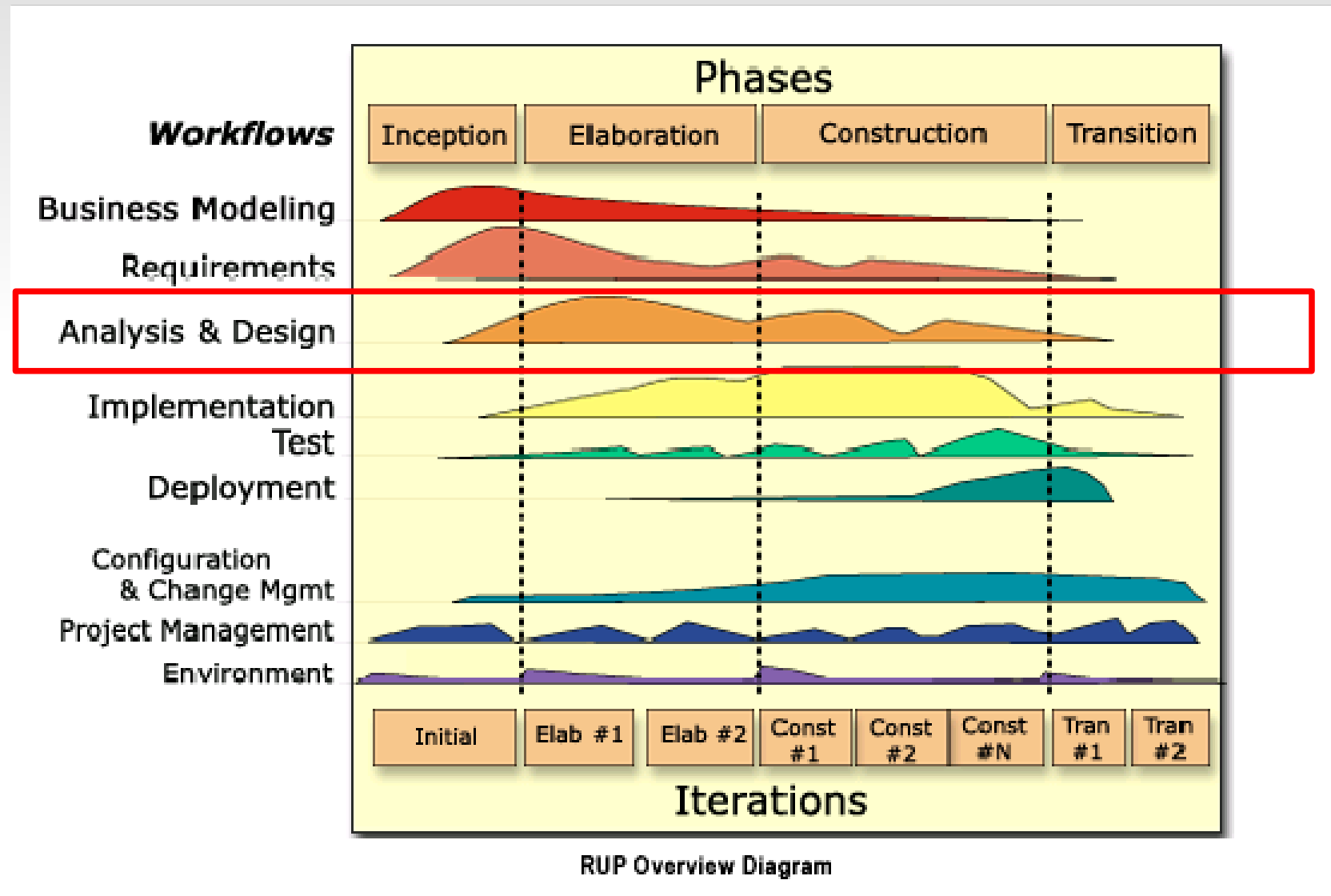


---

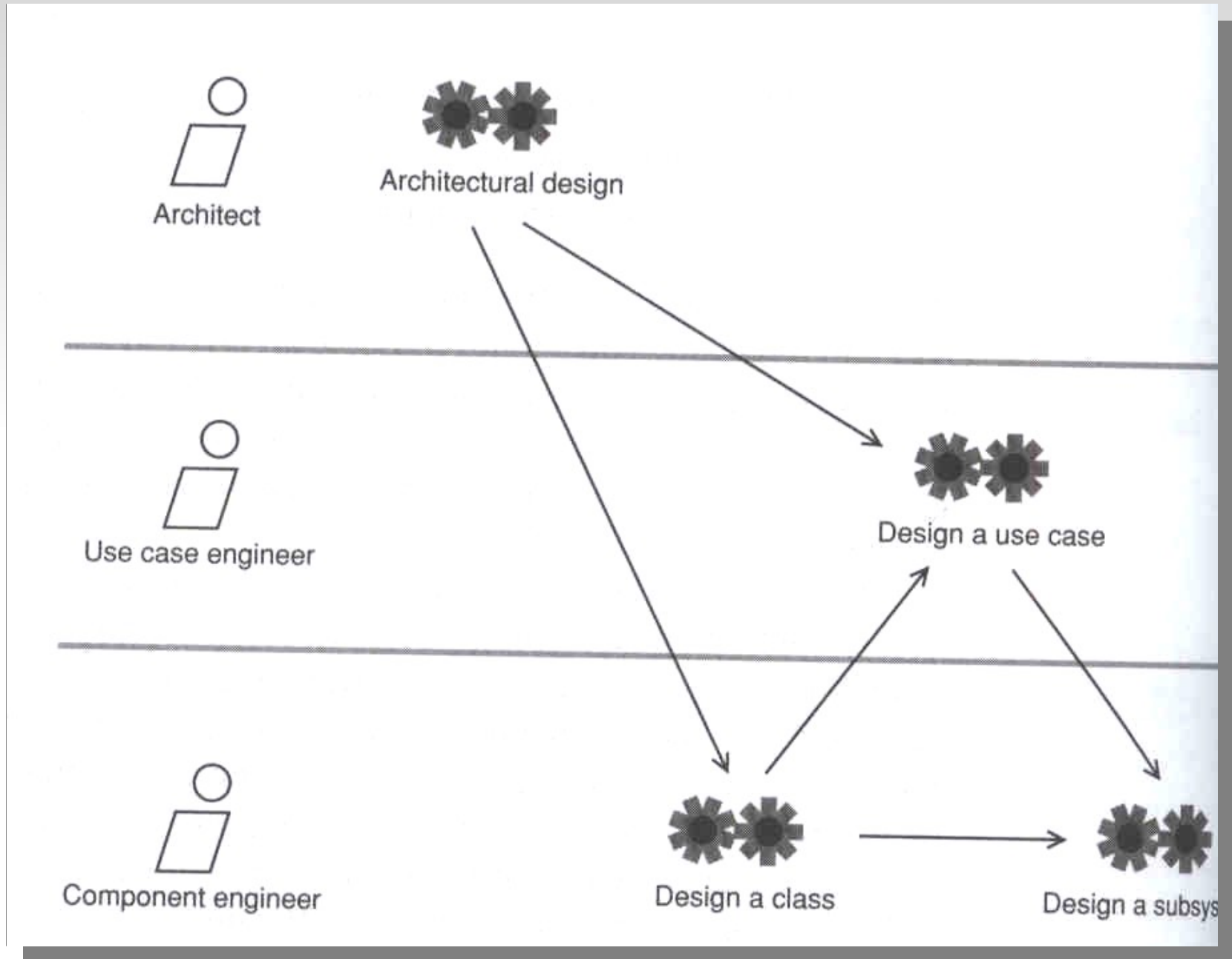
# Návrh (pokr.)

