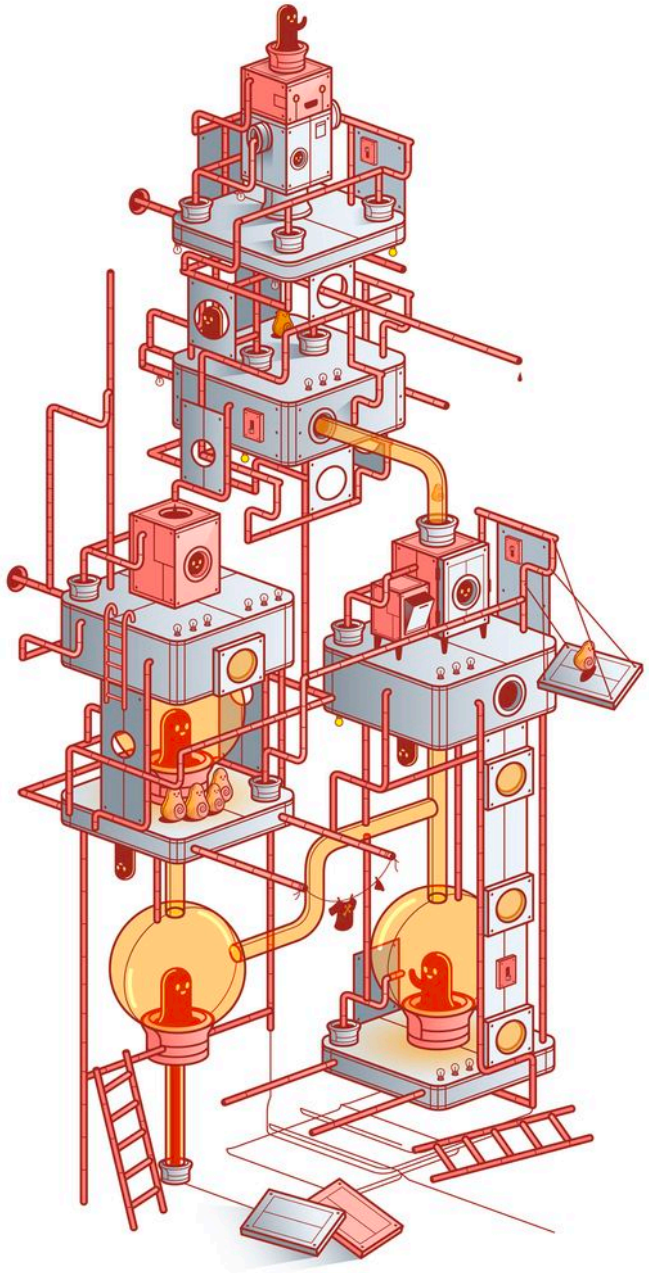


Search . . .

13

Domény, modely a eventy

- DDD - Domain Driven Design
- MDA - Model Driven Architecture
- EDA - Event Driven Architecture

Objektové Modelování - podzim 2022

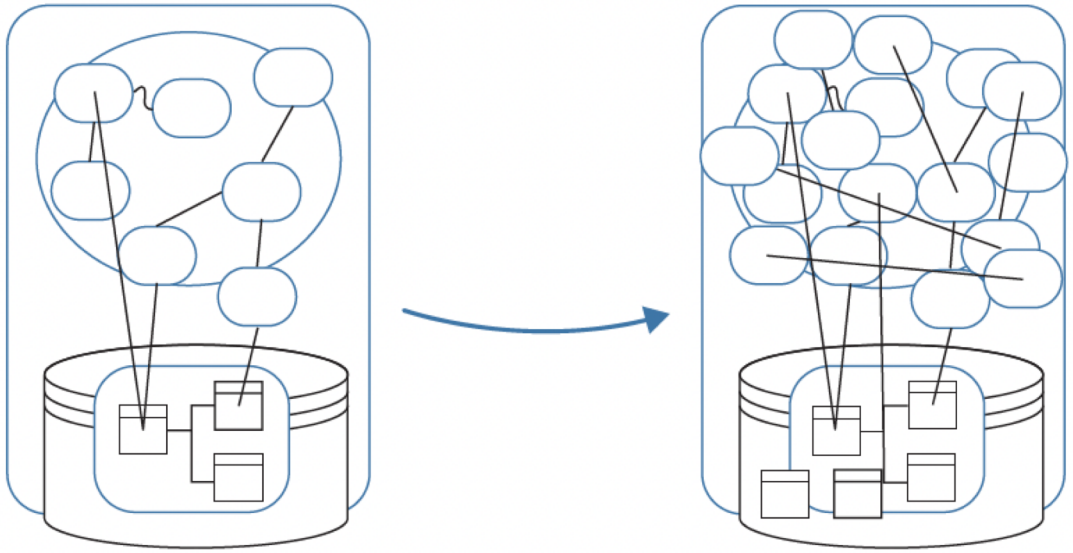
david.kadlecek@cvut.cz, www.czm.fel.cvut.cz

Next >

13 DDD – Domain Driven Design

Metodika softwarového vývoje, která se soustředí na správnou dekompozici problému do domén a realizaci takto dekomponovaného systému jak z pohledu softwarových technologí, tak z pohledu lidí a týmů.

PROBLÉM:



První verze systému vzniká zpravidla rychle a efektivně

Postupem času se ze systému stává monolitické spaghetti, kde je těžké dělat jakoukoliv úpravu a rozšíření (tzv. **Ball of Mud**)

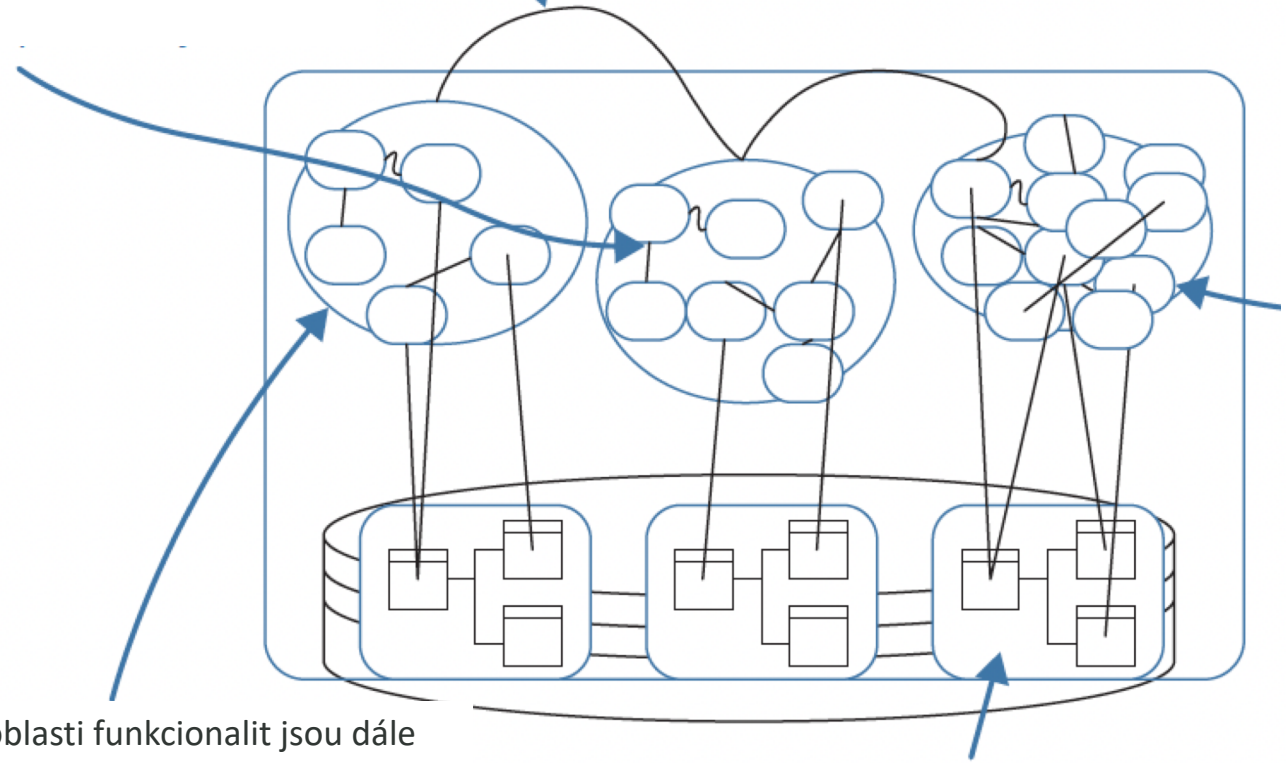
Eric Evans - Domain-Driven Design: Tackling Complexity in the Heart of Software (Addison-Wesley Professional, 2003)

13 DDD – Domain Driven Design

ŘEŠENÍ:

Identifikujeme klíčové části systému, ty izolujeme tak, abychom je mohli rozšiřovat nezávisle

Jednotlivé oblasti mají hranice a interagují spolu pouze přes vystavené API (hraniční přechody) - dokonce nemohou jinak



I tak nám vzniká tzv. **Bull of Mud**, který je ale zapouzdřen v mnohem menším celku a tudíž významně redukujeme množství vazeb, které zvyšují rozsah dopadu změny

Velké oblasti funkcionalit jsou dále rozděleny na podoblasti, které jsou uchápatelné a spravovatelné

Data jsou rozdělena do oblastí, které jsou od sebe opět izolovány hranicemi

13 DDD – Rozdělení problému na (sub)domény

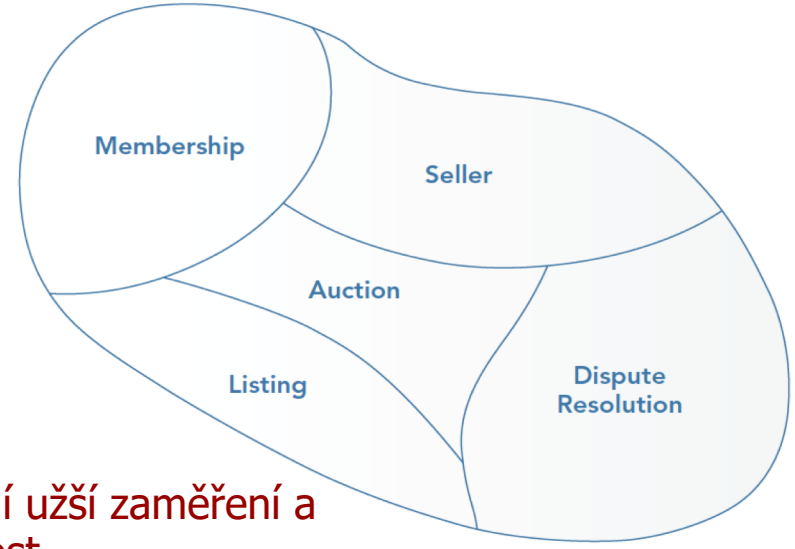
- Jak rozdělit problém správně do subdomén
- Jak identifikovat klíčové domény řešeného problému?
- Jak se zaměřit na oblasti, které jsou důležité pro business?
- Jak vypadají hranice mezi doménami?
- Jak přiřadit lidské zdroje k doménám?

