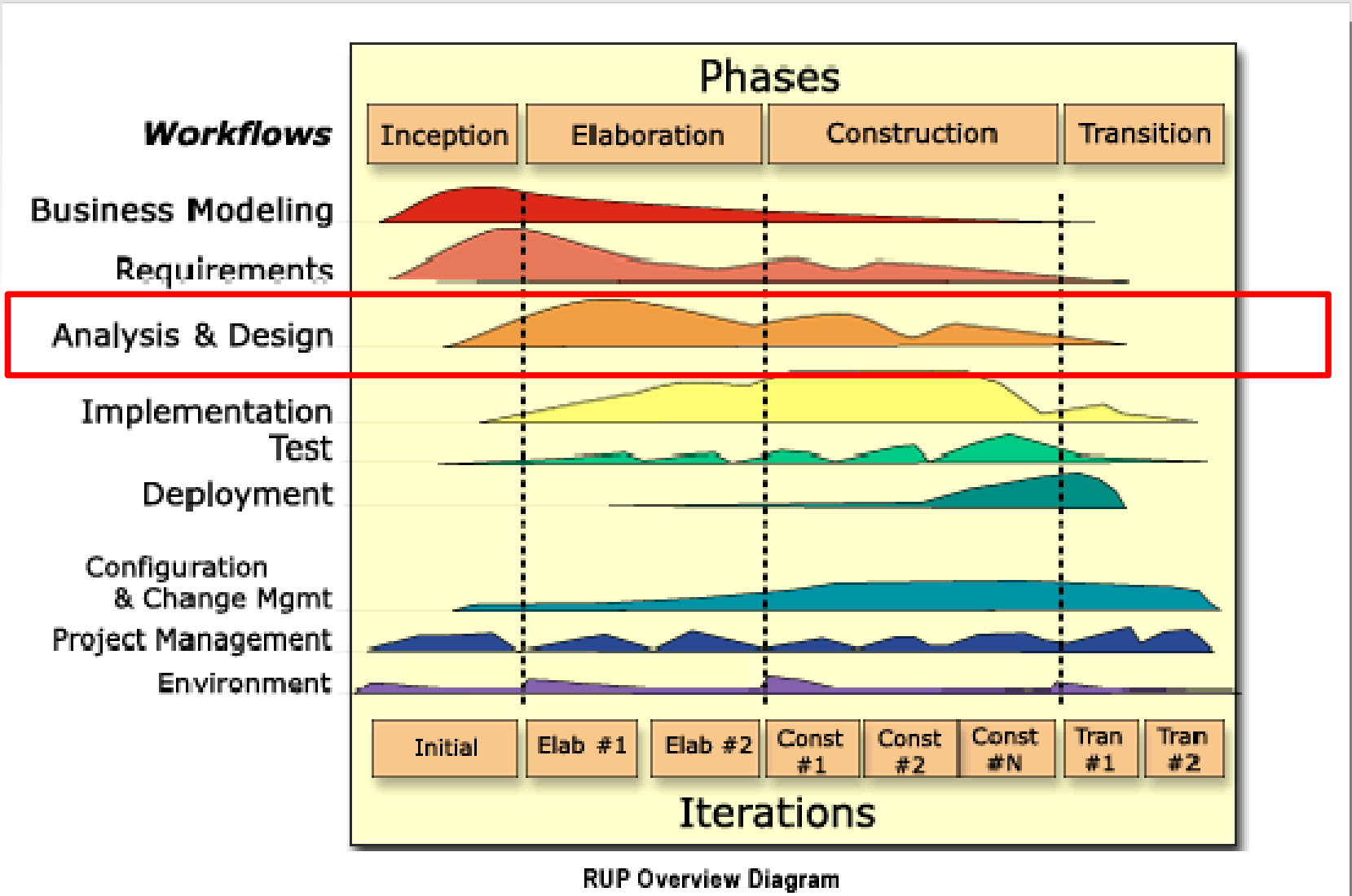
Návrh

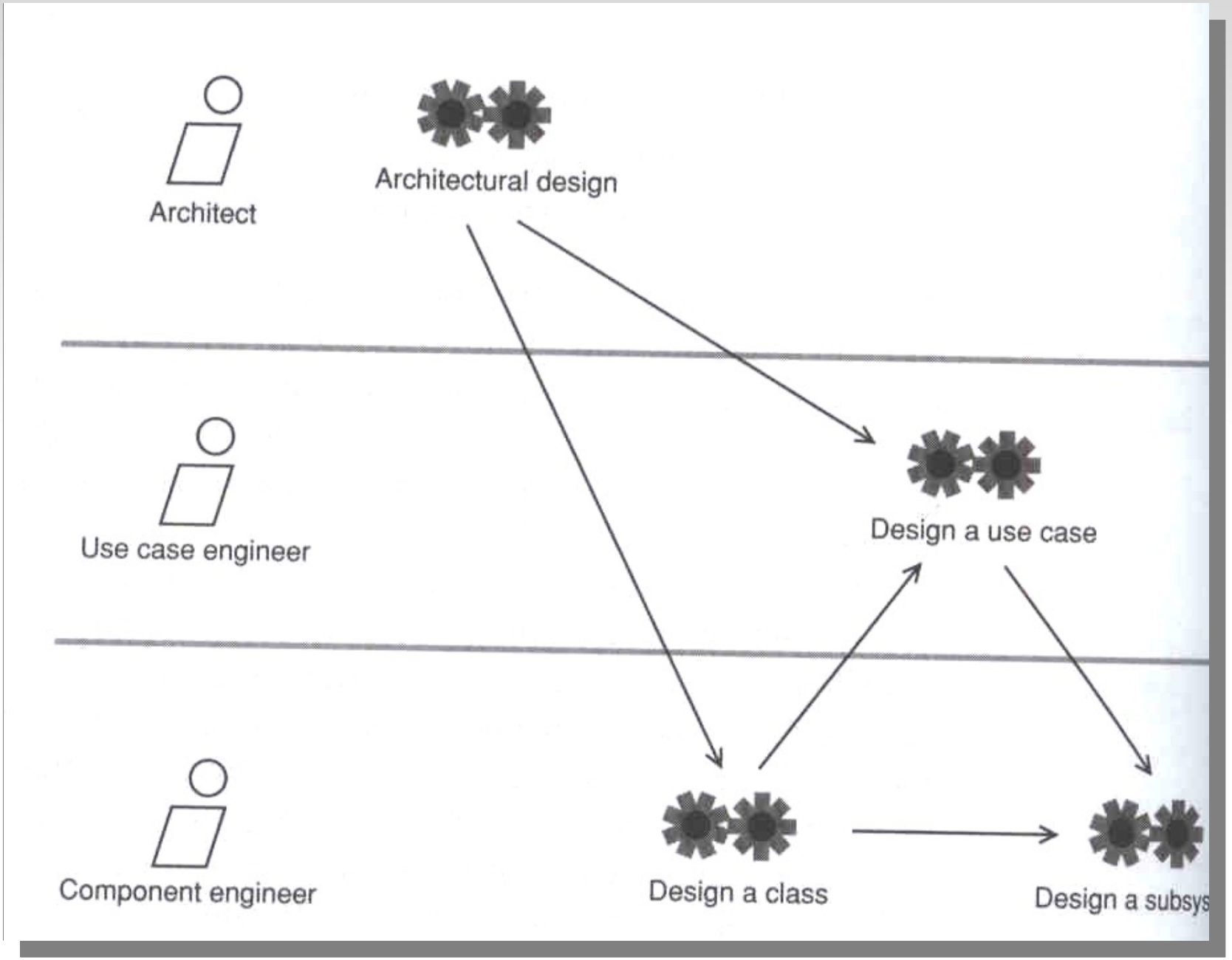
© Radek Ošlejšek
Fakulta informatiky MU
oslejsek@fi.muni.cz

Design workflow

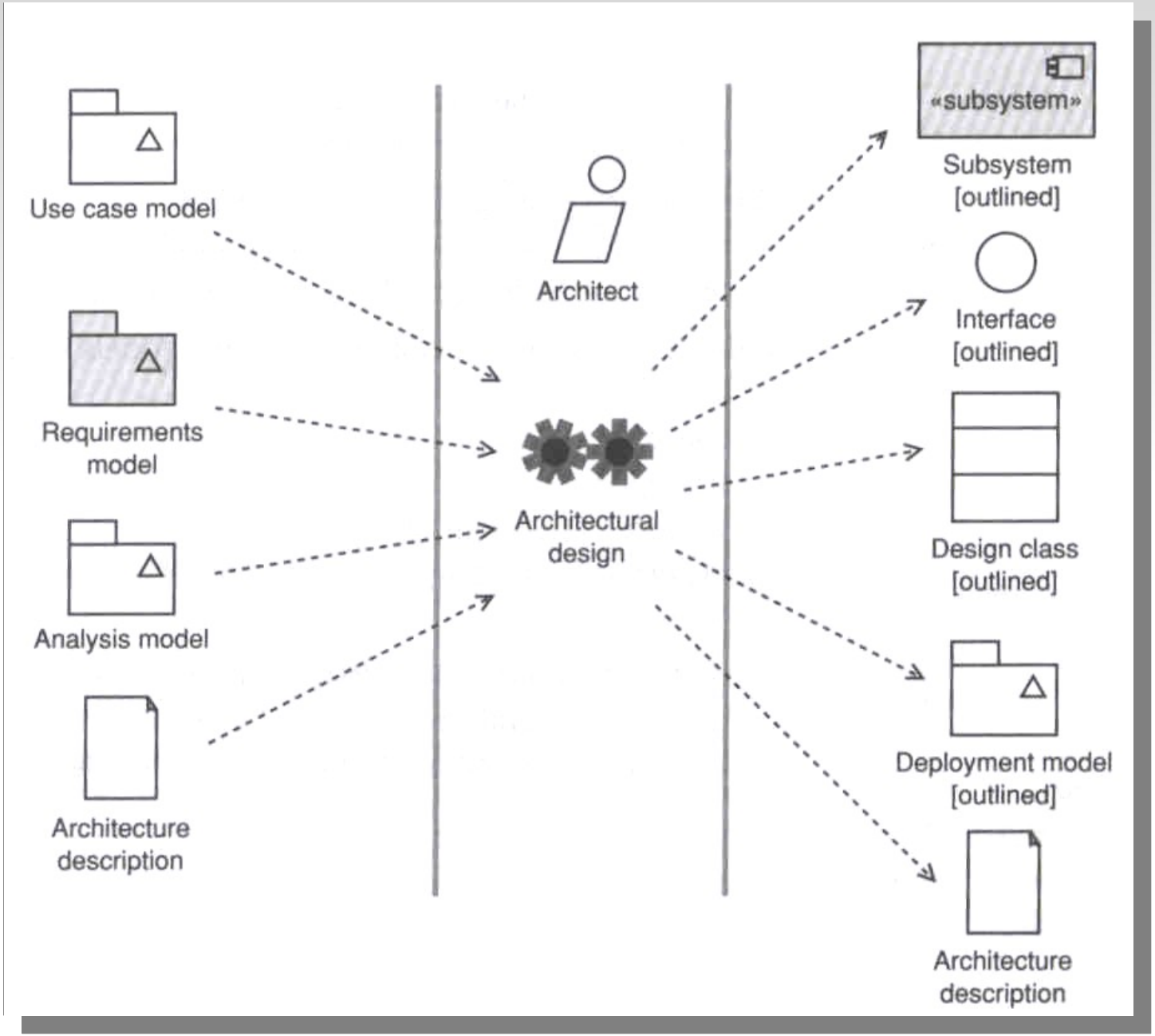


RUP Overview Diagram

Design workflow detail

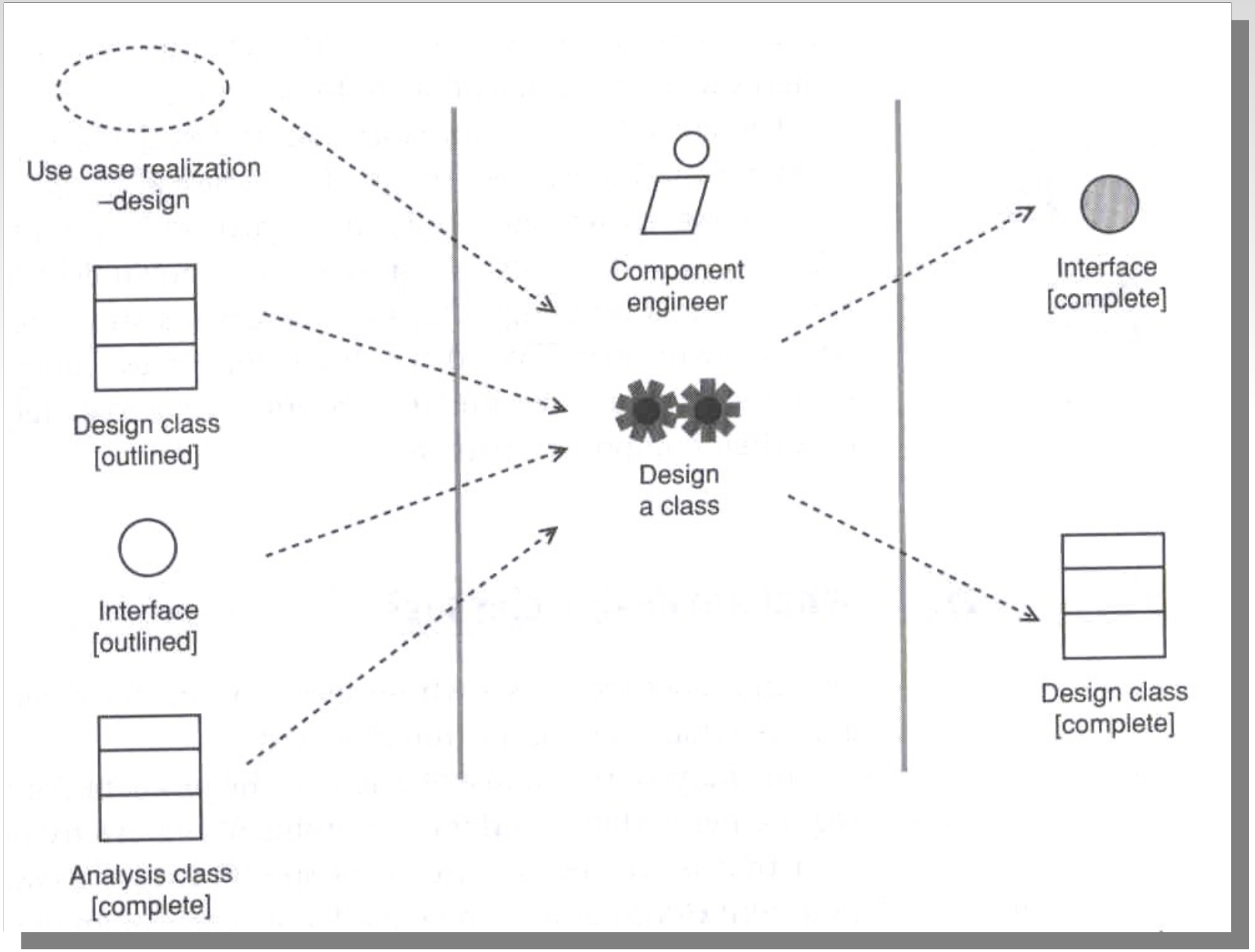


UP aktivita: Architectural design



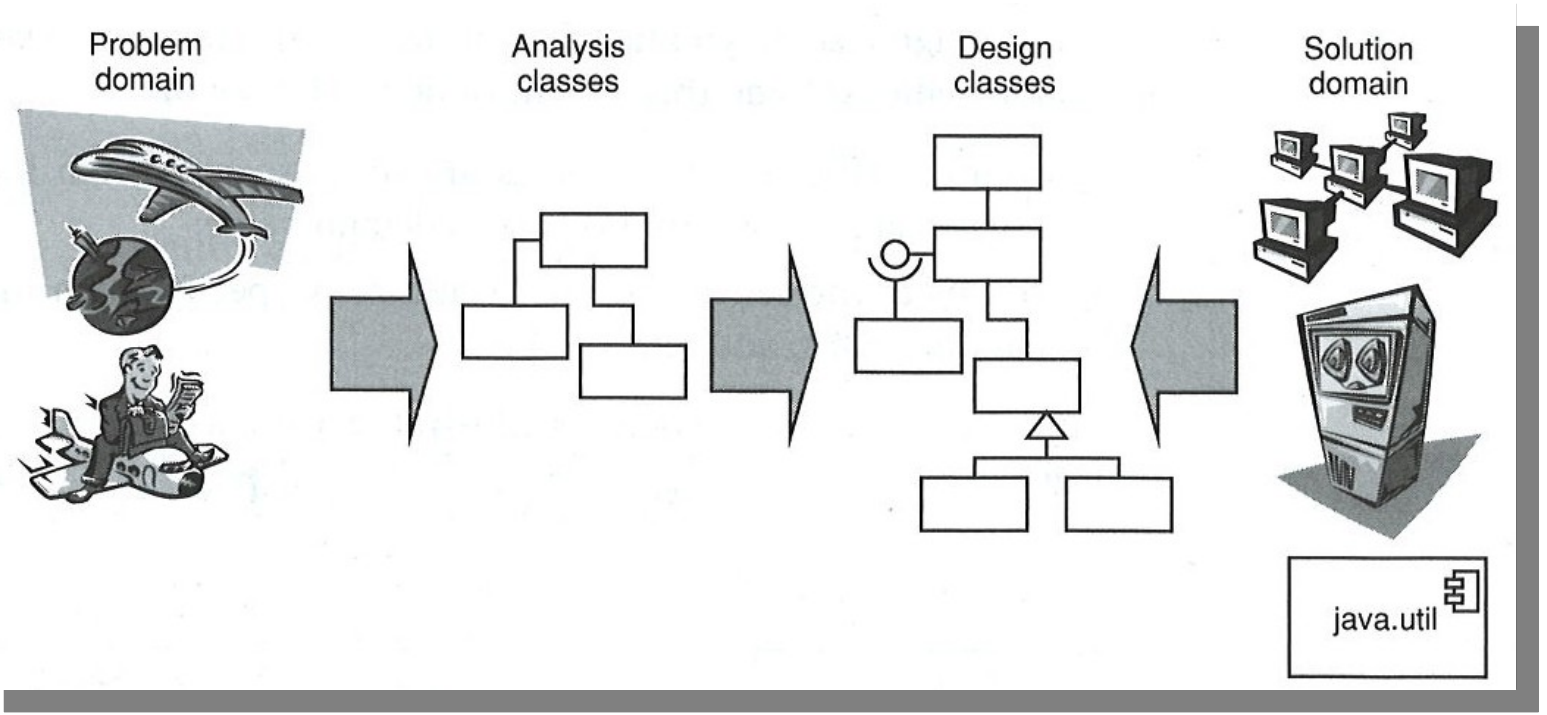
Návrhové třídy
(Design classes)

UP aktivita: Design a class



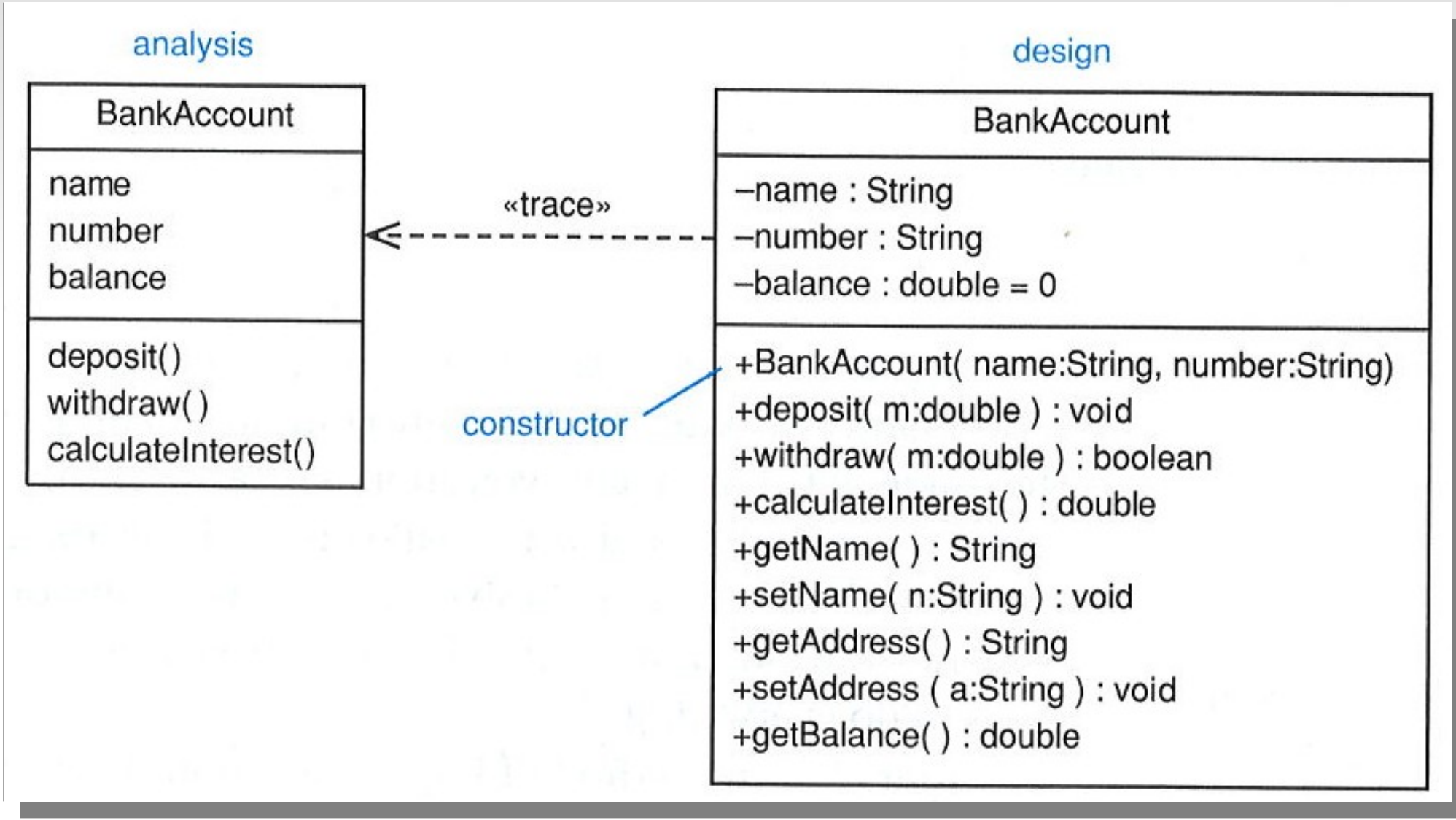
Návrhové třídy – vycházejí z

- „problem domain:
  - rozpracování analytických tříd
  - přidání implementačních tříd
  - rozdělení velkých analytických tříd na menší s jasnější zodpovědností
- „solution domain“
  - zapojení existujících knihoven, middlewaru, GUI a obecných znovupoužitelných elementů (*String*, *Time, Date*, kolekce apod.)