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

# Design workflow



# Design workflow detail

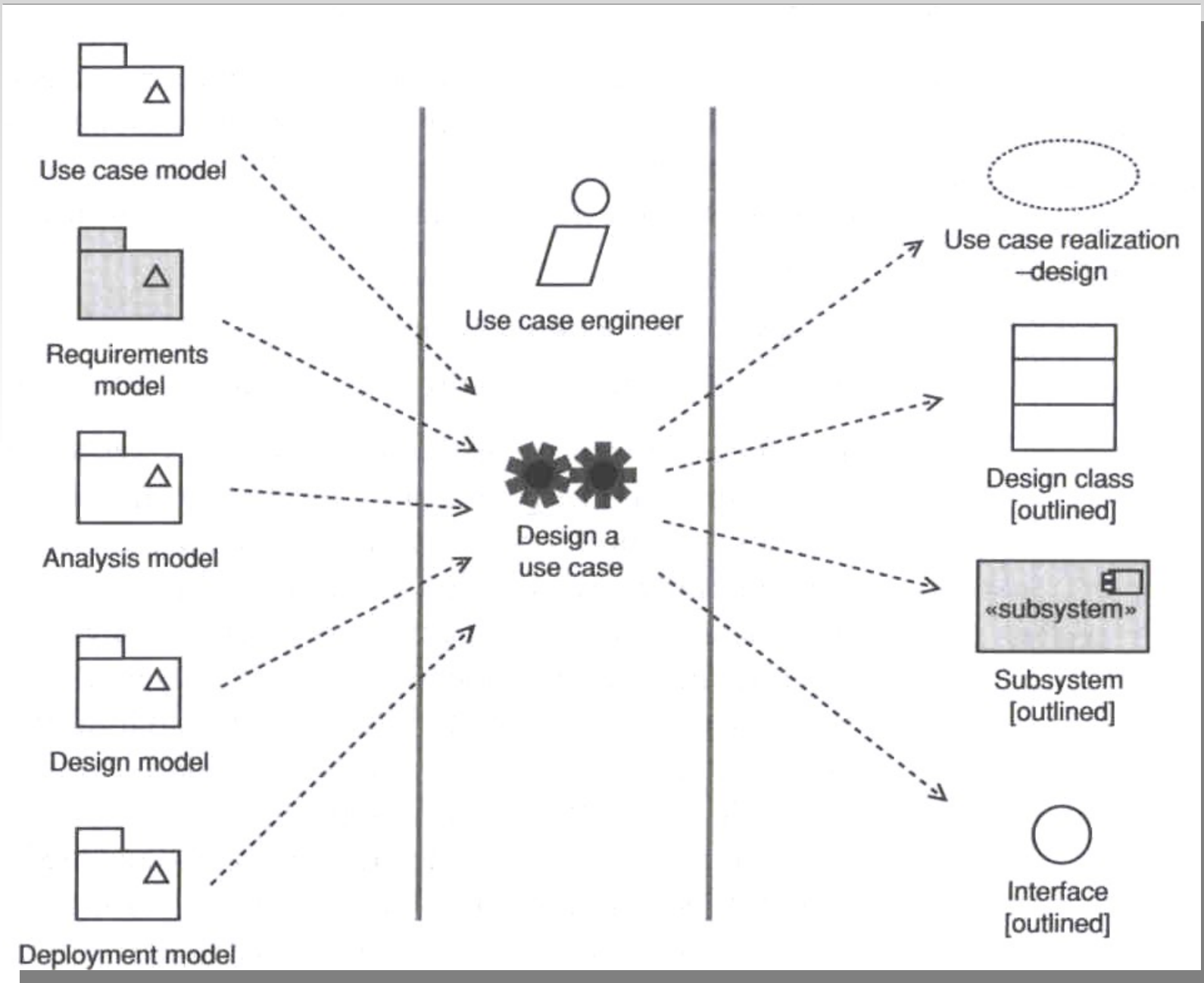


---

# **Realizace případů užití – návrh**

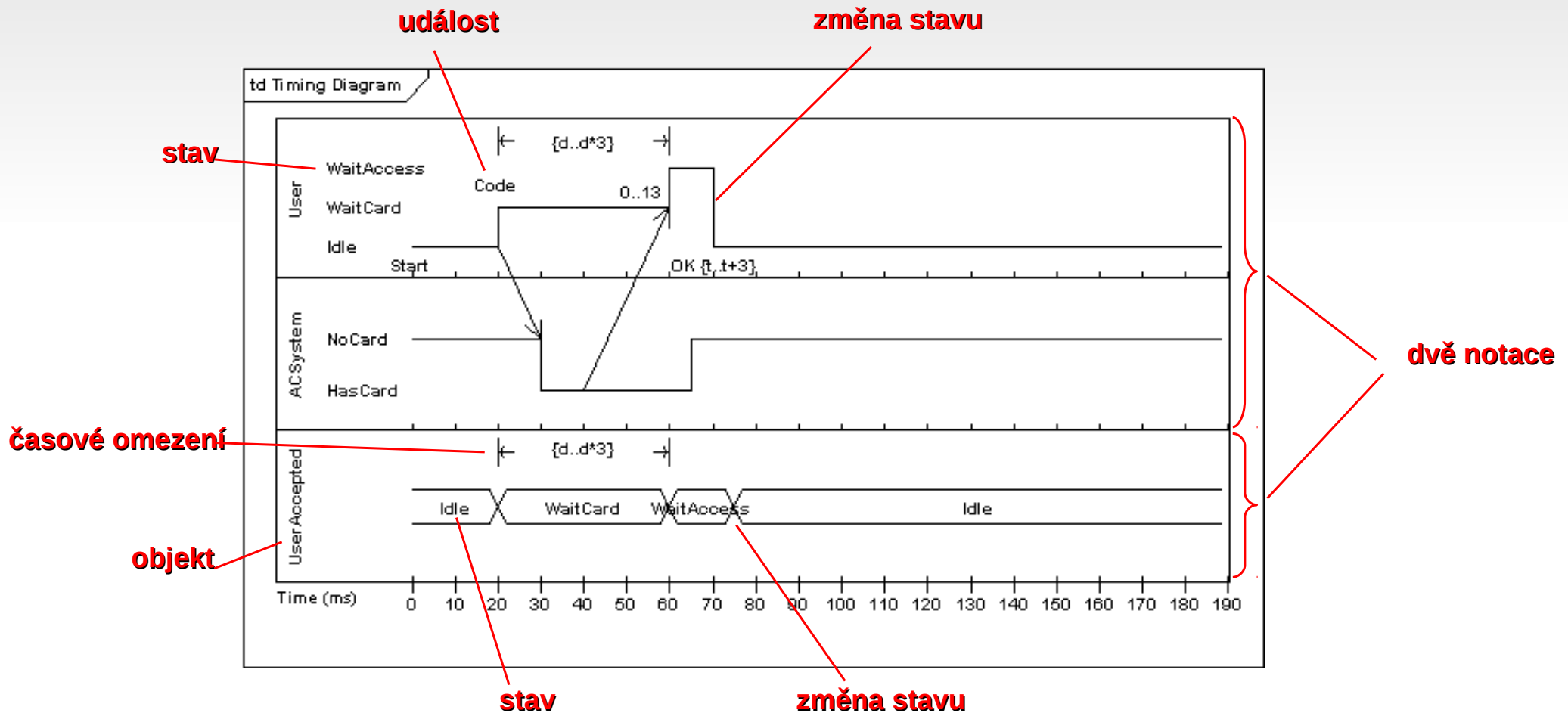
## (Use case realization – design)

# UP aktivita: Design a use case



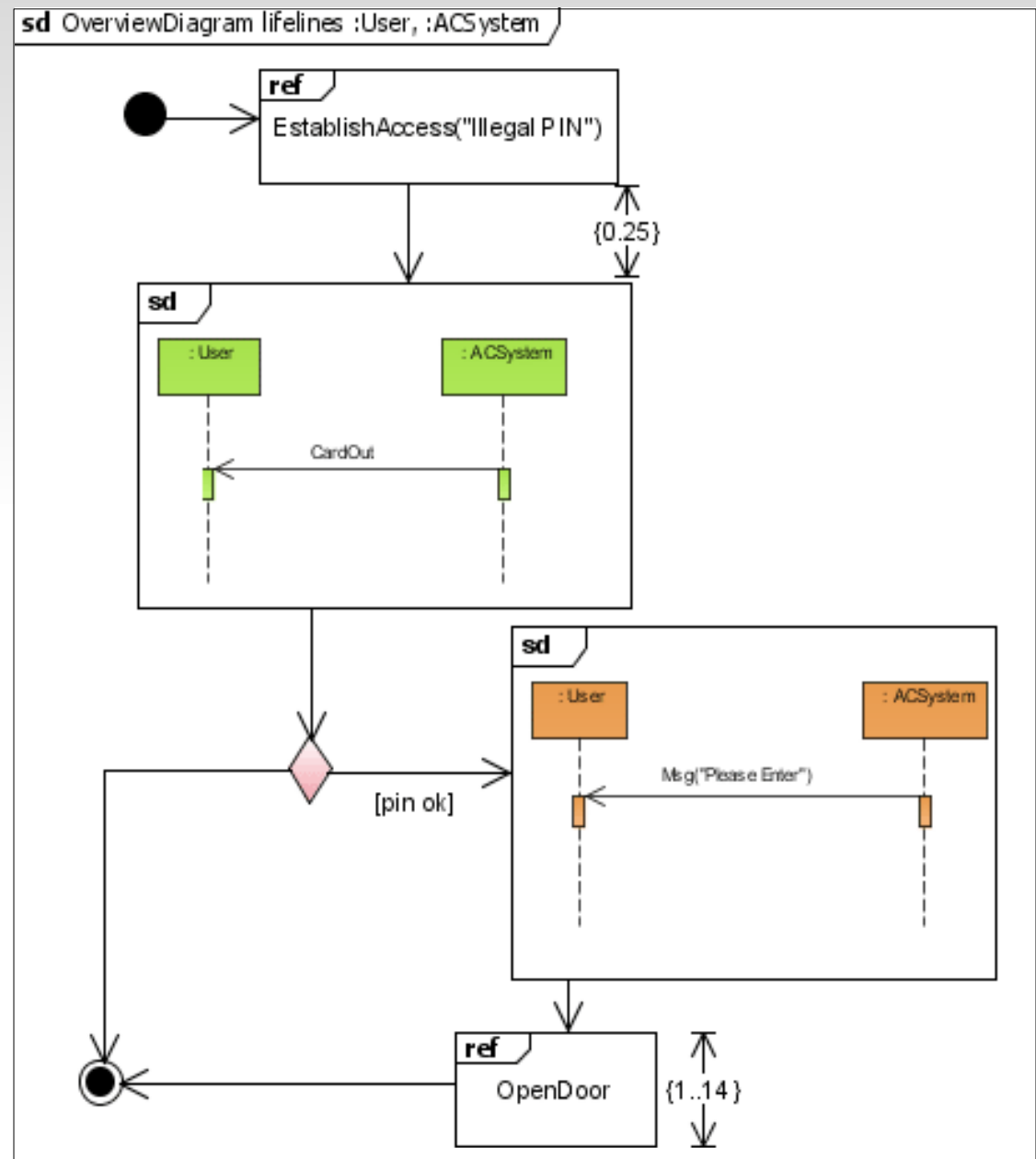
# Diagram časování

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



# Diagram přehledu interakcí

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



---

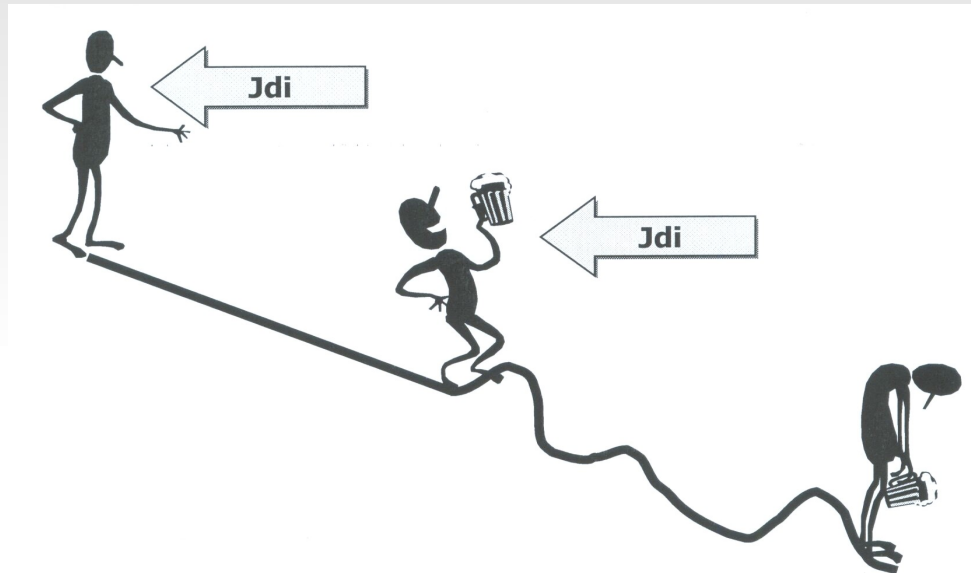
# **Stavové diagramy** (State machines)



