

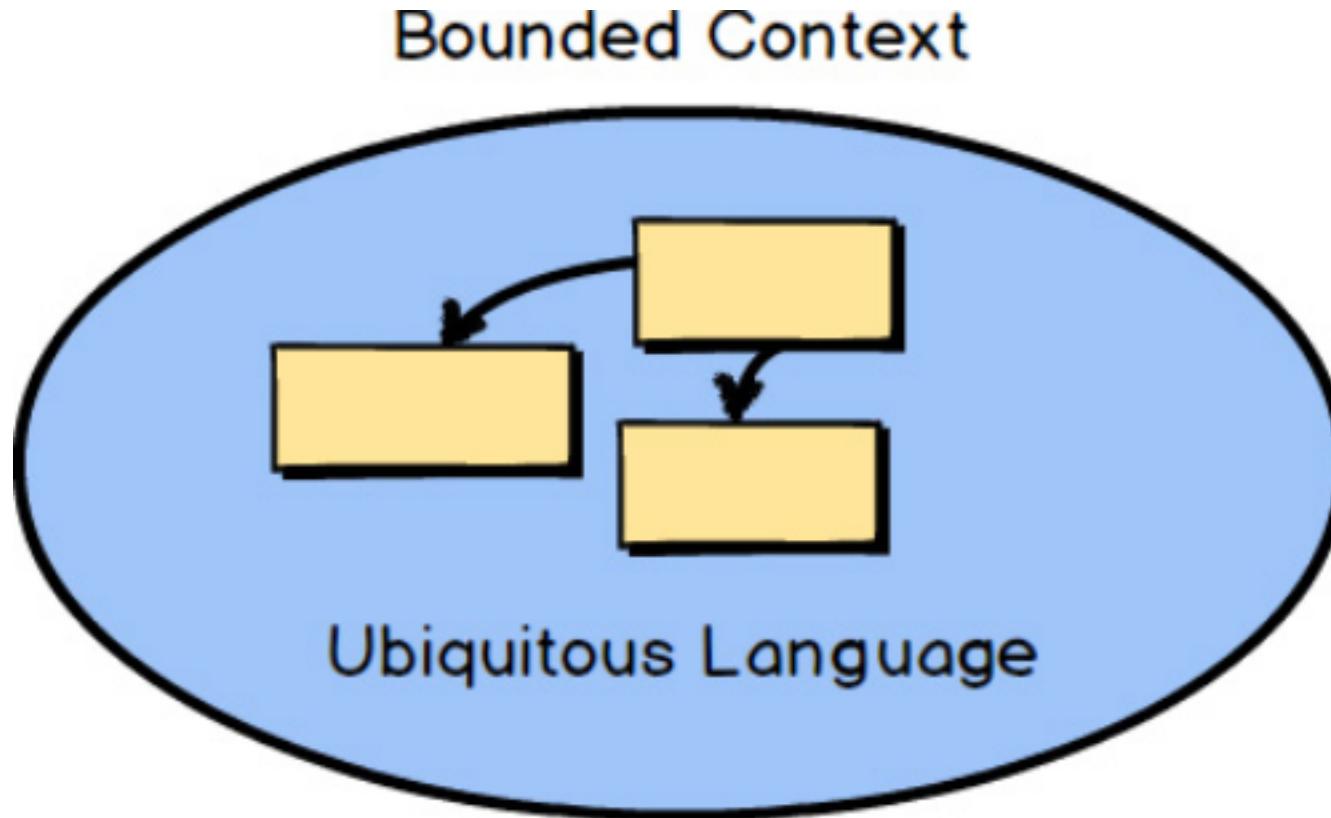
bns it } #} DDD Distilled



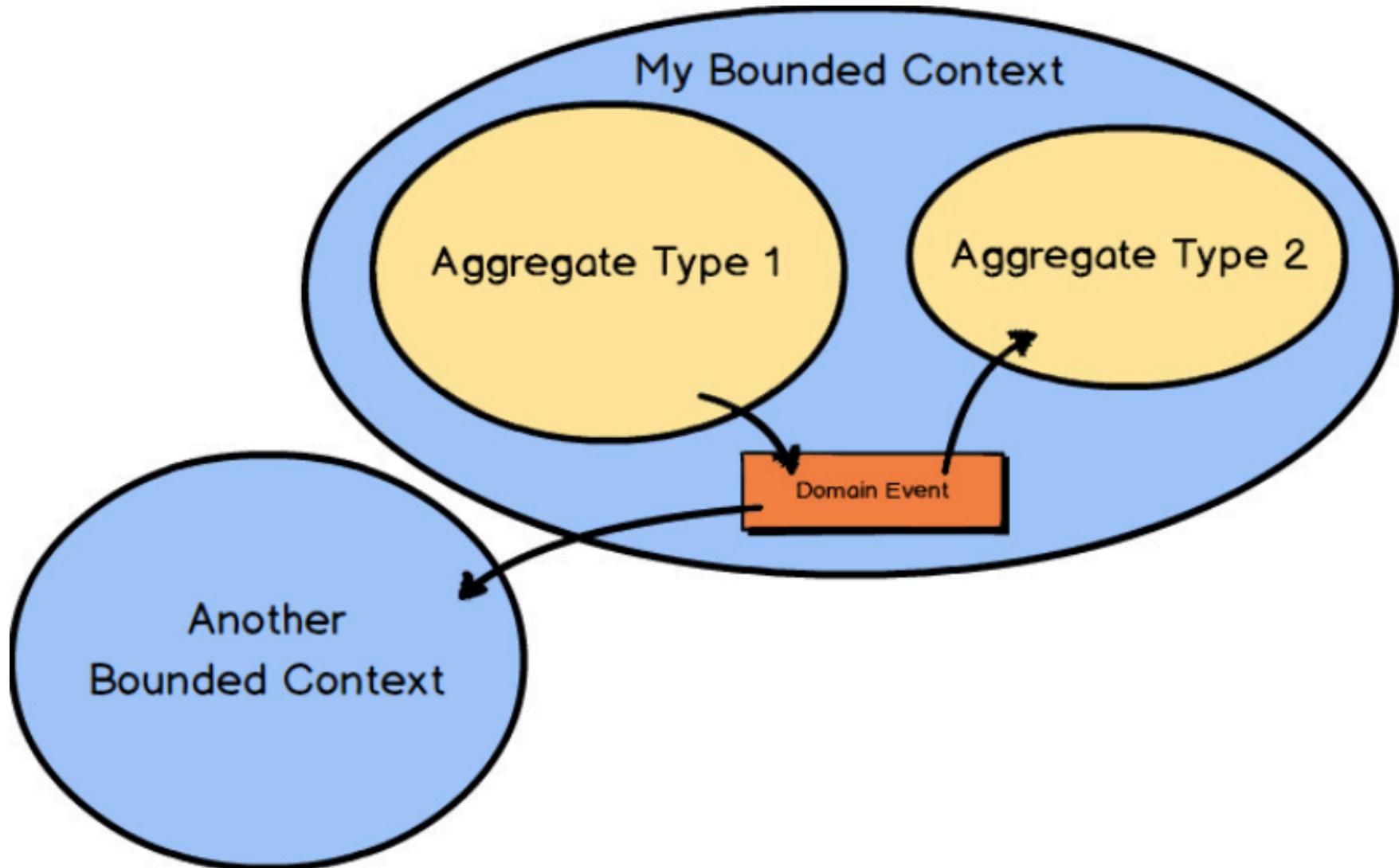
- # Software development is considered a cost center rather than a profit center.
- # Developers are too wrapped up with technology.
- # The database is given too much priority.

- # Developers don't give proper emphasis to naming objects.
- # Often the business stakeholders spend too much time in isolated work.
- # Project estimates are in high demand.

- # There are broken, slow, and locking database queries that block users from performing time-sensitive business operations.
- # Developers attempt to address all current and imagined future needs.
- # There are strongly coupled services, direct calls to another.



Strategic design – Context Maps



- # Learning process is a matter of discovery through group conversation and experimentation.



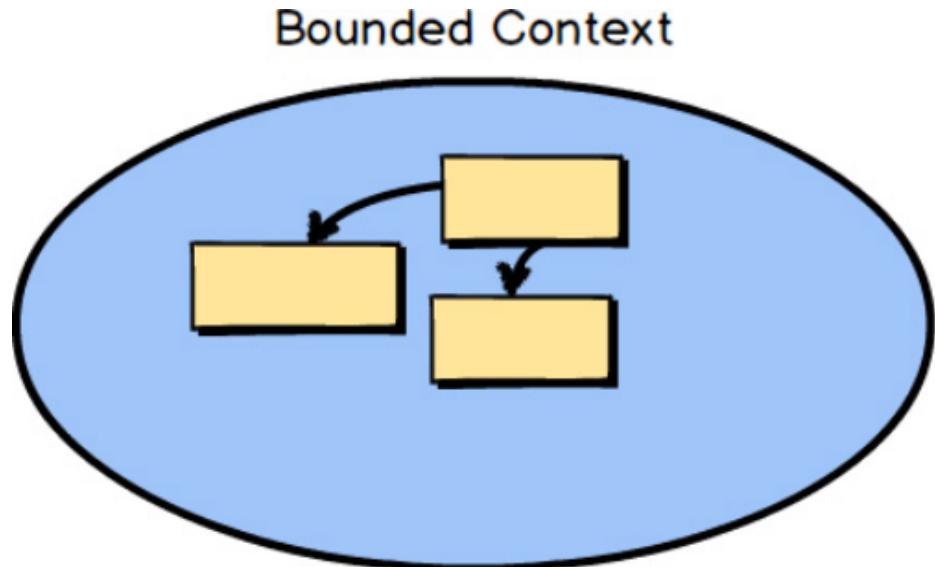
- # Aggregates
- # Entities
- # Factories
- # Policies
- # Value Objects
- # Services
- # Specifications



Bounded Contexts and Ubiquitous Language

bns it } # }

- # Language developed by the team
- # Working in the Bounded Context
- # Spoken by every member of the team
- # Rigorous—strict, exact, stringent, and tight.

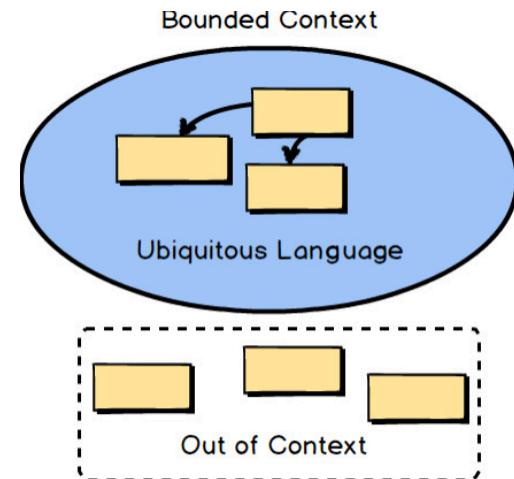


- # Core Domain is developed to distinguish your organization competitively from all others

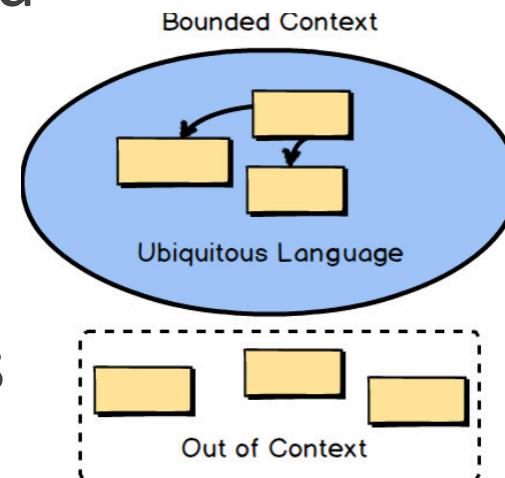
- # Commit your best resources to a Core Domain.

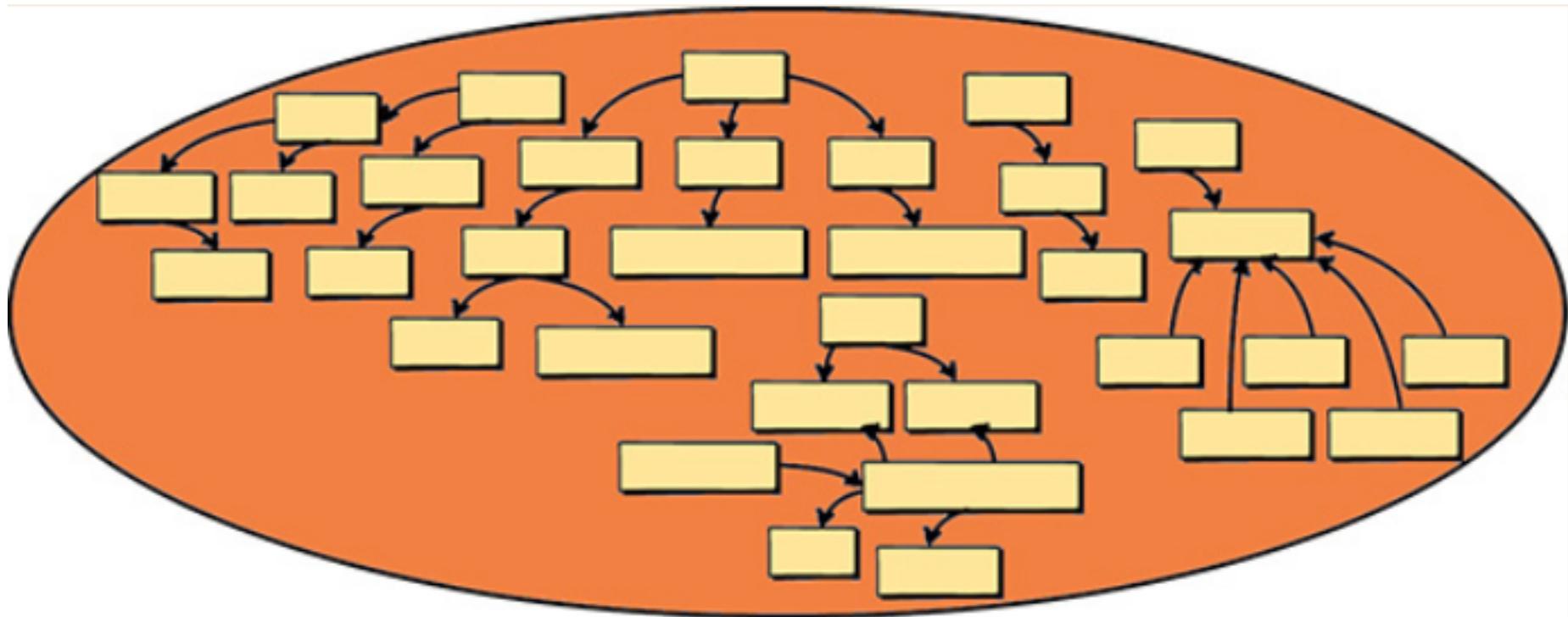
- # There should be one team assigned to work on one Bounded Context.

- # There should be a separate source code repository for each Bounded Context.



- # It is possible that one team could work on multiple Bounded Contexts, but multiple teams should not work on a single Bounded Context.
- # Cleanly separate the source code and database schema for each Bounded Context.
- # Keep acceptance tests and unit tests together with the main source code.

