



Pokročilé informační systémy

Principy, data, modely, architektury

prof. Ing. Tomáš Hruška, CSc.

Ing. Radek Burget, Ph.D.

burgetr@fit.vutbr.cz

Předmět PIS – Cíle předmětu

- Návrhu informačního systému – rozšíření znalostí z IIS
 - Analýza domény a procesů (datové modelování, workflow)
 - Pokročilé architektury systému
 - Systémy pro podporu rozhodování
- Zvládnutí pokročilých technologií
 - Databázová vrstva – objektový datový model, alternativy
 - Aplikační vrstva – aplikační rámce pro enterprise aplikace
 - Prezentační vrstva – klientské programování
- Business intelligence (OLAP systémy)

Technologie

- Cílem je zvládnout základní koncepty a principy **nezávisle na implementační platformě**
- Praktické ukázky (a projekt) budou využívat zejména
 - Java Enterprise Edition (nově Jakarta EE)
 - Alternativy na jiných platformách
 - Objektově-relační mapování (JPA)
 - REST aplikační rozhraní, webový framework (JSF a další)

Hodnocení

- Půlsemestrální zkouška: 19 bodů
- Semestrální zkouška: 51 bodů
- Projekt: 30 bodů (8 + 22)
 - Realizace IS na dané téma v týmu (zadání z AIS)
 - ORM datová vrstva
 - Pokročilý aplikační framework umožňující oddělení datové, aplikační a prezentacní vrstvy

Kontakty

- Ing. Radek Burget, Ph.D.

burgetr@fit.vutbr.cz

C223

- Přednášky, projekty, zkoušky, všechno ostatní

- Ing. Jiří Hynek, Ph.D.

ihynek@fit.vutbr.cz

C235

- Přednášku

Data – informace – znalosti

Data

- Hodnota schopná přenosu, uchování, interpretace či zpracování
- Z hlediska IT jde o *hodnoty různých datových typů*
- Data sama o sobě *nemají sémantiku* (význam), jsou to věty nějakého formálního jazyka
 - Viz pojem *databáze*
 - Hodnoty dat obvykle udávají *stav* nějakého systému

Informace

- *Informace* jsou interpretovaná data
- Mají *sémantiku* (význam)
- Transformaci dat na informace neprovádí informační systém, ale *uživatel*
 - Systém ukládá a transformuje *data*
 - Pro uživatele výsledek znamená *informaci*
- Je nezbytné zajistit shodnou interpretaci dat u všech uživatelů informace
 - vzdělání, školení, zavedení konvencí

Příklad rozdílné interpretace dat

- Údaj 10-12-2005
- V Evropě informace 10. prosince 2005
- V USA informace 12. října 2005
- pro totožná data vznikne *rozdílná informace* jinou *interpretací* dat
- Podobně např. jméno a příjmení

Znalost

- Informace zařazená do souvislostí
- Jejich interpretace je však ještě hůře definovatelná, neboť může jít o celé shluky informací
- Znalosti chápeme často jako *sekundární odvozené informace*
- Některé informační systémy se zabývají pouze *informacemi (transakční)*, některé pracují se *znalostmi (pro podporu rozhodování a plánování)*
- Problematika získávání znalostí z dat (knowledge discovery, data mining)

Příklad: jízdní řád

1 | 

Odjezdy ze zastávky **Semilasso**
směr **Řečkovice**

IDS JMK
1566/1

Zóna 101
ČEROVÁ &
Ondrouškova &
Kubíčkova (o)
Přistaviště &
Zoologická zahrada &
Kamenolom (z) &
Podlesí (o) &
Branka (z) &
Svratecká &
Vozovna Komín &

Zóna 100
Stránského (o) &
Pisárky &
Lipová (o) &
Výstaviště - vstup G2 (z)
Mendlovo náměstí &
Václavská &
Hybešova &
Nové sady &
Hlavní nádraží &
Malinovského náměstí &
Moravské náměstí &
Antonínská &
Pionýrská &
Hrnčířská &
Šumavská &
Kartouzská &
Jungmannova &

Zóna 101
Husitská &
Semilasso &
1 Tylova &
2 Hudcová &
4 Kříšková &
5 Fílkuková &
1 ŘEČKOVICE &

o : zastávka od 20 do 5 hodin na znamení
z : zastávka celodenně na znamení
: bezbariérová zastávka

PRACOVNÍ DNY			SOBOTA			NEDĚLE			
NEPLATÍ 17.4.–18.4., 2.5., 9.5., 30.6.–29.8., 27.10.–29.10., 22.12.2014–2.1.2015			PLATÍ TAKE 20.4., 1.5., 8.5., V NEDĚLE OD 15.6. DO 7.9., 26.10., 16.11., 21.12., 24.12. (do 16 hod.), 25.12., 26.12., 28.12., 31.12.2014 (do 20 hod.), 1.1.2015			NEPLATÍ 20.4., OD 15.6. DO 7.9., 26.10., 16.11., 21.12., 28.12.2014; PLATÍ TAKÉ 21.4., 28.10., 17.11.2014			
2	3	4	5	6	7	8	9	10	
5 22& 32& 42 52& 59	6 05 12& 19 25 32 39 45& 52 58&	7 03 07 13 18& 23 28 33 36 39 43& 46 49 53& 56 59	8 03& 06 09 13 18& 23 28 33 38 43 48 53& 58	9 03 08& 13 18 23 28& 33 38 43 48 53 58&	10 03 08& 13 18& 23 28 33 38& 43 48 53 58	11 03 08 13& 18 23 28& 33 38 43 48& 53 58	12 03 08 13 18& 23 28& 33 38& 43 48 53 58&	13 03 08 13 18 23 28 33& 38 43 48& 53 58	14 03 08& 13 18 23 28 33 38& 43 48& 53 58&
15 03 08 13 18& 23 28 33 38 43 48 53 58	16 03 08& 13 18 23 28& 33 38 43 48 53 58&	17 03 08& 13 18& 23 28 33 38& 43 48 53 58	18 03 08 13& 18 23 28& 33 38 43 48& 53 59	19 05 12 19& 25 32 39 45 52	20 02 12 22 32& 43 58&	21 13 28& 43 58&	22 13 28 43 58&	23	2 13 28& 43 58&
20 02 12 22 32& 43 58&	21 13 28& 43 58&	22 13 28 43 58&	23	2 13 28& 43 58&	3 13 28 43 58&	4 13 28 43 58&	5 13 28 43 58&	6 13 28 43 58&	7 13 28 43 58&
24 02 12 22 32 32& 43 58	25 02 12 22 32 32& 42 52	26 02 12 22 32 32 42& 52	27 02 12 22 32 32 42 42& 52	28 02 12 22 32 32 42 42 52	29 02 12 22 32 32 42 42 52	30 02 12 22 32 32 42 42 52	31 02 12 22 32 32 42 42 52	32 02 12 22 32 32 42 42 52	33 02 12 22 32 32 42 42 52
34 02 12 22 32 32 42 42 52	35 02 12 22 32 32 42 42 52	36 02 12 22 32 32 42 42 52	37 02 12 22 32 32 42 42 52	38 02 12 22 32 32 42 42 52	39 02 12 22 32 32 42 42 52	40 02 12 22 32 32 42 42 52	41 02 12 22 32 32 42 42 52	42 02 12 22 32 32 42 42 52	43 02 12 22 32 32 42 42 52
44 02 12 22 32 32 42 42 52	45 02 12 22 32 32 42 42 52	46 02 12 22 32 32 42 42 52	47 02 12 22 32 32 42 42 52	48 02 12 22 32 32 42 42 52	49 02 12 22 32 32 42 42 52	50 02 12 22 32 32 42 42 52	51 02 12 22 32 32 42 42 52	52 02 12 22 32 32 42 42 52	53 02 12 22 32 32 42 42 52
54 02 12 22 32 32 42 42 52	55 02 12 22 32 32 42 42 52	56 02 12 22 32 32 42 42 52	57 02 12 22 32 32 42 42 52	58 02 12 22 32 32 42 42 52	59 02 12 22 32 32 42 42 52	60 02 12 22 32 32 42 42 52	61 02 12 22 32 32 42 42 52	62 02 12 22 32 32 42 42 52	63 02 12 22 32 32 42 42 52
64 02 12 22 32 32 42 42 52	65 02 12 22 32 32 42 42 52	66 02 12 22 32 32 42 42 52	67 02 12 22 32 32 42 42 52	68 02 12 22 32 32 42 42 52	69 02 12 22 32 32 42 42 52	70 02 12 22 32 32 42 42 52	71 02 12 22 32 32 42 42 52	72 02 12 22 32 32 42 42 52	73 02 12 22 32 32 42 42 52
74 02 12 22 32 32 42 42 52	75 02 12 22 32 32 42 42 52	76 02 12 22 32 32 42 42 52	77 02 12 22 32 32 42 42 52	78 02 12 22 32 32 42 42 52	79 02 12 22 32 32 42 42 52	80 02 12 22 32 32 42 42 52	81 02 12 22 32 32 42 42 52	82 02 12 22 32 32 42 42 52	83 02 12 22 32 32 42 42 52
84 02 12 22 32 32 42 42 52	85 02 12 22 32 32 42 42 52	86 02 12 22 32 32 42 42 52	87 02 12 22 32 32 42 42 52	88 02 12 22 32 32 42 42 52	89 02 12 22 32 32 42 42 52	90 02 12 22 32 32 42 42 52	91 02 12 22 32 32 42 42 52	92 02 12 22 32 32 42 42 52	93 02 12 22 32 32 42 42 52
94 02 12 22 32 32 42 42 52	95 02 12 22 32 32 42 42 52	96 02 12 22 32 32 42 42 52	97 02 12 22 32 32 42 42 52	98 02 12 22 32 32 42 42 52	99 02 12 22 32 32 42 42 52	100 02 12 22 32 32 42 42 52	101 02 12 22 32 32 42 42 52	102 02 12 22 32 32 42 42 52	103 02 12 22 32 32 42 42 52
104 02 12 22 32 32 42 42 52	105 02 12 22 32 32 42 42 52	106 02 12 22 32 32 42 42 52	107 02 12 22 32 32 42 42 52	108 02 12 22 32 32 42 42 52	109 02 12 22 32 32 42 42 52	110 02 12 22 32 32 42 42 52	111 02 12 22 32 32 42 42 52	112 02 12 22 32 32 42 42 52	113 02 12 22 32 32 42 42 52
114 02 12 22 32 32 42 42 52	115 02 12 22 32 32 42 42 52	116 02 12 22 32 32 42 42 52	117 02 12 22 32 32 42 42 52	118 02 12 22 32 32 42 42 52	119 02 12 22 32 32 42 42 52	120 02 12 22 32 32 42 42 52	121 02 12 22 32 32 42 42 52	122 02 12 22 32 32 42 42 52	123 02 12 22 32 32 42 42 52
124 02 12 22 32 32 42 42 52	125 02 12 22 32 32 42 42 52	126 02 12 22 32 32 42 42 52	127 02 12 22 32 32 42 42 52	128 02 12 22 32 32 42 42 52	129 02 12 22 32 32 42 42 52	130 02 12 22 32 32 42 42 52	131 02 12 22 32 32 42 42 52	132 02 12 22 32 32 42 42 52	133 02 12 22 32 32 42 42 52
134 02 12 22 32 32 42 42 52	135 02 12 22 32 32 42 42 52	136 02 12 22 32 32 42 42 52	137 02 12 22 32 32 42 42 52	138 02 12 22 32 32 42 42 52	139 02 12 22 32 32 42 42 52	140 02 12 22 32 32 42 42 52	141 02 12 22 32 32 42 42 52	142 02 12 22 32 32 42 42 52	143 02 12 22 32 32 42 42 52
144 02 12 22 32 32 42 42 52	145 02 12 22 32 32 42 42 52	146 02 12 22 32 32 42 42 52	147 02 12 22 32 32 42 42 52	148 02 12 22 32 32 42 42 52	149 02 12 22 32 32 42 42 52	150 02 12 22 32 32 42 42 52	151 02 12 22 32 32 42 42 52	152 02 12 22 32 32 42 42 52	153 02 12 22 32 32 42 42 52
154 02 12 22 32 32 42 42 52	155 02 12 22 32 32 42 42 52	156 02 12 22 32 32 42 42 52	157 02 12 22 32 32 42 42 52	158 02 12 22 32 32 42 42 52	159 02 12 22 32 32 42 42 52	160 02 12 22 32 32 42 42 52	161 02 12 22 32 32 42 42 52	162 02 12 22 32 32 42 42 52	163 02 12 22 32 32 42 42 52
164 02 12 22 32 32 42 42 52	165 02 12 22 32 32 42 42 52	166 02 12 22 32 32 42 42 52	167 02 12 22 32 32 42 42 52	168 02 12 22 32 32 42 42 52	169 02 12 22 32 32 42 42 52	170 02 12 22 32 32 42 42 52	171 02 12 22 32 32 42 42 52	172 02 12 22 32 32 42 42 52	173 02 12 22 32 32 42 42 52
174 02 12 22 32 32 42 42 52	175 02 12 22 32 32 42 42 52	176 02 12 22 32 32 42 42 52	177 02 12 22 32 32 42 42 52	178 02 12 22 32 32 42 42 52	179 02 12 22 32 32 42 42 52	180 02 12 22 32 32 42 42 52	181 02 12 22 32 32 42 42 52	182 02 12 22 32 32 42 42 52	183 02 12 22 32 32 42 42 52
184 02 12 22 32 32 42 42 52	185 02 12 22 32 32 42 42 52	186 02 12 22 32 32 42 42 52	187 02 12 22 32 32 42 42 52	188 02 12 22 32 32 42 42 52	189 02 12 22 32 32 42 42 52	190 02 12 22 32 32 42 42 52	191 02 12 22 32 32 42 42 52	192 02 12 22 32 32 42 42 52	193 02 12 22 32 32 42 42 52
194 02 12 22 32 32 42 42 52	195 02 12 22 32 32 42 42 52	196 02 12 22 32 32 42 42 52	197 02 12 22 32 32 42 42 52	198 02 12 22 32 32 42 42 52	199 02 12 22 32 32 42 42 52	200 02 12 22 32 32 42 42 52	201 02 12 22 32 32 42 42 52	202 02 12 22 32 32 42 42 52	203 02 12 22 32 32 42 42 52
204 02 12 22 32 32 42 42 52	205 02 12 22 32 32 42 42 52	206 02 12 22 32 32 42 42 52	207 02 12 22 32 32 42 42 52	208 02 12 22 32 32 42 42 52	209 02 12 22 32 32 42 42 52	210 02 12 22 32 32 42 42 52	211 02 12 22 32 32 42 42 52	212 02 12 22 32 32 42 42 52	213 02 12 22 32 32 42 42 52
214 02 12 22 32 32 42 42 52	215 02 12 22 32 32 42 42 52	216 02 12 22 32 32 42 42 52	217 02 12 22 32 32 42 42 52	218 02 12 22 32 32 42 42 52	219 02 12 22 32 32 42 42 52	220 02 12 22 32 32 42 42 52	221 02 12 22 32 32 42 42 52	222 02 12 22 32 32 42 42 52	223 02 12 22 32 32 42 42 52
224 02 12 22 32 32 42 42 52	225 02 12 22 32 32 42 42 52	226 0							

Správa informací

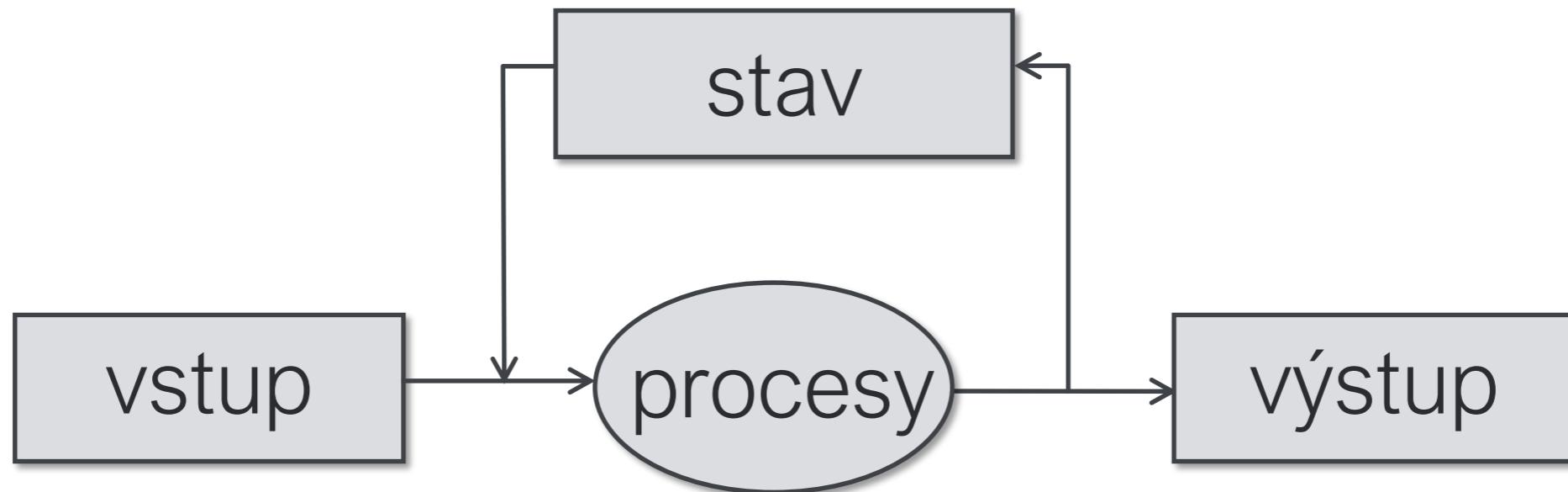
1. Sběr,
2. Uspořádání a příprava,
3. Užití,
4. Rušení a náhrada.

Správa probíhá dle základních funkcí *systému*

- stav, data (zpětná vazba)
- transformace a procesy
- vstup a výstup (komunikace)

INFORMAČNÍ SYSTÉM

Schéma informačního systému



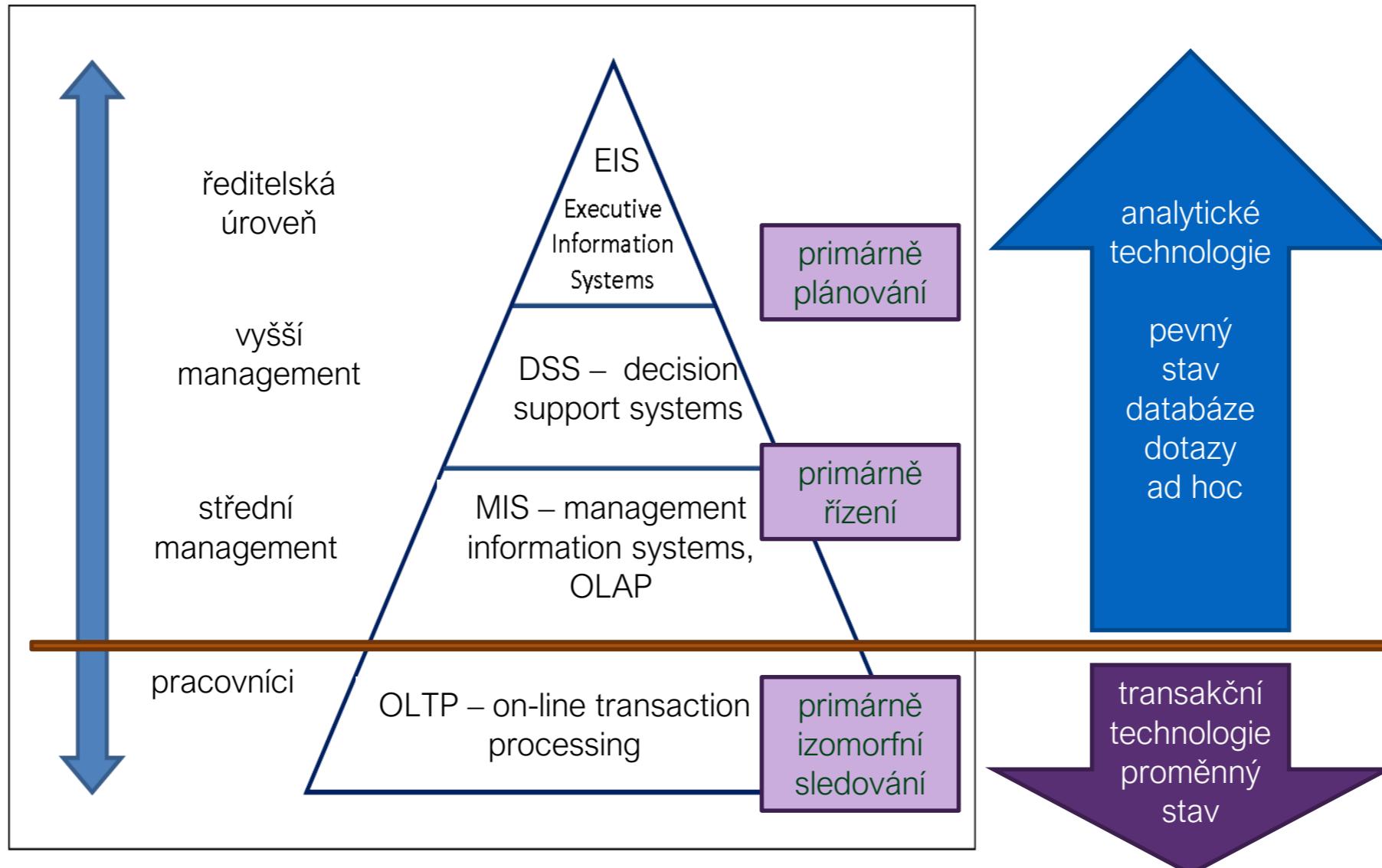
- Modifikované schéma obecného systému
- Data uchovávající *stav* systému a
- *Procesy* realizující transformace často ve formě *transakcí*

Stav informačního systému

- Stavem informačního systému jsou hodnoty dat (typicky reprezentované pomocí nějakého *modelu*) a musíme se zabývat jejich
 - *Persistencí* (přetrváváním),
 - *Konzistencí* (splňování jistých pravidel o možných kombinacích hodnot údajů ve stavu) apod.

Informační systémy podle úrovně rozhodování

Pyramidové schéma



OLTP – On-line transaction processing

- Třída informačních systémů, které zpracovávají transakčně orientované aplikace
- Termín transakční je dvojznačný:
 - *databázové transakce*
 - *komerční transakce*
- Tento termín je rovněž používán v případě, kdy systém odpovídá okamžitě na uživatelské požadavky změnou stavu, příkladem aplikace pro komerční transakce může být např. *bankomat*.

OLTP – On-line transaction processing

- Metodologie, která poskytuje koncovému uživateli přístup k rozsáhlým objemům dat a pomáhá jejich zkoumání.
- podpora pro databázové transakce, které zahrnují i síťové zpracování a mohou být *dlouhé a distribuované*
- užívají zpracování typu klient/server a dovolují transakcím běžet na rozdílných platformách v síti
- pro decentralizované databázové systémy musejí OLTP distribuovat transakce mezi mnoha počítačů a sítí

Návrh informačního systému

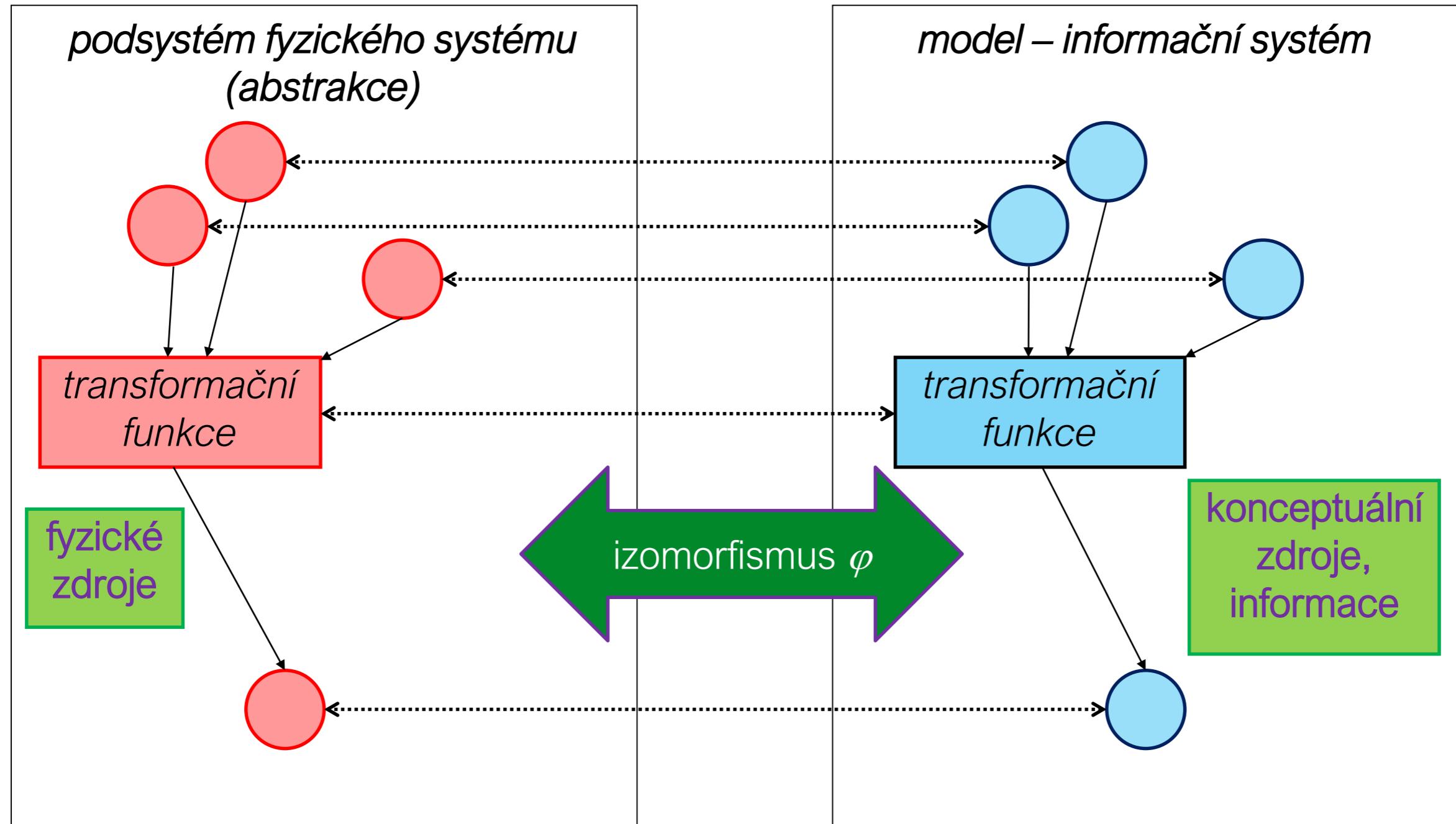
Informační systém jako model

- Informace *modelují skutečné zdroje jiného – obvykle fyzického systému* (např. podniku)
- Informační systém tedy na nehmotné – virtuální úrovni *modeluje svůj fyzický vzor*, pro jehož řízení je obvykle vytvářen. Vzhledem k tomu, že model nikdy *nemůže postihnout veškeré chování a vlastnosti svého vzoru*, je virtuální kopie pořizována vždy na vhodné úrovni *abstrakce*

Izomorfismus

- *Izomorfismus* je zobrazení mezi dvěma matematickými strukturami, které je vzájemně jednoznačné (bijektivní) a zachovává všechny vlastnosti touto strukturou definované.
- Jinými slovy, každému prvku první struktury odpovídá právě jeden prvek struktury druhé a toto přiřazení zachovává vztahy k ostatním prvkům.

OLTP jako model



Nezbytnost abstrakce

- Není možné modelovat všechny zdroje i procesy fyzického systému. Vždy se vybírají jen ty, které jsou pro úroveň řízení, pro kterou OLTP budujeme, podstatné – modelujeme ***pod systém*** původního fyzického systému – ***abstrahujeme***
- OLTP je proto vždy modelem jisté ***abstrakce původního fyzického vzoru***.

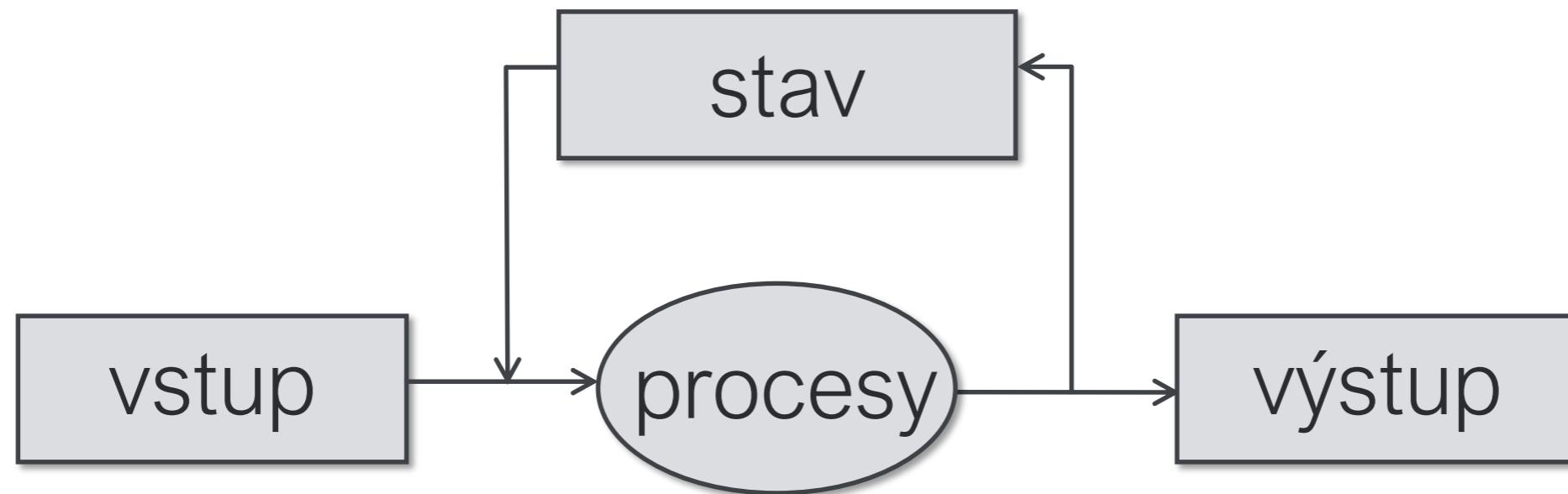
OLTP jako model

- říkáme, že OLTP modeluje nějaký fyzický pod systém.
- mezi OLTP a jeho fyzickým vzorem existuje izomorfismus φ
- pokud je v původním systému funkce nad zdroji, potom v OLTP existuje obraz této funkce pracující s obrazy zdrojů.
- pokud funkce v původním systému má za parametry jisté zdroje a dává jistý výsledek, pak obraz funkce v OLTP mající za parametry obrazy původních zdrojů dává za výsledek obraz původního výsledku.
- To platí i naopak.

Příklad OLTP systému

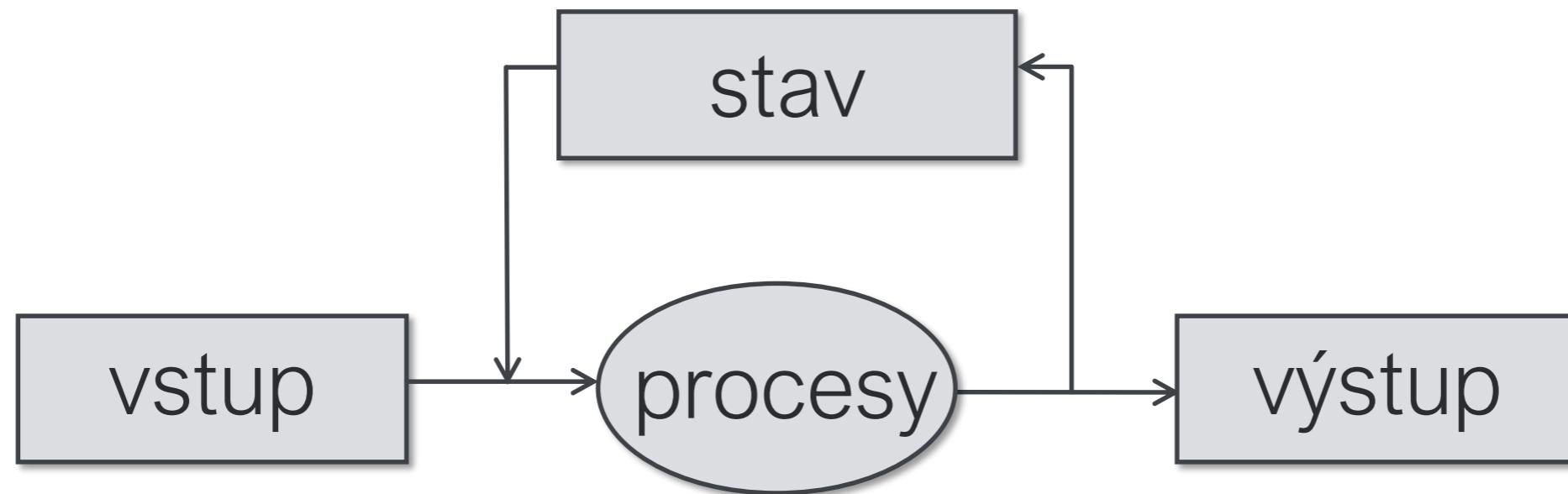
- ve fyzickém systému se pracuje s peněžními zdroji, tj. ***skutečnými penězi***, pak se v informačním systému pracuje s jejich ***virtuálním obrazem***.
- pokud ve fyzickém systému je provedena funkce, která na základě objednávky vytvoří skutečnou fakturu, pak v informačním systému je vytvořen její obraz.

Návrh informačního systému

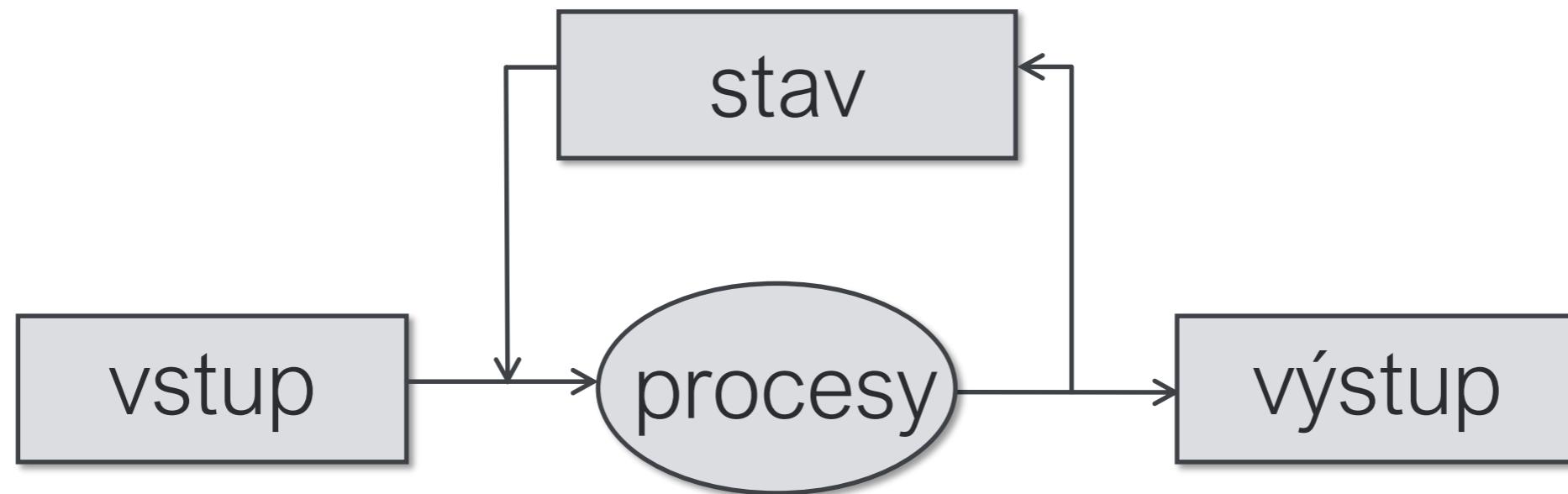


Návrh informačního systému

- S jakými daty pracujeme?
- Analýza domény, model, persistence, konzistence,
...

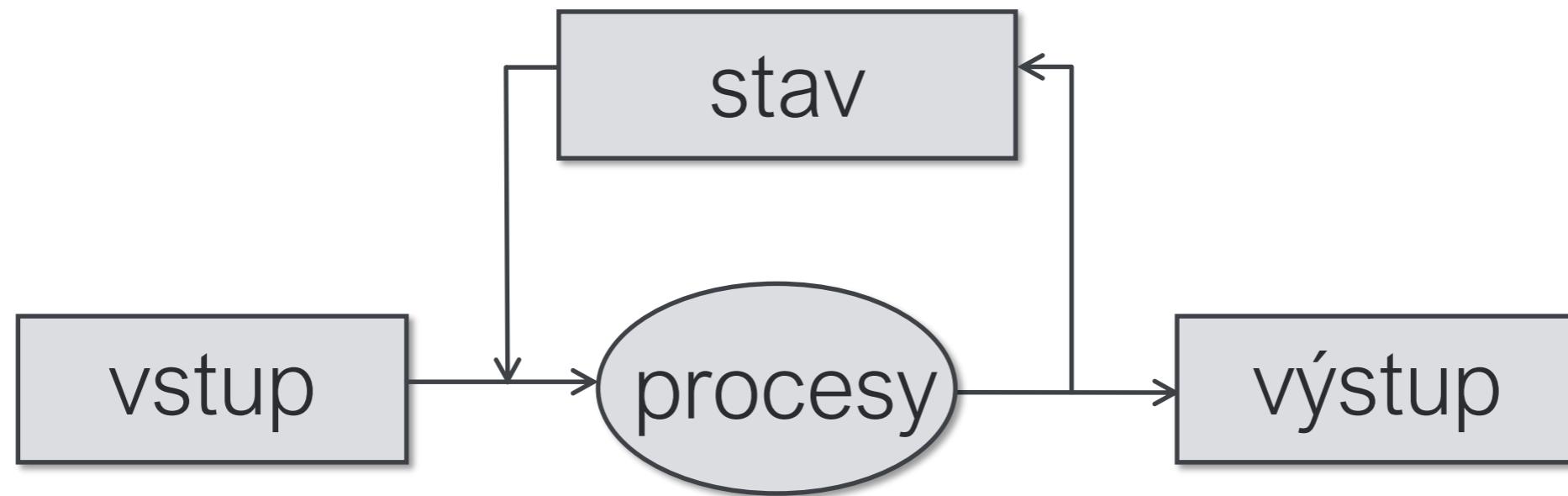


Návrh informačního systému



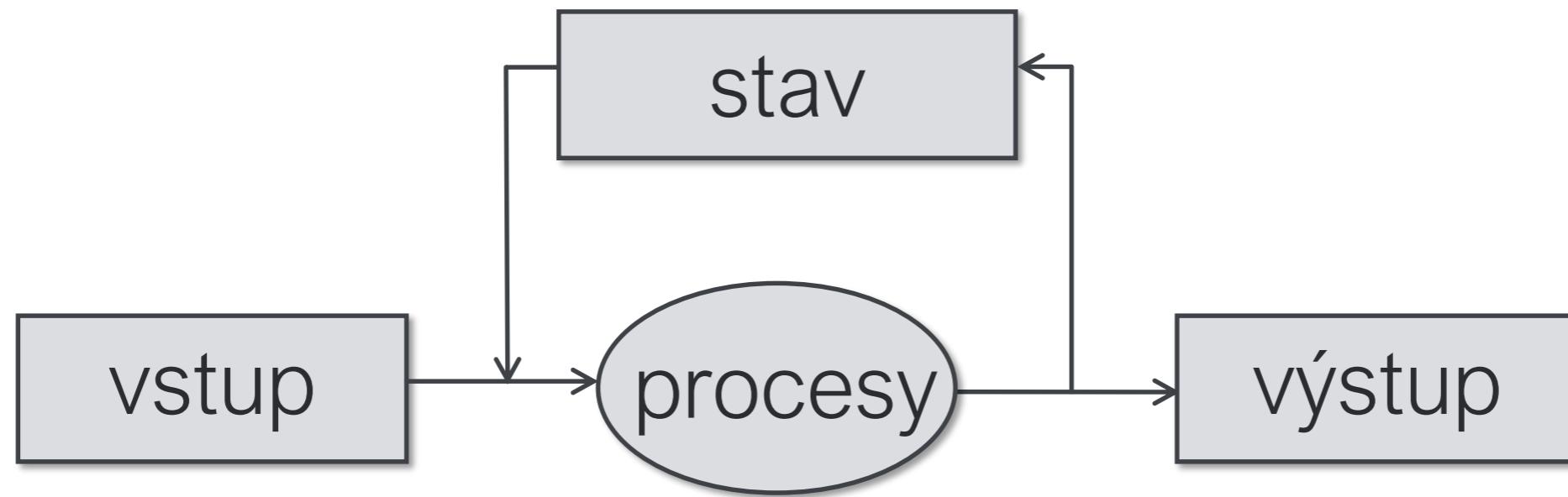
- **Jaké jsou k dispozici vstupy?**
- Jak se informace pořizují, kdo je zadává?

Návrh informačního systému



- **Jak mají vypadat výstupy?**
- Aby to odpovídalo účelu systému?

Návrh informačního systému



- **Jak je třeba data transformovat?**
- Jaké jsou procesy a postupy v cílové doméně?

Netypované a typované informační systémy

Netypovaný informační systém

- Data bez popisu jejich struktury
- Prostředky pro vytvoření obálky a vkládání libovolného počtu položek vesměs různých typů
- Nelze popsat složitější procesy (pouze operace nad strukturou)
- Není standardizovaný vstup a výstup
- Příkladem je systém textových poznámek, myšlenkové mapy, apod.

Typovaný informační systém

- Struktura dat ((meta⁰)data) je součástí definice – meta⁽¹⁾data
- Data jsou vytvářena pouze na základě metadat
- Lze definovat standardní procesy
- Lze použít databázi se strukturovanými daty
- Lze napsat manuály a vyškolit obsluhu
- Vstupy a výstupy lze standardizovat
- Příkladem jsou klasické OLTP systémy s databází

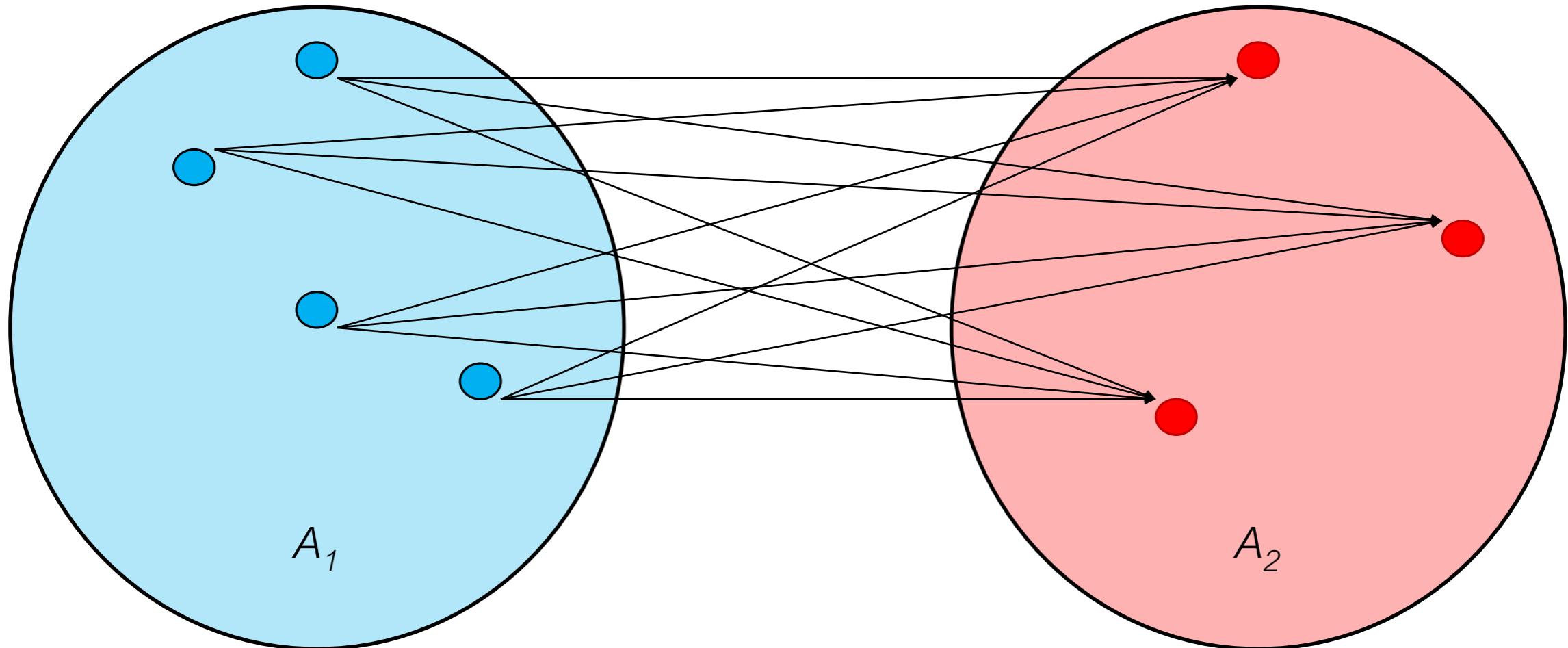
Typované informační systémy - pojmy

úroveň	popis	příklad pro relační databázový model	příklad pro objektový model
meta ⁿ data	atd. cestou odvození uvedených pojmu k základním definicím, se zvyšující se mocninou se používá obecnějších modelů vesměs matematických		
meta ⁵ data	jak vypadá množina, uspořádání atd.	uspořádání je binární relace na množině tranzitivní, reflexivní, antisymetrická	
meta ⁴ data	jak vypadá struktura a kolekce	kartézský součin je množina uspořádaných dvojic; uspořádaná multimnožina je množina obecně obsahující více stejných prvků s definovaným uspořádáním	
meta ³ data	jak vypadá databázový model – základní stavební kameny struktura, kolekce	struktura je kartézský součin dvojic; kolekce je uspořádaná multimnožina (zde a výše už to nezáleží na modelu)	
meta ² data	jak vypadá katalog – databázový model (schéma)	relace je kolekce struktur	objekt je pojmenovaná struktura s libovolnou úrovní vnoření struktur a kolekcí
meta ⁽¹⁾ data = metadata	jak vypadá výskyt - katalog	relace faktura má domény číslo, adresát a klíčem je vázána na relaci položky	objekt faktura má položku číslo a vnořenou prostou struktura adresa, vnořenou kolekci struktur položka atd.
(meta ⁰)dat a = data	výskyt	faktura číslo 1200005, Jan Novák, 1 ks telefon, 2500 Kč, celkem 2500 Kč atd. (výskyt prvku relace nebo objektu)	

Kartézský součin a uspořádaná množina – META⁴DATA

Kartézský součin $A_1 \times A_2$

prerekvizita: kurz Diskrétní matematika (IDA), UMAT



zde prvky jsou všechny dvojice vyjádřené šipkami, v tomto případě konečných množin je jich $4 \times 3 = 12$ (násobí se kardinality všech množin součinu)

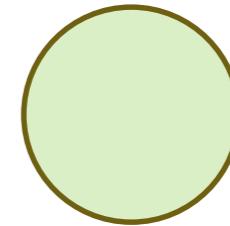
Kartézský součin

- *Uspořádaná n-tice* je množina hodnot ve tvaru $(a_1, a_2, \dots a_n)$
- *Kartézský součin* $A_1 \times A_2 \times \dots \times A_n$ je množina všech uspořádaných n-tic takových, že $a_1 \in A_1, a_2 \in A_2, \dots a_n \in A_n$. Podstatné je, že v uspořádané n-tici každá hodnota je prvkem jediné z množin definice kartézského součinu a to té, která jí indexem odpovídá

Struktura a kolekce – META³DATA

Základní typy

- Vždy musejí existovat nějaké základní datové nestrukturované typy:
 - Celočíselné
 - Reálná čísla
 - Znaky / řetězce
 - Datum/čas
 - Výčtové typy
apod.



Strukturované datové typy

- Strukturované **datové typy** = **meta¹data** (též nazývané datové struktury) popisují, jak z jednodušších datových typů (at' už základních nebo i jednodušších strukturovaných) budovat složitější.
- Existují základní dva způsoby, jak strukturované datové typy vytvářet:
 - *struktura* a
 - *kolekce*.
- Vše je definováno předem, před vznikem hodnoty

Struktura

- Uspořádané n-tice, které jsou prvky kartézského součinu jsou strukturované hodnoty vytvářené:
- *Pevným počtem pojmenovaných dílčích hodnot (dvojic jméno,hodnota) obecně různých typů*
- Jako synonymum pro uspořádanou n-tici (tedy hodnotu) je velmi často užíván termín *struktura* nebo *záznam*. Jako synonymum pro kartézský součin (tedy datový typ) budeme často používat *typ struktury* nebo *typ záznamu*.

Schéma struktury



Příklad datového typu struktura

```
structure FyzOsoba  
properties  
    UplneJmeno:    string  
    Jmeno:        string  
    Prijmeni:     string  
    DatumNarozeni: date  
end structure
```

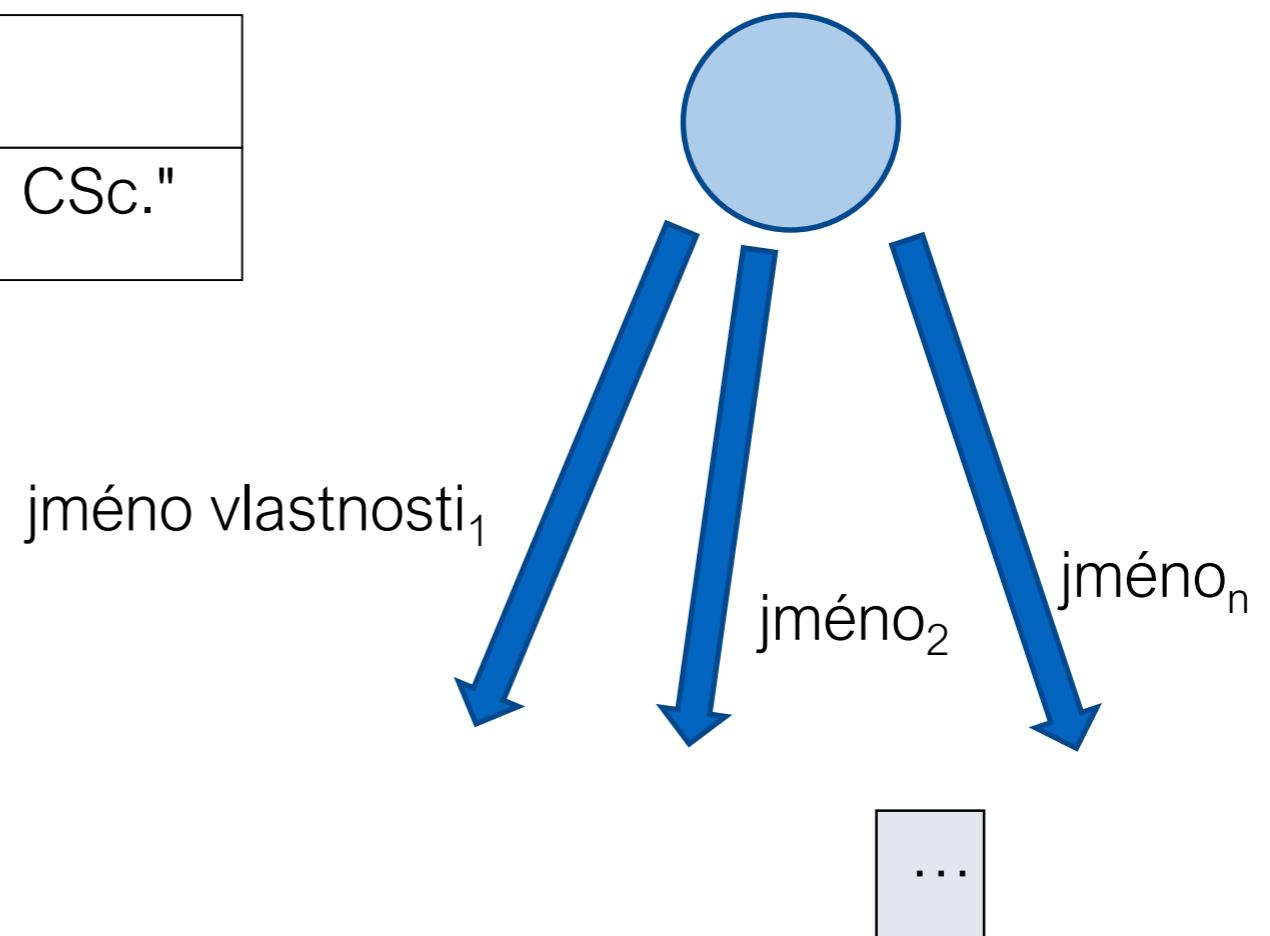
Např. Java

```
class FyzOsoba
{
    private String uplneJmeno;
    private String jmeno;
    private String prijmeni;
    private Date datumNarozeni;

}
```

Hodnota struktury

jména vlastností	hodnoty vlastností
UpIneJmeno:	"Prof. Ing. Jan Novák, CSc."
Jmeno:	"Jan"
Prijmeni:	"Novák"
DatumNarozeni:	24.5.1954



Kolekce

- *Kolekce* (synonyma jsou *řetězec*, *posloupnost*, *seznam*, *soubor*) je, na rozdíl od struktur, tvořena
- *Předem neomezeným počtem hodnot stejných datových typů.*

Typy kolekcí

- **Opakování prvků:** *Množina* obsahuje obvykle každý prvek pouze jednou. Pokud je povoleno, aby daný prvek byl v množině vícekrát, mluvíme o *multimnožině*
- **Pořadí prvků:** kolekce může být *uspořádaná* (záleží na pořadí prvků) nebo *neuspořádaná*
- *Tradiční seznam je uspořádanou multimnožinou*
- Obecně lze vytvářet kolekce s prvky libovolných datových typů.
 - Časté omezení je vytvářet pouze *kolekce s prvky datového typu struktura*

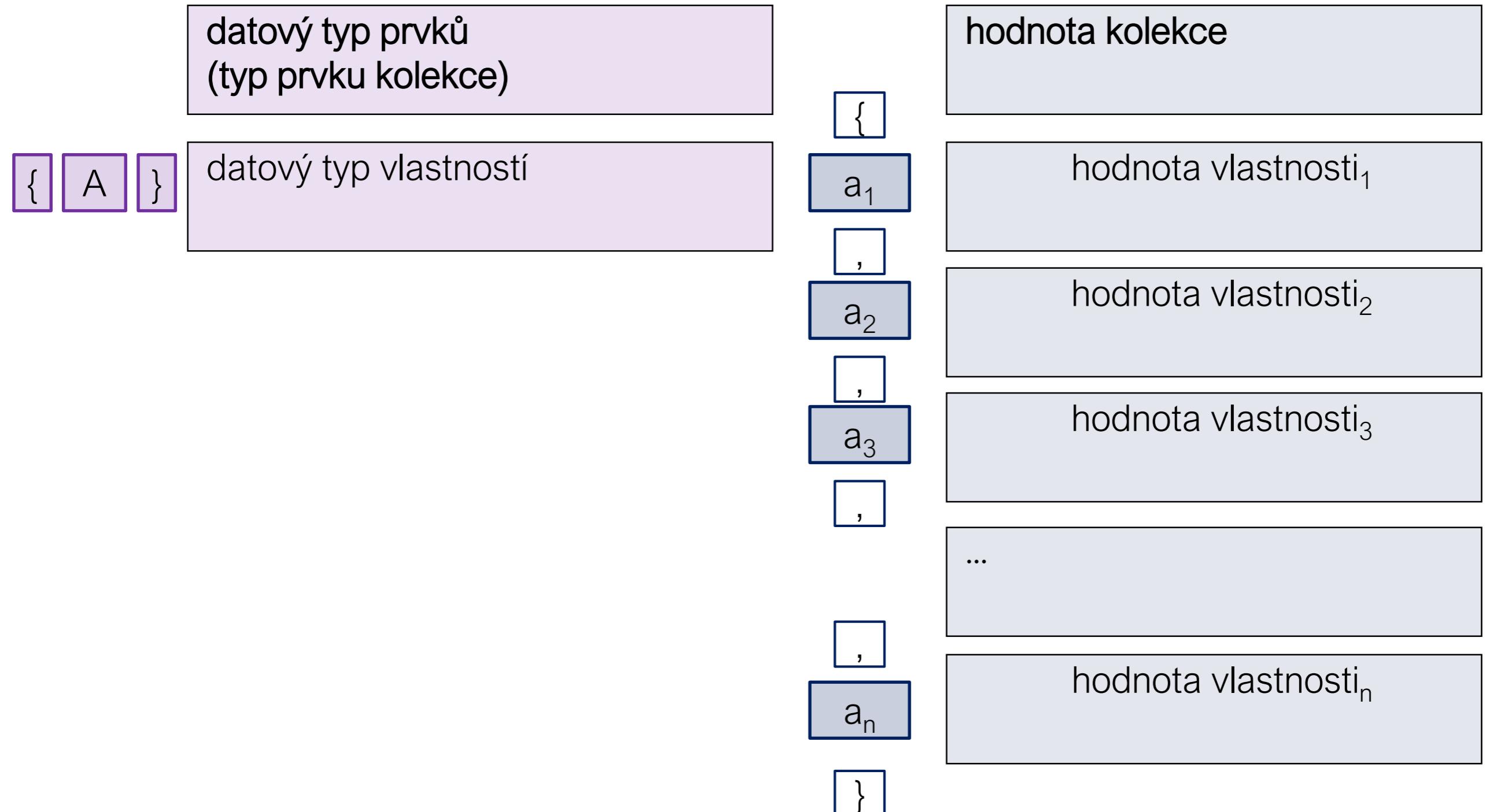
Operace nad množinou

- Vkládání prvku do kolekce (*add*),
- Získání prvku z kolekce (*item*),
- Určení počtu prvků kolekce (*count*) a
- Rušení prvku kolekce (*remove*)
- další
- provádění operací nad všemi prvky (*forall*)

Vlastnosti kolekce

- *Kurzor (iterator)*, což je ukazovátko do kolekce, kterým lze posunovat oběma směry a nastavovat je do různých pozic v kolekci podle různých kriterií.
- Protože v průběhu práce s kurzorem se může kolekce měnit co do obsahu i počtu prvků, dělíme kurzory na *stabilní*, které na tuto skutečnost neberou zřetel a *nestabilní*, které reflektují změny
- Nad kolekcí může existovat jedno nebo více definovaných *uspořádání* jejich prvků podle různých klíčů.

Schéma kolekce



Příklad datového typu kolekce struktur

collection FyzickeOsoby of

structure FyzOsoba

properties

UplneJmeno: string

Jmeno: string

Prijmeni: string

DatumNarozeni: date

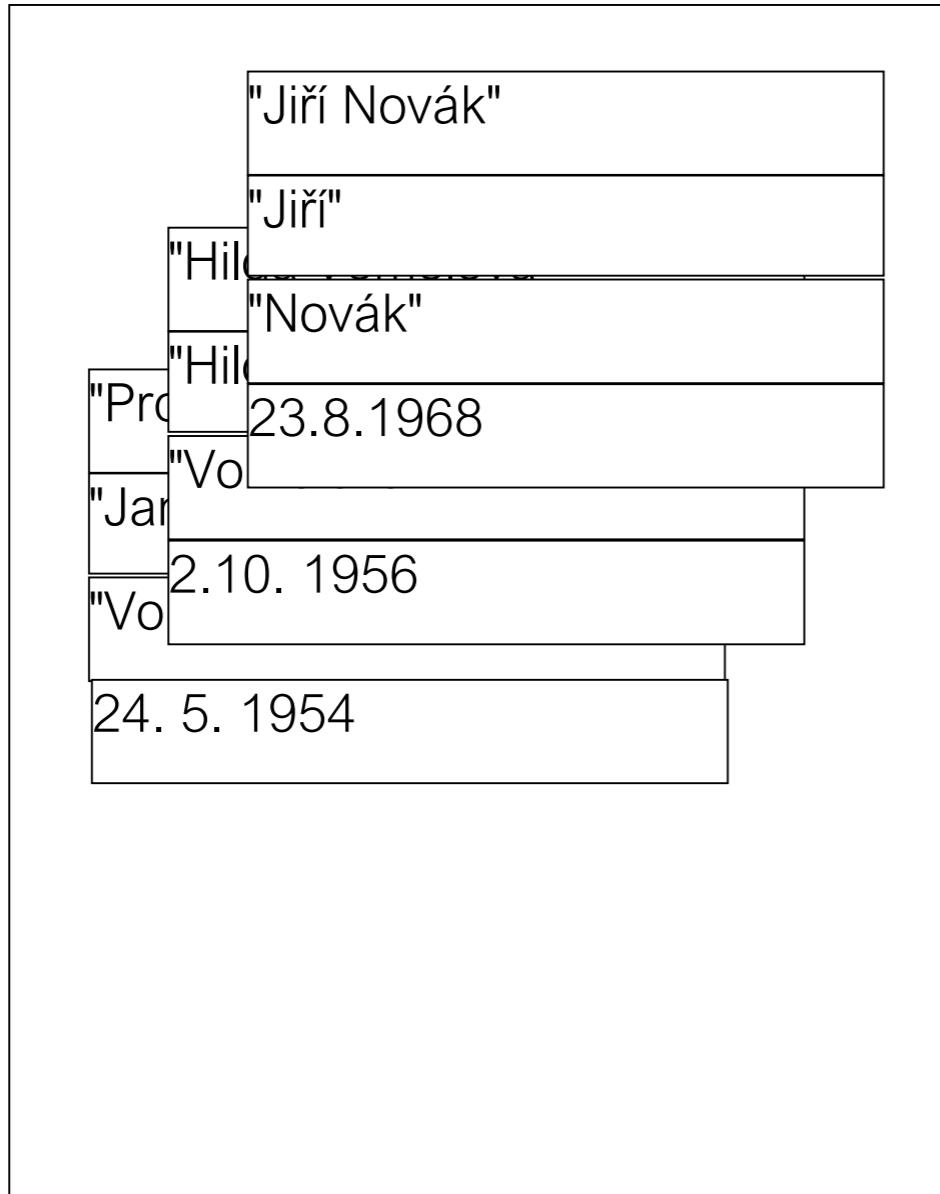
end structure

Např. Java

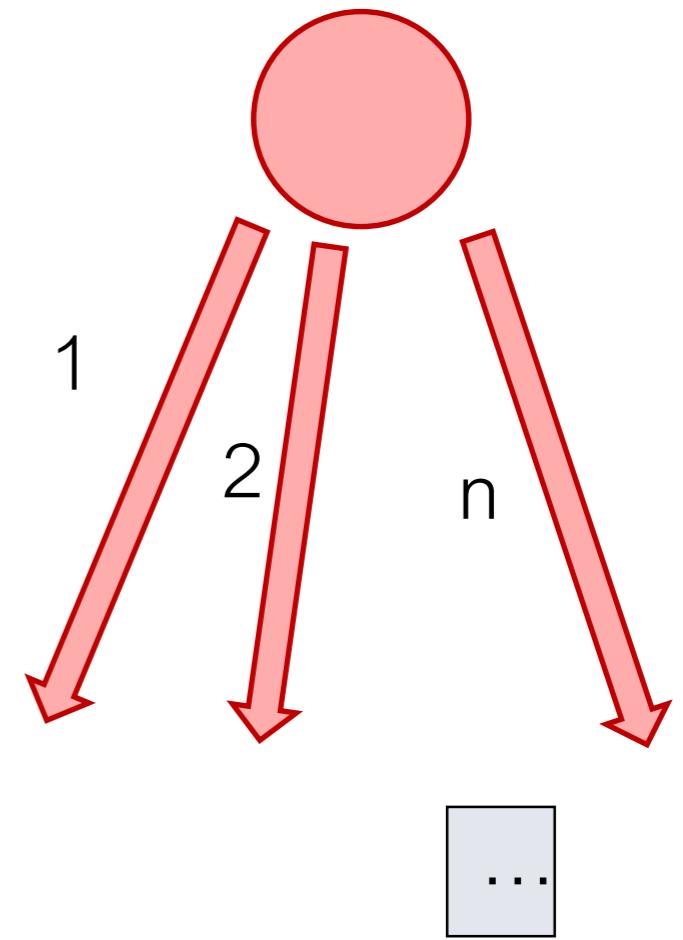
- Collection<FyzOsoba> obecnaKolekce;
 - List<FyzOsoba> seznamOsob;
 - Set<FyzOsoba> mnozinaOsob;
- ...

Hodnota kolekce

{



}



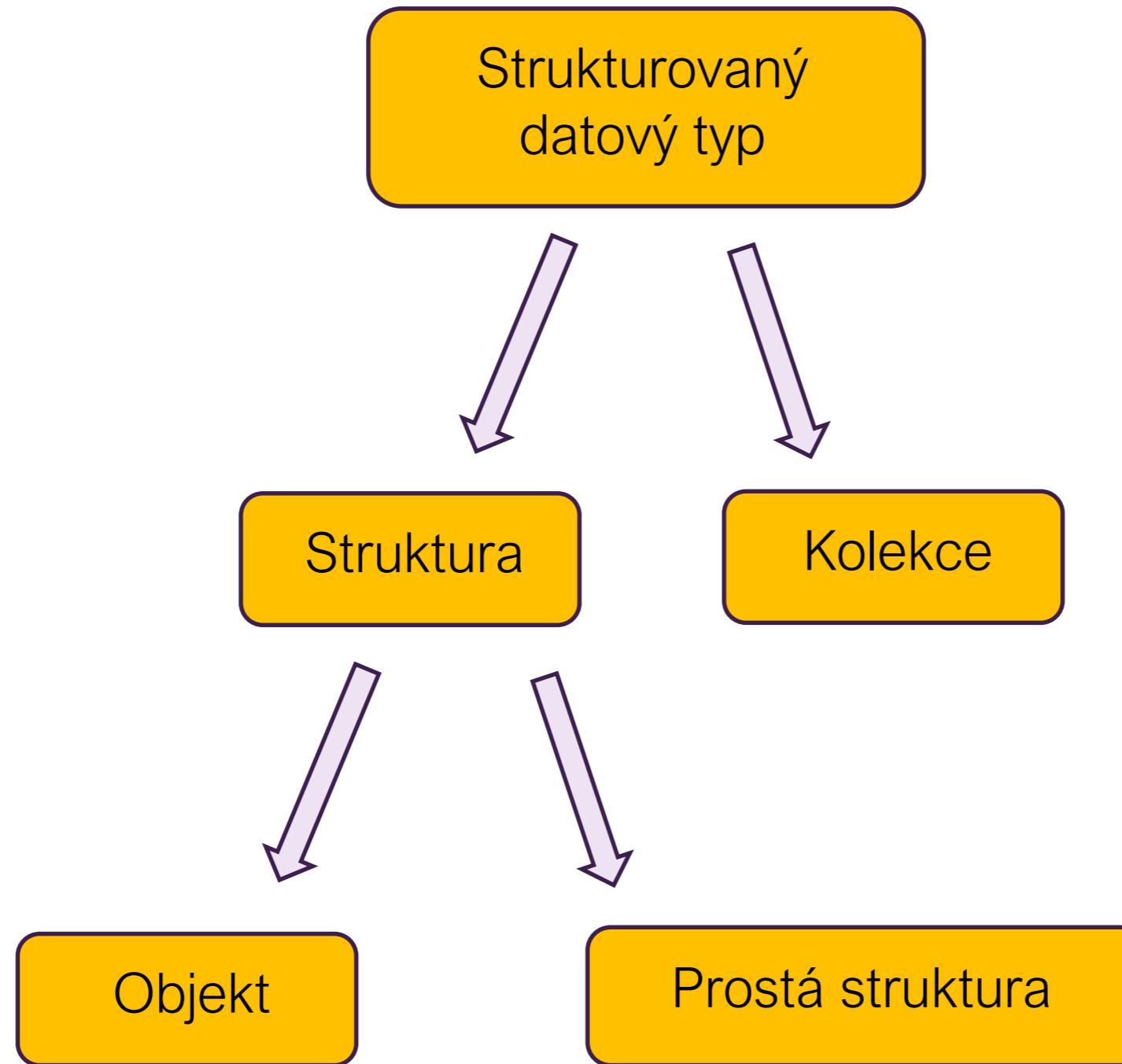
Agregáty

- Vlastnostmi kolekce jsou nejčastěji *agregáty (agregované hodnoty)*, což jsou hodnoty statisticky popisující prvky *kolekce* nejčastěji *číselných hodnot*.
- *počet prvků*,
- *maximum*,
- *minimum*,
- *součet hodnot*,
- *průměr* atd.

Objekt a prostá struktura

- *Objekt je struktura s identifikací.*
- Každému objektu v systému přiřazena *jednoznačná identifikace* nazývaná *OID (object identification)*.
- Objekt je tedy *struktura, jejíž systémovou a obvykle první vlastností je OID*. Hodnotu OID generuje databázový systém při vzniku objektu a po celou dobu činnosti ji nemění
- Tím, že má objekt OID, je *identifikovatelný* a tudíž i *odkazovatelný*. Má to za následek, že může figurovat jako *člen ve vztažích*. To struktura bez identifikace nemůže. Takovou strukturu bez OID budeme nadále nazývat *prostou strukturou*.

Strukturované datové typy



Zanořené kolekce a struktury

- Obecně lze struktury a kolekce libovolně vzájemně vnořovat (v CDL pouze pouze pojmenované typy)

```
concept TYPD [Data=Value]
```

```
...
```

```
end concept
```

```
concept TYPB/TYPYB [Data=Value]
```

```
properties
```

```
  C: integer
```

```
  D: TYPD
```

```
end concept
```

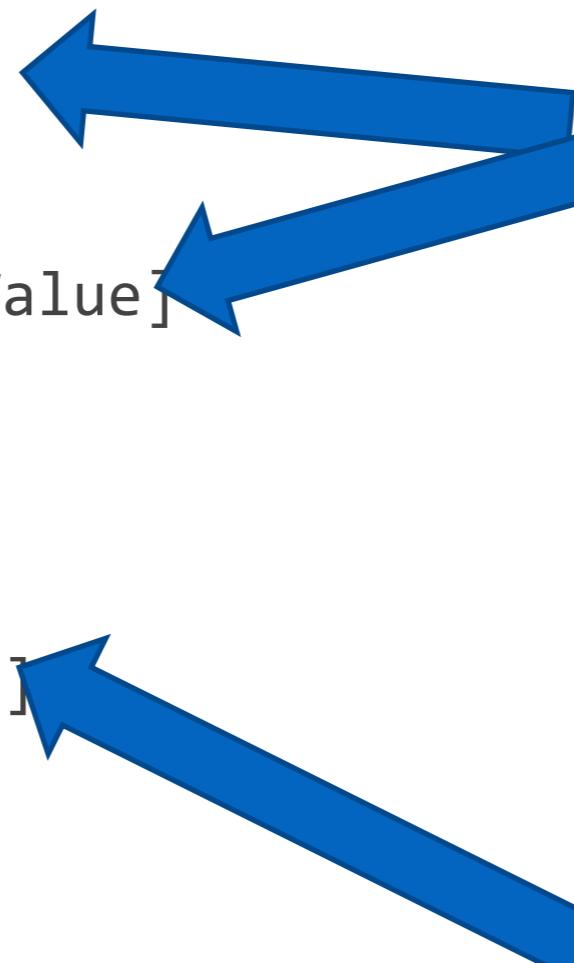
```
concept ZANORENA [Data=Ref]
```

```
properties
```

```
  A: integer
```

```
  B: TYPYB
```

```
end concept
```

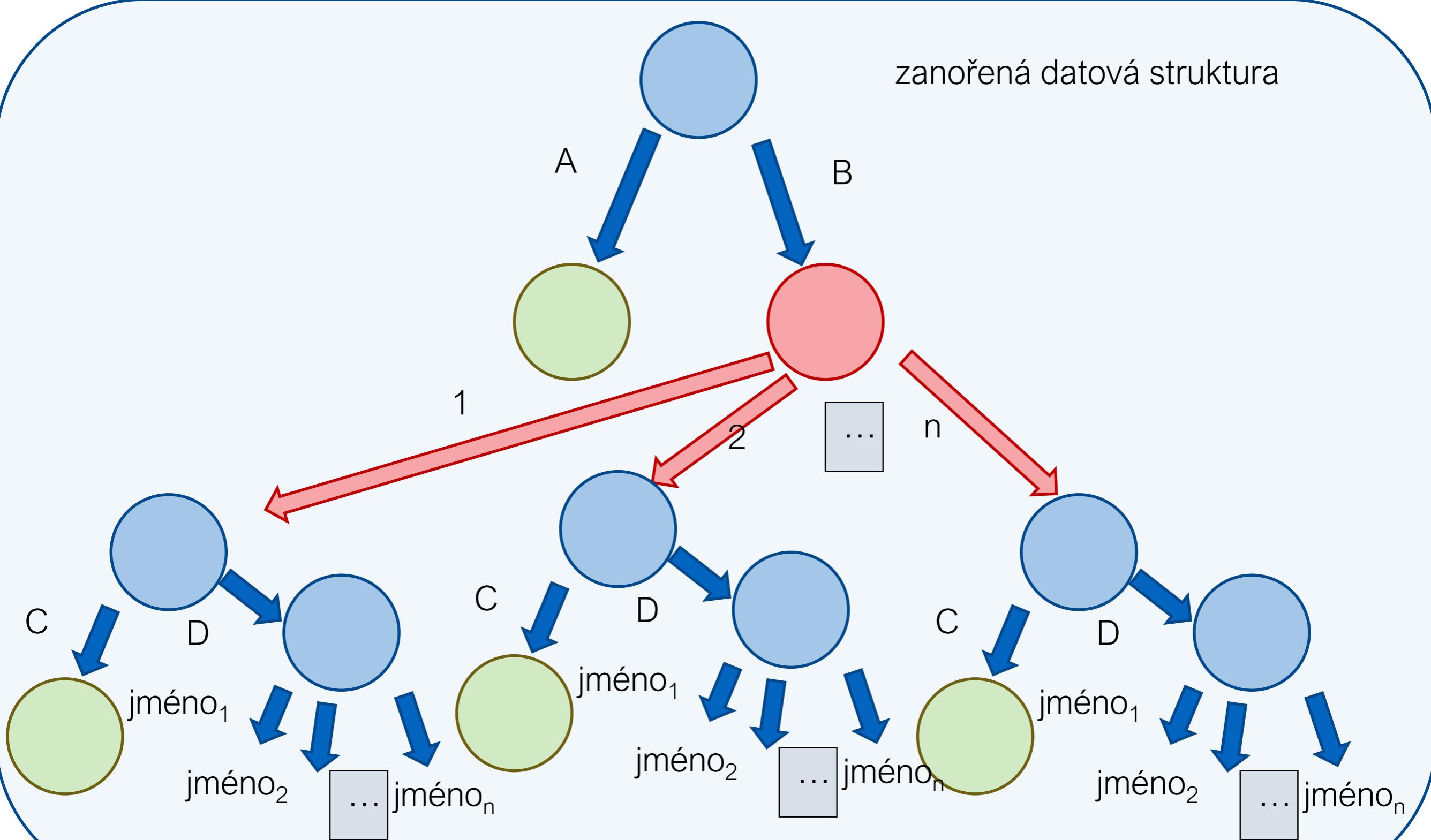


[Data= Value]
(hodnota) značí
prostou strukturu

Atributy v hranatých
závorkách prvků
konceptuálního schématu,
mají implicitní hodnoty

[Data = Ref] (reference)
značí objekt a je možné
tento atribut vynechat,
protože je častější

Graf hodnoty zanořených typů



Datové modelování

Databázové modely

- Modely, které je schopen interpretovat systém pro řízení databázového systému SŘBD
- Jinak též zvané *produkční modely*
- V jejich definičním jazyku musejí být zapsána metadata pro všechny datové struktury uložené v databázi
- Prozatím budeme uvažovat *relační a objektový datový model*

Databázové modely

- Jednoduché (NoSQL)
 - Key-value (MUMPS, Redis, ...)
 - Dokumentové (MongoDB, CouchDB, ...)
 - Sloupcové (Apache HBase, ...)
- Relační datový model
 - Mnoho implementací
- Objektový datový model
 - Objektově-relační mapování (ORM)
- Grafové
 - Grafové databáze (Neo4J, OrientDB, ...)
 - Sémantická úložiště (sémantický web, RDF)

Konceptuální modely

- Slouží pro komunikaci mezi návrháři, případně se zákazníky
- Jsou formálně přesné a *převoditelné* na produkční modely
- Často jsou grafické pro větší přehlednost
- Budeme krátce uvažovat *diagram tříd (UML)* a *E-R diagram* a zejména **CDL**.

Transformace mezi datovými modely

- Slouží nejčastěji pro transformaci konceptuálních modelů na produkční.
- Transformace je tím složitější, čím jsou modely více sémanticky odlišné
- Nejčastěji se uvažuje ***transformace E-R diagramu na relační datový model***

DATABÁZOVÉ MODELY - META²DATA

Relační model dat

- Relace v relačním modelu je *kolekcí struktur*, přičemž datové typy vlastností jsou *jednoduché* (tedy především *ne odkazy/vztahy*)
- Srovnej: *Podmnožina kartézského součinu*

collection of

**structure
properties**

jméno vlastnosti₁: jednoduchý datový typ₁

jméno vlastnosti₂: jednoduchý datový typ₂

...

jméno vlastnosti_n: jednoduchý datový typ_n

end structure

Vztahy

- Relační model dat vztahy **přímo neobsahuje**
 - (Neplést s referenční integritou)
 - Vytváří se až v okamžiku dotazování (JOIN apod.)

Objektový model dat

Vztahy

- Umožňují odkazovat z jedné (strukturované) hodnoty (vlastníka) jinou (člen)
- Musí existovat datový typ *jednoznačné identifikující (odkazující) strukturovanou hodnotu* (disková adresa, OID)
- Vztah je definován prvkem vlastníka typu odkaz (reference) a členem, který je hodnotou odkazu identifikován.

Odkazované struktury (objekty)

concept VLASTNIK

properties

A: integer

B: CLEN

end concept

concept CLEN [Data=Ref]

properties

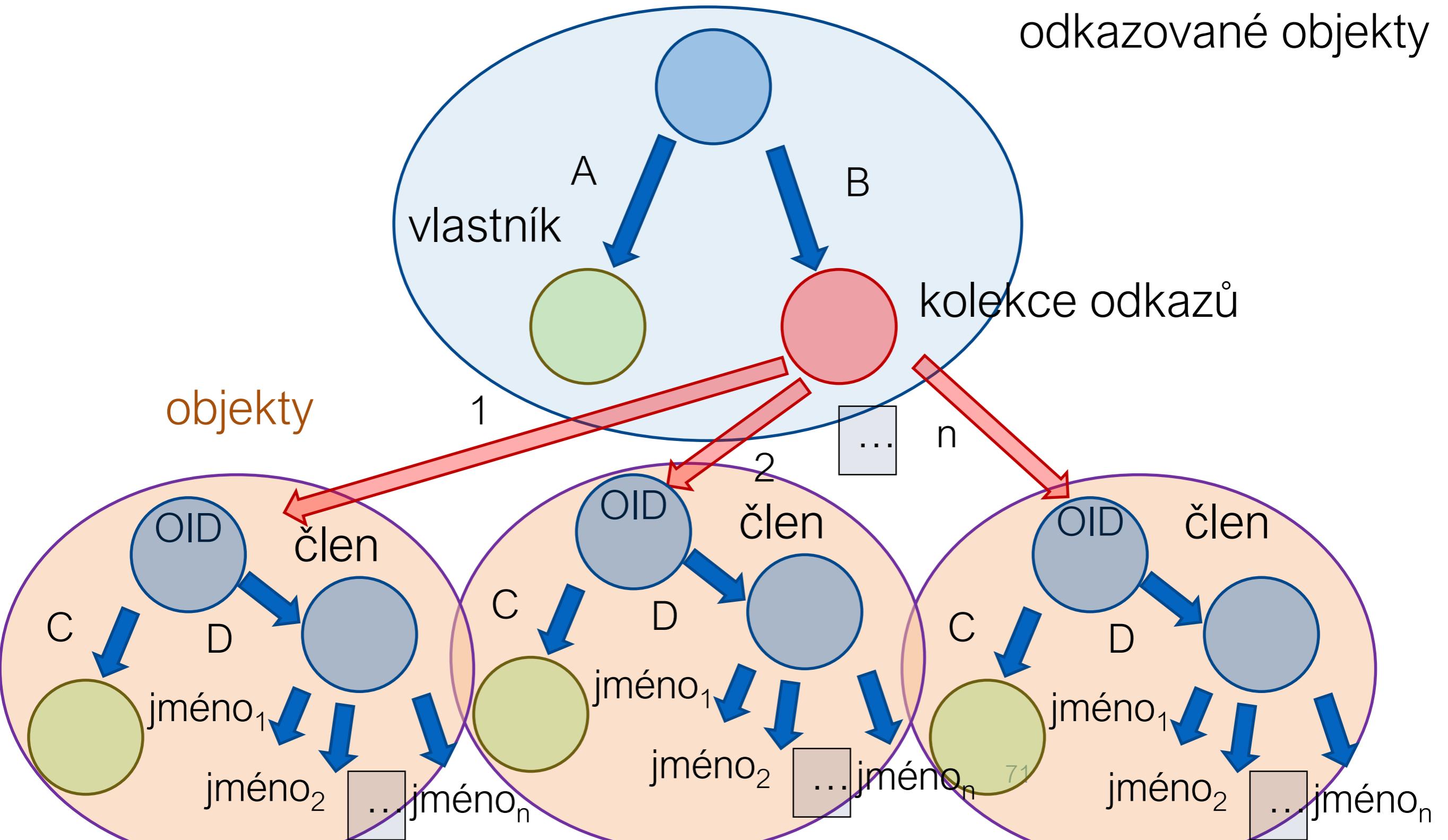
C: integer

D: structure

...

end concept

Graf hodnoty odkazovaných typů



Objektový model dat

- Základní typy + datový typ OID
- Objekt je vždy *strukturou na nejvyšší úrovni*
- Dva druhy neomezeně zanořených struktur
 - kolekce (někdy omezení pouze na kolekce prostých struktur a OID)
 - prosté struktury (ostatní)
- Další vlastnosti – dědičnost, role apod.

Unikátní výskyt v kolekci

- Častým požadavkem na kolekce je *jedinečná hodnota vlastnosti v kolekci*.
- Podle této hodnoty se také často vyhledává.
- Jedinečnost hodnoty v kolekci budeme v definici vyznačovat atributem Key. Půjde o atribut booleovský s implicitní hodnotou False.

Příklad zápisu klíče

```
concept Obcan/Obcane  
properties  
    Oznaceni:      string  
    Adresat:       string  
    RodneCislo:    string [Key]  
    RodnePrijmeni: string  
    Prezdivka:     string  
    Pozice:        string  
end concept
```

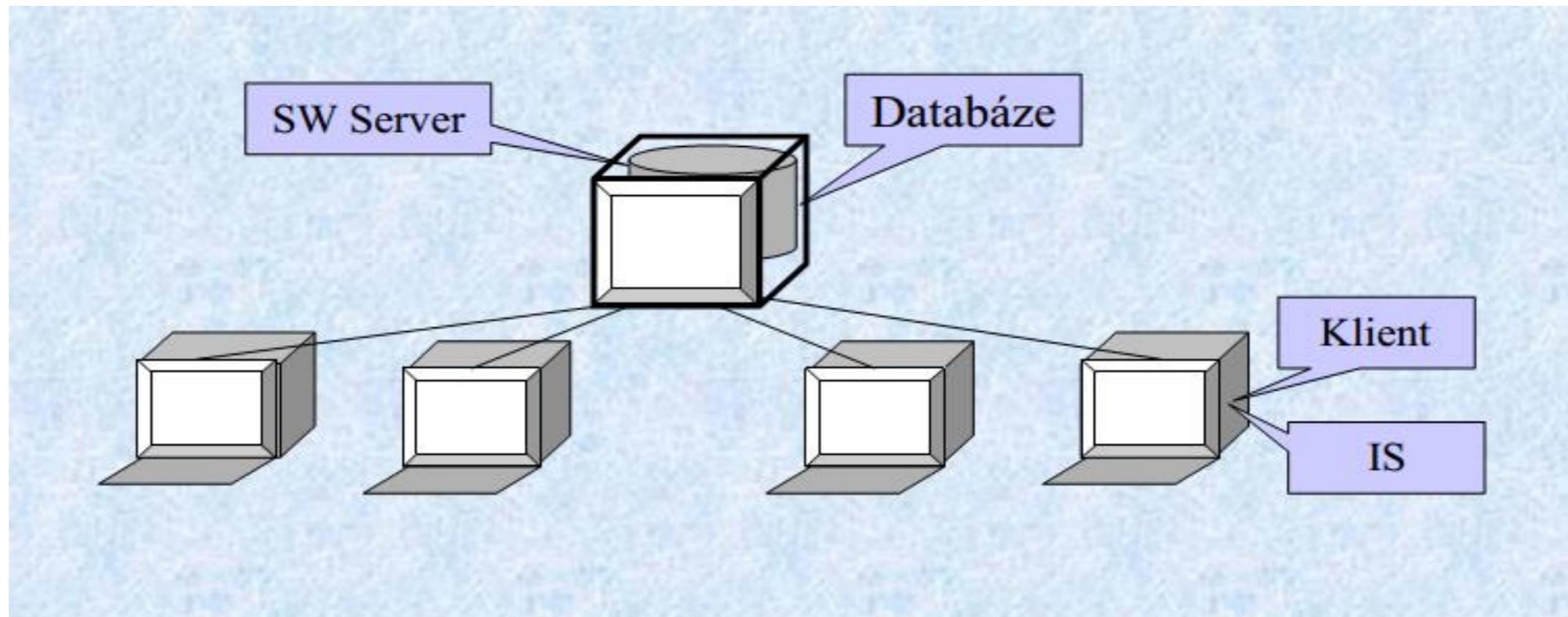
Shrnutí

- Strukturovaná data modelujeme jako **kolekce a struktury**
- Metadata = meta1data – popisují data, se kterými systém pracuje
- Konceptuální modely
 - Výjádření metadat pro účely modelování
 - E-R diagram, Class diagram, CDL
- Produkční (databázové) modely
 - Definice metadat pro konkrétní databázi
 - Relační model, objektový model
 - Alternativní modely – dokumentové, grafové, ...

Architektury a implementace informačních systémů

Architektura klient-server (dvouvrstvá)

- Užity dva druhy oddělených výpočetních systémů *klient* a *server*.
- *Tloušťka* klienta odpovídá jeho "*inteligenci*"



Architektura klient-server

- Na nižší úrovni použita síťová komunikace standardizovaná protokoly Internetu TCP/IP
- Chování klienta a serveru rovněž standardizováno
 - Server specializovaný pro databázové dotazy
 - Po síti se přenášejí pouze dotazy a výsledky
- Ve vyšších vrstvách aplikacích protokolů se nejčastěji komunikuje *serializovanými daty*, případně v SQL

V současnosti nejvíce užívaná architektura – oblast našeho zájmu.

