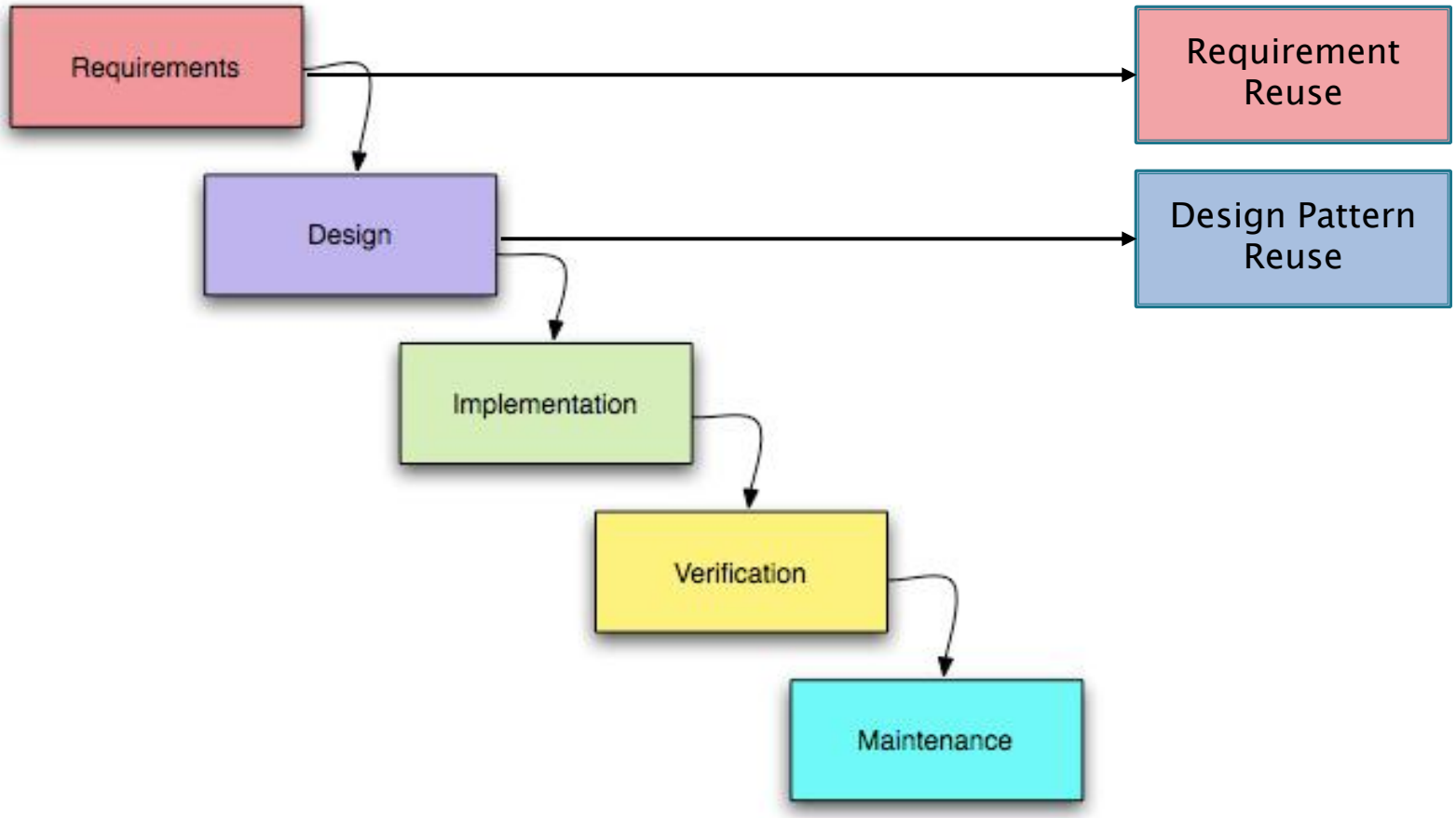
Module 3
Design Pattern Reuse (Part 2)

Dr. Shiping Chen

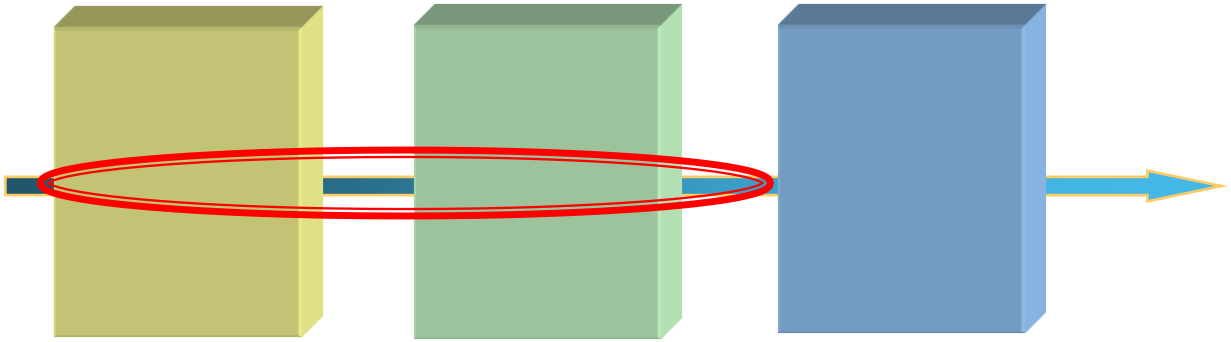
Where are we?