Měl bych mít možnost kdykoliv vzít jednu oblast a s minimálním úsilím a dopadem (minimální rework, testování, organizační změny atd.) do ostatních domén:

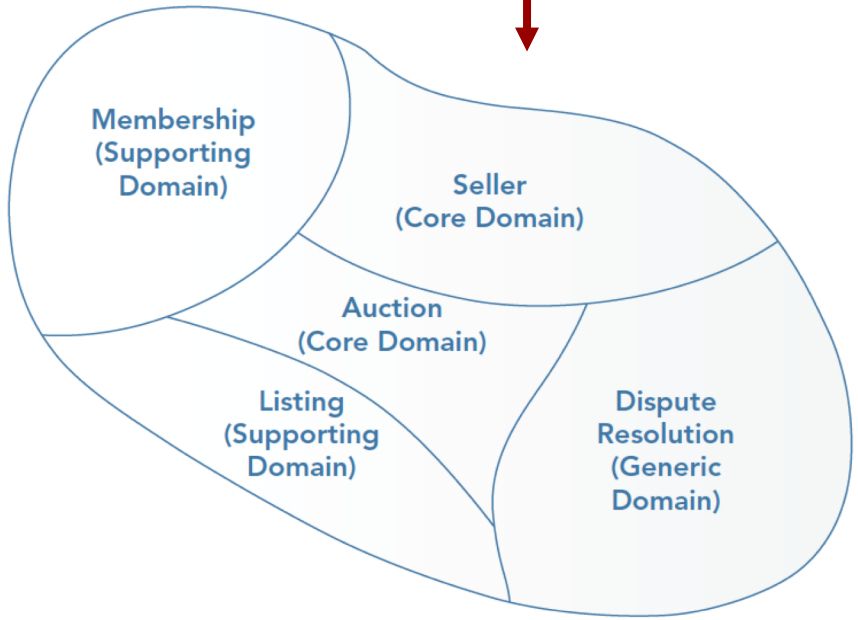
- nasadit opravu
- upravit funkcionalitu
- migrovat funkcionalitu na jinou technologii
- rozšířit z pohledu CPU a datového úložiště
- outsourcovat do jiného oddělení nebo společnosti

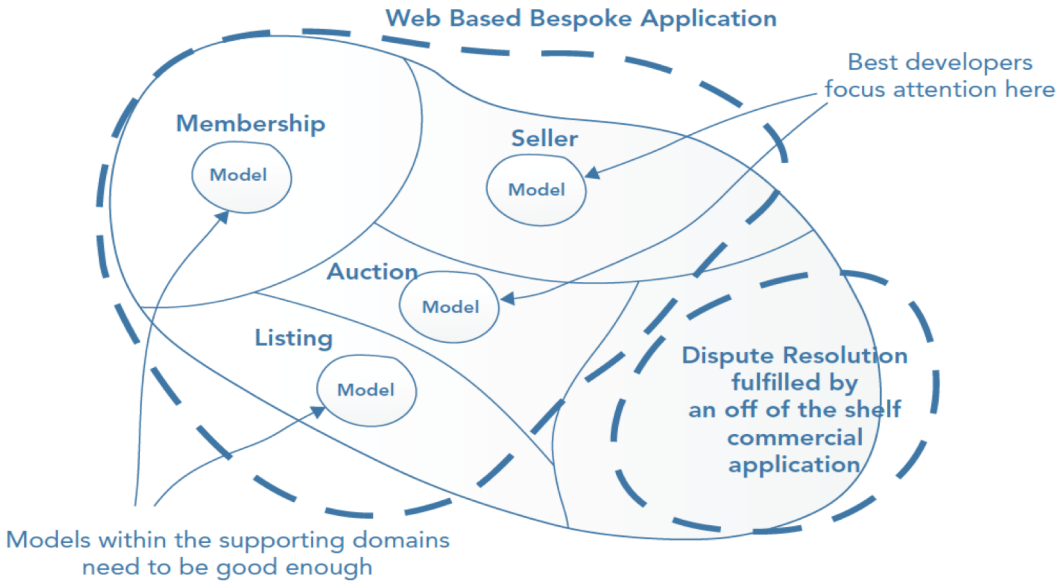
13 DDD - Rozdělení problému na domény

Doména celého problému, který řeším



Domény, které mají užší zaměření a uchopitelnou velikost





13 DDD - DOMÉNY

Domény z předchozího příkladu:

- *Membership* - stará se o registrace, preference, detaily členů (supporting)
- *Seller* - veškeré aktivity prodejce (core)
- *Auction* - řízení a časování aukcí, práce s nabídkami (core)
- *Listings* - seznamy položek pro dražbu (supporting)
- *Dispute resolution* - řešení sporů mezi prodejci a kupci (generic)

Domény můžeme rozdělit do tří základních typů:

- ***Core domain*** - zpravidla reprezentuje core business firmy (přináší nejvíce peněz a kompetitivní výhodu). Dáváme na ní ty nejlepší vývojáře. Na core domain se díváme spíše jako na business produkt než jako na projekt.
- ***Generic domain*** - zpravidla vrstvy a nástroje, které jsou využíváno hlavním (core) businessem.
- **Supporting domain** – ostatní, které jsou spíše podpůrné pro ostatní části

13 DDD - DOMÉNOVÝ MODEL

Klasický softwarový model (objektový, datový…) se snaží o reprezentaci informací ve vrstvě ve které existuje ve formě entit, atributů a vazeb.

Doména v DDD je chápána jako koherentní oblast zájmu businessu. Např. se může jednat o doménu retailového bankovnictví. Souhrn abstrakcí, chování (logiky) a UI interakcí pak určuje model v této doméně.

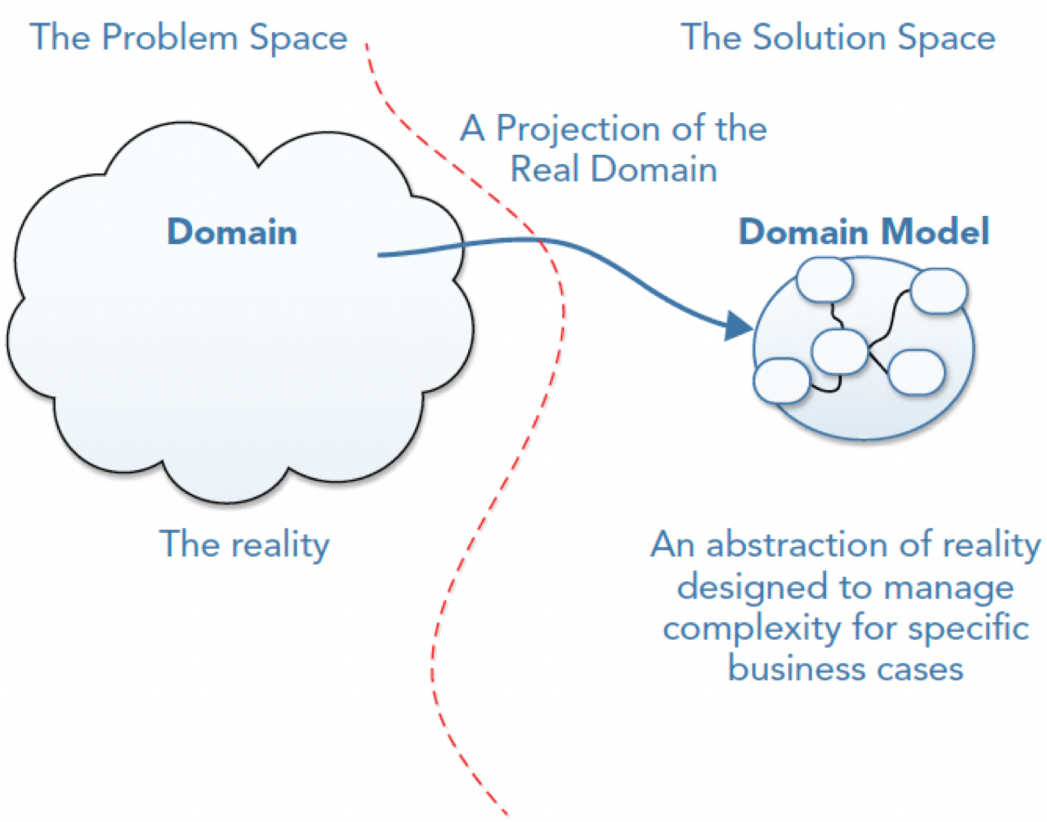
Doménový model v DDD se snaží o kompletní splnění všech požadavků – tedy kromě informací popisuje i chování, omezení, komunikační jazyk…

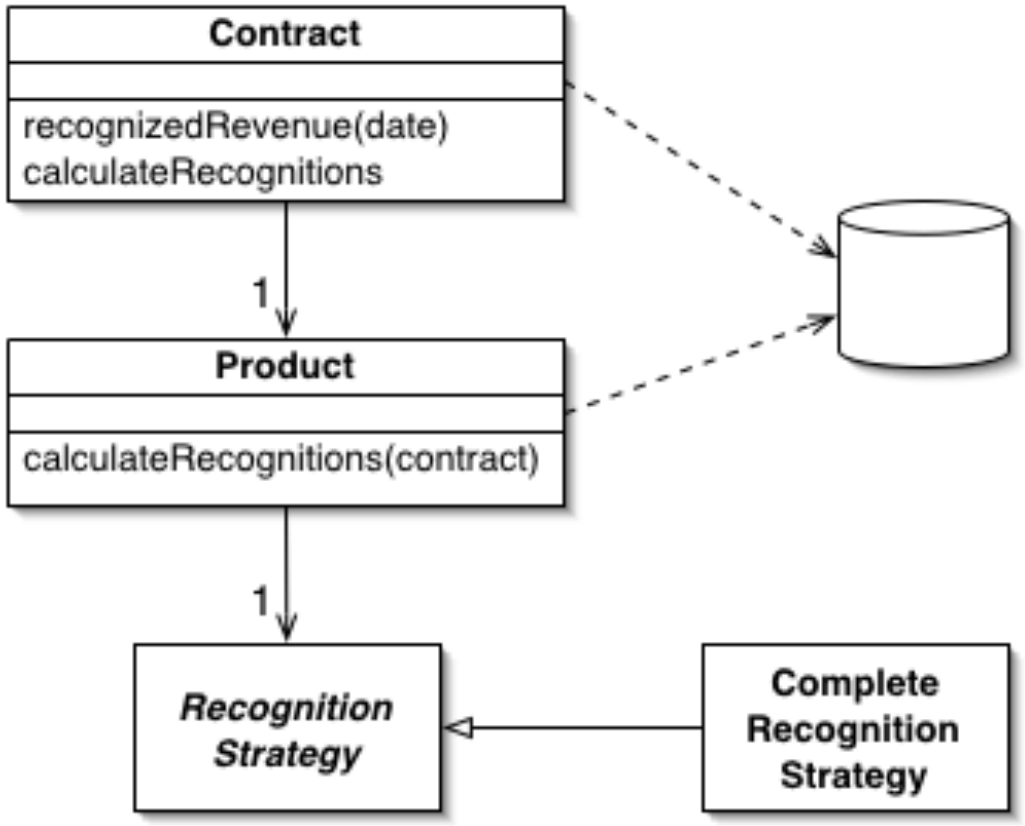
Doménový model v DDD tedy obsahuje:

- Objekty patřící do dané domény – např. v bankovní doméně máme objekty jako banka, účet, transakce…
- Relace mezi těmito objekty (včetně kardinalit)
- Chování, které tyto objekty vykazují při vzájemné interakci (hlavní operace) - např. výpis účtu - *issue statement*, debetování účtu - *debt account*
- Doménový jazyk – jazyk používaný lidmi pracujícími v dané doméně. Např. v doméně retailového bankovnictví *debet*, *kredit*, *portfolio* nebo operace *převod peněz*
- Kontext ve kterém model pracuje – předpoklady a omezení se kterými vyvinutý software musí pracovat/musí zajistit. Např. nový účet musí být založen pouze pro žijícího člověka musí mít nezáporný zůstatek.