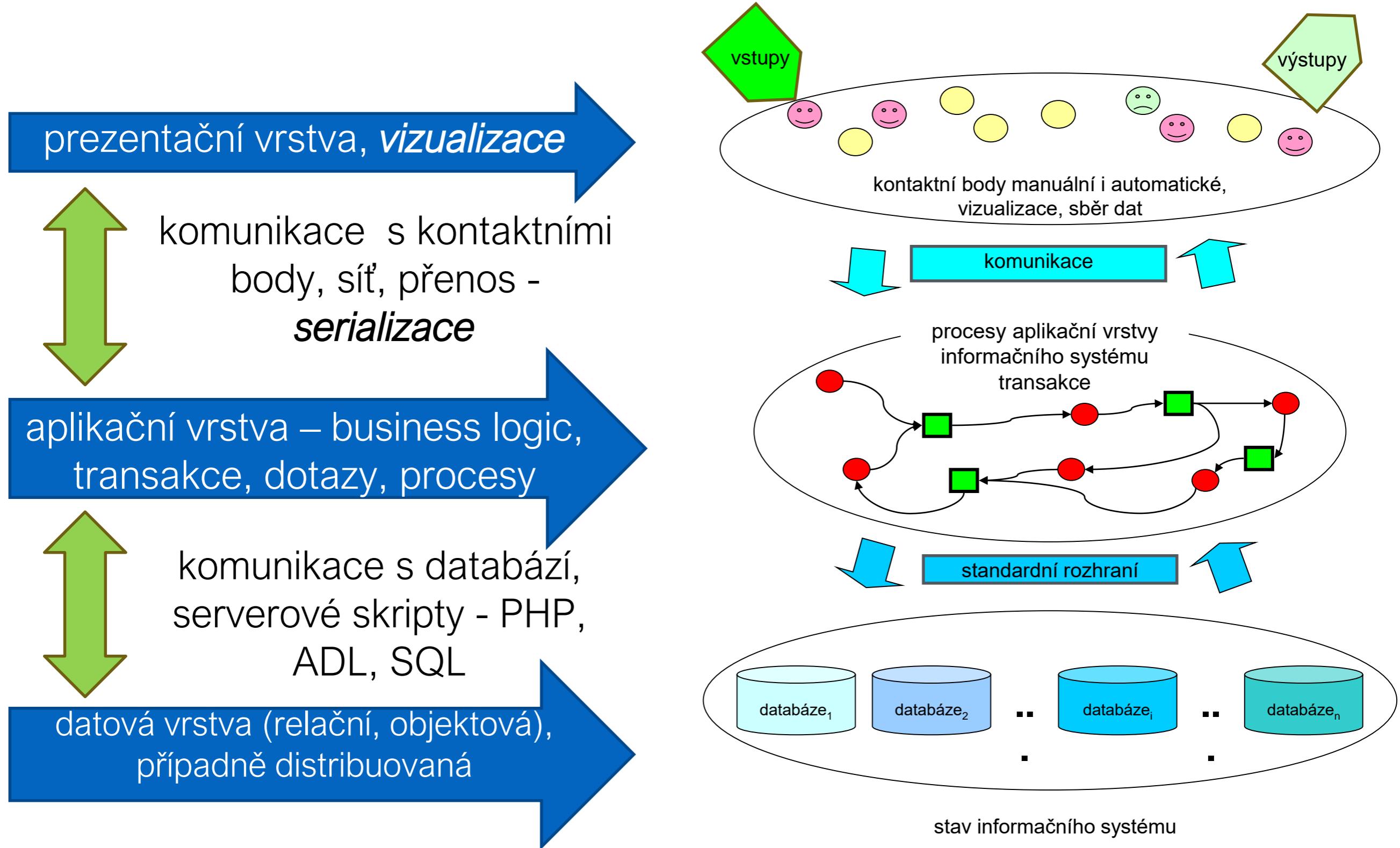
Třívrstvá architektura

- ***Třívrstvá architektura (three-tier architecture)***
- ***Prezentační vrstva*** – **vizualizuje** informace pro uživatele, většinou formou grafického uživatelského rozhraní, může kontrolovat zadávané vstupy, neobsahuje však zpracování dat
- ***Aplikační vrstva*** – jádro aplikace, logika a funkce, výpočty a zpracování dat
- ***Datová vrstva*** – nejčastěji databáze. Může zde být ale také (sítový) souborový systém, webová služba nebo jiná aplikace.

Terminologická odbočka

- **Tier** – fyzická vrstva – jednotka nasazení (deployment)
 - Fyzické členění systému – klient, aplikační server, DB server
 - Tomu odpovídá volba technologií pro realizaci jednotlivých částí
- **Layer** – logická vrstva – jednotka organizace kódu
 - Obvykle řešena v rámci aplikacní vrstvy
 - *Data layer* – část řešící komunikaci s databází
 - *Business layer* – část implementující logiku aplikace
 - *Presentation layer* – komunikace s klientem

Schéma třívrstvé architektury



Technologie třívrstvé architektury

- *Prezentační vrstva* - vizualizace prováděná klientem HTTP
 - Tlustší nebo tenčí klient
- *Komunikace mezi prezentační a aplikační vrstvou*
 - HTTP, přenos dat – serializace
 - Aplikační rozhraní – REST, JAX-RS, SOAP, GraphQL
- *Aplikační vrstva* – aplikační logika (business logic)
 - Java, .NET, PHP, JavaScript, Python, Ruby, ...
 - Různá rámcová řešení (frameworky)

Technologie třívrstvé architektury

- *Komunikace mezi aplikační a datovou vrstvou*
 - Standardizované databázové rozhraní (SQL)
- *Datová vrstva* - relační nebo objektový databázový model
 - *Relační model* viz kurz Databázové systémy
 - *Objektový model* v kurzu Pokročilé informační systémy

Aplikační vrstva – Java

- Java EE umožňuje implementovat *monolitický IS s třívrstvou architekturou*:
 1. Databázová vrstva
 - JPA – definice entit, persistence (*PersistenceManager*)
 - Alternativně: Relační databáze (JDBC), NoSQL (MongoDB), ...
 2. Logická (business) vrstva
 - Enterprise Java Beans (EJB) nebo CDI beans
 - Dependency injection – volné propojení
 3. Prezentační vrstva
 - Webové rozhraní (JSF) nebo API (REST, JAX-RS)

Další platformy – přehled

- Java
 - Existuje mnoho možností kromě „standardní“ J EE
- .NET (Core / Framework)
 - Mnoho řešení na všech vrstvách
- PHP
 - Různé frameworky, důraz na webovou vrstvu
- JavaScript
 - Node.js + frameworky, důraz na web a mikroslužby
- Python, Ruby, ... - podobné principy

Distribuované architektury

- Monolitický systém (typické pro třívrstvou architekturu)
 - Vyhvídí se a nasazuje jako jeden celek
 - + snáze zvládnutelný vývoj, testování
 - - obtížnější a pomalejší nasazování nových verzí
- Distribuované architektury
 - Service-oriented architecture (SOA)
 - Mikroservices (mikroslužby)

Mikroslužby (Microservices)

Architektura orientovaná na služby

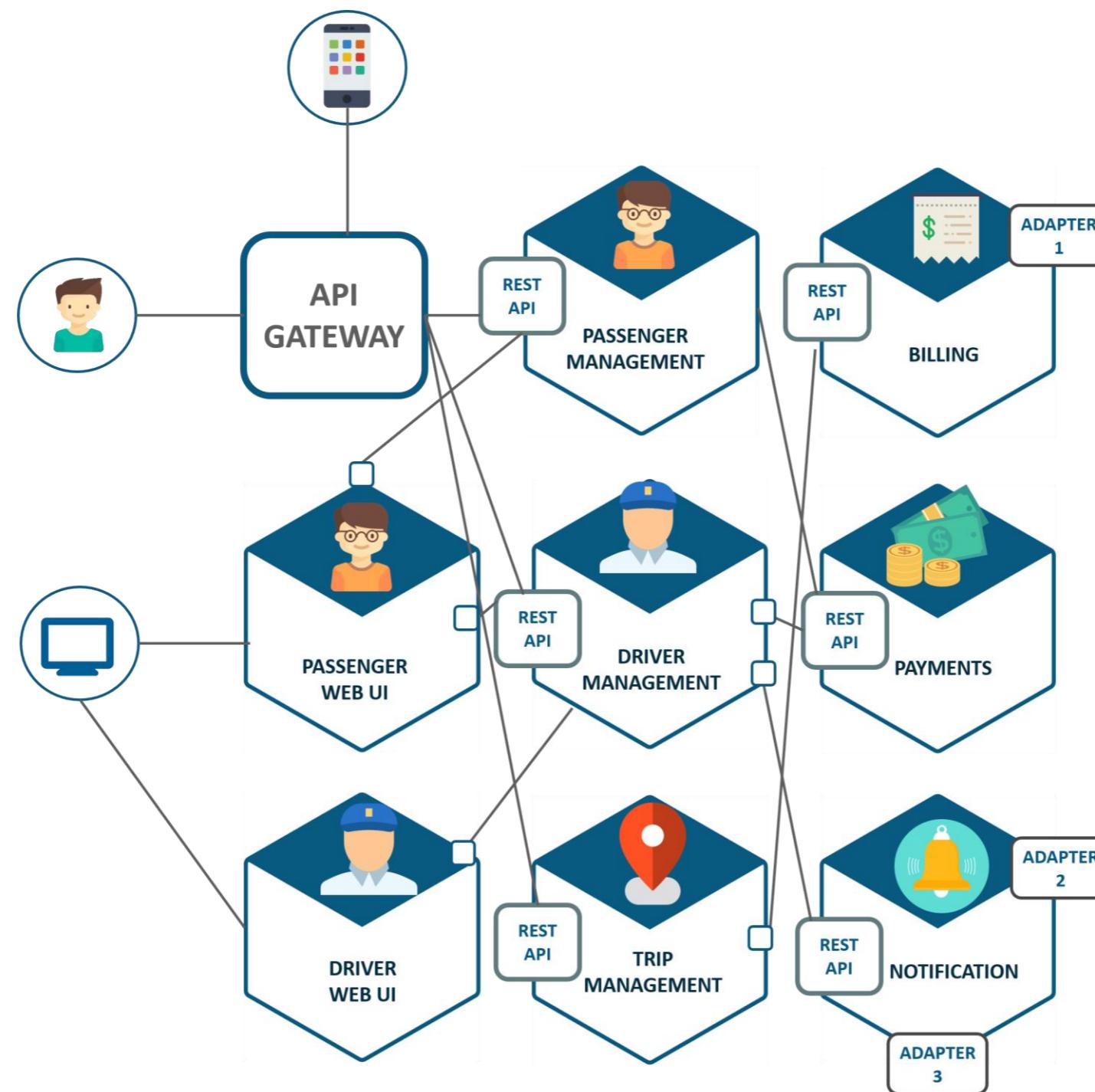
Monolitická architektura

- Jedna aplikace
 - Jedna databáze, webové (aplikační) rozhraní
 - Business moduly – např. objednávky, doprava, sklad, ...
- Výhody
 - Jednotná technologie, sdílený popis dat
 - Testovatelnost
 - Rychlé nasazení – jeden balík
- Nevýhody
 - Rozměry aplikace mohou přerůst únosnou mez
 - Neumožňuje rychlé aktualizace částí, reakce na problémy
 - Pokud použité technologie zastarají, přepsání je téměř nemožné

Mikroslužby

- Aplikace je rozdělena na malé části
 - Vlastní databáze (nepřístupná vně)
 - Business logika
 - Aplikační rozhraní (REST)
- Typicky malý tým vývojářů na každou část (2 pizzas rule)
- Výhody
 - Technologická nezávislost
 - Snadné aktualizace, kontinuální vývoj
- Nevýhody
 - Testovatelnost – závislosti na dalších službách
 - Režie komunikace, riziko nekompatibility, řetězové selhání, ...

Mikroslužby (příklad: Uber)



Vlastnosti mikroslužby

- Vnější API
 - Dostatečně obecné – reprezentuje logiku, ne např. schéma databáze (která je skrytá)
- Externí konfigurace
- Logování
- Vzdálené sledování
 - Telemetrie – metriky (počty volání apod.), výjimky
 - Sledování živosti (Health check)

Implementace mikroslužeb

- V čemkoliv – spojovacím bodem je pouze API
- Node.js (+ express + MongoDB)
 - Populární rychlé řešení
- Java
 - Spring Boot
 - Ultralehké frameworky
Např. Spark - <https://github.com/perwendel/spark>
 - Microprofile

Otázky?



Pokročilé informační systémy

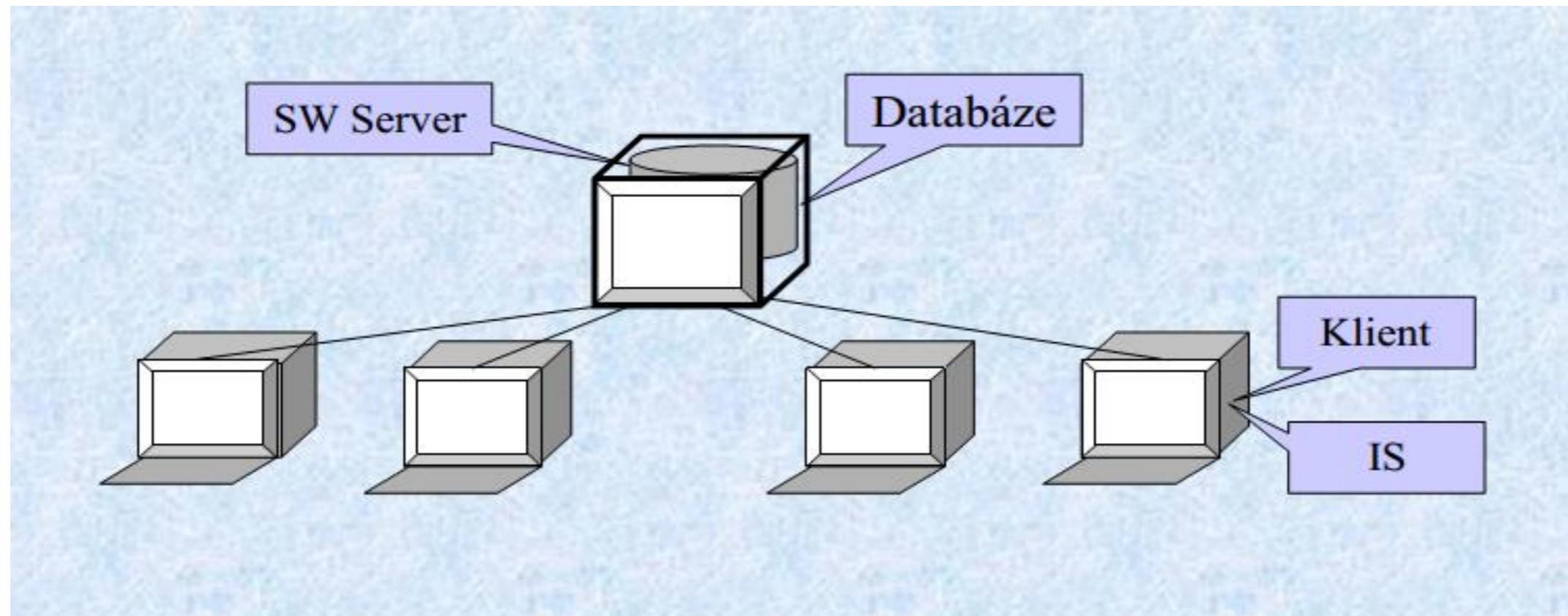
Backend a platforma Jakarta EE

Ing. Radek Burget, Ph.D.
burgetr@fit.vutbr.cz

Architektury a implementace informačních systémů

Architektura klient-server (dvouvrstvá)

- Užity dva druhy oddělených výpočetních systémů *klient* a *server*.
- *Tloušťka* klienta odpovídá jeho "*inteligenci*"



Architektura klient-server

- Na nižší úrovni použita síťová komunikace standardizovaná protokoly Internetu TCP/IP
- Chování klienta a serveru rovněž standardizováno
 - Server specializovaný pro databázové dotazy
 - Po síti se přenášejí pouze dotazy a výsledky
- Ve vyšších vrstvách aplikacích protokolů se nejčastěji komunikuje *serializovanými daty*, případně v SQL

V současnosti nejvíce užívaná architektura – oblast našeho zájmu.

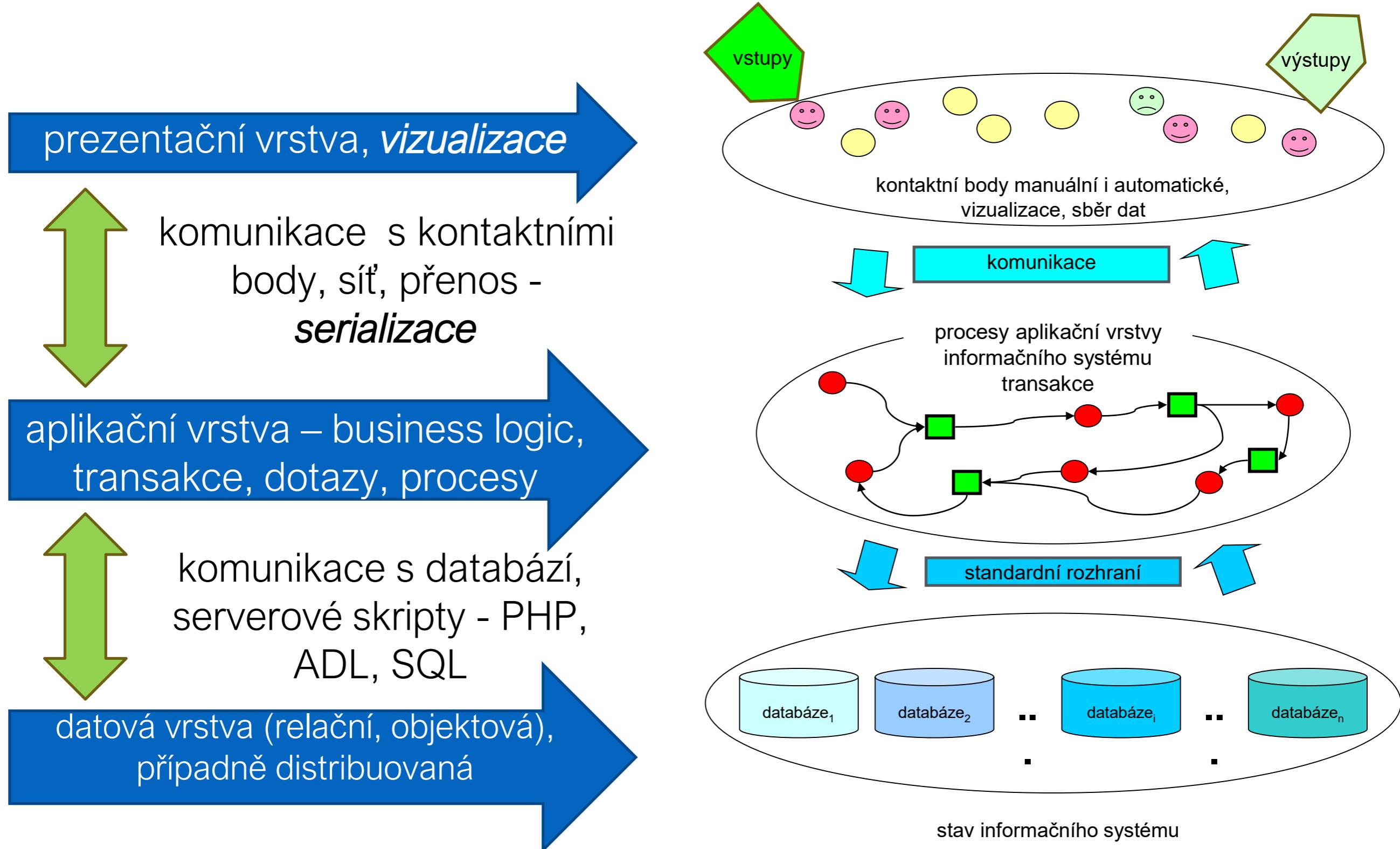
Třívrstvá architektura

- ***Třívrstvá architektura (three-tier architecture)***
- ***Prezentační vrstva*** – **vizualizuje** informace pro uživatele, většinou formou grafického uživatelského rozhraní, může kontrolovat zadávané vstupy, neobsahuje však zpracování dat
- ***Aplikační vrstva*** – jádro aplikace, logika a funkce, výpočty a zpracování dat
- ***Datová vrstva*** – nejčastěji databáze. Může zde být ale také (sítový) souborový systém, webová služba nebo jiná aplikace.

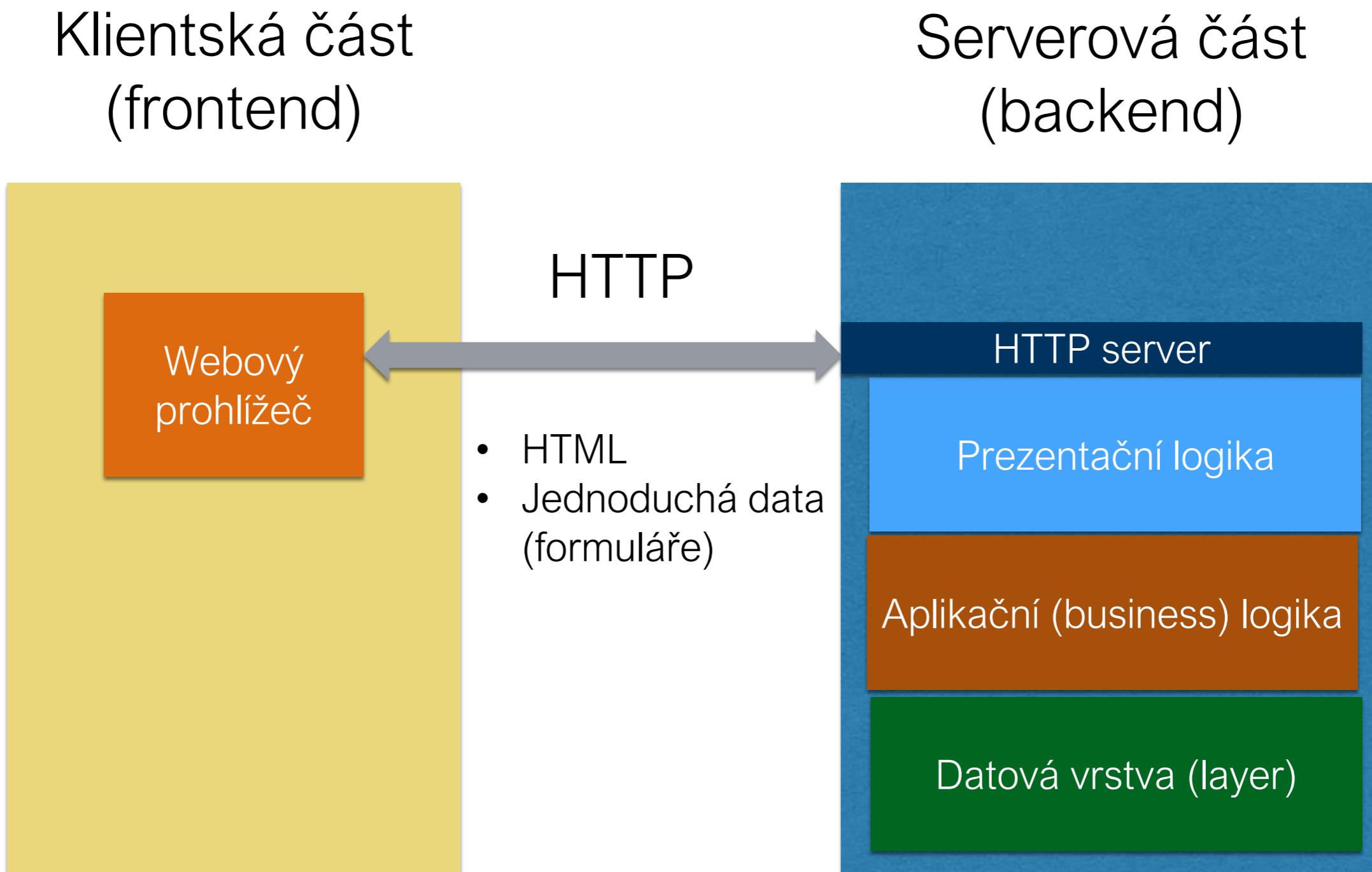
Terminologická odbočka

- **Tier** – fyzická vrstva – jednotka nasazení (deployment)
 - Fyzické členění systému – klient, aplikační server, DB server
 - Tomu odpovídá volba technologií pro realizaci jednotlivých částí
- **Layer** – logická vrstva – jednotka organizace kódu
 - Obvykle řešena v rámci aplikacní vrstvy
 - *Data layer* – část řešící komunikaci s databází
 - *Business layer* – část implementující logiku aplikace
 - *Presentation layer* – komunikace s klientem

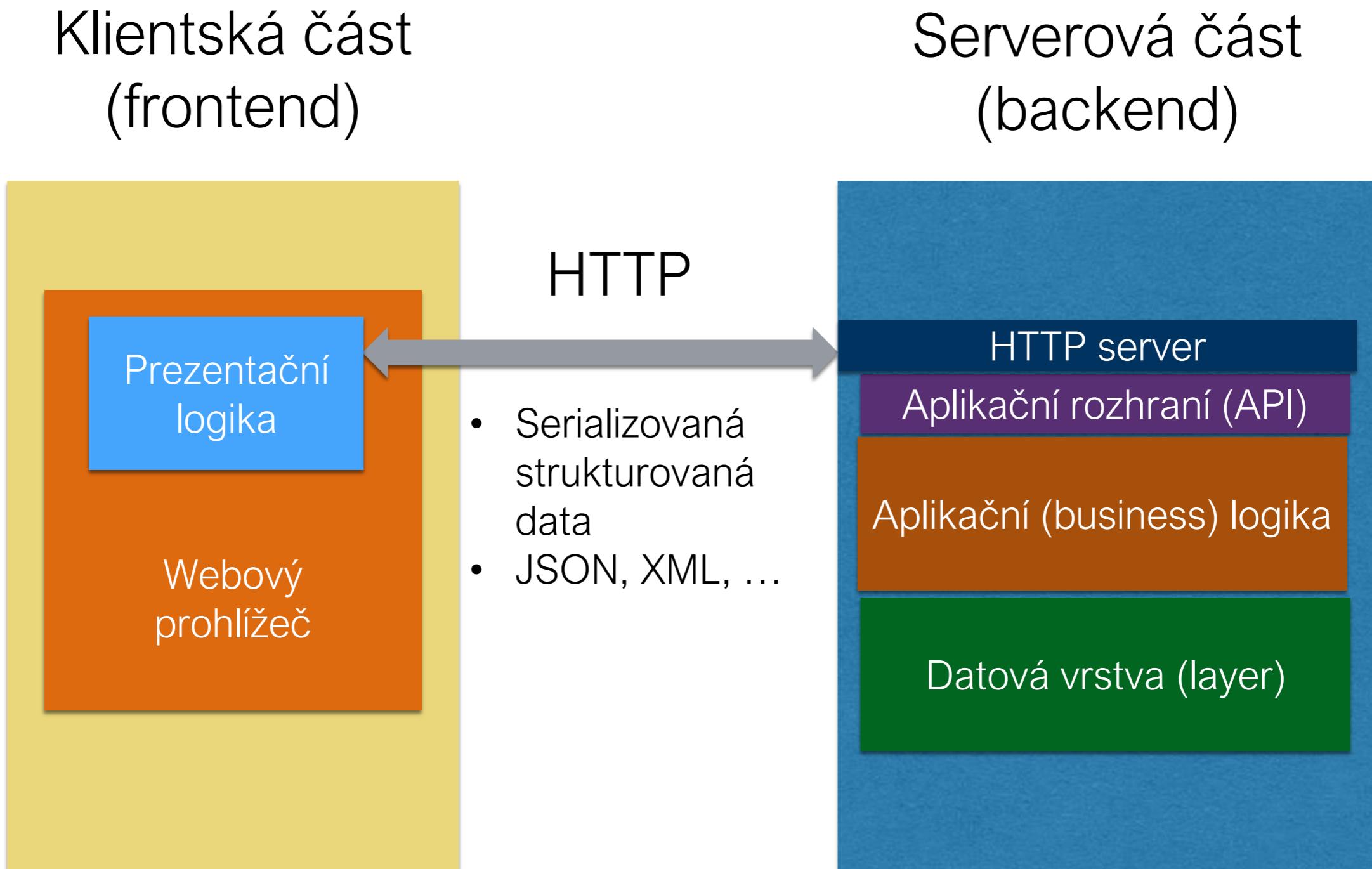
Schéma třívrstvé architektury



Webový IS



Webový IS s aplikačním rozhraním



Technologie třívrstvé architektury

- *Klientská část*
 - HTML, CSS, JavaScript (+ Angular, React, Vue.js, ...)
- *Komunikace*
 - HTTP, **serializace**
 - Aplikační rozhraní – REST, JAX-RS, SOAP, GraphQL
- *Serverová část – jednotlivé vrstvy*
 - Java, .NET, PHP, JavaScript, Python, Ruby, ...
 - Různá rámcová řešení (frameworky)

Technologie třívrstvé architektury

- *Komunikace mezi aplikační a datovou vrstvou*
 - Standardizované databázové rozhraní (SQL)
- *Datová vrstva* - relační nebo objektový databázový model
 - *Relační model* viz kurz Databázové systémy
 - *Objektový model* v kurzu Pokročilé informační systémy

Aplikační vrstva – Java

- Java EE umožňuje implementovat *monolitický IS s třívrstvou architekturou*:
 1. Databázová vrstva
 - JPA – definice entit, persistence (*PersistenceManager*)
 - Alternativně: Relační databáze (JDBC), NoSQL (MongoDB), ...
 2. Logická (business) vrstva
 - Enterprise Java Beans (EJB) nebo CDI beans
 - Dependency injection – volné propojení
 3. Prezentační vrstva
 - Webové rozhraní (JSF) nebo API (REST, JAX-RS)

Další platformy – přehled

- Java
 - Existuje mnoho možností kromě „standardní“ J EE
- .NET (Core / Framework)
 - Mnoho řešení na všech vrstvách
- PHP
 - Různé frameworky, důraz na webovou vrstvu
- JavaScript
 - Node.js + frameworky, důraz na web a mikroslužby
- Python, Ruby, ... - podobné principy

Distribuované architektury

- Monolitický systém (typické pro třívrstvou architekturu)
 - Vyhvídí se a nasazuje jako jeden celek
 - + snáze zvládnutelný vývoj, testování
 - - obtížnější a pomalejší nasazování nových verzí
- Distribuované architektury
 - Service-oriented architecture (SOA)
 - Mikroservices (mikroslužby)

Jakarta EE

Serverová část informačního systému

Java

- Programovací jazyk
 - Silně typovaný, objektově orientovaný
- Platforma pro vývoj a provoz aplikací
 - Virtuální stroj
 - Podpůrné nástroje
 - (překladač, debugger, dokumentace, ...)
- Balík Java SE (JRE nebo JDK)
 - Alternativní implementace (i open source)

Java EE

- V současnosti Jakarta EE 8 (září 2019)
 - Plně kompatibilní s Java EE 8
- Platforma pro vývoj podnikových aplikací a IS v Javě
- Množina standardních technologií a API
 - Enterprise Java Beans (EJB)
 - Transaction API (JTA)
 - Java Persistence API (JPA)
 - Java Message Service (JMS)
 - Java Server Faces (JSF)
 - ... a další

Vrstvy aplikace Java EE

1. Databázová vrstva
2. Business vrstva
 - Implementace chování aplikace
 - Potenciálně distribuovaná
3. Webová vrstva
 - Webové API nebo komponentový serverový framework

Nejdůležitější součásti specifikace

- **Datová vrstva**
 - Java Persistence API (JPA)
- **Business vrstva**
 - Java Transactions API (JTA)
 - Enterprise Java Beans (EJB)
 - Java Messaging Service (JMS)
- **Webová vrstva**
 - Java Server Faces (JSF), Servlet, WebSocket
 - Java API for RESTful Web Services (JAX-RS)
 - Java API for XML Web Services (JAX-WS) – SOAP, ...

Struktura aplikace Java EE

- EJB moduly
 - Definují veřejná rozhraní
 - Implementují chování
 - Lze je odděleně nasadit na servery
 - EJB vrstva zajišťuje distribuované volání
 - Dependency injection, kontrola souběžnosti, přístupu, ...
- Webové moduly
 - Webové rozhraní

Běhové prostředí – Kontejnery

- Prostředí pro běh aplikace na serveru
- EJB kontejner
 - Běh EJB modulů, volání funkcí
- Webový kontejner
 - Běh webové vrstvy
- Java EE kontejner
 - Webový + EJB kontejner

Dostupné kontejnery

- Java EE kontejnery (aplikáční servery)
 - GlassFish (Oracle, open source)
 - Payara (<https://www.payara.fish/>)
 - WildFly (Red Hat, dříve JBoss AS)
 - TomEE (Apache)
 - WebSphere, Open Liberty (IBM)
- Pouze webové servery
 - Tomcat (Apache)
 - Jetty (Mort Bay Consulting)

Instalace vývojového prostředí

Instalace

- Java 11+ – **musí být JDK**
- Java EE server
 - **Payara (Glassfish)**
 - TomEE
 - WildFly, Open Liberty
- Vývojové prostředí
 - **Eclipse for Java EE developers**
 - NetBeans, IntelliJ IDEA
- Databázový server
 - Jakýkoliv relační (MySQL)
 - JDBC ovladač
- Databázový konektor

Konfigurace

- Instalovat Eclipse
- Rozbalit Payara server
- Definovat server v IDE
 - Pro Glassfish nutné rozšíření Eclipse
- Instalovat databázový konektor
 - Např. do /lib serveru
 - Driver definitions v IDE
- Konfigurovat databázové spojení
 - V IDE i na serveru

Tvorba aplikace v Javě

Struktura aplikace

- Archivy s částmi aplikace:
 - MojeApp.war – webová aplikace
 - MojeKomponenta.jar – EJB komponenta
 - MojeKomponenta.ear – enterprise aplikace
- Archiv *.war obsahuje
 - Přeložené třídy
 - Knihovny (thin vs. thick war)
 - Webové soubory

Struktura aplikace (II)

- Kořenový adresář je / webu
- META-INF
 - Informace o archivu
- WEB-INF/lib
 - Potřebné knihovny (*.jar)
- WEB-INF/class
 - Přeložené třídy (*.class)

Konfigurace aplikace

- Deskriptor WEB-INF/web.xml
 - Jméno aplikace
 - Výchozí stránka
 - Mapování servletů
 - Definice filtrů
 - ...

Vytvoření projektu

- New -> Dynamic Web Project
 - Cílové prostředí (Glassfish)
 - Podpora JavaServer Faces 2.x
 - Podpora JPA
- Konfigurace později
 - Project -> Properties
 - Project Facets
 - Přidat JAX-RS, apod.

Alternativna: Maven

- Maven – nástroj pro správu projektu
 - Získání závislostí, sestavení, ...
- Konfigurace v souboru pom.xml
 - Parametry projektu, moduly
 - Závislosti
- Lze vyjít z připravené šablony
 - <https://github.com/DIFS-Teaching/jsf-basic>
- Přeložení a vytvoření distribučního archivu
 - mvn clean package

Java Bean

```
public class Person
{
    private long id;
    private String name;
    private String surname;
    private Date born;

    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    ...
}
```

Persistence

- Pomocí anotací vytvoříme z třídy **entitu** persistence

```
@Entity  
@Table(name = "person")  
public class Person  
{  
    @Id  
    private long id;  
    private String name;  
    private String surname;  
    private Date born;
```

Generované ID

```
@Entity  
@Table(name = "person")  
public class Person  
{  
    @Id  
    @GeneratedValue(strategy = IDENTITY)  
    private long id;  
    private String name;  
    private String surname;  
    private Date born;  
    ...
```

Vztahy mezi entitami

- Anotace @OneToMany a @ManyToOne
- Nastavení mapování
 - V Eclipse pohled JPA
- Kolekce v Javě:
 - Collection<?>
 - List<?> (Vector, ArrayList, ...)
 - Set<?> (HashSet, ...)
 - Map<?, ?> (HashMap, ...)s

Konfigurace presistence

- Soubor persistence.xml
 - Jméno jednotky persistence
 - Odkaz na data source na serveru
 - Případně další parametry pro mapování
 - Např. řízení automatického generování schématu

Implementace business operací

- Enterprise Java Beans (EJB)
 - Zapouzdřují business logiku aplikace
 - Poskytují business operace – definované rozhraní (metody)
 - EJB kontejner zajišťuje další služby
 - Dependency injection
 - Transakční zpracování
 - Metoda obvykle tvoří transakci, není-li nastaveno jinak

Vytvoření EJB

- Instance vytváří a spravuje EJB kontejner
- Vytvoření pomocí anotace třídy
 - @Stateless – bezstavový bean
 - Efektivnější správa – pool objektů přidělovaných klientům
 - @Stateful – udržuje se stav
 - Jedna instance na klienta
 - @Singleton
 - Jedna instance na celou aplikaci

Použití EJB

- Lokální
 - Anotace @EJB – kontejner dodá instanci EJB
- Vzdálené volání – dané rozhraní
 - Rozhraní definované pomocí @Remote

Propojení s JPA

- EJB implementují business operace
 - Často stačí stateless bean
- JPA rozhraní je reprezentováno objektem EntityManager
 - Dodá kontejner pomocí DI

Uložení objektu

```
@PersistenceContext  
EntityManager em;
```

```
Person person = new Person();  
person.setName(„karel“);  
em.persist(person);
```

Změna objektu

```
@PersistenceContext  
EntityManager em;
```

```
person.setName("Karel");  
em.merge(person);
```

Smazání objektu

```
@PersistenceContext  
EntityManager em;
```

```
em.remove(person);
```

Dotazování

- ```
Query q = em.createQuery("...");
q.setParameter(name, value);
q.setFirstResult(100);
q.setMaxResults(50);
q.getResultList();
```

# JPQL dotazy

- ```
SELECT p FROM Person p  
WHERE p.name = "John"
```
- ```
SELECT c FROM Car c
WHERE c.reg LIKE :pref
```
- ```
SELECT  
    NEW myObject(c.type, count(c))  
FROM Car c  
GROUP BY c.type
```

Contexts and Dependency Injection (CDI)

- Obecný mechanismus pro DI mimo EJB
- Omezuje závislosti mezi třídami přímo v kódu
 - Flexibilita (výměna implementace), lepší testování, ...
- Injektovatelné objekty
 - Třídy, které nejsou EJB
 - Různé vlastnosti pomocí anotací
- Použití objektu
 - Anotace @Inject
 - CDI kontejner zajistí získání a dodání instance

CDI – Injektovatelné objekty

- Téměř jakákoli Javaovská třída
- Scope
 - `@Dependent` – vzniká pro konkrétní případ, zaniká s vlastníkem (default)
 - `@RequestScoped` – trvá po dobu HTTP požadavku
 - `@SessionScoped` – trvá po dobu HTTP session
 - `@ApplicationScoped` – jedna instance pro aplikaci
 - *Pozor na shodu jmen se staršími anotacemi JSF*
- Pokud má být přístupný z GUI (pomocí EL)
 - Anotace `@Named`

CDI – Dodání instancí

- Anotace @Inject
- Vlastnost (field)

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    @Inject
    private NumberGenerator numberGenerator;

}
```

CDI – Dodání instancí

- Konstruktor

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    private NumberGenerator numberGenerator;

    @Inject
    public ItemServlet(NumberGenerator numberGenerator) {
        this.numberGenerator = numberGenerator;
    }
    ...
}
```

CDI – Dodání instancí

- Setter

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    private NumberGenerator numberGenerator;

    @Inject
    public void setNumberGenerator(NumberGenerator numberGenerator)
    {
        this.numberGenerator = numberGenerator;
    }

    ...
}
```

Webové API – REST

REST

- Representational state transfer
- Jednoduchá metoda vzdálené manipulace s daty
- Reprezentuje CRUD (Create-Retrieve-Update-Delete) operace
 - Ale ve skutečnosti přistupujeme k **business vrstvě, ne přímo k datům!**
- Úzká vazba na HTTP
- Nedefinuje formát přenosu dat, obvykle JSON nebo XML

Využití HTTP

- Každá položka dat (nebo kolekce) má vlastní URI. Např.:
 - <http://noviny.cz/clanky> (kolekce)
 - <http://noviny.cz/clanky/domaci/12> (jeden článek)
- Jednotlivé metody HTTP implementují příslušné operace s daty

Metody HTTP

- GET
 - Kolekce: získání seznamu položek
 - Entita: čtení entity (read)
- POST
 - Kolekce: přidání prvku do kolekce (create)
- PUT
 - Kolekce: nahrazení celého obsahu kolekce
 - Entita: zápis konkrétní entity (update)
- DELETE
 - Kolekce: smazání celé kolekce
 - Entita: smazání entity

Formát přenosu dat

- Není specifikován, záleží na službě
 - Obvykle JSON nebo XML (schéma záleží na aplikaci)
- Často více formátu k dispozici
 - Např.
<http://noviny.cz/clanky.xml>
<http://noviny.cz/clanky.json>
 - Využití MIME pro rozlišení typu, HTTP content negotiation

REST a Jakarta EE

- JAX-RS API součástí standardu
- Vytvoření služeb pomocí anotací
- Aplikační server zajistí funkci endpointu
 - Mapování URL a HTTP metod na Javaovské objekty a metody
 - Serializace a deserializace JSON/XML na objekty
- Různé implementace
 - JAX-RS – Jersey (Glassfish), Apache Axis
 - Serializace – Jackson, gson, MOXy, ...

REST v Javě

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("username") String
                          userName) {
        ...
        return someXmlString;
    }
}
```

Konfigurace

- JAX-RS servlet ve web.xml
 - Specifikace balíku s REST třídami nebo přímo výčet tříd
 - Specifikace JSON provider (Jackson)
- Alternativně
 - Třída odvozená od javax.ws.rs.core.Application
 - Konfigurace pomocí anotací

Klientská aplikace

- Zasílání REST požadavků
 - Jednotlivé JS frameworky mají vlastní infrastrukturu
- Prezentační logika
 - Navigace
 - Přechody mezi stránkami
 - Výpisy chyb, apod.

Autentizace v REST

- Protokol REST je definován jako ***bezstavový***
 - Požadavek musí obsahovat vše, žádné ukládání stavu na serveru
- To teoreticky vylučuje možnost použití sessions pro autentizaci
 - Technicky to ale možné je
 - Problém např. pro mobilní klienty
- Alternativy pro autentizaci:
 - HTTP Basic autentizace (nutné HTTPS)
 - Použití tokenu validovatelného na serveru – např. JWT
 - Složitější mechanismus, např. OAuth

HTTP Basic

- Standardní mechanismus HTTP, využívá speciální hlavičky
- Požadavek musí obsahovat hlavičku **Authorization**
 - Obsahuje jméno a heslo; nešifrované pouze kódované (base64)
 - **Je nutné použít HTTPS**
- Pro nesprávnou nebo chybějící autentizaci server vrací **401 Authorization Required**
 - V hlavičce **WWW-Autenticate** je identifikace oblasti přihlášení
 - Klient tedy zjistí, že je nutná autentizace pro tuto oblast

JSON Web Token (JWT)

- Řetězec složený ze 3 částí
 1. Header (hlavička) – účel, použité algoritmy (JSON)
 2. Payload (obsah) – JSON data obsahující id uživatele, jehopráva, expiraci apod.
 3. Signature (podpis) – pro ověření, že token nebyl podvržen nebo změněn cestou
- Tyto tři části se kódují (base64) a spojí do jednoho řetězce
 - **xxxxx.yyyyy.zzzzz**

Použití JWT

- Klient kontaktuje *autentizační server* a dodá autentizační údaje
 - Stejný server, jaký poskytuje API, nebo i úplně jiný (např. Twitter)
- Autentizační server vygeneruje podepsaný JWT a vrátí klientovi
- Klient předá JWT při každém volání API
 - Nejčastěji opět v hlavičce:
Authorization: Bearer xxxxx.yyyyy.zzzzz
 - API ověří platnost, role uživatele může být přímo v JWT

Otázky?



Systém Caché

Objektově orientovaný DB systém

Radek Burget
burgetr@fit.vutbr.cz

Datové modely

- **Jednoduché (NoSQL)**
 - Key-value (MUMPS, Redis, ...)
 - Dokumentové (MongoDB, CouchDB, ...)
 - Sloupcové (Apache HBase, ...)
- **Relační datový model**
 - Mnoho implementací
- **Objektový datový model**
 - Objektově-relační mapování (ORM)
- **Grafové**
 - Grafové databáze (Neo4J, OrientDB, ...)
 - Sémantická úložiště (sémantický web, RDF)

Key-value databáze

- Asociativní pole jako datový model
- Typicky řetězcový klíč
 - Často seřazené klíče – efektivní procházení
 - Možná organizace klíčů, např. hierarchie
- Hodnoty různých datových typů
 - Jedna nebo více hodnot
- Např. Redis, MUMPS

Odbočka: MUMPS

- Massachusetts General Hospital Utility Multi-Programming System (60. léta v USA)
 - Postupně vzniklo několik komerčních i open-source implementací
 - InterSystems Caché (1997 -)

Systém MUMPS

- Jazyk pro databázové aplikace
- Předpokládá existenci databáze společné všem aplikacím
- Proměnné začínající ^ se ukládají do databáze – **globály**

```
SET a = 5      //proměnná v RAM  
SET ^a = 5     //proměnná v databázi
```

Ukázky MUMPS

```
; Loop for ever, read a line (quit on end of file), process that line
For Read input Quit:$ZEOF!$Length(input) Do ; input has entire input line
.
. Set i=$Piece(input," ",1) ; i - first number on line is starting integer for the problem
. Set j=$Piece(input," ",2) ; j - second number on line is ending integer for the problem
. Write $FNumber(i","","0)," ",$FNumber(j","","0) ; print starting and ending integers, formatting with commas
.
. Set k=$Piece(input," ",3) ; k - third number on input line is number of parallel streams
. If streams>k Do ; print number of execution streams, optionally corrected
.. Write " (",$FNumber(k","","0)
.. Set k=streams
.. Write "->",$FNumber(k","","0))
. Else Write " ",$FNumber(k","","0)
.
. Set blk=+$Piece(input," ",4) ; blk - size of blocks of integers is optional fourth piece
. Set tmp=(j-i+k)\k ; default / maximum block size
. If blk&(blk'>tmp) Write " ",$FNumber(blk","","0) ; print block size, optionally corrected
. Else Do
.. Write " (",$FNumber(blk","","0)
.. Set blk=tmp
.. Write "->",$FNumber(blk","","0))
.
. ; Define blocks of integers for child processes to work on
. Kill ^limits
. Set tmp=i-1
```

Ukázky MUMPS

```
%DTC
%DTC ; SF/XAK - DATE/TIME OPERATIONS ;1/16/92 11:36 AM
;;19.0;VA FileMan;;Jul 14, 1992
D I 'X1!'X2 S X="" Q
S X=X1 D H S X1=%H,X=X2,X2=%Y+1 D H S X=X1-%H,%Y=%Y+1&X2
K %H,X1,X2 Q
;
C S X=X1 Q:'X D H S %H=%H+X2 D YMD S:$P(X1,".",2) X=X_._.$P(X1,".",2) K X1,X2 Q
S S %=%#60/100+(%#3600\60)/100+(%\3600)/100 Q
;
H I X<1410000 S %H=0,%Y=-1 Q
S %Y=$E(X,1,3),%M=$E(X,4,5),%D=$E(X,6,7)
S %T=$E(X_0,9,10)*60+$E(X_"000",11,12)*60+$E(X_"00000",13,14)
TOH S %H=%M>2&'(%Y#4)+$P("^31^59^90^120^151^181^212^243^273^304^334","^",%M)+%D
S %='%M!%D,%Y=%Y-141,%H=%H+(%Y*365)+(%Y\4)-(%Y>59)+%,%Y=$S(%:-1,1:%H+4#7)
K %M,%D,% Q
;
DOW D H S Y=%Y K %H,%Y Q
DW D H S Y=%Y,X=$P("SUN^MON^TUES^WEDNES^THURS^FRI^SATUR","^",Y+1)_"DAY"
S:Y<0 X="" Q
7 S %=%H>21608+%H-.1,%Y=%\365.25+141,%=%#365.25\1
S %D=%+306#(%Y#4=0+365)#153#61#31+1,%M=%-%D\29+1
S X=%Y_"00"+%M_"00"+%D Q
;
```

Hierarchická organizace dat

- Proměnná (globál) může mít libovolné množství indexů („subscript“)

```
SET ^osoba(1) = „Jan Novák“  
SET ^osoba(1, „vek“) = 25  
SET ^osoba(1, „adresa“, „ulice“) = „Za vodou“  
SET ^osoba(1, „adresa“, „cislo“) = 50
```

Caché Object Script

- Vychází z MUMPS
- Novinky v syntaxi
 - Bloky, while cyklus, ...
- Objektová rozšíření

Caché ObjectScript - příkazy

- **Set** - přiřazení, např. SET A = 1
- **Do** - volání metody nebo rutiny, např. DO X(43)
- **Write** – výpis hodnoty
- **Quit** - konec subrutiny, metody, cyklu
- **Kill** – zrušení proměnné (lokální i globálu)

Řídicí struktury

- **If ... Elseif ... Else** – větvení
 - if a < 5 { ... } else { ... }
- **For** - např. FOR X = 1:1:100
- **While & Do ... While** - smyčky, např. WHILE B < 10 { SET B = B + 1 }.

Zvláštnosti FOR cyklu

```
FOR I = 1:1:5 W I,!  
FOR I = 1:1:5 W I W:I=2 „*“  
FOR I = 1:1:5 {  
    w I  
    w:I=2 „*“  
}
```

Příklad rutiny

```
;  
;soubor vek.mac  
;  
vekosoby(jmeno)  
    SET roknar = ^osoba(jmeno, "rocnik")  
    SET rok = $extract($zdate($horolog, 8), 0, 4)  
    Write rok - roknar
```

Do **vekosoby^vek(„jan“)**

Objektový model v Caché

- Třídy definovány pomocí vnitřního Class Definition Language
 - Třída Class, vícenásobná dědičnost
 - Parametry třídy Parameter
 - Vlastnosti Property (Parametr=hodnota)
 - Metody Method, ClassMethod
 - Pojmenované dotazy

Práce s objekty

- Vytvoření instance třídy
 - set p = ##class(demo.Person).%New()
- Nastavení hodnot vlastností
 - set p.surname = "Clinton"
- Uložení do databáze (persistence)
 - do p.%Save()
- Zjištění identifikátoru
 - w p.%Id()

Přístup o objektům

- Přes ID
 - `set p = ##class(demo.Person).OpenId(id)`
- Pojmenovaný dotaz
- Ad-hoc dotaz (pseudo SQL)

Pojmenovaný dotaz

- Pro danou třídu
- Každá třída má předdefinovaný dotaz **Extent**

```
Set rset =
    ##class(%ResultSet) . %New ("demo.Person:Extent")
Do rset.Execute()
...
set p = rset.GetObject()
```

Vlastní dotaz

```
Query QueryName(Parameter As %String) As %SQLQuery
{
SELECT MyProperty, MyOtherProperty FROM MyClass WHERE
(MyProperty = "Hello" AND MyOtherProperty =
:Parameter) ORDER BY MyProperty
}
```

Ad-hoc dotazy

```
Set dotaz = "SELECT %ID FROM demo.Person WHERE  
name='Jan'"
```

```
Set rset = ##class(%ResultSet).%New()
```

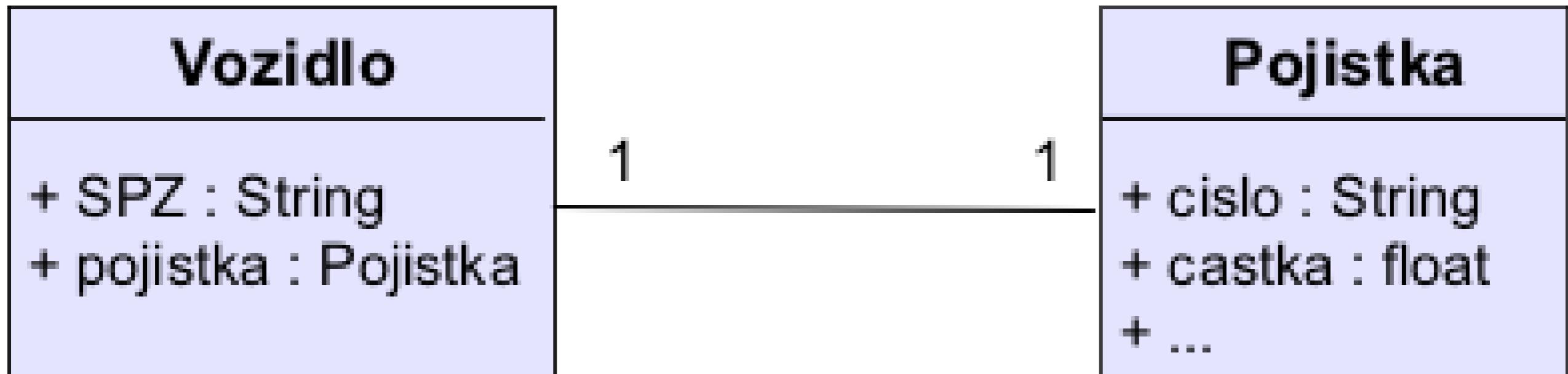
```
Do rset.Prepare(dotaz)
```

```
Do rset.Execute()
```

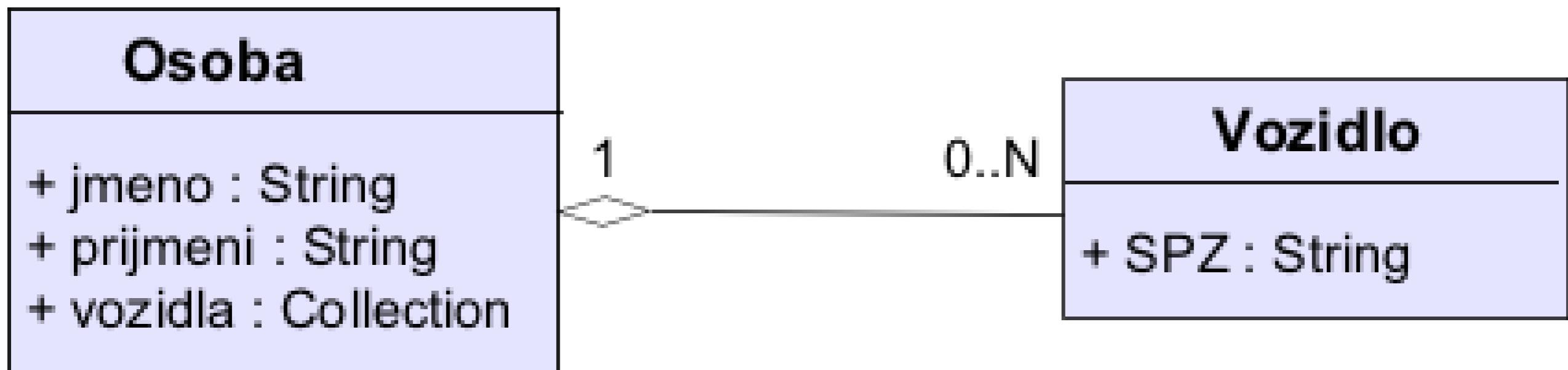
Vztahy v objektové databázi

- Z hlediska objektové databáze
 - Vztahy 1:1
 - Vztahy 1:N
 - Vztahy M:N

Vztahy 1:1



Vztahy 1:N



Kolekce v Caché

- Array
 - Asociativní pole
 - K hodnotám se přistupuje přes klíč
- List
 - Uspořádaný seznam
 - K hodnotám se přistupuje přes index 1..n

List

- Deklarace
 - Property jména as %String [Collection = list]
- Metody
 - jména.Insert("Jan")
 - jména.GetAt(index)
 - jména.SetAt("John", index)
 - jména.InsertAt("Jana", index)
 - jména.Count()

Relation

- Vlastnosti vyjadřující relace

Relation <jmeno> as Typ [parametry]

- Parametry:
 - Cardinality = one | many
 - parent | children
 - Inverse = *vlastnost protitřídy*
- Vždy oboustranné
- Caché zabezpečí konzistenci vztahu

Příklad

```
Class demo.Person Extends ...
```

```
{
```

```
Relationship cars
```

```
As demo.Car
```

```
[ Cardinality = many, Inverse = owner ] ;
```

```
}
```

Příklad

```
Class demo.Car Extends ...
```

```
{
```

```
Relationship owner
```

```
As demo.Person
```

```
[ Cardinality = one, Inverse = cars ];
```

```
}
```

Příklad

```
set c = ##class(demo.Cars) .%New()  
set p = ##class(demo.Person) .%New()  
  
do p.cars.Insert(c)  
//...vlastnost c.owner bude mít hodnotu p  
  
do p.%Save()  
//...uloží se automaticky i c
```

Objektová reprezentace

- Relace se projeví
 - Na straně ONE jako běžná vlastnost daného typu
 - Na straně MANY jako **kolekce**
(konkrétně reprezentovaná datovým typem
%RelationshipObject, se stejným rozhraním, jako ostatní kolekce)

Vztahy Parent – Children

- Objekty Children jsou závislé na Parent
 - Smazání otcovského objektu maže odkazované objekty
 - Závislé objekty nemohou být asociovány s jiným otcovským

Otázky?



Pokročilé informační systémy

Objektový model dat

prof. Ing. Tomáš Hruška, CSc.

Ing. Radek Burget, Ph.D.

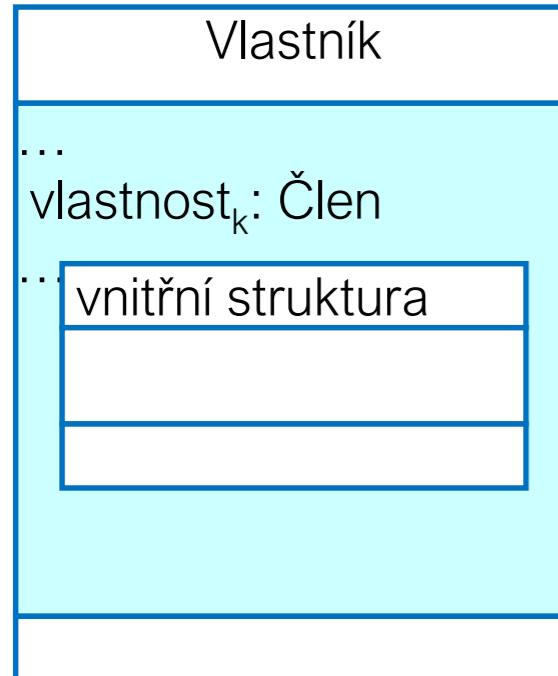
burgetr@fit.vutbr.cz

Objektový datový model

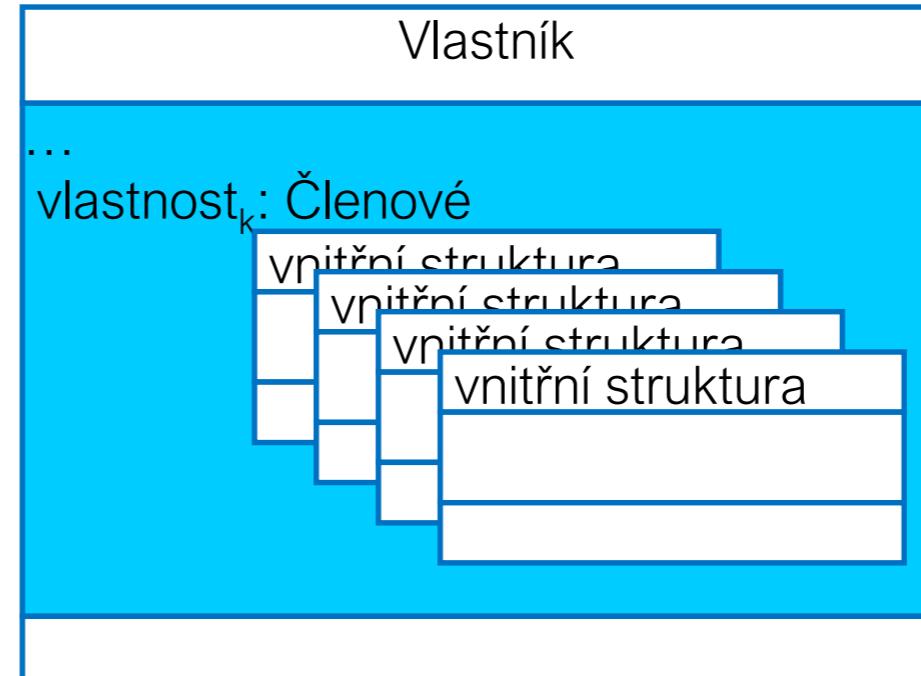
- Motivace: v aplikacích obvykle modelujeme data objektově
 - Objektově-orientované modelování v návrhu IS, UML
- Data reprezentovaná pomocí konceptů objektově orientovaného modelování
 - Třídy, objekty (instance)
- Vztahy (reference)
 - Na rozdíl od relačních databází (nemluvě o NoSQL)

Struktura objektů a vztahy

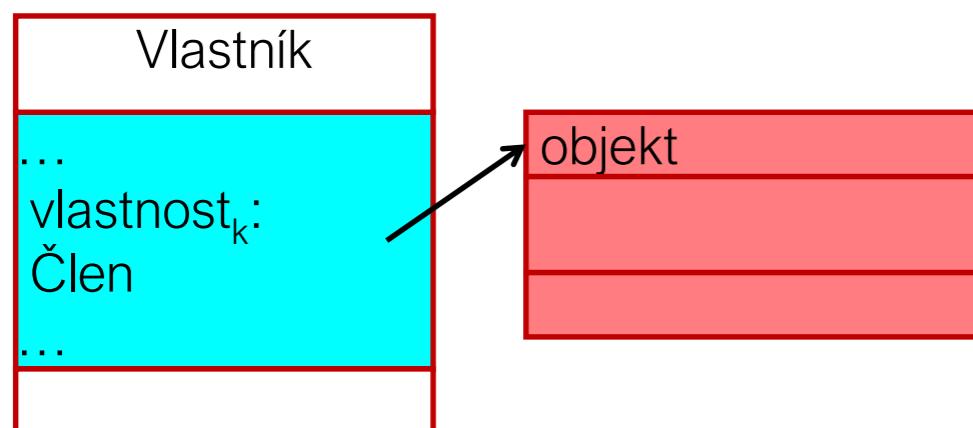
Zanoření a vztahy



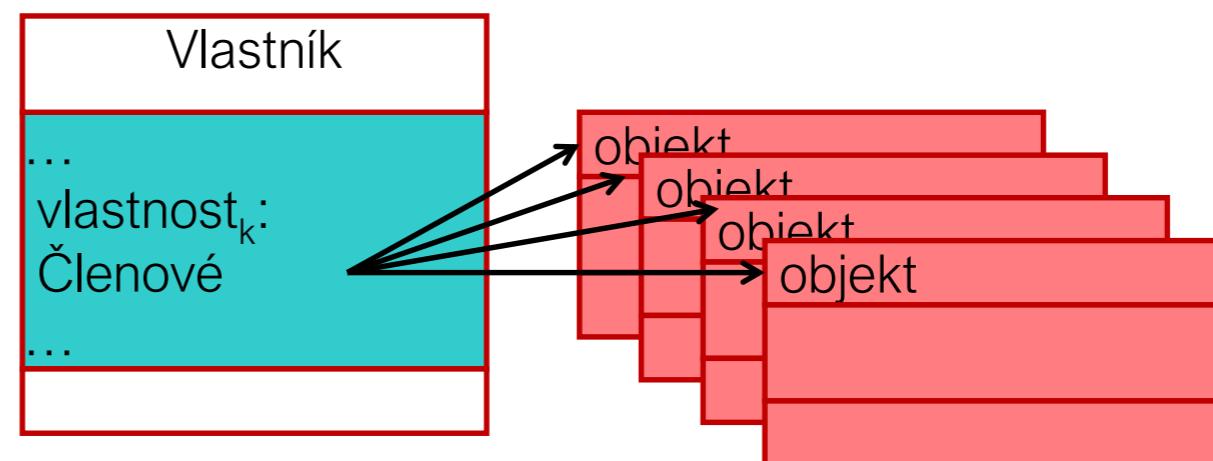
1. člen je jedinou prostou strukturou =
vnoření hierarchie struktur



2. členové jsou v kolekci prostých struktur =
vytváření hierarchie struktur s vnořenými kolekcemi



3. člen je jediným objektem =
vytváření vztahu typu 1:1



4. členové jsou v kolekci objektů = vytváření
vztahu typu 1:N

Vztahy kontra strukturalizace

	struktura	kolekce
vnoření v rámci obálky (strukturovaná data uvnitř)	jmenný prostor nižší úrovně (přístup přes tečku)	prostor přístupný operacemi kolekce
Vztah (k objektu vně)	1:1	1:N

Jmenný prostor nižší úrovňě

concept TYPD [Data=Value]

...

end concept

concept TYPB

properties

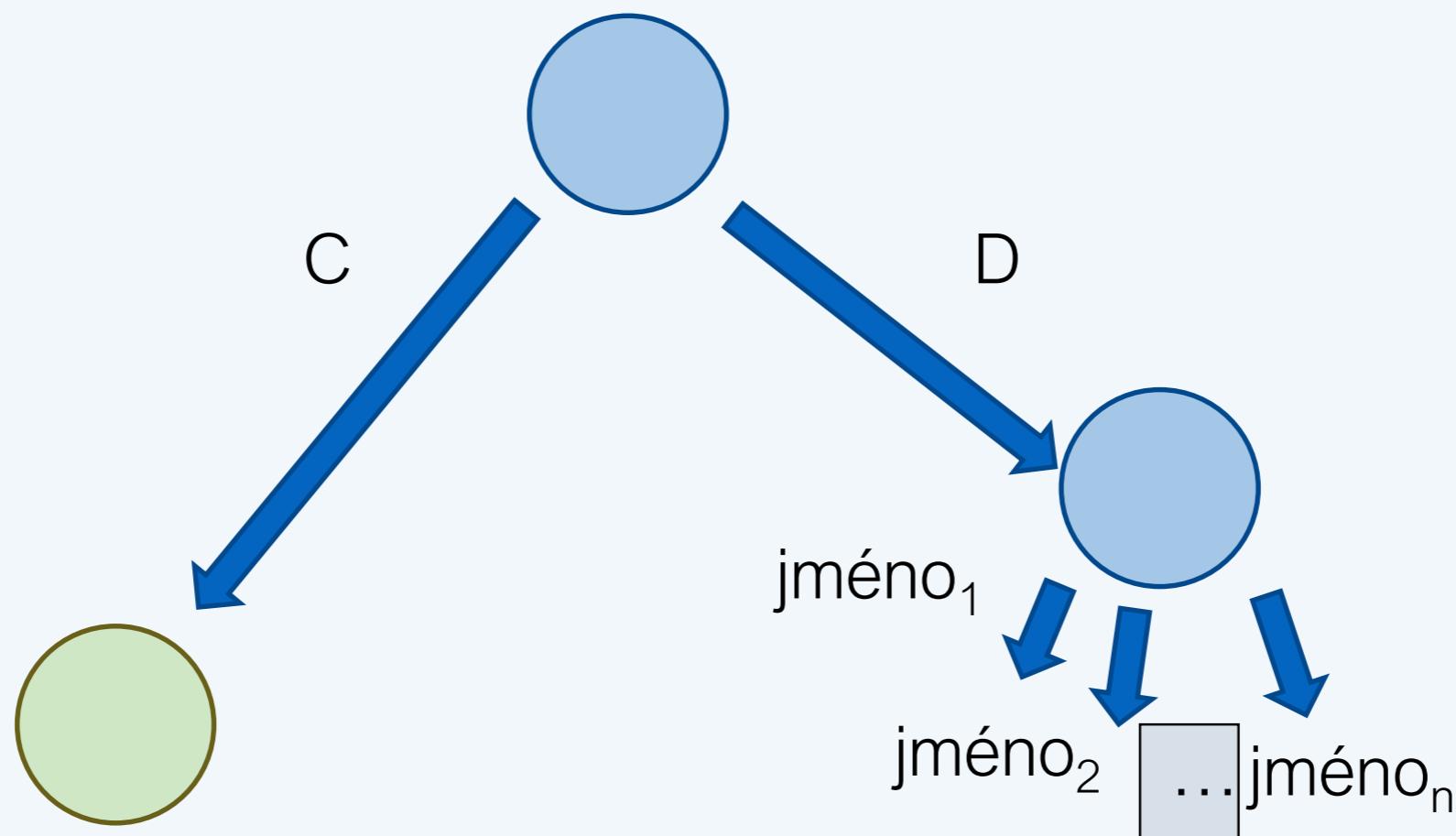
C: integer

D: TYPD

end concept

Jmenný prostor nižší úrovňě

zanořená datová struktura



Prostor přístupný operacemi kolekce

concept TYPB/TYPYB [Data=Value]

properties

C: integer

D: TYPD

end concept

concept ZANORENA

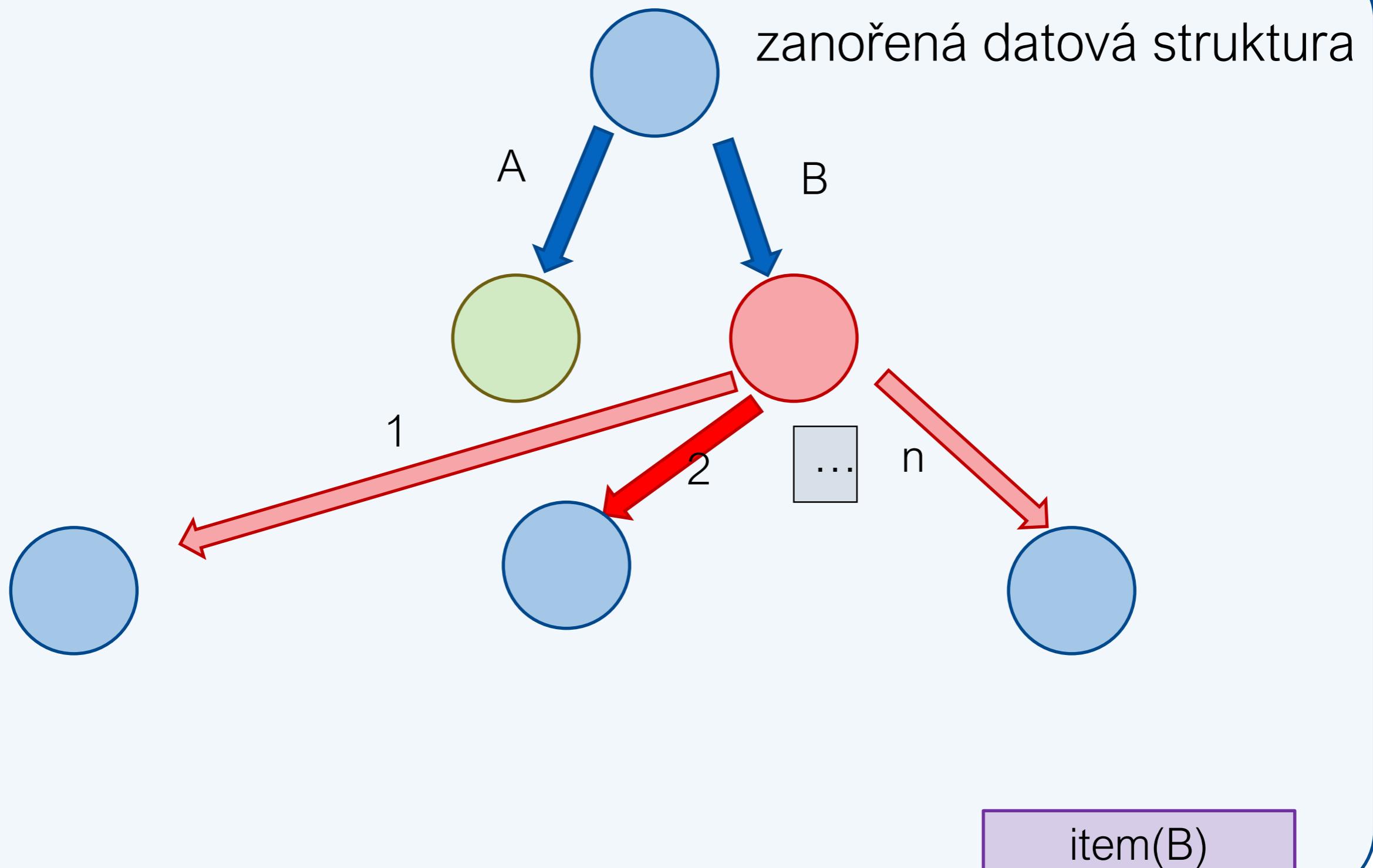
properties

A: integer

B: TYPYB

end concept

Prostor přístupný operacemi kolekce



Vztah 1:1

concept CLEN [Data=Ref]

properties

C: **integer**

D: ...

...

end concept

concept VLASTNIK

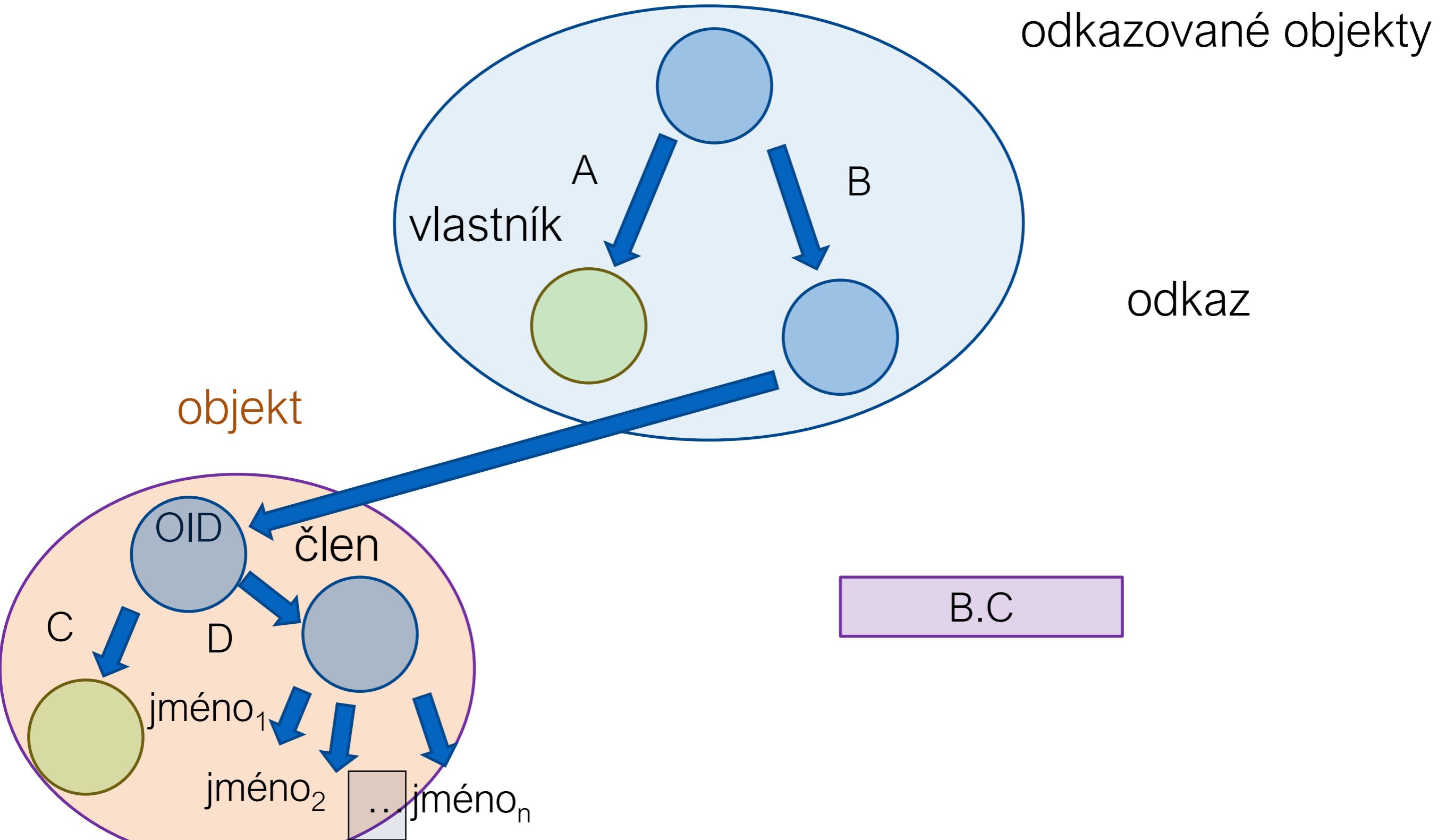
properties

A: **integer**

B: CLEN

end concept

Vztah 1:1



Vztah 1:N

concept VLASTNIK

properties

A: integer

B: CLENOVE

end concept

concept CLEN/CLENOVE [Data=Ref]

properties

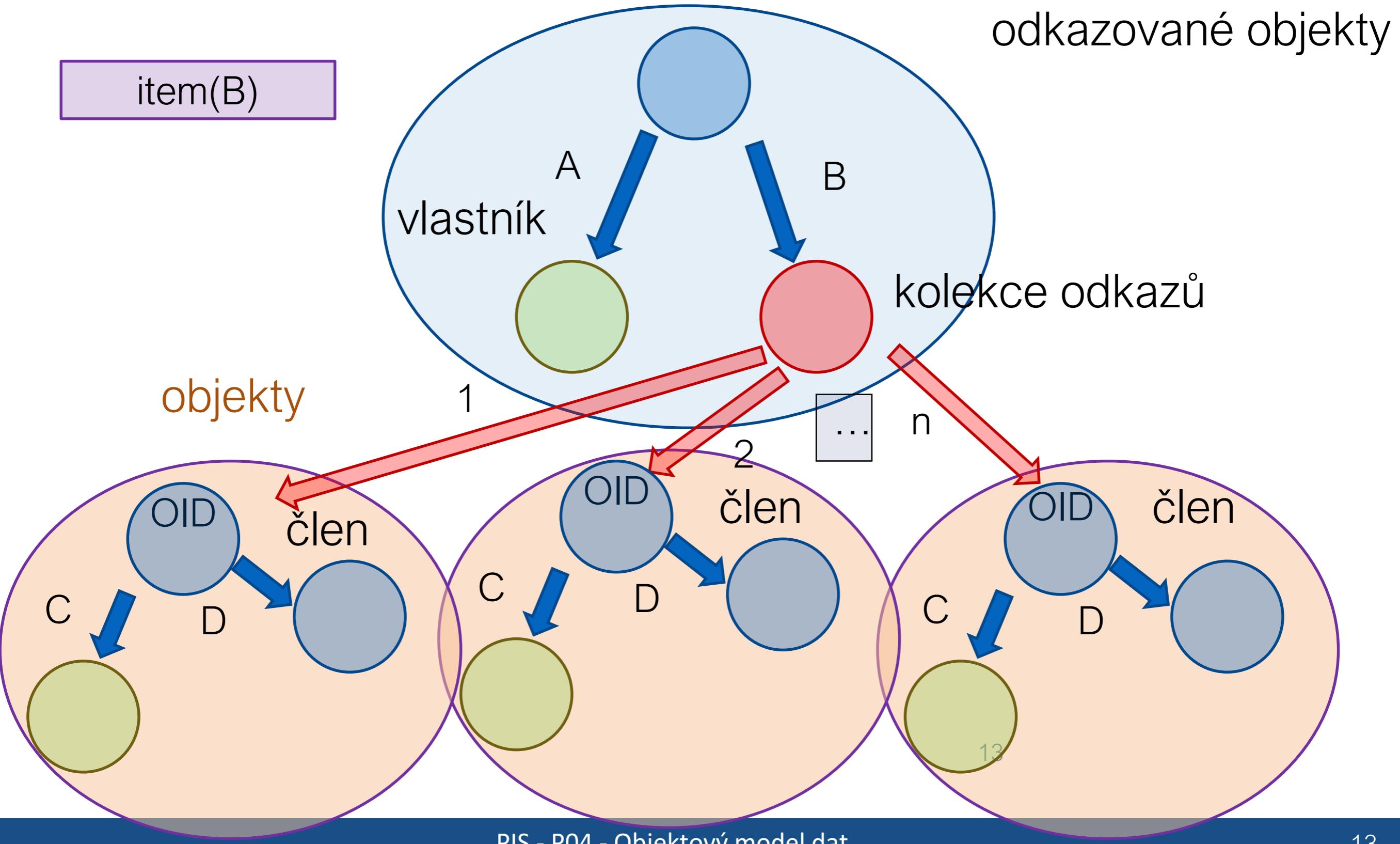
C: integer

D: ...

...

end concept

Vztah 1:N



Příklad vnořené prosté struktury

concept Adresa/Adresy [Data=Value]

properties

vlastnosti adresy

end concept

concept PrvekSAdr/PrvkySAdr

properties

Adresat: **string**

Adresy: Adresy

Adresa: Adresa

end concept

Inverzní vztahy

- Častým modelovaným případem je situace, kdy je požadováno, aby ***vytvoření vztahu V z objektu A na objekt B vyvolalo rovněž vytvoření vztahu W z objektu B na objekt A.***
- Podobně při zrušení vztahu V z objektu A na objekt B musí dojít i ke zrušení vztahu W z objektu B na objekt A.

Inverzní vztahy

- Tuto situaci vyjádříme zápisem atributu **Inverse** ke vztahu V. Datový typ vztahu V určuje, na který objekt vztah povede. Hodnota atributu Inverse udává jméno vztahu W, který má být v objektu B udržován jako inverzní.

Příklad inverzních vztahů

concept A
properties

...

V: B [Inverse=W]

...

end concept

concept B
properties

...

W: A

...

end concept

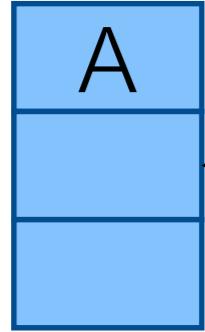
Typy inverzních vztahů

- Inverzní vztahy mohou být jak typu 1:1, tak typu 1:N.
- Je však potřeba si uvědomit poněkud rozdílné použití jména kolekce zde, v atributu `Inverse`, nežli při definici vztahu typu 1:N.
- Je-li použit atribut `Inverse` u vztahu 1:N, znamená to, že ***inverzní vztah bude vytvářen s každým prvkem příslušné kolekce***, nikoliv se vztahem, který by byl vlastností kolekce samotné.

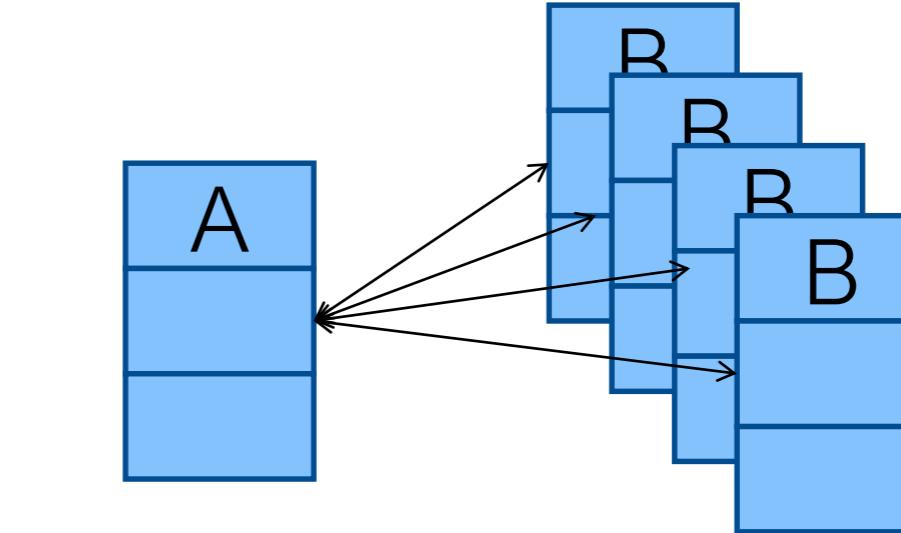
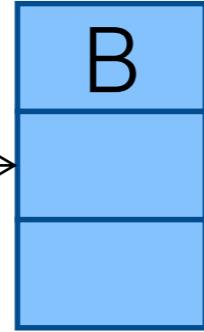
Oboustranné inverzní vztahy

- Inverzní vztahy *nemusejí být vždy oboustranné.*
- Nicméně nejčastějším případem bývají oboustranné inverzní vztahy, kdy popisované vlastnosti inverze vztahu platí jedním i druhým směrem. Tři možné situace oboustranných inverzních vztahů jsou na následujícím slajdu

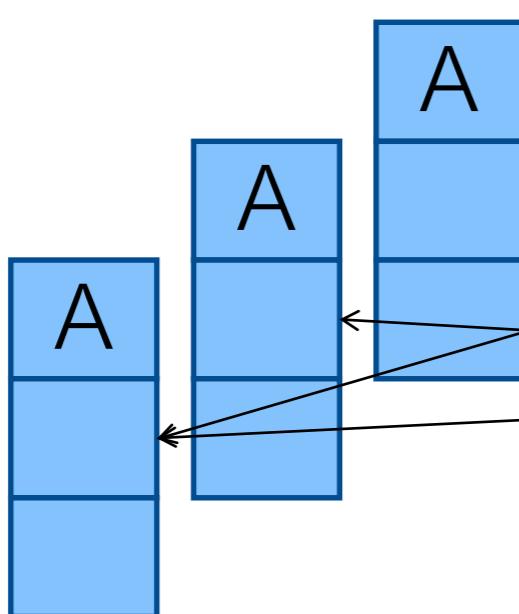
Inverzní vztahy



1:1 proti 1:1



1:N proti 1:1 a naopak



1:N proti 1:N

Příklad

concept PrvekSAdr/PrvkySAdr

properties

Adresat: **string**

Adresy: Adresy [Inverse=CiAdresa]

Adresa: Adresa

end concept

concept Adresa/Adresy

properties

Ucel: DruhAdr

CiAdresa: PrvekSAdr

[Inverse = Adresy]

Adresat: **string**

ObsahAdr: ObsahAdr

end concept



Příklad

- Dva typy objektů.
 - Prvním z nich je objekt modelující prvek s adresou, v němž se vyskytuje vlastnost Adresy typu vztah 1:N na prvek kolekce Adresy.
 - Zde je definována inverze tohoto vztahu vzhledem ke vztahu CiAdresa objektu Adresa.
 - Inverze vztahu je ***oboustranná***. Znamená to, že v objektu Adresa a jeho vlastnosti CiAdresa je definována inverze typu 1:1 na vztah Adresy objektu PrvekSAdr.

Generalizace a specializace (dědičnost)

Dědičnost – vazby mezi typy objektů

- Vazby **mezi typy** struktur. Všechny možné vazby diskutované zde se vyskytují pouze separátně mezi stejnými typy struktur, tedy mezi:
 - **objekty** a
 - **prostými strukturami.**
- Nebudeme uvažovat vazbu mezi typem prosté struktury a typem objektu. Rovněž vazby mezi výčtovými typy a kolekcemi neexistují.

Dědění

- Uvažujme dva obecně různé typy struktur A a B.

concept A/AA

properties

vlastnost_{A1}

vlastnost_{A2}

vlastnost_{A3}

...

end concept

concept B/BB

properties

vlastnost_{B1}

vlastnost_{B2}

vlastnost_{B3}

...

end concept

Dědění

- Vlastnosti struktury A a struktury B jsou ***obecně různé***. To znamená, že jednou krajní situací je, že
 - ***obě struktury jsou typově zcela stejné*** a druhou, že
 - ***jsou zcela různé***.
- Mezi tím je možno nalézt mnoho situací, kdy se struktury částečně shodují co do některých vlastností, jmen vlastností apod.

Diference

- Pokud **mají struktury společné rysy**, bývá často výhodné vyjádřit typ struktury B pomocí typu struktury A. K tomu můžeme použít tří druhů popisu **rozdílu typů (diferencí)**:
 - **přidávání** nové vlastnosti ke stávajícím vlastnostem typu A,
 - **modifikaci (upřesňování)** stávající vlastnosti typu A a
 - **zrušení (vypouštění)** vlastnosti typu A.

Definice typu B z A

- Definici typu struktury B z A můžeme potom provést výrokem:
- Typ B ***obsahuje všechny vlastnosti*** typu A, avšak ***jsou do něj přidány nové vlastnosti*** D, E, F..., ***jsou upraveny vlastnosti*** G,H,I,... následujícím způsobem a vlastnosti J,K,L ... ***byly zrušeny***.
- Toto nazveme ***děděním z A do B.***

Předek a následník

- Pokud definujeme typ určením diferencí, pak tento způsob definice nazýváme ***děděním***. Typ A nazýváme ***předkem*** a typ B ***následníkem***.
- Pokud vzájemné odvozování typů má více kroků, pak typ A, z něhož byl typ B přímo odvozen se nazývá ***přímý předek*** a typ B ***přímý následník***. Pokud odvození proběhlo v několika krocích označujeme typ A pouze slovem předek a typ B následník.
- Jde o binární relace na množině typů struktur a relace předek a následník jsou ***tranzitivními uzávěry relace*** přímého předka a následníka.

Více přímých předků

- Přímých následníků je obecně více vždy
- Podobně můžeme postupovat pokud je přímých předků více. Definice potom zní následujícím způsobem:
- Typ B obsahuje všechny vlastnosti **typů X, Y, Z, ...**, avšak jsou do něj přidány nové vlastnosti D, E, F..., jsou upraveny vlastnosti G,H,I,... následujícím způsobem a vlastnosti J,K,L ... byly zrušeny.