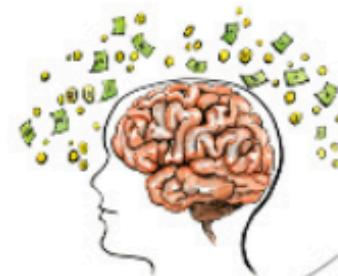




The same word – different meanings

Underwriting

Policy



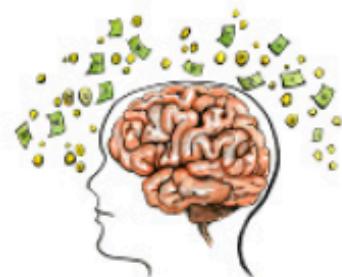
Inspections

Policy



Claims

Policy



Policy in Underwriting

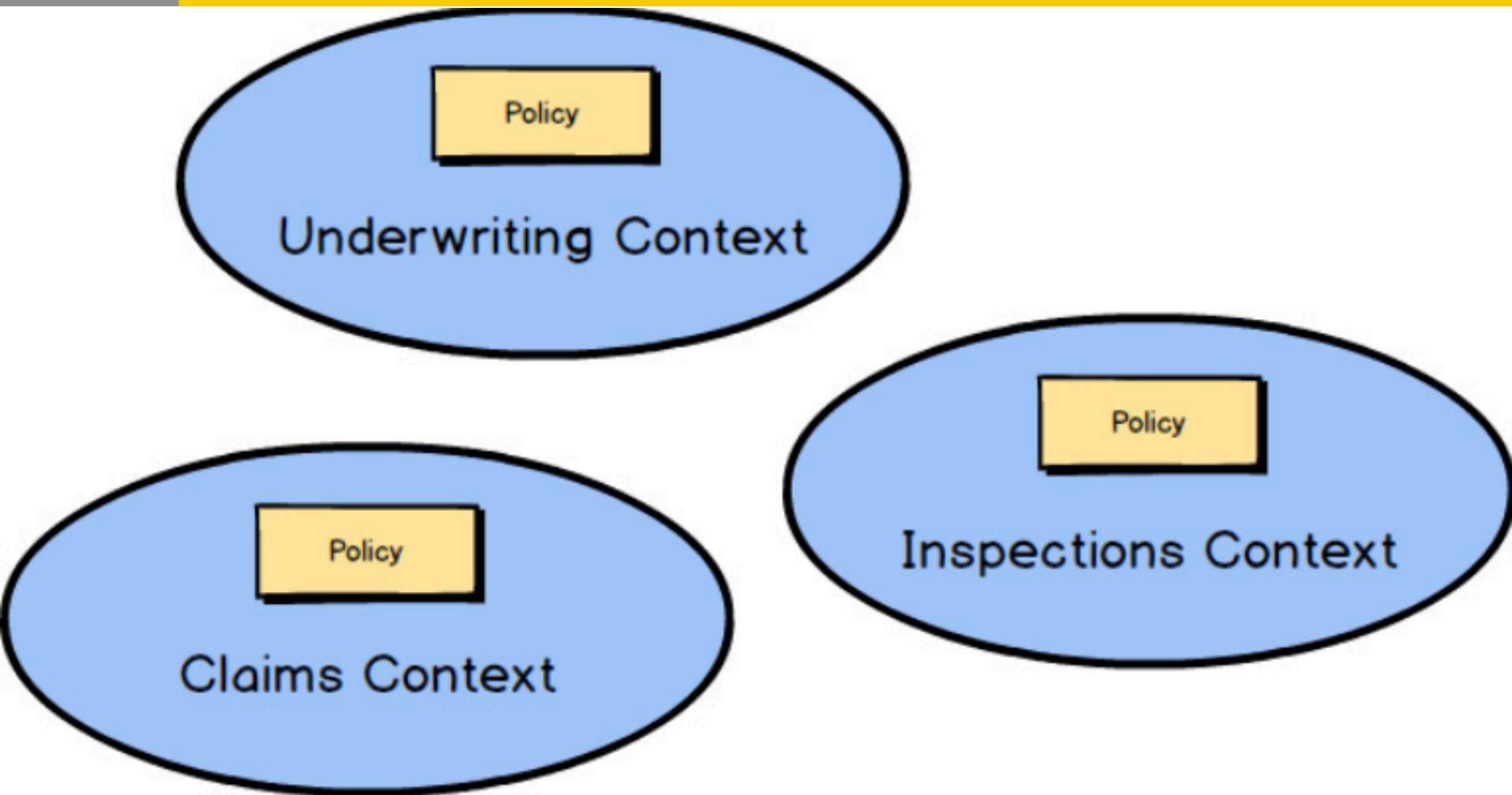
- based on the evaluation of the risks of the insured entity

Policy in Inspections

- based on inspecting a property that is to be insured, information found during inspections - photos and notes
- can be referenced by underwriting to negotiate the final cost

Policy in Claims

- tracks the request for payment
- focused on, for example, damages to the insured property and reviews performed by the claims personnel



Flight

- a single takeoff and landing, where the aircraft is flown from one airport to another
- flight that is defined in terms of aircraft maintenance
- flight that is defined in terms of passenger ticketing, either nonstop or one-stop

Different Bounded Contexts

#}

bns it} Different meanings

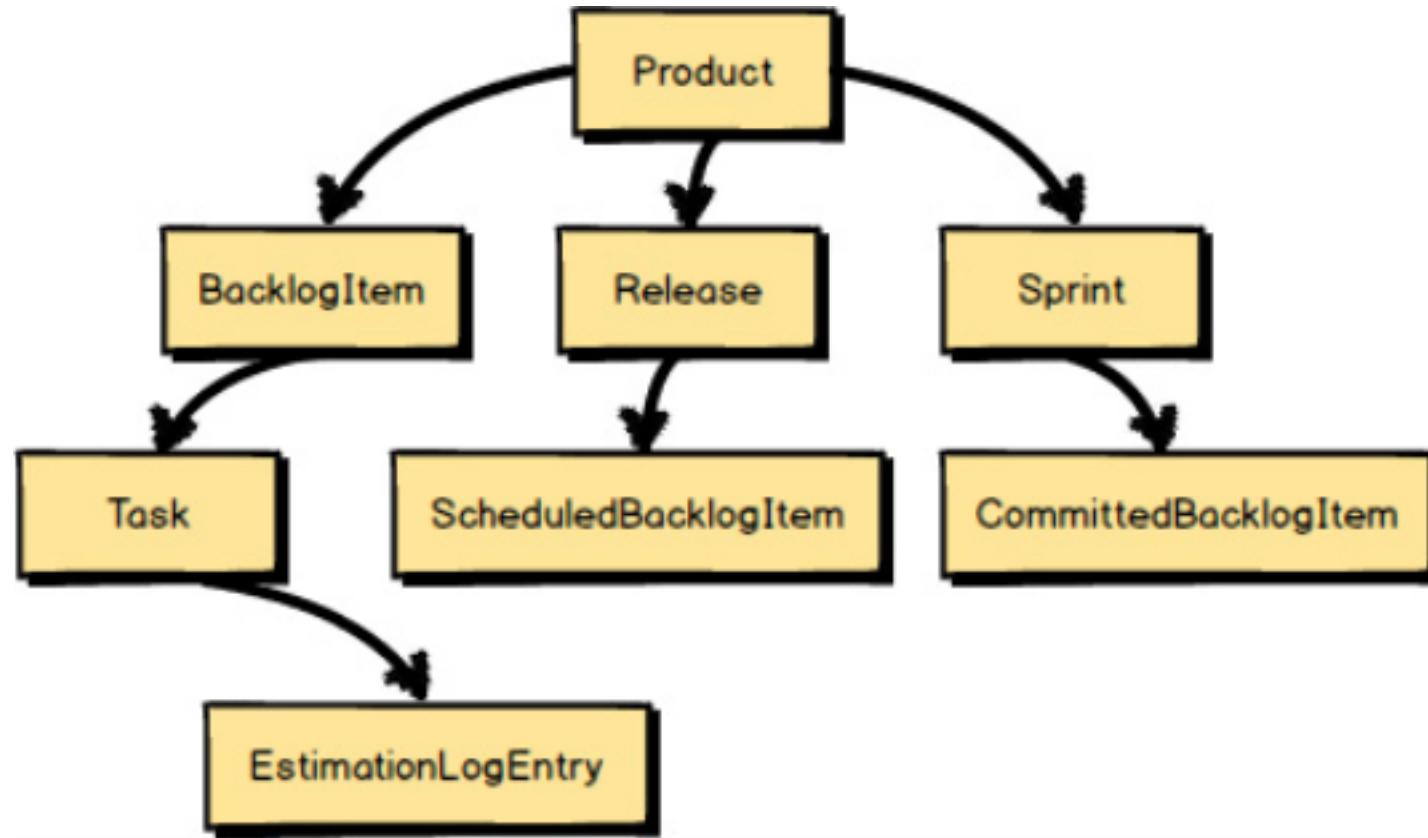
```
public class Lead
{
    public IEnumerable<Opportunity> Opportunities { get; }
    public Person Contact { get; }
}

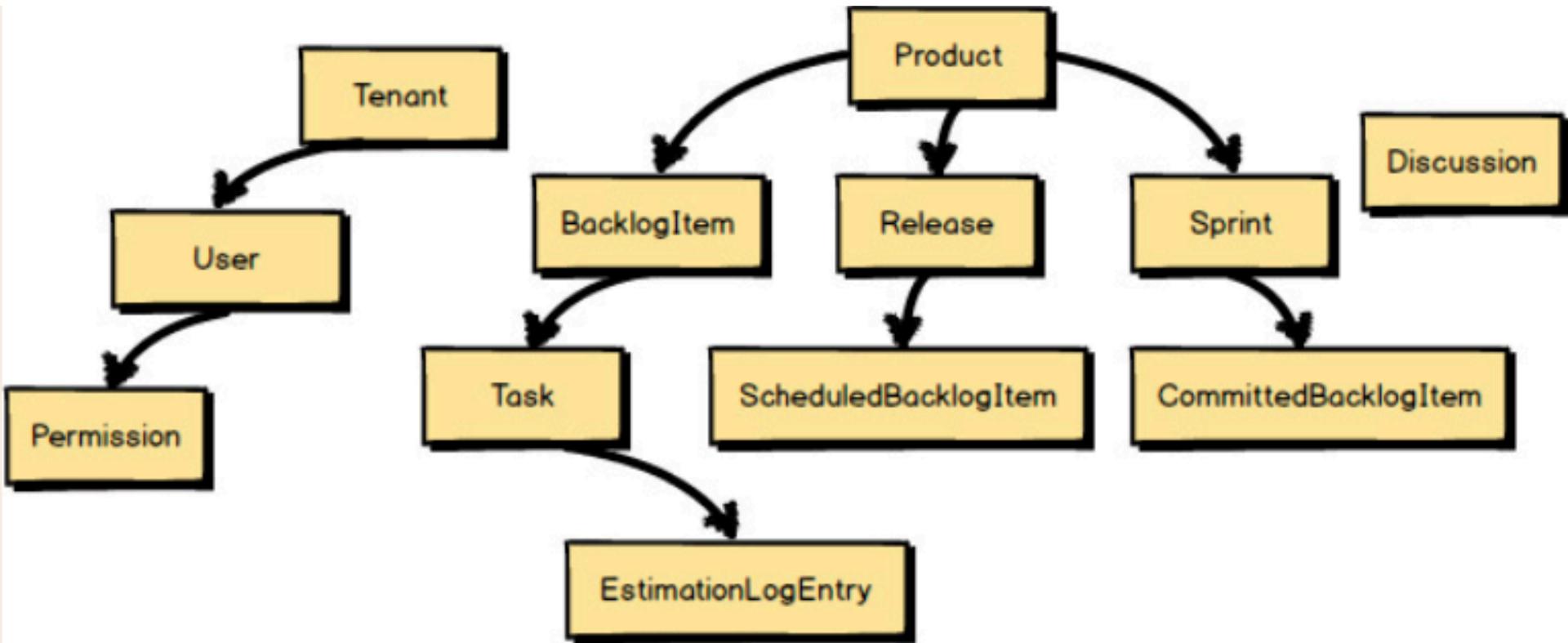
public class Client
{
    public IEnumerable<Invoice> GetOutstandingInvoices();
    public Address BillingAddress { get; }
    public IEnumerable<Order> PurchaseHistory { get; }
}

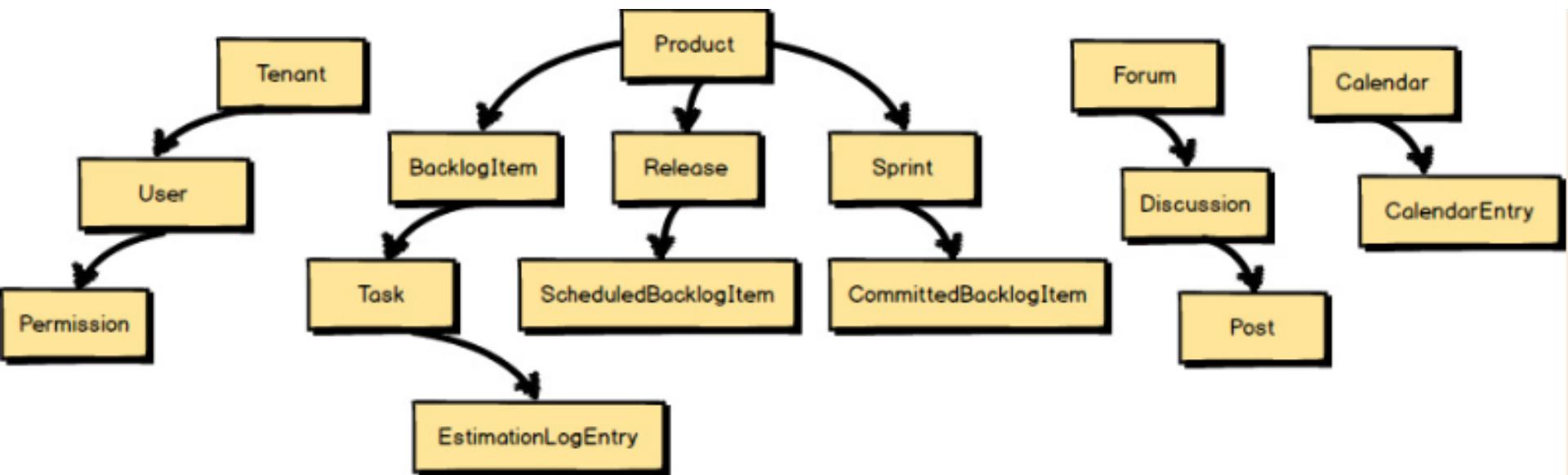
public class Customer
{
    public IEnumerable<Ticket> Tickets { get; }
}
```