Vlastnosti návrhových tříd

- Obsahují všechny nezbytné detaily



Vlastnosti návrhových tříd (pokr.)

- Úplnost
  - veřejné operace definují kontrakt mezi třídou a jejími klienty
  - musí poskytovat všechny nezbytné operace
  - neměla by poskytovat zbytečné operace, o které nikdo nestojí
- Jednoduchost
  - operace by měly být jednoduché, atomické a unikátní (nemělo by existovat více způsobů, jak dosáhnout stejného cíle)
  - porušuje se v odůvodněných případech, např. z důvodu efektivity kódu
- Vysoká soudržnost (koheze, zodpovědnost)
  - každá třída by měla mít jednu dobře definovanou abstrakci a co nejméně vlastností
- Co nejmenší propojení s ostatními třídami (coupling)

Má smysl udržovat dva modely?

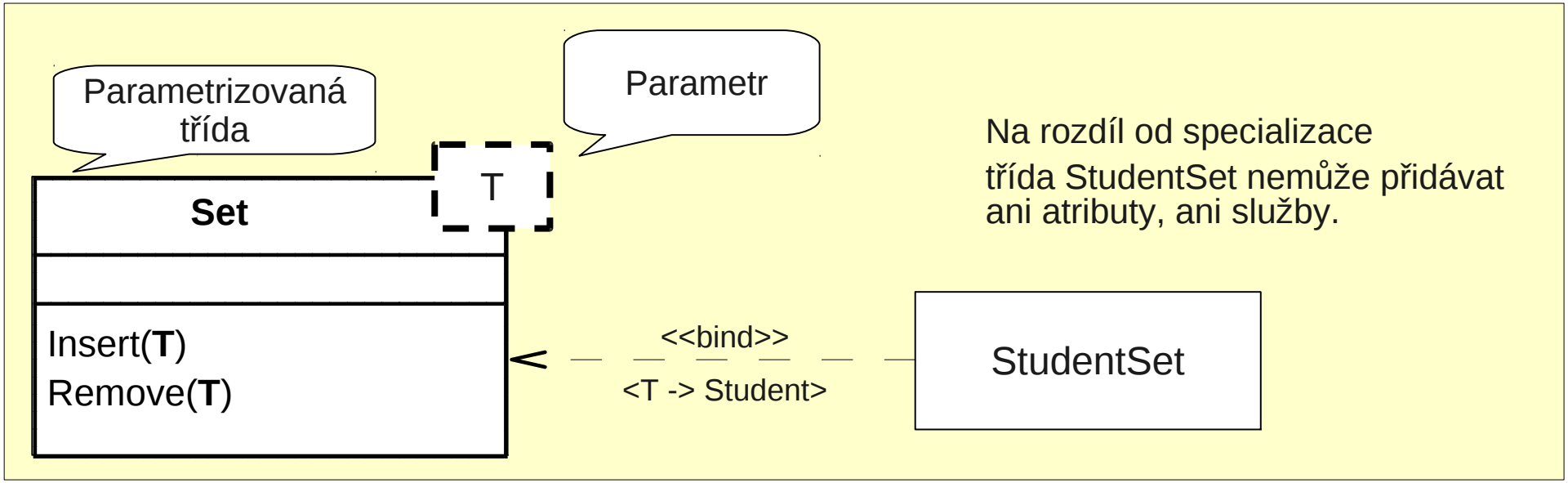
- Strategie, jak přejít od analýzy k návrhu:
  - Postupně přepracujte analytický model na návrhový
    - Pracujete pouze s jediným modelem, ztrácíte analytický pohled
  - Analytický model přepracujte na návrhový a pak použijte CASE nástroje pro vytvoření “analytického pohledu”
    - Pracujete pouze s jediným modelem, ale analytický model vytvořený CASE systémem nemusí být uspokojivý
  - Zamkněte analytický model ve vhodném bodě vývoje a přepracujte *kopii* analytického modelu na návrhový model
    - Máte k dipozici oba modely, mohou se ale rozcházet
  - Spravujte dva oddělené modely, analytický a návrhový
    - Máte k dipozici oba modely které se navíc schodují, správa je ale komplikovaná

Parametrizované třídy, šablony

Parametrizovaná třída definuje *rodinu* tříd.

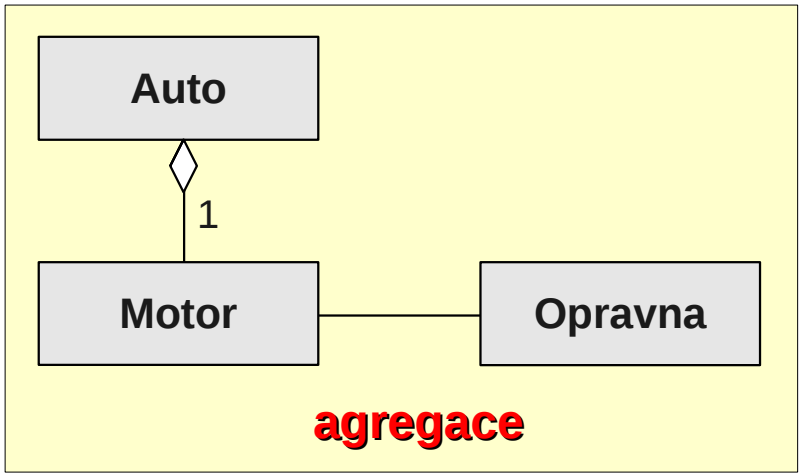
Nelze ji přímo použít, je nutné ji nejprve instanciovat přiřazením potřebných typů. Instanciace teprve vytvoří použitelnou třídu.

Příklad: třída **Set** – reprezentuje kolekci objektů nějaké *třídy.* Tato třída se stává parametrem parametrizované třídy *např. množina **mincí**, množina **studentů**, množina <T>*



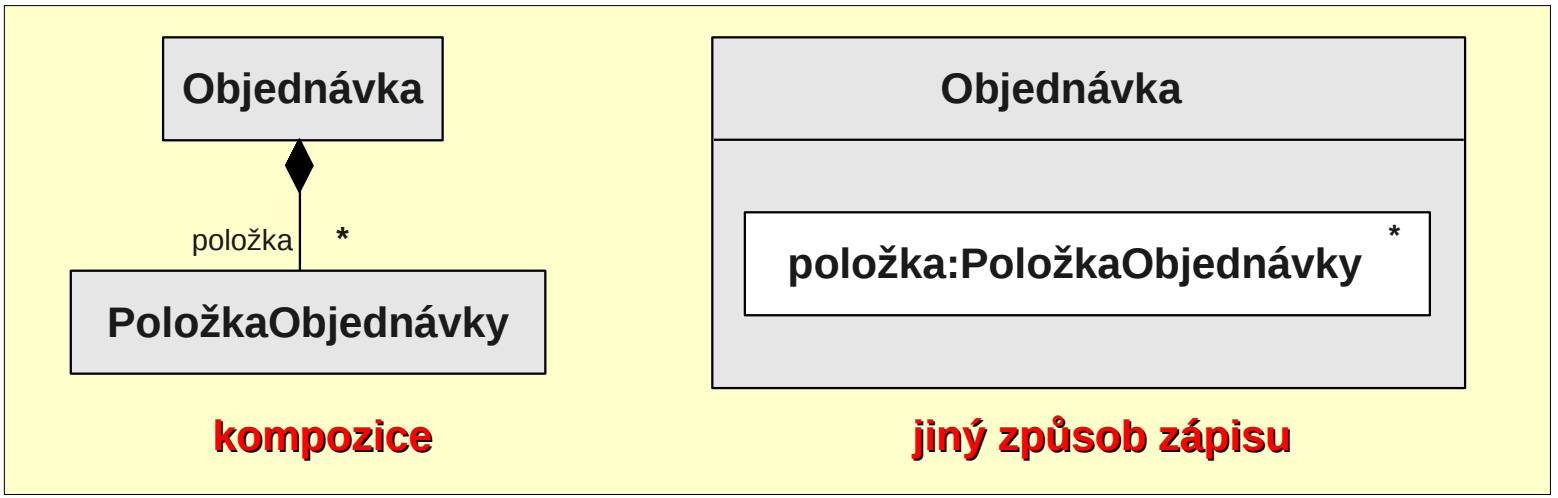
Agregace

- Angl. *Aggregation*
- Speciální případ asociace pro **vztah celek-část**
- Symbol prázdného diamantu
- Tranzitivní
  - pokud je píst součást motoru a motor součástí auta, pak je i píst součástí auta
- Asymetrická
  - pokud je motor součástí auta, pak auto není součástí motoru
  - diagram objektů musí být acyklický
- Sémantika:
  - totéž co asociace, pouze zdůrazňujeme vztah celek-část
  - součásti mohou existovat nezávisle na celku
  - součást může být sdílena více celky



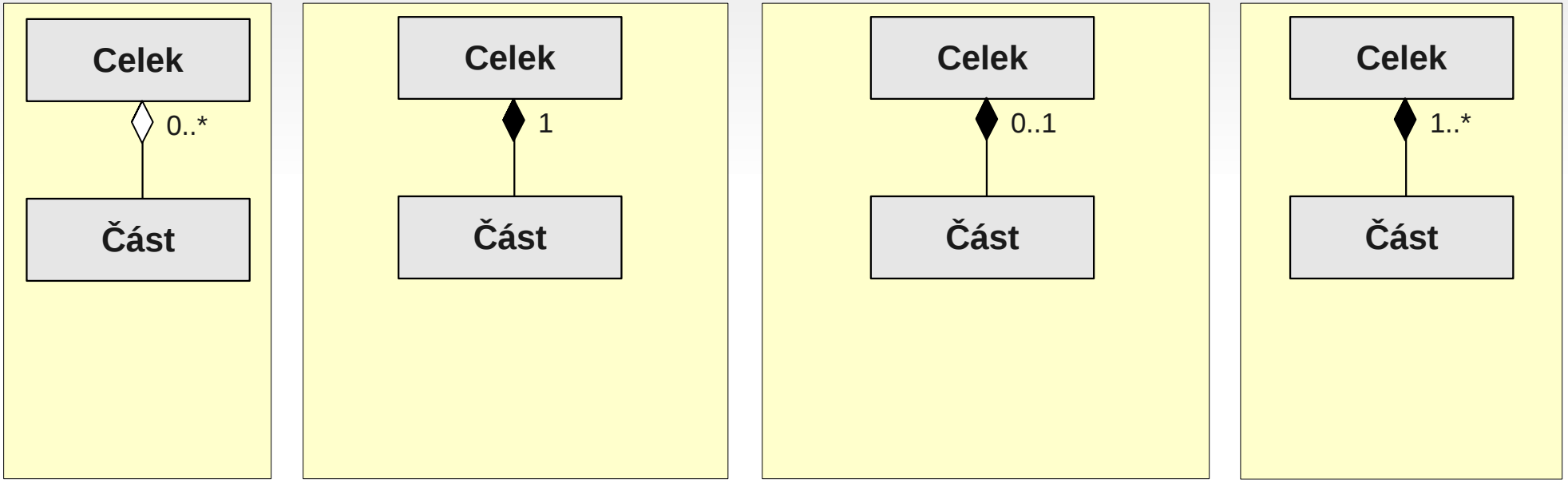
Kompozice

- Angl. *Composition*
- Silnější forma agregace
- Symbol plného diamantu
- Tranzitivní a asymetrická
- Sémantika:
  - na úrovní instancí (objektů) části neexistují vně celku
  - každá část patří pouze do jediného celku (části nelze sdílet)
  - je-li celek zničen, musí zničit i svoje součásti, nebo převést zodpovědnost za ně na jiný objekt



Vlastnosti agregace a kompozice (I)

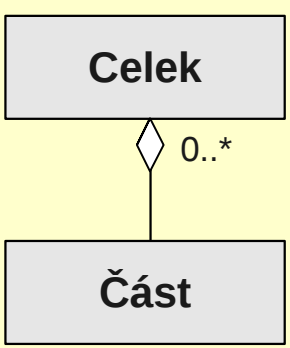
- Které modely jsou správně a které naopak špatně z důvodu vlastností agregace a kompozice?



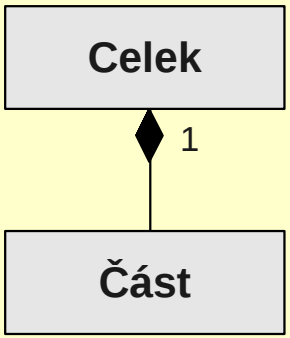
... odpověď na další obrazovce >>

Vlastnosti agregace a kompozice (II)

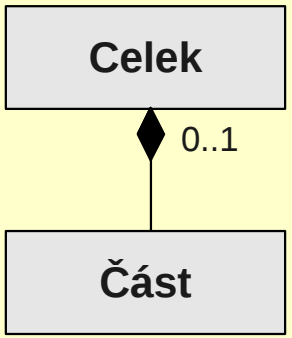
... odpovědi z předchozí stránky:



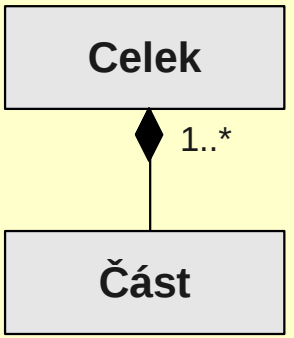
správně



správně
totéž co bez kardinality



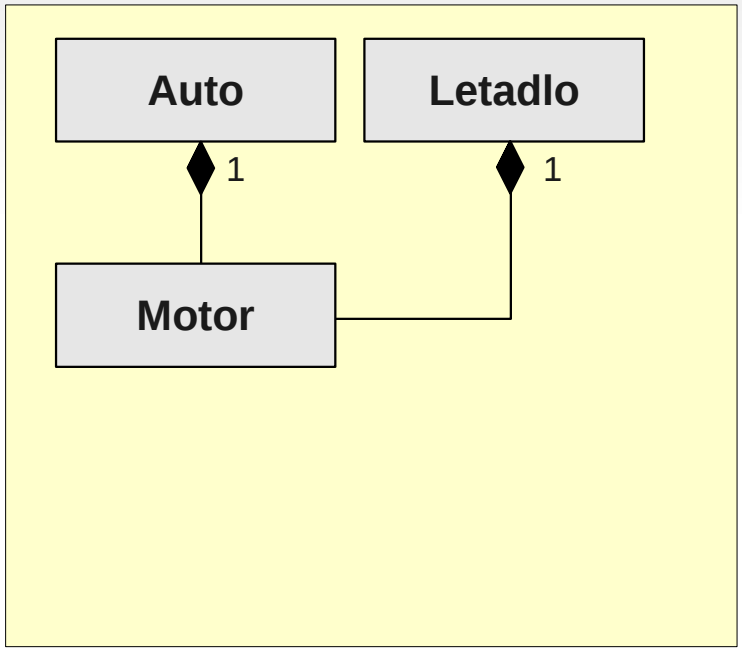
špatně
část neexistuje bez celku

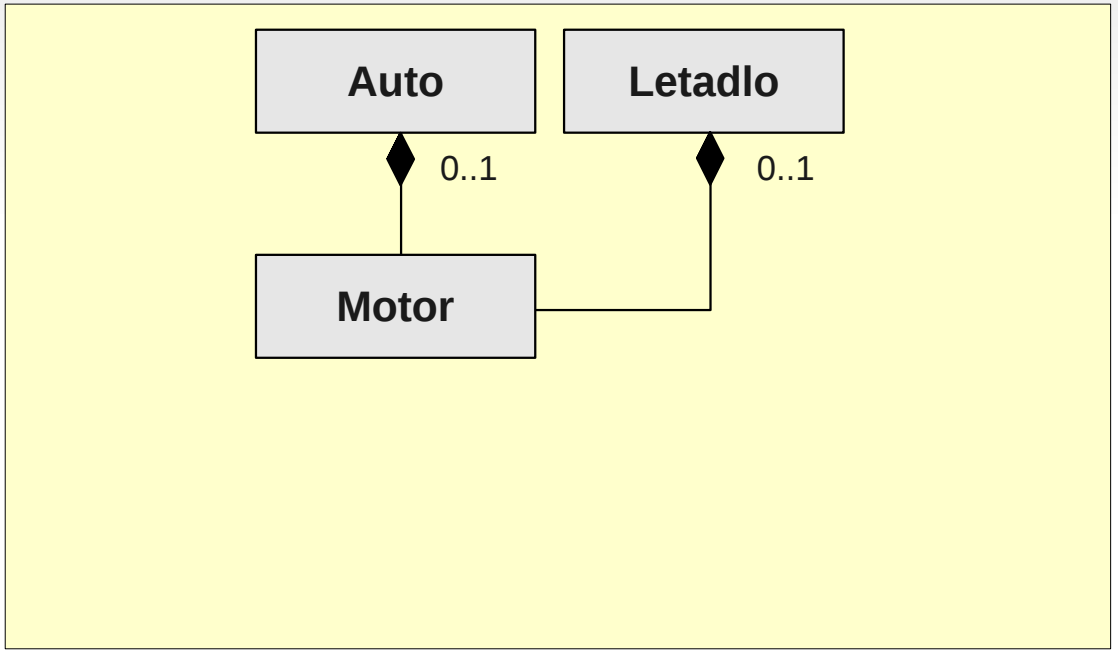


špatně
část je součástí jediného celku

Vlastnosti agregace a kompozice (IV)

- Které modely jsou správně a které naopak špatně z důvodu vlastností agregace a kompozice?

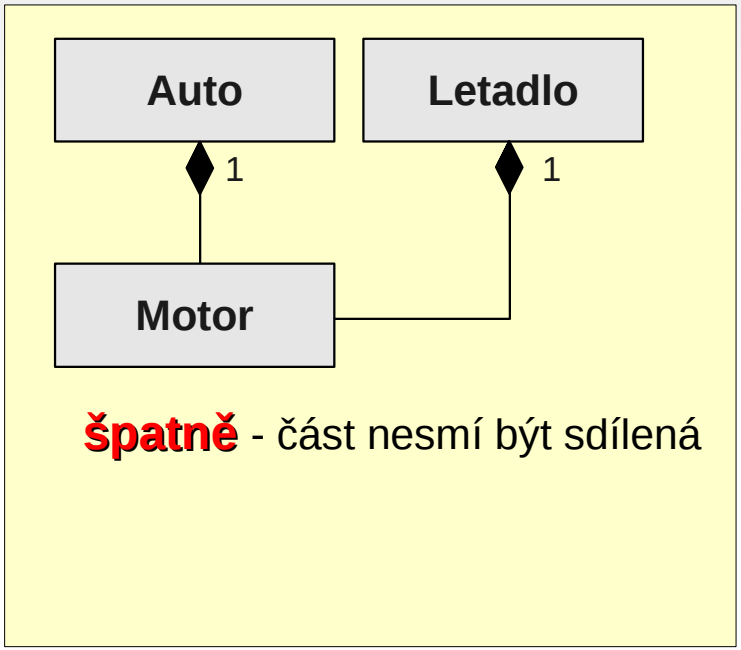




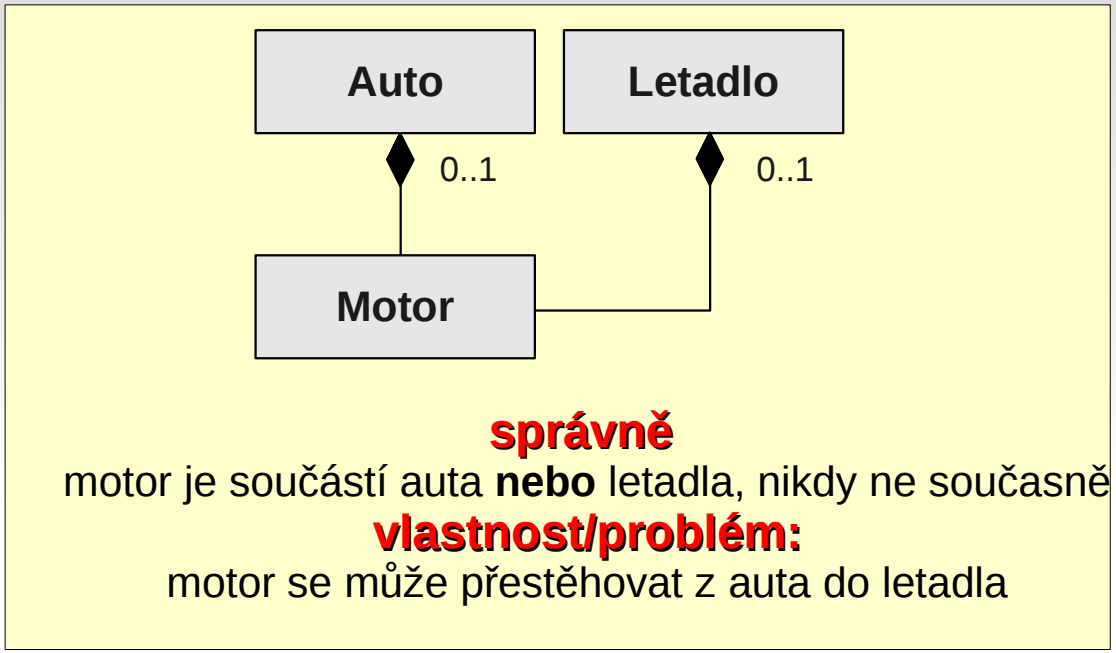
... odpověď na další obrazovce >>

Vlastnosti agregace a kompozice (V)

