Enterprise Architecture Patterns

Transactions

- System transactions
  - Transakce nad všemi zdroji, nejenom datovými objekty v databázi
  - Např. transakce nad datovými objekty v databázi
  - Systémové transakce dělat krátké (pro jeden request)
  - Transakce přes více requestů jsou dlouhé
- Business transaction
  - Typicky se skládá z řady systémových transakcí
  - Nedá se nahradit systémovou transakcí, protože systémové transakce nemají být dlouhé

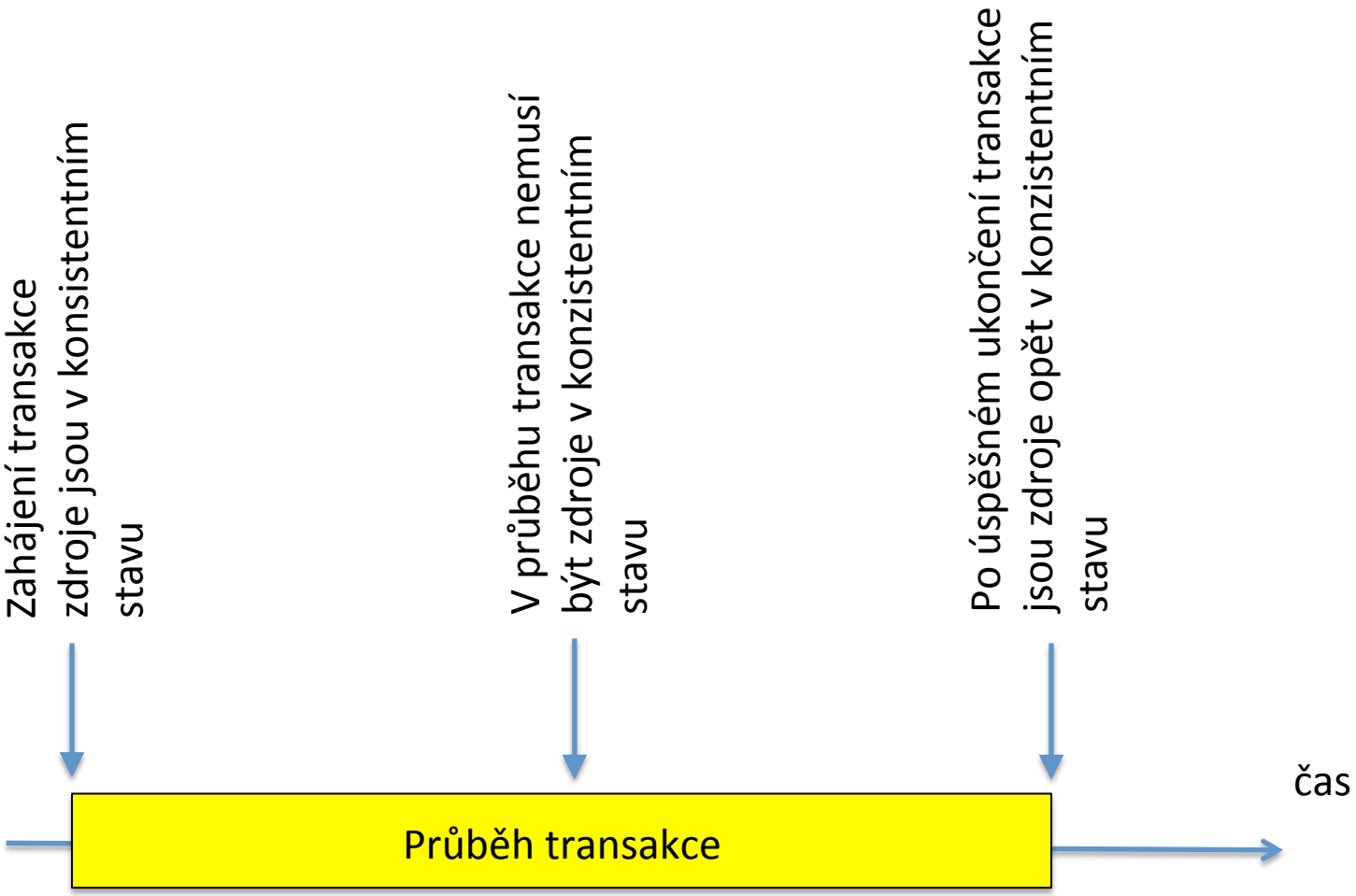
Offline Concurrency

- Zodpovědnost za dodržení ACID vlastností business transakce mezi systémovými transakcemi je ponechána na programátorovi (enterprise) aplikace

Transactions

- ACID
  - Atomicity (atomičnost)
    - Transakce může proběhnout buď celá, nebo vůbec
    - Ukončení transakce operacemi commit/rollback
  - Consistency (konsistence)
    - Transakce provádí sama o sobě správný výpočet
    - Neporušuje konsistency zdrojů (dat)
  - Isolation (izolovanost)
    - Paralelně probíhající transakce si vzájemně neškodí
    - Různé stupně izolovanosti transakcí
  - Durability (trvalost)
    - Výsledky transakce, která byla ukončena operací commit jsou trvalé (i po havárii systému)

Atomičnost transakce



Izolovanost transakcí

- Problémy
  - Dirty read
    - T1:write(A), T2:read(A), …. T1:commit/rollback
  - Lost update
    - T1:write(A), T2:write(A), …, T1:commit
  - Unrepeatable read
    - T1:read(A), T2:write(A), …
  - Phantom
    - V průběhu T1 jiná transakce T2 vytvoří zdroj, který, kdyby existoval při zahájení T1, T1 by s nm pracovala