Outline

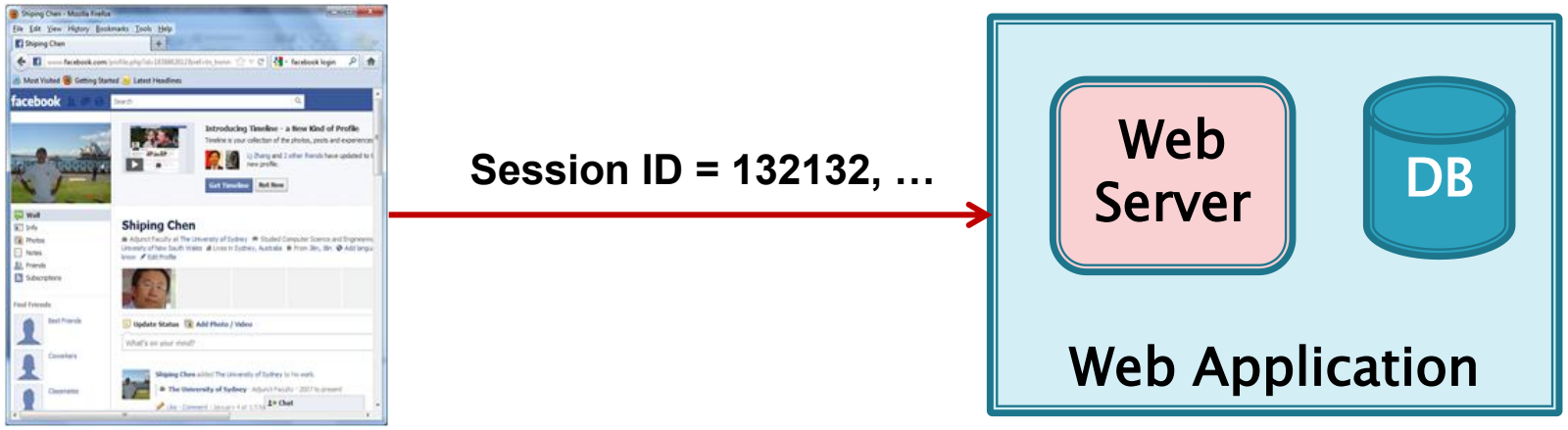
- Session State Patterns
- Domain Logic Patterns
- Message Channel Patterns
- Task 2 of Assignment

Session State Patterns



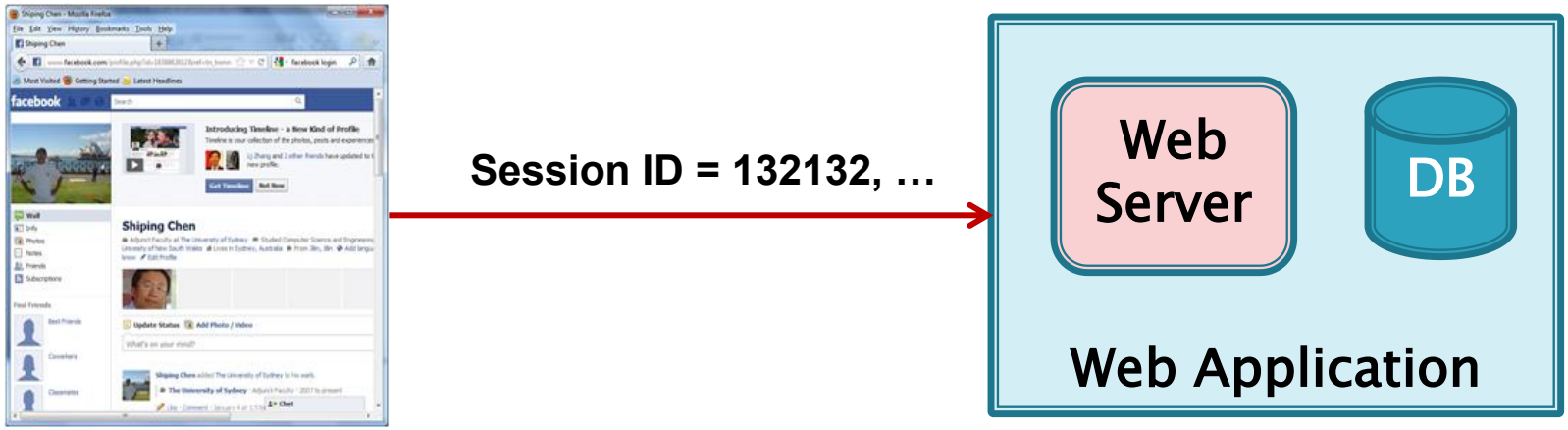
- What is a Session?
  - A semi-permanent interactive information interchange, also known as a conversation, between two or more devices
- What is a Session State
  - The information required to maintain the unique conversation, such as session id, your shop cart, your emails, etc.

