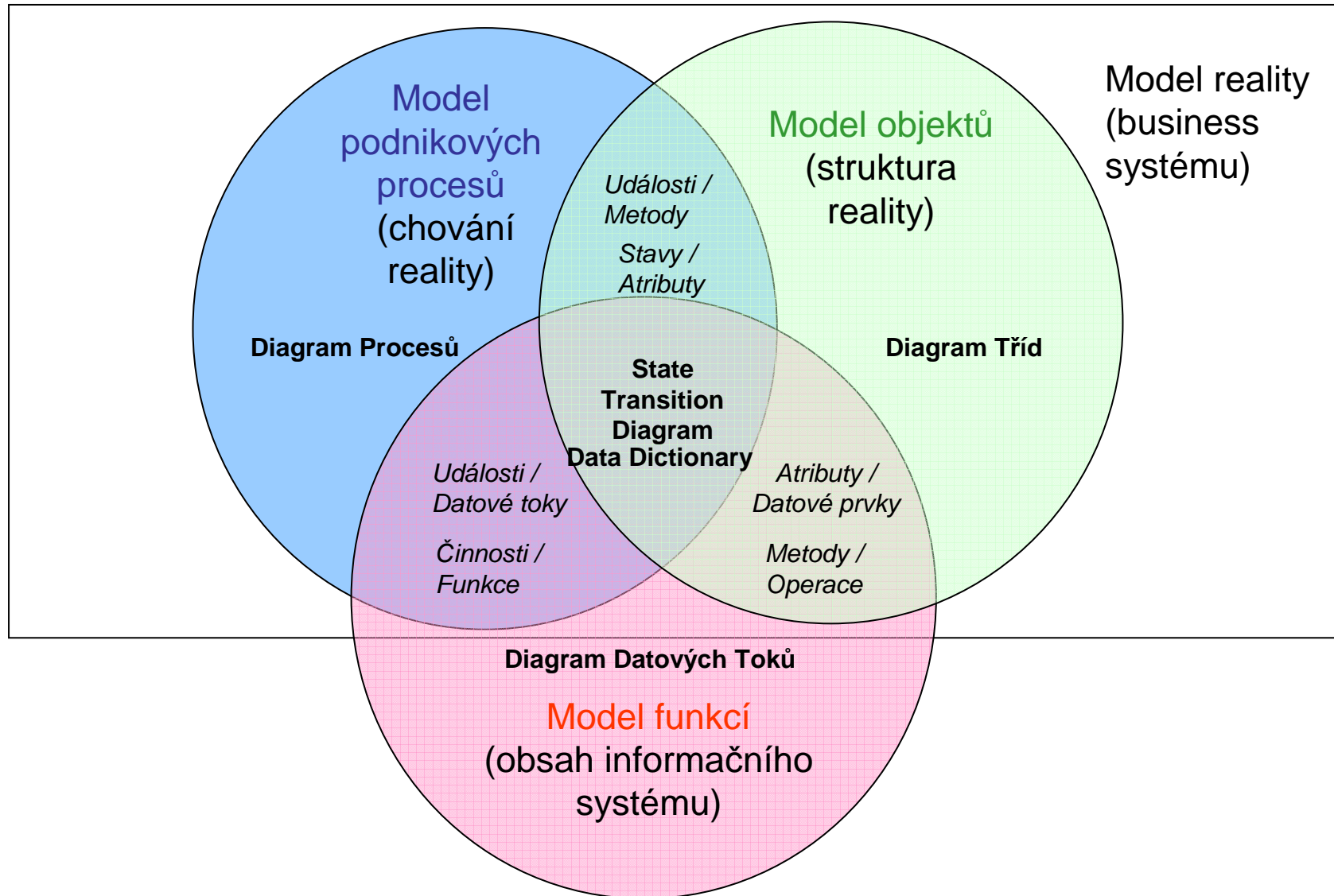
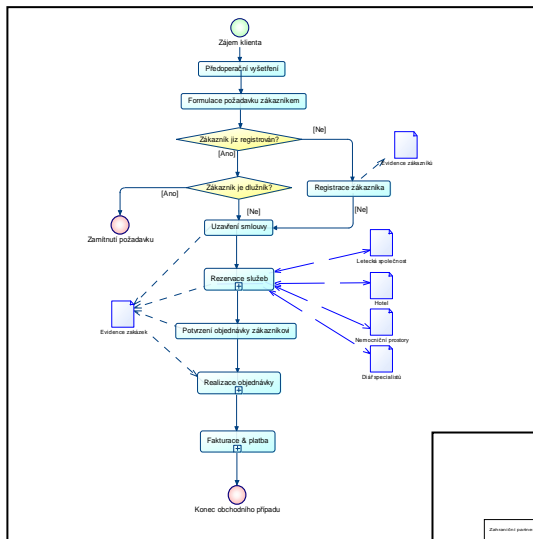


# Přehled analytických modelů



# Přehled analytických modelů

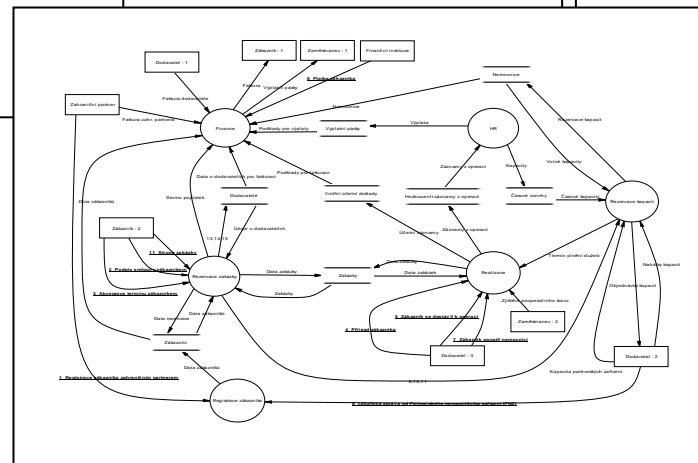
## Model podnikových procesů (Process Diagram)



Údlosti, akce a jejich kontext

Produkty, vstupy, výstupy,  
aktéři, business omezení  
procesů (životní cykly  
objektů)

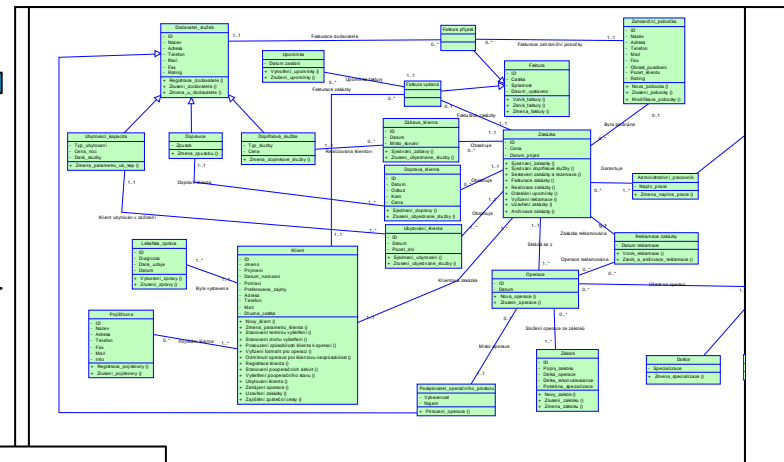
Účelové kombinace ŽC objektů,  
kontext chování objektů



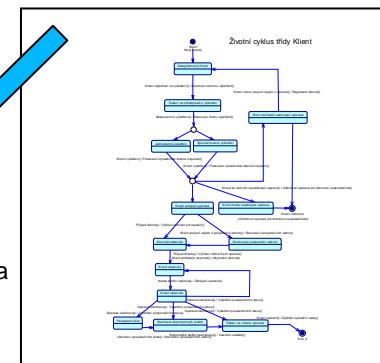
## Model funkcí (Data Flow Diagram)

## Model objektů

## (Class Diagram, State Charts)



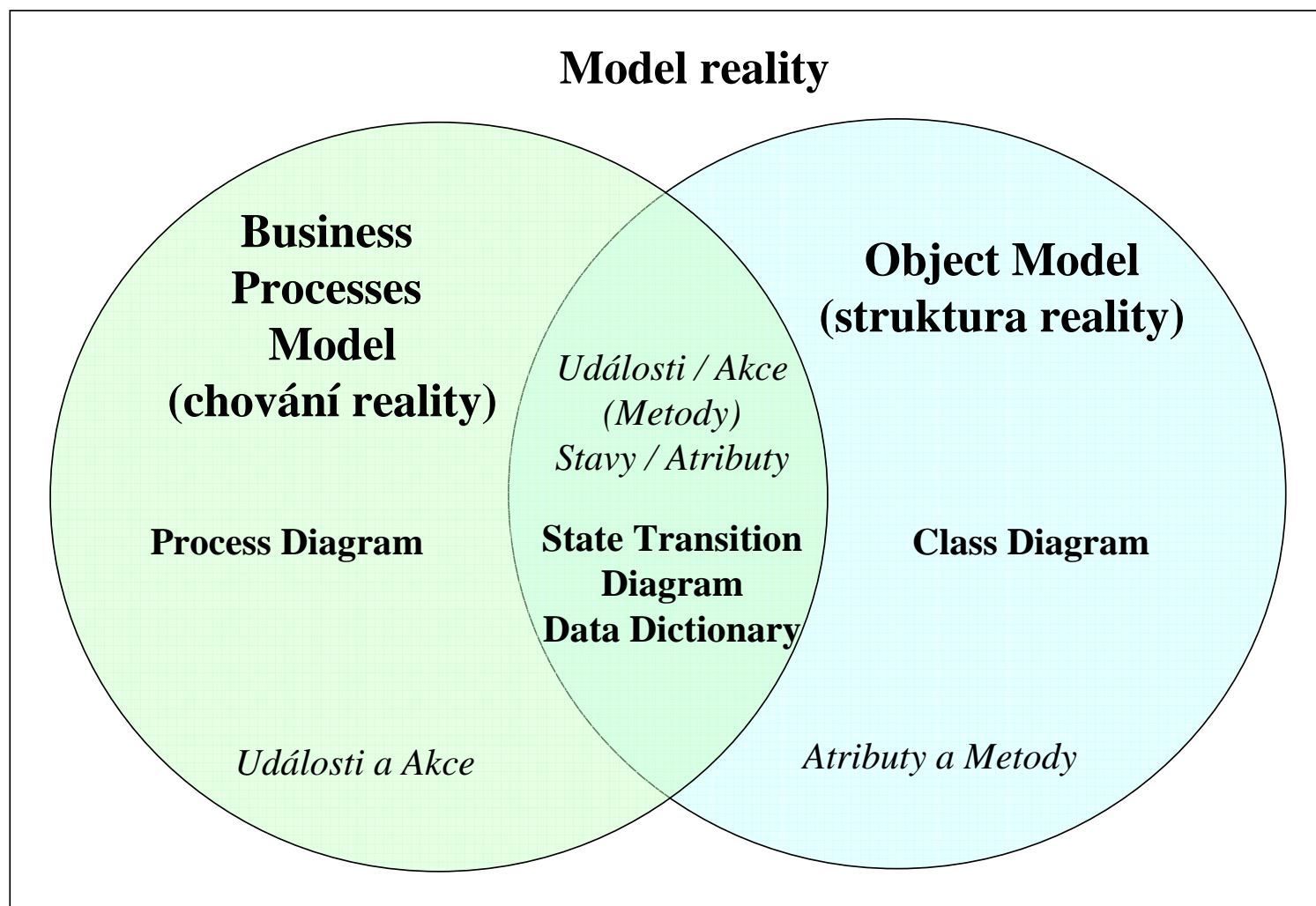
Údlosti, data a  
jejich kontext



# Modelování business a informačního systému

- Konceptuální modelování business objektů
  - Model objektů – Class Diagram
  - Technika návrhu modelu objektů - Normalizace datových struktur
  - Životní cykly objektů – State Chart
  - Konsistence konceptuálních modelů
- Modelování business procesů
  - Model procesů – Process Diagram
  - Provázání procesů s objekty
- Konzistence modelů reality (procesů a objektů)
- Modelování funkčnosti systému
  - Data Flow Diagram
  - Technika návrhu modelu funkcí – Event Partitioning Approach
  - Provázání funkcí s objekty a procesy

# Dvě základní dimenze modelu reality



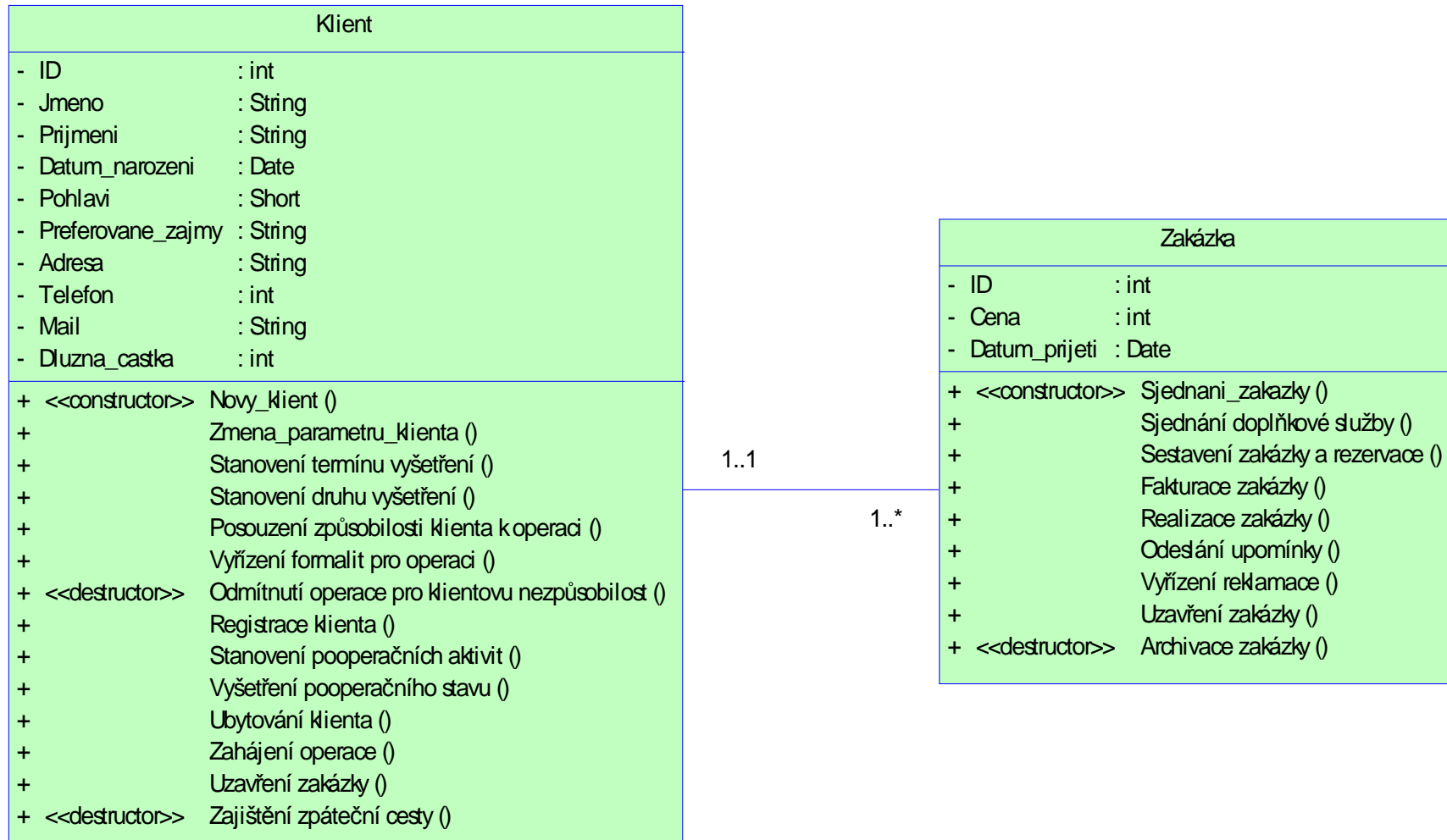
# Konceptuální modelování business objektů

- Model objektů – Class Diagram
- Technika návrhu modelu objektů - Normalizace datových struktur
- Životní cykly objektů – State Chart
- Konsistence konceptuálních modelů

# Model business objektů

- Modeluje statickou strukturu reality, její podstatu nezávislou na konkrétní technologii a implementačním prostředí
- Vyjadřuje typy objektů (třídy, entity) reálného světa a jejich základní (podstatné) vztahy
- Primárním cílem tvorby modelu tříd je pochopit realitu, pojmy používané zákazníkem, složitost reality – chápání reality na úrovni zadavatele - „***konceptuální modelování*** reality“
- Grafické vyjádření = class diagram (**UML**, OMT apod.)

# Diagram tříd - příklad

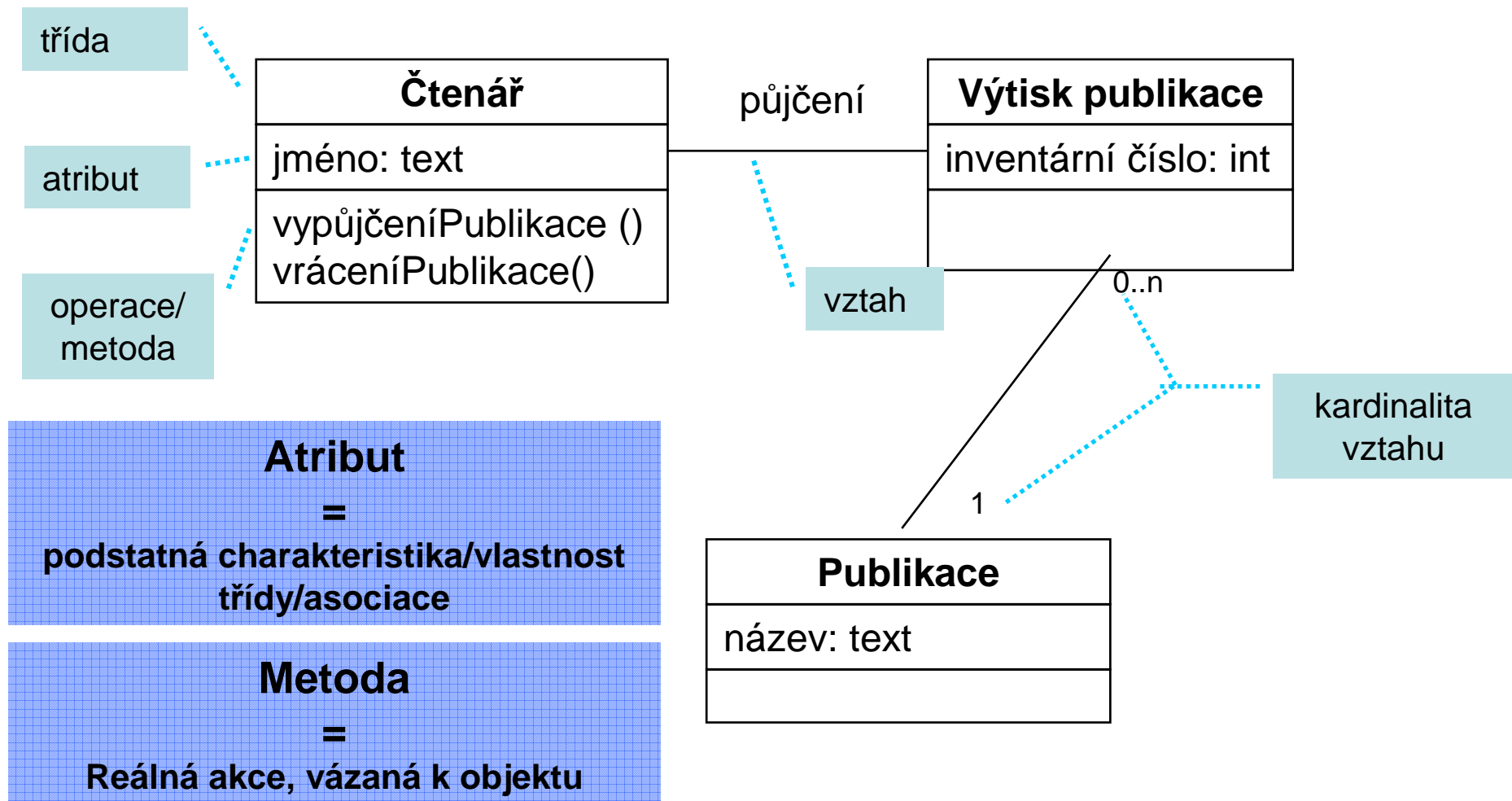


# Odlišnost konceptuálního modelu

- Neobsahuje objekty IS
- Není závislý na programovacím jazyku
- Nerozlišuje viditelnost atributů a metod
- Neobsahuje čtecí metody
- Nezabývá se persistencí
- Nezabývá se implementací vazeb
- Vazby jsou oboustranné
- Každá třída má jeden <<konstruktor>> a alespoň jeden <<destruktor>>, <<transformer>>