Izolovanost transakcí

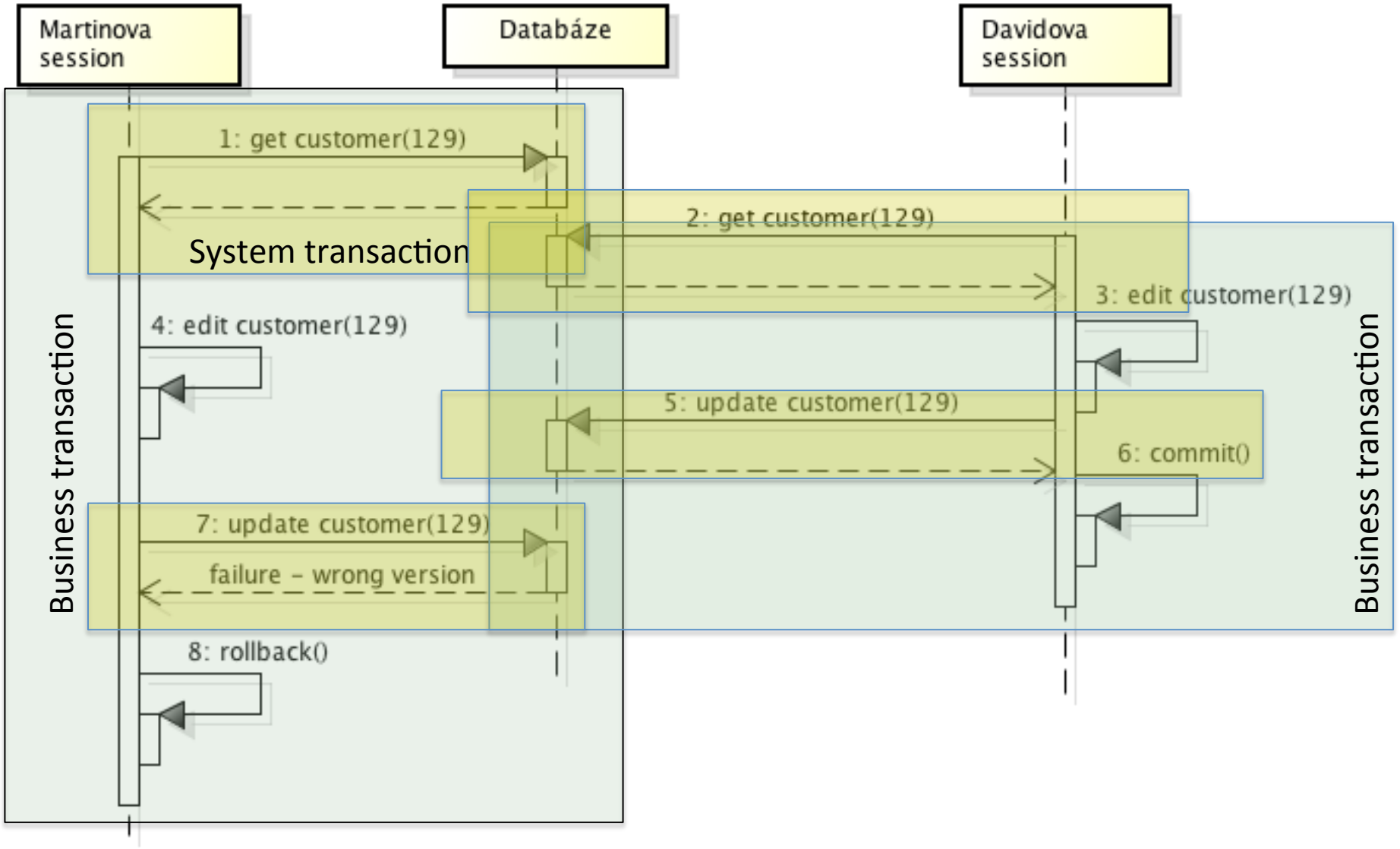
- Stupně izolovanosti transakcí

	Dirty Read	Urepeatable Read	Phantom
Read Uncommited	Ano	Ano	Ano
Read Committed	Ne	Ano	Ano
Repeatable Read	Ne	Ne	Ano
Serializable	Ne	Ne	Ne

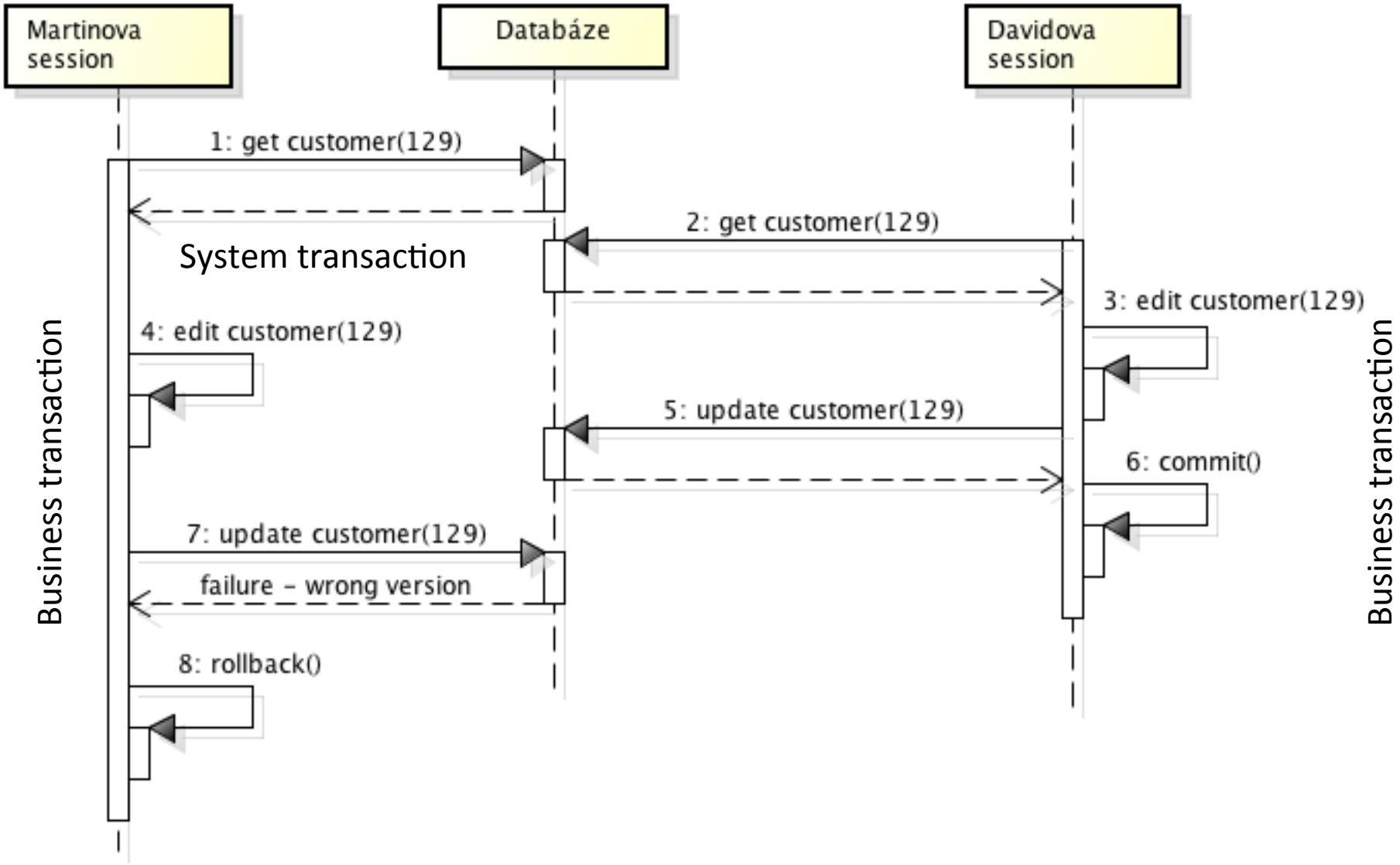
Optimistic versus pessimistic offline concurrency control

