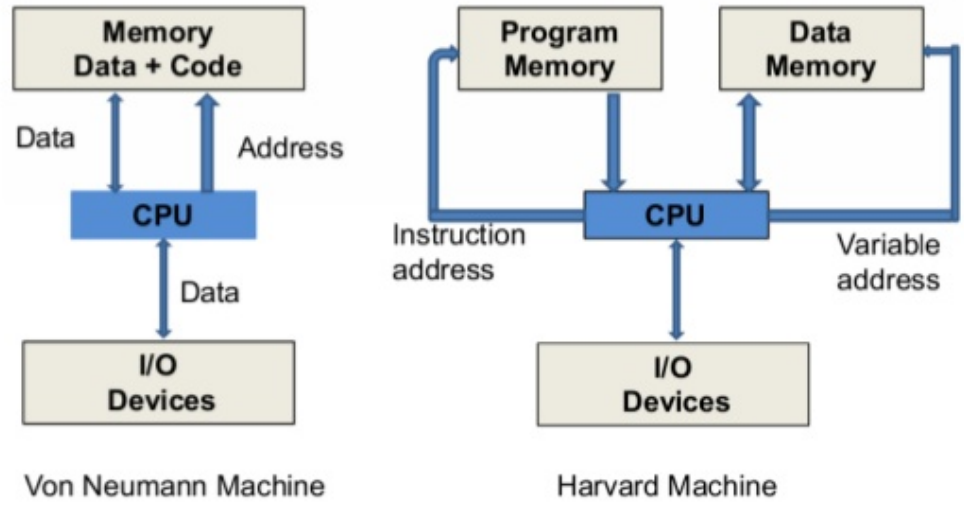
Virtual Private Networks (VPN)

- VPN transmits private data over a public network using encryption methods and tunnels Provides authentication, data integrity, and confidentiality. The advantage is price, flexibility of topology, no management of WAN lines Tunnel - virtual two-point connection over the public network, carries data of one protocol in another protocol.

1.4.5 Computer architectures, their properties, principles of computer operation. Hierarchical arrangement of memories in a computer, basic characteristics of individual memories.

Computer architectures, their properties, principles of computer operation.

- Basic computer principles:
  - It consists of: memory, control unit, arithmetic-logic unit (ALU) and input / output unit
  - The structure of the PC is independent of the type of the solved task
  - The next step depends on the previous one.
  - Instructions and data are in 1 memory. (Late the Harvard concept differently)
  - The program consists of a sequence of instructions, which are called as they are in memory
  - A 2-line system is used to represent instructions, addresses, characters ....
- Two basic concepts: Von Neumann vs. Harvard



- The Harvard concept thus brought above all the division of memory for the program and for the data. → the program can not overwrite itself, we can use different technologies for these two memories, two buses = simple parallelism

Hierarchical arrangement of memories in a computer, basic characteristics of individual memories.

Memories can be divided according to:

- According to the type of access
  - RAM - random access memory
  - SAM - serial access memory
- Podle R/W:
  - RWM - read write memory
  - ROM - read only memory
  - WOM - write only memory
- By cell type:
  - DRAM - dynamic RAM
    - All cells = capacitors. It discharges spontaneously and needs to be constantly charged (otherwise it will discharge after 10ms)
    - The information is therefore stored in the form of a charge (log 1 / log 0)
    - The cells are arranged in a matrix
    - The row is read first, then the column.
    - Variants
      - FP (Fast Page) DRAM, which no longer asks for a line the next time it is read.

        - With (synchronous) DRAM, which uses burst mode - enter ROW and COL and then read the adjacent
    - SRAM - static RAM
        - Information stored by the state of the flip-flop
        - Less memory but faster → cache memory, while DRAM for classic RAMs
        - Memory cells arranged in an array
    - PROM, EPROM, EEPROM, FLASH - programmable memories.
        - DRAM and SRAM do not hold their contents after power off → not suitable for PC boot process → protom ROM memory
        - The main purpose of these memories is to remember data even without power
        - Are there any fuses when they are in their original state, so they conduct current (log 0), when they (programmer) burn them, they do not conduct current (log 1)
        - Therefore, after programming in them (ROM, PROM), no further writing is possible.
        - EPROMs are then memories that can be reprogrammed, the information is stored using an electric charge, which is well insulated. It can be erased with UV radiation.
        - The EEPROMs are then cleared with an electrical pulse. (Electrically Erasable Programmable ROM)
        - Nowadays, FLASH memories, which are improved EEPROMs, the principle is the same.

- L1 vs L2 cache
    - L1
        - Integrated directly on the processor
        - Data from the bus (which is slow) is cached (which is fast) and is then accessed by the processor when needed.
    - L2
        - Between microprocessor and memory.
        - The data traveling between the two parts gets stuck in the cache, and if the microprocessor needs them again, he will touch them there.
        - The cache is controlled by a special controller that tries to predict which data the microprocessor will use.

1.4.6 Basic design properties of RISC processors, principles of accelerating the operation of processors, prediction of jumps. Basic characteristics and principles of operation of Intel family processors from Pentia Pro.

Basic design features of RISC processors, principles of accelerating the operation of processors

- Reduced Instruction Set Computer
- Programmers / compilers usually used only a few instructions - an effort to minimize that instruction set → RISC
- therefore, the most typical feature is a small instruction set.
- One instruction should be completed in each machine cycle. (! = that the instruction lasts one cycle)
- The instructions have a fixed length and a uniform format
- Only the LOAD and STORE instructions are used to work with the main memory
- Unlike CISC, it uses concatenation of instructions. They are queued. See picture:

Krok		Význam
1.	**VI**	Výběr Instrukce
2.	**DE**	Dekódování
3.	**VA**	Výpočet Adresy
4.	**VO**	Výběr Operandu
5.	**PI**	Provedení Instrukce
6.	**UV**	Uložení Výsledku

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}
VI	I_1						I_2						...
DE		I_1						I_2					
VA			I_1						I_2				
VO				I_1						I_2			
PI					I_1						I_2		
UV						I_1						I_2	

Tabulka 4: Postup provádění instrukcí procesorem CISC

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
VI	I_1	I_2	I_3	I_4	I_5	I_6	I_7	...				
DE		I_1	I_2	I_3	I_4	I_5	I_6	I_7				
VA			I_1	I_2	I_3	I_4	I_5	I_6	I_7			
VO				I_1	I_2	I_3	I_4	I_5	I_6	I_7		
PI					I_1	I_2	I_3	I_4	I_5	I_6	I_7	
UV						I_1	I_2	I_3	I_4	I_5	I_6	I_7

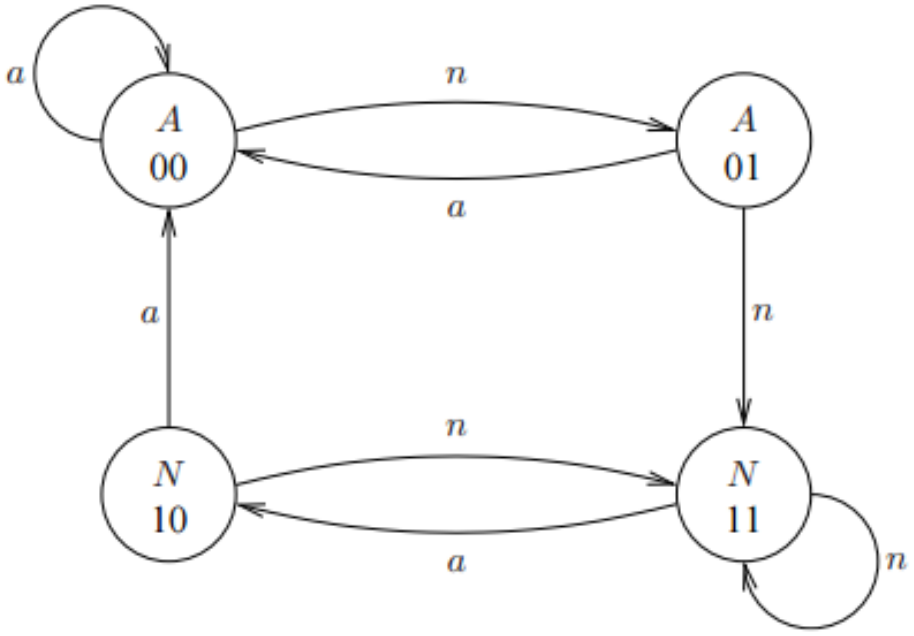
Tabulka 5: Zřetězené provádění instrukcí procesorem RISC

- Instruction concatenation problem:
  - Data - eg when the I5 instruction needs a value at time T7, which I4 does not store until time T9.
  - Structural - instructions need a bus for some of their activities, but only one of them is available. The approach therefore needs to be coordinated, which slows down.

Jump prediction

- There is a problem when we have a jump in the instruction , ie we do not know whether the jump will be performed or not.

- There is no point in pausing processing and waiting for the result - it is better to continue processing and if the jump is not performed, the processed queue of instructions will be used.
- Otherwise, the pending instructions are ignored and the queue starts filling again.
- The jump prediction method is used for this
- One bit is reserved in the instruction format, predicting whether the jump will be performed or not.
- This prediction can be static (appropriate bits are inserted at compile time or directly by the programmer) or dynamic (prediction adapts to current conditions)
- Furthermore, there is a simple one-bit and more accurate two-bit jump prediction, which is shown by the following automaton, where state A says that a jump will be performed and state N will not.



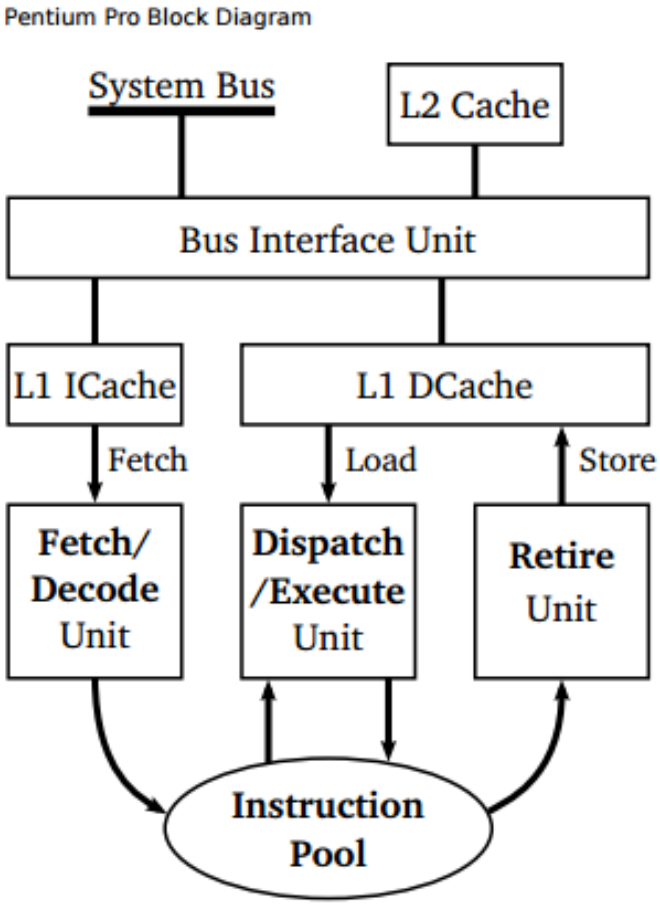
Obrázek 3: Dvoubitová predikce jako stavový automat

-

Basic characteristics and principles of operation of Intel family processors from Pentia Pro.

- Pentium PRO brings major design changes
- The L2 cache is implemented on the processor as a separate chip with the same frequency as the processor

- Transition to RISC architecture, but it was necessary to maintain compatibility with previous processors → this is done by the Fetch / Decode unit, which decodes the instructions into 118-bit RISC instructions
- Instructions are stored in the Instruction Pool (up to 40 microoperations)
- From this pool, the Dispatch / Execute can select out-of-order instructions.
- Executed instructions are stored back in the bank from where they are sent to the registers and L1 cache using the Retire unit.
- Why is it important out-of-order? Because the decoding unit consists of three smaller parts - 2 for simple instructions and 1 for more complex ones, it can produce up to 4 micro-operations in one cycle.
- Other versions of the processors were only modernizations (more transistors, better jump prediction, more cache, etc.)



1.5 Programming

1.5.1 Principles of object oriented programming (OOP) – class, object, encapsulation, inheritance, and polymorphism.

- Class - blueprint/pattern, which is used to create instances. We define properties and behaviour.
- Object - one instance of a class.
- Encapsulation - any information that is not needed by the module clients should remain hidden. (If I want to drive a car, I just need to know that I need to turn the wheel and step on the gas, I don't need to know exactly how it works).
- Inheritance - creating a new class based on an existing class. This is based on extending the original class with new properties.
- Polymorphism - the ability of classes to act in a different role. E.g. if I have classes unified by an interface, I can then put that interface as a parameter to a function and use the function with all classes that implement that interface.

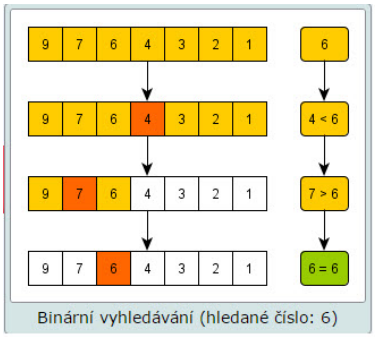
1.5.2 Array based search algorithm – linear (sequential) search algorithm, binary search, informal explanation of their complexities

Sequential

- The most primitive search in the field. We go from the first element to the last. Classic for loop. Its complexity is at worst O(n) - either it is at the end or not in the array at all.

Interval halving (Binary search)

- The fields must be sorted.
- I take the element I'm looking for and compare it to the middle element in the array.
- Depending on whether it's larger or smaller, the element being searched for must necessarily be in one of the halves or not at all.
- This is how I halve the array until I find the element I'm looking for.
- Since I'm halving the array at each step and not dealing with the wrong half at all, this is a $log_2(n)$ complexity.



Binární vyhledávání (hledané číslo: 6)

1.5.3 Sorting algorithms – classification, description of functionality, informal explanation of complexity of selected algorithm.

Classification

- The two basic groups of algorithms are the so-called internal and external sorting.
  - Internal: All data is stored in operational memory and can be accessed freely by the algorithm.
  - External: Not all data is in memory, some of it may be stored on disk. In case we have too much data. Most sorting algorithms work by comparing two elements. But some work on a different principle - e.g. counting sort, which counts the occurrences of each value. Or radix sort.
- Classification of algorithms according to behaviour:
  - Sorting by selection
    - Selection sort
      - Find the largest element in the array and move to the top.
      - We "discard" this element and look for the largest element in the remaining array.
      - Repeat until the array is in descending order.
      - Complexity n^2
  - Sorting by insertion
    - Insertion sort
      - It moves the elements to the right place.
      - In this way, we go from the beginning of the field to the end, looking for the right place to place it.
      - Although the complexity is n^2, for an almost sorted array it is close to n. For arrays with less than 10 elements it is faster than quick sort.
  - Randomization:
    - Bubble sort

· Compares two adjacent elements. If the one on the left is smaller, it swaps them and moves on.

· The inner loop is done $(n-1)+(n-2)+(n-3)+....=(n^2-n)/2$ which is the complexity of n^2

* Heapsort

· It is based on a binary heap whose basic property is that it behaves like a priority queue. If we remove elements from the priority queue one by one, it is obvious that this leads to ordering.

* Quicksort

· In the array, select any element called pivot.

· Rearrange the array so that the elements on one side are larger and on the other side are smaller than the pivot. Repeat this process for both split sides until we reach an array of size 1.

· Thus, if the choice of pivot is wrong (highest or lowest element), the complexity is n^2, since there is no division of the array. However, the expected complexity for a correct choice of pivot is $n * logn$

– Merge sorting

* Merge sort

· It works on the basis of TWO already descending fields. Let's call them A and B, and then field C, which will be the new merged field.

· There is one array on the input, which the merge sort itself recursively splits and descending sorts.

· At each step, we compare the first elements and move the higher one to the end of the new array C.

· We repeat this process until one of the arrays is empty, and then we move the rest of that array into the resulting array C.

1.5.4 Data structures – array, list, queue, stack, tree, graph.

Array

- Indexed from 0.
- Problem with dynamic allocation in some languages - need to use pointers.
- However, it is simple and therefore fast, because elements are stored in memory in turn, occupy all the same space and are quickly accessed

List

- Individual elements contain also a pointer(reference) to the next element. The last element points to NULL

- In a bidirectional list, the pointer(reference) also points to the previous element. The last one points to the next NULL. The first one points to the previous NULL

Queue

- Elements are removed in the order they arrived - FIFO (First In First Out)
- Mostly functions - Put and Get. Also pointers to the first and last element (Head, Tail)

Stack

- Elements are removed in reverse order - LIFO (Last In First Out)
- Mostly functions - Push and Pop. Also pointers to the first element - Top

Tree

- Each element has at most one predecessor and can have more than one successor.
- It is composed of nodes
  - Root - a node without an ancestor
  - Leaf - a node without a successor
  - Internal nodes - classic nodes that are neither a leaf nor a root
- According to the maximum number of descendants we distinguish trees
  - Unární (list)
  - Binary
  - Ternary
- Elements smaller than the root are sorted to the left, elements larger to the right.

Graph

- These are nodes, edges, and incidences. The incidence assigns an edge to two nodes
- Non-oriented graph - edge is assigned and not paired nodes.
- Oriented graph - edge is assigned to a set of nodes.

1.6 Mathematics

1.6.1 Solution of systems of linear equations.

In mathematics and linear algebra, a set of linear equations is a set of linear equations. Usually these equations contain more than one variable. The solution of such equations is then an ordered n-tuple describing the values of individual variables. The system does not have to have any solution, it can have a certain number or infinitely many solutions. The following is a simple demonstration of a system of linear equations. $x - y = -2 \Leftrightarrow 3x + 2y = 0$

In general, any system can be written as:

$$\begin{array}{ccccccc} a_{11}x_1 & + & \dots & + & a_{1n}x_n & = & b_1 \\ \vdots & & \dots & & \vdots & & \vdots \\ a_{m1}x_1 & + & \dots & + & a_{mn}x_n & = & b_m \end{array}$$

Any system can also be written using a matrix. The number of rows indicates the number of equations, the number of columns-1 indicates the number of variables. The last column contains constants.

$$\mathbf{A}' = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{pmatrix}$$

Some equations in a system may be interdependent, or one may be a linear combination of the other. E.g. $x + y = 1 \Leftrightarrow 2x + 2y = 2$

Such equations do not give us new information and one of them is enough for us. Otherwise, we call the equations linearly independent .

I will denote the attic not containing a column of constants as the matrix of the system A. The last column itself is basically the vector b, which we add to the matrix of the system and thus we get the extended matrix of the system A ' . A '= (A — b).

Number of solutions

- We consider a system of linearly independent equations.
- Frobeni's theorem says: "A system has a solution just when the rank of the matrix of system A is the same as the rank of the extended matrix of system A "'
- Systems where the number of linearly independent equations is greater than the number of variables have no solution. We call them predestined .
- Systems where the number of variables is just equal to the number of equations have exactly one solution.
- And systems where the number of variables is greater than the number of equations have infinitely many solutions. Excess variables can be replaced by parameters and the system can be solved using them.

Solution methods

Equivalent operations

- There are 3 basic equivalent operations that we use to solve systems of linear equations. These operations do not change the result of the equations, they only replace them with equivalent and simpler equations.
- Swapping the order of equations
- Multiplication of both sides of the equation by a nonzero number
- Adding a multiple of an equation to another equation (negative multiple = subtraction)

Gaussian elimination method

- We write the coefficients in the extended matrix of the system
- Forward reduction - We convert the matrix to a stepped shape using equivalent line adjustments.
- We apply Frobeni's theorem. This will determine the number of system solutions.
- Back substitution - From the stepped shape of the extended matrix we assemble a system of equations, from which we can easily calculate the solution of the system. We proceed from the last equation, which contains the least unknown, by gradually substituting into "higher" equations.

Gauss-Jordan method

- The procedure is identical to the Gaussian elimination method, but we do not end up in a deficit form, but continue to the so-called standardized deficit form (A = I = unit matrix)

$$\left[\begin{array}{ccc|c} 1 & 0 & 1 & 4 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 2 \end{array}\right] \begin{matrix} \\ -r_3 \\ -r_3 \end{matrix} \mapsto \left[\begin{array}{ccc|c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

Schodkový tvar -> Normovaný schodkový tvar

-

Cramer's rule

- Cramer's rule is especially useful if you want to algorithmize the procedure, it is easy to program. Solving a system using Cramer's rule by hand is usually more complicated than solving it with the Gaussian elimination method.
- Cramer's rule works only on square matrices A. It uses matrix determinants to calculate. I will not describe further.

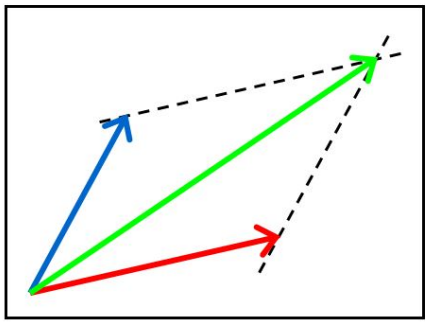
1.6.2 Vector space

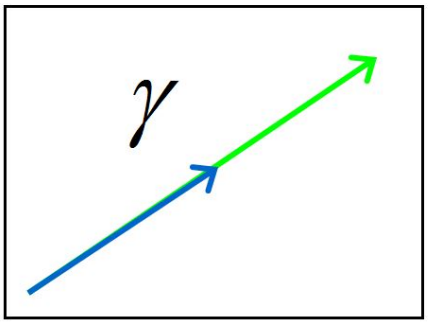
Algebraic operations

- Little info on what an algebraic operation is. This is a representation (we usually consider a binary operation), where we assign a certain result to each ordered pair (a, b) from the set A, also from the set A.
- For example, adding integers is a binary algebraic operation. Consider a pair (2.5). The addition operation also assigns to this pair an element from the set of integers - 7.
- $\mathbb{Z} \times \mathbb{Z} \ni (a, b) \mapsto a + b \in \mathbb{Z}$

Vector space

- Vector space (or also linear space) is a nonempty set V, the elements of which we call vectors. There are two algebraic operations on this set: addition of two vectors (+) and multiplication of vectors by a scalar (•)
- The result of adding vectors is also a vector - eg (a1, a2) + (b1, b2) = (a1 + b1, a2 + b2)
- The result of multiplying a vector by a scalar is also a vector - eg (a1, a2) • b = (a1 * b, a2 * b)
- When proving whether a solid is really a vector space, we must verify the following 8 axioms (the number varies depending on the literature)
- Associative Act for Census $\rightarrow \forall u, v, w \in \mathbb{V} : u + (v + w) = (u + v) + w$
- Existence of a zero vector o $\rightarrow \exists o \in \mathbb{V}, \forall u \in \mathbb{V} : u + o = o + u = u$
- Existence of the opposite vector -uku $\rightarrow \forall u \in \mathbb{V}, \exists(-u) \in \mathbb{V} : u + (-u) = u - u = o$
- Census commutativity $\rightarrow \forall u, v \in \mathbb{V} : u + v = v + u$
- Multiplying the vector by one does not change it $\rightarrow \forall u \in \mathbb{V} : 1u = u$
- Multiplication Association $\rightarrow \forall \alpha, \beta \in \mathbb{R}(\mathbb{C}), \forall u \in \mathbb{C} : \alpha(\beta u) = (\alpha\beta)u$
- Distributivity due to addition $\rightarrow \forall \alpha, \beta \in \mathbb{R}(\mathbb{C}), \forall u \in \mathbb{V} : (\alpha + \beta)u = \alpha u + \beta u$
- Distributivity due to multiplication $\rightarrow \forall \alpha, \beta \in \mathbb{R}(\mathbb{C}), \forall u \in \mathbb{V} : \alpha(a + b) = \alpha a + \alpha b$
- Thus, vector space is defined as a quadruple - Set V (usually R^2), Set T (usually R), addition operations, multiplication operations. The above rules must apply in this area.
- The sum of spaces and their intersection are defined between individual spaces (non-zero, since each vector space contains a zero element)
- The following figure graphically describes the algebraic operations of adding two vectors and multiplying a vector by a scalar (from the left).





-

1.6.3 Linear representation

- Representation between vector spaces X and Y so that sum and product operations by scalar are preserved. This includes linear functions as well as Derivation and Integration.