Pozn. Je možné jít až na úroveň atributů nebo minimálně hlavních atributů (doporučuji)

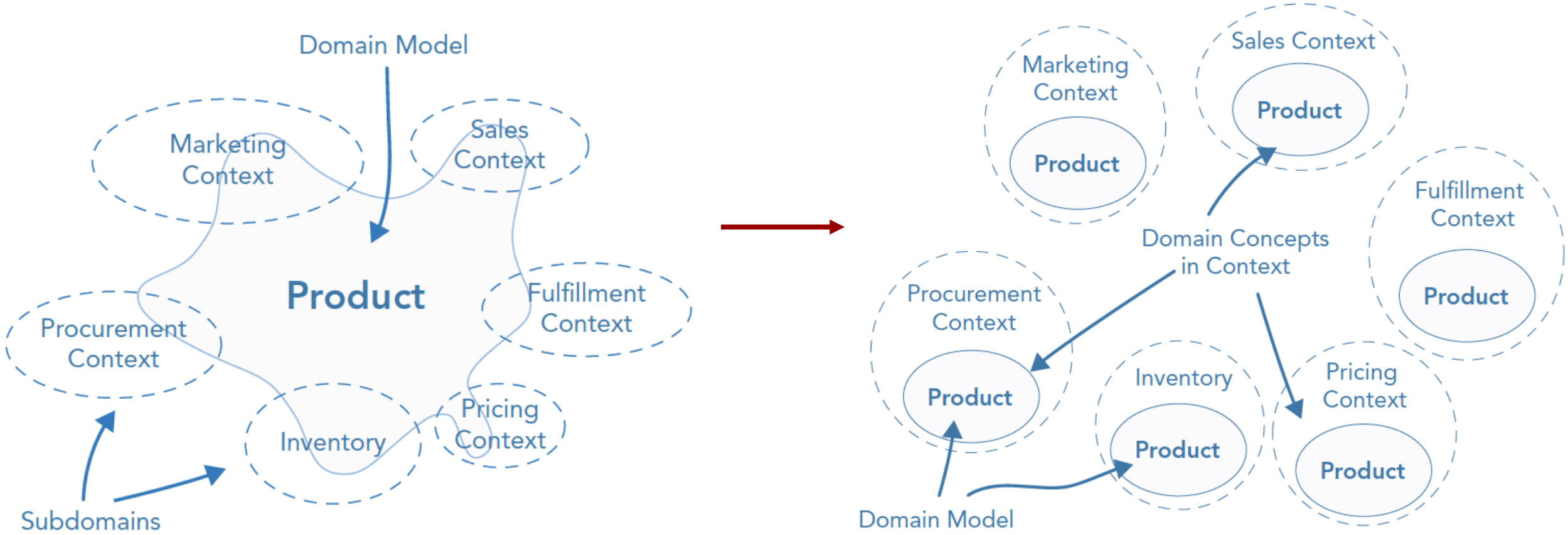
13 DDD - DOMÉNOVÝ MODEL





13 DDD – BOUNDED CONTEXT

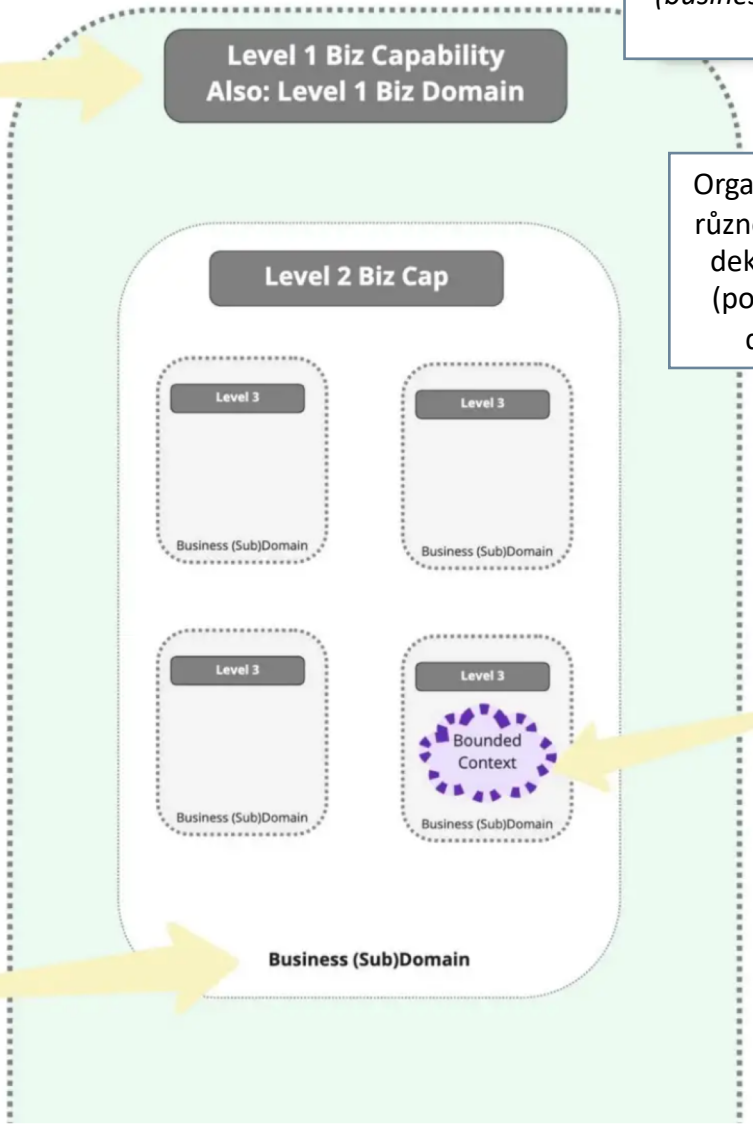
Bounded context je asi nejdůležitější termín z DDD. Zasazuje část doménového modelu do kontextu, kde má svůj unikátní význam a chování. Je to důležité, protože to zároveň tvoří hlavní vodítko pro hranice microservis a transakcí, které pracují nad daty.



13 DDD – Doména, subdoména, bounded context

Organizace může mít jednu nebo více domén na první úrovni

Doména je oblast businessu, které poskytuje spolu související schopnosti *(business capability)*



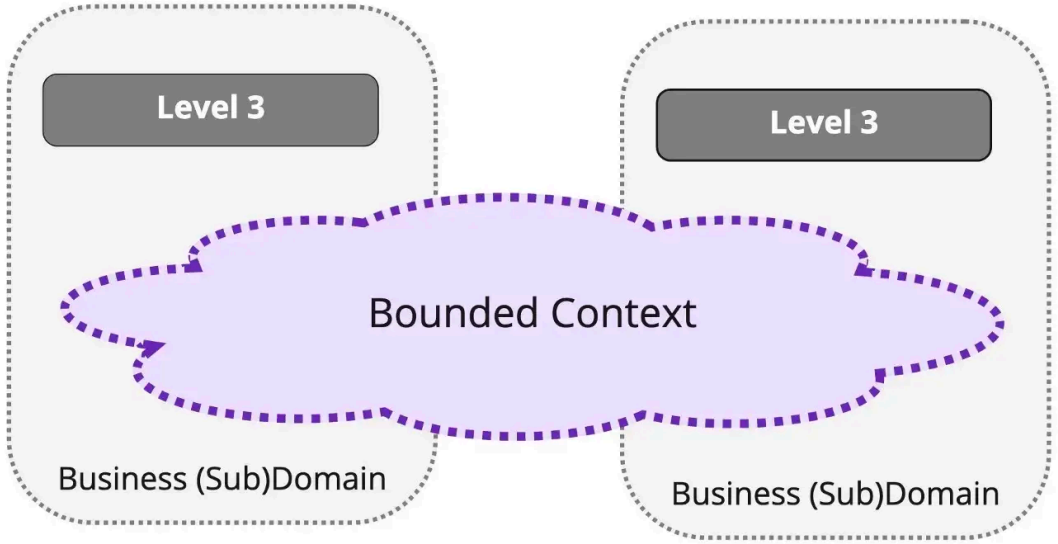
Organizace mají různou hloubku dekompozice (počet vrstev domén)

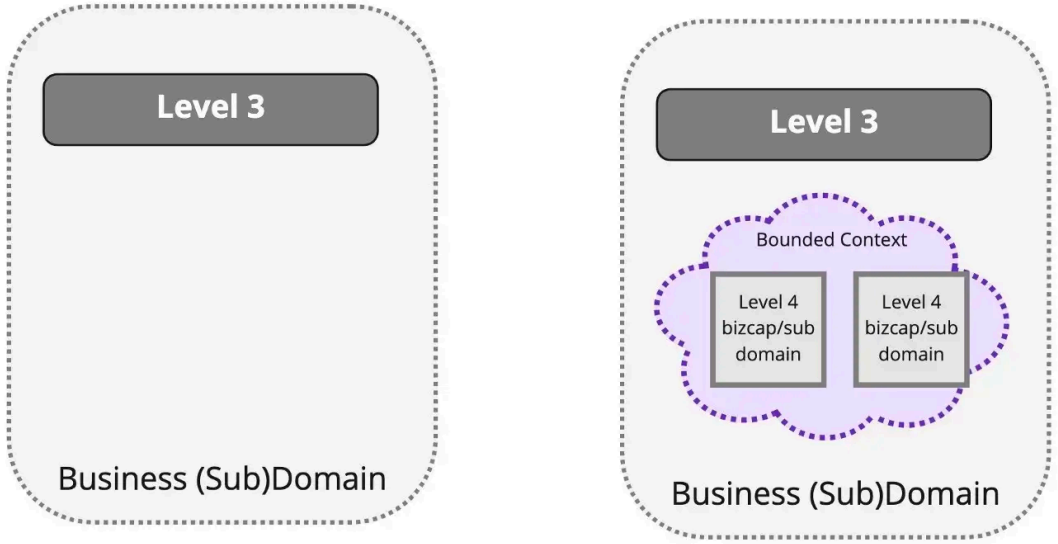
Bounded context nejčastěji "sedí" uvnitř domény = nejjednodušší z pohledu dalších věcí. Ale není to striktní pravidlo (viz další slide)