... odpovědi z předchozí stránky:



špatně - část nesmí být sdílená

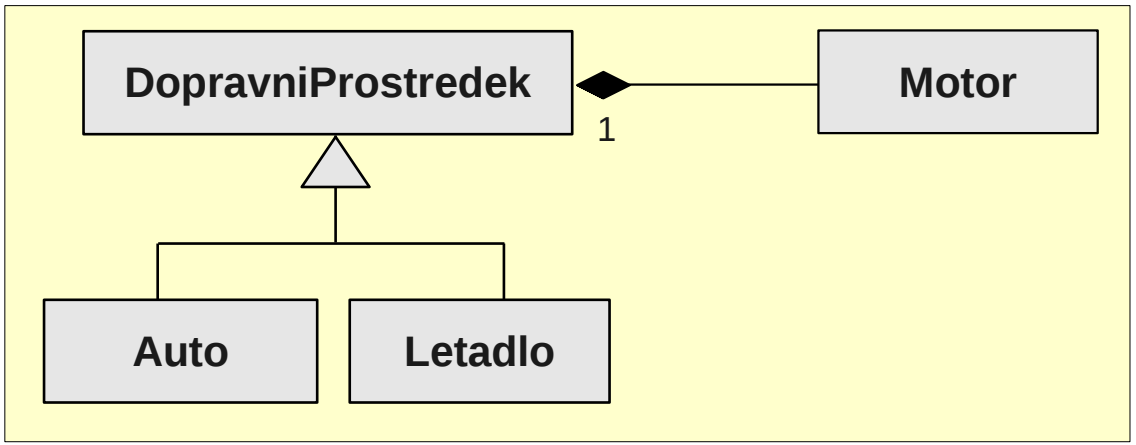


správně

motor je součástí auta **nebo** letadla, nikdy ne současně

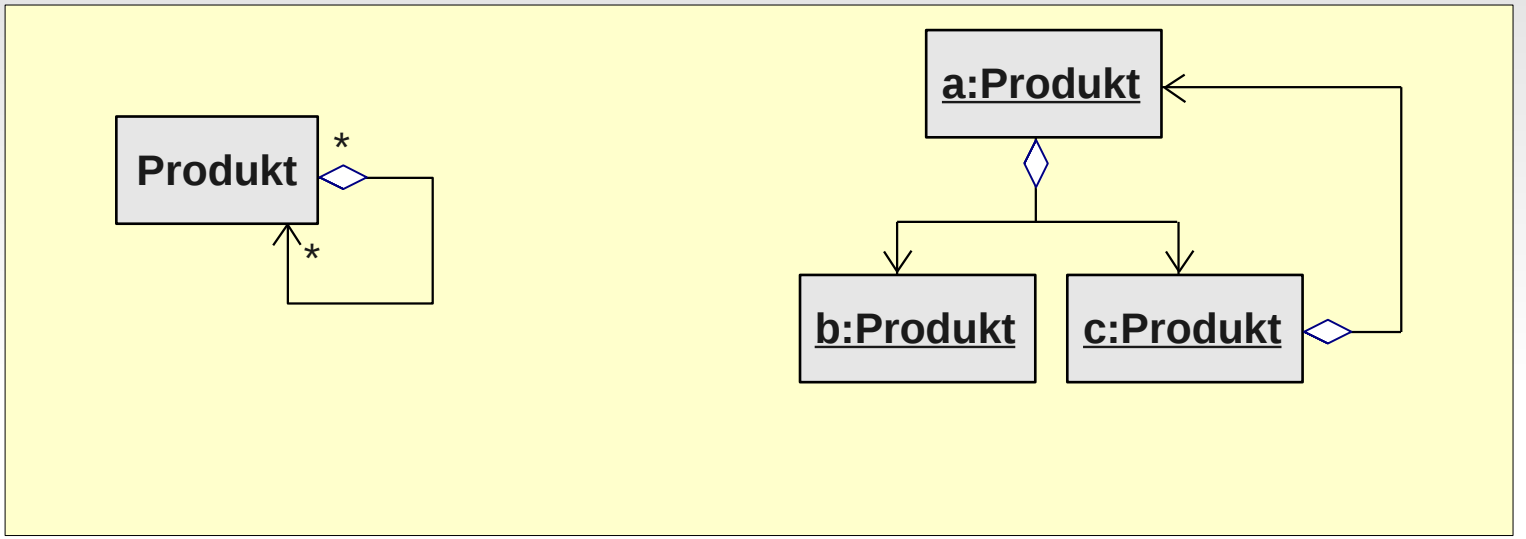
vlastnost/problém:

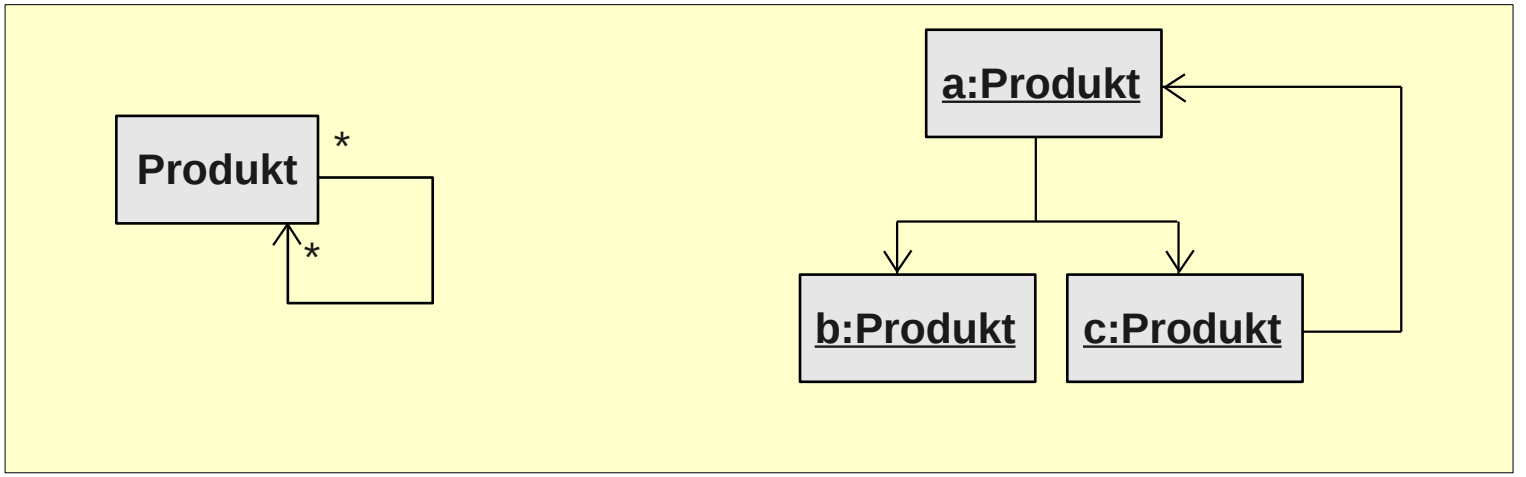
motor se může přestěhovat z auta do letadla



Vlastnosti agregace a kompozice (VI)

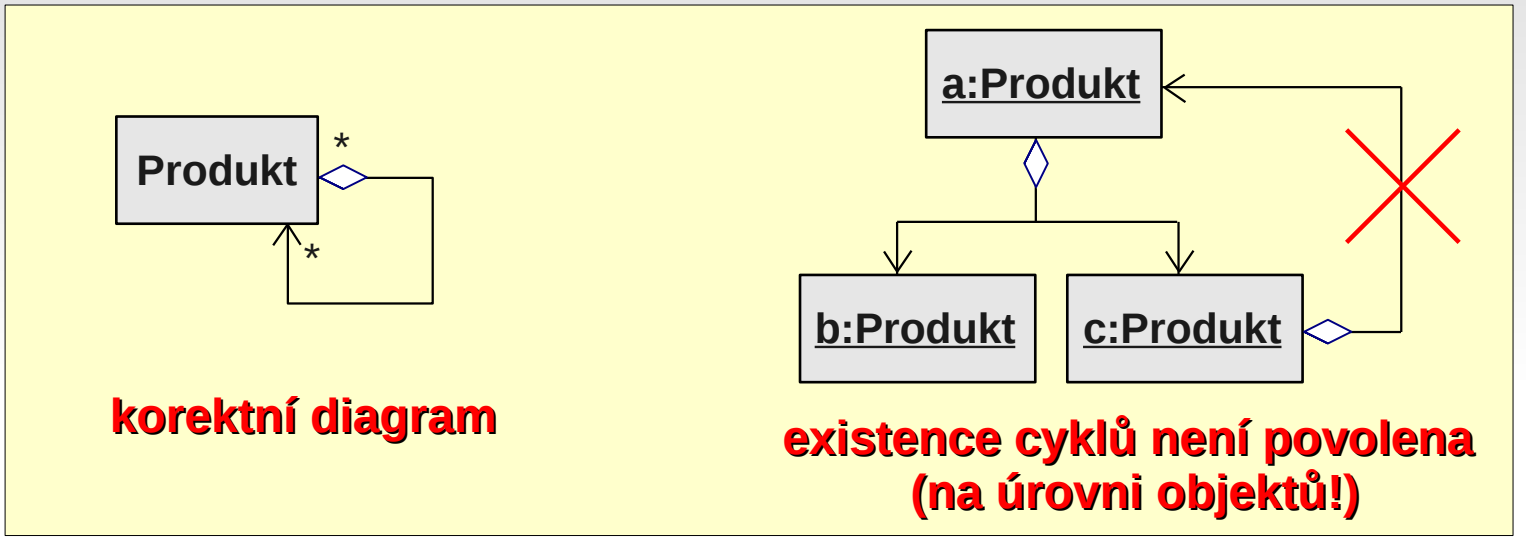
- Které modely jsou správně a které naopak špatně z důvodu asymetrie agregace?

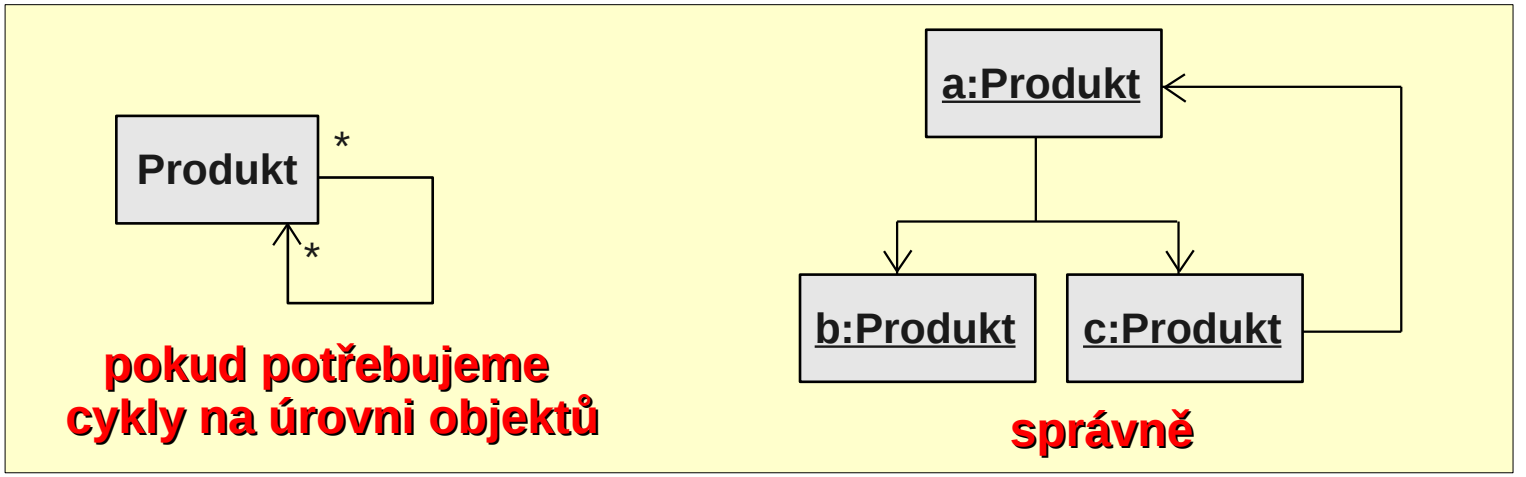




Vlastnosti agregace a kompozice (VII)

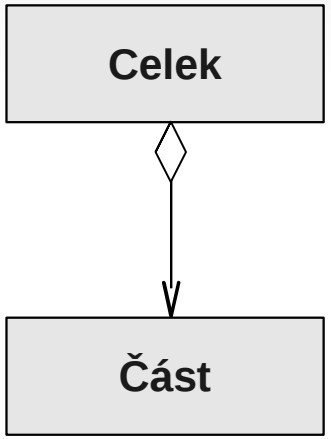
... odpovědi z předchozí stránky:

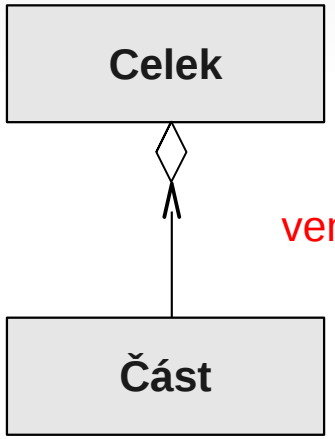




Agregace a směr řízení

- V agregace/kompozici jsou většinou části řízeny svými celky
- Většina asociací z analytického modelu přechází na agregace a kompozice v návrhu

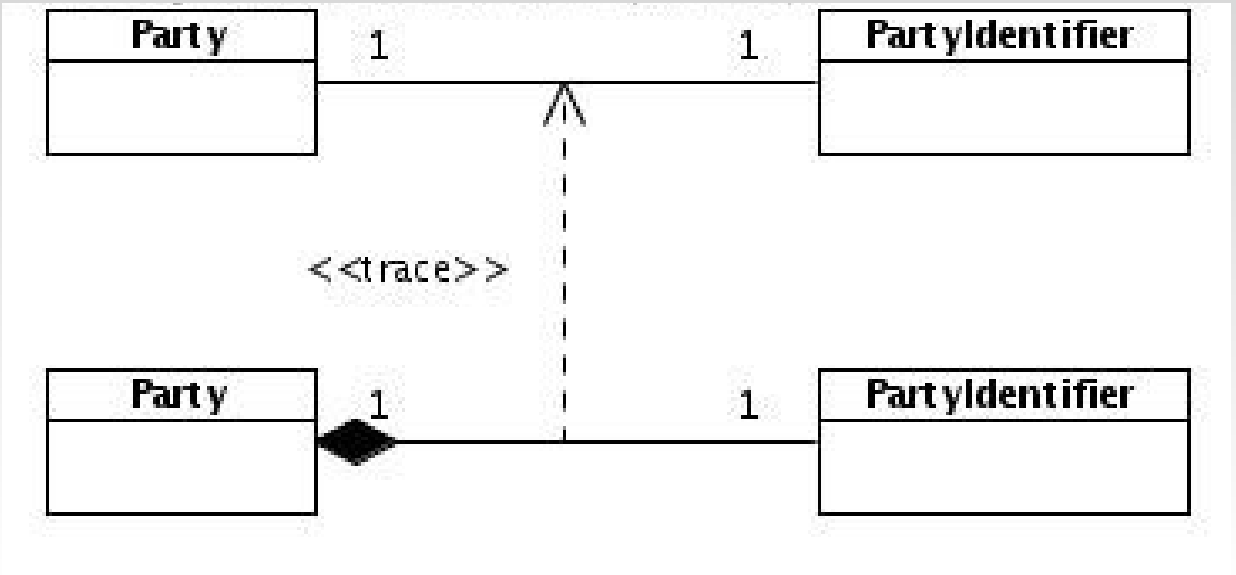




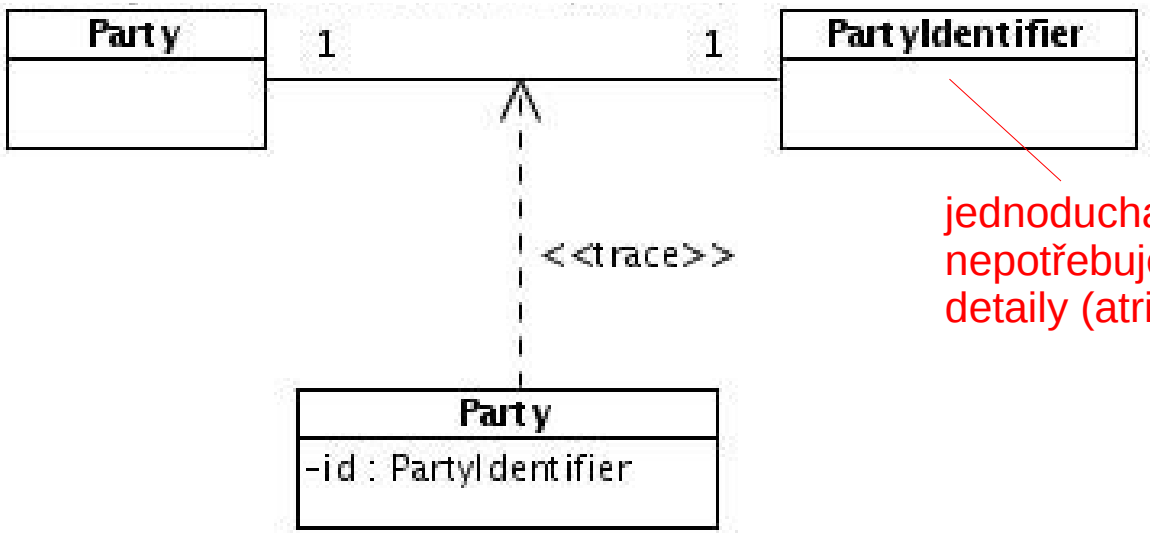
vemi neobvyklé

Návrhové třídy: Asociace 1:1

analýza



analýza



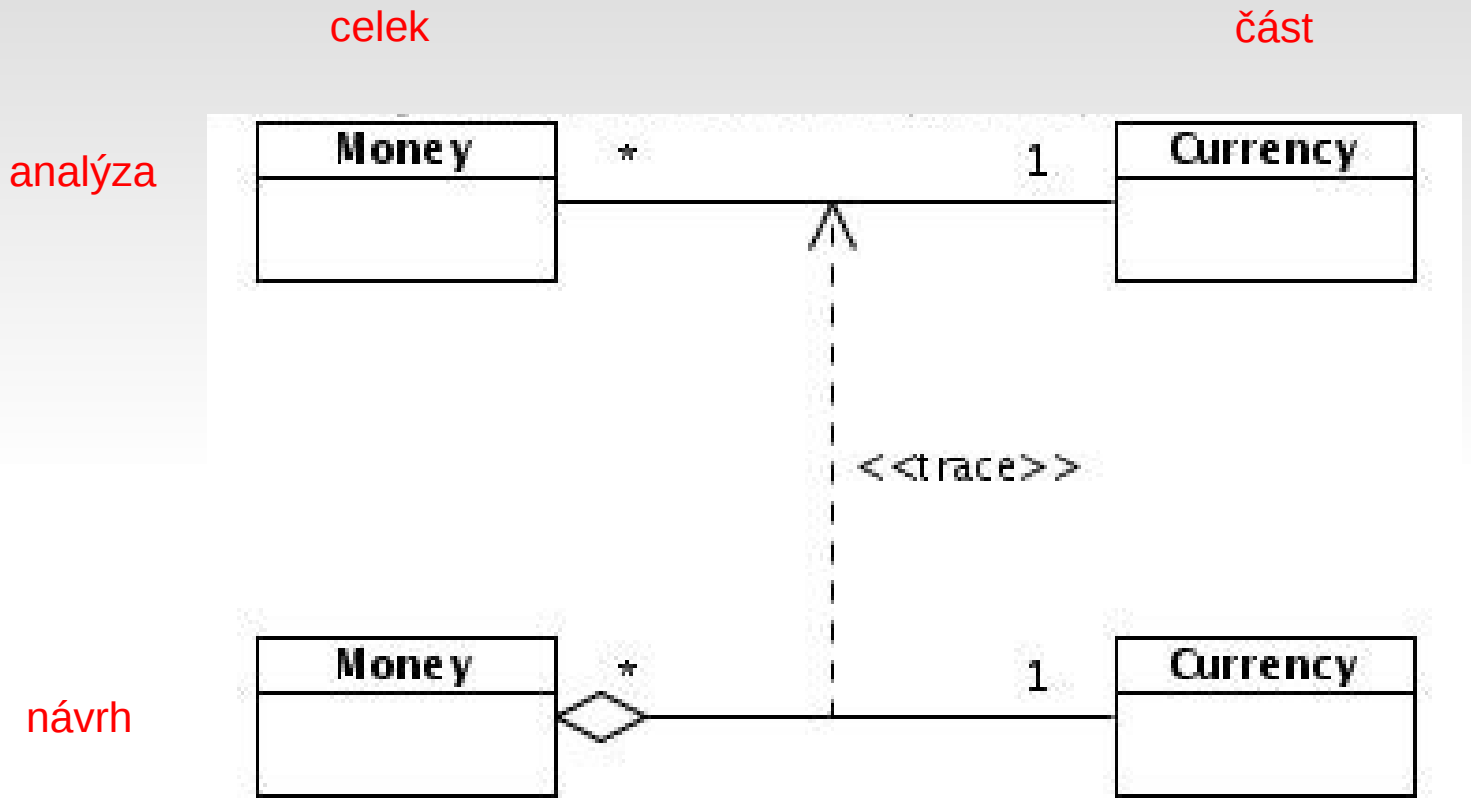
jednoduchá třída,
nepotřebujeme zobrazovat
detaily (atributy, ...)