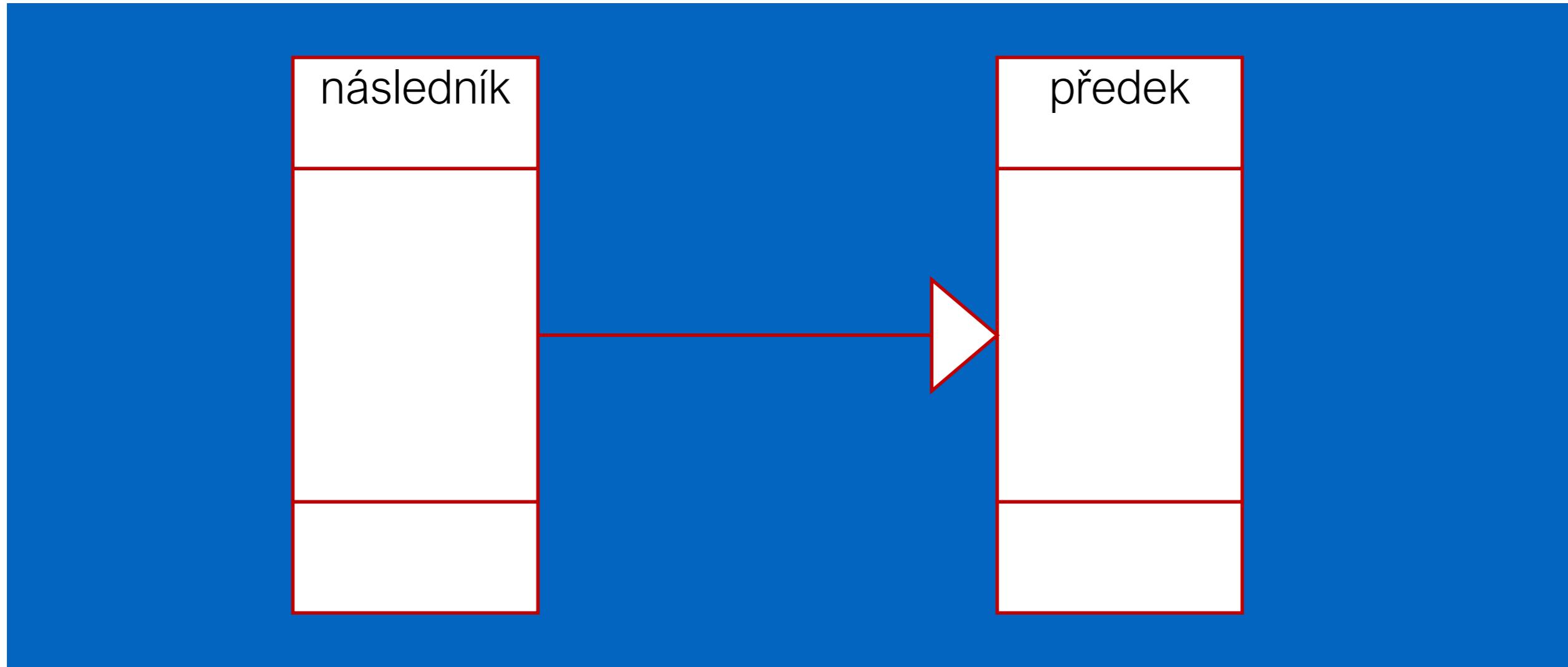
Technika odvozování

- Používáme-li definici typů odvozováním z jiných typů, vzniká hierarchická uspořádaná struktura typů.
- Při přidávání nového typu dochází pak k jeho zařazování do hierarchie typů. Hledá se vždy **přímý předek**. Existují různé možnosti zařazení nového typu, např.:
 - průchodem hierarchie nalézt vhodného předka,
 - vytvořit typ zcela nezávislý na jiných již existujících typech,
 - restrukturalizovat stávající hierarchii typů, čímž získáme vhodného předka,

Generalizace a specializace

- Při budování hierarchického uspořádání typů lze použít při návrhu modelu dvojí postup:
 - Výběrem a sdílením společných charakteristik do nadřízených typů dochází ke ***generalizaci***.
 - Přidáváním nových tříd a doplňováním unikátních vlastností dochází ke ***specializaci***.

Zobrazení dědičnosti



- Ve schématu podle UML jde šipka ve směru generalizace.

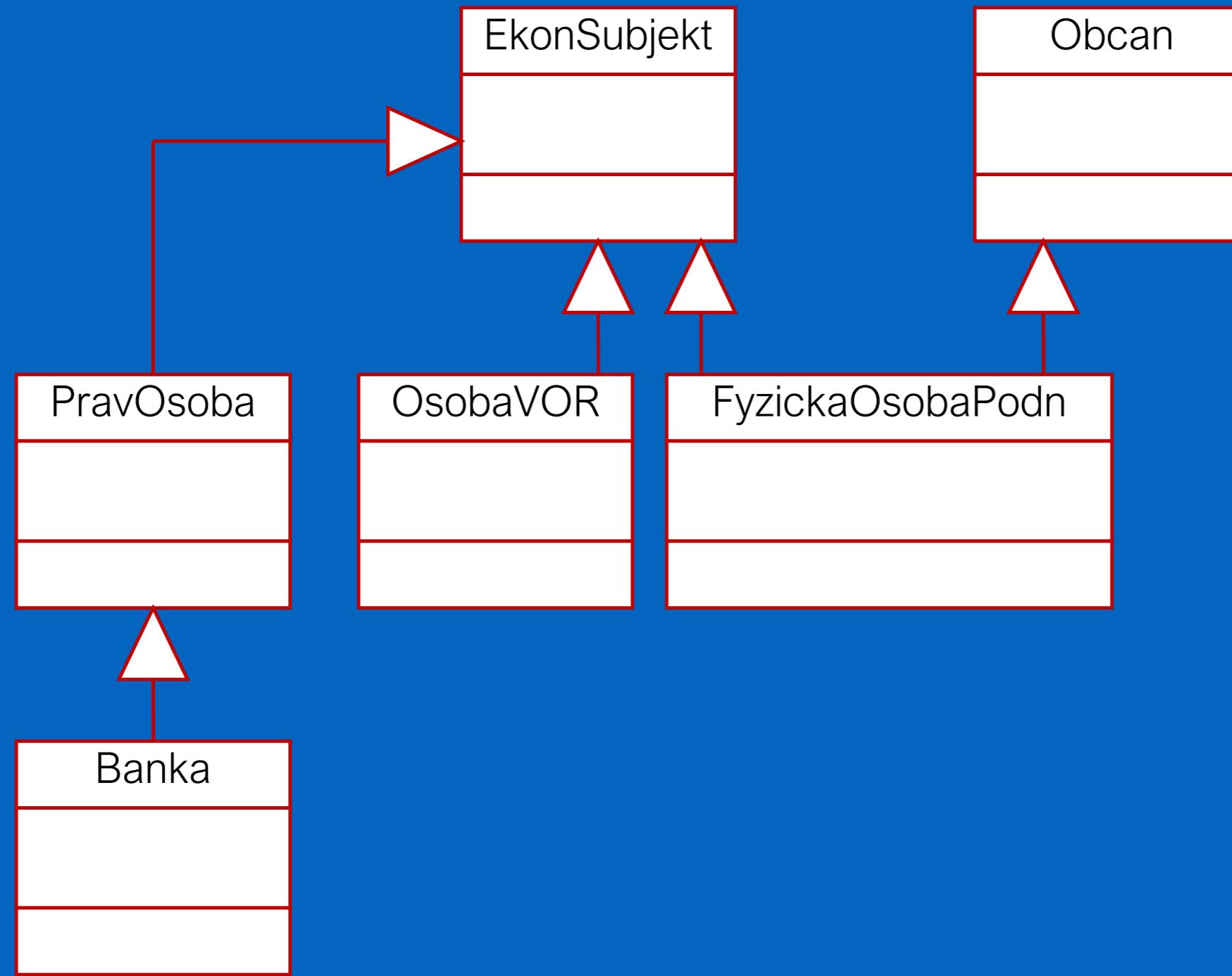
Definice dědičnosti v CDL

- Dědičnost budeme zapisovat do definice typů struktur za klíčové slovo **Inherits** (podobně jako vlastnosti zapisujeme za klíčové slovo **Properties**). Za toto klíčové slovo uvedeme postupně na řádcích jména všech **přímých předků**.
- Typ ve skutečnosti obsahuje vlastnosti všech předků (nejen přímých). Tranzitivní uzávěr počítá překladač.
- Je vhodné se občas přesvědčit, co všechno jsme **nadědili**.

Přidávání vlastností

- Nejčastějším případem diference při dědění je přidávání nových vlastností. Ty budeme definovat tak, že je uvedeme do seznamu vlastností nového typu. Ten pak bude obsahovat nejen všechny vlastnosti *všech předků* (až na další diference), ale i všechny *nově definované vlastnosti*.

Příklad z ekonomické oblasti



Příklad

- V našem příkladu je uvedená část ekonomického modelu osob a adres. Vyskytují se zde běžně známé pojmy z ekonomického života jako
- *ekonomický subjekt* (EkonSubjekt),
- *občan* (Obcan),
- *právnická osoba* (PravOsoba),
- *banka* (Banka),
- *osoba v obchodním rejstříku* (OsobaVOR),
- *fyzická osoba podnikatel* (FyzickaOsobaPodn).

Příklad

- Kromě toho, že některé z uvedených objektů jsou současně partnerem (Partner), což přesahuje náš výřez modelu a popisu vlastností (z mnemotechniky identifikátorů), platí:
 - každá právnická osoba je ekonomickým subjektem,
 - každá fyzická osoba podnikatel je ekonomickým subjektem,
 - každá osoba v obchodním rejstříku je ekonomickým subjektem,
 - každá fyzická osoba podnikatel je současně občanem a
 - každá banka je právnickou osobou.

Příklad

concept Obcan/Obcane

inherits

FyzOsoba

Partner

properties

Oznaceni: **string**

Adresat: **string**

RodneCislo: **string** [Key]

RodnePrijmeni: **string**

Prezdivka: **string**

Pozice: **string**

end concept

Příklad

concept EkonSubjekt/EkonSubjekty

inherits

Partner

properties

Adresat: **string**

ObchJmeno: **string**

ICO: **long [Key]**

DIC: **string [Key]**

PlatceDPH: **boolean**

PravniForma: **PravFormaES**

end concept

Příklad

```
concept FyzOsobaPodn/FyzOsobyPodn
```

```
inherits
```

```
Obcan
```

```
EkonSubjekt
```

```
properties
```

```
Oznaceni:string
```

```
Adresat: string
```

```
ObchJmeno: string
```

```
DIC: string [Key]
```

```
end concept
```

Příklad

concept PravOsoba/PravOsoby

inherits

EkonSubjekt

properties

Oznaceni:**string**

ObchJmeno: **string**

DIC: **string** [Key]

end concept

Příklad

concept OsobaVOR/OsobyVOR

inherits

EkonSubjekt

properties

VypisZOR: VypisZOR

end concept

Příklad

concept Banka/Banky

inherits

PravOsoba

properties

SmerovyKod: **integer** [Key]

SwiftCode: **string**

end concept

Příspěvek

- Připomeňme ještě jednou, že každý podtyp obsahuje všechny vlastnosti svých předků a navíc svoje vlastní nové vlastnosti nazvané **příspěvek**.
- Obecně je možno vlastnosti i **měnit** (např. typ) a **vypouštět** (exclude), v praxi se to však nevyužívá.

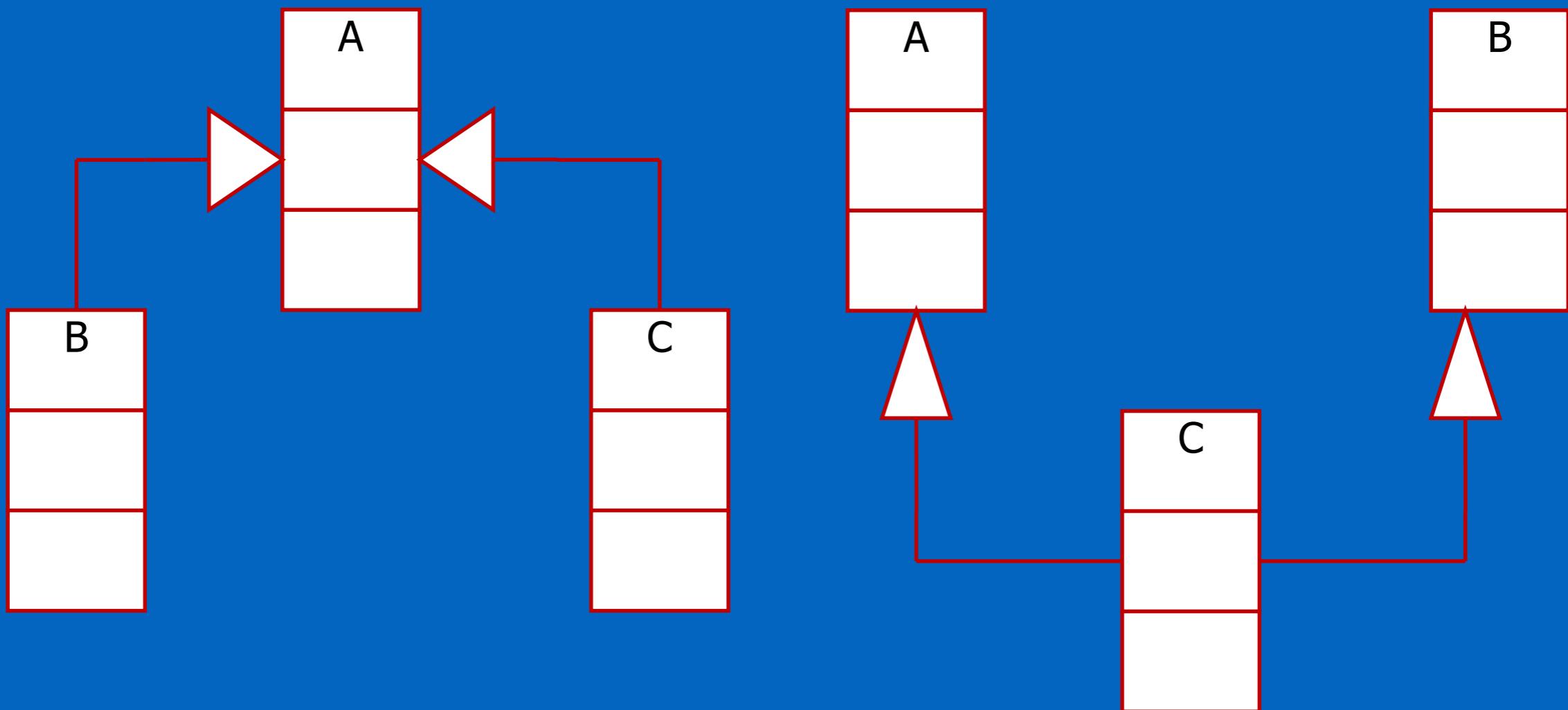
Jednoduchá a vícenásobná dědičnost

- U jednoduché dědičnosti každý následník smí mít **pouze jediného** předka. V grafické podobě dědičnosti to znamená, že ze žádného typu nesmí vycházet více, nežli jedna šipka. Takto zakreslený graf je potom **stromem**.
- U vícenásobné dědičnosti není počet předků omezen. V grafické podobě dědičnosti to znamená, že z každého typu smí vycházet libovolný počet šipek. Takto zakreslený graf je **obecný acyklický graf**.

Jednoduchá a vícenásobná dědičnost

- V žádném případě se v grafu dědičnosti nesmí vyskytovat cyklus, tj. žádný typ nesmí být svým vlastním předchůdcem nebo následníkem. Na obrázku vidíme dva základní grafické nákresy jednoduché a vícenásobné dědičnosti.

Jednoduchá a vícenásobná dědičnost



základní tvar jednoduché dědičnosti

základní tvar vícenásobné
dědičnosti

Typová kompatibilita struktur

- Zavedením hierarchie dědičnosti nad typy jsme stanovili pro každý typ jeho předka (kromě nejvyššího typu v hierarchii). Žádný z typů nemůže být svým vlastním předkem, ani následníkem.

Typová kompatibilita struktur

- Předkové v hierarchii modelují vždy obecnější (generálnější) pojmy a následníci pojmy speciálnější.
- Proto, je-li následníkem osoby např. student, je logické, že každý student je osobou. Nikoliv ovšem naopak. Každá osoba není studentem.
- Podobně, je-li banka následníkem ekonomického subjektu, je každá banka ekonomickým subjektem, ale každý ekonomický subjekt není bankou.

Typová kompatibilita struktur

- Význačnou vlastností je, abychom se na specializovanějšího následníka mohli vždy dívat očima obecnějšího předka. Přeloženo do terminologie typů to znamená, že požadujeme, aby ***každá struktura jistého typu byla zároveň typu všech svých předků***. Struktura není tedy jediného typu, ale je současně více typů, a to svého typu a jeho všech přímých i nepřímých předchůdců.

Typová kompatibilita struktur

- Tedy, pokud máme k dispozici strukturu typu B, která je instancí následníka typu A, pak se může **B vyskytovat všude tam, kde může být A**. Tj. v deklaracích proměnných, hodnotách vlastností, kolekcích, extentech apod.

Typová kompatibilita struktur

- Říkáme, že typ ***B je kompatibilní s typem A***, nikoliv naopak.
- Je třeba si být vždy vědom, že naopak toto tvrzení neplatí. Každá osoba není studentem, každé zvíře není psem a každý ekonomický subjekt není bankou.

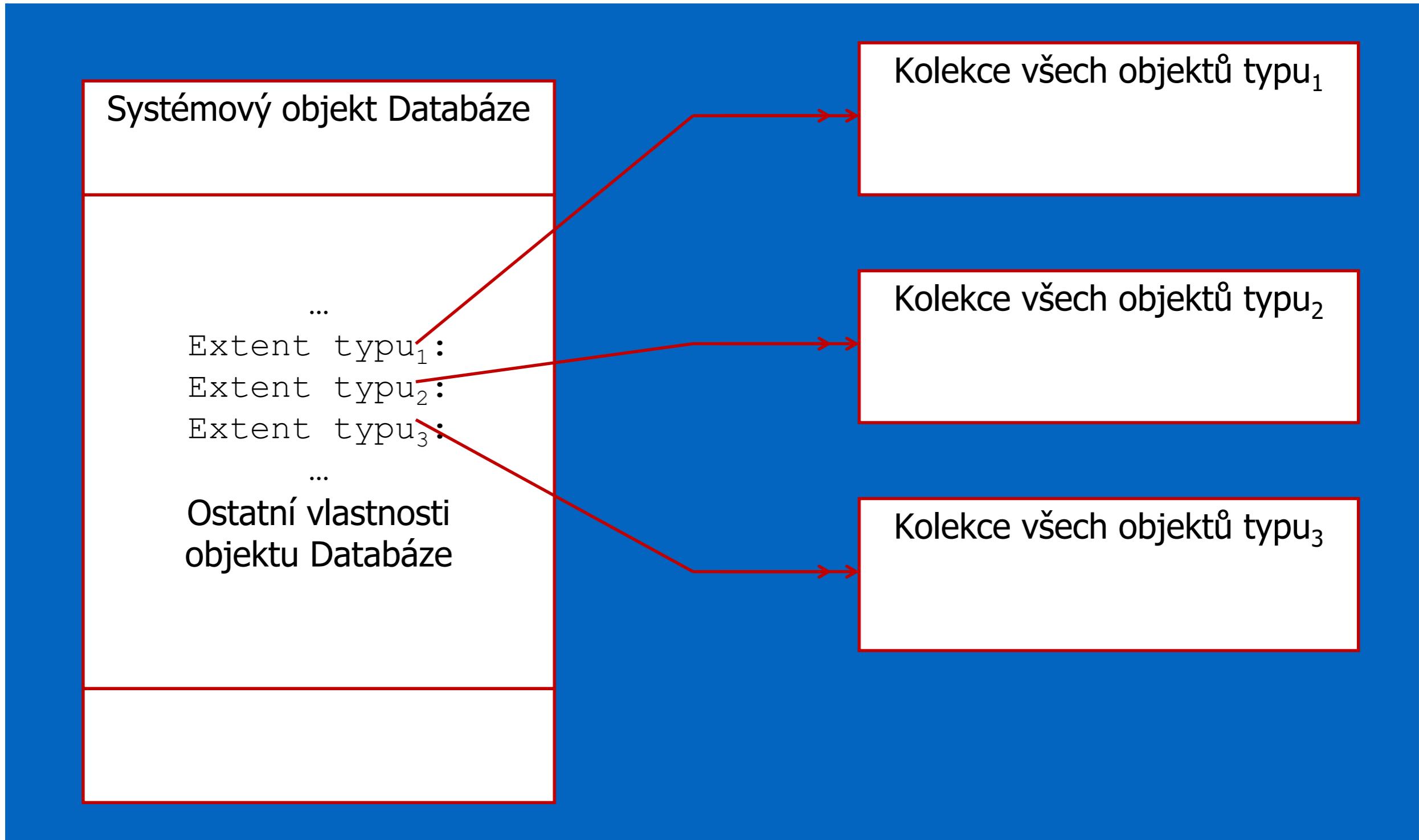
Typová kompatibilita struktur

- Proto deklarujeme-li vlastnosti typu A, resp. kolekce prvků typu A, je třeba vidět, že se v ní budou vyskytovat nejen struktury typu A, ale rovněž všech možných následníků jeho typu. Podobně v kolekci budou nejen prvky typu A, ale i prvky všech následníků typu A.
- Například v kolekci ekonomických subjektů se mohou vyskytovat i právnické osoby, banky, podnikající fyzické osoby i osoby v obchodním rejstříku.

Příklad

- Struktura fyzické osoby má povinnou vlastnost UplneJmeno. Tato vlastnost může být uložena efektivněji.
**concept FyzOsoba/FyzOsoby
properties**
 UplneJmeno: **string**
 Jmeno: **string**
 Prijmeni: **string**
 Pohlavi: Pohlavi
 DatumNarozeni: **date**
 Charakt: **string**
 Oznaceni: **string**
end concept

Obor hodnot, extent



Obor hodnot, extent

- Podobně jako základní typy, jejichž možné hodnoty jsou předem známy, je potom možné znát ***obor všech možných hodnot*** i pro libovolný objekt v databázi. Oborem hodnot pro objekty jistého typu je hodnota kolekce nazývané ***extent***. Představme si, že systém udržuje pro celou databázi jeden ***systémovou strukturu – databázi***, ve které si ukládá hodnoty význačné pro tuto databázi.

Příklad

- Pokud předpokládáme, že pro fyzickou osobu bude systém udržovat extent, napíšeme:

```
concept FyzOsoba/FyzOsoby [Extent]
  properties
    UplneJmeno:string
    Jmeno: string
    Prijmeni: string
    Pohlavi: Pohlavi
    DatumNarozeni: date
    Charakt: string
    Oznaceni:string
  end concept
```

Příklad

- Atribut Extent je typu Boolean a má implicitní hodnotu False.
Je atributem konceptu typu objekt.

Extenty a navigace

- Na rozdíl od relačních databází, kde je nejčastějším prostředkem pro přístup k databázím *dotaz*, je nejčastějším *vstupním bodem* do objektové databáze extent.
- Od něho potom pokračuje navigace po vztazích v databázi. Vytváření extentů je u objektových databází základním prostředkem pro vytváření uživatelské nabídky objektového prohlížeče.

Abstraktní a konkrétní typy

- Při vytváření hierarchie dědičnosti se posléze definované typy struktur rozčlení na dvě kategorie:
 - ty, které slouží jen jako ***stavební kameny*** (vzory) pro vyváření následníků; této kategorie použijeme rovněž, chceme-li ***jednotným způsobem zacházet*** s množinou následníků, které mají množinu společných vlastností (viz kap. 4.4)a
 - ty, které skutečně ***budou mít své instance*** a budou skutečnými strukturami.

Abstraktní a konkrétní typy

- Těm prvním v seznamu říkáme ***abstraktní*** typy. V systému nemůže existovat žádná struktura, která by měla všechny své typy abstraktní. Systém zabrání tomu, aby takovou strukturu bylo možné vytvořit.
- Těm druhým v seznamu říkáme ***konkrétní*** typy.

Příklad

- Výše zmíněný typ modelující ***ekonomický subjekt*** je výhodné deklarovat jako abstraktní. Jeho následníci modelující banku, fyzickou osobu podnikatele apod., jsou již konkrétní.
- Ekonomický subjekt jako takový je pouze stavebním kamenem a nemůže nikdy existovat samostatně. Na všechny konkrétní následníky ekonomického subjektu však můžeme pohlížet jednotným způsobem jako na ekonomický subjekt.

Příklad

- Nicméně *abstraktní typ může mít extent*. Víme, že v extentu jsou i výskyty všech následníků daného datového typu (zde ekonomického subjektu), které již mohou být konkrétní.
- Abstraktnost typu struktury vyjádříme booleovským atributem Abstract konceptu.

Příklad

concept EkonSubjekt/EkonSubjekty [Abstract][Extent]

inherits

Partner

properties

Adresat: **string**

ObchJmeno: **string**

ICO: **long**

DIC: **string** [Key]

PlatceDPH: **boolean**

PravniForma: **PravFormaES**

end concept

Extenty u děděných objektů

- Zdůrazněme, že *v extentu typu A se budou vyskytovat všechny objekty typu A, ale i objekty všech následníků typu A.*
- Naopak, při vzniku není objekt zařazen pouze do extentu typu, který byl pro něj zvolen jako typ vzniku, ale rovněž do všech extentů jeho všech předchůdců.

Extenty u děděných objektů

- Musí platit, že pro každý objekt musí být deklarován alespoň jeden extent. Na druhé straně deklarace extentů pro všechny úrovně dědění může značně zneefektivnit operace vytváření a rušení objektu.

Objektový DB model v Javě

Objektově-relační mapování, Java Persistence API

Java EE – Objektově-relační mapování

- Java Persistence API (JPA)
 - Více implementací (EclipseLink, Hibernate, DataNucleus, ...)
 - Existují alternativy (JDO)
- Primárně počítá s mapováním do relačních tabulek
 - Využívá JDBC
 - Mnoho ovladačů pro různé databáze

Java Bean

```
public class Person
{
    private long id;
    private String name;
    private String surname;
    private Date born;

    public String getName()
    {
        return name;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    ...
}
```

Persistence

- Pomocí anotací vytvoříme z třídy **entitu persistence**

```
@Entity  
@Table(name = "person")  
public class Person  
{  
    @Id  
    private long id;  
    private String name;  
    private String surname;  
    private Date born;
```

Generované ID

```
@Entity  
@Table(name = "person")  
public class Person  
{  
    @Id  
    @GeneratedValue(strategy = IDENTITY)  
    private long id;  
    private String name;  
    private String surname;  
    private Date born;  
    ...}
```

Vztahy mezi entitami

- Anotace @OneToMany a @ManyToOne
- Nastavení mapování
 - V Eclipse pohled JPA
- Kolekce v Javě:
 - Collection<?>
 - List<?> (Vector, ArrayList, ...)
 - Set<?> (HashSet, ...)
 - Map<?, ?> (HashMap, ...)s

Konfigurace presistence

- Soubor persistence.xml
 - Jméno jednotky persistence
 - Odkaz na data source na serveru
 - Případně další parametry pro mapování
 - Např. řízení automatického generování schématu

Implementace business operací

- Enterprise Java Beans (EJB)
 - Zapouzdřují business logiku aplikace
 - Poskytují business operace – definované rozhraní (metody)
 - EJB kontejner zajišťuje další služby
 - Dependency injection
 - Transakční zpracování
 - Metoda obvykle tvoří transakci, není-li nastaveno jinak

Vytvoření EJB

- Instance vytváří a spravuje EJB kontejner
- Vytvoření pomocí anotace třídy
 - @Stateless – bezstavový bean
 - Efektivnější správa – pool objektů přidělovaných klientům
 - @Stateful – udržuje se stav
 - Jedna instance na klienta
 - @Singleton
 - Jedna instance na celou aplikaci

Použití EJB

- Lokální
 - Anotace @EJB – kontejner dodá instanci EJB
- Vzdálené volání – dané rozhraní
 - Rozhraní definované pomocí @Remote

Propojení s JPA

- EJB implementují business operace
 - Často stačí stateless bean
- JPA rozhraní je reprezentováno objektem EntityManager
 - Dodá kontejner pomocí DI

Uložení objektu

```
@PersistenceContext  
EntityManager em;
```

```
Person person = new Person();  
person.setName(„karel“);  
em.persist(person);
```

Změna objektu

```
@PersistenceContext  
EntityManager em;
```

```
person.setName("Karel");  
em.merge(person);
```

Smazání objektu

```
@PersistenceContext  
EntityManager em;
```

```
em.remove(person);
```

Dotazování

- ```
Query q = em.createQuery("...");
q.setParameter(name, value);
q.setFirstResult(100);
q.setMaxResults(50);
q.getResultList();
```

# JPQL dotazy

- ```
SELECT p FROM Person p  
WHERE p.name = "John"
```
- ```
SELECT c FROM Car c
WHERE c.reg LIKE :pref
```
- ```
SELECT  
    NEW myObject(c.type, count(c))  
FROM Car c  
GROUP BY c.type
```

ORM pomocí JPA – doplnění

- Asociace A -> B
 - Třída A obsahuje vlastnost typu B nebo kolekci B (podle kardinality)
 - + potenciálně inverzní (obousměrný) vztah
 - Anotace @OneToOne, @OneToMany, @ManyToOne, @ManyToMany
 - Reprezentace v relační databázi
 - @JoinColumn, @JoinTable
- Slabé entitní množiny
- Dědičnost

Vložené entity

```
@Entity  
public class Person {  
  
    @Id  
    private String idperson;  
    private String name;  
    private String area;  
    private String city;  
    private String zipcode;  
  
    // getters and setters  
  
}
```

- Položky adresy chceme reprezentovať strukturou

Vložené entity (II)

```
@Entity  
public class Person {  
  
    @Id  
    private String idperson;  
    private String name;  
    private Address address;  
  
    // getters and setters  
}  
  
public class Address {  
  
    private String area;  
    private String city;  
    private String zipcode;  
  
    // getters and setters  
}
```

- Jak reprezentovat vztah?
- (@OneToOne je v tomto případě neefektivní)

Vložené entity (III)

```
@Entity
public class Person {

    @Id
    private String idperson;
    private String name;
    @Embedded
    private Address address;

    // getters and setters
}

@Embeddable
public class Address {

    private String area;
    private String city;
    private String zipcode;

    // getters and setters
}
```

- Sloupce area, city, zipcode budou přímo v tabulce Person

Slabá entitní množina

```
@Entity
public class Person {

    @Id
    private String idperson;
    private String name;
    @ElementCollection
    @CollectionTable(
        name=„ADDRESSES“,
        joinColumns=
            @JoinColumn(name=„OWNER“))
    private List<Address> addresses;
    // getters and setters
}

@Embeddable
public class Address {

    private String area;
    private String city;
    private String zipcode;
    // getters and setters
}
```

- Adresy ve zvláštní tabulce ADDRESSES
- Lze i reprezentovat metadata u vztahu:
<https://blog.zvestov.cz/software%20development/2015/04/15/jpa-vazebni-tabulky-s-metadaty>

Základní datový typ

```
@Entity  
public class Person {  
  
    @Id  
    private String idperson;  
    private String name;  
    @ElementCollection  
    @CollectionTable(  
        name=,,ADDRESSES“,  
        joinColumns=  
            @JoinColumn(name=,,OWNER“))  
    @Column(name=,,PHONE_NUMBER“)  
    private List<String> phones;  
  
    // getters and setters  
  
}
```

Pořadí u seznamů

```
@Entity
public class Person {

    @Id
    private String idperson;
    private String name;
    @ElementCollection
    @CollectionTable(
        name=„ADDRESSES“,
        joinColumns=
            @JoinColumn(name=„OWNER“))
    @OrderColumn(name=„ORD“)
    private List<Address> addresses;

    // getters and setters
}

    @Embeddable
    public class Address {

        private String area;
        private String city;
        private String zipcode;
        // getters and setters
    }
}
```

- Nový sloupec ORD v tabulce adres

Pořadí u seznamů

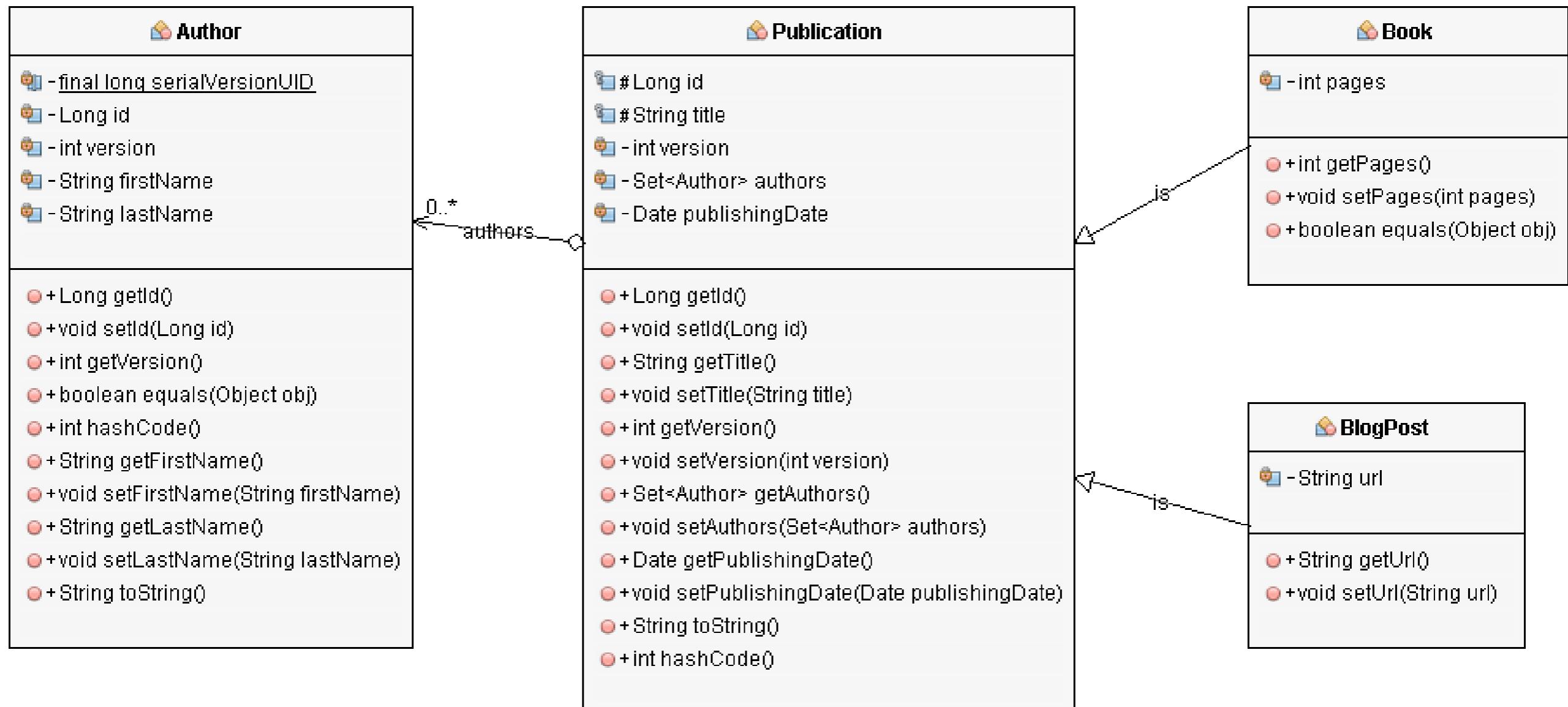
```
@Entity  
public class Person {  
  
    @Id  
    private String idperson;  
    private String name;  
    @ElementCollection  
    @CollectionTable(  
        name=„ADDRESSES“,  
        joinColumns=  
            @JoinColumn(name=„OWNER“))  
    @OrderBy(name=„priority ASC“)  
    private List<Address> addresses;  
  
    // getters and setters  
  
}  
  
@Embeddable  
public class Address {  
  
    private int priority;  
    private String area;  
    private String city;  
    private String zipcode;  
  
    // getters and setters  
}
```

- Totéž i pro @OneToMany

Dědičnost

- Mapování do relačního schématu
 - Vlastnosti nadtídy jsou dostupné v odvozených třídách
 - Jedna tabulka nebo více tabulek?
- Typová kompatibilita
 - Odvozená třída je typově kompatibilní s nadtídou
 - Tvorba extentu jednotlivých tříd?
- Viz např.
<https://thoughts-on-java.org/complete-guide-inheritance-strategies-jpa-hibernate/>

Dědičnost – příklad



Mapped Superclass

```
@MappedSuperclass
public abstract class Publication {

    @Id
    protected Long id;

    protected String title;

    ...
}
```

```
@Entity
public class Book extends Publication {
    private int pages;
    ...
}
```

```
@Entity
public class BlogPost extends Publication {
    private String url;
    ...
}
```

Mapped Superclass – výsledek



- Třída *Publication* není entitou
 - Nemá tabulku v databázi
 - **Nelze specifikovat vztah publikace – autor**
- Vhodné pro efektivní definici sdílených vlastností

Tabulka pro každou třídu

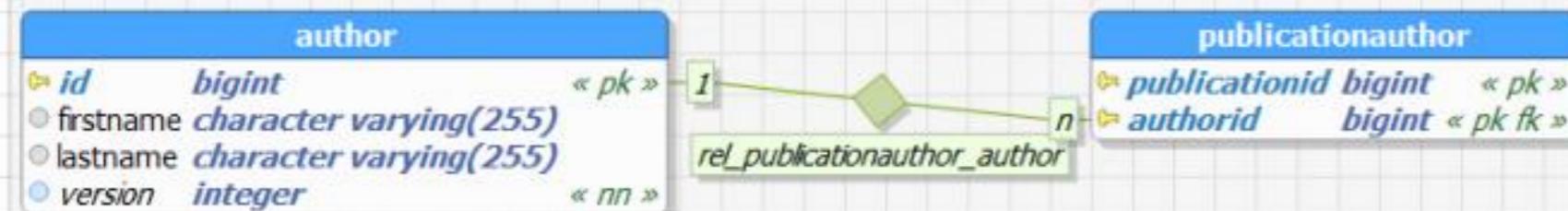
```
@Entity  
@Inheritance(strategy =  
    InheritanceType.TABLE_PER_CLASS)  
public abstract class Publication {  
  
    @Id  
    protected Long id;  
  
    protected String title;  
  
    @ManyToMany  
    @JoinTable(...)  
    private Set<Author> authors;  
  
    ...  
}
```

```
@Entity  
public class Book extends Publication {  
  
    private int pages;  
  
    ...  
}  
  
@Entity  
public class BlogPost extends Publication {  
  
    private String url;  
  
    ...  
}
```

Tabulka pro každou třídu – výsledek

public

SQL off



book	
id	bigint « pk »
publishingdate	date
title	character varying(255)
version	integer
pages	integer

blogpost	
id	bigint « pk »
publishingdate	date
title	character varying(255)
version	integer
url	character varying(255)

- Oddělené tabulky pro jednotlivé třídy
- Třída Publication je entita
 - Lze definovat vztah Publication – Author
- Dotazy nad třídou Publication nejsou efektivní
 - Vede na JOIN nad konkrétními tabulkami
 - Např. for (Publication p : author.getPublications()) { ... }

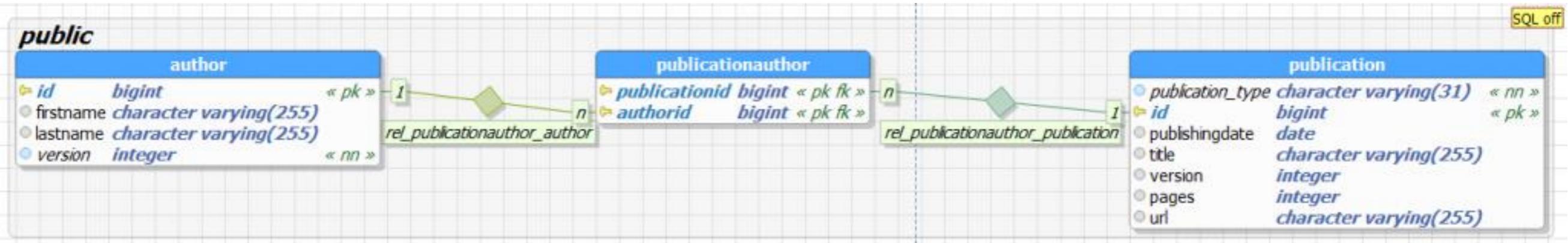
Jediná tabulka

```
@Entity  
@Inheritance(strategy =  
    InheritanceType.SINGLE_TABLE)  
@DiscriminatorColumn(name =  
    "Publication_Type")  
public abstract class Publication {  
  
    @Id  
    protected Long id;  
  
    protected String title;  
  
    @ManyToMany  
    @JoinTable(...)  
    private Set<Author> authors;  
  
    ...  
}
```

```
@Entity  
@DiscriminatorValue("Book")  
public class Book extends Publication {  
  
    private int pages;  
  
    ...  
}
```

```
@Entity  
@DiscriminatorValue("Blog")  
public class BlogPost extends Publication {  
  
    private String url;  
  
    ...  
}
```

Jediná tabulka – výsledek



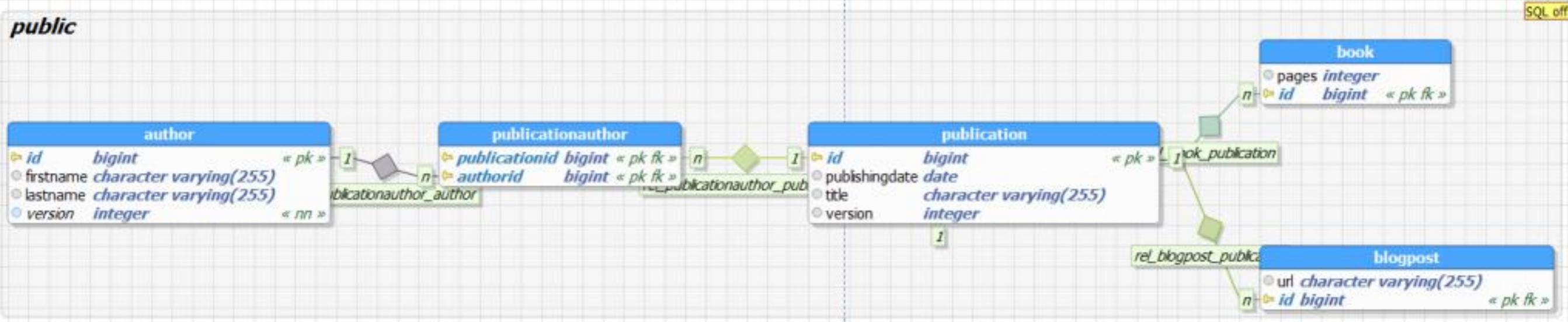
- Jedna tabulka pro všechny odvozené třídy
 - Jeden sloupec slouží jako diskriminátor
- Efektivní dotazování
 - Filtrování podle hodnoty diskriminátoru
 - Snadná reprezentace vztahu Publication – Author
- Hodnoty nevyužitých vlastností jsou nulové
 - Nelze specifikovat *not null* nad vlastnostmi podtříd – omezuje kontrolu integrity dat

Joined

```
@Entity  
@Inheritance(strategy =  
    InheritanceType.JOINED)  
public abstract class Publication {  
  
    @Id  
    protected Long id;  
  
    protected String title;  
  
    @ManyToMany  
    @JoinTable(...)  
    private Set<Author> authors;  
  
    ...  
}
```

```
@Entity  
public class Book extends Publication {  
  
    private int pages;  
  
    ...  
}  
  
@Entity  
public class BlogPost extends Publication {  
  
    private String url;  
  
    ...  
}
```

Joined – výsledek



- Tabulka pro každou třídu včetně Publication
- Snadná reprezentace vztahů, možnost integritních omezení
- Neefektivní dotazování
 - Vždy vede na JOIN více tabulek

Otázky?



Pokročilé informační systémy

Jakarta EE – Business a prezentační vrstva

Ing. Radek Burget, Ph.D.
burgetr@fit.vutbr.cz

Implementace business operací

- Enterprise Java Beans (EJB)
 - Zapouzdřují business logiku aplikace
 - Poskytují business operace – definované rozhraní (metody)
 - EJB kontejner zajišťuje další služby
 - Dependency injection
 - Transakční zpracování
 - Metoda obvykle tvoří transakci, není-li nastaveno jinak

Vytvoření EJB

- Instance vytváří a spravuje EJB kontejner
- Vytvoření pomocí anotace třídy
 - @Stateless – bezstavový bean
 - Efektivnější správa – pool objektů přidělovaných klientům
 - @Stateful – udržuje se stav
 - Jedna instance na klienta
 - @Singleton
 - Jedna instance na celou aplikaci

Použití EJB

- Lokální
 - Anotace @EJB – kontejner dodá instanci EJB
- Vzdálené volání – dané rozhraní
 - Rozhraní definované pomocí @Remote

Contexts and Dependency Injection (CDI)

- Obecný mechanismus pro DI mimo EJB
- Omezuje závislosti mezi třídami přímo v kódu
 - Flexibilita (výměna implementace), lepší testování, ...
- Injektovatelné objekty
 - Třídy, které nejsou EJB
 - Různé vlastnosti pomocí anotací
- Použití objektu
 - Anotace @Inject
 - CDI kontejner zajistí získání a dodání instance

CDI – Injektovatelné objekty

- Téměř jakákoli Javaovská třída
- Scope
 - `@Dependent` – vzniká pro konkrétní případ, zaniká s vlastníkem (default)
 - `@RequestScoped` – trvá po dobu HTTP požadavku
 - `@SessionScoped` – trvá po dobu HTTP session
 - `@ApplicationScoped` – jedna instance pro aplikaci
 - *Pozor na shodu jmen se staršími anotacemi JSF*
- Pokud má být přístupný z GUI (pomocí EL)
 - Anotace `@Named`

CDI – Dodání instancí

- Anotace @Inject
- Vlastnost (field)

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    @Inject
    private NumberGenerator numberGenerator;

}
```

CDI – Dodání instancí

- Konstruktor

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    private NumberGenerator numberGenerator;

    @Inject
    public ItemServlet(NumberGenerator numberGenerator) {
        this.numberGenerator = numberGenerator;
    }
    ...
}
```

CDI – Dodání instancí

- Setter

```
@WebServlet(urlPatterns = "/itemServlet")
public class ItemServlet extends HttpServlet {

    private NumberGenerator numberGenerator;

    @Inject
    public void setNumberGenerator(NumberGenerator numberGenerator)
    {
        this.numberGenerator = numberGenerator;
    }

    ...
}
```

Webové API – REST

REST

- Representational state transfer
- Jednoduchá metoda vzdálené manipulace s daty
- Reprezentuje CRUD (Create-Retrieve-Update-Delete) operace
 - Ale ve skutečnosti přistupujeme k **business vrstvě, ne přímo k datům!**
- Úzká vazba na HTTP
- Nedefinuje formát přenosu dat, obvykle JSON nebo XML

Využití HTTP

- Každá položka dat (nebo kolekce) má vlastní URI. Např.:
 - <http://noviny.cz/clanky> (kolekce)
 - <http://noviny.cz/clanky/domaci/12> (jeden článek)
- Jednotlivé metody HTTP implementují příslušné operace s daty

Metody HTTP

- GET
 - Kolekce: získání seznamu položek
 - Entita: čtení entity (read)
- POST
 - Kolekce: přidání prvku do kolekce (create)
- PUT
 - Kolekce: nahrazení celého obsahu kolekce
 - Entita: zápis konkrétní entity (update)
- DELETE
 - Kolekce: smazání celé kolekce
 - Entita: smazání entity

Formát přenosu dat

- Není specifikován, záleží na službě
 - Obvykle JSON nebo XML (schéma záleží na aplikaci)
- Často více formátu k dispozici
 - Např.
<http://noviny.cz/clanky.xml>
<http://noviny.cz/clanky.json>
 - Využití MIME pro rozlišení typu, HTTP content negotiation

REST a Jakarta EE

- JAX-RS API součástí standardu
- Vytvoření služeb pomocí anotací
- Aplikační server zajistí funkci endpointu
 - Mapování URL a HTTP metod na Javaovské objekty a metody
 - Serializace a deserializace JSON/XML na objekty
- Různé implementace
 - JAX-RS – Jersey (Glassfish), Apache Axis
 - Serializace – Jackson, gson, MOXy, ...

REST v Javě

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("username") String
        userName) {
        ...
        return someXmlString;
    }
}
```

Konfigurace

- JAX-RS servlet ve web.xml
 - Specifikace balíku s REST třídami nebo přímo výčet tříd
 - Specifikace JSON provider (Jackson)
- Alternativně
 - Třída odvozená od javax.ws.rs.core.Application
 - Konfigurace pomocí anotací

Klientská aplikace

- Zasílání REST požadavků
 - Jednotlivé JS frameworky mají vlastní infrastrukturu
- Prezentační logika
 - Navigace
 - Přechody mezi stránkami
 - Výpisy chyb, apod.

Autentizace v REST

- Protokol REST je definován jako **bezstavový**
 - Požadavek musí obsahovat vše, žádné ukládání stavu na serveru
- To teoreticky vylučuje možnost použití sessions pro autentizaci
 - Technicky to ale možné je
 - Problém např. pro mobilní klienty
- Alternativy pro autentizaci:
 - HTTP Basic autentizace (nutné HTTPS)
 - Použití tokenu validovatelného na serveru – např. JWT
 - Složitější mechanismus, např. OAuth

HTTP Basic

- Standardní mechanismus HTTP, využívá speciální hlavičky
- Požadavek musí obsahovat hlavičku **Authorization**
 - Obsahuje jméno a heslo; nešifrované pouze kódované (base64)
 - **Je nutné použít HTTPS**
- Pro nesprávnou nebo chybějící autentizaci server vrací **401 Authorization Required**
 - V hlavičce **WWW-Autenticate** je identifikace oblasti přihlášení
 - Klient tedy zjistí, že je nutná autentizace pro tuto oblast

JSON Web Token (JWT)

- Řetězec složený ze 3 částí
 1. Header (hlavička) – účel, použité algoritmy (JSON)
 2. Payload (obsah) – JSON data obsahující id uživatele, jehopráva, expiraci apod.
 3. Signature (podpis) – pro ověření, že token nebyl podvržen nebo změněn cestou
- Tyto tři části se kódují (base64) a spojí do jednoho řetězce
 - **xxxxx.yyyyy.zzzzz**

Použití JWT

- Klient kontaktuje *autentizační server* a dodá autentizační údaje
 - Stejný server, jaký poskytuje API, nebo i úplně jiný (např. Twitter)
- Autentizační server vygeneruje podepsaný JWT a vrátí klientovi
- Klient předá JWT při každém volání API
 - Nejčastěji opět v hlavičce:
Authorization: Bearer xxxxx.yyyyy.zzzzz
 - API ověří platnost, role uživatele může být přímo v JWT

JWT v Javě

- Součástí standardu Microprofile
 - Ne přímo součást Jakarta EE
 - Ale dostupná na běžných serverech (Payara)
- Lze snadno spojit s JAX-RS
- <https://github.com/javaee-samples/microprofile1.4-samples/tree/master/jwt-auth>

Serverová prezentační vrstva

Java Server Faces (JSF)

Prezentační vrstva

- Webové API + JS tlustý klient
 - Prezentační logika na klientovi
 - Business operace přístupné pomocí API (REST, SOAP, ...)
 - Klientská aplikace může využívat klientský framework
 - Angular, React.js, Vue.js, ...
- Serverový framework
 - Prezentační logika na serveru
 - Server generuje HTML (CSS, JavaScriptový) kód
 - Komponentově orientované frameworky

Prezentační vrstva na serveru

- Funkčnost zajišťuje webový kontejner
- Pro Java EE standardně 2 vrstvy
 1. Facelets
 2. Java Server Faces
- Existují další webové frameworky (Struts, Spring, Vaadin, ...)

Java Servlet

- Třída implementující chování serveru
- Obecný **javax.servlet.GenericServlet**
 - Metody init(), destroy(), service()
- HTTP **javax.servlet.http.HttpServlet**
 - Metody doGet(), doPost(), ...
- Vytváření instancí řídí server
- Jedna instance může obsluhovat více požadavků

Java Server Pages

- Dynamické webové stránky
 - HTML (XHTML, XML, ...) kód
 - Vložený kód v Javě (*scriptlet*)
 - Definované značky
 - Výrazy – unified expression language (EL)
- Umožňuje definovat **knihovny značek** (tag libraries)
 - Chování každé značky implementováno jako třída
 - XML deskriptor knihovny

Příklad JSP

```
<%@ page language="java,"  
    contentType="text/html; charset=ISO-8859-2,"  
    pageEncoding="ISO-8859-2"%>  
<!DOCTYPE html>  
<html>  
<head>  
    <title>My JSP Page</title>  
</head>  
<body>  
    <h1>Hello World!</h1>  
    <p>  
        <%  
            out.println("Current time: " + new java.util.Date());  
        %>  
    </p>  
</body>  
</html>
```

Zpracování JSP stránky

- Stránka se překládá na servlet (třídu)
- Překlad zajišťuje kontejner
 - Skripty – vloženy do stránky
 - Tagy – volání metod příslušných tříd
 - Výrazy – volání evaluátoru

Facelets

- Nástupce JSP od Java EE 6
- Založeno na XML
 - Obvykle XHTML
- Podpora existujících i nových tag libraries
- Šablony komponent a stránek

Facelets

- Překlad XHTML stránek na interní objektovou reprezentaci – **Facelet**
 - Zpracování šablon
 - Související soubory (resources)
- Možnost definice knihoven značek
 - Vystačíme s existujícími
- Zpracování výrazů jazyka EL
 - Přístup k objektům, jejich vlastnostem a metodám

Odbočka: XML namespaces

- Motivace: možnost kombinovat v jednom dokumentu značky z více XML jazyků (např. (X)HTML + SVG)
- Namespace
 - Jmenný prostor, ve kterém jsou jména značek (a atributů) unikátní
 - Je identifikován jednoznačným identifikátorem – URI
 - V rámci jednoho dokumentu má přiřazen zkrácený identifikátor – *prefix*
- Použití v XML
 - Definujeme prefix pomocí zabudovaného atributu `xmlns`
 - Značky z daného jazyka zapisujeme
`<prefix:jméno>...</prefix:jméno>`
 - Jeden jmenný prostor je výchozí – bez prefixu

Příklad XML namespaces

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">

<ui:composition template="template.xhtml">
    <ui:define name="content">
        <h2>Welcome</h2>
        <p><a href="person_list.xhtml">Simple
           forms demo</a></p>
    </ui:define>
</ui:composition>
```

Java – Namespaces

- Facelets tags

`xmlns:ui="http://xmlns.jcp.org/jsf/facelets"`

- JSF Core – obecné definice

`xmlns:f="http://xmlns.jcp.org/jsf/html"`

- JSF HTML – obsah stránky

`xmlns:h="http://xmlns.jcp.org/jsf/core"`

Šablony

- Šablona definuje vzorovou stránku
 - Proměnné části jsou označeny pomocí <ui:insert name=,,>
- Jednotlivé stránky využívají šablonu
 - Místo <body> se použije <ui:composition> a obsah proměnných částí se definuje pomocí <ui:define>.

Java Server Faces

- MVC framework nad Facelets
- Značky pro generování základních prvků uživatelského rozhraní
 - Rozšiřitelné knihovny komponent
- Řídící servlet implementující chování

Nadstavbové knihovny

- [PrimeFaces](#)
- [Apache MyFaces](#)
- OmniFaces
- RichFaces (JBoss, není dále vyvíjen)
- ...

Aplikace nad JSF

- Založeno na vzoru Model-View-Controller
- Model – Business vrstva (služby)
- View – JSF kód
 - XHTML kód využívající komponenty
 - Managed beans
- Controller – FacesServlet, konfigurovatelný

Expression language

- Podobný JavaScriptu
- Zápis v # { ... } nebo \$ { ... }
(zpožděné vs. okamžité vyhodnocení)
- Abstrakce sdílených objektů v libovolném kontextu

Sdílené objekty a EL

- `# { objekt }` – vyhledá objekt daného jména v kontextu stránky, požadavku, session a aplikace
- Metody: `# { customer.sayHello }`
- Vlastnosti: mapují se na set / get
 - `# { customer.name }`

Logika uživatelského rozhraní

- EJB služby – model
- Managed beans
 - Spravované objekty přístupné přes EL
 - Implementují chování
- Validators
 - Kontrola vstupních dat
- Converters
 - Převod dat na řetězec a zpět

Managed beans

- CDI beans
- Vlastnosti přístupné pomocí get() a set()
- Vlastnosti vázané na prvky GUI
- Scope:
 - none – vždy nový objekt je vytvořen na požádání
 - request – pro každý požadavek bude vytvořena nová instance
 - session – zachován po celou dobu sezení
 - application – po dobu běhu kontejneru

Validátory

- Položky povinné a nepovinné
 - Atribut required
- Zabudované
 - f:validateDoubleRange
 - f:validateLongRange
 - f:validateLength
- Uživatelské validátory
 - f:validator

Konvertory

- Zabudované konvertory
 - <f:convertDateTime>
 - <f:convertNumber>
- Obecný konvertor podle identifikátoru
 - <f:converter>
- Zabudované konvertory javax.faces.*
 - Boolean, Byte, Character, DateTime, Double, Long, ...
- Uživatelské konvertory

Definice chování

- Každá stránka produkuje výsledek ve formě řetězce (outcome)
 - Statický (zadaný konstantou)
 - Dynamický (generovaný metodou)
 - Často po provedení nějaké akce
- Přechody mezi stránkami externě v XML souboru faces-config.xml

Autentizace

- Informace o přihlášení reprezentované session beanem
- Filtrování požadavků filtrem definovaným ve WEB-INF/web.xml
 - Alternativně pouze označeným anotací @WebFilter

Alternativa: Vaadin

- Komponentový server-side framework
- Veškerá specifikace GUI v Javě
 - Využívá existujících komponent
- Servlet zajišťující komunikaci s prohlížečem
 - Neřeší programátor
- <https://github.com/vaadin/framework>
- <https://vaadin.com/framework>

Otázky?



Pokročilé informační systémy

Architektury a principy napříč platformami

Ing. Radek Burget, Ph.D.
burgetr@fit.vutbr.cz

Třívrstvá architektura

- Java EE umožnuje implementovat *monolitický IS s třívrstvou architekturou*:
 1. Databázová vrstva
 - JPA – definice entit, persistence (*PersistenceManager*)
 - Alternativně: Relační databáze (JDBC), NoSQL (MongoDB), ...
 2. Logická (business) vrstva
 - Enterprise Java Beans (EJB) nebo CDI beans
 - Dependency injection – volné propojení
 3. Prezentační vrstva
 - Webové rozhraní (JSF) nebo API (REST, JAX-RS)

Další platformy – přehled

- Java
 - Existuje mnoho možností kromě „standardní“ J EE
- .NET (Core / Framework)
 - Mnoho řešení na všech vrstvách
- PHP
 - Různé frameworky, důraz na webovou vrstvu
- JavaScript
 - Node.js + frameworky, důraz na web a mikroslužby
- Python, Ruby, ... - podobné principy

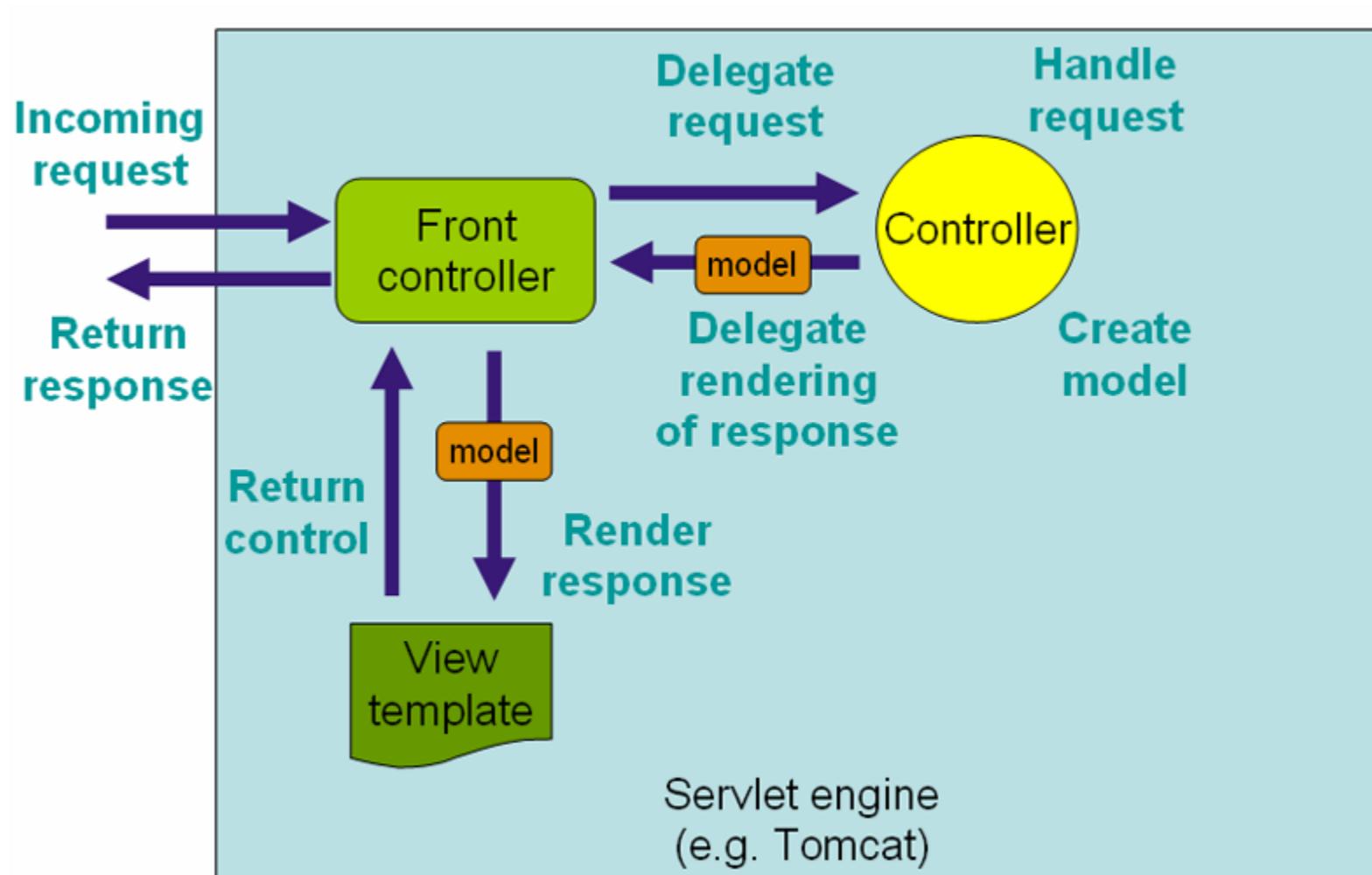
Java – alternativy

- Databázová vrstva
 - Hibernate ORM – implementuje JPA, ale i vlastní API
 - NoSQL databáze – Hibernate OGM, EclipseLink, ...
- Business vrstva
 - Spring framework – alternativa EJB pro dependency injection, transakce, správa sezení, ...
- Prezentační vrtstva
 - Spring MVC (controllers + JSP / Thymeleaf...)
 - Struts, Play!, ...
- <https://www.dailyrazor.com/blog/best-java-web-frameworks/>

Spring

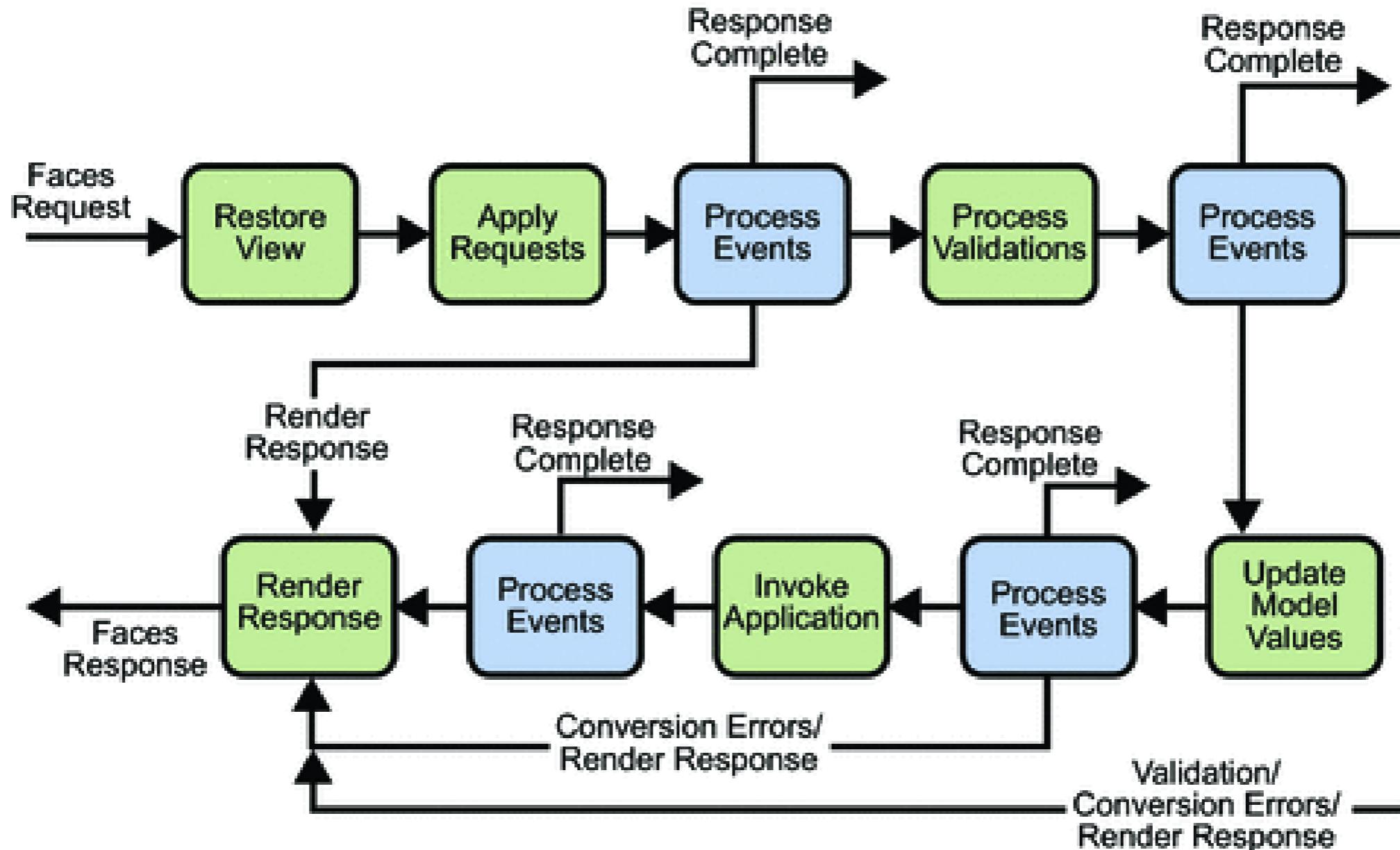
- Vznikl jako alternativa k EJB
 - Využití POJO místo (tehdy složitých) EJB
 - Omezení požadavků na infrastrukturu
- Modulární struktura, mnoho součástí
- Dependency injection
 - Podobně jako v J EE, anotace @Bean, @Autowire, ...
 - Opět field, constructor, setter injection
- Spring MVC
 - Tradiční MVC přístup, bližší ostatním frameworkům
 - Ukázková aplikace
<https://github.com/spring-projects/spring-mvc-showcase>

Zpracování požadavku Spring MVC



- Handler matching – anotace v controller třídách, vrací popis view (různé formáty) nebo přímo výsledný obsah
- View matching – výběr view podle výsledku (pokud je)
- <https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html>

Pro srovnání: JSF



Spring Boot

- Přístup „Vše je v aplikaci“ (včetně serveru)
 - Na rozdíl od Java EE – „Server umí vše“ (thin WARs)
- Usnadňuje vytvoření aplikace a konfiguraci závislostí
 - Maven nebo Gradle šablony
 - Spring moduly (MVC, Security, ...), Thymeleaf, JPA,...
- Snadné vytvoření funkční aplikace
 - Třída reprezentující celou aplikaci
 - Konfigurace pomocí anotací
 - Spustitelná main() metoda
- <https://www.baeldung.com/spring-boot-start>

.NET

- .NET Core / .NET Framework
- Databázová vrstva
 - Entity Framework, (LINQ, Dapper, ...)
<https://docs.microsoft.com/cs-cz/ef/core/modeling/>
- Business vrstva (služby)
 - ASP.NET Core (dependency injection, middleware)
<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/>
- Webová vrstva
 - Razor (MVVM) – two-way data binding, HTML
<https://dotnet.microsoft.com/apps/aspnet/web-apps>
 - ASP.NET Core MVC – logika+model v C#, view v HTML, REST, ...
<https://docs.microsoft.com/cs-cz/aspnet/core/tutorials/first-mvc-app/start-mvc>

Entity Framework – entita

```
[Table("Product")]
public class Product
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public int CategoryId { get; set; }

    [Required, StringLength(50)]
    public string Name { get; set; }

    [Required]
    [DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [Required]
    public int Stock { get; set; }

    [ForeignKey("CategoryId")]
    public virtual Category Category { get; set; }
}
```

PHP

- PHP je rozšiřující modul HTTP serveru
 - Žádný trvale běžící kontejner
 - + stabilita řešení
 - - efektivita, možnost udržovat stav napříč požadavky
- PHP Frameworky
 - Laravel – (MVC) <https://laravel.com/>
 - Symfony – (MVC) <https://symfony.com/>
 - Nette – (MVP) <https://nette.org/>
 - ...
- Správa závislostí – composer

Databázová vrstva

- Různé vlastní přístupy
- Laravel
 - Fluent query builder – specifikace SQL dotazů v PHP
`DB::table('users')->where('name', 'John')->first();`
 - Eloquent ORM
- Nette
 - Nette database – parametrizovatelné SQL dotazy
- **Doctrine** – pokročilé ORM, podobné JPA
 - Integrovatelné do všech frameworků

Doctrine: Entita

```
<?php

use Doctrine\ORM\Annotation as ORM;

/*
 * @ORM\Entity @ORM\Table(name="products")
 */
class Product {

    /** @ORM\Id @ORM\Column(type="integer")
     * @ORM\GeneratedValue **/
protected $id;

    /** @ORM\Column(type="string") **/
protected $name;

    // ... (other code)

}
```

Doctrine: Uložení objektu

```
$product = new Product();  
  
$product->setName(„Tatranky”);  
  
$entityManager->persist($product);  
  
$entityManager->flush();  
  
echo "Created Product with ID "  
. $product->getId() . "\n";
```

Business vrstva v PHP

- Obvykle v podobě služeb – services
- Framework poskytuje DI kontejner, který registruje služby
 - Procedurálně v PHP nebo externí konfigurační soubor
- Při vytváření controlleru framework dodá závislosti
 - Obvykle *constructor injection*
- Příklady
 - Laravel <https://laravel.com/docs/5.8/container#resolving>
 - Symfony
https://symfony.com/doc/current/components/dependency_injection.html
 - Nette <https://doc.nette.org/cs/2.4/dependency-injection>

Zpracování požadavku v PHP

1. Požadavek na kořenový dokument (index.php)
2. Bootstrapping frameworku
 - Načtení konfigurace
 - Inicializace součástí, rozšíření, služeb (DI)
 - Obnova session
3. Dekódování parametrů požadavku
 - Směrování požadavku – routing
4. Volání aplikáční logiky
 - Vytvoření instance controlleru
 - Volání metody podle požadavku (handler)
5. Vytvoření odpovědi (view rendering)

Zpracování požadavku v PHP – příklady

- Přiřazení controllerů k URL je definováno odděleně
`Route::get('user/{id}', 'UserController@show')`
- Controller konfiguruje a vrací view
- <https://laravel.com/docs/5.8/controllers>

Zpracování požadavku v PHP – Symfony

- Controller je přiřazen k URL pomocí anotací
- Vrací objekt Response
 - Různé druhy, případně včetně obsahu, přesměrování,
...
 - Případně sám zajišťuje použití šablon
- <https://symfony.com/doc/current/controller.html>

Zpracování požadavku v PHP – Nette

- Požadavek vyřizuje *presenter* (metoda renderXyz (params))
- Presenter si sám spravuje model (není formalizováno)
- Předává data do view (template) nebo přímo odesílá odpověď (sendResponse ())
- <https://doc.nette.org/cs/2.4/presenters>

JavaScript – node.js

- Standardní řešení pro JS na serveru
- V8 JavaScript Engine + knihovny
- Procedurální implementace zpracování HTTP požadavků
 - Obdobně jako servlety
- Ukázka:
<https://nodejs.org/en/docs/guides/getting-started-guide/>
- Správce balíků npm
 - Jednoduchá instalace závislostí (knihoven)

Databázová vrstva

- Knihovny pro podporu relačních DB serverů k dispozici v rámci platformy node.js
 - Např. MySQL
<https://expressjs.com/en/guide/database-integration.html#mysql>
- Existují i ORM řešení
- Např. Sequelize
 - Podpora MySQL, SQLite, PostgreSQL, MSSQL
 - <https://github.com/sequelize/express-example>

Sequelize

```
const User = sequelize.define('user', {
  firstName: {
    type: Sequelize.STRING
  },
  lastName: {
    type: Sequelize.STRING
  }
});

// Vytvoří tabulku
User.sync({force: true}).then(() => {
  // Table created
  return User.create({
    firstName: 'John',
    lastName: 'Hancock'
  });
});

// Dotaz
User.findAll().then(users => {
  console.log(users)
})
```

Business vrstva

- Implementace v JS, žádné standardní řešení
- Modularizace řešena na úrovni node.
- Případné DI řešení
 - <https://www.npmjs.com/package/node-dependency-injection>

Webová vrstva

- Velké množství frameworků s různými přístupy
 - <http://nodeframework.com>
- Express
 - Mapování HTTP požadavků na funkce v JS
<http://expressjs.com/en/guide/routing.html>
 - Views pomocí několika template engines
<http://expressjs.com/en/guide/using-template-engines.html>
- Full stack frameworky
 - Těsnější integrace s frontendem, např. Meteor

Mikroslužby (Microservices)

Architektura orientovaná na služby

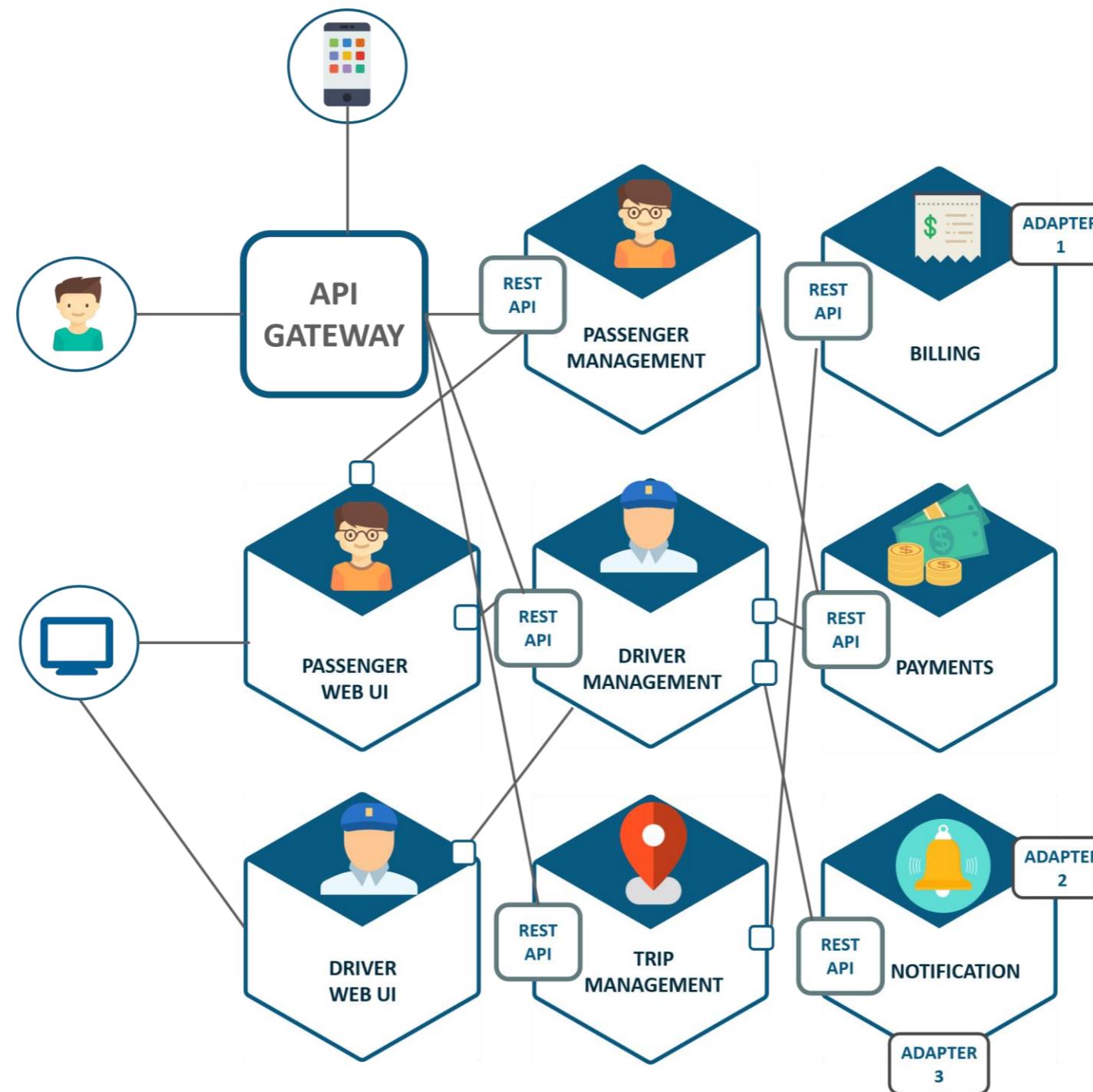
Monolitická architektura

- Jedna aplikace
 - Jedna databáze, webové (aplikační) rozhraní
 - Business moduly – např. objednávky, doprava, sklad, ...
- Výhody
 - Jednotná technologie, sdílený popis dat
 - Testovatelnost
 - Rychlé nasazení – jeden balík
- Nevýhody
 - Rozměry aplikace mohou přerůst únosnou mez
 - Neumožňuje rychlé aktualizace částí, reakce na problémy
 - Pokud použité technologie zastarají, přepsání je téměř nemožné

Mikroslužby

- Aplikace je rozdělena na malé části
 - Vlastní databáze (nepřístupná vně)
 - Business logika
 - Aplikační rozhraní (REST)
- Typicky malý tým vývojářů na každou část (2 pizzas rule)
- Výhody
 - Technologická nezávislost
 - Snadné aktualizace, kontinuální vývoj
- Nevýhody
 - Testovatelnost – závislosti na dalších službách
 - Režie komunikace, riziko nekompatibility, řetězové selhání, ...

Mikroslužby (příklad: Uber)



Vlastnosti mikroslužby

- Vnější API
 - Dostatečně obecné – reprezentuje logiku, ne např. schéma databáze (která je skrytá)
- Externí konfigurace
- Logování
- Vzdálené sledování
 - Telemetrie – metriky (počty volání apod.), výjimky
 - Sledování živosti (Health check)

V čem tvořit mikroslužby?

- V čemkoliv – spojovacím bodem je pouze API
- Node.js (+ express + MongoDB)
 - Populární rychlé řešení
- Java
 - Spring Boot
 - Ultralehké frameworky
Např. Spark - <https://github.com/perwendel/spark>
 - Microprofile

Eclipse Microprofile

- <https://microprofile.io/>
- Standard založený na Java (Jakarta) EE
 - Podmnožina rozhraní JEE (např. CDI, JAX-RS, JSON-B)
 - Specifická rozhraní pro mikroslužby
 - Config, Health Check, Metrics, JWT Auth, REST Client, ...
- Příklad aplikace
 - <https://github.com/cicekhayri/payara-micro-javaee-crud-rest-starter-project>

Micropattern – další API

- Config
 - Externí konfigurace služby – zdroje, priority, ...
- Fault Tolerance
 - Řešení výpadků kvůli závislosti služeb
 - Timeout, Retry, ...
- Health Check
 - Vzdálené zjištění živosti mikro služby

Micropattern – další API (II)

- JWT Authentication
- Metrics
 - Statistiky o využití služby – vzdálené měření výkonu
- OpenAPI
 - Generování formalizované dokumentace API služby
- REST Client
- Příklady
 - <https://github.com/payara/Payara-Examples/tree/master/micropattern>

Aplikační rozhraní

Alternativy k REST

Standardizace API

- Předchůdci REST
 - Snaha o maximální standardizaci volání serverových služeb přes HTTP
 - Vznik komplikovaných standardů webových služeb (SOAP atd.)
 - Obtížně použitelné bez podpůrných technologií – omezení na konkrétní implementační platformy
- REST – zjednodušení v reakci na komplikovanost WS
 - Flexibilita, ale žádný standard – mnoho ad hoc řešení
 - GraphQL

XML-RPC

- Předchůdce webových služeb
- Jednodušší – stále používané
- Definované XML zprávy pro předání parametrů i výsledku
- Podpora datových typů včetně polí, seznamů a struktur

XML-RPC volání

```
<?xml version="1.0"?>

<methodCall>

    <methodName>trida.jePrvocislo</methodName>

    <params>

        <param>

            <value><int>1345</int></value>

        </param>

    </params>

</methodCall>
```

XML-RPC výsledek

```
<?xml version="1.0"?>

<methodResponse>

    <params>

        <param>

            <value><boolean>0</boolean></value>

        </param>

    </params>

</methodResponse>
```

Volání v PHP

```
function xmlrpc($url, $method, $params, $types = array(), $encoding = 'utf-8') {  
    foreach ($types as $key => $val) {  
        xmlrpc_set_type($params[$key], $val);  
    }  
    $context = stream_context_create(array('http' => array(  
        'method' => "POST",  
        'header' => "Content-Type: text/xml",  
        'content' => xmlrpc_encode_request($method, $params,  
array('encoding' => $encoding))  
    )));  
    return xmlrpc_decode(file_get_contents($url, false, $context), $encoding);  
}
```

Webové služby (Web Services)

- Vzdáleně volané podprogramy umístěné na serveru
- Volání pomocí protokolu HTTP
 - Předání vstupních parametrů
 - Vrácení výsledku
- Předávají se XML data definovaná protokoly
 - WSDL – popis rozhraní služby
 - SOAP – volání služby

Popis rozhraní služby: WSDL

- Web Services Description Language
- Platformově nezávislý popis rozhraní
 - XML dokument
 - Využíva XML Namespaces a XML Schema
- Definuje
 - Názvy funkcí
 - Jejich parametry
 - Způsob volání (vstupní URL)

Příklad popisu služby

```
<message name="jePrvocisloRequest">
    <part name="cislo" type="xsd:long"/>
</message>
<message name="jePrvocisloResponse">
    <part name="return" type="xsd:boolean"/>
</message>
<portType name="Cisla">
    <operation name="jePrvocislo" parametrOrder="cislo">
        <input message="m:jePrvocisloRequest"
name="jePrvocisloRequest"/>
            <output message="m:jePrvocisloResponse"
name="jePrvocisloResponse"/>
    </operation>
</portType>
```

Příklad popisu služby (II)

```
<binding name="cislaSoapBinding" type="m:Cisla">  
    ...  
</binding>  
<service name="CislaService">  
    <port binding="m:cislaSoapBinding" name="cisla">  
        <wsdlsoap:address location="http://nekde.cz/cisla" />  
    </port>  
</service>
```

Použití WSDL

- Na základě popisu lze automaticky generovat rozhraní v cílovém jazyce
 - „Stub“ - zástupnou metodu, která implementuje volání skutečné metody přes HTTP
- Rovněž lze generovat WSDL popis z rozhraní v cílovém jazyce
- Současné vývojové nástroje umožňují vytvoření WS z funkce „jedním kliknutím“
 - Např. v Eclipse

Volání služby: SOAP

```
<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    env:encodingStyle="
        http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xs="http://www.w3.org/1999/XMLSchema"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
<env:Header/>
<env:Body>
    <m:jePrvocislo xmlns:m="urn:mojeURI">
        <cislo xsi:type="xs:long">1987</cislo>
    </m:jePrvocislo>
</env:Body>
</env:Envelope>
```

Odpověď služby

```
<env:Envelope
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <env:Body>
        <ns1:jePrvocisloResponse
            xmlns:ns1="urn:mojeURI"
            env:encodingStyle="
                http://schemas.xmlsoap.org/soap/encoding/">
            <return xsi:type="xsd:boolean">true</return>
        </ns1:jePrvocisloResponse>
    </env:Body>
</env:Envelope>
```

Komplexní příklady

- WSDL
<http://www.w3.org/TR/wsdl20-primer/#basics-greath-scenario>
- SOAP
<http://www.w3.org/TR/soap12-part0/#L1165>
- W3C Web Services Activity
<http://www.w3.org/2002/ws/#documents>

Vyhledání webových služeb

- Idea centrálního registru
 - Protokol UDDI (Universal Description, Discovery and Integration)
 - Poskytovatelé ukládají WSDL popisy, klienti prohledávají
- Přehled služeb daného poskytovatele
 - WSIL (Web Services Inspection Language)
 - Seznam služeb v souboru /inspection.wsil

Implementace WS

- Java EE
 - JAX-WS API součástí standardu
 - Vytvoření webových služeb pomocí anotací
 - Běží na JavaEE aplikačním serveru
- PHP
 - Rozšíření SOAP
 - Třídy SoapServer, SoapClient, ...

Java EE

- Součástí Java EE je *Java API for XML Web Services (JAX-WS)*
- Server
 - Definice služeb pomocí anotací tříd a metod
 - Automaticky generuje WSDL popis
- Klient
 - Automatické generování proxy třídy z WSDL popisu

Java EE Implementace

```
package helloservice.endpoint;

import javax.jws.WebService;
import javax.jws.webMethod;

@WebService
public class Hello {

    public Hello() { }

    @WebMethod
    public String sayHello(String name) {
        return "Hello, " + name + ".";
    }
}
```

PHP

- Rozšíření SOAP
- Server
 - Třída **SoapServer**
 - Registruje třídy a metody implementující službu
- Klient
 - Třída **SoapClient**
 - Zpracuje WSDL a zpřístupní vzdálené metody

PHP Soap Server

```
<?php

function sayHello($name) {
    return "Hello, " . $name;
}

$server = new SoapServer(null,
    array('uri' => "urn://helloservice/endpoint"));

$server->addFunction('sayHello');
$server->handle();

?>
```

PHP SOAP Klient

```
$client = new SoapClient(null,  
    array('location' => "http://.../simple_server.php",  
          'uri'      => "urn://my/namespace") );  
  
$name = "Karel";  
$result = $client->  
    __soapCall("sayHello", array($name));  
  
print $result;
```

PHP s WSDL

```
$soap = new SoapClient(  
    'http://api.search.live.net/search.wsdl');  
  
print_r($soap->__getFunctions());  
  
$ret = $soap->Search(...);
```

Popis služeb v REST

- Obdoba WSDL pro REST
- WADL (Web Application Description Language)
 - Založený na XML
 - Podpora v Javě – např. Payara
- OpenAPI <https://swagger.io/specification/>
 - Používá YAML (alternativně JSON)
 - Např. Payara <http://localhost:8080/openapi/>

GraphQL

- <https://graphql.org/>
- Motivace: klient (klienti) potřebují v různých situacích různá data
 - Např. stránka „seznam osob“ vs. „detail osoby“
 - REST endpoint vrací vždy stejnou strukturu
 - Redundance dat (nevyužijeme všechna data)
 - Více dotazů ((ne)efektivita, složitější logika klienta)
- Řešení GraphQL
 - Popis datového modelu API
 - Dotaz na API specifikuje požadovaný tvar odpovědi

GraphQL – datový model

- Datový model API (ne nutně serverové aplikace)
- Jednoduché datové typy: Int, Float, String, Boolean, ID, enum
- Uživatelské typy (*types*) = struktury
 - Vlastnosti (parametrizovatelné) – jméno, parametry, typ
 - Typy jednoduché, struktury (vztahy), kolekce
- Speciální typy reprezentující volání API (*root types*)
 - Query – čtení dat
 - Mutation – změna dat
- GraphQL Schema Definition Language (SDL)

GraphQL – definice typů

```
type Person {  
    name: String!  
    age: Int!  
    posts: [Post!]!  
}
```

```
type Car {  
    type: String!  
    reg: String!  
    owner: Person!  
}
```

```
type Query {  
    allPersons: [Person!]!  
    findPerson(name: String!): Person!  
}
```

```
type Mutation {  
    createPerson(name: String!, age: Int!): Person!  
}
```

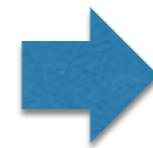
GraphQL – dotazy

```
{  
  allPersons {  
    name  
  }  
}
```



```
{  
  "allPersons": [  
    { "name": "Jan" },  
    { "name": "Karolína" },  
    { "name": "Alice" }  
  ]  
}
```

```
{  
  findPerson(name: "James") {  
    name  
    age  
    cars {  
      type  
    }  
  }  
}
```



```
{  
  "findPerson": {  
    "name": "James",  
    "age": 28,  
    "cars": [  
      { type: "Fiat" },  
      { type: "Tesla" }  
    ]  
  }  
}
```

GraphQL – modifikace

```
mutation {
  createPerson(name: "Bob", age: 36) {
    name
    age
  }
}
```



```
"createPerson": {
  "name": "Bob",
  "age": 36
}
```

GraphQL přes HTTP

- Jediné endpoint URL

- Odeslání přes GET

`http://myapi/graphql?query={me{name}}`

- Odeslání přes POST

- Data application/json

`{"query": "{me{name}}"}`

- Data application/graphql

`{me{name}}`

Otázky?