- Optimistic:
  - Pravděpodobnost konfliktu je malá
  - Předpokládáme, že konflikt nenastane
    - Neděláme prevenci konfliktu
    - Konflikt řešíme, až když nastane – ošetření výjimky
    - Nejobvyklejší implementace – každý zdroj má přiřazeno číslo verze
    - Obdoba řízení současného přístupu ke zdrojovým kódům systémy typu csv/svn.

Optimistic offline concurrency control



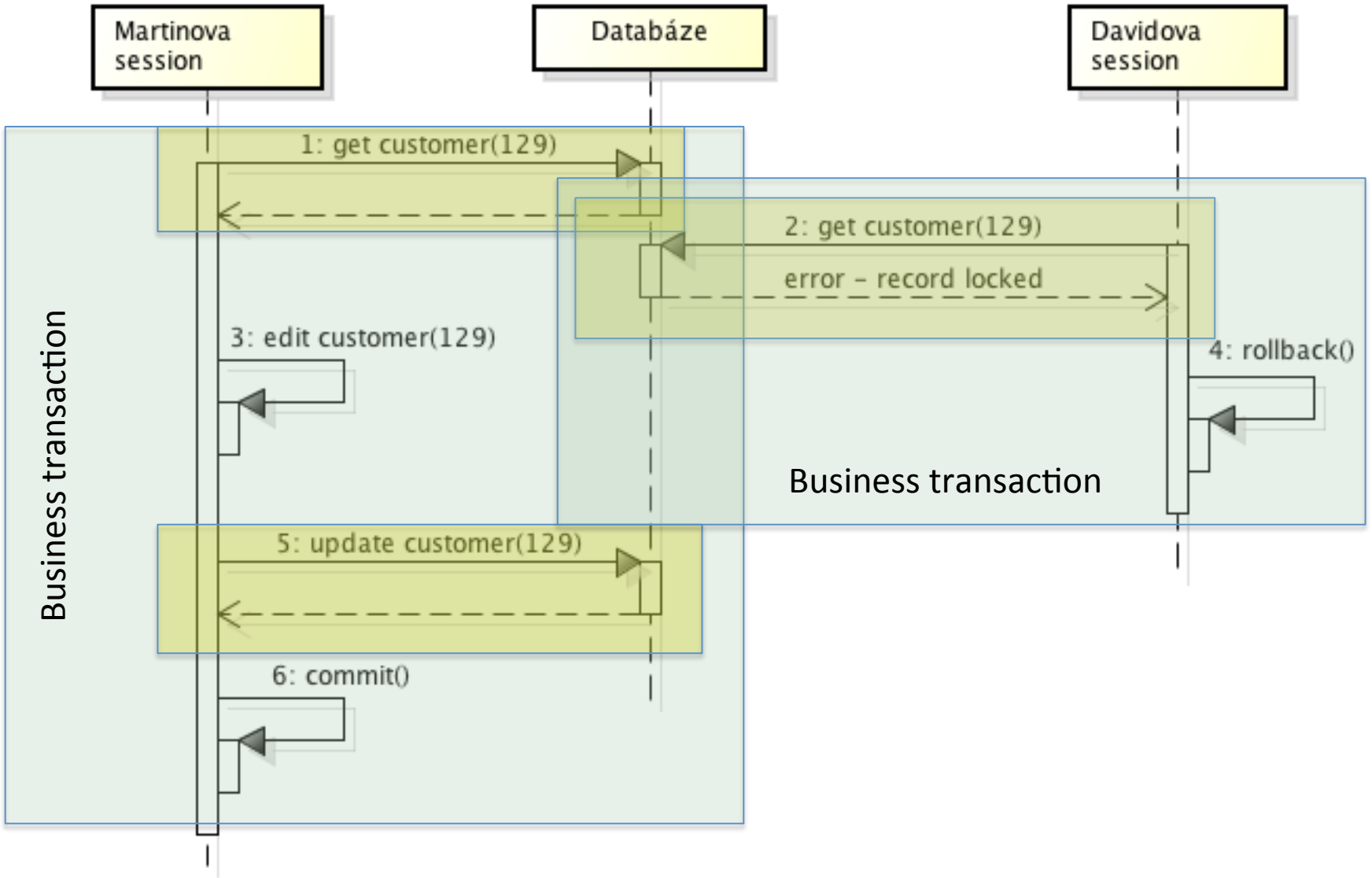
Optimistic offline concurrency control



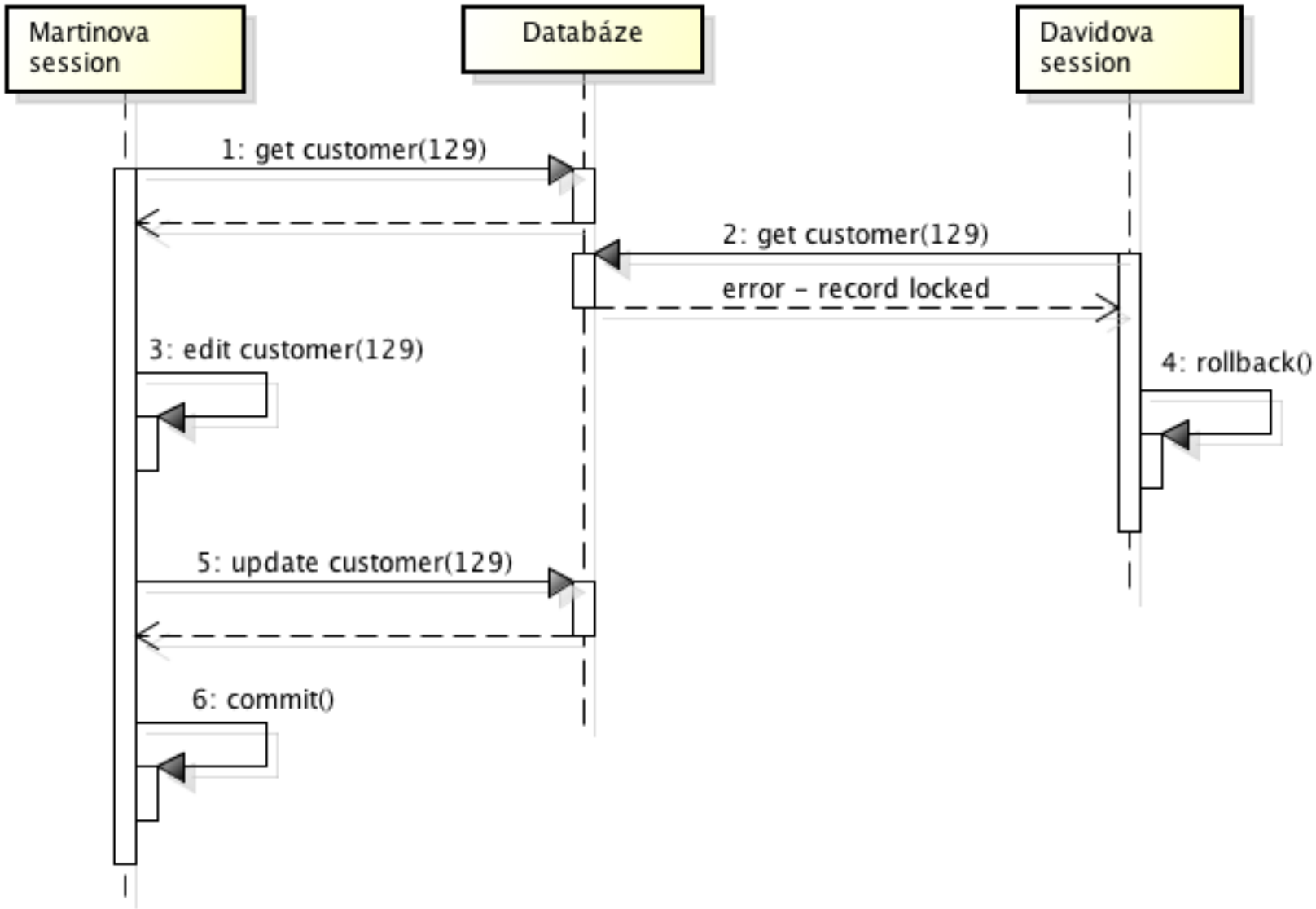
Optimistic versus pessimistic offline concurrency control

- Optimistic
- Pessimistic:
  - Pravděpodobnost konfliktu je velká
    - Preventivně bráníme vzniku konfliktu
    - Nejobvyklejší implementace – zamykání zdrojů
    - Business transakce musí získat zámek zdroje předtím, než se zdrojem začne pracovat
  - Nebo uživatel nesmí přijít ani o část své práce

Pessimistic offline concurrency control



Pessimistic offline concurrency control



Optimistic versus pessimistic concurrency control

- Pessimistic:
  - Lock manager
    - Dobré, pokud je součástí doménového modelu
    - V (i) paměti nebo v (ii) databázi drží seznam zamčených objektů. Pokud aplikace běží na clusteru serverů, musí být tento seznam uložen v databázi.
