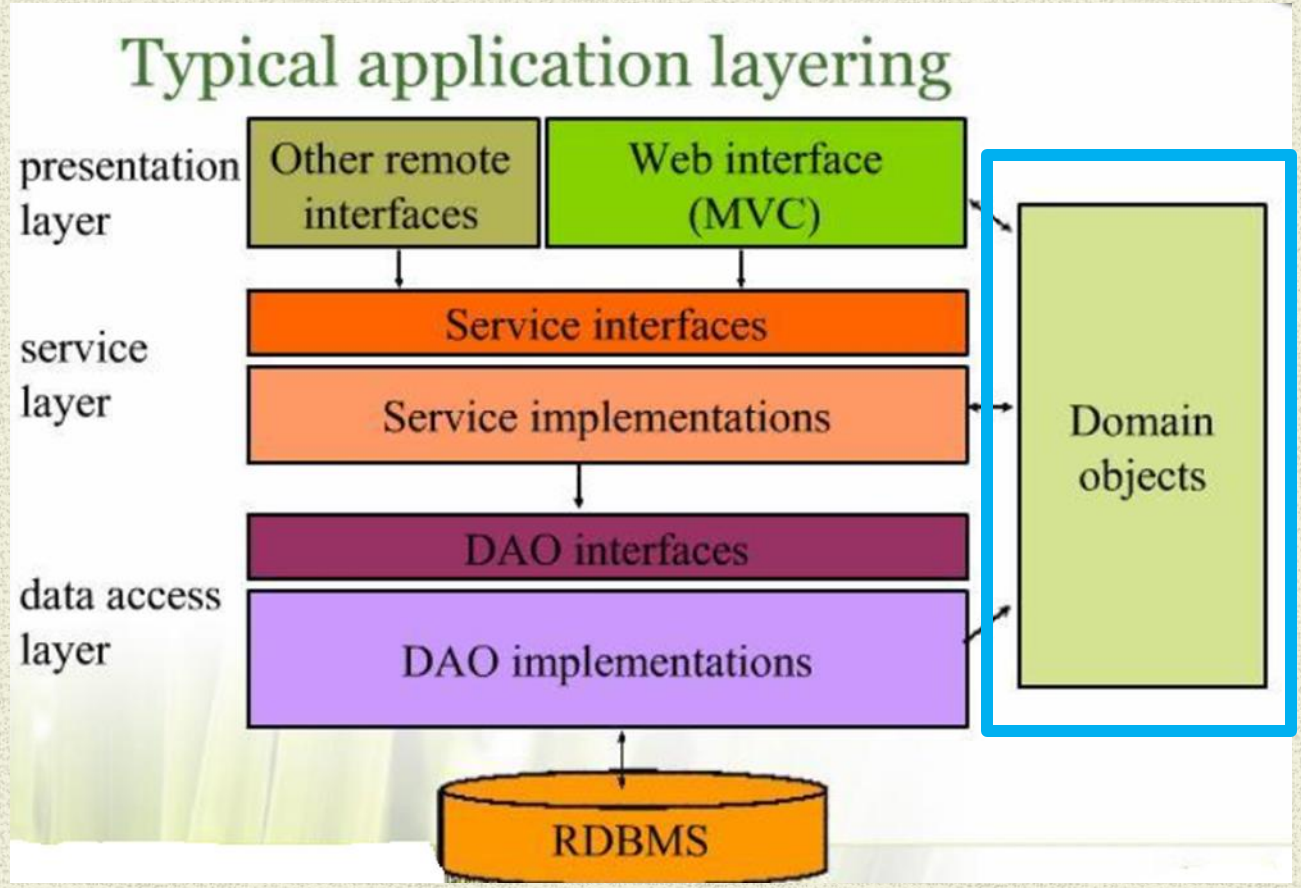
ENTERPRISE INFORMATION

Knowledge is Power

The Domain Model is Essential



Domain Model

- **Application Design is Driven by the Business Domain [Model].**
  - **The domain model is the central, organizing component of the Enterprise**
- **ALL application functionality is derived from it.**
- The rest of the Enterprise is involved in modifying, validating, moving, translating and presenting …the Domain Model - **DATA**

Analysis & Design of the Domain Model

- Capture the essence of business information
- Identify the structure and relationships of the business entities
- Identify the rules that have to be applied to guarantee the integrity of data

PROCESS [SIMPLIFIED]

- Analysis of the problem domain
- Conceptual view of the business [Business Model]
- ***Conceptual OO model***

Leads to

- Detailed Requirements Analysis
- Detailed Design [Class Diagrams]

Domain Driven Design[DDD]

