CS544 EA

Overview

Intro: Principles and Patterns

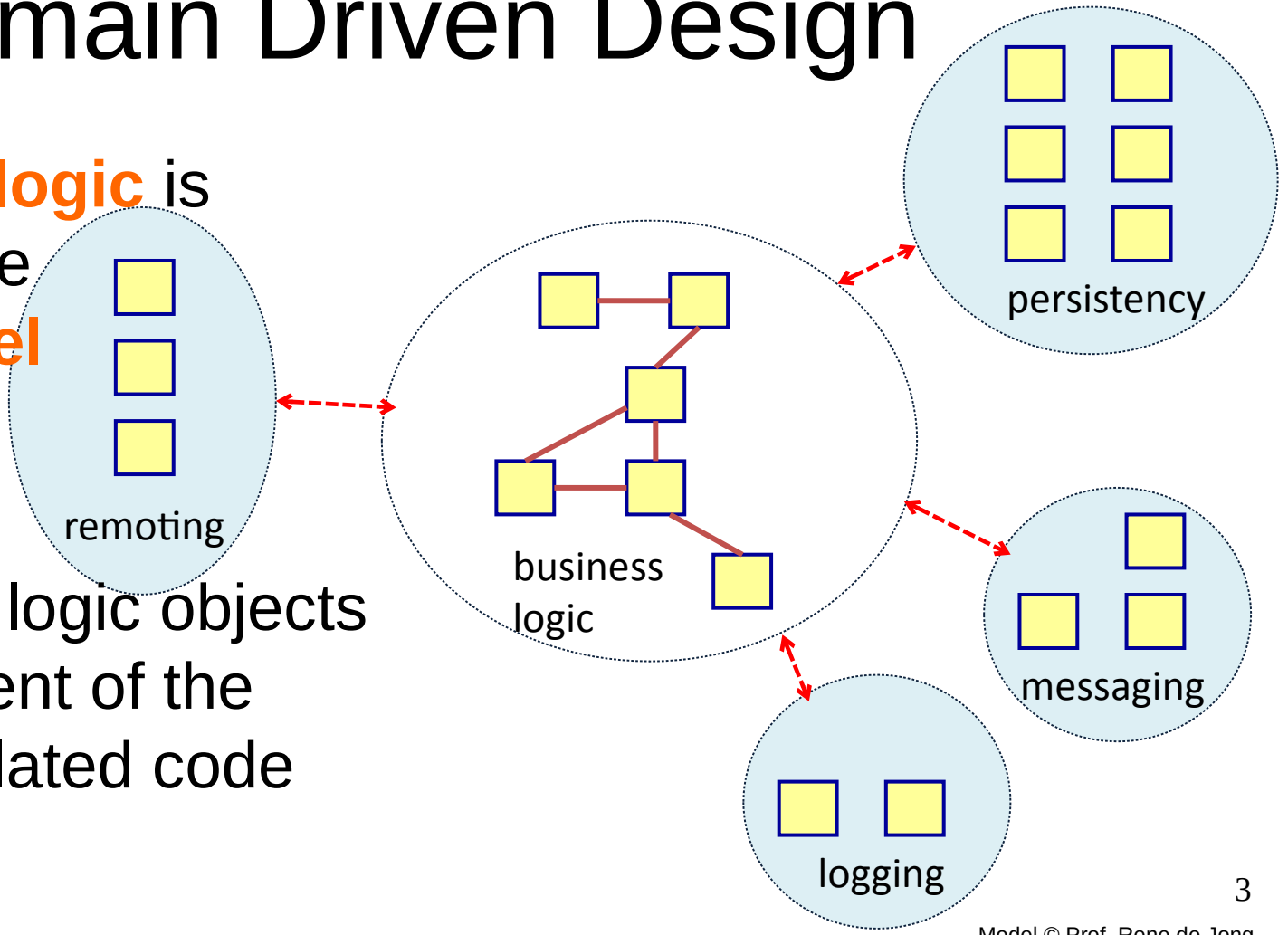
Principles

- Certain principles and patterns are present when creating an enterprise application
  - Whether this is a service or a monolith

- We give a brief overview of the most important of these
  - So that you can **keep the bigger picture in mind** when we go into deeper details in future lectures.