Why Session State?



- Security: allow a server to check if the request is from a valid logined user;
- Personalized web page: allow a server to provide personalized services, such as your profile, your shopping cart, etc.

3 Design Patterns for Session State Managment



- Client Session State Management
- Server Session State Management
- Database Session State Management

Client Session State

How it works	Store session state on the client
Pros	• Improve server performance with stateless servers • Improve server performance with easily clustering • Improve server reliability with easily failover recovery
Cons	• Communication overhead • Programming overhead • Not Secure
When to use	• Must be used for session ID anyway • small amount of session state data • Not serious applications

Server Session State

How it works	Keep the session state on the server system in a serialized form
Pros	• Simple, no need programming • Can handle complex session state objects • Secure
Cons	• Hard clustering and failover recovery • Memory cost • Some performance overhead, even not too much
When to use	• Complex session state data • don't want to look after it • Strong secure systems

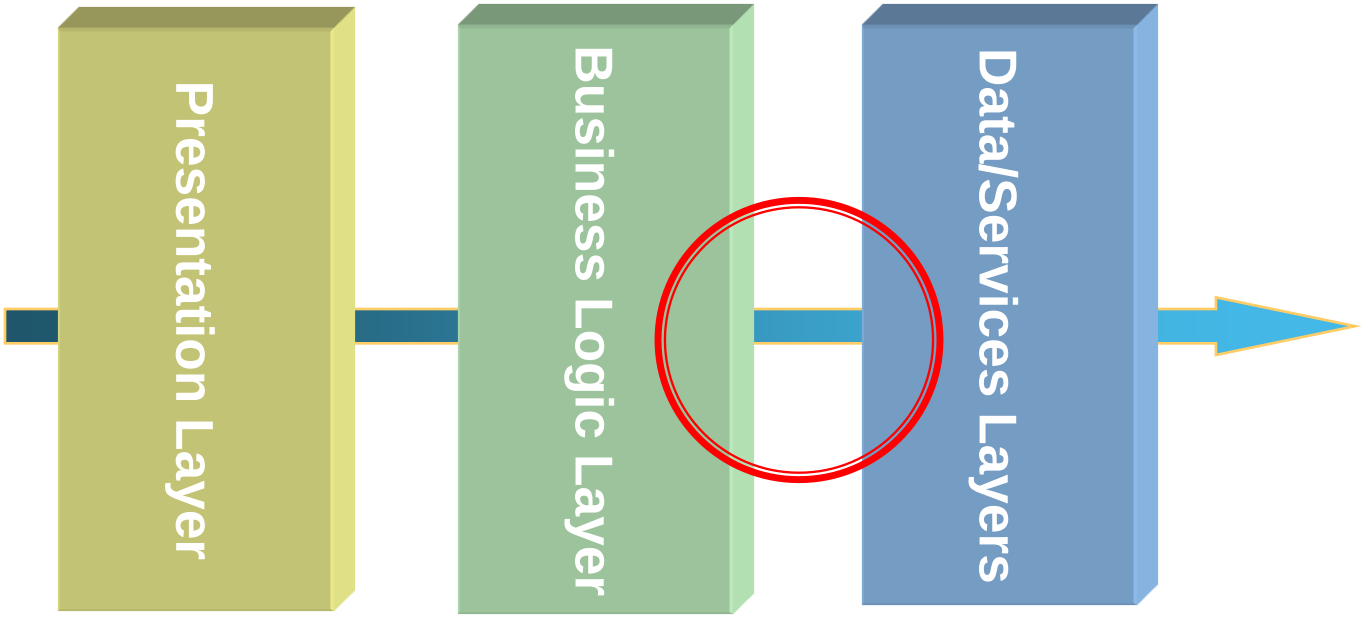
Database Session State

How it works	Store session state as committed data in the database
Pros	• Can handle large session state objects • Can handle multiple types of session state data • Make clustering and Failover recovery easier • Secure
Cons	• Programming overhead • Maybe performance overhead • need carefully managing transactions
When to use	• Large, complex and multiple types session state data • tasteful server clustering • Strong secure systems

Put IT All Together

Patterns	Client-SS	Server-SS	DB-SS
Programming Overhead	M	L	H
Security Risk	H	L	L
Clustering/FT Difficulty	L	H	L
Performance Overhead	H	M	H
Data Process Capability	L	M	H

Domain Logic Patterns



- To address how to manage the business data and their relationship
- It mainly refers to M in MVC architecture