Nechť U, V jsou vektorové prostory. Zobrazení A: U → V se nazývá lineární zobrazení (operátor), jestliže pro každé dva vektory u, v ∈ U a skalár α platí:

1. A(u + v) = A(u) + A(v)

2. $A(\alpha u) = \alpha A(u)$

- –

PŘÍKLAD 1 Funkce $y = ax$ je lineární transformace $\mathbb{R}$ pro libovolné pevně zvolené $a \in \mathbb{R}$, neboť

$$a(u + v) = au + av \quad \text{a} \quad a(\alpha u) = \alpha au$$

pro libovolná čísla u, v a α.

PŘÍKLAD 2 Funkce $f : y = 2x + 1$ není lineární transformace $\mathbb{R}$, neboť $f(2 + 2) = f(4) = 9 \neq 10 = f(2) + f(2)$.

- –

- Superposition = if we already know the results of x and y, then if we want to calculate f (x + y), it is enough to add x and y (it follows from the definition of a linear representation)
- The fastest way to find out if a representation is NOT linear is to see if, after substituting for all variables 0, we get 0. (since $0 + 0 = 0$) ... if not then it is not a linear representation, if so, we must continue verify.
- Every linear representation is a function, not every function is a linear representation. Each linear representation can be written in a matrix (speeds up the work, I write ASAP).
- View = maps an image from set M to a pattern from set N
- Definition field = set of all patterns
- Value range = set of all images

Zero Space / Display Core

This is such a subset of the domain that A is displayed on the zero vector. Resp. a set of patterns whose images are zero vectors.

1.6.4 Derivation of a real function

- A derivative of a function is a change (increase or decrease) in the image of that function in relation to an (ideally) infinitesimal change in its arguments. The opposite process to derivation is integration . At some point, the derivative can be represented as the direction of the tangent (angle) and tells us how the function changes in the immediate vicinity. The zero derivative at a point means that the tangent is horizontal with the x-axis, so the function does not decrease or increase. This is a so-called "stationary point", which may or may not have a local extreme.

- We can derive the derivative again and get the so-called second derivative, generally a higher order derivative. It logically describes the change of the first derivative in the immediate vicinity. Using the first derivative, we look for stationary points, ie whether the function decreases or increases at a given point. Using the second derivative, we can find out whether the stationary point is a local minimum (f '= 0 and f' '¿ 0) or a maximum (f' = 0 and f " ¡0). In general, the second derivative itself indicates whether fce is convex (f "¿ 0) or concave (f " ¡0).

Toto je seznam některých derivací elementárních funkcí.

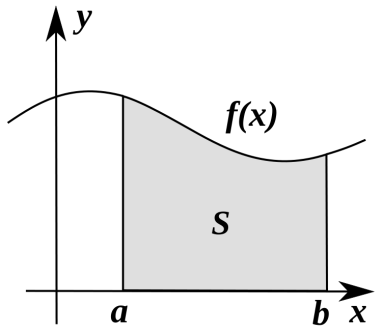
Funkce	Definiční obor funkce	Derivace	Def. obor první derivace
Polynomy			
$f(x) = c$ (c je konstanta)	$\mathbb{R}$	$f'(x) = 0$	$\mathbb{R}$
$f(x) = x^c$ (c je konstanta)	záleží na c*	$f'(x) = cx^{c-1}$	záleží na c
Speciálně: $f(x) = x$	$\mathbb{R}$	$f'(x) = 1$	
Mocniny, logaritmy			
$f(x) = c^x$ (c je konstanta, $c > 0$)	$\mathbb{R}$	$f'(x) = c^x \ln c$	$\mathbb{R}$
$f(x) = e^x$ (e je Eulerovo číslo)	$\mathbb{R}$	$f'(x) = e^x$	$\mathbb{R}$
$f(x) = \log_a x$ (a je konstanta, $a \neq 1, a > 0$)	$x > 0$	$f'(x) = \frac{1}{x \cdot \ln a}$	$x > 0, a \neq 1, a > 0$
Speciálně: $f(x) = \ln x$	$x > 0$	$f'(x) = \frac{1}{x}$	$x > 0$
Goniometrické funkce			
$f(x) = \sin x$	$\mathbb{R}$	$f'(x) = \cos x$	$\mathbb{R}$
$f(x) = \cos x$	$\mathbb{R}$	$f'(x) = -\sin x$	$\mathbb{R}$
$f(x) = \operatorname{tg} x$	$\mathbb{R} - k\pi/2$	$f'(x) = \frac{1}{\cos^2 x}$	$\mathbb{R} - k\pi/2$
$f(x) = \operatorname{cotg} x$	$\mathbb{R} - k\pi$	$f'(x) = -\frac{1}{\sin^2 x}$	$\mathbb{R} - k\pi/2$
Cyklometrické funkce			
$f(x) = \arcsin x$	$\langle -1; 1 \rangle$	$f'(x) = \frac{1}{\sqrt{1-x^2}}$	$(-1, 1)$
$f(x) = \arccos x$	$\langle -1; 1 \rangle$	$f'(x) = -\frac{1}{\sqrt{1-x^2}}$	$(-1, 1)$
$f(x) = \operatorname{arctg} x$	$\mathbb{R}$	$f'(x) = \frac{1}{1+x^2}$	$\mathbb{R}$
$f(x) = \operatorname{arccotg} x$	$\mathbb{R}$	$f'(x) = -\frac{1}{1+x^2}$	$\mathbb{R}$
Hyperbolické funkce			
$f(x) = \sinh x$	$\mathbb{R}$	$f'(x) = \cosh x$	$\mathbb{R}$
$f(x) = \cosh x$	$\mathbb{R}$	$f'(x) = \sinh x$	$\mathbb{R}$
$f(x) = \operatorname{tgh} x$	$\mathbb{R}$	$f'(x) = \frac{1}{\cosh^2 x}$	$\mathbb{R}$
$f(x) = \operatorname{cotgh} x$	$\mathbb{R} - [0]$	$f'(x) = -\frac{1}{\sinh^2 x}$	$\mathbb{R} - [0]$
Hyperbolometrické funkce			
$f(x) = \operatorname{argsinh} x$	$\mathbb{R}$	$f'(x) = \frac{1}{\sqrt{1+x^2}}$	$\mathbb{R}$
$f(x) = \operatorname{argcosh} x$	$x \geq 1$	$f'(x) = \frac{1}{\sqrt{x^2-1}}$	$x > 1$
$f(x) = \operatorname{argtgh} x$	$\lvert x \rvert < 1$	$f'(x) = \frac{1}{1-x^2}$	$\lvert x \rvert < 1$
$f(x) = \operatorname{argcotgh} x$	$\lvert x \rvert > 1$	$f'(x) = \frac{1}{1-x^2}$	$\lvert x \rvert > 1$

•

1.6.5 Definite and indefinite integral

- Together with derivation, they form the two main operations of mathematical analysis. The concept of integral is a generalization of concepts such as area, volume, sum or sum.

- Let us have a function f of a real variable x on the interval [a, b]. $\int_a^b f(x)dx$
- By (definite) integral we mean the content of a surface in a two-dimensional plane, which is bounded by the graph of the function f, the x-axis and the vertical lines x = aax = b.



- An indefinite integral is one that is not "bounded" by points a, b. We are looking for a set of functions which, when differentiated, give the original function. We call this set the Primitive function F. (F '(x) = f (x)) and denote it

1.6.6 Combinatorics

Uspořádané výběry	Bez opakování	Variace bez opakování	$V(n,k) = \frac{n!}{(n-k)!}$
		Permutace bez opakování	$P(n) = V(n,n) = n!$
	S opakováním	Variace s opakováním	$V^*(n,k) = n^k$
		Permutace s opakováním	$P^*(n_1,n_2,\ldots,n_k) = \frac{n!}{n_1! \cdot n_2! \cdot \ldots \cdot n_k!}$
Neuspořádané výběry	Bez opakování	Kombinace bez opakování	$C(n,k) = \frac{n!}{(n-k)!\,k!}$
	S opakováním	Kombinace s opakováním	$C^*(n,k) = \frac{(n+k-1)!}{(n-1)!\,k!}$

- Variations - How many options how to choose from k elements of n , depending on the order (1,2)! = (2,1)
- Permutation - An arrangement of n elements, not selected. Repetition = some element is in the set several times, but we do not distinguish between them (permutation = variation where n = k)
- Example without repetition = number of different arrangements of the word "ABC" with repetition = number of different arrangements of the word "AAC".

- Combination - Variation, regardless of order = how many options to select an unordered k-tuple from n. (1,2) = (2,1)
- Combination number - indicates the number of combinations, we denote by reading $\binom{n}{k}$. It is also called the binomial coefficient .
- Pascal's triangle - a triangle where the number on the next line will be the sum of adjacent numbers one line up. Derived from the combination theorem.

1.6.7 Graphs and their use

Concepts

- An undirected graph is a pair G = (V, E), where V is a set of vertices and E is a set of disordered two-element sets of vertices = edges
- An oriented graph is also a pair, but E is a set of ordered two-element sets of vertices.
- Neighboring vertices are vertices connected by an edge
- Loop one edge leading from one node to itself (Usually not allowed in unoriented graph - the so-called pseudograph is created)
- Subgraph - The set of its vertices is a subset of the original, and the same is true for edges, where an edge can only exist between included vertices.
- Factor - subgraph containing all its vertices (differs only by edges)
- Vertex degree - the number of edges associated with the vertex, the oriented graph has two degrees - input and output
- Sequence - sequence of vertices and edges; it begins and ends at the top
- Move - a sequence in which there is no edge more than once.
- Euler's move - a move that includes all edges of the graph
- An undirected continuous graph has Euler's move just when - it contains two nodes of odd degree and the rest has even degree.
- An oriented continuous graph has an Euler thrust just when - one node has an output stage higher than the input stage, the output stage one lower than the input stage, and the rest have the same output and input stages.
- A graph containing Euler's move is called an Euler graph
- Euler's circuit - the same as Euler's move, moreover ends at the place where it started (Closed Euler's move)

- An undirected continuous graph has an Euler move just when all degrees have an even degree
- An oriented continuous graph has an Euler move just when all vertices have the same input and output stage
- Path - a move in which no vertices are repeated
- Hamilton's Path - A path containing all nodes
- Circle / cycle - Closed path, cycle is in oriented, circle in non-oriented; similarly, there is a Hamiltonian circle
- Link -
- Tree - minimal continuous graph without circles, removing any edge will make it discontinuous, adding will create a circle
- Forest - A set of unconnected trees.
- Root tree - an oriented tree in which one vertex is cut as a tree
- Spanning tree - A subgraph that contains all vertices (= factor) and is also a tree (does not contain circles)
- Isomorphism of graphs - the same graphs otherwise named / twisted (there is a complete representation from one graph to another). Same number of edges, vertices, same degrees, same neighbors ...

Types of graphs

- Oriented / non-oriented - oriented / non-oriented edges (described above)
- Rated / unrated - Edges can have a value (price), usually used, for example, to show the distance between cities, etc., Edges can also have a condition, etc.
- Complete chart - each vertex associated with each
- Planar graph (planar) - a graph whose edges do not intersect at any point
- "k - partite" - a graph that can be divided into k parts so that the vertices in one part do not share any edge with each other. Respectively, each vertex is adjacent only to a vertex from another group. We are most interested in a bipartite graph (does not contain odd circles)
- Network -oriented graph, where each vertex has a non-negative rating ($x >= 0$)
- Symmetric = for each vertex, the number of edges from X to Y is the same as from Y to X

- Chromatic number - the minimum number of colors with which the graph can be colored so that adjacent vertices do not have the same color.

Algorithms

- Kruskal's algorithm - search for the minimum skeleton of the graph (we add all edges with the lowest possible price so that there is no circle)
- Dijkstra's algorithm - finding the shortest path in an oriented graph. We start at the beginning, go through all the edges and update the distance. We take the vertex with the smallest distance and update the vertices for all its edges. We move on until we have any peak to go through. The result is the shortest distances to all vertices in the graph relative to the origin.
- Graph coloring - NP complete problem.

Using a graph

- Trees - efficient search and sorting
- Description of relations - eg who is whose friend, what is the divisor of what, etc.
- The shortest way - Dijkstra's algorithm
- Maximum flow
- Vending machines
- "Units": 3

2 Subject Computer Science and Technology

A. Subject Computer Science and Technology

1. Introduction to Theoretical Computer Science

a) Interpretations and models in first-order predicate logic. Resolution logic.

Interpretations and models

For experts:

- Interpretation = each predicate symbol is assigned an n-ary relation from the given Universe

ANOTHER EXAMPLE OF INTERPRETATION $\mathcal{A}$:

- UNIVERSE $A = \{a, b, c, d, e, f, g\}$
- $P^{\mathcal{A}} = \{b, d, e\}$
- $Q^{\mathcal{A}} = \{a, b, e, g\}$
- $R^{\mathcal{A}} = \{(a, b), (a, e), (a, g), (b, b), (c, e), (f, c), (f, g), (g, a), (g, g)\}$

- Valuation = for a given interpretation I with the universe U is the valuation of a function that assigns elements from the universe to individual variables
- We therefore assign truth values in a given interpretation with a given universe and in a given valuation
- Model = Valuation for a given interpretation in which the given sentence is true

For dummies:

- Interpretation = the body of the predicate that the predicate evaluates
- Valuation = assignment of specific values to variables
- Model = if the interpretation is true at any valuation, it is a model

Resolution method

1. Transfer to CNF
2. Every elementary disjunction is a clause
3. This is proof by dispute -> we negate the conclusion and look for an empty clause = dispute
4. We assign one-liter clauses to other clauses containing the opposite clause and derive new clauses (b a! B + c => c)
5. We search until new clauses can be found, or until we find an empty clause ⊥

b) Nondeterministic finite automata, the closure properties of the class of regular languages with respect to different operations on languages.

Nondeterministic automata

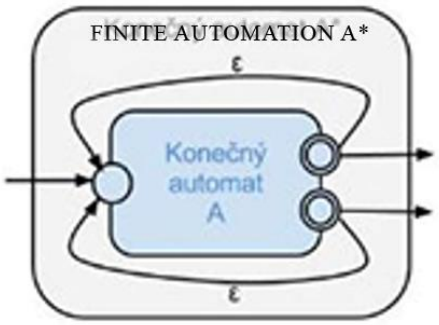
- Compared to deterministic automata, they can have multiple initial states
- Unlike DKA, there may not be exactly as many rules for each state as the size of the set of received symbols. There may be more (repeat) or fewer, or they may accept an empty word.

Closedness of regular languages

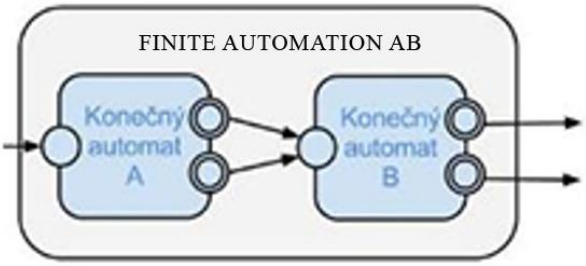
It was previously defined what a regular language is. Another definition could be that a Regular Language is a language that can be described by a finite state machine (NDKA / DKA). Then there is the regular language class REG, which contains all regular languages. When we select x languages of the REG class and perform a certain operation, then we get the language of the REG class again.

"The closure of a set over an operation means that the result of the operation with any

elements from the set will again fall into that set."
For example, we can say that the REG class is closed to iteration. It is easy to make an NDKA describing an iteration - the ergo resulting language can be described by a finite automaton - the ergo language is regular.



KA - ITERATION



KA - CONCATENATION OF 2 LANGUAGES

In short - the REG class is closed to (incomplete list) - unification, intersection, complement, concatenation, iteration, or eg mirroring

c) Regular expressions and their relation to finite automata.

A string describing a whole set of strings, specifically a regular language.

Definition of regex:
1. alphabet Σ
2. ∅, ε, and (where a ∈ Σ) are regular expressions
3. If α, β are regular expressions, then i (α + β), (α · β), (α *) are regular expressions
For (α · β) the dot is usually not written (concatenation) and can be used directly αβ.
Since both a regular expression and a finite state machine describe a regular language, then we can say that each regular expression can be converted to a finite state machine and vice versa.
Algorithms and algorithmic problems, computational models
Described above in sections 4 and 5 for ICT.

d) Context-free languages and grammars.

Context-free grammar is formally a quadruple G = (Π, Σ, S, P)
- Π = set of nonterminals (A, B, C ...)
- Σ = set of terminals = characters themselves (a, b, c, ... / 1,2,3, ...)
- S = initial nonterminal
- P = finite set of rewriting rules

Context-free language = language accepted by the so-called stack automaton. It is a set of languages whose subset is formal languages.
A context-free language is a set of all words generated by a context-free grammar G.
Derivation = sequence of derivation of a certain word in a certain grammar; we distinguish between left and right derivatives, according to the direction of nonterminal replacement
Derivation tree = graphical representation of derivation.
Grammar equivalence = generate the same language; There is no word that is in one, but not in the other; Algorithmically undecidable
Grammar ambiguity = there are two different derivation trees for getting a certain word in a given grammar.

e) Computational complexity of problems, complexity classes.

Described above in Part 6 - "Computational Complexity of Algorithms, Asymptotic Notation"

2. **Computer Architectures, Computer Networks**

a) IEEE 802 standards. Ethernet. IEEE 802.11 wireless networks.

IEEE 802 standards

IEEE 802 is a group of standards dealing with LAN networks, including:

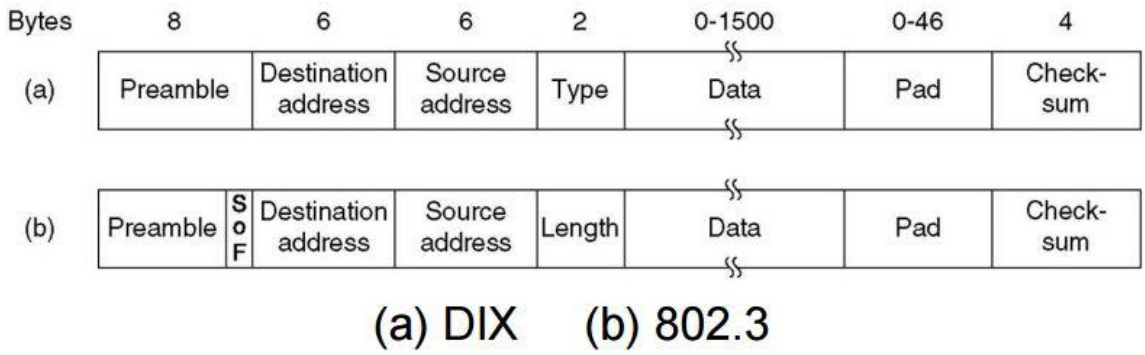
- 802.2 - LLC
- 802.3 - Ethernet
- 802.5 - Token Ring
- 802.11 - WiFi

Defines the division of OSI-RM Layer 2 into MAC and LLC sublayers.

- LLC
  - Multiplexing. -> Allows multiple network protocols (IP, IPv6, IPX, DECnet, AppleTalk, X.25, CONS) to be used on the same network at the same time.
  - provides error correction.
- MAC
  - Access to the medium, conflict resolution ...

Ethernet

- Identical to TS
- Summary of technologies for LAN networks.
- IEEE 802.3 standard
- Naming by format: {Mbps | GbpsG} {Base | Broad} {seg_len / 100 [m] | -medium}. Eg: 10Base-T
- Coax, TP, Optics.
- For Half-Duplex CSMA / CD (media access)
  - 1. Listens to see if the media is free.
  - 2. If it is free, it starts transmitting and listens for any signal from another station. If, yes, a collision has occurred (multiple stations can detect free media and start broadcasting). Stops the transmission and sends a jam signal.
  - 3. The station waits a random amount of time before trying to retransmit. If several consecutive collisions occur, the time increases exponentially.
- There are multiple versions of the frames. The most common is DIX / Ethernet II.



(a) DIX (b) 802.3

Basic comparison of some Ethernet variants

- 10Base5 = Thick Ethernet - 10mm coax, up to 500m long segments
- 10Base2 = Thin Ethernet - 5mm coax, up to 185 long segments
- 10Base-T - Star topology, UTP3 cables, range 100m from the hub
- Fast Ethernet - similar to 10Base-T, 100Base-TX (UTP5), 100Base-FX (single / multi mode optical fiber)
- Gigabit Ethernet - 1000Base-T, 1000Base-SX
- 10 Gigabit Ethernet - full duplex only (no CSMA / CD), UTP6; 10GBaseT

- IEEE 802.11 wireless networks
- These are wireless networks operating in the 2.4 and 5 GHz bands.
- Several 802.11-802.11ac standards ...
- 802.11a
    - OFDM modulation
    - about 5GHz
    - about 19 channels

b) Routing in computer networks. Routing protocols.

Routing in general

- Circuit-switched
    - Derived from old telephone networks
    - created a channel for two sides throughout the communication
    - about inefficient
- Packet-switched
    - Dividing data into packets and sending from source to destination
    - Messy, every packet could go the way he wanted
- Virtual channels
    - Compromise between Circuit and Packet switched
    - A (virtual) channel is created as in circuit-switched

Routing variants

- Centralized vs. Distributed
    - There is a central Routing Control Center (RCC) in the centralized
    - RCC collects information about routers and calculates routing tables for them
    - Routers periodically send their information to the RRC
    - In a distributed network, each router knows its surroundings and exchanges routing information
- Static vs. Dynamic
    - In static mode, admin creates routing tables manually.
    - It's more secure, less CPU intensive but non-adaptive. Used in small intranets.
    - For dynamic, one of the protocols is used to build routing tables.

Routing protocols

Distance Vector Algorithm (DVA) or Link State Algorithm (LSA).

TWO

- Routers do not know the topology of the whole network, only their neighbors - next hop address
- Routing table is periodically broadcast to neighbors (for newer versions then Trigerred update)
- Hop count is specified as the unit
- Slow convergence when changing topology
- An example is RIP

LSA

- Routers know the topology of the entire network. This information is stored in a database for them.
- Each router calculates the shortest route to all the others (Djikstra)
- Fast convergence when changing network topology
- The example is OSPF

c) Topologies of computer networks, transmission media, deterministic and non-deterministic media access protocols.

Topology of computer networks, media, conflict and collision-free methods of media

sharing.

Topology

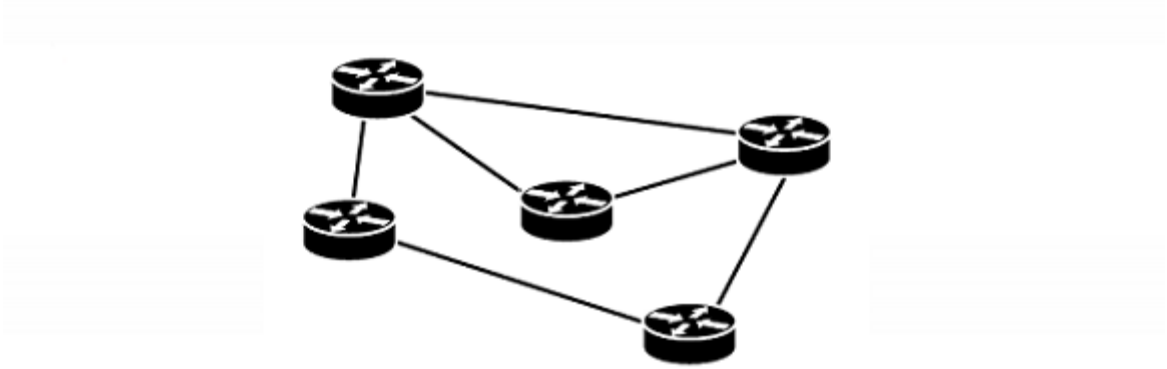
In topologies, we speak mostly LAN vs WAN topologies.

LAN topology:

- Bus
  - One backbone cable, terminator on both sides.
  - Reception to all stations, low security
- Star
  - One central element in the middle
- Tree
  - Star expansion - more stars connected together.
- Circle - stations connected via a medium to a circle

WAN topology:

- Mesh



- Point-to-point links between routers
- Alternate paths may exist
- Hub-and-spoke and full-mesh are the most favorite examples

Average

- Metallic
  - Asymmetric
    - Coax
    - The core is made of copper wire
    - more resistant to interference, but expensive
  - Symmetric
    - Twisted pair
    - Typical media in LAN networks
    - Baud rate according to cable quality (UTP cat1-8)
    - Contains 4 twisted pairs of wires, - twisting suppresses electrical noise
    - 2 types - shielded (STP) and unshielded (UTP). At STP, everything is insulated, the mantle itself -> quality
- Optical
  - High transmission speeds, no electromagnetic interference, low losses
  - Light / radiation is transmitted on the principle of total reflection - two horns with different refractive index
  - Fiber types:

- Single-species
- Transmits only one "vid".
- Expensive, but there is no fashion dispersion.
- Multimodal
- Low price, but there is a fashion dispersion. This is solved, for example, by a gradient fiber (refractive index is not the same in the main fiber).