návrh

návrh

Asociace 1:1 často vedou na kompozici

Návrhové třídy: Asociace M:1

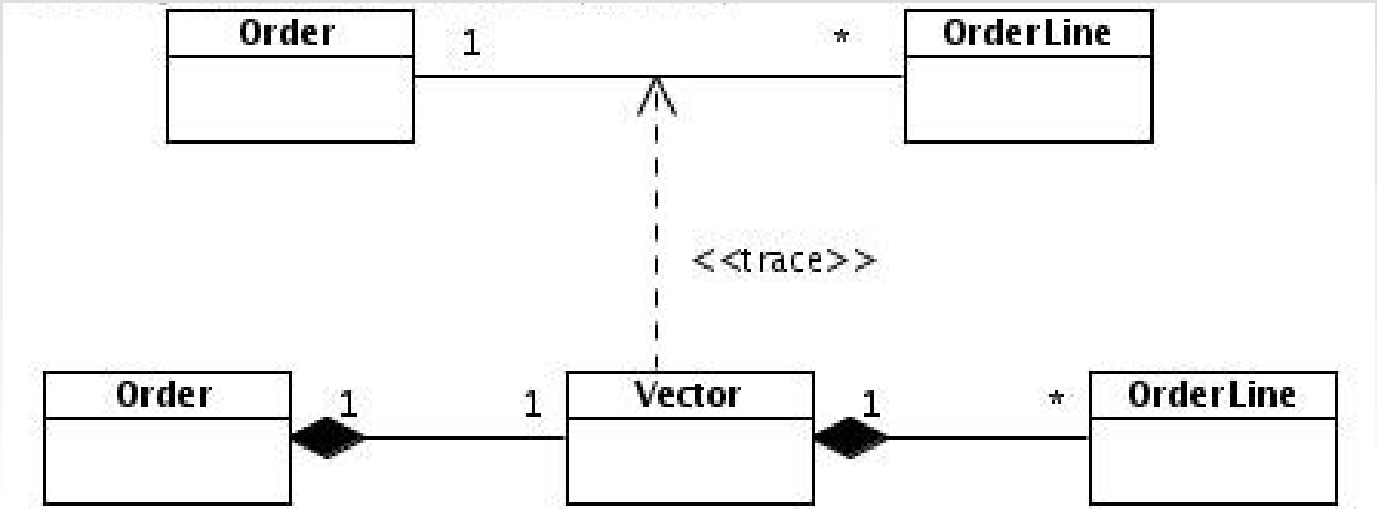


Asociace M:1 často vedou na agregaci, pokud není potřeba dělat cykly na úrovni objektů

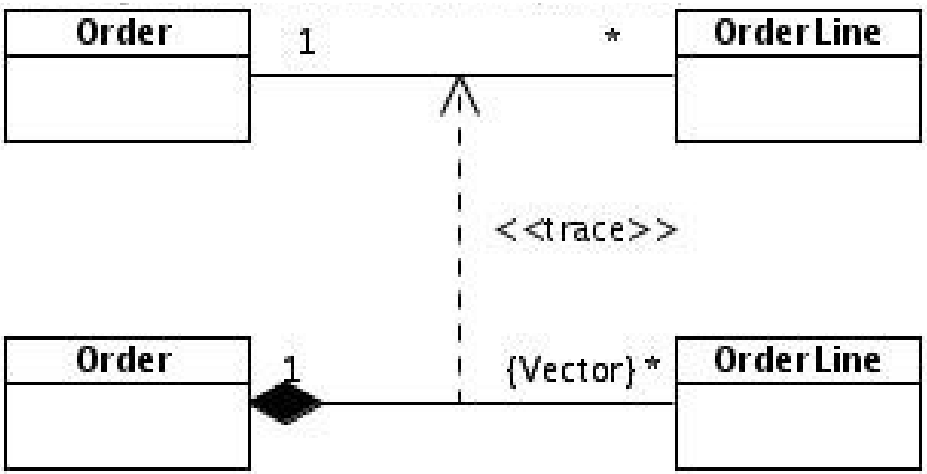
Návrhové třídy: Asociace 1:M (= kolekce)

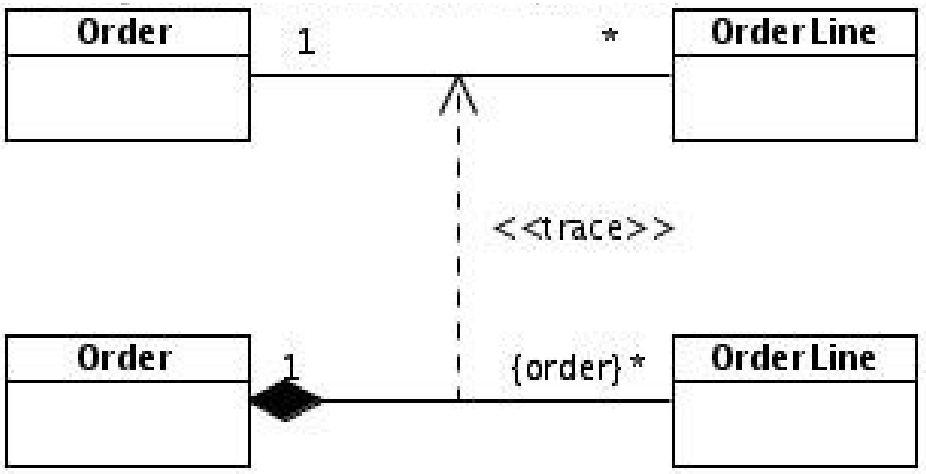
celek část

analýza



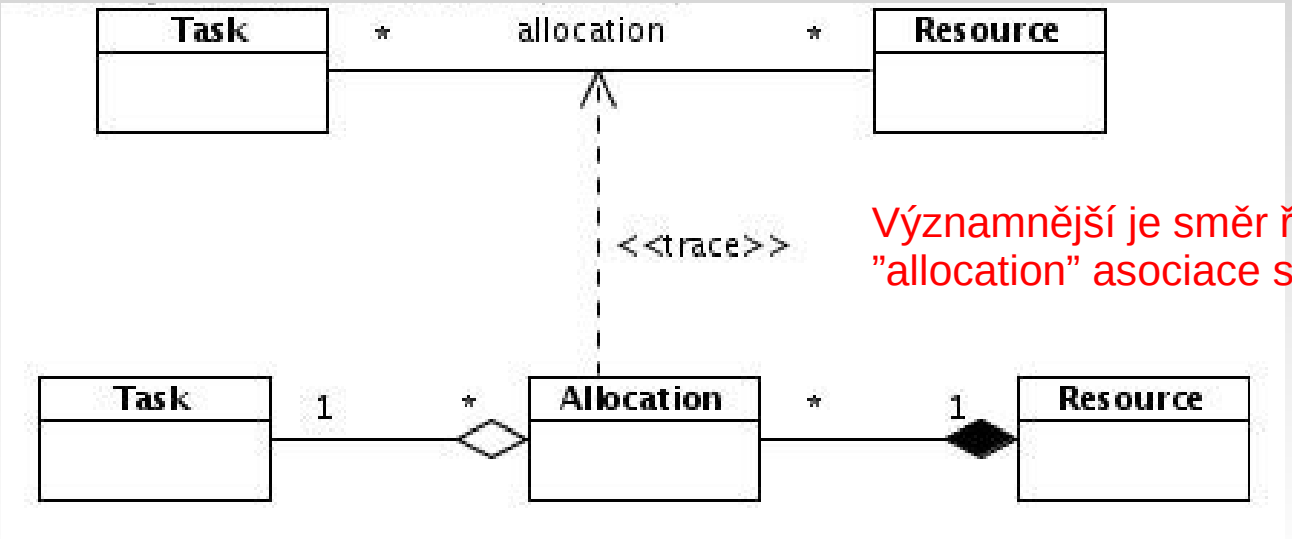
návrh





Návrhové třídy: Asociace M:N

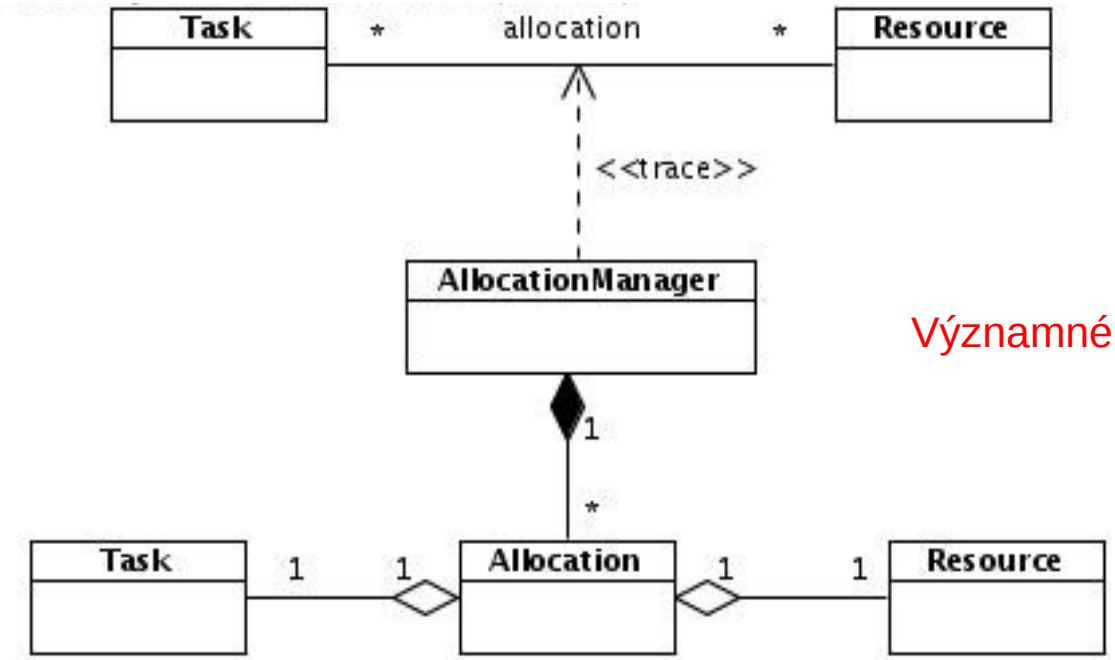
analýza



Významnější je směr řízení je Resource -> Task, ”allocation” asociace se změní ve třídu „Allocation”

návrh

analýza

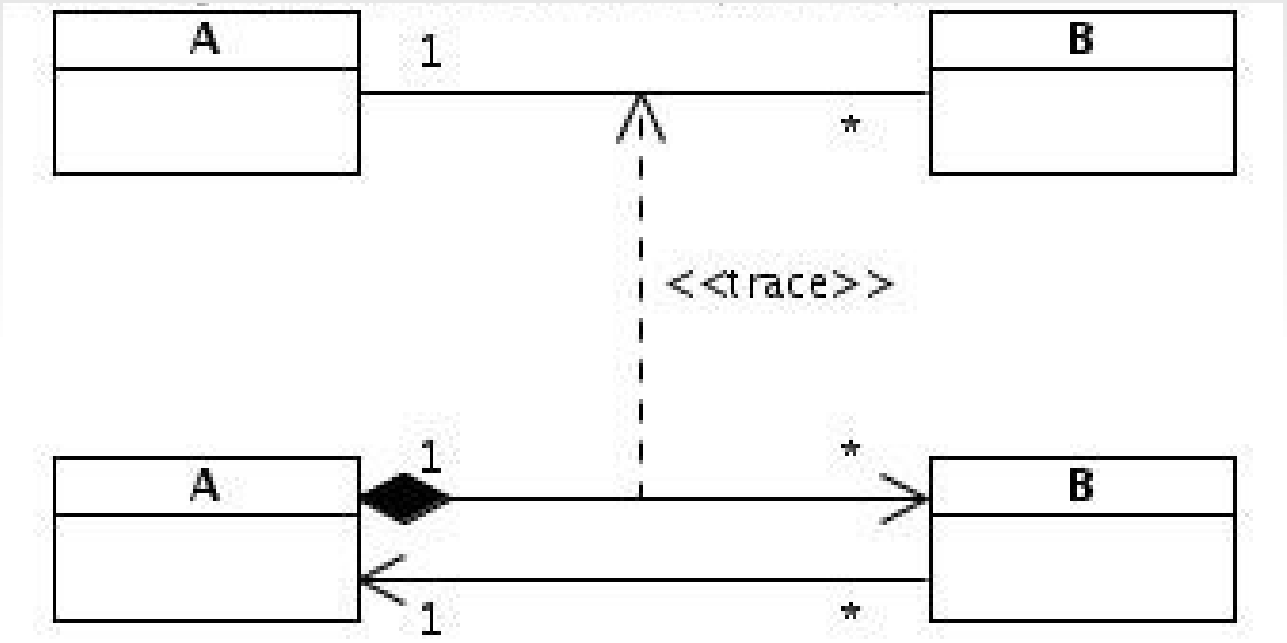


návrh

Významné jsou oba směry řízení (přístupu)

Návrhové třídy: Obousměrné asociace

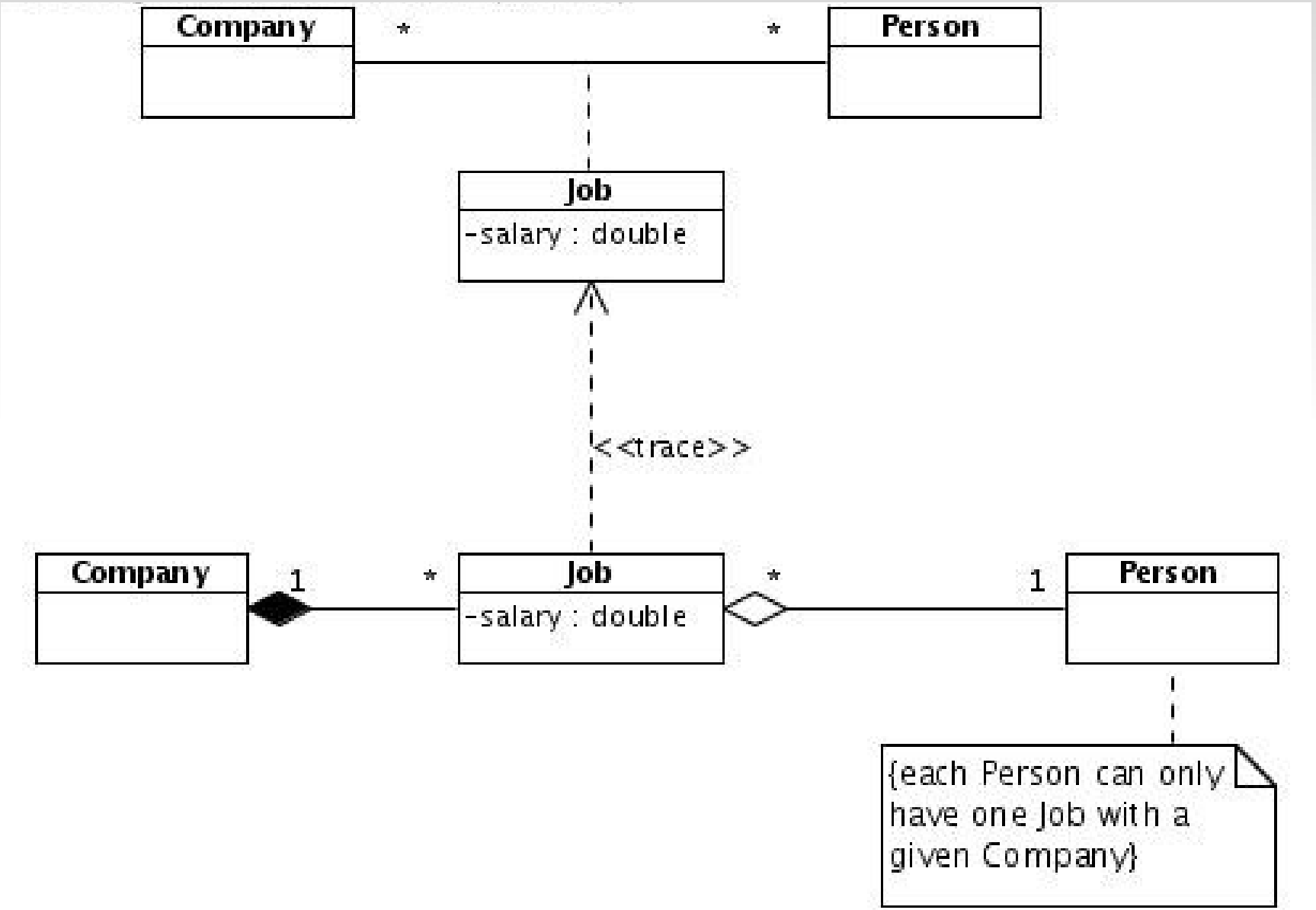
analýza



návrh

Návrhové třídy: Asociační třídy

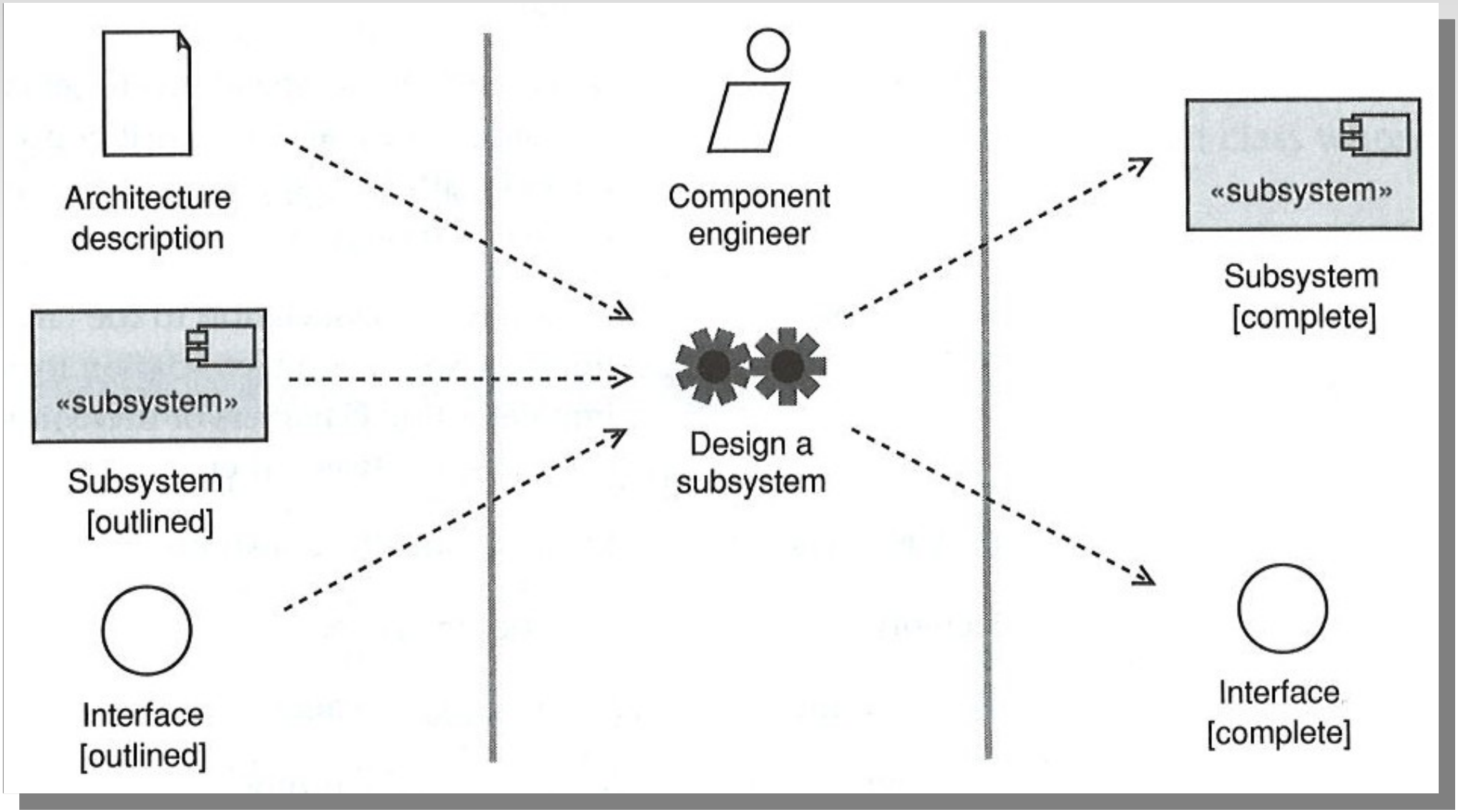
analýza



návrh

Rozhraní a komponenty
(Interfaces and components)