Technique for clarifying Domain Model complexity

With an “eye” towards simplification

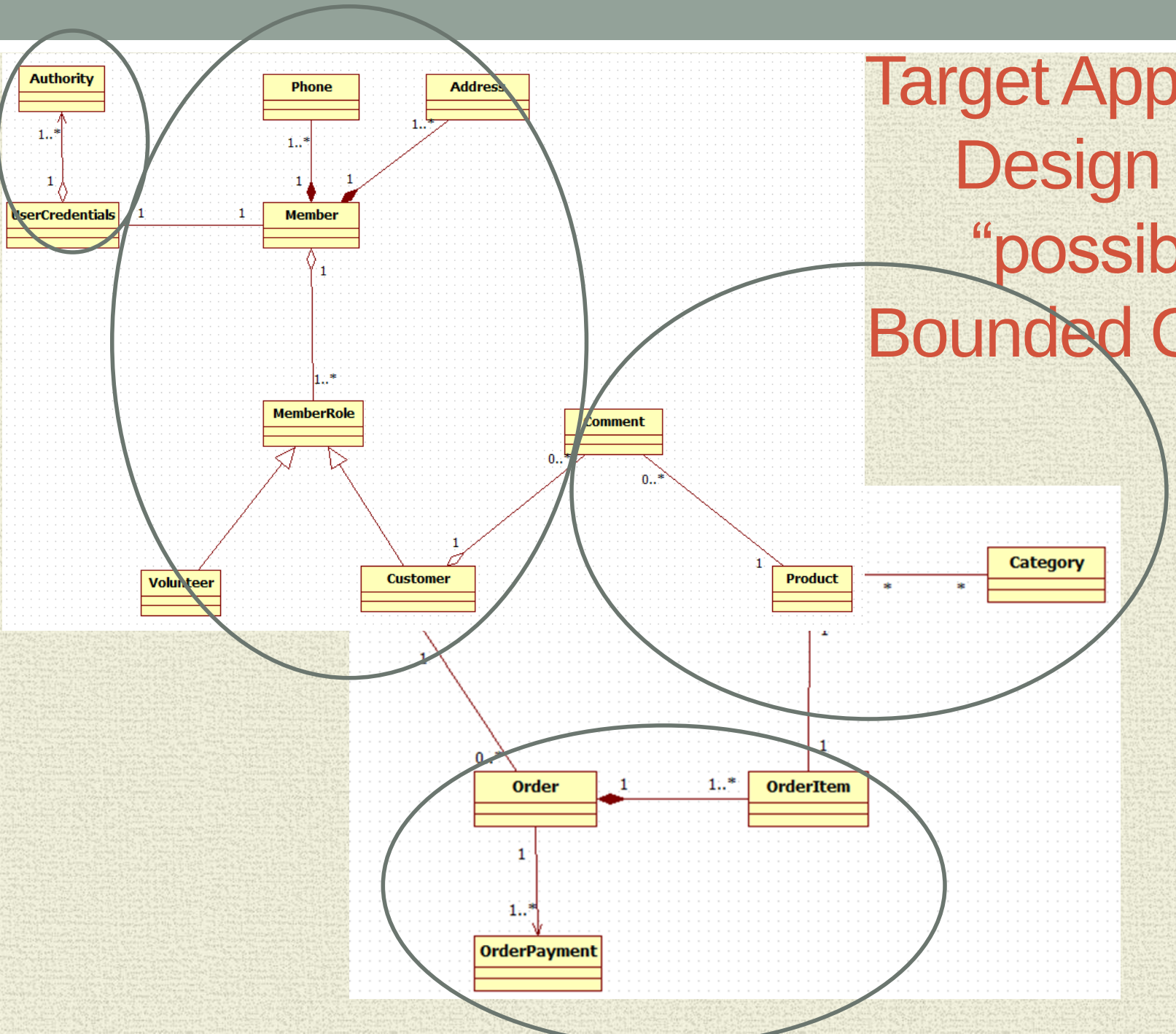
Recognizes that a “single” domain model for a large system is not feasible or cost-effective.