Media sharing methods

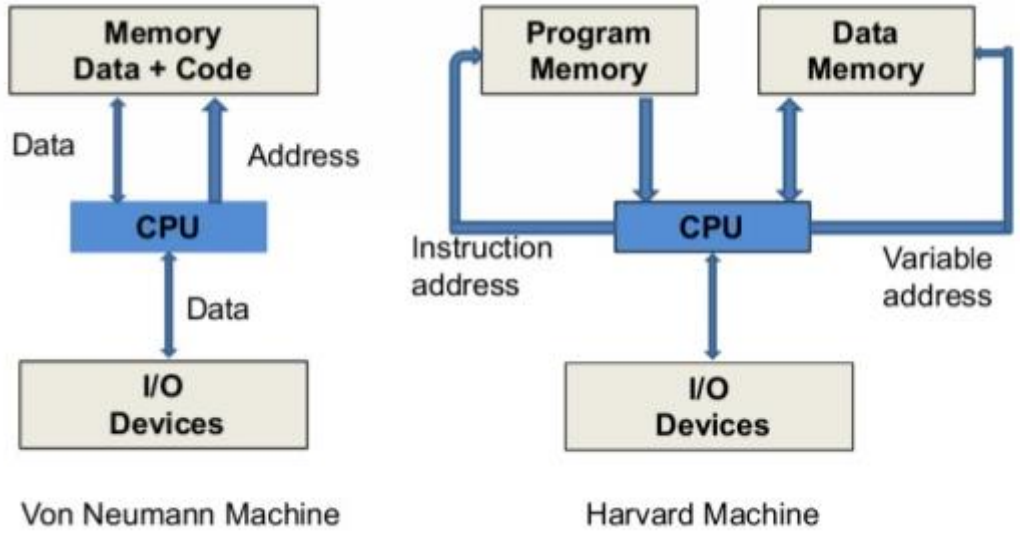
- Collision - CSMA / CD (Carrier Sense Multiple Access with Collision Detection)
  - 1. Listens to see if the media is free.
  - 2. If it is free, it starts transmitting and listens for any signal from another station. If, yes, a collision has occurred (multiple stations can detect free media and start broadcasting). Stops the transmission and sends a jam signal.
  - 3. The station waits a random amount of time before trying to retransmit.
- Carrier Sense Multiple Access with Collision Avoidance (CSMA / CA)
  - 1. If the medium is free, it will start transmitting, if the other party does not confirm the reception, it will wait and try again.
  - 2. If the media is busy, it waits for a random amount of time and then checks the media again.
  - 3. It is possible to use RTS / CTS (RTS - Request To Send, CTS - Clear To Send) before sending the packet itself.

d) Microcomputers, basic construction features. Common integrated peripherals and their characteristics.

Microcomputers, basic construction features

For a number of non-demanding applications, small computers integrated in a single housing are produced - thus forming a single unit -> monolith.

The Harvard concept of computers is mostly used, ie there is a separate memory for data and program. The advantage, then, is to use different memory technologies for the data and for the program.



- For DATA:
- Energy dependent memories (RWM-RAM)
- The contents are not retained after disconnection
- Memory cell implemented by a flip-flop

- Pro PROGRAM:

- ROM type memories (EPROM, FLASH EEPROM)
- The contents are retained even after disconnection

According to the purpose of the memory cell, we can find 3 basic types of memory in monolithic PCs:

- Work registers
  - for current data (results of operations)
  - operand of machine instructions
- Universal notebook registers
  - about storing the most frequently used data
- RWM data memory
  - about storing large and less used data
  - Usually the instruction file does not allow manipulation of this data, only their transfer

Synchronization

The machine and clock cycles need to be synchronized. Sometimes integrated internally - directly on the chip, but due to overheating, etc. higher deviations occur. If it is the exact time executed by the instruction -> external synchronization source. These can be:

- Silicon crystal
- Ceramic resonator
- LC or RC circuit

RESET monolithic PCs

The initial state of the computer is precisely defined -> referred to as RESET. All registers will be set to (usually) 0, etc.

Interference protection

Usually, we do not have an ideal environment and this causes interference:

- Mechanical influences - we must be able to securely connect it to the board, it must be resistant to vibration and shock.
- Electromagnetic influences
- Programmer errors

Usual integrated peripherals, their characteristics.

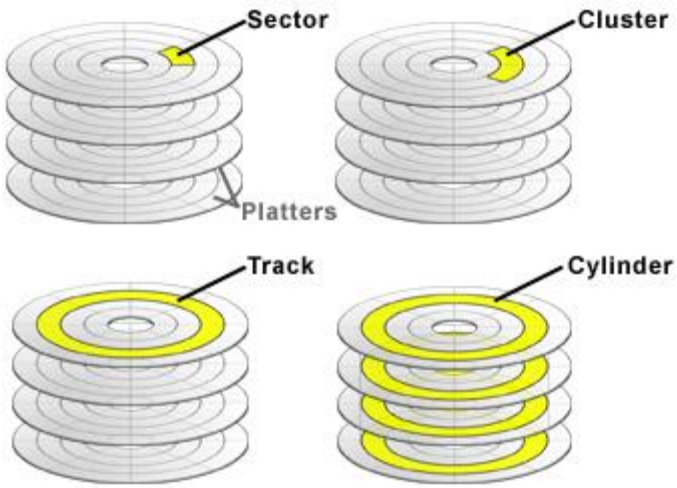
= circuits that ensure the communication of the microcomputer with the environment.

- Entrance and exit gates
  - The simplest and most commonly used I / O interface
  - 4 or 8 single-bit pins, where 1/0 can be read and written at the same time
- Counters and timers
  - The counter is incremented by some external signal
  - The timer is incremented by an internal clock signal
- A / D, D / A converters - conversion of physical (ANALOG) quantities (voltage, current, resistance) to (DIGITAL) numerical form.

e) External computer memories: hard drives and optical media. Displays: CRT, LCD, OLED, E-ink.

Hard disks

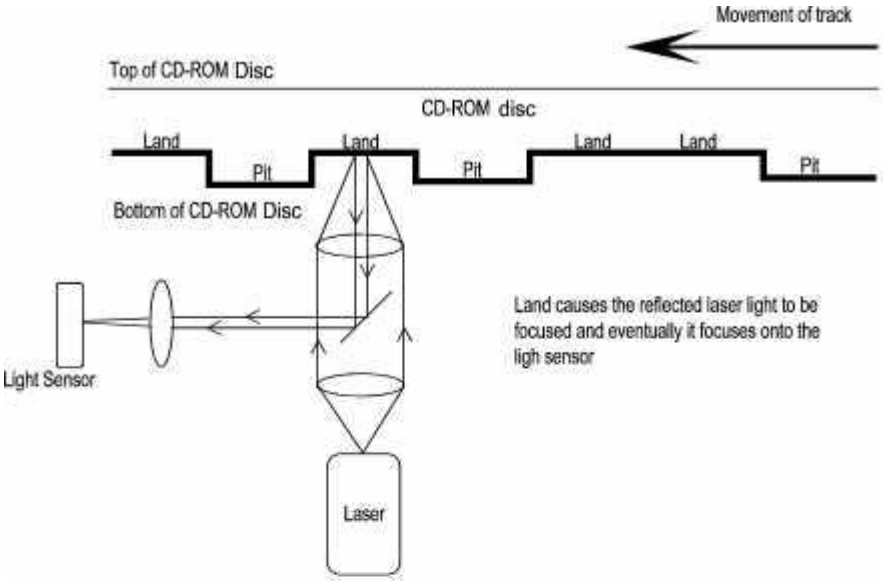
- Data is stored in bytes on the disk
- Bytes are arranged in groups of 512 -> sectors. It is also the smallest data unit that can be R / W to disk.
- The sectors are grouped into tracks and the tracks are grouped into a cylinder.

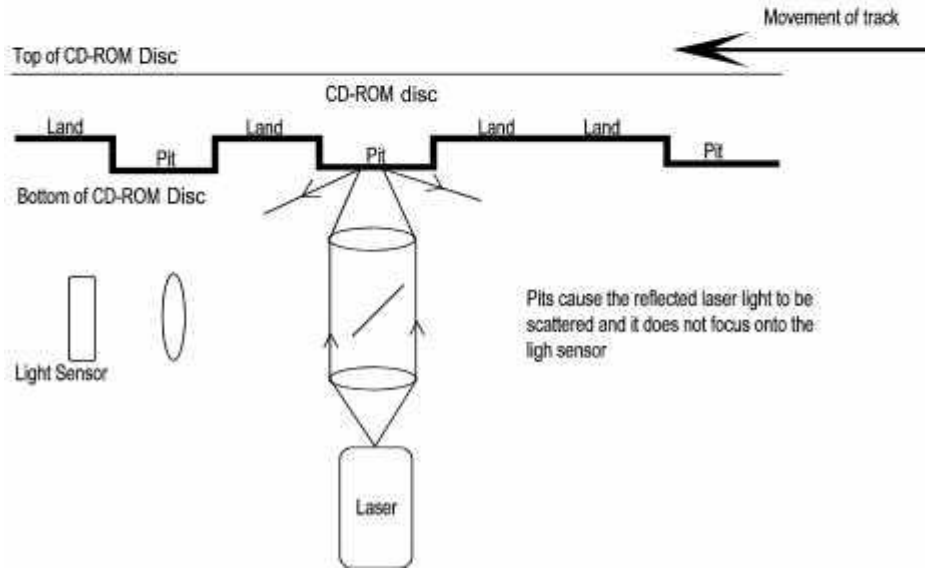


- R / W method on HDD:
- Implemented using magnetization directions
- Head to read - it takes the most time to move the head, so the fastest to read are files whose sectors are on the same track and the tracks are placed on top of each other.
- Recording head - An electric current passes through the coil, creating a magnetic field that magnetizes the surrounding environment (air gap) according to the direction of flow.

Optical media

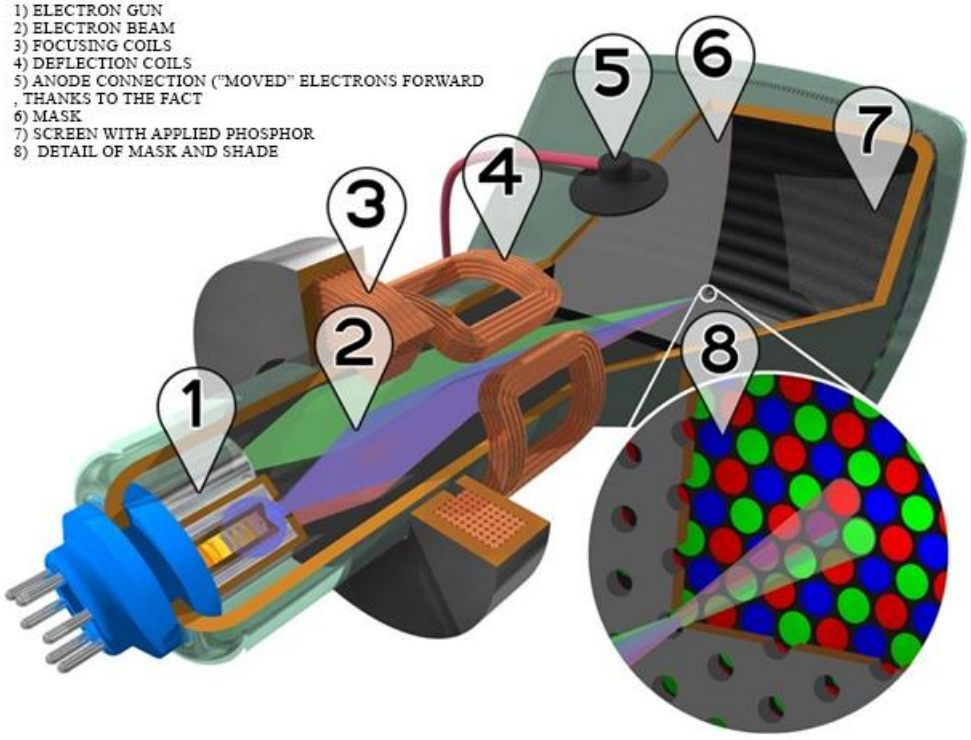
- Pits and Lands.
- The principle is based on the reflection of rays - different reflection of the beam from pits and land.
- Spiral track -> the player must change the disc rotation speed (CLV - constant Linear Velocity)
- The difference between CD and DVD is in the dimensions of the holes and the mainland.



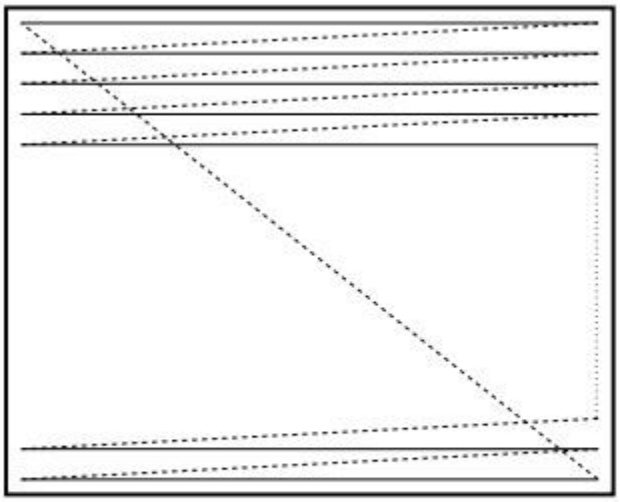


CRT

1) ELECTRON GUN
2) ELECTRON BEAM
3) FOCUSING COILS
4) DEFLECTION COILS
5) ANODE CONNECTION ("MOVED" ELECTRONS FORWARD , THANKS TO THE FACT
6) MASK
7) SCREEN WITH APPLIED PHOSPHOR
8) DETAIL OF MASK AND SHADE



- The CRT consists of a glass flask with a vacuum inside.
- Because the monitor is an analog technology, the digital signal from the graphics must first be converted to an analog signal that the monitor will understand. DAC converters (Digi to Anal converter) are used for this. These create "analog tables" that contain the necessary voltage levels to display the R G B color.
- At the beginning of the flask is an electron gun that serves as a cathode.
- It fires a stream of electrons at high speed for each of the 3 colors. These are negatively charged.
- When firing is in charge of Whenelt's cylinder, which is around the cathode. It has a negative potential and thus directs the electrons into a narrow beam and at the same time, by changing the voltage, it can regulate the amount of electrons fired and thus the brightness.
- The electrons continue to pass through a series of grids to get them to the screen. They also have special functions - one of them (grid g3) electrons the blade and the other (g6) causes convergence (if there is poor convergence, then the image is out of focus. See link
- Electrons continue to pass around the deflection coils. These determine exactly where the electron beam should hit the screen. The electrons move gradually from the upper left corner of the screen to the right, then go down a line and continue again from left to right. This is called rasterization.
- Once the lower right corner is reached, one refresh cycle is completed. The entire path of the beam across the screen is referred to as the "field" (?? 420 smoke weed ??). The rendering shows this picture well.

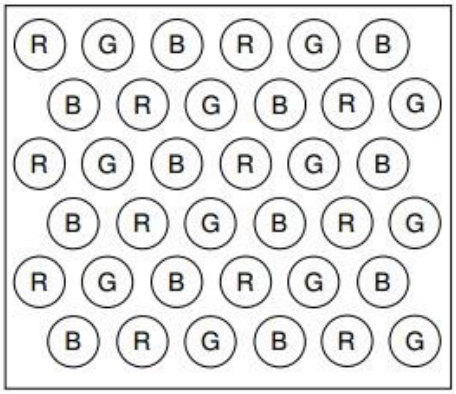


Obrázek 2: Princip řádkování

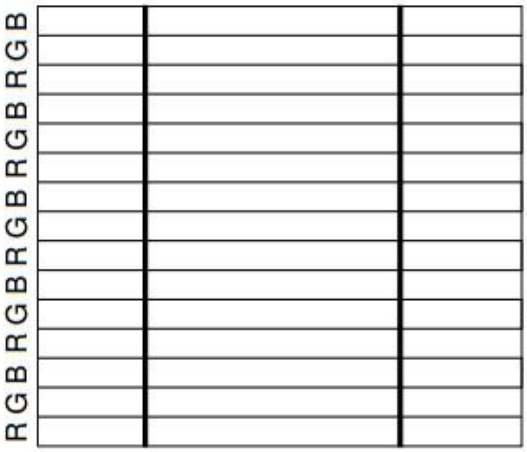
Figure 2: Line spacing principle

- The screen refreshes 60x per second (classic 60Hz refresh rate, also known from LCD)
- It must pass through the mask before the electron hits the screen. This is because the electrons repel each other, so the transmitted beam would be blurred.
- The mask is either a metal plate with circular holes (called Invar and the holes are arranged in triangles) or (l) Trinitron is used. He solved it so that the mask was not a metal plate, but horizontal wires (which held in place 2 more vertical wires, which were placed in the thirds, see the picture below)
- The mask is made of metal, so it is susceptible to thermal expansion and magnetic fields. This causes the electrons not to land directly on the spot and distort the colors. This is solved by rounding the mask (therefore the glass ("screen") of the monitor is rounded)

- As soon as the electrons pass through the mask, they hit the phosphor. It is a special substance that receives energy from electrons and emits it as light. The phosphor is applied to the screen to form a pixel matrix. See picture

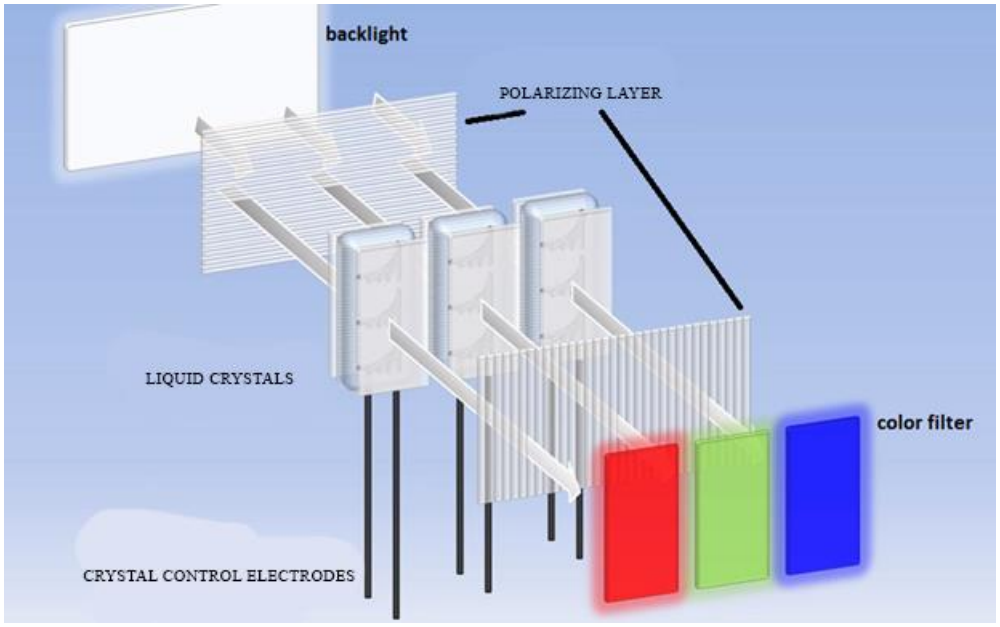


Invar



Trinitron

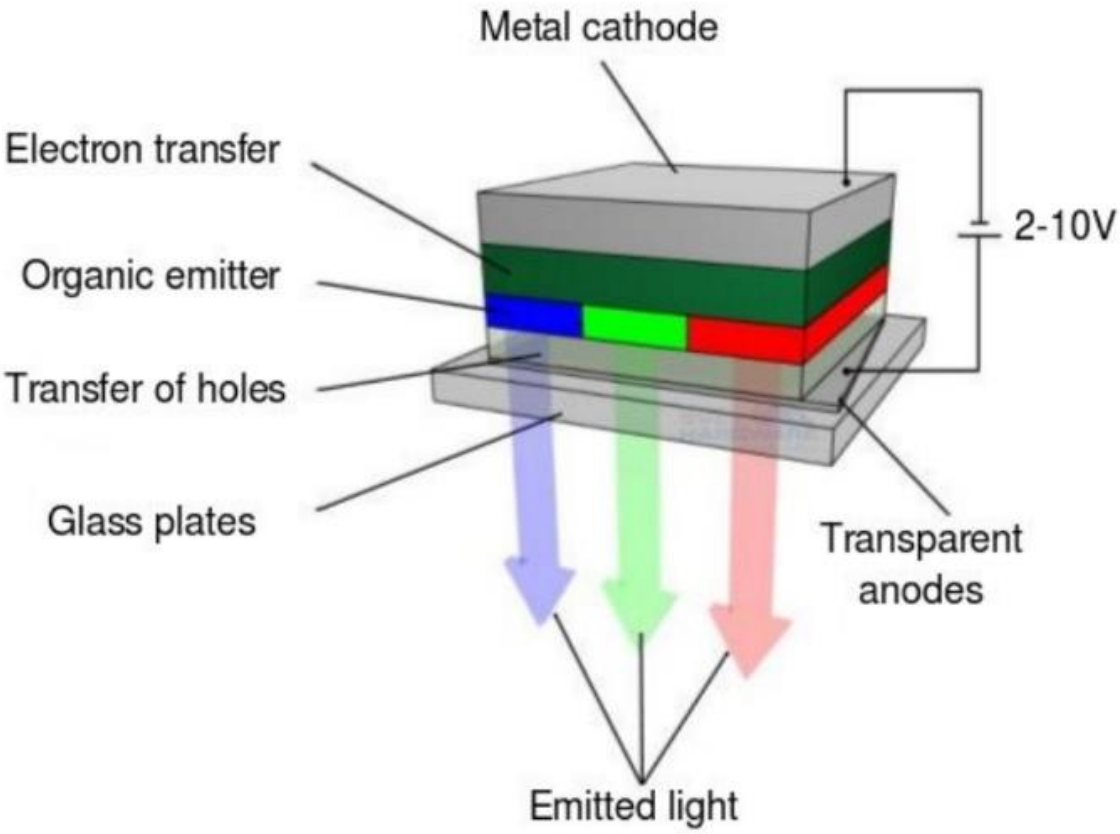
LCD



- The main active element - liquid crystals.
- Polarizing filters rotated 90 °
- The liquid crystal rotates the stream of light to pass through the second polarizing filter.
- We can regulate the crystals with the help of electrodes - depending on how much current we let into them, we can regulate how much light reaches the color filter and thus create different colors (RGB base).

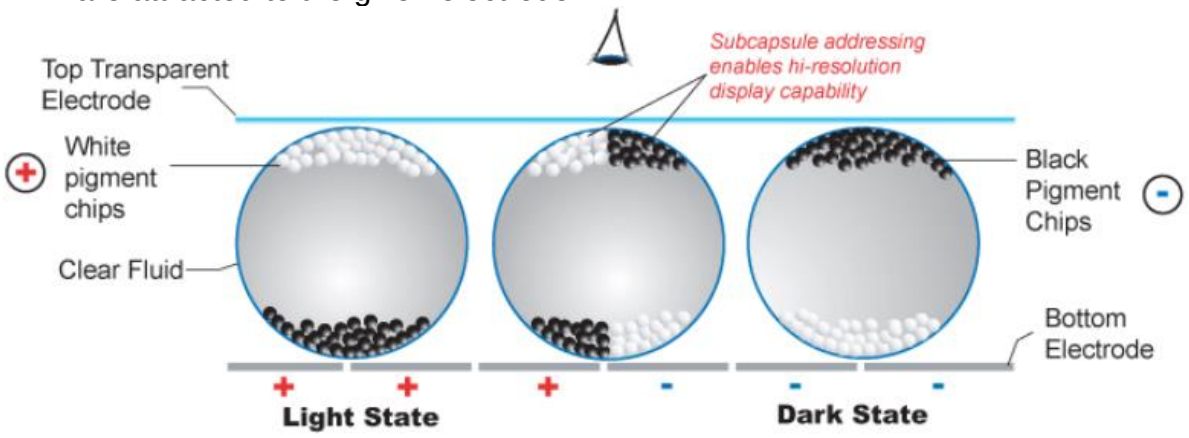
OLED

- There are organic layers between the cathode and the anode, which emit photons (collision of electrons and holes) as current flows from the cathode to the anode. This phenomenon is called recombination
- Organic layers are a light source in their own right - no backlight is needed.
- High contrast, very thin, full color, installation even on a flexible surface, low consumption.