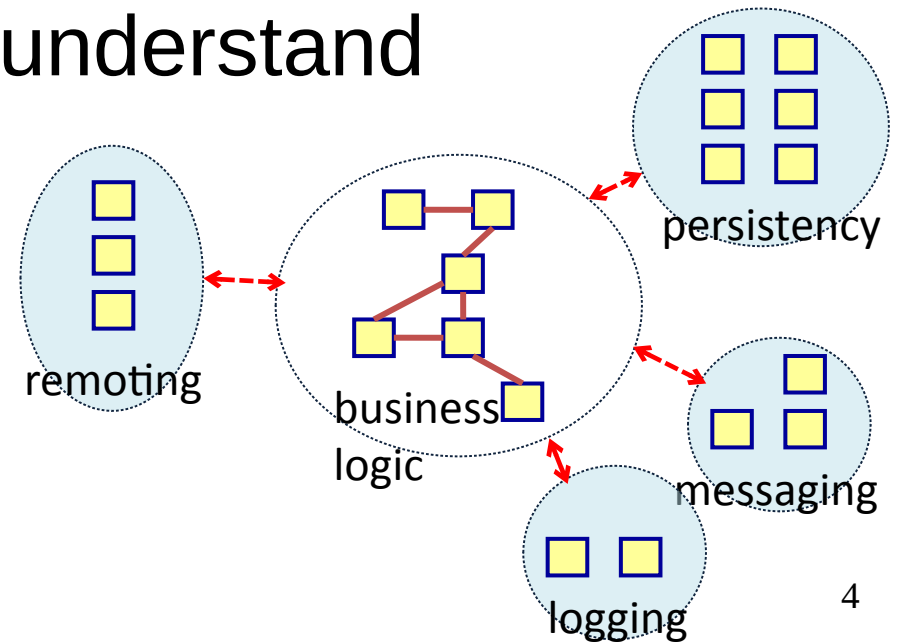
Domain Driven Design

- All **business logic** is captured in the **domain model** that reflects the real world
- The business logic objects are independent of the technology related code



Advantages of DDD

- Business logic is **independent of technology** changes – switching between tech is easy
- Business logic is easy to understand
  - Easy to write, test, modify