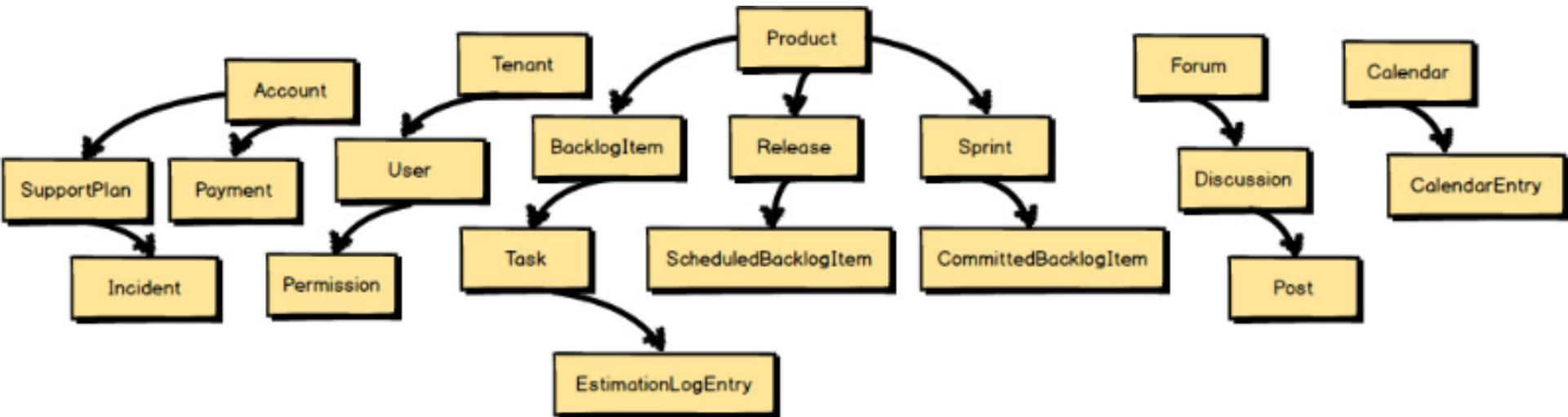
bns it} # Scrum Management Tool - Example

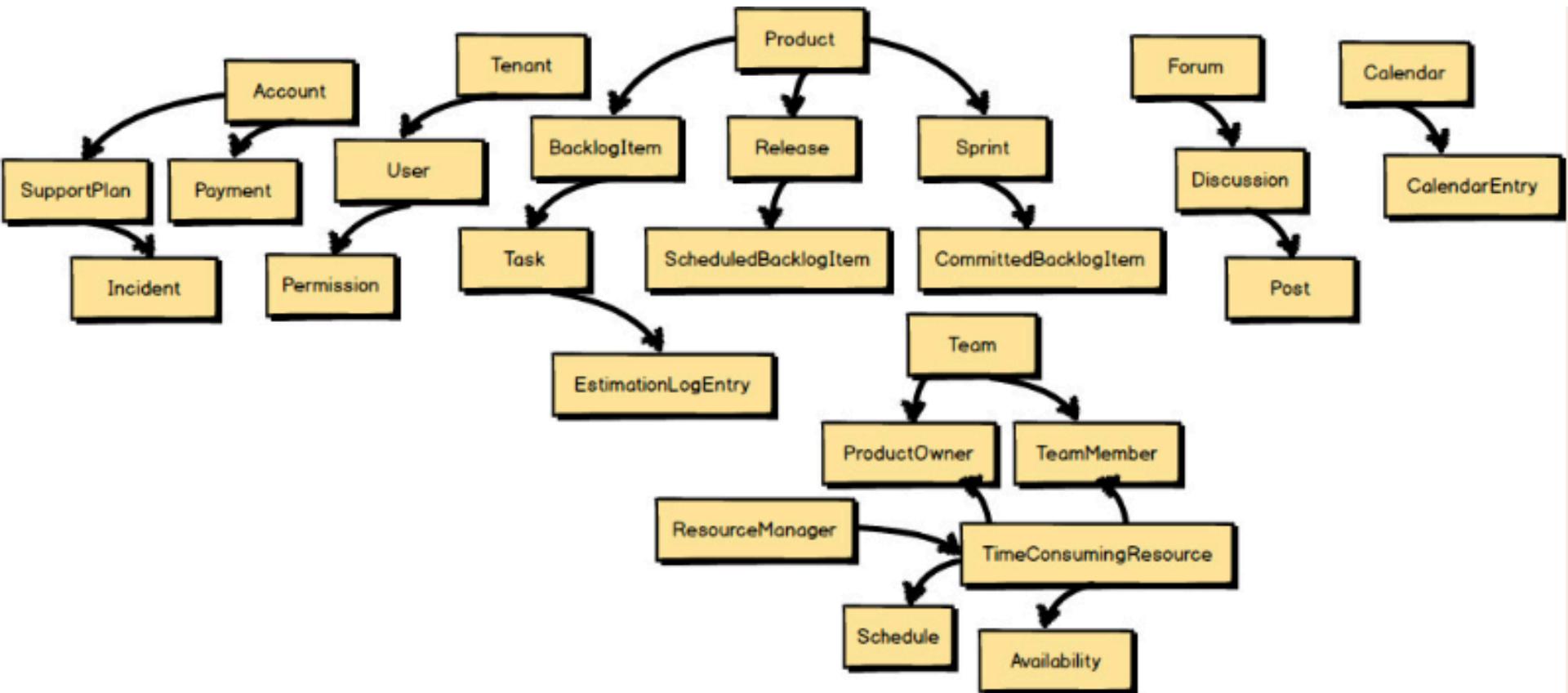
- # You can plan product development – features, releases, sprints
- # Available to many users and customers (SaaS)
- # Different Support Plans and Incident Management
- # Discussions through forums
- # Development event in calendar
- # Team management and resource planning

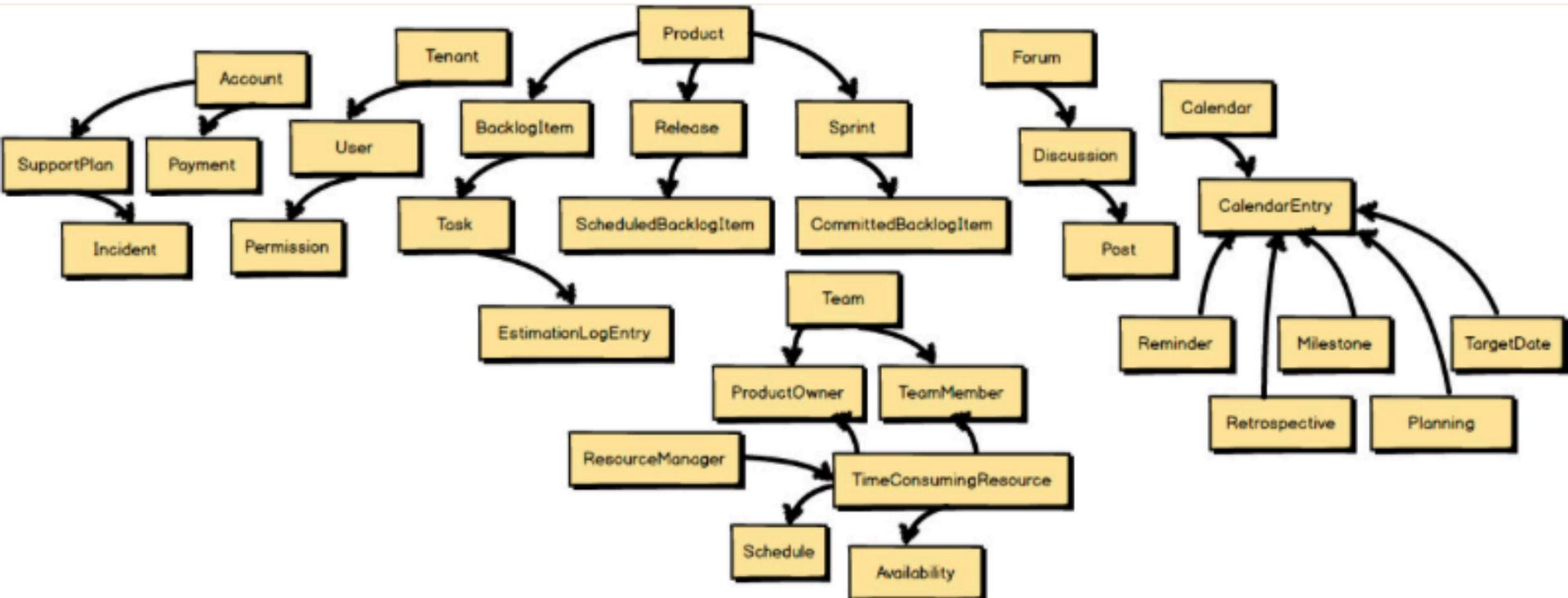


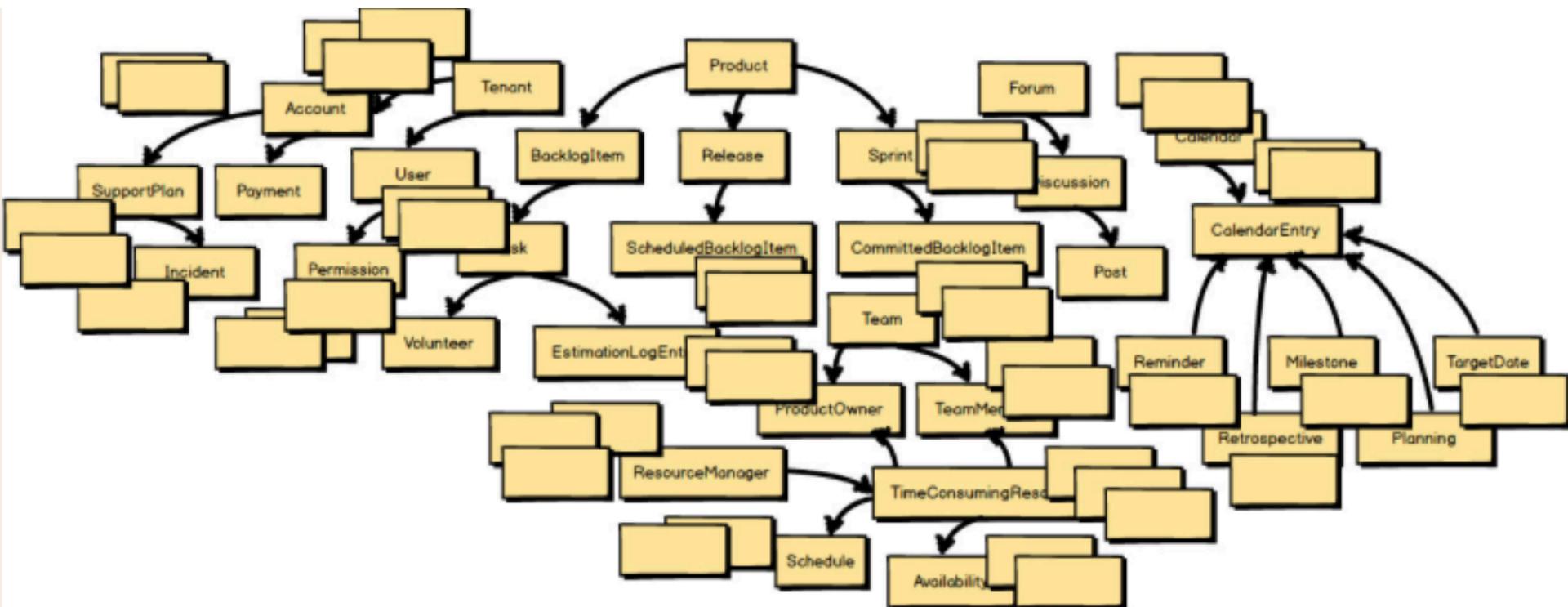






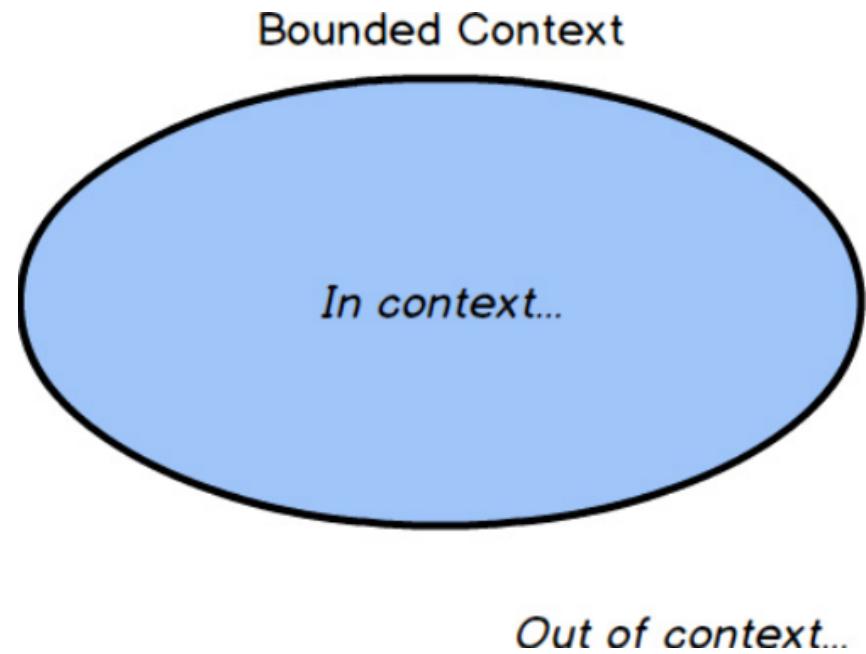
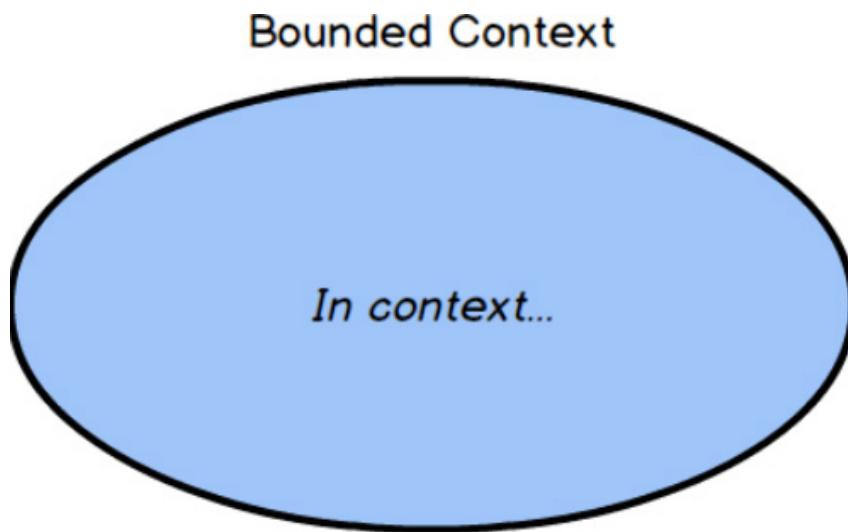