E-ink

- Individual points formed by capsules.
- The capsule contains an electrophoretic solution.
- In solution - negatively charged black particles (ink) + positively charged white particles.
- Capsules placed between the electrodes and by means of voltage the particles are attracted to the given electrode.

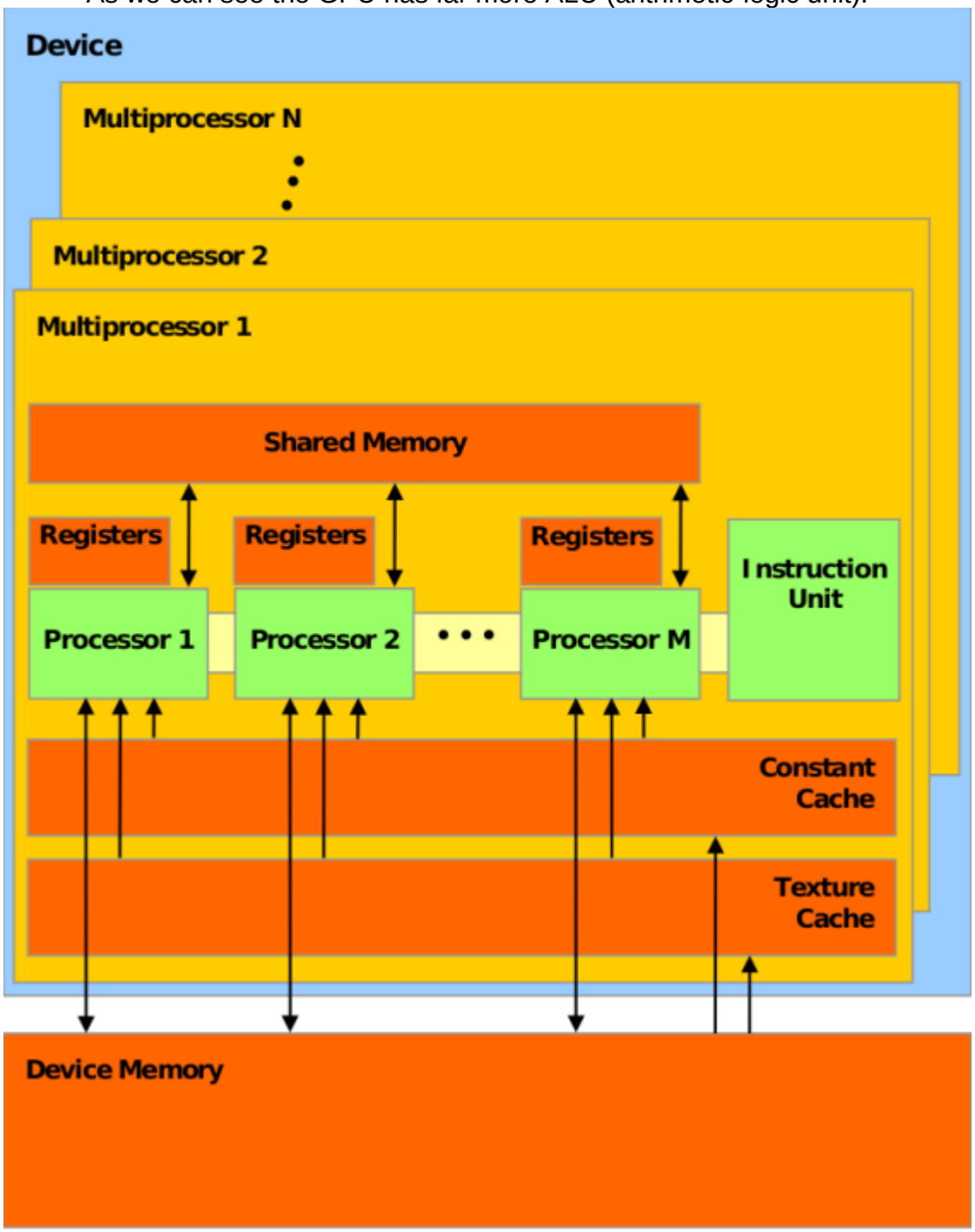


f) Parallel graphical processor architectures (e.g. CUDA, OpenCL, etc.).

CUDA

- = NVIDIA software and hardware extensions that allow us to use GPU performance for computing.
- Unified programming of all NVIDIA cards. It is a C / C ++ extension

- As we can see the GPU has far more ALU (arithmetic-logic unit).



- The GPU is divided into several Multiprocessors

- The GPU has a shared memory (Device Memory), which consists of three parts Gloval, Texture and Constants memory. -> Multiprocessors can (only) share data via this memory.
- One multiprocessor then contains several processors and they exchange data with each other via shared memory in the multiprocessor.
- In addition, each Multiprocessor contains a cache of constants and textures, to speed up work. (both are READ-ONLY)

3. Programming

a) Recursion – examples of recursive algorithms, complexity of recursive algorithms, elimination of recursion.

A function that calls itself. Contains a condition for termination. Classical recursive functions: factorial, fibonation sequence.

Factorial

```
public static int faktorial(int n) {
    //podminky pro vyskoceni
    if (n < 0) return -1;
    if (n == 0 || n == 1) return 1;

    //rekurzivni volani
    return n * faktorial(n - 1);
}

public static int faktorialBezRekurze(int n) {
    int vysledek = 1;
    //for cyklus ktery jede od 1 do n a prepisuje vysledek
    for (int i = 1; i <= n; i++) {
        vysledek = vysledek * i;
    }
    return vysledek;
}
```

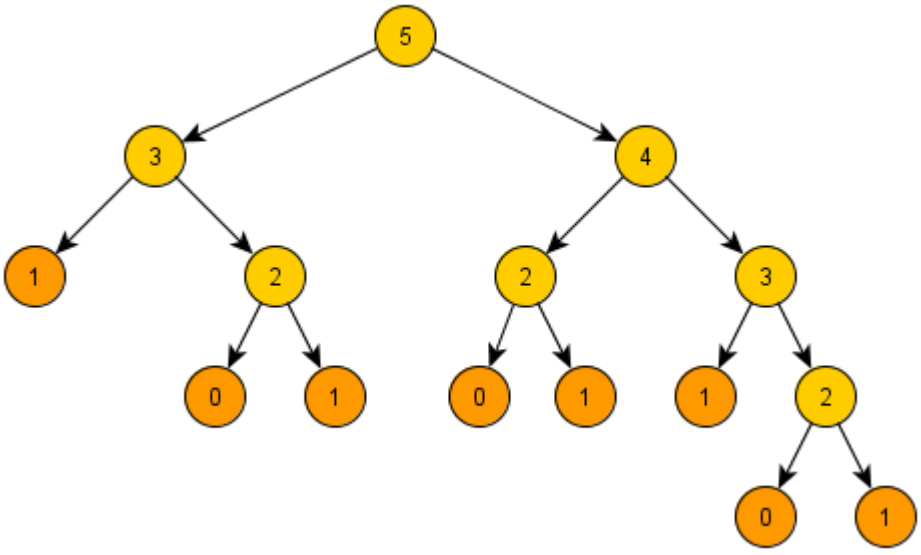
- In this case, it is a complexity of O (n), because it runs n times.

Fibonac sequence

- A sequence of natural numbers, where each additional number is the sum of the previous two.
- Often solved by inefficient recursive method.

```java
//fibonacciho cisla:
//1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987
public static int fibonacci(int index){
    if(index == 0) return 0;
    else if(index == 1) return 1;
    else return fibonacci(index - 1) + fibonacci(index - 2);
}

public static int fibonacciBezRekurze(int to)
{
    if (to == 0)
    {
        return 0;
    }
    if (to == 1)
    {
        return 1;
    }
    int prvni = 0;
    int druhe = 1;
    int result = 0;
    for (int i = 1; i < to; i++)
    {
        result = prvni + druhe;
        prvni = druhe;
        druhe = result;
    }
    return result;
}
```

- In this recursive method, however, there is a very inefficient calculation of O (2 ^ n) and below we see that for example for the sequence n = 5 we calculate the value 3 twice, we calculate the value 2 3 times and we even look at trivial values (0,1). 8x



- In this case, we can remove:
    - For cycle, see. code.
    - Using dynamic programming - keep the calculated values in some collection and instead of counting the values we already have calculated, just pull them out of this collection. This reduces the complexity from O (2 ^ n) to O (n).

Delete recursion

- It can usually be replaced by a for loop.
    - We save time for function calls, including parameter passing
    - For the cycle, the compiler can apply optimizations (condition prediction), while for recursive calls it cannot.
- Recursion is stored on the stack, and a record containing information and parameters is created for each function called.
- When it is finished, this entry is removed from the tray and advanced.
- We cannot always replace recursion with a cycle. But what always works is replacing recursion with your own stack. This is because more information is stored on the default stack than is needed in some cases.

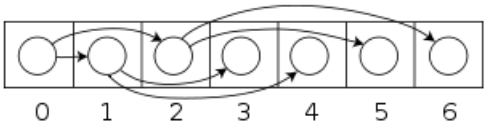
b) Tree data structures – binary tree, B-tree, description of related algorithms, explanation of complexity of selected algorithm.

Tree data structures in general

- It consists of a root and inner and end nodes. = Root, node, leaf.
- Specific trees usually have a defined maximum number of descendants - binary, ternary, B-tree (it has a maximum of N descendants).
- They contain subtrees - each inner node can be viewed separately as the root of our own mini-tree.
- Properties:
    - Height - maximum depth / level of the tree
    - Width - number of nodes at one level
    - Path - a sequence of nodes from the root to the searched node
    - Path length - number of path edges (number of sequence nodes - 1)
    - Balance - all sheets are not much deeper than any other sheet. Or else - the depth of the subtrees differs by a maximum of x *.
    - - the exact number varies depending on the specific implementation, but it is usually 1.
- Browse the tree
    - Wide - scroll through the levels
    - In height / depth - starts at the root and goes to the offspring. If we come across a letter or the node has already been visited, we return.
    - We determine 3 basic methods in order.
        - Preorder - root -> left subtree -> right subtree
        - Inorder - left subtree -> root -> right subtree
        - Postorder - left subtree -> right subtree -> root

Binary tree

- Each node has 0-2 descendants.
- Each node has exactly one parent. The exception, of course, the root.
- Nodes contain a key to search for. Keys smaller than the root in the left subtree, keys larger than the root in the right subtree.
- Efficient sorting and searching. For exported trees, this is O (log_n). Therefore, it is important to have a balanced tree.
- Implementation:
    - Dynamic structure, ie data and references to the left and right descendant. The downside is that we have frequent null pointers when we have no children and waste memory.
    - Using arrays - we use implicit data structures (ie data structure that uses very little memory besides the actual data elements)



FOR THE NODE WITH INDEX I WE FIND DESCENDANTS:

LEFT -> **2i+1**
RIGHT -> **2i+2**

PARENTS FOR THE NODE WITH INDEX I CAN BE FOUND:

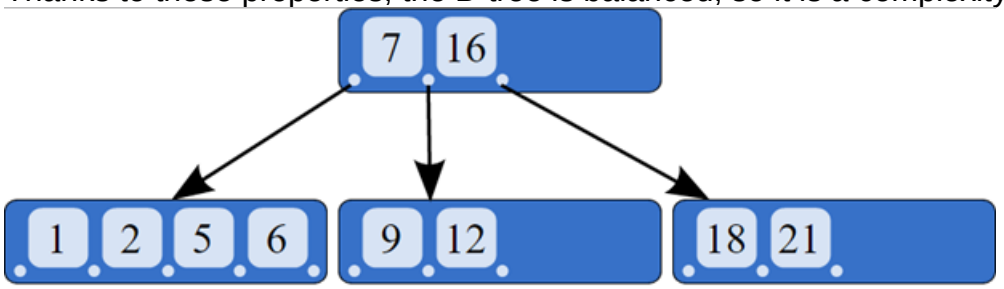
$$\left\lfloor \frac{i-1}{2} \right\rfloor$$

B-tree

It is a tree specific in that it has N descendants.

- The root has at most N offspring. Resp 2-N.
- Internal nodes are half-full, so they have N / 2 to N descendants.
- All sheets are at the same level.

Thanks to these properties, the B-tree is balanced, so it is a complexity of O (log_n).



B-TREE, WHERE N = 5
EACH NODE THEREFORE CONTAINS N/2 TO N-1 KEYS

- Searching is easy, you just compare the number and proceed correctly.
- Insert more or less the same, but when the node overflows, it must split and the middle element goes to the front.

c) Implementation of OOP in programming languages – description and comparison.

- C ++ allows a class to inherit from multiple classes, Java from only one
- Java contains interfaces - we can implement multiple interfaces and seem to mimic multiple inheritance but more securely
- C ++ can have separate methods (the program must have main and then it doesn't matter where the other methods are), in Java all methods belong to a class.
- Java is Pure OOL so is C #, c ++ is mixture of OOL and structured language, so is python.
- C ++ static types, Java and c # support generics (dynamic types)
- Some languages allow you to easily change classes (even at runtime?)
- Java forces the programmer to program everything as OOP - all classes inherit from the Object class
- Working with objects in memory - destructor vs. Garbage Collector
- in C ++ Class something = Object; v Jave Class neco = reference
- c ++ references are rebindable c * ++; in Java they are always "static" (arithmetic cannot be done)
- C # vs java = operator overload in C #, better exception handling in c # and more can be found in why is c # the best.eu aka MSDN
- C ++ has pointers, c # has a delegate, java dries :(

What more do you want? Talk about PPE and then mention the differences here. Ez.

d) Java technology, .NET technology.

Java technology

How Java works:

- Java is platform independent (OS) thanks to JVM. Byte-code itself is platform independent, but there is a separate JVM for each platform.
- Java compiler converts source code to Java byte-code, which is used for JVM.
- Java Runtime Environment (JRE), supplementing the JVM with a just-in-time (JIT) compiler that converts bytecode to native machine code.
- The JRE also includes Class Libraries - due to platform independence, it cannot rely on OS libraries, so it has its own on everything.
- All of this is included in the JDK - Java Development Kit
- Individual technologies / platforms:
- Java ME (Micro Edition)
- Java SE (Standard edition)
- Java EE (Enterprise edition)
- Java FX

Their differences:

- They differ mainly in libraries and different packages (SE - defines basic types and objects of the Java programming language to high-level classes that are used for networking, security, database access, graphical user interface (GUI) development, and XML parsing) and their use.
- Java ME is used for applications on devices with limited resources (memory, display) such as mobile phones, smart TVs, game consoles.
- Java EE extends SE with functionality for larger IS and other enterprise applications based on multi-tier architecture.

.NET technology

- A set of technologies for the web, Windows and mobile devices
- The standardized kernel specification is called the Common Language Infrastructure (CLI)
- As with Java, the platform can be divided according to focus on
    - .NET Micro Framework - for embedded devices
    - .NET Compact Framework - for mobile devices
    - .NET Framework - for desktop PCs
- .NET supports multiple languages, all languages are translated into universal CLI code by the compiler
- The most common languages are C #, Visual Basic .NET, Object Pascal; we can also meet F #, J #, C ++, ...

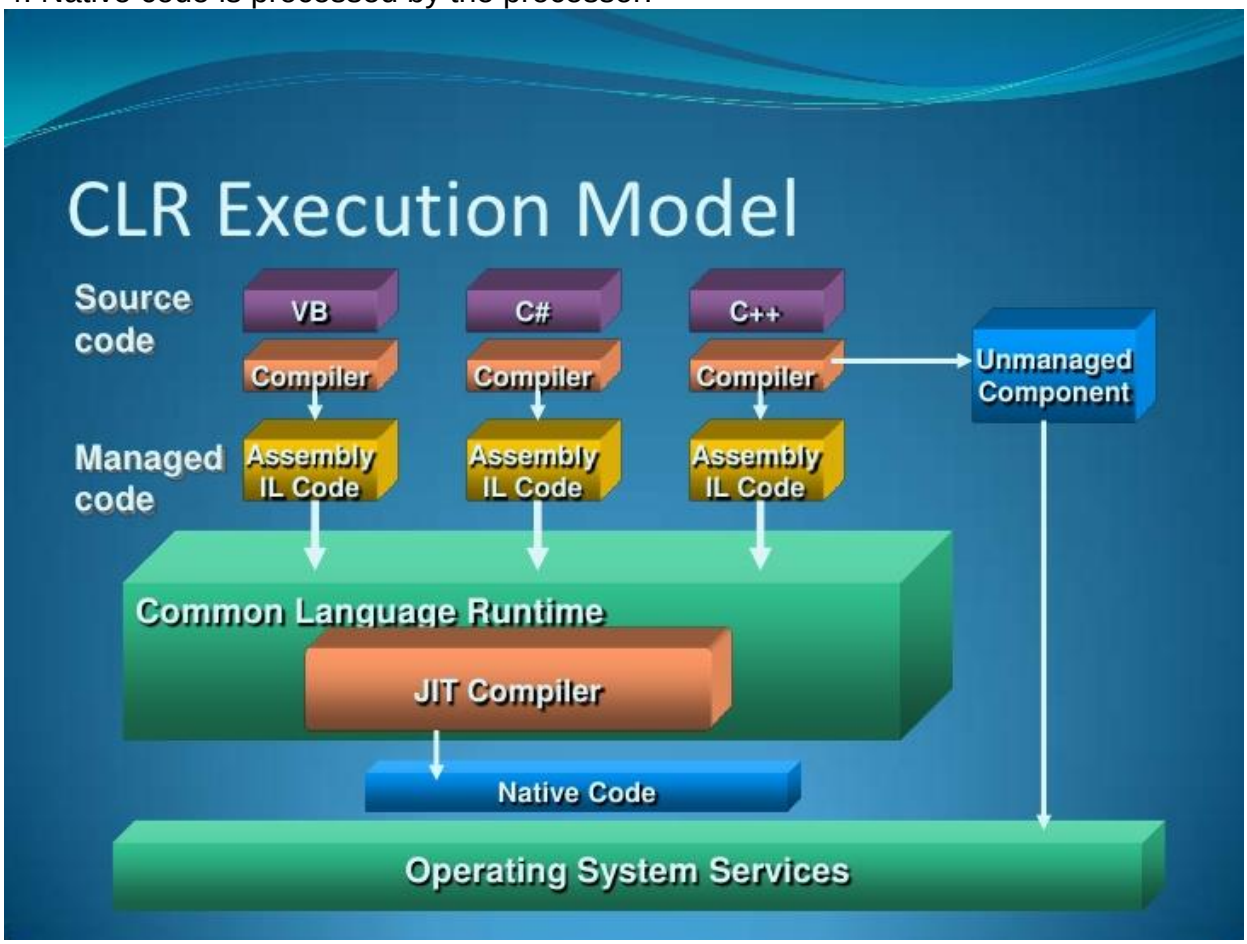
Components

- ASP.NET - Web application development technology
- WCF - Windows Communication Foundation - technologies for web services and communication infrastructure (API) development
- WPF - Windows Presentation Foundation - technology for GUI
- LINQ - provides object access to data in DB, XML and IEnumerable objects

Basic concepts

- CLI - defines the basic platform for the .NET architecture
- CLR - Common Language Runtime - runtime for .NET
- CLS - Common Language Specification - a common language for .NET
- CIL - Common Intermediate Language, formerly MSIL
- Assembly - a piece of Intermediate code that is designed to run the CLR
- JIT - Just In Time compilation
- GC - Garbage Collector - clears memory
- CTS - Common Type System - common data units for all languages

Compilation process -> run
1. The source code is converted to CIL.
2. The CIL is then converted to bytecode and a .NET assembly is created.
3. After performing the .NET assembly, its bytecode passes through the operational JIT compiler to generate native code.
4. Native code is processed by the processor.



e) Scripting languages.

Typical features

- Interpreted language (uncompiled source code + interpreter)
- No need to declare variables
- Dynamic type check = automatic conversion between strings, logical values and numbers
- Advanced data types, error recovery that does not result in script termination, and more ...

A program written in a scripting language is called a **script**. Typically, a script benefits from the fact that it does not have to be compiled, and often forms an extensible (parametric) part of a software project that can change without the need to recompile the main executable each time. Scripts can therefore be found in games, more complex software solutions or as a major part of dynamic websites and the like. Some programming languages have the ability to compile into machine code as well as interpretation; others also allow an intermediate stage, precompiling into an internal language, the execution of which is faster than classical interpretation.
Typical shortcuts = PHP, Perl, Python, Ruby, JavaScript, Lua
Some languages can be interpreted and compiled - eg Python. Some languages may contain an intermediate stage (precompilation) - Chrome's V8 engine precompiles Javascript files.

Benefits

- No need to compile -> quick change, the ability to serve as an addition to the main compiled program

- Easier maintenance, development and administration, (testing)
- Possible interpretation of code from a string at runtime
- Avoidance

Disadvantages

- No need to compile -> necessarily slower than precompiled program
- Untype -> greater code complexity when verifying the correctness of variables, greater error rate
- Higher memory requirements and poorer memory access (greater integrity limitations)

4. **Introduction to software engineering**

a) Software process. Definition of software process, software process models, software process maturity.

Processing by Žoltá: http://lucie.zolta.cz/index.php/statnice-vsb/8-skola-vsb/softwarove-inzenyrstvi/34-softwarovy-proces

Process definition and division

"A software process is a set of steps designed to create or modify a software work."

- This is a set of procedures that we must perform to create a software product
- According to this process, we then define projects related to individual orders (so-called process instances)
- The project can be divided into implementation, development and management. This is a technical and non-technical part, which follows its methodology.
- The project is thus divided into the methodology of software system implementation, development methodology and project management methodology
- The individual steps / activities in the process can run in parallel, so they need to be coordinated in some way
- At the same time, it must be ensured that they can be repeated, because the main goal is to achieve results with consistent quality
- Activities are provided by people (with the given skills, knowledge and technical resources needed for implementation)

The level of maturity of the software product vendor

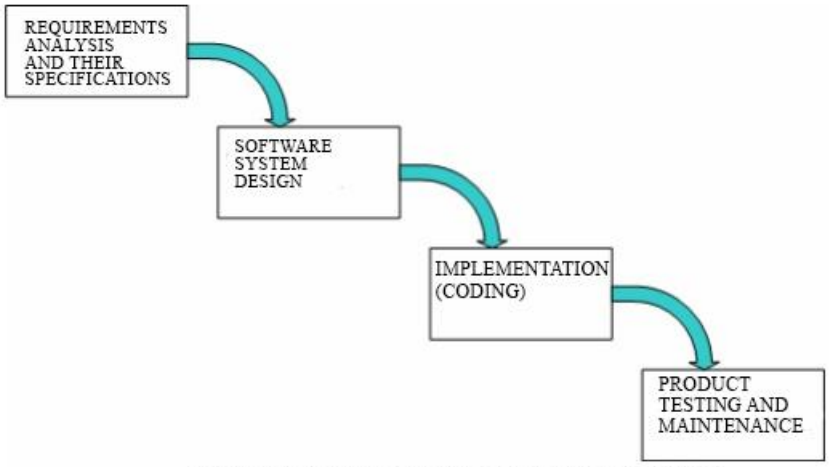
The level is set on the SEI (Software Engineering Institute) scale. The evaluation model has 5 levels and we call it CMM (Capability Maturity Model)

1. Initial - the company does not have a defined software process and each project is solved on a case-by-case basis (ad hoc)
2. Repeatable - the company has identified repeatable procedures in individual projects and is able to reproduce these in each new project
3. Defined - the software process is defined (and documented) based on the integration of previously identified repeatable steps
4. Managed - based on a defined software process, the company is able to manage and monitor it
5. Optimized - feedback information obtained by a long-term process of monitoring the software process is used in favor of its optimization

Software process models

To date, there is no detailed and precisely defined reference software process. However, it can be said that the basis of almost all became the so-called waterfall model, which can be found in various modifications and extensions in most current approaches.