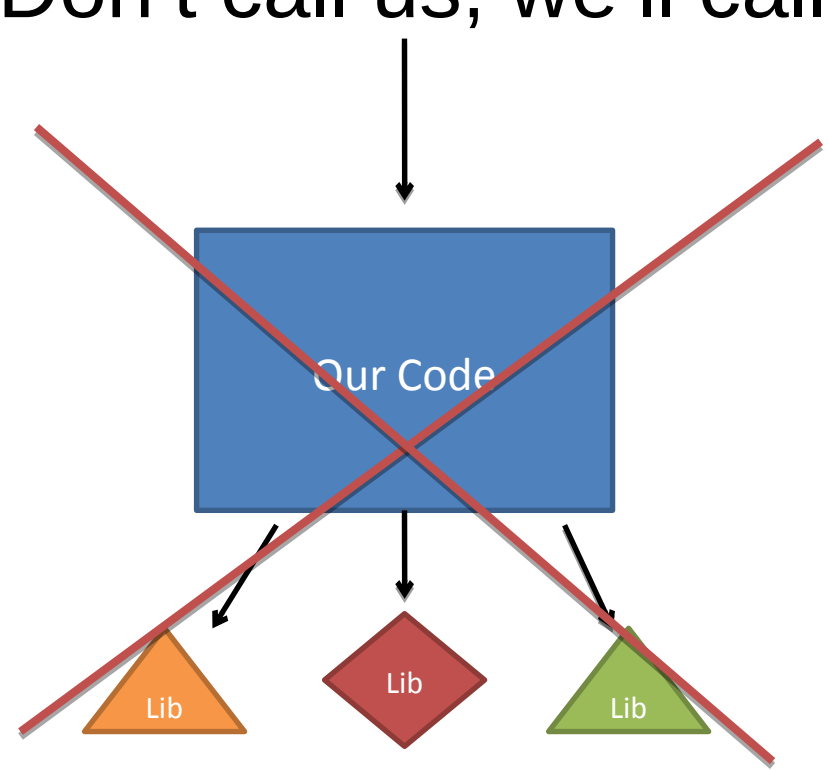


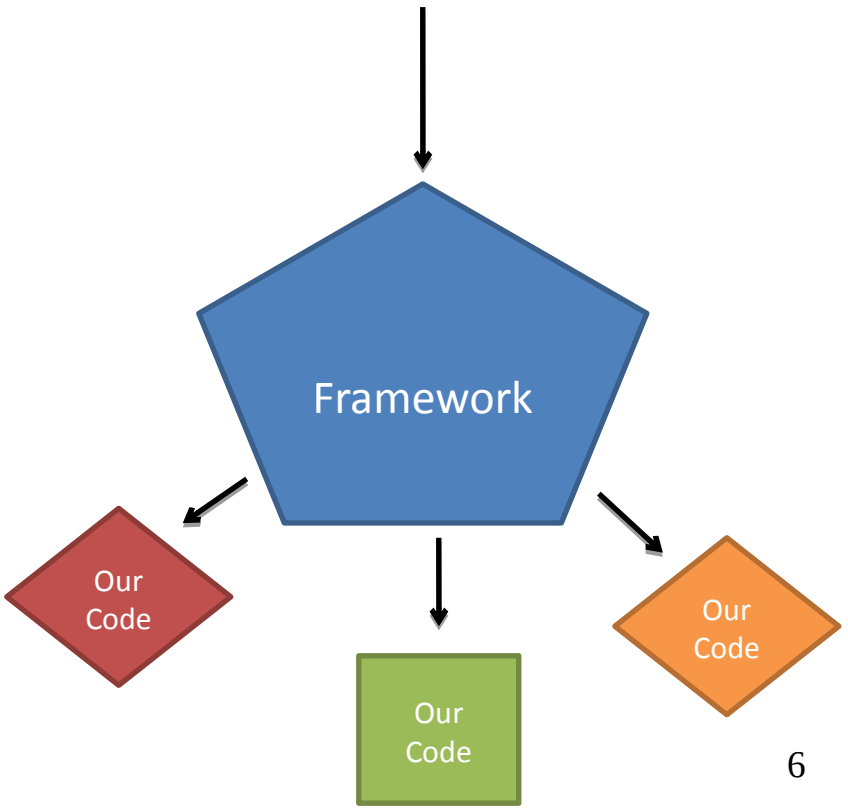
Containers

- A container is a piece of software that manages your objects. Common Examples:
  - Web (servlet) Container
  - EJB / Spring Bean Container
  - JPA EntityManager
- The primary principle that containers use is **Inversion of Control** (IoC)

IOC / Hollywood Principle

- Don’t call us, we’ll call you





Java Bean Standard

- An important part of IoC is that the container **creates and manages** your objects.
- To facilitate this, your classes should adhere to the Java Bean Standard
    - All properties are private (use getters & setters)
    - A public, no-argument constructor
    - Implements Serializable

Dependency Injection

- Loose Coupling is another important aspect
  - To be able to interchange objects with different versions depending on the context
- The container **creates and then connects** the objects together – injects one into the other.

```
public class CustomerService {

  @Inject
  private EmailService emailService;

}
```

Aspect Oriented Programming

- Some functionalities are cross-cutting concerns
  - Things you need everywhere, like:
  - Transactions, Security, Logging, …