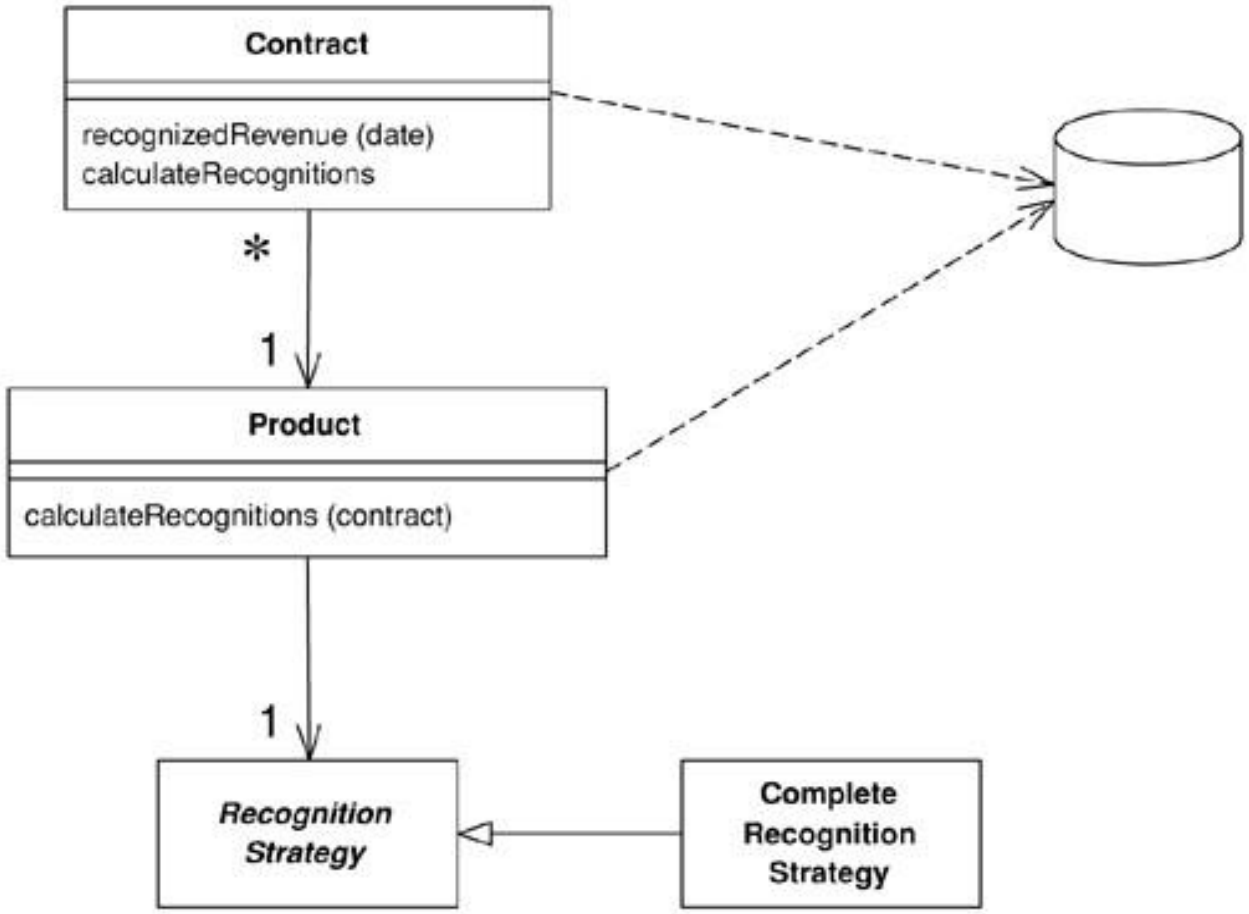
Transaction Script

```
-- cr_spatial_index.sql –
declare
   cursor c1 is SELECT DISTICT sdogid from POLYGON_SDOGEOM;
   gid number;
   i number;
begin
    i := 0;
    for r in c1 loop
          ......
    endloop
    commit;
end;
```

Transaction Script: Pros vs. Cons

How it works	Organizes business logic by (SQL) procedures where each procedure handles a single request from the presentation
Pros	• Simple • Easily understanding • Fast with less performance overhead
Cons	• Becoming messy as business gets complicated • due to a large amount of duplicated code • Hard to maintain and reuse
When to use	• Simple systems • Small systems • Stable systems with less changes