Where is the core?
} What are the Bounded Contexts?
bns it }

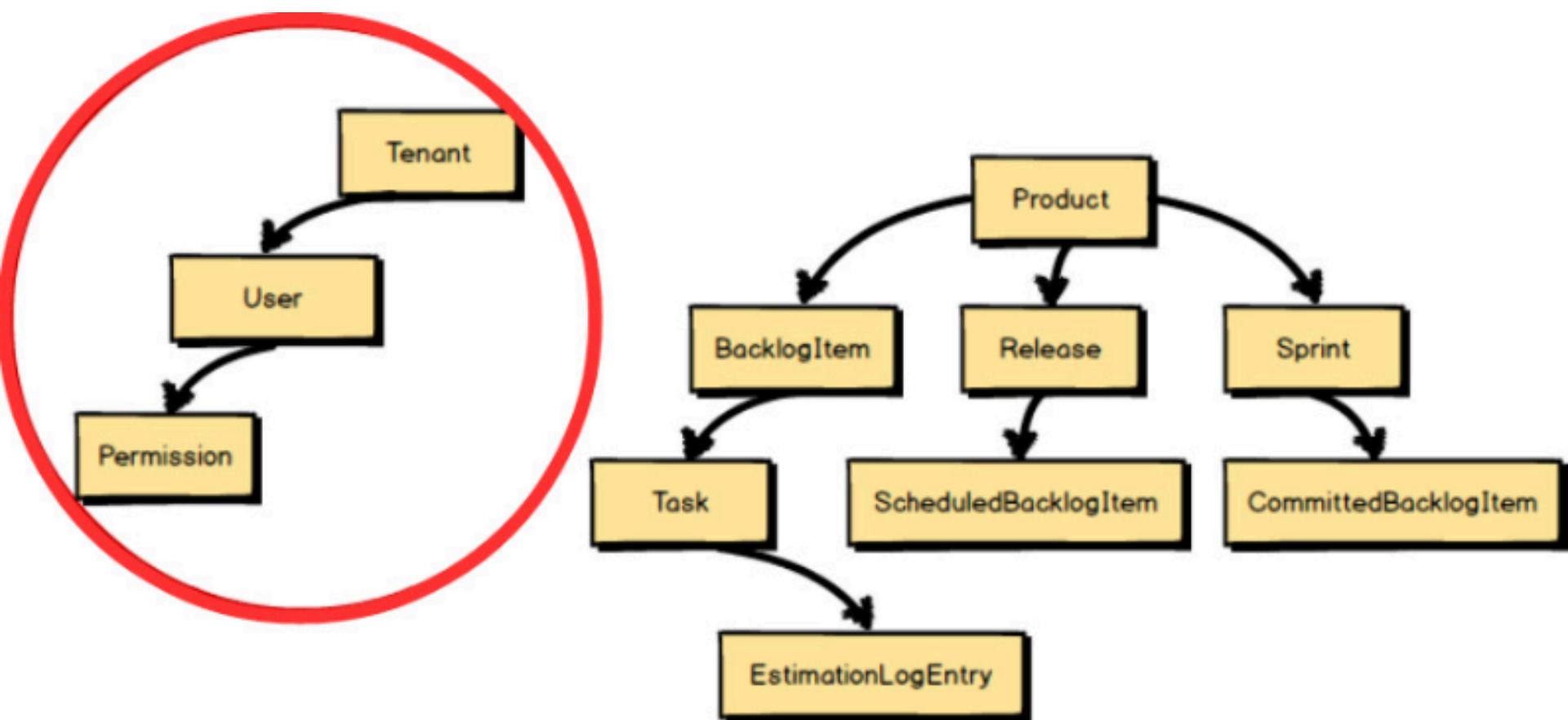
Bounded contexts definition

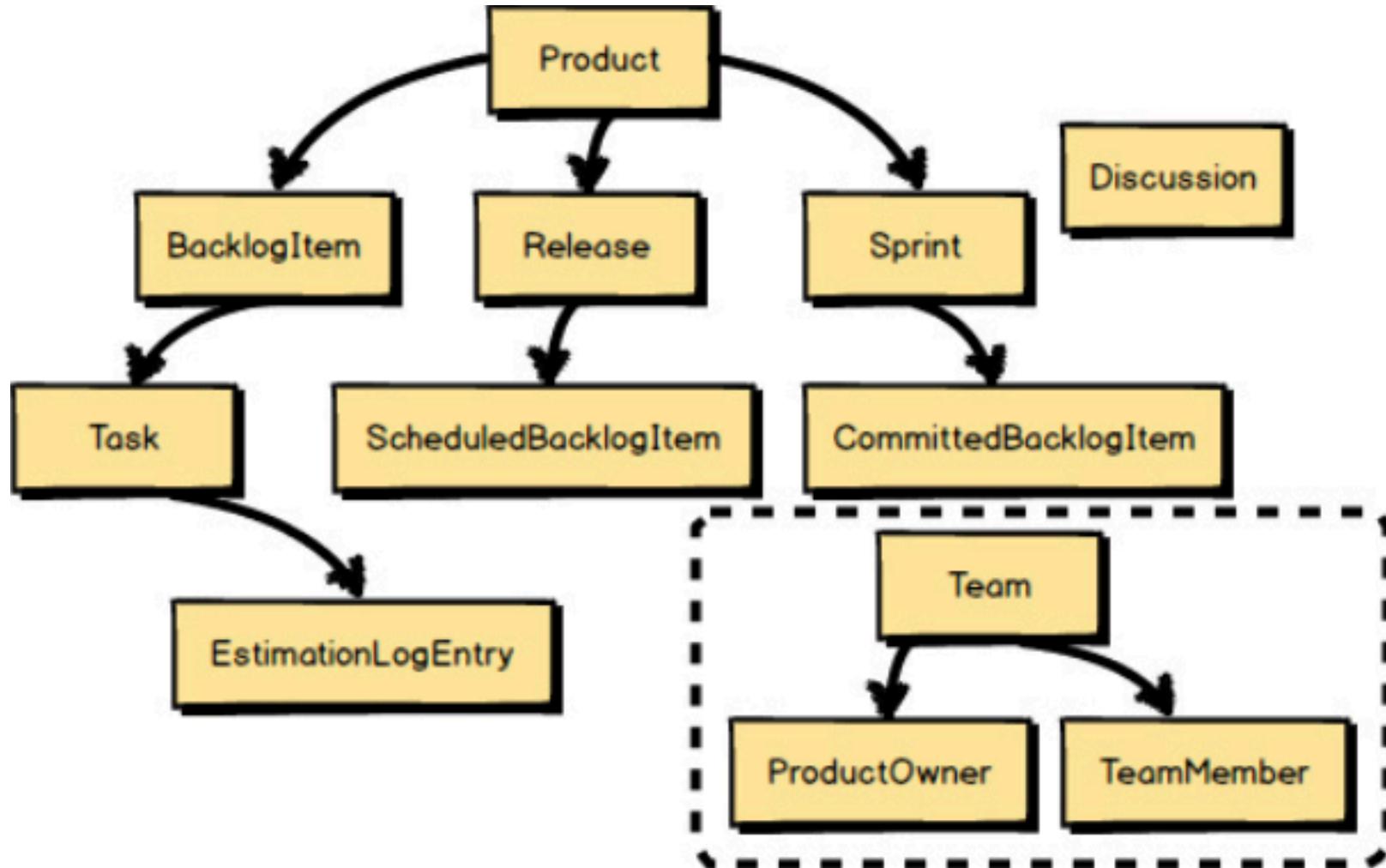


Product Owner or Domain Expert?

Focus on Business Complexity, Not Technical
Complexity

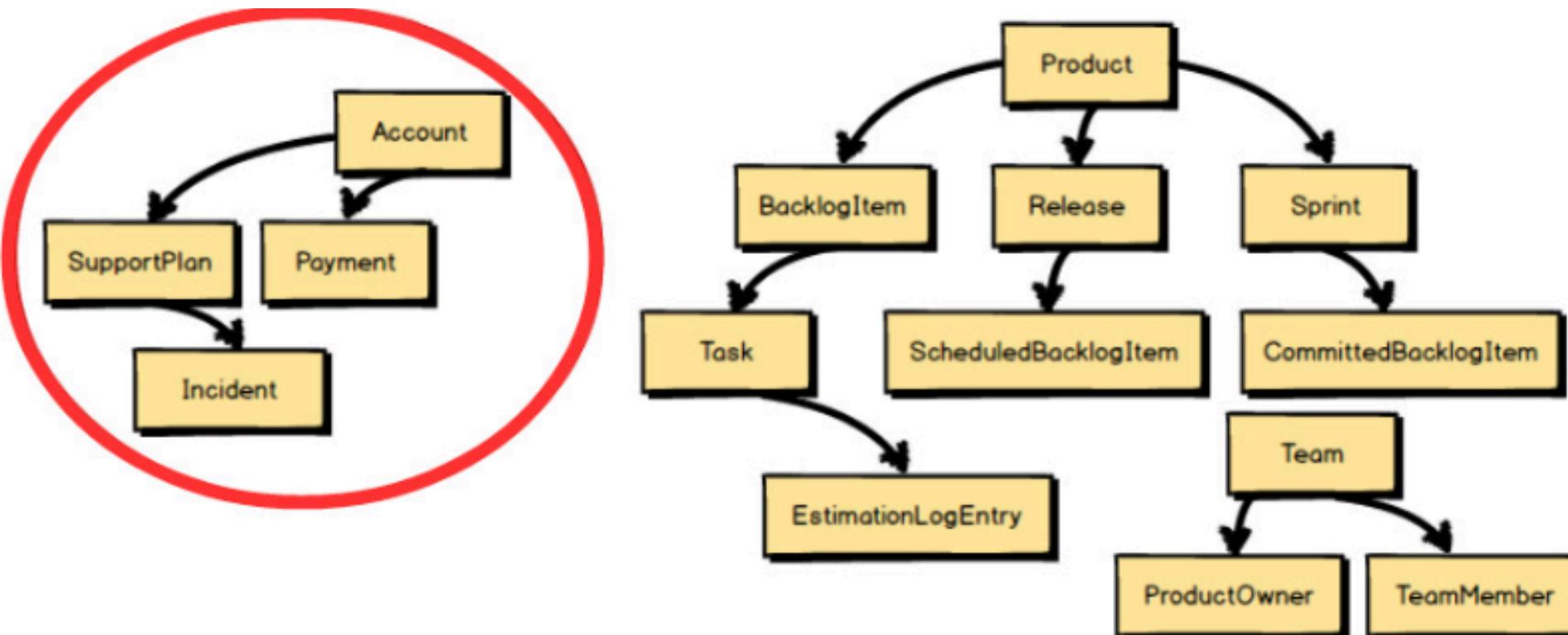
Where is the core?