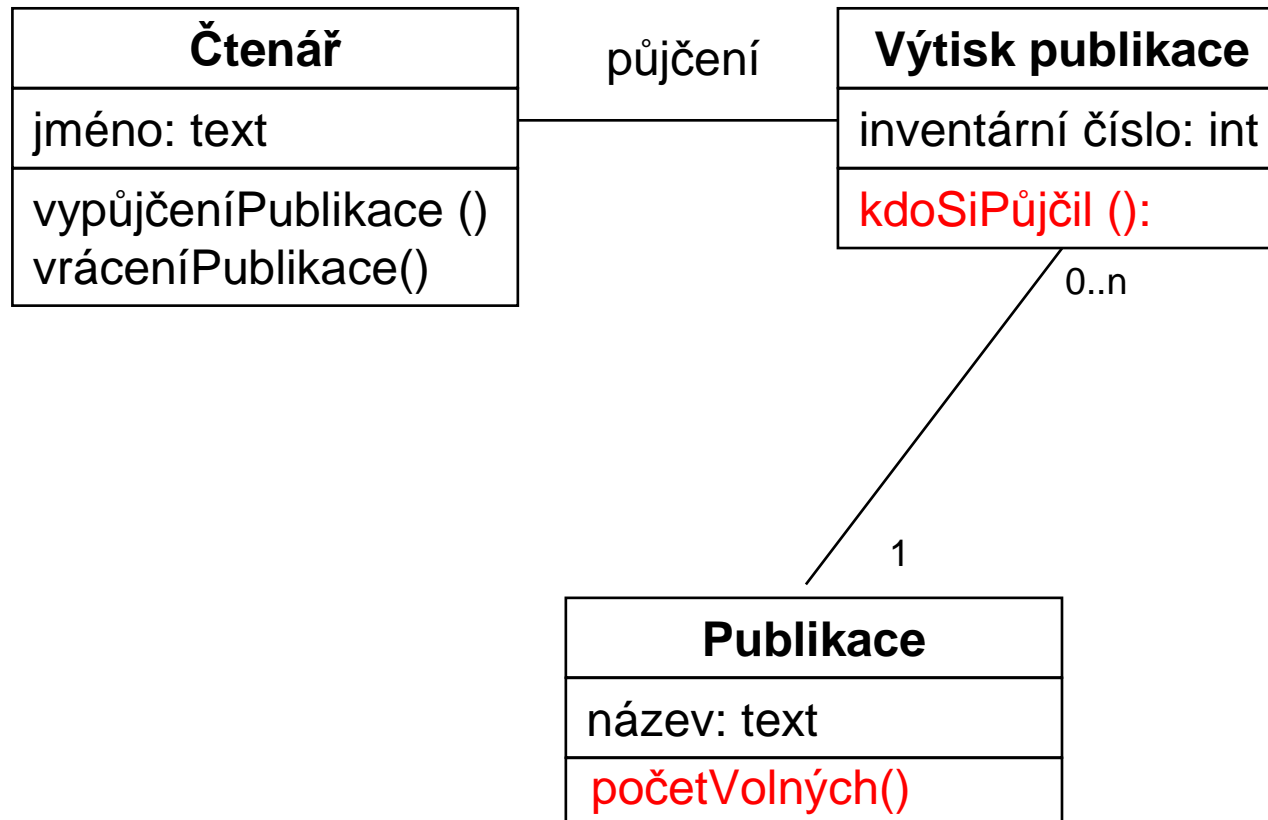


# Diagram tříd



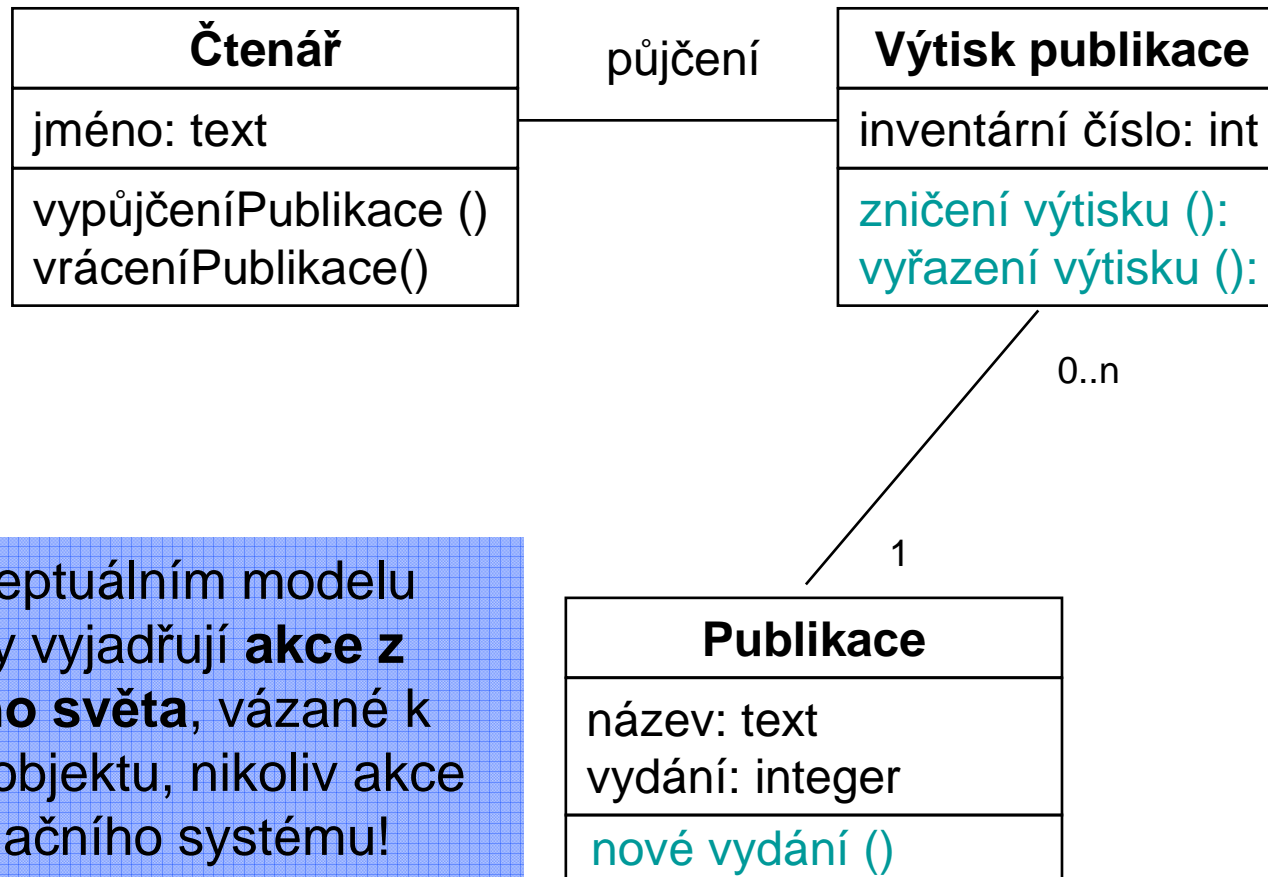
# DIAGRAM TŘÍD

konceptuální versus implementační metody



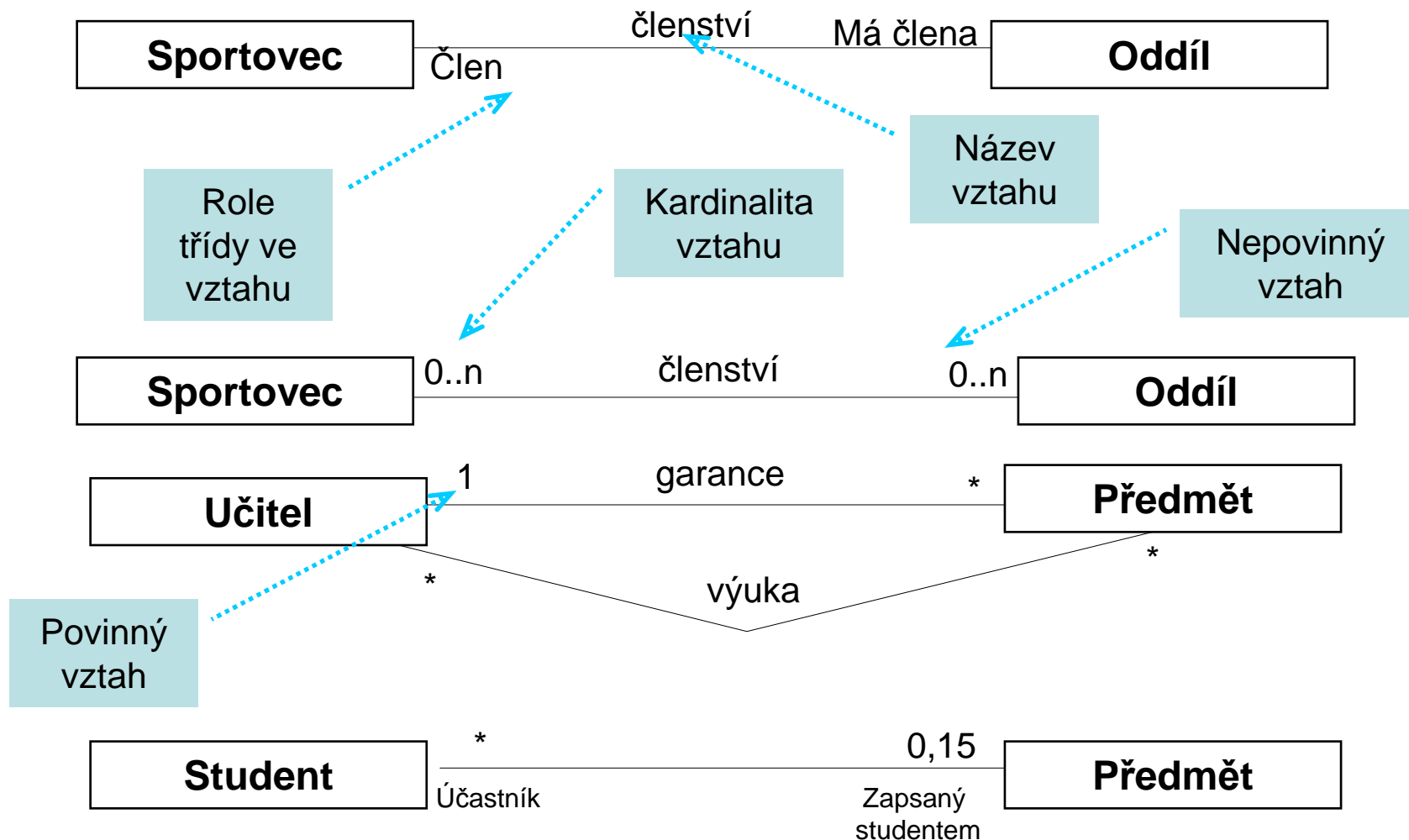
# DIAGRAM TŘÍD

konceptuální versus implementační metody

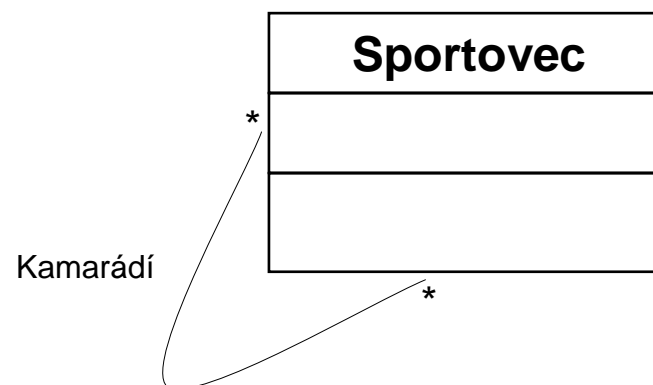
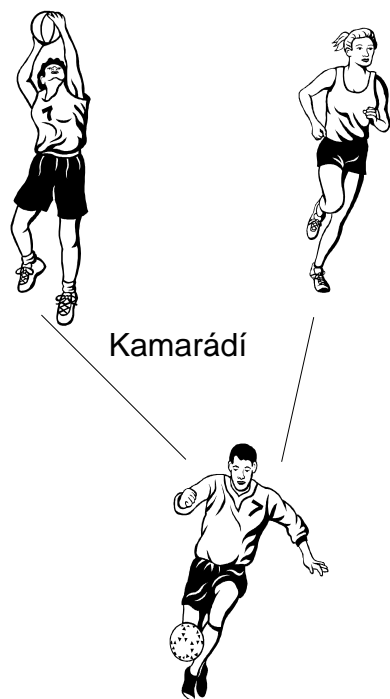


V konceptuálním modelu metody vyjadřují **akce z reálného světa**, vázané k danému objektu, nikoliv akce informačního systému!

# Diagram tříd - asociace



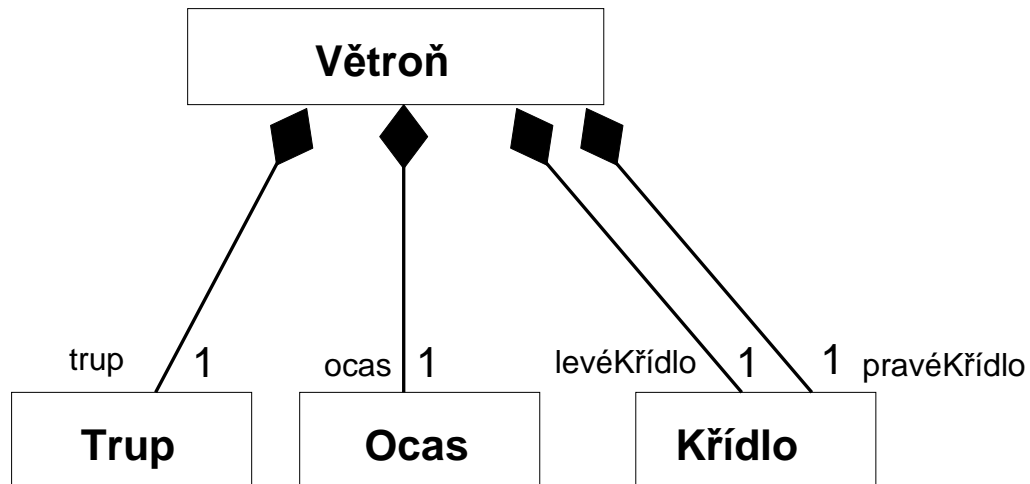
# Model tříd – asociace / vztah



**Reflexivní vztah (asociace)**

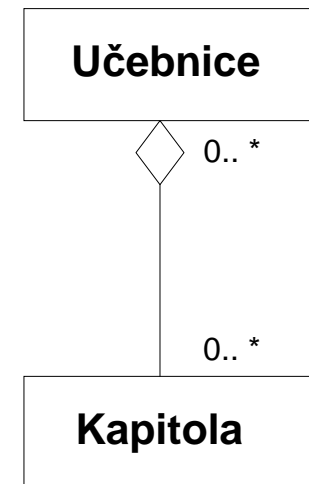
# Model tříd - vztah celku a částí

## Kompozice (složení)



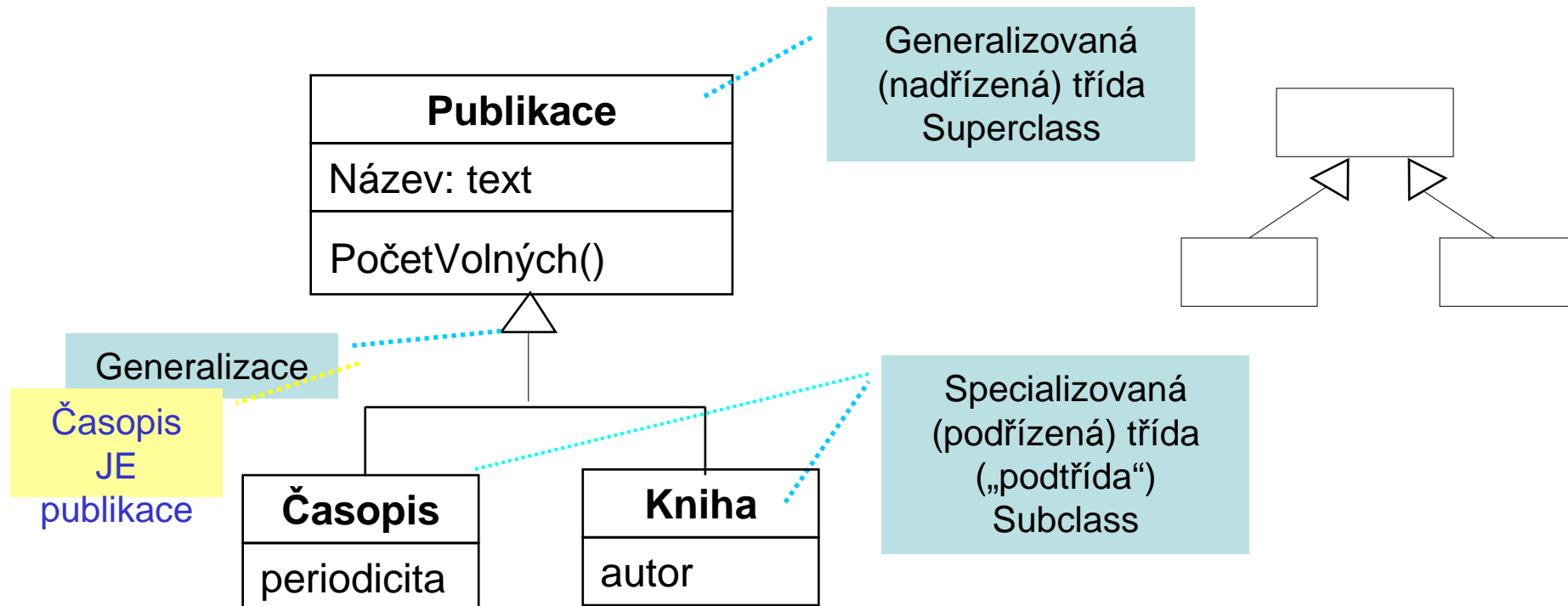
- Objekt kompozice **neexistuje** bez svých komponent
- Objekt komponenty může být v jakýkoliv okamžik součástí jen jedné kompozice
- Komponenty budou pravděpodobně různých typů
- Není-li uvedena kardinalita, předpokládá se přesně 1

## Agregace (seskupení)



- Seskupený objekt **může existovat** i bez svých tvořících objektů
- Jeden objekt může být v jednom okamžiku konstituentem více seskupení
- Konstituenti typického seskupení patří do jedné třídy

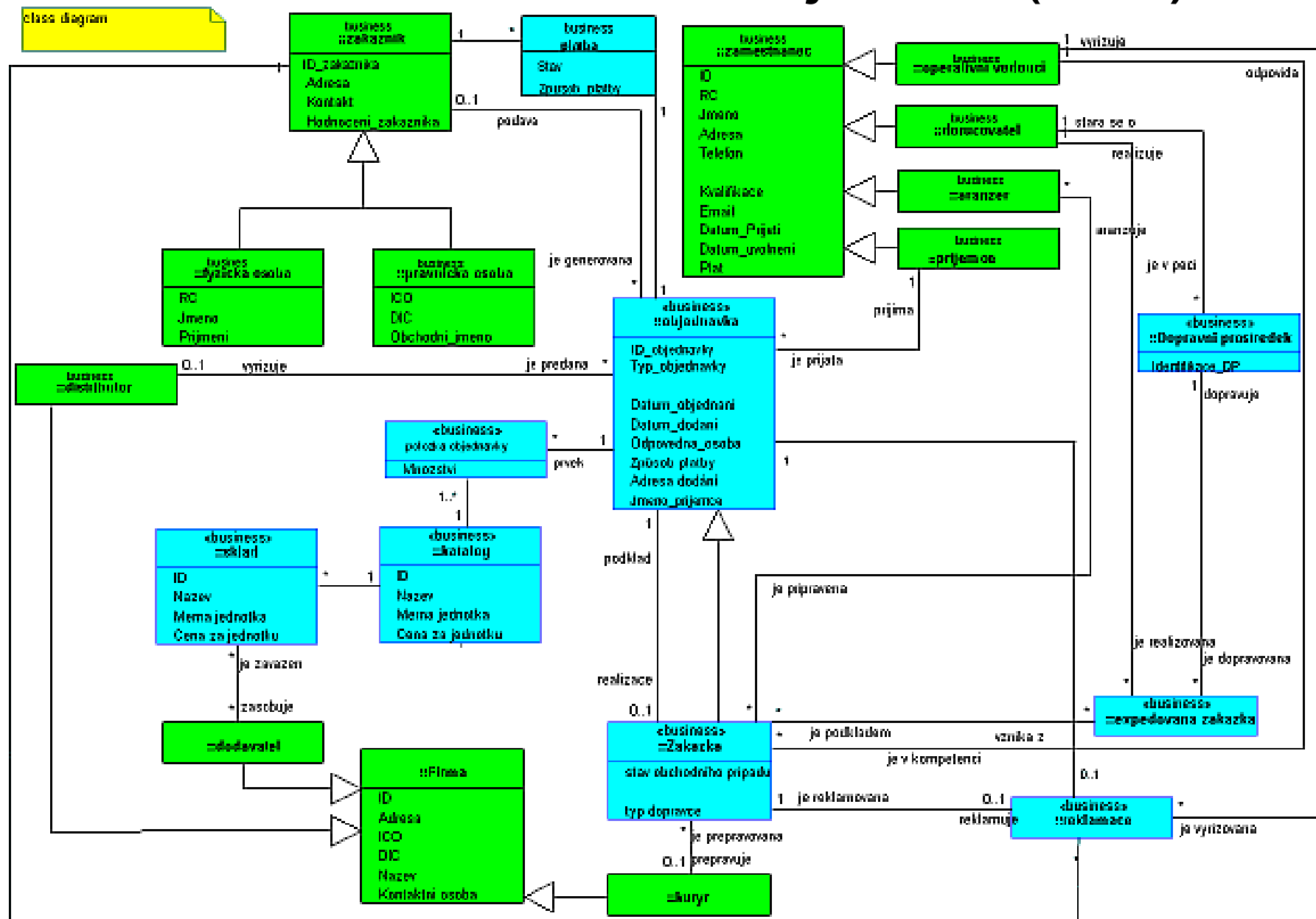
# Generalizace/Specializace



Specializované třídy mají stejné vlastnosti (atributy, operace, asociace) generalizované třídy a něco navíc

**Dědičnost** = jeden ze způsobů realizace generalizace/specializace, pomocí něhož třída nadřízená implicitně definuje všechny atributy a operace třídy podřízené, jako by byly definované přímo v ní (podřízené)

# Příklad modelu objektů (tříd)





# Modelování business objektů

## Normalizace datových struktur

# První normální forma

datová struktura nesmí obsahovat opakující se položky

OS - CISLO (\*)  
JMENO  
PLAT  
TYM  
NAZEV - PROGRAMU  
DAT - ZAH - PRACE  
PLAN - DOKONC - PRACE  
VELIKOST - PROGR  
PROGR - JAZYK

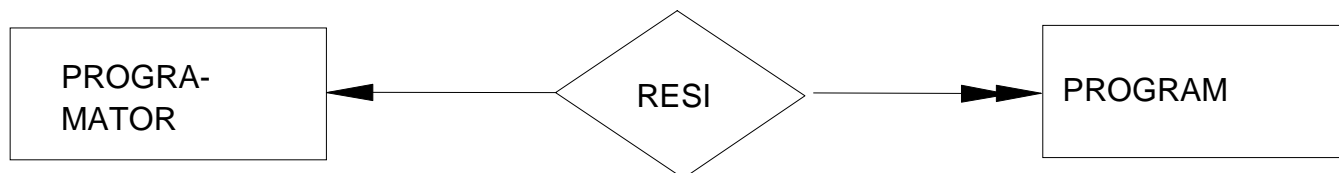
nx

PROGRAMATOR

OS - CISLO (\*)  
JMENO  
PLAT  
TYM

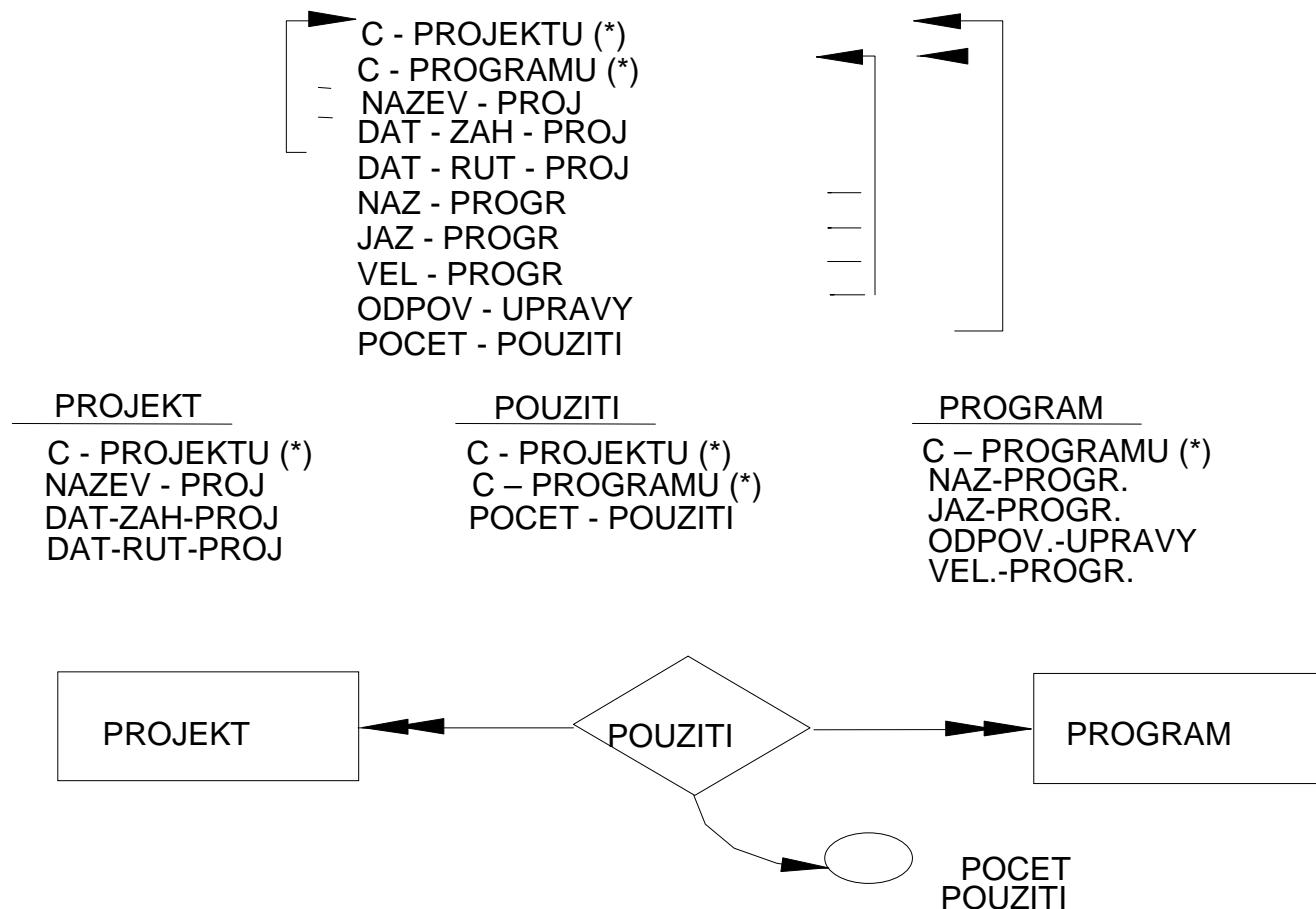
PROGRAM

NAZEV - PROGRAMU (\*)  
DAT - ZAH - PRACE  
VELIKOST - PROGR  
PLAN - DOKONC - PRACE  
PROGR - JAZYK  
OS - CISLO



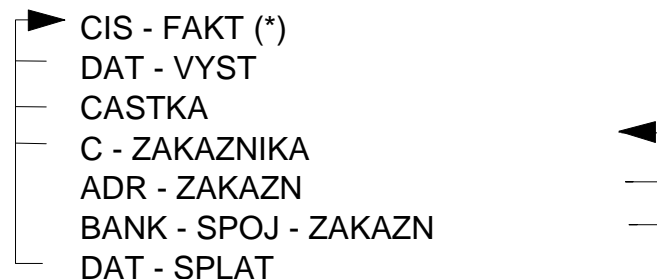
# Druhá normální forma

datová struktura, která obsahuje složený primární klíč, musí obsahovat pouze takové neklíčové položky, které závisí na celém složeném klíči