    - Sdílené (shared locks) zámky pro čtení, výhradní (exclusive locks) zámky pro zápis.
    - Kompatibilita zámků: SH x SH ano, SH x EX ne, EX x EX ne
    - Business transakce nemanipuluje se zámky přímo, ale zásadně prostřednictvím lock managera, který je jejich vlastníkem

Optimistic versus pessimistic concurrency control

- Pessimistic:
  - Lock manager
  - Protocol lock managera
    (i) kdy zamknout – jde-li to, pak dříve než zdroj získám (mám jistotu, že pracuji s nejaktuálnější verzí zdroje). Úplně nejlepší je zamknout všechny zdroje dříve, než s nimi uživatel začne pracovat, nemá-li to vliv na omezení průchodnosti systému.
    (ii) co zamknout – typicky ID zdroje (znám ho, podle něj vyhledávám),
    (iii) kdy odemknout – nejlépe na konci business transakce
    (iv) co dělat, když nelze zamknout – nejjednodušší je transakci zrušit (rollback)

Optimistic versus pessimistic concurrency control

- Pessimistic:
  - Lock manager
  - Protocol lock managera
  - Tabulka zámků spravovaná lock managerem
    - Serializovaný přístup
      - Tabulka zámků v paměti – serializace na úrovni progr. jazyka
      - Tabulka zámků v databázi – serializace pomocí systémové transakce na stupni izolovanosti SERIALIZABLE (současné inserty a ready)