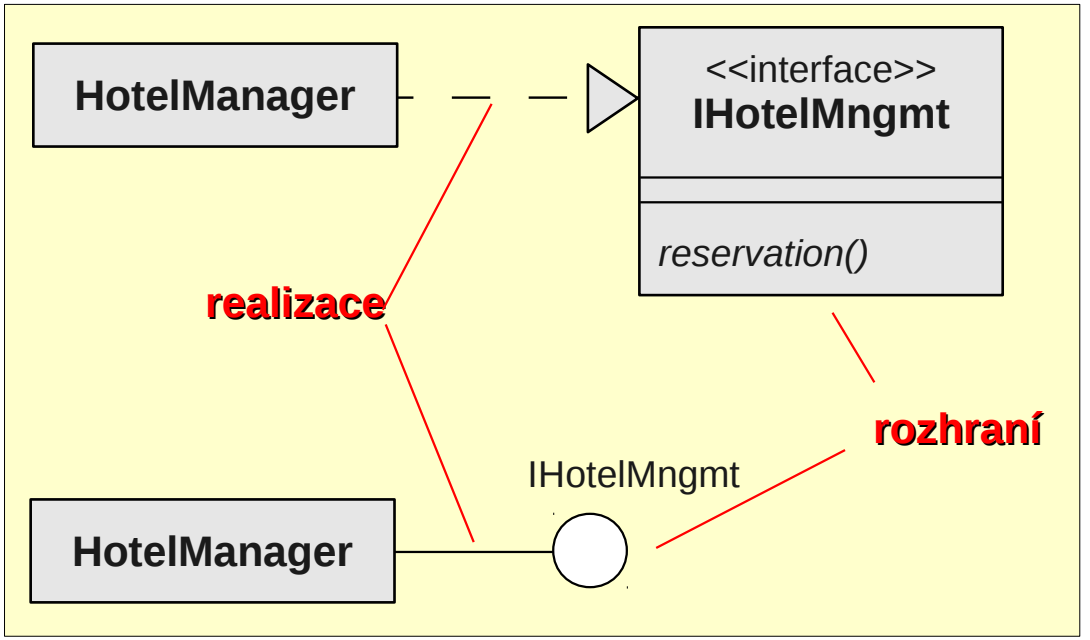
UP aktivita: Design a subsystem



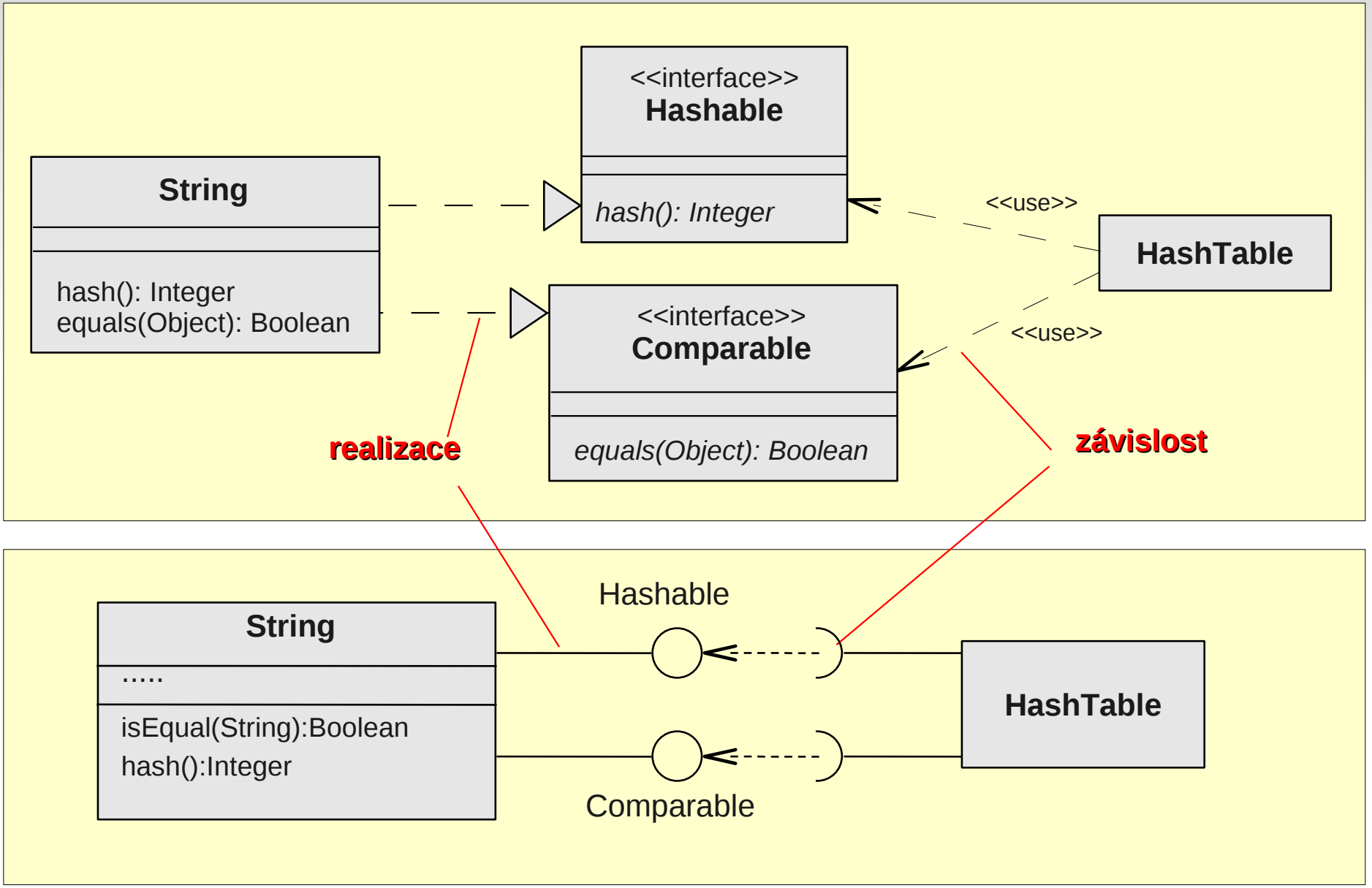
Rozhraní

- Angl. *Interface*
- Speciální třídy, které definují *externě viditelné služby* (tzv. kontrakt) nějaké třídy nebo komponenty, bez *specifikace interní struktury* (atributy, stavy, implementace metod).
- Často popisují pouze část, nikoliv celé chování příslušné třídy.
- Používají stejné vztahy jako třídy, navíc ještě vztah *realizace* (angl. *realization*).
- Dvě notace.

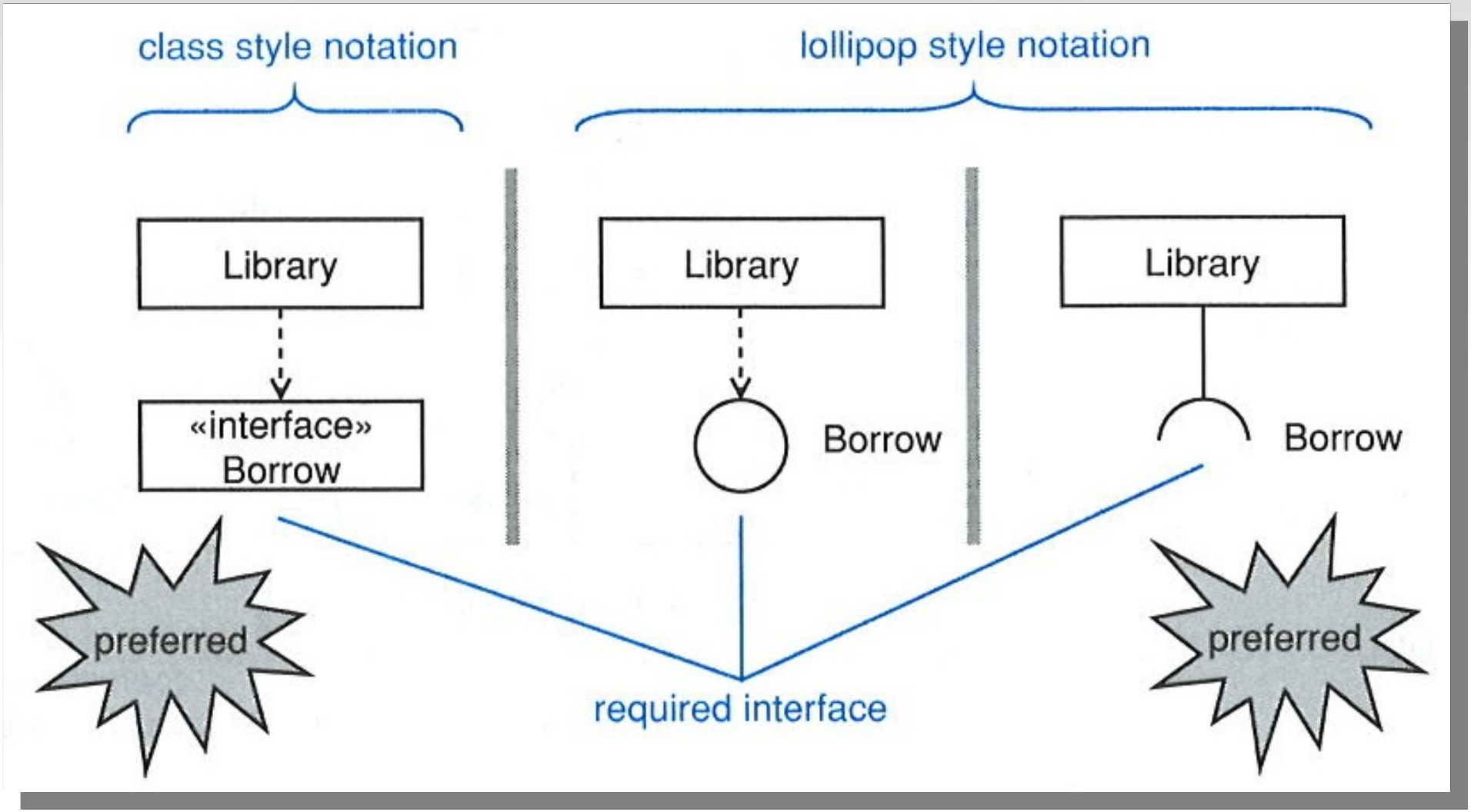
Rozhraní = nabídka služeb realizovaných třídami



Rozhraní: poskytované vs. požadované



Rozhraní: notace



Modelování komponent

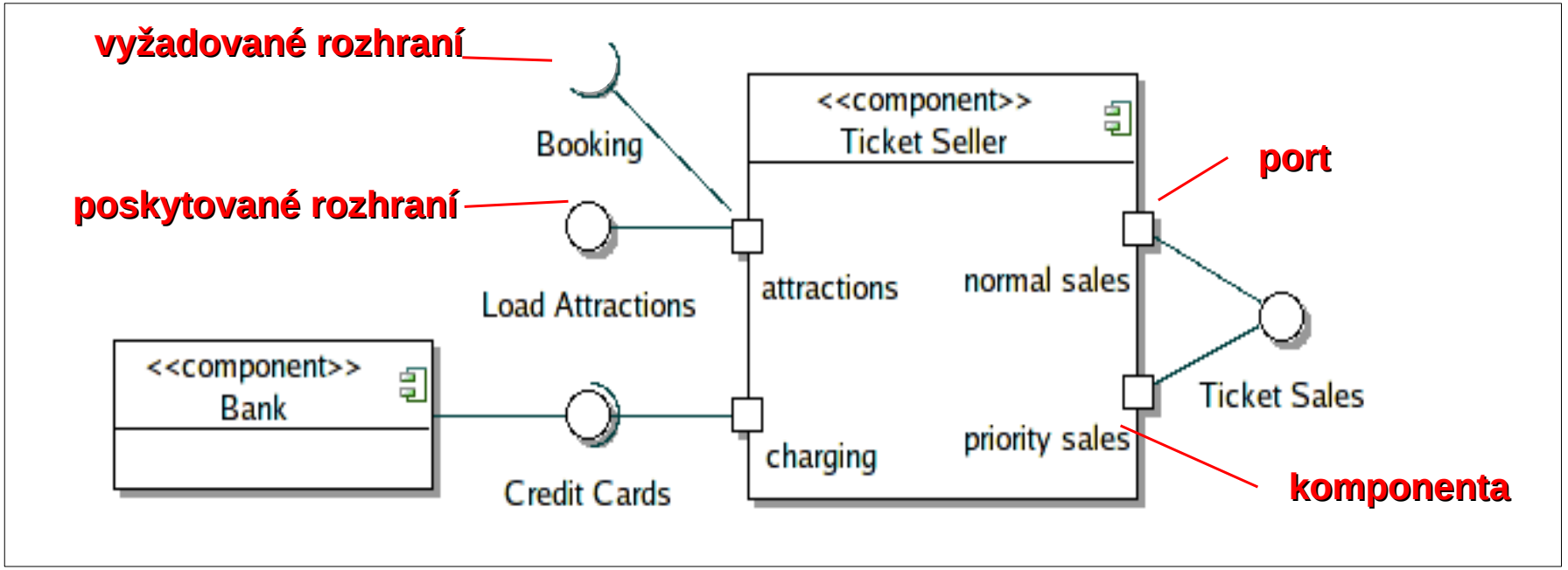
- Model v etapě návrhu
- Spojení konceptu balíků a rozhraní
- Co je to komponenta?
  - Komponenta je jeden nebo více objektů sdružených do definovaného programového rozhraní
  - Komponenta poskytuje ucelený soubor služeb a je navržena pro vícenásobné použití
  - Komponenta je samostatně spustitelná a připojitelná za běhu

Komponenty
větší celky
více rozhraní
poskytují služby
plně zapouzdřené
obecně pochopitelné

Objekty/třídy
jemné subjekty
jedno rozhraní
poskytují operace
využívají dědičnost, asociace
pochopitelné pro vývojáře

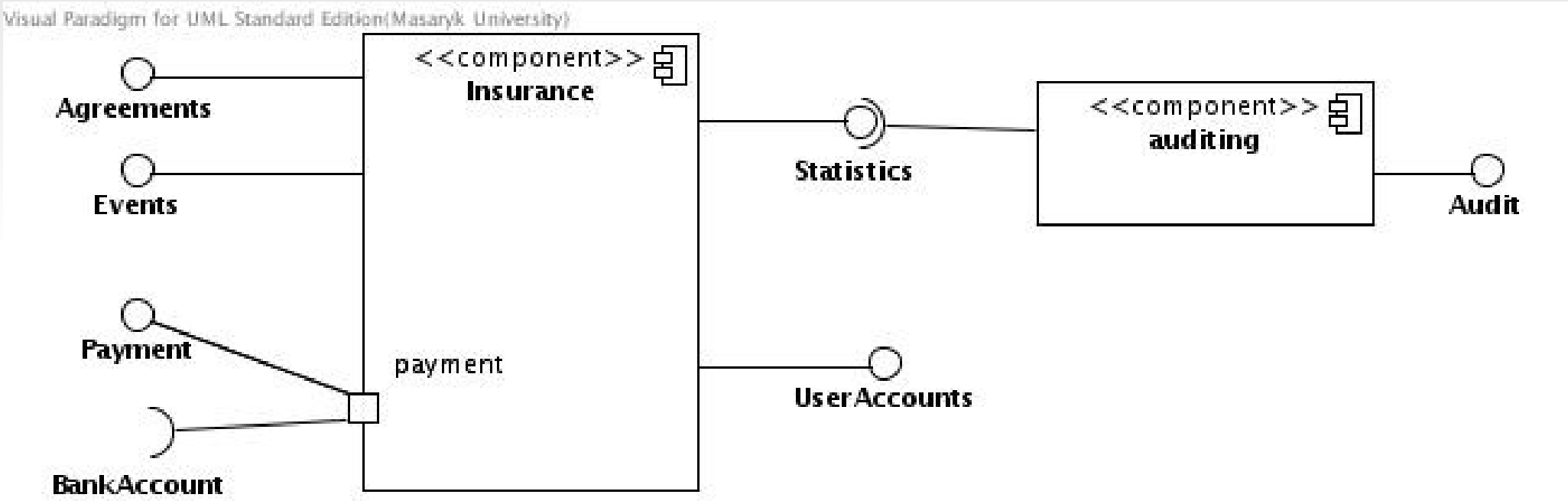
Diagram komponent

- Angl: *Component Diagram*
- Ukazuje strukturu kódu
  - *rozhraní* – kolekce operací které jsou poskytovány nebo vyžadovány komponentou, viz. třídy
  - *komponenta* – vyměnitelná část systému; může obsahovat vnitřní strukturu (propojené komponenty nižší úrovně)
  - *port* – "okno" do uzavřené komponenty akceptující zprávy z/do komponenty odpovídající specifikovanému rozhraní



Pojišťovna: Diagram komponent

Demo



Realizace případů užití – návrh
(Use case realization – design)

UP aktivita: Design a use case

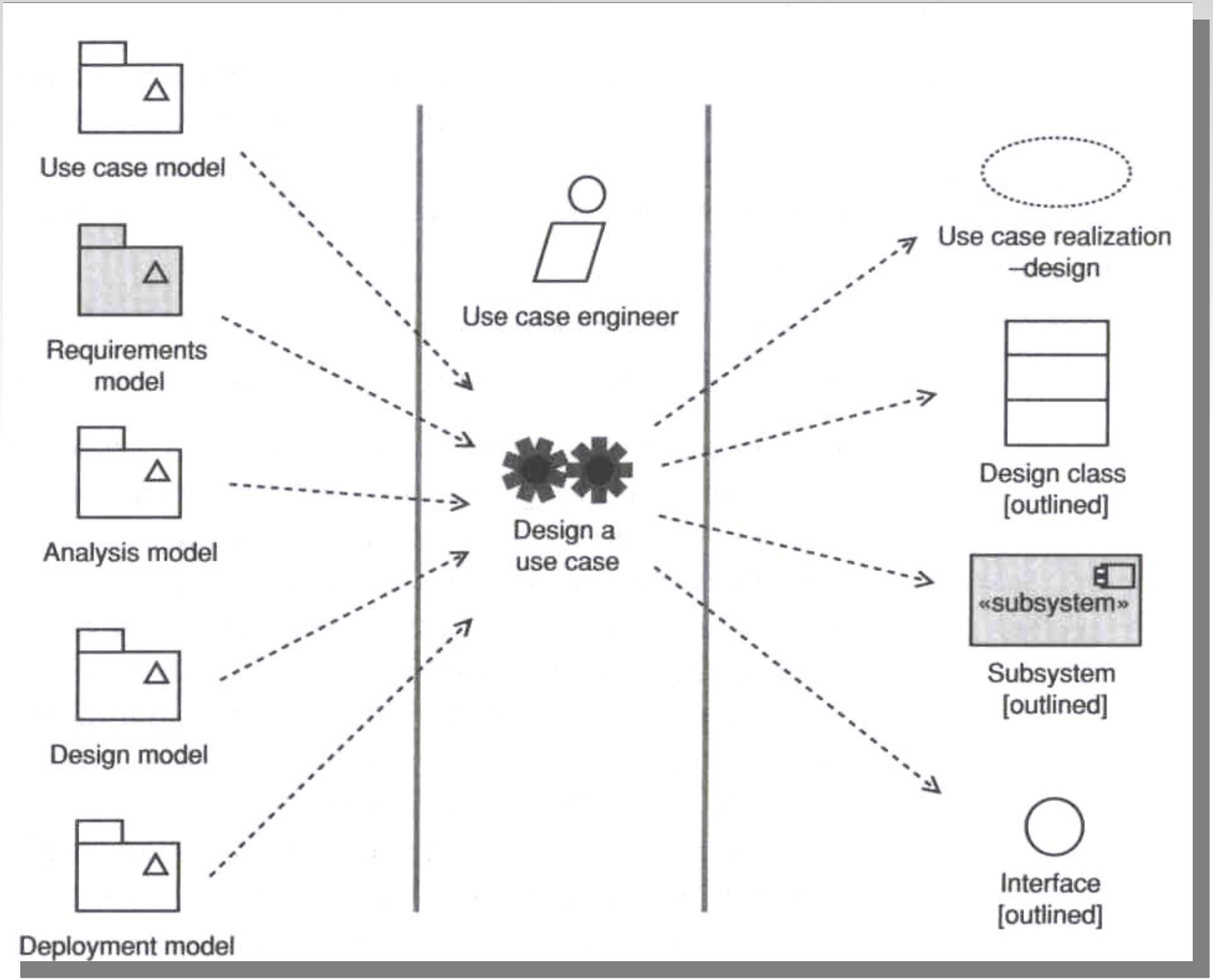


Diagram časování

- Angl.: *Timing Diagram*
- Modelují časová omezení mezi změnami stavů různých objektů