# Stavový diagram

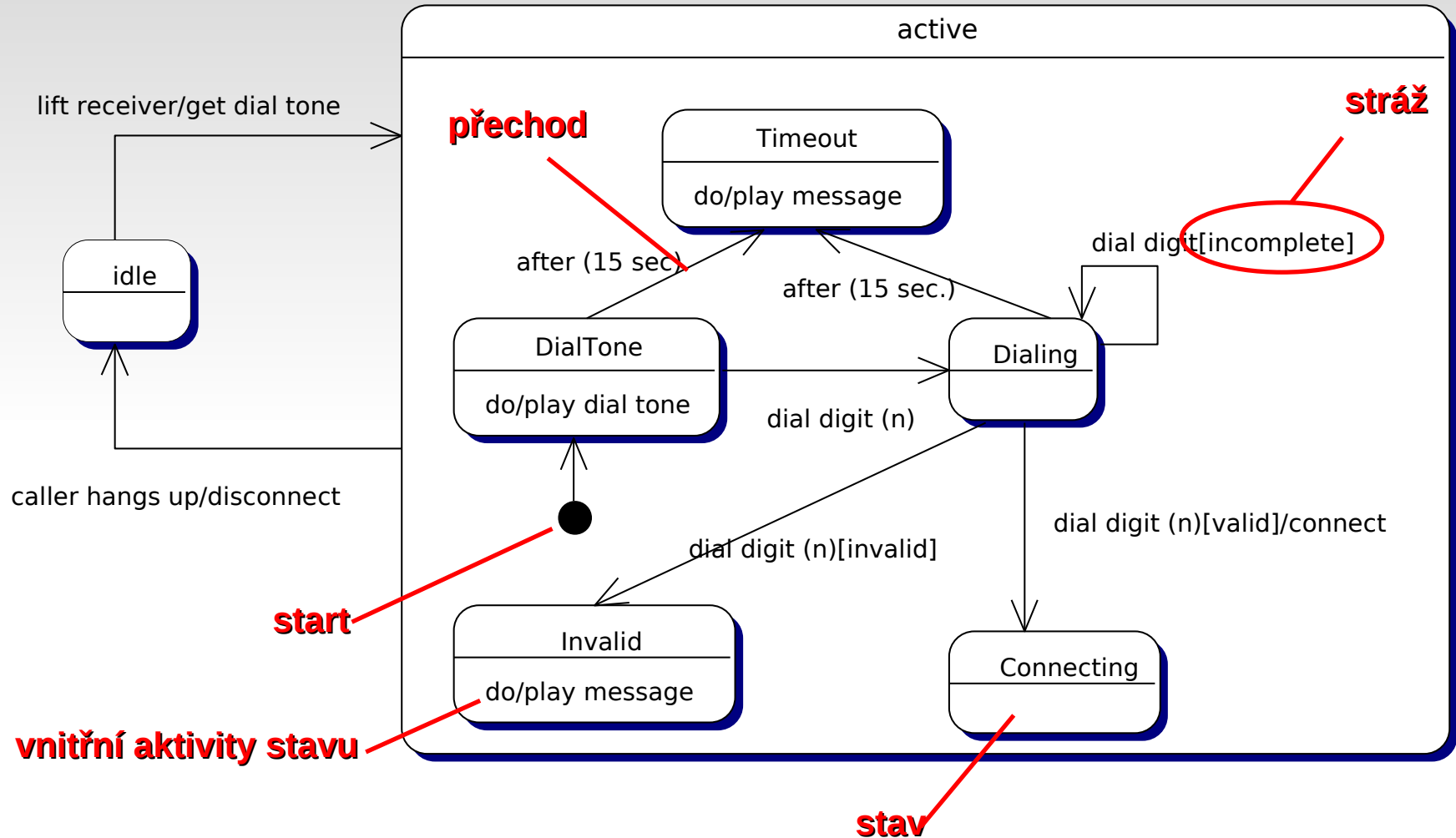
## State Transition Diagram – STD

**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**
- Příklad: *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)
- Příklad špatně: *teplota > 100*
- Příklad 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)**

- Příklad: *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)*

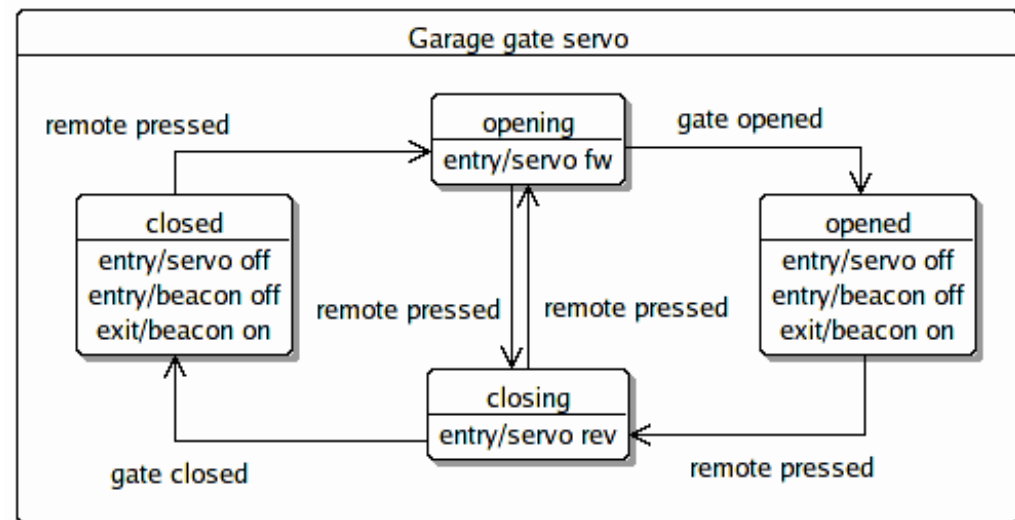
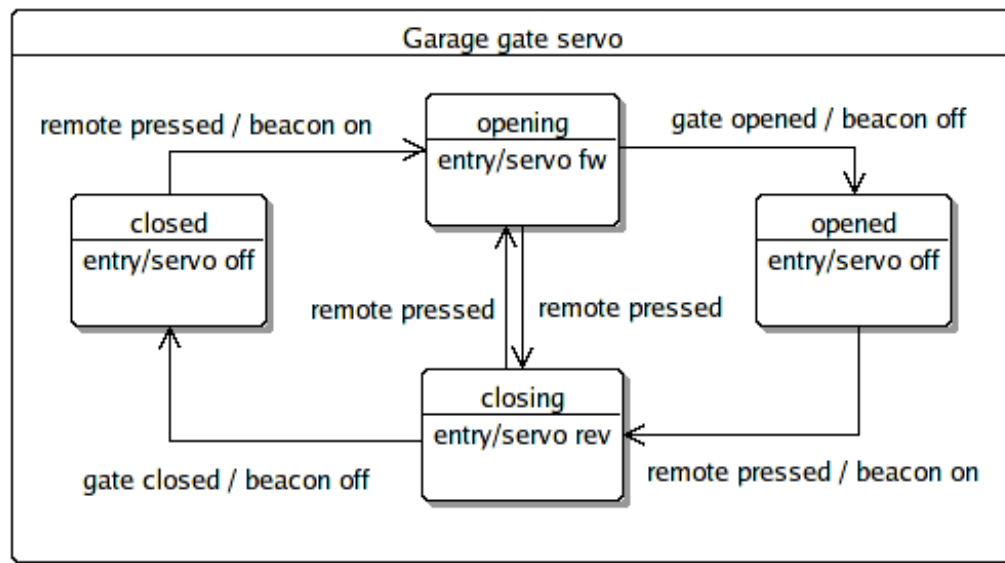
# STD: Vnitřní události stavu

1) Vyvolání vnořeného stavového automatu (událost **do**):

- **do / 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**):

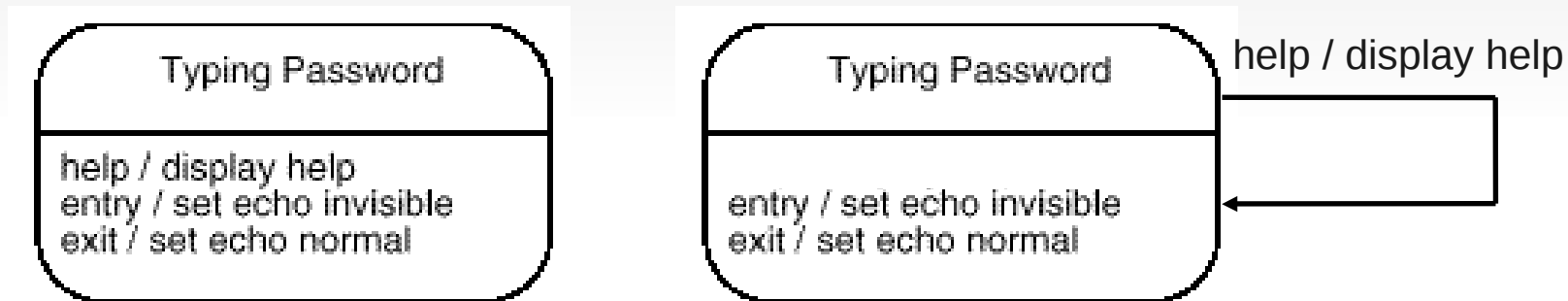
- alternativa k zobrazování akcí v přechodech
- používá se pokud všechny přechody z/do stavu vykonávají stejnou akci
- nepoužívat na úrovni analýzy, ale až v návrhu