Domain Model

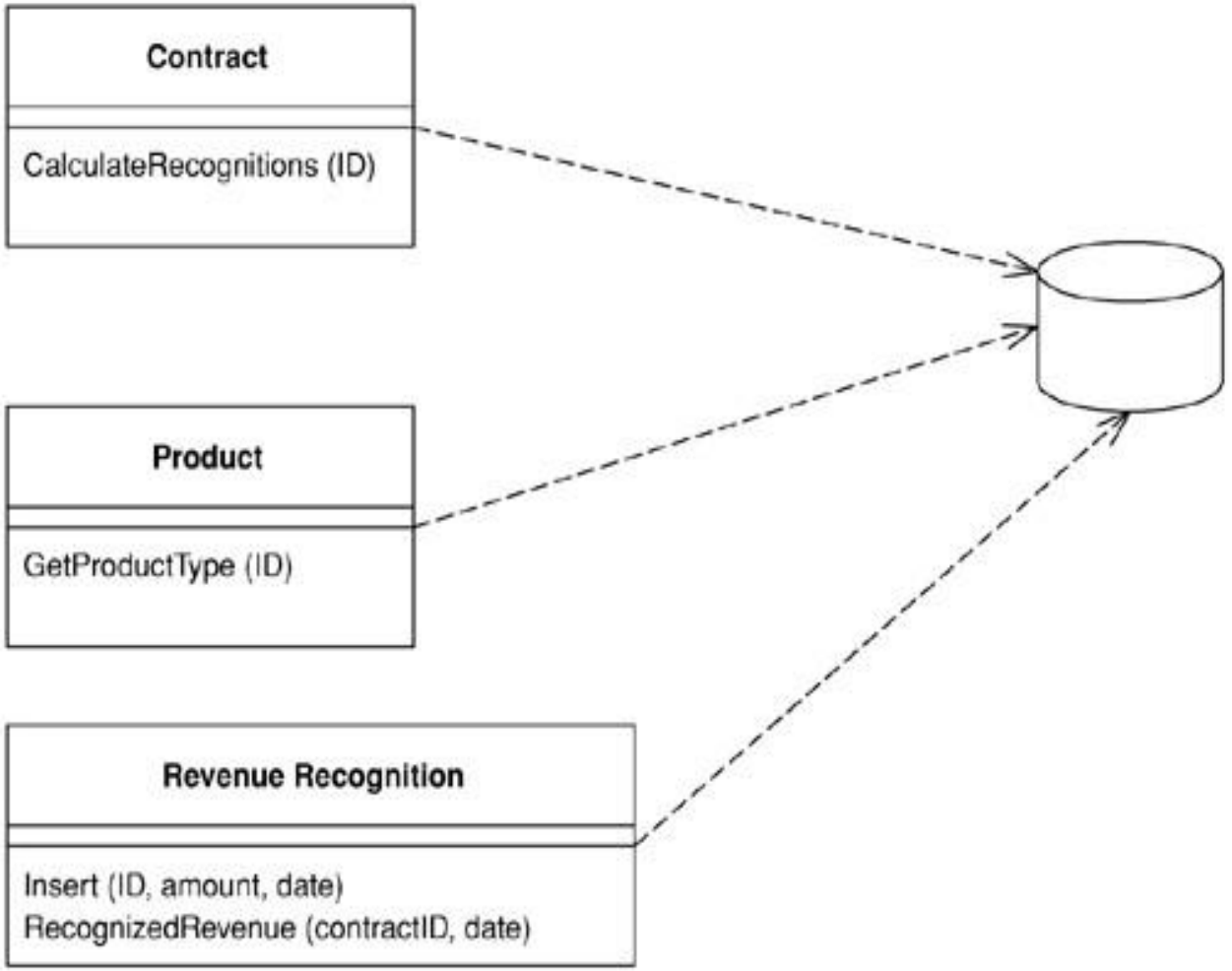


From: “Patterns of Enterprise Application Architecture”, Martin Fowler *et al.*

Domain Model: Pros vs. Cons

How it works	Building an object model of the domain that incorporates both behavior and data
Pros	• Natural and easily understanding • Can handle complicated business • Easily reuse
Cons	• Expensive at beginning • Hard to maintain
When to use	• Complicated, and • Large systems