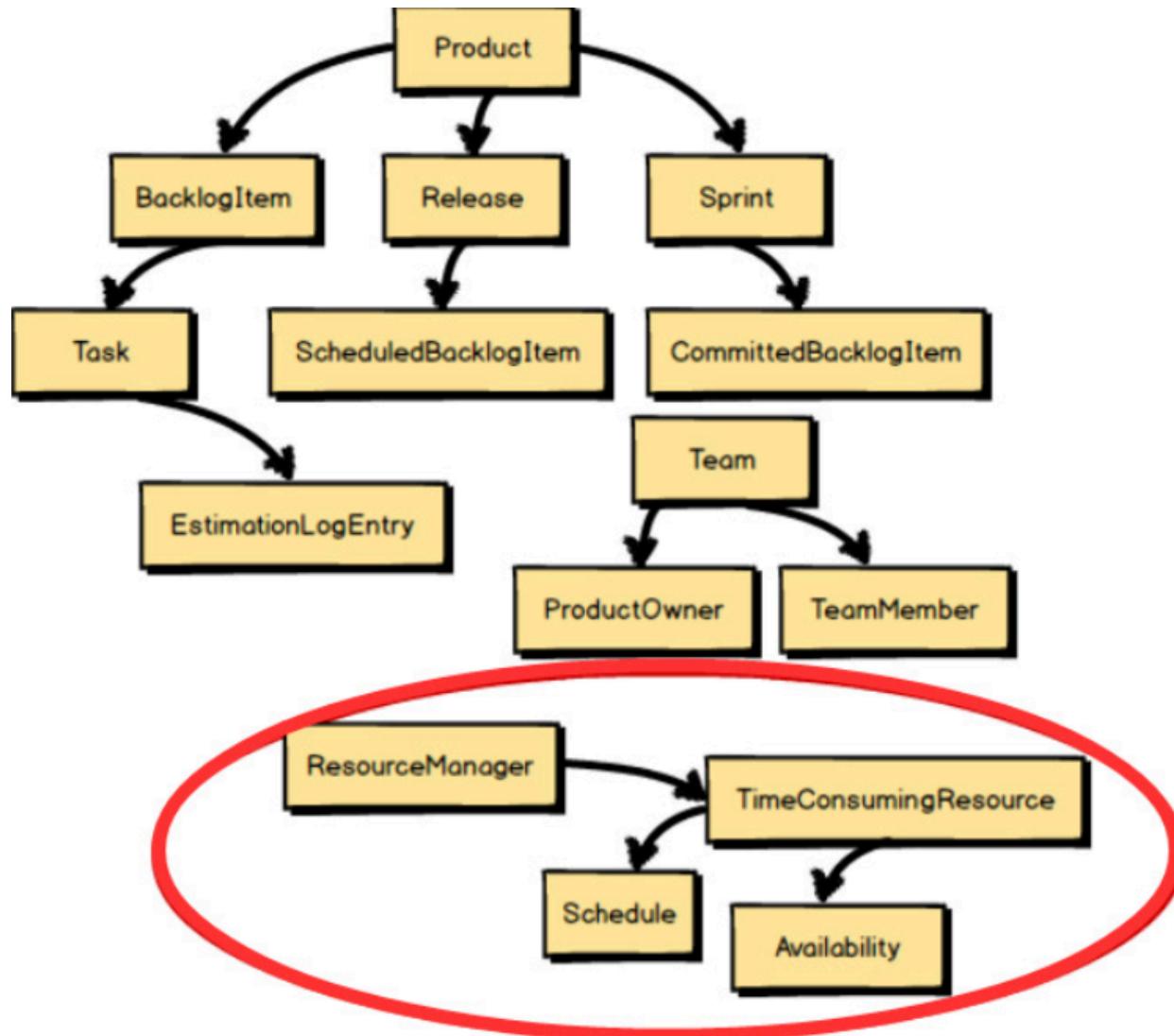


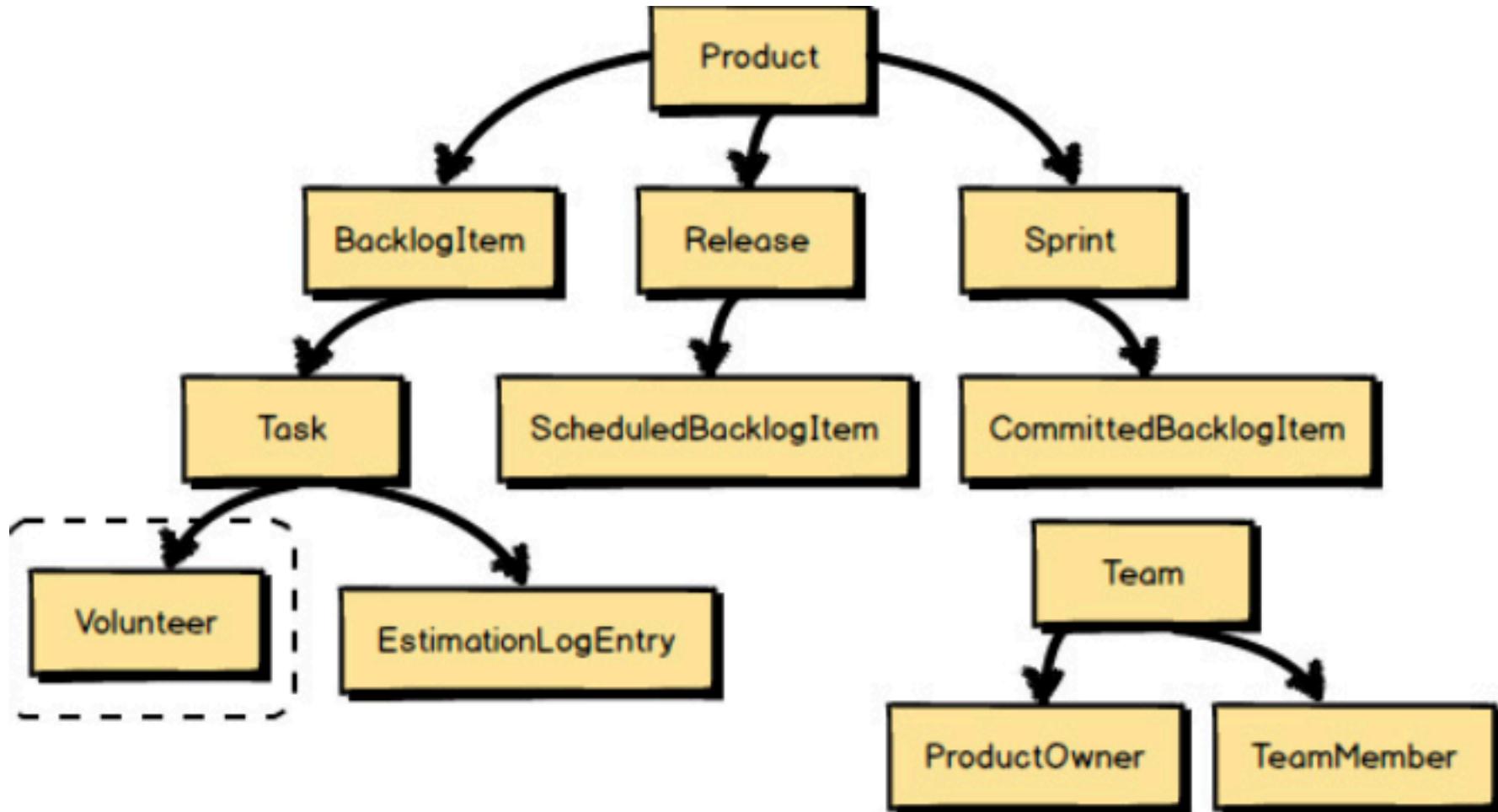


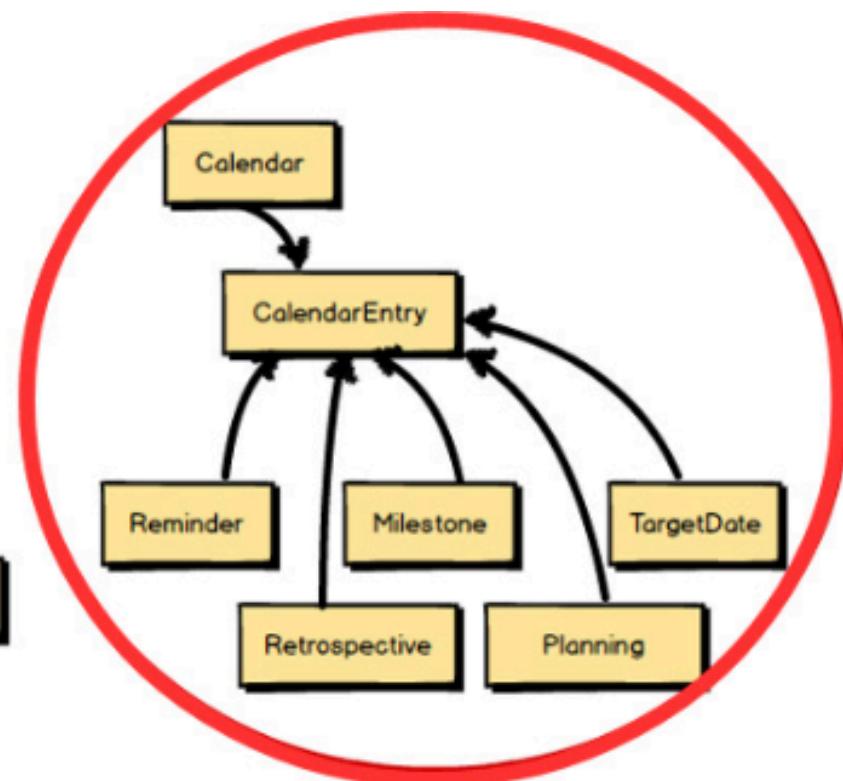
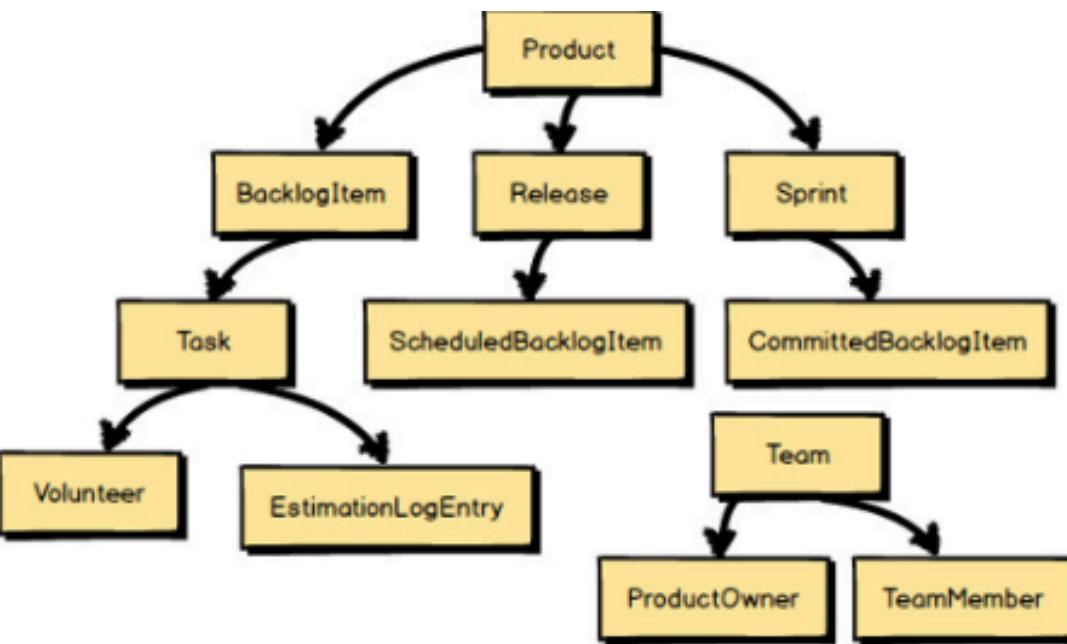
#}

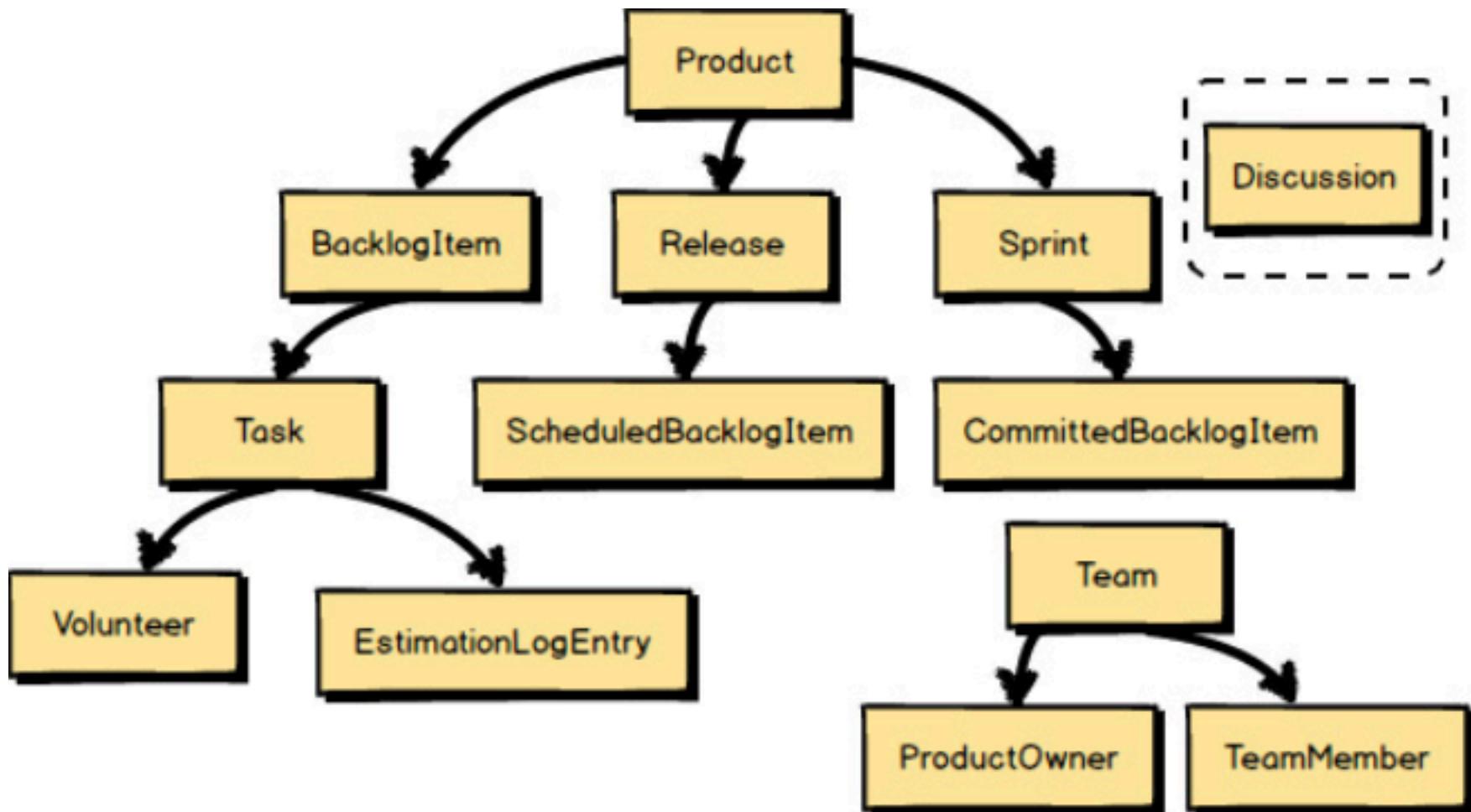
bns it} Where is the core?