# Třetí normální forma

datová struktura nesmí obsahovat tranzitivní závislosti, tj. všechny neklíčové položky musí záviset na primárním klíči přímo (ne přes jinou neklíčovou položku)



FAKTURA

CIS - FAKT (\*)  
DAT - VYST  
CASTKA  
DAT - SPLAT  
C - ZAKAZNIKA

ZAKAZNIK

C - ZAKAZNIKA (\*)  
ADR - ZAKAZN  
BANK - SPOJ - ZAKAZN



# Normalizace datových struktur

- Kromě uvedených tří normálních forem je možné v odborné literatuře nalézt popis ještě řady dalších normálních forem, které se vždy zaměřují na určité speciální problémy v datových strukturách, které nejsou dostatečně odfiltrovány prvními třemi normálními formami. Zejména se jedná o 4. a 5. normální formu, které se soustřeďují na nezávislosti či párové cyklické závislosti mezi atributy tvořícími primární klíč.
- S použitím normalizace dat je možno získat datové struktury, které co nejvěrněji odrážejí existující entity a jejich vztahy. Tyto datové struktury ovšem odpovídají konceptuální úrovni pohledu na data a je zapotřebí je před implementací přizpůsobit (zoptimalizovat) dle zvoleného technologického a implementačního prostředí.

# Modelování business objektů

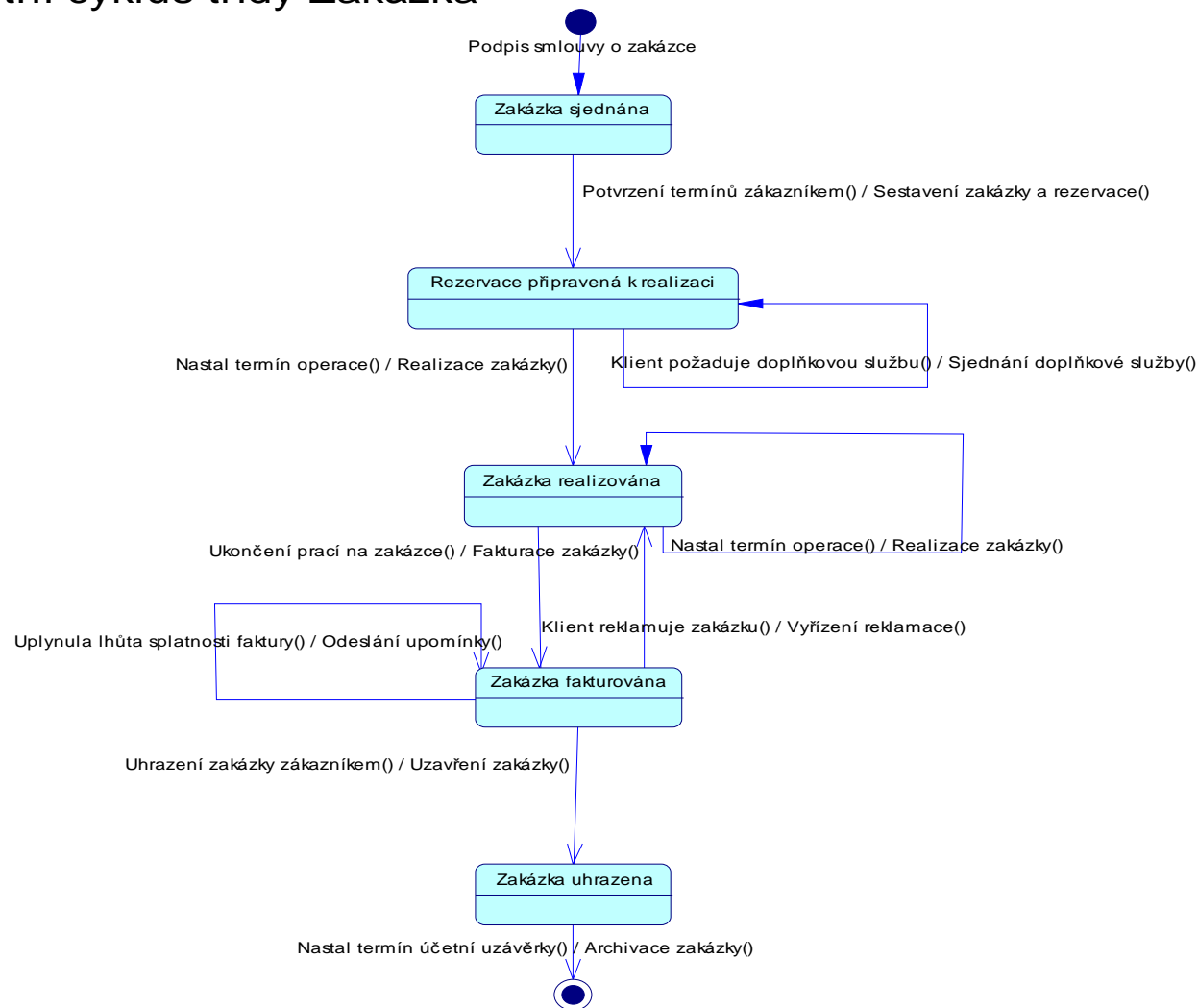
## životní cykly objektů

# Statechart

- Notace UML
- Jeden Statechart pro každou neprimitivní třídu
- Zachycuje **životní cyklus třídy**: všechno možné chování objektu - všechny možné posloupnosti volání metod
- Objekt nesmí přecházet ze stavu jinak než definuje některý z přechodů v statechartu
- Volání metod jinak než popisuje statechart je nepřípustná operace

# Statechart - příklad

## Životní cyklus třídy Zakázka

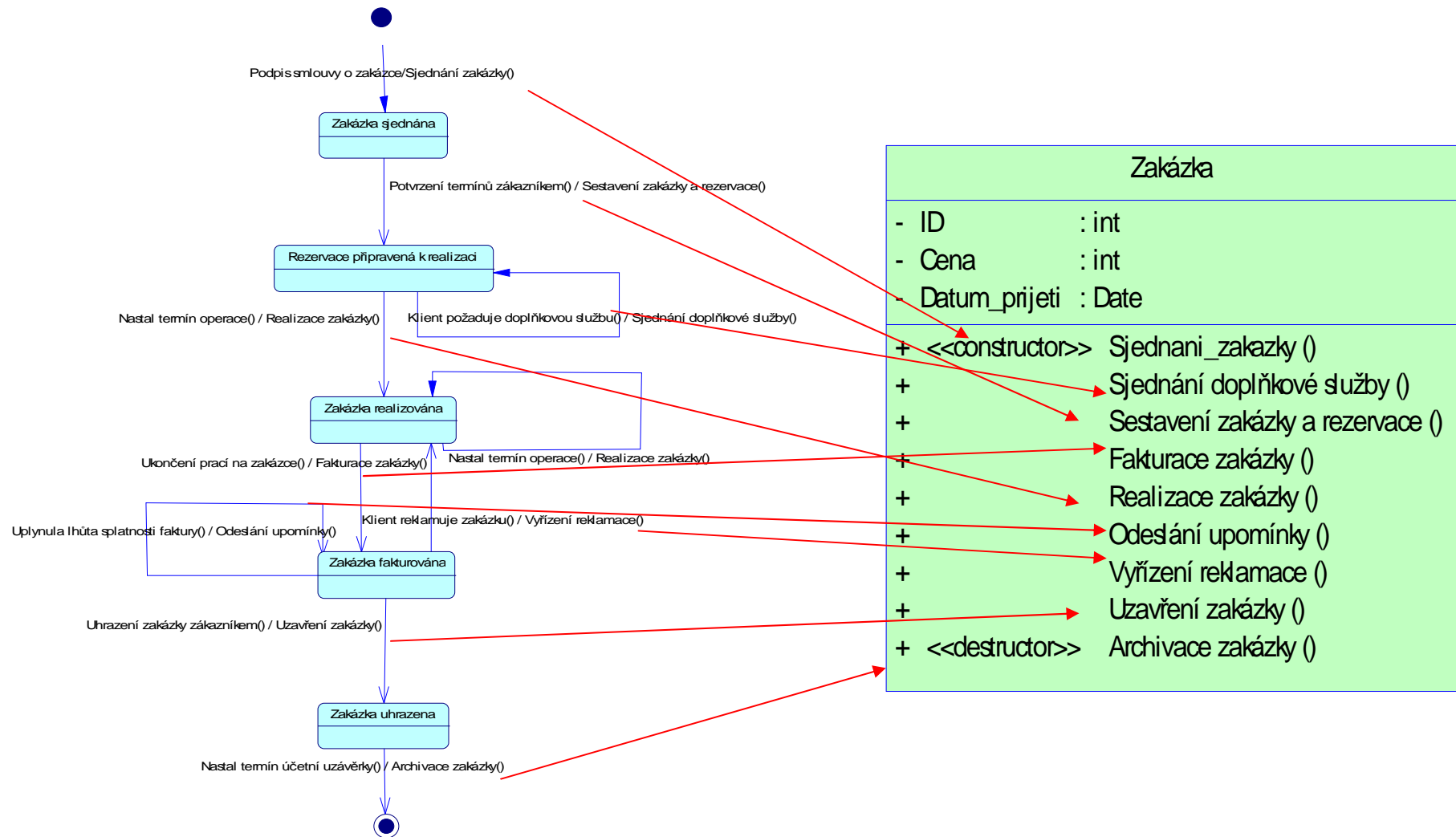




# Provázání konceptuálních diagramů

- Každá neprimitivní třída má právě jeden statechart
- Každá metoda třídy je použita u přechodu v statechartu a naopak každý přechod v statechartu je popsán metodou třídy
- <<konstruktor>> je použit u přechodu z počátečního stavu a <<destruktory>> u přechodů do koncových stavů, ostatní přechody používají <<transformery>>

# Příklad provázání



# Modelování business procesů

- Model procesů – Process Diagram
- Provázání procesů s objekty
- Konzistence modelů reality

# Prvky popisu podnikového procesu

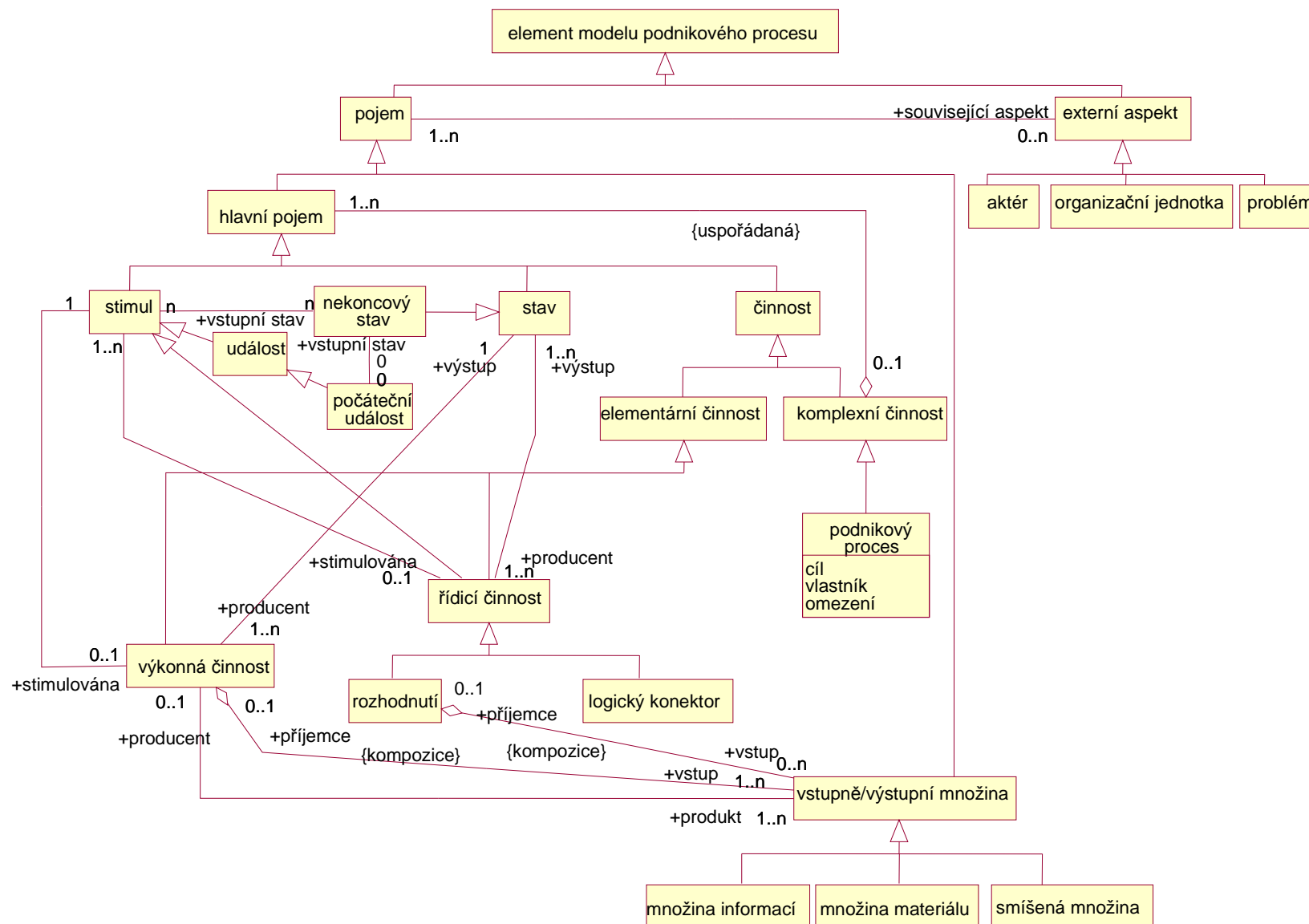
## **Základní elementy**

- podnět
  - událost (externí)
  - stav (interní)
- činnost
  - výkonná činnost ("výroba" výstupů ze vstupů)
  - řídicí činnost (řízení procesu)




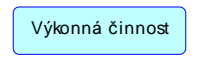
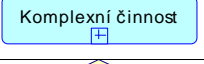
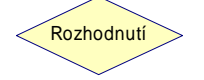

## **Externí elementy**

- vstupně/výstupní množina
  - materiál (surovina nebo produkt)
  - data (řídicí data procesu)
  - smíšená množina
- externí aspekty
  - aktér (účastník, nebo "oběť" činností procesu)
  - organizační jednotka spojená s procesem
  - problém spojený s procesem






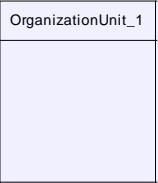

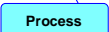
# Meta-model podnikového procesu



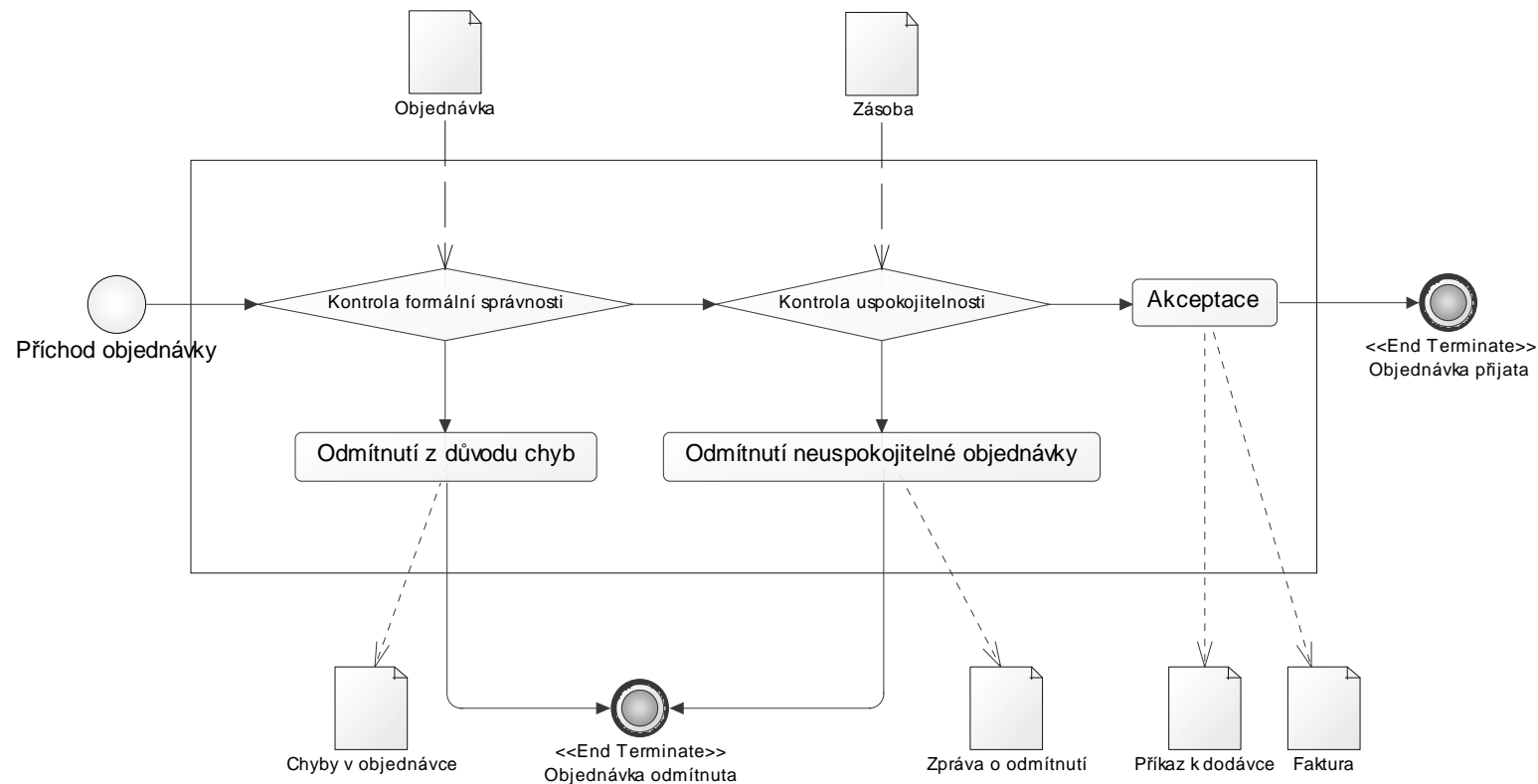
# Elementy popisu procesu 1

Konstrukt	Použitý symbol	Popis
Událost	 <<Event General>> Obecná událost	<p>Vnější podnět činnosti. Informace o skutečnosti nastalé mimo proces (nezávisle na něm).</p> <p><i>V nástroji Power Designer lze vyjádřit použitím symbolu „start“ doplněného názvem události. Start lze použít vícenásobně – pro každou událost. Pro popis formy vstupu, jímž je událost signalizována (pokud je s událostí spojen nějaký hmotný, či informační vstup, např. u událostí časovaných (periodických) lze použít bohatý repertoár symbolů BPMN, diskutovaný níže a vhodný i pro rozlišení událostí časovaných od běžných (<b>business</b>).</i></p>
Stav procesu	 <<Parallel(AND)>> Vnitřní stav procesu   <<End Terminate>> Koncový stav obecný	<p>Vnitřní podnět činnosti. Výsledek činnosti logicky předcházející. Místo mezi činnostmi procesu.</p> <p><i>V notaci Power Designeru, lze vyjádřit použitím „synchronizace“.</i></p> <p>Koncový stav procesu.</p> <p><i>V nástroji Power Designer lze použít symbol „End“. Pro vyjádření formy výstupu, s nímž je koncový stav případně spojen, obsahuje jazyk BPMN bohatou paletu symbolů, podobně jako u událostí (viz Událost).</i></p>
Činnost	 Výkonná činnost   Komplexní činnost	<p>Základní element procesu – zpracování vstupů na výstupy. Činnost je z principu dekomponovatelná, čili může být nahlížena jako samostatný proces (komplexní činnost).</p> <p><i>Dekompozice(nastavení volby „Change to Composite“) je graficky znázorněna smyčkou v boxu činnosti.</i></p>
Rozhodovací činnost	 Rozhodnutí   <<Complex>> Rozhodnutí (BPMN)	<p>Elementární (dále nedekomponovatelná) činnost, jejímž výstupem je nic více, než rozhodnutí o dalším postupu procesu.</p>

# Elementy popisu procesu 1

Logická spojka (primitivní rozhodnutí)	 <<Parallel(AND)>> AND   <<Inclusive(OR)>> OR   <<Data-XOR>> XOR - vylučnost dat  <<Event-XOR>> XOR - vylučnost událostí	<p>Primitivní rozhodovací činnost, která nepotřebuje žádné dodatečné (informační) vstupy.</p> <p><i>V nástroji Power Designer jsou z nabídky BPMN použitelné standardní stereotypy rozhodovací činnosti AND, OR a (nikoliv nezbytné) dva podtypy XOR (datový a událostní).</i></p>
Množina dat	 Vstup / výstup	<p>Množina údajů, či surovin, které slouží jako zdroj pro provedení činnosti procesu nebo je jejich výstupem (obecný zdroj). Příklady: výrobní plán, strategický plán investic, dodací list apod.</p> <p>Lze použít i jako množina materiálu v kombinaci s informací. Příklad: dodávka společně s dodacím listem.</p>
Smíšená množina		
Množina materiálu		
Aktér		Abstraktní účastník (osoba – její role, útvar, systém, orgán, objektivní entita) procesu.
Organizační jednotka		<p>Organizační část organizace, v níž proces probíhá.</p> <p><i>V notaci BPMN jsou organizační jednotky použitelné pouze ve formě tzv. „swim lanes“ (plavečkových drah), uzavírajících všechny činnosti náležející dané jednotce (roli). To, žel, poněkud redukuje možnosti popisu procesu, nezávislého na organizační struktuře.</i></p>
Problém	 	<p>Problém, spojený s procesem v jistém jeho místě (stavu).</p> <p><i>V nástroji PowerDesigner lze vyjádřit poznámkou („note“), nebo rovnou do popisu procesu.</i></p>