Waterfall model



GIANT. 2.1: WATERFALL MODEL OF SOFTWARE PROCESS

- Divides the SW process into 4 basic phases:
- requirements analysis and their specification, software system design, implementation (coding), testing and maintenance of the created product
- The principle of a waterfall is that the next set of activities associated with a given phase cannot begin before the previous one ends. In other words, the results of the previous phase "flow" as inputs to the next phase

Disadvantages:

- The delay between entering a project and creating an executable system is too long
- The result depends on the complete and correct entry of the requirements for the final product
- The final product quality given by meeting all requirements cannot be revealed until the final software system is ready

Incremental model

- As the name suggests, its most typical feature is the increment of system versions, which then include an ever-widening range of functions defined gradually during its creation.
- Basically, these are a number of smaller waterfalls with a significantly shorter life cycle, each of which meets a new set of additional requirements

Spiral model

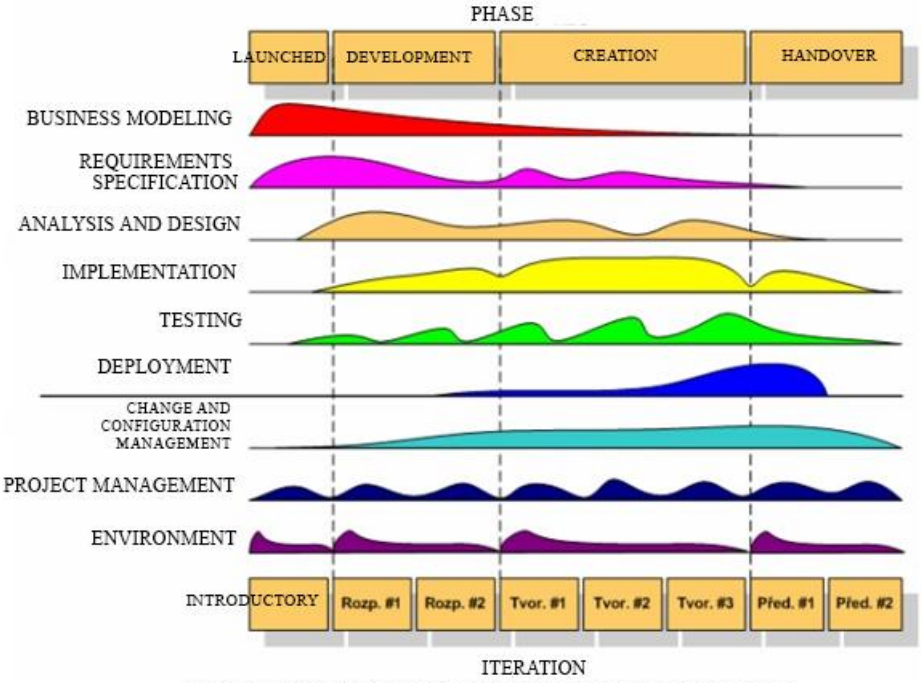
- Includes additional phases in its life cycle, such as the creation and evaluation of a prototype verifying the functionality of the target system, with each cycle packing additional requirements specified by the client

RUP (Rational Unified Process)

- Defined by a group of large companies coordinated by Rational
- It is a disciplined approach to assigning tasks and responsibilities within a development organization.
- It differs from the waterfall model in that it must adhere to the following principles:
- the software product is developed in an iterative way (at the end of each iteration there is an executable code, which is verified with the client and we process the comments in the next iteration)
- requirements placed on it are managed (RUP describes how to attract requests from contracting authorities and how to organize them, monitor their changes and document them)
- existing software components are used (components are either interconnected ad-hoc (on a case-by-case basis) or through a component architecture - Common Object Request Broker Architecture (COBRA), Component Object Model (COM), or some architecture using the Internet))
- the model of the software system is visualized (using UML - the purpose is to

hide details and create code using a language based on the graphical representation of the building blocks of the application)

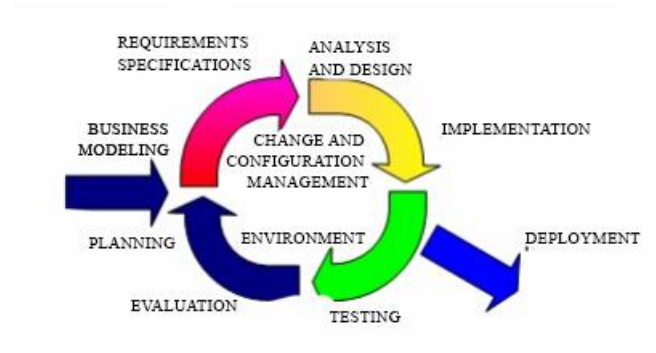
- product quality is verified on an ongoing basis (it is included in all activities, it applies to all participants in software solution development)
- system changes are managed (Change management allows you to ensure that every change is acceptable and all system changes are traceable)



GIANT. 2.3: SCHEMATIC REPRESENTATION OF THE RUP PROCESS

1. Initiation, where the original idea is developed into the vision of the end product and the framework of how the whole system will be developed and implemented is defined
2. Development is the phase devoted to the detailed specification of requirements and the development of the architecture of the final product
3. The work is focused on the complete execution of the required work. The resulting software is created around the designed framework (architecture) of the software system
4. Delivery is the final stage when the created product is handed over for use. This phase also includes other activities such as beta testing, training, etc.

Each phase can be further broken down into several iterations:



Static process structure

The purpose of the process is to specify who appears in it, what to create, how to create it and when to create it.

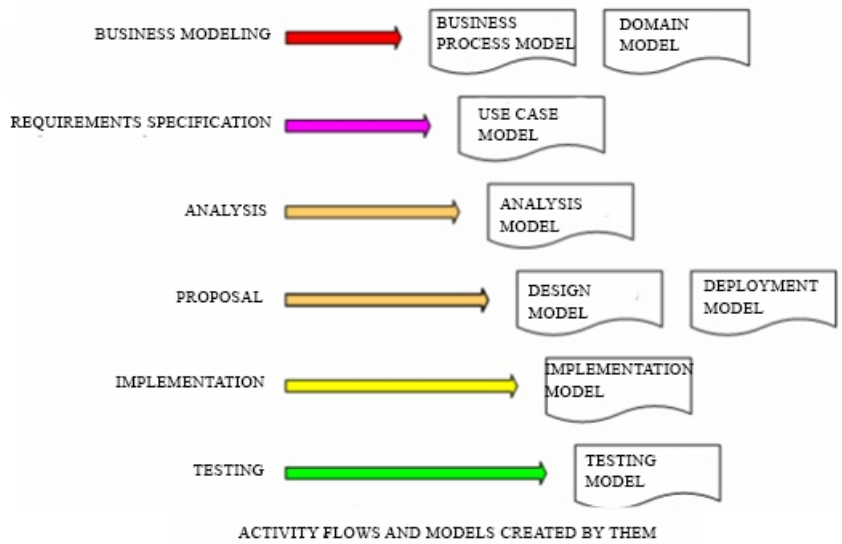
- Roles (staff) defining the behavior, competencies and responsibilities of individuals (analyst, programmer, project manager, etc.) or groups of people working in teams
- Artifacts representing entities that are created, modified or exploited in the

process (models, documentation, source codes, etc.). The model is a basic artifact (it is a simplification of reality, in order to better understand the developed system)
- Activities performed by staff to create or modify artifacts (source code compilation, design creation, etc.)
- Flows (workflow) of activities representing sequences of activities creating the required products (business modeling, requirements specification, etc.)

Process flows

In the SW process, there are basic flows, the result of which is an artifact (part of the final product) / model, "looking at the created system from a given point of view of abstraction (simplification)" and then supporting flows that do not create anything but are needed to implement the base year. The basic flows are:

- Business modeling describing the structure and dynamics of a company or organization
- Requirements specification defining its functionality by specifying the so-called use cases of the software system
- Analysis and design focused on software product architecture specifications
- Implementation representing custom software development, component testing and integration
- Testing focused on activities related to verifying the correctness of the software solution in all its complexity
- Deployment dealing with the configuration of the final product on the target computer infrastructure



ACTIVITY FLOWS AND MODELS CREATED BY THEM

The most important support flows are:

- Change and configuration management dealing with the issue of managing individual versions of created artifacts reflecting the development of changes in the requirements imposed on the software system
- Project management, including the issue of staff coordination, ensuring and adhering to the budget, planning activities and control of achieved results. An integral part is the so-called risk management, ie the identification of problematic situations and their solutions
- The environment and its management is the flow of activities that provide the development organization with the methodology, tools, and infrastructure that support the development team

b) Requirements engineering discipline. UML diagrams used in RE phase.

inzenyrstvi/35-specifikace-pozadavku

Definition

The goal of the requirements specification is to describe what the software system should do by specifying its functionality. Requirement specification models are used to agree the assignment between the development team and the client

Actors and functional specifications through use cases

The models that are created within this activity are based on use case use cases. These consist of:

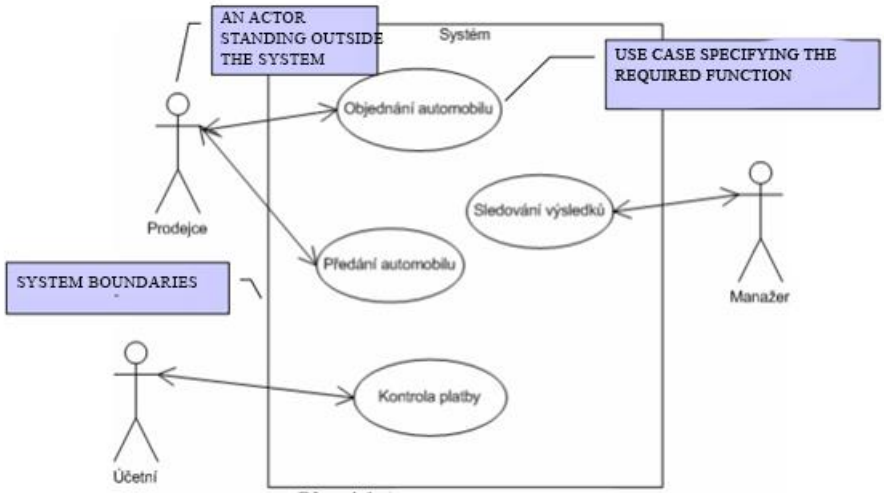
- Actors - defining users or other systems that will interact with the developed software system
- Use cases - specifying patterns of behavior implemented by the software system. Each use case can be understood as a sequence of consecutive transactions performed in a dialogue between the actor and the software system itself.

Use Case diagrams and sequence diagrams are used to visualize requirements.

- Use Case - describes the relationships between actors and individual use cases
- Sequence diagrams - show the interaction of objects organized by time

Use Case diagram

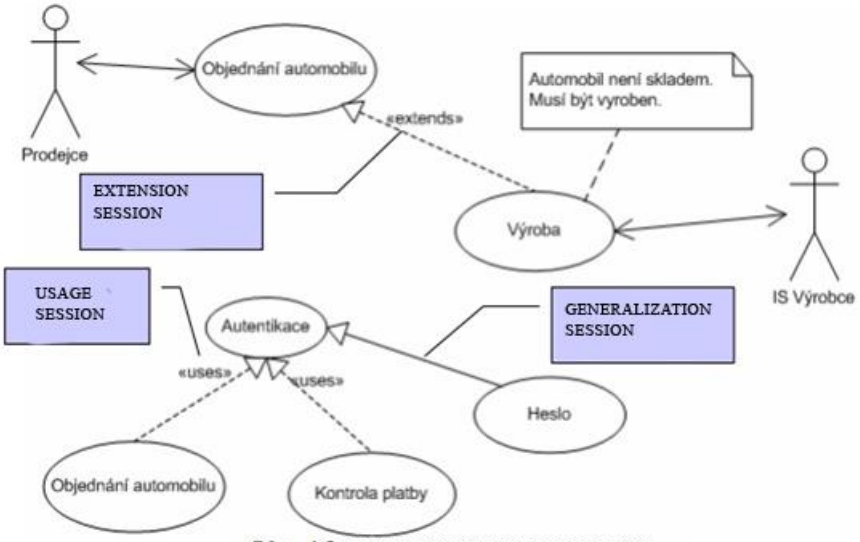
Their purpose is to show who will operate the system (actors) and what functionality (use cases) the system should have. The input for compiling the diagram is a business model (specifically business process models, the result of their analysis is a list of required functions).



Obr. 4.1: USE CASE DIAGRAM

You still have to add sessions to a UML like this. There are a total of 3: (see second picture)

- Session uses - << \ uses >>, expresses the situation when one quarrel is used by other use cases.
- The session extends - << \ extends >>, expresses the situation when one scenario extends another or "represents variant passages through the scenario described by it"
- Generalization / specialization / generalization relation - expresses the relationship between the general scenario and its special case. At the same time it expresses the inheritance of the actors (User is further divided into sellers, accountants, etc.)

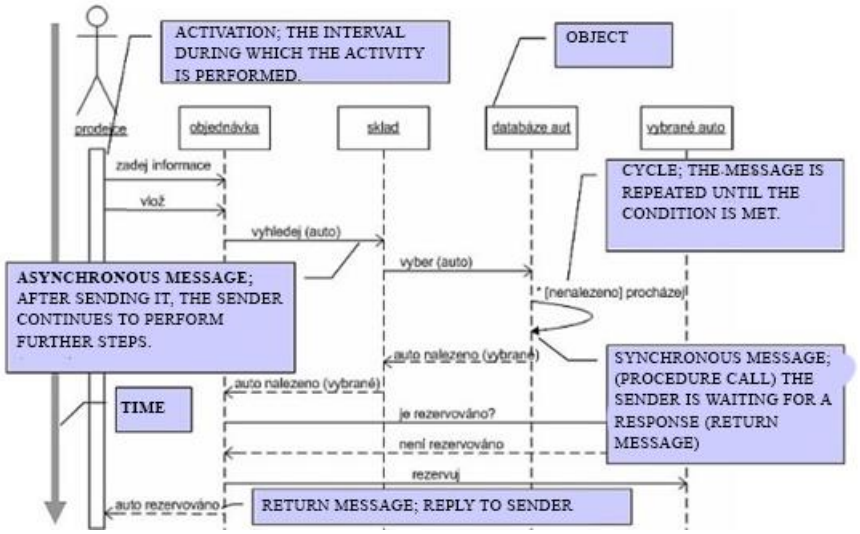


Obr. 4.2: STRUCTURE OF USE CASE

Sequence diagram

The sequence diagram describes the functions of the system from the point of view of objects and their iterations. Communication between objects is shown in time, so we can read the life cycle of the object from the diagram.

Sekvenční (Iterační) diagram popisuje funkce systému z pohledu objektů a jejich iterací. Komunikace mezi objekty je znázorněná v čase, takže z diagramu můžeme vyčíst i životní cyklus objektu.



c) Definition of a discipline “Design”. UML diagrams used in this discipline. Design pattern – classification, description and examples.

Proposal

Implementation design is the second phase of software development. It is an abstraction of source code that will serve as the main document to programmers in the next implementation phase.

The following UML diagrams are applied in the implementation design phase:

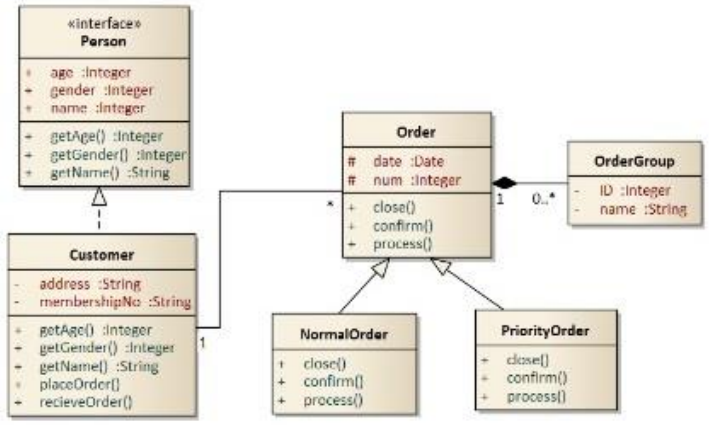
- Class diagram - a static view of the system showing the classes and the relationships between them.
- Sequence diagram - describes iterations between objects in time (see here).
- Collaboration diagram - focuses similarly to sequential iterations between objects, but without time sequence.
- The state diagram describes the life cycle of the object and the states in which it can be found.

- Deployment diagram - describes the system configuration (HW and SW deployment)

Návrh implementace je druhá fáze vývoje softwaru. Jde o abstrakci zdrojového kódu, která bude sloužit jako hlavní dokument programátorům v další implementační fázi.

V fázi návrh implementace se uplatňují tyto UML diagramy:

- Diagram tříd - statický pohled na systém zobrazuje třídy a vztahy mezi nimi.
- **Sekvenční diagram** - popisuje iterace mezi objekty v čase (viz. zde).
- **Diagram spolupráce** - zaměřuje se podobně jako sekvenční na iterace mezi objekty, ale bez časové posloupnosti.
- **Stavový diagram** - popisuje životní cyklus objektu a stavy ve kterých se může nacházet.
- **Diagram nasazení** - popisuje konfiguraci systému (rozmístění HW a SW).



Třídní diagram

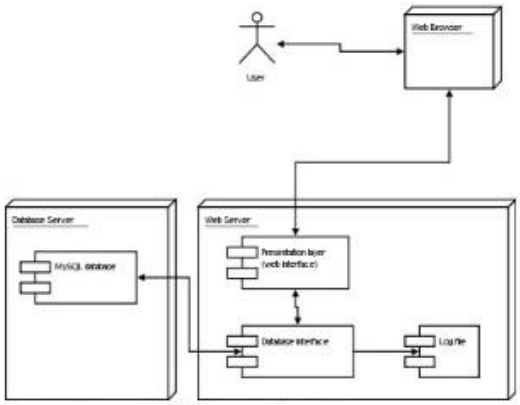


Diagram nasazení ukázka webového servru

Design patterns

Design patterns are methodologies (templates) for solving various problems that a developer may encounter. Object-oriented design patterns typically show the relationships and interactions between classes and objects without specifying the implementation of a particular class.
Design patterns are divided into the following 3 basic groups:

- Creational Patterns - solves problems related to creating objects in the system. The aim of these design patterns is to describe the procedure for selecting the class of a new object and ensuring the correct number of these objects. These are usually dynamic decisions made while the program is running.
- Structural Patterns - represent a group of design patterns focusing on the possibilities of arranging individual classes or components in a system. The aim is to make the system clearer and use the possibilities of code structuring.
- Behavioral Patterns - They are interested in the behavior of the system. They can be based on classes or objects. For classes, they use the principle of inheritance when designing solutions. The second approach addresses the cooperation between objects and groups of objects, which ensures the achievement of the desired result.

Each of these groups contains many design patterns, which we will now elaborate on:

Creational patterns

- Abstract Factory - Defines an interface for creating families of objects that are interdependent or related without specifying a specific class. The client is

shielded from creating specific object instances.

- Factory Method - Defines an object creation interface that lets children decide which object will actually be created. * Factory method lets classes pass creation to children.
- Builder - Separates the creation of a complex of objects from their representation so that the same creation process can be used for other representations.
- Singleton - If you need the class to have a maximum of one instance.
- Prototype (Clone) - Specifies the type of objects to be created using the prototype object. New objects are created by copying this prototype object.
- Lazy Initialization - Delays creating an object, counting a value, or performing a process until it is first needed.
- Object pool - Allows you to avoid the costly creation and release of resources by recycling objects that are no longer used.
- Multiton

Structural patterns

- Adapter - If you need two classes that do not have a compatible interface to work together. The adapter converts an interface of one class to an interface of another class.
- Bridge - Separates the abstraction from the implementation so that the two can be arbitrarily different.
- Composite - Composes objects into a tree structure and allows the client to work with both individual and composite objects in the same way.
- Decorator - We will use it if we have some objects to which we need to add additional functions at runtime. The new object retains the original interface.
- Facade - Provides a single interface to the interface set in the subsystem. Defines a higher-level interface that simplifies the use of the subsystem.
- Flyweight - Suitable for use if you have too many small objects that are very similar.
- Proxy - Offers a replacement or placeholder for another to control access to that object.

Behavioral patterns

- Observer - When there are many other objects dependent on one object, this pattern will give you a way to let everyone know when something changes.
- Command - Encapsulate the request as an object, allowing you to parameterize clients with different requests, queues, or log requests, and support take-back operations.
- Interpreter - Creates a language, which means defining grammar rules and determining how to interpret the resulting language.
- State - Allows an object to change its behavior if its internal state changes. An object looks like it has become an instance of another class.
- Strategy - Encapsulates some type of algorithms or objects to change so that they are interchangeable for the client.
- Chain of responsibility - Addresses how to send a request without specifying the object that will process it.
- Visitor - Represents an operation that should be performed on object structure elements. The visitor allows you to define new operations without changing the classes of elements he is working on.
- Iterator - Provides a way to access the elements of a group object sequentially without exposing the object's internal representation.
- Mediator - Allows you to communicate between two components of a program without having to interact directly and thus have to know the exact methods provided.

- Memento - Capture and store an object's internal state in an external object without breaking the encapsulation so that the object can be returned to that state at any time later.
- Template method Defines the skeleton of how an algorithm works, leaving some steps to the children. This allows descendants to modify certain steps of the algorithm without being able to change the structure of the algorithm.

d) Object oriented paradigm. Concept class, object, interface. Basic features of object and relation with class. Basic relations among classes and interfaces. Class vs. instance features.

Object-oriented paradigm.
Object-Oriented Programming (0OP) is a software development methodology. The OOP paradigm describes how to develop and write a program and how to think about a problem.

Basic paradigm of PPE
- When solving the problem, we create a model of the described reality, we describe the entities and the interaction between the entities
- We abstract from irrelevant details - when describing / modeling an entity, we omit irrelevant properties of entities
- The solution procedure is in many cases more efficient than in the procedural approach (not always), where the tasks are solved as a sequence of commands

PPE objectives
- It is driven by the desire for component reusability.
- It decomposes a complex task into subcomponents that can be solved independently if possible.
- Bringing the structure of a computer solution to the real world (communicating objects).
- Hide solution implementation details from the user.

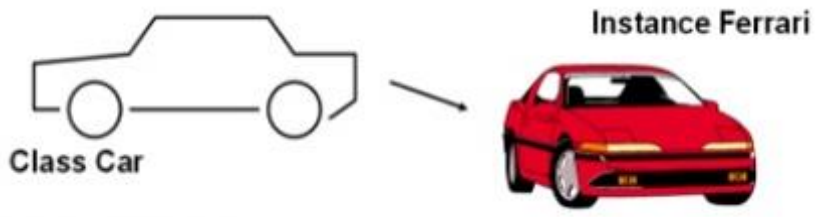
OOP! = Classes and objects - is an approach to how to properly design a program structure

OOP model

OOP defines a program as a set of cooperating components (objects) with a precisely defined behavior and state. PPE methods mimic the appearance and behavior of a real-world object with the possibility of great abstraction. The PPE model consists of several of the following designs:

Object
The individual elements of modeled reality (both data and related functionality) are grouped into entities in the program, called objects. Objects remember their state and externally provide operations that are accessible as calling methods. The object is also an instance of tidy.

Class
The class serves as a template for instantiating classes - objects. It groups objects of the same type and captures their nature at a general level. (So a safe class is not its own information, it's just a template; data contains only objects.) The class defines the data and methods of the objects.



Interface
The interface prescribes the class that will be derived from it, what methods or properties it must implement. The derived object can also implement other methods. The interface does not define any variables - only constants. nor does it contain implemented methods - it contains only abstract methods without their implementation (their headers). Tida can implement any number of interfaces (unlike inheritance). If you create an interface that inherits from another interface, it automatically inherits all its methods and constants.

Abstract class
It's such a hybrid between the interface and the classic class. Since the classical class, it has the ability to implement properties (variables) and methods that will be performed the same on all derived classes. In turn, it gained the ability to contain empty abstract methods that each derived subclass must implement itself. With these advantages, the abstract class also has a few limitations, namely that one subclass cannot inherit more than one abstract class and takes over from the interface the limitation that it cannot create a separate instance (by the new operator).
1. Abstract class
- may contain specific methods as well as methods without implementation (abstract)
- Is expand only one class

2. Interface
all methods are without implementation
implement multiple interfaces in one Is class

Features of PPE
- Abstraction - the programmer, and therefore the program he creates, can abstract from some details of the work of individual objects. Each object works like a black box that can perform specific activities and communicate with the environment without requiring knowledge of the way it works internally.
- Encapsulation - guarantees that the object cannot directly access the "insides" of other objects, which could lead to inconsistencies. Each object externally exposes the interface through which (and in no way otherwise) the object is manipulated.
- Composing - An object can contain other objects.
- Delegation - An object can use the services of other objects by requesting them to perform an operation.
- Inheritance - objects are organized in a tree way, where objects of some kind can inherit from another type of objects, thus taking over their abilities, to which they only add their own extensions. This idea is usually implemented by dividing objects into classes, with each object being an instance of a class. Each class can then inherit from another class (in some programming languages from several other classes).