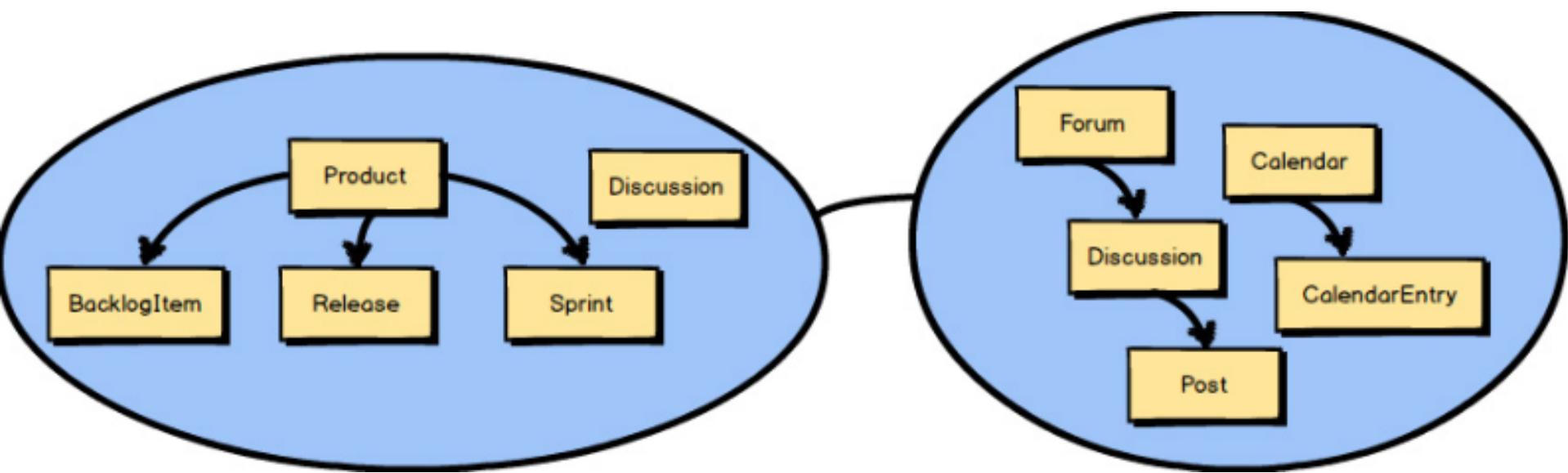


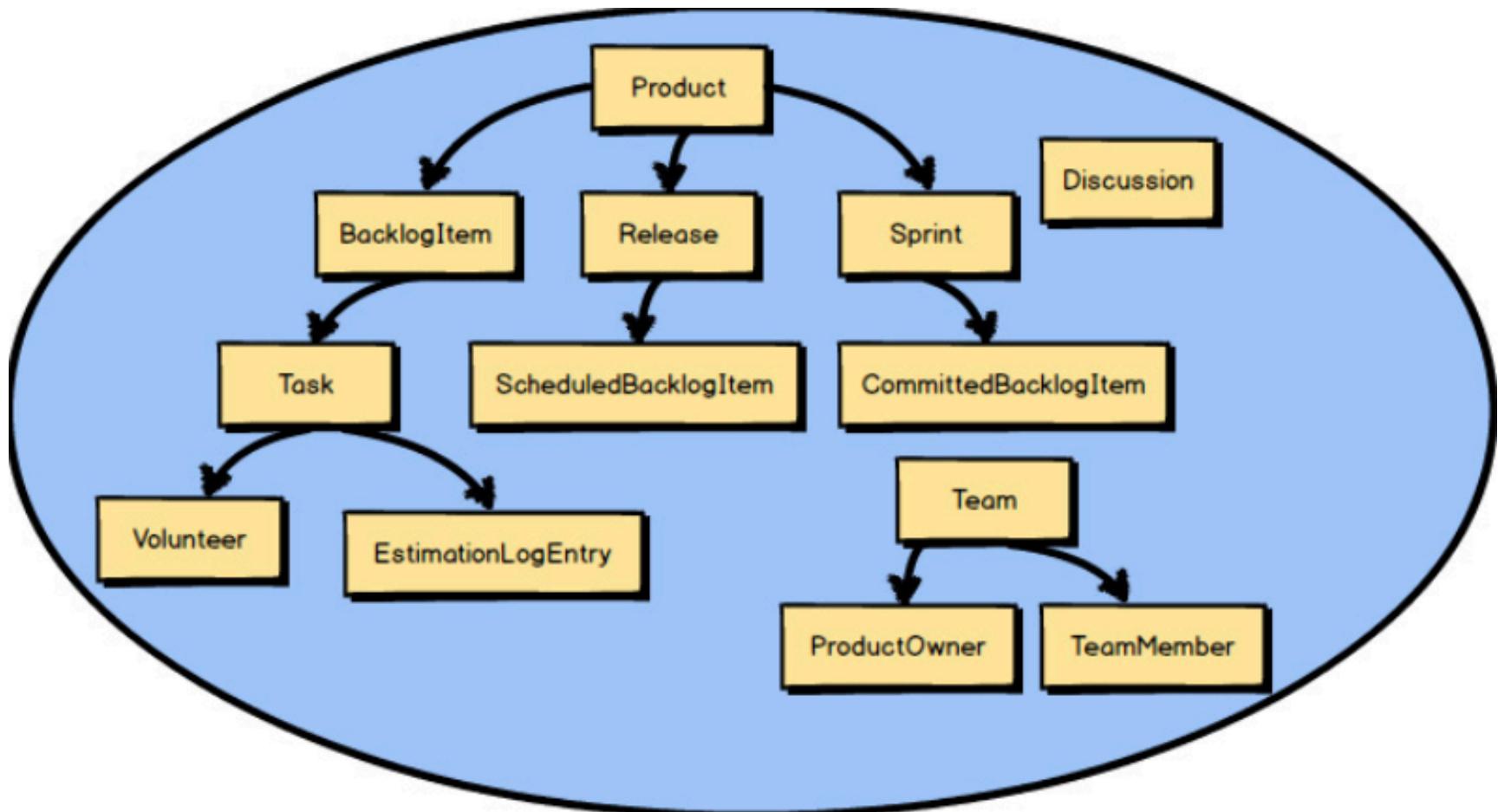


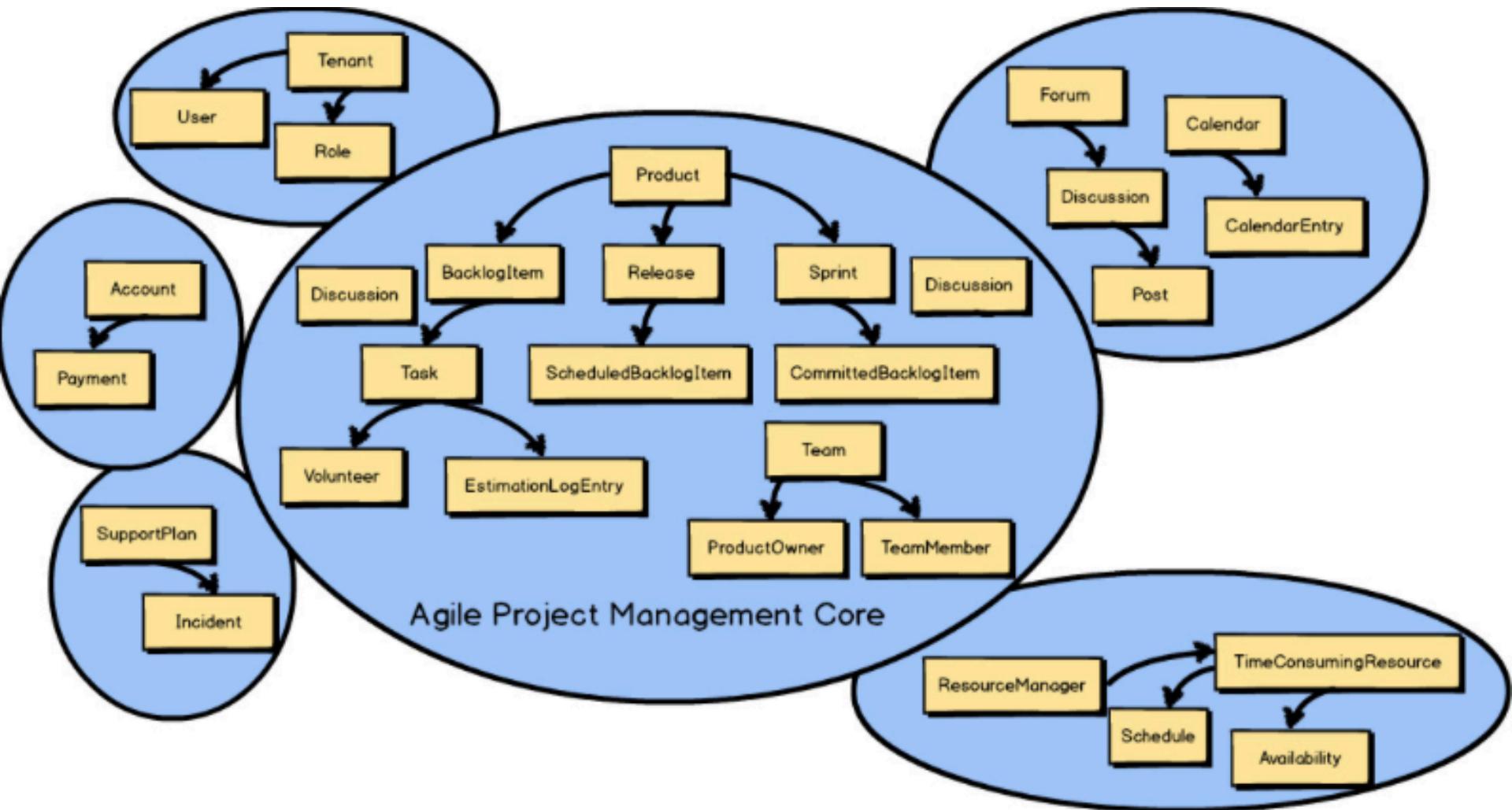




Where is the core?

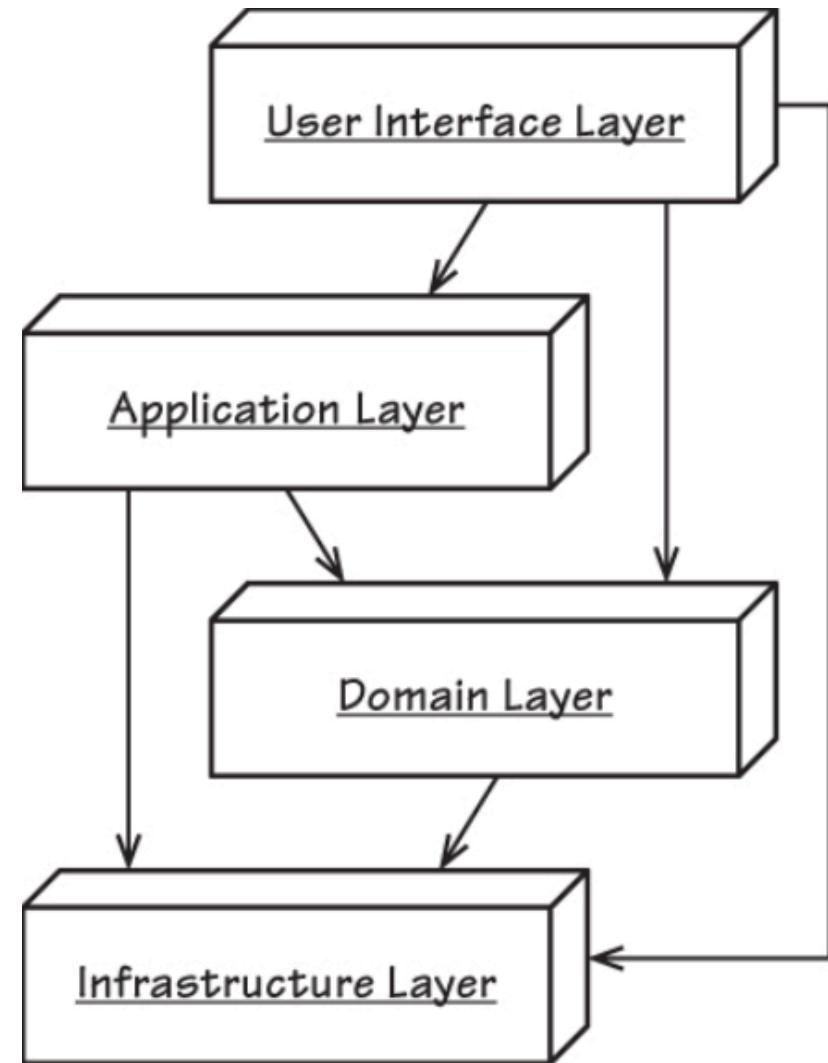




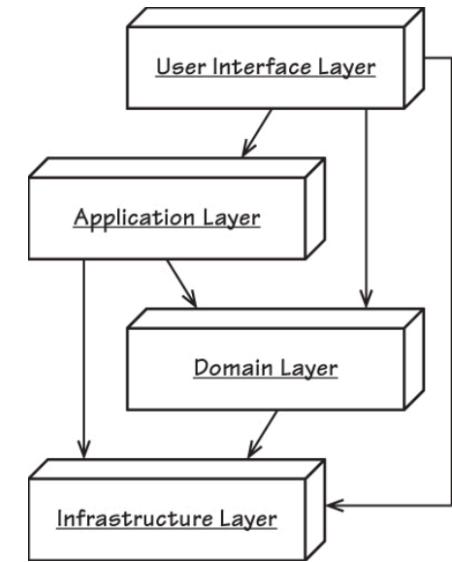


bns it } **#** Architecture

- # Strict Layers Architecture
- # Relaxed Layers Architecture
- # Communication backwards with Observer or Mediator pattern



- # Out of the scope od DDD
- # Any valid model like MVC, Presentation Model, MVP
- # Operations attributed to User Experience
- # UI validations, no business logic



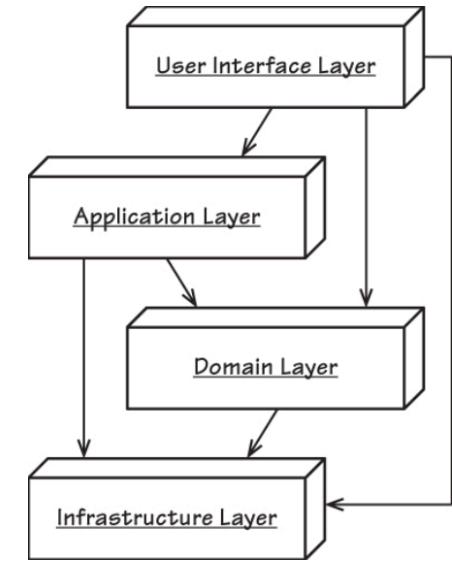
Application Services (not Domain Services)

Transactions and security

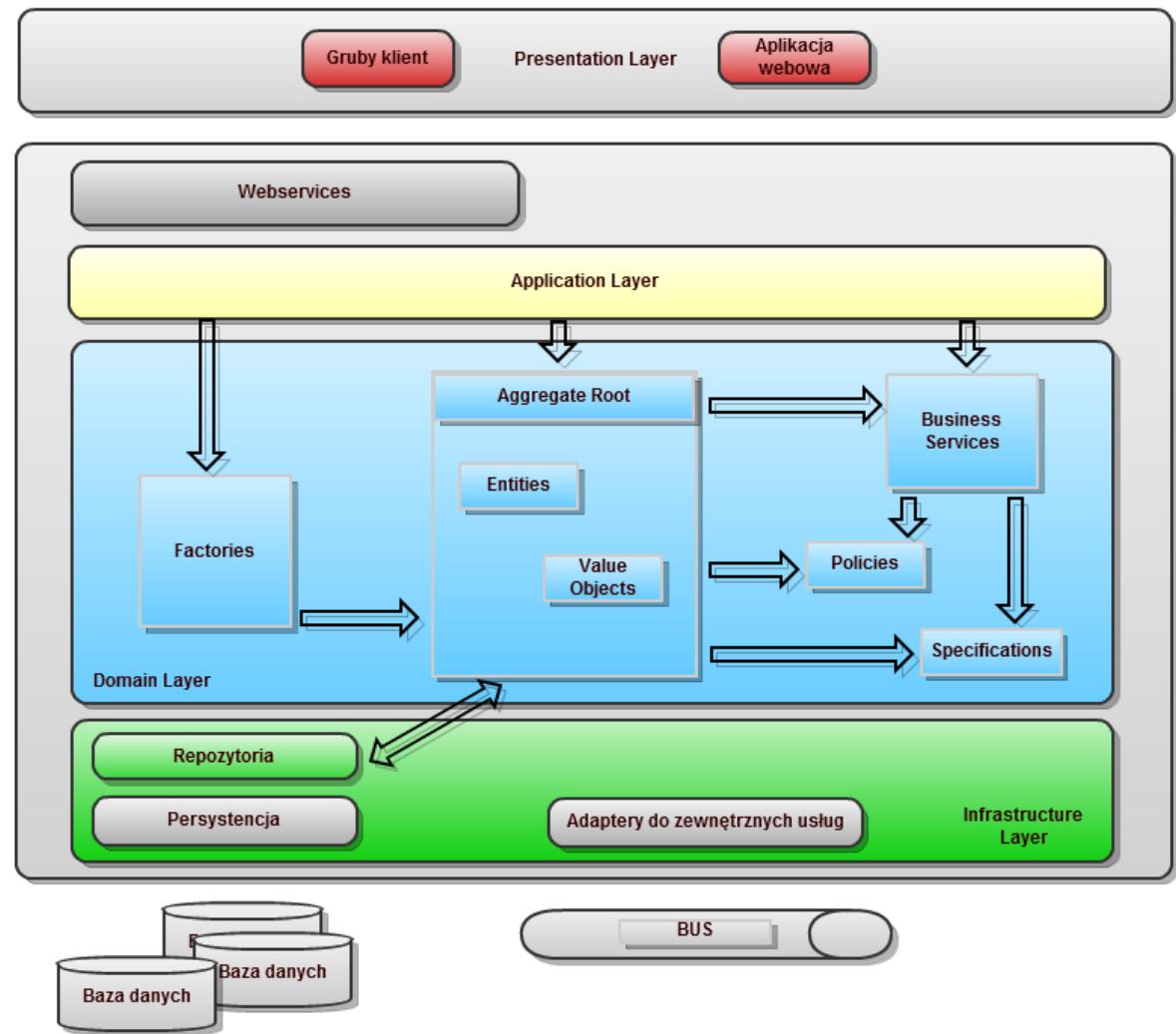
Sending event notifications or exposing services to the world

Use cases execution - orchestration

No domain logic



- # Events
- # Entities
- # Value Objects
- # Factory
- # Policy
- # Specification
- # Services