DDD divides up a large system into Bounded Contexts

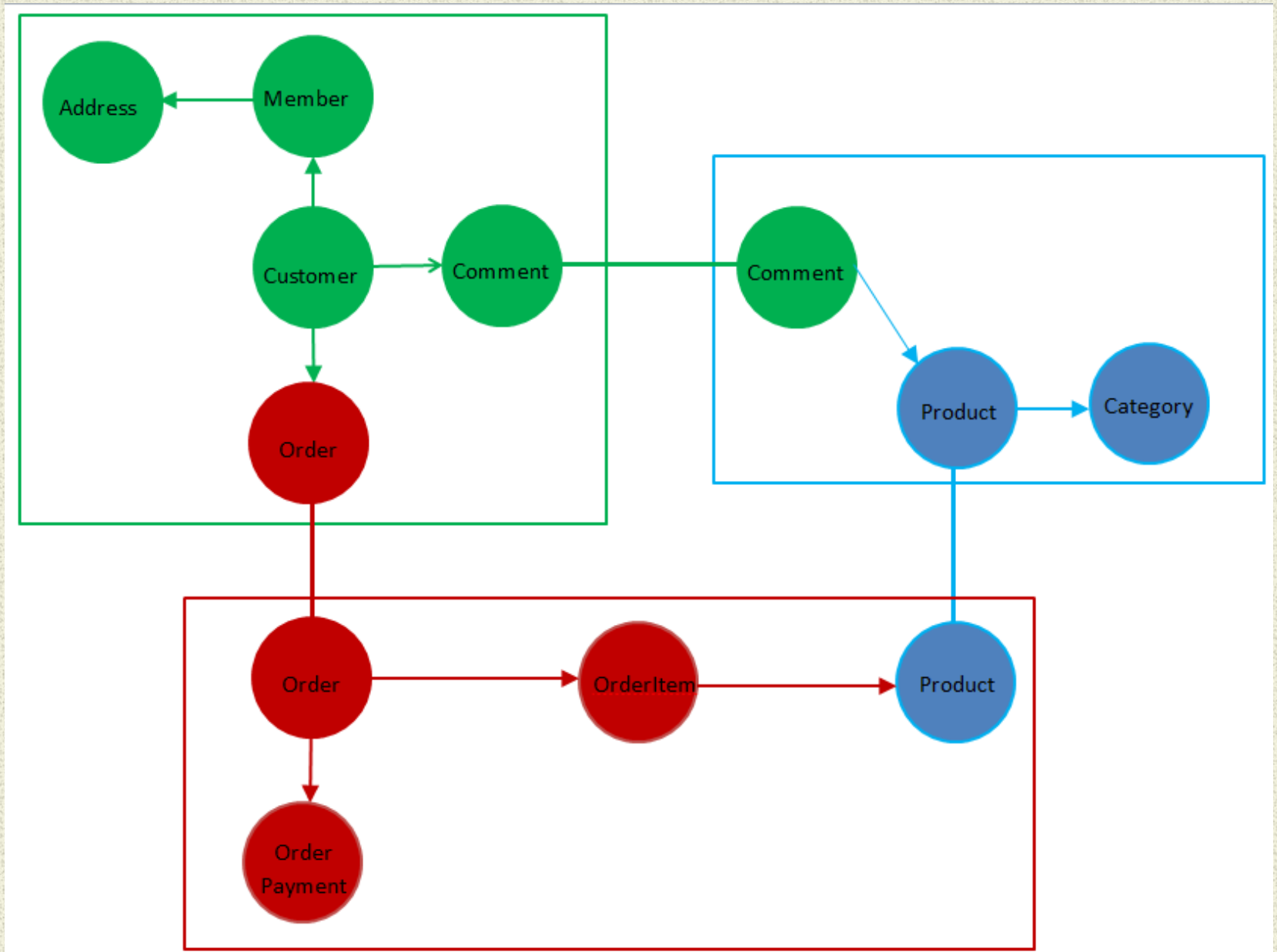
Bounded Context is a central DDD pattern