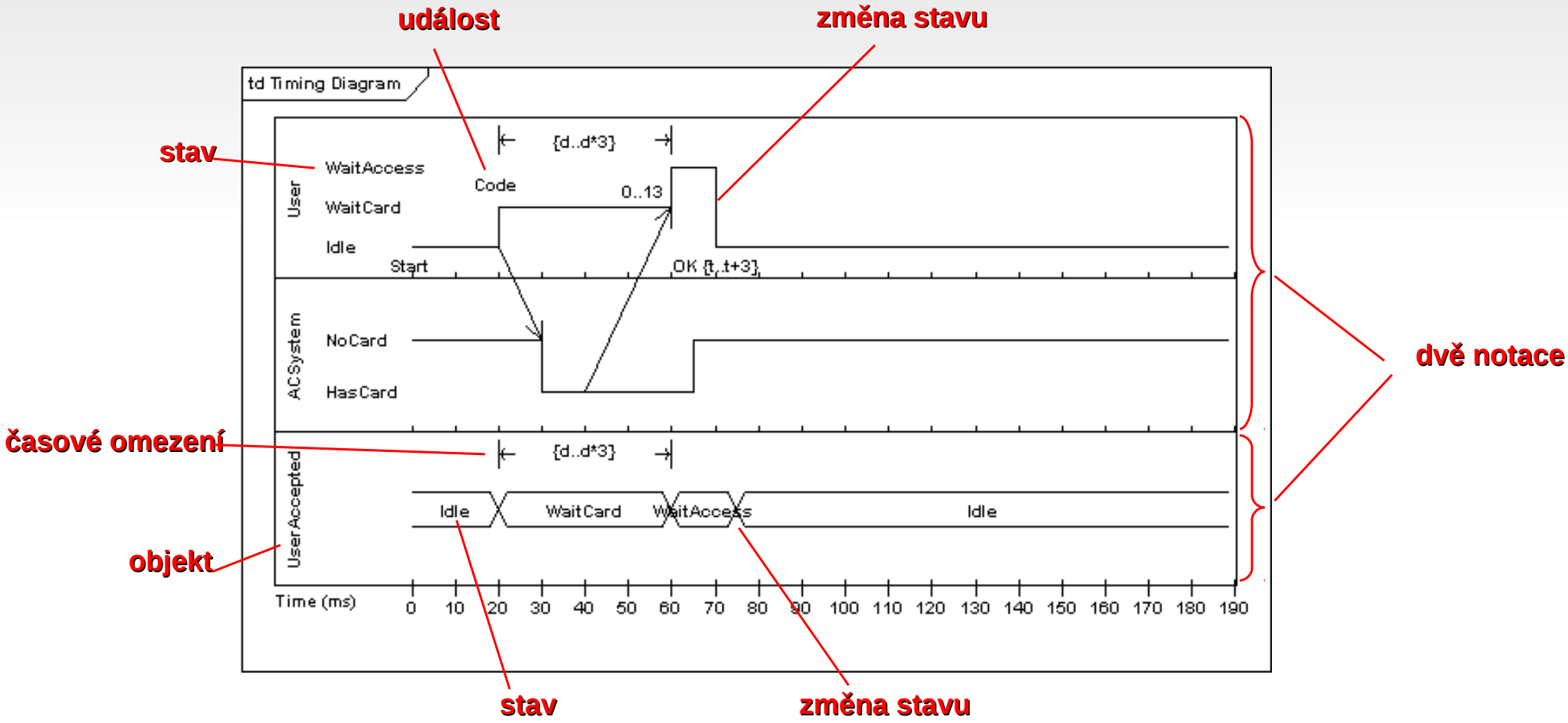
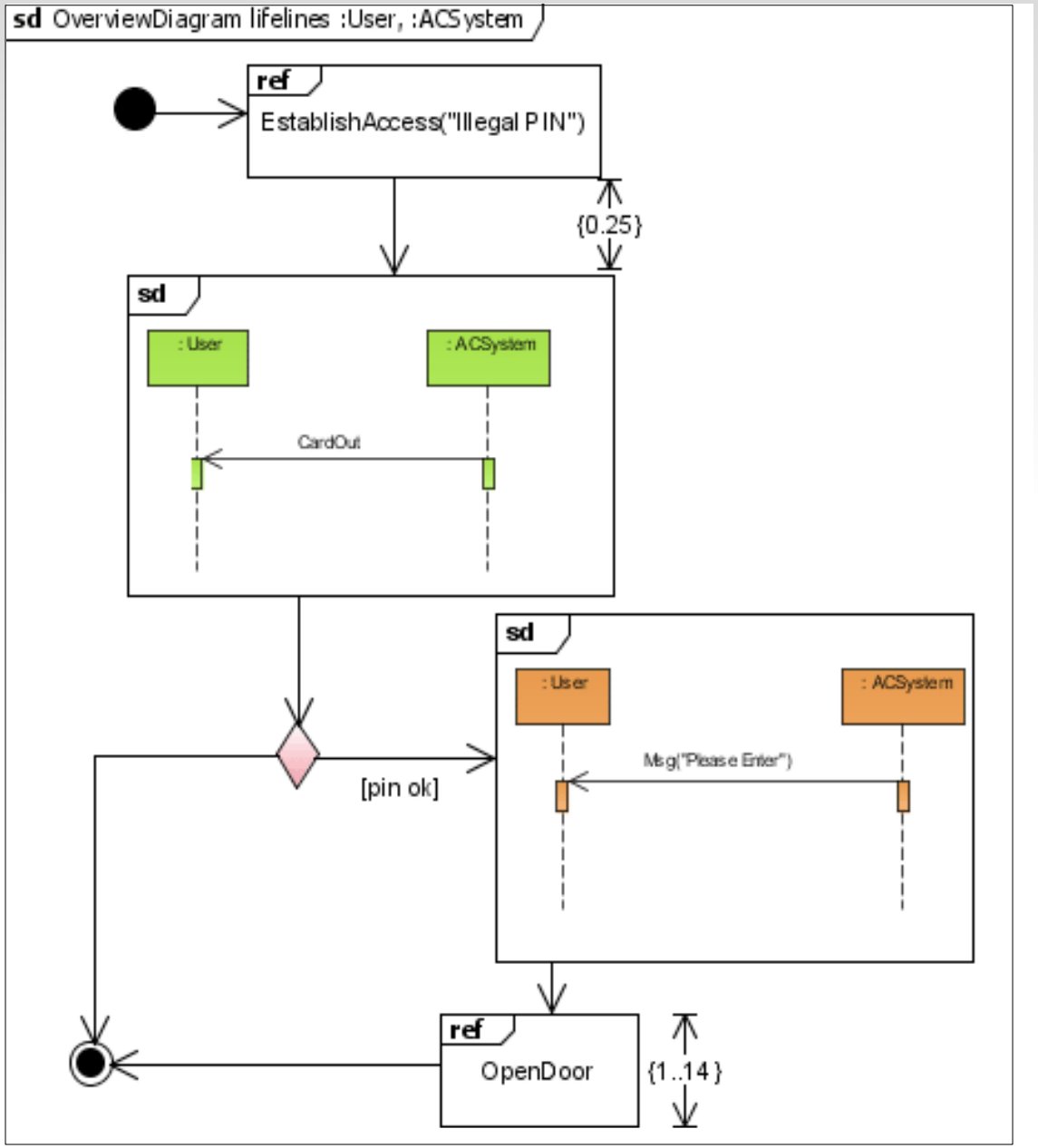


Diagram přehledu interakcí

- Anlg.: *Interaction Overview Diagram*
- Kombinuje sekvenční diagramy a digramy aktivit

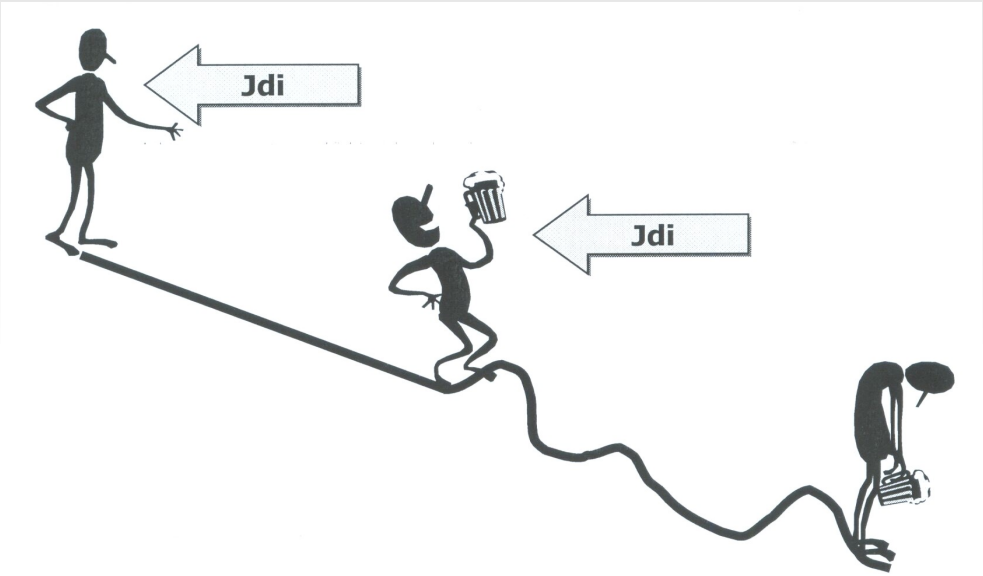


Stavové diagramy
(State machines)

Stavový diagram

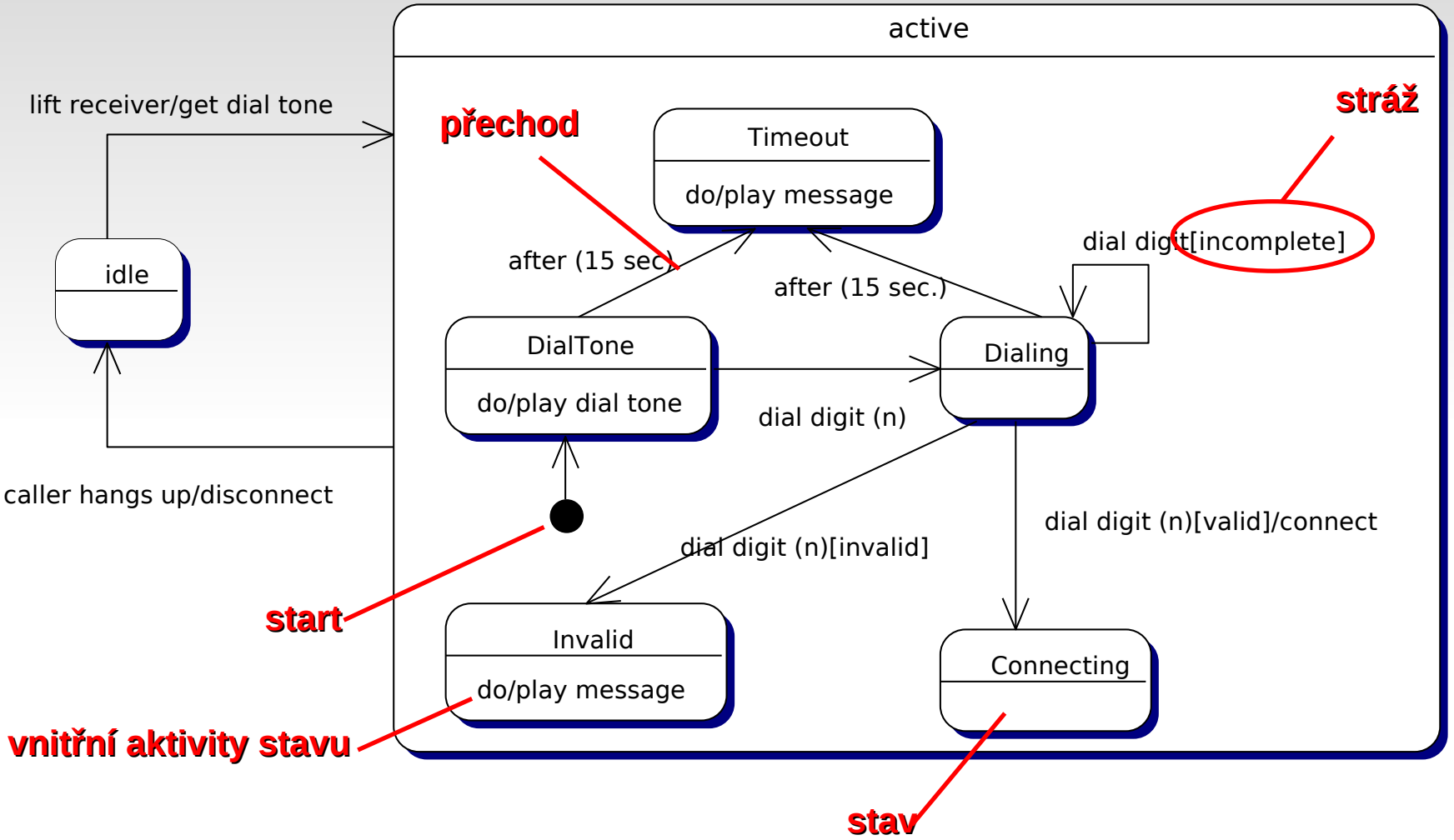
State Transition Diagram – STD

Stavový diagram reprezentuje **stavový automat,** který je grafem **stavů** a **přechodů** a popisuje **odezvu objektu** dané třídy na přijetí **vnějšího stimulu**.



- jeden diagram pro každou třídu vykazující zvláštní (důležité) chování
- zpráva je stimul odpovídající události na kterou má objekt reagovat
- objekt reaguje změnou svého stavu a/nebo provedením operace
  - objekt může reagovat na stejnou zprávu různě v závislosti na stavu
- stavový diagram reprezentuje šablonu pro všechny objekty třídy

STD: UML notace



přechod: událost [stráž] / akce

STD: přechody

- **Notace:**

  událost (seznam_argumentů) [stráž] / akce_výraz ^ vysílací_doložky

- **Událost**
  - vnější stimul, který může vést ke změně stavu
  - libovolné jméno, kromě ***entry***, ***exit*** a ***do***
  - Př: *stisknuté tlačítko (n)*
- **Stráž** (nepovinná)
  - podmínka je platná v určitém časovém rozmezí a tudíž **není stimulem**
  - booleovský výraz používající parametry spouštěcí události a atributy, propojení nebo stavy objektu, kterému patří stavový diagram, nebo objektu, který je dosažitelný (pomocí propojení)
  - Př. špatně: *teplota > 100*
  - Př. správně: *signál z čidla (teplota) [teplota > 100]*
- **Akce_výraz** (nepovinná)
  - reakce na událost
  - atomická nepřerušitelná operace, operace jsou prováděny sekvenčně
  - mohou přiřazovat hodnoty atributům a spojením
- **Vysílací_doložka** (nepovinná)
  - Př: *cílový-výraz . jméno-operace-nebo-signálu (seznam-argumentů)*

STD: příklady přechodů

Příklad komplexního přechodu mezi stavy:

```
right-mouse-down(location) [location in window] /
 object := pick-object(location) ^ object.highlight()
```

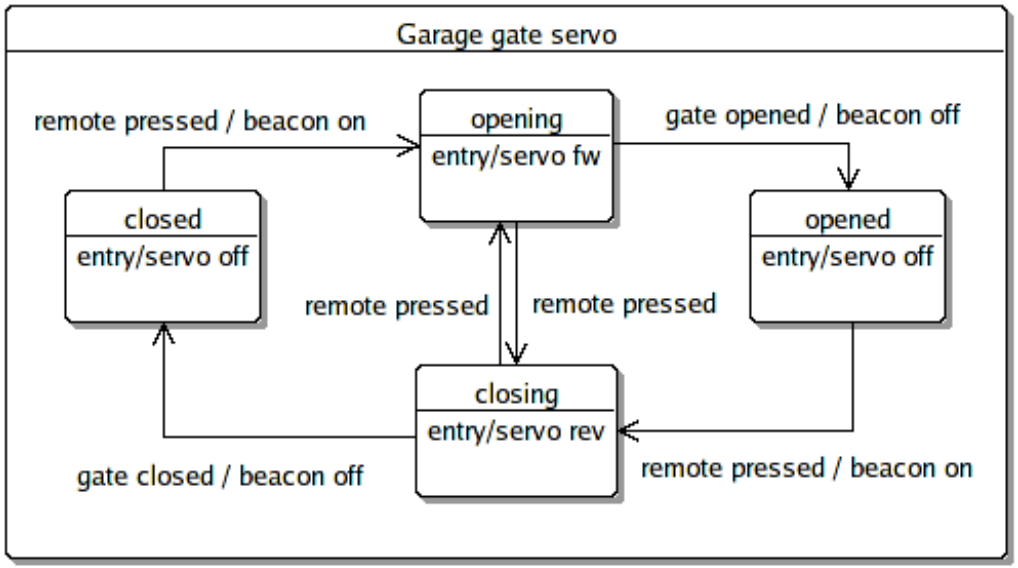
Příklady událostí:

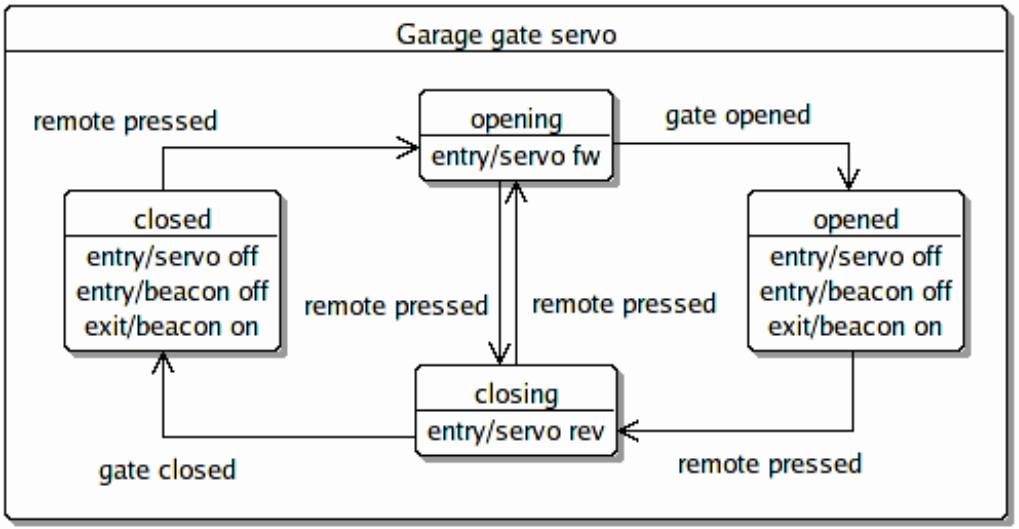
- objekt přijme volání operace (*CallEvent*)
  Př: *jméno-operace (seznam parametrů), tj. startAutopilot(normal)*
- explicitní signál od jednoho objektu nebo systému k jinému (*SignalEvent*); signály jsou asynchronní, vznikají, vznikají a zanikají, mají parametry a hierarchie dědičnosti => jsou modelovány jako třídy se stereotypem <<*signal*>>
  Př: *jméno-třídy (seznam parametrů), tj. Collision(5.3)*
- vypršení stanovené doby od výskytu jiné události (*TimeEvent*)
  Př: *after (2 seconds), at (11:49PM)*
- podmínka je splněna (*ChangeEvent*)
  Př: *when (altitude < 1000)*

STD: Vnitřní události stavu

1) Vyvolání vnořeného stavového automatu (událost ***do***):
- **do *l jméno_vnořeného_stavového_automatu (seznam argumentů)***
- vyvolání stavového automatu zakresleného v jiném diagramu
  ==> jeden ze způsobů rozdělení diagramu
- ostatní aktivity považujte za stavové automaty „bez diagramu“

2) Vstupní a výstupní akce (událost ***entry*** a ***exit***):
- alternativa k zobrazování akcí v přechodech
- používá se pokud všechny přechody z/do stavu vykonávají stejnou akci
- nepoužívat na úrovni analýzy, ale až v návrhu

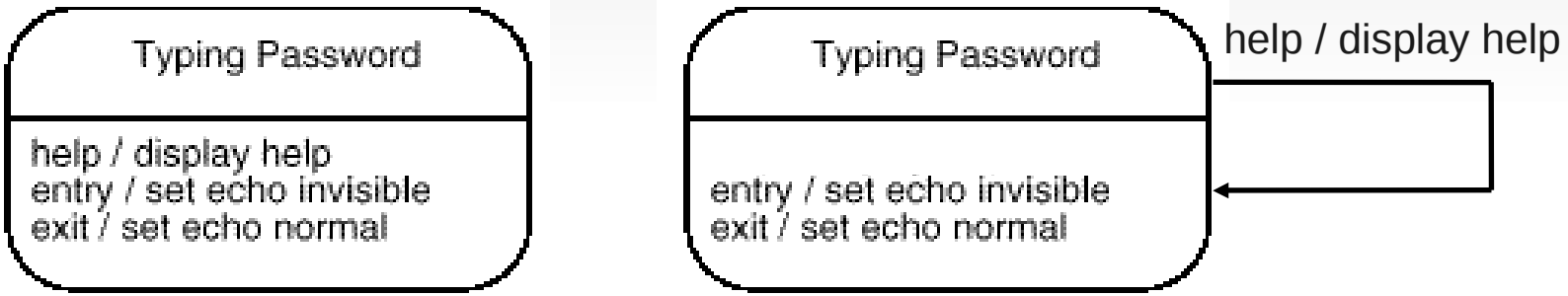




STD: Vnitřní události stavu (II)

3) Ostatní vnitřní události:

- události, které nastanou v daném stavu a *ponechávají původní stav*
- alternativa k externí akci na lokálním přechodu
- Př: když nastane událost ‘help’, je provedena akce ‘display help’:

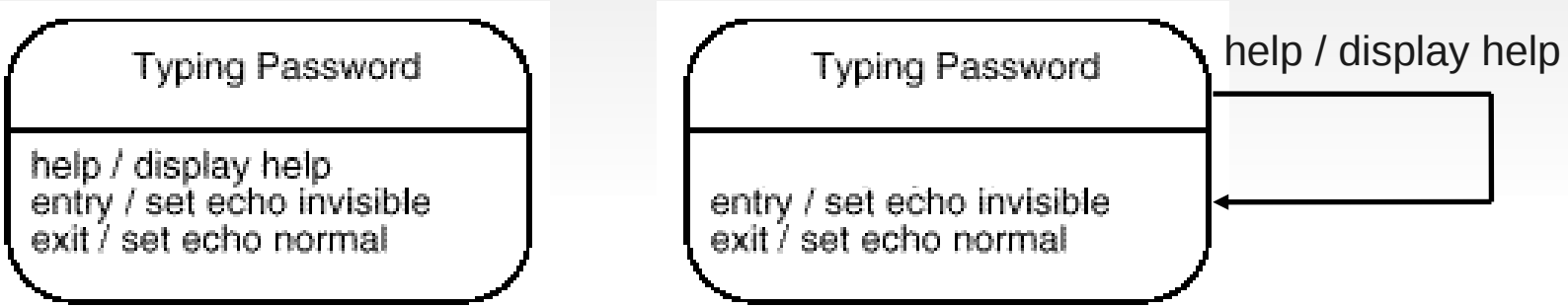


rozdíl mezi těmito diagramy?

STD: Vnitřní události stavu (III)

3) Ostatní vnitřní události:

- události, které nastanou v daném stavu a *ponechávají původní stav*
- alternativa k externí akci na lokálním přechodu
- Př: když nastane událost ‘help’, je provedena akce ‘display help’:



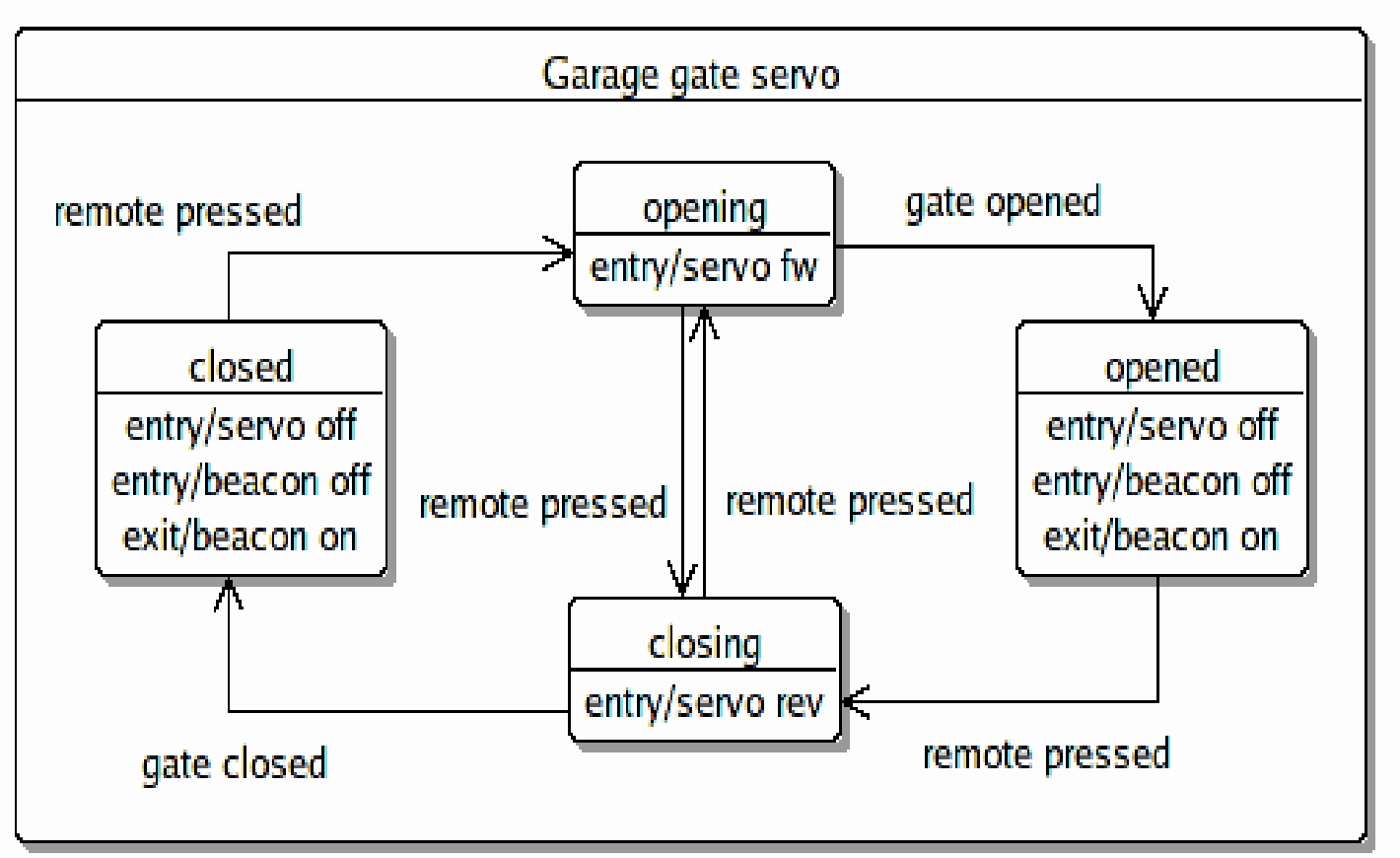
rozdíl mezi těmito diagramy?