#}

bns it} Application service

```
@Transactional

public void commitBacklogItemToSprint(
    String aTenantId, String aBacklogItemId, String aSprintId) {

    TenantId tenantId = new TenantId(aTenantId);

    BacklogItem backlogItem =
        backlogItemRepository.backlogItemOfId(
            tenantId, new BacklogItemId(aBacklogItemId));

    Sprint sprint = sprintRepository.sprintOfId(
        tenantId, new SprintId(aSprintId));

    backlogItem.commitTo(sprint);

}
```

```
public class Product extends Entity {  
  
    private Set<ProductBacklogItem> backlogItems;  
    private String description;  
    private ProductDiscussion discussion;  
    private String discussionInitiationId;  
    private String name;  
    private ProductId productId;  
    private ProductOwnerId productOwnerId;  
    private TenantId tenantId;  
  
    public void initiateDiscussion(DiscussionDescriptor aDescriptor) {  
        public BacklogItem planBacklogItem(  
            BacklogItemId aNewBacklogItemId,  
            String aSummary,  
            String aCategory,  
            BacklogItemType aType,  
            StoryPoints aStoryPoints) {
```

```
public class Sprint extends Entity {

    private Set<CommittedBacklogItem> backlogItems;
    private Date begins;
    private Date ends;
    private String goals;
    private String name;
    private ProductId productId;
    private String retrospective;
    private SprintId sprintId;
    private TenantId tenantId;

    public void commit(BacklogItem aBacklogItem) {
    public void uncommit(BacklogItem aBacklogItem) {
    public void reorderFrom(BacklogItemId anId, int anOrderOfPriority)
    public void nowBeginsOn(Date aBegins) {
    public Date ends() {
    public String goals() {
```

#}

bns it} Repository

```
public interface ProductRepository {  
  
    public Collection<Product> allProductsOfTenant(TenantId aTenantId);  
  
    public ProductId nextIdentity();  
  
    public Product productOfDiscussionInitiationId(TenantId aTenantId,  
                                                String aDiscussionInitiationId);  
  
    public Product productOfId(TenantId aTenantId, ProductId aProductId);  
  
    public void remove(Product aProduct);  
  
    public void removeAll(Collection<Product> aProductCollection);  
  
    public void save(Product aProduct);  
  
    public void saveAll(Collection<Product> aProductCollection);  
}
```

```
public class Colour
{
    public int Red { get; private set; }
    public int Green { get; private set; }
    public int Blue { get; private set; }

    public Colour(int red, int green, int blue)
    {
        this.Red = red;
        this.Green = green;
        this.Blue = blue;
    }

    public Colour MixInTo(Colour other)
    {
        return new Colour(
            Math.Avg(this.Red, other.Red),
            Math.Avg(this.Green, other.Green),
            Math.Avg(this.Blue, other.Blue));
    }
}
```

#}

bns it} Value Object

```
public class BusinessPriorityRatings extends ValueObject {  
  
    private int benefit;  
    private int cost;  
    private int penalty;  
    private int risk;  
  
public final class Repetition extends AssertionConcern {  
  
    private Date ends;  
    private RepeatType repeats;  
  
public final class TimeSpan extends AssertionConcern {  
  
    private Date begins;  
    private Date ends;
```

```
public interface TripService
{
    float GetDrivingDistanceBetween(Location a, Location b);
}
```

```
public class AuthorizationService extends AssertionConcern {

    private GroupRepository groupRepository;
    private RoleRepository roleRepository;
    private UserRepository userRepository;
```

```
public boolean isUserInRole(TenantId aTenantId, String aUsername, String aRoleName)
```

```
public boolean isUserInRole(User aUser, String aRoleName) {
```

```
public boolean isUserInRole(User aUser, String aRoleName) {  
    this.assertArgumentNotNull(aUser, "User must not be null.");  
    this.assertArgumentNotEmpty(aRoleName, "Role name must not be null.");  
  
    boolean authorized = false;  
  
    if (aUser.isEnabled()) {  
        Role role = this.roleRepository().roleNamed(aUser.tenantId(), aRoleN  
  
        if (role != null) {  
            GroupMemberService groupMemberService =  
                new GroupMemberService(  
                    this.userRepository(),  
                    this.groupRepository());  
  
            authorized = role.isInRole(aUser, groupMemberService);  
        }  
    }  
  
    return authorized;  
}
```

```
class GoldCustomerSpecification : ISpecification<Customer>
{
    public bool IsSatisfiedBy(Customer candidate)
    {
        return candidate.TotalPurchases > 1000.0m;
    }
}

if (new GoldCustomerSpecification().IsSatisfiedBy(employee))
    // apply special discount
```

```
var spec = new PizzaSpecification()
    .BasedOn(new MargaritaPizzaSpecification())
    .WithThickCrust()
    .WithSwirl(Sauces.Bbq)
    .WithExtraCheese();

var pizza = new PizzaFactory().CreatePizzaFrom(spec);
```

Policy aka GoF Strategy

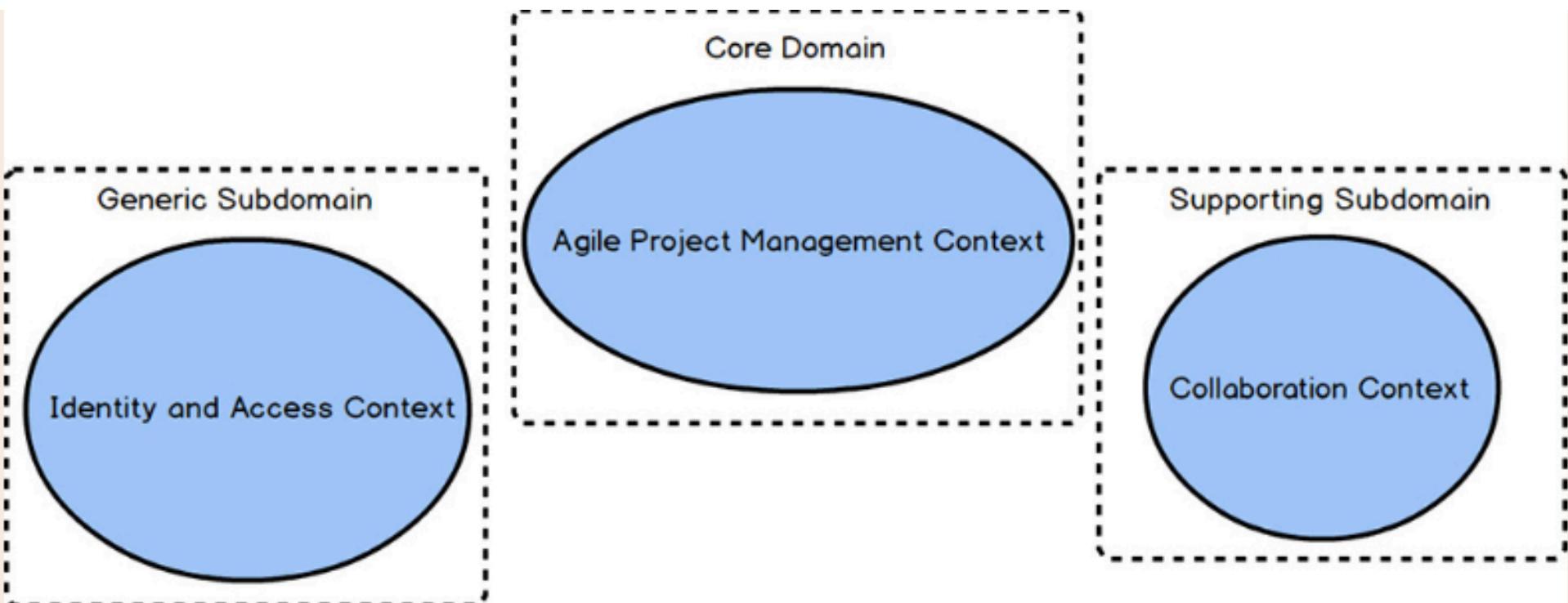
```
@DomainPolicy
public interface TaxPolicy {
    /**
     * calculates tax per product type based on net value
     */
    public Tax calculateTax(ProductType type, Money net);
```

bns it} #} Subdomain types

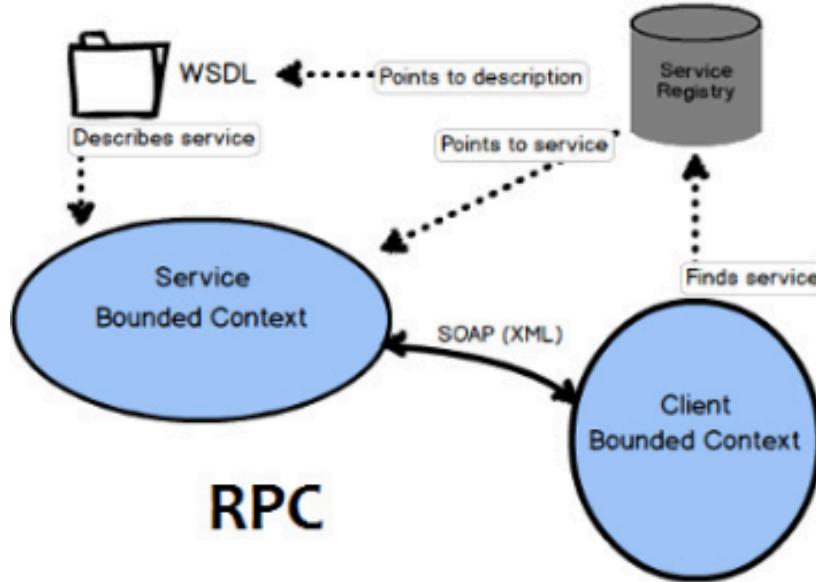
- # Where you are making a strategic investment
- # Highest priority
- # What your organization must excel at
- # How your organization will distinguish from others

- # Supports the core domain
- # Off-the-shelf solution doesn't exist
- # Not the most important one
- # Good to outsource

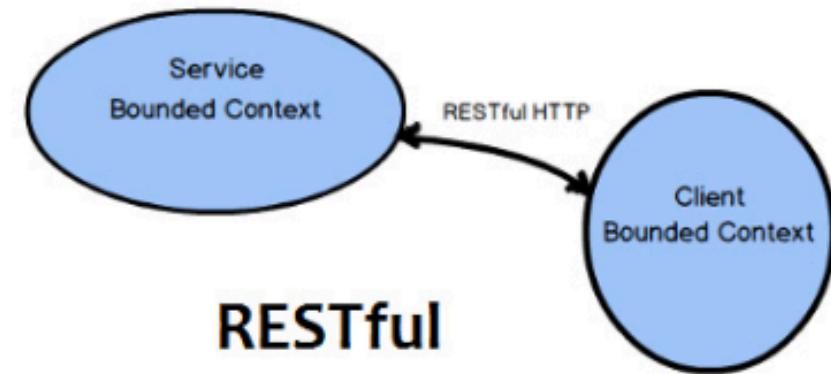
- # Necessary but generic
- # Avoid investing heavily in this area
- # Possible to buy off-the-shelf product
- # Don't dedicate best developers to it
- # Eg. Identity management, security



bns it} #} Integrating Bounded Contexts

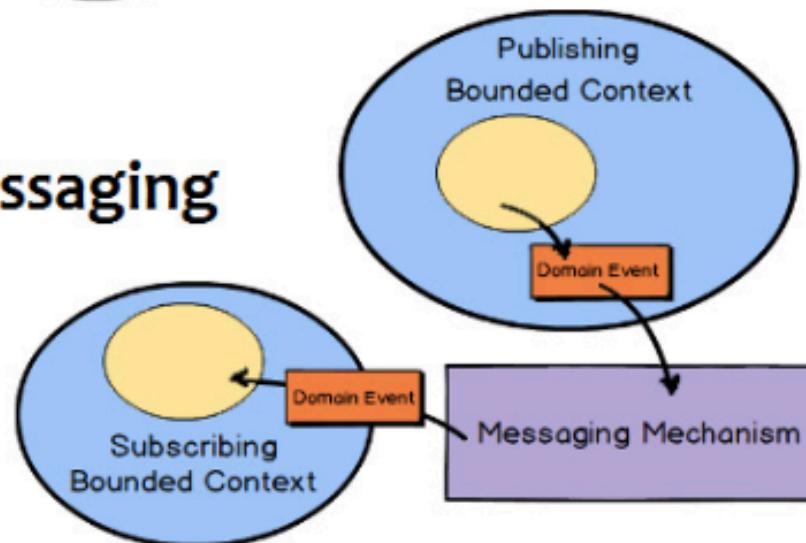


RPC

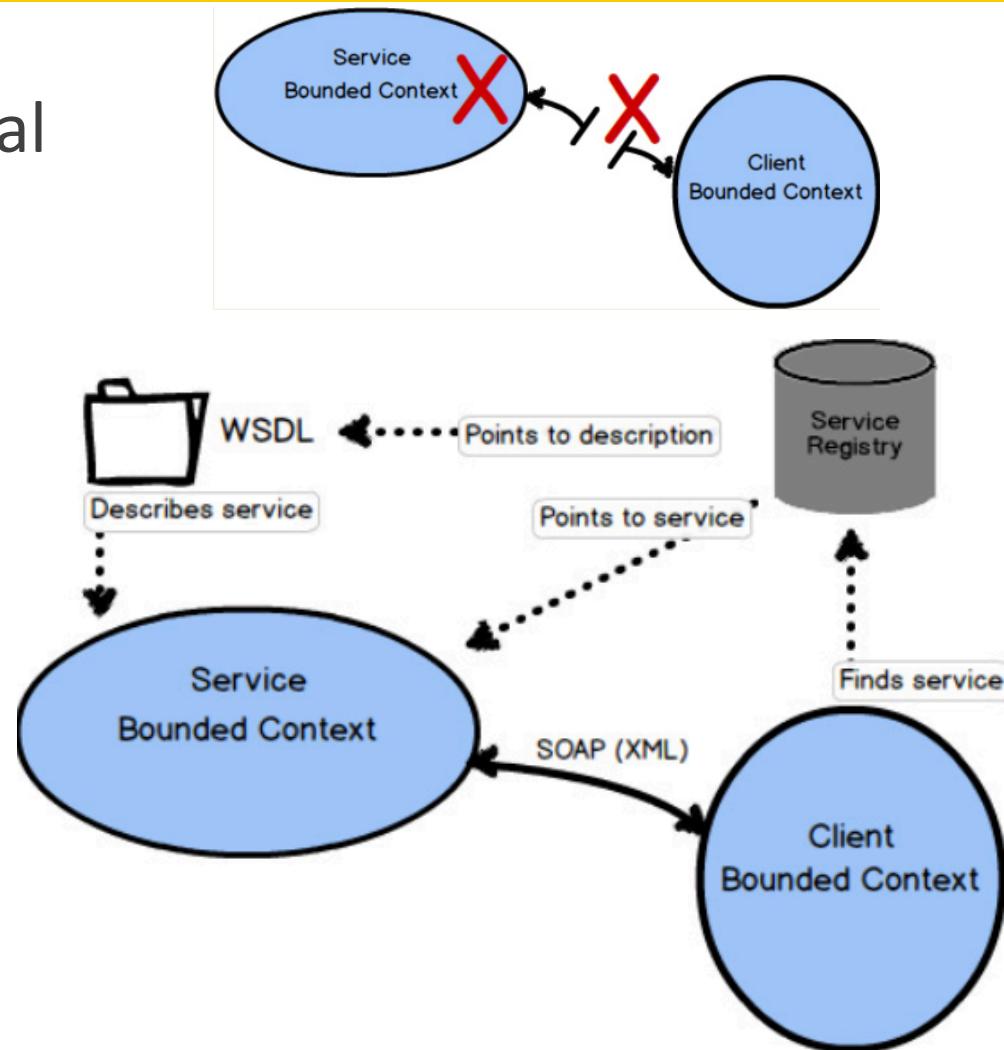
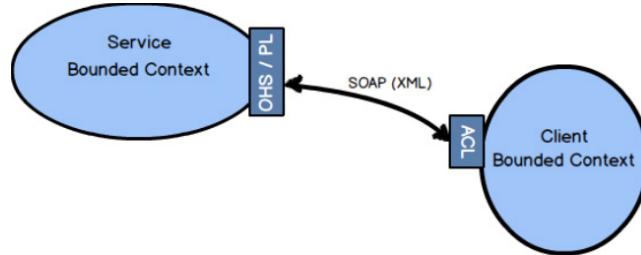