# 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':

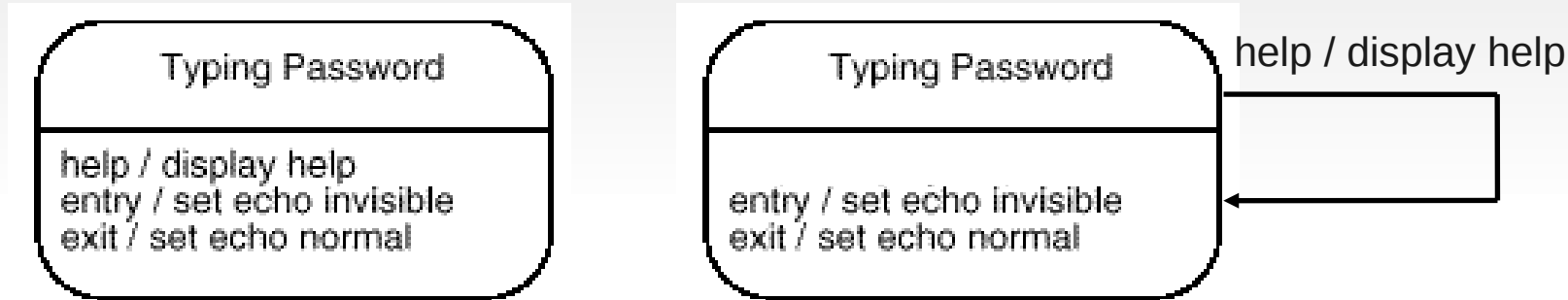


rozdíl mezi těmito diagramy?

# 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
- Příklad: když nastane událost 'help', je provedena akce 'display help':



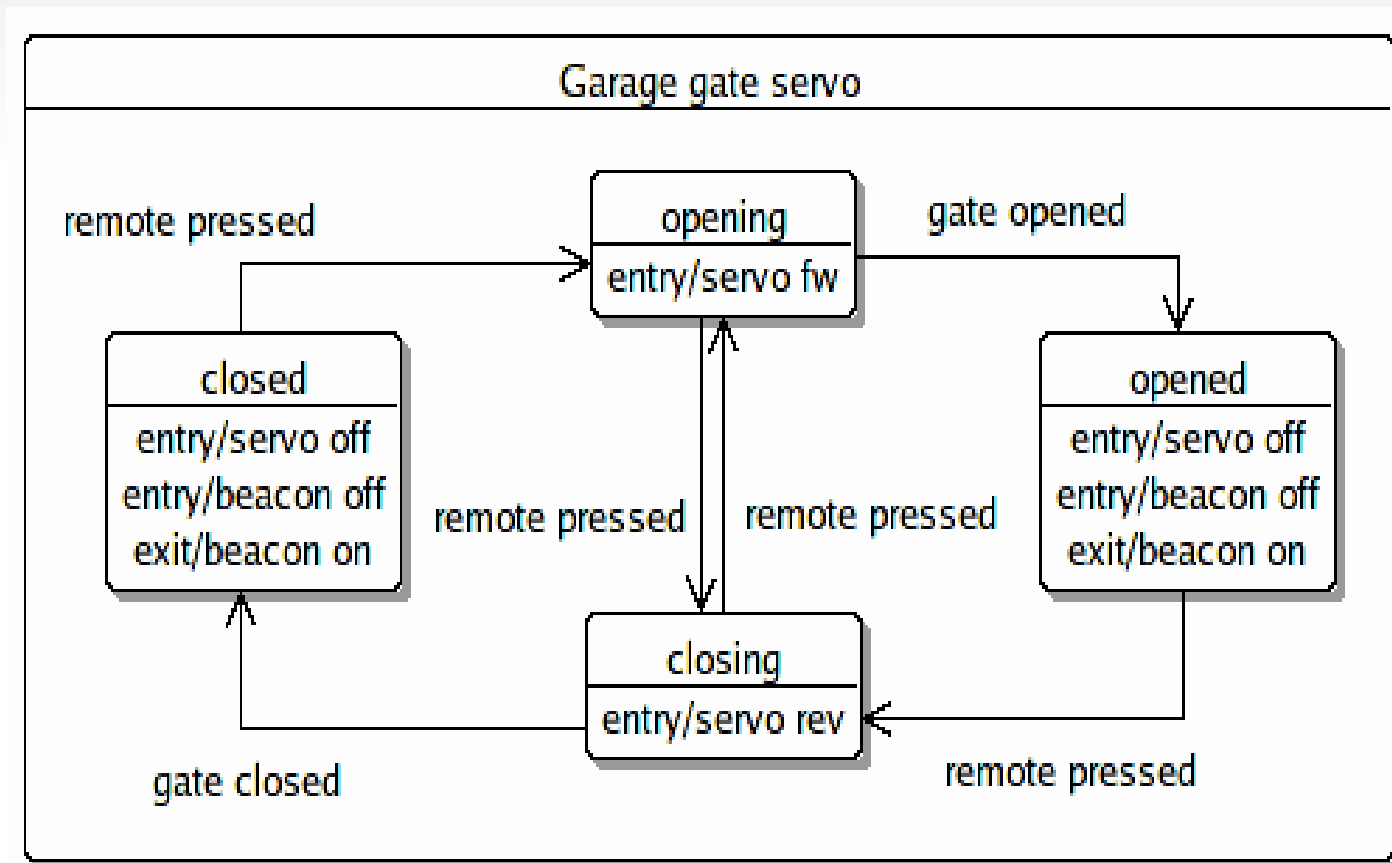
rozdíl mezi těmito diagramy?

- **externí akce** opouští stav a znovu do něj vstupuje  
=> spustí se **entry** 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:

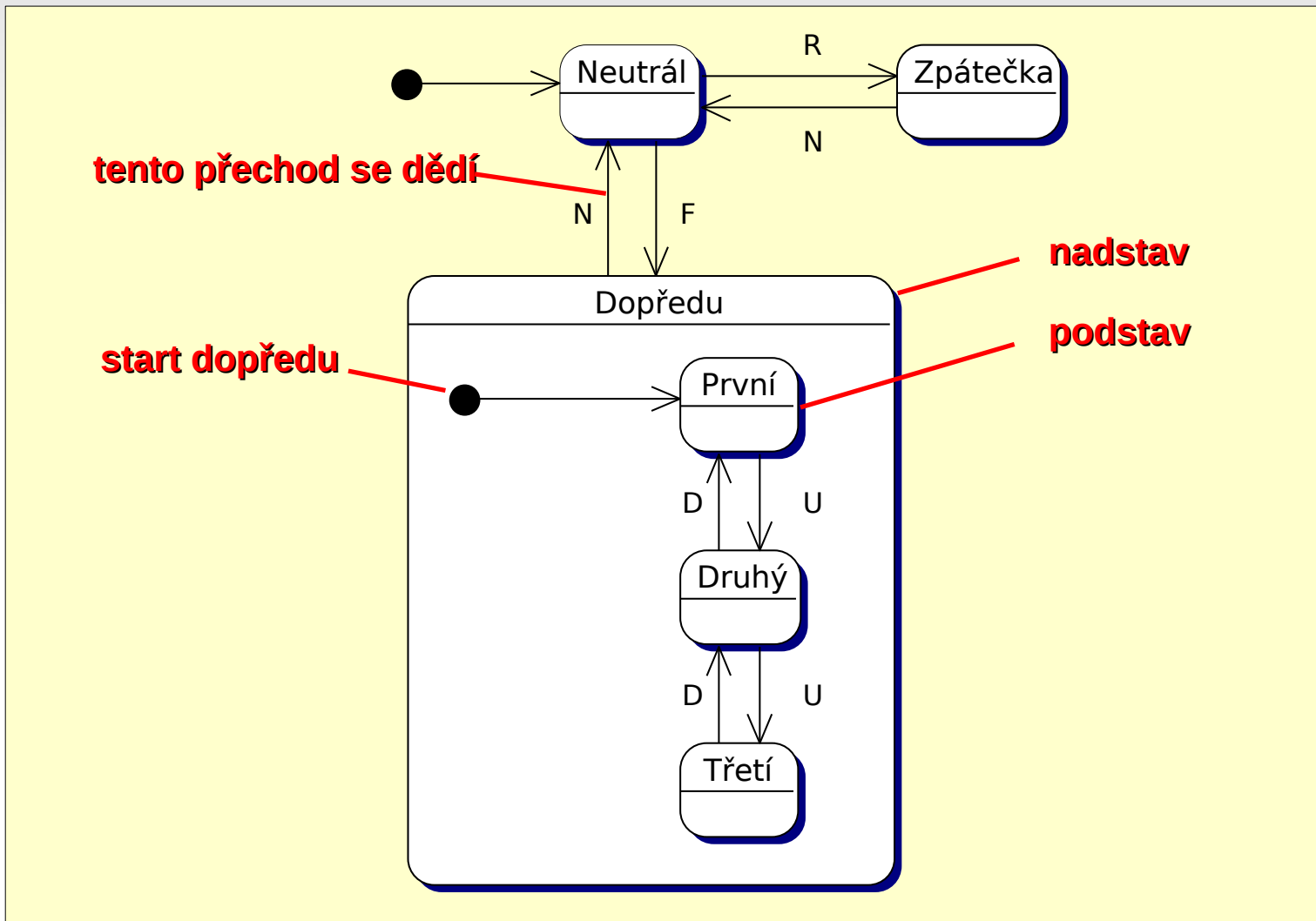
1. akce vstupních přechodů
2. *entry* akce
3. Vnitřní akce a *do* (vnořené STD)
4. *exit* akce
5. akce výstupních přechodů