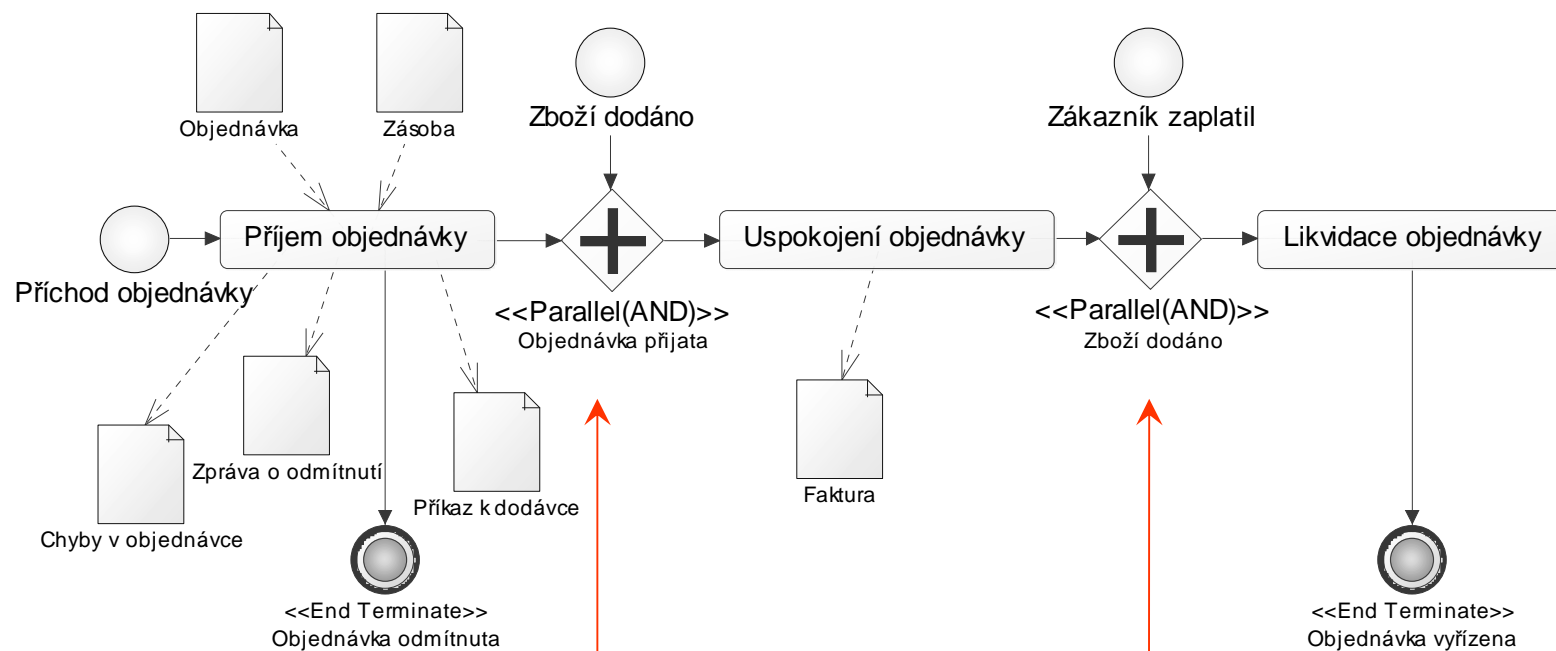
# Primitivní proces (Příjem objednávky)





# Komplexní proces

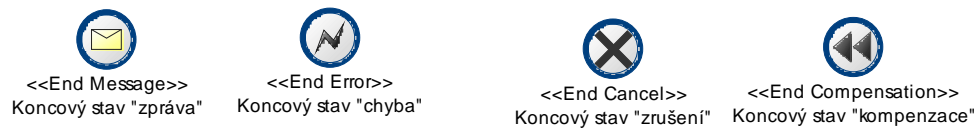
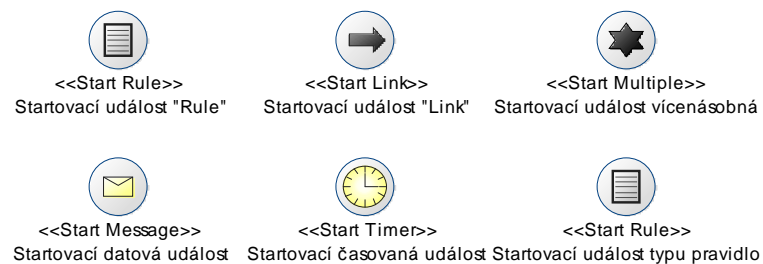
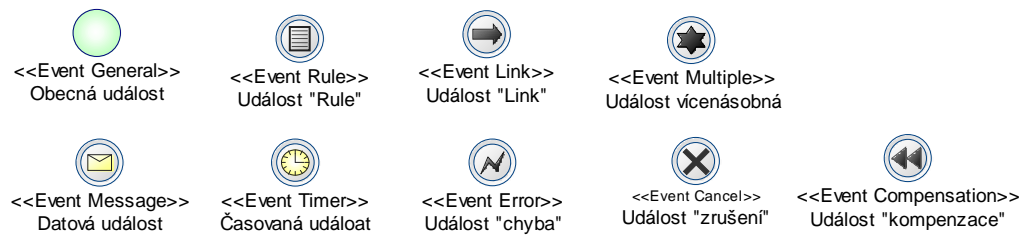
## Vyřízení objednávky zákazníka



Čekání na  
akci skladu

Čekání na akci  
zákazníka

# Události a stavy v BPMN



# Technika

## Tři úrovně zjednodušení modelu

Úroveň	Popis	Účel zjednodušení
úroveň 0	Plná složitost. Použity všechny elementy.	
úroveň 1	Model bez aktérů, problémů a organizačních jednotek	Popis procesu samotného bez ohledu na externí aspekty (aktéry, problémy a organizaci). Není možná analýza externích aspektů procesu (např. v rámci informační analýzy současného stavu).
úroveň 2	Model úrovně 1 bez vstupů a výstupů (hmotné, informační, či smíšené množiny)	Popis procesu samotného bez ohledu na vstupy a výstupy činností. Model popisuje toliko posloupnost činností a jejich řízení (podněty). Není možné popsat podstatu zpracování.
úroveň 3	Model úrovně 2 bez stavů a řídicích činností. .	Popis procesu samotného bez ohledu na vstupy a výstupy činností. Model popisuje toliko posloupnost výkonných činností. Není možné popsat vnitřní řízení procesu.

# Paměť procesu

## Proč:

- **Potřeba uložit informaci o momentálním stavu procesu** v případě řízení komplexních procesů (majících často složité vazby k jiným procesům).
- **Potřeba snížit složitost popisu procesu.**

## Paměť procesu obsahuje:

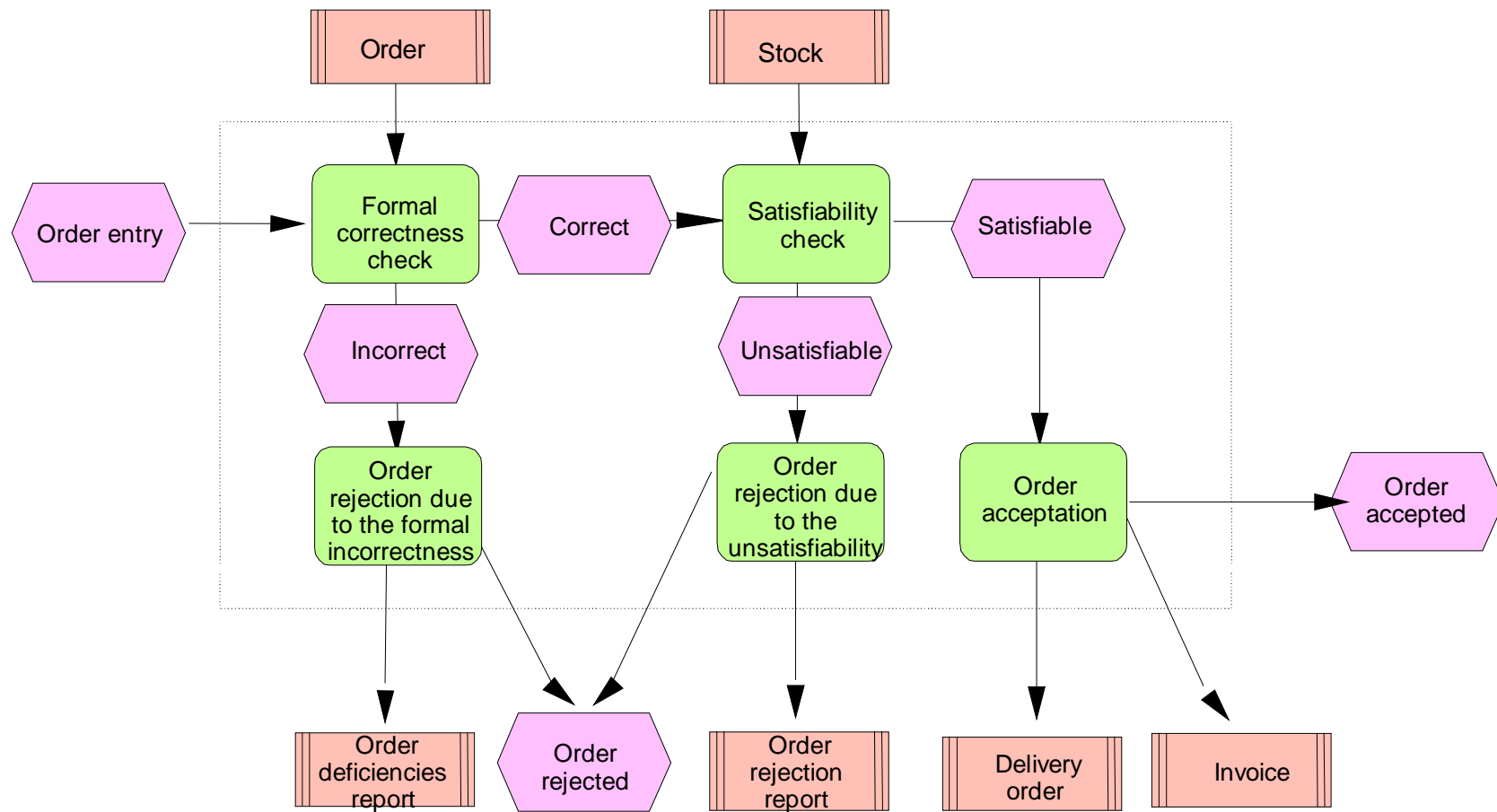
- identifikaci momentálního **stavu procesu**
- **atributy** momentálního stavu procesu
- **data získaná** činnostmi procesu  
(jakmile jsou data získána, existují uvnitř procesu a mohou být bez omezení používána činnostmi procesu (tzv. globální přístup k datům))

## Důsledky:

- kritérium **rozdílu mezi primitivním a komplexním procesem.**  
(proces, nevyžadující ukládání informace o svém stavu je možné považovat za jednoduchý algoritmus (a též jej tak implementovat))
- ukazuje na **možný paralelismus v procesu** nebo přinejmenším na potřebu komunikace s okolními procesy.

# Primitive Process (Order Receiving)

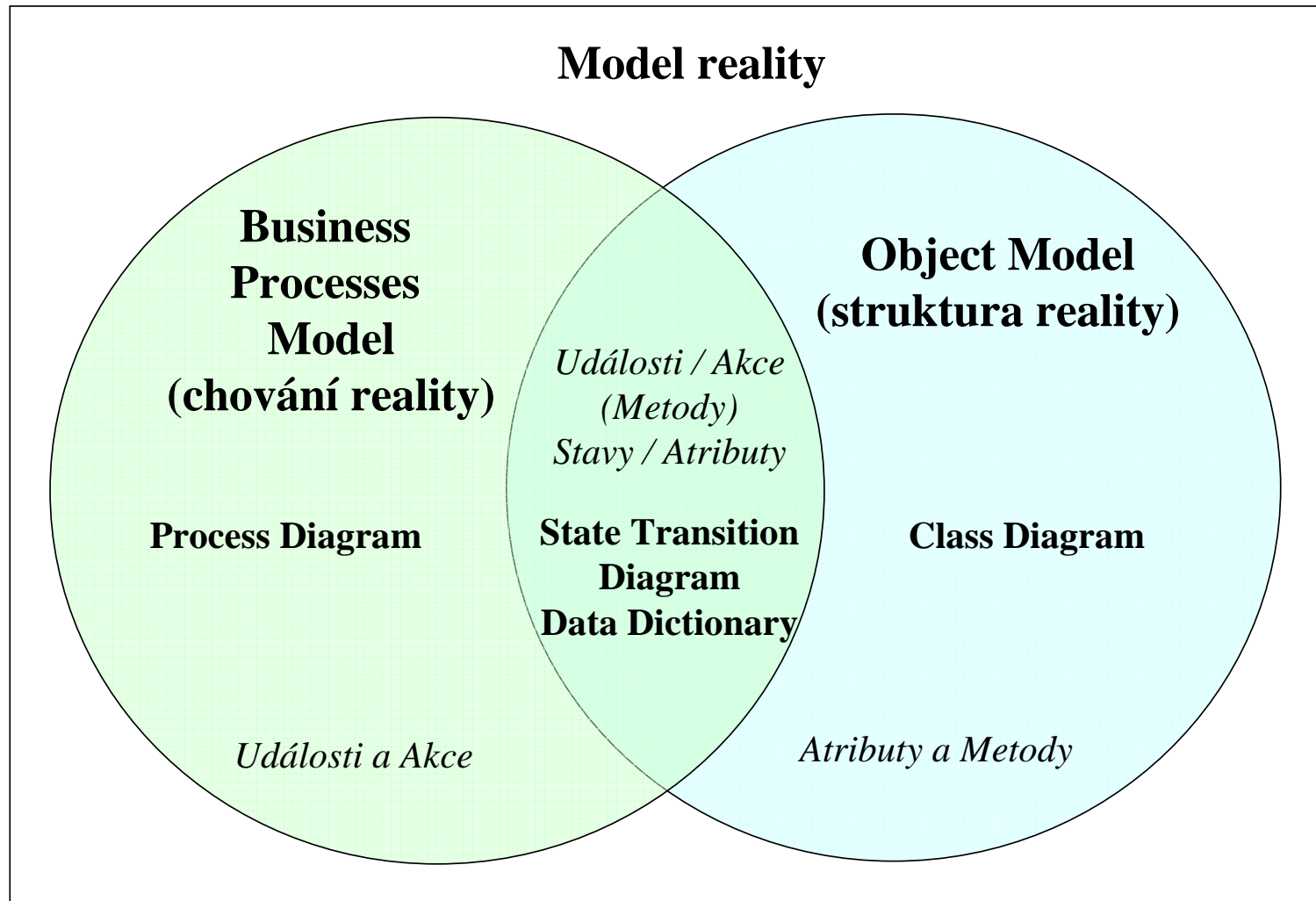
## - Aris Notation



# Konzistence procesů a objektů

# Konzistence procesů a objektů

## Dvě základní dimenze modelu reality



# Konzistence procesů a objektů

**Přehled potřeby konzistenčních pravidel ve věci externích skutečností (různé významy téže skutečnosti)**

<b>Fakt</b>	<b>Model objektů</b>	<b>Model podnikových procesů</b>
Událost	Podnět ke: <ul style="list-style-type: none"> <li>• Změně vnitřního stavu objektu</li> <li>• Možné komunikaci s jinými objekty (poslání zprávy) pokud se jedná o tzv. "společnou akci"</li> </ul>	Podnět k: <ul style="list-style-type: none"> <li>• Provedení činnosti</li> <li>• Změně stavu procesu</li> <li>• Produkci výstupu</li> <li>• Možné komunikaci s jinými procesy (koordinace procesů)</li> </ul>
Výstup	Důsledek: <ul style="list-style-type: none"> <li>• Akce objektu</li> <li>• Změny vnitřního stavu objektu</li> </ul>	Důsledek: <ul style="list-style-type: none"> <li>• Provedení činnosti (produkt činnosti)</li> <li>• Změny stavu procesu</li> </ul>

**Přehled potřeby konzistenčních pravidel ve věci vnitřních pojmů (různé významy téhož pojmu)**

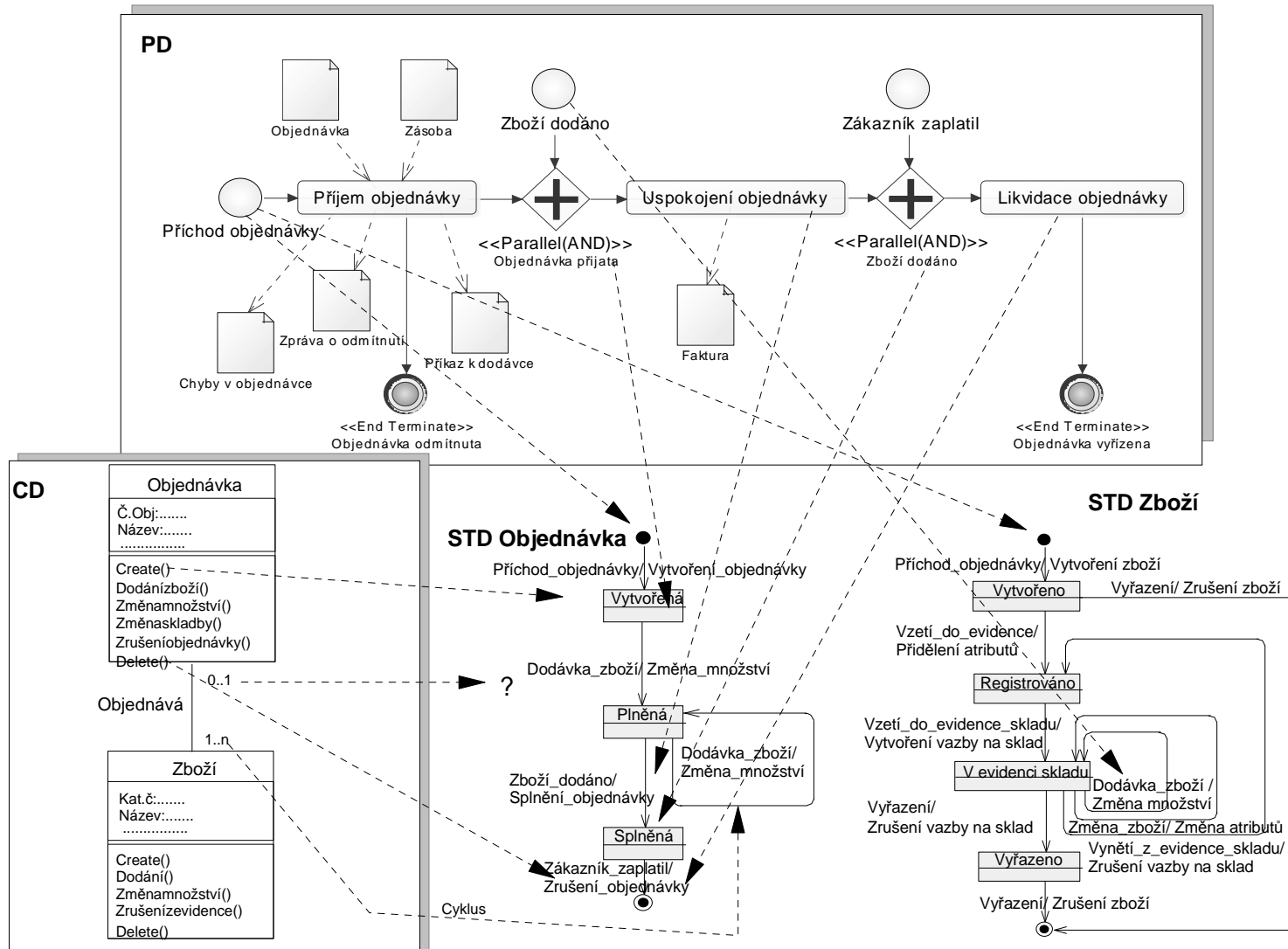
<b>Pojem</b>	<b>Model objektů</b>	<b>Model podnikových procesů</b>
Akce	Akce provedená/připuštěná objektem Má ze následek: <ul style="list-style-type: none"> <li>• Změnu stavu objektu</li> <li>• Možný výstup</li> <li>• Možnou komunikaci s jinými objekty (poslání zprávy) pokud se jedná o tzv. "společnou akci"</li> </ul>	Činnost procesu Má ze následek: <ul style="list-style-type: none"> <li>• Změnu stavu procesu</li> <li>• Možný výstup - produkt procesu</li> <li>• Možnou komunikaci s jinými procesy (koordinace procesů)</li> </ul>
Stav	Stav životního cyklu objektu <ul style="list-style-type: none"> <li>• Východisko (podnět) akce</li> <li>• Výsledek akce</li> </ul>	Stav běhu procesu <ul style="list-style-type: none"> <li>• Východisko (podnět) činnosti</li> <li>• Výsledek činnosti</li> </ul>



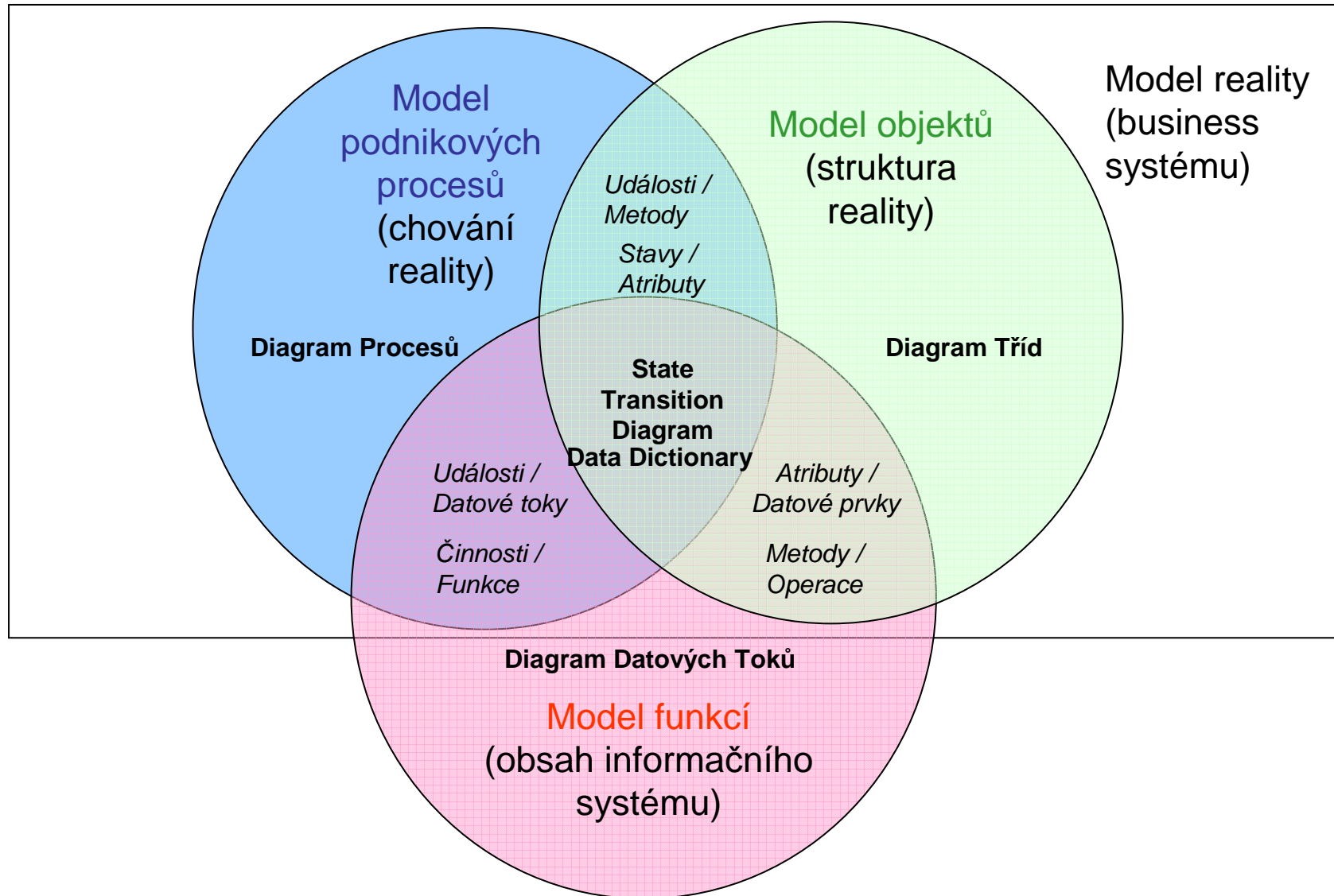
# Provázání procesů s objekty

- Každá třída objektů z modelu tříd musí být zastoupena v modelu procesů v alespoň jednom z jeho vstupů, či výstupů a/nebo aktérů, či jiných externích aspektů.
- Každý vstup, či výstup procesu, jakož i každý externí aspekt procesu, musí být zastoupen v modelu tříd jako třída, nebo asociace mezi třídami, či jako kombinace obojího.
- Každá událost, specifikovaná v popisech přechodů ve stavovém diagramu životního cyklu třídy, musí korespondovat s událostí, specifikovanou v popisu nějakého (nějakých) business procesu (procesů).