Procesy, pokročilé modely procesů, cesta k workflow

INTEGRITA A KONZISTENCE

Databázová integrita

- databáze vyhovuje zadaným pravidlům – ***integritním omezením (IO)***. Tato integritní omezení bývají nejčastěji součástí definice databáze a za jejich splnění zodpovídá **systém řízení báze dat (SŘBD)**
- mohou být zadána výrazem (*deklarativně*) nebo programem (*procedurálně*)
- Integritní omezení se mohou týkat *jednotlivých hodnot* vkládaných do polí databáze (například známka z předmětu musí být v rozsahu 1 až 5)

Databázová integrita

- může jít o podmínu na *kombinaci hodnot* v některých polích jednoho záznamu (například datum narození nesmí být pozdější než datum úmrtí).
- může se týkat *i celé množiny záznamů daného typu*
- může jít o požadavek na *unikátnost hodnot daného pole či kombinace polí* v rámci celé množiny záznamů daného typu, které se v databázi vyskytují (například číslo průkazu v záznamech o osobách).

Integrita datovým typem

- *Datový typ* je množina hodnot spolu s operacemi, které je možné nad těmito hodnotami provádět.
- je vlastností jisté části modelu (proměnné, části jiného datového typu apod.) a *omezuje* její použití (jde vlastně o *integritní omezení* části modelu). Omezuje je tak, že tato část modelu může:
 - *nabývat pouze jisté množiny hodnot* a
 - *může s ní být prováděna pouze jistá omezená množina operací*.
- Výhodou zavedení datového typu pro jistou část modelu je zejména možnost kontrolovat (a to nejčastěji předem), zda se s touto částí zachází korektně (zda ukládaná hodnota je správná a použitá operace správně použita).

Konzistence

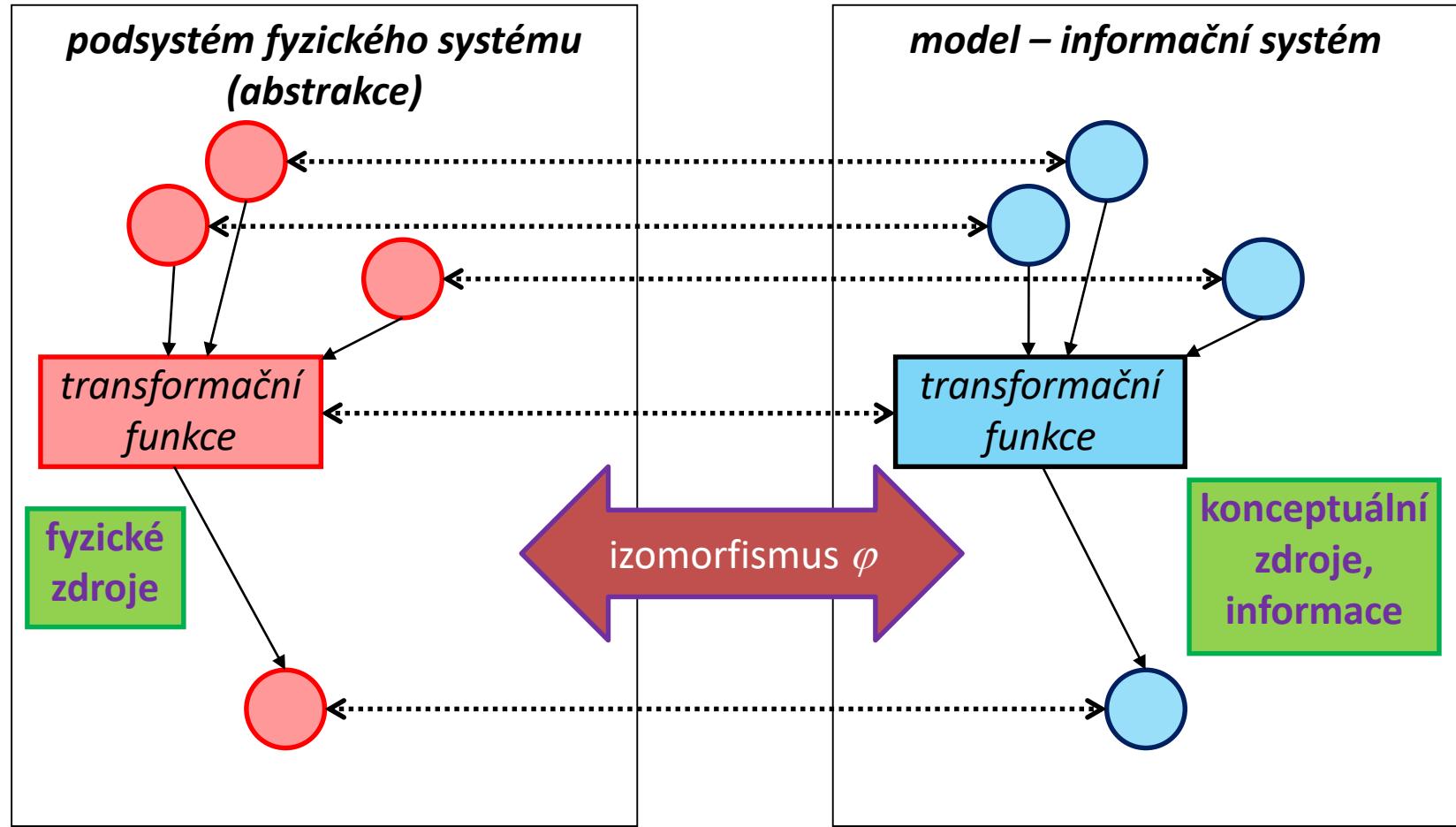
- DB musí splňovat všechna *integrální omezení* (IO)
- Udržování *interní konzistence* (redundantních, vícenásobně uložených/replikovaných dat v distribuovaných systémech)
- Dodržování *pravidel daných modelovaným systémem*

OLTP JAKO MODEL

Izomorfismus

- *Izomorfismus* je zobrazení mezi dvěma matematickými strukturami, které je vzájemně jednoznačné (bijektivní) a zachovává všechny vlastnosti touto strukturou definované.
- Jinými slovy, každému prvku první struktury odpovídá právě jeden prvek struktury druhé a toto přiřazení zachovává vztahy k ostatním prvkům.

OLTP jako model



Nezbytnost abstrakce

- Není možné modelovat všechny zdroje i procesy fyzického systému. Vždy se vybírají jen ty, které jsou pro úroveň řízení, pro kterou OLTP budujeme, podstatné – modelujeme ***pod systém*** původního fyzického systému – ***abstrahujeme***
- OLTP je proto vždy modelem jisté ***abstrakce původního fyzického vzoru.***

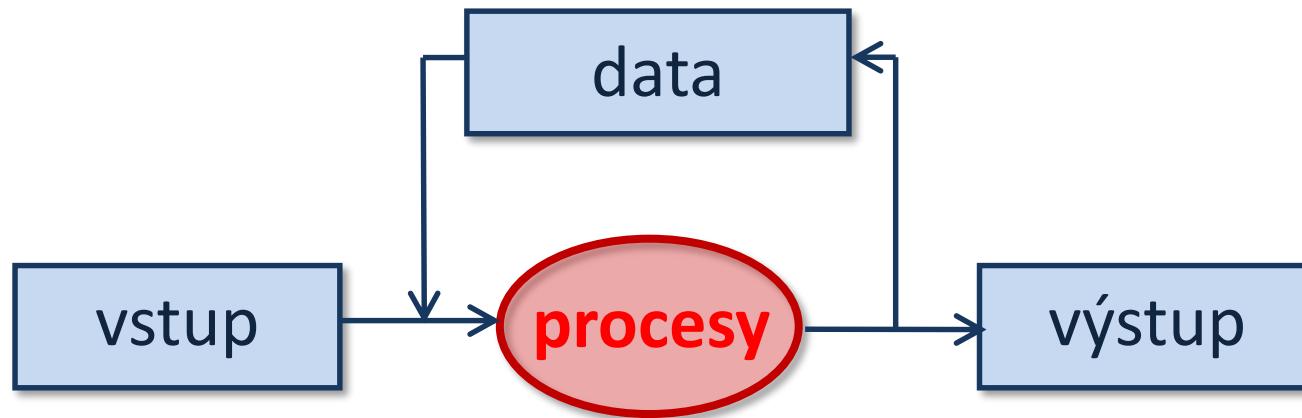
OLTP jako model

- říkáme, že OLTP modeluje nějaký fyzický podsystém.
- mezi OLTP a jeho fyzickým vzorem existuje izomorfismus φ
- pokud je v původním systému funkce nad zdroji, potom v OLTP existuje obraz této funkce pracující s obrazy zdrojů.
- pokud funkce v původním systému má za parametry jisté zdroje a dává jistý výsledek, pak obraz funkce v OLTP mající za parametry obrazy původních zdrojů dává za výsledek obraz původního výsledku.
- To platí i naopak.

Příklad OLTP systému

- ve fyzickém systému se pracuje s peněžními zdroji, tj. ***skutečnými penězi***, pak se v informačním systému pracuje s jejich ***virtuálním obrazem***.
- pokud ve fyzickém systému je provedena funkce, která na základě objednávky vytvoří skutečnou fakturu, pak v informačním systému je vytvořen její obraz.

Procesy ve schématu informačního systému

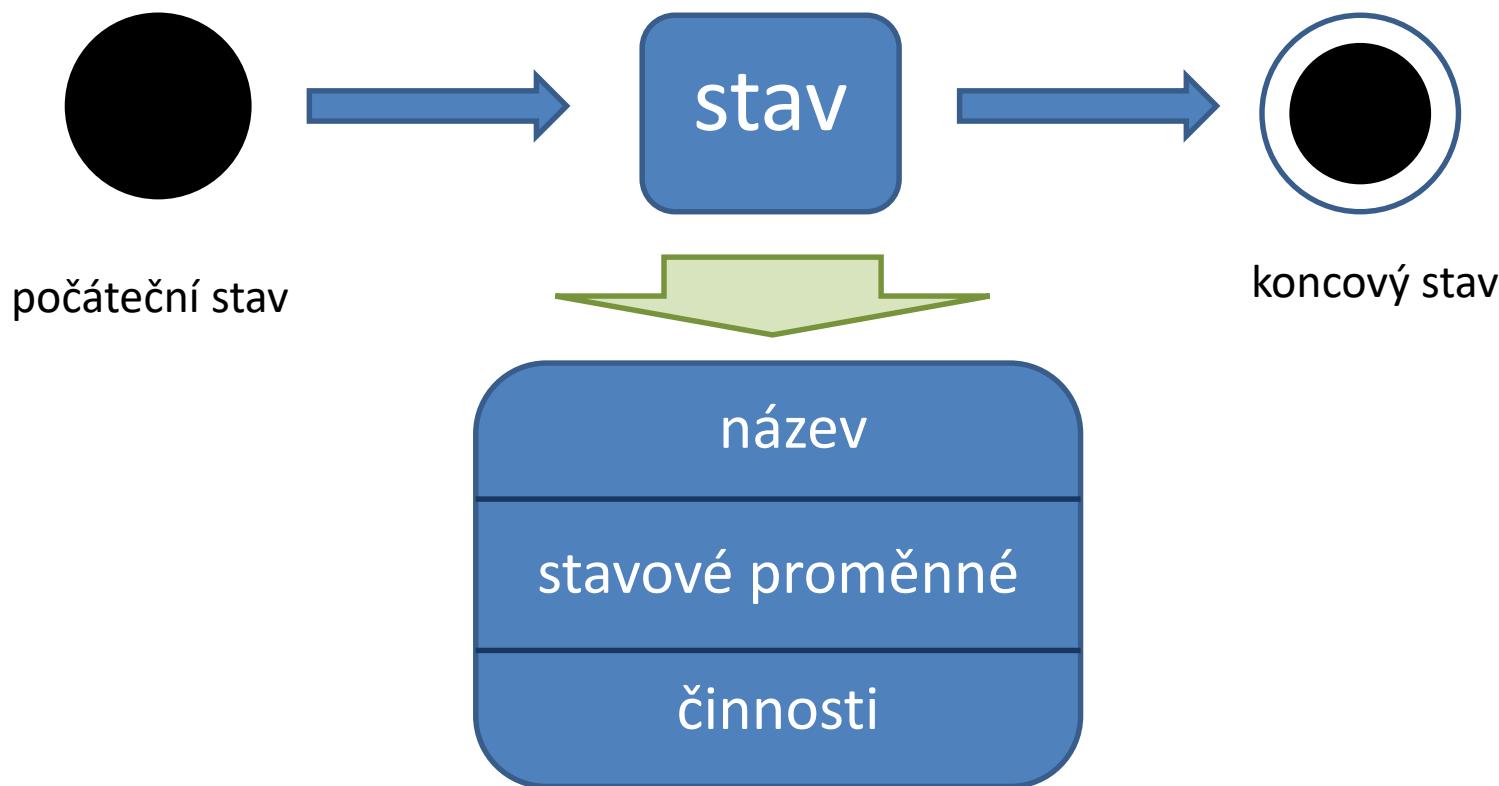


- ***data*** uchovávající ***stav*** systému a
- ***procesy*** realizující transformace často ve formě ***transakcí***.

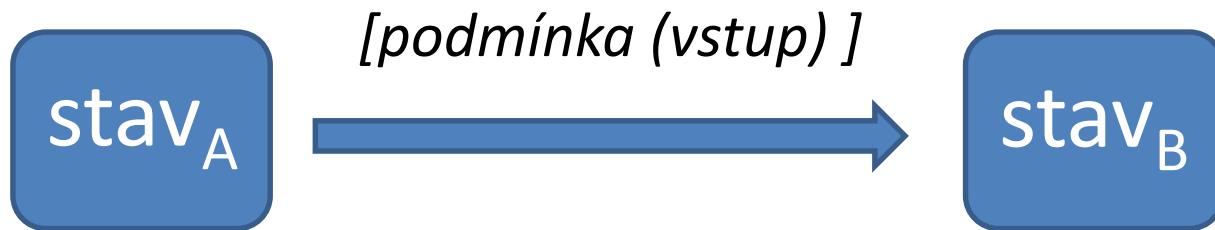
PROCESY A JEJICH DEFINICE

Stavový diagram UML

- Symboly UML používané ve stavových diagramech



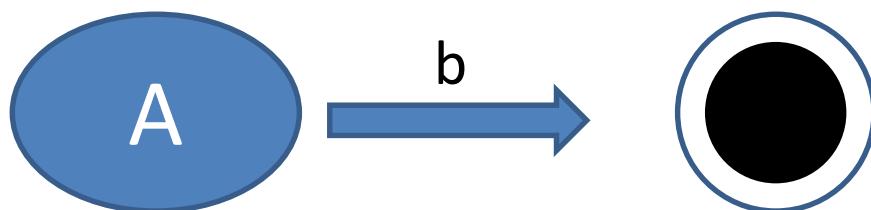
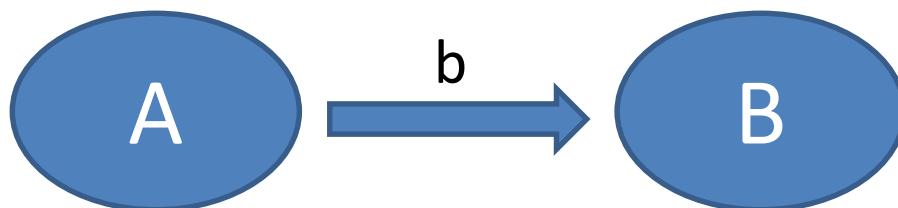
Strážní podmínky



Sekvenční procesy

- model = regulární gramatiky, **konečné automaty**

$A \rightarrow bB$ nebo $A \rightarrow b$



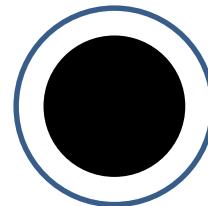
koncový stav

Sekvenční procesy

- stavový diagram (UML)

$\text{stav}_A \rightarrow [vstup_b] \text{ stav}_B$

$\text{stav}_A \rightarrow [vstup_b] \text{ (a konec)}$

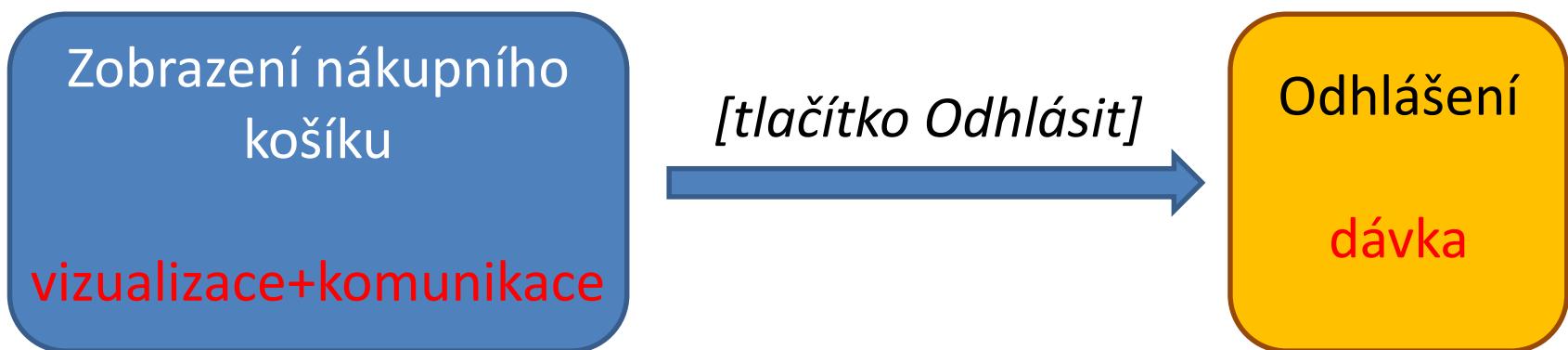


- vstup může být i prázdný

koncový stav

Příklad

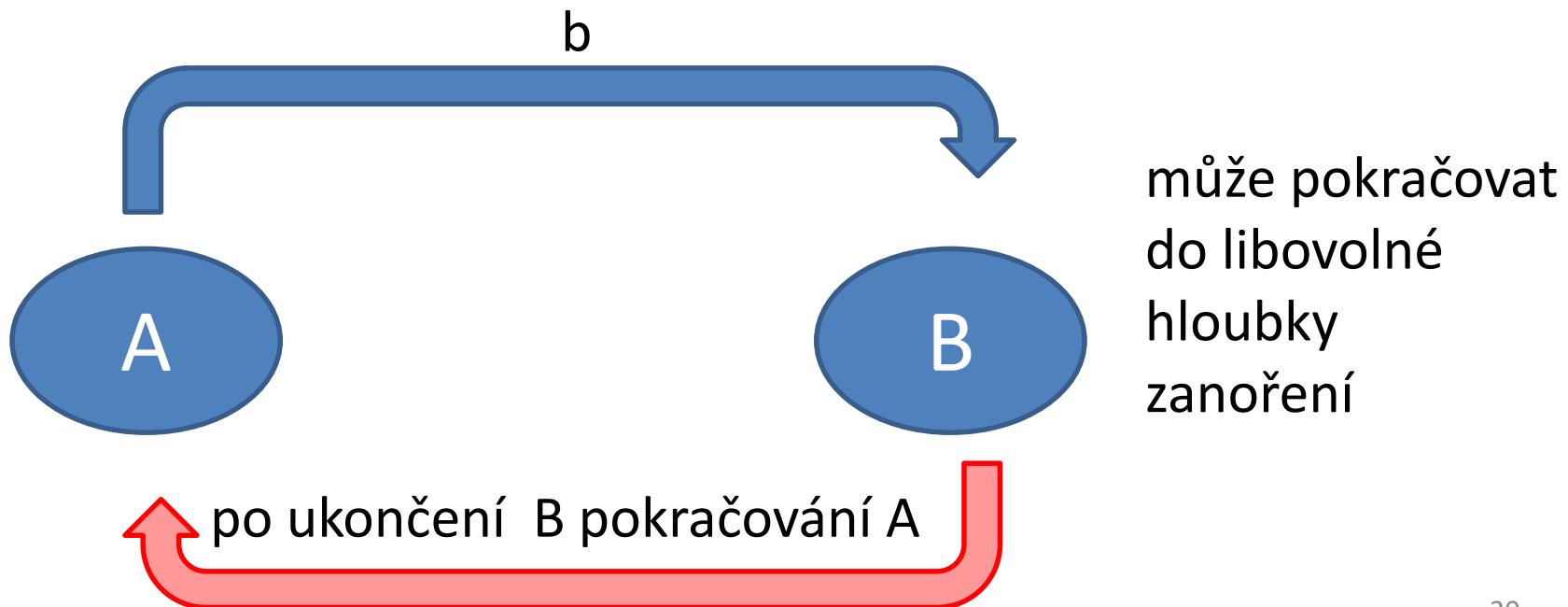
'Zobrazení nákupního košíku' -> *[tlačítko Odhlásit]*
'Odhlášení'



Hierarchické procesy

- model = bezkontextové gramatiky (speciální pravidlo), zásobníkový automat

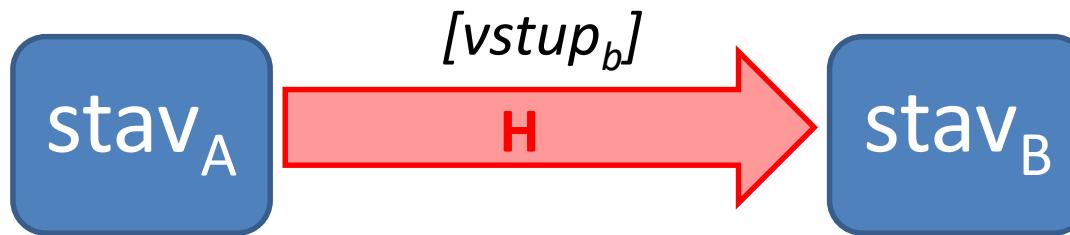
$A \rightarrow b B$ pokračování_A



Hierarchické procesy

- stavový diagram (varianta **UML**)

$\text{stav}_A \rightarrow [vstup_b] \text{ stav}_B$ pokračování_A



- Vstup musí být vždy označen, jinak může vzniknout nekonečný cyklus

Příklad

'Zobrazení nákupního košíku' -> [*tlačítko Přepočítat*]
'Úprava množství v košíku'

Zobrazení nákupního
košíku
vizualizace+komunikace

[*tlačítko Přepočítat*]



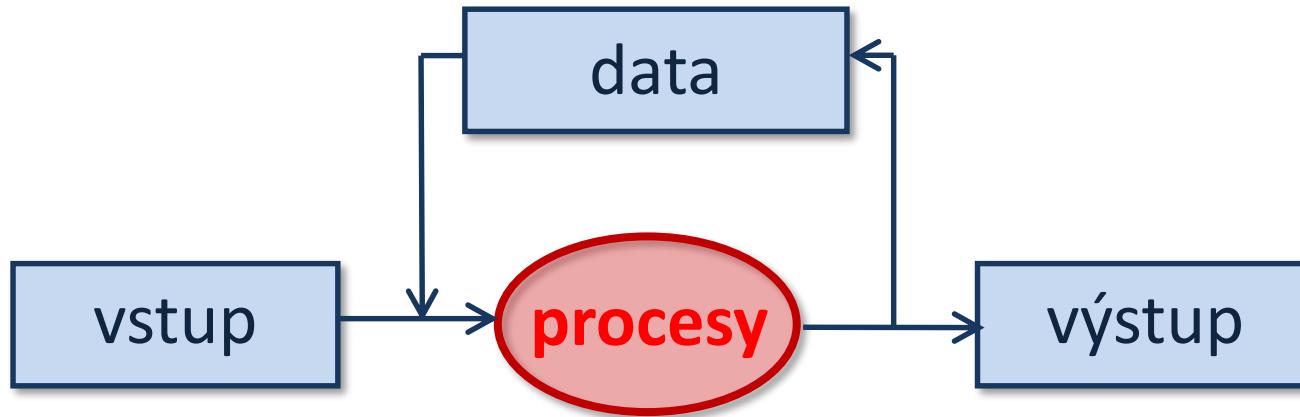
Úprava množství
v košíku
dávka
transakce T

- v dávkových stavech se často vyskytuje transakce

Obecné procesy

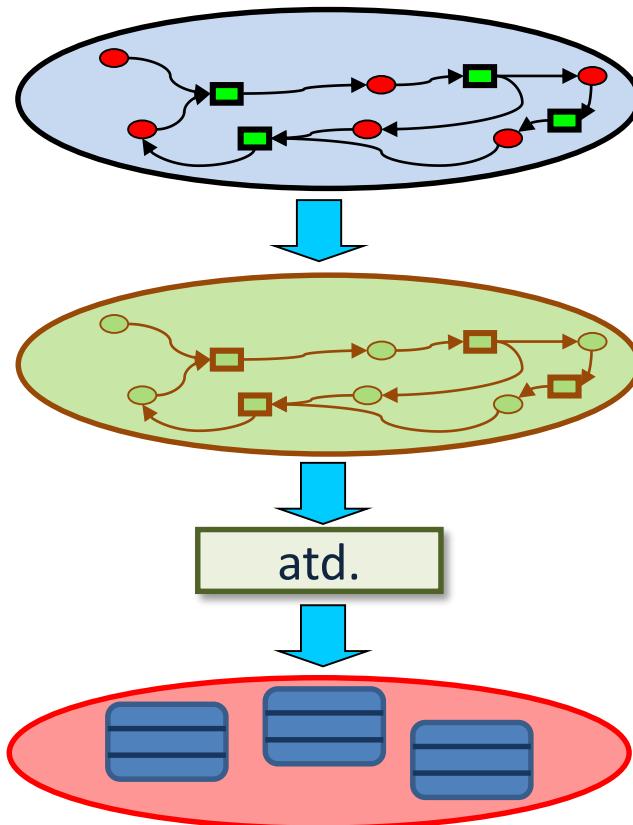
- model = neomezené gramatiky, Turingův stroj
 $a A b \rightarrow c B d$
- diagram neexistuje, popisuje se v nějakém programovacím jazyku

Procesy ve více úrovních



- typy *procesů* se liší využitím paměti (všechno jsou modelovány stavovými stroji s různým typem paměti - kontextu)
- pokud bychom považovali za paměť data IS v databázi, jde většinou o obecné procesy.

Řešení obecností dekompozicí



řízení - sekvence, hierarchie

dekompozice stavů

na dně hierarchie obvykle
obecný proces - *transakce*

(DATABÁZOVÉ) TRANSAKCE

Motivační otázky vzniku transakcí

- Co se stane, pokud dojde k poruše během práce s důležitými zdroji dat?
- Které operace prováděné nad daty systém stihl před poruchou skutečně provést a které ne?
- Co se stane, když více uživatelů současně bude modifikovat tentýž údaj?
- Budou údaje v databázi stále smysluplné ?
- Hledáním odpovědí a jejich aplikací při zajištění spolehlivosti se zabývají ***transakční modely*** a celý obor ***transakčního zpracování***.

Pojem transakce

- **Transakce** představuje jednotku práce vykonávanou v databázovém (informačním nebo podobném) systému nad databází a zpracovávanou **souvislým a bezpečným způsobem nezávisle na jiných transakcích**. Transakce v databázovém prostředí má dva základní účely:
 - Poskytnout bezpečnou jednotku práce, která dovoluje správné **zotavení z poruch** a udržuje databázi v konzistentním stavu i v případě poruchy systému, když je zastaveno provádění (úplně nebo částečně) a některé operace nad databází zůstávají nedokončené nebo v nejistém stavu
 - Poskytnout **izolaci programům přistupujícím k databází současně**. Pokud tato izolace není poskytnuta, výstupy programů jsou potenciálně chybové.

Pojem transakce

- ***skupina operací*** (akcí) prováděných jako celek (bud' celá dávka nebo nic)
- modelování stavu popisovaného výseku reálného světa
 - popis a provádění nerozlučných příkazů
 - první historické zmínky – 60. léta
 - důležitý pojem v oblasti databází
- ***Transakce*** je speciální druh programu, který je spouštěn v aplikaci ***OLTP*** (On Line Transaction Processing)

Systém pro zpracování transakcí - TPS

- systém (platforma, databázový systém)
podporující provádění transakcí – transakční systém
- zajišťuje speciální *vlastnosti transakcí* (atomičnost, nezávislost, trvanlivost)
- angl. *Transactional Processing System* (zkratka **TPS**)

Základní vlastnosti transakce

- Žádoucí vlastnosti transakcí jsou:
 - **Atomičnost** (Atomicity) – každá transakce je dokončena zcela nebo vůbec
 - **Konzistence** (Consistence) – databázová konzistence (správná reflexe stavu reálného světa a dodržování omezujících pravidel pro hodnoty)
 - **Izolovanost** (Isolation, Independence) – souběžné provádění má totožný efekt jako sekvenční
 - **Trvanlivost** (Durability) – odolnost proti ztrátě již dokončených změn
- V databázové praxi se pro tyto vlastnosti užívá akronym **ACID**.

důležitý pojem: ACID

Kdo co zajišťuje

- Programátor je zodpovědný za vytváření konzistentních transakcí. TPS považuje
 - *konzistenci* za zajištěnou programátorem (případně částečně systémem pro kontrolu integritních omezení)
- a zaopatřuje
 - *atomičnost,*
 - *izolovanost* a
 - *trvanlivost,*
- což jsou vlastnosti nutné pro zajištění ***souběžného spouštění konzistentních transakcí*** a případného ***zotavení z chyb či poruch.***

Úkol transakce

- Úkolem transakce je udržovat model (abstraktní, datový) stavu skutečného světa během jeho změn v **konzistentním stavu**.

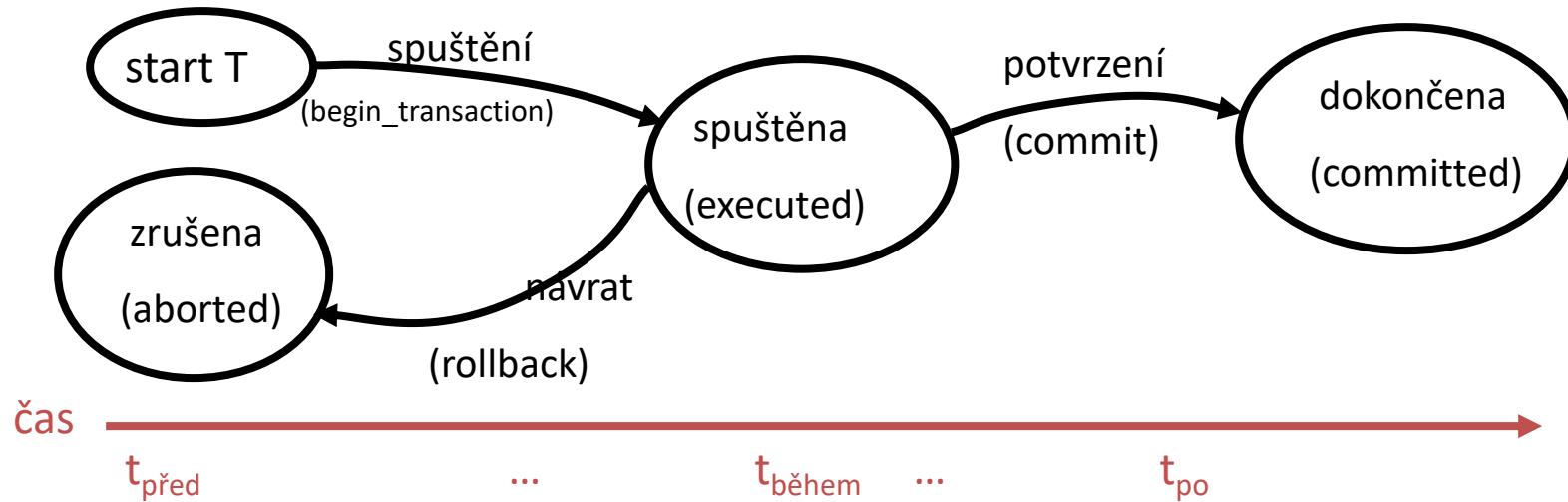
Atomičnost

BUĎ

- Provedení kompletní transakce

NEBO

- Neprovedení ani jedné operace z transakce
 - Implementačně řešeno návratem do původního stavu před spuštěním transakce



Důvody zrušení prováděné transakce

Nepředvidatelné:

- havárie systému

V režii TPS:

- porušení integritního omezení
- porušení izolovanosti souběžných transakcí
- detekované uváznutí (deadlock)

V režii transakce/programátora:

- na požadavek samotné transakce (rollback)

Bankomat

- Transakce výběru hotovosti z bankomatu se skládá z nejméně dvou základních akcí:
 - snížení stavu účtu o vybranou částku a
 - vydání příslušného obnosu hotovosti.
- Atomické spuštění této transakce znamená, že pokud je potvrzena, tak jsou provedeny obě akce a pokud je zrušena, tak ani jedna.

Konzistence

- zajišťuje uživatel
- Databáze má dvě role vzhledem k modelovanému nosiči:
 1. pasivní = kontrola a hlášení chyb
 2. aktivní = vynucování platnosti pravidel daných aplikacní doménou
- Konzistence souvisí s oběma rolemi a má dvě formy:
 1. Konzistence datového modelu
 2. Konzistence s reálným světem (izomorfismus modelu)

Konzistence datového modelu

- DB musí splňovat všechna integrit. omezení (IO)
 1. Udržování ***interní konzistence*** (redundantních dat)
 2. Dodržování ***pravidel*** daných reálným světem
- ***Transakce nesmí po svém dokončení porušit/porušovat žádné integritní omezení!***
- U nekonzistentní DB nedefinujeme chování transakcí.

PS: Díky atomičnosti transakcí nevadí dočasná nekonzistence během provádění.

Izolovanost

- zabývá se vícenásobným souběžným přístupem
- řeší ji TPS

Sekvenční zpracování, plán

- **sekvenční zpracování transakcí** = v jeden okamžik je rozpracována nejvýše 1 transakce (zákaz souběžnosti více transakcí)
 - + zachování konzistence
 - špatná efektivita/propustnost
- **souběžné zpracování (concurrent execution)**= využití paralelismu (několik CPU, několik I/O jednotek)
- **plán transakce** = pořadí operací uvnitř této transakce
- (vykonávací) **plán** = **sloučení** (NE pouze zřetězení) plánů souběžných transakcí (operace se mohou promíchat)

Izolovanost, nezávislost

- **Izolovanost** = výsledný efekt vykonání plánu souběžných transakcí je stejný jako jejich sekvenční zpracování.
- **Uspořadatelný plán** = splňuje izolovanost a transakce jsou konzistentní
- Analogie s problémem kritické sekce (v OS) při přístupu ke sdíleným prostředkům
 - zamykání záznamů v DB (trpí výkonnost)
- V praxi – různé **úrovně izolovanosti**
 - volba optima mezi správnou funkčností a rychlostí

Trvanlivost

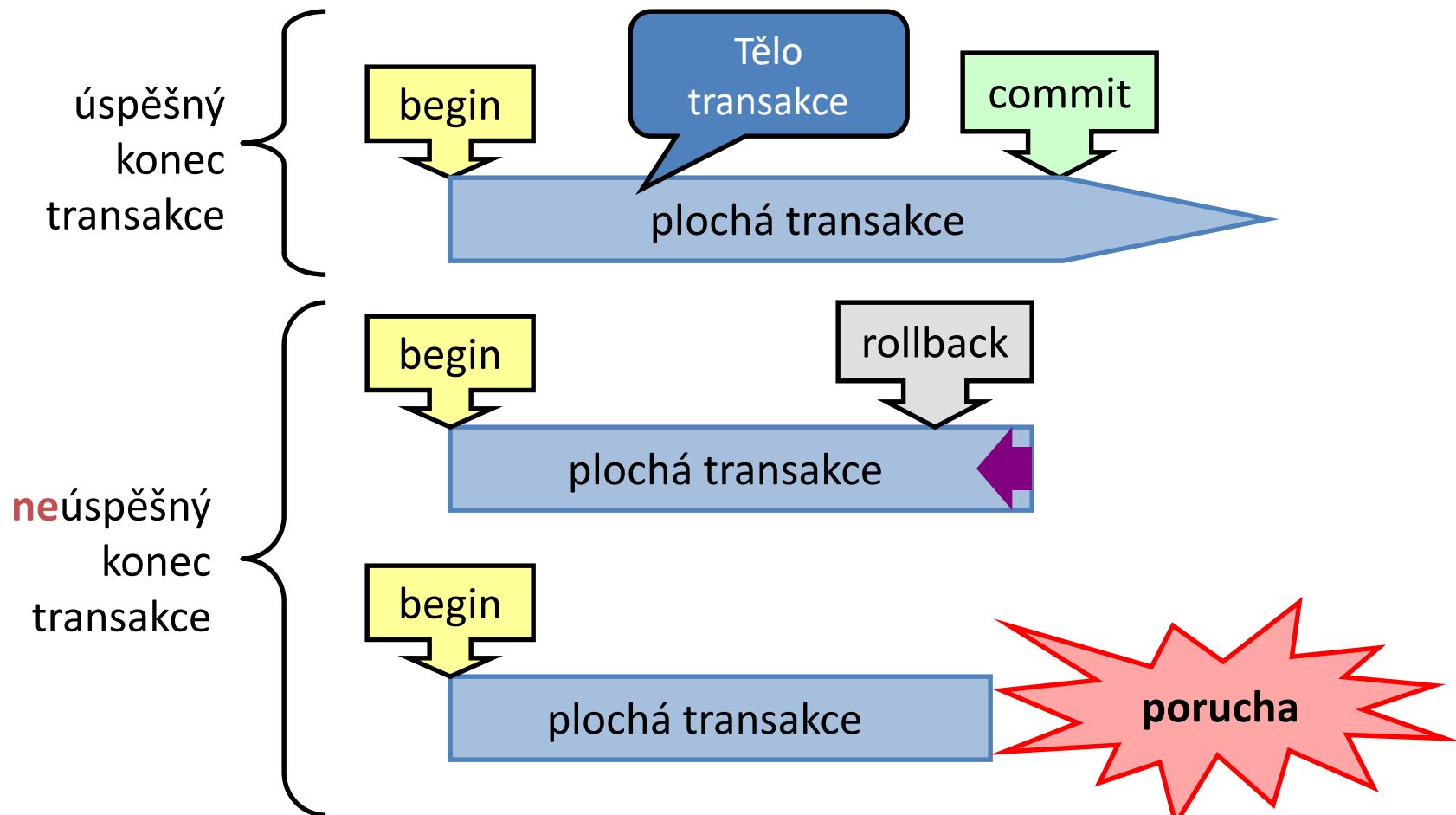
- zajišťuje TPS
- ***Trvanlivost*** (stálost) změn zanesených úspěšně dokončenými transakcemi
- ***Dostupnost*** = rychlosť uvedení systému do původního funkčního stavu po havárii
 - Nonstop dostupnost (např. zrcadlení disků)
 - Pomalejší dostupnost (např. obnova DB z pásky)
- Úrovně kvality trvanlivosti
 - např. odolnost vůči selhání CPU, selhání 1/více disků, živelné pohromě, úmyslnému útoku
- **kvalita × cena**

MODELY PROCESŮ (TRANSAKCÍ)

Plochá transakce

- Minimální model, který není vnitřně strukturovaný
- Obvykle obecný proces = popis v programovacím jazyce
- začátek transakce (begin)
- tělo transakce (sekvence operací)
- konec transakce
 - Úspěšný (commit)
 - Neúspěšný (rollback či abort nebo porucha)
- Žádné **částečné zotavení** ani *savepoints*

Plochá transakce (Flat transaction)



Struktura ploché transakce

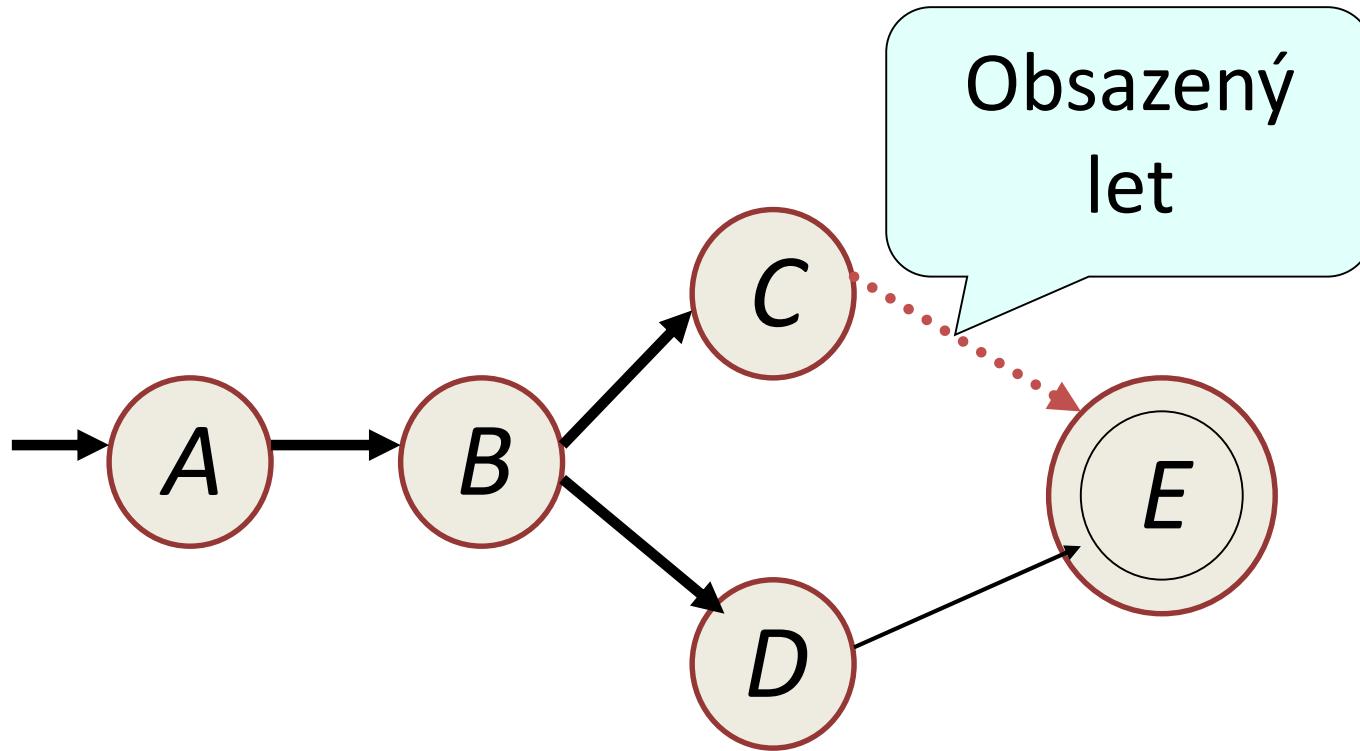
```
begin_transaction();
```

```
... blok programu ...
```

```
commit();
```

- kde blok může obsahovat příkaz `abort()`;
- v programu se mohou vyskytovat ***lokální nedatabázové proměnné***

Příklad ploché transakce hledání volného letu z A do E



Příklad ploché transakce

- transakce plánování cest, která provádí rezervace letů na cestě z bodu A do bodu E .
- prvním pokusem je uspořádat cestu přes body A, B, C, E .
- druhou možností je cesta přes body A, B, D, E .
- v prvním kroku rezervuje transakce let z A do B , následně z B do C .
- nyní však plánovač zjistí, že z bodu C do E již není žádný volný let, takže ploché transakci nezbude nic jiného, než ***provést zrušení všech prozatím provedených rezervací*** místo toho, aby provedla pouze částečný návrat o jednu rezervaci a provedla rezervaci z bodu B do bodu D , odkud je již volný let do cílového bodu E .

Dekompozice konečným automatem

SEKVENČNÍ MODELY

Body návratu (Savepoints)

- ***savepoint*** = bod návratu (synonymum = ***milník***) v transakci pro částečný návrat (partial rollback)
- i více bodů návratu v 1 transakci
 - odlišení identifikátorem/číslem
 - $sp_i := \text{create_savepoint}();$
- ***částečný návrat*** obnoví DB kontext
 - $\text{rollback}(sp_i);$
 - ***nemění se hodnoty lokálních proměnných***
 - ***poté se pokračuje*** za příkazem návratu ve vykonávání transakce
- není aktivních více transakcí, vše v rámci jediné transakce

Příklad

```
begin_transaction();
S1;
sp1 := create_savepoint();
S2;
sp2 := create_savepoint();
S3;
sp3 := create_savepoint();
...
if (condition)
{
    rollback(sp2);
    ...
}
...
commit();
```

- po provedení rollback(sp2); již nemá smysl pracovat s proměnnou sp3, protože identifikuje již neexistující milník. Součástí částečného návratu bylo i odstranění milníků mezi místem volání návratu (rollback(sp2);) a cílem návratu (sp2 := create_savepoint());.

Body návratu

- *Lokální proměnné* v transakci **nejsou** provedením částečného návratu (ani návratu) **změněny**.
- *Provedením částečného návratu tedy není iluze, jako by se daná část transakce vůbec nepovedla, ale pouze jako by nepovedla žádné změny v databázi.*
- Důležitým rozdílem mezi návratem (rollback) a zrušením transakce (abort) je, že **abort** transakce dále nepokračuje v provádění transakce, kdežto po **rollback** (v případě umístění milníku na úplný začátek transakce lze mluvit i o návratu) se provede zbytek transakce.

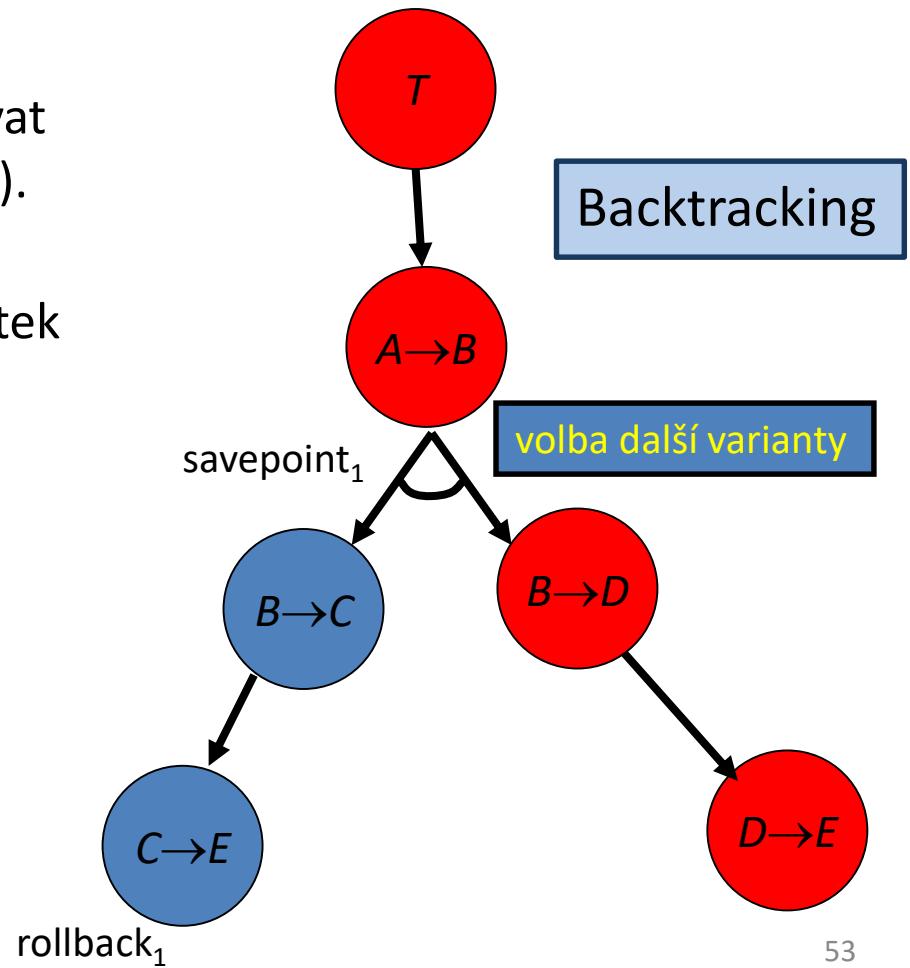
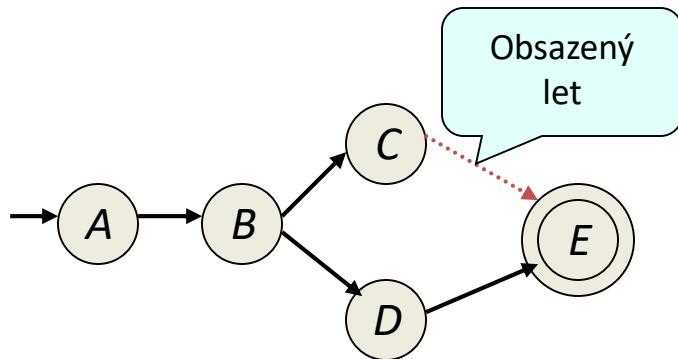
Příklad bodů návratu

Model transakce T pro rezervaci cesty z A do E :

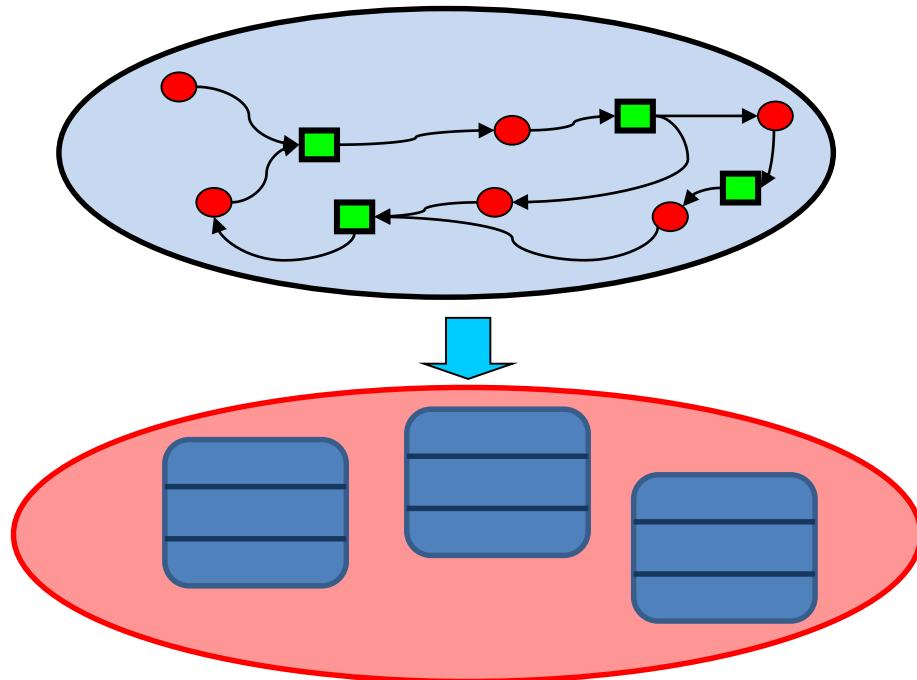
Je rozpracována jediná transakce se zásobníkovou strategií (musí existovať programově **zásobník** bodů návratu).

Na začátku a v každém větvení se vytvoří bod návratu. Návrat na začátek znamená neúspěch.

Možné cesty z A do E :



Dvouúrovňové schéma



řízení - sekvence,
hierarchie

dekompozice stavů

stavy - obecný proces
někdy *transakce*

- Standardně jsou procesy dekomponovány na
řízení a činnost stavů - transakce

Dvouúrovňové schéma

- Obecné procesy lokalizujeme do jednotlivých stavů

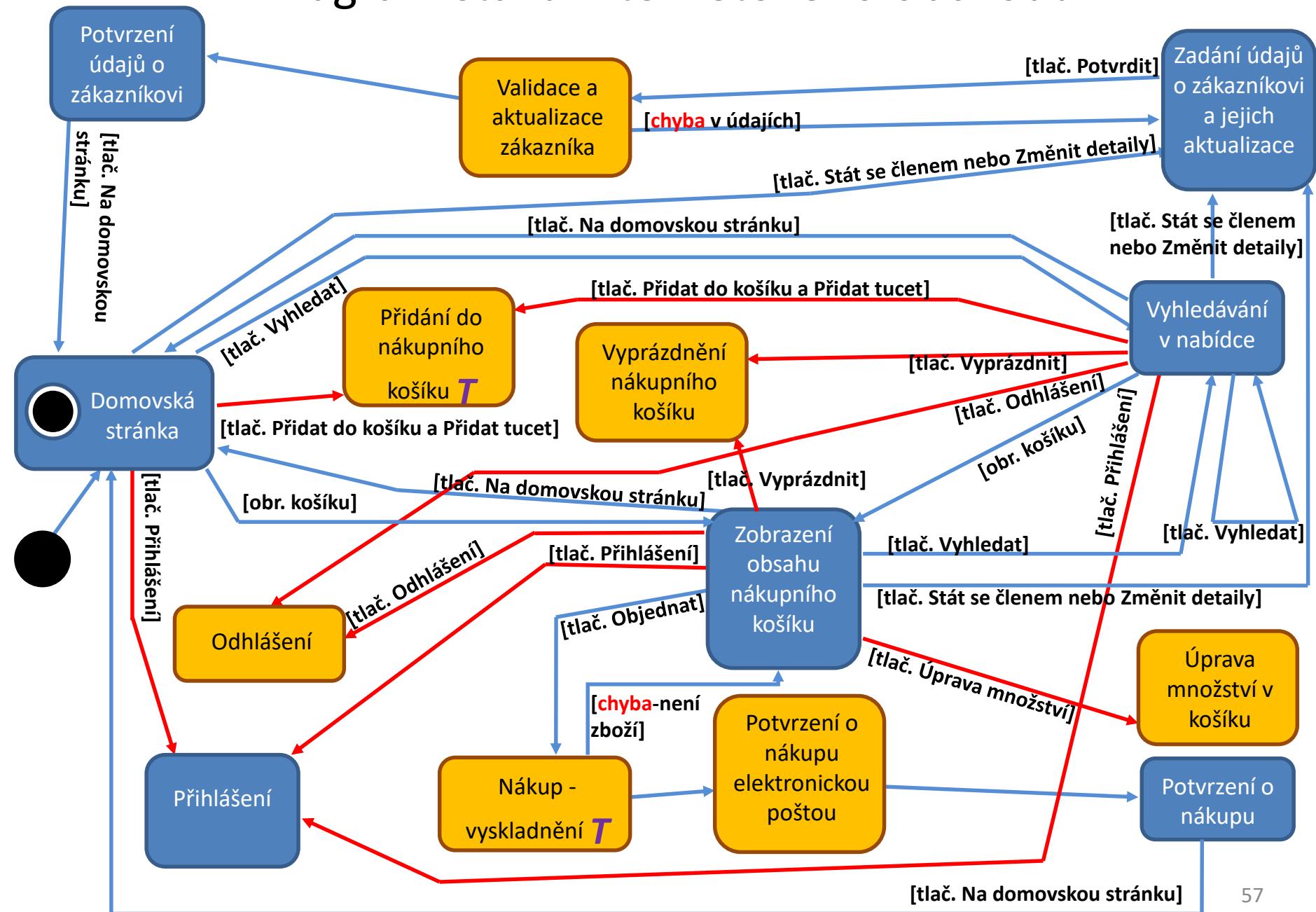


- Potom se proces přechodů mezi stavy značně zjednoduší

Dvouúrovňové schéma

- pro modelování informačních systémů na nejvyšší úrovni ***vystačíme se sekvenčními a hierarchickými procesy*** (širší kontext kromě zanoření se většinou nemodeluje)
- obecné procesy ve stavech (pracující s databází) pak modelujeme v obecném programovacím jazyce, někdy jako ***transakce*** podle různých modelů.

Diagram stavů internetového obchodu



Poznámky k diagramu stavů

- Větvení
 - převažující v komunikačních stavech (vizualizováno tlačítky) - událost způsobená uživatelem na klientovi
 - v dávkových stavech porušením konzistence (chyby) - kontrola konzistence prováděná na serveru
- Zabránit vzniku cyklů z hran bez označení
- ***Transakce T*** v dávkových stavech (ne ve všech)

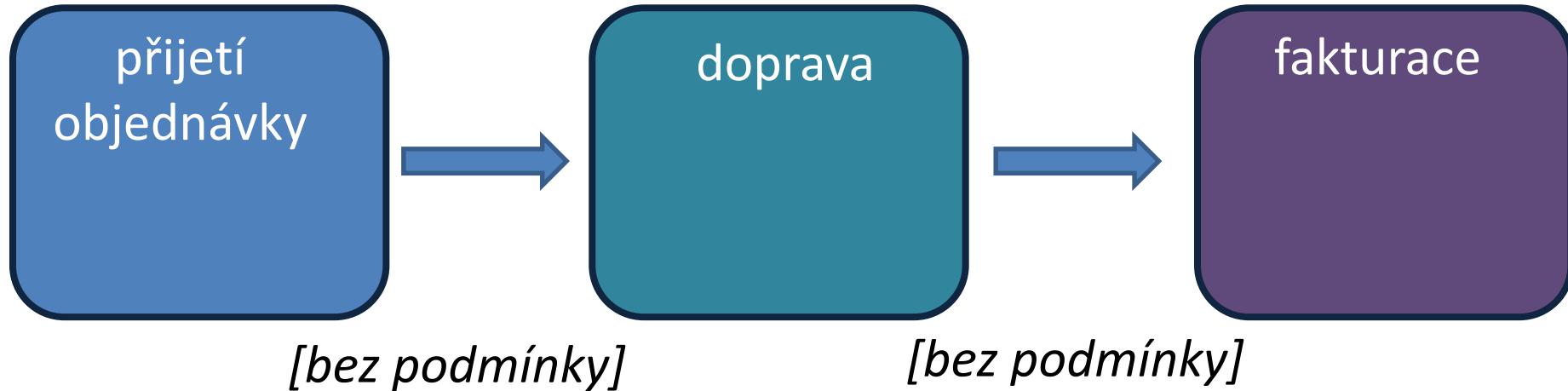
Chained transaction

ZŘETĚZENÉ TRANSAKCE – DVOUÚROVŇOVÁ DEKOMPOZICE

Motivace

- Většina aplikací se skládá ze sekvence transakcí.
- Například objednávkový katalogový systém, který se skládá ze tří transakcí:
 - přijetí objednávky,
 - doprava,
 - fakturace.

Slučování řízení a transakcí



- Převedení sekvence stavů s přechody bez podmínek na jedinou modelovanou strukturu
- Dochází ke slučování vrstev dvouúrovňového modelu

Zřetězené transakce - úvod

- v některých situacích je nutno ***dekomponovat*** na menší i transakce, které to nevyžadují ani kvůli distribuovanosti nebo lepší strukturovanosti, ale ***z důvodu vylepšení výkonnosti.***
- především ***dlouhoto trvající transakce*** (at' už z důvodu velkého množství prováděných operací nebo nutnosti delších čekání mezi jednotlivými operacemi).
- dalším důvodem dekompozice je v případě poruchy ***zabránění ztrátě celé zatím provedené práce***, kterou transakce vykonala

Zřetězené transakce

- Jednoduchou optimalizací je tzv. **zřetězení (angl. *chaining*)**, kdy po potvrzení jedné transakce je automaticky vytvořena nová transakce, která v dané sekvenci transakcí následuje.
- Tímto např. omezíme režii nutnou na zasílání příkazu `begin_transaction()` databázovému systému

Schéma zřetězené transakce

```
begin_transaction();
```

```
S1;
```

```
commit();
```

```
S2;
```

```
commit();
```

```
...
```

```
Sn-1;
```

```
commit();
```

```
Sn;
```

```
commit();
```

- kde S_i je tělo i -té transakce, kterou budeme dále označovat jako podtransakci ST_i .

Chování zřetězené transakce

- každý příkaz commit() způsobí ***trvanlivost změn předchozí podtransakce***, takže v případě havárie v podtransakci ST_i budou změny z podtransakcí ST₁, ..., ST_{i-1} zachovány.

Odlišné vlastnosti zřetězených transakcí

- Při využití pro dekompozici dlouhých transakcí (například připisování úroků na desítky tisíc bankovních účtů) je potřeba uvažovat následující soubor vlastností, kterými se zřetězené transakce *odlišují* od prozatím celkem striktních požadavků na **ACID** vlastnosti transakcí

Trvanlivost

- při poruše nejsou ztraceny výsledky předchozích podtransakcí v řetězu (sekvenci) na rozdíl třeba od klasické ***sekvenční dekompozice transakcí s milníky*** (savepoints), kdy je celá transakce zrušena (proveden kompletní návrat).
- Tím pádem zřetězená transakce jako celek (celý řetěz podtransakcí) ***není atomická***. TPS nemá žádnou zodpovědnost za restartování řetězu při zotavování z poruchy.

Komunikace mezi transakcemi v řetězu

- Problém využívání ***lokálních proměnných*** pro komunikaci mezi podtransakcemi (například předávání identifikátoru posledního zpracovaného účtu při procházení seznamu účtů) ***nastane v případě poruchy***, kdy všechny obsahy lokálních proměnných zaniknou a nemohou být tak využity při dokončení zřetězené transakce.

Komunikace mezi transakcemi v řetězu

- Alternativou je komunikace přes ***databázové proměnné***. Před potvrzením podtransakce jsou lokální proměnné uloženy do databázových položek a stanou se tak součástí databázového kontextu, který dokáže přežít pád systému. Při dokončování havarované transakce pak lze například zjistit, kterým záznamem dále pokračovat.
- Nevýhodou komunikace přes databázové proměnné je jejich ***viditelnost ostatním souběžným transakcím*** (např. v jiných aplikacích). Toto musí být ošetřeno (např. využitím ***zámků*** nad záznamy databázových proměnných).

Databázový kontext

- není udržován mezi dvěma sousedními podtransakcemi v řetězu. Například pokud je potvrzena podtransakce ST_i , tak se uvolní všechny jí aktivované zámky nebo používané databázové kurzory.
- souběžné transakce tak mají v časovém úseku mezi dokončením ST_i a startem ST_{i+1} volný přístup i k položkám, které byly v ST_i zamknuty.
- může se tak stát, že ST_{i+1} již bude pracovat s jinými hodnotami v těchto položkách, protože budou změněny nějakou souběžnou transakcí.
- v kontrastu s dlouho trvajícími transakcemi je sice ***každá podtransakce řetězu izolovaná, ale celá zřetězená transakce izolovaná není.***

Databázový kontext

- Porušení izolovanosti zřetězené transakce jako celku při správném návrhu a použití tohoto typu transakcí vede ke zvýšení výkonnosti, protože je zkrácena doba zamykání záznamů v databázi a je tak větší prostor pro souběžné provádění transakcí

Konzistence

- podtransakce při potvrzení uvolňují databázový kontext a zviditelnějí tak provedené změny ostatním souběžným transakcím.
- v případě, že tyto souběžné transakce vyžadují spouštění v konzistentní databázi, tak musíme požadovat, aby každá podtransakce v řetězu byla konzistentní. Všimněme si, že v případě použití ***milníků*** tento problém neřešíme, protože je celá transakce ***atomická*** a ***izolovaná***.
- Konzistence podtransakcí je vyžadována také z důvodu, že při havárii systému ***nezajišťuje TPS***, aby byla přerušená zřetězená transakce ***dokončena*** (je proveden návrat pouze aktuálně spuštěné podtransakce).
- ***Zřetězené transakce nejsou izolované ani atomické.***

Alternativní sémantika pro zřetězené transakce

- pokouší se eliminovat některé předchozí nevýhody a porušení ACID

Schéma alternativní sémantiky

```
begin_transaction();
```

```
    S1;
```

```
    chain();
```

```
    S2;
```

```
    chain();
```

```
    . . .
```

```
    Sn-1;
```

```
    chain();
```

```
    Sn;
```

```
commit();
```

- kde chain() odlišuje starou a novou sémantiku a jedná se o příkaz ***potvrzení podtransakce ve zřetězené transakci*** s lehce pozměněnou sémantikou oproti příkazu commit().

chain()

- Příkaz chain() potvrzuje podtransakci ST_i , začíná novou ST_{i+1} , ale
- na rozdíl od commit() **mezitím neuvolňuje databázový kontext** (např. aktivované zámky nad položkami databáze) ani databázové kurzory.
- Pokud byla tedy nějaká položka zamknuta v ST_i , tak bude zamknuta i v ST_{i+1} , pokud nebyl zámek explicitně uvolněn.
- Souběžné transakce navíc nevidí stav databáze ani v časovém bodě mezi ST_i a ST_{i+1} . Pokud to ST_{i+1} očekává, tak dokonce může ST_i zanechat databázi nekonzistentní.
Zřetězena transakce s alternativní sémantikou je jako celek izolovaná, i když na úkor výkonnosti.

Dopředný návrat (roll forward)

- daní za toto řešení je ***komplikovanější zotavovací fáze***.
- nyní již striktně nevyžadujeme konzistenci po dokončení podtransakce uvnitř řetězu, tak může při návratu ke stavu po poslední dokončené podtransakci ***zůstat databáze v nekonzistentním stavu***.
- ***Úplný návrat nelze provést***, protože změny již dokončených podtransakcí byly potvrzeny a porušili bychom tak trvanlivost.
- Řešením je rozšíření zotavovací procedury, která po restartu TPS zajistí ***dokončení zřetězené transakce***. Tj. obnoví se databázový kontext (včetně zámků) a znova se provede kvůli havárii nedokončená podtransakce a všechny po ní následující až do konce řetězu. Pak zůstane ***zachována izolovanost i atomičnost celé zřetězené transakce s alternativní sémantikou***. Tento způsob zotavení se nazývá ***dopředný návrat*** (angl. ***roll forward***).

Kompenzující transakce

- mějme zřetězenou transakci, kterou jsme vytvořili místo jedné dlouho trvající transakce, a předpokládejme, že po potvrzení (dokončení) několika podtransakcí potřebuje transakce provést úplný návrat (abort).
- bohužel protože zřetězené transakce ***nezaručují izolovanost ani atomičnost***, tak nelze garantovat, že nebyla databáze před spuštěním návratu změněna nějakou souběžnou transakcí, což je zásadní problém.

Možnost implementace návratu

- Možnosti implementace **částečného návratu a úplného návratu transakcí**:
- v plochých a zanořených transakcích, které zaručují ACID vlastnosti, lze použít tzv. **fyzickou obnovu** neboli **fyzické logování** (angl. *physical restoration, logging*), kdy si uložíme stav dané databázové položky před její změnou, poznačíme si, že byla změněna, a při návratu provedeme obnovení její původní hodnoty.

Nevýhody fyzického logování

- Odčinění změn způsobených zřetězenou transakcí (neuvážuji alternativní sémantiku) je však složitější, protože pouhé fyzické logování je nedostatečné, protože mezi jednotlivými podtransakcemi je databázový kontext viditelný i ostatním souběžným transakcím a ty mohou provést nějakou změnu předtím, než spustíme návrat.
- Provedením fyzického obnovení bychom tak mohli potenciálně porušit trvanlivost změn od jiných transakcí. Tento problém řeší tzv. ***kompenzace*** (angl. *compensation*).

Kompenzující transakce

- ***Kompenzující transakce*** je speciální druh transakce, který zajišťuje provedení ***logické obnovy důsledků spuštění jiné transakce***.
- Obsahuje posloupnost akcí nutných pro uvedení databáze do stavu, v jakém by byla při neprovedení transakce, kterou kompenzační transakce kompenzuje. Kompenzační transakce ***nemusí revertovat nutně všechny změny***, což je dáno především aplikační doménou a důležitostí těchto změn pro konzistenci databáze i konzistenci s reálným světem.

Příklad kompenzující transakce

- *Registraci studentů do předmětu.*
- Odregistrační (kompenzační) transakce logicky **ruší efekt úspěšné registrační transakce.**
- *Registrace zvýší počet studentů ve třídě a naopak deregistrace jej o jednoho sníží.*
- Kompenzace tak správně provede návrat i v případě, že byla souběžně provedena registrace dalšího studenta.
- Příkladem operace z registrační transakce, která nemusí být nutně kompenzována může být registrace i neúspěšně zaregistrovaného studenta do bulletinu, jehož účel je informovat o nových předmětech na daném ústavu čistě z reklamních důvodů

ZOTAVITELNÉ FRONTY

Zotavitelné fronty - motivace

- aplikace nevyžaduje úzké semknutí sekvence akcí do jediné transakce (izolované jednotky)
- postačí, když je zaručeno provedení jisté sekvence akcí (často transakcí)
- požadavkem je, aby po dokončení jedné akce byla ***někdy provedena další.***
- na rozdíl však od ***zřetězených transakcí*** může být mezi jednotlivými akcemi podstatná ***časová proluka.***

Příklad aplikace pro zotavitelnou frontu

- ***Zadání a specifikace problému katalogového a objednávkového systému.***
- Hlavní aktivita v tomto systému se skládá z vytvoření
 - objednávky,
 - expedice a
 - fakturace.
- Tyto tři úkoly lze vykonat třemi oddělenými transakcemi. Jediné co požadujeme, aby transakce ***fakturace a expedice byla provedena kdykoli po úspěšném dokončení objednávky***, a to i v případě havárie systému bezprostředně po objednání.

Zotavitelná fronta - definice

- **Zotavitelná fronta** (angl. *recoverable queue*) je mechanismus na plánování transakcí pro budoucí vykonání a zajištění, že vykonání bylo skutečně v aplikaci provedeno. Základní sémantika vychází z klasické fronty, která obsahuje operace:
- **Vlož.** Transakce vkládá do fronty záznam o práci naplánované k provedení, právě když je transakce potvrzena.
- **Vyber.** Záznam je pak někdy později vyzvednut jinou transakcí, která danou práci provede. Tato transakce bývá většinou spuštěna serverem, který periodicky kontroluje frontu a vybírá z ní pracovní požadavky.
- **Vkládaný/vybíraný záznam** obsahuje informaci o akci, která je plánována, a o datech, která je potřeba mezi jednotlivými transakcemi v řetězu předávat (například identifikátor objednávky).

Vlastnosti zotavitelné fronty

- Zotavitelná fronta ***musí být trvanlivá***, aby byla schopna přežít havárii systému. Navíc atomicita transakce vyžaduje od vložení/výběru z fronty následující koordinaci s potvrzením (resp. způsobem ukončení) transakce:
 - Vloží-li transakce do fronty záznam a později je zrušena, tak musí být tento záznam z fronty odstraněn.
 - Vybere-li transakce z fronty záznam a později je zrušena, tak musí být tento záznam do fronty navrácen.
 - Dokud není transakce T dokončena (potvrzena), tak nelze jinými transakcemi vybírat záznamy, jež byly transakcí T vloženy, protože může být transakce T ještě případně zrušena.

Trvanlivost zotaviteľné fronty

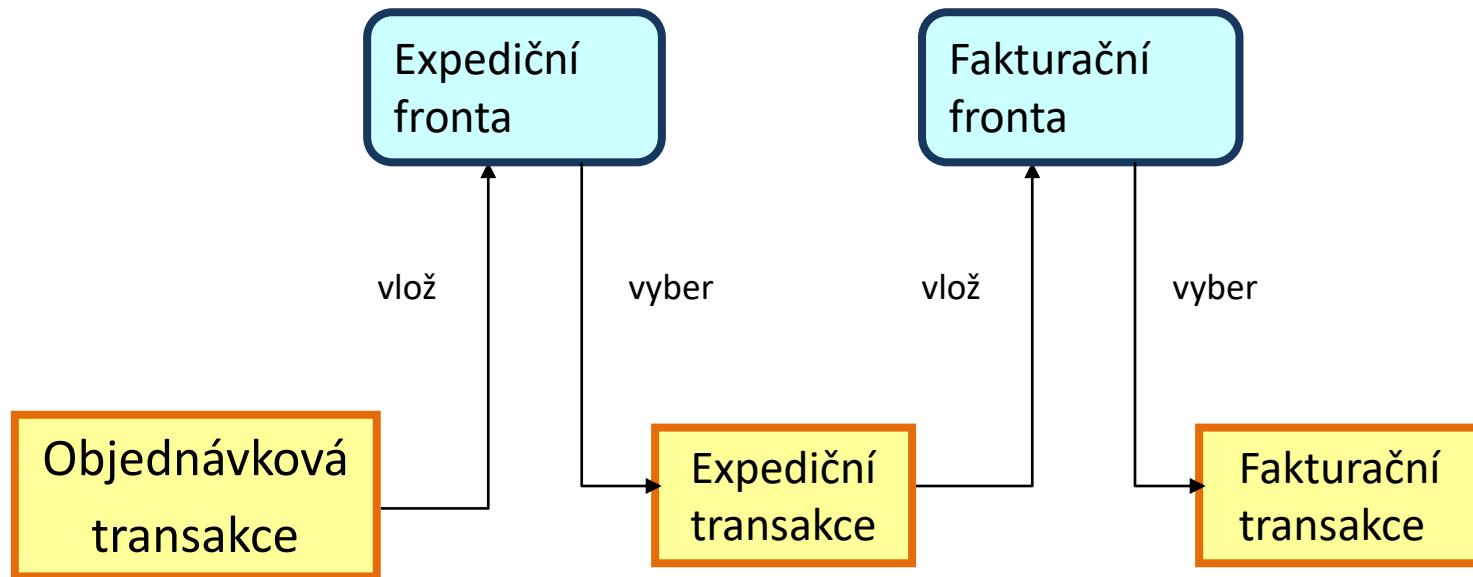
- Trvanlivost zotaviteľné fronty lze sice implementovať prostredníctvím tabuľky v databázi, ale častý prístup k této tabuľke pak tvoří **výkonnostnú slabinu** celého systému (angl. **bottleneck**). Proto je vhodné využiť k realizaci **oddelený aplikačný modul**.
- Jde tedy o ďalší funkčný blok systému

Scénáře využití zotavitelných front

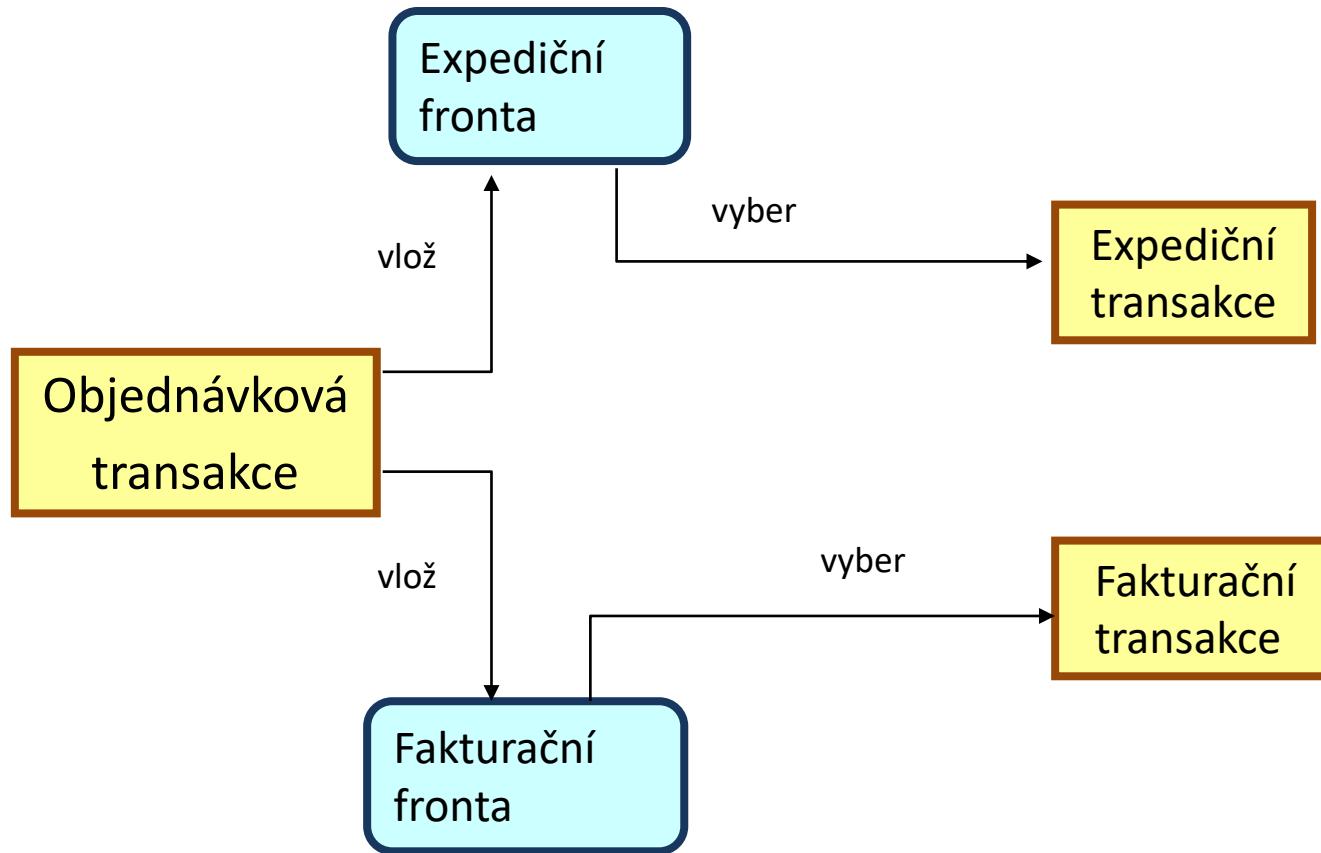
- Sekvence a paralelismus směřují k pozdějším schématům ve workflow
- Doposud se paralelismus v modelech **neřešil**.
Předpokládalo se **řešení pomocí TPS**.



Organizace zotaviteľné fronty – řetěz s pevným uspořádáním (pipeline)



Organizace zotaviteľné fronty – podpora paralelismu



Paralelismus

- využití ***paralelismu***, protože povoluje, aby byla transakce fakturační i expediční spuštěna v libovolný okamžik po dokončení objednávky (nevýhoda – zákazník může dostat dříve fakturu než samotné zboží, ale obchodně je to v pořádku).

REÁLNÉ UDÁLOSTI

Reálné události

- transakce s reálnou událostí může být zrušena.
- problém je v dosažení atomičnosti, protože reálné události nelze většinou vrátit zpět (odčinit).
- Například pokud bankomat vydá zákazníkovi hotovost a před potvrzením transakce systém havaruje, tak TPS nedokáže zajistit, aby byly navráceny všechny provedené akce. Může sice vrátit databázi do původního stavu, ale od zákazníka ***Ize jen těžko požadovat navrácení hotovosti*** (alespoň ne se 100% jistotou ☺).

Reálné události

- **Reálné (fyzické) události** (angl. *real-world events*) nelze vrátit zpět, a proto musí být atomičnost transakcí s takovými událostmi dosažena jiným způsobem než návratem (*rollback*).
- Nejpřirozenější je použít **dopředný návrat** (*forward roll*) s využitím zotavitelných front pro dosažení sémantiky – fyzická událost je provedena, když a jen když je transakce potvrzena.

Reálné události

- Transakce T vloží požadavek R na provedení fyzické události na zotavitelnou frontu Q před svým potvrzením. Pokud je však T zrušena, tak je R odstraněno z Q . Je-li T úspěšně potvrzena, tak je díky trvanlivosti Q někdy v budoucnu zajištěno obsloužení R .

Havárie reálné události

- Problém nastává v případě havárie transakce pro fyzickou událost T_{RW} , kdy nevíme, zda byla fyzická akce již provedena nebo ještě ne.

Možné řešení

- Předpokládejme, že fyzické zařízení provádějící reálnou událost obsahuje **čítač** C , který se inkrementuje atomicky s provedením akce. Dále předpokládejme, že tento čítač může číst i transakce T_{RW} . Pak po provedení akce provede T_{RW} načtení a uložení aktuální hodnoty čítače do databáze (**záznam** označme D) před vlastním potvrzením T_{RW} . Když se pak musí systémem zotavit z náhlé havárie, tak načte čítač C a porovná s hodnotou v D . Porovnání může mít dva výsledky:

Možné řešení

- $C = D$. Tedy žádná další fyzická akce nebyla provedena od potvrzení poslední T_{RW} .
- $C \neq D$. C musí být o jedničku větší, což znamená, že fyzická akce byla provedena, ale T_{RW} byla zrušena (a navrácena, včetně případného provedení aktualizace záznamu D i obnovení záznamu pro T_{RW} v zotavitelné frontě).

Zotavení

- Systém tedy z této situace ($C \neq D$) usoudí, že fyzická akce již byla provedena a cíl transakce T_{RW} byl splněn, i když nebyla potvrzena, a proto se před znovu-nastartováním celého systému provede odstranění záznamu pro T_{RW} ze zotavitelné fronty.



Technologie informačních systémů

Workflow systémy

Radek Burget
burgetr@fit.vutbr.cz

Vývoj architektur IS

- 60. léta – řada samostatných aplikací
 - vlastní uživatelské a datové rozhraní
 - vlastní metody ukládání dat
 - vlastní komunikace s uživatelem
- 70. léta – osamostatnění dat
 - databázové systémy
- 80. léta – osamostatnění uživatelského rozhraní
 - Windows API, X Window, ...
- 90. léta – osamostatnění řídicích procesů
 - workflow systémy

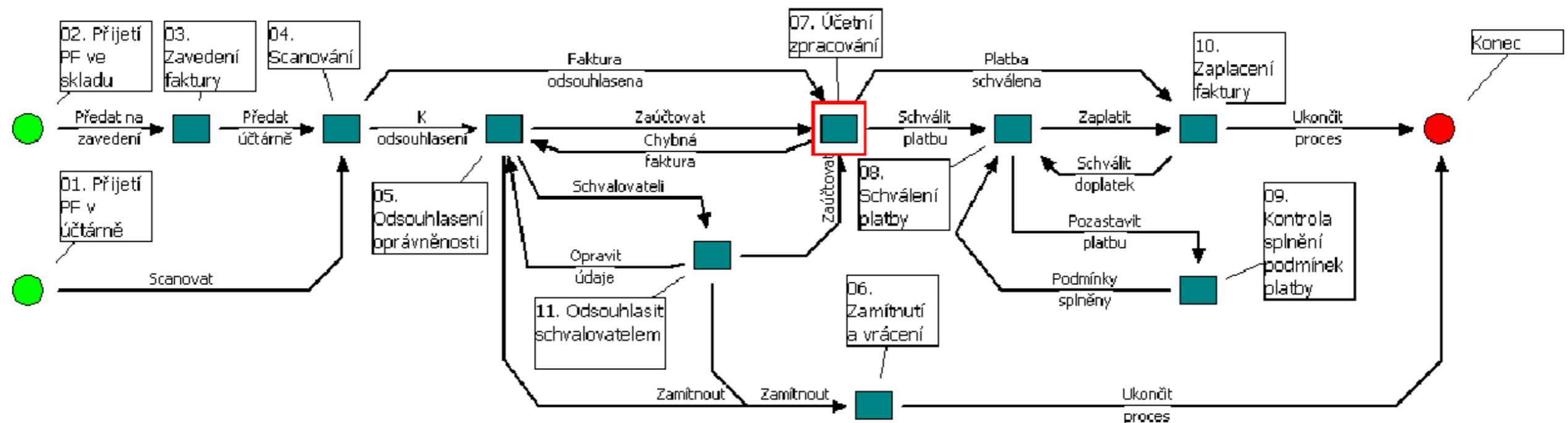
Podnikové (business) procesy

- Podnikový proces je koordinační mechanismus napříč organizačními jednotkami distribuovaný v čase a prostoru;
- integruje a koordinuje distribuované zdroje a poskytuje správnou informaci správnému jednotlivci ve správný čas k vykonání přiděleného úkolu.

CO – JAK – KDY – KDO

- Z hlediska technologie jde o
 - Popis procesů mimo vlastní implementaci IS
 - Infrastrukturu schopnou zajistit vykonání popsaných procesů
 - Doplňkové funkce: monitorování, analýza, ...

Příchozí faktura



Další příklady procesů

- Recenze příspěvků na konferenci
 - zaslání příspěvku, předání recenzentům, recenze příspěvků, zpráva autorovi, ...
- Zařízení služební cesty
 - objednávka letenek, ubytování, vypůjčení auta, zaplacení poplatků, schválení cesty, ...
- Vyřízení reklamace
 - obdržení požadavku, rozhodnutí o oprávněnosti, odpověď, ...
- Sledování pacientů v nemocnici
 - příjem, RTG, EKG, krev, diagnóza, léčení, ...

Příklady procesů (II)

- Vyřízení žádosti o půjčku
 - žádost o půjčku, analýza rizik, schválení, sledování splátek, uzavření případu, ...
- Vývoj programů
 - návrh, specifikace, implementace, ...
- Zápis studentů do dalšího ročníku
 - předběžný zápis, kontrola studia, zápis, změny, ...
- Výběrové řízení na zakázky
 - zadání, vyhodnocení nabídek, výběr dodavatele, řešení námitek, realizace, ...

Workflow

- Procedurální automatizace business procesu prostřednictvím správy sekvence pracovních aktivit a vyvolání příslušných lidských nebo IT zdrojů příslušejících k těmto aktivitám.
- Rozdíl mezi Business procesem a workflow je neurčitý, pojmy jsou často zaměňovány.
 - Workflow je konkrétní popis realizace procesu
 - BP lze chápat na obecnější úrovni



Systémy pro řízení business procesů

Zařazení workflow do infrastruktury

- Samostatný systém
 - Např. webové nebo jiné rozhraní
 - Sdílení podkladů a dokumentů
 - Např. naskenované faktury k vyřízení
 - Sledování stavu úkolů
 - Vývoj zpracování, zadávání doplňujících informací
- Integrace s existující infrastrukturou
 - E-mail (upozorňování na úkoly)
 - Účetní software, group managemet, ...