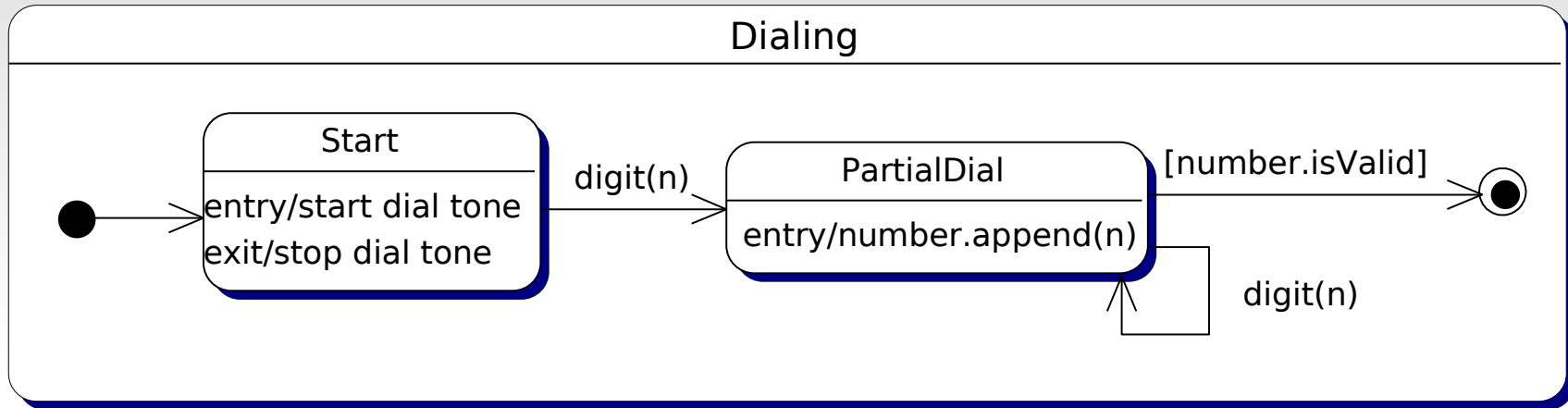
# STD: vnořené stavy = generalizace

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



# STD: vnořené stavy (or-vztah)

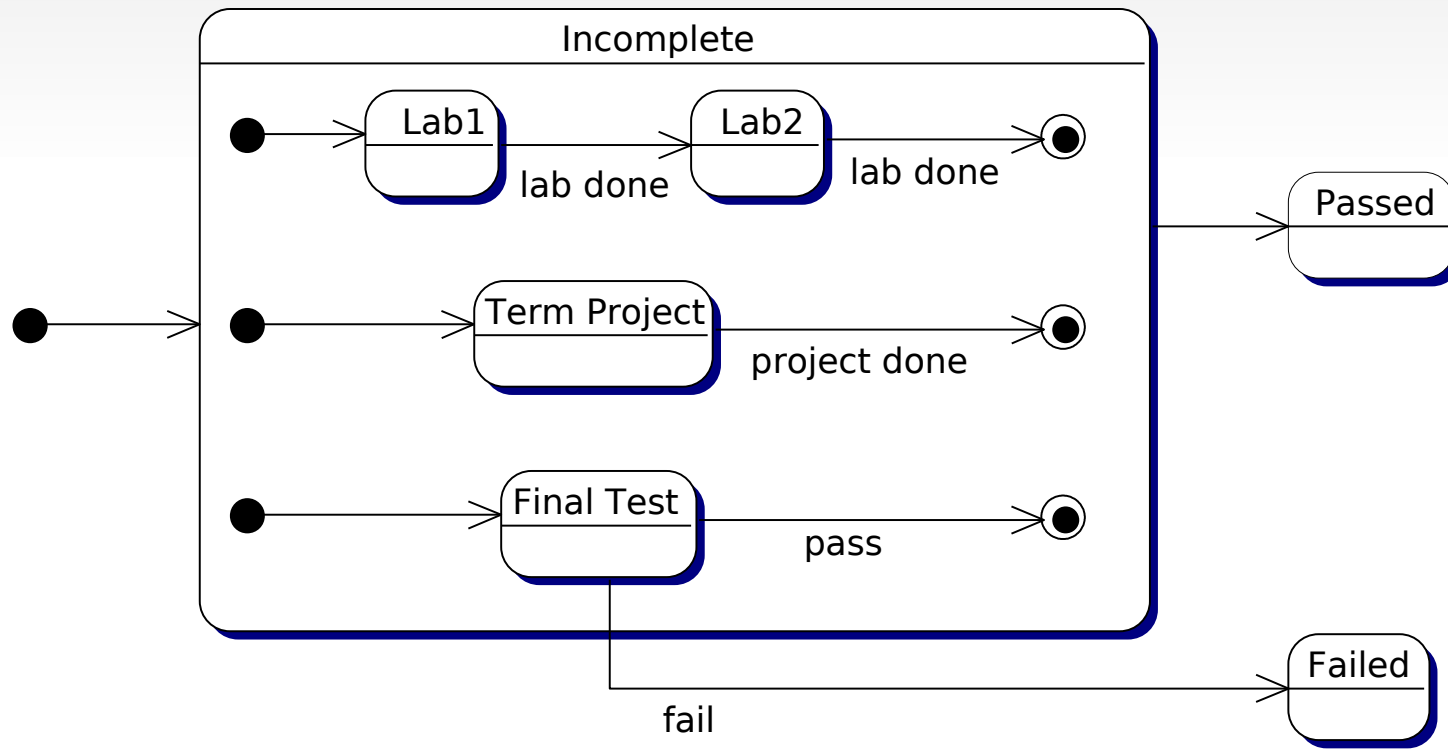
- stav 'Vytáčení' je rozložen s použitím **or-vztahu** na **vzájemně výlučné dílčí stavy**



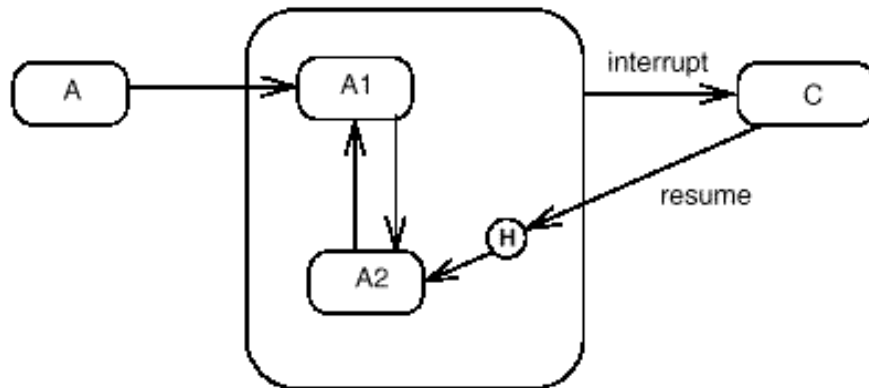
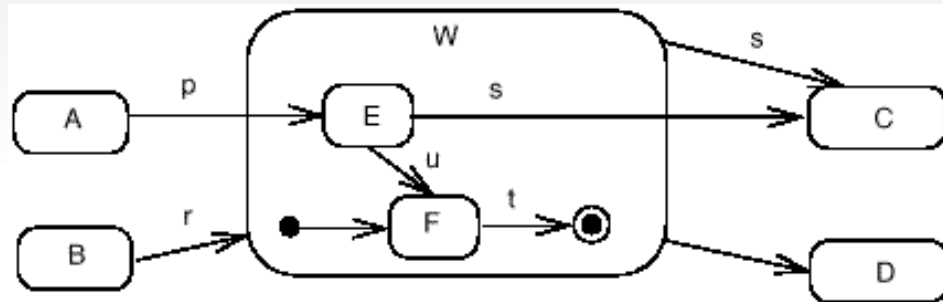
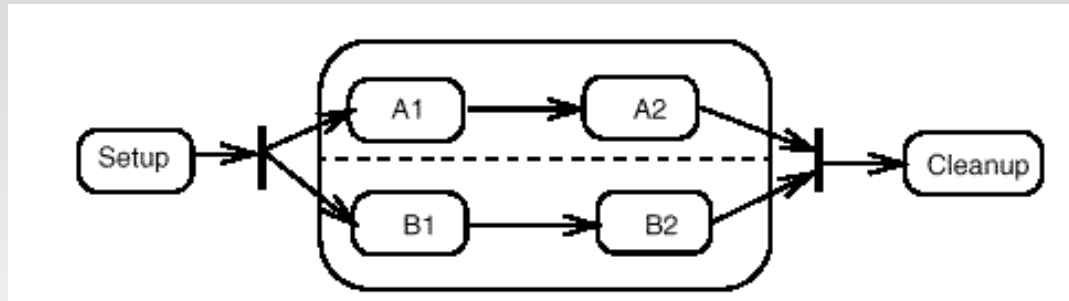
- počáteční stav - koncový stav během 'Vytáčení'
- aktivity dány pomocí vstupní a výstupní akce
- události mohou být „metodami“ určitých tříd
- co popisuje stavový diagram ? metodu, třídu, ...?