Termín doména a subdoména jsou zaměnitelné, jedná se o hloubku zanoření

DDD – Doména, subdoména, bounded context

Jedna microservice nepřesahuje hranice bounded contextu, v jednom bounded contextu může být více microservis



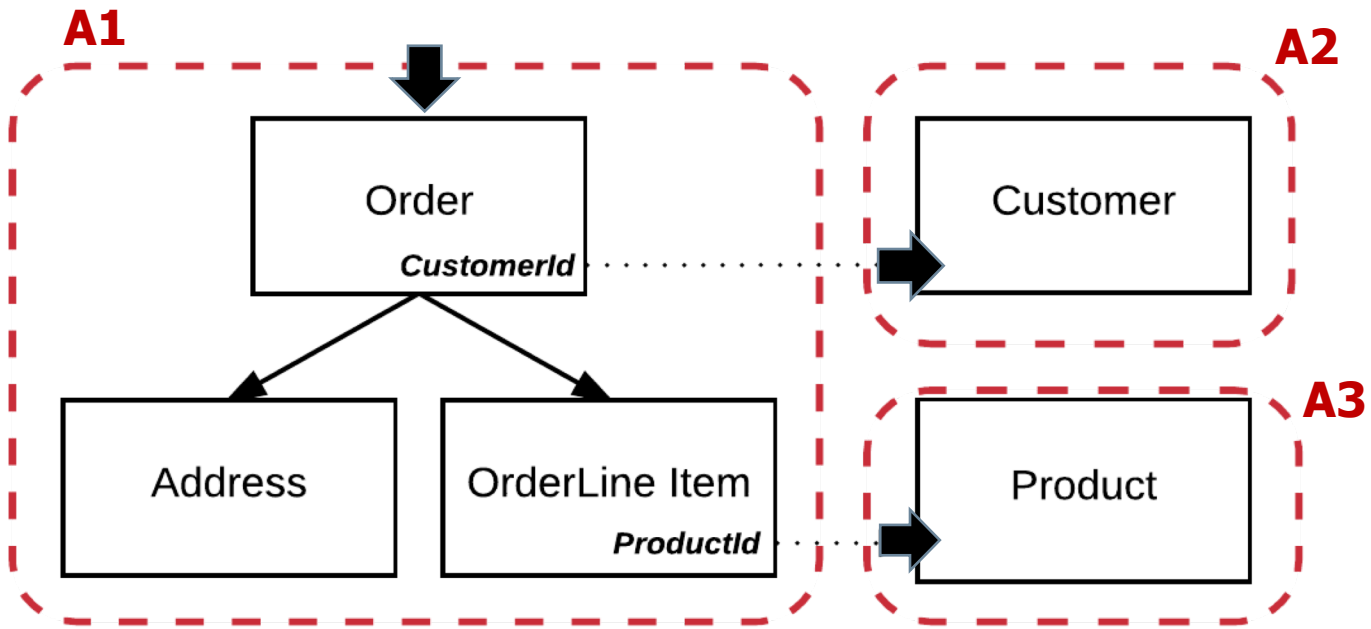


13 DDD – Agregát

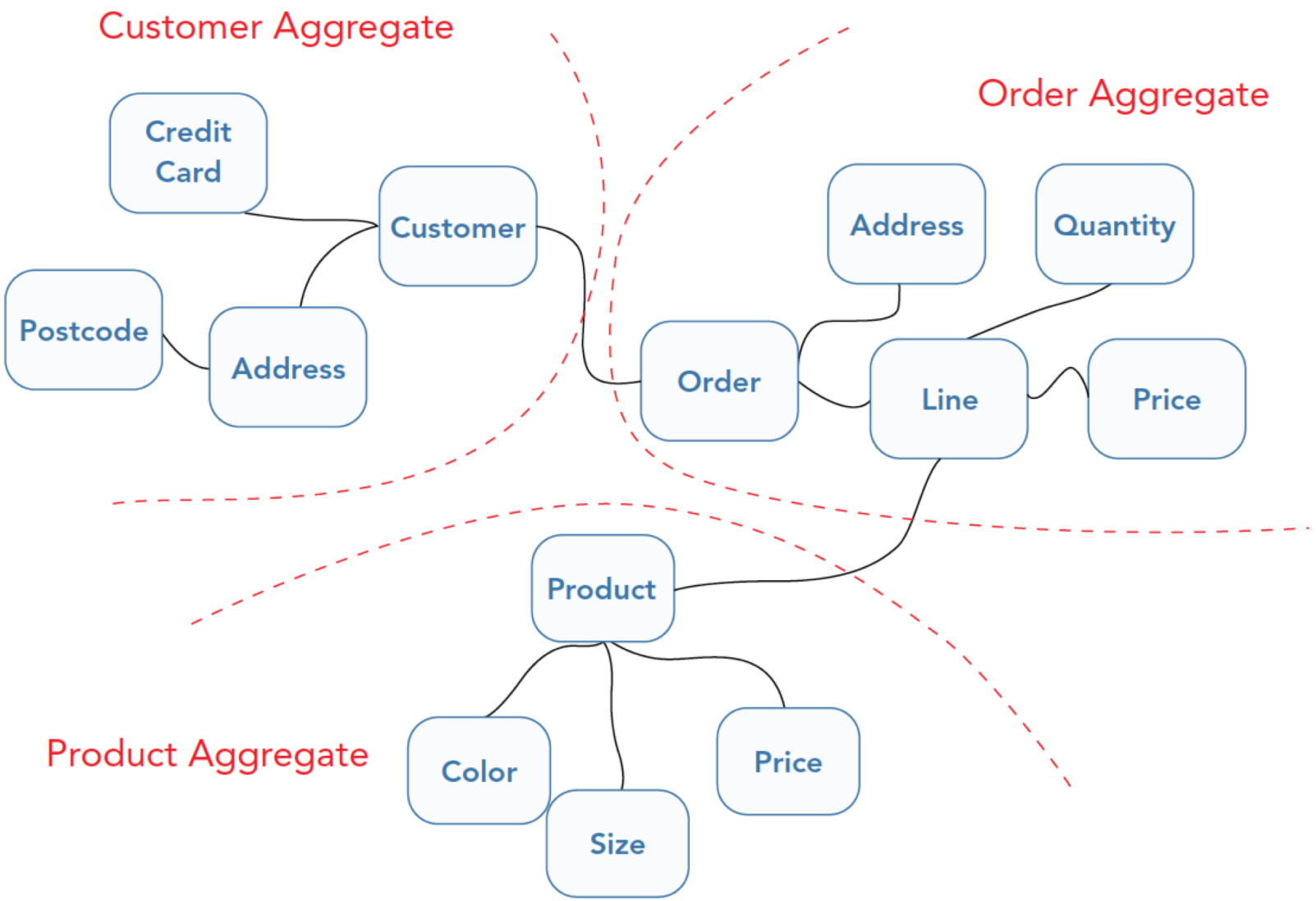
Agregát je skupina objektů (nebo tabulek), ke kterým se mohu chovat jako k uzavřené jednotce. Pracuji s ním vždy zkrz tzv. ***Agregate Root***

Pravidla:

- Agregát má vždy jeden **Agregate root** (kořenový) objekt a sadu dalších navázaných objektů
- Agregát se linkuje na další agregáty pomocí ***Id*** jejich root objektů *(vzpomeňte na lazy loading pattern pomocí Ghost)*
- Jeden agregát = jeden command, který s ním pracuje
- Scope transakce = agregát



13 DDD – Agregát

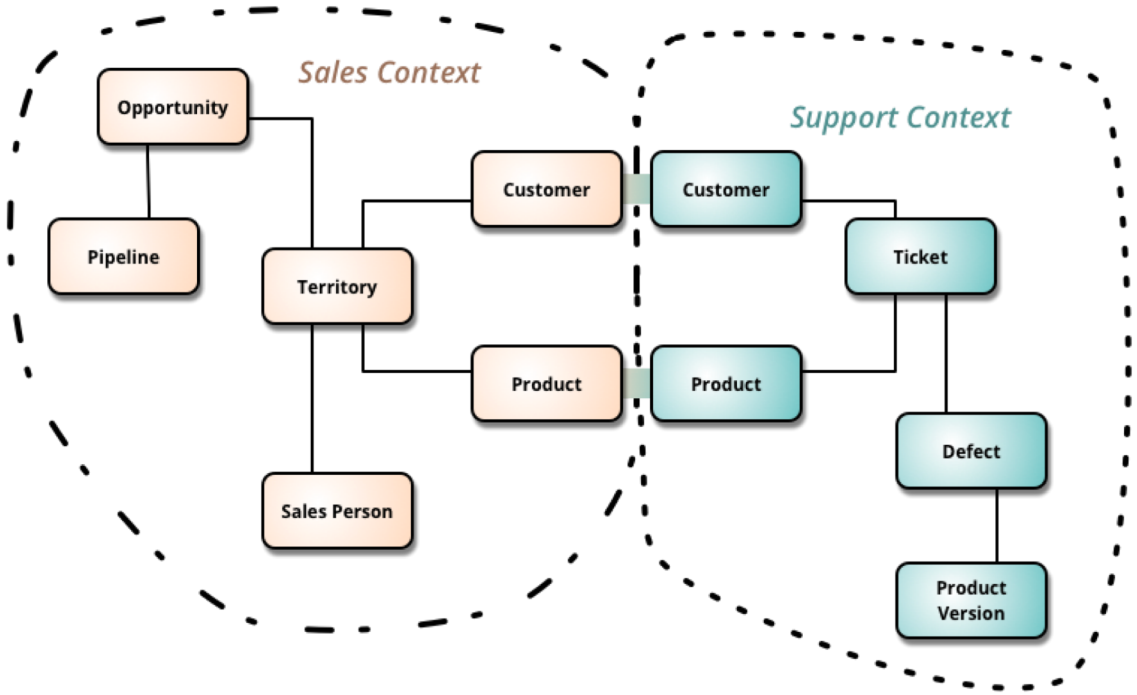


13 DDD – Agregát a bounded context

Ty samé entity mohou sedět ve více bounded kotextech

1. autorizační microservisa může mít doménový model s *user_id* a dalšími detaily pro autorizaci
2. order microservice může mít doménový model s *customer_id* a detaily zákazníka
3. shipping microservisa může mít doménový model s *customer_id*, jeho adresou atd.