- Polymorphism - the referenced object behaves according to the class of the instance. If several objects provide the same interface, they are handled the same way, but their specific behavior varies by implementation. In the case of an inherited polymorphism, this means that in the place where an instance of a class is expected, we can substitute an instance of any of its subclasses. For an unconditional inherited polymorphism, it is sufficient that if the interfaces (or their desired parts) of the different classes match, then they are polymorphic to each other. Polymorphism is often explained in the picture with animals, all of which have the Speak method in their interface), but each performs it in its own way.

- Genericity - is the possibility of the programming language to define instead of types only "type patterns", where the types of variables used in the definition (ADT types) are output outside the definition as parameters and are determined later by the client application.

Example:
List <T> .where T is any type. When we allocate a Sheet, we use it
List <int> intList = new List <int> (), and this list then only accepts int numbers.
The spell of genericity then stands out in combination with inheritance, when not only objects of type T can be added to the list, but also objects of all possible heirs of class (type T).

Class vs. instance properties

Installation variables

- specify the properties of objects
- access from the class or using the get () and set () methods
- arise with the object

Static variables

- do not describe object properties
- exist only once per class
- can exist without an object
- access from the class or by class name

e) Mapping of UML diagrams to source code.

Mapping UML to code

https://www.milosnemec.cz/clanek.php?id=199

Good explanation here too

The mapping of UML diagrams to code occurs in the first part of the implementation phase of development. It is an automatic generation of source code from a UML diagram.
For example, it generates individual interfaces, classes with variables, and empty methods from a class diagram. A very nice tutorial on how to map a class diagram to code can be found here.
It then reads from the component diagrams and adds the required components to the generated code. A component is an existing and replaceable part of a system with a given interface through which the system uses it. It is a component written in source code, or an already compiled component in binary code, or other components

represented by database tables, documents, etc.
The task of the implementation itself is then to complete the bodies of methods, the behavior of which can be described in the activity diagram.

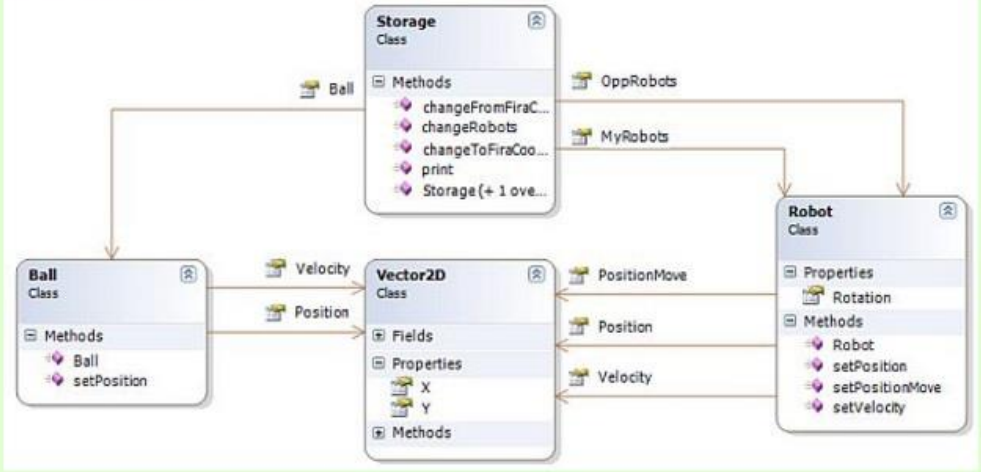
K mapování UML diagramů na kód dochází v první části implementační fáze vývoje. Jde o automatické vygenerování zdrojového kódu z UML diagramu.

Z ***diagramu tříd*** například vygeneruje jednotlivé rozhraní, třídy s proměnnými a prázdnými metodami. Velmi hezký tutoriál o tom jak se mapuje class diagram na kód najdete zde.

Z ***diagramů komponent*** pak vyčte a do generovaného kódu přidá požadované komponenty. **Komponenta** je už existující a vyměnitelná část systému s daným rozhraním skrze které jí systém používá. Jde o komponentu napsanou v zdrojovém kódu, nebo už zkompilovanou komponentu v binárním kódu, či další komponenty reprezentovanými databázovými tabulkami, dokumenty, apod.

Úkolem samotné implementace je pak dopsat těla metod, jejichž chování může být pospáno v ***diagramu aktivit***.

Ukázka kódu z diagramu tříd




```
public class Storage
{
        public Ball Ball { get; set; }
        public Robot[] MyRobots { get; set; }
        public Robot[] OppRobots { get; set; }

        public Storage( ){}
}

public class Ball
{
        private Vector2D Position { get; set; }
        private Vector2D Velocity { get; set; }

        public Ball(){}
        public void setPosition(double x, double y){}
}

public class Robot
{

        private Vector2D Position { get; set; }
        private Vector2D Velocity { get; set; }
        private Vector2D PositionMove {  get; set; }
        private double Rotation { get; set; }

        public Robot(){}
        public void setPosition(double x, double y){}
        public void setPositionMove(double x, double y){}
        public void setVelocity(double left, double right){}
}

public class Vector2D
{
        private double x { get; set; }
        private double y { get; set; }

...
```

f) Memory management (in languages C/C++, Java, C#, Python), virtual machine.

Memory management

Memory management is a set of methods in computer science that the operating

system uses to allocate operating memory to individual running processes.
Memory management is fully automated in both Java and C #. Memory is freed by a separate thread that runs at low priority and provides continuous monitoring of unused blocks of memory. Memory is allocated by the new operator.

Memory management levels

- Technical equipment
  - registry, cache
- Operating system
  - virtual memory
  - segmentation, paging
- Applications
  - Memory allocation
  - Memory regeneration
    - Manual - delete, dispose, free (), ...
    - Automatic - garbage collection

Garbage collector (GC) - is usually part of the runtime environment (programming language), or an additional library, supported by the compiler, hardware, operating system, or any combination of the three. Its task is to automatically determine which part of the program's memory is no longer in use and prepare it for further reuse.

Virtual machine

In computer science, a virtual machine is software that creates a virtualized environment between the computer platform and the operating system in which the end user can run the software on the abstract machine.

Hardware virtual machine

The original meaning for a virtual machine, sometimes called a hardware virtual machine, refers to several individual identical work environments on a single computer, each running an operating system. This allows an application written for one OS to be used on a machine running another OS, or to provide a sandbox that provides a higher level of isolation between processes than is achieved when executing several processes at once (muititasking) on the same OS.
One use may also provide the illusion to many users that they are using an entire computer, which is their "private machine, isolated from other users, even though they all use one physical machine." Another advantage may be that starting and booting the virtual machine can be much faster than with a physical machine, because tasks such as hardware initialization can be skipped.
Similar software is often referred to as virtualization and virtual servers. The host software that provides this capability is often referred to as a hypervisor or virtual machine monitor.

Application virtual machine

Another meaning of the virtual machine is computer software that isolates the applications used by the user on the computer. Because virtual machine versions are written for different computer platforms, any application written for a virtual machine can run on any of the platforms, instead of having to create separate versions of the application for each computer operating system. An application running on a computer uses an interpreter or Just in time compilation.
One of the best-known examples of a virtual machine application is the Java Virtual Machine from Sun Microsystem. The Java Virtual Machine can process the Java bytecode, which is usually created from the source code of the Java programming language. Due to the fact that the JVM is available on many platforms, it is possible to

create an application in Java only once and run it on any of the platforms for which the JVM is developed (eg Windows, Linux).

g) Support for parallel execution, threads.

Parallel execution

- Divide a task into subparts and run them at the same time
- As a result, we have a new programming danger that we must be careful about - for example, concurrency
- Concurrency means that the subparts access the variable at the same time and fight with each other.
- It can be solved, in several ways, most ideally programmed in such a way that there is no concurrence
- However, if it is absolutely necessary and concurrency will occur, we must use eg TSL (Test and Set Lock)

Threads

- Here the sub-parts (defined for parallel processing) are controlled by one thread
- "Self-executed computing that runs in a single process."
- Specifically in Java, two ways:
    - By implementing the Runnable interface.
    - Inheritance from the Thread class.
    - Both methods require the implementation of the Run () function - that is, what will happen in the thread.
    - The second method offers more flexibility in handling multiple threads.

Every thread in Java is an instance of the java.lang class. Thread. This class provides thread start, stop, and termination. The thread must implement a run method that defines the activity of the thread. This method is passed control after starting the thread by the start method.

Life cycle

During its life, a thread goes through a sequence of the following states:

- New - no system resources are allocated to the thread immediately after creation, the thread is not running.
- Runnable - after the start method, the thread is ready to run. There can be multiple threads in this state, but only one of them (on a single - processor computer) is in a running state ".
- Not runnable - the thread enters this state if it is paused by calling one of the sleep, wait or suspend methods, or by waiting for the input / output operation to complete.
- Dead - the thread enters this state by terminating the run method or calling the stop method.

Multithreaded applications and debugging

The main problems with multithreaded applications are related to synchronization

- Deadlock is a situation where the successful completion of the first action is conditional on the previous completion of the second action, while the second action can be completed only after the completion of the first action. We can prevent a deadlock, for example, by having the process apply for all the resources it needs at once. Either he gets them all or he doesn't get one.
- race conditions - access of multiple threads to shared variables and at least one thread does not use synchronization mechanisms. A thread reads a value while another thread writes. Writing and reading are not atomic and data may be invalid. Locks are used to prevent concurrency, which ensures that only one

thread can access a designated resource file or piece of code at a time.

- Starvation - Starvation - a condition in which resources are constantly denied to a thread. Without these resources, the program will never complete its task.

Example: The following program creates two parallel threads, which always print their name and then wait for the specified number of milliseconds.

```
using System;
using System.Threading;

public class Vlakno
{
        string jmeno;
        int interval;

        public Vlakno(string jmeno, int interval)
        {
                this.jmeno = jmeno;
                this.interval = interval;
        }

        public void Run()
        {
                while( true ) {
                        Console.WriteLine("Vlakno {0}", jmeno);
                        Thread.Sleep(interval);
                }
        }

        public static void Main()
        {
                Vlakno v1 = new Vlakno("v1", 500);
                Vlakno v2 = new Vlakno("v2", 1000);

                Thread t1 = new Thread(new ThreadStart(v1.Run));
                Thread t2 = new Thread(new ThreadStart(v2.Run));
                t1.Start();
                t2.Start();
                t1.Join();       // Čekat, dokud vlákno 't1' nedokončí práci
                t2.Join();
        }
}
```

h) Error handling in modern programing languages.

Error handling

Exceptions represent certain situations in which the calculation must be interrupted and the procedure handed over to the place where the exception will be treated and a decision will be made on how to proceed with the calculation. Older programming languages usually had no support for handling errors during the calculation. For all functions that may have encountered an error, it was still necessary to test the various symptoms and special return values that reported the error, and in case we forgot to test the error, the program continued to run and did not respond to the error at best. a completely different place where it was already difficult to trace the source of the error. Java, like C ++, uses a structured handler method to handle exceptions. This means that the programmer can specify for a specific section of the program how a particular type of exception should be handled. If an exception occurs, the nearest parent block in which the exception is handled is always found.

Exceptions are represented as objects in Java. These objects are instances of classes derived generally from the Throwable class with two subclasses, Error and Exception.

Each of these subclasses corresponds to one group of exceptions. The first group contains exceptions, the occurrence of which represents a serious problem in the operation of the application and which the programmer should not catch. The second group consists of exceptions, the treatment of which makes sense - for example, exceptions such as an attempt to open a non-existent file, incorrect number format, etc. This group should also include all exceptions defined by the user.

Try-catch statement

Exception handling always refers to a block of a program constrained by a try statement followed by one or more catch blocks describing how individual exceptions are handled.

Example if we want to handle errors when working with a file (in Java):

```
InputStream fs = null;
try {
   fs = new FileInputStream("data.txt");
} catch( FileNotFoundException e ) {
   System.err.println("Soubor data.txt nenalezen");
   System.exit(2);
}
```

Example of error handling for incorrect number parsing (for example, we parse a text string):

```
class TiskCisel {
   public static void main(String[] args) {
      try {
         int n = Integer.parseInt(args[0]);
         for(int i = 1; i <= n; i++)
            System.out.println(i);
      } catch( NumberFormatException e ) {
         System.err.println("Chyba: Nespravny format argumentu");
      }
   }
}
```

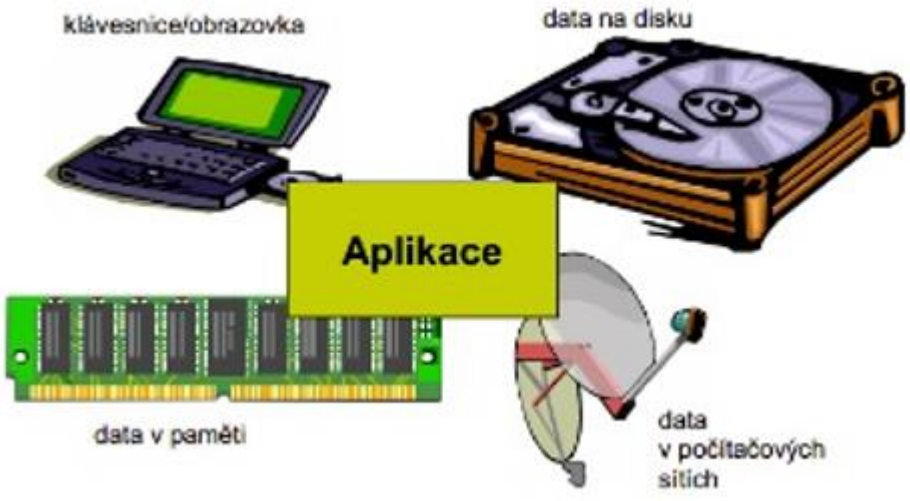
If we need to raise our own exception in the program, we must first create a suitable class derived from the Exception class or some of its subclasses.

```
class MojeVyjimka extends Exception {
   MojeVyjimka(String msg) { super(msg); }
}


class Vyjimky {
static void zpracuj(int x) throws MojeVyjimka
   {
     System.out.println("Volani zpracuj " + x);
     if( x < 0 )
        throw new MojeVyjimka("Parametr nesmi byt zaporny");
   }

   public static void main(String args[])
   {
      try {
         zpracuj(1);
         zpracuj(-1);
      } catch( MojeVyjimka e ) {
         e.printStackTrace();
      }
   }
}
```

i) Principles of data streams – for input/output operation. Differences between character and byte oriented data streams.

Principle of data streams

Data streams (hereinafter only streams) are sequences of data. The stream is defined by its input and output, which can be, for example, a disk file, a device (keyboard or mouse input, output display) or another program.



I / O device (I / 0):

- character: input - keyboard, output - monitor, printer.
- block: hard drive, flash drive, SD, microSD ...

From the keyboard we read sequentially character by character (sequential access), for a binary disk file we can freely access the selected part of the data - random access. Primitive streams only transmit binary data, specialized streams can manipulate the transmitted data. These handling capabilities vary depending on the type of data being transmitted.

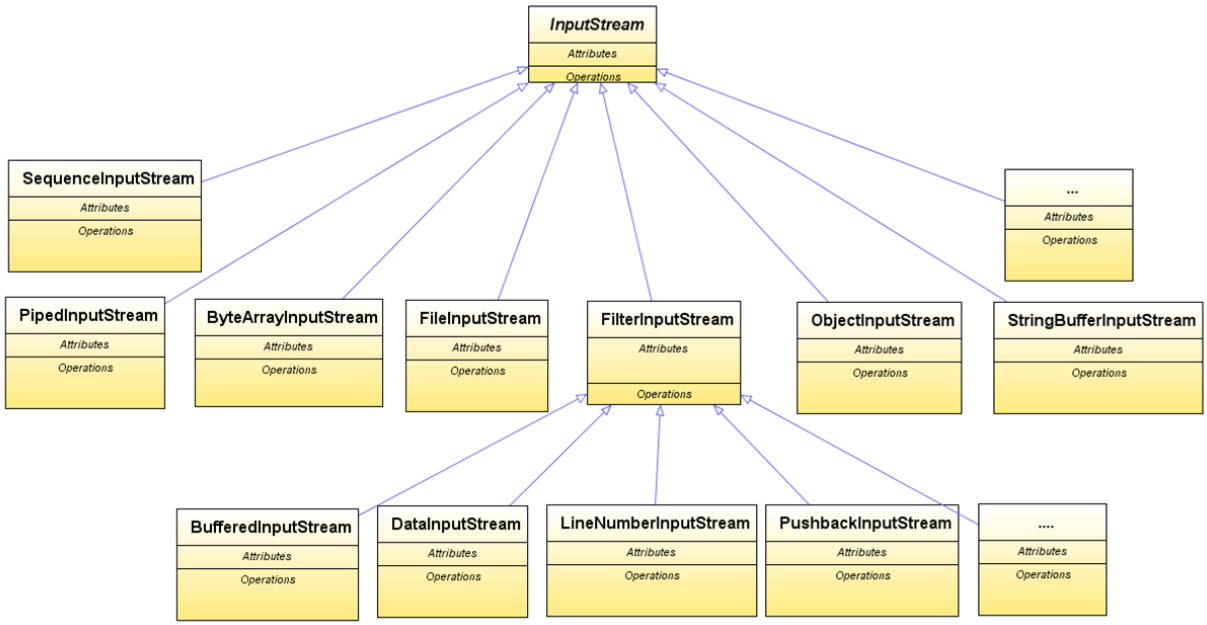
Types of transmitted information:

- Binary data - pictures, sound, objects, ... - data stored in the same form as they are stored on a computer.
- Text data - represent lines of text in ASCII (abc) or UNICODE (S)

Types of currents

All Streams inherit from InputStream, the key method of Read ();

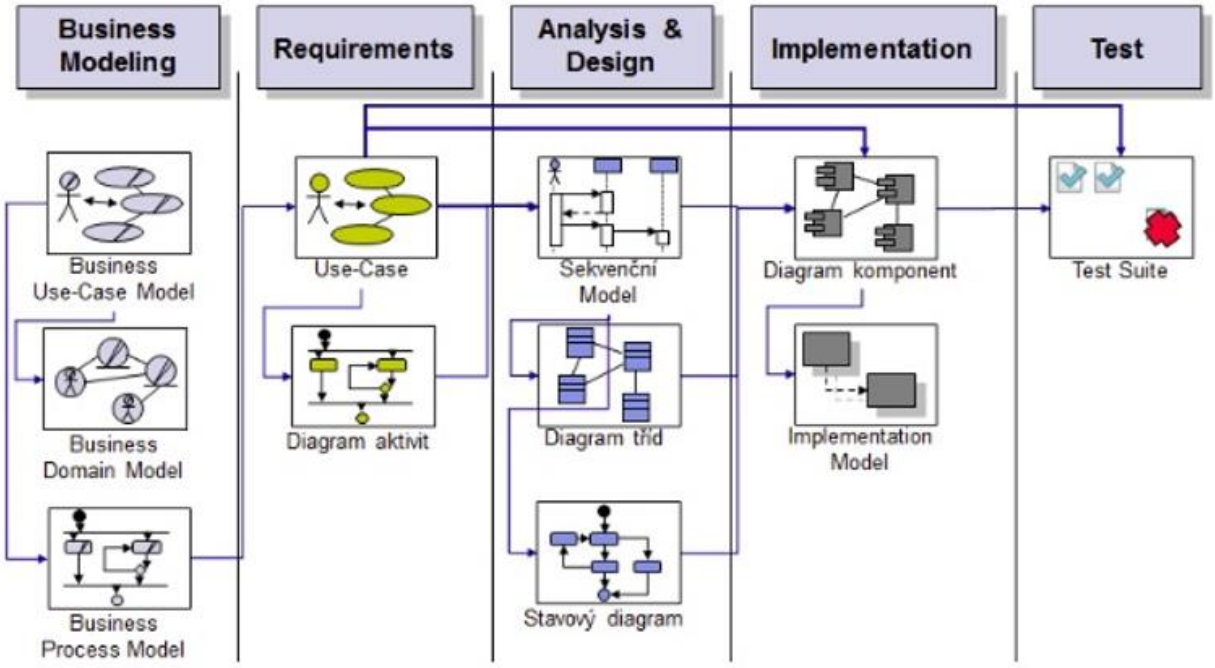
- Housing flows
  - FileInputStream - reads data byte by byte.
  - Most low-level. If we know that the file we are reading will need text and we want to work with the text, it is better to use more specified Streams
- Character streams
  - FileReader - reads char data by char (16 bits by default, but encoding can also be specified)
- Buffered streams
  - BufferedReader - only wraps other InputStreams
  - Does not read data by char / byte after each of (eg) disks, instead transfers them to the buffer and then reads
  - This makes it faster and more efficient



j) Unified modelling language (UML) – types of diagrams and its usage during software development cycle.

UML

UML is a unified diagramming language. Defines standards for a uniform diagram structure. With it, we can implement a variety of schemes across the software development process.
UML language enables specification (structure and model), visualization (graphs), construction (code generation eg from cdlass diagram) and documentation of software system artifacts.
Each of the UML diagrams is also used in a different phase of software development.



UML division

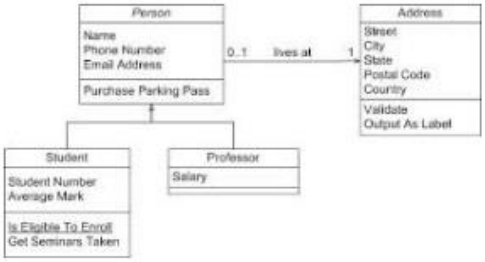
Affecting various aspects of the system:

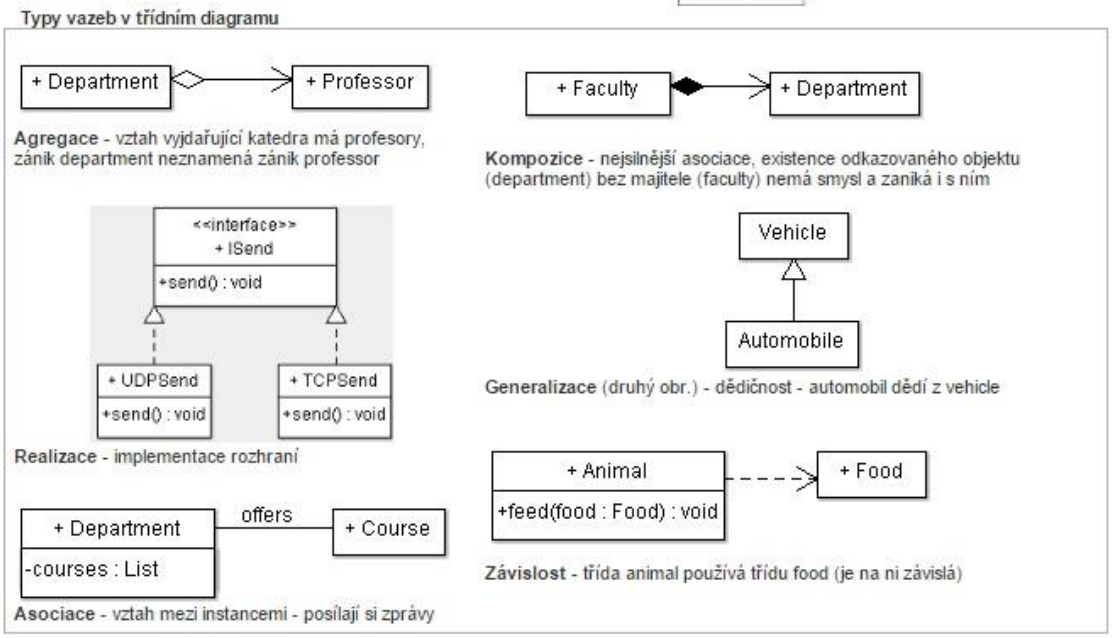
1) Functional diagram view
   - Use case
2) Logical preview
   - Class diagram
   - Object diagram
3) Dynamic preview describing behavior
   - Status diagram
   - Activity diagram
   - Interaction diagrams
   - Sequence diagrams
   - Cooperation diagrams
4) Implementation preview
   - Component diagram
   - Deployment diagram

Class diagram

Class diagram - the class structure diagram provides a logical view of the system. It shows data structures, operations on objects and also their bindings. The class diagram is used for ERD creation and implementation design.