DDD influences Microservices

Target Application Design Model "possible" Bounded Contexts

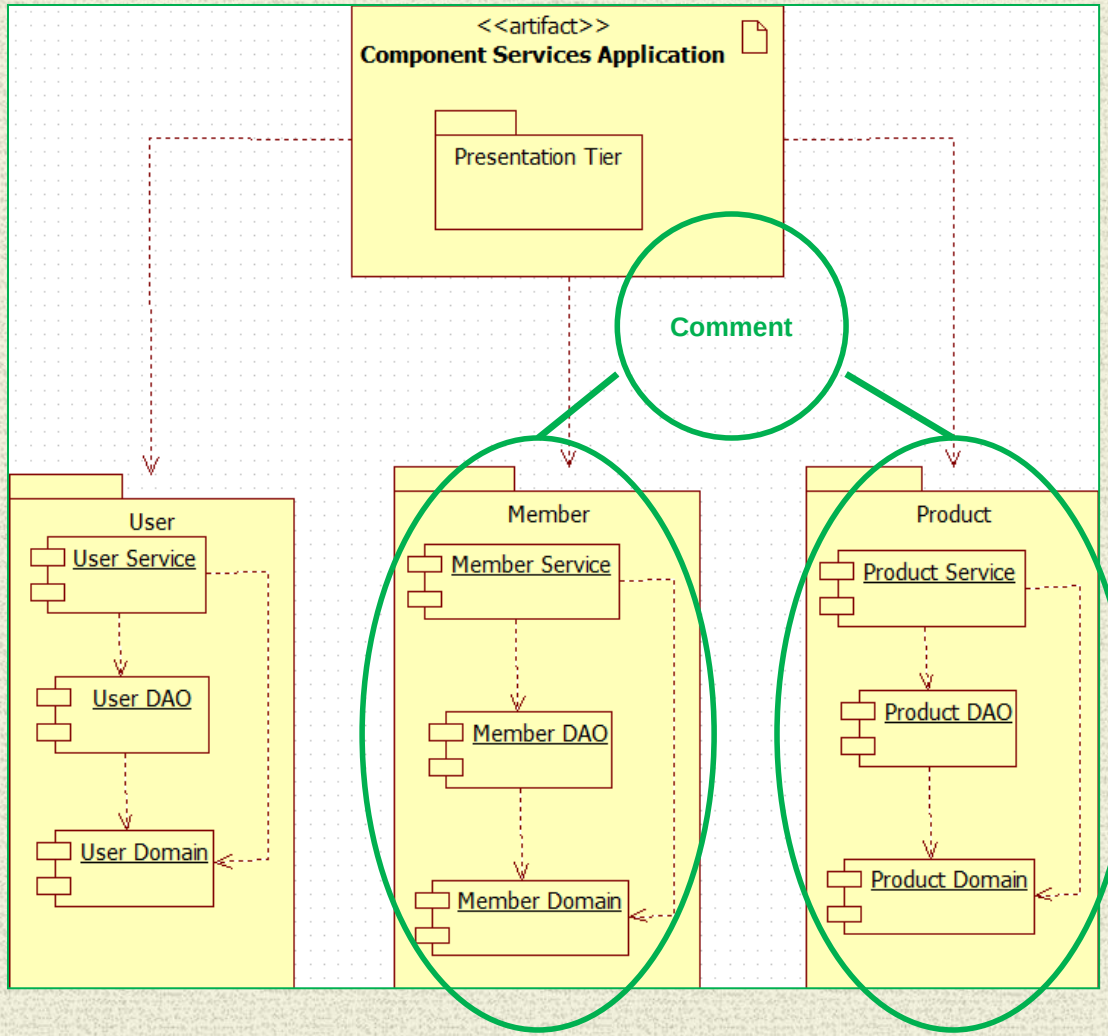


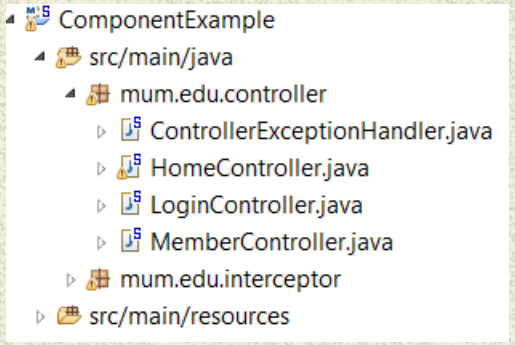
Bounded Context Model

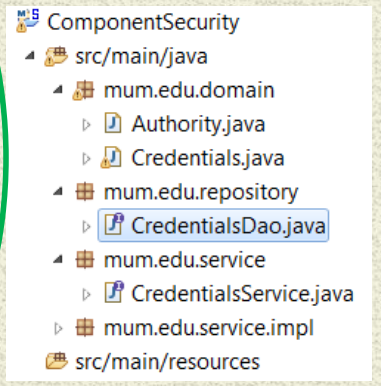


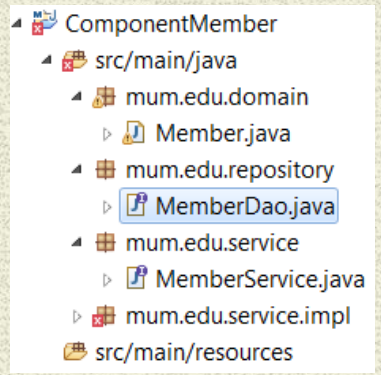
Component N-Tier

We'll look at Bounded Context for Product in Lesson 13



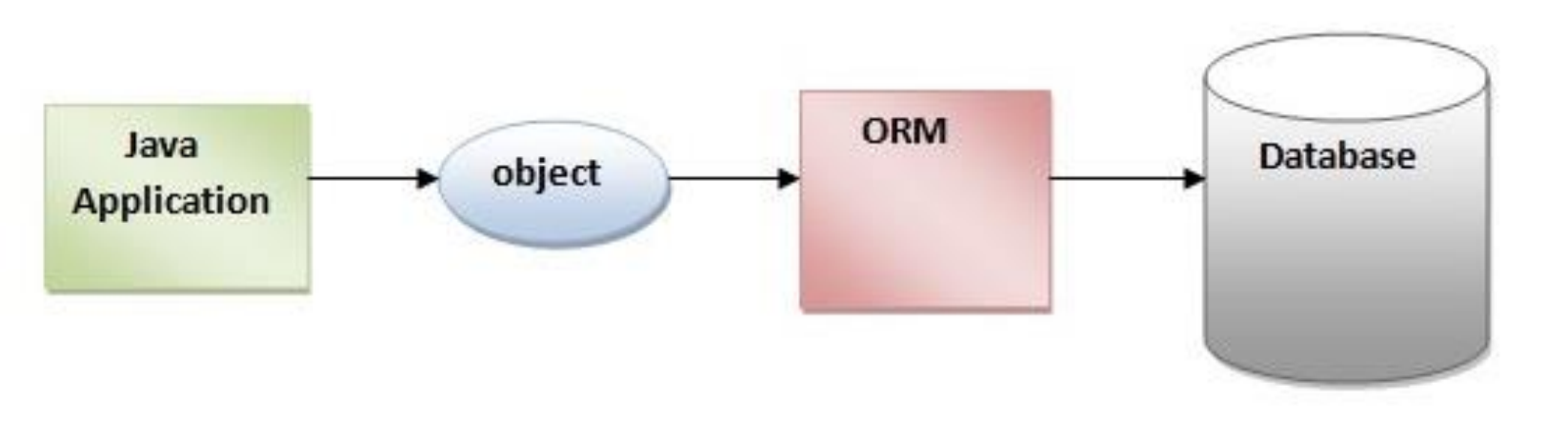






Basic Function of ORM

Acts as a "Gateway" between OO Domain & Relational Database



Maps Object to Relational Model & vice versa

Object-Relational Mapping [ORM]
The Domain Model in a Relational World

Enterprise Applications employ OO in design and implementation

&

Relational Databases for Persistence

There exists a MISMATCH

ORM tools essentially present a relational database from an object oriented viewpoint

The ORM is not for enhancing the Domain Model, it is simply a tool to “overcome” the O/R differences & to hide SQL.

ORM Impedance Mismatch

2 Different Technologies – 2 different ways to operate

EXAMPLE

- **OO traverse objects through relationships**
- Category category = product.getCategory();
- **RDB join the data rows of tables**
- SELECT c.* FROM product p,category c where p.category_id = c.id;

- **OTHERS:**

Many-to-many relationships

Inheritance

Collections

Keys;Foreign Keys

Main Point

An Object Relational Mapping framework provides an Object-Oriented approach to data storage; simplifying the access to the database and effortlessly handling the persistence management for us.

Science of Consciousness: *Transcendental Meditation is an effortless technique to bring us to the simple state of awareness*

ORM USE CASE

For applications based on a rich domain model.

complex business rules

complex interactions between entities

Value is dealing with the full complexity of object/relational persistence.

On the other hand:

An application with only a few entities and simple relationships could be adequately server by direct database table-oriented solutions

WARNING WARNING ORM IS *HARD*

- **“Let the ORM deal with the database.”** - anonymous

BIG MISTAKE!!!

- Works for small applications and loads, but it soon falls apart once “things” get interesting.

- **ORM == 80% of the mapping problems**
- **The other 20% requires “manual labor” at times by a Relational Database expert [RE: DBA type]**

- ***Basic ORM benefit is automating the grunt work*** – relieves the developer from writing all that tedious boiler plate CRUD column to property mapping code.

Basic ORM features

- Mapping Classes To Tables
- Out Of The Box CRUD Functionality
- Hydrating Entities **
- Executing Custom “OO” Queries
- Cache management
- Concurrency support
- Transaction management