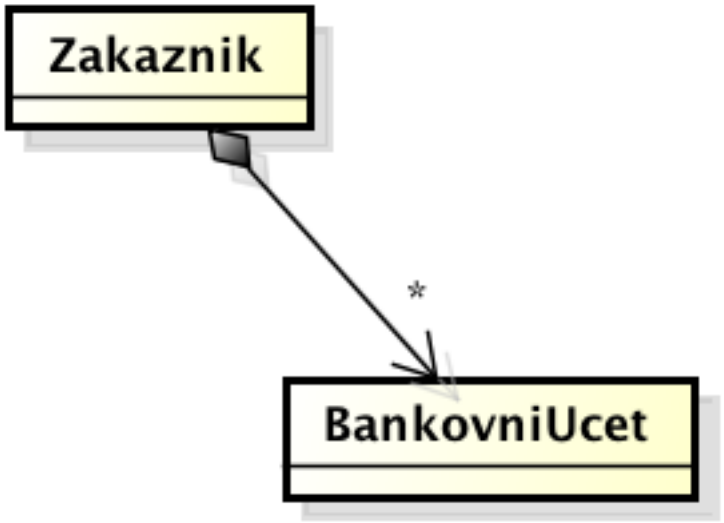
Optimistic versus pessimistic concurrency control

- Pessimistic:
  - Lock manager
  - Protocol lock managera
  - Tabulka zámků spravovaná lock managerem
  - Nebezpečí deadlocku
    - Nečekat na zámek, raději vyhodit výjimku, pokud se nepodaří zámek získat hned.
    - Pozor EJB transkace – čekají na zámek!

Optimistic versus pessimistic concurrency control

- Pessimistic:
  - Lock manager
  - Protocol lock managera
  - Tabulka zámků spravovaná lock managerem
  - Nebezpečí deadlocku
  - Lost transactions
    - Klientský proces “spadne” uprostřed transakce
    - U webové aplikace typicky uživatel transakci nedokončí
    - Transakce není schopna uvolnit zámky
    - Timeout na úrovni aplikace nebo lépe na úrovni aplikačního serveru (typicky timeout http session) – po timeoutu se uvolní (nebo zinvalidní) zámky

Coarse-grained Lock

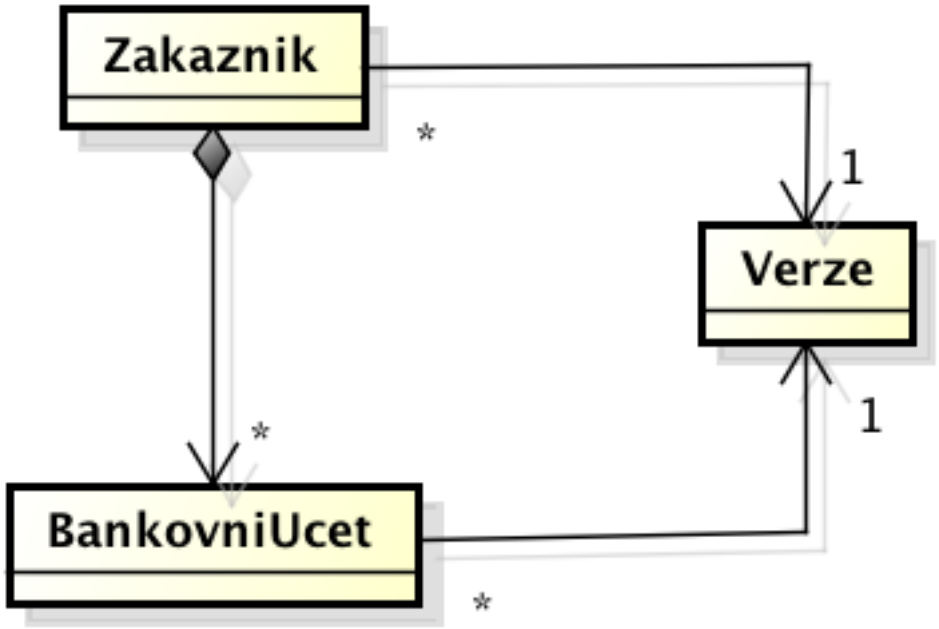


Zamykání skupin objektů jedním zámkem

Výhody oproti zamykání jednotlivých objektů:

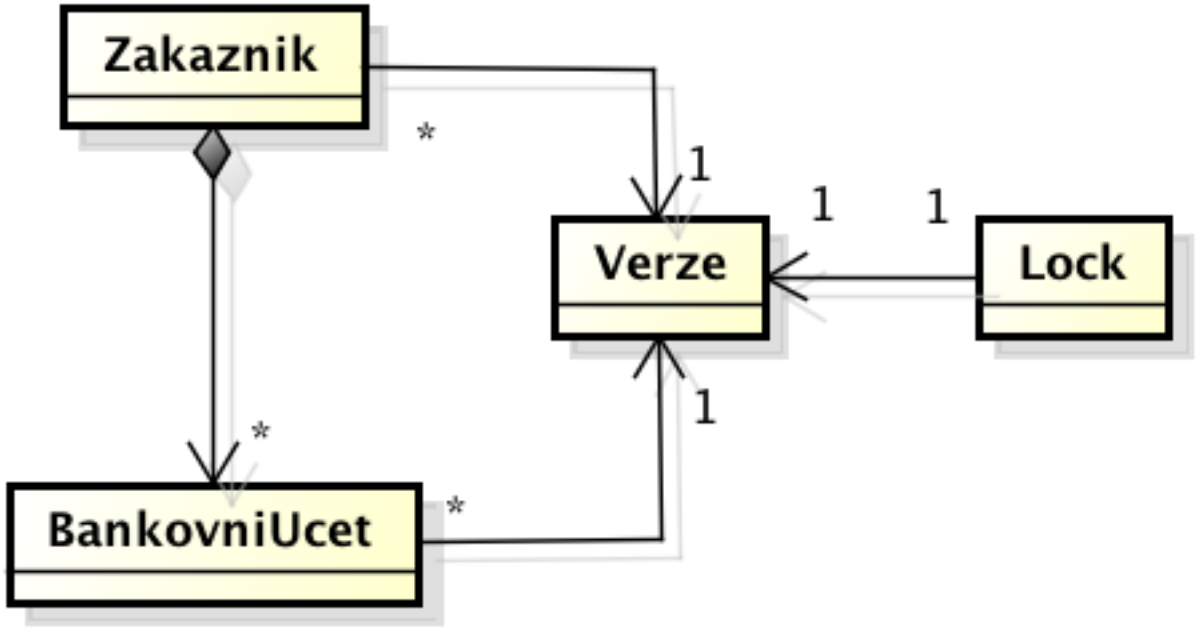
- Optimistické zamykání – pro zamčení (opatření verzí) je třeba načíst velké množství objektů
- Pesimistické zamykání – rozsáhlá tabulka zámků – sériový přístup – časově dlouho trvající transakce při velkém množství obvjektů