# Příklad provázání



# Přehled analytických modelů



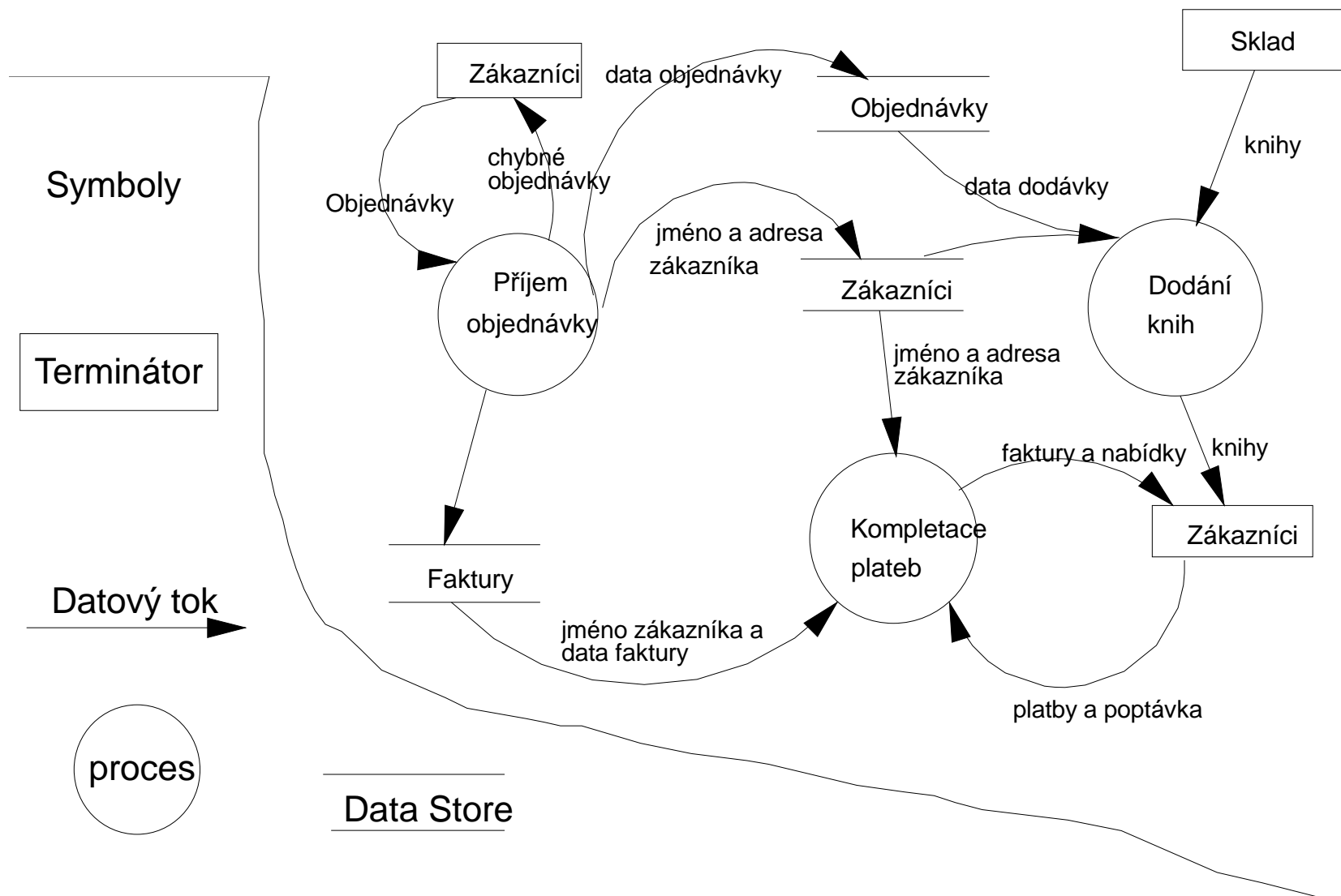
# Modelování funkčnosti systému

- Data Flow Diagram
- Technika návrhu modelu funkcí
  - Event Partitioning Approach
- Provázání funkcí s objekty a procesy

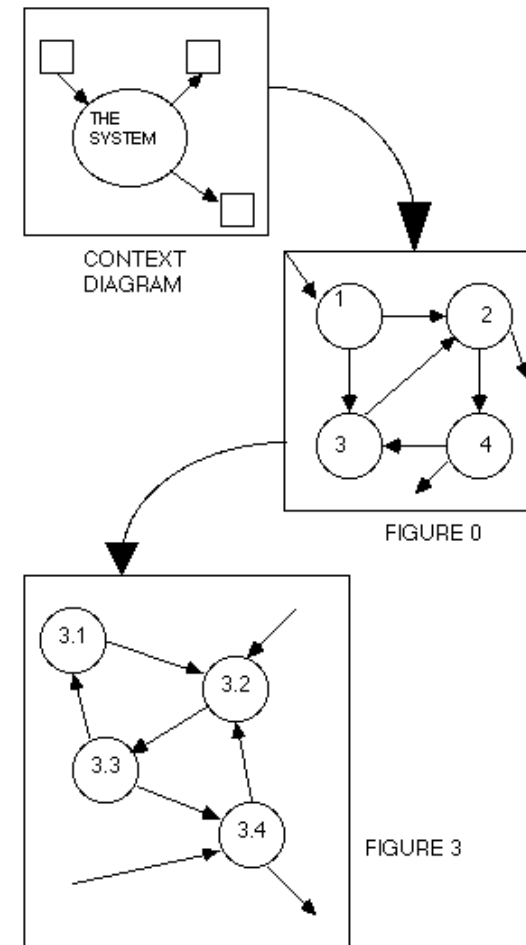
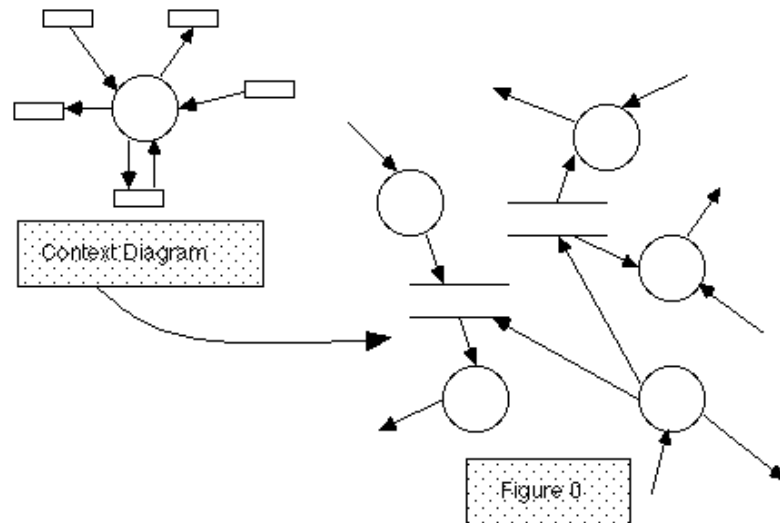
# Data Flow Diagram (DFD)

- Notace Yourdon (DeMarco)
- Použití:
  - v analýze - modelování funkčnosti systému
    - cílem je popsat chování informačního systému – funkce a jejich vazby:
      - Datové toky
      - Datastorey (úložiště dat)

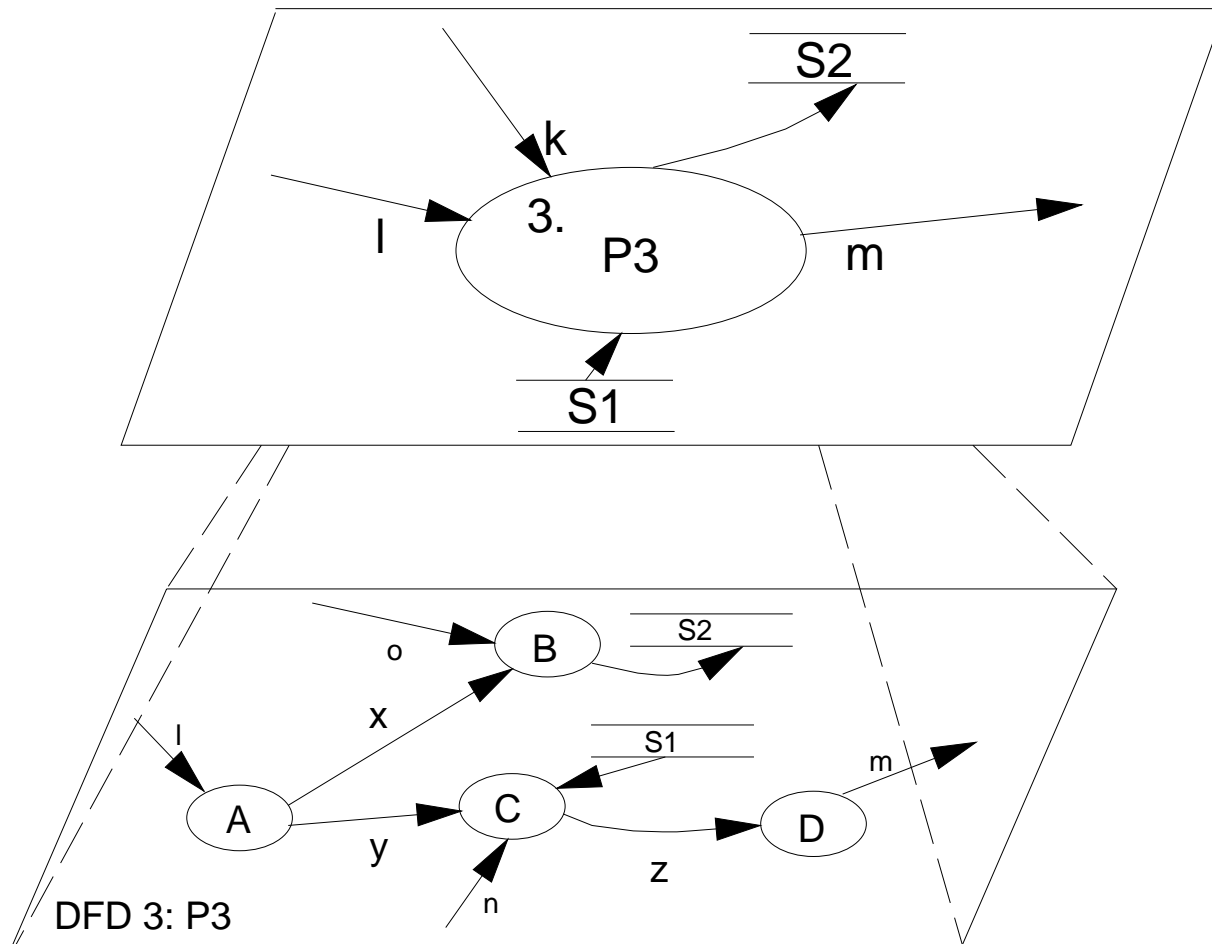
# Data Flow Diagram (DFD)



# Hierarchie DFD



# Konzistence hierarchie DFD

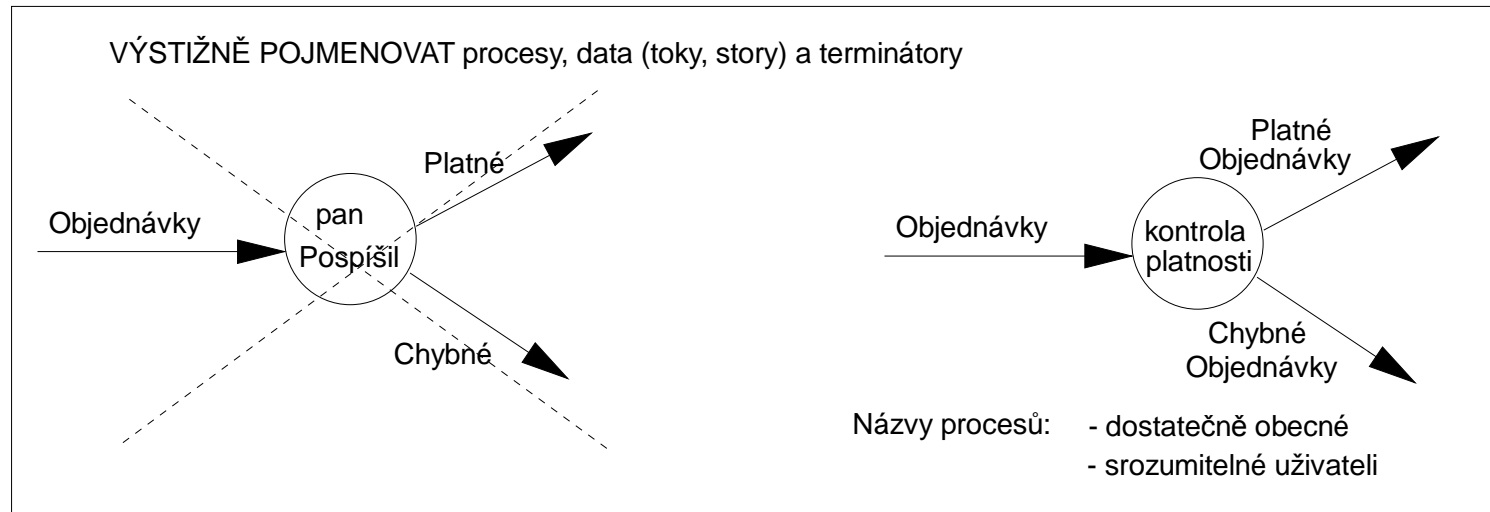


Data Dictionary

$$k = n + o$$



# Pravidla tvorby DFD



## OČÍSLOVAT procesy

- číslo identifikuje proces v rámci úrovně
- číslo určuje příslušnost procesu do nadřazeného procesu



## Volit SNESITELNOU SLOŽITOST DFD

- DFD s příliš procesy je nesrozumitelný (rozdělit do úrovní)
- Jeden DFD = 7 +/- 2 procesy

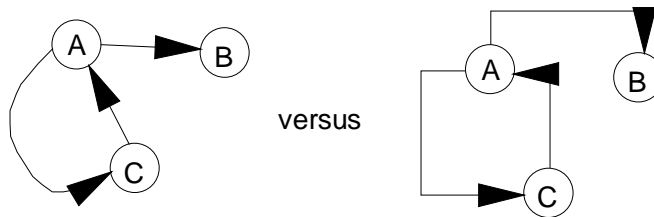
vždy musí být úplný

# Pravidla tvorby DFD



Volit DOSTATEČNĚ ESTETICKÉ USPOŘÁDÁNÍ DFD

- Velikost a tvar bublin
- Oblé versus hranaté spojnice

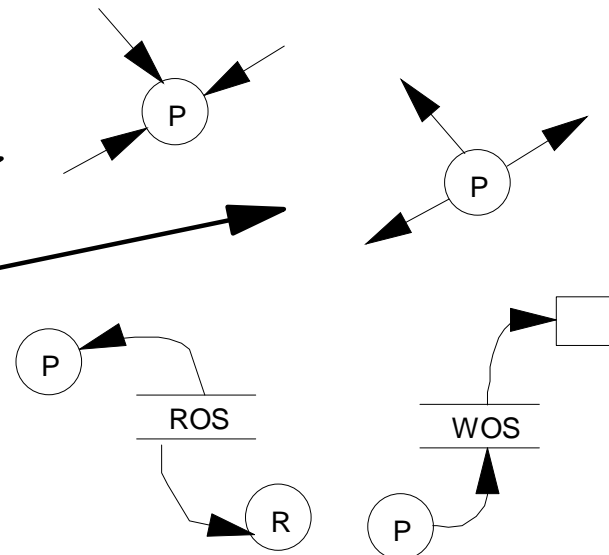


Sledovat FORMÁLNÍ SPRÁVNOST DFD

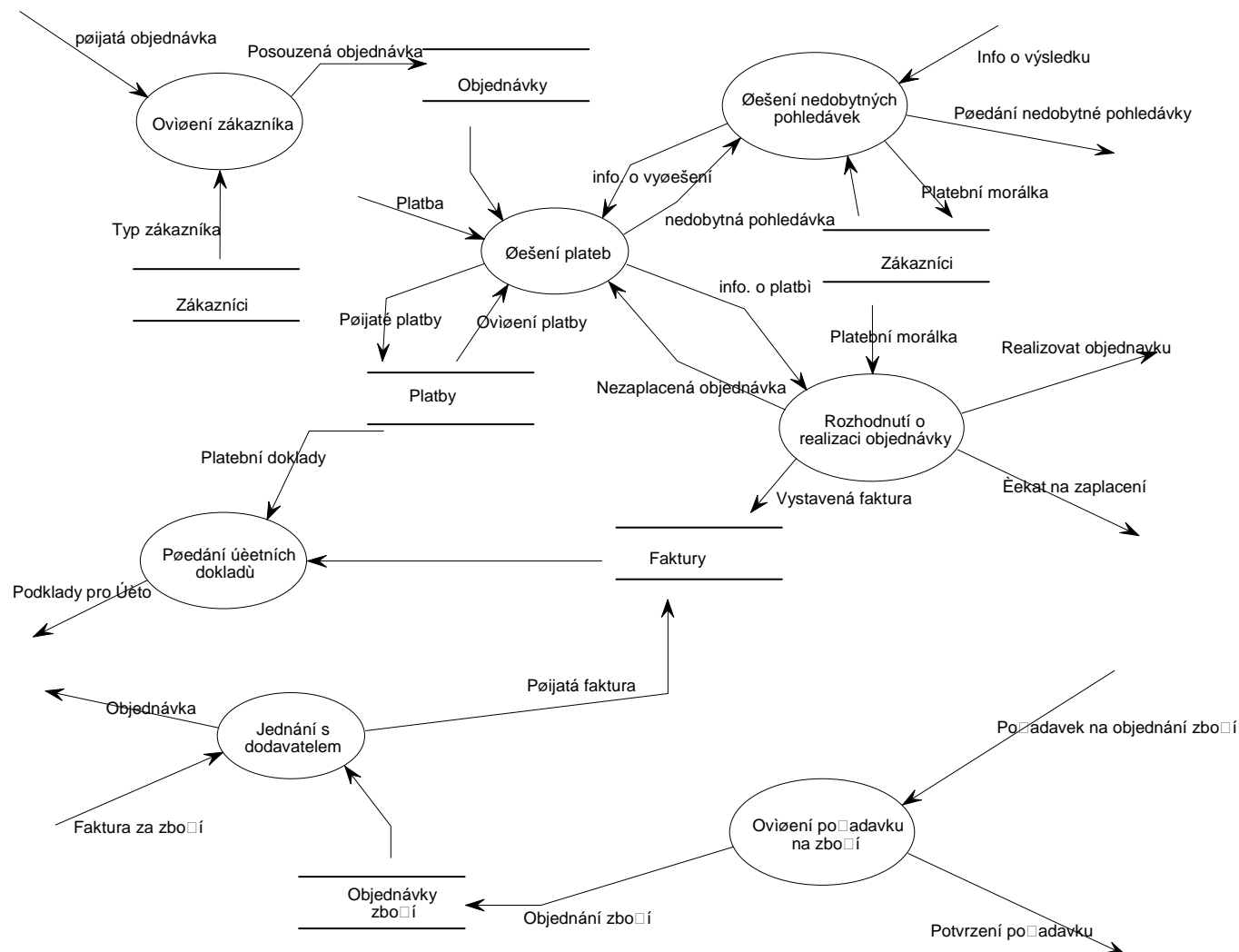
t.j. soudržnost uvnitř DFD

DFD NESMÍ obsahovat:

- ČERNÉ DÍRY
- Samogenerující procesy
- Neoznačené toky a procesy
- READ ONLY a WRITE ONLY story



# Příklad DFD - finanční řízení



# Realizace DFD

Specializace diagramu tříd, 4 standardní stereotypy:

- DataStore (třída)
- Funkce (třída)
- Terminátor (třída)
- DataFlow (asociace)

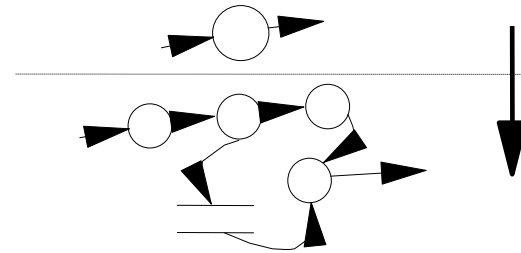
Pravidla konzistence DFD:

- DataStore musí mít alespoň jeden vstupní DataFlow a jeden výstupní DataFlow.
- DataFlow smí spojovat pouze Funkci a Funkci, Funkci a DataStore nebo Terminátor a Funkci.
- DataFlow Terminátor -> Funkce musí mít přiřazenu událost
- Funkce musí mít alespoň jeden DataFlow