** “Automatically” Populate Table data to Object including Relationships

Brief ORM History

NeXT Enterprise Object Framework 1994

RogueWave DBTools – 1994

TopLink for Smalltalk - 1995

TopLink for Java – 1997

ObjectExtender for Smalltalk 1998

Hibernate 2002

iBatis Database layer [SQL Maps & DAO] 2002

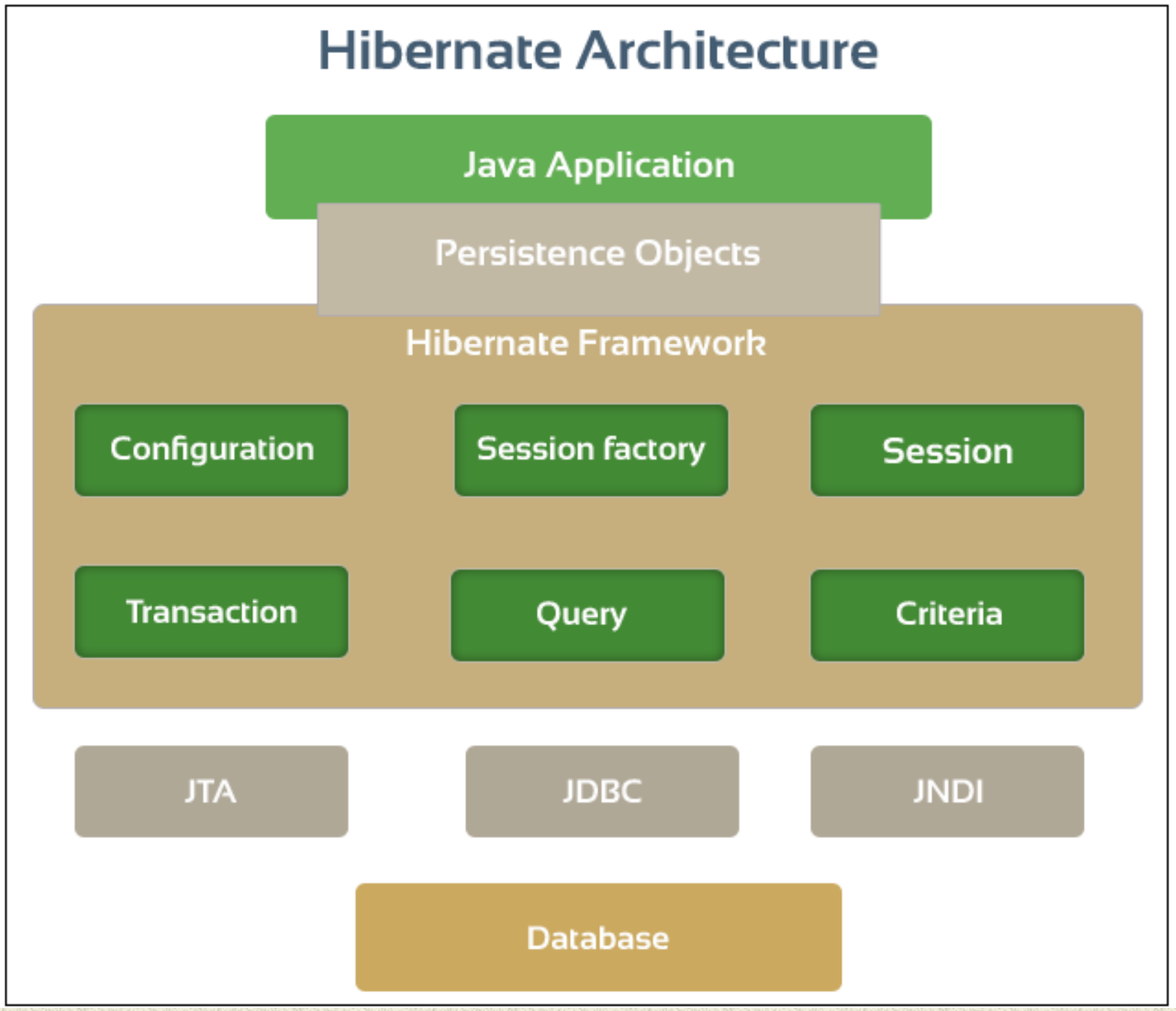
Java Data Objects 2002

Java Persistence API 2006

Java Persistence API

- JPA is a specification – not an implementation.
- JPA 1.0 (2006). JPA 2.0 (2009).
- Standardizes interface across industry platforms
- Object/Relational Mapping
  - **Specifically Persistence for RDBMS**

- Major Implementations [since 2006]:
  - Toplink - Oracle implementation [donated to Eclipse foundation for merge with Eclipselink 2008]
  - Hibernate - Most deployed framework. Major contributor to JPA specification.
  - OpenJPA - (openjpa.apache.org) which is an extension of Kodo implementation.


Hibernate Architecture
Java Application
Persistence Objects
Hibernate Framework
Configuration
Session factory
Session
Transaction
Query
Criteria
JTA
JDBC
JNDI
Database

Range of ORM implementations

- JDBC “ORM”
- Hibernate XML
- Hibernate Annotations
- Hibernate Spring Transactions
- Hibernate Spring – JPA
- Hibernate Spring Data

JDBC "ORM" - VehicleDAOImpl

```
public void insert(Vehicle vehicle) {
  String sql= "INSERT INTO VEHICLE (VEHICLE_NO, COLOR, WHEEL, SEAT)"
              + "VALUES (?, ?, ?, ?)";
      Connection conn = null;
      try {
          conn = dataSource.getConnection();
          PreparedStatement ps = conn.prepareStatement(sql);
          ps.setString(1, vehicle.getVehicleNo());
          ps.setString(2, vehicle.getColor());
          ps.setInt(3, vehicle.getWheel());
          ps.setInt(4, vehicle.getSeat());
          ps.executeUpdate();
          ps.close();
      } catch (SQLException e) {
          throw new RuntimeException(e);
      } finally {
          if (conn != null) {
              try {
```

Database "managed" transaction JDBC Connection is in *auto-commit* mode by default

SEE JDBCDao DEMO

Hibernate GenericDAOImpl

BACK

```
@Autowired
protected SessionFactory sessionFactory;
  protected Session getSession() {return sessionFactory.getCurrentSession(); }
```

Explicitly managed Session & Transaction

```
    @Override
    public void save( T entity ){
     Transaction tx=null;
     try {
          tx = this.getSession().beginTransaction();
        this.getSession().save(entity);
         tx.commit();
     }
     catch (Exception e) {
         if (tx!=null) tx.rollback();
         throw e;
     }
     finally {
     getSession().close();
     }
```