- **Třídní diagram** (***Class diagram***) - diagram struktury tříd poskytuje logický náhled na systém. Znázorňuje datové struktury, operace u objektů a také jejich vazby. Třídní diagram se využívá pro tvorbu ERD a při návrhu implementace.





Object diagram

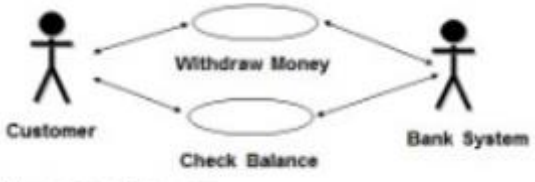
Object diagram (Object diagram) shows instances of classes - objects, sometimes called an instance diagram. It is a picture of objects and their relationships in the system at a certain point in time, which we want to emphasize for some reason. It is used in data analysis for ERD

- **Objektový diagram** (***Object diagram***) zobrazuje instance tříd - objekty, někdy nazýván jako ***instanční diagram***. Je snímkem objektů a jejich vztahů v systému v určitém časovém okamžiku, který chceme z nějakého důvodu zdůraznit. Využívá se v datové analýze pro ERD.

Objektový diagram

Use case diagram

Use-case diagram provides a functional view of the system (who works with the system and how). It is used for the implementation of DFD, in the development phase of requirements specification.

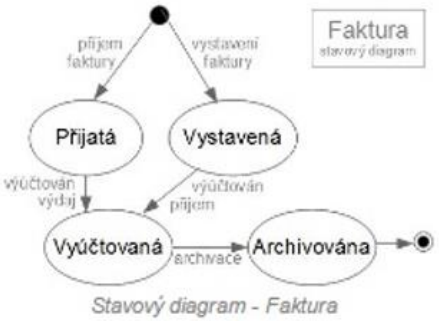


Use-case diagram

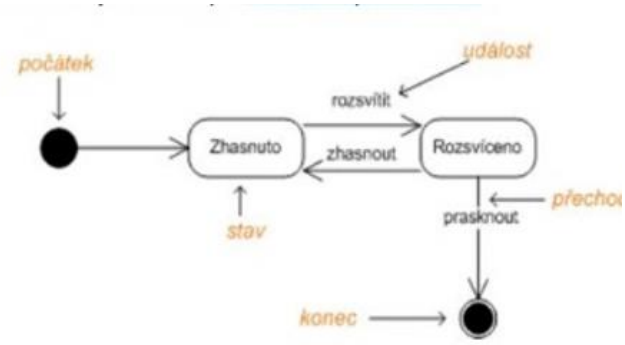
State diagram

State diagram shows the life cycle of objects and the states they enter during their

lifetime. It contains 3 basic elements status, event, transition. Used in implementation design.



Stavový diagram - Faktura



Stavový diagram - životní cyklus žárovky

Activity diagram

Activity diagram - graphical representation of algorithms. This diagram is used to model procedural logic, processes, and workflow capture. Each process in the activity diagram is represented by a sequence of individual steps that determine the control flow. This diagram also contains a decision symbol (diamond) with the condition that the flow of the process continues based on the response of one of the branches. The activity diagram is used to create mini-specifications and other detailed analyzes and to specify requirements.

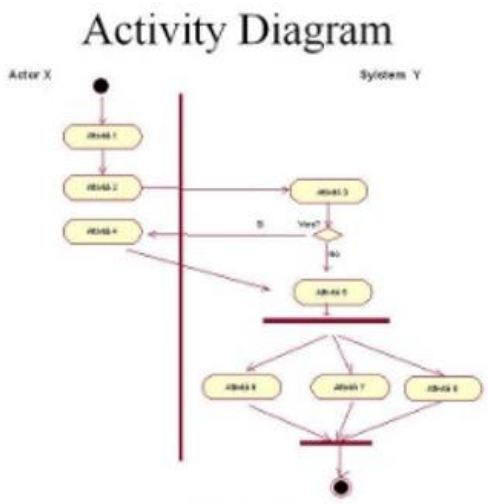


Diagram aktivit

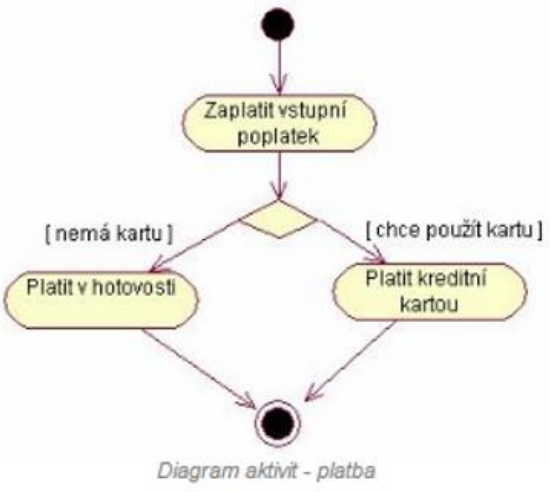


Diagram aktivit - platba

Sequence diagram

Sequence diagram describes the functions of the system from the point of view of objects and their iterations: Communication between objects is represented in time, so we can read the life cycle of the object from the diagram.

Sekvenční (Iterační) diagram popisuje funkce systému z pohledu objektů a jejich iterací. Komunikace mezi objekty je znázorněná v čase, takže z diagramu můžeme vyčíst i životní cyklus objektu.

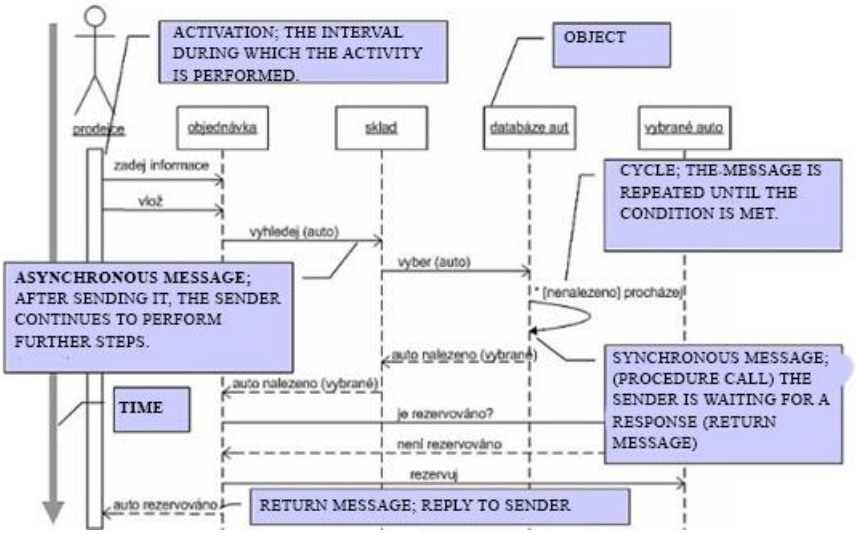


Diagram of cooperation

Colaboration diagram - a dynamic view of the system showing the behavior of objects in the system. When describing mutual communication between objects, the cooperation diagram prioritizes their topology, ie their distribution and mutual connection. Similar to sequential but without time. Use in the implementation phase.

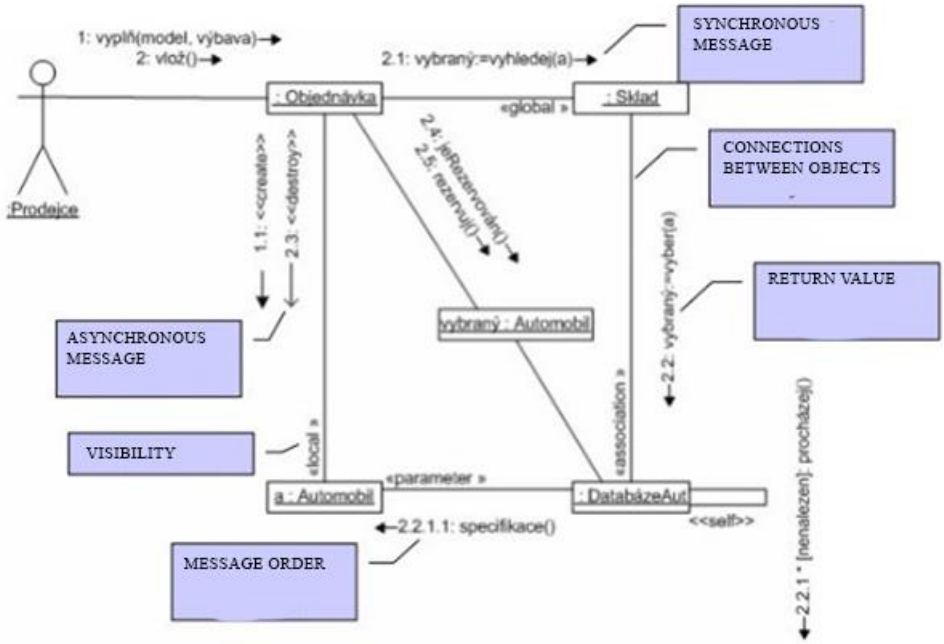


Diagram spolupráce - objednávka auta

The time sequence of sending messages is expressed by their serial number. The return value is expressed by the assignment operator: =. The retransmission of the message is indicated by the symbol * and in square brackets the conditions of the repetition of the cycle. In addition, this diagram introduces the following types of visibility of interconnected objects:

<local>> expresses a situation where an object is created in the body of the operation and is canceled after its execution;
<<global>> specifies a globally visible object;
<<parameter>> expresses the fact that the object is passed to the other as an argument to the message sent to it:
<<association>> specifies a permanent link between objects (sometimes also called familiar connections).

Component diagram

Component Diagram - Illustrates the organization and dependencies between software components. The source components are files created by the programming language used. The executable components diagram specifies all components created by us and those provided by the implementation environment.

- **Diagram komponent** - ilustruje organizaci a závislosti mezi softwarovými komponentami. Zdrojové komponenty tvoří soubory vytvořené použitým programovacím jazykem. Diagram spustitelných komponent specifikuje všechny komponenty vytvořené námi i ty, které nám dáva k dispozici implementační prostředí.

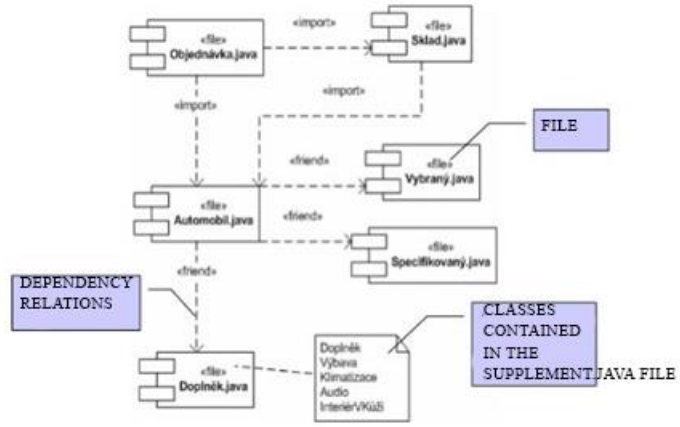


Diagram komponent - binární kód

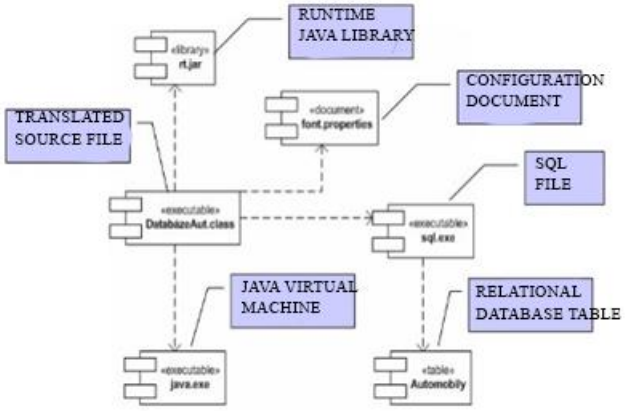


Diagram komponent - runtime prostředí

Deployment diagram

Deployment diagram - specified not only in terms of configuration of technical means, but especially in terms of deployment of implemented software components on individual technical means represented in the computer.

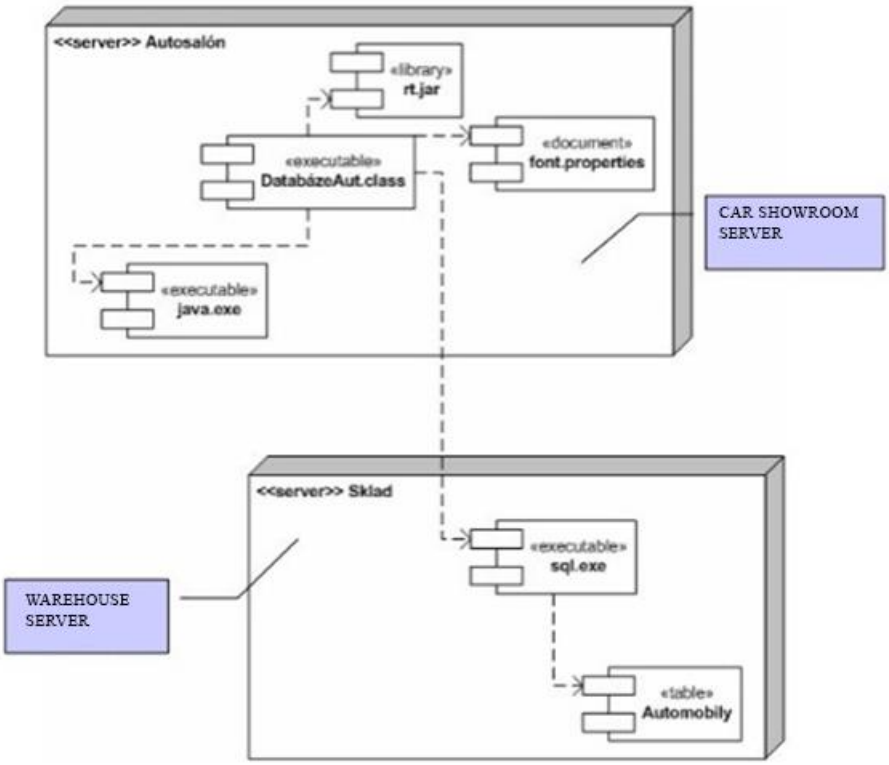


Diagram nasazení

k) Structure and usage of compiler. Description of source code and result program. How do an interpreter and compiler work.

Translator

A compiler is a program that translates programs from one programming language to another. Therefore, the compiler reads the source program written in the source language, translates it into an equivalent program (target program), but it is written in another (target) language.
The compilers also use higher programming languages such as Prolog. Lips, etc. Translators have also been used wherever the meaning of a text needs to be understood (semantic analysis). For example, structured editors, automatic code completers (bracket completion), formatting and typesetting programs (LaTex).
An important element of the compiler are diagnostic messages that inform about errors that occurred during the translation and which the compiler did not deal with.
According to the translation approach, we divide the translators into

- Compilers (compilers) - translates the source program code into code in the target language, which is then run and executed. The advantage of these compilers is that the program is compiled once and only the target program is run.
- Interpreters (interpretive compilers) - in contrast, interpreters skip the step of creating the target program and translate the source program directly into the actions that are performed. See. picture on the right.

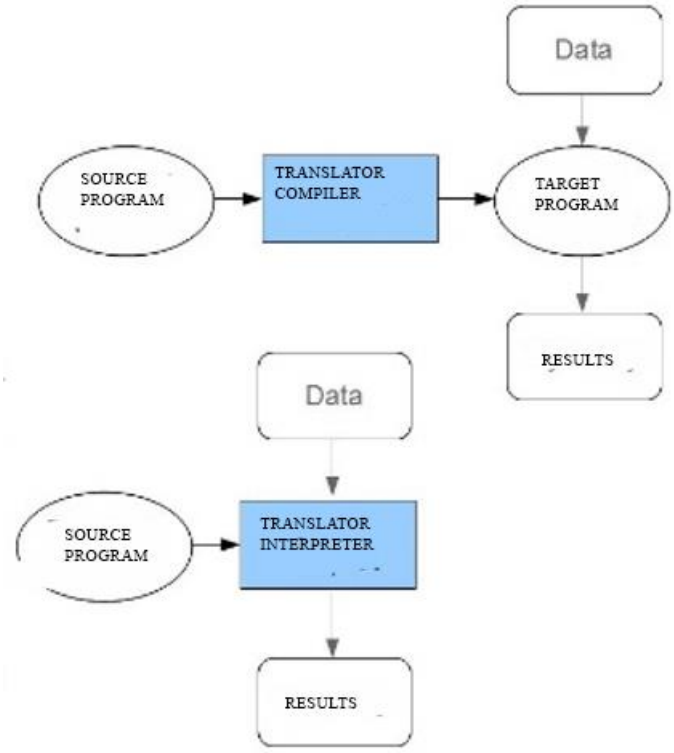
Depending on the architecture of the computer, we divide the target programs into:

- pure machine code - contains only the instructions of the computer instruction file, does not presuppose the existence of an operating system (OS) and is therefore fully dependent on the specific HW.
- extended machine code - uses OS subroutines in addition to the computer instruction file. It therefore depends on both the hardware and the software of the computer. Fortran compiler.

- virtual machine code - contains only virtual instructions that do not depend on either HW or SW of the computer. As a result, translators are portable, just add the interpreter of the virtual code.

According to the format, we divide the target code into:

- Symbolic format - a classic source with instructions in which one is familiar. This is most often an assembler.
- Relocative binary format - code in binary form, but without resolved references to external symbols. Addresses are relative from some initial section.
- Absolute binary format - binary executable code.



How does it work

The compiler performs two basic activities - it analyzes the source code and then translates it into the target program. The analysis includes lexical, syntactic and structural analysis.
The operation of the compiler is as follows: Source code -> pre-processor -> lexical analyzer -> parser -> intermediate code creation -> optimizer -> final code generation -> device-dependent optimization
Part of generating a target program is code optimization.

Lexical analysis

The text of the source code rewrites the lexical analyzer into a sequence of lexical units (lexemes), ie expressions having the meaning as identifier (variable), operator, assignment symbol. apod.

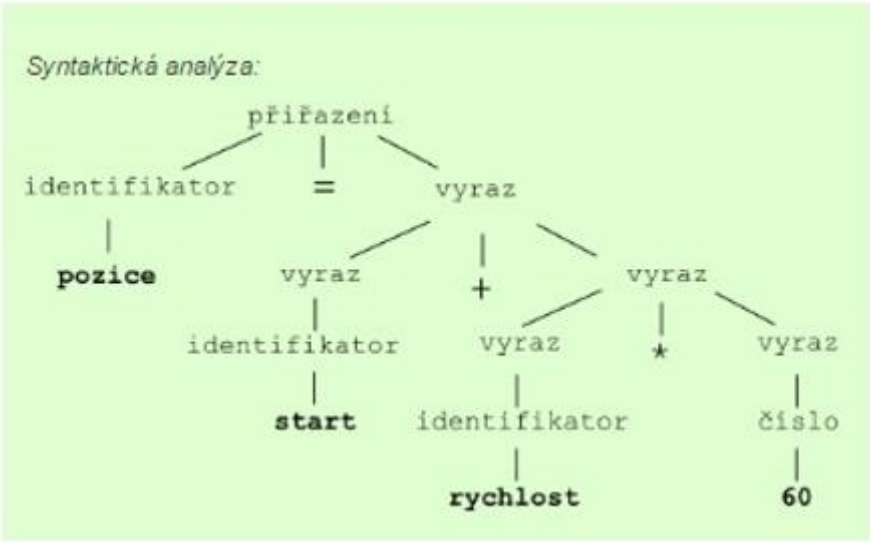
Source code:
position = start + speed * 60
Lexical analysis:
<id, position> <=> <id, start> <+> <id, speed> <*> <num, 60>

These lexemes are represented in the form of tokens that are provided for processing by the parser. The task of the lexical analyzer is also to remove comments and

whitespace from the source program.
Lexical analyzer functions:
1. finding and recognizing a lexical symbol
2. encoding of lexical symbols,
3. omitting unnecessary characters and symbols
4. calculation of synthesized attributes of lexical symbols.
In practice, the lexical analyzer is implemented using a finite state machine.

Syntactic analysis

Parsing converts lexical units to phrases. These are usually represented by a derivation tree. In this phase, context-free grammars are used, which construct the derivation machine from the expression.
The input is therefore tokens (eg parenthesis, literal, variable, keyword, symbol, etc.), which is for the parser a further indivisible basic building unit, which is stored in the sheets of the loaded data tree.



Semantic analysis

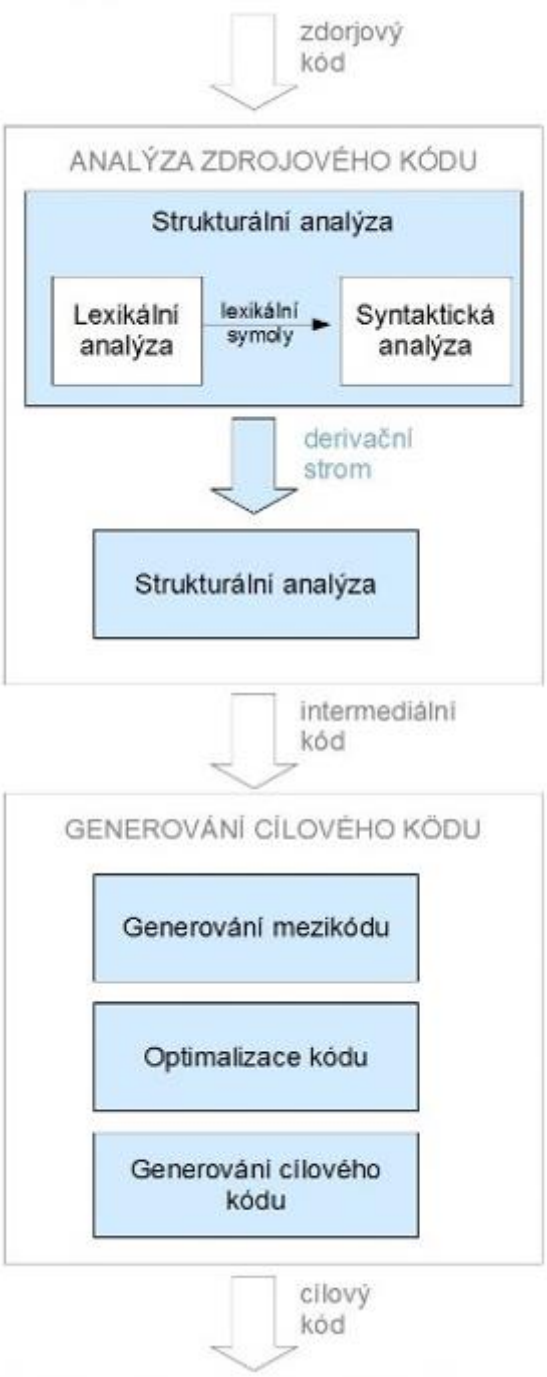
Semantic analysis gradually goes through symbols or groups of symbols obtained from syntactic analysis and is assigned meaning. For example, if a group of symbols represents the use of a particular variable, then the parser determines whether the variable is already declared (if required, does not require the Perl or PHP programming language, for example) and whether it is used correctly with respect to its data type. For example, the operation checks whether the operands are of the correct type or performs the necessary data type conversion.

Semantic analysis therefore mainly processes declarations, which it stores in its data structures, there is a type check and other checks that verify whether the code corresponds to the specification of the source language.

Intermediate code generation

Some compilers generate intermediate code (explicit intermediate code representation). It is a kind of code for an abstract computer, which Ize can easily create and at the same time easily convert into target code. For some interpretive compilers, this is the last stage and they are already executing the intermediate code straight away.

The intermediate codes at this stage can take various forms, one of which is the three-address code you see in the example on the right. The three-part code is similar to computer instructions and consists of a sequence of instructions of up to three operands.



Generating (three-address) intermediate code:
temp1 = internal (60)
temp2 = speed * temp1
temp3 = start + temp2
position = temp3

Code optimization

Improves code to make it faster or shorter. Some optimization methods are trivial, for example simplifying various calculations and reducing intermediate code variables as shown in the example on the right.