# Postup tvorby funkčního modelu

■ *Top-down funkční dekompozice*

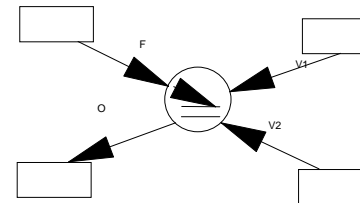
nebo



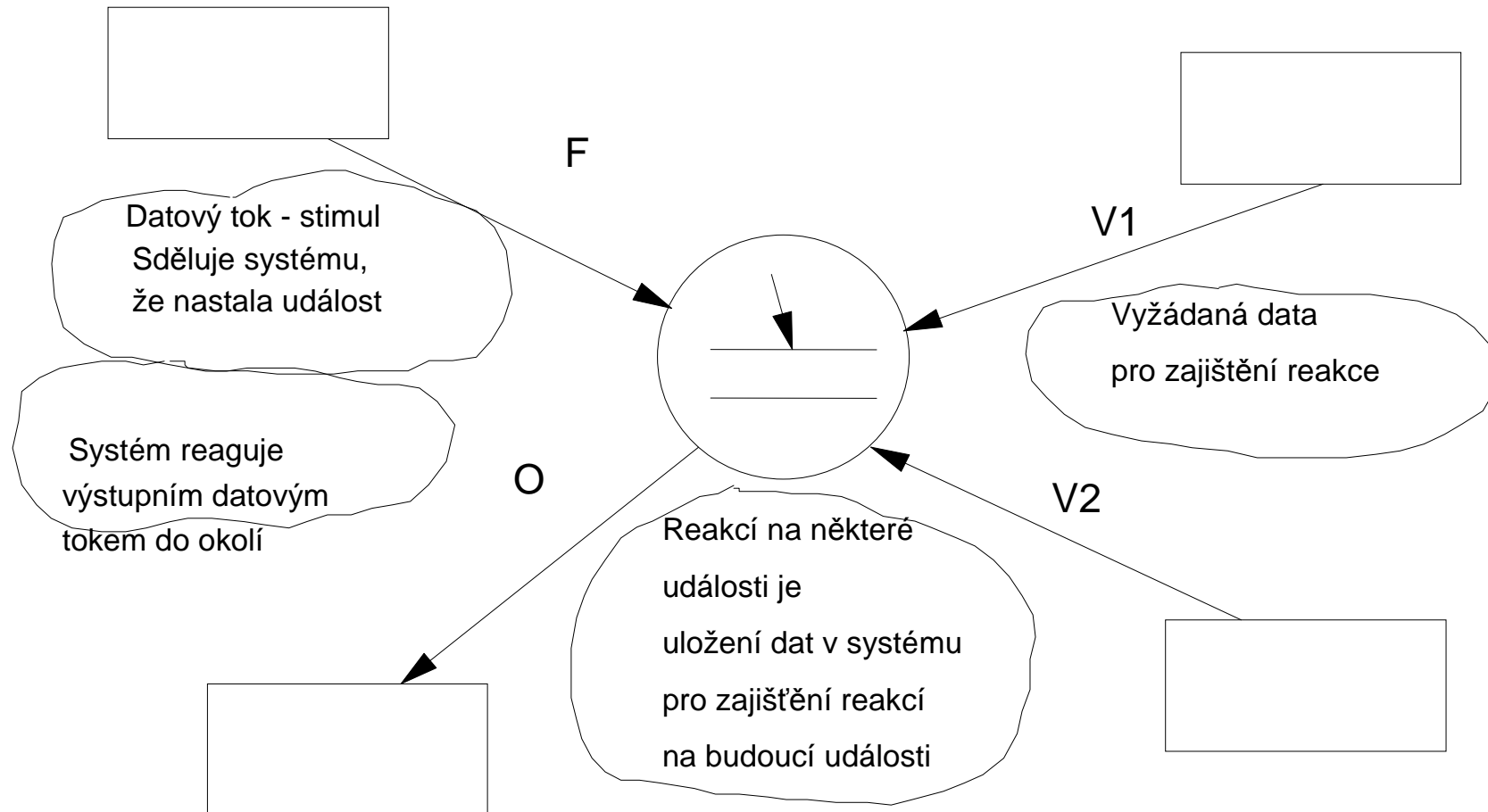
■ *Dle výstupních datových toků (výstupy -> procesy -> vstupy)  
+ kompozice vyšších úrovní a dekompozice na nižší úrovně*

nebo

■ *Dle událostí (událost -> proces -> vstupy+výstupy)  
+ kompozice vyšších úrovní a dekompozice na nižší úrovně*



# Událost -> stimul -> reakce



# Event Partitioning Approach



Pro každou UDÁLOST vytvořit PROCES



Každý PROCES pojmenovat podle REAKCE systému na událost



Ke každému procesu doplnit VSTUPY a VÝSTUPY a případně DATA STORY.

"Jaká data funkce potřebuje, co je jejím výstupem ?"



KONTROLA KONSISTENCE

t.j. balancování výsledku s kontextovým diagramem.



KOMPOZICE MEZIÚROVNĚ (úroveň 0)

přístupem INFORMATION HIDING (skrývání Data Storů)

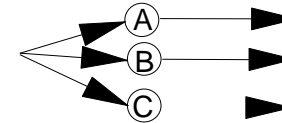
- mezi funkcemi vyhledat LOKÁLNÍ DATA STORY

- tento DS se svými funkcemi tvoří FUNKCI VYŠŠÍ UROVNĚ  
(vytvoření diagramu vyšší úrovně)

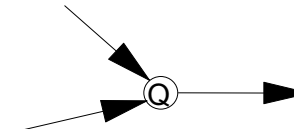
- rozpustit původní diagram do SUBDIAGRAMŮ

Poznámky:

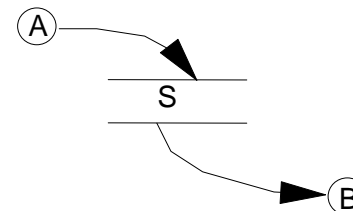
\* 1 událost & různé reakce



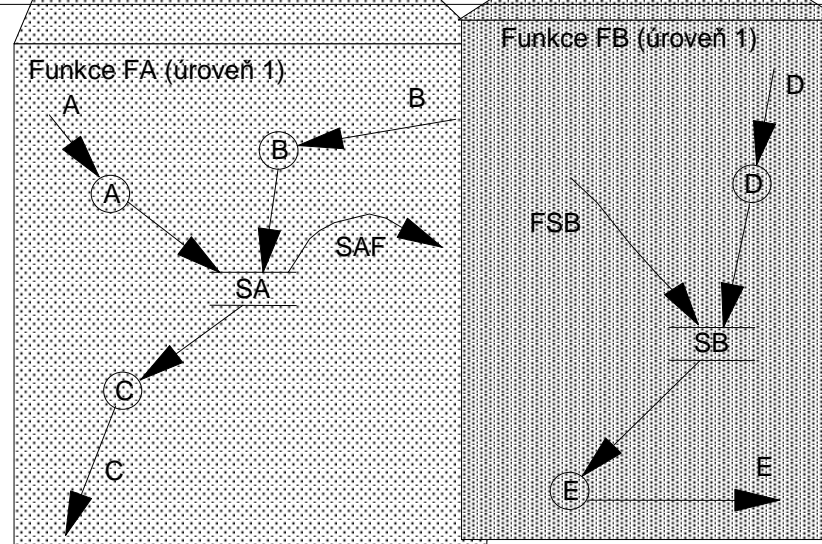
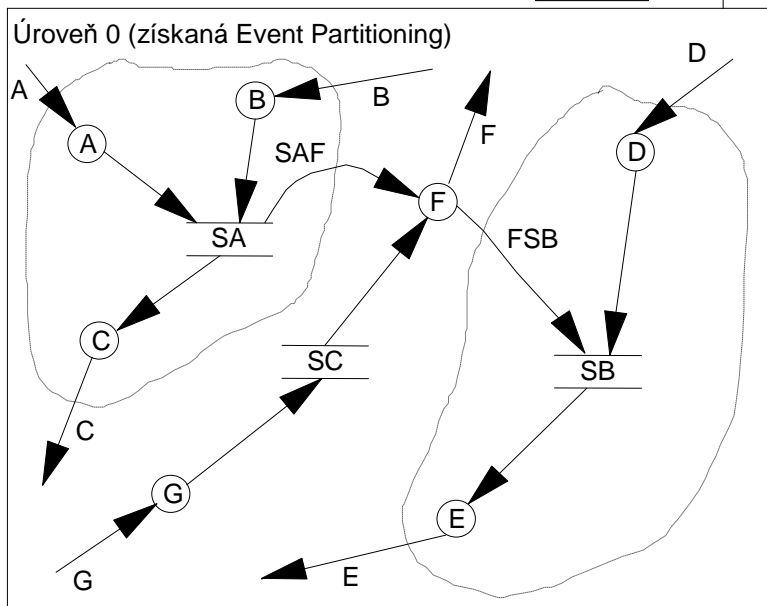
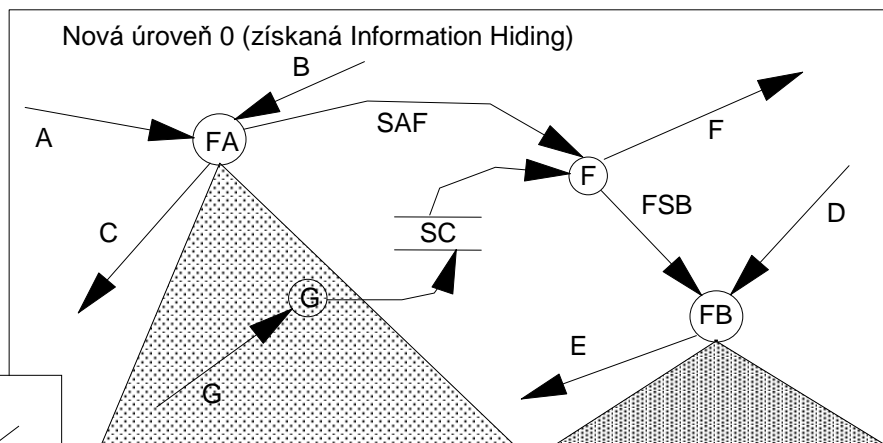
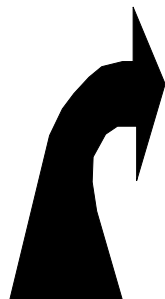
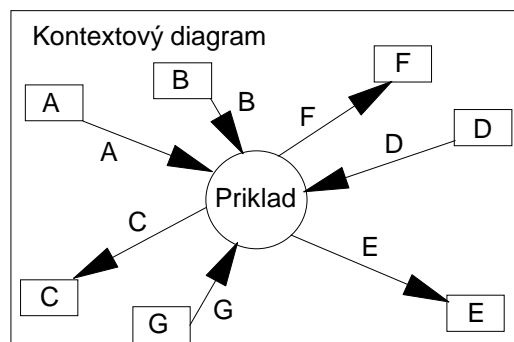
\* Více událostí & shodná reakce



\* Data Store je PŘIROZENOU FORMOU komunikace nesynchronizovatelných procesů. Jde o ESENCIÁLNÍ DS !



# Postup tvorby funkčního modelu ( Kompozice meziúrovně - Information Hiding )

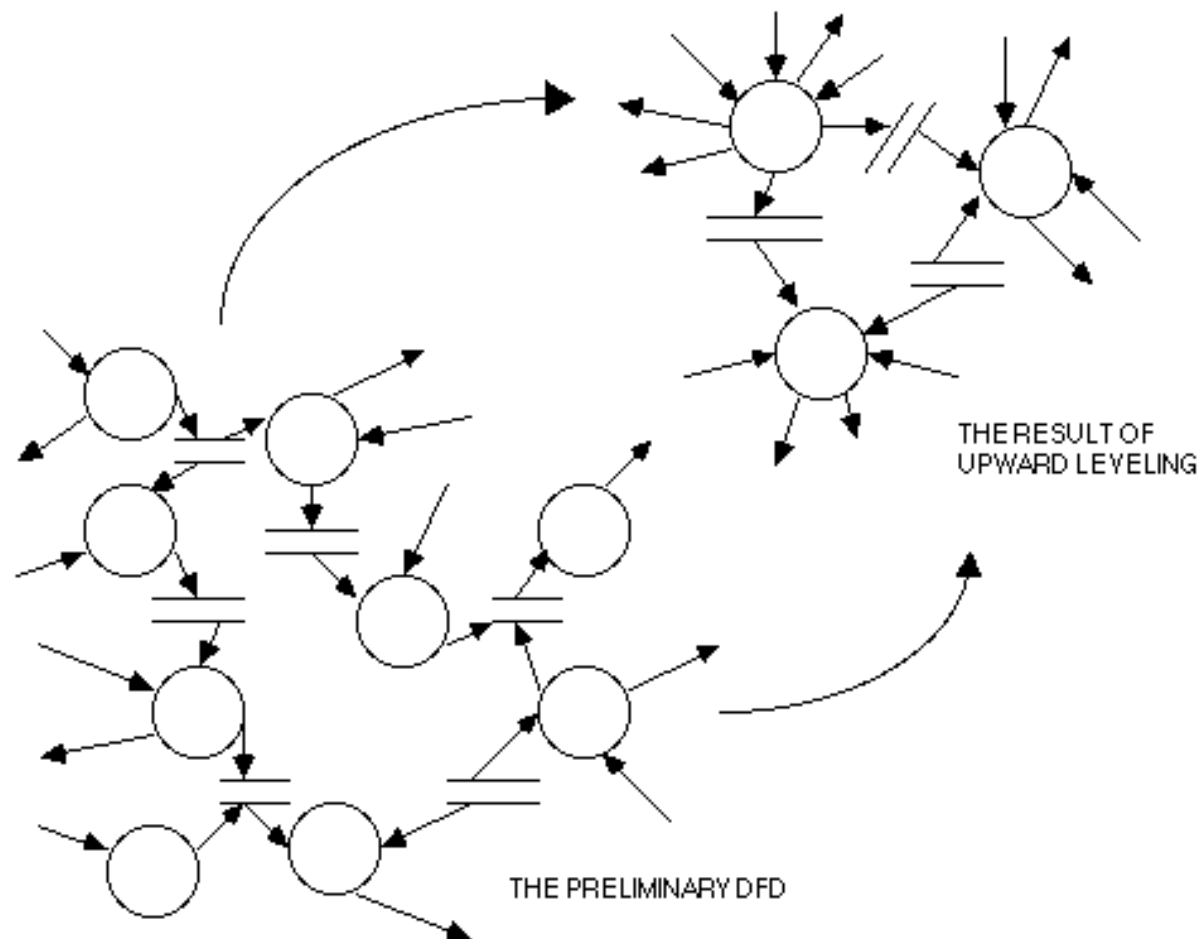


(c) MEWA, 1990

(c) MEWA, 1994



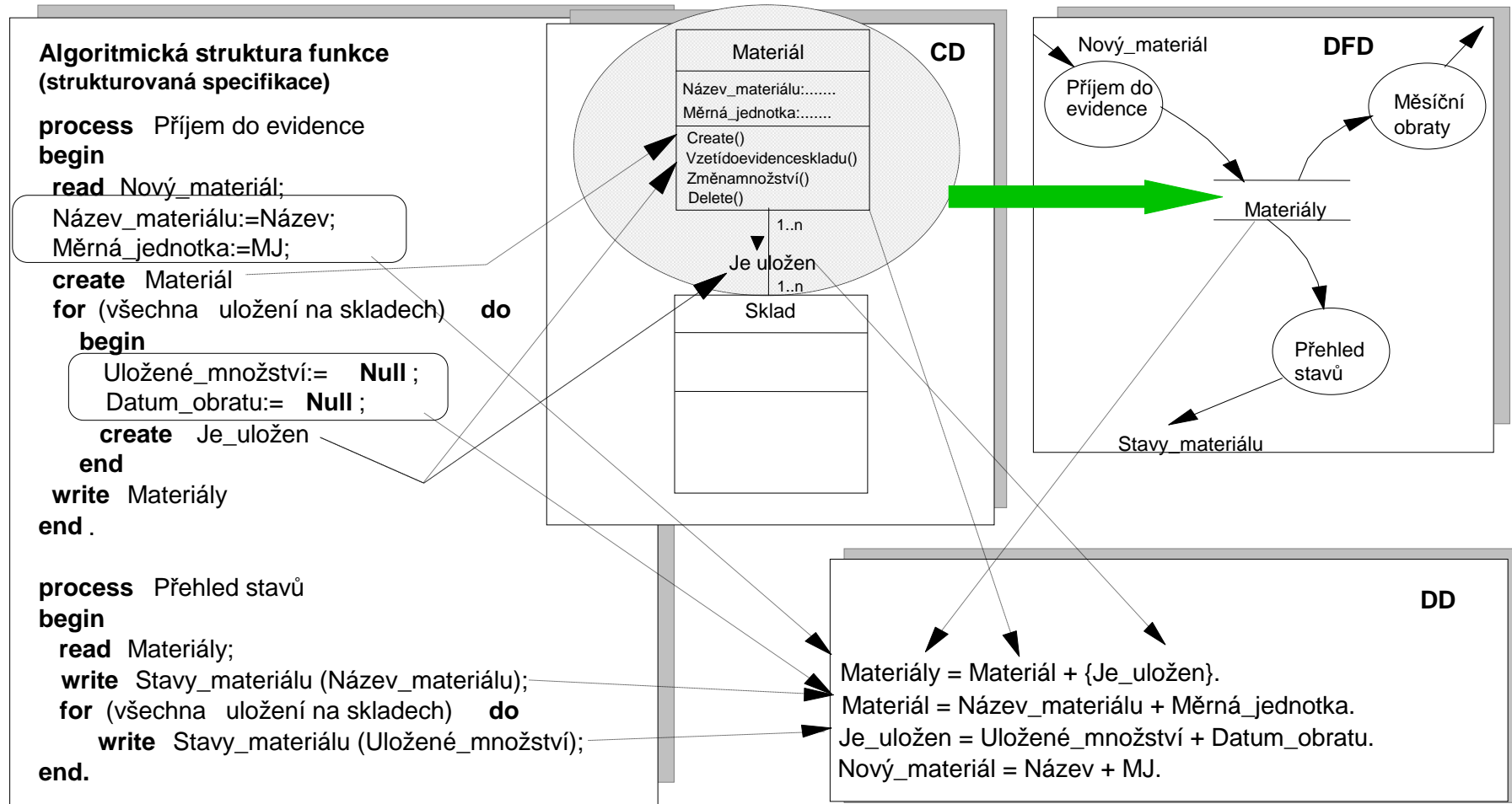
# Tvorba abstraktního DFD skládáním z detailů



# Provázání DFD s objekty

- Každý elementární Datastore v DFD musí být v CD zastoupen jako třída, nebo asociace, anebo kombinace obojího.
- Atributy každého elementárního Datastore z DFD musí být datovou strukturou atributů tříd, jimiž je tento Datastore v CD zastoupen.
- Metody každé elementární funkce z DFD musí být algoritmickou strukturou metod tříd, jimiž jsou v CD zastoupeny Datastory, spojené datovými toky s touto funkcí

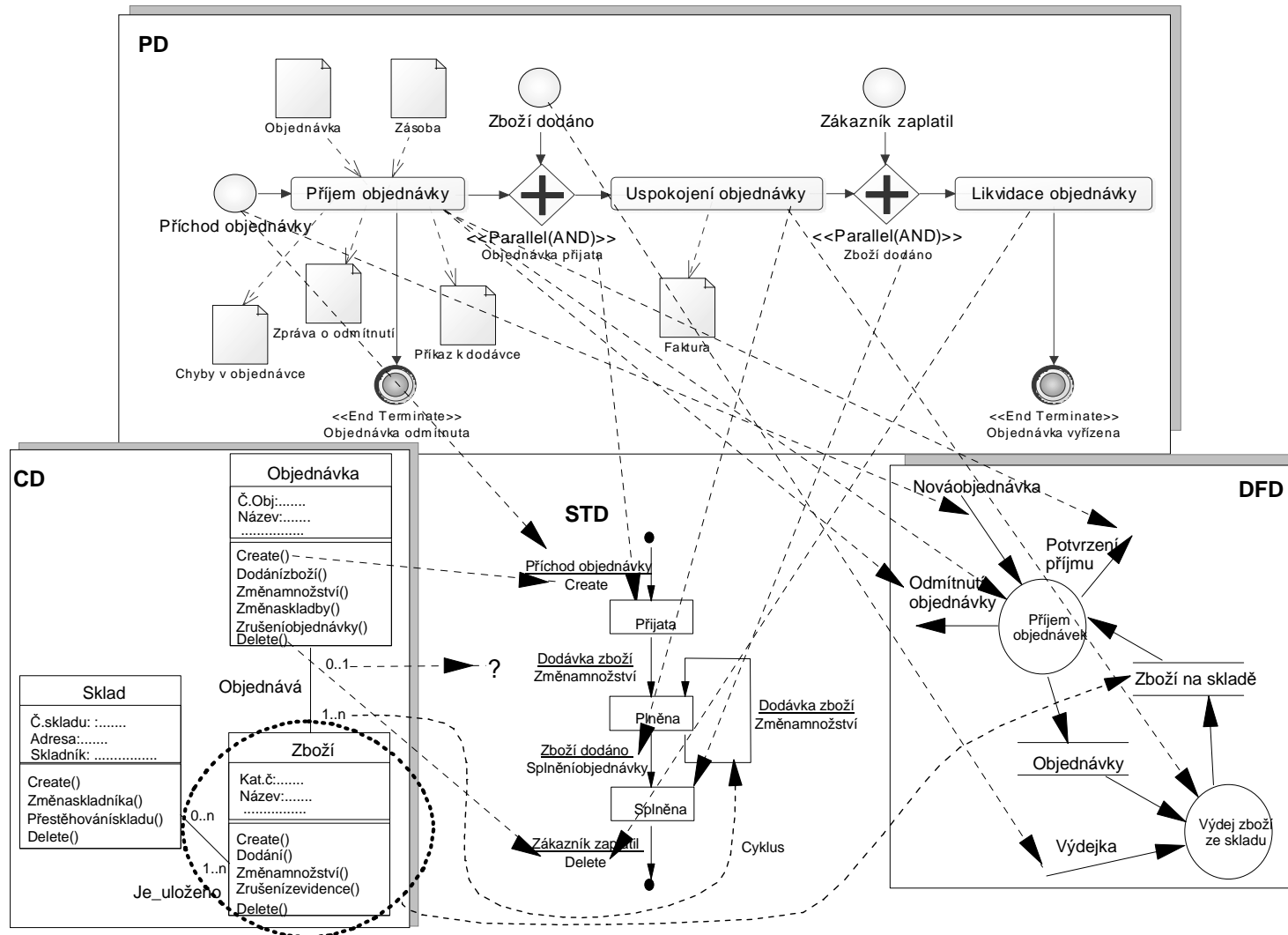
# Příklad provázání DFD s objekty



# Provázání DFD s procesy

- Každý proces má vazbu alespoň na 1 funkci
- Každá funkce má vazbu alespoň na 1 proces
- Každá událost v procesním modelu má vazbu na vstupní tok v DFD
- Každý elementární vstupní datový tok v DFD od terminátoru (tj. zvnějšku systému) musí odpovídat nějaké události, specifikované v popisu nějakého (nějakých) business procesu (procesů) v PD.
- Každý stav každého procesu v PD musí korespondovat s některým(i) elementárním(i) Datastorem(y) v DFD a naopak každý elementární Datastore v DFD musí korespondovat s některým(i) stav(y) procesů(ů) v PD. Jde o korespondenci M:N.