Table Model

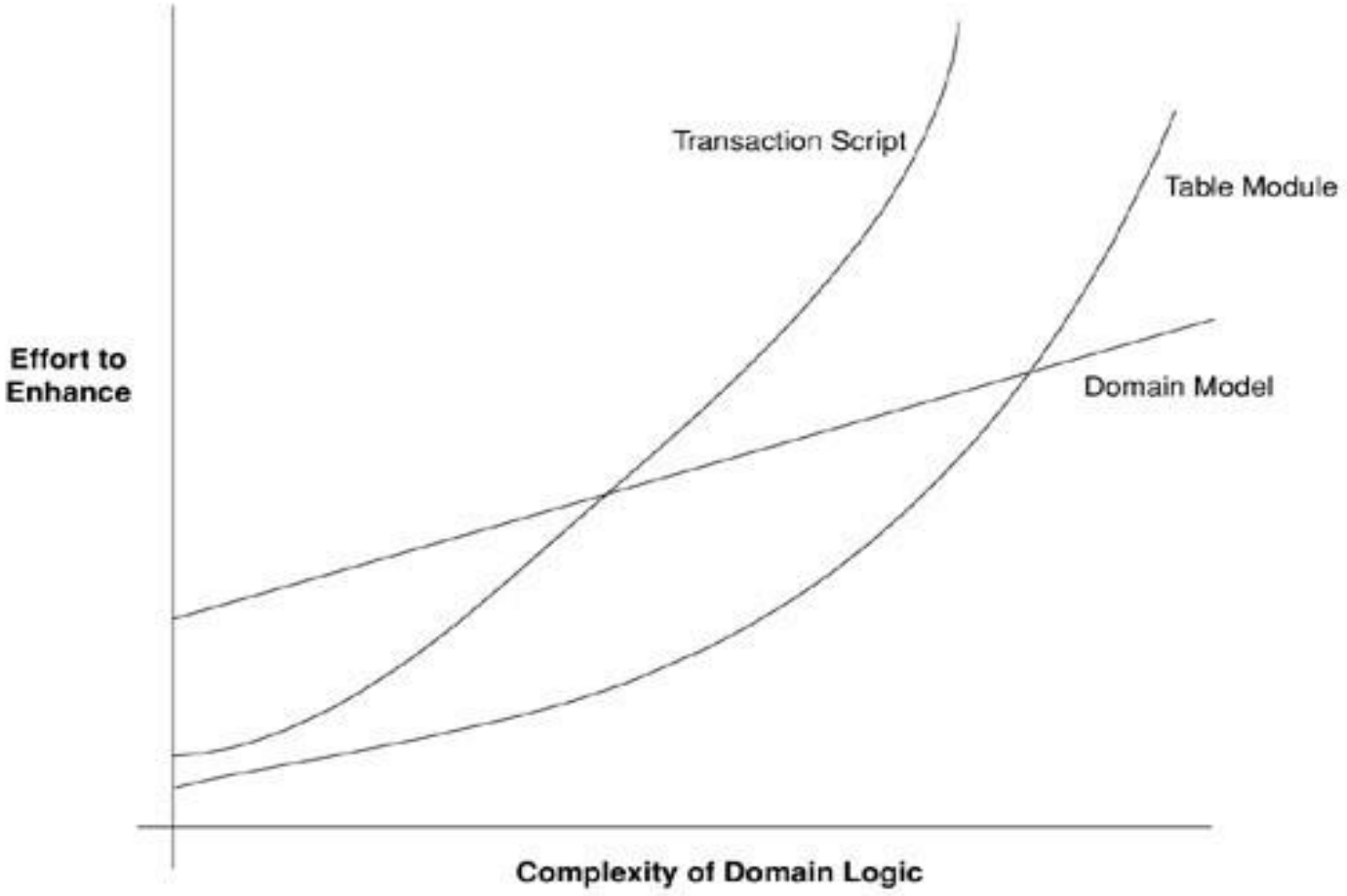


From: “Patterns of Enterprise Application Architecture”, Martin Fowler *et al.*

Table Model: Pros vs. Cons

How it works	A single instance that handles the business logic for all rows in database table or view
Pros	• Natural and easily understanding • Easily reuse for simple business • Less resource demand
Cons	• Hard to handle complicated business • Hard to handle concurrency • Poor performance
When to use	• Simple systems • Less change • want reuse

Put it All Together

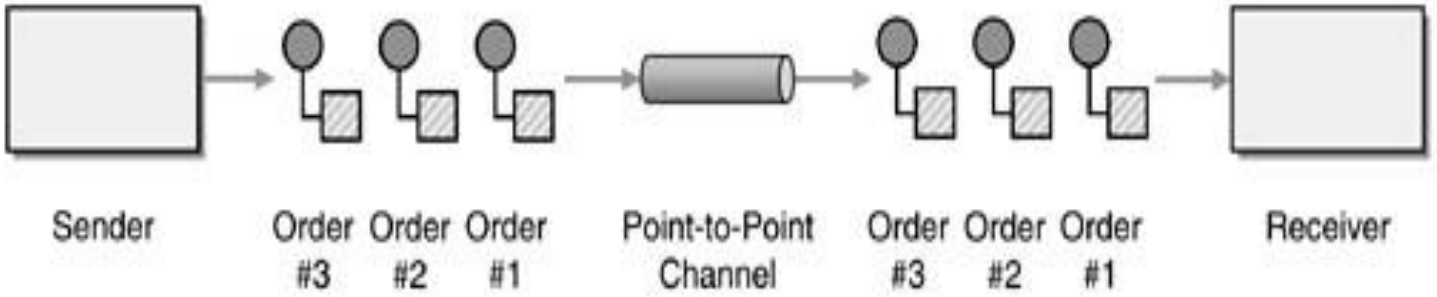


Messaging Channel Patterns

- Point-to-Point Channel Pattern
  - One-to-One
- Publish-Subscribe Channel Pattern
  - **One-to-Many**
  - Many-to-One
  - Many-to-Many