Příklad 2 jiných bounded kontextů, kde se částešně překrývají entity *Customer* a *Product*

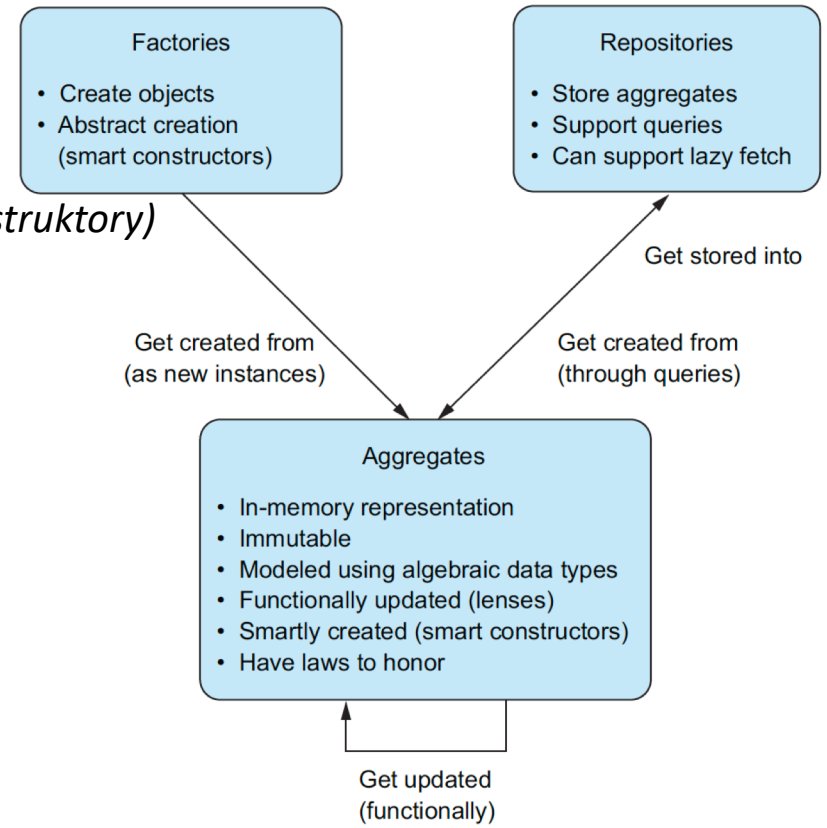


13 DDD - Životní cyklus doménového modelu

Factory
- *Vytváří objekty*
- *Abstrakce vytváření creation (smart konstruktory)*

Repository
- *Perzistuje agregáty*
- *Podporuje další dotazy*
- *Může podporovat lazy loading*

Agregáty
- *Sedí v paměti*
- *Immutable*
- *Modelované jako ADT*
- *Updatovány funkcionálně (lenses)*
- *Vytvářeny přes Factory*



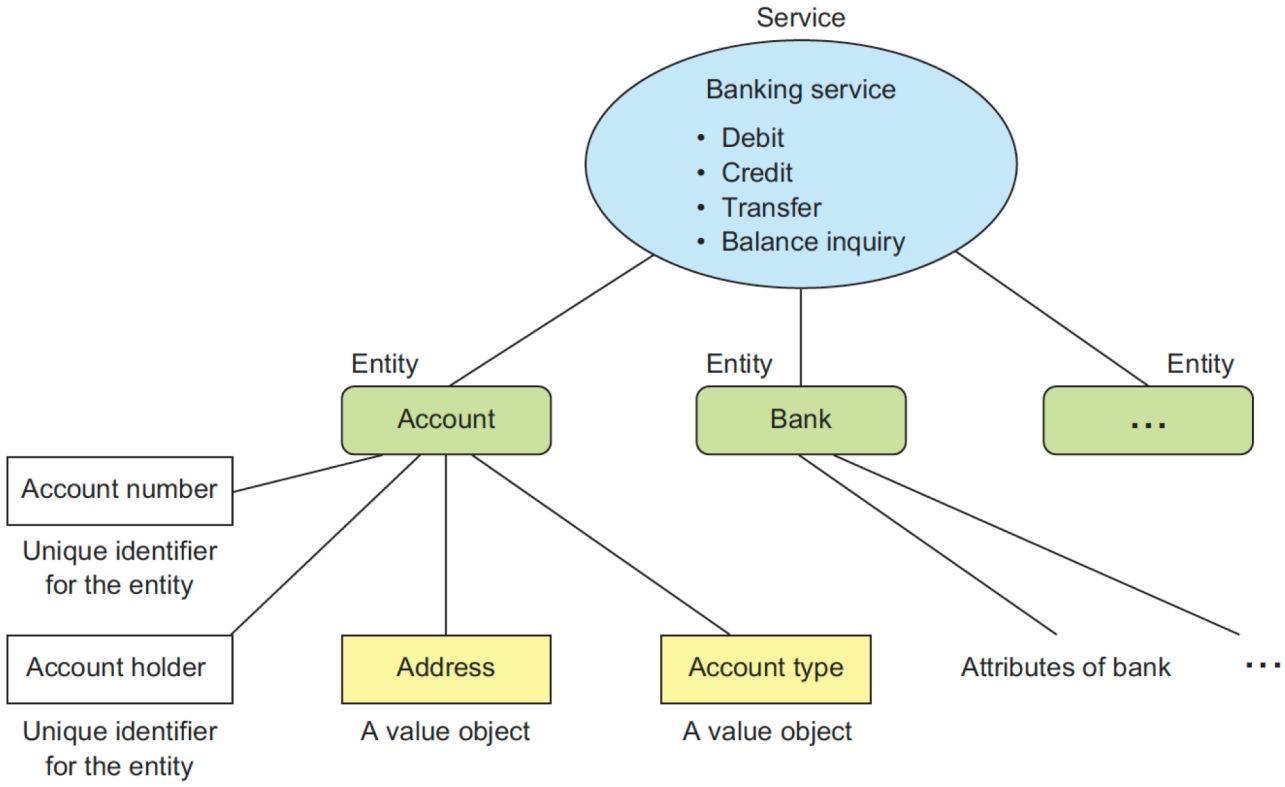
Pro zájemce, není nutno chápat na zkoušku: "Relational Lenses, a Language for Updatable Views," by Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan (http://dl.acm.org/citation .cfm?id=1142399).

13 DDD – entita, value objekt, service

Entita
- Má vlastní identitu
- Prochází různými fázemi životního cyklu
- Má konečný životní cyklus v daném businessu

Value objekt
- Immutable
- Může být volně sdílen mezi entitami

Service
- Realizuje operace
- Vyšší úroveň abstrakce než entita nebo value objekt
- Může pracovat s více entitami a value objekty
- Obvykle realizuje business use case



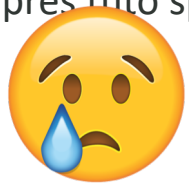
13 DDD – Doménový Event

- Doménový event reprezentuje změnu v doméně
- Doménové eventy by měly být v jazyce domény ve které jsou používány
- Doménové event unikátní typ —tento typ je definován v doménovém modelu
- Doménové event je soběstačný (Self-contained) —obsahují kompletní informace popisující nastalou změnu nebo změnu, která má být provedená
- Doménové event eventy mohou být dále rozděleny na podtypy:
  - **Event** – immutable stav a hodnota entity, která nastala během provádění operací v doméně (mezi službami)
  - **Command** – asynchroní forma RPC (Remote Procedure Call), kdy říkáme konzumujícímu systému jakou změnovou operaci má provést
  - **Query** – podobné jako command, ale očekává vrácení výsledku (např. hodnoty nějaké entity) bez změny stavu

13 DDD – Anticorruption layer

Bounded context - je o kontrole hranice, důsledném zapouzdření a izolaci

ANTICORRUPTION LAYER - zařizuje “čistou a jasnou” hranici mezi doménami. Veškerá komunikace a transformace dat mezi doménami probíhá přes tuto specializovanou vrstvu (API)



Q: Uffff… když mám entitu Product ve více bounded kontextech, tedy ve více microservisách, co když jeho data mohu modifikovat ve všech těchto microservisách? Kde je pak single source of truth dat o produktu?
A:

1) *V ideálním případě mám jedinou microservice, které mi zapouzdřuje veškeré operace nad produktem a pracuje se s jediným agregátem pro produkt.*
2) *Pokud ne, tak:*
    i) *Někdy mám výhodu v tom., že každý bounded context/microservice má trochu jiný agregát - v jednom se např. spravují ceny k produktu, jinde jeho parametry a fotografie*
    ii) *Jindy v jednom bounded contextu data k produktu čtu i zapisuji, v druhém pouze čtu (např. přes materialized view)*
    iii) *Občas se bohužel stane, že v obou bounded contextech zapisuju ta samá data. Pak to řeším přes Ságu a eventuální konzistenci - viz následující slidy*

13 DDD – SHRNUTÍ

Bounded Context - zasazuje část modelu do kontextu, kde má svůj unikátní význam, chování a izolaci

Agregát - objektový graf který spravujeme jako celek (Agregát root - kořen agregátu)

Entita - business objekt reprezentující doménový koncept se stabilní identitou

Value objekt - immutable objekt reprezentující kompozitní hodnotu objektu

Domain Service - domain behavior, které se nepersistuje (např. vypočti daň)

Doménový Event - objekt reprezentující změnu v doméně

Repository - fasáda odpovědná za perzistenci agregátu

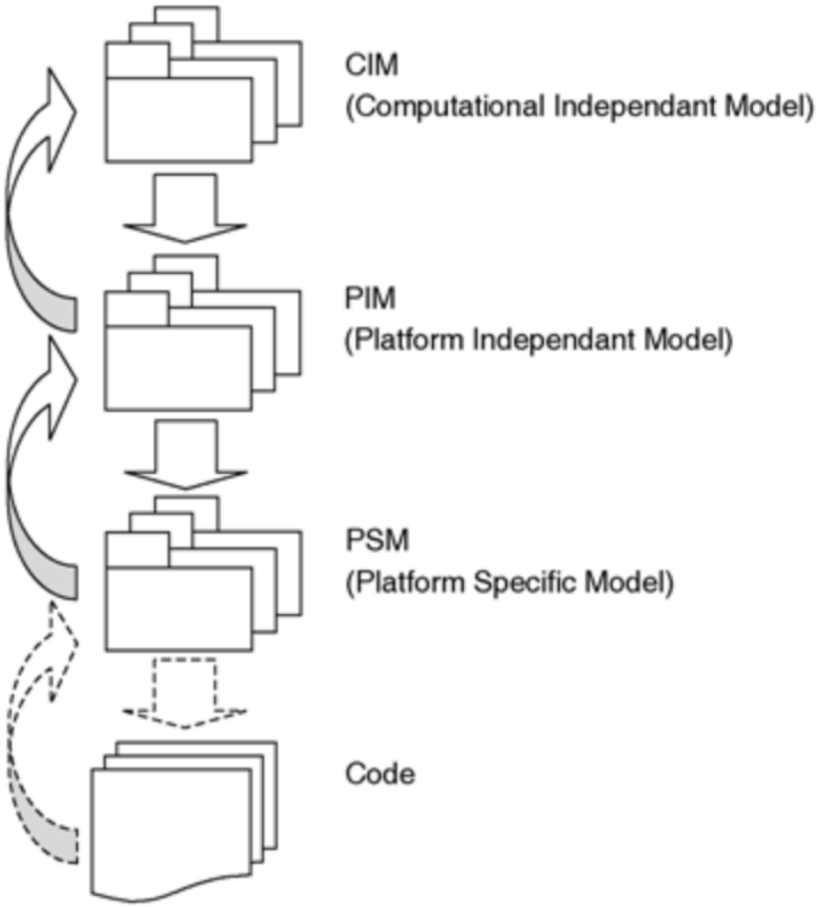
Factory – abstrakce vytváření objektů

MDA/MDD – Model Driven Architecture a Model Driven Development

Computation Independent Model (CIM) – označujeme jako primární model. Tento model odráží systémové a softwarové znalosti z business pohledu.