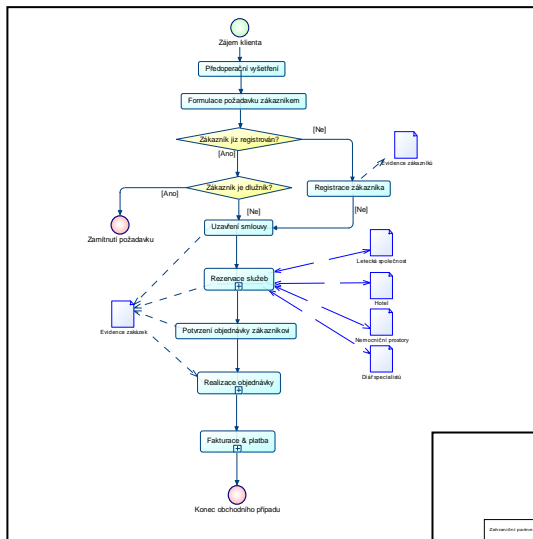
# Příklad provázání DFD s procesy a objekty



Shrnutí konzistence procesů,  
objektů a funkcí

# Přehled analytických modelů

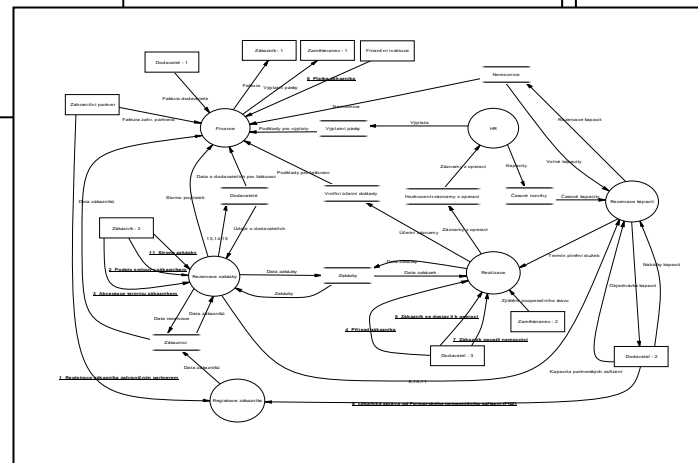
## Model podnikových procesů (Process Diagram)



Údlosti, akce a jejich kontext

Produkty, vstupy, výstupy,  
aktéři, business omezení  
procesů (životní cykly  
objektů)

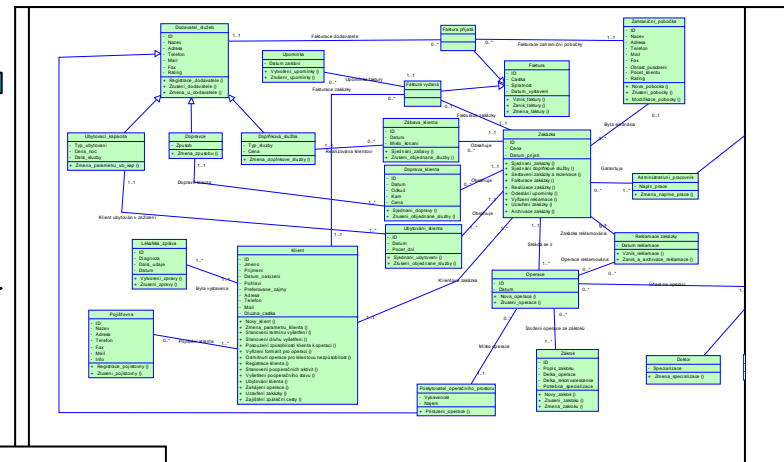
Účelové kombinace ŽC objektů,  
kontext chování objektů



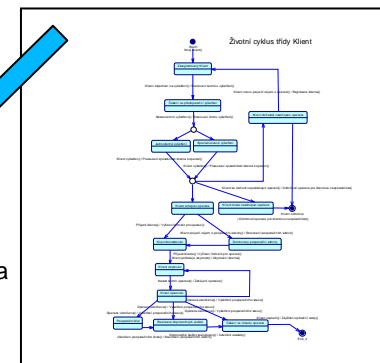
## Model funkcí (Data Flow Diagram)

## Model objektů

## (Class Diagram, State Charts)



Údlosti, data a  
jejich kontext



# Vztahy mezi modely a nástroji analýzy

## *Vztahy mezi nástroji ve funkčním modelu*

### 1. Vztahy mezi DFD a Data Dictionary

- \* Každý DATOVÝ TOK a DATA STORE musí být definován v DATA DICTIONARY.
- \* Každý DATOVÝ PRVEK a DATA STORE definovaný v DD musí být použit v DFD.

### 2. Vztahy mezi DFD a Specifikací algoritmu

- \* Každá FUNKCE v DFD musí být buď popsána jako jednoduchý algoritmus, nebo musí představovat DFD NIŽŠÍ ÚROVNĚ (nikoliv obojí).
- \* Každá SPECIFIKACE ALGORITMU musí být obsažena coby FUNKCE NEJNIŽŠÍ ÚROVNĚ v některém DFD.
- \* Každému VÝSTUPNÍMU DATOVÉMU TOKU z funkce v DFD musí ve specifikaci algoritmu odpovídat WRITE a každému VSTUPNÍMU READ.

### 3. Vztahy mezi Specifikací algoritmu a DFD & Data Dictionary

Každý ODKAZ NA DATA VE SPECIFIKACI ALGORITMU musí představovat buď:

- \* NÁZEV DATOVÉHO TOKU, nebo DATA STORE, spojeného se specifikovaným procesem,
- \* LOKÁLNÍ DATA specifikovaného procesu NEBO
- \* NÁZEV KOMPONENTY DATOVÉHO TOKU, NEBO DATA STORU, spojeného se specifikovaným procesem tak, jak je tato komponenta uvedena v DD.

### 4. Vztahy mezi DD a DFD & Specifikací algoritmu

Každý DATOVÝ PRVEK v DD musí být použit:

- \* ve SPECIFIKACI ALGORITMU, nebo
- \* v DFD, anebo
- \* při popisu JINÉHO DATOVÉHO PRVKU.



# Vztahy mezi modely a nástroji analýzy

## Vztahy mezi nástroji procesního, objektového a funkčního modelu

### 5. Vztahy mezi CD a DFD & Specifikací algoritmu

- \* Každý elementární DATASTORE v DFD musí být v CD zastoupen jako OBJEKT (třída), nebo VZTAH, anebo KOMBINACE OBOJÍHO.
- \* Každý elementární DATASTORE musí být v DD popsán jako struktura atributů tříd z CD.
- \* SPECIFIKACE ALGORITMU musí obsahovat operace CREATE a DELETE pro každou TŘÍDU a VZTAH, uvedený v CD a těmto operacím musí odpovídat příslušné metody této třídy.
- \* DATOVÉ ELEMENTY (atributy) každého TŘÍDY v CD musí být NASTAVENY některým procesem v DFD a také některým POUŽITÝ a těmto operacím musí odpovídat příslušné metody této třídy.

### *Elementární DataStore*

je DataStore u něž není objektivní důvod k jeho rozkladu do podrobnější struktury DataStoreů.

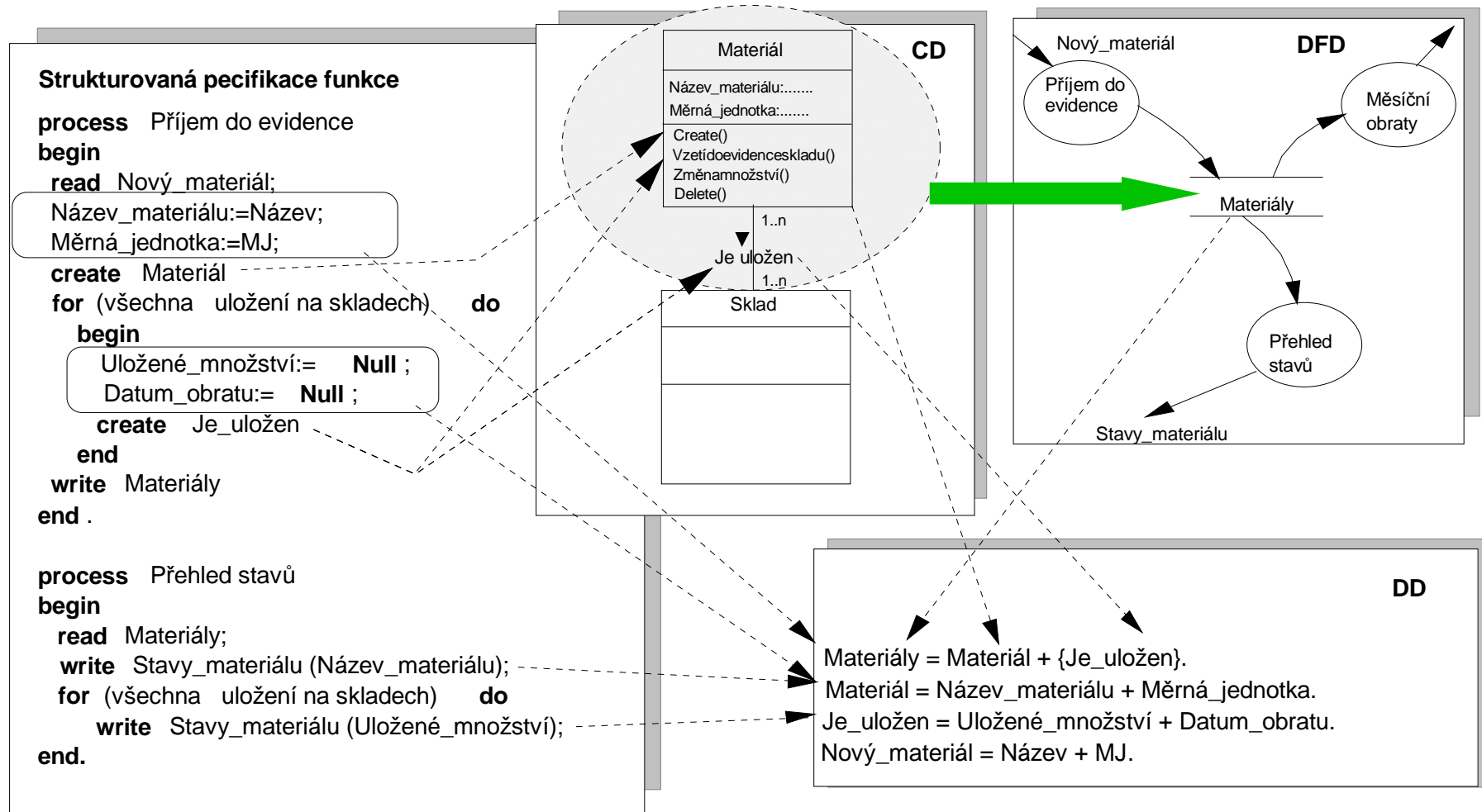
### *Primitivní třída*

je třída, jejíž životní cyklus je natolik jednoduchý, že jej lze popsat typově (zrození - existence - zánik), přičemž není objektivní důvod specifikovat strukturu existence třídy podrobněji, nežli jako obecnou možnost změny atributu(ů) třídy.

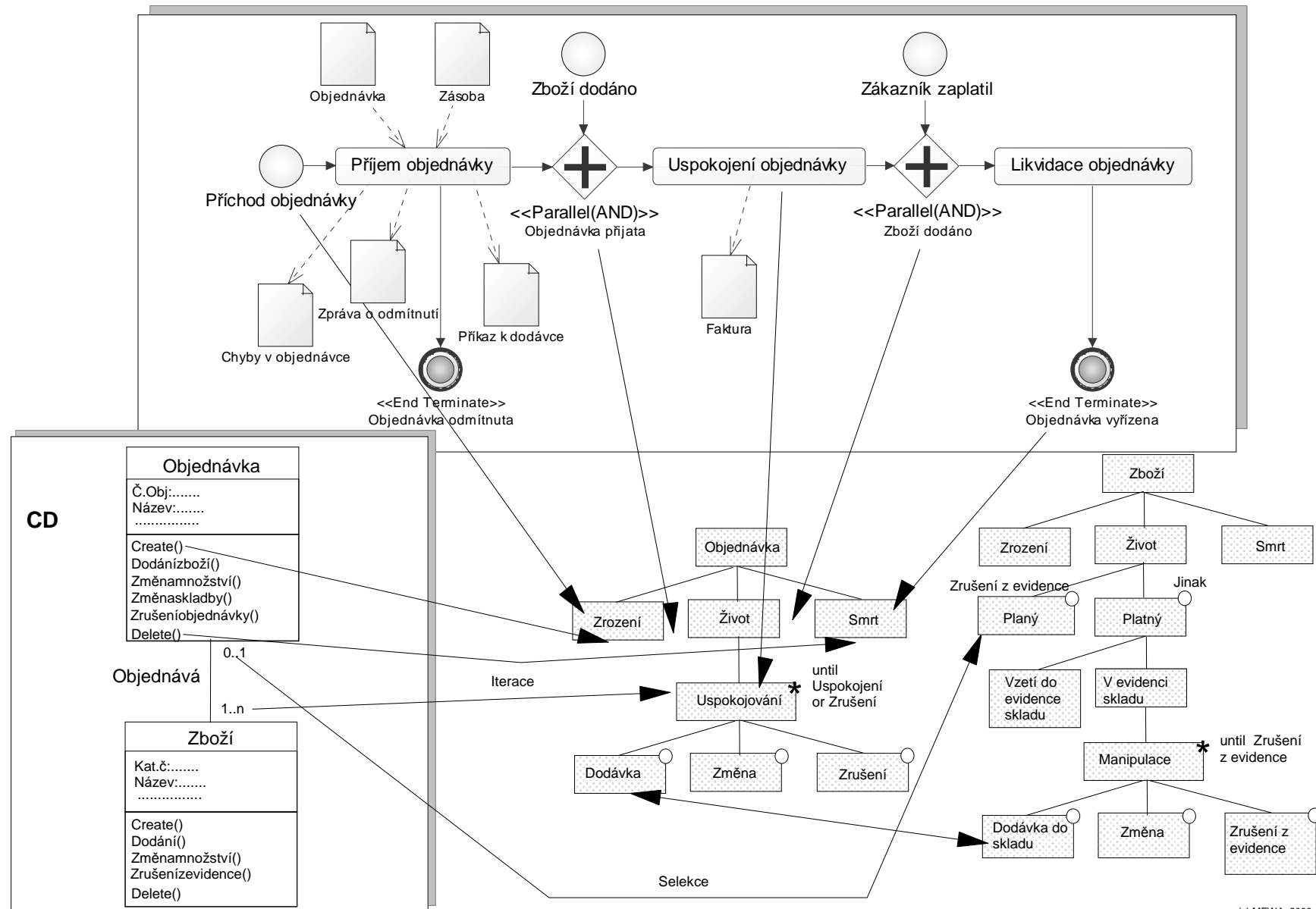
### 6. Vztahy mezi Diagramem Tříd (CD) a STD a DFD a Modelem Procesů (PD)

- \* Každá TŘÍDA v CD musí mít svůj STD, popisující její životní cyklus (s výjimkou "primitivních tříd").
- \* Každá PODMÍNKA v STD odpovídá VSTUPNÍMU TOKU v DFD, UDÁLOSTI v PD a metodě třídy.
- \* Každá AKCE v STD odpovídá VÝSTUPNÍMU TOKU v DFD, ČINNOSTI v PD a METODĚ třídy.

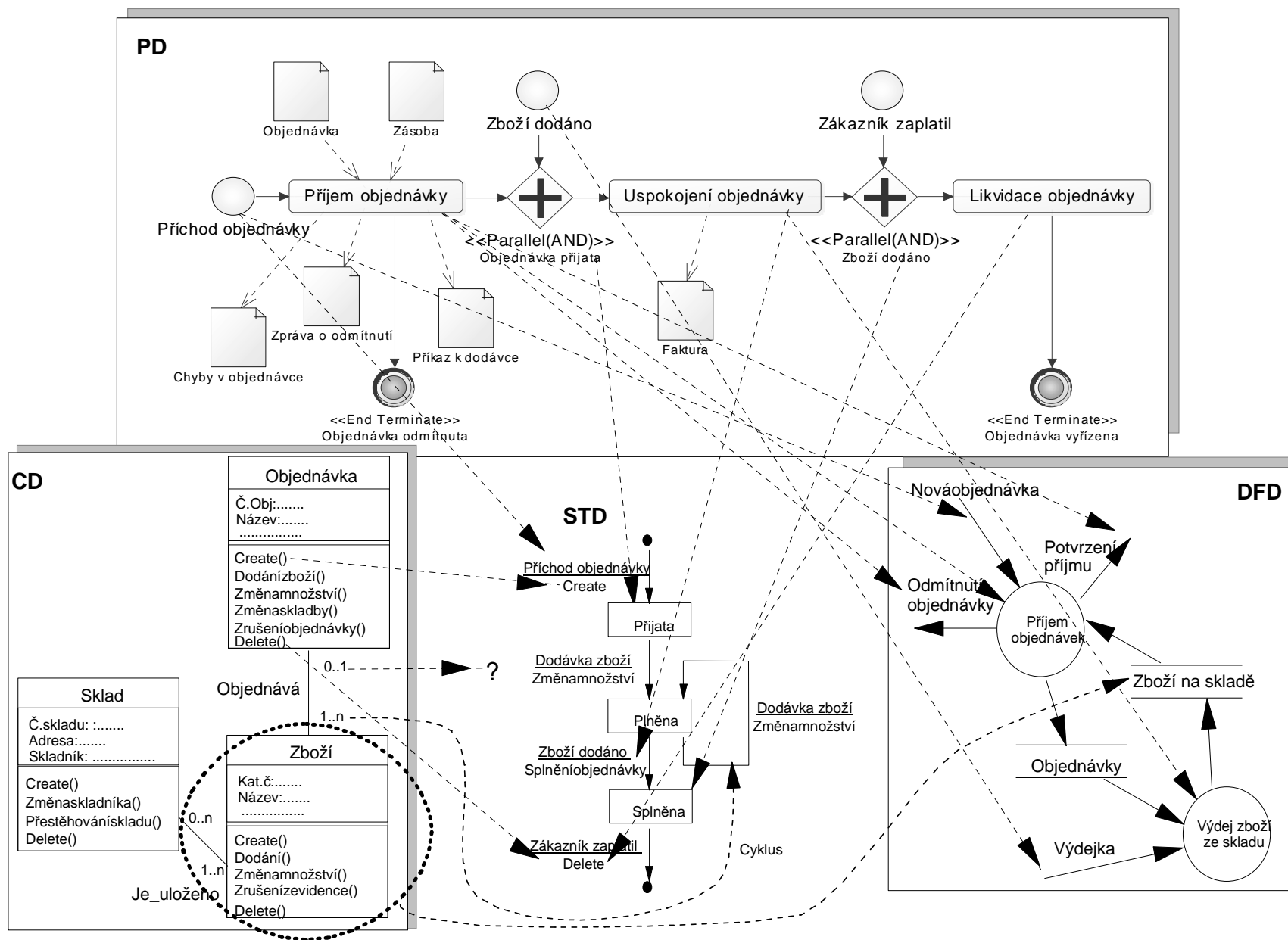
# Vztahy mezi funkčním a objektovým modelem



# Vztahy mezi procesním a objektovým modelem



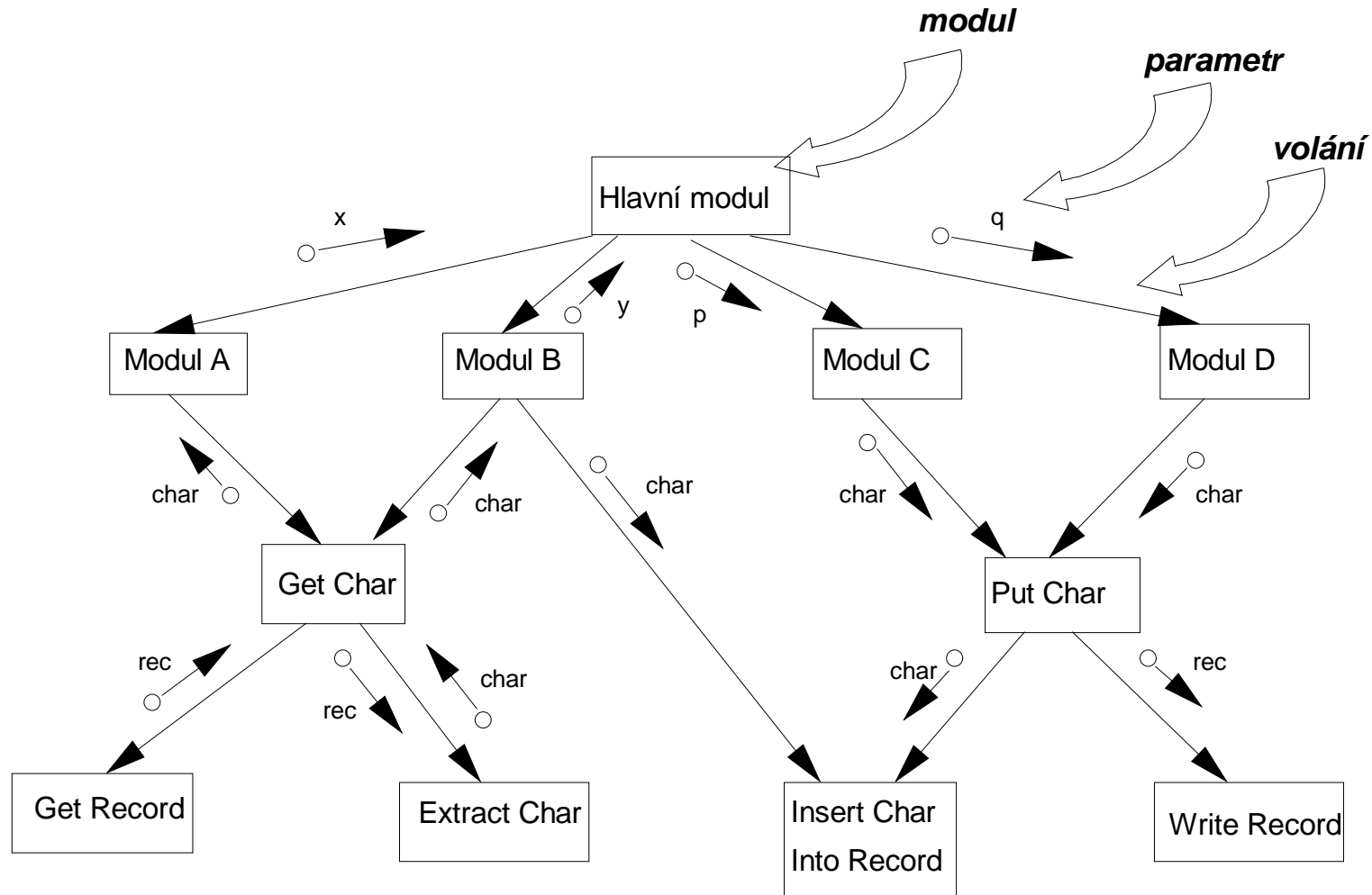
## Vztahy mezi procesním, objektovým a funkčním modelem prostřednictvím STD