Coarse-grained Lock



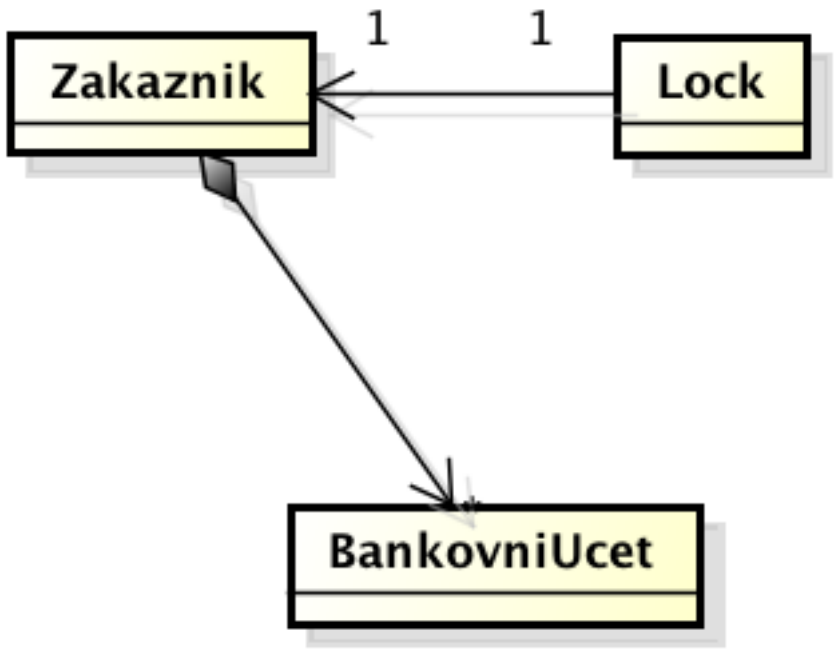
Optimistické zamykání – sdílený objekt verze

Coarse-grained Lock



Pesimistické zamykání – zamykání sdíleného objektu verze

Coarse-grained Lock



Pesimistické zamykání – zamykání kořene

Business transakce

- Neimplementuj svůj vlastní lock manager nebo transakční monitor
- Pochop dobře, jak funguje transakční mechanismus Tvého aplikačního serveru

Client Session State

- Pokud je server zcela bezestavový (stateless), musí se stav sešny přenášet mezi klientem a serverem – může jít o velký objem dat
- Pro tenkého webového klienta – 3 možnosti
  - **Parametry v URL** – negativa: (i) omezená délka URL, (ii) parametry zobrazeny v URL stránky, (iii) problémy s bookmarkováním stránky
  - **Skrytá vstupní pole** (hidden fields) – skryta ve smyslu, že nejsou na stránce zobrazena
  - **Cookies** – problémy: (i) uživatel může cookies zakázat, (ii) cookies organizovány po doménách – co když více aplikací v doméně

Client Session State

- Udržování stavu na klientovi může být výhodou při clusteringu, failover
- Obsahuje-li stav citlivá data, měl by být šifrován => režie
- Server by měl příchozí data revalidovat, aby nebyla poškozena jejich konzistence
- Stav na klientovi v minimální variantě obsahuje sessionId – odkazující na session, která drží stav na serveru
- Session steeling – uživatel modifikuje sessionID tak, aby získal session někoho jiného

Server Session State

- Stav “sešny” typicky reprezentován
  - Binárně (BLOB – Binary Large OBject) – problém s verzováním
  - Textově (typicky XML)
- Udržován
  - Lokálně
    - v paměti aplikačního serveru
      - HashMap, jejímž klíčem je sessionID
    - Ve filesystému aplikačního serveru
    - V lokální DB aplikačního serveru
    - Problém v případě clusteru aplikačních serverů, při failover/switchover
  - Ve sdílené databázi
    - Stav uložen nestrukturovaně (např. jako BLOB) – v opačném případě viz Database Session State
    - Umožňuje clusterování, failover/switchover
    - Musí se řešit zapomínání stavu ukončených a “vyprchaných” sessions

Database Session State

- Speciální případ předchozího “server session state” – stav uložen jako strukturovaná data
- Dva případy
  - V “ostrých” tabulkách
    - Přidat sloupec SessionID, if null => ostrá data, if not null => pracovní data rozpracované sešny
    - Rozpracovaná (pending) data nemusí být konzistentní => nemusí být na ně aplikovatelná integritní omezení
  - V “pending” tabulkách
- Mazat data nedokončených (abandoned) a přerušených (cancelled) sessions

Gateway

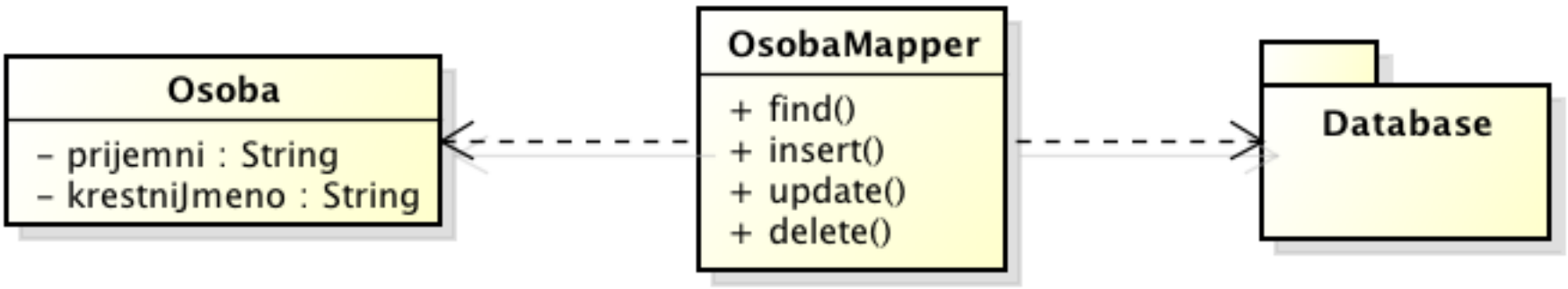
- Jednotlivé vrstvy mívají specifická API (JDBC/SQL pro relační DB, W3C/DOM pro XML) poplatná dané technologii
- Cílem vzoru Gateway je izolovat klienta této vrstvy od "technologických" API
- Zdrojový kód Gateway lze často vygenerovat (např. JAXP)
- Gateway je typicky tvořena jednou třídou, ale nemusí tomu tak být nutně
- Pro vývoj je vhodné, aby Gateway byla interface se 2 implementacemi:
  - "ostrá" implementace volá reálnou službu dané vrstvy,
  - "testovací" implementace volá tzv. "Service Stub"