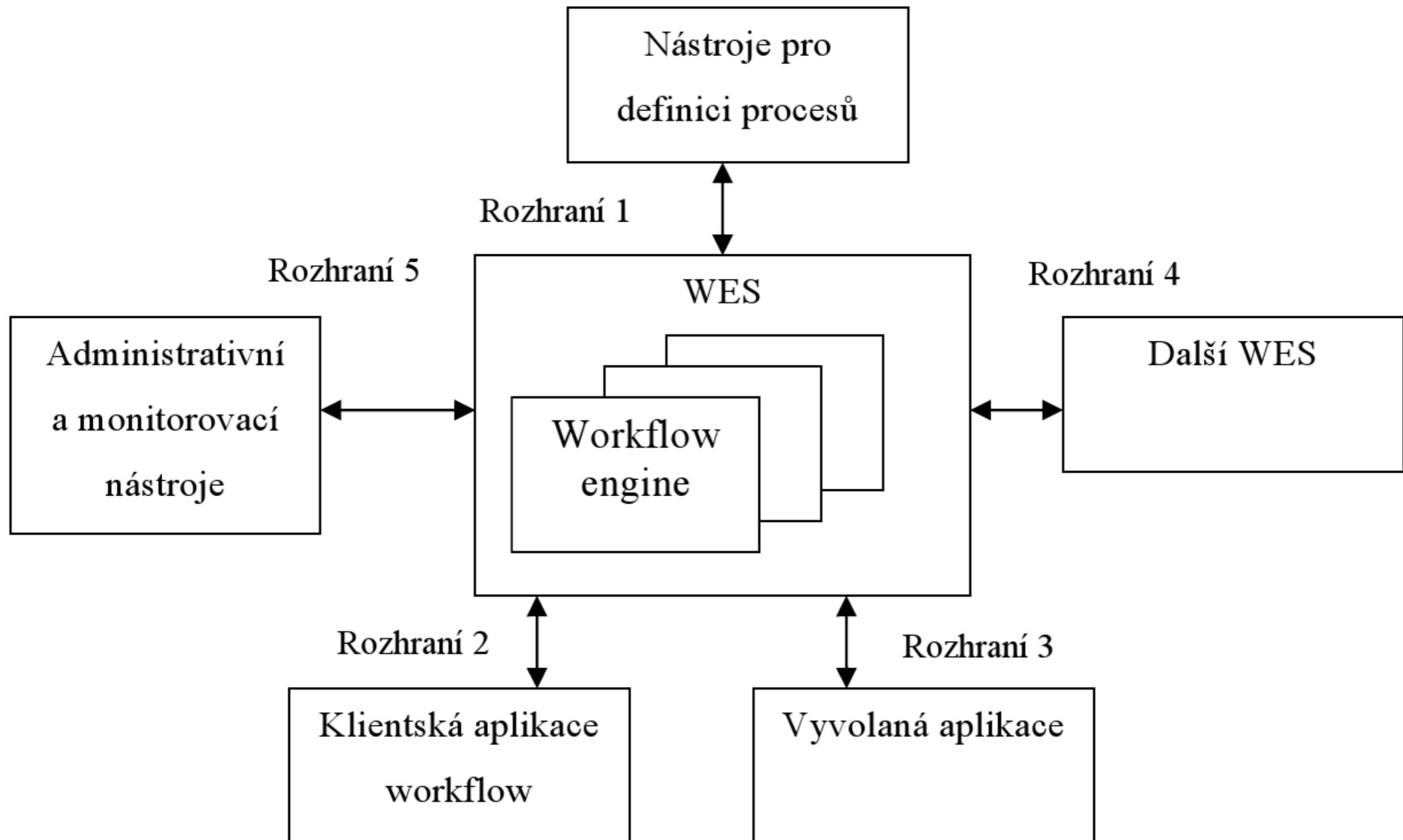
Standardizace

- Existuje množství SW nástrojů realizujících myšlenku workflow
- Potřeba integrace systémů – nutnost standardizace
- Workflow Management Coalition (WfMC)
 - založena 1993
 - nevýdělečná mezinárodní organizace prodejců, uživatelů, analytiků a univerzitních / výzkumných skupin (asi 130 členů)
 - tvorba standardů v oblasti
 - terminologie,
 - spolupráce a propojení wf systémů
 - tři komise a pracovní skupiny

Hlavní standardy

- Workflow Reference Model
- Workflow Client Application Application Programming
- Glossary
- Interoperability Abstract Specification
- Audit Data Specification
- Process Definition Interchange
- Interoperability Internet e-mail MIME Binding
- Objektový model (IDL a OLE)
- Bezpečná spolupráce wf systémů

Referenční model workflow



Prvky workflow

- WES = workflow enactment service
(workflow servery)
 - Zajišťuje vykonání správné činnosti pomocí správného prostředku ve správný čas
 - Složen z jednoho nebo více *workflow engines*
- Workflow engine
 - Interpretace definice procesu
 - Vytváří instance procesů a řídí jejich vykonávání
 - Zajišťuje přechody mezi aktivitami a vytváření pracovních položek
 - Další funkce pro správu a dohled

Prvky workflow

- Klientské aplikace workflow
 - Provádí jednotlivé úkoly
 - Interakce uživatelů s workflow
- Vyvolané aplikace
 - Spouštěné v souvislosti se započetím úkolu apod.

Prvky workflow (II)

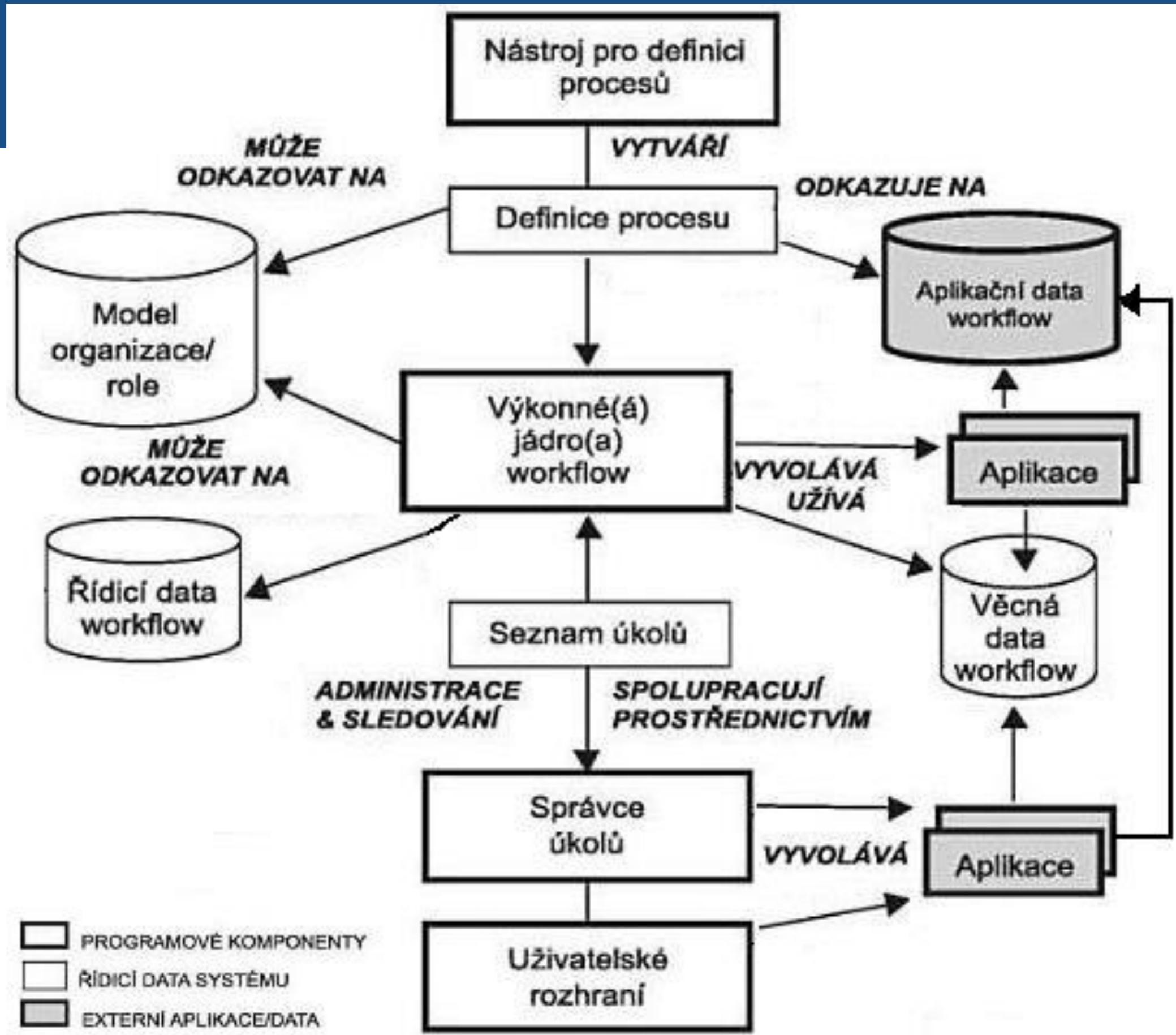
- Nástroje pro definici procesů
 - umožňují definici a rozplánování procesů na počítači
- obvykle grafické nástroje
- prvky modelu:
 - **zprávy** zaslané účastníkům procesu,
 - **události**, které mohou nastat,
 - **rozhodnutí**, která je třeba učinit;
- základní prvky určují charakter modelu

Prvky workflow (III)

- Nástroje pro simulaci procesů
 - Co se stane, když ... ?
 - Ověření modelu, predikce
- Nástroje pro verifikaci procesů
 - Bude každá objednávka vyřízena?
 - Bude každá reklamace vyřízena do 14 dnů?
 - Matematické metody – Petriho sítě
- Nástroje pro administraci

Rozhraní workflow systému

1. Pro nástroje pro definici procesů
2. Pro workflow klienty
3. Pro volané aplikace
4. Pro komunikaci s jinými WFM systémy
5. Pro administraci a monitorování



Data ve workflow

- Model organizační struktury
 - Role, vztahy nadřízený – podřízený
- Definice procesu
 - Činnosti, přidělení rolím, rozhodovací pravidla
- Seznam úkolů
 - Aktuální úkoly pro konkrétní uživatele
 - Uživateli bud' skryt (postupné přidělování úkolů) nebo přístupný (uživatel si volí pořadí, možno i více úkolů současně)

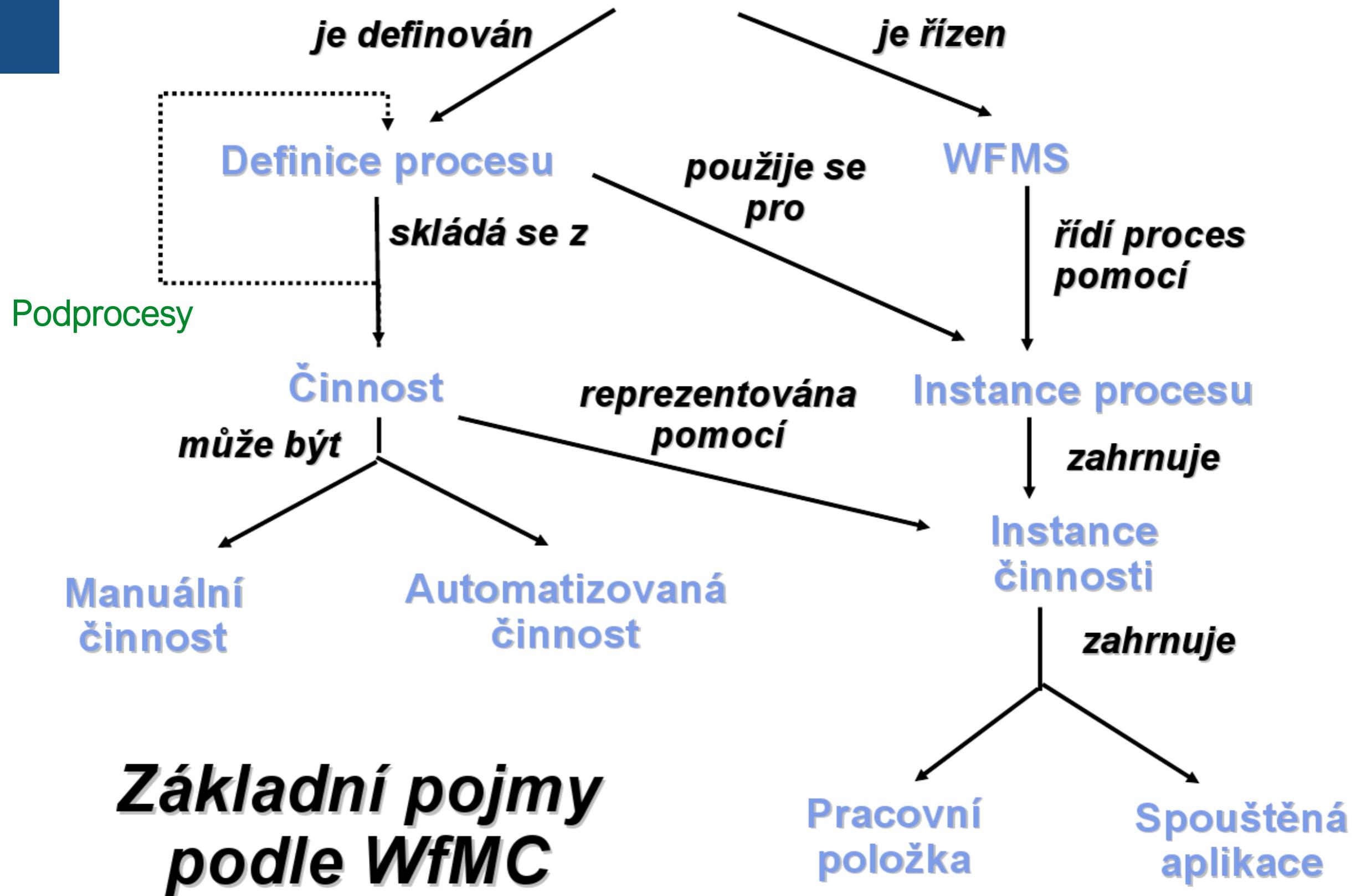
Data ve workflow

- Řídicí data workflow
 - Interní data WF systému nutná pro zajištění chodu příp. zotavení po havárii
 - Nedostupná externím aplikacím
- Věcná data workflow
 - Zpracování jádrem workflow systému
 - Používána pro rozhodování o dalším postupu
 - Dostupná i aplikacím
- Aplikační data workflow
 - Specifická data aplikací podporujících proces
 - Nejsou přístupná WF systému

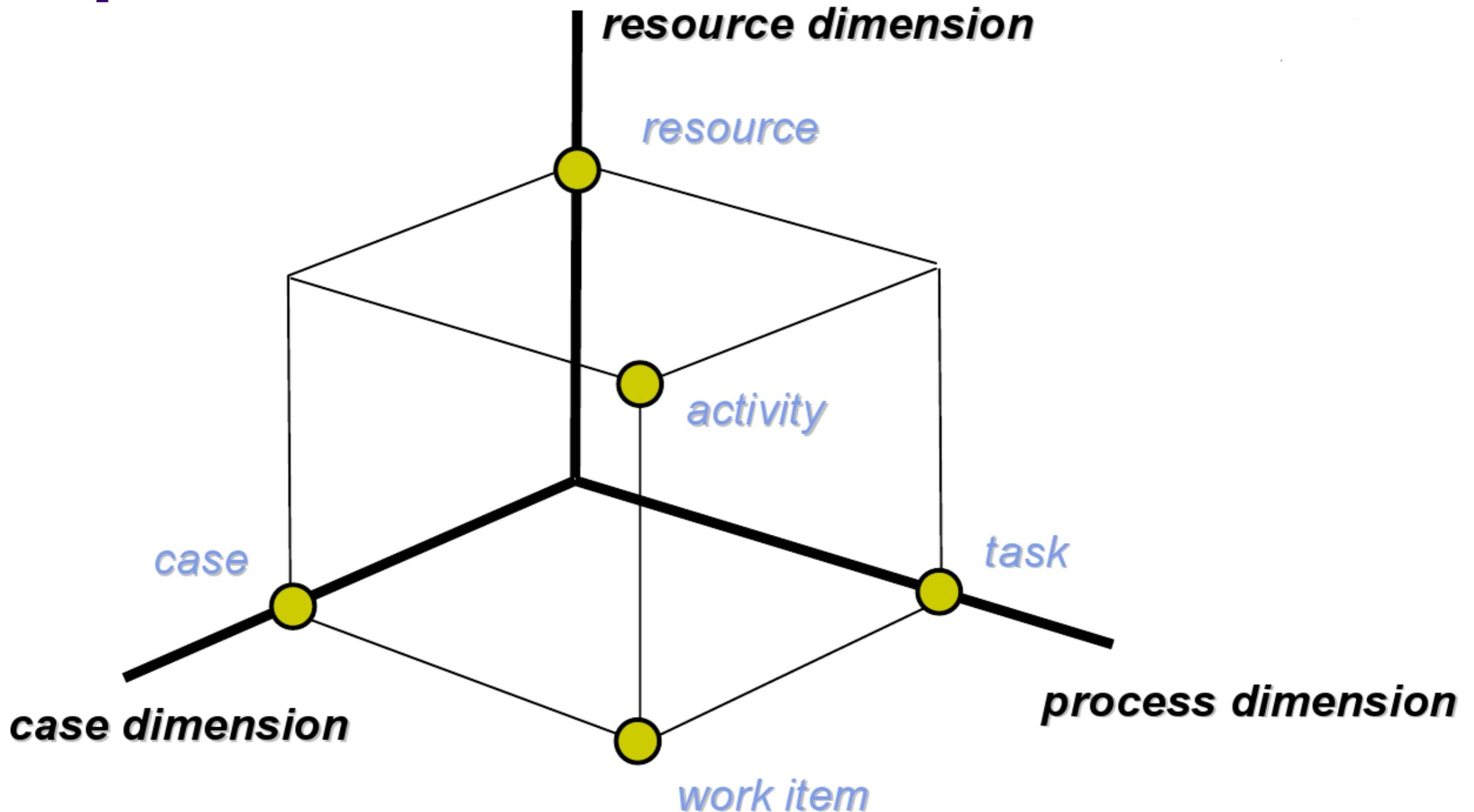
Reprezentace a provádění procesu

- Standardy WfMC definují základní pojmy používané pro reprezentaci workflow

Podnikový proces



3D pohled na workflow



3D pohled na workflow

- **případ (case)**
 - konkrétní řešený problém (žádost o půjčku)
 - obvykle jej generuje externí zákazník
 - zpracovává se prováděním úloh v určitém pořadí
 - na základě definice workflow procesu
- **úloha (task)**
 - krok provádění procesu
 - charakterizuje se podmínkami platnými před (*precondition*) a po (*postcondition*) provedení

3D pohled na workflow

- **zdroj (resource)**
 - zařízení (fax, tiskárna) nebo osoba (účastník, dělník, zaměstnanec)
 - vytvářejí **třídy zdrojů** na základě podobných charakteristik
 - **role** je třída založená na schopnostech svých prvků (např. programátoři)
 - **organizační jednotka** je třída založená na struktuře organizace (např. reklamační oddělení)

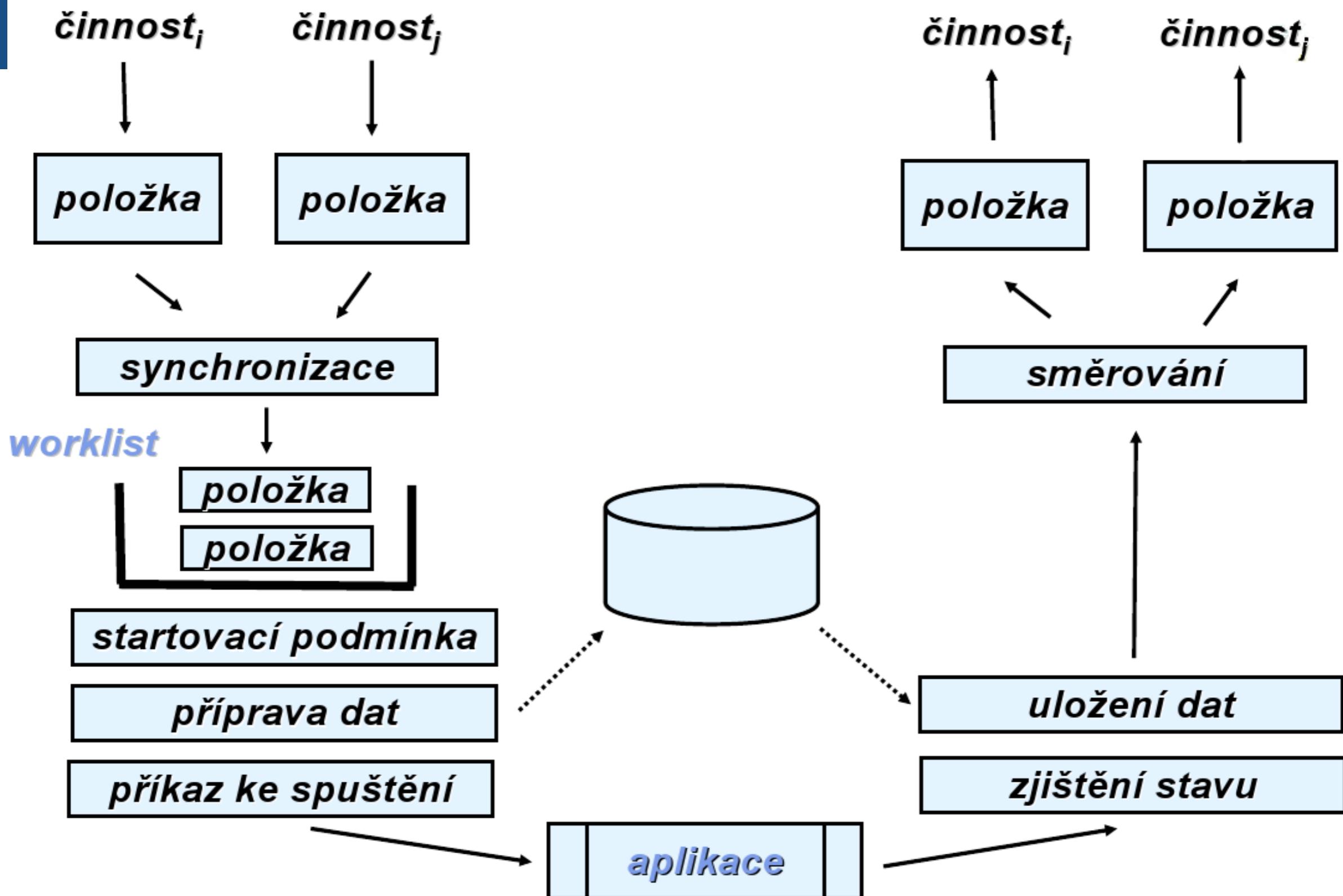
3D pohled na workflow

- **pracovní položka, požadavek (work item)**
 - úkol řešený pro konkrétní případ, např. „vrátit panu Novákovi peníze za reklamované zboží“
- **činnost (activity)**
 - úkol řešený pro konkrétní případ a využívající konkrétní zdroj
 - vytváří frontu požadavků (*worklist*)

Role

- práci vykonávají **kategorie pracovníků**
- jedna osoba může mít více rolí, mnoho osob má stejnou roli
- role jsou autorizovány provádět požadavky z front spojených s činnostmi
- požadavky na zpracování se přidělují staticky nebo dynamicky (load balancing)

Struktura (automatizované) činnosti



Struktura činnosti

- **Pracovní položka a fronta požadavků**
 - požadavky na provedení aplikace
 - strukturované zprávy obsahující parametry pro provedení činnosti
 - maximální doba provedení činnosti (připomenutí, předání jinam)
 - synchronizace paralelních instancí workflow
 - různé strategie: FIFO, LIFO, priority

Struktura činnosti

- **Příprava k provedení vybrané činnosti**
 - vyhodnocení vstupní podmínky na základě dat
 - závislých na řešeném případu
 - získání vstupních dat pro činnost
- **Akce jako jádro činnosti**
 - interaktivní: výběr položky uživatelem spustí provedení činnosti
 - automatické: příchod položky do fronty způsobí provedení činnosti

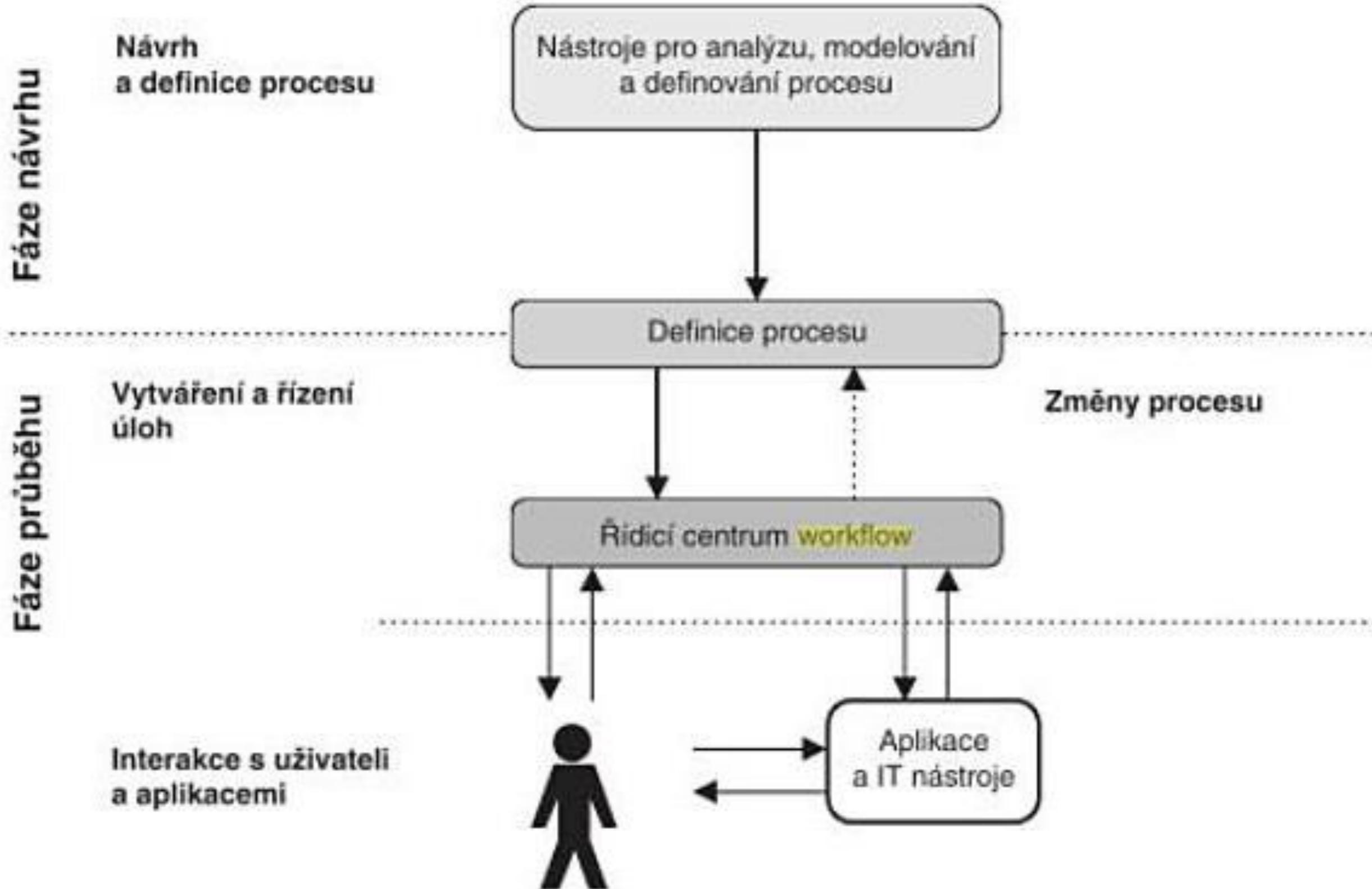
Struktura činnosti

- **Závěrečná analýza**
 - monitorování provádění aplikace: úspěch, chyba, havárie
 - uložení výsledků aplikace – konverze a uložení dat do společné paměti
- **Směrování**
 - přesun požadavků k dalším činnostem
 - na základě stavu (návratového kódu), výsledku



Modelování business procesů

Fáze vývoje workflow



Cíle BPM

- Formální popis procesů probíhajících v organizaci
- Možnost řízení takto popsaného procesu pomocí WFM systému
- Možnost analýzy, verifikace
 - Zvýšení efektivity

Standardy pro modelování

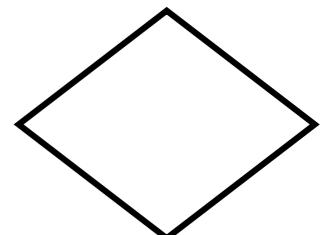
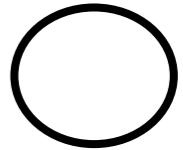
- Business Process Modeling Notation (BPMN)
 - Grafická notace pro specifikaci procesů
 - Diagramy BPD (Business process diagram)
- Jazyky pro popis procesu
 - BPEL (BPEL4WS, WS-BPEL) – procedurální
 - XPDL – deskriptivní
 - BPMN XML serializace (definována od BPMN 2.0)
- Je definováno mapování
 - Ukládání BPMN grafů v XPDL
 - Mapování BPMN -> BPEL

Elementy BPMN

- Objekty toku (flow objects)
 - Event (událost)
 - Activity (aktivita)
 - Gateway (brána)
- Spojovací objekty (connection objects)
 - Sequence flow (sekvenční tok)
 - Message flow (tok zpráv)
 - Association
- Plavecké dráhy (swimlanes)
- Artefakty (artifacts)

Objekty toku

- Událost
 - Ovlivňuje tok procesu
 - Začátek nebo konec procesu
- Aktivita
 - Práce, činnost která se má vykonat
 - Atomická nebo podproces
- Brána
 - Rozhodování, paralelní zpracování



Spojovací objekty

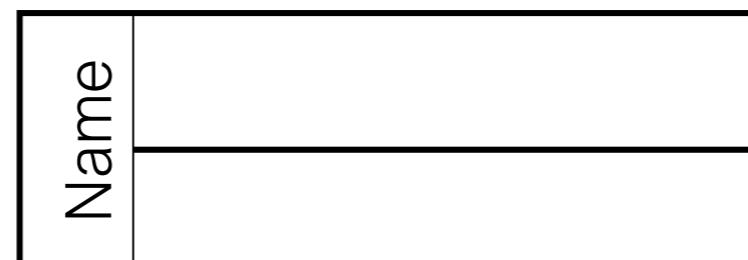
- Sekvenční tok
 - Navazující aktivity (pořadí)
- Tok zpráv
 - Zpráva mezi dvěma účastníky procesu
- Asociace
 - Propojuje objekt s dodatečnou informací

Plavecké dráhy (swimlanes)

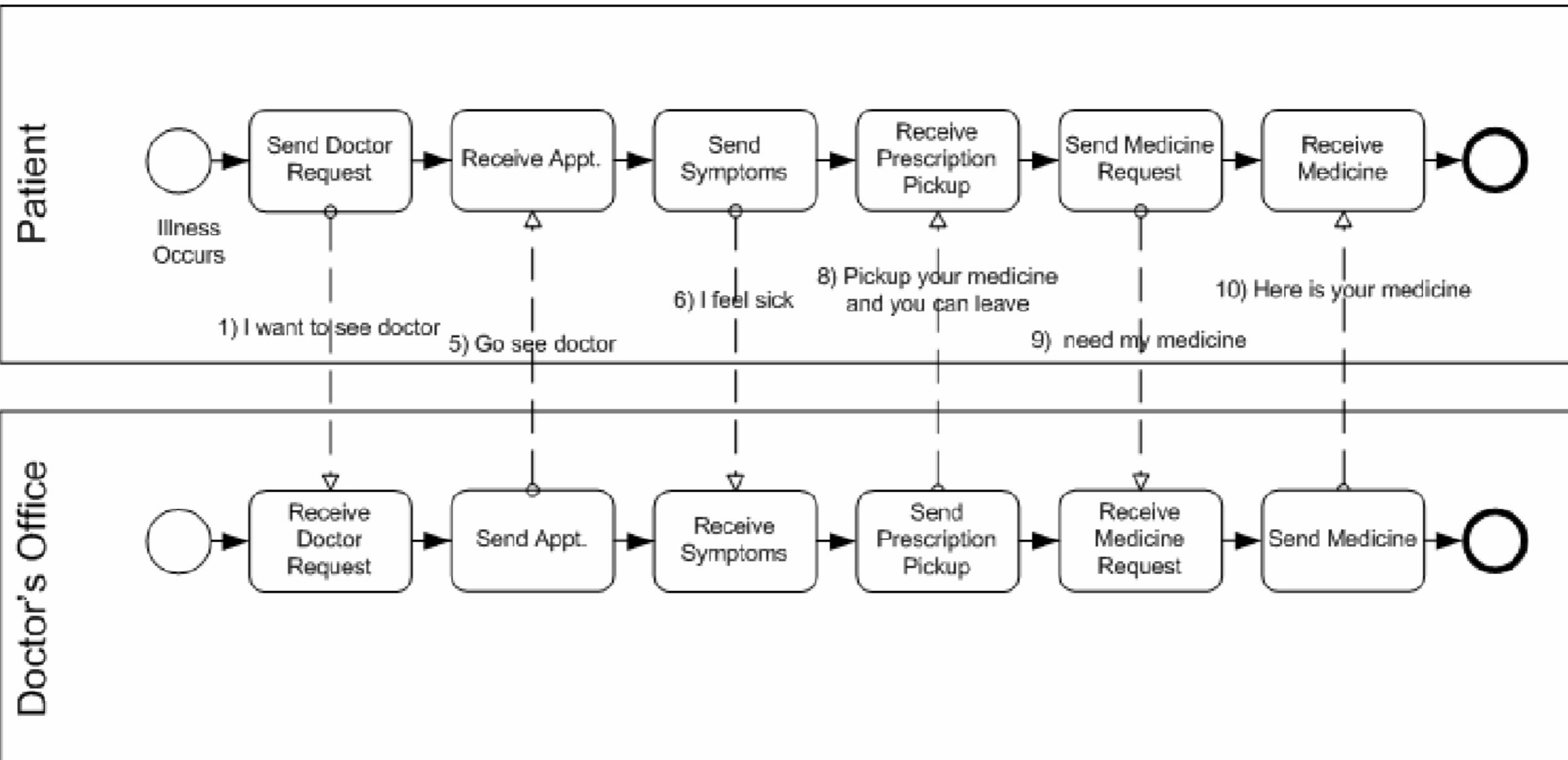
- Pool
 - Reprezentuje účastníky v procesech
 - Mezi pooly se komunikuje zprávami



- Swimlane
 - Kategorizují aktivity



Příklad poolu

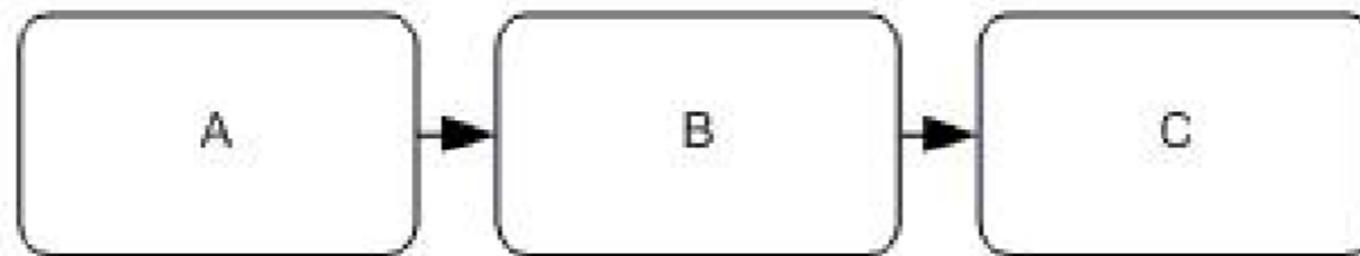


Směrování – řízení toku činnosti

- Rozdelení toku (split)
- Spojení (join)
- Různé druhy
 - XOR – vzájemné vyloučení
 - OR – nebo
 - AND – současně

Workflow patterns

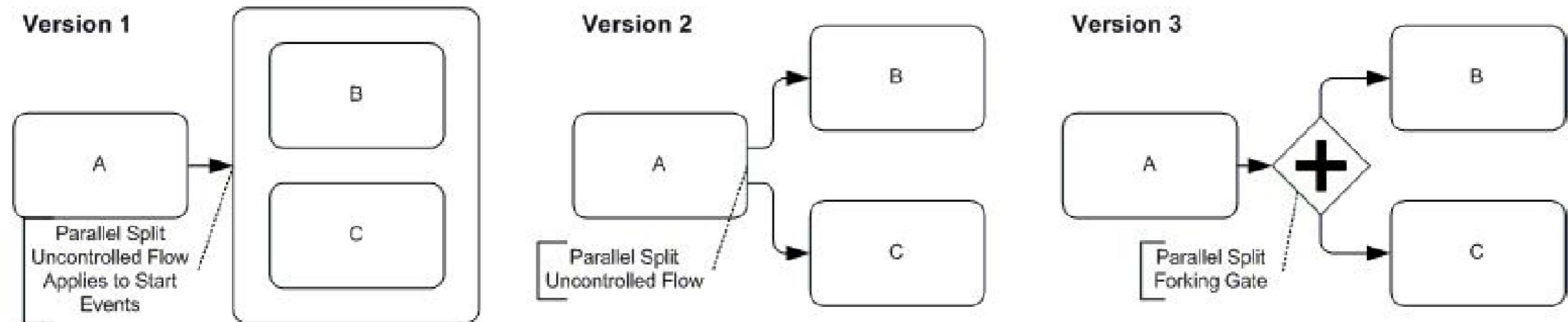
- Sekvence
 - Pracovní úkol je v procesu povolen, až je dokončeno provedení předcházejícího úkolu v procesu.



<http://www.workflowpatterns.com/>

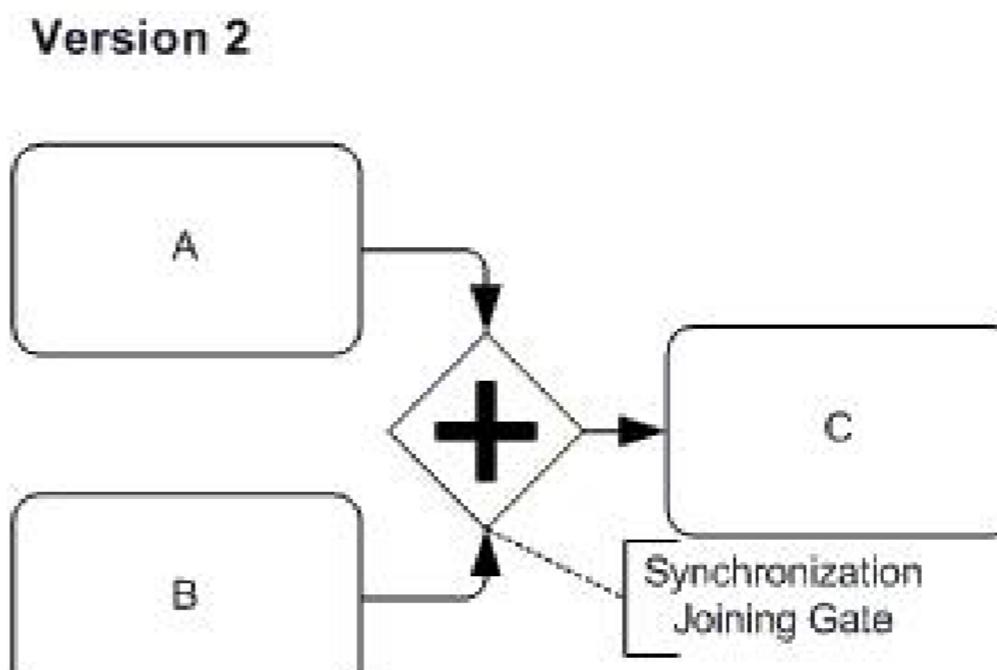
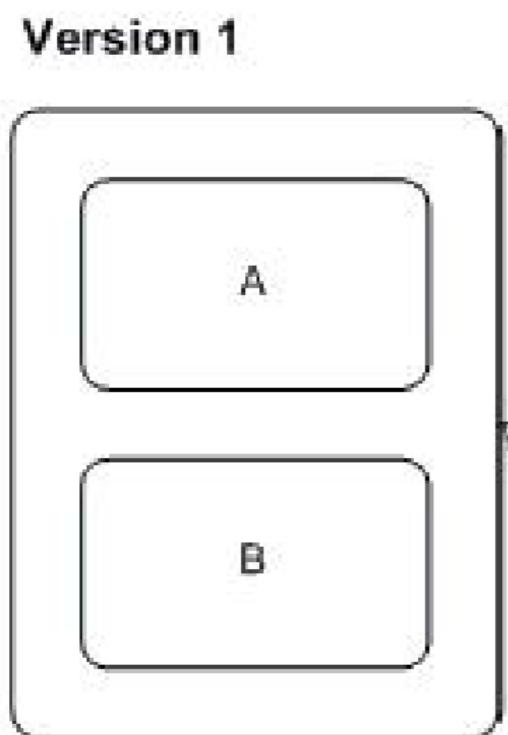
Workflow patterns

- Parallel split (AND-split)
 - Rozděluje tok procesů (workflow) do dvou a více paralelních vláken.



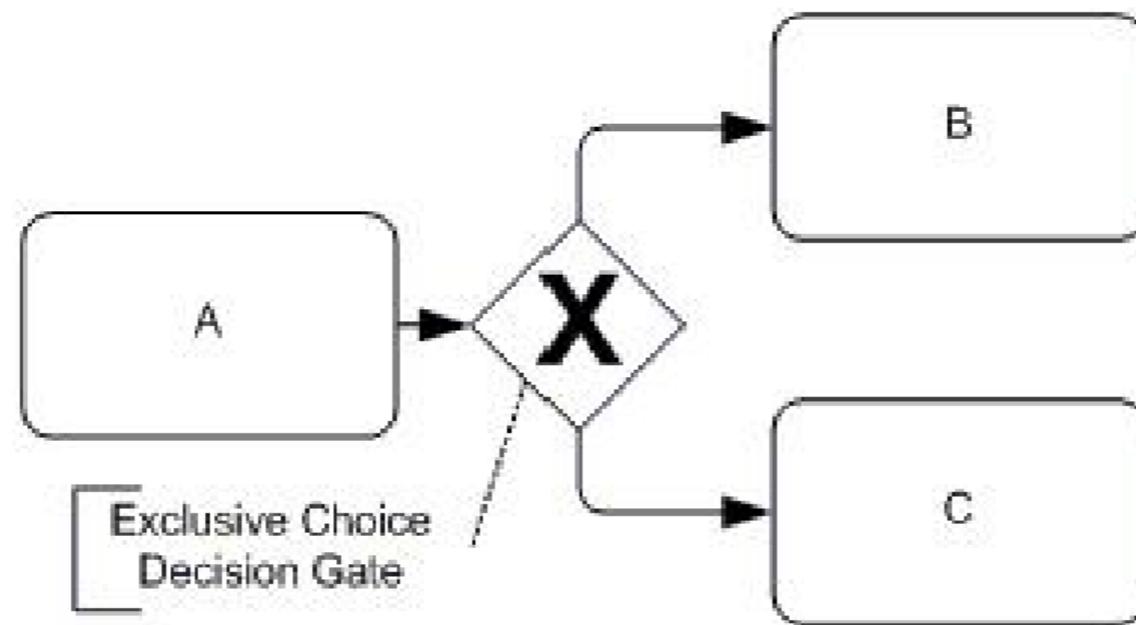
Workflow patterns

- Synchronizace (AND-join)
 - Dalším úkolem se pokračuje, až po dokončení všech předchozích vláken.



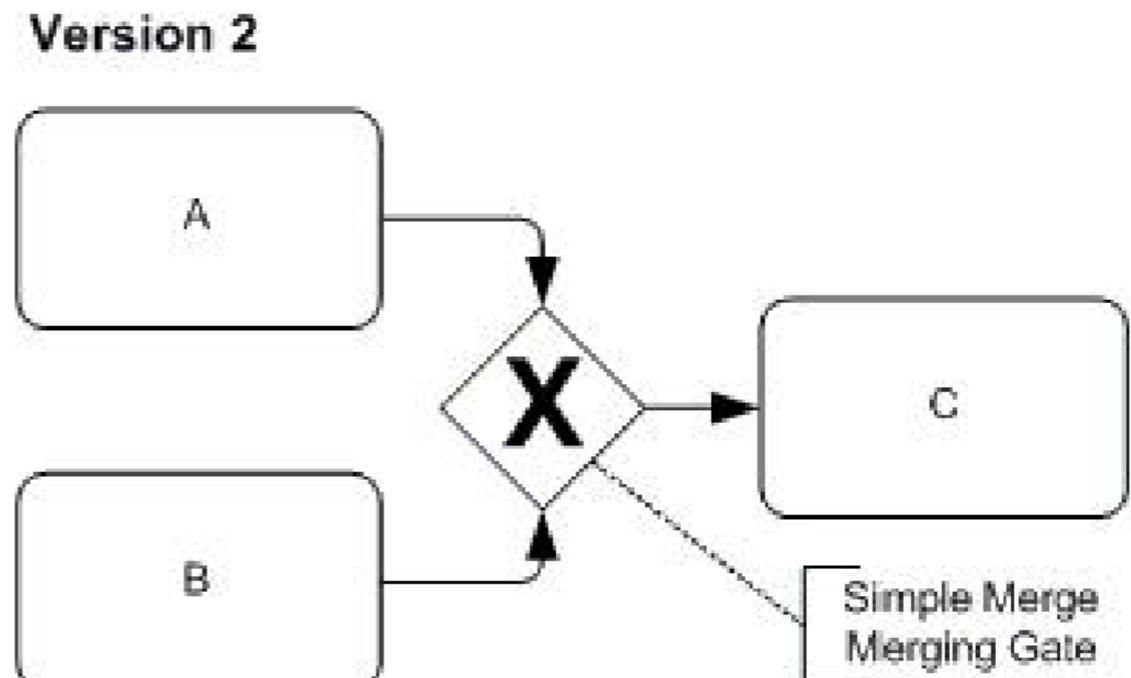
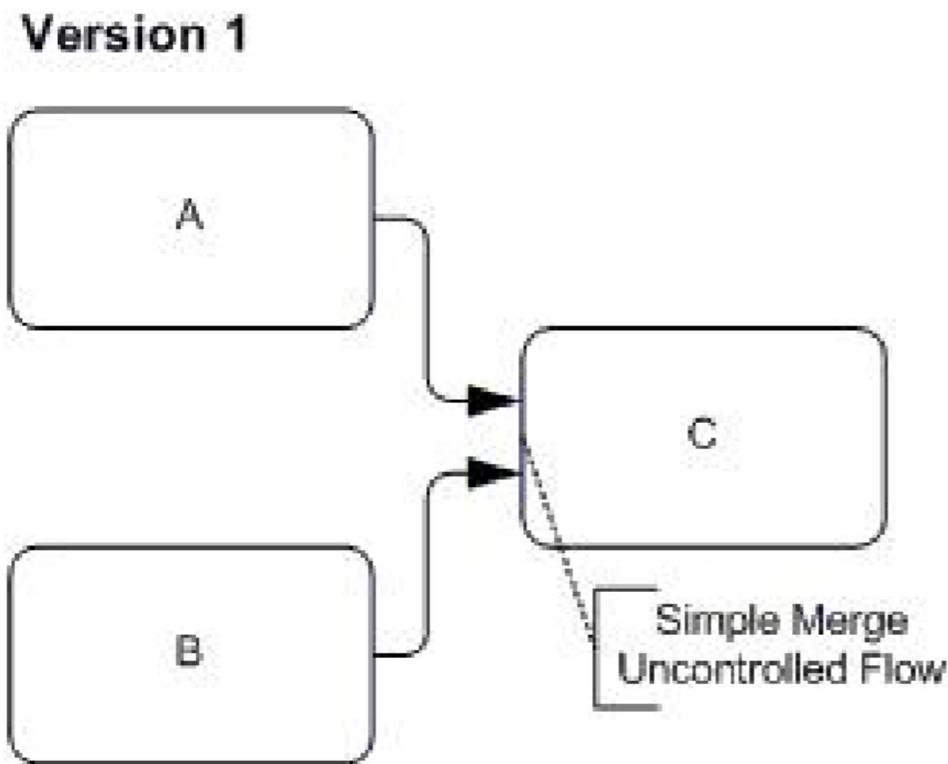
Workflow patterns

- Výlučné rozhodnutí (XOR-split)
 - Rozděluje tok procesů na dvě nebo více větví, které jsou vzájemně výlučné. Podle podmínky v Gateway se vstupuje do jedné z větví.



Workflow patterns

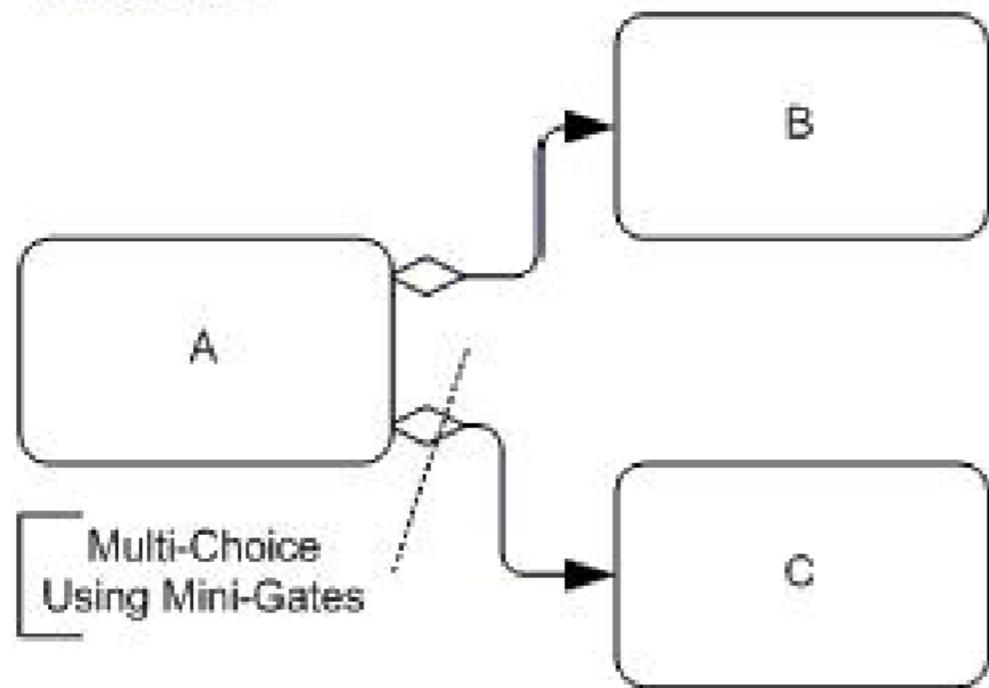
- Jednoduché spojení (XOR-merge)
 - Spojení dvou nebo více nezávislých větví do jedné. Navazující aktivita začne okamžitě, jakmile jedno vlákno dosáhne svého konce.



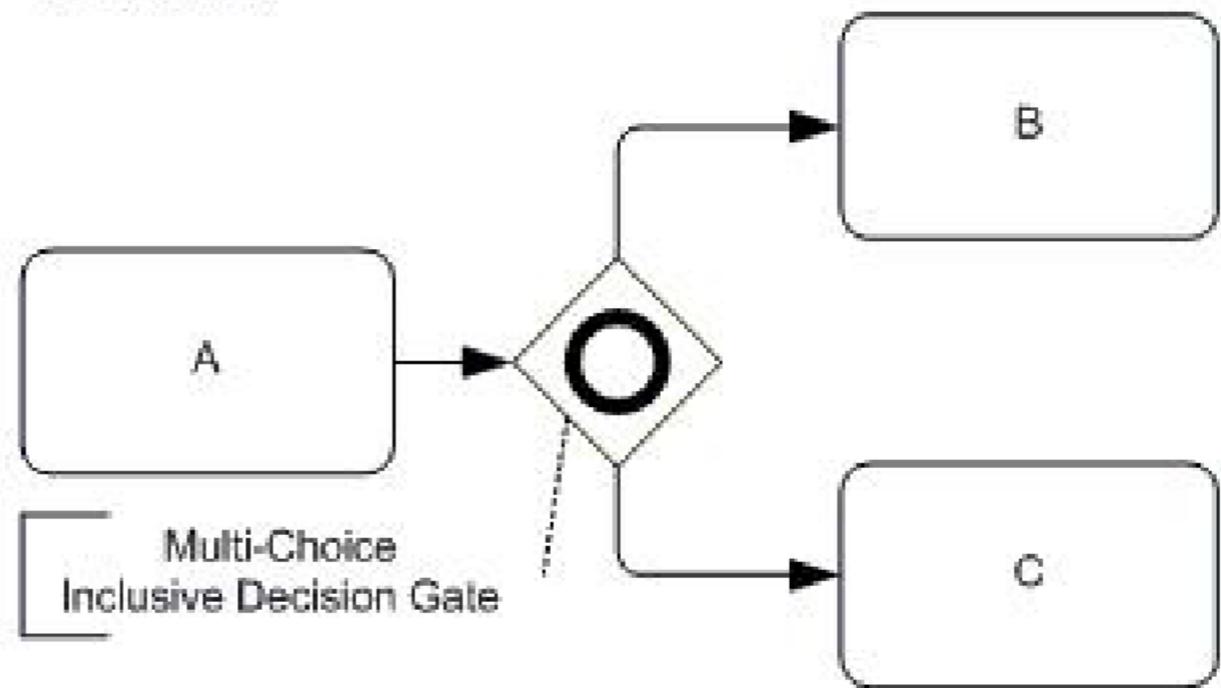
Workflow patterns

- Multi-choice (OR-split)
 - Jedna nebo více variant

Version 1

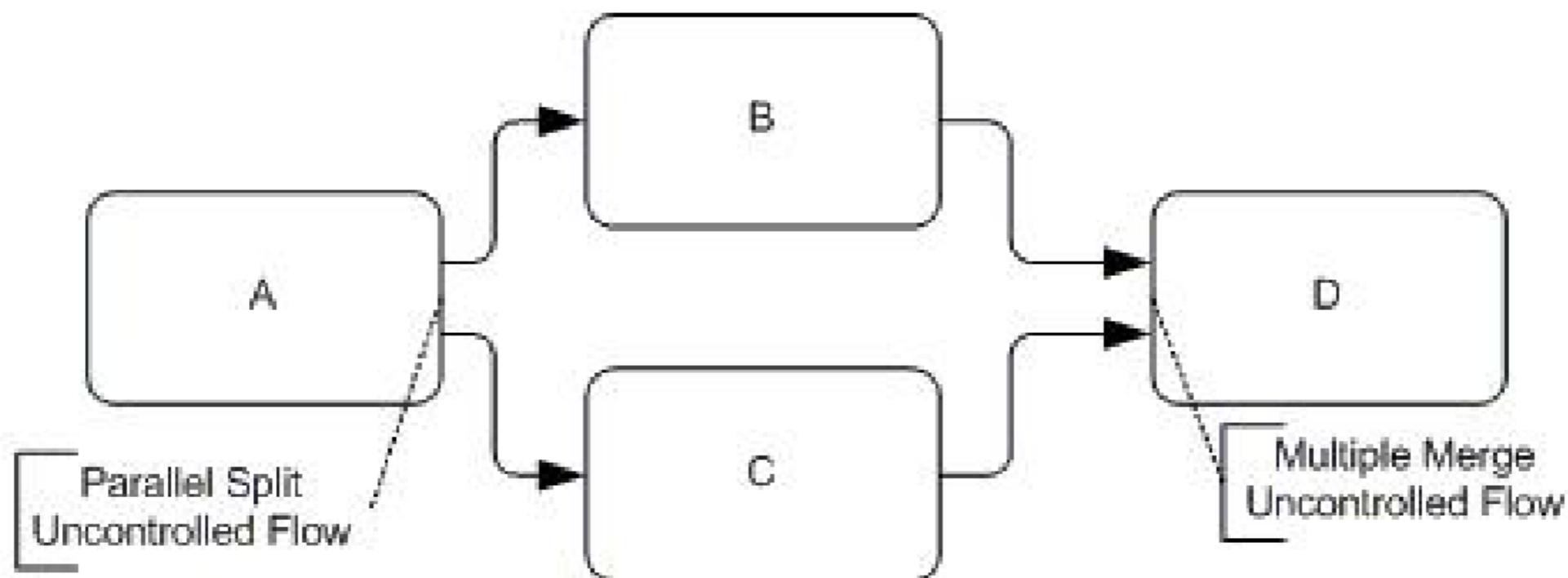


Version 2



Workflow patterns

- Structured Synchronizing Merge (OR-join)
- Skončí všechno, co začalo (kontext)



Pravidla pro přechod mezi činnostmi

- Lhůta (deadline)
- Vstupní podmínka (pre-condition)
 - Musí být splněna pro spuštění činnosti
 - Vyhodnocována WF systémem
- Výstupní podmínka
 - Musí být splněna pro ukončení činnosti
 - Do té doby činnost trvá, příp. se opakuje
- Přechodová podmínka
 - Umožňuje určit pořadí zpracování činností
 - Např. mimořádné situace

Flexibilita workflow

- Podmínky pro chod organizace se mění
 - Změna legislativy, restrukturalizace, ...
- Zajištění flexibility
 - Dopředné – uvažujeme všechny možné situace ve workflow
 - Zpětné – změna workflow za běhu

Dynamická změna workflow

- Pomocí základních změn (evolution patterns)
 - Vždy je definováno, zda lze změnu provést a jak
- Převod existujících instancí
 - Concurrent to completion
 - Migrace na finální schéma
 - Jen za určitých podmínek
 - Migrace na ad-hoc schéma
 - Verifikace výsledku
 - Bude WF stále dělat to, co má?

Verifikace workflow

- Analýza cest
 - Dosažitelnost stavů
 - Petriho sítě



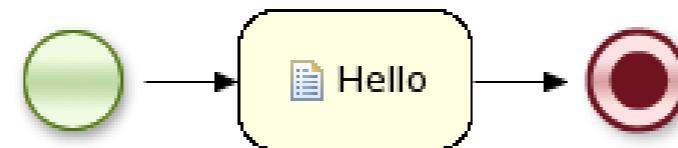
Technologie

Jazyk XPDL

- XML Process Description Language
- Jazyk popisující BPMN graf
- Aplikace XML
- Prvky dokumentu
 - Package, application
 - Workflow process
 - Activity
 - Transition
 - Participant
 - DataField
 - DataType

XML Serializace BPMN 2.0

- Obdobné použití, jako XPDL



```
<process processType="Private" isExecutable="true" id="com.sample.HelloWorld",
  name="Hello World" >

  <!-- nodes -->
  <startEvent id="_1" name="StartProcess" />
  <scriptTask id="_2" name="Hello" >
    <script>System.out.println("Hello World");</script>
  </scriptTask>
  <endEvent id="_3" name="EndProcess" >
    <terminateEventDefinition/>
  </endEvent>

  <!-- connections -->
  <sequenceFlow id="_1-_2" sourceRef="_1" targetRef="_2" />
  <sequenceFlow id="_2-_3" sourceRef="_2" targetRef="_3" />

</process>
```

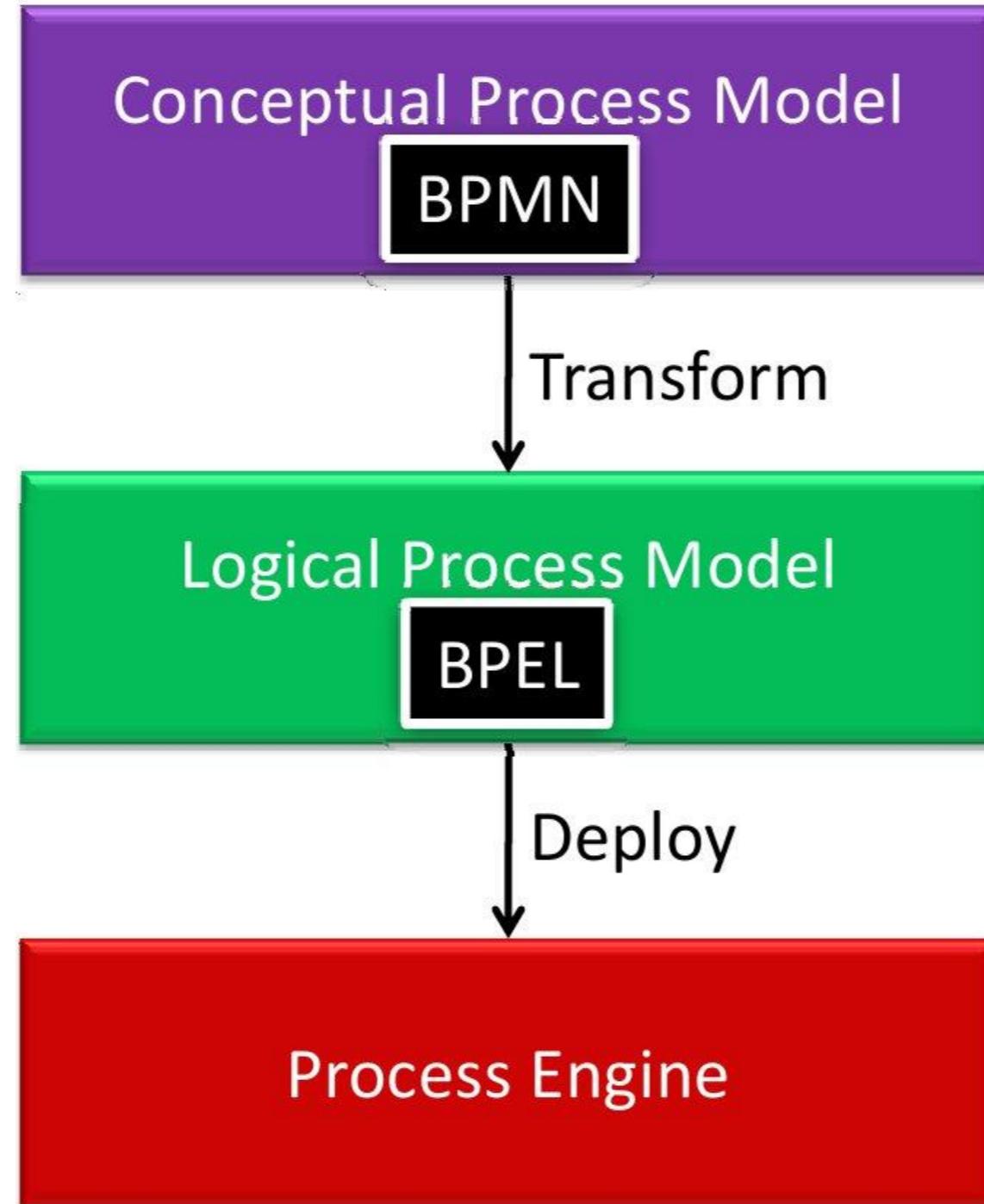
Jazyk BPEL

- Business Process Execution Language
- De-facto průmyslový standard
- Procedurální jazyk, opět XML
- Předpokládá implementaci úkolů pomocí webových služeb
 - Orchestrace volání webových služeb
- BPMN 2.0 obsahuje významnou podmnožinu ekvivalentních BPEL
 - Je možno použít BPMN 2.0 engine místo BPEL engine

Webové služby

- Standard komunikace v distribuované aplikaci
 - Vzdálené volání funkcí
 - Výměna dokumentů
- Standardní jazyky
 - Popis rozhraní služby (WSDL)
 - Výměna zpráv (SOAP)
- Komunikace
 - Obvykle HTTP

Vrstvy modelování procesů



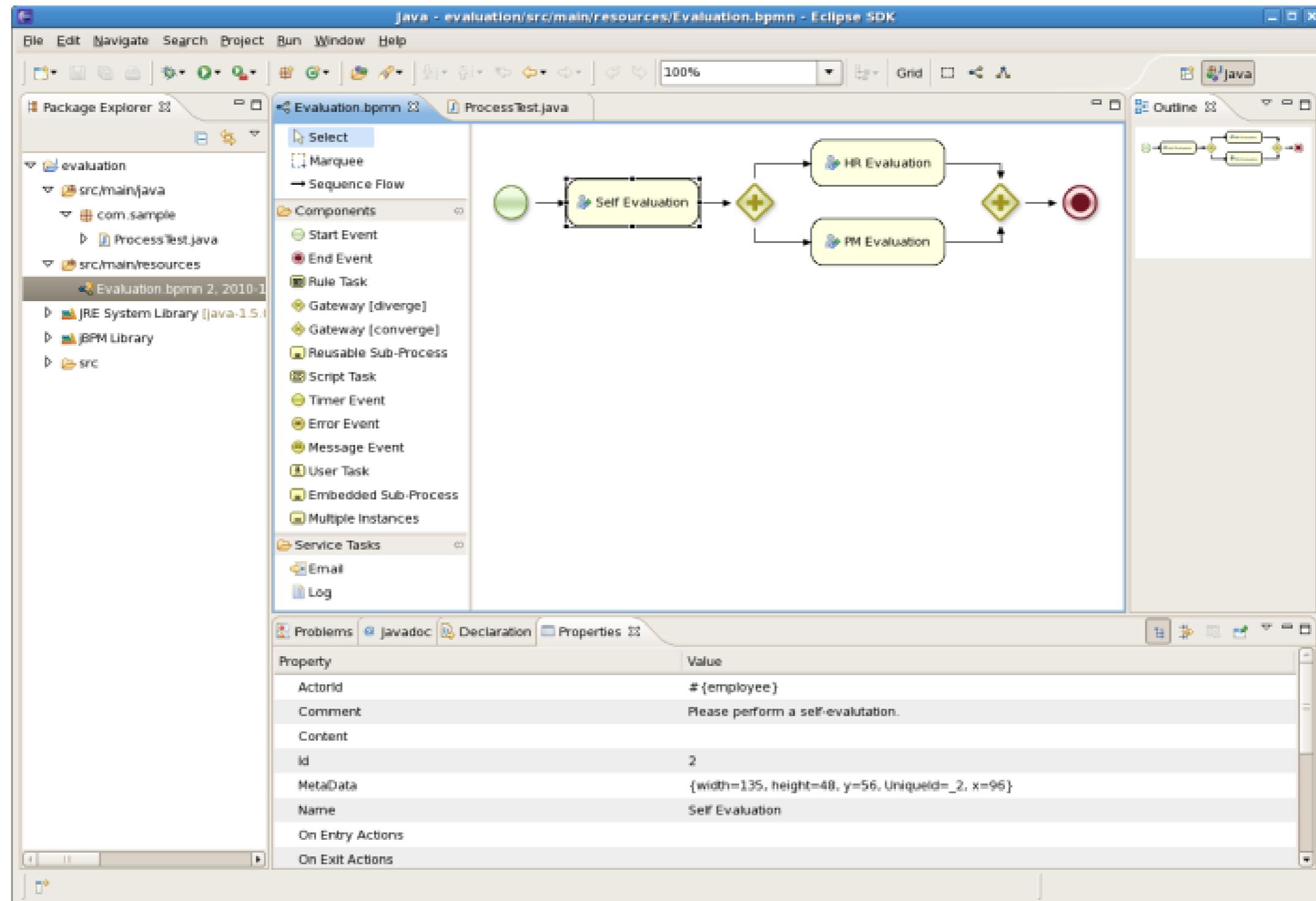
BPEL & BPMN engines

- Apache ODE
- MS BizTalk Server
- Oracle BPEL Process Manager
- IBM WebSphere Process Server
- jBoss jBPM
- ...

jBoss jBPM

- Framework implementující workflow, BPM a orchestraci procesů
- Běží na jBoss nebo jiném JEE serveru
- Snadná integrace Java EE aplikací
 - Web services, Java Messaging, JDBC, EJB
- Zajišťuje správu stavů a úloh
- Měří časy provádění kroků, logování
- Unifikuje správu workflow procesů

jBPM Eclipse Plugin



Jazyky v jBPM

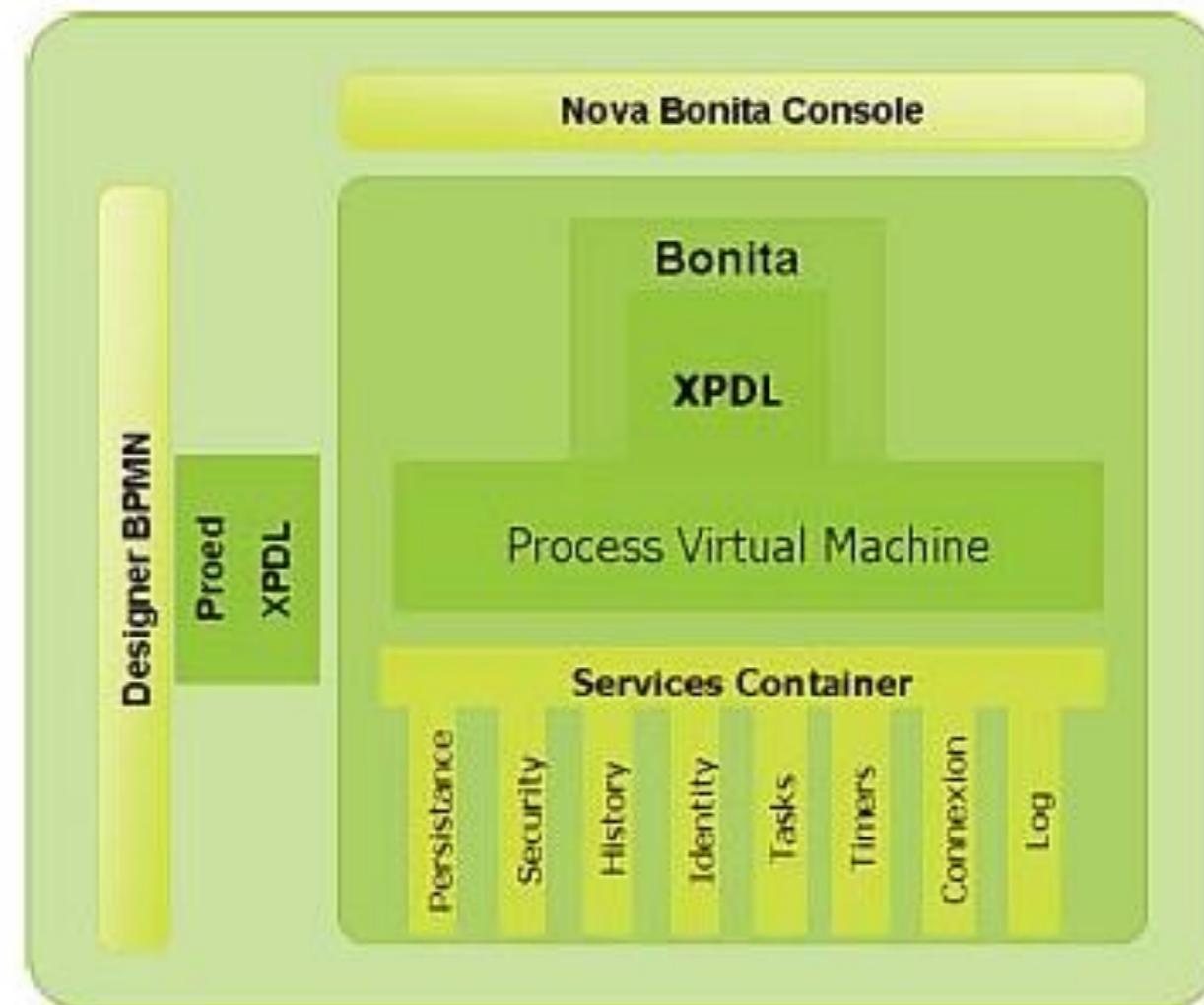
- Grafický editor BPMN
 - Eclipse
- BPMN 2.0 serializace
 - Workflow management
 - Správa úkolů prováděných lidmi
- BPEL
 - Web service orchestration v rámci JBoss BPEL serveru
 - Opuštěno v novějších verzích jBPM

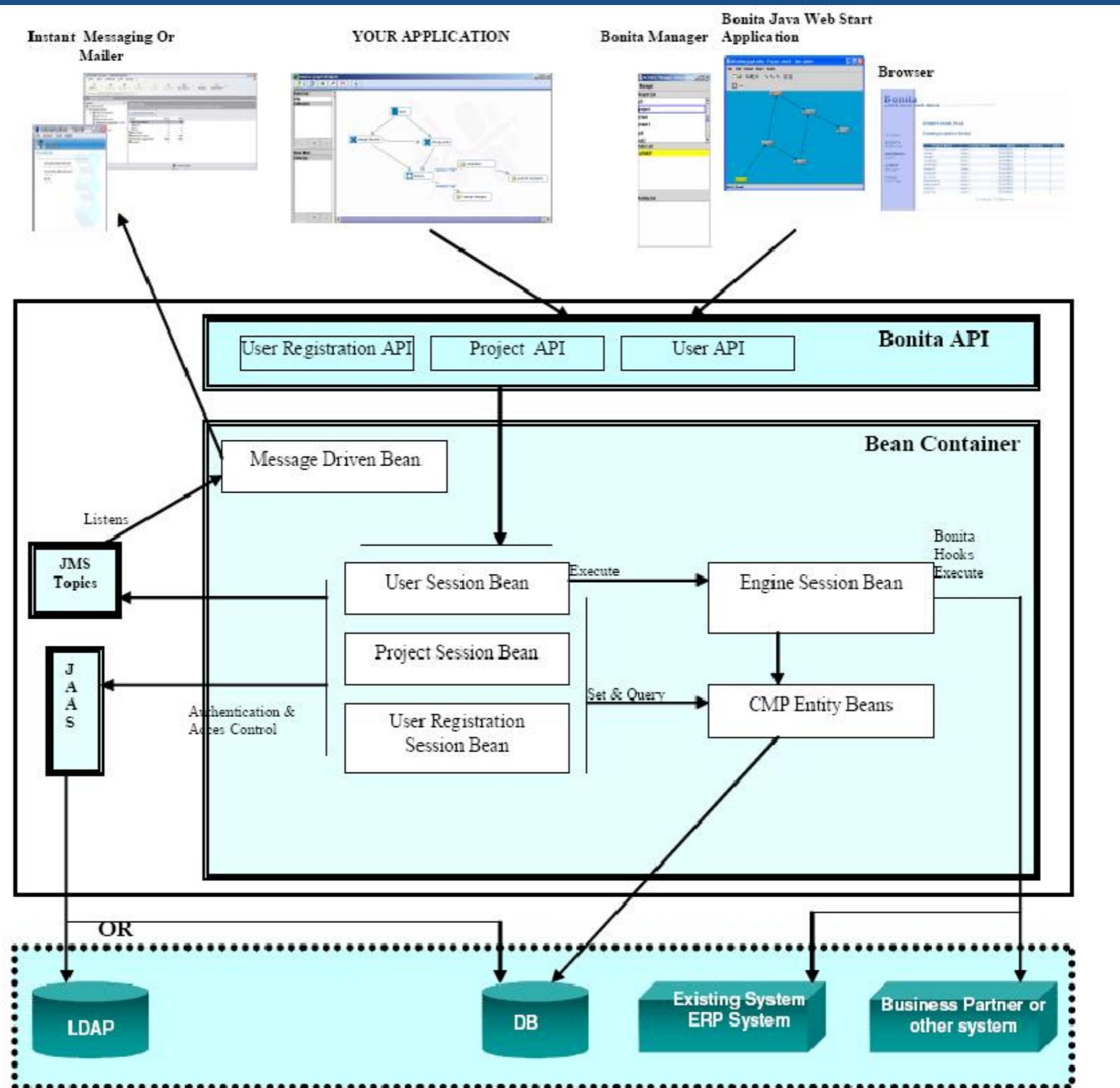
AgilPro

- Open source nástroje pro modelování procesů
- AgilPro LiMo
 - Grafický editor procesů
- AgilPro Simulator
 - Simulátor procesů
- Založeno na platformě Eclipse + JWT (Java Workflow Tooling)
 - Vlastní formát JWT
 - Export/import z XPDL, BPMN, atd.

Bonita

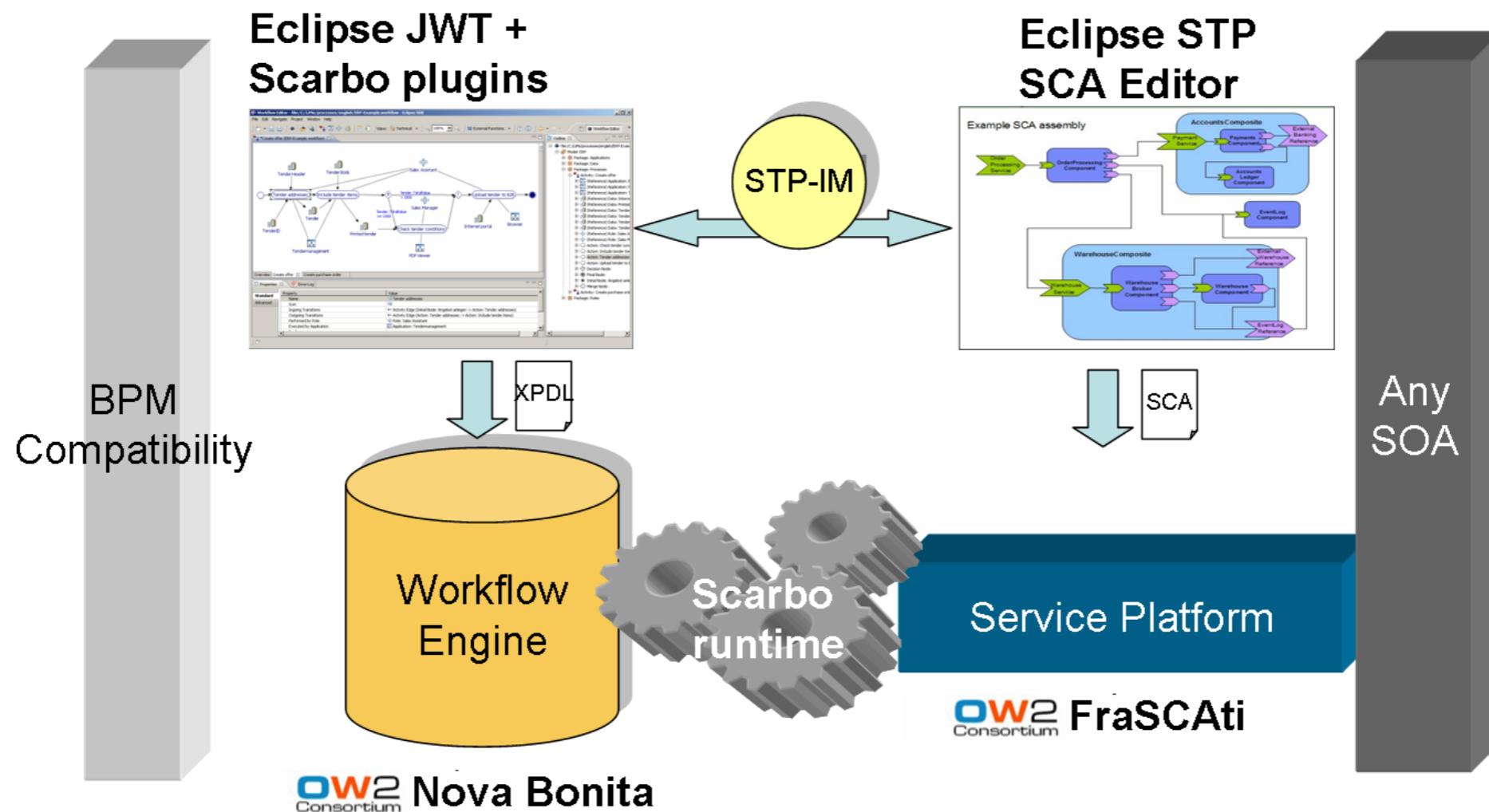
- Open source projekt
 - Java EE a XPDL
 - API pro vytváření i provoz workflow
- Workflow prováděný virt





Scarbo

- Grafický modeler + engine
- Založeno na Eclipse a Bonita



Literatura

- M. Beneš: Úvod do technologie workflow systémů (slidy)
- T. Novotný: Workflow – BPM Systémy
- P. Opletal: Procesy a workflow (slidy)
- V. Mates, T. Hruška: Workflow, studijní opora

Otázky?



Pokročilé informační systémy

Grafové databáze, sémantický web, ontologie

Ing. Radek Burget, Ph.D.
burgetr@fit.vutbr.cz

Datové modely

- **Jednoduché (NoSQL)**
 - Key-value (MUMPS, Redis, ...)
 - Dokumentové (MongoDB, CouchDB, ...)
 - Sloupcové (Apache HBase, ...)
- **Relační datový model**
 - Mnoho implementací
- **Objektový datový model**
 - Objektově-relační mapování (ORM)
- **Grafové**
 - **Grafové databáze (Neo4J, OrientDB, ...)**
 - Sémantická úložiště (sémantický web, RDF)

Neo4j

- Tradiční grafový model – graf (V, E)
 - *Labeled property graph*
- Uzly (*nodes*)
 - label(s) – lze použít jako typ uzlu (:Person)
 - properties [key: value, key: value, ...]
- Hrany (vztahy, *relationships*)
 - Typ (:OWNS_VEHICLE)
 - properties [key: value, key: value, ...]

Cypher

- Jazyk pro dotazování a manipulaci s daty (i serializaci grafu)

```
CREATE (sally:Person { name: 'Sally', age: 32 })
CREATE (john:Person { name: 'John', age: 27 })
CREATE (sally)-[:FRIEND_OF { since: 1357718400 }]->(john)
```

- Dotaz (výsledkem je tabulka)

```
MATCH (sally:Person { name: 'Sally' })
MATCH (john:Person { name: 'John' })
MATCH (sally)-[r:FRIEND_OF]-(john)
RETURN r.since AS friends_since
```

Implementace

- Embedded databáze nebo samostatný server
 - bolt://localhost:7687
- Java API
 - Vytváření a procházení grafu (Node, ...)
 - Zasílání Cypher dotazů
 - Možnost mapování Java POJOs
- API pro další platformy
 - .NET, JavaScript, Python

Sémantický web – RDF

Datové modely

- **Jednoduché (NoSQL)**
 - Key-value (MUMPS, Redis, ...)
 - Dokumentové (MongoDB, CouchDB, ...)
 - Sloupcové (Apache HBase, ...)
- **Relační datový model**
 - Mnoho implementací
- **Objektový datový model**
 - Objektově-relační mapování (ORM)
- **Grafové**
 - Grafové databáze (Neo4J, OrientDB, ...)
 - **Sémantická úložiště (sémantický web, RDF)**

Účel sémantického webu

- Aplikační databáze
 - Lokální úložiště
 - Sémantika dat je vázána na aplikaci
 - Např. názvy sloupců v relační databázi
 - Špatně se sdílí např. na webu
- Myšlenka sémantického webu
 - Každý údaj má explicitně určenou sémantiku
 - Identifikace pomocí URI
 - Údaje na sebe mohou vzájemně odkazovat

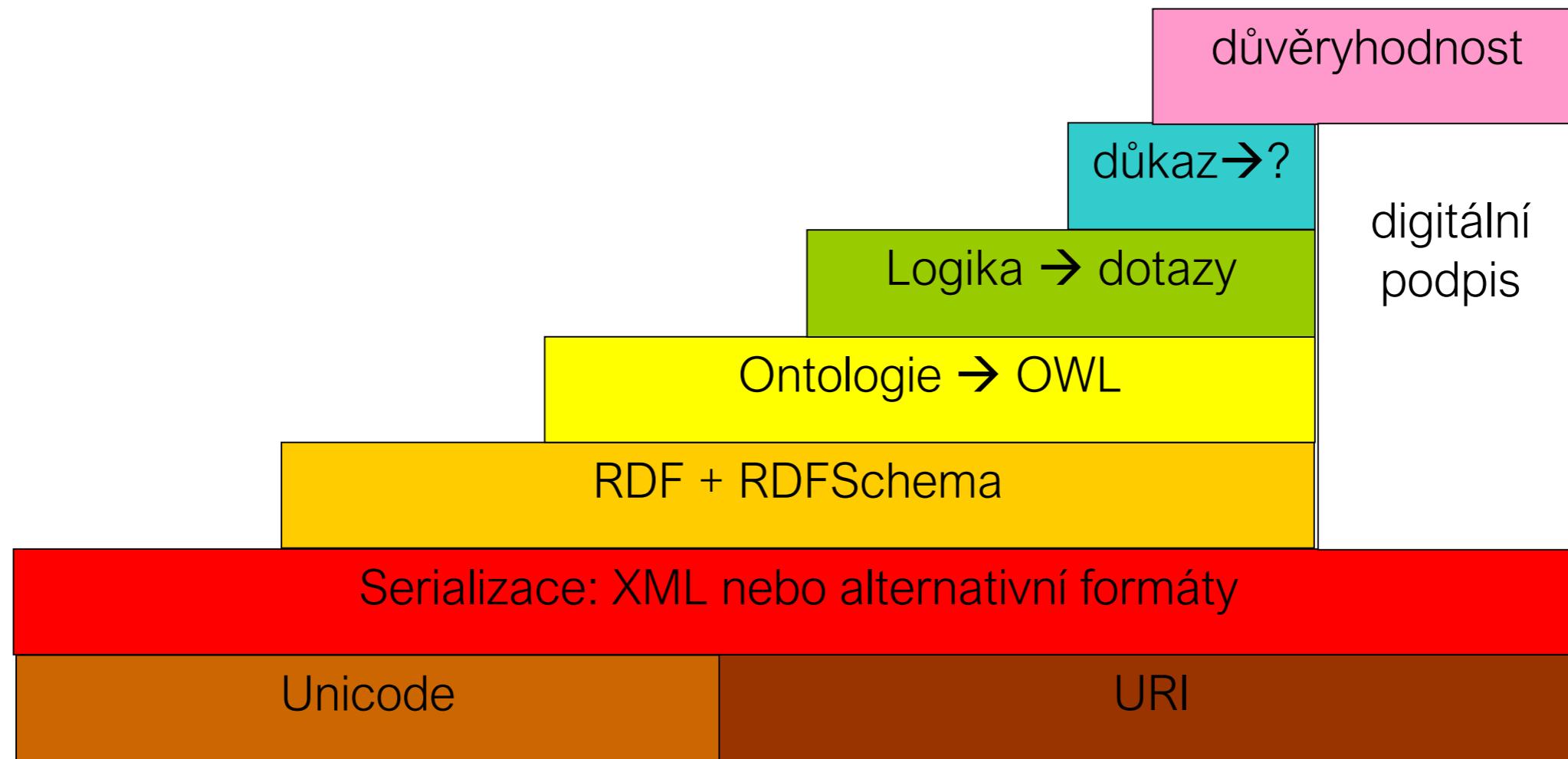
Web a sémantický web

- World Wide Web (web)
 - Základní jednotkou je dokument
 - „Web of documents“
- Semantic Web (sémantický web)
 - Základními jednotkami jsou data
 - „Web of Data“, „Linked data“

Technologie sémantického webu

- Technologie standardního webu
 - HTTP, URI
- Nástroje pro reprezentaci znalostí
 - Reprezentace dat (faktů)
 - XML, RDF, ...
 - Sémantika
 - Ontologie
 - Technologie pro reprezentaci ontologie

Vrstvový model sémantického webu



Reprezentace faktů

XML a RDF

Cíle a prostředky

- Cíle
 - Reprezentace strukturovaných dat a jejich významu (sémantiky)
 - Možnost sdílet data a jejich sémantiku napříč aplikacemi
- Běžná reprezentace dat v IS:
 - Relační/objektové/NoSQL databáze – vázané na aplikaci
 - Veřejné API + serializace (JSON, XML) – není definována sémantika

Serializace – příklad

```
<nabidka>
  <polozka>
    <velikost>3+1</velikost>
    <lokalita>Brno-střed</lokalita>
    <cena mena="czk">2 200 000</cena>
  </polozka>
  <polozka>
    <velikost>2+1</velikost>
    <lokalita>Kuřim</lokalita>
    <cena mena="czk">450 000</cena>
  </polozka>
</nabidka>
```

Problémy

- Význam elementů je specifický pro danou aplikaci
 - Je definován v programovém kódu, který generuje nebo načítá serializovaná data
 - Obdobně jako např. sloupce v relační databázi
- Jiná aplikace může stejným elementům přiřadit jiný význam
 - Např. <velikost>2+1<velikost> vs. <velikost>55m²</velikost>

Reprezentace sémantiky

- Odlišení značek v různých aplikacích
 - Např. XML namespaces
 - Řeší kolize značek – syntaktický problém
- Oddělená definice významu značek
 - Např. doprovodný dokument vysvětlující význam a případy použití
- Navíc ale potřebujeme definovat sémantické vztahy
 - Např. byt je věc, která má umístění, velikost a cenu
 - Pokud možno formálně => **Ontologie**

Reprezentace faktů

- XML
 - Mapování elementů na vlastnosti ontologií
 - Pouze hierarchická struktura – omezující
- RDF
 - Grafová struktura
 - Lze zapsat pomocí XML nebo jiných jazyků

RDF

- RDF je datový model standardizovaný W3C
 - Zaměřeno na data sdílená na venek
 - Snadné propojení dat z různých zdrojů a na různých schématech (linked data) (<http://lod-cloud.net/>)
- Existují různá úložiště
 - RDF úložiště napsané v Javě <http://rdf4j.org/>
 - Původně známé jako *Sesame*, nyní pod Eclipse Foundation
 - Blazegraph (Java) <https://www.blazegraph.com/>
 - OpenLink Virtuoso (C++) <https://virtuoso.openlinksw.com/>

RDF trojice

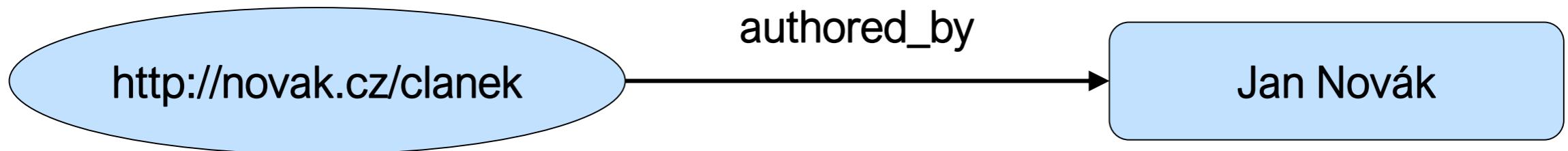
- Základním prvkem je **RDF trojice**

subjekt – predikát – objekt

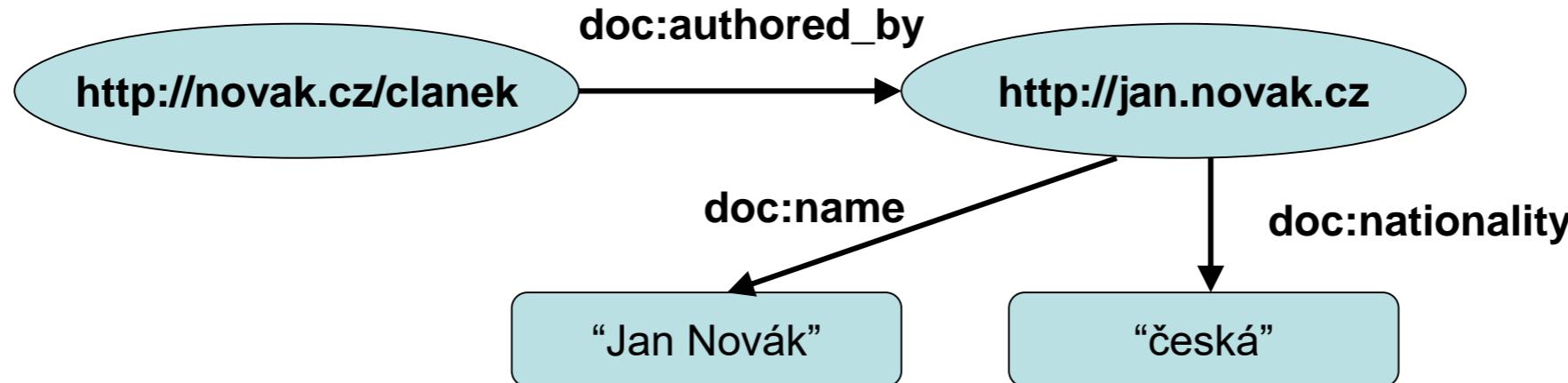
RDF trojice – tvrzení (statement)

- Autorem **dokumentu X** je pan Y
- Subjekt: **dokument X**
- Predikát: **je autorem**
- Objekt: pan Y
- Jednotlivé zdroje reprezentované pomocí **URI** nebo **literálem**

RDF tvrzení (II)

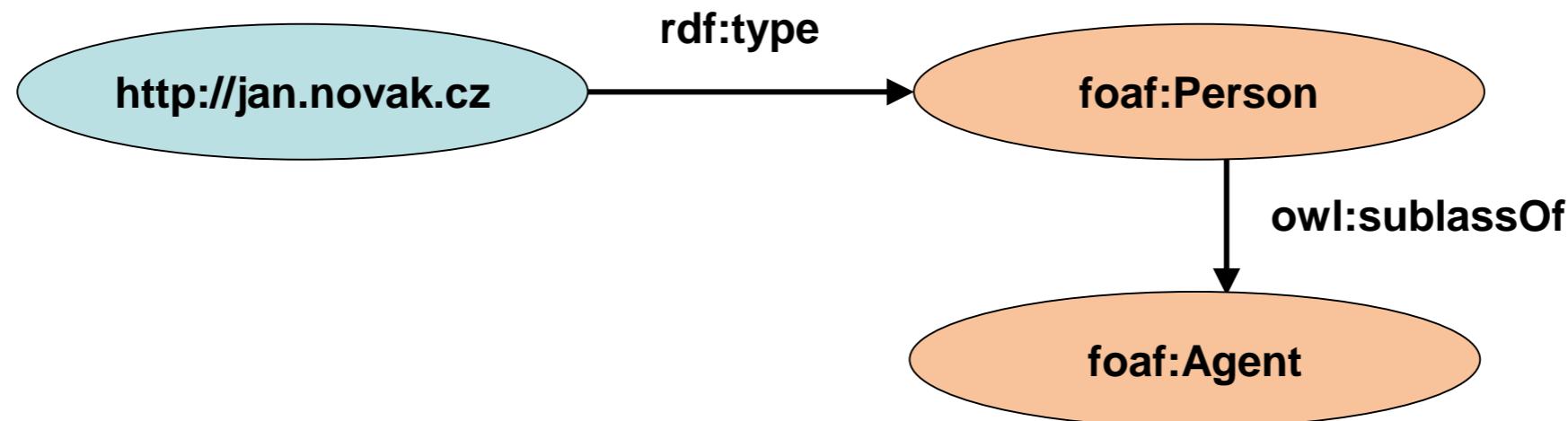


RDF Graf



- RDF graf lze rozložit na trojice subjekt – predikát – objekt
- Subjekt a predikát jsou vždy URI
 - doc: je prefix URI, který se expanduje
 - Např. doc:name => http://my.docs.com/#name
- Objekt je URI nebo literál (různých datových typů)

Schéma – Ontologie



- RDF data lze propojit s metadaty (ontologií, schématem)
 - Pomocí predikátu `rdf:type`
 - Definice metadat opět pomocí RDF
 - Je možné (ale ne nutné) spojit data i metadata do jednoho grafu.