**Design systému**

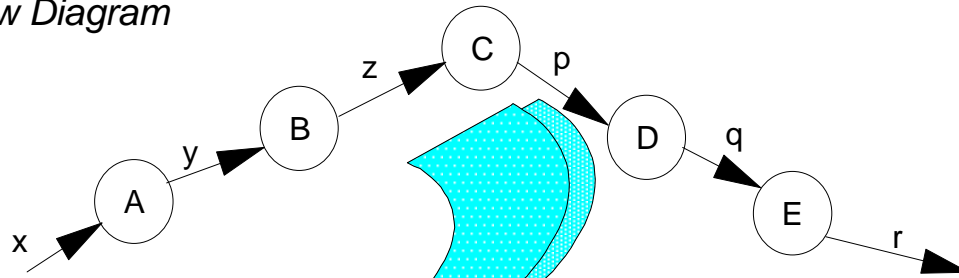
# Model struktury programového systému

## Structure Chart

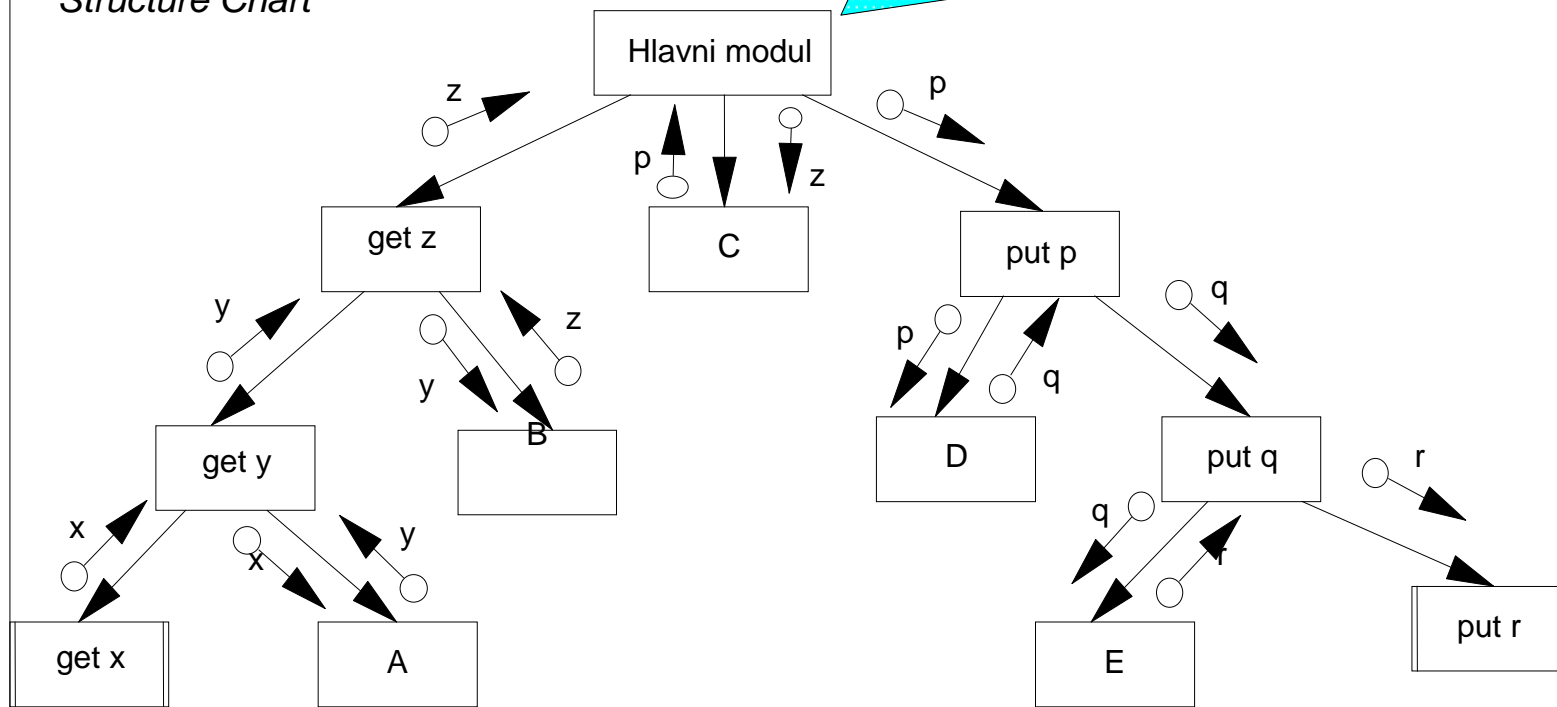


# Transakční a transformační analýza

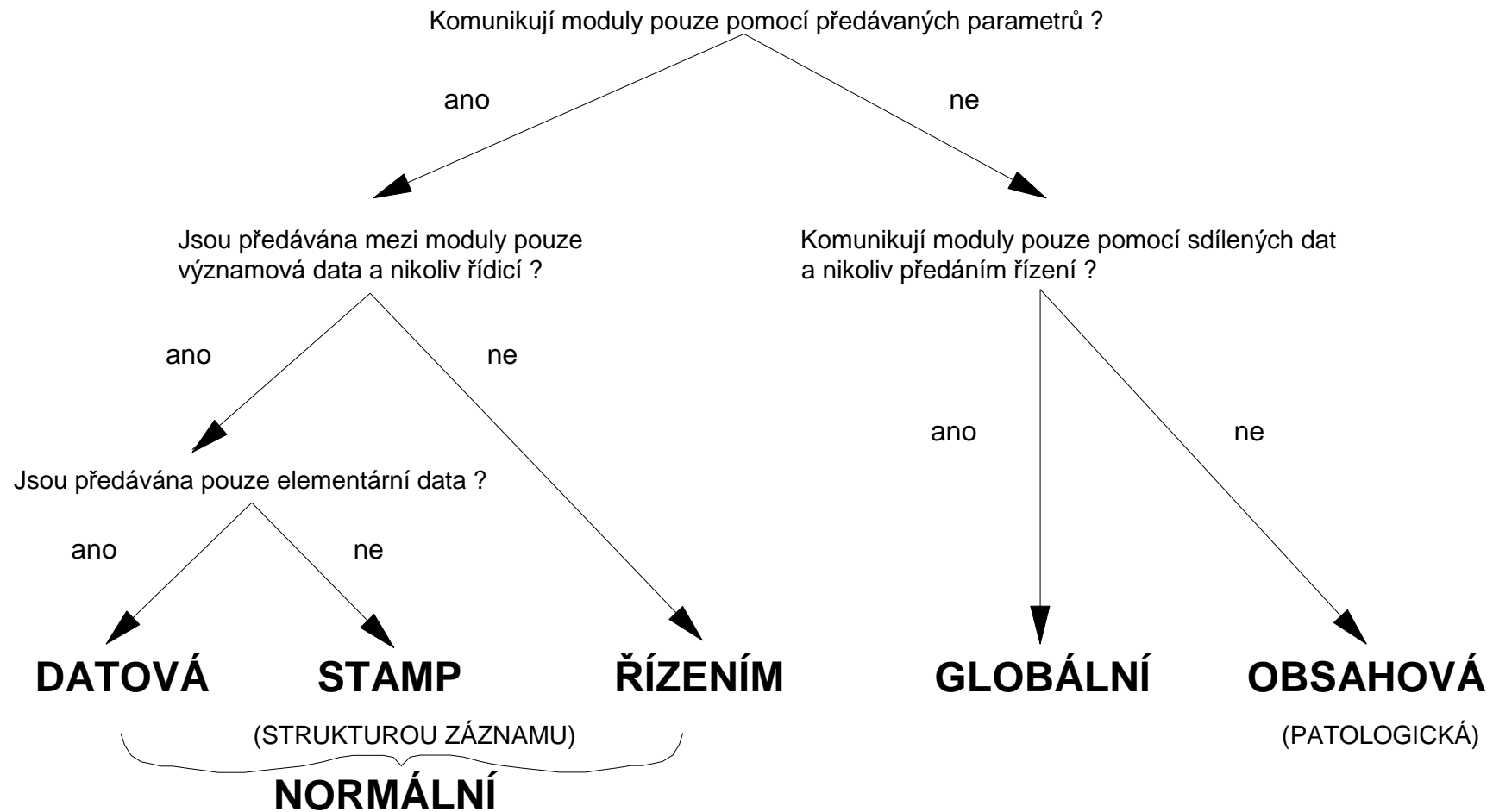
*Data Flow Diagram*



*Structure Chart*



# HODNOCENÍ SPŘAŽENOSTI





# OO Design systému

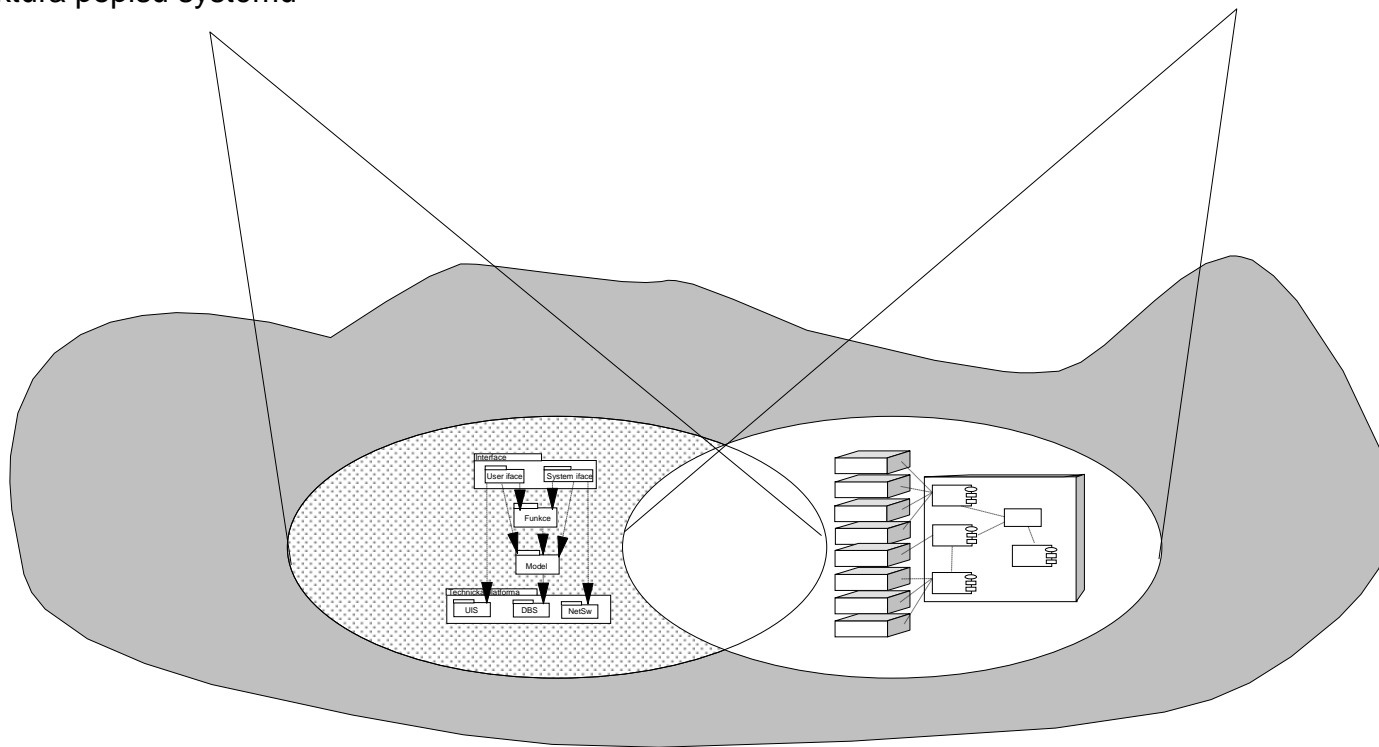
## Komponentová versus procesní architektura

### Architektura komponent:

- třídy
- statické aspekty
- propojení komponent
- logický pohled
- struktura popisu systému
- 

### Architektura procesů:

- objekty
- dynamické aspekty
- koordinace procesů
- fyzický pohled
- struktura chování systému
- 

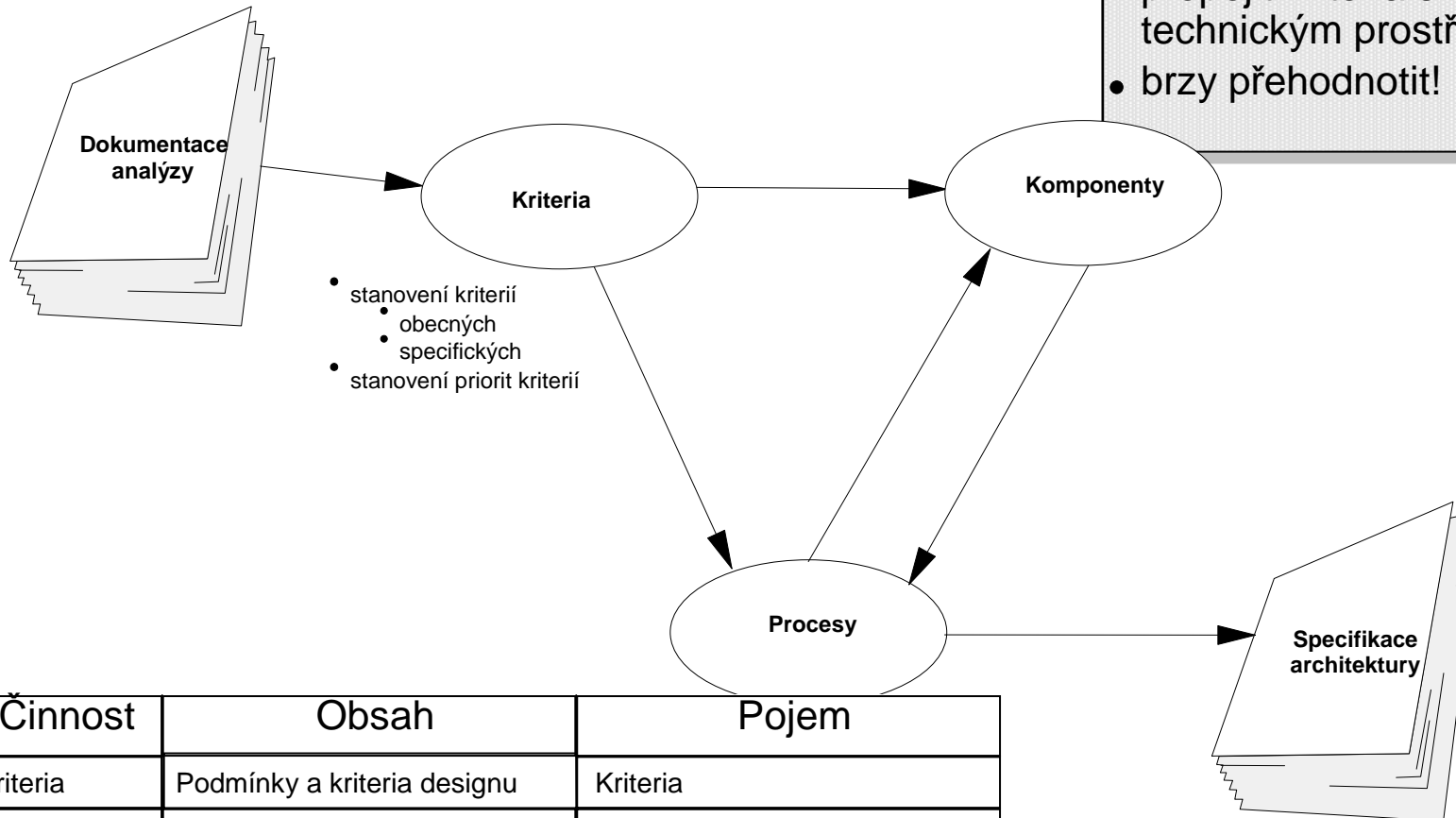


# OO Design systému

Postup architektonického designu

## Principy:

- definovat kriteria a jejich priority!
- propojit kriteria s technickým prostředím!
- brzy přehodnotit!



Činnost	Obsah	Pojem
Kriteria	Podmínky a kriteria designu	Kriteria
Komponenty	Jak je systém strukturován	Komponentová architektura
Procesy	Distribuce a koordinace procesů	Procesní architektura

# OO Design systému

## Kriteria designu

**Kriterium** = žádaná vlastnost architektury

Kriterium	Je měřítkem
Použitelnost	přizpůsobitelnosti systému organizačnímu, provoznímu a technickému kontextu
Bezpečnost	imunity vůči neautorizovanému přístupu k datům a zařízením
Efektivnost	schopnosti ekonomicky využít technickou platformu
Správnost	naplnění uživatelských požadavků
Spolehlivost	naplnění požadované přesnosti výkonu funkcí
Udržovatelnost	nákladů na lokalizaci a opravu chyby
Testovatelnost	nákladů na ujištění, že instalovaný systém správně provádí své určené funkce
Pružnost	nákladů na modifikaci instalovaného systému
Srozumitelnost	úsilí potřebného k příslušnému porozumění systému
Znovupoužitelnost	možnosti použít části systému v jiných systémech
Přenositelnost	nákladů na přenos systému na jinou technickou platformu
Interoperabilita	nákladů na propojení systému s jinými systémy

### Principy:

- dobrý design nemá kritické slabiny!
- dobrý design vyvažuje více kriterií!
- dobrý design je
  - použitelný,
  - pružný,
  - srozumitelný!

### Postup:

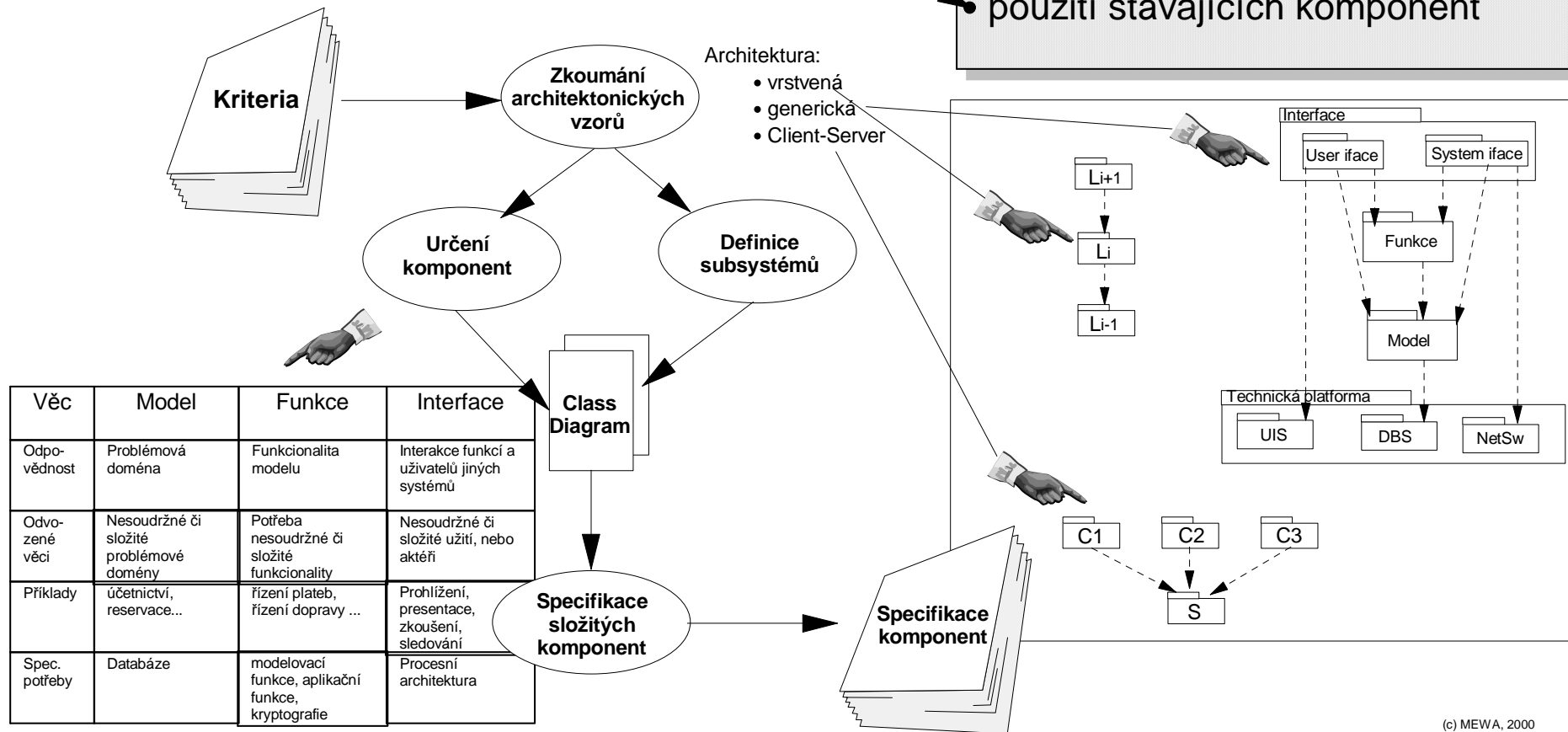
- zvážení obecných kriterií
- analýza specifických podmínek:
  - technických (stávající HW/SW, použití vzorů a komponent, nákup komponent...)
  - organizačních (kontrakty, plán vývoje IS, WBS a obsazení rolí...)
  - personálních (kompetence, zkušenosti věcné a technické...)
- stanovení priorit

# OO Design systému Komponenty

**Komponenta** = souhrn programových částí, tvořících celek s definovanými odpovědnostmi

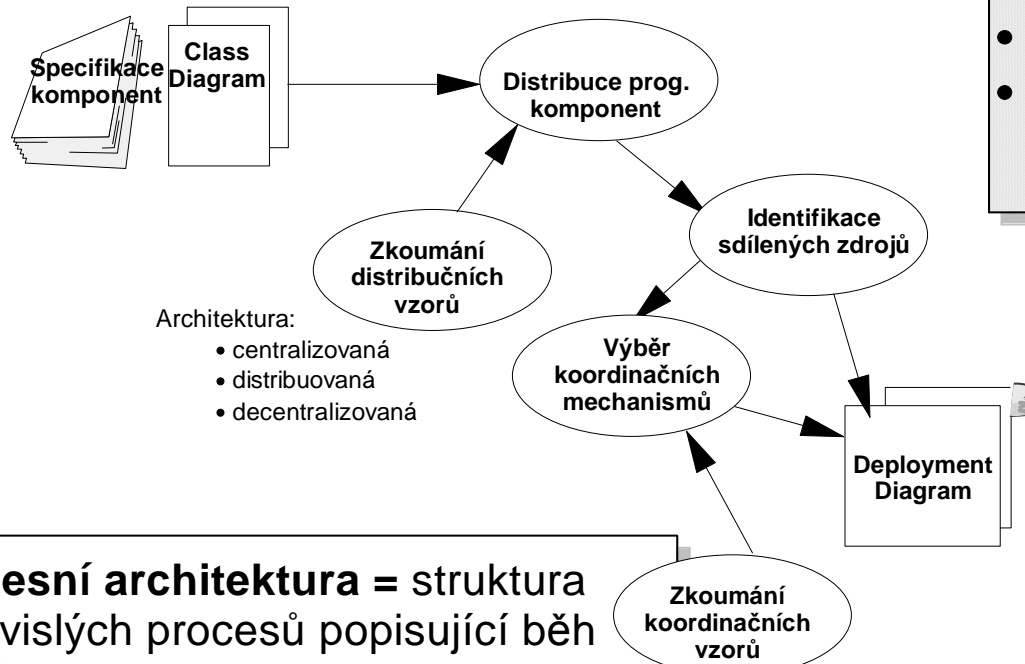
## Principy:

- redukce složitosti rozdělením komponent podle oblastí zájmu (architektonické vzory)
- uvažování stabilních kontextových struktur (stabilní aspekty reality a podmínek práce systému)
- použití stávajících komponent



# OO Design systému

## Procesy



### Principy:

- zaměření na architekturu bez úzkých míst
- distribuce komponent na procesory
- koordinace sdílení zdrojů s aktivními objekty

**Procesní architektura** = struktura nezávislých procesů popisující běh systému

**Procesor** = zařízení schopné provádět program

**Programová komponenta** = fyzický modul programového kódu

**Aktivní objekt** = objekt, přiřazený procesu

### Koordinace:

- určeným monitorem
- centrálním dispečerem
- kritickými hodnotami stavů
- asynchronní výměnou dat (buffering)