# 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

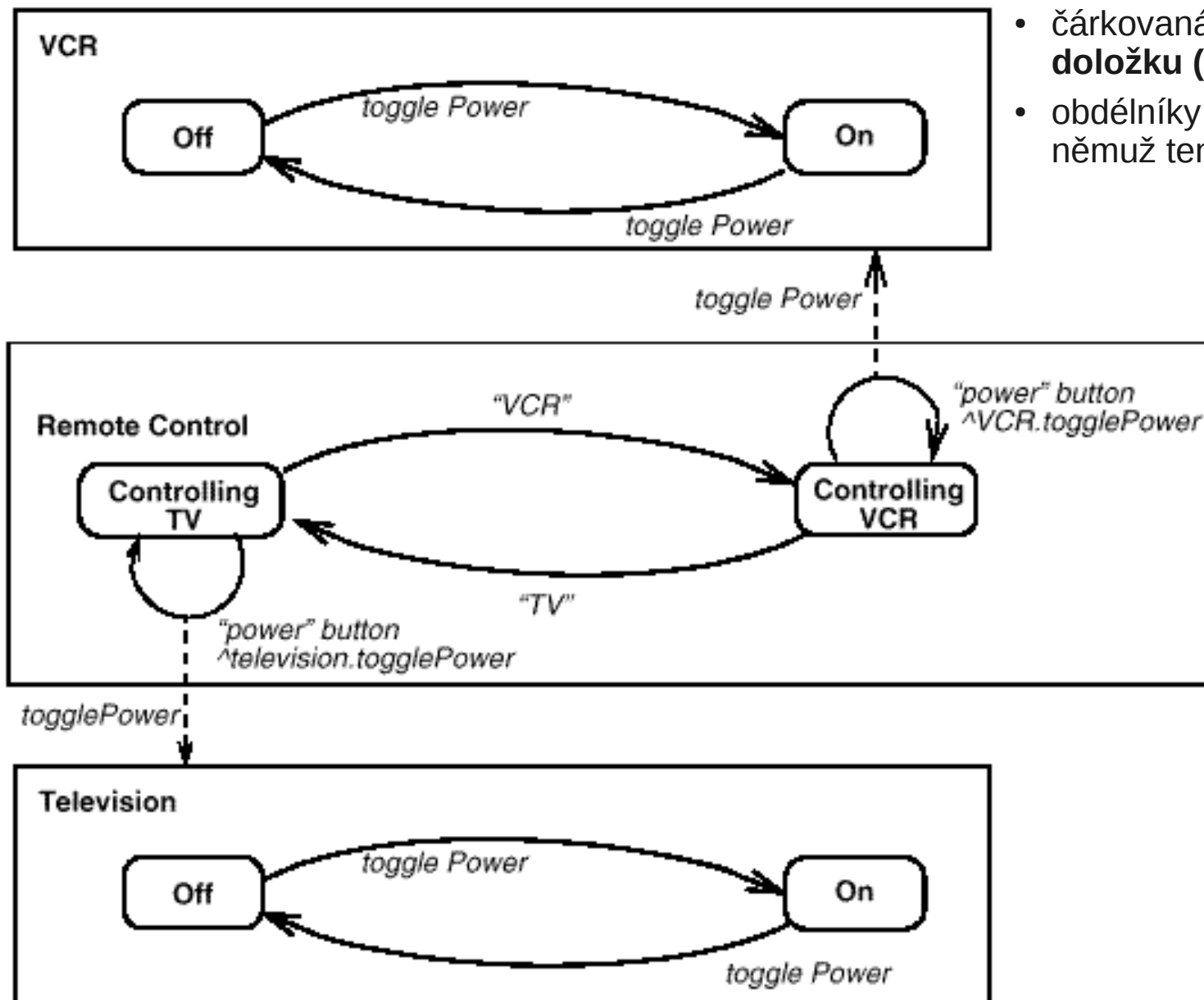


# STD: komplexní přechody



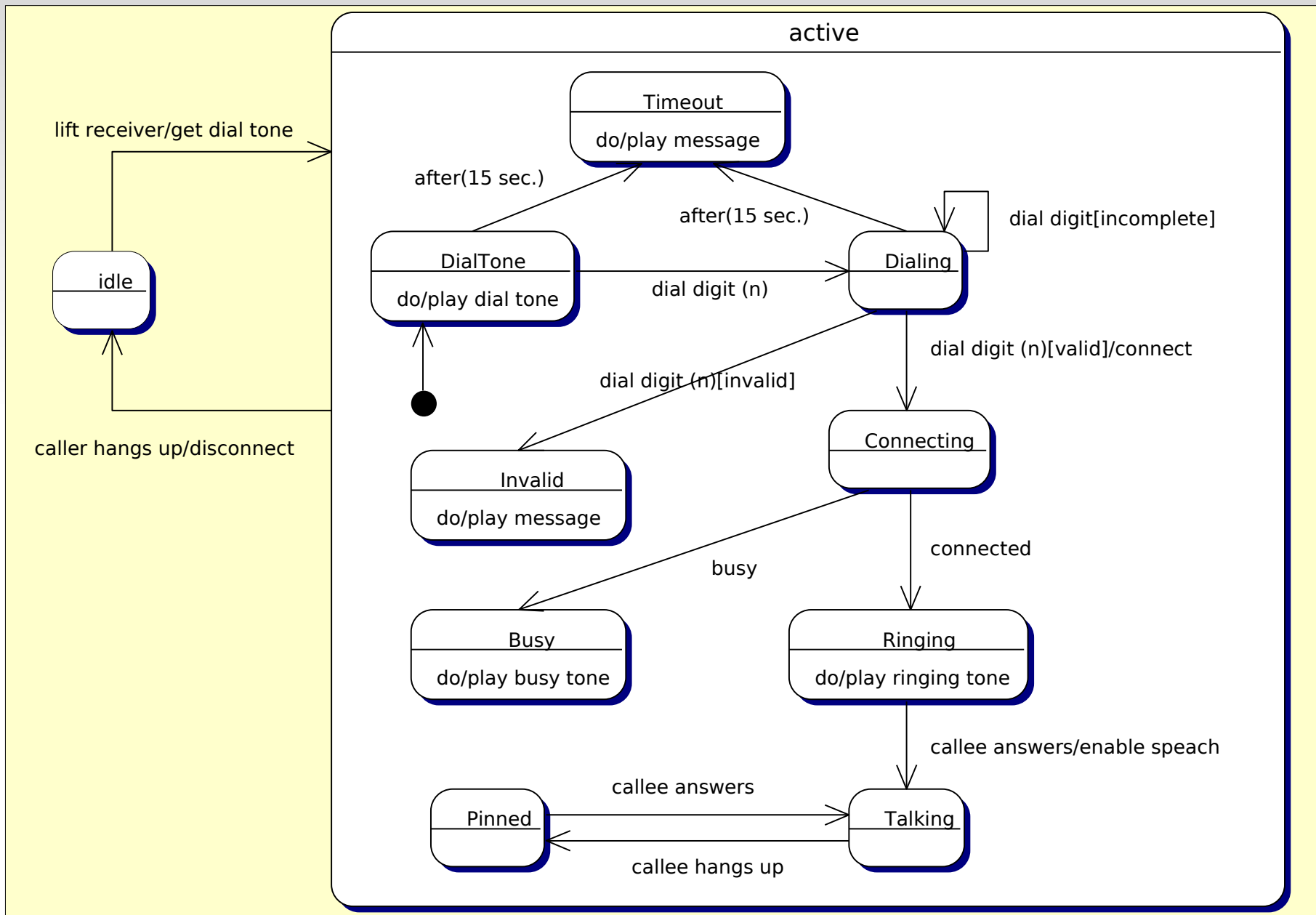
- **komplexní přechody:** všechny události před závorou musí nastat, než dojde k přechodu do stavů za závorou
- **dědění přechodů:** podstavy E a F dědí přechod s událostí 's', podstav E tento zděděný přechod navíc předdefinovává
- **indikátor historie:** A2 je vybráno poprvé, později po *resume* následuje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