Ukládání a přenos RDF dat

- Uložení do RDF úložiště (např. RDF4J)
 - Rozložení na trojice a uložení do interní struktury
 - Následně možnost dotazování (jazyk SPARQL)
- Serializace do souboru a zpět – několik variant
 - RDF/XML (standard W3C)
 - N-triples (N3)
 - Turtle (podmnožina N3)

XML Serializace

```
<rdf:RDF  
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
    xmlns:doc="http://dokumenty.cz/def#">  
  
<rdf:Description rdf:about="http://novak.cz/clanek">  
..   <doc:authored-by  
..     rdf:resource="http://jan.novak.cz" />  
</rdf:Description>  
  
<rdf:Description rdf:about="http://jan.novak.cz">  
  <doc:name>Jan Novák</doc:name>  
  <doc:nationality>česká</doc:nationality>  
  <rdf:type  
    rdf:resource="http://xmlns.com/foaf/0.1/Person" />  
</rdf:Description>  
</rdf:RDF>
```

Serializace do Turtle

```
@prefix doc: <http://dokumenty.cz/def#> .  
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

```
<http://novak.cz/clanek>  
  doc:authored-by <http://jan.novak.cz> .
```

```
<http://jan.novak.cz>  
  doc:name "Jan Novák" ;  
  doc:nationality "česká" ;  
  a foaf:Person .
```

RDF jako databáze

- Repozitář – úložiště RDF trojic
- Dotazování – jazyk SPARQL
- Lokální úložiště:
 - Virtuoso <http://virtuoso.openlinksw.com/>
 - RDF4J (dříve Sesame) <http://rdf4j.org/>
 - Blazegraph <https://www.blazegraph.com/product/>
- Globální
 - DBpedia <http://dbpedia.org>
 - <http://dbpedia.org/resource/Berlin>
 - <http://dbpedia.org/sparql>

RDF4J

- Java API (embedded) nebo samostatně běžící server přístupný přes HTTP REST API
- Různé druhy úložišť
 - Memory, Native, relační databáze, rozšiřitelné o další
- Strategie vyhodnocování SPARQL dotazů
 - Možnost implementace vlastní strategie
- Podpora kontextu (RDF čtveřice)
- Podpora transakcí

Dotazování – SPARQL

- Výsledkem dotazu je
 - CSV (tabulka) – dotaz SELECT
 - Nebo nový graf – dotaz CONSTRUCT

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX yago: <http://dbpedia.org/class/yago/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbprop: <http://dbpedia.org/property/>
SELECT ?place ?name ?label WHERE {
    ?place rdf:type dbpedia-owl:Country .
    ?place dbprop:commonName ?name .
    ?place rdfs:label ?label .
    OPTIONAL { ?place dbprop:yearEnd ?yearEnd }
    FILTER (!bound(?yearEnd))
}
```

Veřejné databáze

- DBpedia <http://dbpedia.org>
 - <http://dbpedia.org/resource/Berlin>
 - <http://dbpedia.org/sparql>
- Mnoho dalších
 - <http://lod-cloud.net/>

Ontologie

Slovníky pro sémantický web

Pojem ontologie

- Původně obecnější význam (filozofie)
- Nástroj pro sdílení významu pojmů, které se vyskytují v cílové oblasti
- „*Formální, explicitní specifikace sdílené konceptualizace*“
- Definují základní pojmy modelovaného světa a vztahy mezi nimi
- Sdílené a opakovatelně použitelné

Účel ontologií

- Porozumění mezi lidmi (experty)
- Porozumění mezi počítačovými aplikacemi
 - Dodání významu jednotlivým URI v sémantickém webu
- Návrh znalostních aplikací

Typy ontologií

- Terminologické (lexikální)
 - Seznam termínů v dané oblasti
 - Jejich vzájemné vztahy (taxonomie)
 - Např. *WordNet*
- Informační ontologie
 - Databázové systémy – pokročilejší schémata
- Znalostní ontologie
 - Aplikace umělé inteligence
 - Koncepty formálně definované pomocí logických formulí

Typy ontologií (II)

- Generické ontologie
 - Zákonitosti a vztahy mezi obecnými pojmy
 - „Upper ontology“, např. SUMO
- Doménové ontologie
 - Konkrétní oblast (např. podnikové, lékařství, ...)
- Aplikační ontologie
 - Pro konkrétní aplikaci

Prvky ontologií

- Třídy (koncepty)
- Individua (objekty, instance)
- Vlastnosti (role, atributy)
- Meta-sloty (facety)
- Primitivní datové typy
- Axiomy (pravidla)

Koncepty – třídy

- Množiny konkrétních objektů
- Žádné procedurální metody
- Třídy *definované* a *primitivní*
 - Podle definice příslušnosti individua
- Dědičnost tříd (často vícenásobná)

Individua – objekty – instance

- Konkrétní objekty reálného světa
- Individuum nemusí být nutně instancí třídy
- Vzhledem k určení ontologií se často nepoužívají
 - Reprezentují konkrétní data

Relace – atributy – sloty – vlastnosti

- Pojetí vlastnosti je jiné, než u OO modelování
- Vlastnost = relace
 - Samostatně definovaný prvek
 - Obvykle binární relace
- Možná dědičnost relací (má otce, má předka)
 - Nadřazená relace obsahuje všechny prvky podřazené relace
- Funkce – speciální relace
 - Hodnota argumentu n jednoznačně určena předchozími $n-1$ argumenty

Meta-slots, omezení na slotty

- Vlastnosti vlastností
 - Vztah podřízená – nadřízená vlastnost
- Globální omezení
 - Definiční obor a obor hodnot vlastnosti
- Lokální omezení – *facet*
 - Např. kardinalita
 - Hodnota vlastnosti má-otce aplikované na třídu **osoba** je **právě jedna instance třídy osoba.**

Primitivní hodnoty, datové typy

- Argumentem relace může být *primitivní hodnota* (ne objekt)
 - Číslo, řetězec, výčtová hodnota, ...
 - Datatype slot vs. objektový slot
- Můžeme uvažovat dato-typové třídy (datové typy) a dato-typové instance (hodnoty)
- Dato-typové sloty obvykle deklarujeme jako funkční (mají pouze jednu hodnotu)

Axiomy, pravidla

- Logické formule vymezující vztahy tříd
 - Ekvivalence, subsumfce
 - Obvykle součást definice tříd

Ontologické jazyky

RDF Schema, OWL

RDF Schema

- Sémantické rozšíření RDF
 - V podstatě **ontologie**
- Umožňuje definici
 - Tříd
 - Binární relace (definiční obor, obor hodnot)
 - Hierarchie nad třídami i relacemi
- Definice opět pomocí RDF
 - Např. skola:Student rdfs:subClassOf skola:Osoba
- XML syntax nebo jiná
 - Namespace (prefix obvykle **rdfs**)
<http://www.w3.org/2000/01/rdf-schema#>

Třídy

- Třída je přiřazena ke zdroji pomocí `rdf:type`
 - `skola:Osoba rdf:type rdfs:Class`
 - V XML:

```
<rdf:Description
  rdf:about="&skola;Osoba">
    <rdf:type rdf:resource="&rdfs;Class" />
  </rdf:Description>
```
 - Nebo

```
<rdfs:Class
  rdf:about="&skola;Osoba" />
```

Odvozené třídy

- Podtřídy `rdfs:subClassOf`
 - Např.
`skola:Student rdfs:subClassOf skola:Osoba`
 - V XML
 - `<rdfs:Class rdf:about="&skola;Student">`
 - `<rdfs:subClassOf`
 - `rdf:resource="&skola;Osoba" />`
 - `</rdfs:Class>`

Třídy v RDFS

- rdfs:Class – třída (je instancí rdfs:Class)
- rdfs:Resource – třída jakéhokoliv zdroje
 - instance rdfs:Class
- rdfs:Literal
 - instance rdfs:Class
 - Podtřída rdfs:Resource
- rdfs:Datatype
 - instance i podtřída rdfs:Class
 - každá instance Datatype je podtřídou Literal

Třídy v RDFS (II)

- rdfs:XMLLiteral
 - instance rdfs:Datatype
 - podtřída rdfs:Literal
- rdfs:Property
 - instance rdfs:Class

Vlastnosti v RDFS

- Vlastnosti jsou instance rdfs:Property
 - skola:maZapsano rdf:type rdfs:Property
- **rdfs:Range** – typ objektů (obor hodnot)
 - skola:maZapsano rdfs:range skola:Predmet
- **rdfs:Domain** – typ subjektů (def. obor)
 - skola:maZapsano rdfs:domain skola:Student
- rdfs:subPropertyOf
 - Vlastnost je „podvlastností“ jiné vlastnosti

OWL

- Rozšíření RDFS o pokročilé vlastnosti
- Různé verze
 - OWL Lite – zjednodušená, kvůli implementaci
 - OWL DL – omezení RDF(S) pro podporu DL
 - OWL Full – max. kompatibilita s RDF(S)
- Namespace

<http://www.w3.org/2002/07/owl#>

Definice tříd v OWL

- Kombinace s RDFS
- Třídu lze definovat pomocí logických podmínek
 - Identifikátorem třídy (žádné prvky)
 - Výčtem prvků (instancí)
 - Omezením vlastností
 - Sjednocením nebo průnikem dvou a více tříd
 - Doplňkem

Definice ontologie

```
<owl:Ontology rdf:about="">  
  <rdfs:comment>An example OWL ontology</rdfs:comment>  
  <owl:priorVersion>  
    <owl:Ontology  
      rdf:about="http://www.w3.org/TR/2003/WD-owl-guide-  
      20030331/wine"/>  
  </owl:priorVersion>  
  <owl:imports  
    rdf:resource="http://www.w3.org/TR/2003/CR-owl-  
    guide-20030818/food"/>  
  <rdfs:label>Wine Ontology</rdfs:label>  
</owl:Ontology>
```

Definice třídy identifikátorem

```
<owl:Class rdf:about="#Person"/>

<rdf:Description rdf:ID="Person">
    <rdf:type resource="#owl;Class" />
</rdf:Description>
```

Totéž v Turtle

@prefix rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>> .

@prefix owl: <<http://www.w3.org/2002/07/owl#>> .

@prefix foaf:<<http://xmlns.com/foaf/0.1/>>.

foaf:Person rdf:type owl:Class .

foaf:Person a owl:Class .

Definice výčtem prvků

- Jakýkoliv objekt v OWL patří do třídy owl:Thing

```
<owl:Class rdf:id="WineColor">  
  <owl:oneOf rdf:parseType="Collection">  
    <owl:Thing rdf:about="#White"/>  
    <owl:Thing rdf:about="#Rose"/>  
    <owl:Thing rdf:about="#Red"/>  
  </owl:oneOf>  
</owl:Class>
```

Definice omezením vlastností

- Definujeme omezení nějaké vlastnosti
 - hodnoty nebo kardinalita

```
<owl:Restriction>
  <owl:onProperty rdf:resource="vlastnost"/>
  <!-- omezení hodnoty nebo kardinality -->
</owl:Restriction>
```

Druhy omezení

- owl:allValuesFrom
 - Všechny hodnoty dané vlastnosti (pokud nějaké jsou) jsou dané třídy
- owl:someValuesFrom
 - Alespoň jedna hodnota je dané třídy
- owl:hasValue
 - Vlastnost má konkrétní hodnotu
- owl:cardinality (minCardinality, maxCardinality)
 - Kardinalita

Příklad omezení

```
<owl:Class rdf:id="FoodLover">  
  <rdfs:subClassOf rdf:resource="#Person" />  
  
  <rdfs:subClassOf>  
    <owl:Restriction>  
      <owl:onProperty rdf:resource="#loves" />  
      <owl:allValuesFrom rdf:resource="#Food" />  
    </owl:Restriction>  
  </rdfs:subClassOf>  
  
</owl:Class>
```

Příklad omezení – funkční vlastnost

```
<owl:Restriction>  
  <owl:onProperty rdf:resource="#father"/>  
  <owl:cardinality  
    rdf:datatype="&xsd;nonNegativeInteger">  
    1</owl:cardinality>  
</owl:Restriction>
```

Definice průnikem

```
<owl:intersectionOf  
    rdf:parseType="Collection">  
    <owl:Class rdf:about="#class1"/>  
    <owl:Class rdf:about="#class2"/>  
</owl:intersectionOf>
```

Definice sjednocením

```
<owl:unionOf rdf:parseType="Collection">  
  <owl:Class rdf:about="#class1"/>  
  <owl:Class rdf:about="#class2"/>  
</owl:intersectionOf>
```

Definice doplňkem

```
<owl:Class>  
  <owl:complementOf>  
    <owl:Class rdf:about="#Student"/>  
  <owl:complementOf/>  
</owl:Class>
```

Ostatní operátory nad třídami

- owl:equivalentClass
 - Stejná třída (např. z jiné ontologie)
- owl:disjointWith
 - Disjunktní třída

Definice vlastností

- RDFS konstruktory

```
<owl:ObjectProperty rdf:ID="studuje">  
  <rdfs:domain rdf:resource="#Student"/>  
  <rdfs:range rdf:resource="#Obor"/>  
</owl:ObjectProperty>
```

- Vztahy mezi vlastnostmi

- owl:equivalentProperty – stejné hodnoty

- owl:inverseOf – inverzní vlastnost

```
<owl:ObjectProperty rdf:ID="maStudenta">  
  <owl:inverseOf rdf:resource="#studuje"/>  
</owl:ObjectProperty>
```

Definice vlastností (II)

- Omezení kardinality

```
<owl:FunctionalProperty  
    rdf:about="studuje"/>
```
- Symetrická vlastnost
 - owl:SymmetricProperty
- Tranzitivní vlastnost
 - owl:TransitiveProperty

Data-typové vlastnosti

- RDF Literály
- XSD datové typy
 - Namespace <http://www.w3.org/2001/XMLSchema>
 - xsd:string, xsd:normalizedString, xsd:boolean, xsd:decimal, xsd:float, xsd:double, xsd:integer, xsd:nonNegativeInteger xsd:positiveInteger, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, xsd:hexBinary, xsd:base64Binary, xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:anyURI, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName

Individua

- Zápis jménem třídy

```
<Region rdf:id="Morava" />
```

- je ekvivalentní zápisu

```
<owl:Thing rdf:id="Morava" />
```

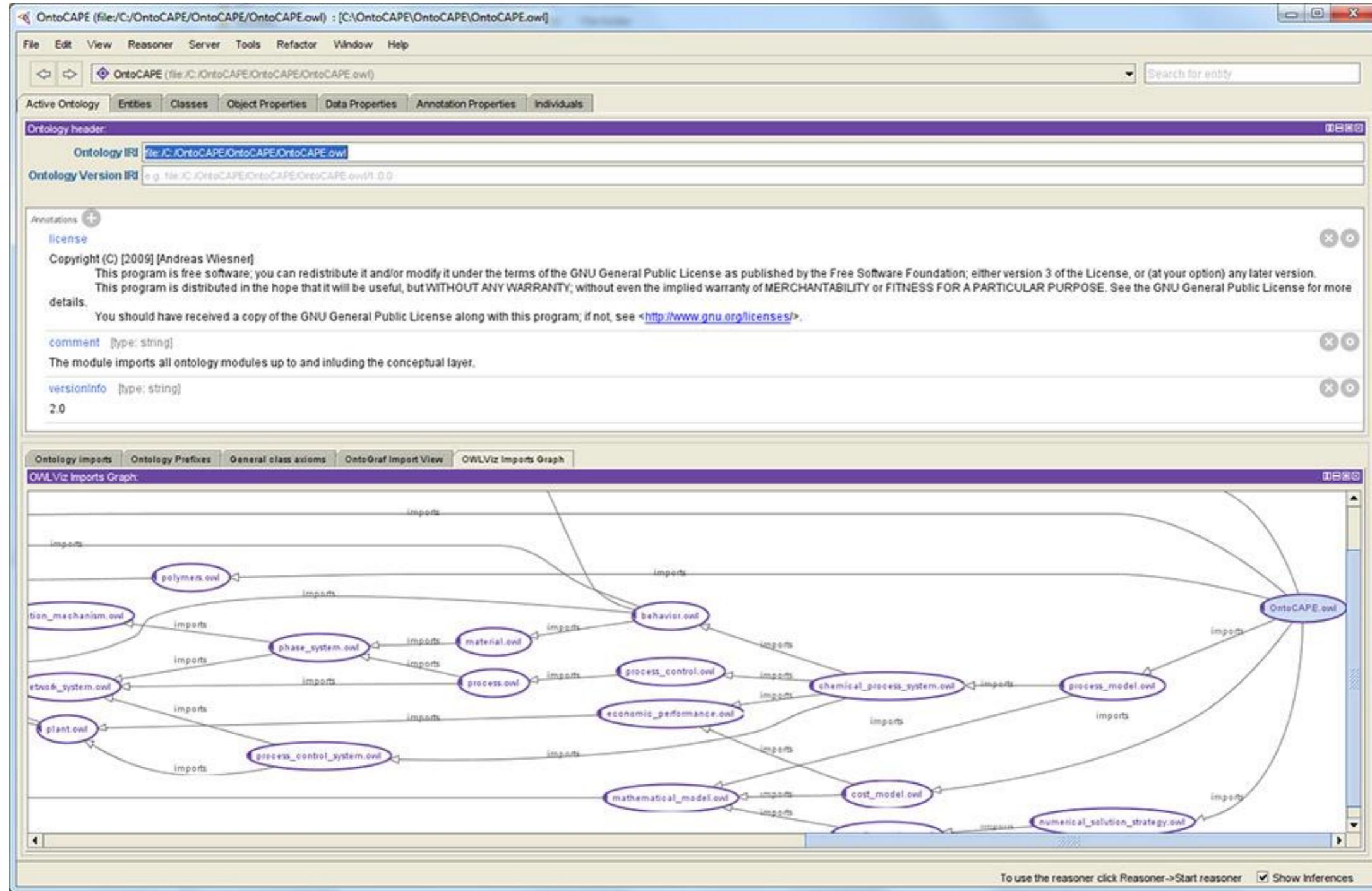
```
<owl:Thing rdf:about="#Morava">
```

```
  <rdf:type rdf:resource="#Region"/>
```

```
</owl:Thing>
```

Editor Protégé

<http://protege.stanford.edu/>



Existující ontologie

- Důraz na maximální využití existujících ontologií
 - Je možno kombinovat koncepty a vlastnosti z různých ontologií
- Přehled
 - [http://protegewiki.stanford.edu/index.php/Protege Ontology Library#OWL ontologies](http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library#OWL_ontologies)

Dublin core

- Metadata dokumentů
- Použití zejména v knihovnictví
- Definuje vlastnosti dokumentů:

```
<rdf:Description      rdf:about="http://www.w3schools.com">  
  <dc:description>W3Schools</dc:description>  
  <dc:publisher>Refsnes Data as</dc:publisher>  
  <dc:date>2008-09-01</dc:date>  
  <dc:type>Web Development</dc:type>  
  <dc:format>text/html</dc:format>  
  <dc:language>en</dc:language>  
</rdf:Description>
```

Friend-of-a-friend (FOAF)

- Ontologie pro popis osob a jejich vzájemných vztahů
<http://www.foaf-project.org/>
- Třídy pro popis osob
 - foaf:Agent, foaf:Person, ...
- Vlastnosti
 - foaf:name, foaf:knows, ...

FOAF příklad

@prefix foaf:<<http://xmlns.com/foaf/0.1/>>.

@prefix dbr:<<http://dbpedia.org/resource>>.

dbr:Luke_Skywalker foaf:knows dbr:Han_Solo .

dbr:Luke_Skywalker foaf:name "Luke Skywalker" .

Další ontologie

- Music ontology
 - <http://musicontology.com/>
- Event ontology
 - <http://motools.sourceforge.net/event/event.html>
- Time ontology
 - <http://www.w3.org/TR/2006/WD-owl-time-20060927/>
- Geo ontology
 - <http://www.w3.org/2003/01/geo/>

Ontologie a RDF databáze

- DBpedia.org
 - Vlastní ontologie + použití existujících
 - <http://dbpedia.org/resource/Berlin>
 - [http://dbpedia.org/page/Novak Djokovic](http://dbpedia.org/page/Novak_Djokovic)
- Např.
 - Vlastnost Birth place
 - <http://dbpedia.org/ontology/birthPlace>

SPARQL

- Jazyk pro dotazování v RDF datech
- Syntax SQL + Turtle
- SELECT ?var WHERE { vzor RDF stromu }
- Např.
 - <http://dbpedia.org/sparql>

RDF a Webové stránky

Web of Documents vs. Web of Data

Jednoduchá anotace dokumentu

```
<html xmlns:html="...">  
<head>  
<rdf:RDF  
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
    xmlns:dc="http://purl.org/dc/elements/1.1/">  
<rdf:Description rdf:about="http://www.moje.cz/"  
    dc:creator="Jan Novak"  
    dc:title="Titulek dokumentu"  
    dc:description="Popis"  
    dc:date="1999-09-10"/>  
</rdf:RDF>  
</head>
```

Anotace částí obsahu

- Současnost – mikroformáty
- Doplnění sémantiky kombinací existujících značek a atributů
- Příklad: hCalendar

```
<ol class="schedule">
<li>2006
  <ol>
    <li class="vevent">
      <strong class="summary">Fashion Expo</strong> in
      <span class="location">Paris, France</span>:
      <abbr class="dtstart" title="2006-10-20">
        Oct 20</abbr> to
      <abbr class="dtend" title="2006-10-23">22</abbr>
    </li>
```

Návrh W3C – RDFa

<p xmlns:dc="http://purl.org/dc/elements/1.1/"
about="http://www.example.com/books/wikinomics">
In his latest book
<cite property="dc:title">Wikinomics</cite>,
Don Tapscott
explains deep changes in technology,
demographics and business.
The book is due to be published in
October
2006.
</p>

Problémy

- Není validní XHTML
 - Jiné návrhy řešení: eRDF
- Mikroformáty jsou rozšířenější
 - Jak překlenout mezeru?

GRDDL [griddle]

- Je definován GRDDL profil stránky
 - Atribut profile elementu <head>
 - Informuje klienta o přítomnosti GRDDL
- Je dodána XSL transformace na RDF
 - Element <link> v hlavičce dokumentu
- Umožňuje transformovat libovolný mikroformát na RDF
 - XHTML -> RDF

Otázky?

Online Analytical Processing & Business Intelligence

DATA, INFORMACE, ZNALOSTI - ZOPAKOVÁNÍ

Data

- hodnota schopná přenosu, uchování, interpretace či zpracování
- z hlediska IT jde o ***hodnoty*** různých ***datových typů***
- data sama o sobě ***nemají sémantiku*** (význam), jsou to věty nějakého formálního jazyka určující ***syntaxi***
- hodnoty dat obvykle udávají ***stav*** nějakého systému

Informace

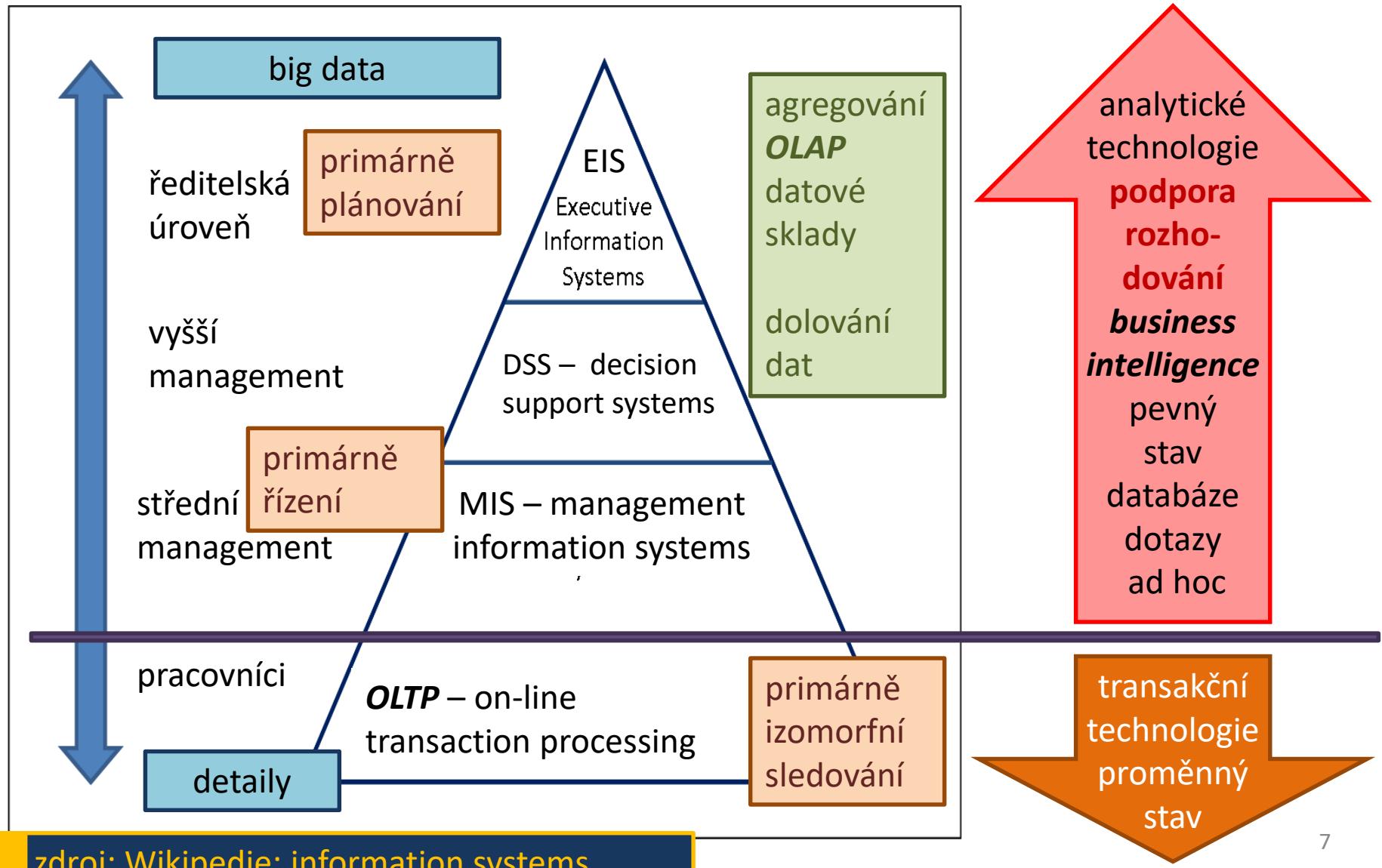
- *informace* jsou interpretovaná *data* lidmi nebo strojově
- po interpretaci mají *sémantiku* (význam)
- transformaci dat na informace provádí *uživatel* nebo *specializovaný systém*
- je nezbytné zajistit shodnou interpretaci dat u všech uživatelů informace
 - vzdělání, školení, zavedení konvencí, *ontologie*

Znalosti

- informace zařazená do souvislostí (často s redukovaným množstvím dat)
- interpretace je však ještě hůře definovatelná, neboť může jít o agregaci informací
- znalosti chápeme často jako ***sekundární odvozené informace***
- některé informační systémy se zabývají pouze ***informacemi (transakční - OLTP)***, některé pracují se ***znalostmi a velkými daty (pro podporu rozhodování a plánování – OLAP)***
- ***znalosti*** jsou často získávány operacemi nad ***velkými daty (agregace, data mining, apod.)***

DATOVÉ SKLADY A ON-LINE ANALYTICAL PROCESSING (OLAP) BUSINESS INTELLIGENCE

Pyramidové schéma



Motivace, příklady

- manager potřebuje vědět, kterým klientům může nabídnout po telefonu úvěrovou kartu, u kterých klientů je vysoké riziko odchodu ke konkurenci
- manager potřebuje znát vývoj tržeb za posledních třicet dní v členění dle regionů a produktů či jak se liší skutečný výkon společnosti od plánovaného

Pojem *business intelligence*

- procesy, technologie a nástroje potřebné k *přetvoření dat a informací do znalostí* pro podporu rozhodování na různých úrovních
- *Vstup*: velké objemy (big data) primárních (produkčních) dat
- *Výstup*: znalosti, které lze využít v procesu rozhodování

Prostředky *business intelligence*

1. *Datové sklady (data warehouses)*

- systém převodu a uložení dat pro analýzu, definovaní formálního modelu dat

2. *OLAP (On-line Analytical Processing)*

- rozhraní pro manipulaci s modelem a zpřístupnění výsledků uživateli a dalším aplikacím

3. *Data Mining (dolování dat)*

viz specializovaný kurz Získávání znalostí z databází

DATOVÉ SKLADY (DATA WAREHOUSES)

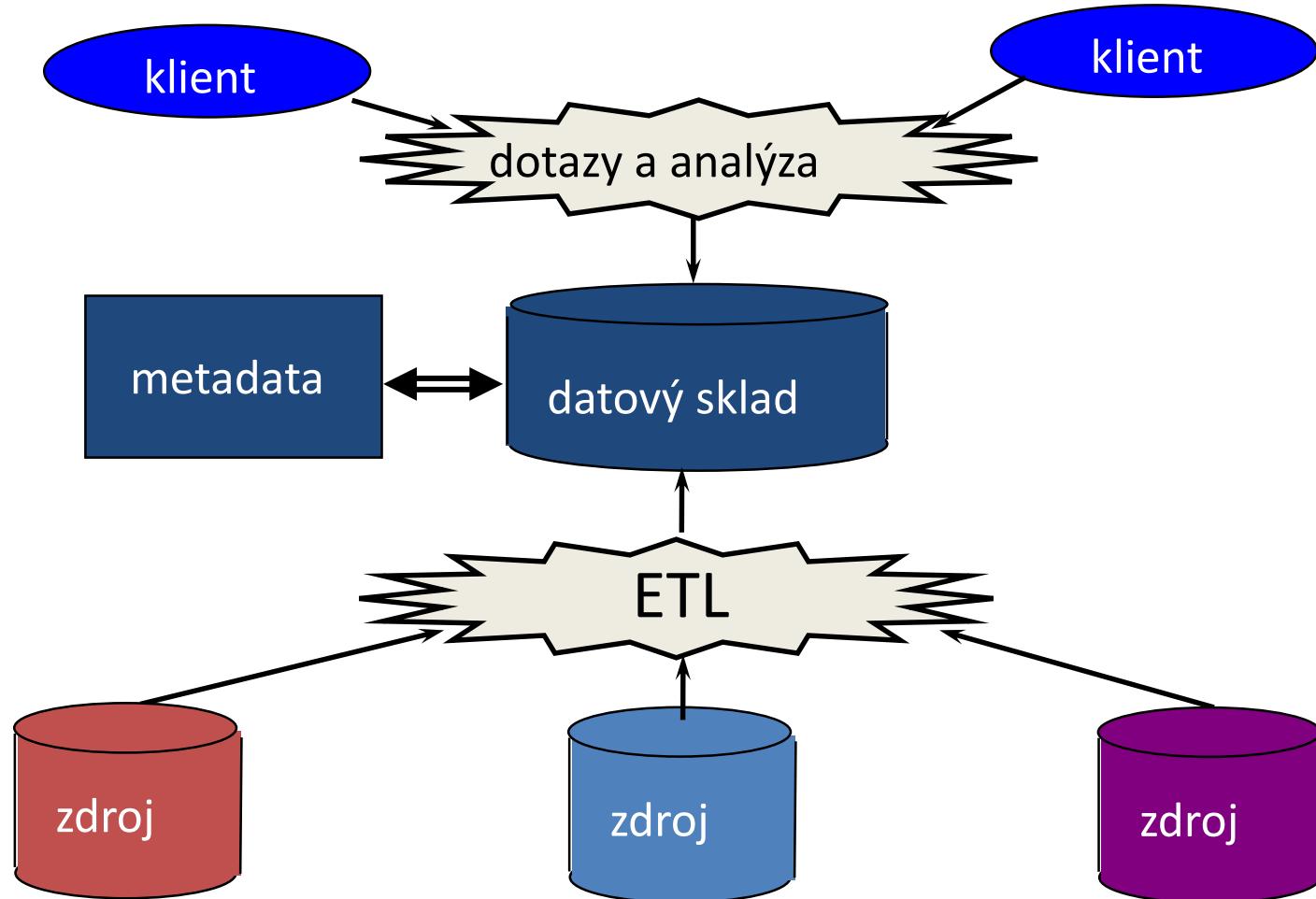
Pojem datového skladu - slovně

- Podnikově strukturovaný depozitář *subjektově orientovaných, integrovaných, časově proměnlivých, historických dat* použitých na získávání **znalostí** a podporu rozhodování
- obsahuje operační i agregovaná data.

Pojem datového skladu

- **Datovým skladem** nazýváme technologii
 - natažení (extrakce a transformation)
 - uložení (loading) a
 - poskytovánídat pro **podporu rozhodování prováděnou analýzou informací a vytvářením znalostí**
- je typicky provozován **oddeleně** od základní **operační databáze** (též **databáze detailů** nebo **produkční databáze**)

Rámcová architektura datového skladu



Nevhodnost produkčních databází jako datových skladů

- slouží především pro ***ukládání*** primárních detailních dat ***izomorfního modelu*** (***modelování reálného systému***)
- výsledkem vesměs pevného počtu dotazů jsou především data explicitně uvedená v databázi bez dalších agregačních úprav
- výhodné především pro ***jednoduché transakce*** (vkládání, mazání...) - OLTP, naopak ***nevhodné pro složitější analýzu velkých dat***

Nevhodnost produkčních databází jako datových skladů

- obvykle ***decentralizovanost*** systémů OLTP
 - data potřebná pro analýzu jsou většinou ***uložena v různých heterogenních DB na různých serverech***, není většinou k dispozici integrovaný zdroj údajů a je složité tato data integrovat a homogenizovat
- ***nehomogenní struktura údajů*** – různé názvy vlastností, datové typy...
- nevhodnost technologie pro analýzy pro standardní výpočtové prostředky a modely

Nevhodnost produkčních databází jako datových skladů

- degradace výpočetního výkonu databázového stroje – neustále *se opakujícími stejnými aggregačními výpočty*
- případně nejsou uchovávány historické údaje – uchovávají se zpravidla *pouze aktuální data*

Vhodný model = vícerozměrnost

- aby bylo možno provádět komplexní analýzu a vizualizaci, jsou data v datovém skladu typicky modelována ***multidimensionálně***.
- jiný datový model, nežli relační
- klade důraz na ***strukturu pro budoucí dotazy ad hoc vyžadující agregační a statistické výpočty***

Multidimenzionální databáze

- slouží jako platforma pro získání ***agregovaných údajů***
- výpočty, které by se případně opakovaně prováděly, ***mohou být spočteny předem a uloženy (materializovány)*** z důvodu rychlého přístupu k agregovaným datům – zapojení učících algoritmů
- redundance zde není tak podstatným problémem (data jsou ***read-only, nevzniká problém udržování konzistence a vícenásobného přístupu***)

DEFINICE MULTIDIMENZIONÁLNÍHO MODELU

XML for Analysis

XMLA

XMLA úvod

- ***XML for Analysis*** (zkráceně ***XMLA***)
- Průmyslový standard pro přístup k datům v analytických systémech jako je OLAP a data mining.
- je založen na průmyslových standardech XML, SOAP and HTTP.
- je udržováno společností ***XMLA Council*** se členy Microsoft, Hyperion a SAS , kteří jsou oficiálními zakladateli.

XMLA historie

- XMLA bylo navrženo společností Microsoft jako následník systému OLE DB for OLAP v dubnu 2000.
- V lednu 2001 byl přibrán XMLA navržený společností Hyperion.
- Verze 1.0 byla zveřejněna v dubnu 2001
- V září 2001 byl ustaven XMLA Council.
- V dubnu 2002 byla přibrána SAS jako zakladající člen XMLA Council.
- Během času přebralo tento standard více než 25 společností.

XMLA použití

- XMLA je vhodná syntaxe pro zápis součástí multidimenzionálního modelu
- Budeme používat pro příklady jak definice multidimenzionální kostky, tak pro operace nad ní

XMLA API

- XMLA sestává pouze ze 2 metod SOAP methods.
- *Discover*
- *Execute*

XMLA Discover

- Navržena pro modelování všech zjišťovacích metod z OLE DB
- Zahrnuje schémata jako *rowset*, *properties*, *keywords*, ale taky *kostku* atd.
- Umožňuje definovat, jak a co bude zkoumáno, tak možná omezení nebo vlastnosti

XMLA Execute

- Má 2 parametry:
- *Command* – Příkaz, který má být proveden.
Může být v jazycích **MDX**, DMX nebo SQL.
- *Properties* - seznam v jazyce XML sestávající z vlastností jako je Timeout, Catalog name atd.
- Výsledkem příkazu *Execute* může být *Multidimensional Dataset* nebo *Tabular Rowset*.

Implementace XMLA

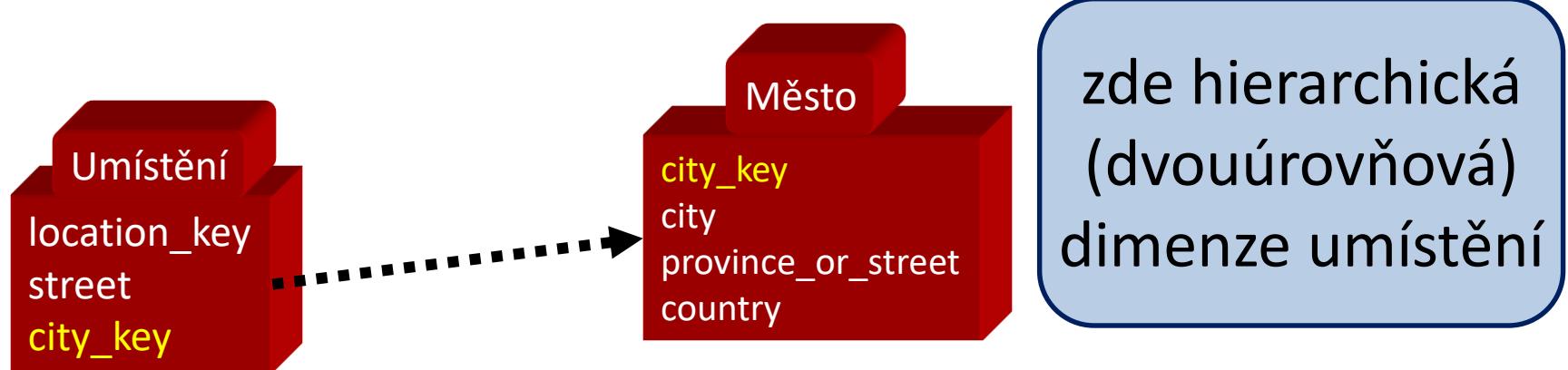
- dostupný systém *icCube*
- budeme používat pro příklady zejména v multidimenzionálním dotazovacím jazyce MDX.

MULTIDIMENZIONÁLNÍ MODEL

DIMENTE

Dimenze

- *Dimenze* je **uspořádatelná** množina hodnot **diskrétního** základního typu (integer, výčet, čas) nebo množina jejich struktur **hierarchicky organizovaných**



Dimenze příklady

jednorozměrné

čas

leden

únor

březen

duben

...

prosinec

katalog

kabáty

kalhoty

bundy

čepice

...

trička

místo

Brno

Praha

Ostrava

Plzeň

...

Olomouc

hierarchické

čas

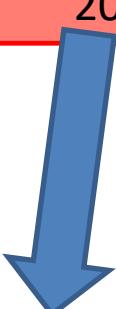
2000

2001

2002

...

2015



leden

únor

...

prosinec

leden

únor

...

prosinec

leden

únor

...

prosinec

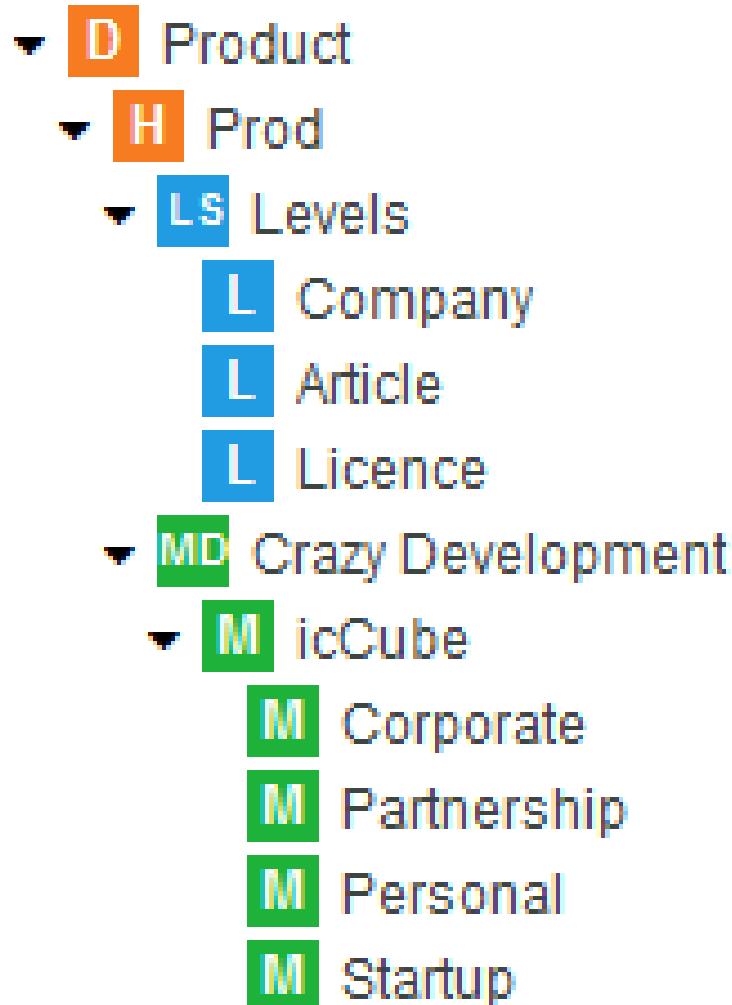
Příklad definice dimenze Geography v icCube (hierarchie Geo a Economy)



Příklad – Geography v Excelu

[Geo]. [Continent]	[Geo].[Country]. [Name]	[Geo]. [Country]. [Key]	[Geo]. [City]	[Economy]. [Partnership]	[Economy]. [Country]
America	Canada	CA	Quebec	NAFTA	Canada
America	Canada	CA	Toronto	NAFTA	Canada
America	United States	US	Los Angeles	NAFTA	United States
America	United States	US	New York	NAFTA	United States
America	United States	US	San Francisco	NAFTA	United States
America	Mexico	ME	Mexico	NAFTA	Mexico
America	Venezuela	VE	Caracas	NAFTA	Venezuela
Europe	France	FR	Paris	EU	France
Europe	Spain	ES	Barcelona	EU	Spain
Europe	Spain	ES	Madrid	EU	Spain
Europe	Spain	ES	Valencia	EU	Spain
Europe	Switzerland	CH	Geneva	None	Switzerland
Europe	Switzerland	CH	Lausanne	None	Switzerland
Europe	Switzerland	CH	Zurich	None	Switzerland

Příklad definice dimenze Product hierarchie Prod pro icCube



Příklad Product v Excelu

[Prod].[Company]	[Prod]. [Article]	[Prod]. [Licence]
Crazy Development	icCube	Corporate
Crazy Development	icCube	Partnership
Crazy Development	icCube	Personal
Crazy Development	icCube	Startup

Příklad definice dimenze Time hierarchie Calendar pro icCube



Příklad Time v Excelu

[Calendar].[Year]	[Calendar].[Quarter]	[Calendar].[Month]	[Calendar].[Day]	[Calendar].[Day].[Key]
2010Q1 2010	January 2010	January 1,2010	January 1, 2010	January 1, 2010
2010Q1 2010	January 2010	January 2,2010	January 2, 2010	January 2, 2010
2010Q1 2010	January 2010	January 3,2010	January 3, 2010	January 3, 2010
2010Q1 2010	January 2010	January 4,2010	January 4, 2010	January 4, 2010
2010Q1 2010	January 2010	January 5,2010	January 5, 2010	January 5, 2010
2010Q1 2010	January 2010	January 6,2010	January 6, 2010	January 6, 2010
2010Q1 2010	January 2010	January 7,2010	January 7, 2010	January 7, 2010
2010Q1 2010	January 2010	January 8,2010	January 8, 2010	January 8, 2010
2010Q1 2010	January 2010	January 9,2010	January 9, 2010	January 9, 2010
2010Q1 2010	January 2010	January 10,2010	January 10, 2010	January 10, 2010
2010Q1 2010	January 2010	January 11,2010	January 11, 2010	January 11, 2010
2010Q1 2010	January 2010	January 12,2010	January 12, 2010	January 12, 2010
2010Q1 2010	January 2010	January 13,2010	January 13, 2010	January 13, 2010
2010Q1 2010	January 2010	January 14,2010	January 14, 2010	January 14, 2010
2010Q1 2010	January 2010	January 15,2010	January 15, 2010	January 15, 2010
2010Q1 2010	January 2010	January 16,2010	January 16, 2010	January 16, 2010

není to celé atd.

Definice multidimenzionální kostky

- Nechť existuje *uspořádaná množina n dimenzí* $\{D_1, D_2, D_3, \dots, D_n\}$
- tj. *existuje relace uspořádání R (<)* nad množinou dimenzí

čas

katalog

místo

Definice multidimenzionální kostky

- počet různých prvků relace R je $n!$ (počet **permutací** nad n , počet různých uspořádání)
- je to také **počet stěn n -dimenzionální kostky**. Proto má **3D kostka na vrhcáby** 6, tj. $3!$ stěn a dvoudimenzionální čtverec 2, tj. $2!$ stěn.

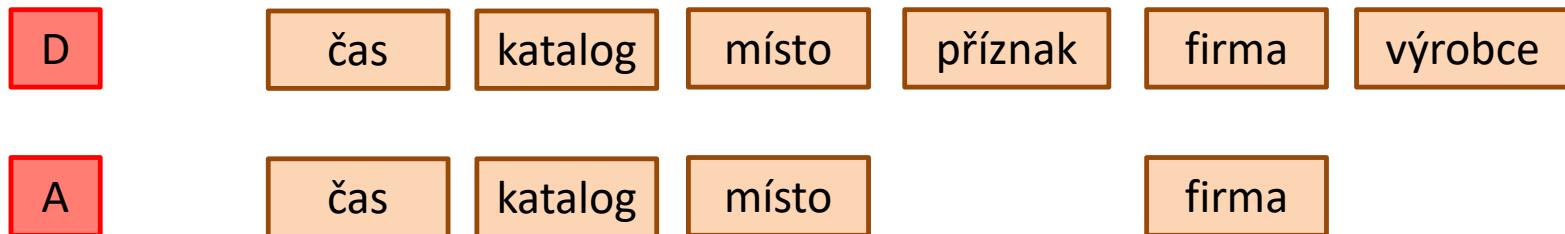


Definice multidimezionální kostky

- Nechť existuje ***uspořádaná podmnožina aktivních dimenzí***
 $\{A_1, A_2, A_3, \dots, A_m\}$, kde $m \leq n$,
 $A_1 = D_{i1}, A_2 = D_{i2}, A_3 = D_{i3}, \dots, A_m = D_{im}$ a
 $D_{i1} < D_{i2} < D_{i3} < \dots < D_{im}$

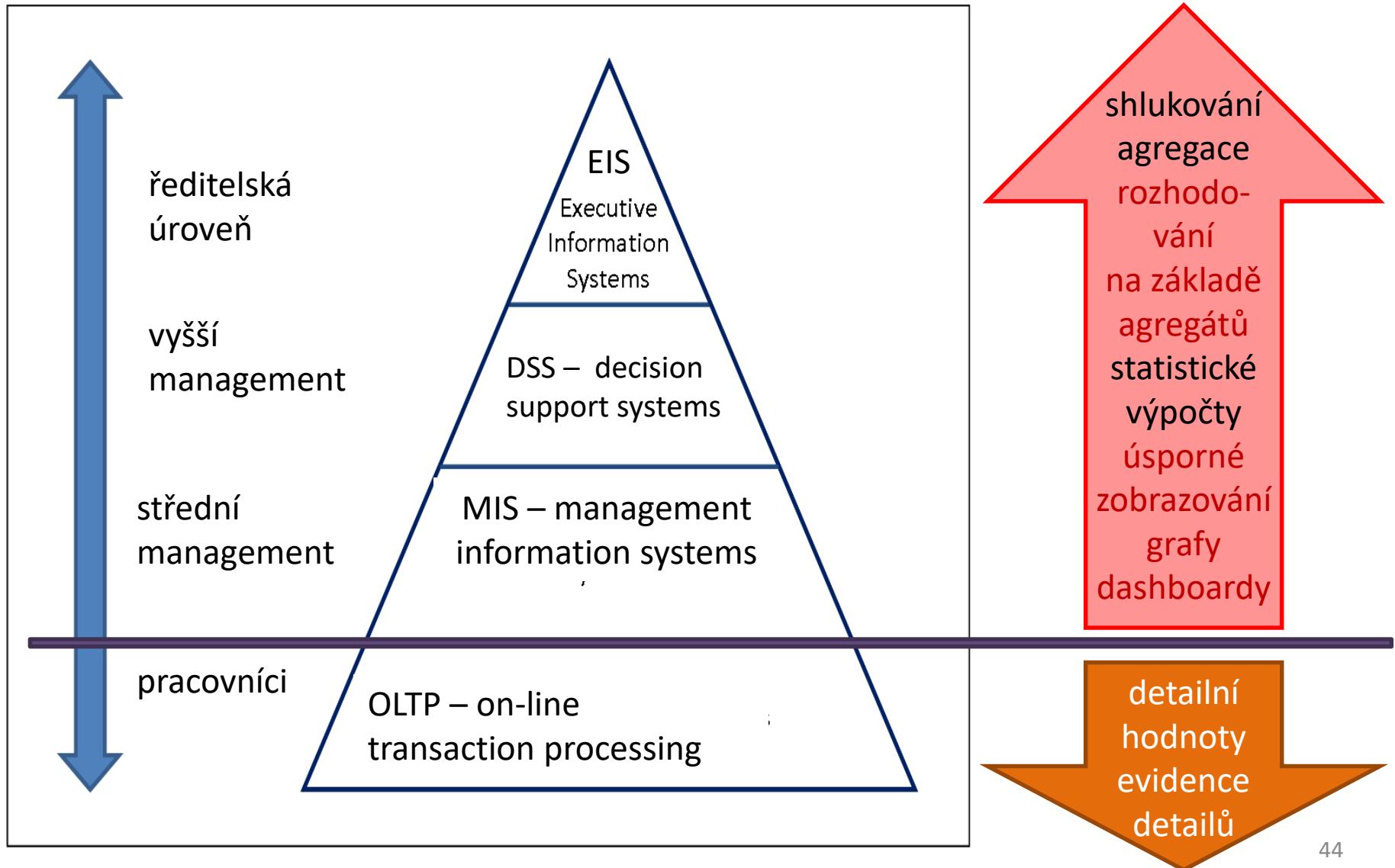
Definice multidimezionální kostky

- Aktivní dimenze budou uspořádány stejně jako v původní množině všech dimenzí



ZOPAKOVÁNÍ POJMŮ AGREGACE A AGREGAČNÍCH FUNKCÍ

Shlukování, agregace



Agregace

- **agregace** (z lat. *ad-*, při- a *grex, gregis*, stádo) znamená připojení, přičlenění, případně také **shrnutí či shlukování**.
- **agregační funkce** – shlukující **několik (množinu obvykle číselných) údajů** do **jediné hodnoty** jistou **agregační funkcí**
- **agregát** – výsledek aggregace, spojení několika strojů, hodnot a pod.

Agregační funkce

- **Agregační funkce** jsou funkce, které shlukují (různým výpočtem) dohromady množiny hodnot do jediné hodnoty
- Obecně známé jsou:
- **počet**
- **součet**
- aritmetický **průměr** (nadále je aritmetický průměr a průměr synonymum)
- **maximum**
- **minimum**
- Méně známé jsou:
- **medián**
- **modus**

Modus

- Modus je hodnota, která se vyskytuje v množině hodnot nejčastěji.

Medián

- prostřední hodnota oddělující horní a dolní polovinu uspořádané datové množiny

Porovnání průměru, mediánu a modu

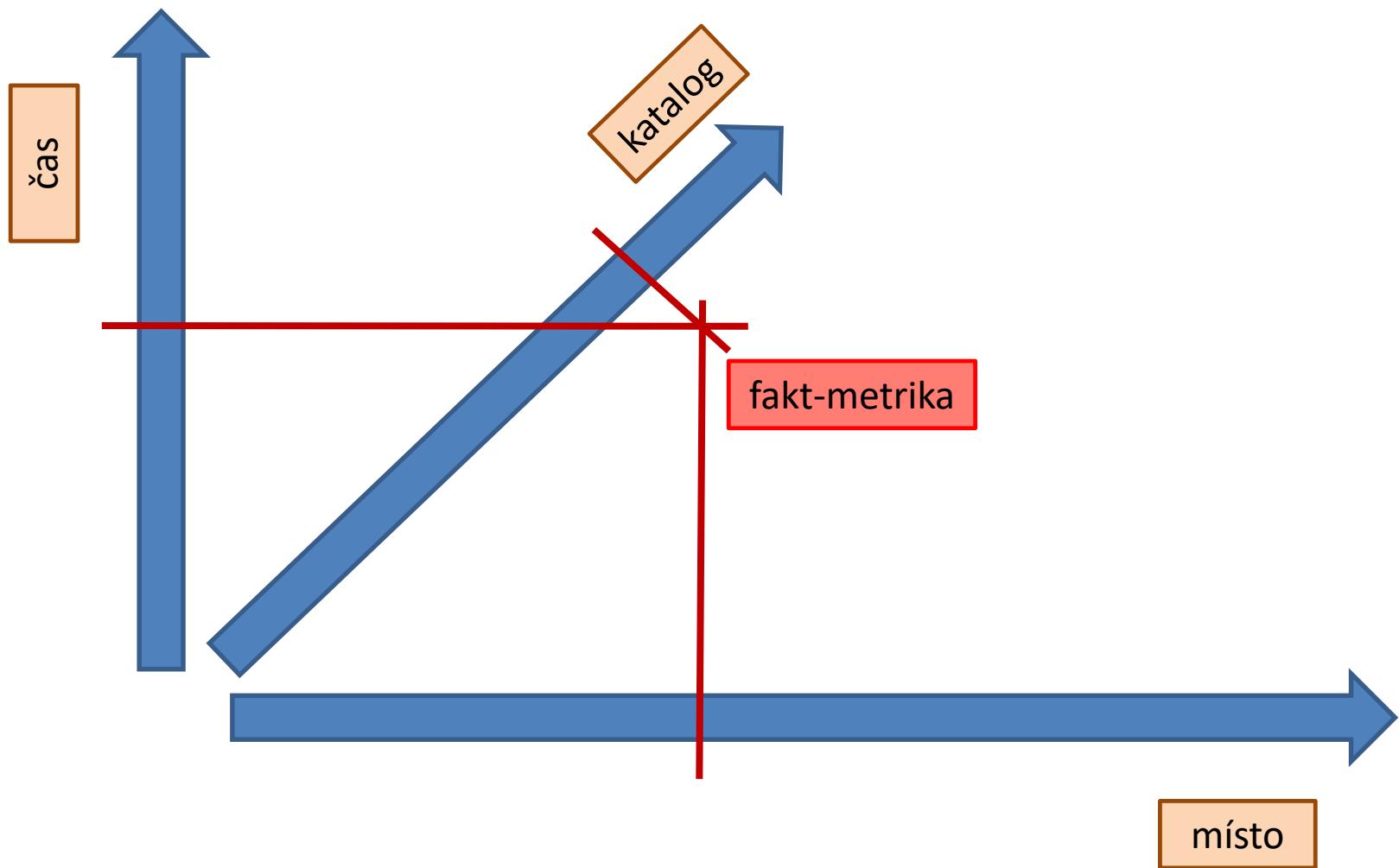
{ 1, 2, 2, 3, 4, 7, 9 }

funkce	popis	výpočet	výsledek
průměr	součet hodnot dělený počtem	$(1+2+2+3+4+7+9) / 7$	4
medián	prostřední hodnota oddělující horní a dolní polovinu uspořádané datové množiny	1, 2, 2, 3, 4, 7, 9	3
modus	nejfrekventovanější hodnota v datové množině	1, 2, 2, 3, 4, 7, 9	2

Definice multidimezionální kostky

- **Multidimenzionální kostka** je funkce $g_m (A_1 \times A_2 \times A_3 \times \dots \times A_m) = F$, kde $f \in F$ nazýváme **fakt (míra, measure)** a $A_1 \times A_2 \times A_3 \times \dots \times A_m$ je **kartézský součin** dimenzí.
- **Fakt (míra, measure)** je libovolná **agregovatelná** hodnota (lze ji sčítat, průměrovat, řetězit apod.), tedy existují kostky počtů, aritmetických průměrů, součtů apod.

Definice multidimenzionální kostky



Příklad definice míry Amount

→  **Measures**

 **Amount**

Příklad míry Amount v Excelu

[Calendar].[Day]. [Key]	[Geo]. [City]	[Prod]. [Licence]	[Measures]. [Amount]
May 12, 2010	Madrid	Personal	1
May 13, 2010	Barcelona	Personal	2
May 14, 2010	Paris	Personal	4
May 15, 2010	Lausanne	Personal	8
May 16, 2010	Lausanne	Corporate	16
May 17, 2010	Lausanne	Partnership	32
May 18, 2010	Zurich	Partnership	64
May 19, 2010	Geneva	Corporate	128
May 20, 2010	New York	Corporate	256
May 21, 2010	New York	Corporate	512

ZOPAKOVÁNÍ - SVAZ

Částečně uspořádaná množina

- **Částečně uspořádaná množina** se skládá z množiny S a relace částečného uspořádání \leq . Obvykle ji, ale současně někdy také její graf, označujeme $[S; \leq]$.

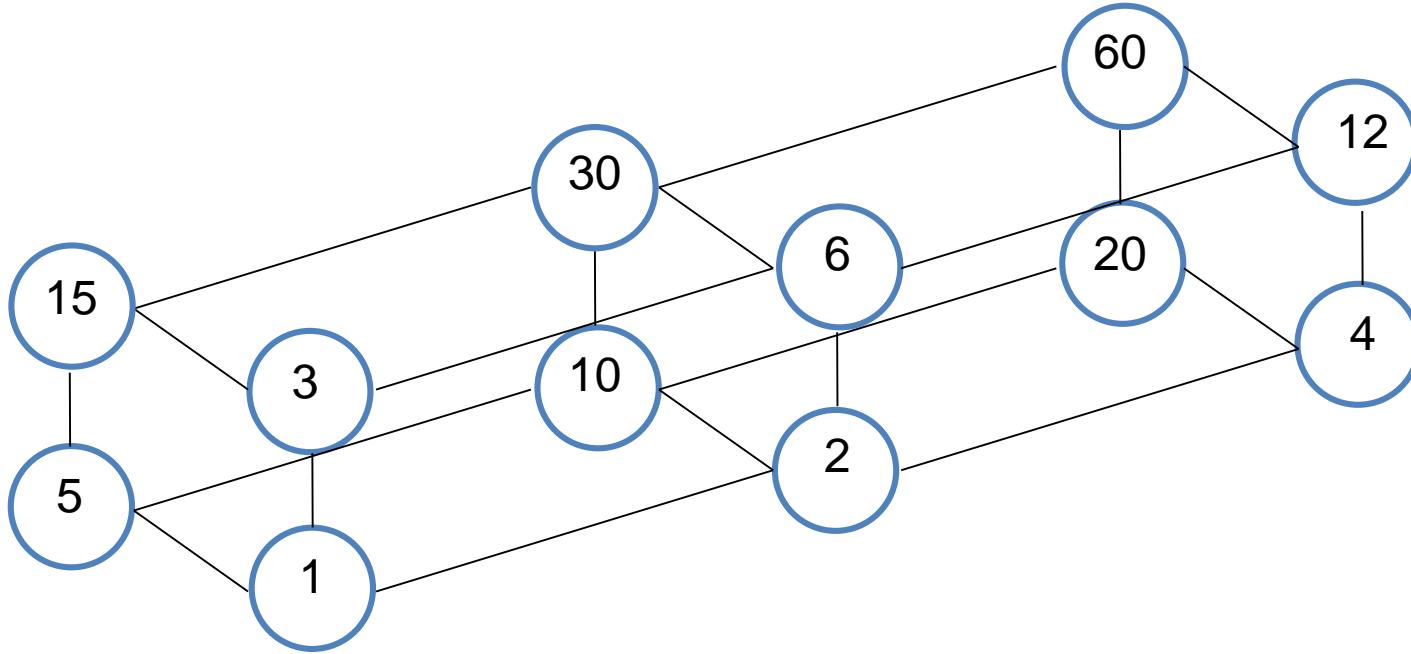
Přímý předchůdce, přímý následník

- Necht' $[S; \leq]$ je částečně uspořádaná množina. Pak prvek a nazveme **bezprostředním** (nebo také **přímým**) **předchůdcem** prvku b , pokud $a \leq b$ a neexistuje takový prvek c , pro který by platilo $a \leq c \leq b$. Relaci bezprostředního (přímého) předchůdce označujeme symbolem $<$. Inverzní relaci k nazýváme **bezprostředním** (**přímým**) **následníkem**.

Přímý předchůdce, přímý následník

- Relace bezprostředního předchůdce není *ani reflexivní, ani symetrická, ani tranzitivní*.
- Relace přímého předchůdce je také podmnožinou původní relace částečného uspořádání.

Hasseův diagram



Množina $A = \{ 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60 \}$
všech dělitelů čísla 60 je částečně uspořádaná podle
dělitelnosti

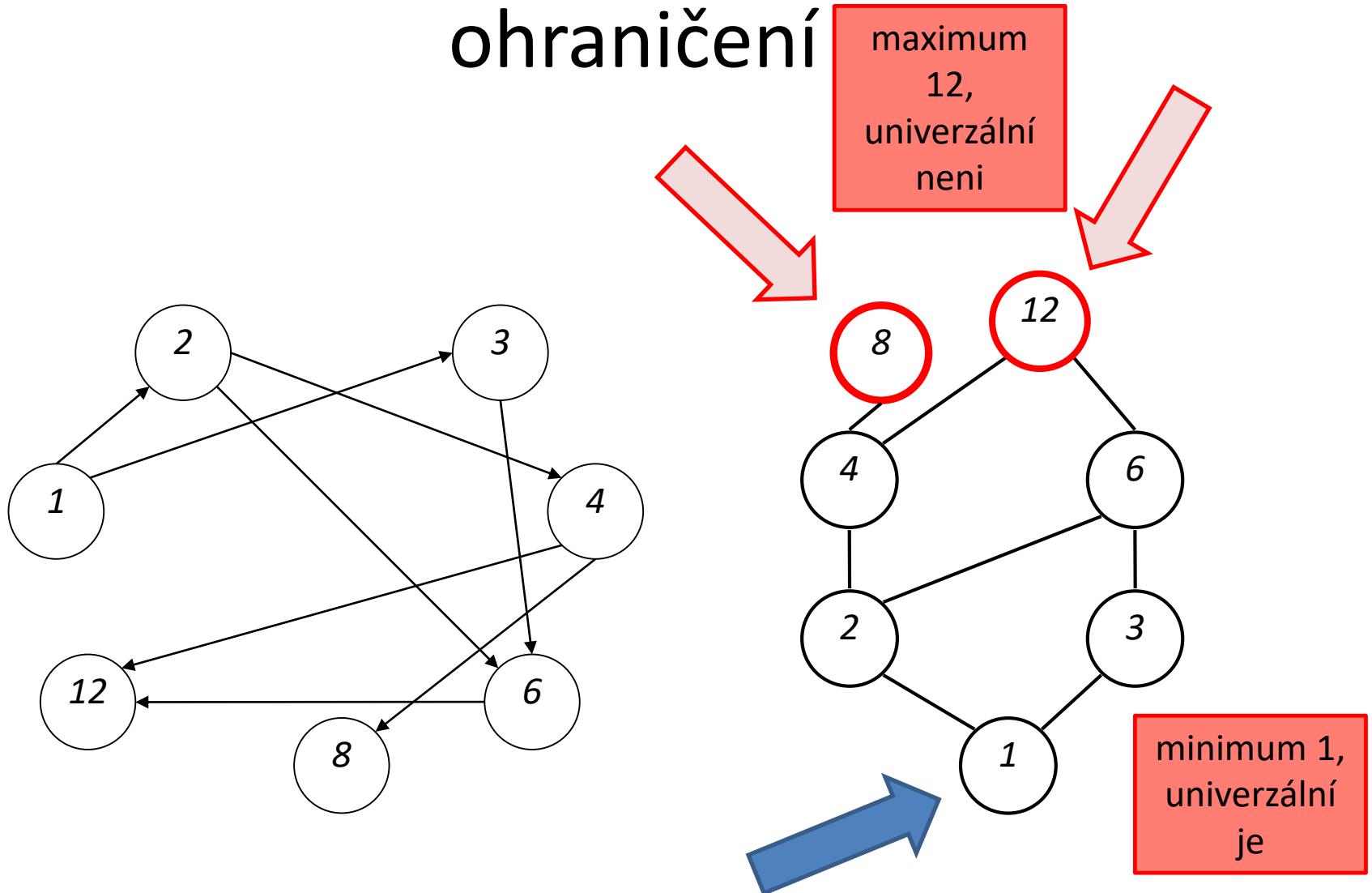
Maximální a minimální

- Prvek $\max \in [S; \leq]$ se nazývá **největší (maximální)**, pokud neexistuje prvek $a \in [S; \leq]$ různý od \max , pro který by platilo platí $\max \leq a$. Podobně se pro inverzní relaci \geq definuje **nejmenší (minimální)** prvek \min .

Univerzální horní a dolní ohraničení

- Prvek $l \in [S; \leq]$ se nazývá ***univerzální horní ohraničení***, pokud pro všechny $a \in [S; \leq]$ platí $a \leq l$. Podobně se pro inverzní relaci \geq definuje ***univerzální dolní ohraničení*** prvek O .

Příklad univerzálního dolního ohrazení



Univerzální horní a dolní ohraničení

- Konečná částečně uspořádaná množina má vždy maximální a minimální prvek. Nemusí ovšem mít horní, resp. dolní univerzální ohraničení.
- částečně uspořádaná množina z předchozího obrázku Hasseova diagramu má univerzální dolní ohraničení (tj. prvek 1), ale nemá horní univerzální ohraničení (ale má maximální prvek 12).

Spojení

- Nechť a, b jsou prvky částečně uspořádané množiny $a, b \in [S; \leq]$. ***Spojením*** prvků nazýváme prvek c , pro který platí $a \leq c$ a $b \leq c$ a neexistuje prvek $x \in (S - \{c\})$ takový, že $a \leq x \leq c$ a také $b \leq x \leq c$. Pokud dva prvky mají ***jediné spojení***, hovoříme o ***nejmenším horním ohrazení*** a označujeme jej $a \vee b$.

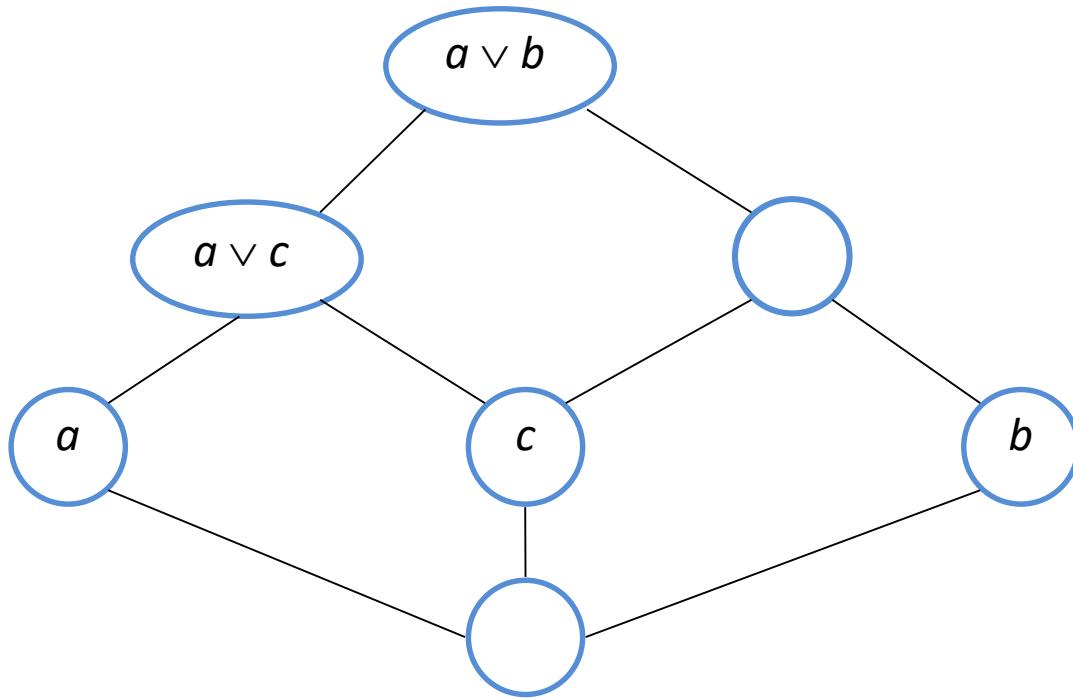
Průsek

- Nechť a, b jsou prvky částečně uspořádané množiny $a, b \in [S; \leq]$. **Průsekem** prvků nazýváme prvek d , pro který platí $d \leq a$ a $d \leq b$ a neexistuje prvek $x \in (S - \{d\})$ takový, že $d \leq x \leq a$ a také $d \leq x \leq b$. Pokud dva prvky mají **jediný průsek**, hovoříme o **největším dolním ohrazení** a označujeme jej $a \wedge b$.

Spojení a průsek

- Spojení e dvou libovolných prvků a a b z konečné uspořádané množiny $[S; \leq]$ zjistím poměrně snadno z Hasseova diagramu.
- Skutečnost, že $a \leq e$ znamená, že existuje řetězec hran vedoucích vzhůru z a do e .
- To stejné platí i pro $b \leq e$. e je potom společným prvkem ležícím na obou řetězcích takový, že žádný jiný nemá takovou vlastnost.
- Z toho také pochází i název spojení. Spojení dvou prvků a a b je takový vrchol, který poprvé spojí řetězce hran vedoucí vzhůru vycházející z prvků a a b .

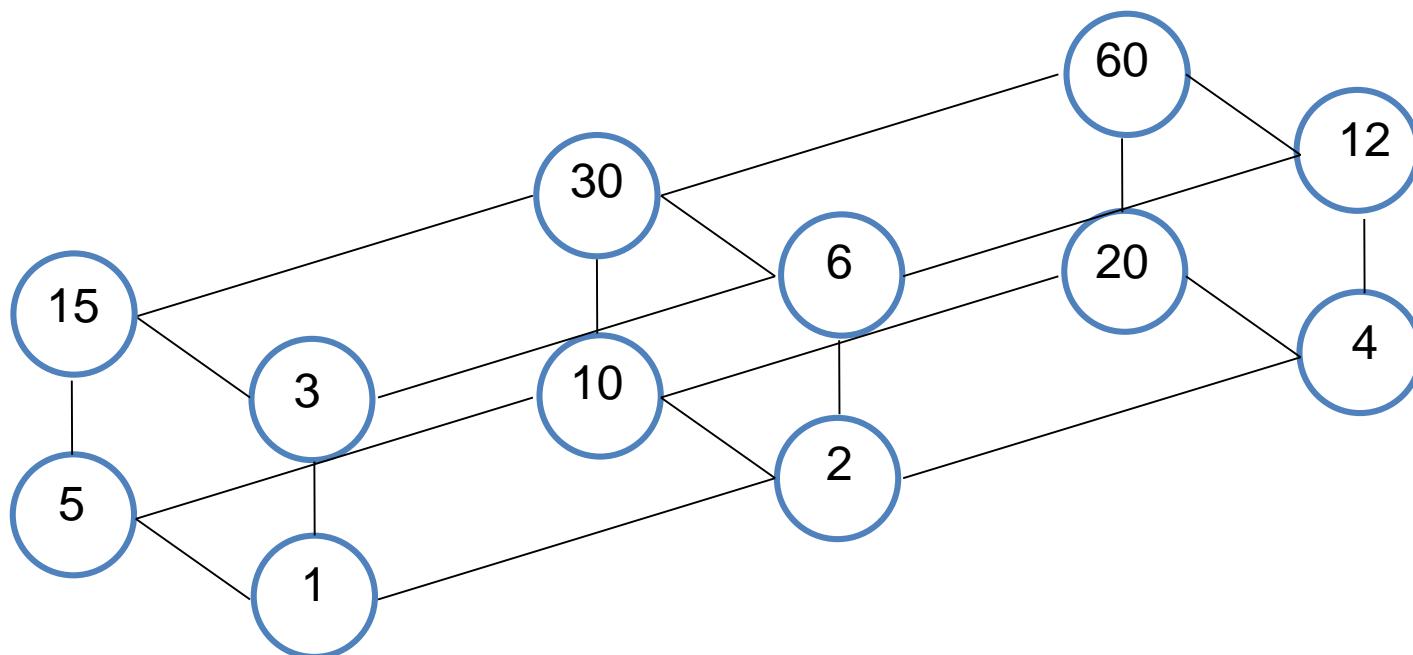
Spojení pomocí Hasseova diagramu



Svaz

- **Svaz** (anglicky *lattice*, což je anglicky mřížka a souvisí to s podobou Hasseova diagramu pro svaz) je
- **částečně uspořádaná množina** $[S; \leq]$
- každé dva prvky mají *jediné spojení a jediný průsek*.
- svaz označujeme $[S, \vee, \wedge]$.

Příklad svazu dělitelnosti



Příklad svazu

- každá dvojice prvků má jediné spojení a jediný průsek.
- množina $A = \{ 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60 \}$ společně se spojením ***nejmenší společný násobek*** a průsekem ***největší společný dělitel*** je ***svazem***.

**ZPĚT K MULTIDIMENZIONÁLNÍMU
MODELU**

Definice multidimezionální kostky

- ***Podkostky (kuboidy)*** odvozené od jednoho uspořádání dimenzí tvoří ***poset*** aneb ***částečně uspořádanou množinu.***

Částečně uspořádaná množina podkostek

- Poset nad podkostkami kostky \mathbf{g} je definován následovně:

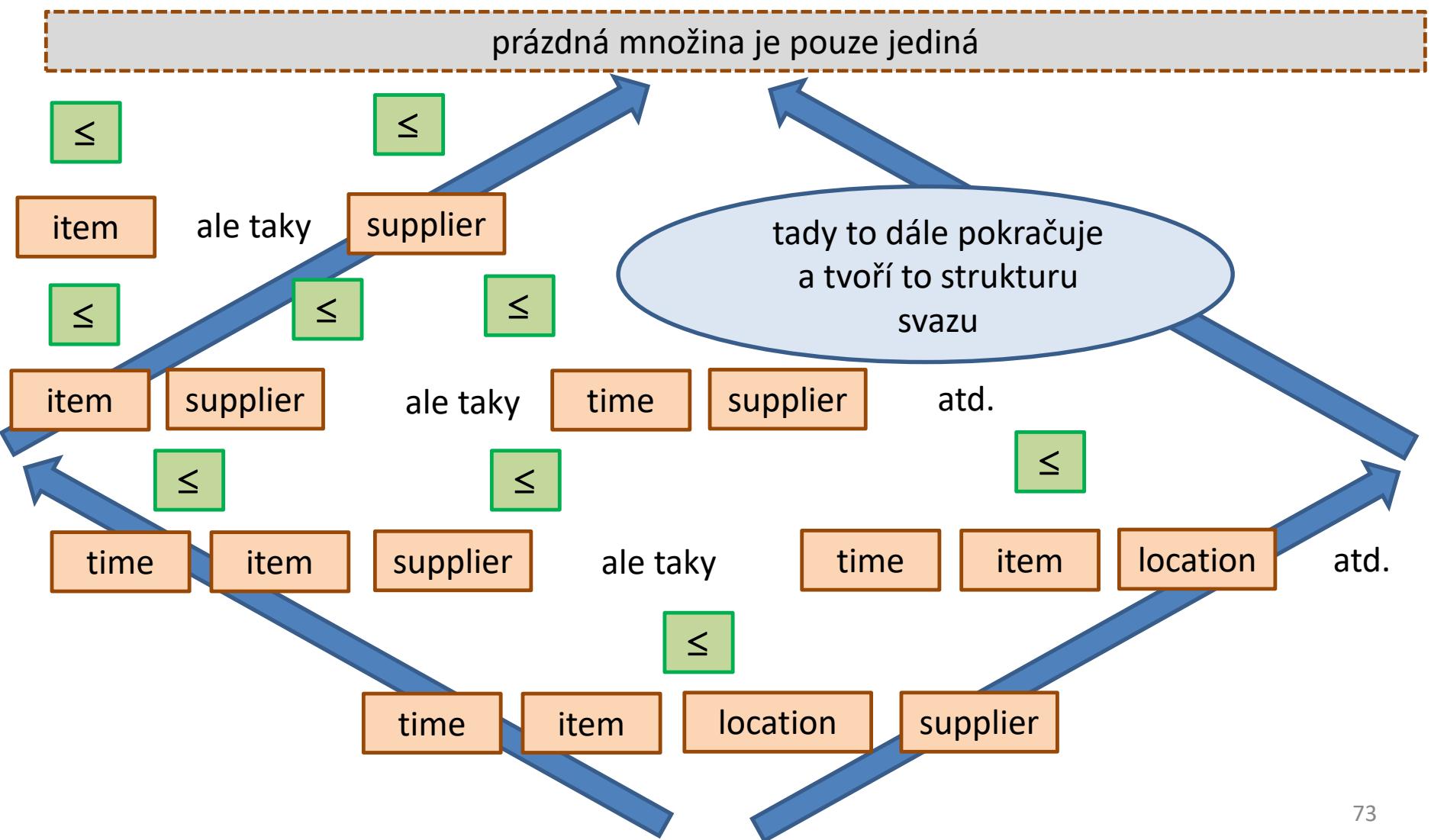
$$\mathbf{g}_m (A_1 \times A_2 \times A_3 \times \dots \times A_i \times \dots \times A_m)$$

\leq

$$\mathbf{g}_{m-1} (A_1 \times A_2 \times A_3 \times \dots \times A_{m-1})$$

- Podkostka je přímo větší, pokud má o jednu méně dimenzí a ostatní jsou uspořádány stejně v obou podkostkách

Příklad podkostek a relace \leq



Částečně uspořádaná množina podkostek

- ***Multidimensionální podkostky*** pro jedno uspořádání dimenzí a jednu agregační funkci tvoří ***svaz***

Svaz podkostek

- Co je to **svaz** viz také

zdroj: studijní opora: Matematické základy modelování informačních systémů

- **Univerzální horní ohrazení** - vrcholová podkostka **all**
- **Univerzální dolní ohrazení** - základní kostka

$$g_n(D_1 \times D_2 \times D_3 \times \dots \times D_n)$$

Svaz podkostek kostky g_n

- **Průsek** \wedge

$$g_m(A_1 \times A_2 \times A_3 \times \dots \times A_i \times \dots \times A_m) \wedge$$

$$g_m(A_1 \times A_2 \times A_3 \times \dots \times A_j \times \dots \times A_m) =$$

$$g_{m+1}(A_1 \times A_2 \times A_3 \times \dots \times A_i \times \dots \times A_j \times \dots \times A_{m+1})$$

- **Spojení** \vee

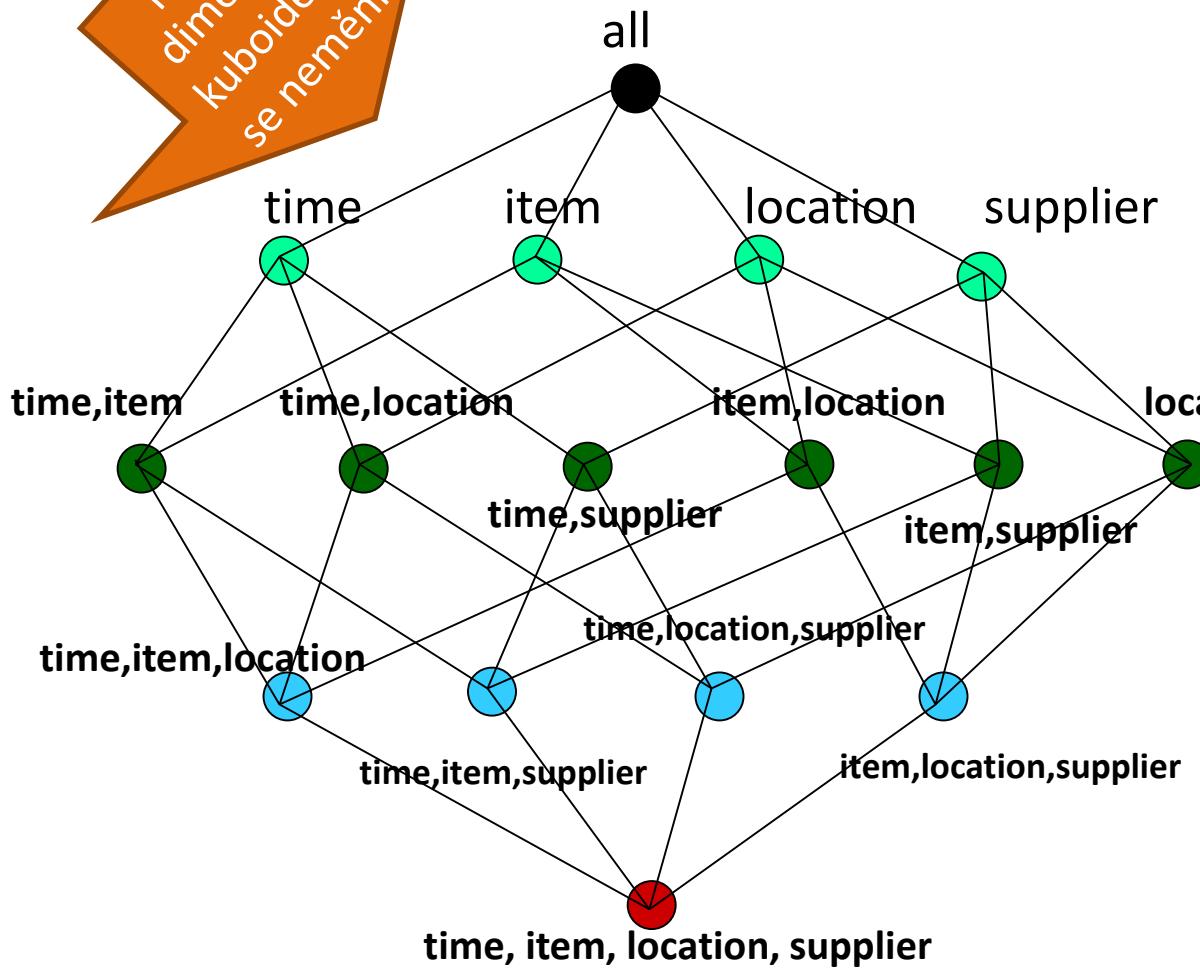
$$g_m(A_1 \times A_2 \times A_3 \times \dots \times A_i \times \dots \times A_m) \vee$$

$$g_m(A_1 \times A_2 \times A_3 \times \dots \times A_j \times \dots \times A_m) =$$

$$g_{m-1}(A_1 \times A_2 \times A_3 \times \dots \times A_{m-1})$$

Kostka jako svaz kuboidů

pořadí
dimenzi v
kuboidech
se nemění



0D (vrcholový) kuboid

1D kuboidy

2D kuboidy

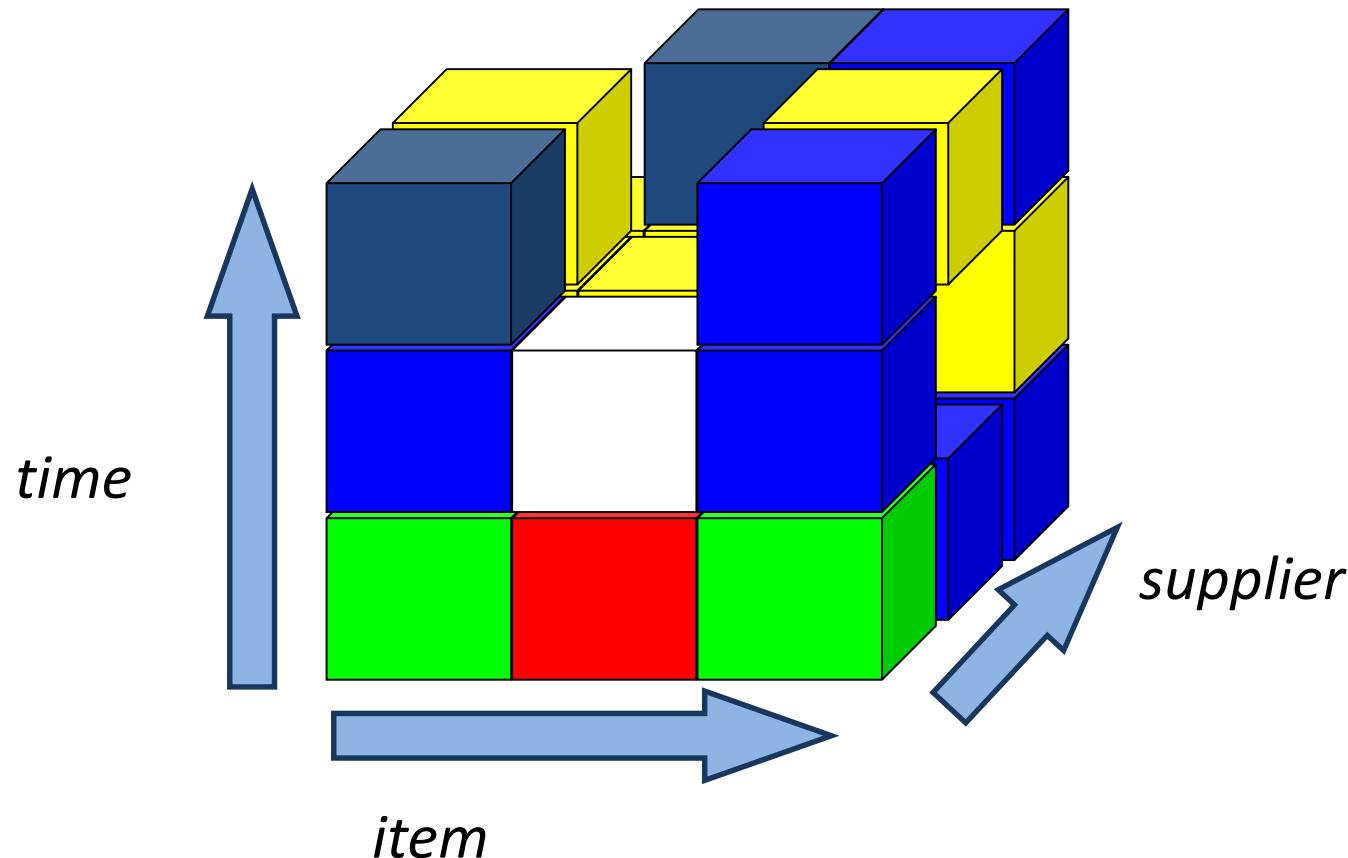
3D kuboidy

4D (základní) kuboid

Model multidimenzionální kostky

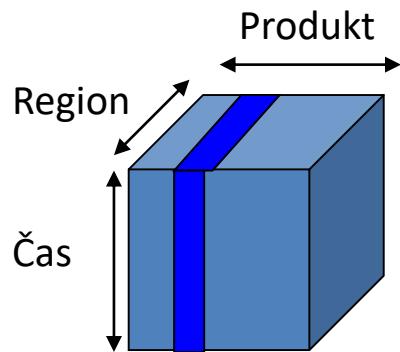
- Každá dimenze A_i má ***možnou kardinalitu*** danou *datovým typem dimenze* (i nekonečnou) a ***skutečnou kardinalitu* k_i** , (tj. skutečný počet prvků dimenze daný ***existujícími ohodnocenými fakty***)
- Proto bude výsledná kostka "vykotlaná", tj. ***řídká n-rozměrná matice***.