Platform Independent Model (**PIM**) - model softwarového systému nebo obchodního systému, který je nezávislý na konkrétní technologii použité k jeho implementaci.

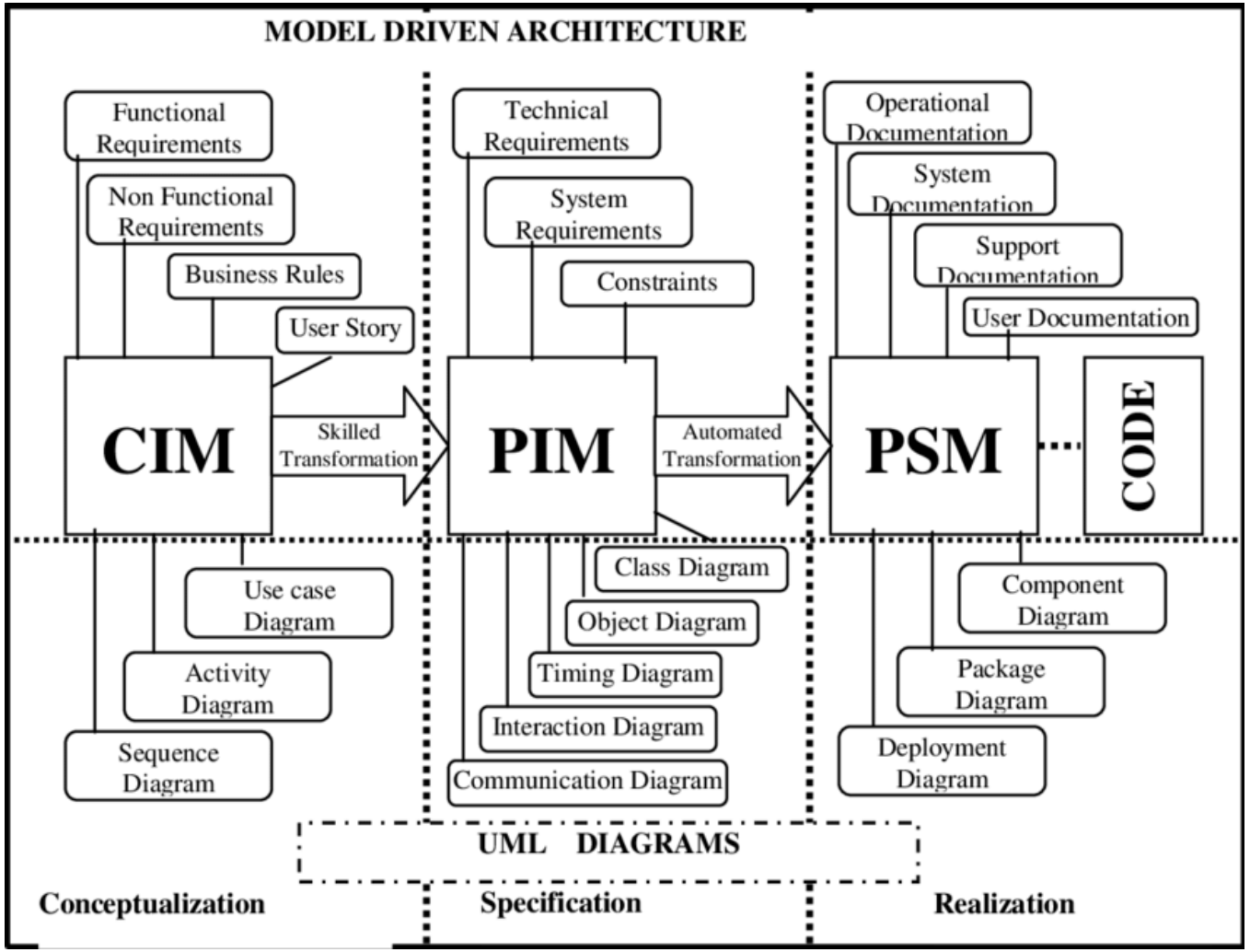
Platform Specific Model (PSM) - je model softwaru nebo obchodního systému, který je propojen s konkrétní technologickou platformou (např. konkrétní programovací jazyk, operační systém, formát souboru dokumentu nebo databáze). Pro vlastní implementaci systému jsou nezbytné modely specifické pro platformu.



MDA – Model Driven Architecture

MDA vychází z premisy, že model je základ všeho. Snažím se všechno modelovat pomocí např. UML. Jak se zvyšuje maturita projektu, tak se na základě abstraktnějších modelů vytvářejí konkrétnější a detailnější modely. Jednotlivé úrovně modelu jsou propojené nebo vznikají transformací jeden z druhého. Na nejdetailnější úrovni pak generuji části aplikace/kódu z modelu.

V případě potřeby změny, provádím změnu ideálně na úrovních, kde je ta změna relevantní a to směrem zleva doprava



13 EDA - EVENT DRIVEN ARCHITECTURE/PROGRAMMING

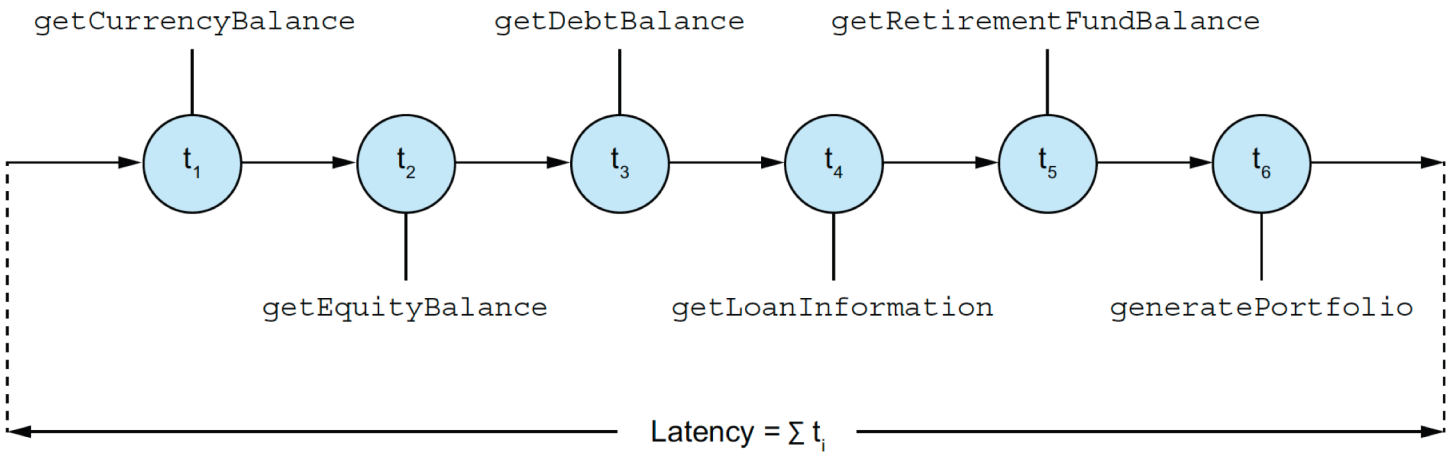
Místo abychom interakci mezi funkcionalitami a moduly řešili formou blokujících synchronních volání, tak interakce řešíme pomocí *eventů* (co se stalo), *commandů* (co se má provést/provolat) a *queries* (chci informaci)

Event-driven programming models make events an important architectural element

13 EDA - EVENT DRIVEN ARCHITECTURE/PROGRAMMING

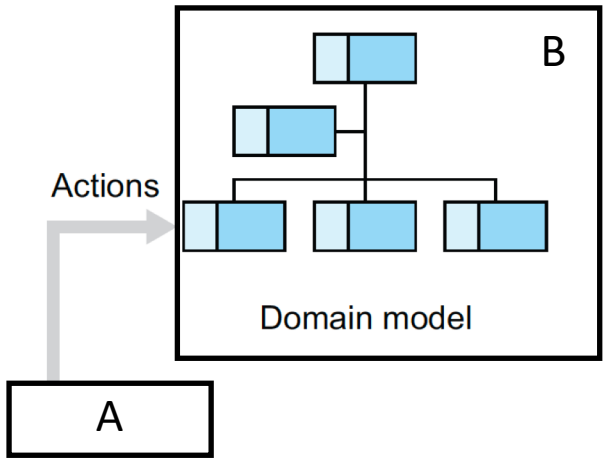
Nevýhody synchronní integrace:

- *Horší performance*
  - *Volané funkce blokují všechny thready, které je volají (dokud není požadavek odbaven)*
  - *Exekuce je sekvenční*
- *Svazování modelu přes pevná rozhraní*
- *Náročné na zásahy*