Service Stub

- Vyvíjíme SW, který využívá nějakou službu třetí strany
- Zpomalení vývojového procesu
  - Nemůžeme monitorovat komunikaci na straně vzdálené služby
  - Zpřístupnění služby může vyžadovat složitý protokol
  - Nemusíme být schopni rozhodnout, kde je problém – u nás nebo u “nich”
- Pro testování použijeme Service Stub, který běží lokálně a simuluje onu službu třetí strany

Mapper

- Vrstva izolující dva subsystémy
- Typický případ “Data Mapper”



Layer Supertype

- Pro všechny objekty (téhož druhu) ve vrstvě vytvoř supertype (superclass)
- Supertype poskytuje
  - Správu ID (při automatickém generování)
  - Persistenci
  - Metody
    - equals()
    - hash()

Organizace doménové logiky

- Transaction script
- Domain Model (preference)
- Table Module

Organizace doménové logiky

- Transaction script
  - Business objekt obsahuje spíše data než logiku
  - Logika centralizována do jedné/několika tříd
  - Transakčnímu skriptu se poskytnou business objekty ke zpracování
  - Při rozšiřování funkcionality se musí transakční skript modifikovat

Organizace doménové logiky

- Transaction script
- Domain Model
  - Business objekty obsahují data i logiku
  - Doménová logika rozptýlena mezi business objekty
  - Přehlednější z hlediska objektového návrhu
  - Při rozšiřování funkcionality se vytvoří (odvodí) nový typ business objektu, který poskytuje příslušnou funkcionalitu

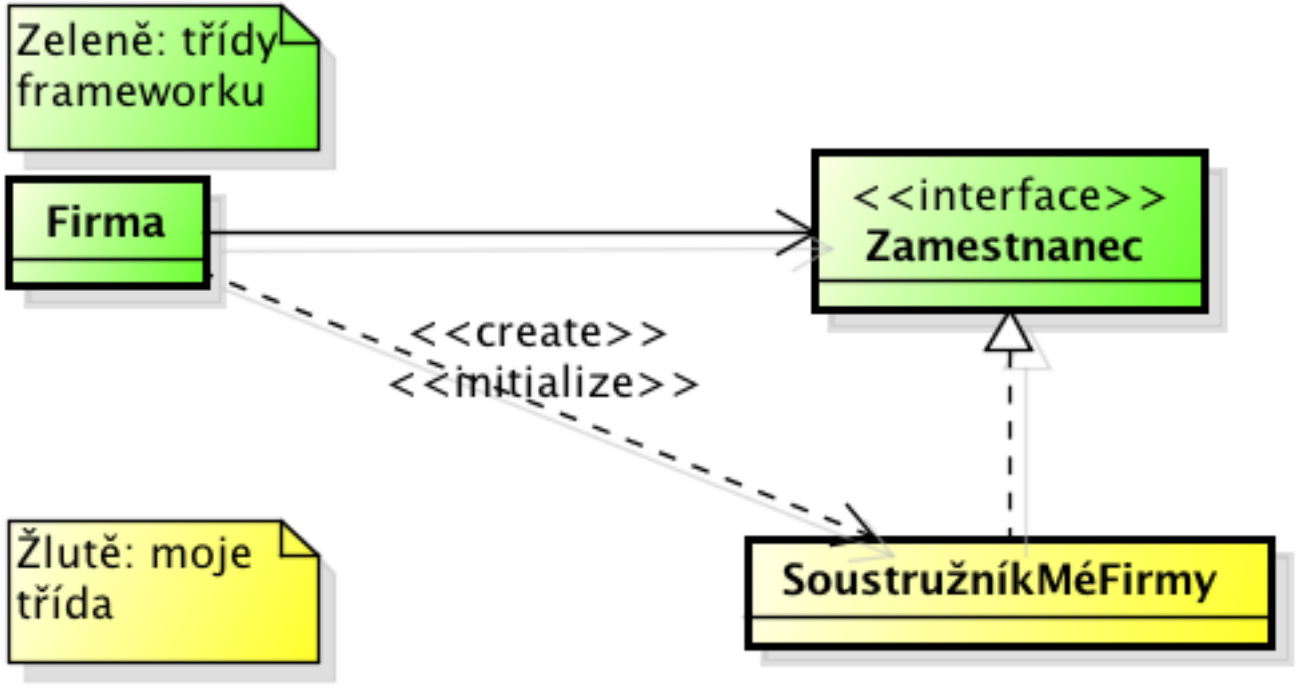
Organizace doménové logiky

- Transaction script
- Domain Model
- Table Module
  - Organizace doménové logiky poplatná struktuře databáze
  - Logika soustředěna v singletonech per table
  - Výsledkem zpracování není business objekt, ale result set
  - Může být výhodné pro některé frameworky pracující s result sety ???
  - Neodpovídá příliš zásadám objektového návrhu