Multidimenzionální kostka jako řídká matic

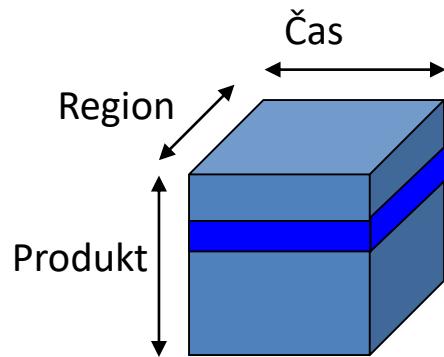


3D kostka - příklad 3! otočení

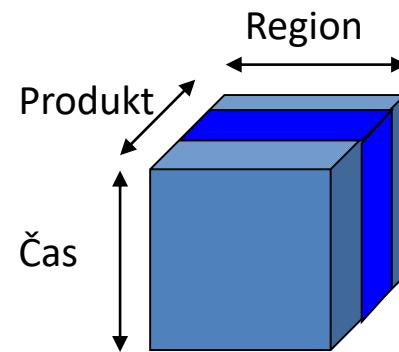
- Princip multidimenziorní kostky



Analýza údajů
pro určitý produkt-
produkt, region, čas



Analýza pro určité
časové období -
čas, region, produkt



Analýza údajů podle
regionálních kritérií -
region, produkt, čas

je možný i produkt, čas, region - čas, produkt, region a region, čas, produkt

Typy agregačních výpočtů-distributivní

- **Distributivní:** Pokud můžeme kostku rozdělit na části, pro něž se dané funkce vypočte zvlášť a poté jednotlivé výsledky spočítat dohromady, a neovlivní to výsledek funkce, pak je funkce distributivní
- např. počet, **součet**, min, max
- **součet** je distributivní, tj. celkový součet je součtem dílčích součtů

Typy agregačních výpočtů - algebraické

- *Algebraické*: jde o výsledek funkce, která má M parametrů, z nichž každý může být získán výpočtem distributivní funkce.
- Příkladem je průměr, který lze spočítat jako součet/počet
- *průměr* je algebraický, tj. celkový průměr **není obecně průměrem dílčích průměrů**

Kdo tomu nevěří

- Platí to, pouze *jsou-li počty vzorků v obou množinách stejné*
- $\{4,5,6\}$, součet=15, průměr= $15/3=5$
- $\{100,200,300\}$, součet=600,
průměr= $600/3=200$
- $\{5,200\}$, součet=205, průměr= $205/2=102.5$
- $\{4,5,6,100,200,300\}$, součet=615,
průměr= $615/6=102.5$

Kdo tomu nevěří

- nejsou-li stejné:
- $\{1,2,3,4,5,6\}$, součet=21, průměr= $21/6=3.5$
- $\{100,200,300\}$, součet=600,
průměr= $600/3=200$
- $\{3.5,200\}$, součet=203.5,
průměr= $203.5/2=101.75$
- $\{1,2,3,4,5,6,100,200,300\}$, součet=621,
průměr= $621/9=69$

Typy aggregačních výpočtů - holistické

- **Holistické (celostní)**: pro tyto jednotky neexistuje algebraické funkce,
- např. zde patří funkce pro zjištění nejčastěji se vyskytující položky

Typy agregačních výpočtů

- Budeme se zabývat hlavně ***distributivními*** (počet, součet) a
- ***algebraickými*** agregáty (aritmetický průměr).

Neagregované - detailní hodnoty

- Předpokládejme, že existuje n dimenzí $\{D_1, D_2, D_3, \dots, D_n\}$
- Každá z n dimenzí má ***skutečnou kardinalitu*** k_i , prvků pro všechna $1 \dots i \dots n$.
- Fakt dané kostky pro agregační funkci ***agr***,
detail $(D_1 \times D_2 \times D_3 \times \dots \times D_n) = F$, kterou je existující hodnota na průsečíku hodnot dimenzí $(d_1, d_2, d_3, \dots, d_i)$ nazveme detailní hodnotou nebo ***detailem***.

Kostky počet, součet a průměr

- Uvažujme nyní výpočet funkcí (multidimenzionálních kostek)
 - *počet,*
 - *součet,*
 - *průměr*
- pro uspořádanou podmnožinu aktivních dimenzí $\{A_1, A_2, A_3, \dots, A_m\}$ a dané uspořádání dimenzí $\{D_1, D_2, D_3, \dots, D_n\}$.

První krok agregace detailů

- Pro dané uspořádání dimenzí $\{D_1, D_2, D_3, \dots, D_n\}$ platí pro jednotlivé podkostky ***počet_i***, ***součet_i***, a ***průměr_i*** ($d_1, d_2, d_3, \dots, d_i$) a detailní hodnoty ***h*** ohodnocené hodnotami všech dimenzí ($d_1, d_2, d_3, \dots, d_n$):
- ***součet_n*** ($d_1, d_2, d_3, \dots, d_n$) = $\Sigma \text{ detail } (d_1, d_2, d_3, \dots, d_n)$
- ***počet_n*** ($d_1, d_2, d_3, \dots, d_n$) = $\Sigma 1$ přes všechny hodnoty ***detail*** ($d_1, d_2, d_3, \dots, d_n$) s týmiž hodnotami dimenzí
- ***průměr_n*** ($d_1, d_2, d_3, \dots, d_n$) = ***součet_n*** ($d_1, d_2, d_3, \dots, d_n$) / ***počet_n*** ($d_1, d_2, d_3, \dots, d_n$)

Podkostky

- Potom pro dané uspořádání $n > m \geq 0$ dimenzí $\{A_1, A_2, A_3, \dots, A_m\}$ platí pro příslušné podkostky počet_m , součet_m a průměr_m a jejich přímé následníky $m+1$ v částečném uspořádání $\{A_1, A_2, A_3, \dots, A_i, \dots, A_m\}$:
- $\text{součet}_m(d_1, d_2, d_3, \dots, d_m) = \sum \text{součet}_{m+1}(d_1, d_2, d_3, \dots, d_i, \dots, d_m)$ přes všechny hodnoty z D_i
- $\text{počet}_m(d_1, d_2, d_3, \dots, d_m) = \sum \text{počet}_{m+1}(d_1, d_2, d_3, \dots, d_i, \dots, d_m)$ přes všechny hodnoty z D_i ,
- $\text{průměr}_m(d_1, d_2, d_3, \dots, d_m) = \text{součet}_m(d_1, d_2, d_3, \dots, d_m) / \text{počet}_m(d_1, d_2, d_3, \dots, d_m)$

Příklad detailních hodnot

hodnoty dimenzií seřazeny např. vzestupně podle uspořádání

time	item	location	fakt
22.6.	párek	Brno	4
22.6.	párek	Brno	9
	párek	Praha	21
22.6.	rohlík	Brno	12
22.6.	rohlík	Brno	4
22.6.	rohlík	Brno	3
22.6.	rohlík	Praha	16
22.6.	rohlík	Praha	6
23.6.	rohlík	Brno	5
23.6.	rohlík	Praha	14

celkový součet se nemění

94

více hodnot faktů pro stejné hodnoty dimenzií

První krok pro kostku *součet*

time	item	location	fakt
22.6.	párek	Brno	13
22.6.	párek	Praha	21
22.6.	rohlík	Brno	19
22.6.	rohlík	Praha	22
23.6.	rohlík	Brno	5
23.6.	rohlík	Praha	14

- není více hodnot faktů pro stejné hodnoty dimenzií
 - nebylo dimenzií

Podkostka *součet* bez dimenze location

time	item	fakt
22.6.	párek	34
22.6.	rohlík	41
23.6.	rohlík	19
94		

Podkostka *součet* bez dimenze item

time	fakt
22.6.	75
23.6.	19
94	

Podkostka *součet* bez dimenzí

fakt
94

při nulovém počtu dimenzí
jde o celkový agregát

94

Jiné pořadí dimenzí v prvním kroku

location	time	item	fakt
Brno	22.6.	párek	13
Brno	22.6.	rohlík	19
Brno	23.6.	rohlík	5
Praha	22.6.	párek	21
Praha	22.6.	rohlík	22
Praha	23.6.	rohlík	14

94

Podkostka *součet* bez dimenze item

location	time	fakt
Brno	22.6.	32
Brno	23.6.	5
Praha	22.6.	43
Praha	23.6.	14

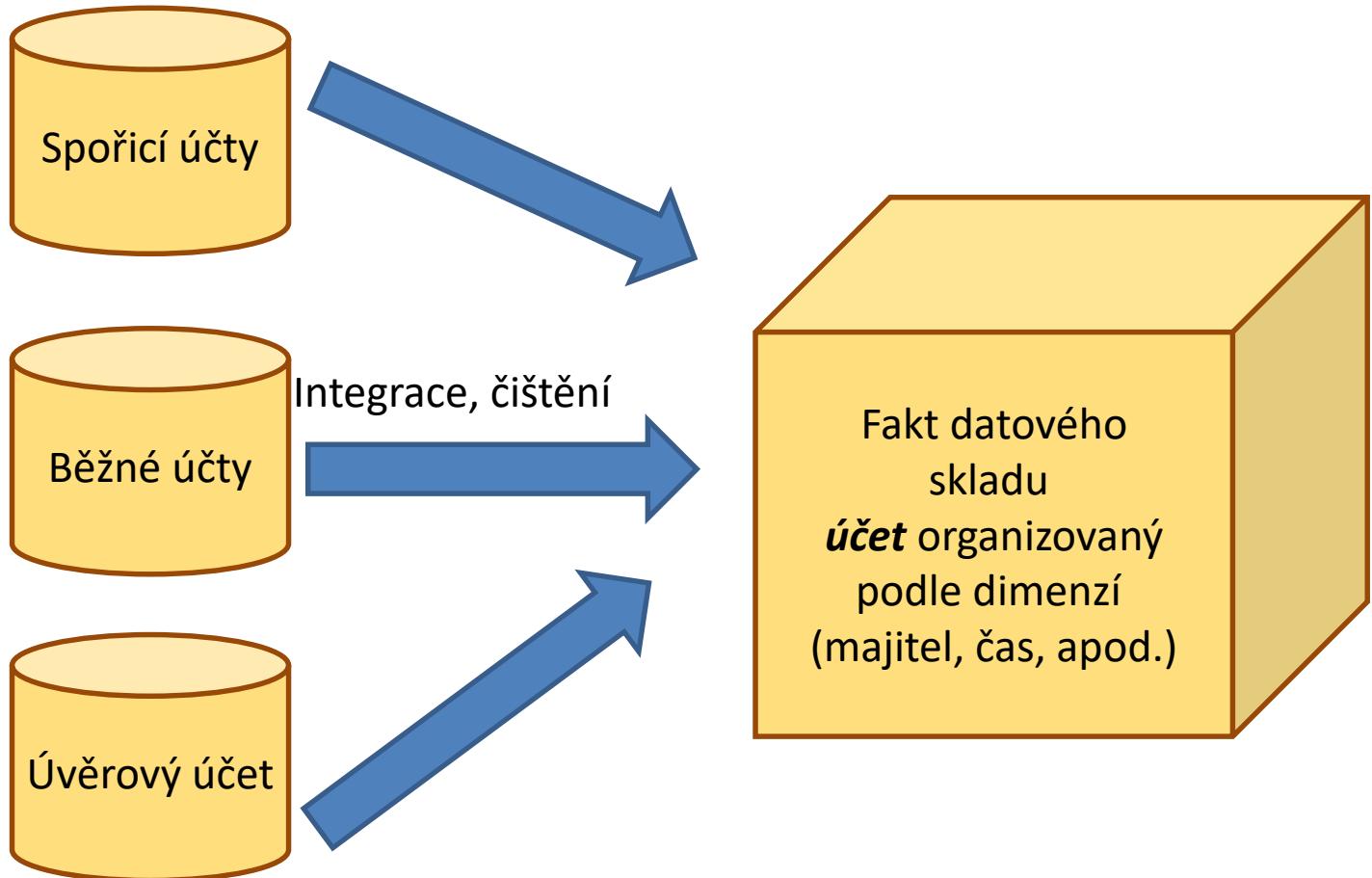
VLASTNOSTI DATOVÉHO SKLADU

Orientace podle dimenzí

- Fakta jsou zapisována podle vlastností vyjádřeného průsečíkem *n-dimenzí*

Integrace příklad

Data z různých zdrojů



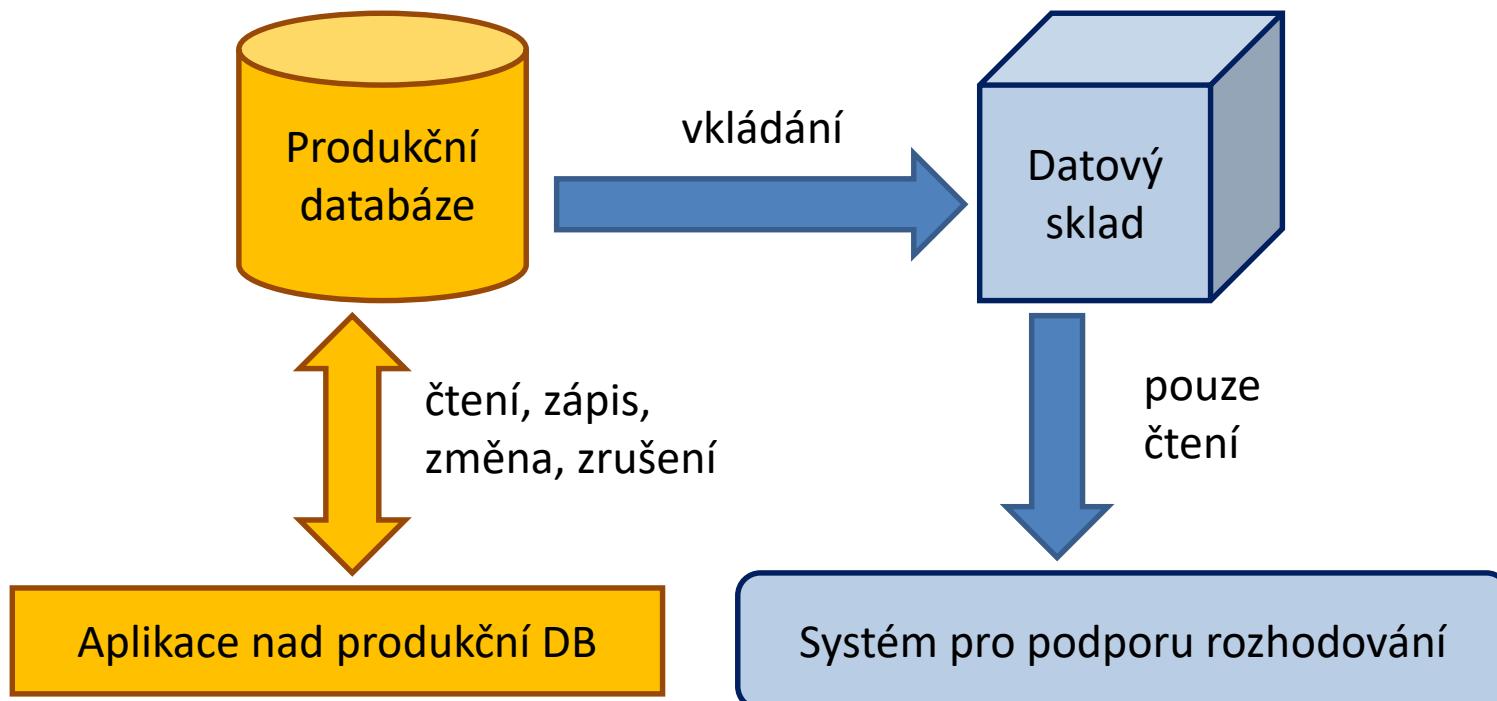
Integrace

- data týkající se konkrétního předmětu se ukládají **pouze jednou**
 - jednotná terminologie, jednotky veličin
- vytvořen případně **integrací** několika heterogenních zdrojů dat - relační databáze, textové soubory, on-line transakce
 - problém nekonzistentních zdrojů dat
- nutnost **úpravy, vyčištění a sjednocení** (integrace) vstupních dat
 - je nutné ověřit konzistenci v pojmenování proměnných, jejich struktury a jednotkách pro různé zdroje dat

Čas

- klíčový atribut (*lineární, uspořádaný a spojity*), vhodný na spojité grafy
- Časový horizont datového skladu je zpravidla podstatně delší než u produkční databáze
 - Produkční databáze: pouze současně aktuální data
 - Data v datovém skladu: poskytují informace z historické perspektivy (např. posledních 5-10 let)
- Každá klíčová struktura v datovém skladu
 - obsahuje čas, explicitně nebo implicitně
 - klíč u produkčních dat nemusí vždy obsahovat čas

Neměnnost



Neměnnost

- zpravidla **fyzicky oddělené** uložení dat transformovaných z produkčních databází
- v datových skladech se data **nemění**, manipulace s daty je tedy jednodušší.
 - dva typy operací: **vkládání** dat a **čtení** dat
 - optimalizace a normalizace ztrácí smysl
 - **nepotřebuje se** zpracování **transakcí**, zotavení, mechanismy **pro řízení souběžného přístupu**

Produkční databáze x datový sklad

- ***Uživatelé a orientace systému:***
 - běžný uživatel x ***manager***
- ***Datový obsah:***
 - současná, detailní x ***historická, sloučená***
- ***Návrh databáze:***
 - ER model + aplikace x ***multidimenzionální kostka, dimenze, fakta***
- ***Pohled na data:***
 - aktuální, lokální x ***agregovaný***
- ***Přístupové vzory:***
 - jednoduchá aktualizace x ***read-only, komplexní dotazy***

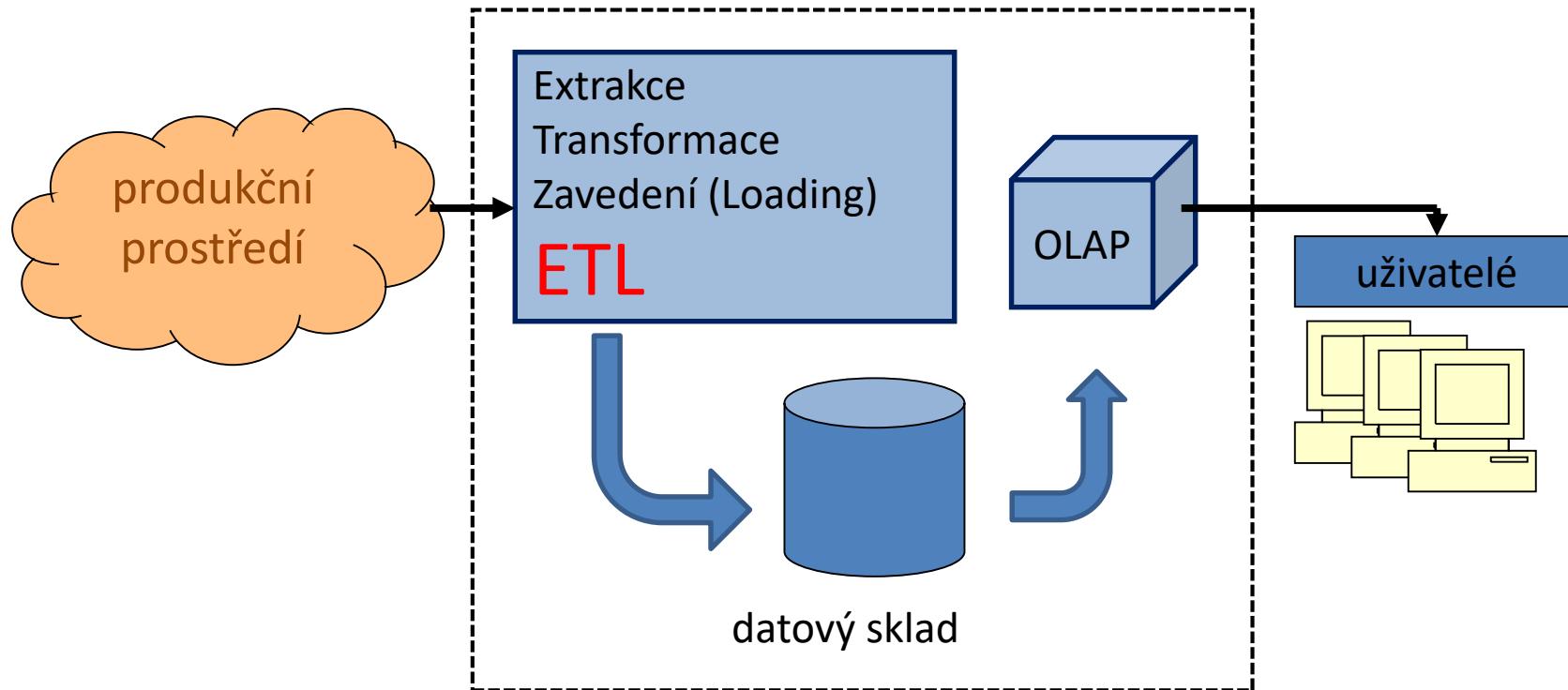
Porovnání vlastností

Vlastnost	Produkční DB	Datový sklad
Čas odezvy	ms - s	s - hod
Operace	DML, např. SQL	Jen čtení, např. MDX
Původ dat	30 – 60 dní	Snímky za čas. úsek
Organizace dat	Podle aplikace	Podle dimenzí
Velikost	Malá až velká	<i>big data</i>
Zdroje dat	operační, interní	operační, interní, externí
Činnosti	Transakce (OLTP)	Analýza (OLAP)

Shrnutí požadavků na datový sklad

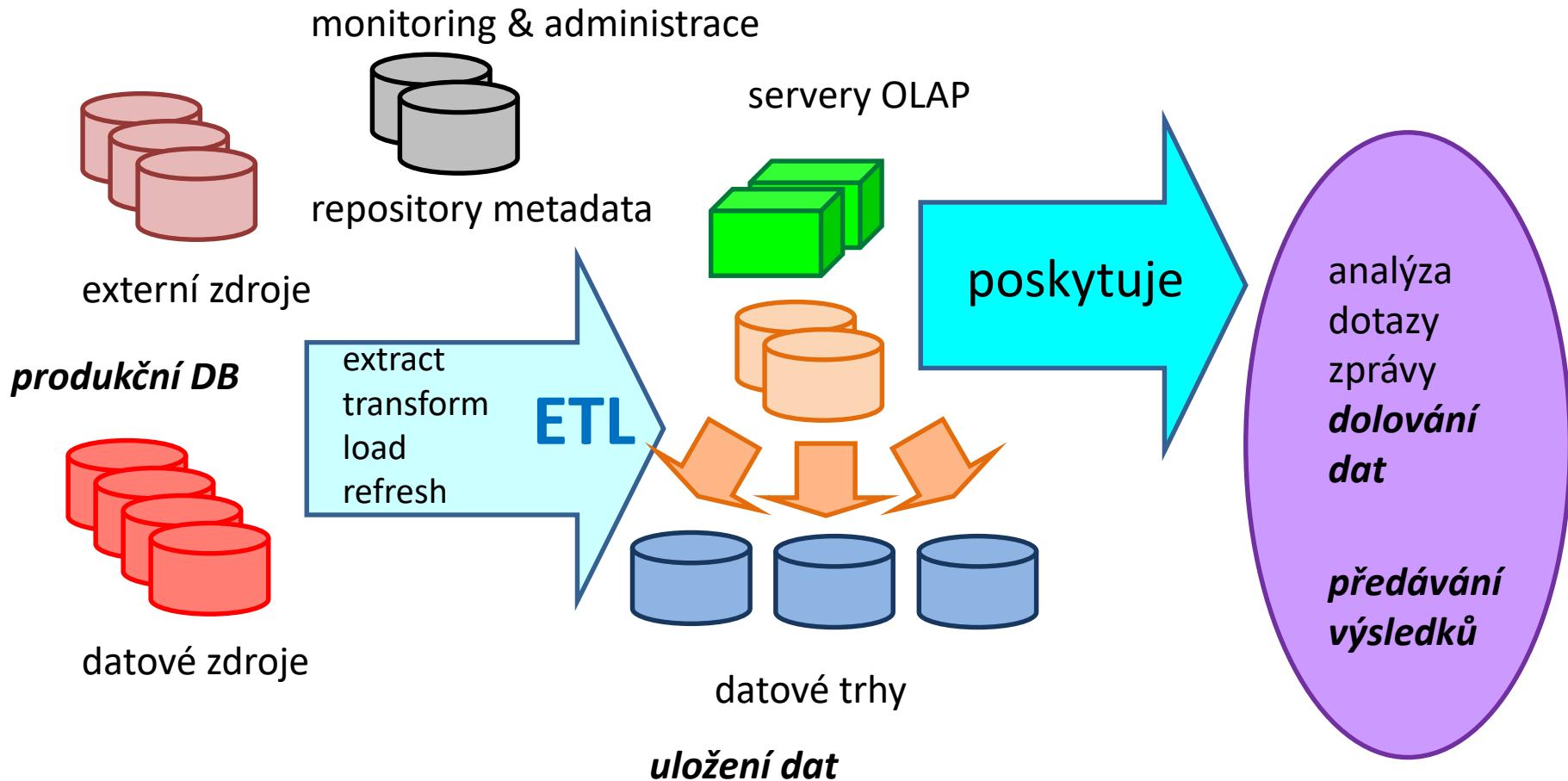
- schopnost *agregace*
- databáze navržená pro *analytické* dotazy
- možnost *integrovat* data z více aplikací
- častá operace *čtení* z databáze
- nahrání dat po určité časové periodě
- možnost využití *současných i historických dat*

Celkové schéma datového skladu



Získání údajů -> úprava a zavedení do datového skladu
-> analýza -> zpřístupnění uživatelům

Architektura datového skladu podrobnější



Architektura datového skladu

- 3 hlavní části
- **Získání dat**
 - Zdrojová data + místo přípravy dat (*ETL*)
- **Uložení dat (datová kostka)**
 - Datový sklad + datové trhy + uložení metadat
- **Předávání výsledků**
 - přes model multidimenzionální databáze samotné získání informací (*OLAP*, data mining, tiskové sestavy atd.)

Získání dat

- ***extrahování dat*** z mnoha produkčních databází a externích zdrojů
- čištění, transformování a integrování těchto dat - ***Extraction, Transformation, Loading - ETL***
- ***periodické doplňování*** datového skladu tak, aby odrážel změny v produkčních databázích a přenos dat z datového skladu, nejčastěji do pomalejší archivní paměti.

Uložení dat

- k hlavnímu datovému skladu je přidruženo několik ***datových trhů (data marts)*** různých dílčích uživatelů.
- data ve skladu a trhu jsou uložena a spravována jedním nebo více skladovými servery, které prezentují multidimenzionální pohled na data OLAPu a/nebo přímo různým koncovým prostředkům

Uložení dat

- Jde o oddělené skladiště pro uložení velkého množství především historických dat
- Je navrženo ***pro analýzu***, ne pro rychlý přístup k datům
- ***read-only***
- Musí být přístupné pro více druhů nástrojů – odpovídající rozhraní

Předávání výsledků

- OLAP
- a jiné prostředky
- dotazovací,
- generátory zpráv,
- analytické a
- prostředky ***dolování dat (data mining)***.

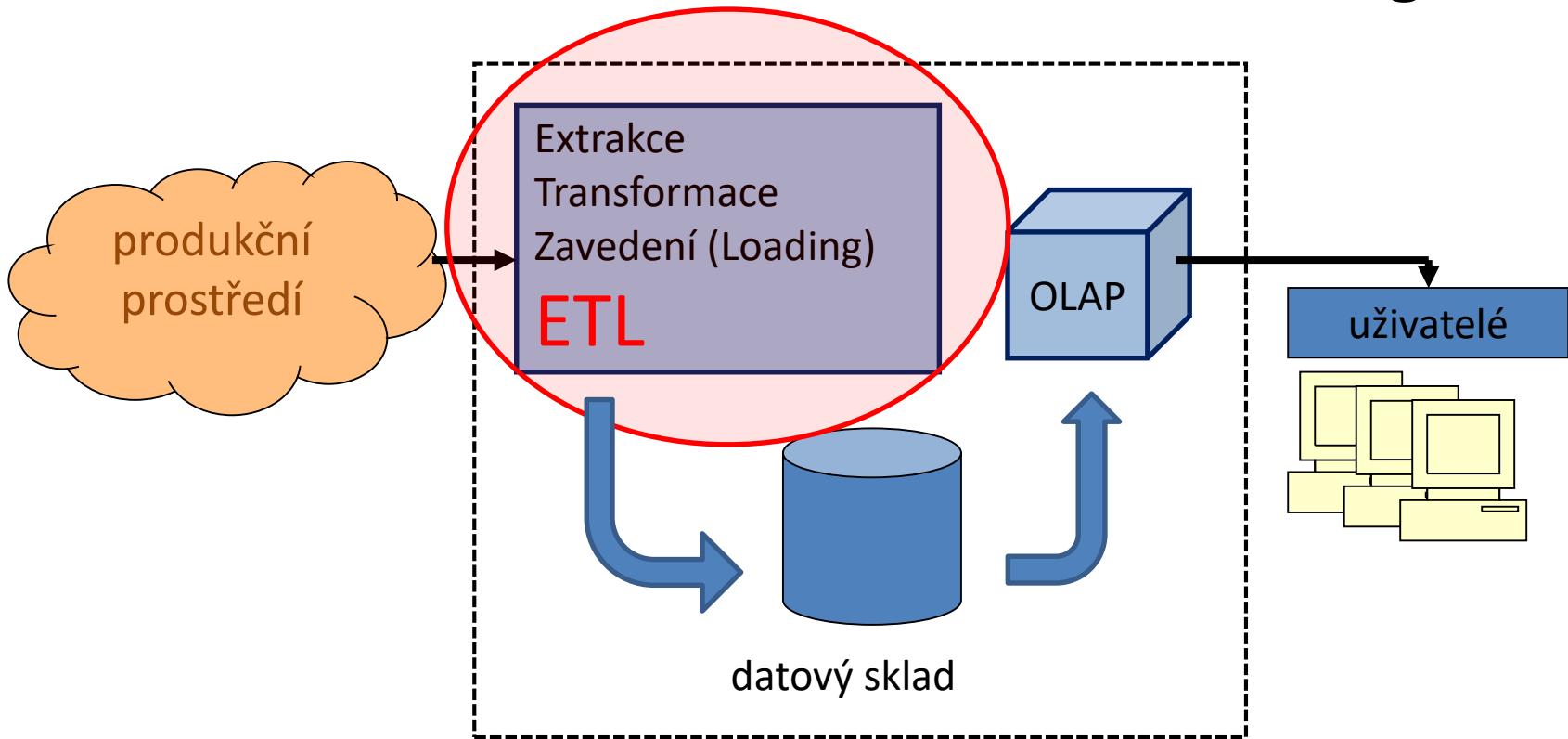
Předávání výsledků

- Poskytuje informace pro různé uživatele
 - **Začínající uživatelé**: tiskové sestavy, jednoduché dotazy
 - **Běžní uživatelé**: statistická analýza, různá zobrazení dat, předdefinované dotazy
 - **Pokročilí uživatelé**: provádění multidimenzionaální analýzy, formulace vlastních OLAP dotazů, používání data miningu

ZÍSKÁNÍ DAT - EXTRACTION, TRANSFORMATION, LOADING

Příprava údajů – etapa ETL

- Klíčová úloha správy datového skladu
- ETL = Extraction, Transformation, Loading



Typy zdrojových dat

- ***Produkční data***
 - Data získaná z různých produkčních DB podniku pomocí dotazů
- ***Interní data***
 - Data uložená v privátních souborech (zpravidla Excel XLS) zaměstnanců organizace
- ***Externí data***
 - Data z různých zdrojů, která mohou být pro organizaci užitečná (webovské služby)

Etapa ETL

- ***Extrakce*** – výběr dat různými metodami
- ***Transformace*** – ověření, čištění, integrace a časové označení dat
- ***Loading*** – přesun dat do datového skladu
- ***Hlavní cíl:*** integrace údajů

Hlavní úkoly ETL procesu

- *Určit zdroje dat*, interní i externí
- Příprava *mapování* mezi zdrojovými a cílovými daty
- Stanovení *metadat* pro extrakci dat
- Určit *pravidla pro transformaci a čištění* dat
- Plán pro *agregaci* tabulek
- Návrh oblasti přípravy dat
- Napsat procedury pro nahrávání dat

Extrakce

- **Zdroj:** Data z nehomogenního produkčního prostředí, popř. z archivních dat
- Různé možnosti extrakce
 - **Periodická extrakce** – z interních zdrojů
 - **Občasná extrakce** – z externích zdrojů (např. Internet)
 - **První extrakce** – provádí se především z archivních dat

Extrakce - součásti procesu

- *Identifikace zdrojů* (struktury a aplikace)
- Stanovení *metody extrakce* pro každý zdroj
 - manuální – napíši si sám SQL příkazy
 - s využitím nástrojů
- *Frekvence extrakcí* pro každý zdroj – většinou se liší pro různé zdroje
- Stanovení *časového okna* pro extrakci – kdy ji provádět
- *Paralelní x sériová* extrakce pro jednotlivé zdroje dat
- Zpracování *výjimek* při extrakci

Postup při identifikaci zdrojů

- Výpis všech ***datových položek potřebných v tabulce faktů***
- Výpis všech ***dimenzií***
- Pro každou cílovou položku najít zdroj a jeho položku
- Je-li více zdrojů pro jednu cílovou položku, vybrat preferovaný zdroj
- Identifikace vícenásobných zdrojů pro jeden cíl – stanovení ***konsolidačních pravidel***
- Identifikace vícenásobných cílů na jeden zdroj – stanovení ***dělících pravidel***
- Určení ***implicitních hodnot***
- Zjištění ***chybějících hodnot ve zdrojových datech***

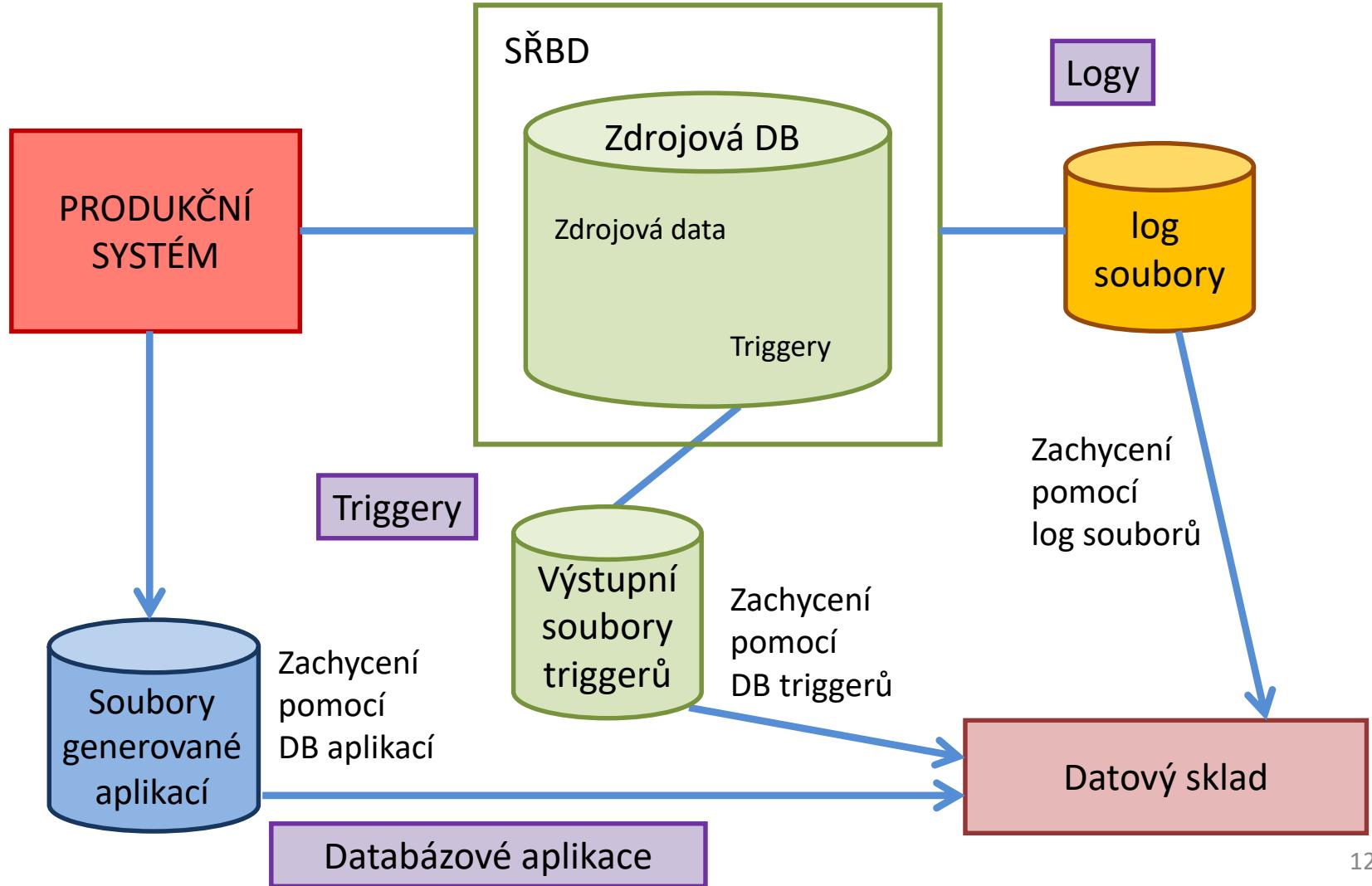
Metody extrakce

- Metoda extrakce statických dat
 - Vytvoření obrazu zdrojové databáze na výstupu
 - Používá se při iniciálním nahrávání dat do skladu
- Metody extrakce při aktualizaci dat
 - Metody ***přímé extrakce***
 - Metody ***odložené extrakce***

Přímá extrakce

- Liší se způsobem zachycení změn v DB od posledního nahrání
 - Zachycení pomocí ***log souborů*** (vytvořených databází)
 - Zachycení pomocí databázových ***triggerů***
 - Při každé změně se spustí trigger, který zapíše změnu do souboru
 - Zachycení pomocí samotných ***databázových aplikací***
 - Editace aplikace tak, aby ukládala záznamy o provedených změnách v DB

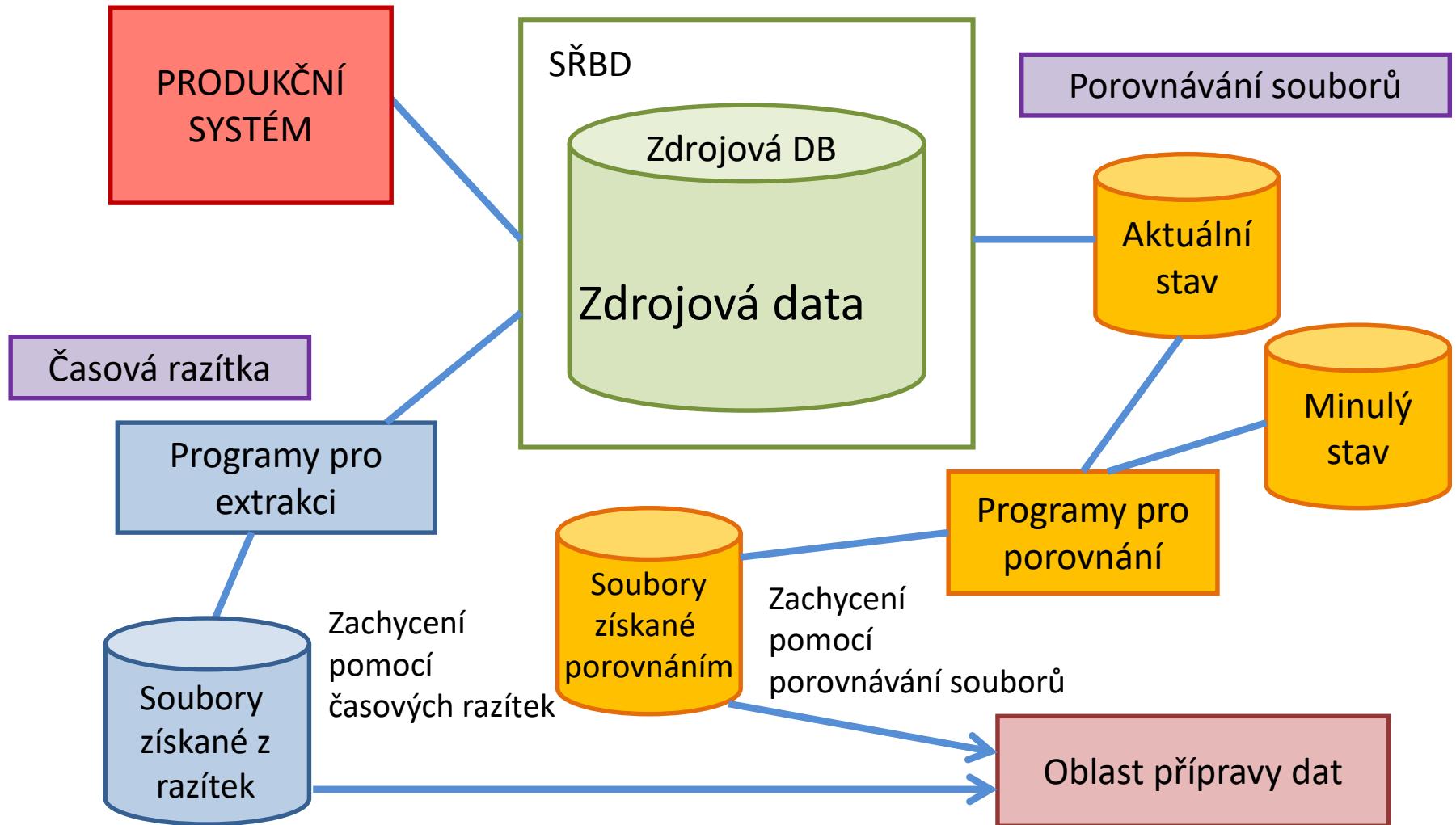
Přímá extrakce



Odložená extrakce

- Nezachycují změny při jejich vzniku, ale až při nahrávání se **porovnává zdrojová a cílová DB**
 - Zachycení pomocí **časových razítek**
 - Razítka jsou označeny záznamy, které byly přidány nebo editovány – ty se pak při nahrávání dat naleznou (problém s mazáním)
 - Zachycení pomocí **porovnávání souborů**
 - Vytvoří se soubor s kopíí dat ve stavu současném a včerejším, pak se soubory porovnají (velmi neefektivní)

Odložená extrakce



Transformace

- Cílem je zvýšit kvalitu vstupních dat a zvýšit jejich použitelnost pro cílového uživatele
- Někdy je kvalita vstupních dat velmi proměnlivá -> čištění dat (odstranění nekvalitních dat)
- Často je potřeba odstranit tzv. ***anomálie***, které v klasických databázích běžně vznikají

Anomálie

- Příklady anomálií:
 - Přechod z MS-DOSu na Windows – např. kódování češtiny
 - Lidský faktor – různé překlepy, pravopisné chyby
- Potřeba rozdělení složených atributů na atomické

Časté problémy

- ***Nejednoznačnost údajů***
 - Např. různě uložená informace o pohlaví zákazníka (M, muž, Muž atd...)
- ***Chybějící hodnoty***
 - Tyto hodnoty je potřeba doplnit, popř. ignorovat nebo označit nějakým příznakem
- ***Duplicitní hodnoty***
 - Většinou není příliš velký problém je odstranit, někdy je to však časově náročné

Transformace

- ***Konvence názvů pojmu a objektů***
 - je nutné sjednotit terminologii používanou různými zdroji dat
- ***Různé peněžní měny***
 - problém vznikne např. při přechodu z CZK na Euro
- ***Formáty čísel a textových řetězců***
 - použití různých datových typů pro ukládání čísel (např. řetězce)

Transformace

- ***Referenční integrita***
 - Neustálé změny v reálném světě zkreslují data – např. i po zrušení oddělení firmy zůstanou v DB údaje o jeho zaměstnancích
- ***Chybějící datum***
 - Časový aspekt je v datových skladech velmi důležitý, ve vstupních datech však čas často chybí, často je nutné jej doplnit

Loading

- Přesun údajů a jejich uložení do tabulek datového skladu
- Pokud možno by měl probíhat automatizovaně
- Rozlišujeme podle periody přesunů...
 - to závisí především na požadavcích aplikace
 - většinou jde o časově náročnou operaci, především u iniciálního přenosu

3 typy loadingu (nahrávání)

- ***Iniciální nahrávání***
 - Nahrávání všech dat do prázdného skladu
- ***Inkrementální nahrávání***
 - promítnutí změn v DB do datového skladu
(provádí se periodicky)
- ***Přepis dat***
 - kompletní smazání obsahu skladu a nahrání aktuálních dat

Módy nahrávání dat

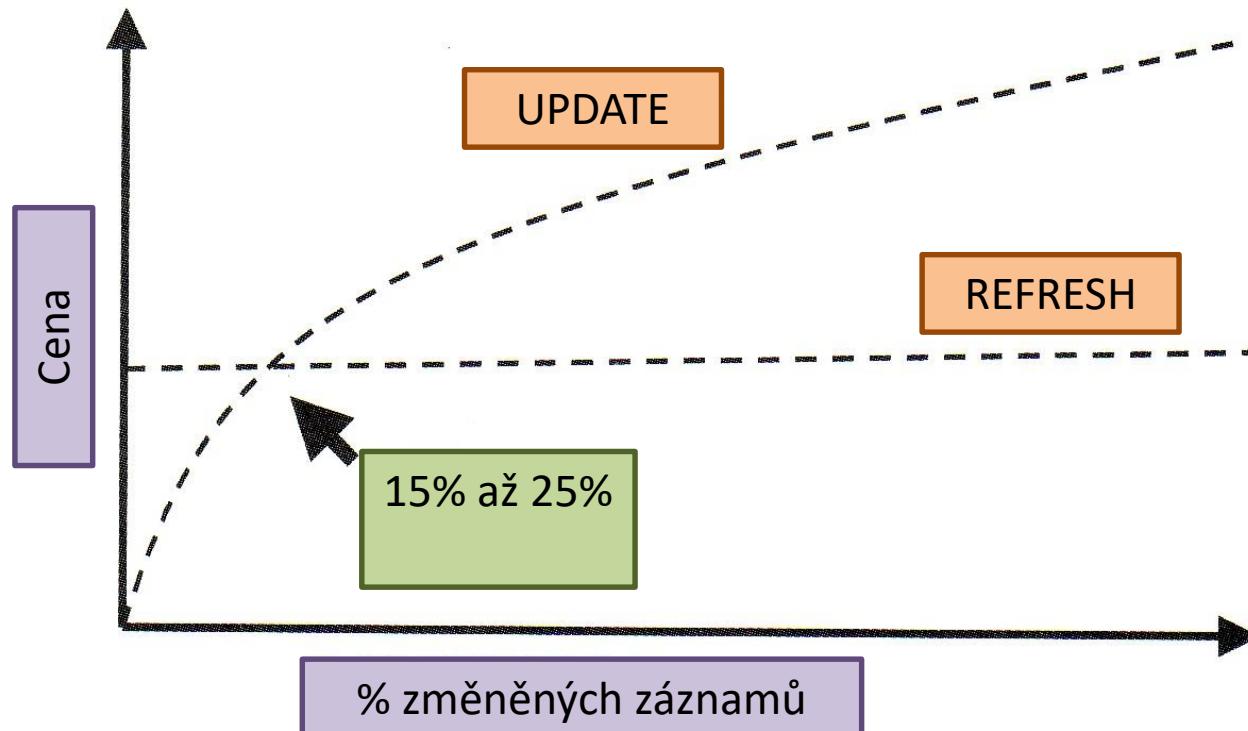
- **Nahrání (Load)**
 - Pokud cílová tabulka obsahuje data, pak jsou smazána a nahrazena aktuálními
- **Přidání (Append)**
 - Přidání nových dat ke stávajícím, při duplicitě může uživatel zvolit další postup
- **Destruktivní sloučení**
 - Stejné jako přidání, při stejných klíčích se přepíše hodnota daného řádku
- **Konstruktivní sloučení**
 - Při stejných klíčích se přidá nový prvek a označí se jako nový, starý v datovém skladu zůstane

Refresh x Update

Po počátečním naplnění jsou data v datovém skladu udržována v aktuálním stavu pomocí:

REFRESH - kompletní natažení ve specifikovaném intervalu

UPDATE - aplikace inkrementálních změn



Architektury serveru OLAP

- **Multidimenzionální OLAP (MOLAP)**
 - Multidimenzionální paměťový stroj založený na polích (array) - techniky řídkých matic
 - Rychlé indexování předzpracovaných agregovaných dat
- **Relační OLAP (ROLAP)**
 - Užívá relační nebo rozšířená relační SŘBD
 - Zahrnují optimalizaci back-endu SŘBD, implementaci aggregační navigační logiky a dodatečných pomůcek a služeb
 - Velká možnost škálování
- **Hybridní OLAP (HOLAP)**
 - Uživatelská flexibilita kombinující obě předchozí metody
- **Specializovaný SQL server**
 - specializovaná podpora pro SQL dotazy nad schématy hvězda/sněhová vločka

Multidimezionální OLAP (MOLAP)

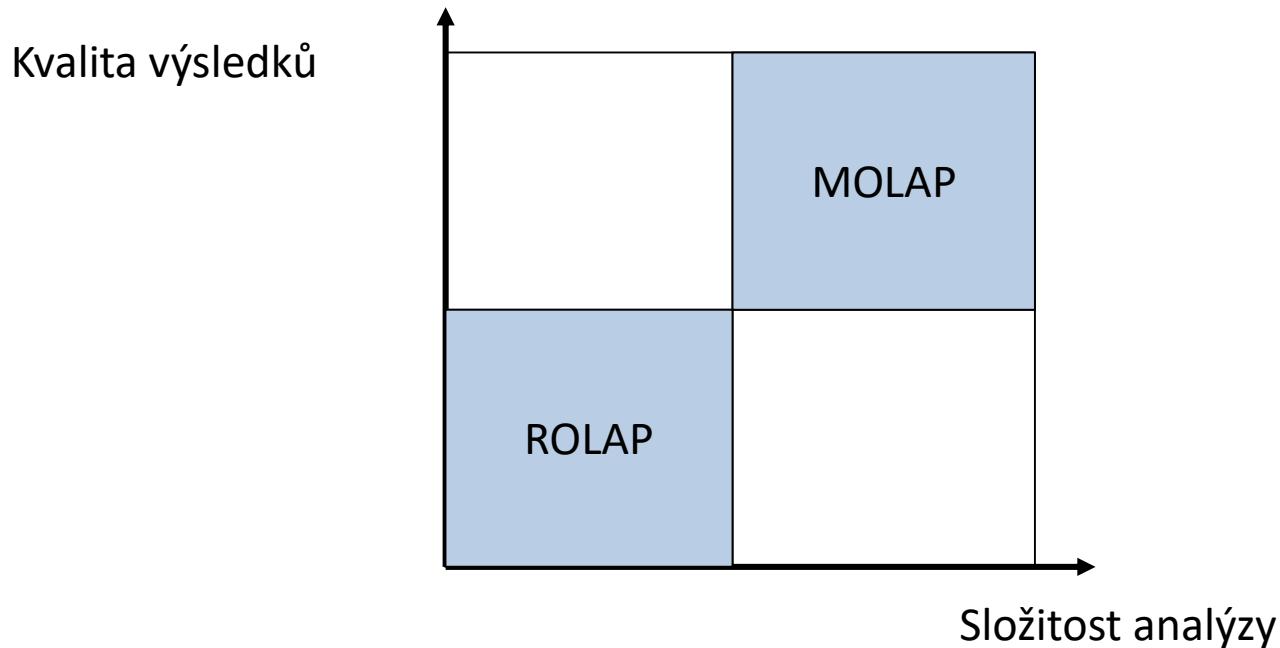
- Data se ukládají do vlastních datových struktur
- Databáze konstruována pro rychlé vyhledávání údajů
- **Výhoda:** maximální výkon
- **Nevýhoda:** redundance údajů, velké prostorové nároky

Relační databázový OLAP (ROLAP)

- Údaje jsou získávány z *relačních tabulek*, jsou uživateli předkládány jako multidimenzionální pohled
- Data jsou uložena jako záznamy relační tabulky
- Žádná redundance

ROLAP x MOLAP

- Záleží na důležitosti analýzy
- Záleží na složitosti dotazů uživatelů



ROLAP x MOLAP - uložení dat

- **ROLAP**
 - Data **uložena** v relačních tabulkách
 - Možnost získat detailní i agregovaná data
 - Velké datové prostory
 - Veškerý přístup k datům prostřednictvím datového skladu
- **MOLAP**
 - Data **neuložena** v relačních tabulkách
 - Různá aggregovaná data uložena v multidimenz. databázi
 - Průměrně velké datové prostory
 - Přístup jak do MDDB (sumy), tak do datového skladu (detailní data)

ROLAP x MOLAP - technologie

- ***ROLAP***
 - Použití komplexních SQL dotazů k získání dat ze skladu
 - Datové kostky jsou vytvářeny za běhu
 - Multidimenziona lní pohledy vytváří prezentační vrstva
- ***MOLAP***
 - Vytváření datových kostek předem
 - Použití technologie pro ukládání multidim. dat v polích, ne tabulkách
 - Technologie pro zpracování řídkých matic

ROLAP x MOLAP - funkce a vlastnosti

- ***ROLAP***
 - Známé prostředí a možnost použití známých nástrojů
 - Limitované použití komplexních analýz
 - Omezené použití operace drill-across
- ***MOLAP***
 - Rychlejší přístup
 - Velká knihovna funkcí pro komplexní analýzu
 - Snadná analýza bez ohledu na počet dimenzí
 - Rozšiřující prostor pro operace drill-down a slice-and-dice

Hybridní OLAP (HOLAP)

- Kombinace MOLAP a ROLAP
- Data *v relačních tabulkách*, agregace se ukládají do multidimenzionálních struktur
- Využití multidimezionální cache

KONCEPTUÁLNÍ NÁVRH DATOVÉHO SKLADU

Fakta (míry) a dimenze

- Navrhujeme v relačním modelu - tabulky
- Každá datová kostka obsahuje 2 typy údajů – **fakta (míry)** a **dimenze**
- **Fakta**
 - Největší relace v DB, zpravidla jen jedna
 - Obsahuje numerické měrné jednotky obchodování
 - V kombinaci s relacemi dimenzí tvoří základní schémata

Fakta a dimenze

- ***Dimenze (číselníky)***
 - logicky nebo hierarchicky uspořádané údaje
 - textové popisy obchodování
 - jsou menší a nemění se tak často
 - nejčastěji: časové, geografické a produktové dimenze (stromové struktury)

Kontinent

Země

Územní celek

Město

Druh produktu

Kategorie

Subkategorie

Název produktu

Rok

Kvartál

Měsíc

Týden

Konceptuální modelování kostky

- **Schéma hvězdy** - *jedna tabulka faktů* je ve středu připojena k množině relací dimenzí
- **Schéma sněhové vločky** - zjemnění schématu hvězdy, kde existuje *hierarchie dimenzí* normalizovaná do množiny navázaných relací dimenzí
- **Konstelace faktů** - více relací faktů, které sdílejí relace dimenzí - je možné chápat jako kolekci hvězd, proto se také někdy nazývá *schéma galaxie*

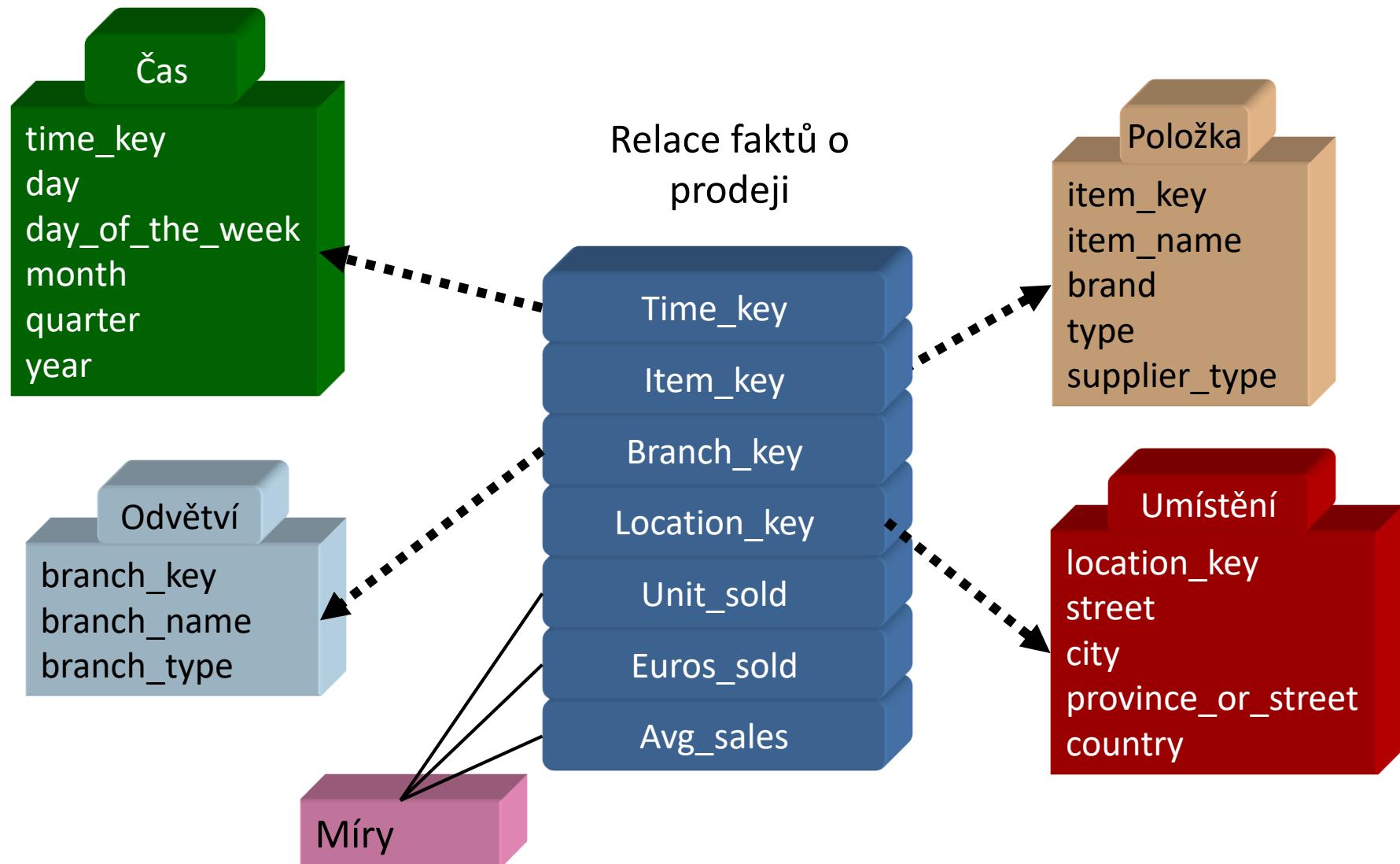
Schéma hvězdy - star

- Každá relace dimenze sestlává z množin, které odpovídají hodnotám dimenze.
- Hvězdicové schéma ***neposkytuje explicitně podporu pro hierarchii atributů.***
- Lze obejít organizačně

Schéma hvězdy - star

- Relace faktů obsahuje cizí klíče do relací dimenzí, ty se vztahují k jejím primárním klíčům
- Snadno pochopitelné
- Relace dimenzí jsou však nejsou normalizované, je to tedy poměrně pomalé

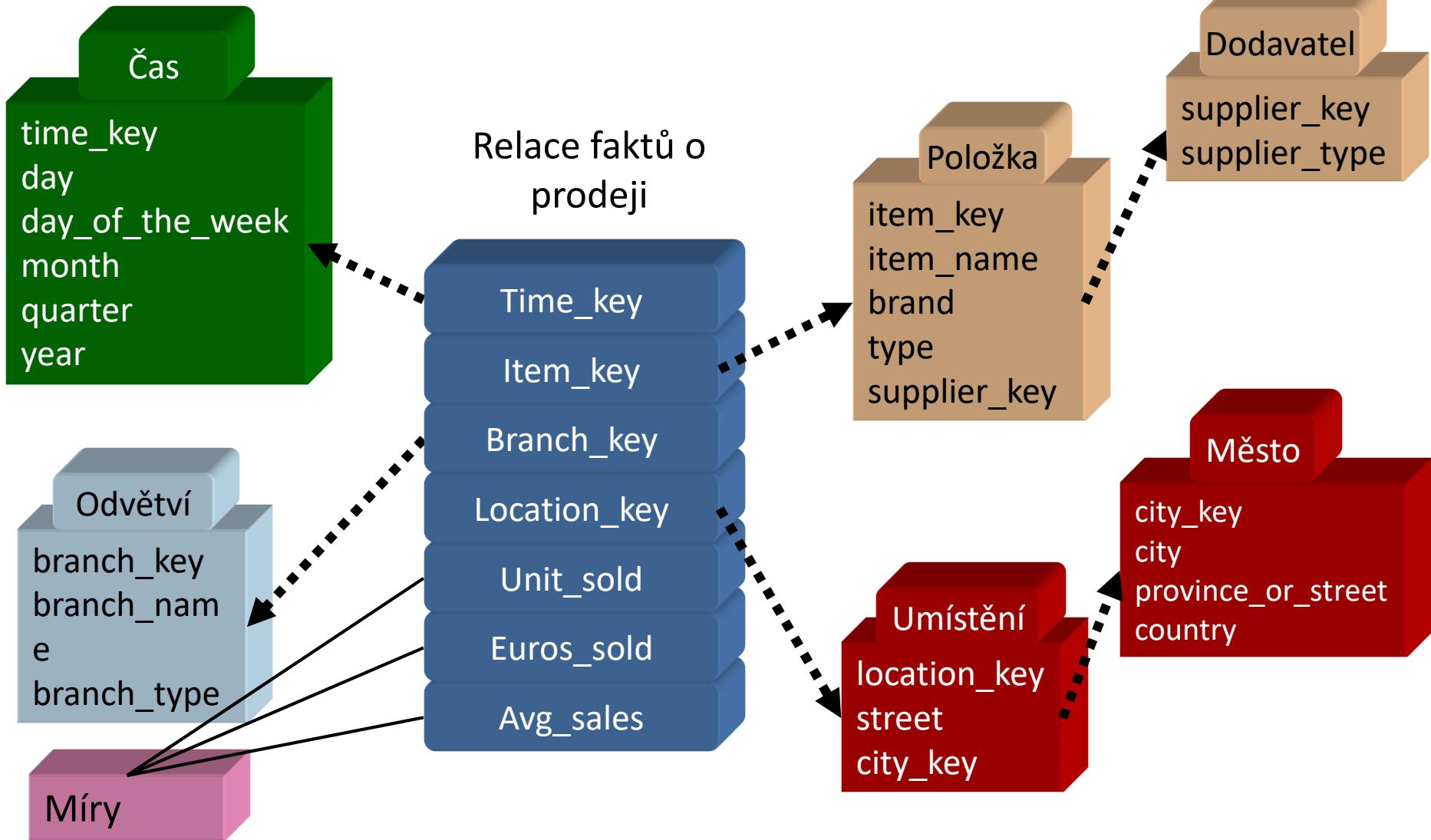
Příklad schématu hvězda - star



Sněhová vločka - snowflake

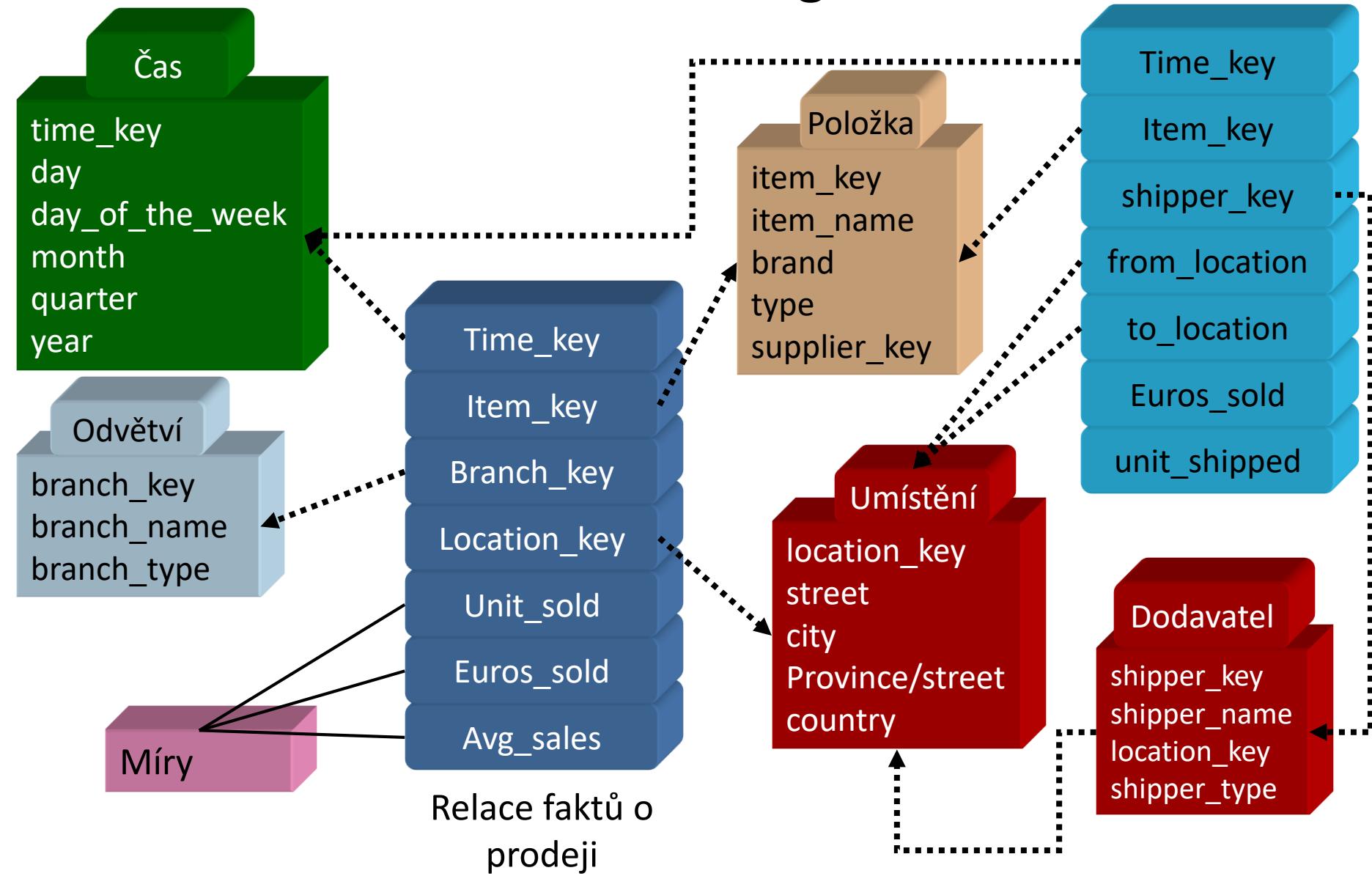
- *Schéma sněhové vločky* poskytuje zjemnění hvězdicového schématu tak, že **hierarchie dimenzí** je explicitně reprezentována normalizováním relací dimenzí. To vede k výhodné údržbě relací dimenzí.
- Nicméně nenormalizovaná struktura dimenzionální rewe hvězdicovém schématu může být mnohem vhodnější pro procházení dimenzemi.

Příklad schématu sněhová vločka



Relace faktů o
dodávce

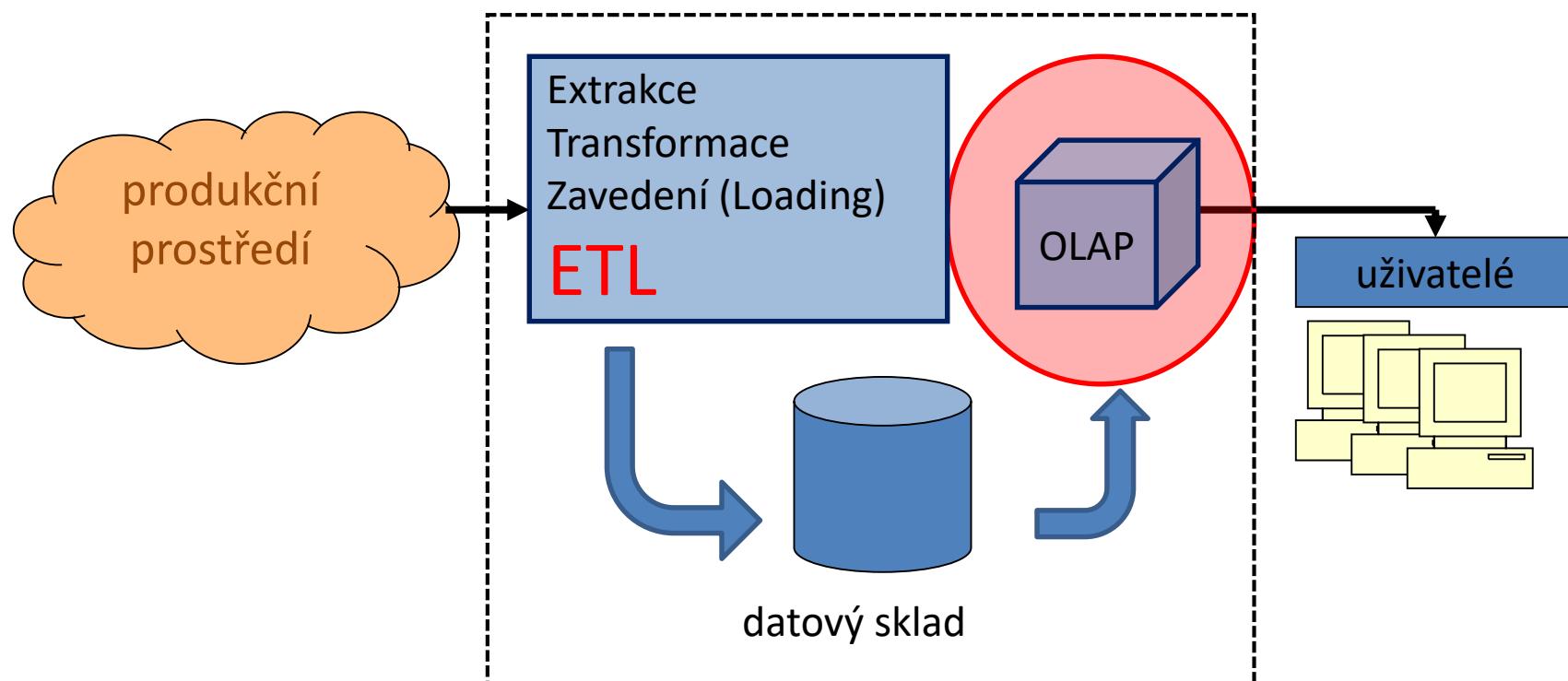
Příklad konstelace faktů - galaxie



OLAP

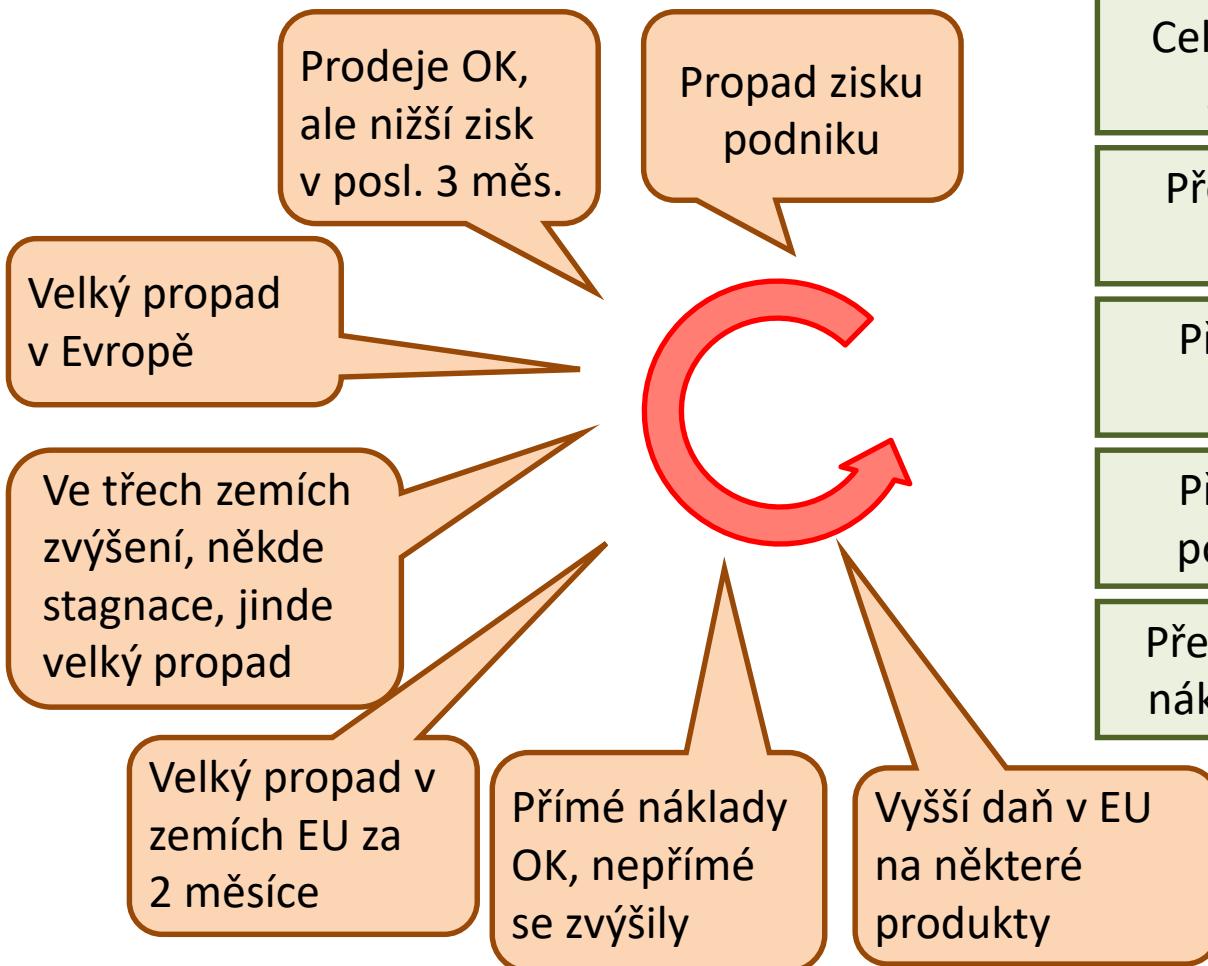
Analýza OLAP

- Analytické zpracování údajů v datovém skladu podle uživatelských dotazů



Příklad komplexní analýzy

Myšlenkový pochod při analýze



Sekvence dotazů při této analýze

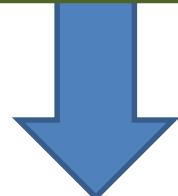
Celosvětové měsíční prodeje za posledních 5 měsíců

Přehled měsíčních prodejů po regionech

Přehled prodejů v Evropě po zemích

Přehled prodejů v Evropě po zemích, po produktech

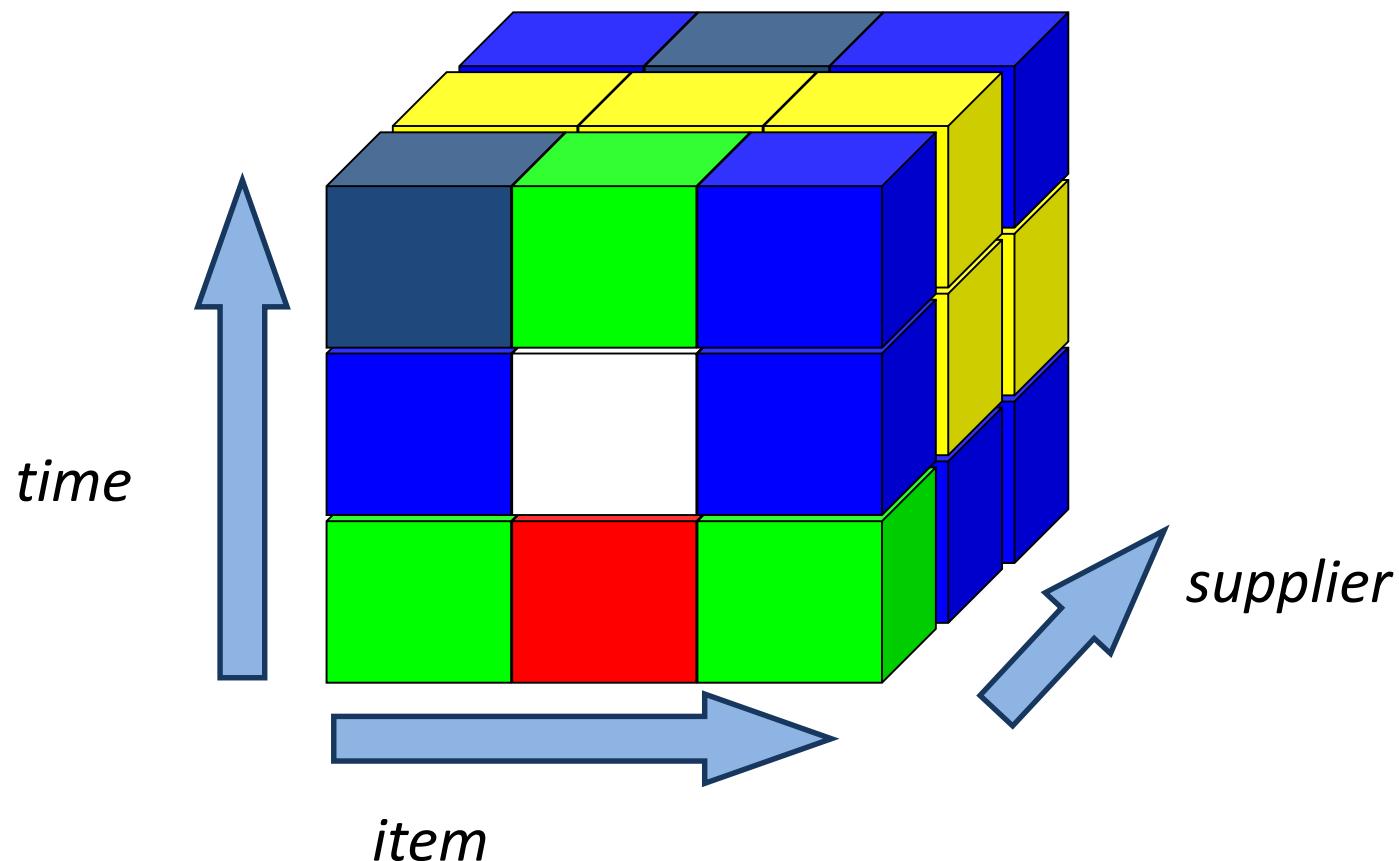
Přehled přímých a nepřímých nákladů v evropských zemích



Požadavky na OLAP systémy (příklady)

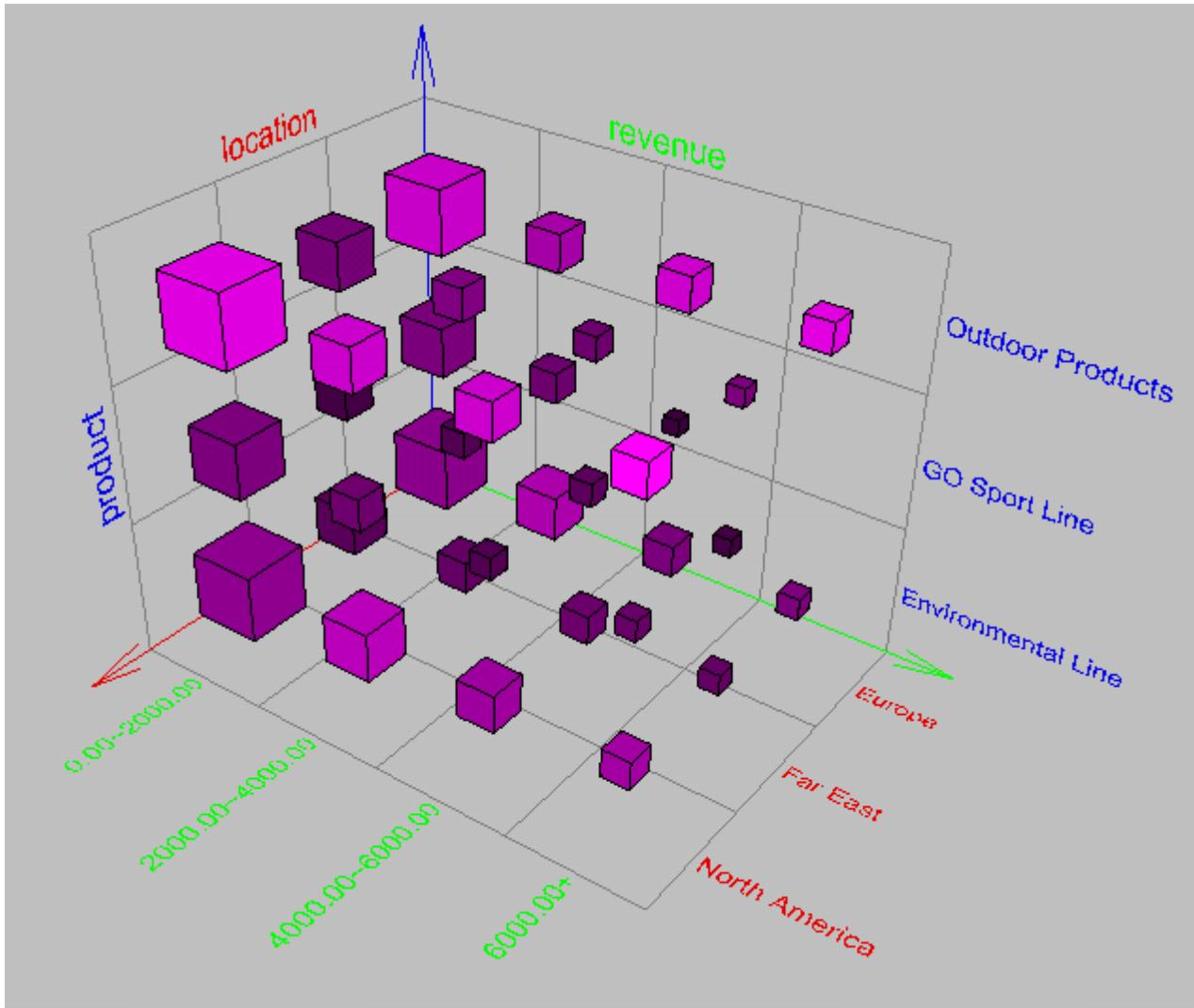
- Poskytování ***agregačních*** funkcí podle hierarchií
- Možnost ***detailního pohledu - zooming*** na data
- Jednoduché kalkulace, např. výpočet zisku (prodeje – náklady)
- Sdílení kalkulací za účelem procentuálního vyjádření vzhledem k celku
- Algebraické rovnice pro výpočet klíčových indikátorů
- Přenos průměrů a procentuálních vyjádření
- Analýza trendů statistickými metodami