Source code optimization:
temp1 = speed * 60.0
position = start + temp1

Target code generation

The last step of the compiler is to generate the target code. The result is usually a relocatable machine code or program in an assembler. At this stage, all primeins are allocated memory space and the intermediate code instructions are translated into machine instructions.

The instructions in the example on the right contain F because we work with decimal values. The first operand of the instruction is the source, the second the target. The first instruction therefore says move the variable speed to registers 2, the next instruction is then multiplied by the constant 60 register 2, etc.

Target code generation:
MOVF speed, R2
MULF # 60.0, R2
MOVF start, R1
ADDF R2, R1
MOVE R1, position

5. **Data Processing Theory, Database and Information Systems, Information Systems Development**

a) Database systems modelling, conceptual modelling, data analysis, functional analysis.

DS modeling

Data model - a set of concepts that can be used to describe the structure of a database; 3 main categories

- Conceptual - a model independent of the database technology used
- Database - technology dependent, but not language specific (for relational databases -> Relational data model)
- Physical - language dependent, represents physical storage of data on media (cache vs hard disks, heap, etc. see another chapter)

In DAIS, the Data Model sometimes means only a data dictionary. see below.

Conceptual model

We specify the following terms

- Entity - real world object - practically a record in a table
- Attribute - entity property - practically a column in a table
- Entity type - a set of entities with the same attributes - practically a data table
- Key - one or more attributes that uniquely identify an entity in a set of entities
- Integrity constraints - a condition that must be met for an entity to be valid
- Relationship is recorded between entity types
    - Relationship cardinality - division of relations into 1: 1, 1: M, M: M
    - The obligation of the relationship

We illustrate the graphical conceptual model using the ER Diagram (entity relationship diagram)

FIGURE : Crow's foot notace - Oracle

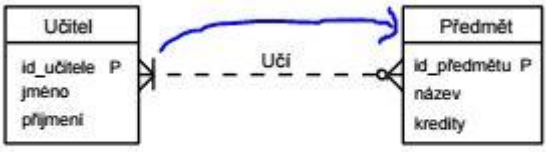


FIGURE : Crow's foot notace - Toad

The blue arrow indicates the binding obligation and the table to which it relates. The diagram shown in both cases describes that the teacher must have a subject.
The last possible representation is the use of UML diagrams, specifically a class diagram directly mapping to the database (persistent flag)

Analysis

Data analysis

- the output is a conceptual model
- we analyze the data that will be in the database, their properties, we usually write in a table in the so-called Data Dictionary

kategorie

Komentář k tabulce : Kategorie programů

Sloupec	Typ	Nulový	Výchozí	Odkazuje na	Komentáře	MIME
id	int(11)	Ne			Identifikátor kategorie	
rodic	int(11)	Ne	0	kategorie -> id	Identifikátor rodiče kategorie	
nazev	varchar(200)	Ne			Název kategorie	

licence

Komentář k tabulce : Licence programů

Sloupec	Typ	Nulový	Výchozí	Odkazuje na	Komentáře	MIME
id	int(11)	Ne			Identifikátor licence	
nazev	varchar(200)	Ne			Název licence	
url	varchar(200)	Ne			Stránka obsahující licenci	

software

Komentář k tabulce : Databáze programů

Sloupec	Typ	Nulový	Výchozí	Odkazuje na	Komentáře	MIME
id	int(11)	Ne			Identifikátor programu	
nazev	varchar(200)	Ne			Název programu	
popis	text	Ne			Popis programu	
url	varchar(200)	Ne			Internetová stránka programu	
kategorie	int(11)	Ne	0	kategorie -> id	Identifikátor kategorie programu	
licence	int(11)	Ne	0	licence -> id	Identifikátor licence programu	

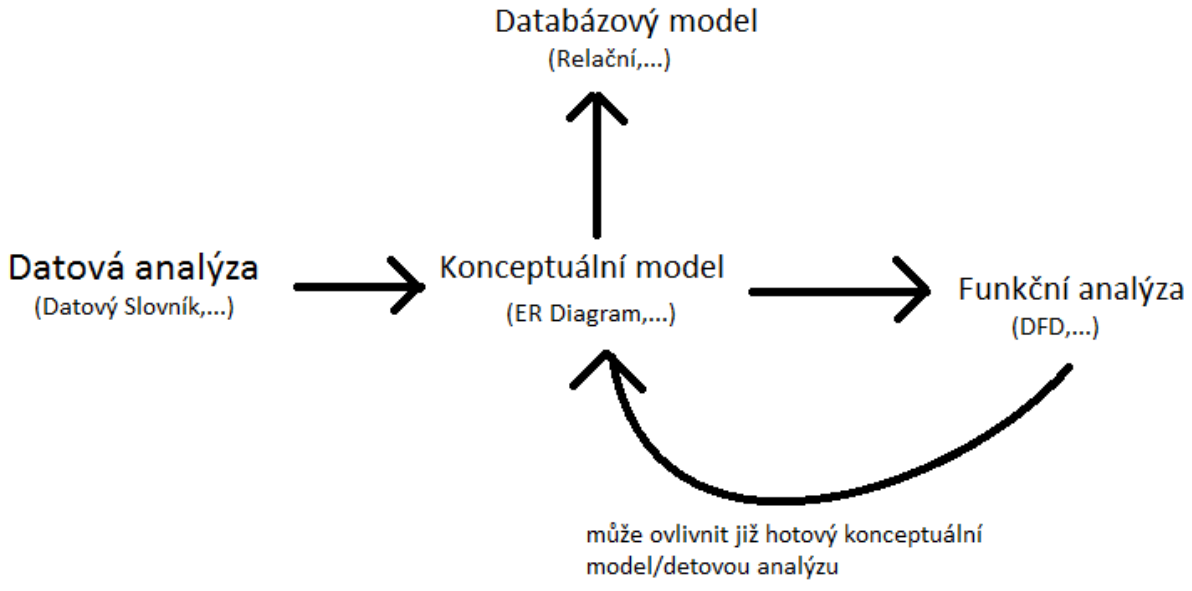
- We then create a specific conceptual model related to a specific data dictionary

Functional analysis

- solves system functions
- we list all the functions of the system, we solve their inputs and outputs, who can access it.
- We describe non-trivial functions in more detail, or we also write pseudocode.

Other analyzes can be state analysis (entities can take on some states - completed, created, in progress ...), requirements analysis (respectively specifications), graphical interface analysis

WORKFLOW Modelování DB/IS



b) Relational data model; function dependencies, decomposition and normal forms.

Database model specific to relational databases

Session = table whose rows are records, its columns are attributes, some of the attributes can be foreign keys, ie refer to the attributes of another session / table
On relations we have 3 basic query operations based on relational algebra that do not change the model itself

- selection - selection of certain lines (WHERE, IN, LIKE)
- projection - selection of certain columns (SELECT * / SELECT name)
- connection - (JOIN)

Other operations to change the relational model (update) - CREATE, INSERT, DELETE, UPDATE ...

Functional dependence + decomposition

Functional dependence - a relationship between X-> Y attributes, such that X will always affect Y. For example, name-> type, sprite -> drink, it is not possible for two records containing a sprite to have a different type. On the contrary, we cannot say that. Other examples can be order number -> customer, items; birth number -> date of birth)
Functional dependencies prevent problems such as rendundance (unnecessary repetition of records) or inconsistencies.

1. Data analysis
2. Making one super table where all attributes belong to one table
3. Determination of functional dependencies
4. Divide the table into smaller ones so that redundancy is eliminated (by using functional dependencies) and at the same time all functional dependencies are preserved.

Armstrong's axioms

There can be many functional dependencies even within one table. These axioms describe deriving these dependencies without having to write them all down.
The operation on axioms is similar to the operations on sessions / views -> Unification, Decomposition, Transitivity.

Normal forms

- 1NF - contains only atomic attributes (no attribute can be decomposed; eg address can be divided into street, city, etc.)
- 2NF - 1NF + in the case of a compound key, the left side of the dependency must form the entire key
- 3NF - 2NF + all dependencies are formed on the left side only by primary keys.
- BCNF - 3NF + no key candidates (combinations of attributes that uniquely define a row) do not share the attribute

There are also 4NF and 5NF

c) Query languages, relational algebra, SQL; DML, DDL.

Relational algebra

We work with sessions (sets); in addition to the obvious operations (unification, intersection, difference and Cartesian product), we also have defined operations of selection, projection and connection.

SQL

Standard, multiple revisions named by year as SQL-86, SQL: 2011 ...
Standard not always when implementing a specific language (MSSQL, MySql, Oracle SQL, ...) all constructions were available, the creators of the language created them themselves. Languages are not fully compatible.
Some SQL Servers also offer procedural extensions (Oracle: PL / SQL, Microsoft: T-SQL), but these are diametrically different.
SQL is a declarative language = statements describe what we want, but not HOW

JMD / DML vs JDD / DDL

JMD - Data manipulation language = SELECT, UPDATE, DELETE, INSERT
JDD - Data definition language = CREATE, ALTER, DROP

d) Transactions, recovery, log file, ACID, COMMIT and ROLLBACK operations.

Transaction

1. An operation that is commonly used to group many operations, among which the database is in an invalid / inconsistent state. The use of transactions reduces the number of database accesses.
2. Each transaction must meet the learned characteristics - ACID
    0. A - Atomicity - All operations in the transaction or none will take place
    1. C - Consistency - transition from one consistent state to another (there can be no violation of the integrity constraint)
    2. I - Isolation -Each transaction is independent of other transactions
    3. D - Durability - Past operations are permanent and persist even in the event of a system failure
3. **Commit** - performs all operations and writes them to the database.
4. **Rollback** - operation used in case of error. Cancels any changes made to the transaction.

Recovery

1. Recovery occurs after a system error to return the database to the correct state.
2. We classify system errors into several categories
    0. Power failure
    1. System failure
    2. DBMS failure (DBMS)
3. Recovery techniques
    0. Deferred update - write to DB only after COMMIT - NO-UNDO / REDO
    1. Immediate update - write immediately to DB - NO-REDO / UNDO
    2. Combined update - a combination of both

e) Procedural extensions of SQL: PL/SQL, triggers, cursors, bind variables, bulk operations.

Procedural extensions of SQL language

- PL-SQL for Oracle, T-SQL for Microsoft (and Sybase)
- Extensions enabling more efficient work with individual records
- They allow exceptions to be caught
- Introduction to working with cursors and variables
- Optimization of procedure performance, running directly on the server, independent of the language of the application itself

Triggers

- A function that performs an operation immediately after a pre-specified DML operation

```
CREATE OR REPLACE TRIGGER del_student
BEFORE DELETE ON student
FOR EACH ROW
BEGIN
  INSERT INTO hist_stud(login, name, surname)
   values(:OLD.login, :OLD.name, :OLD.surname);
END;
```

Výpis 18.6: Příklad jednoduchého triggeru

Procedures

- They can be anonymous or named
- Performs a specific operation, they do not have a return value
- It can accept the declared number of parameters. Some of these parameters

may be output.

```
DECLARE
  v_name VARCHAR2(30) := 'michal.kratky@vsb.cz';
BEGIN
  INSERT INTO Email VALUES (v_name);
END;
```

NAMED

```
CREATE [OR REPLACE] PROCEDURE jmeno_procedury
  [(jmeno_parametru [mod] datovy_typ, ... )]
IS|AS
  definice lokálních proměnných
BEGIN
  tělo procedury
END [jmeno_procedury]
```

Function

- Unlike procedures, they have a specified return type and always return a value

```
CREATE [OR REPLACE] PROCEDURE jmeno_procedury
  [(jmeno_parametru [mod] datovy_typ, ... )]
RETURN navratovy_datovy_typ
IS|AS
  definice lokálních proměnných
BEGIN
  tělo procedury
END [jmeno_procedury]
```

Cursors

- Cursors are auxiliary variables created after an SQL statement is executed. Used to scroll through this command
- Two types
  - Default cursor - automatically created after DML command (INS, DEL, UPD)
  - Explicit cursor - declared in the definition part of the procedure.

f) Physical database design; the heap table, indices (B-tree), table data clustering.

Heap table

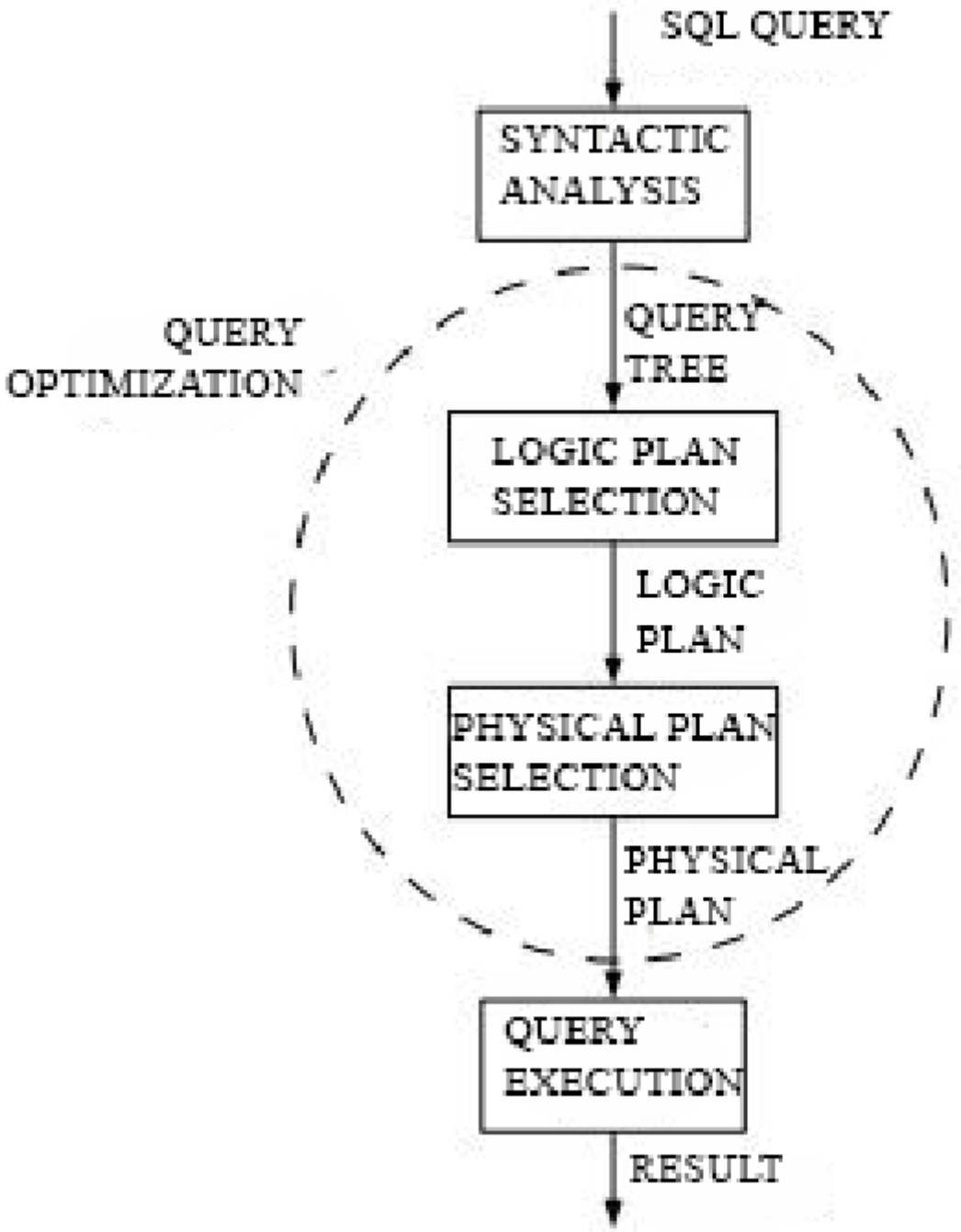
- Default selection at CREATE TABLE
- Records are inserted in the 1st free position or at the end -> quick insert O (1)
- Records are not sorted by default -> O (n) SELECT
- Records are not deleted by default, only flagged -> SHRINK command

Clustering of records

- Records are immediately sorted according to the primary key when created -> inefficient insert
- Since the records are already sorted, the records can be selected, for example, by binary search
- Keys are stored in special structures
  - **B + tree** - balanced tree, pointer always to the next element, O log (n)
  - **Compound index** - an index on multiple fields, when searching for only one of the attributes, the complexity is the same as if the key did not exist (suitable for specific combinations)
  - **Hash table** - keys are not organized, but searching for one record is quite fast
  - **Bitmap index** - suitable for keys that take on only small values (gender, marital status, booleans, etc.)
- Records are not deleted by default, only flagged -> SHRINK command

g) Query evaluation in database systems; query execution plan.

- SQL queries are not executed in the database exactly as declared.
- There is an optimizer that modifies commands into a lower cost form
- Optimization is based on relational algebra.
- The number of alternative plans grows exponentially, only a few plans are evaluated -> the final price may not necessarily be the lowest possible.
- Optimization has 4 main steps
    - Syntactic analysis - Query conversion to relational algebra (tree generation)
    - Logical plan selection - selection of the best tree (relational algebra)
    - Physical plan selection - selection of specific algorithms (SQL) of the query subpart
    - Execution of the query - Compilation of plans = joining physical plans according to the selected logical plan
- Two essential parts of any optimization
    - The plan is minimized and redundant parts are omitted
    - Rearrangement of individual operations so that the result takes as little time and resources as possible



Parts of query evaluation

h) Object-relational data model.

- Due to the costly migration of relational data, an object-relational data model was created
- Introduction of collections, inheritance, references / indicators, indicators
- Object types and their functions are stored together with the data in the database (there is no need to create them again for each application)
- Data types can then be used in new RELATION and OBJECT tables.

```
CREATE OR REPLACE TYPE TDepartment AS OBJECT (
  name VARCHAR2(45),
  shortcut NUMBER(3),
  chief REF TTeacher);
```

i) Data layer of information systems; API, frameworks and implementations; transactions in programming languages, security, object-

relational mapping.

- The data layer of the information system separates the application from the database. These are the classes and functions that ensure communication with the database.
- Design patterns on this layer
    - Data Access Object / Data Mapper
    - Active Record (not exactly a typical shortcut suitable for a three - tier arch.
    - Row Data Gateway - instance = one table row; contains the search class that instantiates us. It can implement Table Data Gateway
    - Table Data Gateway - instance = table; can be implemented in the RDGateway search class

Existing API

- Database API = interface to access the database. For example, execute QUERY from the programming environment.
- ODBC = Open database Conectivity = independent of the programming language, it contains drivers for practically all relational databases, but also for example CSV
- JAVA -> JDBC
- .NET -> ADO.NET

ORM

- Due to the costly migration of relational data, an object-relational data model was created
- Introduction of collections, inheritance, references / indicators, indicators
- Object types and their functions are stored together with the data in the database (there is no need to create them again for each application)
- Data types can then be used in new RELATION and OBJECT tables.

j) Concurrency in database systems, anomalies of concurrency, techniques and implementations; serial and serializable plans, isolation levels of transactions in SQL.

- Concurrency = multiple accesses to the same data at the same time
- The following conditions may occur
    - Read + Read => no problem no worries
    - Read + Write => Inconsistent analysis = A sequence of reads is in progress, but records already read may change due to another transaction, resulting in inconsistencies.
    - Write + Read => Unconfirmed dependency = the second transaction reads after a write, but which has not yet been COMMITnuted and was later ROLLBACKnuted.
    - Write + Write => Loss of update = the first transaction writes successfully, but the second overwrites it

- Ztráta aktualizace (**Lost Update**):

Transakce A		Transakce B	
READ	t1		
		t2	READ
WRITE	t3		
		t4	WRITE

Dirty Write

- Nepotvrzená závislost (**Uncomitted dependancy**)

Transakce A		Transakce B	
		t1	WRITE
READ	t2		
		t3	ROLLBACK

Dirty Read

Transakce A v čase t2 pracuje s daty, který byly rollbacknuty (jejich update byl zrušen)

- Nekonzistentní analýza (**Inconsistent analysis**)

acc1 = 20€ | acc2 = 30€ | acc3 = 50€			
Transakce A		Transakce B	
READ acc1	t1		
suma = 20			
READ acc2	t2		
suma = 50			
		t3	READ acc3
		t4	WRITE acc3
		suma = 60	
		t5	READ acc2
		t6	WRITE acc2
		suma = 20	
		t7	COMMIT
READ acc3	t8		
suma = 110			

Non-repeatable read

Z acc2 přeneseme 10€ do acc 3

Transakce A pracuje s nekonzistentní databází -> má nekonzistentní analýzu. Výsledná suma všech účtů má být 100, ne 110.

Concurrency solution

- Pessimistic = locking = we assume that concurrence occurs
  - There are two types of locks
  - S = read lock = transaction holding this lock, can read from the record but not write
  - X = write lock = transaction holding this lock, can read from and write to the record.
  - Lock S can get multiple transactions at once. If an X lock exists, it can only be owned by one transaction, and there cannot be an S.
  - If the transaction wants to write and has S (and no other lock S is on the records), S is upgraded to X.
  - If, for example, two transactions hold S and both want to write (ask for X), they will wait indefinitely for each other => deadlock.
  - Deadlock can be solved - timeout, graph detection, modification of locks with timestamps (Wait-die / Wound-wait)
- Optimistic solution = versioning

Insulation levels

1. Read uncommited - the transaction can also read intermediate transactions of another transaction
2. Read commited - sees states only before the start of the query, locks can be released before the end of the transaction
3. Repeatable read - states cannot change during the transaction, but new ones can be added in the meantime
4. Serializable - maximum level of isolation = if there is a lock can not be added, changed, deleted ...

k) Architecture and structure of information system. Rules and principles. Components, connectors, configurations. Decomposition. The relationship of architecture, design and implementation of information systems.

Architecture

- This is a very abstract concept of a view of the whole system
- includes the basic structure / organization of that system
- ie. interconnection of components, their physical location, its principles, relations with the environment ...
- each party - developer, customer, investor, ... understands what is going on

Proposal

- It is based on architecture, it is a concretization of problems and their solutions
- ergo HOW and with WHAT the system works

Deployment

- describes the KDE system running (on which HW, platform,…).

IS structure

- Component - a separate part of the system with an interface
- Connector - a channel between components, specifically between their interfaces
- Configuration - specific connection of components using connectors

l) Three competencies of information system and three-tier architecture. The logical and physical architecture of information systems. Patterns for enterprise architecture. Patterns of domain logic, data access, object- relational behavior. Principles of object-relational mapping and mapping of inheritance.

3 Competences + 3-tier architecture

- Corresponds to a three-tier architecture
- Communication with the user (presentation layer)
- Information processing and temporary retention (business layer)
- Permanent data storage (database layer)
- Logical and physical architecture of information system.
- Logical = division into eg three-tier architecture. This is more or less the standard. Sometimes including the service layer.
- Physical = specific real - world component deployments (server deployments, etc.)

P of EAA

- Patterns for domain logic
    - Transaction script - each query from the translation layer (to the business

layer) has its own function.
    - Domain model - division of the business layer into a "network of connected objects".
    - Table module - one instance manages one table in the database (all its rows)
    - Service layer - covers the whole structure / business logic and clearly defines what can be done with the application.
- Data Layer (ORM) patterns
    - Table data gateway
    - Row data gateway
    - Active record
    - Data mapper (DTO + DAO)
- Object-relational behavior
    - Unit of work
    - Lazy load
    - Identity map - the object lives in memory only once. If we reach for the object for the first time - it will be created (new), if someone else wants it, then it will send this one and it will not create new ones from the DB.
- Object-relational structures
    - Identity field - mapping of the primary key to the integer in the given instance.
    - Foreign Key Mapping - mapping a foreign key as an object reference.
    - Assocation Table Mapping - If we have M: N, we need an association table in the DB.
    - Dependent mapping - one class can map both the whole and its parts (Album <composition> -> Track -> AlbumMapper, no need for TrackMapper)
    - Embedded value - mapping of own data type to another object's table.
    - Serialized LOB - BLOB (Graph) to DB ... About to store a huge structure for backup? When it doesn't make sense to create a new Relational Model ...
- Inheritance mapping
    - Single table inheritance - one table for all classes. the disadvantage is the NULL values in the DB.
    - Class table inheritance - there is a table for each class, but they are not independent. Only things that add value are defined in them. The disadvantage is to create a single object to write to x tables.
    - Concrete table inheritance - there is a specific table for each class. The downside is data redundancy.

m) Systems development life cycle, Zachman Framework. Tasks, roles, issues. Principles of information system development. Principles and phases of Unified Process. Robust and agile approaches to the information system development.