- ***externí akce*** opouští stav a znovu do něj vstupuje
  => spustí se ***enty*** a ***exit*** akce
- ***entry***, ***exit*** a ***interní akce*** často vedou na privátní operace dané třídy,
  externí akce často vedou na veřejné operace

STD: Pořadí vykonávání akcí

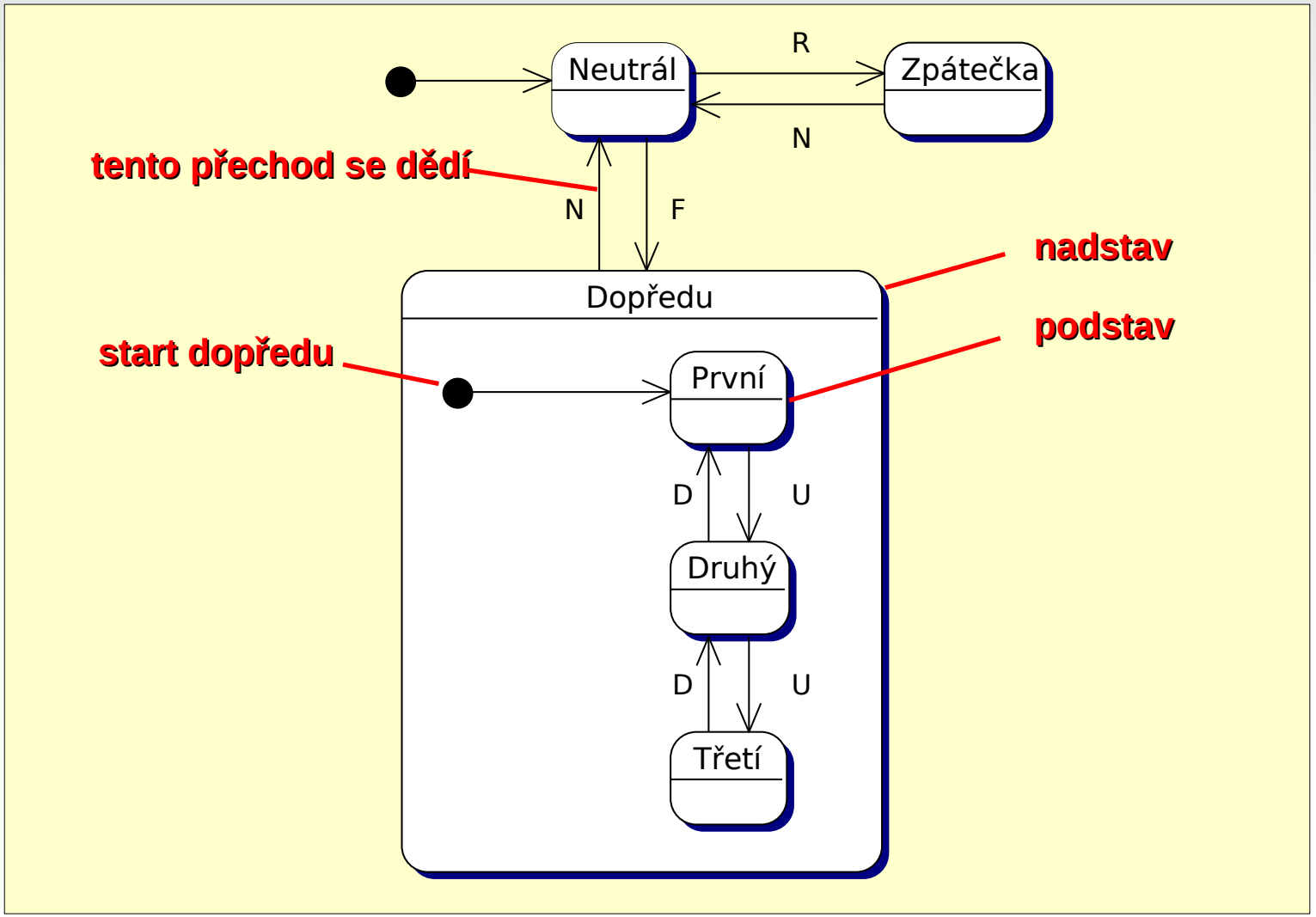
Pořadí vykonávání akcí:

1. akce vstupních přechodů
2. *entry* akce
3. Vnitřní akce a *do* (vnořené STD)
4. *exit* akce
5. akce výstupních přechodů



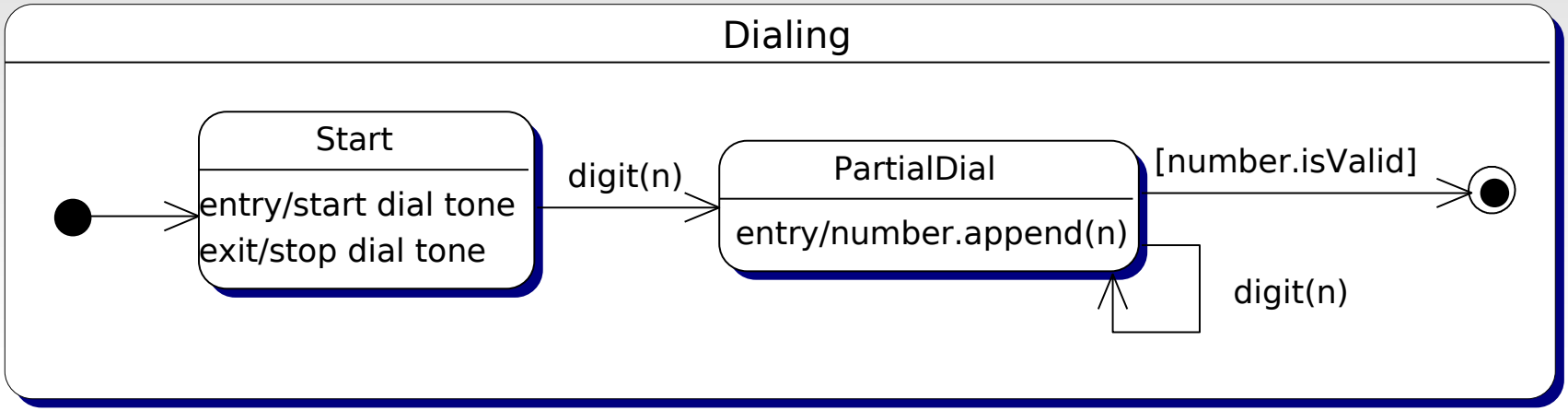
STD: vnořené stavy = generalizace

- Podstavy dědí všechny přechody nadstavů, mohou je ale předefinovat
- Použití generalizace je velmi vhodný postup pro zpřehlednění složitých STD



STD: vnořené stavy (*or*-vztah)

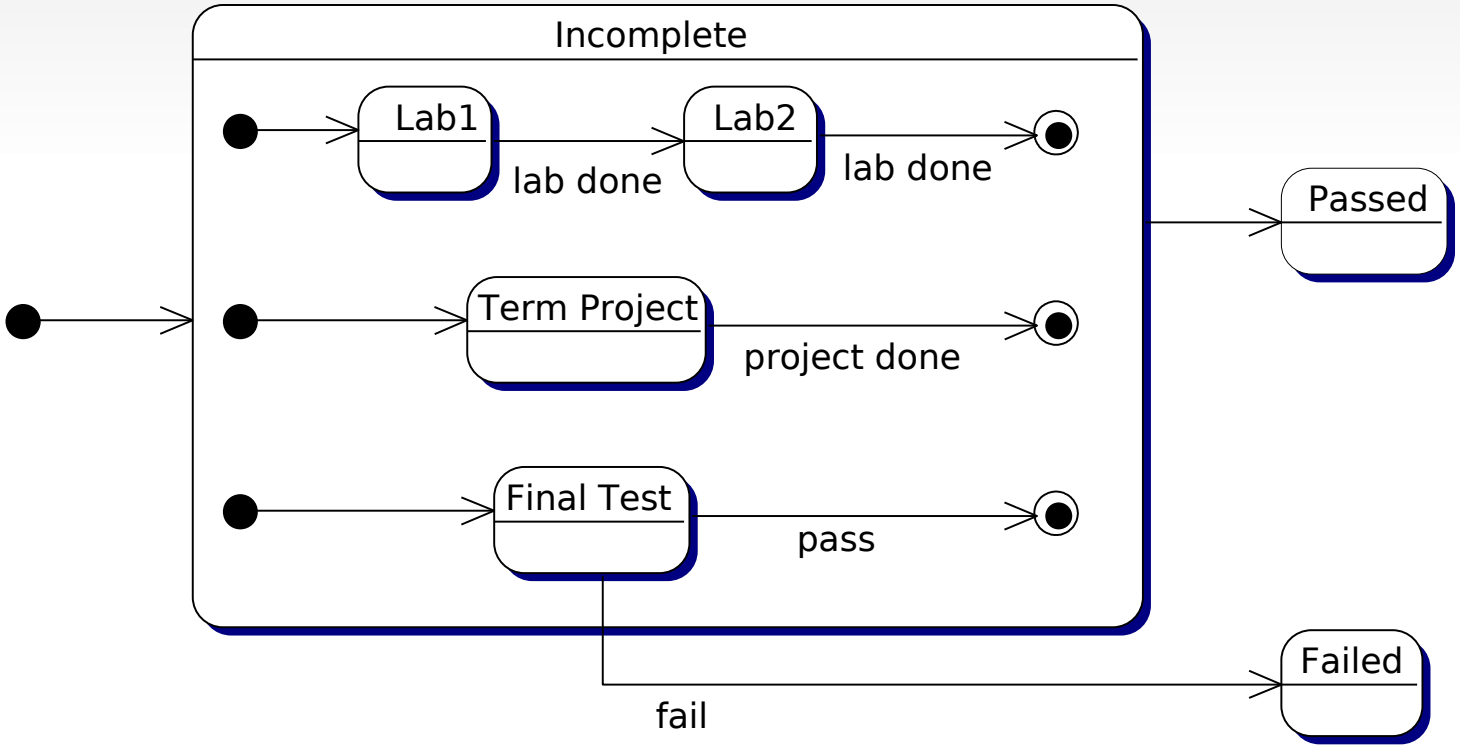
- stav ‘Vytáčení’ je rozložen s použitím ***or*-vztahu** na **vzájemně výlučné dílčí stavy**



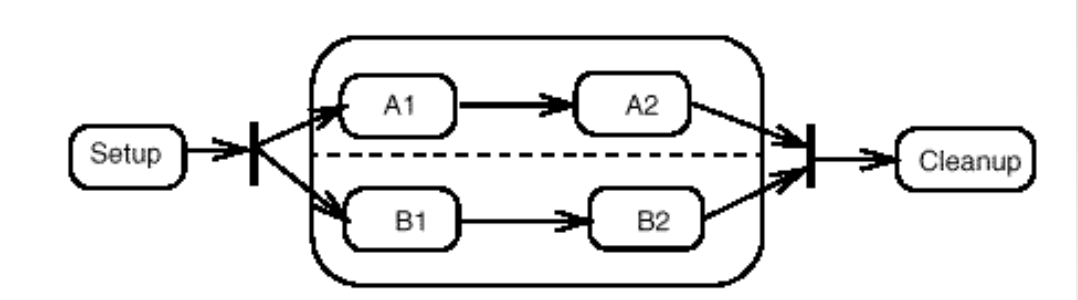
- **počáteční** stav - **koncový** stav během ‘Vytáčení’
- **aktivity** dány **pomocí vstupní a výstupní akce**
- **události** mohou být „metodami“ určitých tříd
- co **popisuje** stavový diagram ? metodu, třídu, …?

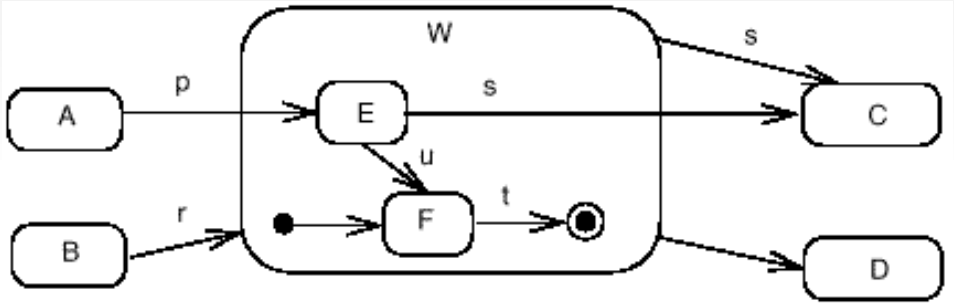
STD: vnořené stavy (*and*-vztah)

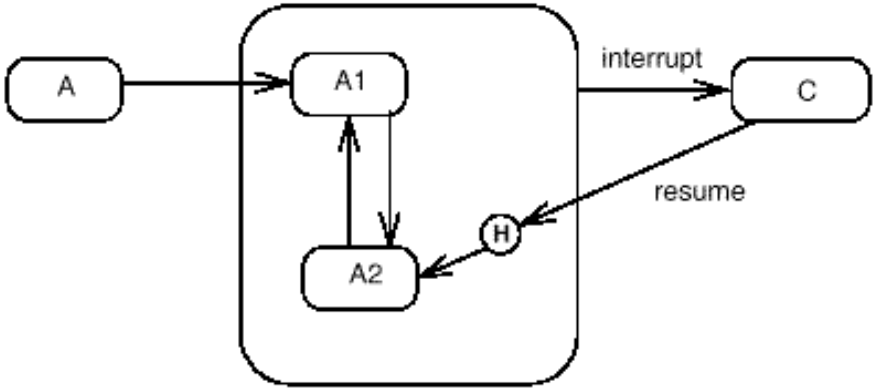
- stav ‘Incomplete’ je rozložen s použitím ***and*-vztahu** na **souběžné dílčí stavy**
- paralelní stavy uvnitř jedné třídy => třída má více zodpovědností => je dobré třídu přezkoumat



STD: komplexní přechody

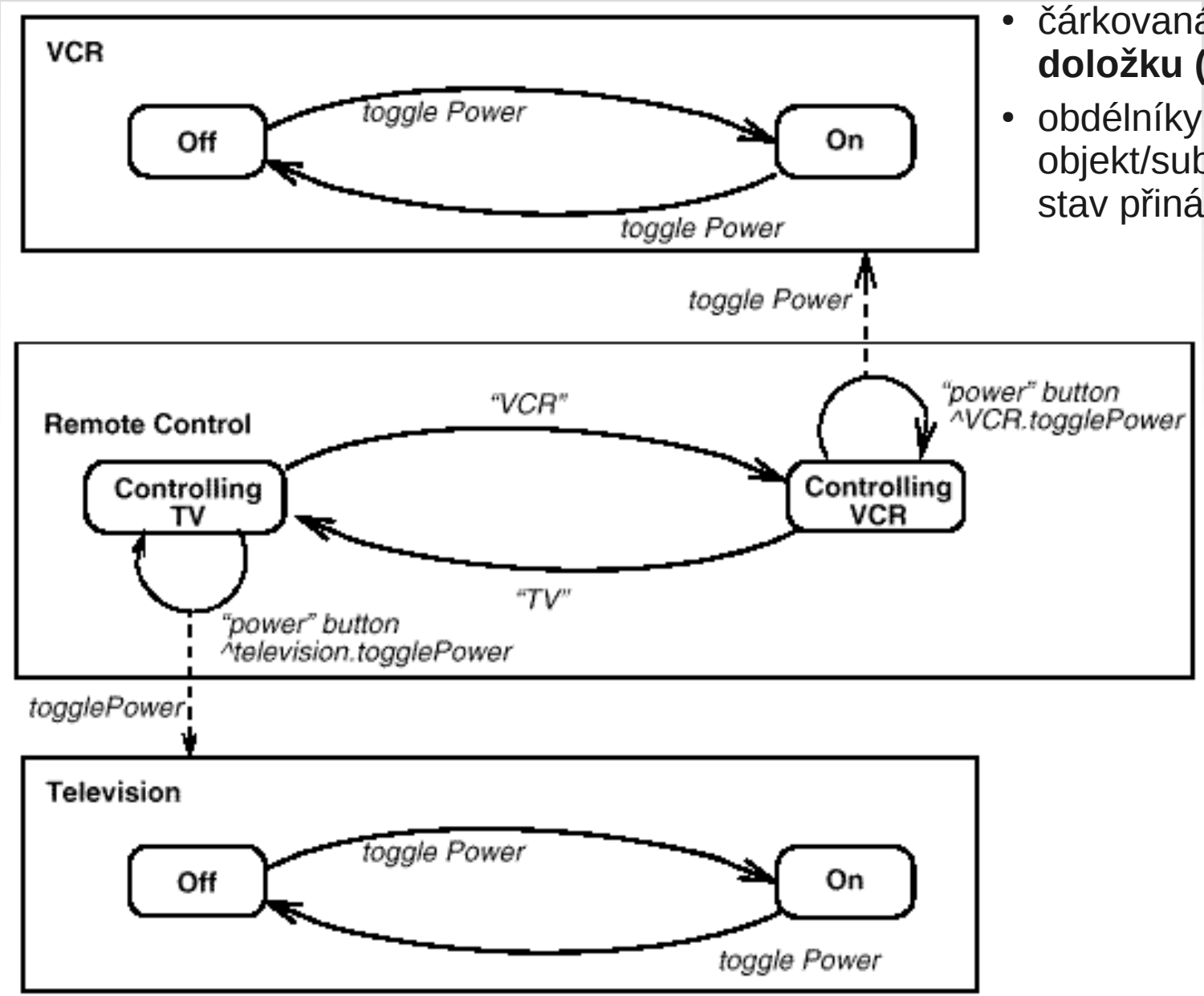






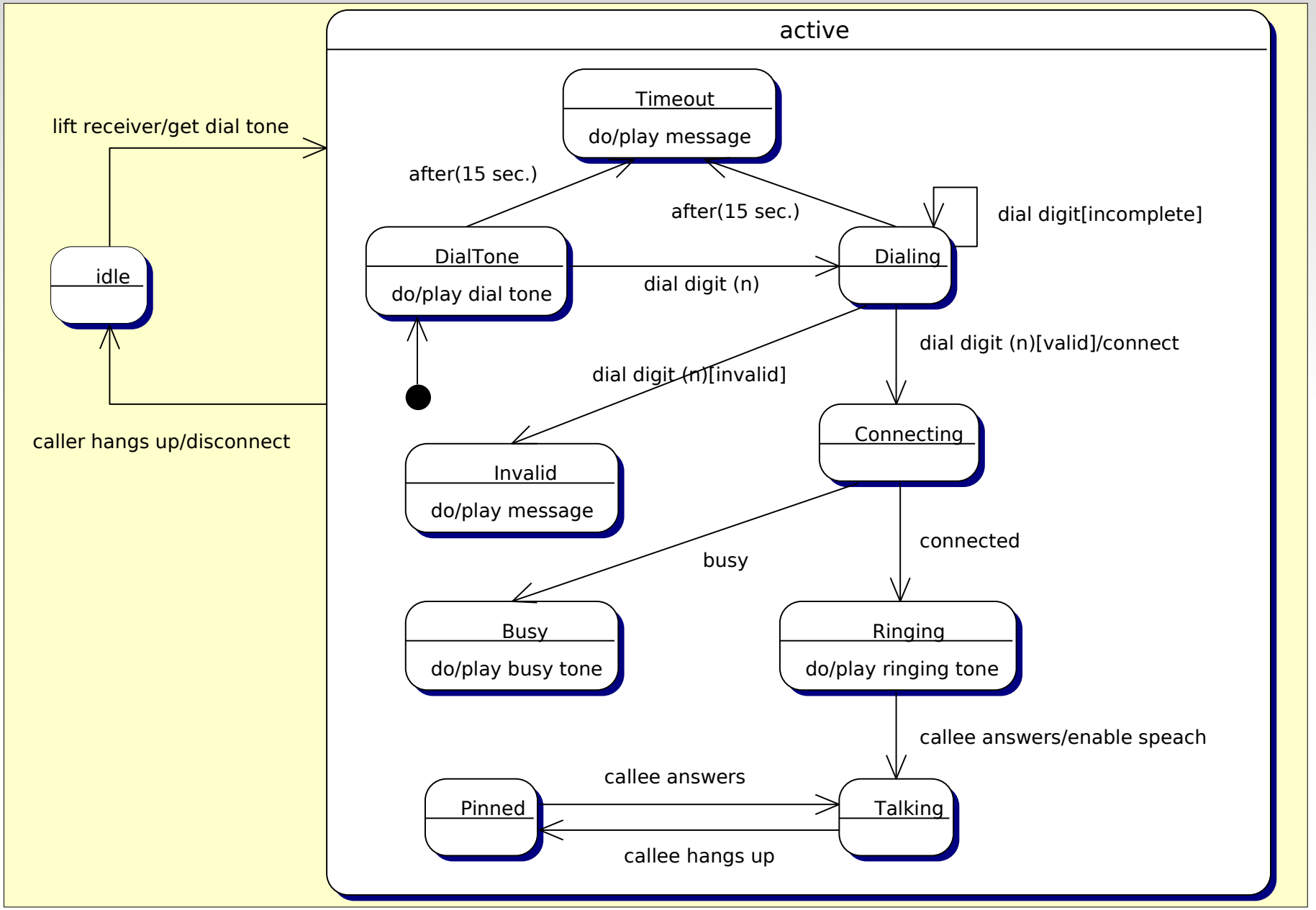
- **komplexní přechody**: všechny události před závorou musí nastat, než dojde k přechodu do stavů za závorou
- **dědění přechodů**: podstavy E a F dědí přechod s událostí 's', podstav E tento zděděný přechod navíc předefinovává
- **indikátor historie**: A2 je vybráno poprvé, později po *resume* následuje přechod do stavu, který byl aktivní předtím, než došlo k přerušení