Datová kostka 3D příklad



OPERACE NAD DATOVOU KOSTKOU

Prohlížení datové kostky

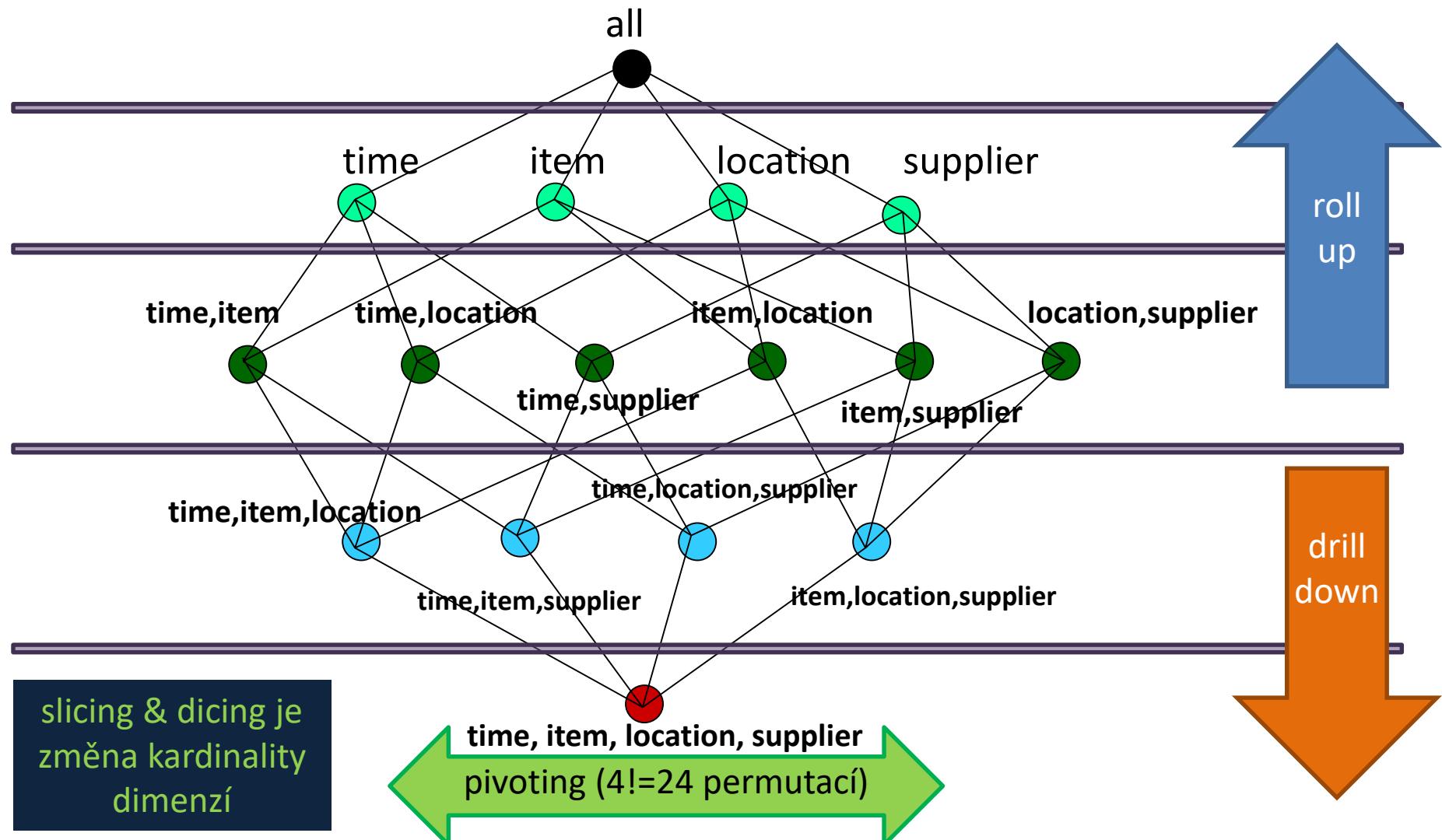


- vizualizace
- operace
- interaktivní manipulace

Operace nad datovou kostkou

- *roll-up* – (vyrolování) vzrůst úrovně agregace
- *drill-down* (zavrtání) – snížení úrovně agregace a zvýšení detailu
- *pivoting* (přetočení) – změna relace R pro uspořádání dimenzí
- *slicing & dicing* (seříznutí) – výběr projekce

Operace nad 4D datovou kostkou



slicing & dicing je
změna kardinality
dimenzi

time, item, location, supplier
pivoting (4!=24 permutací)

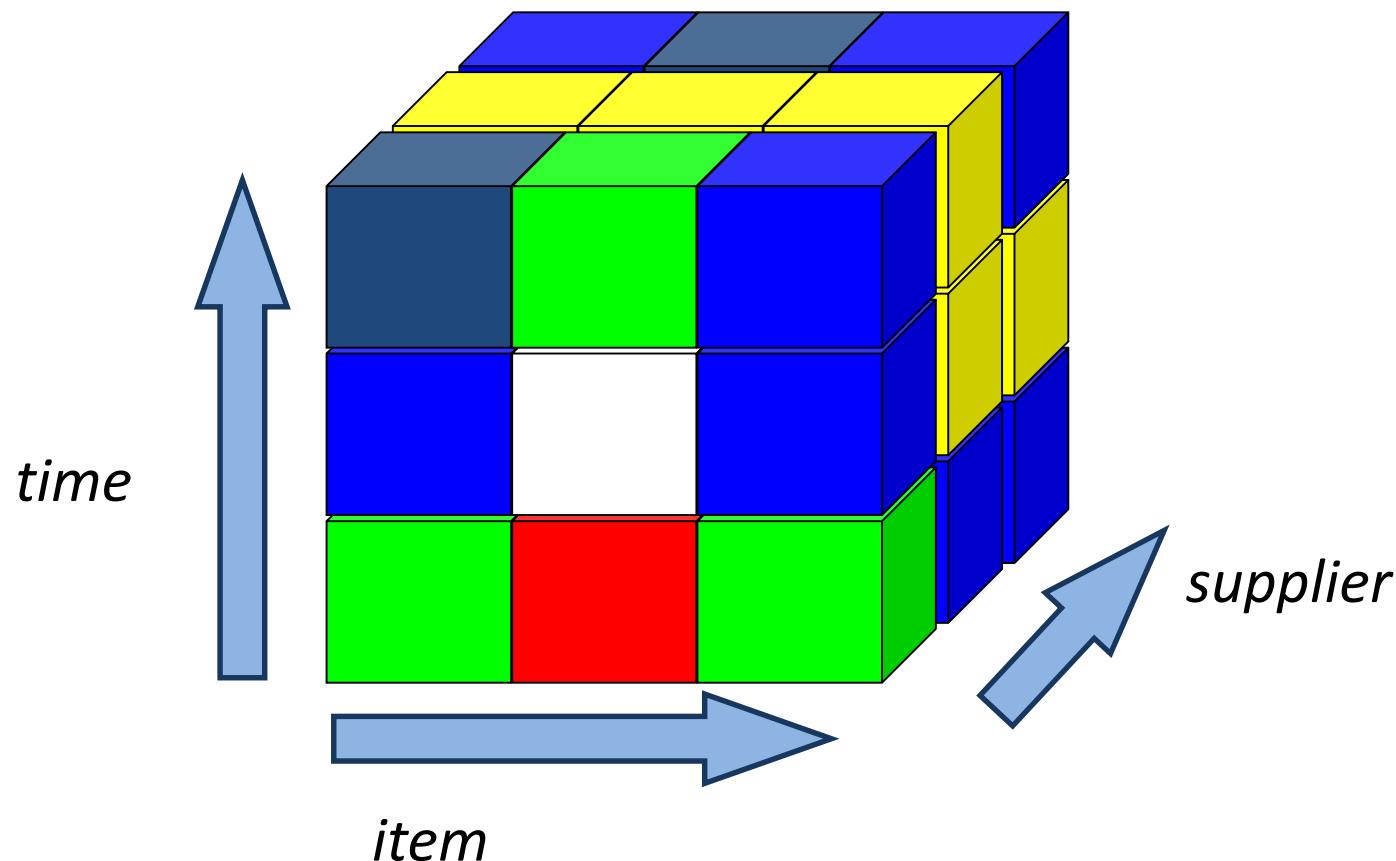
Roll-up

- **Posun o jednu úroveň výše v uspořádání kuboidů**
- **Vstup:** uspořádaná množina m aktivních dimenzí $\{A_1, A_2, A_3, \dots, A_i, \dots, A_{m-1}, A_m\}$, kde $m \geq 1$
- **Výstup:** uspořádaná množina $m-1$ aktivních dimenzí $\{A_1, A_2, A_3, \dots, A_m\}$, tj. A_i bylo deaktivováno.
- Nejčastěji se deaktivuje nejmenší dimenze A_m , tj. z $\{A_1, A_2, A_3, \dots, A_m\}$ vznikne $\{A_1, A_2, A_3, \dots, A_{m-1}\}$

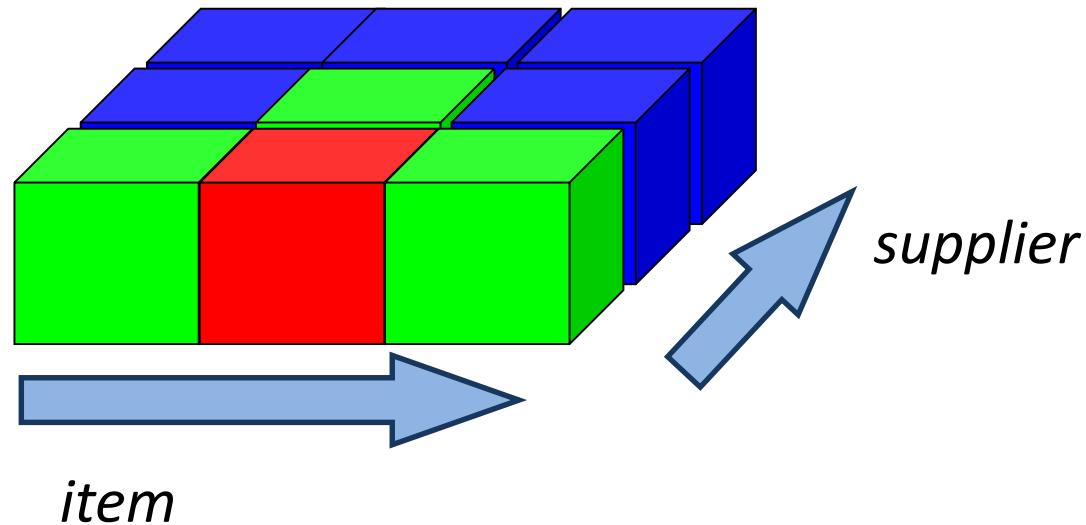
Drill-down

- **Posun o jednu úroveň níže v uspořádání kuboidů**
- **Vstup:** uspořádaná množina m viditelných dimenzí $\{A_1, A_2, A_3, \dots, A_m\}$, kde $m \leq n$
- **Výstup:** uspořádaná množina $m+1$ aktivních dimenzí $\{A_1, A_2, A_3, \dots, A_i, \dots, A_{m-1}, A_m\}$, kde A_i je vybrána z neaktivních D , tak, aby platila definice aktivních dimenzí
- Nejčastější variantou je přidání nejmenší neaktivní dimenze na konec, tedy vznikne $\{A_1, A_2, A_3, \dots, A_m, A_{m+1}\}$
- Pro $m=n$ bude výsledkem **detail** všech hodnot
- Je možné provést i **zavrtání do základního kuboidu**, neboť i jeho fakta jsou agregací detailních hodnot pro všechny aktivní dimenze

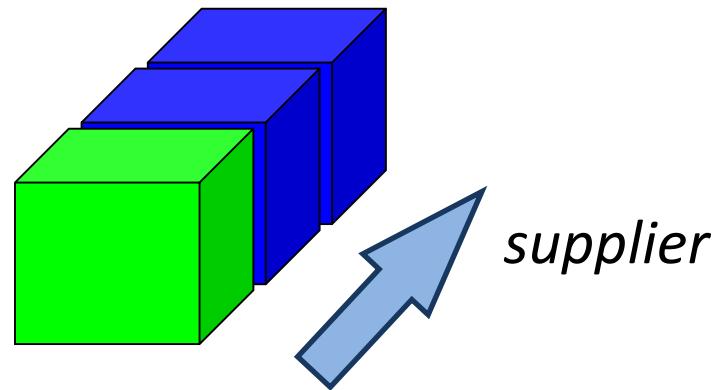
Před operací roll-up



Po operaci roll-up (bez dimenze *time*)

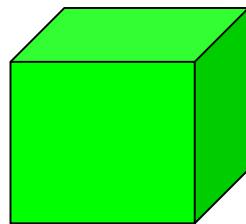


Po operaci roll-up (bez dimenze *item*)



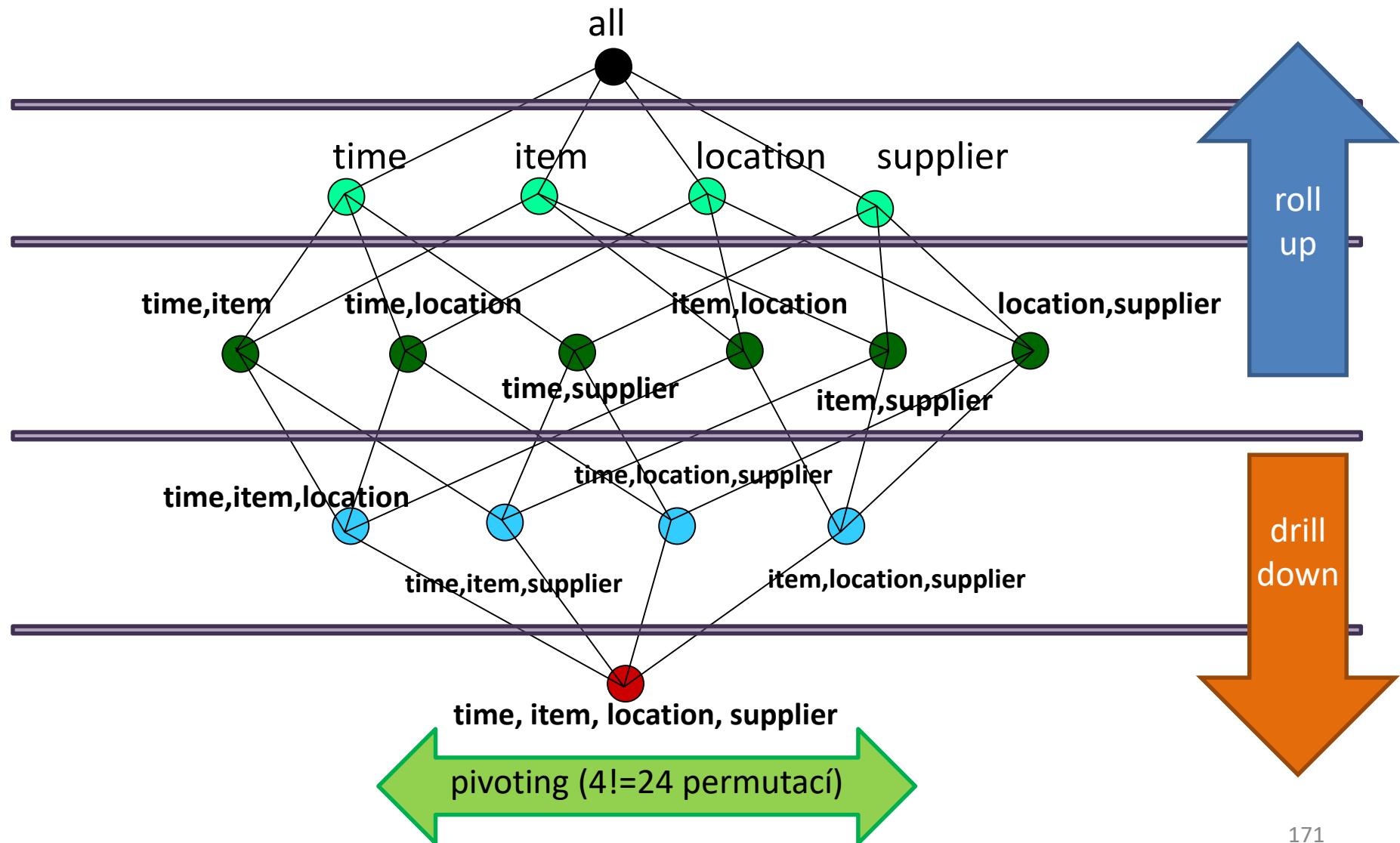
supplier

Po operaci roll-up (bez dimenzí)

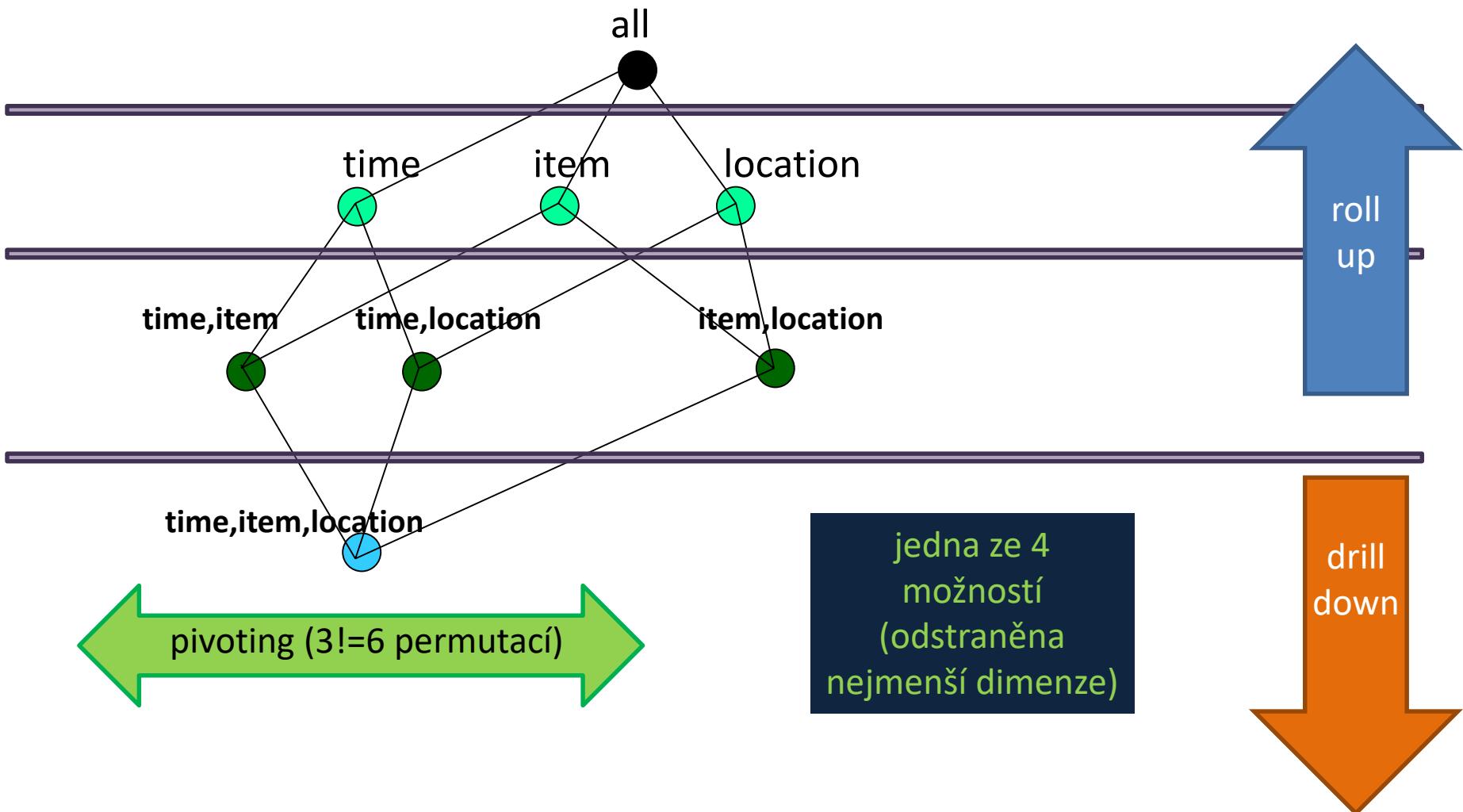


pokud existuje definovaná hodnota agregační funkce,
musí existovat i kostička představující agregovaný fakt

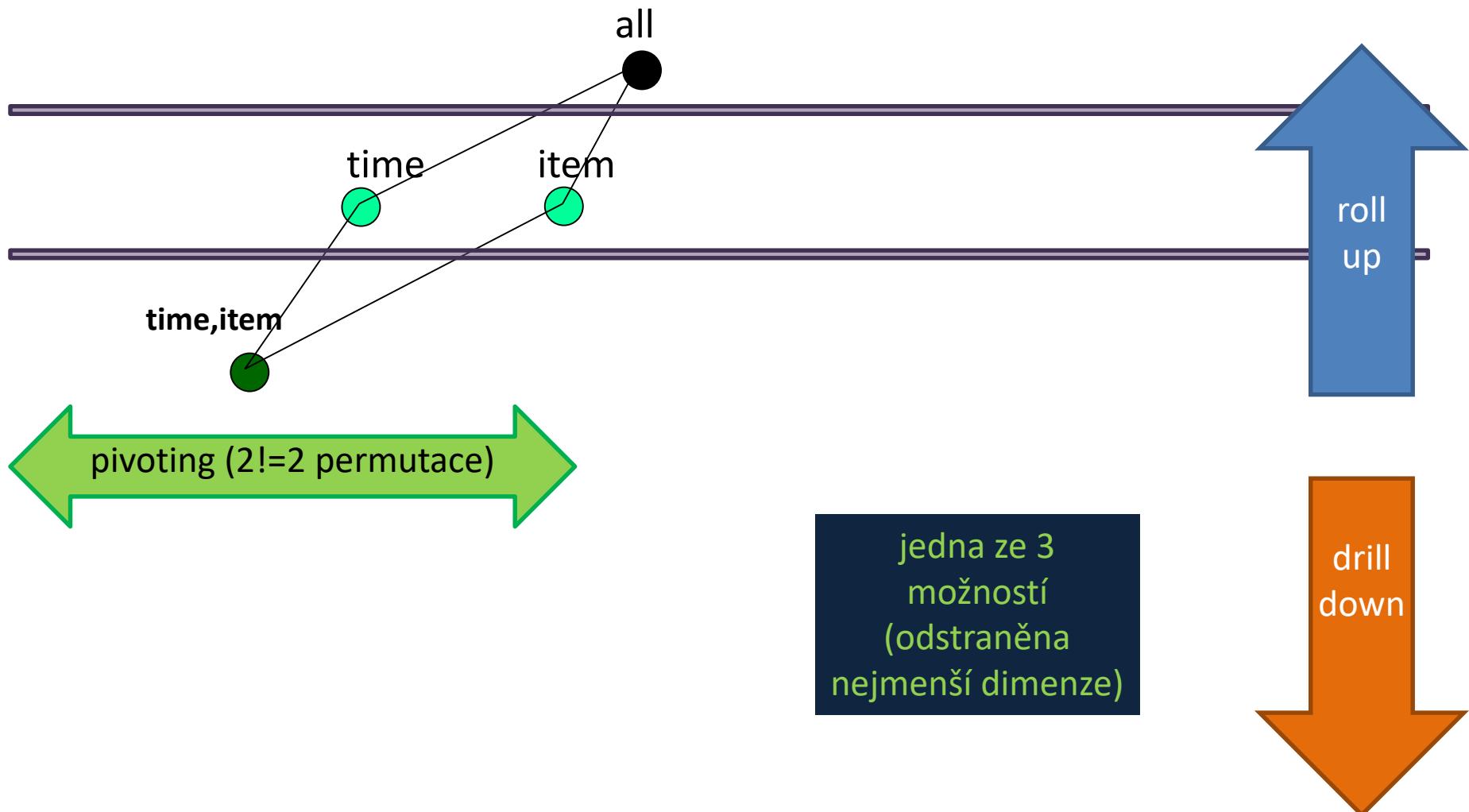
Roll-up nad 4D datovou kostkou



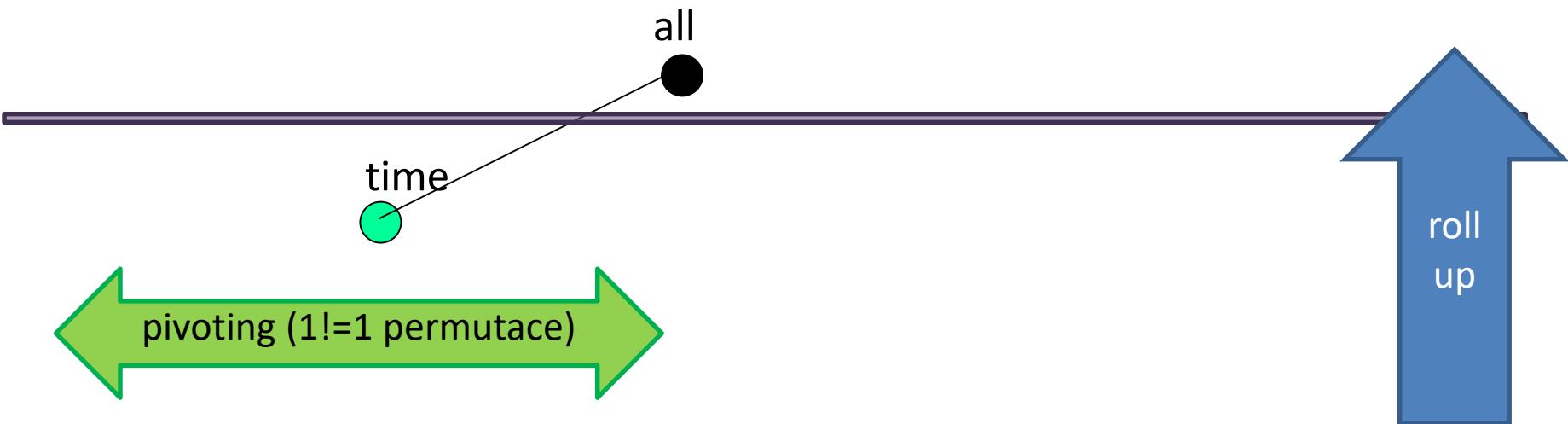
Vyrolovaná datová kostka 3D



Vyrolovaná datová kostka 2D

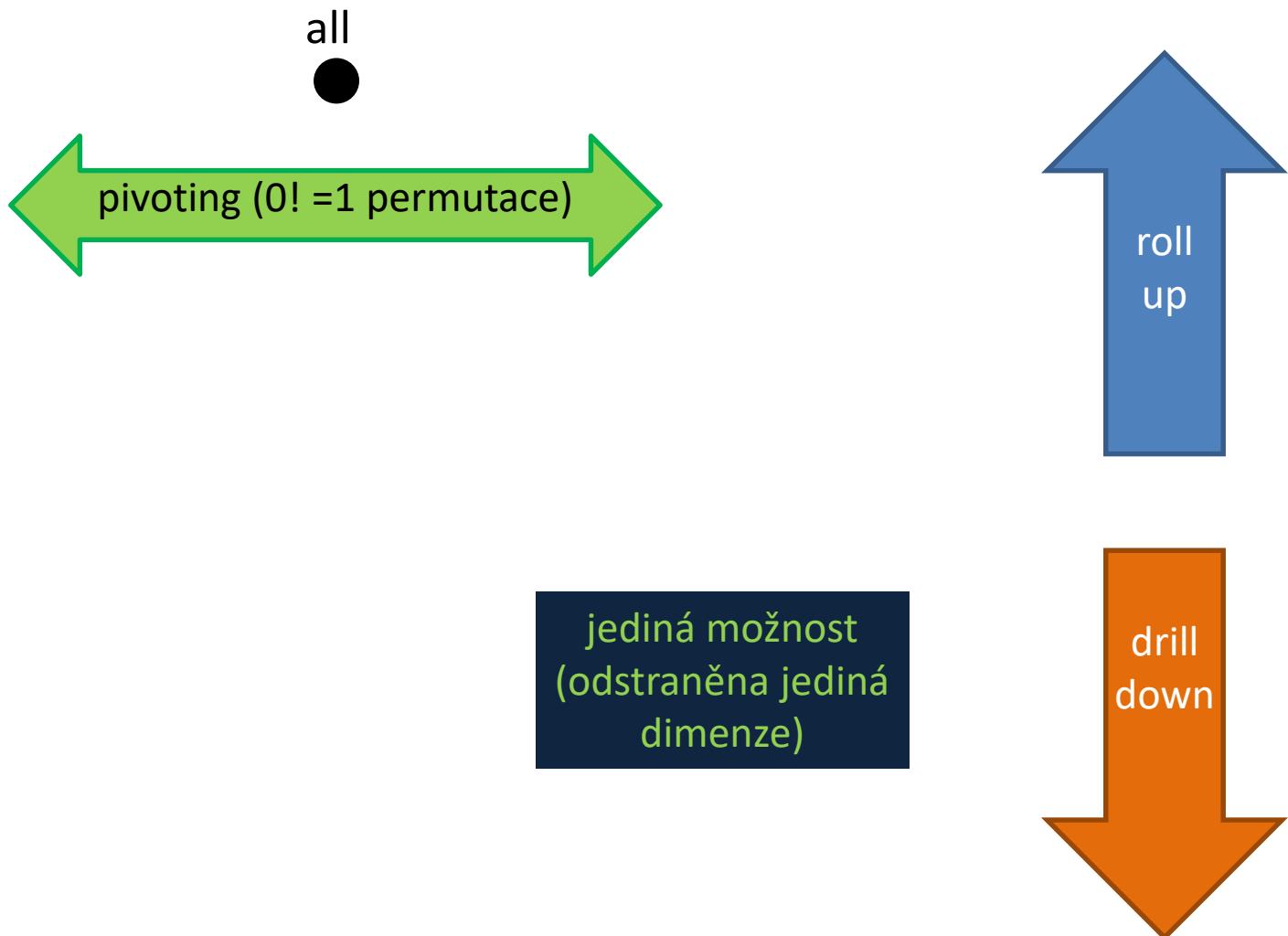


Vyrolovaná datová kostka 1D



jedna ze 2
možností
(odstraněna
nejmenší dimenze)

Vyrolovaná datová kostka 0D



Příklad použití drill-down

LINE	TOTAL SALES
Clothing	\$12,836,450
Electronics	\$16,068,300
Video	\$21,262,190
Kitchen	\$17,704,400
Appliances	\$19,600,800
Total	\$87,472,140

1

aktivní
dimenze je
produkt

2

zaktivizována
dimenze čas

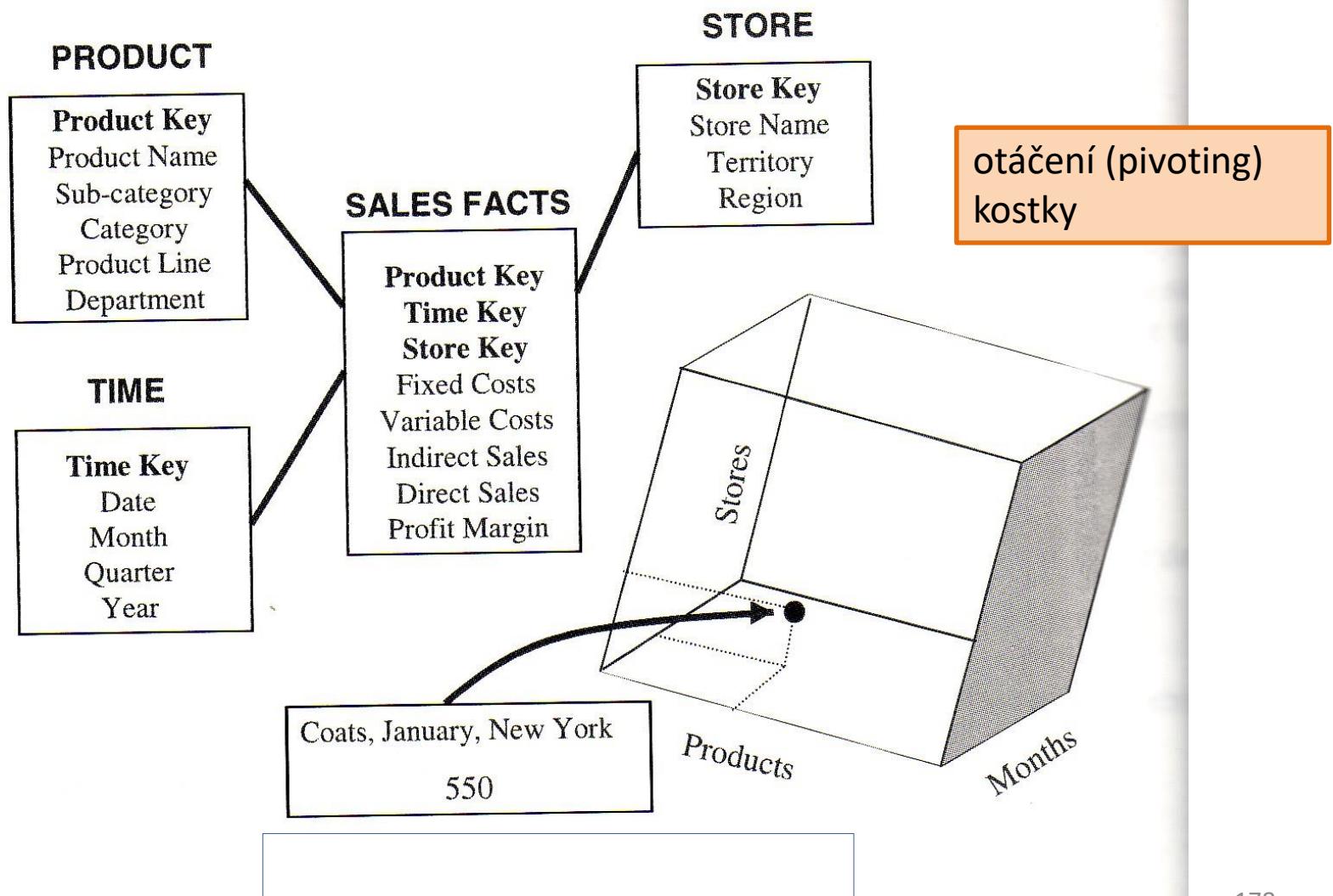
LINE	1998	1999	2000	TOTAL
Clothing	\$3,457,000	\$3,590,050	\$5,789,400	\$12,836,450
Electronics	\$5,894,800	\$4,078,900	\$6,094,600	\$16,068,300
Video	\$7,198,700	\$6,057,890	\$8,005,600	\$21,262,190
Kitchen	\$4,875,400	\$5,894,500	\$6,934,500	\$17,704,400
Appliances	\$5,947,300	\$6,104,500	\$7,549,000	\$19,600,800
Total	\$27,373,200	\$25,725,840	\$34,373,100	\$87,472,140

mírou je
objem
prodejů

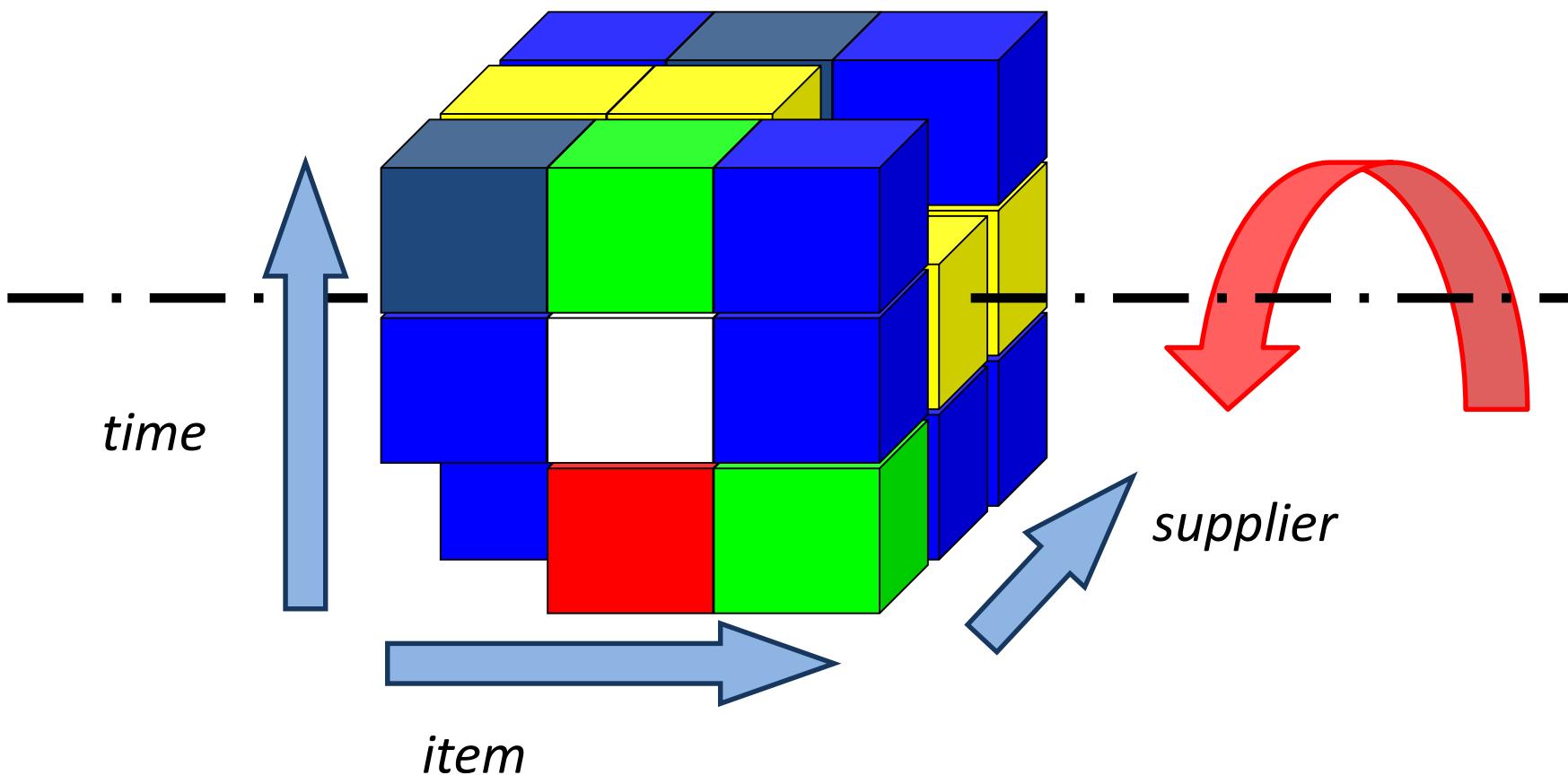
Pivoting

- **Změna uspořádání R nad stejnou množinou dimenzí**
- **Vstup:** uspořádaná množina dimenzí $\{D_1, D_2, D_3, \dots, D_n\}$, kde uspořádání je jedna z možných relací R , kterých je $n!$
- **Výstup:** uspořádaná množina dimenzí $\{D_{x1}, D_{x2}, D_{x3}, \dots, D_{xn}\}$ a obecně jiná relace uspořádání R (jedna z možných $n!$)
- Jde o **otočení jedné ze stěn kostky k sobě**, proto **pivoting**.

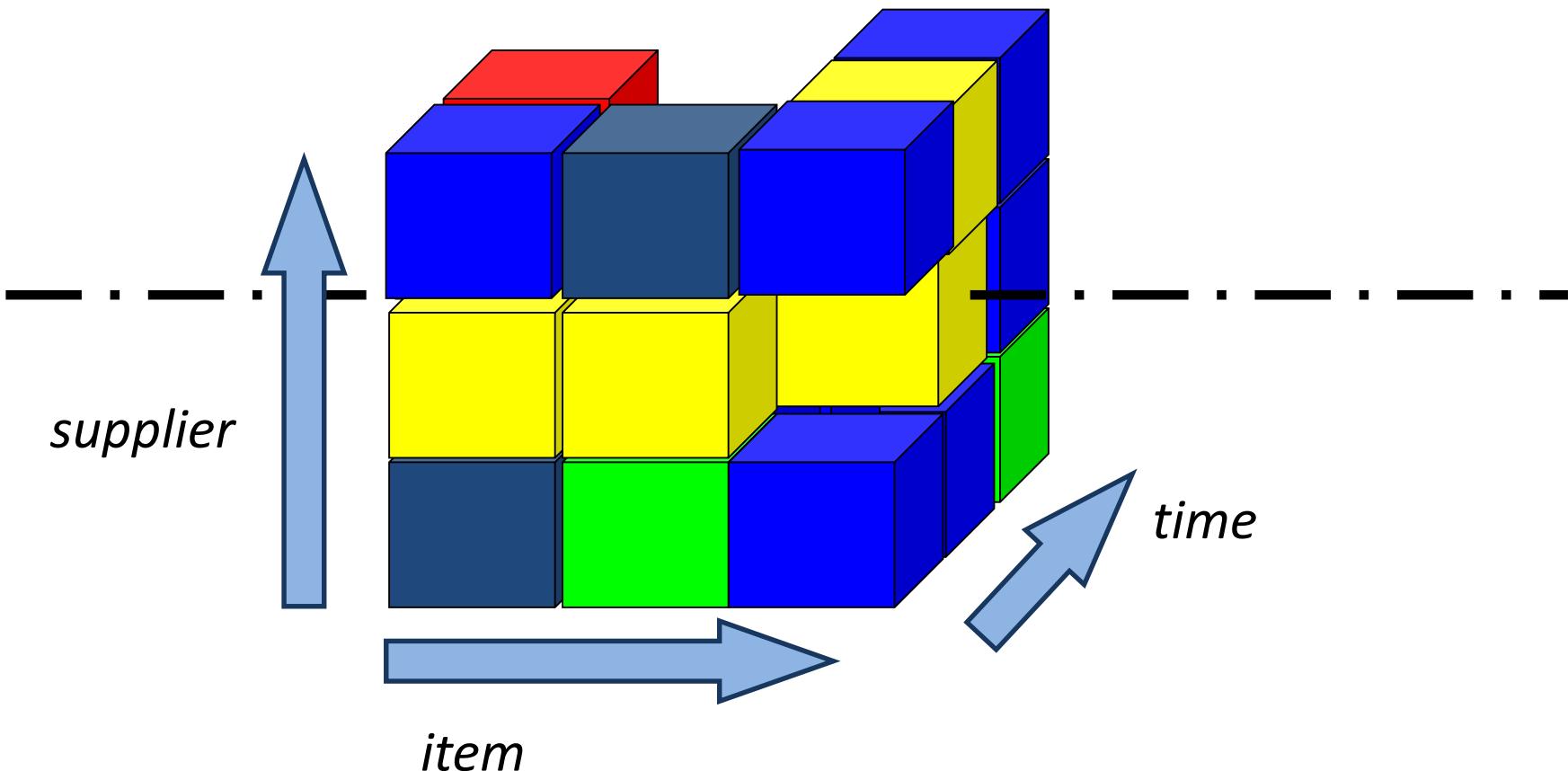
Příklad otáčení



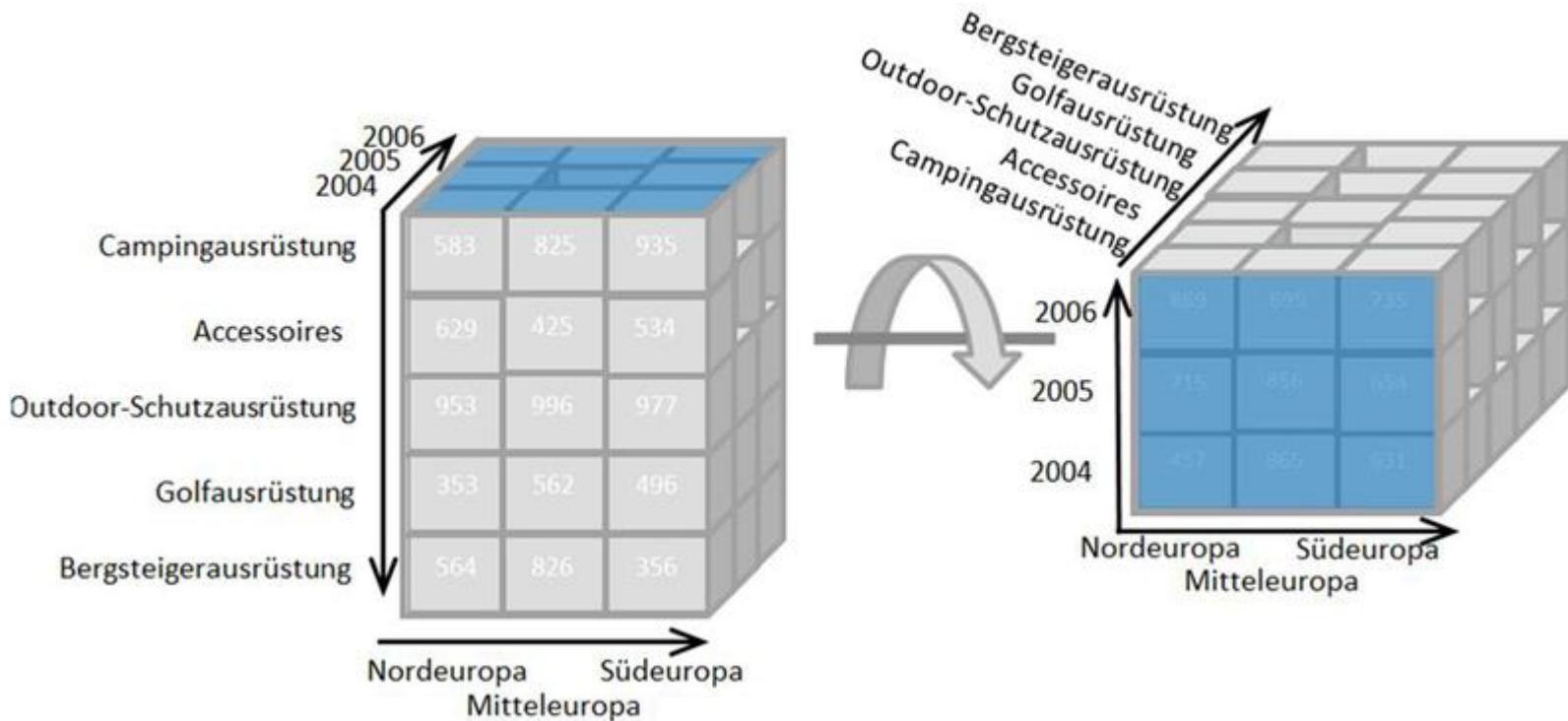
Před operací pivoting



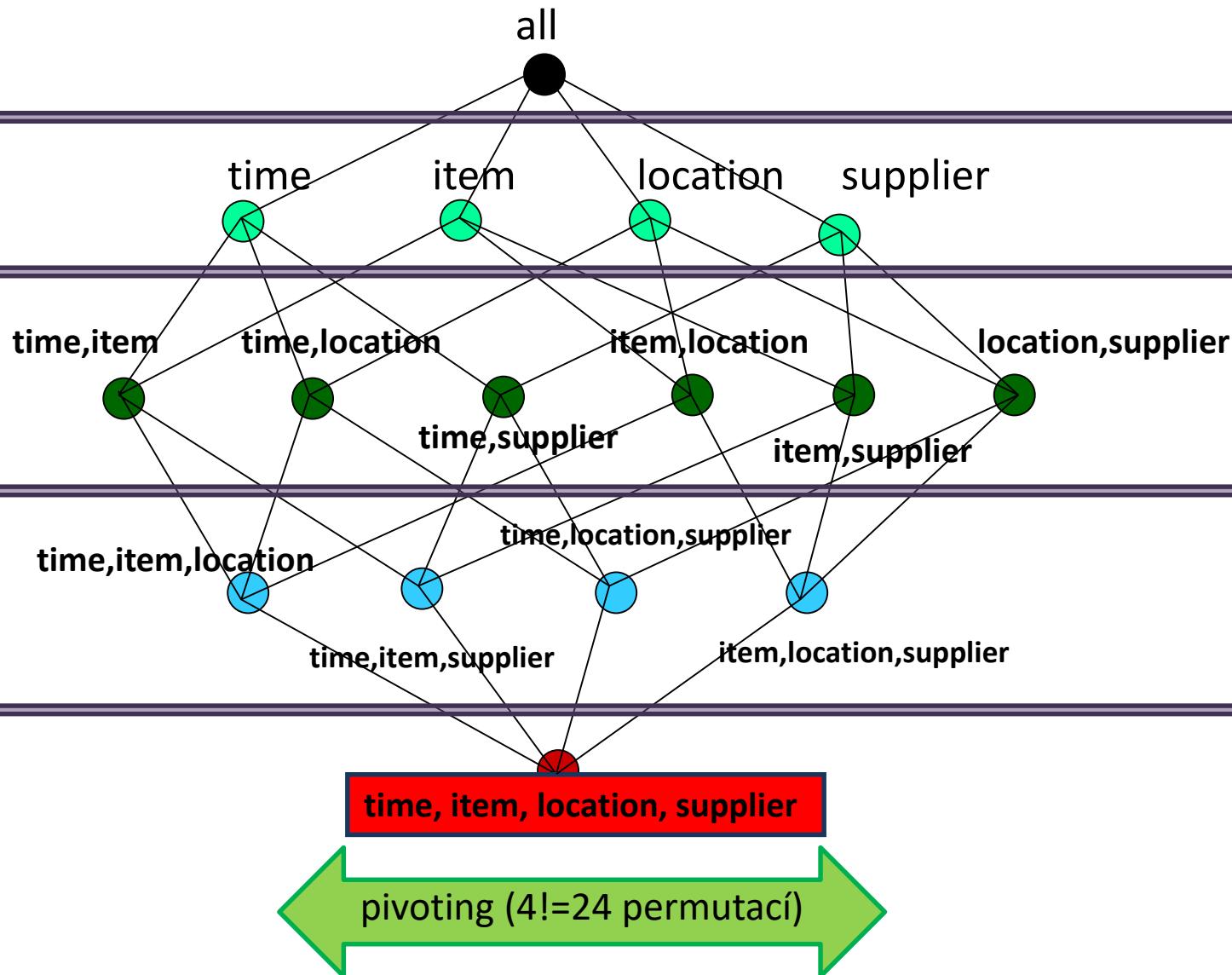
Po operaci pivoting



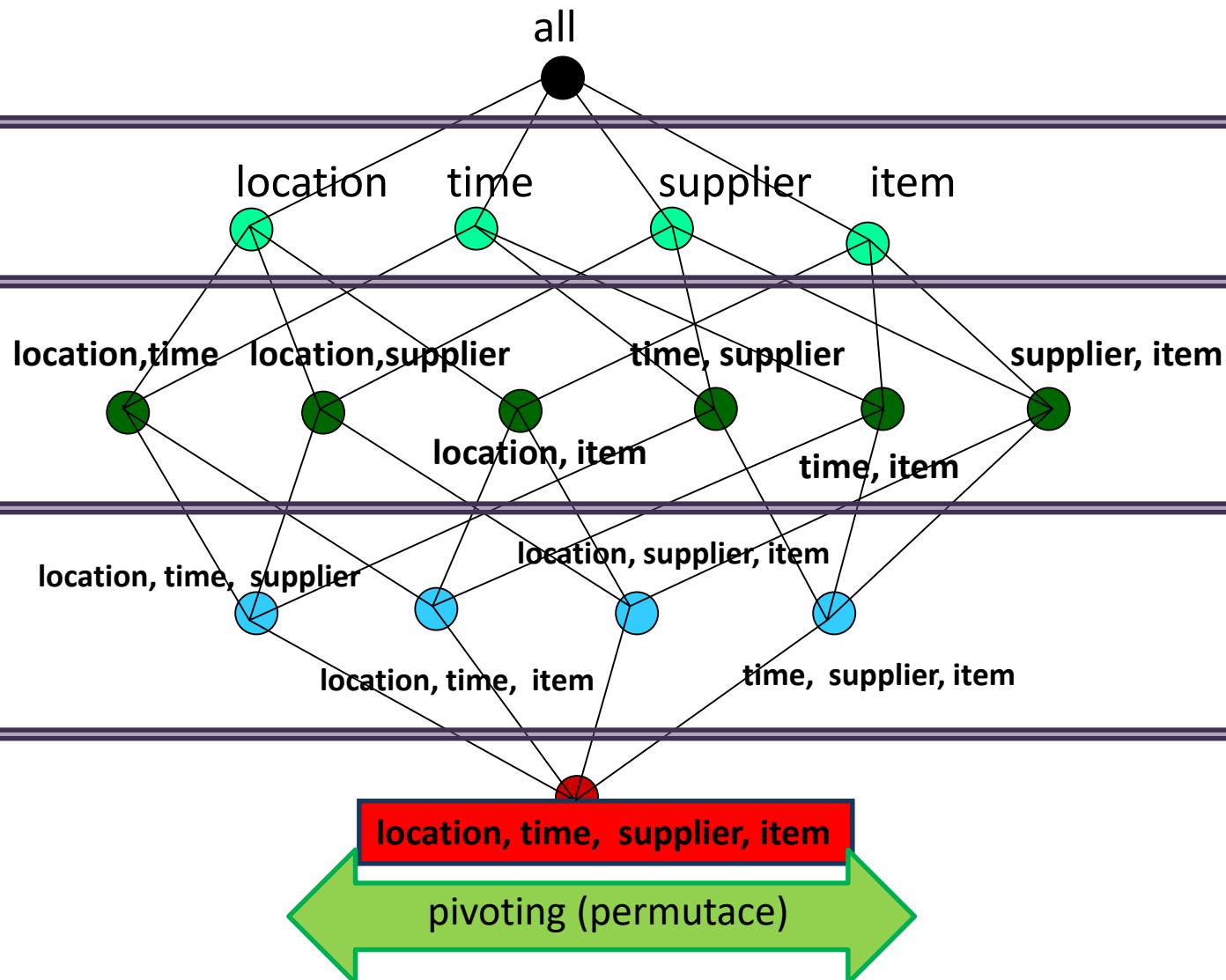
Pivoting



Původní relace R



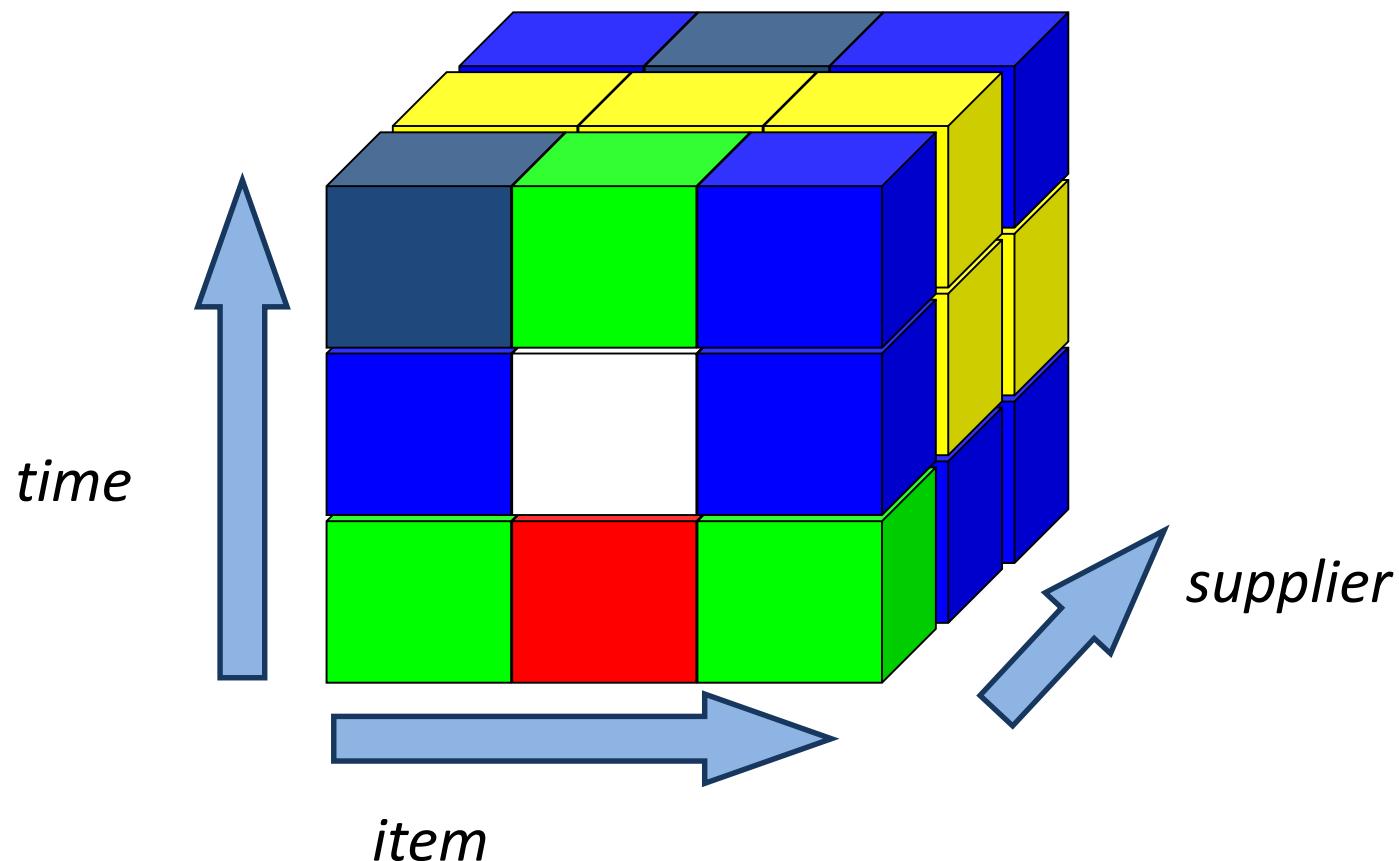
Otočená datová kostka



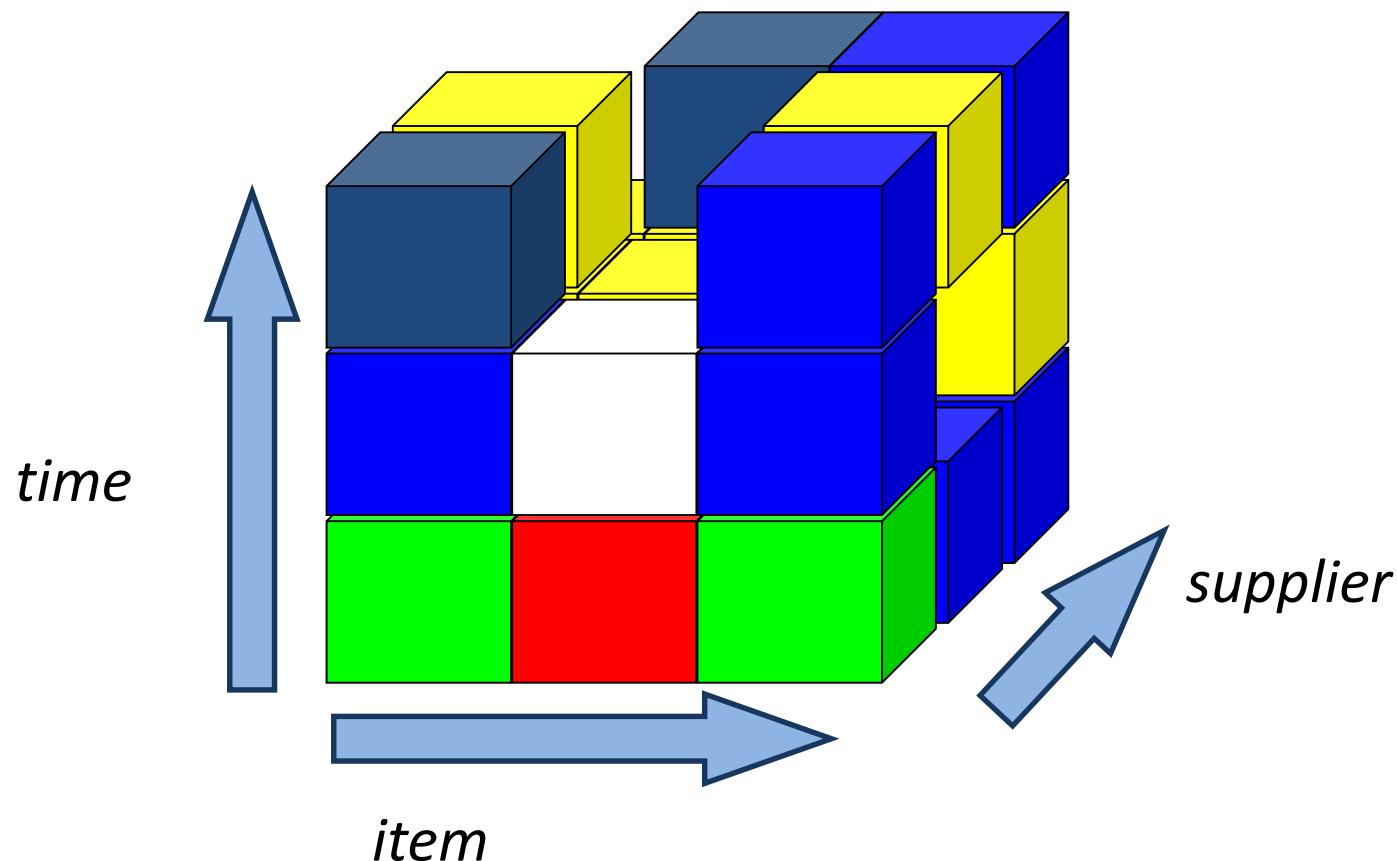
Slicing and Dicing

- **Změna skutečné kardinality jedné nebo více dimenzí**
- **Vstup:** uspořádaná množina dimenzí $\{D_1, D_2, D_3, \dots, D_n\}$, kde $k_1, k_2, k_3, \dots, k_i, \dots, k_n$ jsou skutečné kardinality jednotlivých dimenzí
- **Výstup:** uspořádaná množina dimenzí $\{D_1, D_2, D_3, \dots, D_n\}$ a obecně jiné $k_1, k_2, k_3, \dots, I_i, \dots, k_n$, kde $k_i <> I_i$
- Změnu lze provést výběrem, nastavením **filtru** ve **tvaru predikátu** apod.
- Výsledek ovlivňují i **filtry neaktivních dimenzí**

Před operací slicing and dicing



Po operaci provedené výběrem



VIZUALIZACE MULTIDIMENZIONÁLNÍ KOSTKY A OLAP OPERACÍ NA 2D PRŮMĚTNĚ

Kolekce struktur

- Nelze zobrazovat $m > 3$, ale obecně stejně zobrazujeme na 2D průmětnu
- Funkce $g(A_1 \times A_2 \times A_3 \times \dots \times A_m) = F$ je vlastně kolekcí struktur, kterou vizualizujeme **tabulkou**

**collection of
structure
properties**

A_1 : jednoduchý datový typ₁

A_2 : jednoduchý datový typ₂

...

A_m : jednoduchý datový typ_m

F : agregovatelný datový typ

end structure

Tabulka

D:\DATAFIX\Vema\EKOS1700 FAK\Vema TH

Soubor Úpravy Stránka

VMMI0102 - Fakturační případy vydané

Servis Viditelnost zpět Náhled tisku Přechody Verze

Největší dlužníci

Výběr	1-12	Firma	Nákl. stř. nejvyšší	Nákl. stř. vyšší	Nákl. stř. základní	Celkem	Zbývá
		>40000000 <31.10 1					>0.00
1	45860951-Kristian s.r.o.	24.06.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	61 244.00	61 244.00
2	42035709-MEDIK s.r.o.	20.05.01	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	50 473.90	50 473.90
3	46218487-BALLSTREET	04.03.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	59 187.78	59 187.78
4	46471219-Johansson s.r.o.	18.04.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	7 870.45	7 870.45
5	47952425-A B S tech.činnosti	22.04.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	5 390.00	5 390.00
6	42587912-BSIO a.s.	30.03.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	3 856.70	3 856.70
7	47526869-AKO s.r.o.	30.03.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	18 100.00	18 100.00
8	99990001-Kutálek Petr, Ing.	15.01.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	1 428.00	1 428.00
9	47935469-Computer office	06.02.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	1 439.60	1 439.60
10	45872463-Renovace a.s.	07.04.04	1-Brno	3-výroba	31-výroba nábytku	121.40	121.40
11	47952425-A B S tech.činnosti	20.03.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	16.50	16.50
12	49385719-Ekolab s.r.o.	30.03.04	1-Brno	9-rozvahové pracoviště	91-rozvahové pracoviště	70 759.90	70 759.90
	12					279 888.23	143 427.23

Ladění++

Tabulka

- pokud máme více (k) funkcí g se stejnými dimenzemi (***galaktické schéma***), lze je vizualizovat v jediné tabulce

collection of

structure

properties

A_1 : jednoduchý datový typ₁

A_2 : jednoduchý datový typ₂

...

A_m : jednoduchý datový typ_m

F_1 : fakt- agregovatelný datový typ₁

F_2 : fakt- agregovatelný datový typ₂

...

F_k : fakt- agregovatelný datový typ_k

end structure

Tabulka

- Lze provádět všechny 4 operace:
- ***roll-up*** – ubírání dimenzionálních sloupců zprava i obecně
- ***drill-down*** – přidávání dimenzionálních sloupců napravo i obecně
- ***pivoting*** – změna pořadí dimenzionálních sloupců
- ***slicing & dicing*** – filtry, případně výběr ad hoc např. zaškrťtaváním

Sloupce, řádky a stránky

- omezený způsob vhodný pouze pro $\leq 3D$ kostky

Příklad

- Příklad: zjistit prodeje jednotlivých položek v jednotlivých měsících roku v jednotlivých obchodech
- Výsledek ve kompaktním tvaru kontingenční tabulky s jediným faktem

		Products					
		<u>COLUMNS:</u> PRODUCT dimension					
		Hats	Coats	Jackets	Dresses	Shirts	Slacks
ROWS: TIME dimension	Jan	200	550	350	500	520	490
	Feb	210	480	390	510	530	500
	Mar	190	480	380	480	500	470
	Apr	190	430	350	490	510	480
	May	160	530	320	530	550	520
	Jun	150	450	310	540	560	330
	Jul	130	480	270	550	570	250
	Aug	140	570	250	650	670	230
	Sep	160	470	240	630	650	210
	Oct	170	480	260	610	630	250
	Nov	180	520	280	680	700	260
	Dec	200	560	320	750	770	310

Další příklady dotazů OLAP

- ***Celkové prodeje všech produktů za 5 let***
 - Řádky: roky 2000, 1999, 1998 atd.
 - Sloupce: součet prodejů pro všechny produkty
 - Stránky: jedna stránka pro každý obchod
- ***Porovnej prodeje všech produktů a obchodů mezi roky 1999 a 2000***
 - Řádky: roky 2000, 1999, průměry, rozdíly
 - Sloupce: jeden sloupec pro každý produkt
 - Stránka: pouze jedna – pro všechny obchody
- ***Porovnej totéž, ale pouze u zlevněných produktů***
 - Řádky: roky 2000, 1999, 1998 atd.
 - Sloupce: jeden sloupec pro jeden produkt, ale pouze vybrané produkty
 - Stránky: jedna stránka pro každý obchod

Vícedimenzionální hyperkostky

- **Problém:** více než tři dimenze
- Uložení v datovém skladu není problém, dimenzí může být libovolný počet
- Problém je se zobrazením výsledků OLAP analýzy – nevystačíme s *řádky, sloupcí a stránkami*

Příklad – zobrazení 4 dimenzí

multidimenzionální struktura				zobrazení ve 3D	
STORE	TIME	PRODUCT	METRICS		
New York	Jan	Hats	Sales	<u>PAGE</u> : Store Dimension	
San Jose	Feb	Coats	Cost	<u>ROWS</u> : Time Dimension	
Dallas	Mar	Jackets		<u>COLUMNS</u> : sloupce kombinované s dvoječkou (dosti nesystematické)	

New York Store

	Hats:Sales	Hats:Cost	Coats:Sales	Coats:Cost	Jackets:Sales	Jackets:Cost
Jan	450	350	550	450	500	400
Feb	380	280	460	360	400	320
Mar	400	310	480	410	450	400

Příklad – zobrazení 6 dimenzí

multidimenzionální struktura

DEMO	PROMO STORE		TIME		PRODUCT METRICS	
Life Style	Type	New York	Jan	Hats	Sales	Sales
	Coupon	San Jose	Feb	Coats	Cost	

HOW DISPLAYED ON A PAGE

PAGE: Demographics & Promotion Dimensions combined

ROWS: Store & Time Dimensions combined

COLUMNS: Product & Metrics combined

Life Style : Coupon

		Hats		Coats	
		Sales	Cost	Sales	Cost
New York	Jan	220	170	270	220
	Feb	190	140	230	180
Boston	Jan	200	160	240	200
	Feb	180	130	220	170

přechod ke kontingeční tabulce

Kontingenční tabulka (Pivot table)

- řádky i sloupce jsou dimenze i hierarchické
- fakta jsou na průsečíku dimenzí
- fakt je proto v zásadě **pouze jediný**, více faktů se v průsečíku nedá zobrazit
- řeší se to opakováním tabulky, záložkami nebo více sloupcí, což se zdá méně přehledné

Kontingenční tabulka

- dimenze lze přesunovat a zaměňovat – pivoting
- zakrýváním hierarchie dimenzí lze zvyšovat a snižovat agregaci – roll-up, drill-down
- všechny sloupce lze zneviditelnit nebo nevyužívat – slice & dice
- prostorově méně náročná, avšak nejsou vidět všechny míry a hierarchie dimenzí může být nepřehledná

Prvotní data ve formě relace

The screenshot shows a Microsoft Excel spreadsheet titled "041031 Kontingenční tabulka.xls". The table consists of 18 rows and 9 columns. The columns are labeled B through J. The first few rows contain headers and some descriptive text. Rows 3 through 9 contain numerical data. Row 14 is highlighted in orange and contains a single cell with a black border, indicating it is selected. The status bar at the bottom shows "Připraven" and "123".

	B	C	D	E	F	G	H	I	J
1									
2		firma	splatnost	nákl. nejvyšší	nákl. vyšší	nákl. základní	celkem	zbývá	
3		A	1.1.2002	1	11	111	10000	3000	
4		A	1.1.2002	1	11	111	34567	8388	
5		A	1.2.2003	1	11	111	84732	8366	
6		A	12.4.2003	1	11	111	76355	3444	
7		B	1.1.2003	1	11	112	11000	4000	
8		C	1.1.2003	1	12	121	12000	0	
9		D	1.1.2004	2	21	211	13000	5000	
10									
11									
12									
13									
14									
15									
16									
17									
18									

Kontingenční tabulka

The screenshot shows a Microsoft Excel spreadsheet titled "041031 Kontingenční tabulka.xls". The table is a contingency table with the following structure:

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3	Součet z celkem	nákl. nejvyšší	nákl. vyšší	nákl. základní								
4		1										
5		11		Celkem z 11	12	Celke m z 12			2		Celkem z 2	Celkový součet
6	splatnost	firma	111	112		121			211			
7	1.1.2002	A	44567		44567		44567					44567
8	Celkem z 1.1.2002		44567		44567		44567					44567
9	1.1.2003	B		11000	11000		11000					11000
10		C				12000	12000	12000				12000
11	Celkem z 1.1.2003			11000	11000	12000	12000	12000	23000			23000
12	1.2.2003	A	84732		84732		84732					84732
13	Celkem z 1.2.2003		84732		84732		84732					84732
14	12.4.2003	A	76355		76355		76355					76355
15	Celkem z 12.4.2003		76355		76355		76355					76355
16	1.1.2004	D							13000	13000	13000	13000
17	Celkem z 1.1.2004								13000	13000	13000	13000
18	Celkový součet		205654	11000	216654	12000	12000	228654	13000	13000	13000	241654
19												
20												
21												
22												
23												
24												
25												
26												

Pivoting

- operace otočení provede přesun dimenzí mezi souřadnicemi x a y případně změní jejich pořadí v hierarchii

Pivoting

Microsoft Excel - 041031 Kontingenční tabulka.xls

The screenshot shows a Microsoft Excel spreadsheet titled "041031 Kontingenční tabulka.xls". The window includes a menu bar with Czech labels like "Soubor", "Úpravy", "Zobrazit", etc., and a toolbar with various icons. The main area displays a pivot table with the following structure:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1															
2															
3	Součet z celkem			firma	splatnost										
4	nákl. nejvyšší	nákl. vyšší	nákl. základní		1.1.2002	1.2.2003	12.4.2003	Celkem z A	B	Celkem z B	C	Celkem z C	D	Celkem z D	Celkový součet
5	1	11	111		44567	84732	76355	205654	11000	11000					205654 11000
6			112												
7															
8		Celkem z 11			44567	84732	76355	205654	11000	11000					216654
9		12	121												12000
10		Celkem z 12													12000
11	Celkem z				44567	84732	76355	205654	11000	11000	12000	12000	12000		228654
12	2	21	211												13000 13000
13		Celkem z 21													13000 13000
14	Celkem z														13000 13000
15	Celkový součet				44567	84732	76355	205654	11000	11000	12000	12000	13000	13000	241654
16															
17															
18															
19															
20															
21															
22															
23															
24															
25															
26															

The status bar at the bottom shows "Připraven" and "123".

Roll-up a drill-down

- V kontingenční tabulce jsou operace zvětšování a zmenšování detailů, které postupně odstraňují nižší patra hierarchických klíčů a počítají zadanou operaci (nejčastěji součtový aggregát)
- Operace ***drill-down*** je opačná k ***roll-up***, tj. zmenšuje detaily.

Roll-up a drill-down

Microsoft Excel - 041031 Kontingenční tabulka.xls

Soubor Úpravy Zobrazit Vložit Formát Nástroje Data Okno Nápověda Acrobat Nápověda – zadejte dotaz

Arial 10 B I U % 000 ,00 ,00 180% 180%

D11

	A	B	C	D	E	F	G	H	I
1									
2									
3	Součet z celkem			firma	splatnost				
4				A	B	C	D	Celkový součet	
5	nákl. nejvyšší	nákl. vyšší	nákl. základní						
6	1			205654	11000	12000		228654	
7	2							13000	13000
8	Celkový součet			205654	11000	12000	13000	241654	
9									
10									
11									
12									
13									
14									

Zdrojová data \ Kontingenční tabulka \ Graf /

Připraven 123

Slicing and dicing

- Je možné zadávat filtry nad dimenzemi

[stáhnout odkaz a pak spustit](#)

GRAFY V MULTIDIMENZIONÁLNÍ KOSTCE

Grafické zobrazení

- K multidimenzionálnímu pohledu na hodnoty je velmi dobré přiřadit transformovaný pohled pomocí grafů

Jednoduchý graf (histogram)

- Kolekce dvojic (x-vodorovná, y-svislá)
- x – výčtový, číselný, rovnoměrný i nerovnoměrný
- y – číselný, měřitelný
- nutno redukovat počet dimenzí a faktů na 1

Koláčový (výsečový) graf - Pie

- Kolekce dvojic (x-označení, y-velikost úhlu výseče)
- x – výčtový, číselný, rovnoměrný i nerovnoměrný
- y – číselný **nezáporný**, úhel = $360^{\circ} \times (\text{hodnota}/\text{součet})$
- Obvyklé vyjádření v procentech (y ve stupních)/3,6
- nutno redukovat počet dimenzí i faktů na 1

Násobný graf (histogram)

- Kolekce dvojic (x -vodorovná, (y_1, y_2, \dots, y_n) - svislá)
- x – výčtový, číselný, rovnoměrný i nerovnoměrný
- y_i – číselný, měřitelný
- nutnost redukovat počet dimenzí na 1, počet faktů není třeba omezovat

Operace kostky u grafů

- Operace nad kostkou silně omezeny:
- *roll-up* – neužívá se
- *drill-down* – speciální provedení bude diskutováno separátně
- *pivoting* – pouze jedna dimenze - nemá smysl
- *slicing & dicing* – filtry, případně výběr ad hoc např. zaškrťtaváním

Kombinovaná operace drill-down u živých grafů

- **živé grafy**
- předpokládáme uspořádání dimenzí $\{D_1, D_2, D_3, \dots, D_n\}$
- pokud klikneme v grafu na hodnotu viditelné dimenze D_i , která je jediná, nastaví se pro dimenzi D_i **filtr rovnosti s kliknutou hodnotou (slicing)** a aktuální dimenzí v grafu se stane dimenze D_{i+1} (**drill-down**). Lze provést pouze pro $i <= n$.

Multi Dimensional eXpressions – dotazy pro multidimenzionální kostku

MDX (XMLA EXECUTE PRO MULTIDIMENZIONÁLNÍ KOSTKY)

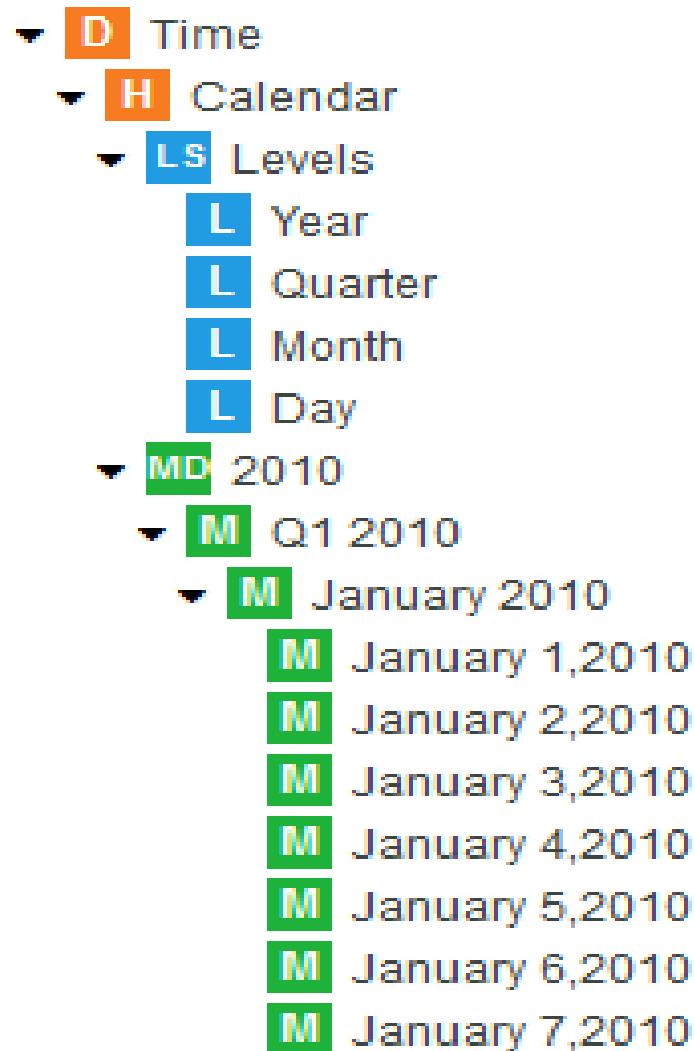
Kostka Sales - dimenze Geography



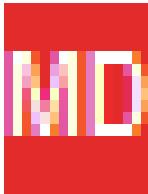
Sales - dimenze Product

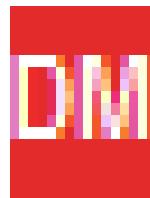
- ▼  **D** Product
- ▼  **H** Prod
- ▼  **LS** Levels
 -  **L** Company
 -  **L** Article
 -  **L** Licence
- ▼  **MD** Crazy Development
- ▼  **M** icCube
 -  **M** Corporate
 -  **M** Partnership
 -  **M** Personal
 -  **M** Startup

Sales - definice dimenze Time hierarchie Calendar



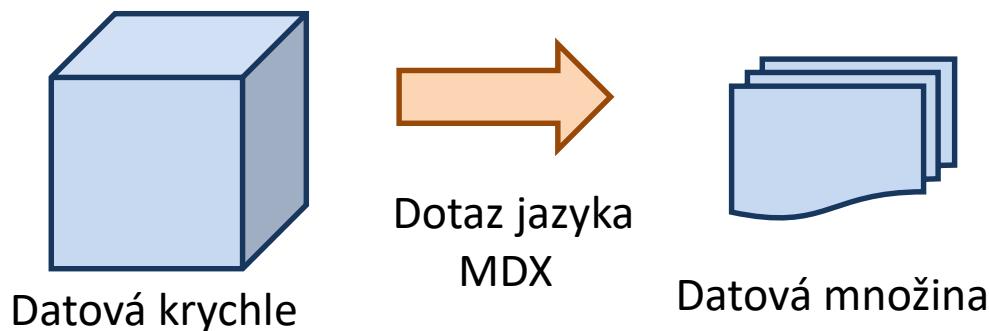
Sales - definice míry Amount

→  **Measures**

 **Amount**

Jazyk MDX

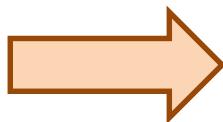
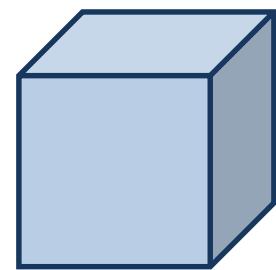
- MDX = **M**ulti**D**imensional **E**Xpressions
- Dotazovací jazyk pro navigaci v multidimenzionálních údajích
- Výsledek dotazu – zobrazitelná tabulka (buďto 2D nebo některé náhradní techniky)



Jazyk MDX - příklad

- Příklad příkazu SELECT

```
SELECT <specifikace_osy> ON COLUMNS,  
      <specifikace_osy> ON ROWS  
FROM  <krychle>  
[WHERE <filtr>]
```



MDX

Datová krychle

	Čechy	Slovensko
	Zisk	Zisk
Mléko	1299343	2145441
Pivo	4545784	1954001
Rohlíky	214147	774755
Víno	7474587	7847111

Množina (set)

- Více výsledných buněk v tabulce

```
{[William Wong],[Andrea Donlon]}\n{([Non-consumable],[USA]),([Beverages],[Mexico])}
```

- Definuje množinu více průsečíků dimenzí

Úplná agregace bez dimenzí

pouze sloupce

```
SELECT [Measures].[Amount] ON COLUMNS  
FROM [Sales]
```

- Výpis jedné hodnoty v jednom sloupci, celková utržená částka za zboží.

Amount
1023

```
SELECT  
    [Measures] . [Amount]  ON COLUMNS  
FROM  
    [Sales]
```

Amount

1023

Uspořádaná n-tice

```
SELECT ([Product].[Prod].[Licence].[Corporate],  
        [Geography].[Geo].[Continent].[America]) ON COLUMNS  
FROM [Sales]
```

pouze sloupce

- Výpis uspořádané n-tice v jednom sloupci, celková utržená částka za zboží.

Corporate
America
768

```
SELECT  
    ([Product].[Prod].[Licence].[Corporate],  
     [Geography].[Geo].[Continent].[America]) ON COLUMNS  
FROM  
    [Sales]
```

Corporate
+ America

768

Řádky i sloupce

```
SELECT {[Product].[Prod].[Licence].[Corporate],  
        [Product].[Prod].[Licence].[Partnership],  
        [Product].[Prod].[Licence].[Personal]} ON COLUMNS,  
        {[Geography].[Geo].[Country].[Spain],  
         [Geography].[Geo].[Country].[United States],  
         [Geography].[Geo].[Country].[Switzerland]} ON ROWS  
FROM [Sales]  
WHERE ([Time].[Calendar].[Year].[2010])
```

- Výpis tabulky s řádky i sloupci
- WHERE specifikuje omezení

	Corporate	Partnership	Personal
Spain			3
United States	768		
Switzerland	144	96	8

Řádky i sloupce

```
SELECT {[Product].[Prod].[Licence].[Corporate],  
        [Product].[Prod].[Licence].[Partnership],  
        [Product].[Prod].[Licence].[Personal]} ON COLUMNS,  
    {[Geography].[Geo].[Country].[Spain],  
     [Geography].[Geo].[Country].[United States],  
     [Geography].[Geo].[Country].[Switzerland]} ON ROWS  
FROM [Sales]  
WHERE ([Time].[Calendar].[Year].[2010])
```

RESULT

	<i>Corporate</i>	<i>Partnership</i>	<i>Personal</i>
⊕ Spain			3
⊕ United States	768		
⊕ Switzerland	144	96	8

Další dimenze

- Až 128
- COLUMNS, ROWS, PAGES, SECTIONS, CHAPTERS, AXIS (číslo)
- osy číselovány od 0
- $0=x$, $1=y$, $2=z$

Řádky a sloupce jako osy

```
SELECT {[Product].[Prod].[Licence].[Corporate],  
        [Product].[Prod].[Licence].[Partnership],  
        [Product].[Prod].[Licence].[Personal]} ON AXIS(0),  
        {[Geography].[Geo].[Country].[Spain],  
         [Geography].[Geo].[Country].[United States],  
         [Geography].[Geo].[Country].[Switzerland]} ON AXIS(1)  
FROM [Sales]  
WHERE ([Time].[Calendar].[Year].[2010])
```

- Výpis tabulky s řádky i sloupci
- WHERE specifikuje omezení

	Corporate	Partnership	Personal
Spain			3
United States	768		
Switzerland	144	96	8

Řádky a sloupce jako osy

```
SELECT { [Product].[Prod].[Licence].[Corporate],  
        [Product].[Prod].[Licence].[Partnership],  
        [Product].[Prod].[Licence].[Personal] } ON AXIS(0),  
{ [Geography].[Geo].[Country].[Spain],  
  [Geography].[Geo].[Country].[United States],  
  [Geography].[Geo].[Country].[Switzerland] } ON AXIS(1)  
FROM [Sales]  
WHERE ([Time].[Calendar].[Year].[2010])
```

RESULT

	<i>Corporate</i>	<i>Partnership</i>	<i>Personal</i>
⊕ <i>Spain</i>			3
⊕ <i>United States</i>	768		
⊕ <i>Switzerland</i>	144	96	8

Fakta

```
SELECT [Measures].MEMBERS ON COLUMNS,  
       [Product].MEMBERS ON ROWS  
FROM [Sales]
```

- Vrací fakta pro každý produkt se součtem na každé úrovni součtů

	RESULT
	<i>Amount</i>
▼ <i>Crazy Development</i>	1023
▼ <i>icCube</i>	1023
<i>Corporate</i>	912
<i>Partnership</i>	96
<i>Personal</i>	15
<i>Startup</i>	

Všechny prvky MEMBERS

```
SELECT Measures.MEMBERS ON COLUMNS,  
 {[Store].[Store State].[CA],  
 [Store].[Store State].[WA]} ON ROWS  
 FROM [Sales]
```

- Vrací fakta pro státy California a Washington

Funkce Children

```
SELECT Measures.MEMBERS ON COLUMNS,  
 {[Geography].[Geo].[Country].[United States].CHILDREN,  
 [Geography].[Geo].[Country].[Switzerland].CHILDREN} ON ROWS  
 FROM [Sales]
```

- pracuje s hierarchií a pro sloupce užije všechny následníky států United States a Switzerland.
- podobně lze užít i *další operace na stromem*

Funkce Children

```
SELECT Measures.MEMBERS ON COLUMNS,  
  { [Geography].[Geo].[Country].[United States].CHILDREN,  
    [Geography].[Geo].[Country].[Switzerland].CHILDREN } ON ROWS  
FROM [Sales]
```

RESULT

	<i>Amount</i>
<i>Los Angeles</i>	
<i>New York</i>	768
<i>San Francisco</i>	
<i>Geneva</i>	128
<i>Lausanne</i>	56
<i>Zurich</i>	64

From jako filtr - poddotazy

- Jiný způsob slice v klauzuli FROM
- Použití poddotazu

```
SELECT  
[Measures].MEMBERS ON 0,  
[Geography].[Geo].MEMBERS ON 1  
FROM  
(SELECT [Time].[Calendar].[Year].[2010] ON 0 FROM [Sales])
```

From jako filtr - poddotazy

```
SELECT  
    [Measures].Members ON 0,  
    [Geography].[Geo].Members ON 1  
FROM  
    (SELECT [Time].[Calendar].[Year].[2010] ON 0 FROM [Sales])
```

	Amount
All Regions	1023
America	768
Canada	
Quebec	
Toronto	
Mexico	
Mexico	
United States	768
Los Angeles	
New York	768
San Francisco	
Venezuela	
Caracas	
Europe	255
France	4
Paris	4
Spain	3
Barcelona	2
Madrid	1
Valencia	
Switzerland	248
Geneva	128
Lausanne	56
Zurich	64

Efektivní výpočet datových kostek

- *Materializace datové kostky*
- Materializace je *vypočtení aggregačních hodnot předem*
- Materializace každého kuboidu (*plná materializace*), žádného (*bez materializace*) nebo jen některých (*částečná materializace*)
- Výběr podkostek, které se budou materializovat
- Na základě velikosti, sdílení nebo frekvence přístupů

KONEC