- **Copy pasting** the same **code is bad**
  - Don’t Repeat Yourself (DRY)

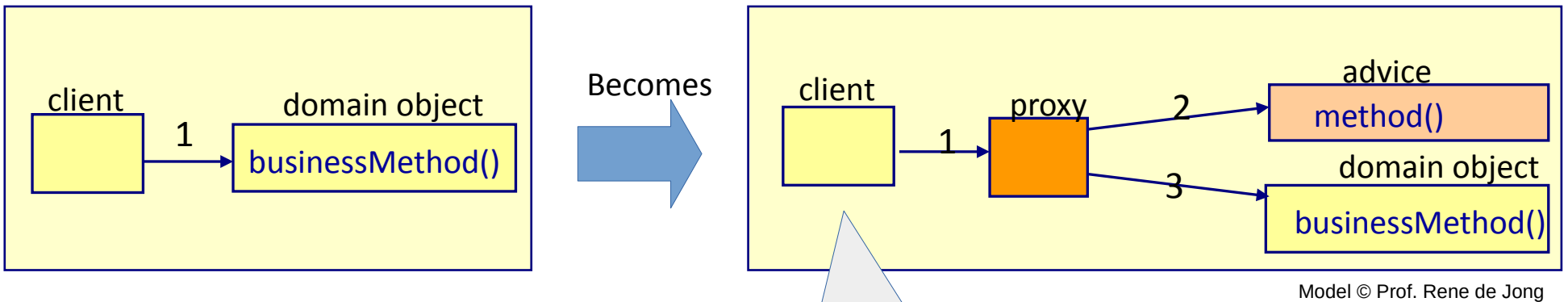
Advice

- With AOP you can **write** code for a cross-cutting-concern **once** (in an advice method)
- **Then specify all the points** (before or after which methods), that the advice should run
- This is called an Aspect:
  - Advice plus the points where it should run

Containers Implement AOP

- The container can inject any object that implements the same interface as the requested object.
- A **proxy object** can be generated **to look just like** the real thing (to be injected instead).
- The proxy **calls the Advice** method (before or after) and then calls the real method on to the actual object.
  - This also called the interceptor pattern

Visually


client
1
domain object
businessMethod()
Becomes
client
1
proxy
2
advice
method()
3
domain object
businessMethod()
Model © Prof. Rene de Jong
Client does not know it's not talking to domain object.

Object Relational Mapping

- Relational databases are often a good fit for business applications
  - Relational databases are what's traditionally used

- There are **differences between OO and relational** models.
  - Use Object Relational Mapping (ORM) to bridge the gap

Mapping

- Main **Areas of Mismatch** / mapping need
  - Identity Mapping
  - Associations Mapping
  - Inheritance Mapping

- Java Persistence Query Language (JPQL)
  - Primarily used for data retrieval, like SQL but OO

Repositories and DTOs

- Two of the most common data related patterns are:
  - Repositories aka Data Access Objects (**DAO**)
    - All database related code for a domain class is kept in the repository for that domain class. Eg. Person and PersonDao
  - Data Transfer Objects (**DTO**)
    - When a set of data (not domain objects) needs to go between layers
    - Separate DTO class is made to hold and transfer the data

Integration

- The two primary ways for integration are:
  - **Remote Invocation**
    - In the past many mechanism: CORBA, RMI, ...
    - Currently primarily with **Web Services**: REST, SOAP
    - Often Synchronous – request waits for response
  - **Messaging**
    - Sending without requiring a direct response (asynchronous)
    - Standards like JMS, AMQP, MSMQ,
    - Either Publish / Subscribe or Point to Point sending

Summary

- Containers use IoC which gives us:
  - Dependency Injection (loose coupling)
  - Aspect Oriented Programming (interceptors)
- Object Relational Mapping is used so we can work in an OO language while storing in a RDB
- Integration uses remote invocation or messaging
- We see Unity in Diversity
  - it’s more or less always the same principles / patterns that are used.