# 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?

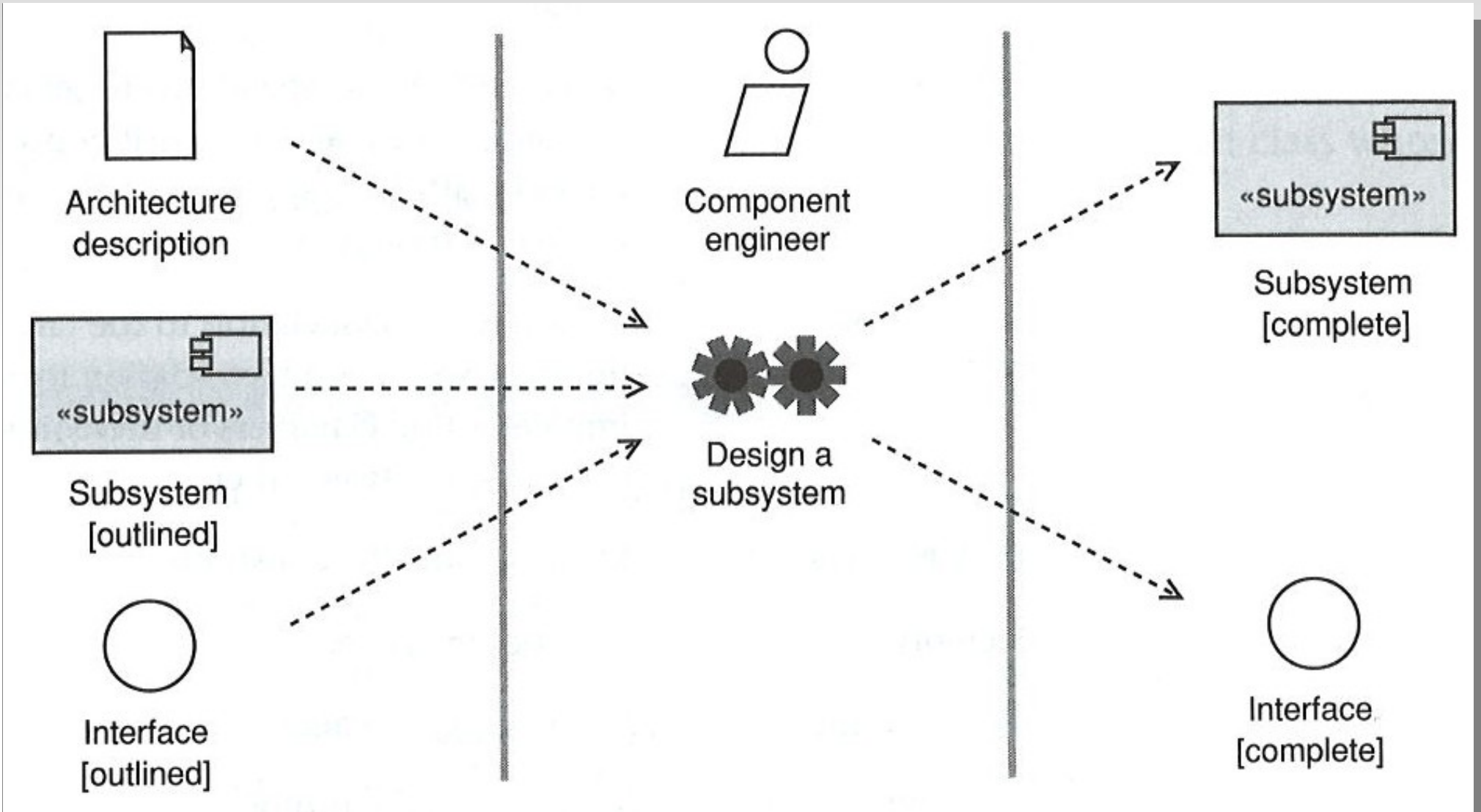


---

# **Rozhraní a komponenty**

## (Interfaces and components)

# UP aktivita: Design a subsystem

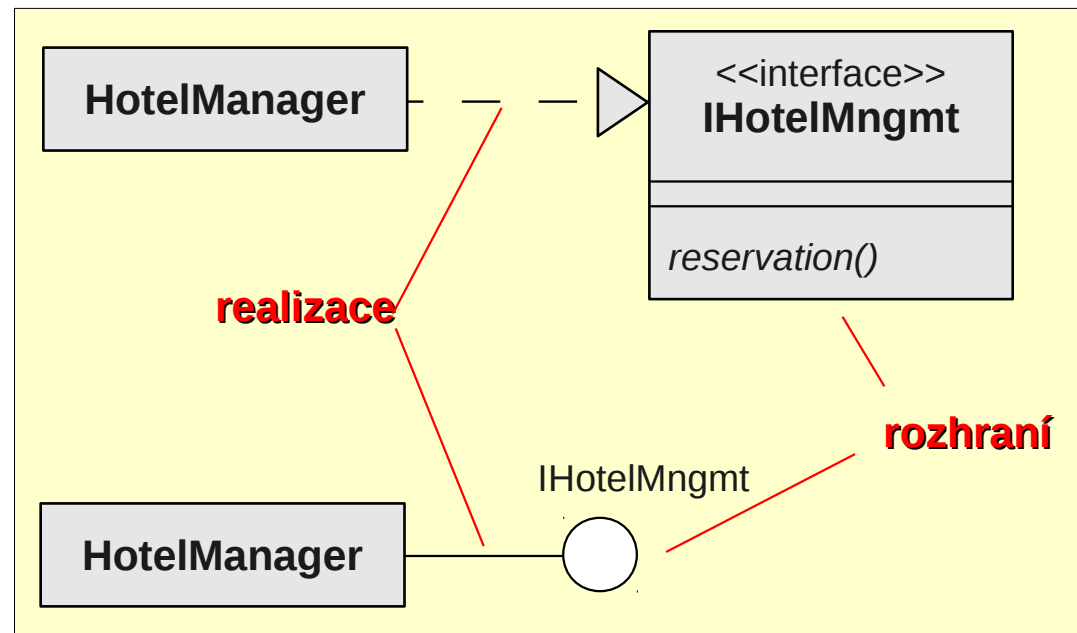


# Rozhraní

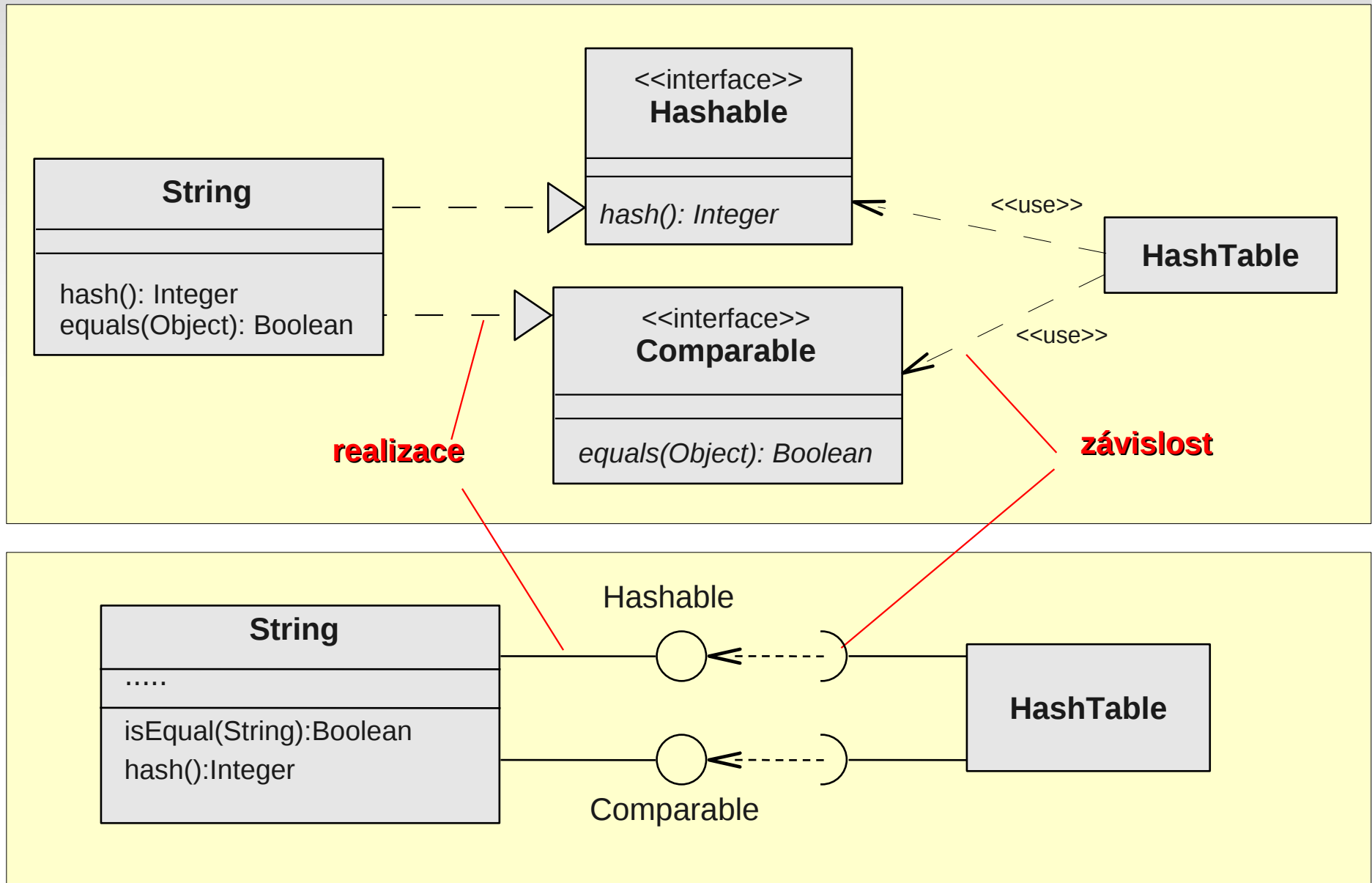
- Angl. *Interface*
- Speciální třídy, které definují *externě viditelné služby* (tzv. kontrakt) nějaké třídy nebo komponenty, bez *specifikace interní struktury* (atributy, stavy, implementace metod).
- Často popisují pouze část, nikoliv celé chování příslušné třídy.
- Používají stejné vztahy jako třídy,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



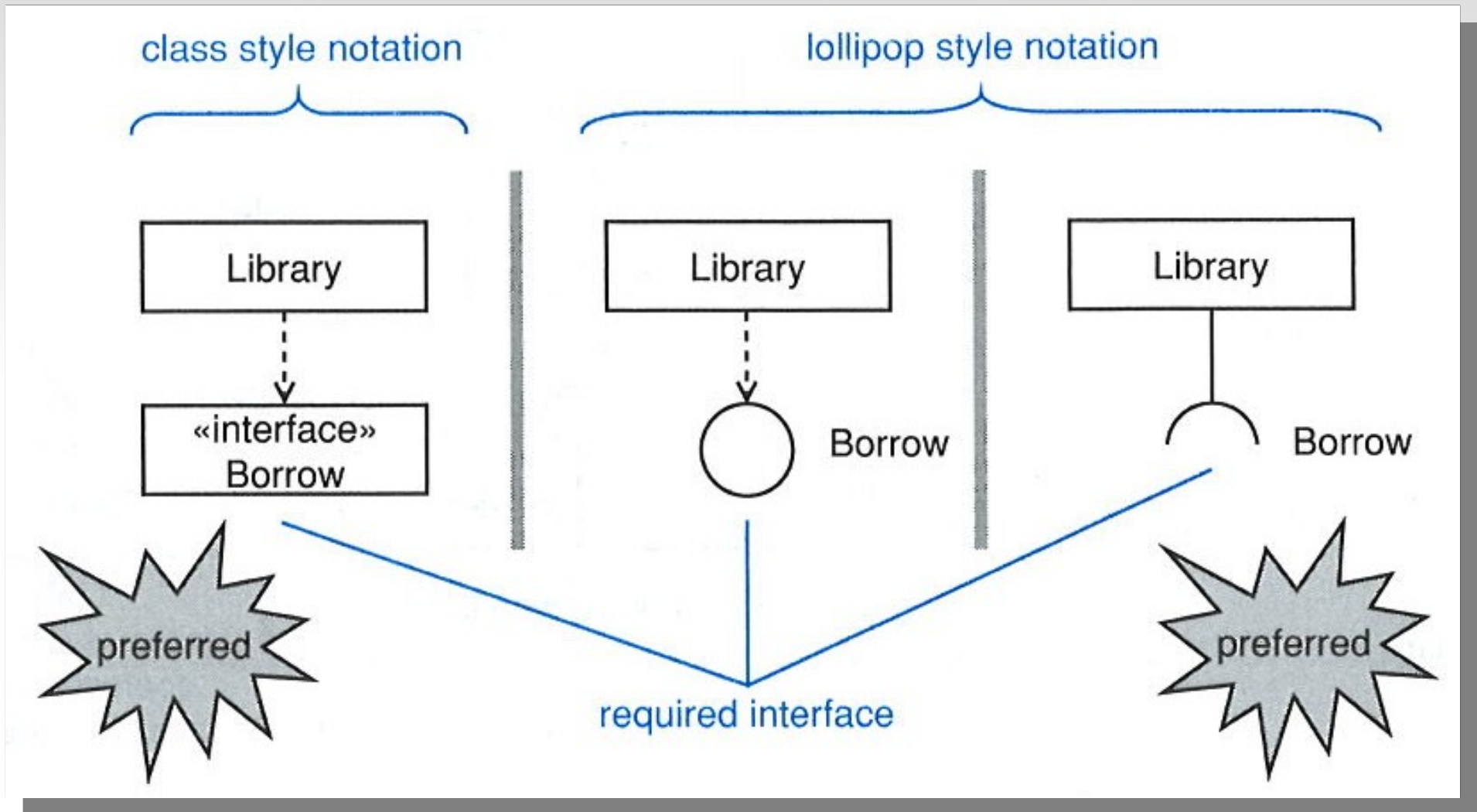
Rozhraní = nabídka služeb realizovaných třídami