RESTful

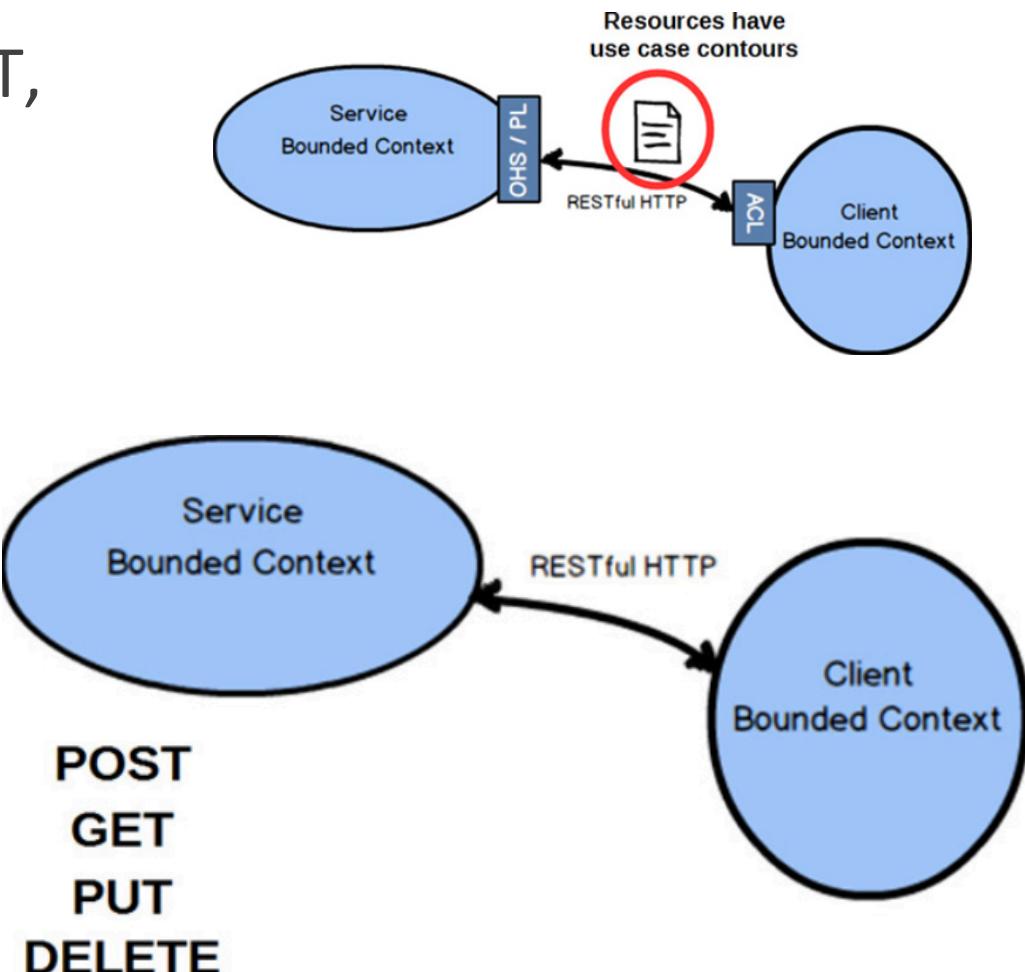
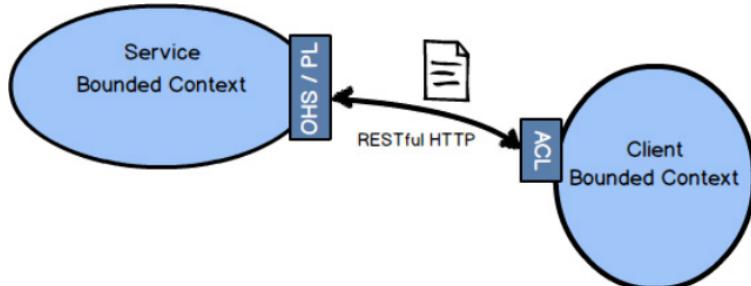
Messaging



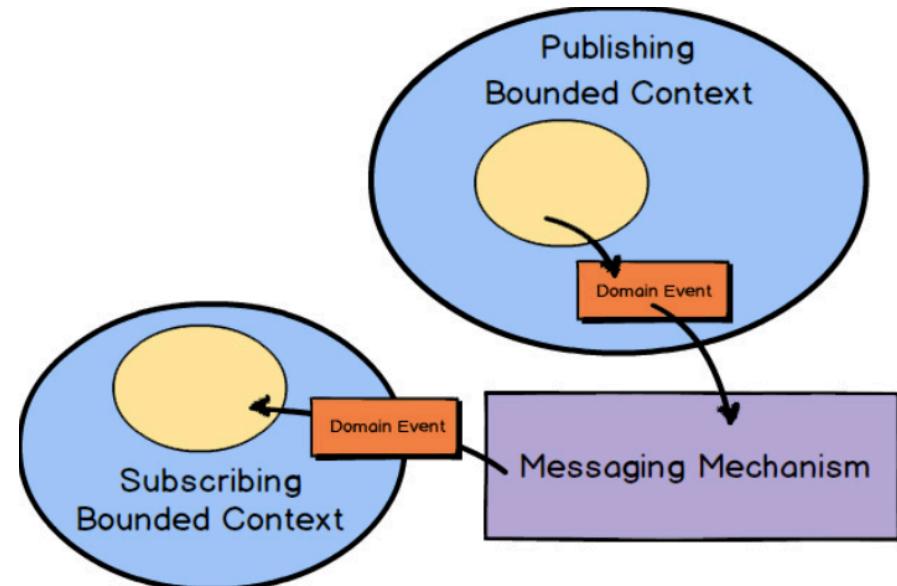
- # Look like a simple, local procedure invocation
- # Potential for network failure or latency
- # Strong coupling



- # Operations: POST, GET, PUT, and DELETE
- # Potential for network failure or latency
- # Reliability, scalability



- # Remove much of the temporal coupling
- # Anticipate the latency of message exchange
- # It's possible to accomplish asynchronous messaging using REST
- # Avoid Integration Train Wrecks



bns it} #} Command Query Responsibility Segregation

Command/Query Separation

```
while ( (row = rowSet.next()) != null) {  
  
    //...  
    while ( rowSet.hasNext() ) {  
        rowSet.moveNext();  
        row = rowSet.get();  
        //...  
    }  
}
```

- # Korzystanie z efektów ubocznych prowadzi do nieczytelnego kodu
- # Metoda może mieć tylko jedno przeznaczenie:
 - Odczyt stanu obiektu (*query*)
 - Zmiana stanu obiektu (*command*)

- # Jeśli weźmiemy pod uwagę perspektywę danych w systemach informatycznych mamy dwa rodzaje operacji
 - Command – modyfikują stan systemu
 - Query – zapytania, służą do pobrania fragmentu stanu systemu

Spójność danych

- Commands – wymagają wykonania w transakcjach, aby zachować spójność
- Queries – duża część operacji odczytu nie wymaga dbania o spójność (wyjątek: Query for update)

Przechowywanie danych

- Commands – zazwyczaj dane muszą być przechowywane w 3. postaci normalnej, aby zachować transakcyjność operacji
- Queries – wyniki zapytań można przechowywać w 1. postaci normalnej, aby uniknąć kosztownych JOINów

Skalowalność

- Commands – ponieważ w dużej części systemów operacje modyfikacji danych stanowią mniejszość, toteż ta część ma mniejsze wymagania odnośnie skalowalności
- Queries – operacji tego typu jest więcej (często o rzęd wielkości) toteż skalowalność ma tu duże znaczenie.

Rozwiązanie

- **Oddziel w systemie części do zapisu i odczytu**
- Część do zapisu podąża za dotychczasową architekturą
- Operacje odczytu są dokonywane z dodatkowego źródła danych, które jest aktualizowane na podstawie przesyłanych zdarzeń
- Część związana z Query to mała hurtownia danych

Przeniesienie idei na architekturę

Część Command

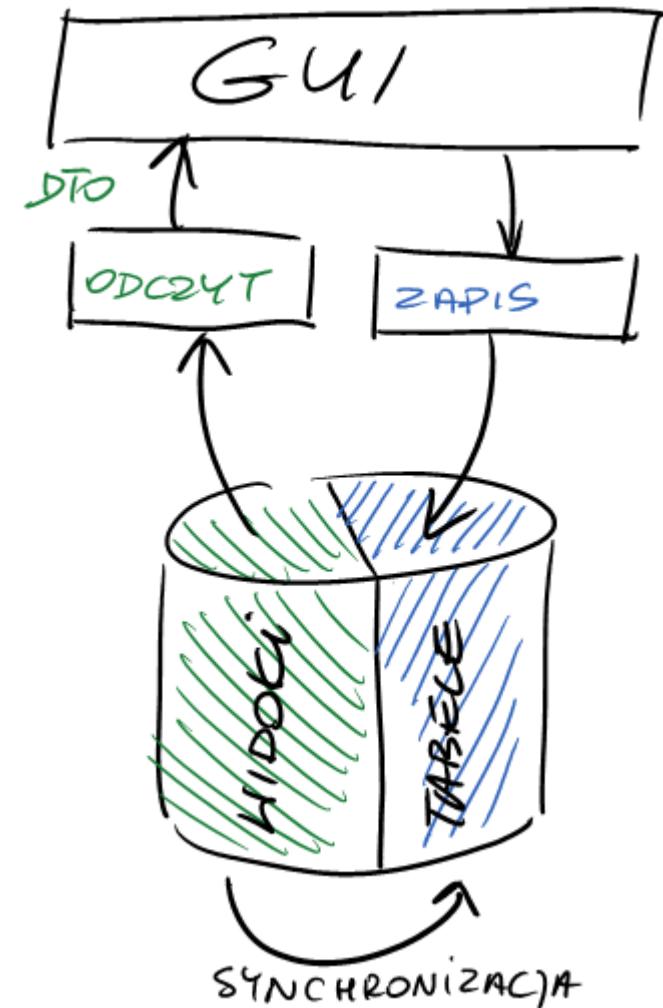
- Szkielet ORM

Część Query

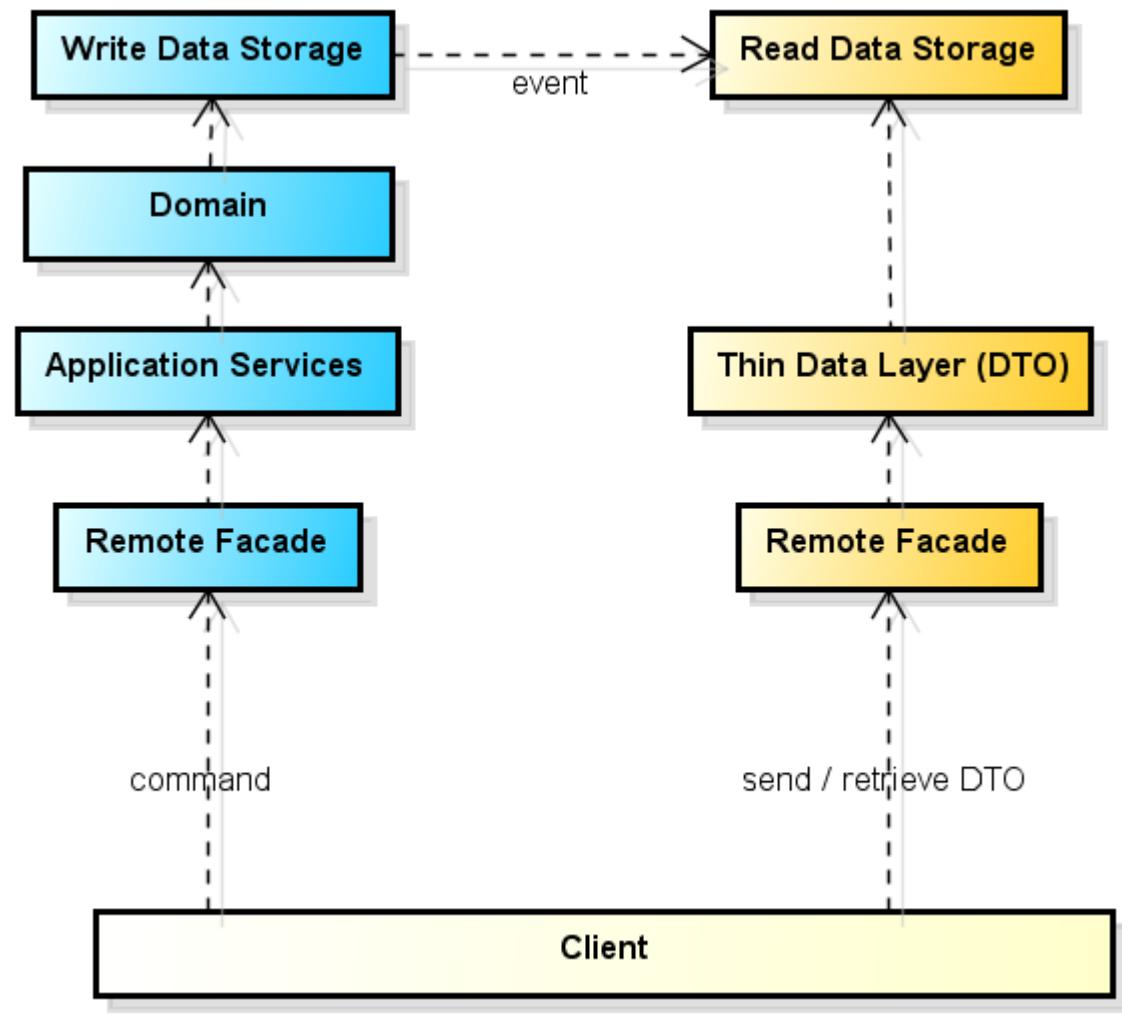
- Widoki bazodanowe
- Obiekty transferowe zaprojektowane „pod potrzeby” konkretnych widoków

Synchronizacja

- Wbudowany mechanizm bazy danych
- Zewnętrzna usługa harmonogramu inicjuje przeliczenie widoków



CQRS bardziej szczegółowo



powered by astah*

Część *Command*

- Specjalizowane polecenia zmieniające stan systemu

Część *Query*

- Specjalizowana baza danych (niekoniecznie relacyjna)
- Rozbudowany mechanizm zapytań

Synchronizacja

- Poprzez mechanizm zdarzeń
- Możliwość odtworzenia stanu systemu z dowolnego momentu