Interakce ve stavových diagramech



- čárkovaná šipka označuje **vysílací-doložku (send-clause)**
- obdélníky označují objekt/subsystém, k němuž tento stav přináleží

STD: příklad


active
lift receiver/get dial tone
idle
caller hangs up/disconnect
Timeout
do/play message
after(15 sec.)
after(15 sec.)
dial digit[incomplete]
DialTone
do/play dial tone
dial digit (n)
Dialing
dial digit (n)[valid]/connect
dial digit (n)[invalid]
Connecting
Invalid
do/play message
busy
connected
Busy
do/play busy tone
Ringing
do/play ringing tone
callee answers/enable speach
callee answers
Pinned
Talking
callee hangs up

Proč vytváříme STD

- Pochopení posloupnosti chování objektu v průběhu času
- Vyjasňuje závislost chování na stavu
- Odhaluje skryté atributy
- Pomáhá rozpoznat chybějící a skryté operace
- Definuje sekvence operací a zpráv

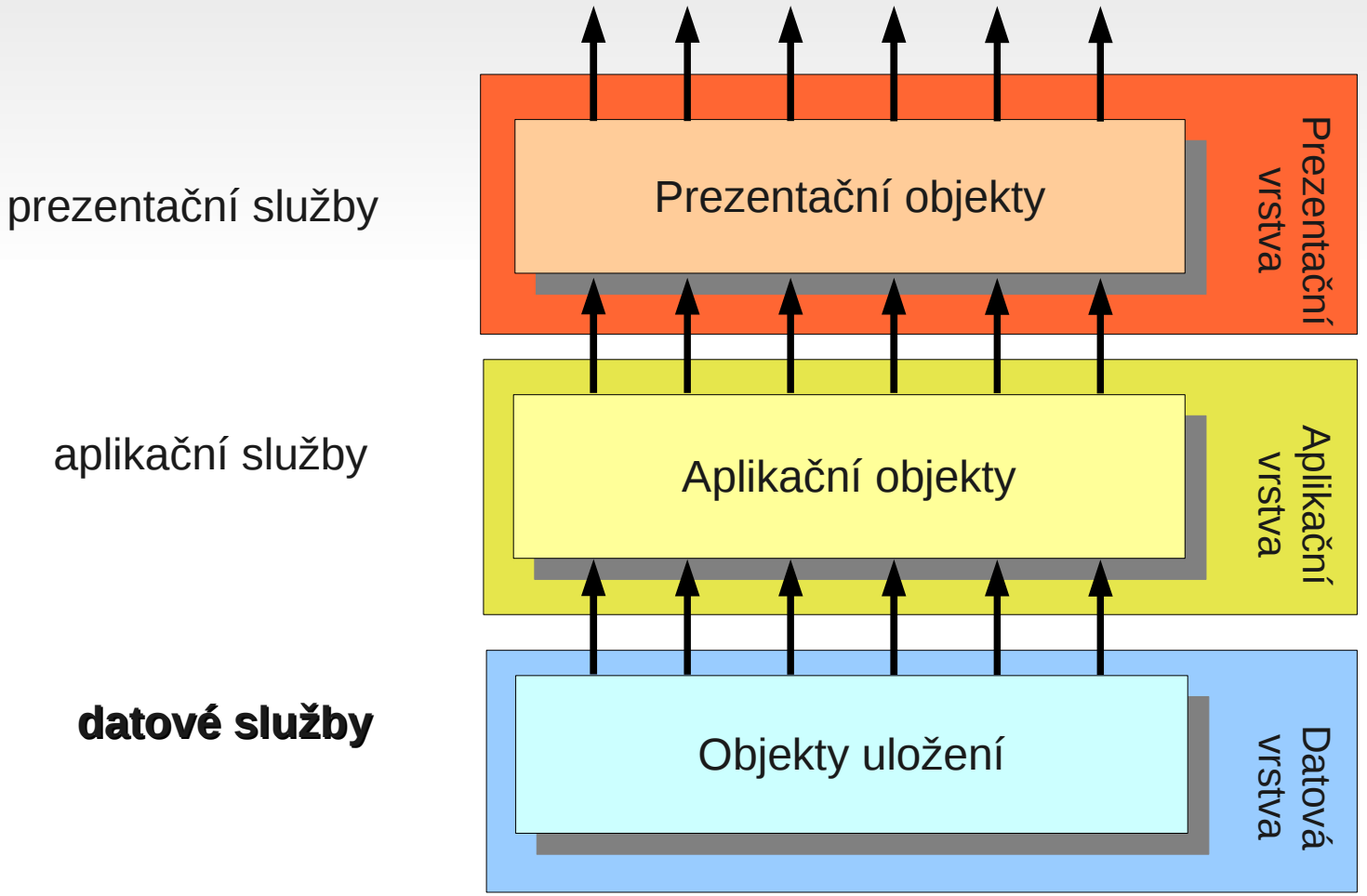
STD: kontrola konzistence

- Kontrola konzistence STD
  - Každý stav musí mít předchůdce (kromě iniciálního stavu)
  - Stavy bez následníka (kromě koncového stavu) jsou podezřelé
  - Projděte efekty vstupních událostí systému a zkontrolujte soudržnost
  - Ověřte, že stejná událost na různých STD je modelována konzistentně
- Konzistence mezi STD a diagramy tříd
  - Vyskytují se všechny aktivity a akce jako operace na diagramu tříd?
  - Podporuje diagram tříd plně stavové diagramy?
    - existují atributy pro modelované stavy?
    - existují atributy pro vyhodnocení podmínek?
    - jsou pro všechny přechodové akce veřejné operace?
    - jsou pro ***entry***, ***exit*** a ***interní akce*** privátní operace?

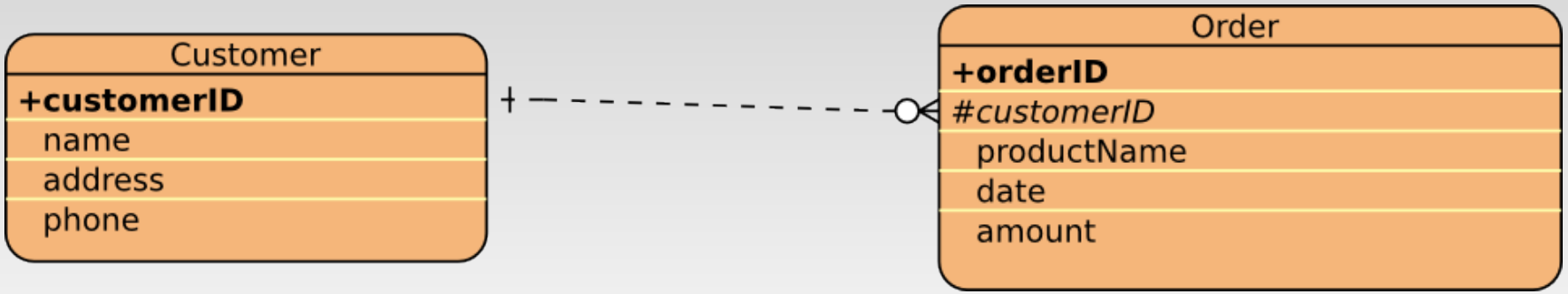
Uložení persistentních objektů

Proč potřebujeme model objektů uložení?

- Široké nasazení relačních databází, které ale nepodporují přímo struktury objektového modelu => nutnost mapování



Relační databáze

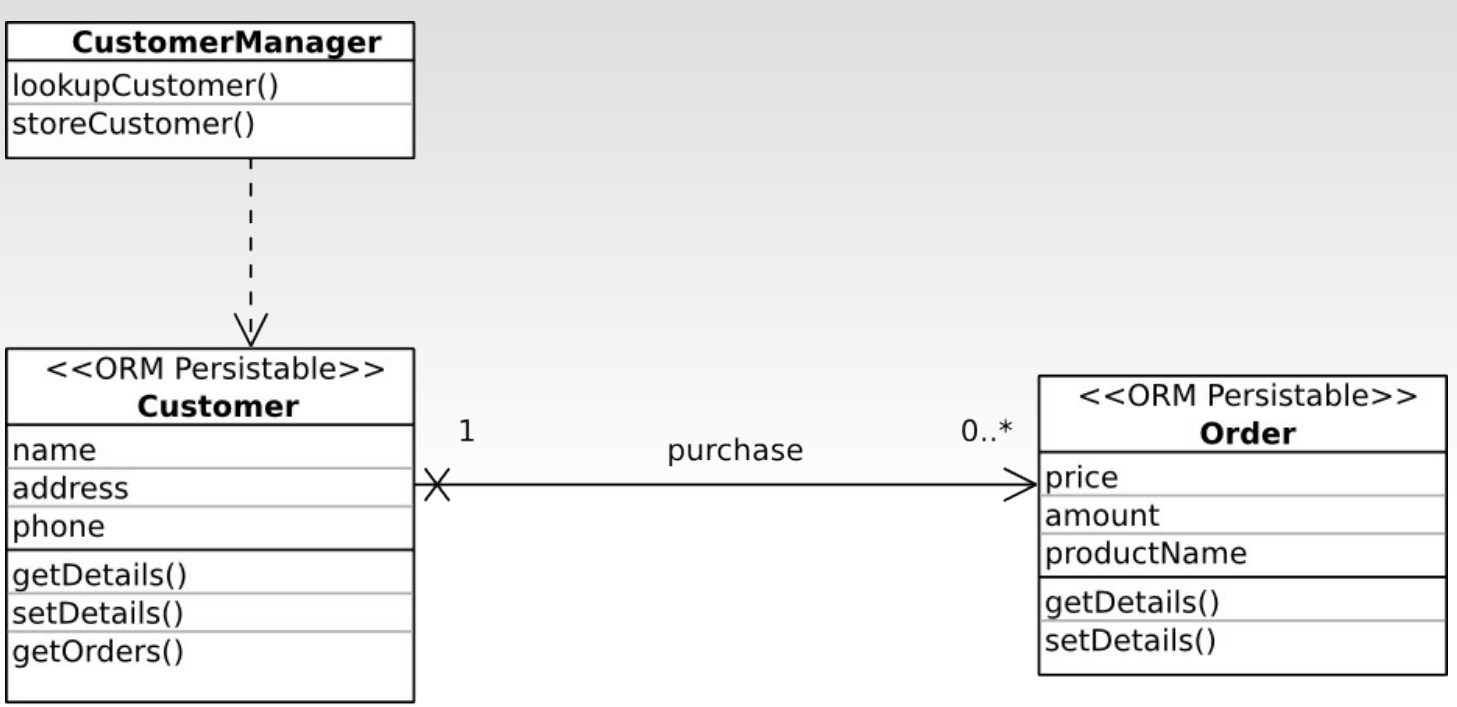


- Relační technologie
  - Uložení dat v tabulkách
  - Řádky jsou záznamy, sloupce typy
  - Tabulky jsou propojeny vazbami
  - Primární/cizí klíče
  - Kardinalita/násobnost vazeb
  - Relační algebra (SQL) pro snadný přístup k datům
- Objektová technologie
  - Třídy obsahují data i operace
  - Asociace s násobností
  - Dědičnost
  - „Asociace a objekty se udržují v paměti“ => operace nad daty jsou řešeny interakcí objektů
  - Př: vyhledání všech studentů, kteří mají zapsaný daný kurz – rozdíl při použití SQL oproti interakci objektů

ORM – základní mapování

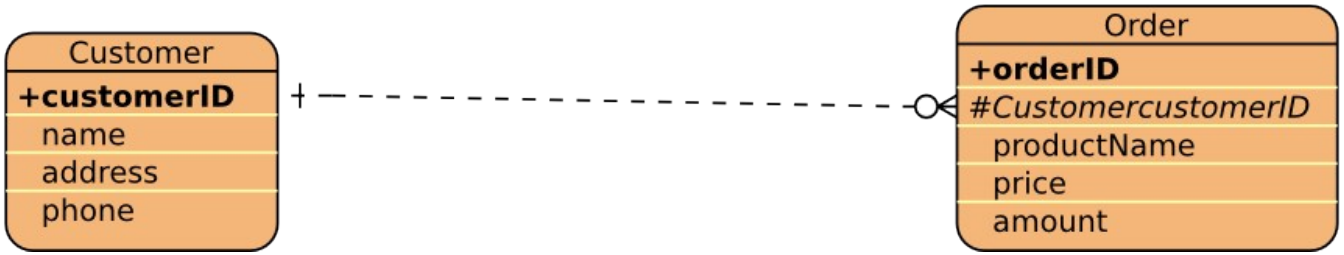
- Object-relational mapping, ORM
  - Persistentní třída definuje entitní množinu (tabulku)
  - Objekt definuje entitu (záznam, řádek tabulky)
  - Atributy třídy se stávají atributy entity (sloupce tabulky)
  - Klíč je vybrán z atributů nebo je vytvořen nový
  - Asociace/agregace/kompozice tříd definuje relaci (propojení tabulek cizími klíči)
  - Dědičnost tříd
    - mapování 1:1
    - zahrnutí do nadtřídy
    - rozpuštění do podtříd

ORM – základní mapování (II)



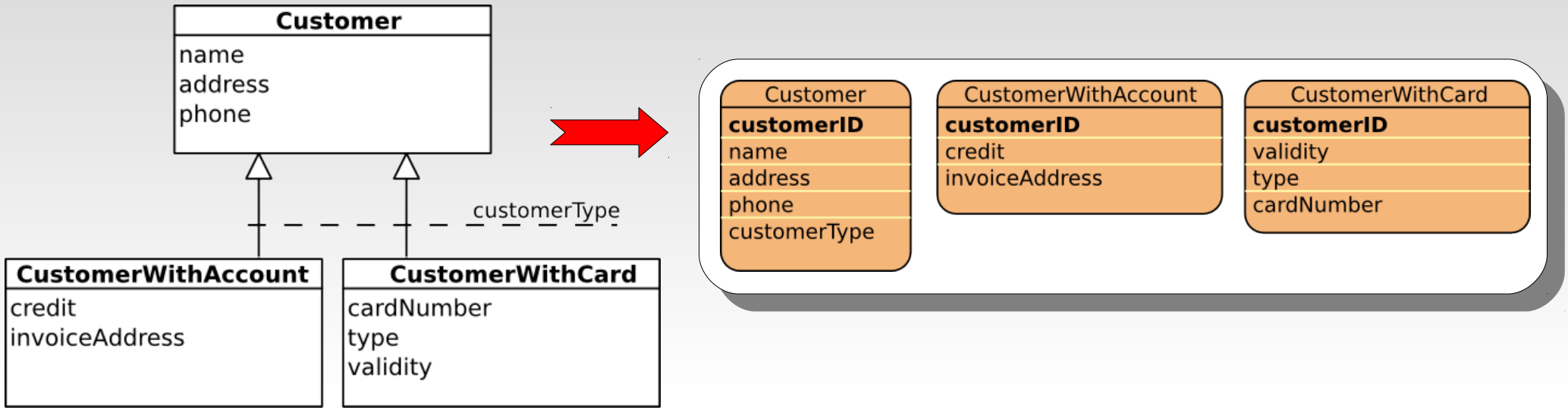
objekty pro práci s DB (SQL dotazy)

persistentní objekty



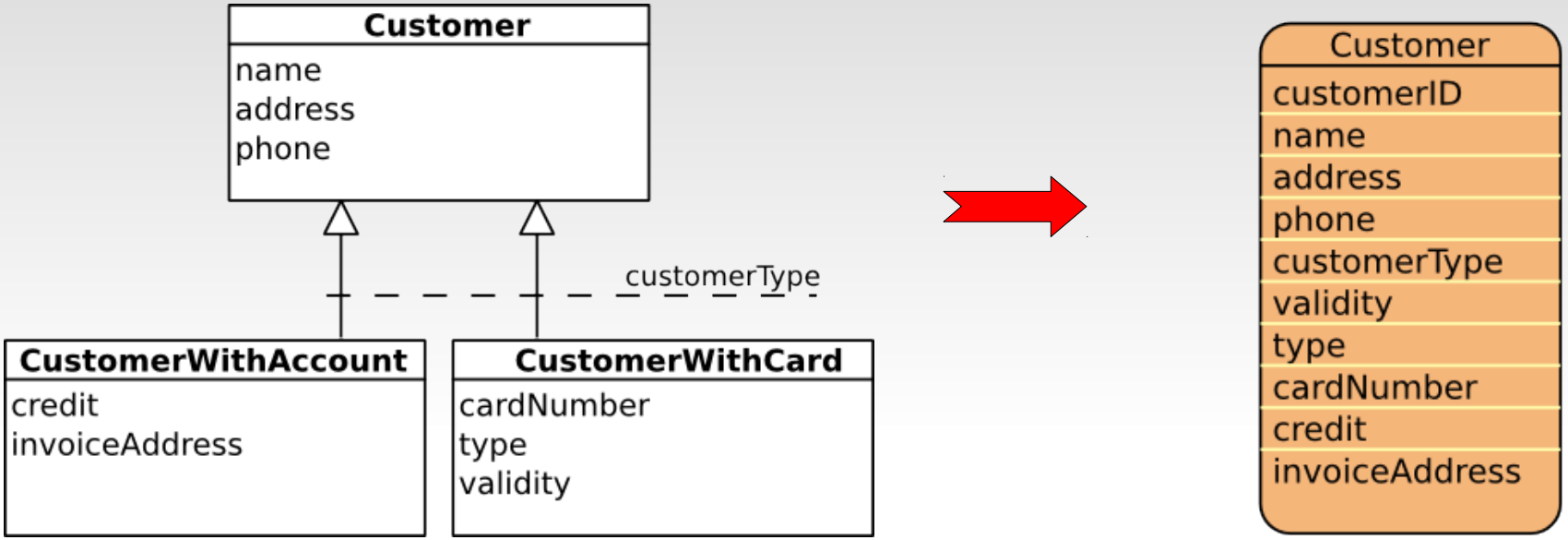
relační schéma

Dědičnost: mapování 1:1



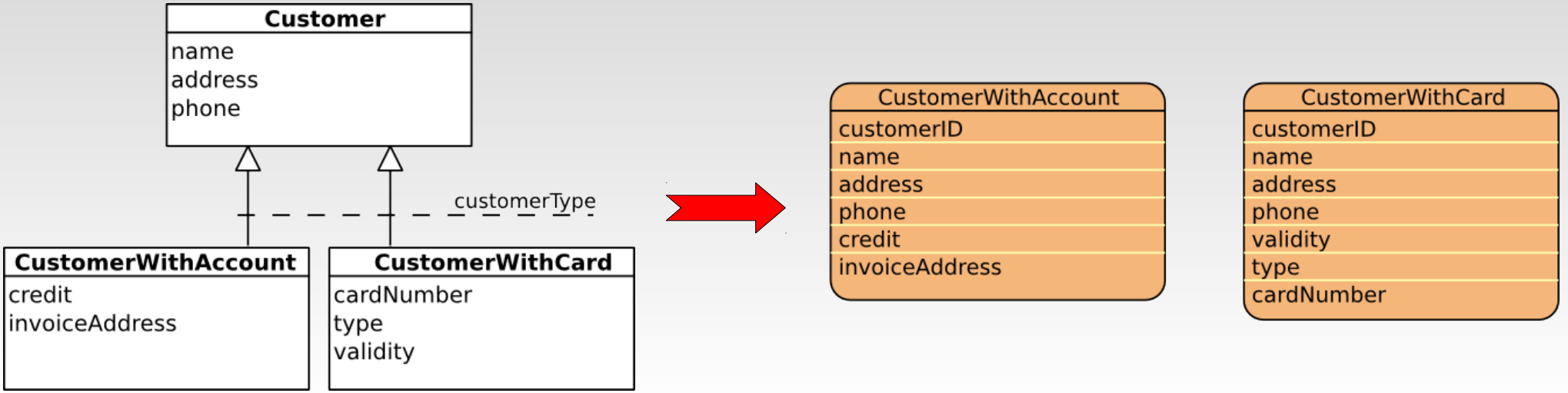
- Každá třída se stává tabulkou
- Všechny tabulky mají stejný primární klíč
- Diskriminátor se stává atributem
  - prohledávají se pouze tabulky příslušné podtřídy
- Jedna instance třídy je uložena v několika tabulkách
  - složitější přístup k datům

Dědičnost: zahrnutí do nadtřídy



- Všechny atributy podtříd jsou zahrnuty do jedné tabulky
- Některé atributy mohou obsahovat hodnotu NULL
  - porušení čtvrté normální formy
- Vhodné v případě menšího počtu podtříd s málo atributy

Dědičnost: rozpuštění do podtříd



- Atributy nadtřídy jsou přeneseny do tabulek pro všechny neabstraktní podtřídy
- Vhodné při:
  - Nadtřída má málo atributů
  - Existuje mnoho podtříd (košatý strom větvení)
  - Podtřídy mají hodně atributů