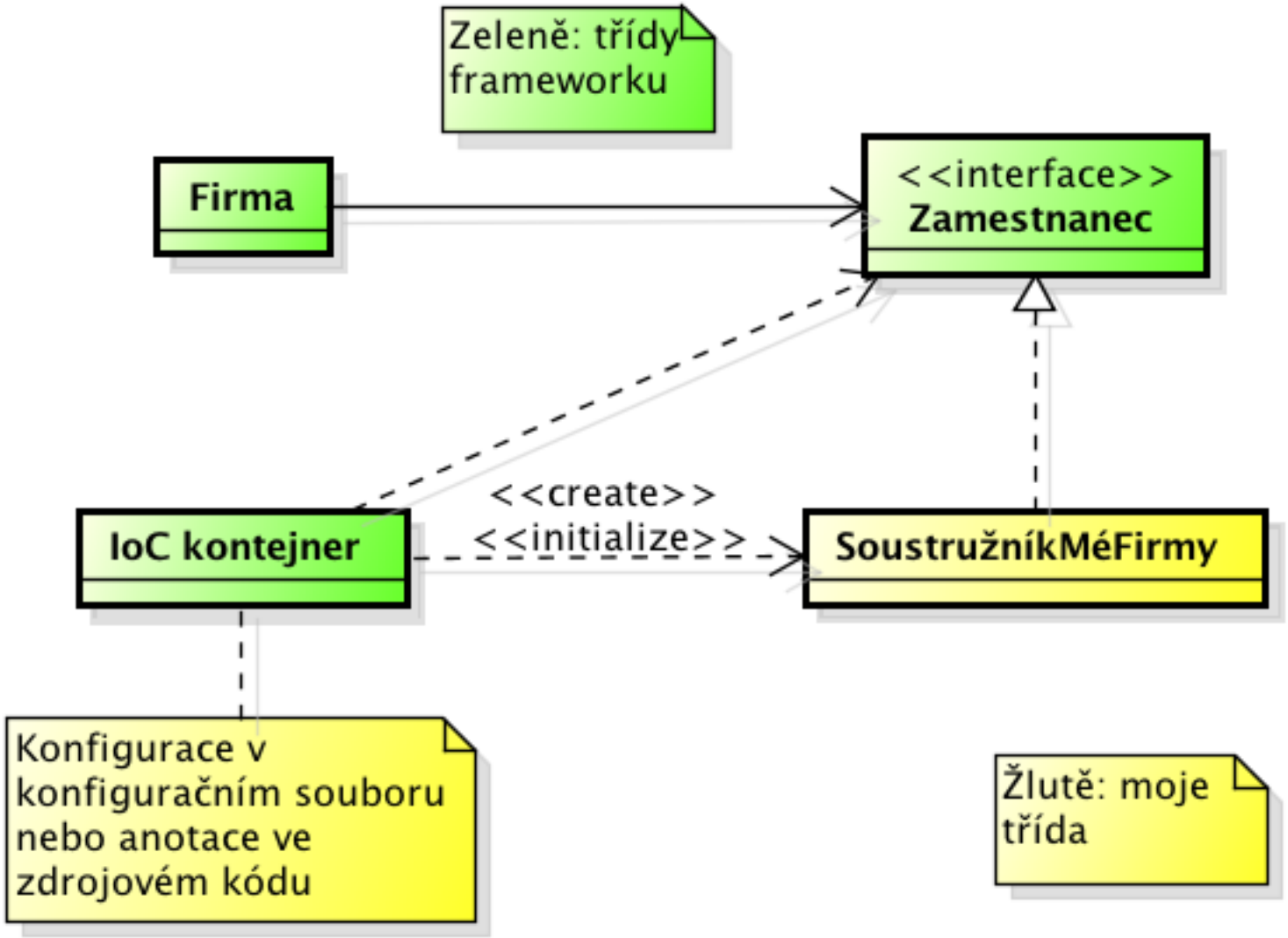
Organizace doménové logiky

- Transaction script
- **Domain Model** (preference)
- Table Module

Inversion of control



Inversion of control



Dependency injection

Způsob, jak realizovat IoC

Service Layer

- Typicky jednoduché API nad doménovou logikou
- Organizováno dle business transakcí odpovídajících případům užití
- Zahrnuje zabezpečení
- Extrémy:
  - Pouhá (tenká) fasáda, doménová logika soustředěna v nižší vrstvě
  - Sada transakčních skriptů obsahujících většinu doménové logiky, nížší vrstva pouze data wrappery
- Někde mezi je controller-entity architektura
  - Kontrolery (služby) jsou transakční skripty tvořící API

Service Layer

- EJB
  - Service layer realizována pomocí Session beans
    - Stateless
    - Statefull (pozor dvě injekce = 2 různé instance, každá má svůj stav)
  - Dependency Injection možná pouze pro Session beans (service layer), nikoliv pro Entity Beans (business objects), proto service layer je spíše transakční script a business objects jsou spíše wrappery datových objektů

Objektově-relační mapování

- JPA 2.0 (budou věnovány 2 přednášky), Hibernate
- Separace SQL přístupu od doménové logiky
  - Row Data Gateway – instance per řádek tabulky
  - Table Data Gateway – instance per tabulka – má metody zapouzdřující SQL dotazy, vrací ResultSet (někdy kolekce uložených procedur)
  - Active Record – objekt obsahuje (dědí) metody pro select/insert/update/delete plus metody obsahující doménovou logiku
  - Data Mapper – odděluje business objekty (jejich metody = výhradně doménová logika) od manipulace s databází (tu poskytuje Data Mapper)

Objektově-relační mapování

- JPA 2.0 (budou věnovány 2 přednášky), Hibernate
- Separace SQL přístupu od doménové logiky
  - Organizace vzhledem ke struktuře databáze
    - Row Data Gateway – instance per řádek tabulky
    - Table Data Gateway – instance per tabulka – má metody zapouzdřující SQL dotazy, vrací ResultSet (někdy kolekce uložených procedur)
  - Oddělení doménové logiky od obsluhy persistence
    - Active Record – objekt obsahuje (dědí) metody pro select/insert/update/delete plus metody obsahující doménovou logiku
    - Data Mapper – odděluje business objekty (jejich metody = výhradně doménová logika) od manipulace s databází (tu poskytuje Data Mapper)

Objektově-relační mapování

- Preference
  - Row Data Gateway
  - Data Mapper