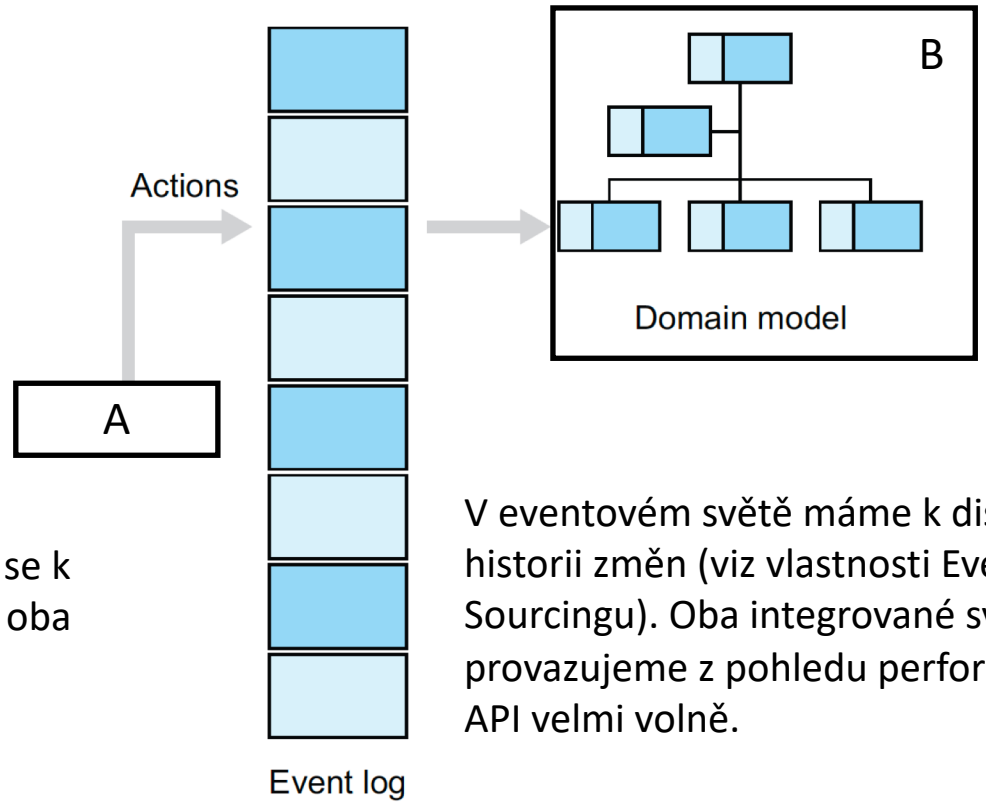
EDA - COMMAND FIRST vs EVENT FIRST PATTERN

Command first pattern



V synchronním světě updatujeme doménový model a nedostaneme se k historii změn. Provazujeme pevně oba integrované systémy z pohledu performance a API.

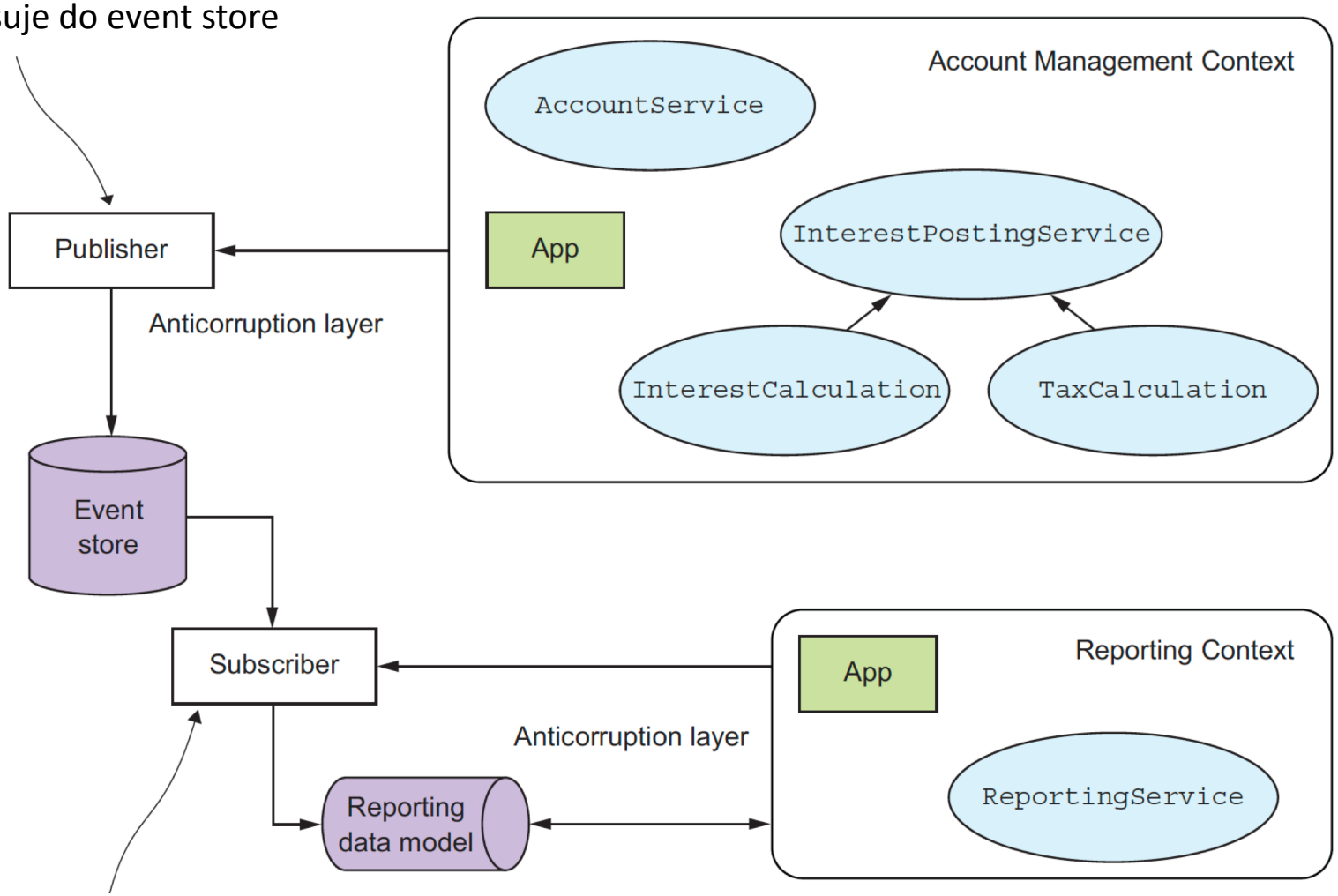
Event first pattern



V eventovém světě máme k dispozici i historii změn (viz vlastnosti Event Sourcingu). Oba integrované systémy provazujeme z pohledu performance a API velmi volně.