SEE HibernateSolo DEMO

Hibernate XML Domain Mapping File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
     "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="edu.mum.domain">
    <class name="Vehicle" table="VEHICLE">
        <id name="id" column="ID">
            <generator class="native"/>
        </id>
        <property name="color" column="COLOR" />
        <property name="wheel" column="WHEEL" />
        <property name="seat" column="SEAT" />
        <property name="vehicleNo" column="VEHICLE_NO" />
    </class>
</hibernate-mapping>
```

Annotation-based Domain Mapping

```
@Entity(name = "VEHICLE")
public class Vehicle {
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name="ID")
private long id;

@Column(name="VEHICLE_NO")
   private String vehicleNo;
@Column(name="COLOR")
   private String color;
@Column(name="WHEEL")
    private int wheel;
@Column(name="SEAT")
    private int seat;
```

Annotations INSTEAD of XML mapping files

SEE HibernateAnnotations DEMO

Spring ORM Support

Comprehensive transaction support is one of the compelling reasons to use the Spring Framework.

- Integration with Hibernate, Java Persistence API (JPA)…
- **Hibernate Support**

  First-class integration support through IoC/DI

  Easier testing

  Resource management

  Integrated transaction management

Spring Framework Data Access

Hibernate Spring Managed Transactions

..On Service Layer

```
@Service
@Transactional   → Starts Transaction
public class VehicleServiceImpl implements edu.mum.service.VehicleService
{
        public void save( Vehicle vehicle) {
                vehicleDao.save(vehicle);
        }

---

public abstract class GenericDaoImpl<T> implements GenericDao<T> {
@Autowired
    protected SessionFactory sessionFactory;

 protected Session getSession() {
     return sessionFactory.getCurrentSession();
}
 public void save( T entity ){
     this.getSession().save(entity);
}
```

Reduction in code*:
manage transaction
open/close session
** Compared to Hibernate Solo*

SEE HibernateTransactions DEMO

JPA Example

```
public abstract class GenericDaoImpl<T> implements GenericDao<T>
{

@PersistenceContext
    protected EntityManager entityManager;

    protected Class<T> daoType;

public void setDaoType(Class<T> type) {
                daoType = type;
}

    @Override
    public void save( T entity ){
        entityManager.persist( entity );
     }
```

EntityManager “replaces” Session

SEE HibernateSpringJPA DEMO
HibernateSpringJPAJConfig
HibernateSpringBoot

JPQL - Data Object Queries

JPA Query Language

- JPQL is different from SQL in that it operates on objects, attributes and relationships instead of tables and columns.

Queries are declared in the DAO implementation

```
@Repository
public class MemberDaoImpl extends GenericDaoImpl<Member> implements MemberDao {