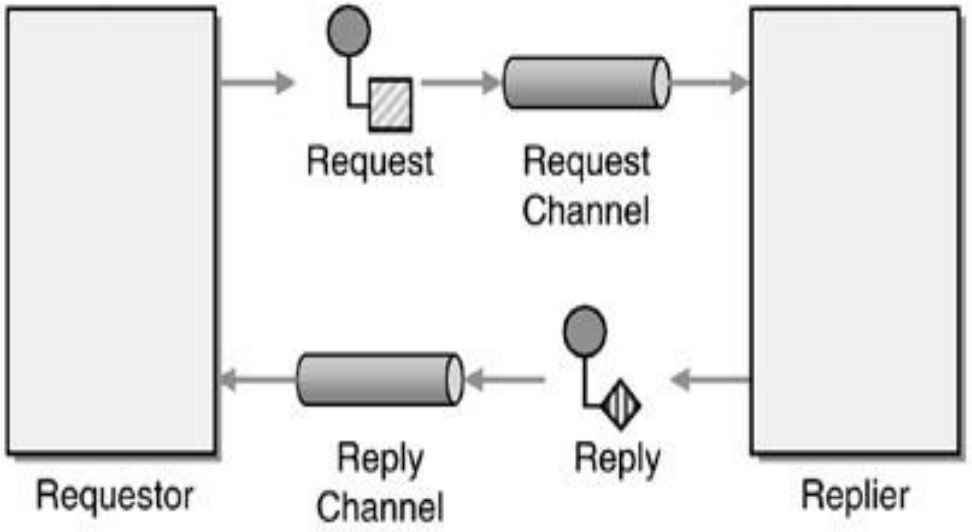
Point-to-Point (P2P) Channel

Send the message on a *Point-to-Point Channel*, which ensures that only one receiver will receive a particular message.



Request-Reply Using P2P

Send a pair of *Request-Reply* messages, each on its own channel.



P2P: Pros vs. Cons

How it works	Two software components can communicate asynchronously
Pros	• No shaking hands between the sender and receiver • The sender can send-and-forget • Good performance and scalability
Cons	• The sender cannot get a response immediately • The sending can go wrong
When to use	• When you don't expect a reply at runtime • when sending performance is important