13 EDA – PUBLIKOVÁNÍ a SUBSKRIPCE K EVENTŮM

Publikuje doménové eventy a zapisuje do event store

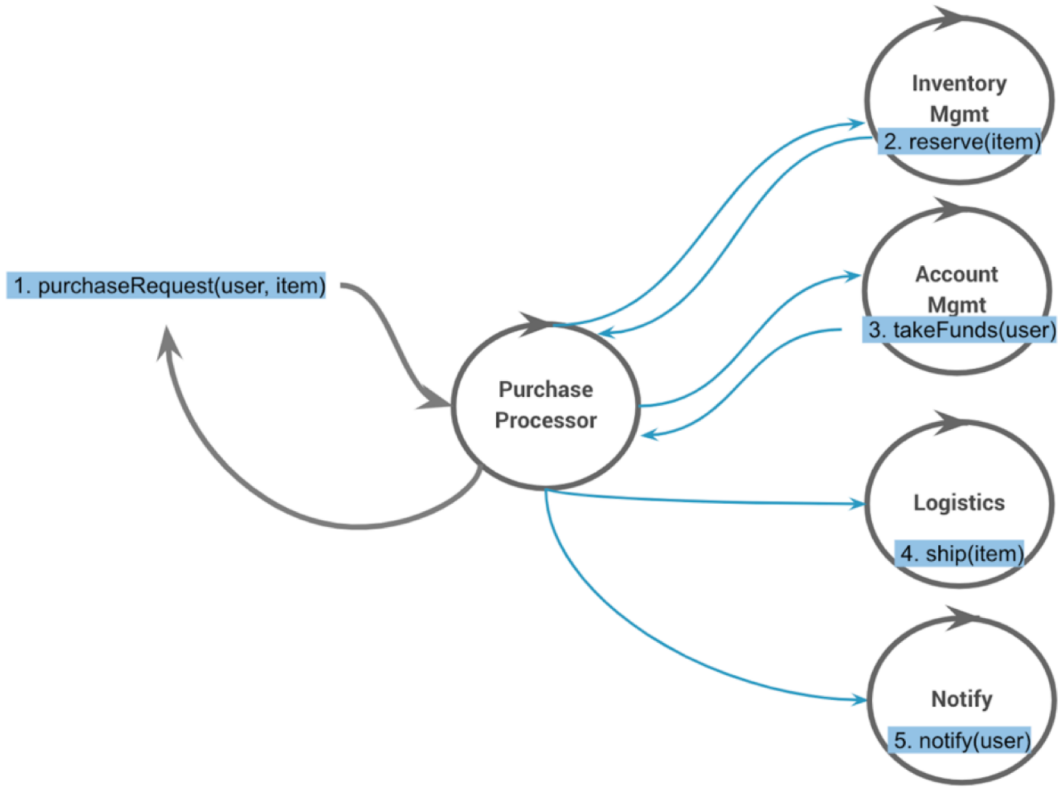


Subskribce k doménovým eventům a update datového modelu pro reporting

13 EDA - COMMAND FIRST vs EVENT FIRST PATTERN

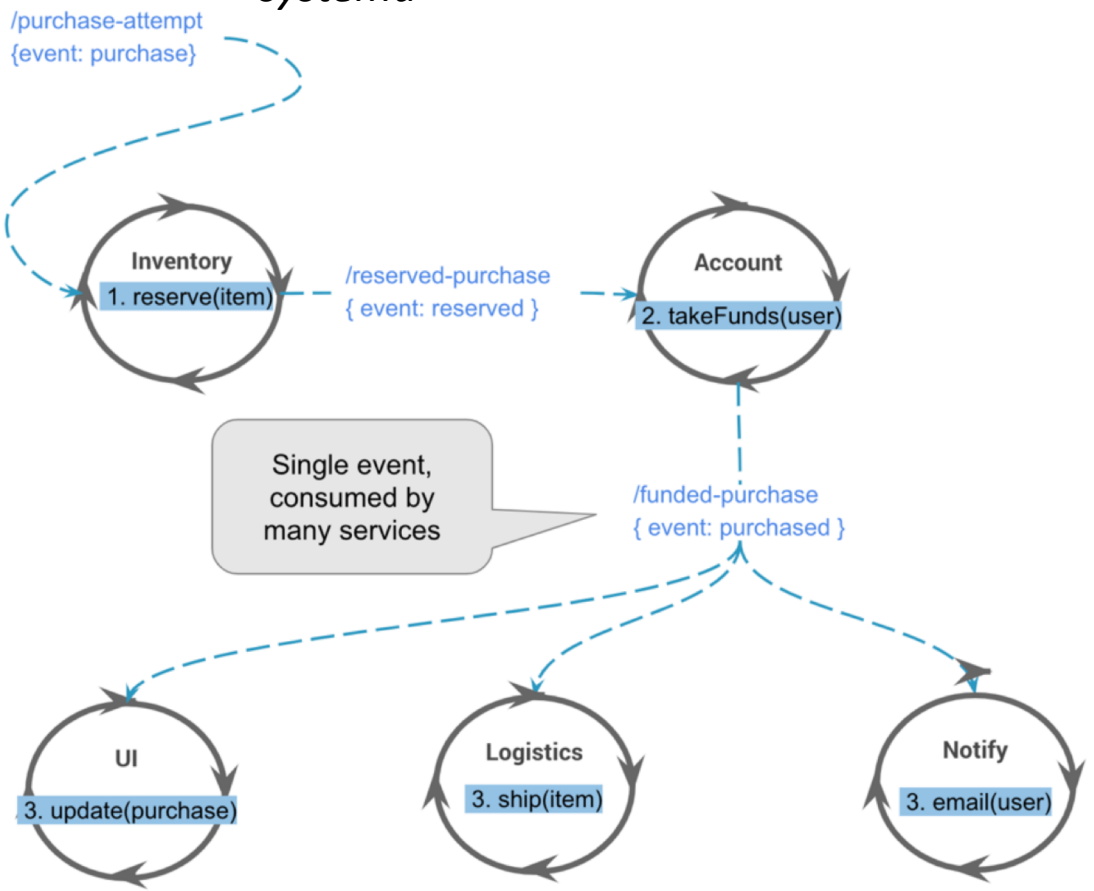
Command first pattern

1:1 interakce – synchronní volání je pouze mezi dvěma systémy



Event first pattern

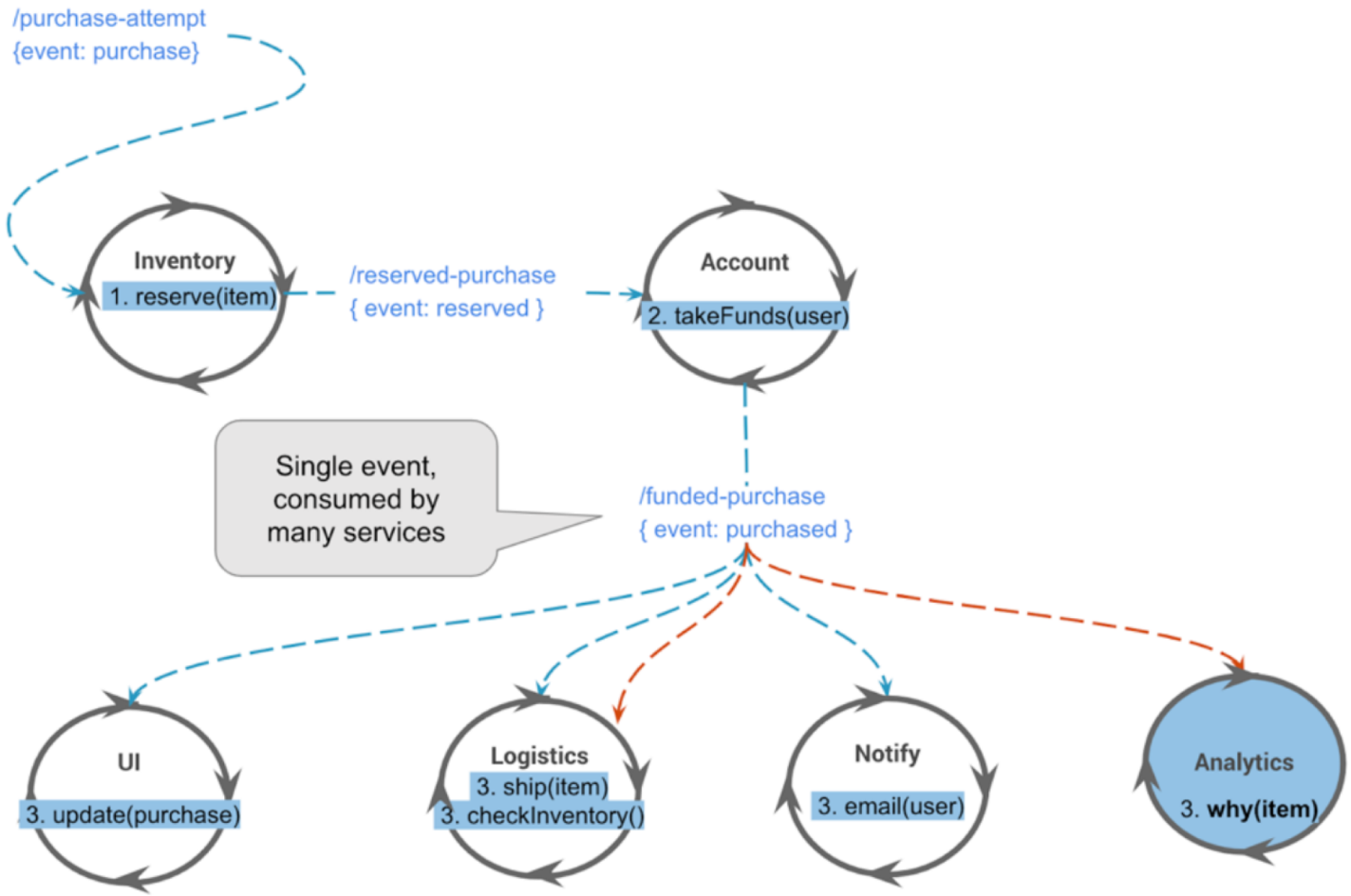
M:N interakce - Cílem eventu může být i více systémů



13 EDA - EVENT FIRST PATTERN

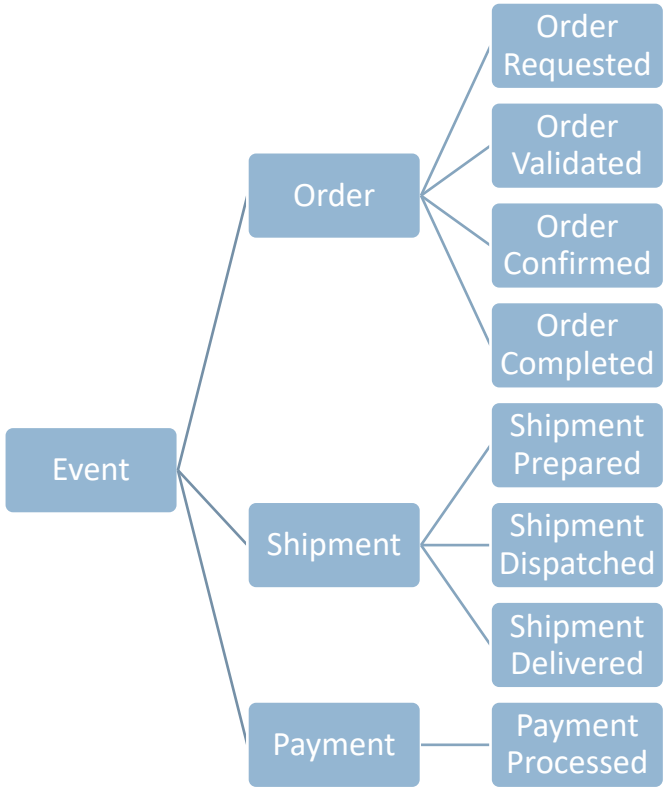
Event first pattern je také otevřený pro přidávání logiky a změny logiky jednoduše tím, že upravíme subskripci k eventům

Event first pattern je pro konstrukci funkcionality stejné jako IoC (Inversion of Control – viz. dependenci injection framework) pro konstrukci objektů

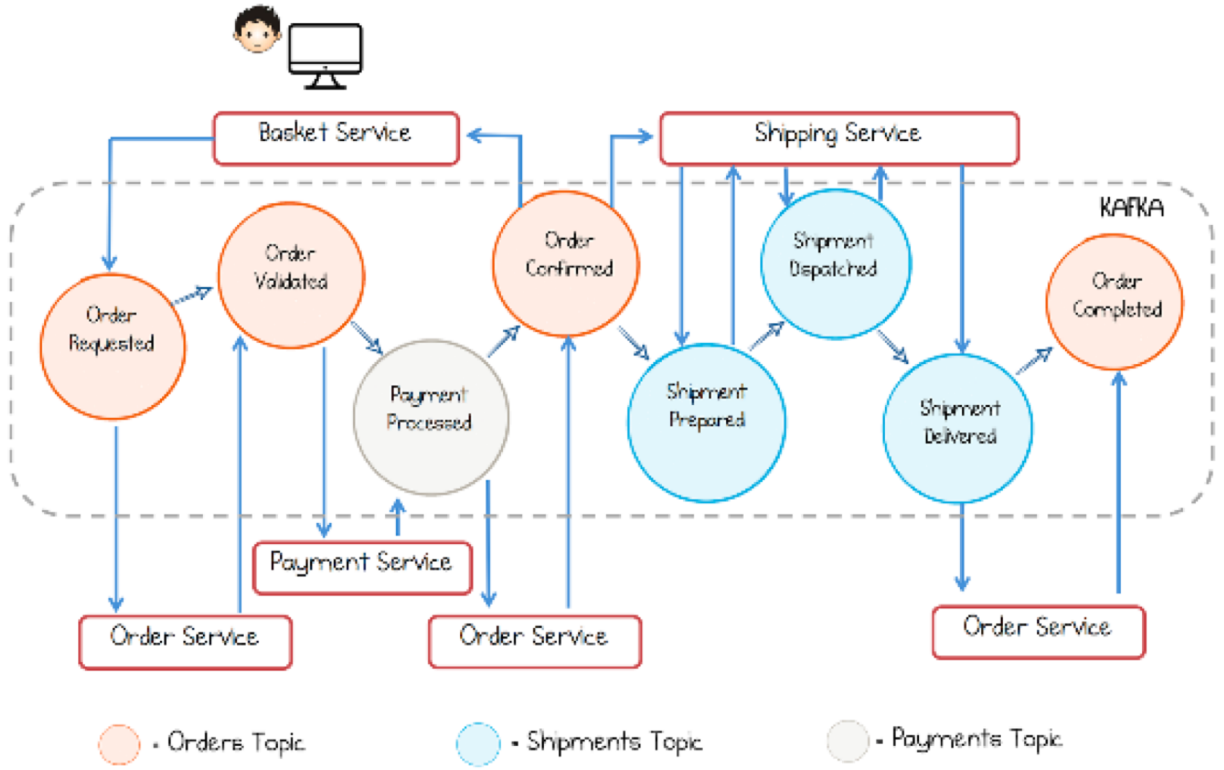


13 EDA - EVENT FIRST PATTERN

Taxonomie eventů
Eventy se snažíme uspořádat do taxonomie, která definuje typovou hierarchii. Toto typovou hierarchii pak používáme pro subscribování komponent (služeb) k určitým typům eventů



Orchestrace logiky pomocí eventů
V ekosystému pak vznikají eventy na které reagují komponenty (servis), které jsou k nim registrované. Akcí vznikají další eventy.



Pozn. Kafka je nástroj pro integraci a procesování eventů, Topic je její mechanismus pro subskripci k typům eventů a publikování eventů ostatním