   public MemberDaoImpl() {
       super.setDaoType(Member.class );
   }
   public Member findByMemberNumber(Integer memberNumber) {
          Query query = entityManager.createQuery("select m from MEMBER m
                                          where m.memberNumber =:number");
   return (Member) query.setParameter("number", memberNumber).getSingleResult();
   }
```

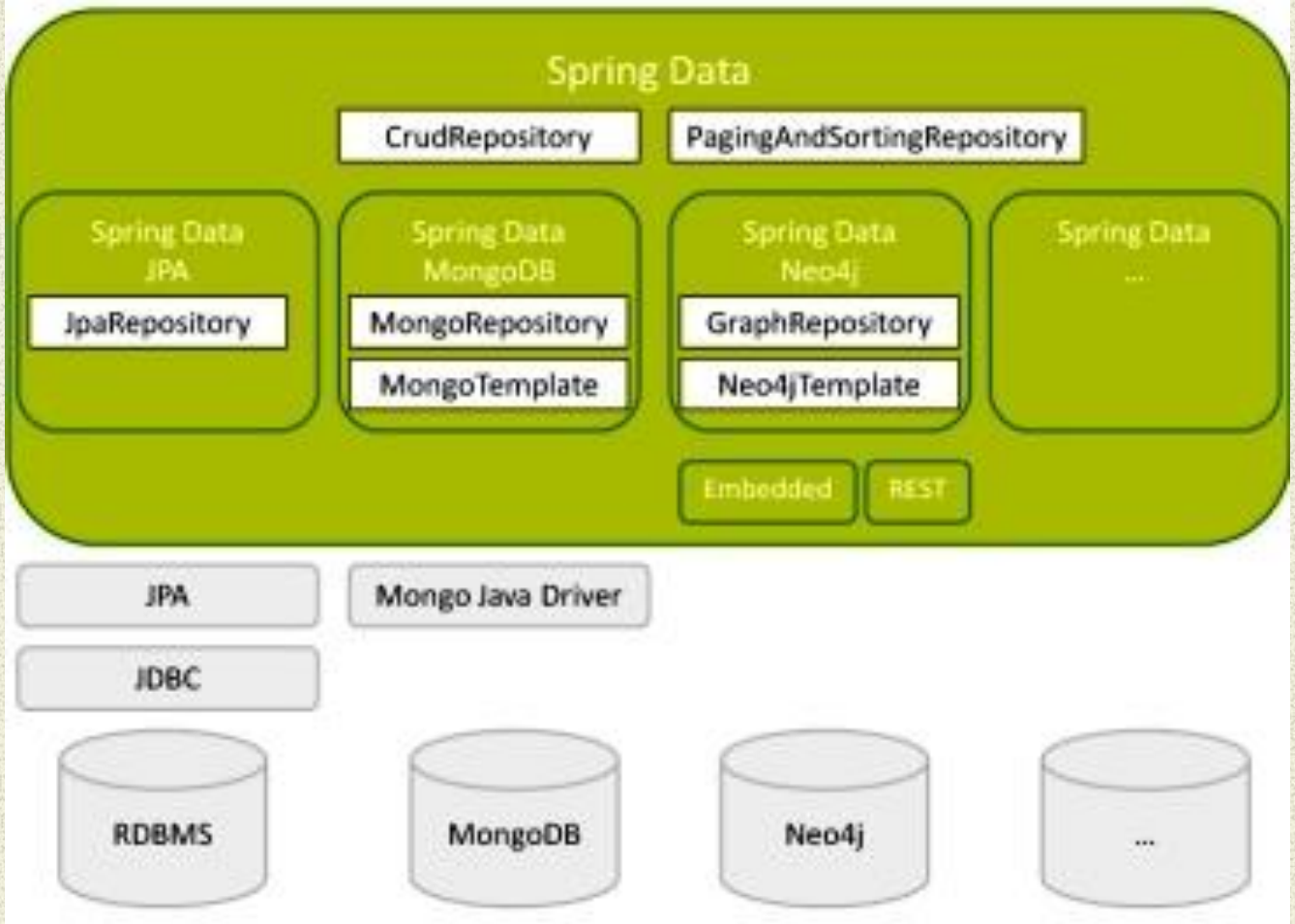
Spring Data

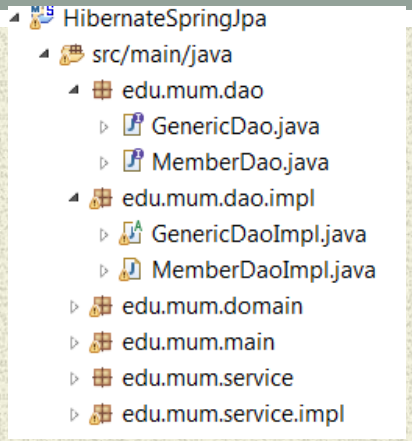
- Spring Data
  - High level Spring project whose purpose is to unify and ease the access to different kinds of persistence stores, both relational database systems and NoSQL data stores.

- Hibernate ORM
  - (Hibernate for short) is an object-relational mapping Java library; a framework for mapping an object-oriented domain model to a traditional relational database. Distributed under the GNU Lesser General Public License

Spring Data Project

- High level Spring project whose purpose is to unify and ease the access to different kinds of persistence stores, both relational database systems and NoSQL data stores.

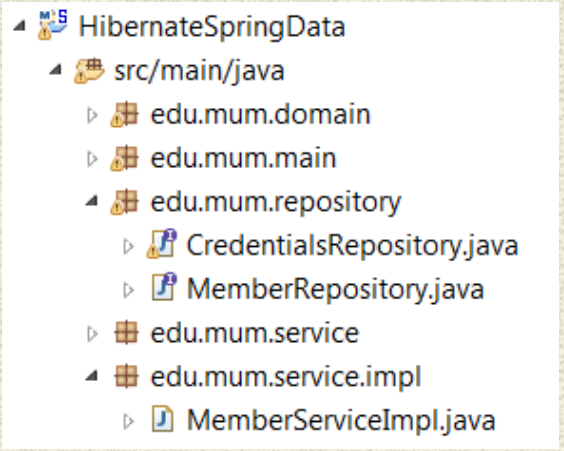




Spring Data

AUTO-GENERATES the DAO

No Need for GenericDAO, etc.




```
@Repository
public class MemberDaoImpl extends GenericDaoImpl<Member> implements MemberDao {
   public MemberDaoImpl() {
       super.setDaoType(Member.class );
   }
   public Member findByMemberNumber(Integer memberNumber) {
          Query query = entityManager.createQuery("select m from MEMBER m
                                                   where m.memberNumber =:number");
     return (Member) query.setParameter("number", memberNumber).getSingleResult();
   }
```

BECOMES

```
@Repository
    public interface MemberRepository extends CrudRepository<Member, Long>
    {
      public Member findByMemberNumber(Integer memberNumber);
    }
```

See HibernateSpringData

Main Point

- JPA is a specification not an implementation. It provides a consistent, reliable mechanism for data storage and retrieval that alleviates the application developer from the details involved in the persistence layer.

- *The mechanism of transcending allows the individual to tap into the Home of all the Laws of Nature alleviating the stress of mundane day to day issues.*