# 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**

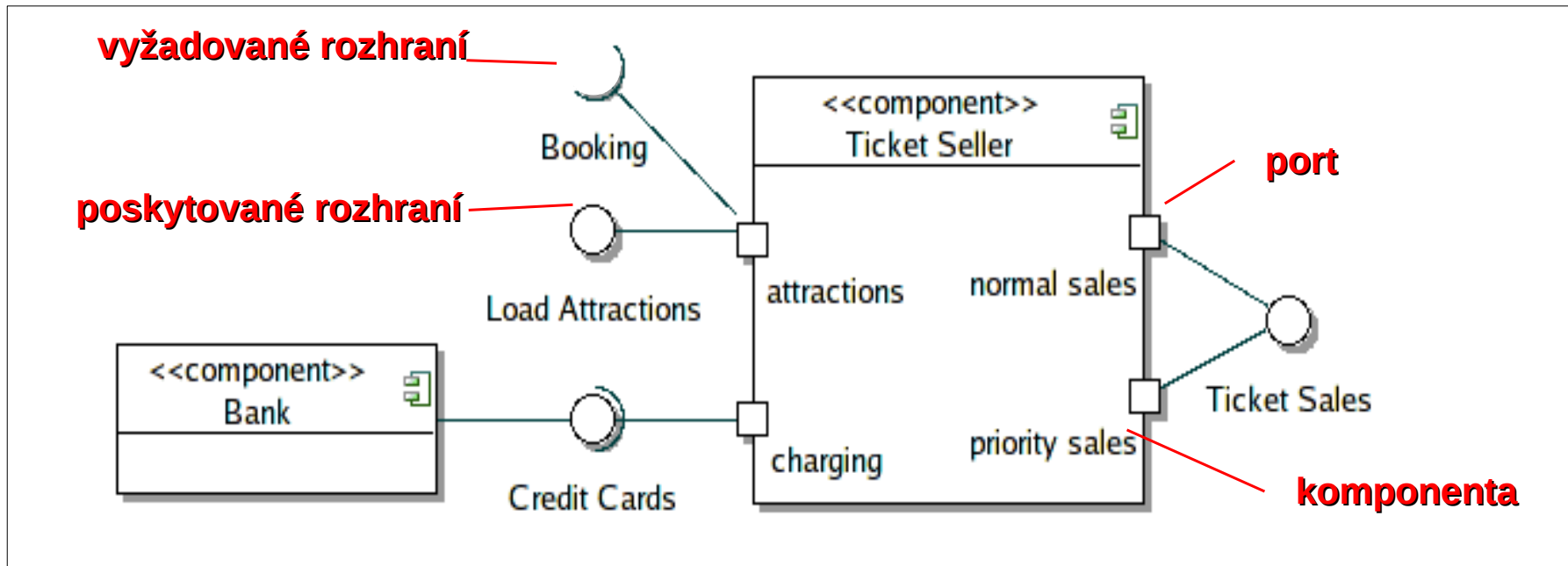
větší celky  
více rozhraní  
poskytují služby  
plně zapouzdřené  
obecně pochopitelné

## **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



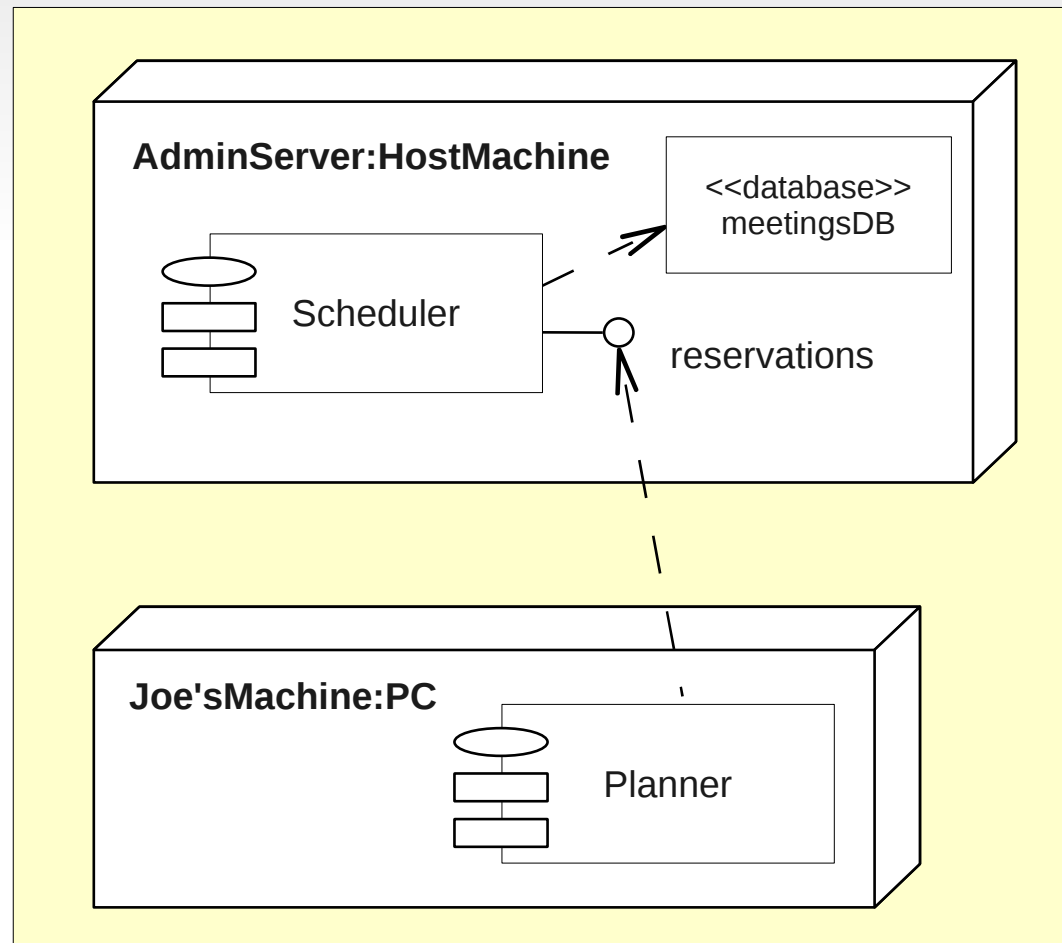
---

# **Diagram rozmístění** *(Deployment Diagram)*



# Diagram rozmístění

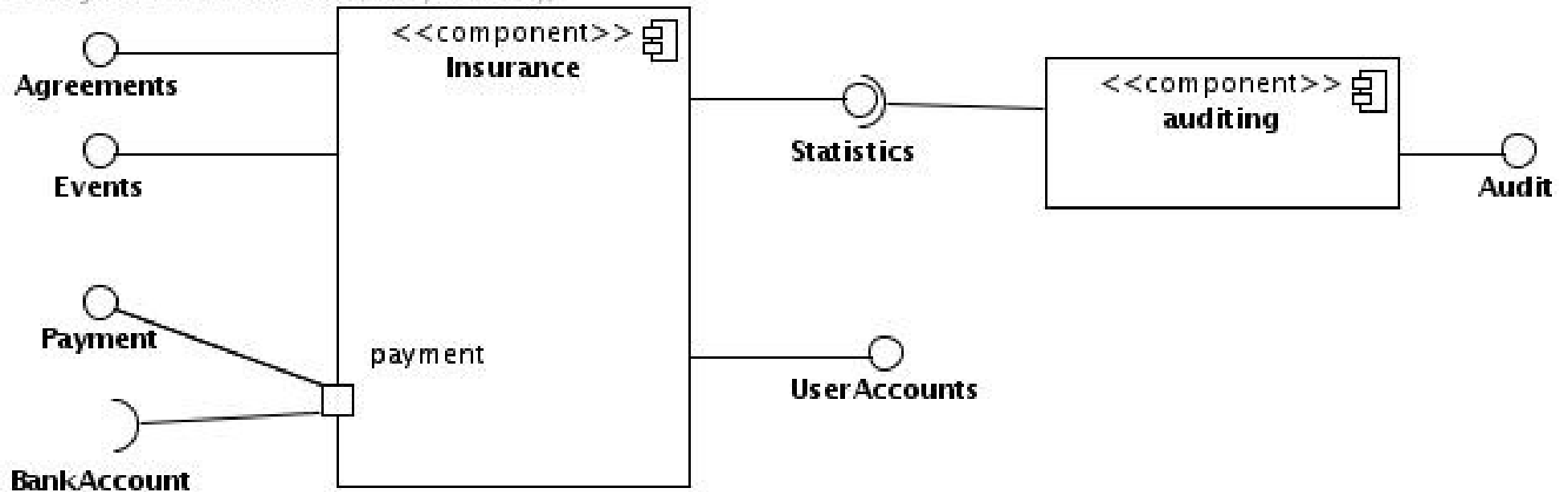
- Angl: *Deployment Diagram*
- Model v etapě návrhu a implementace
- Ukazuje strukturu spustitelného programu
- Fyzické rozmístění systému



# Pojišťovna: Diagram komponent

Demo

Visual Paradigm for UML Standard Edition(Masaryk University)



# Pojišťovna: Diagram rozmístění

Demo

Visual Paradigm for UML Standard Edition(Masaryk University)