Publish-Subscribe (Pub/Sub) Channel

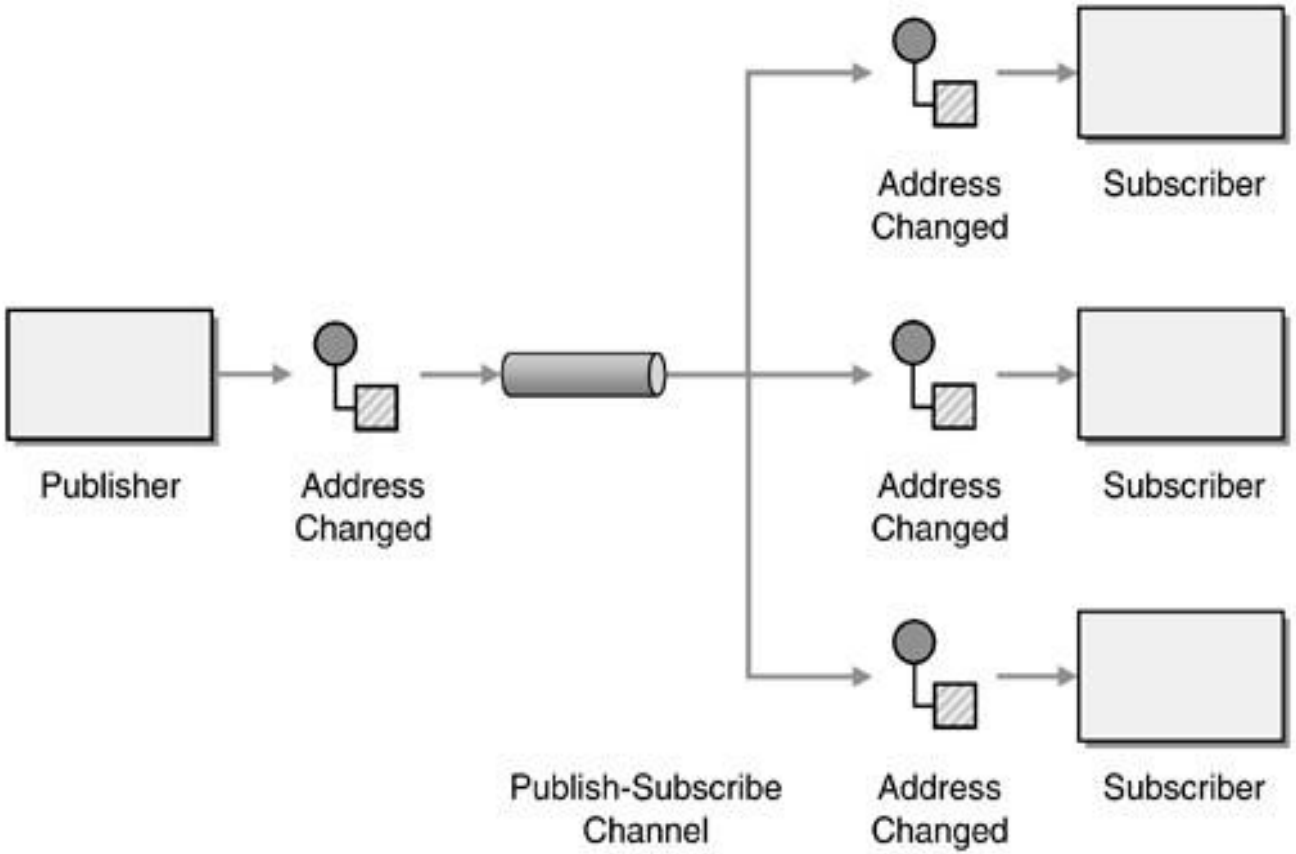


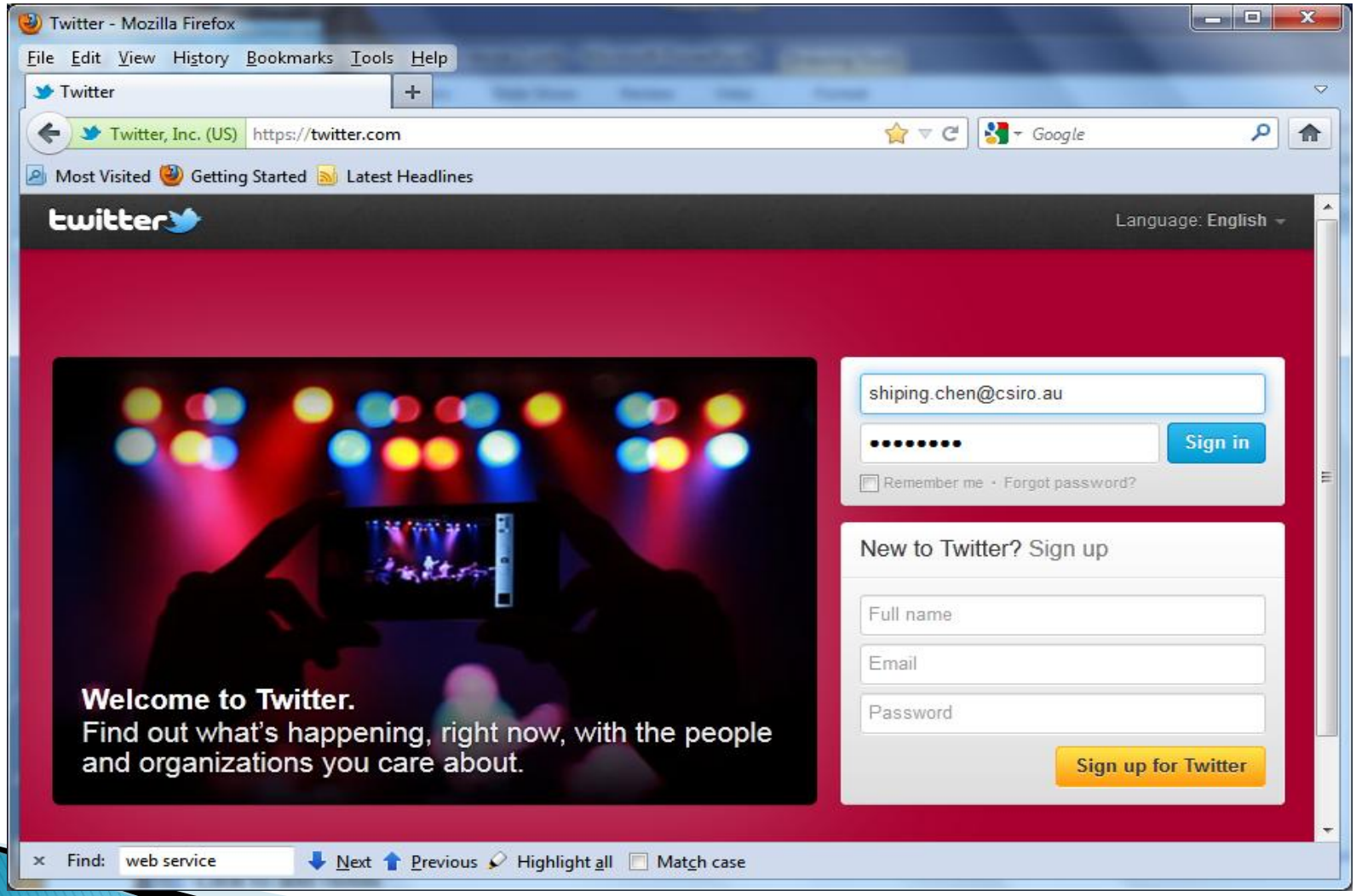
Table Model: Pros vs. Cons

How it works	An entity sends messages to multiple recivers via a topic

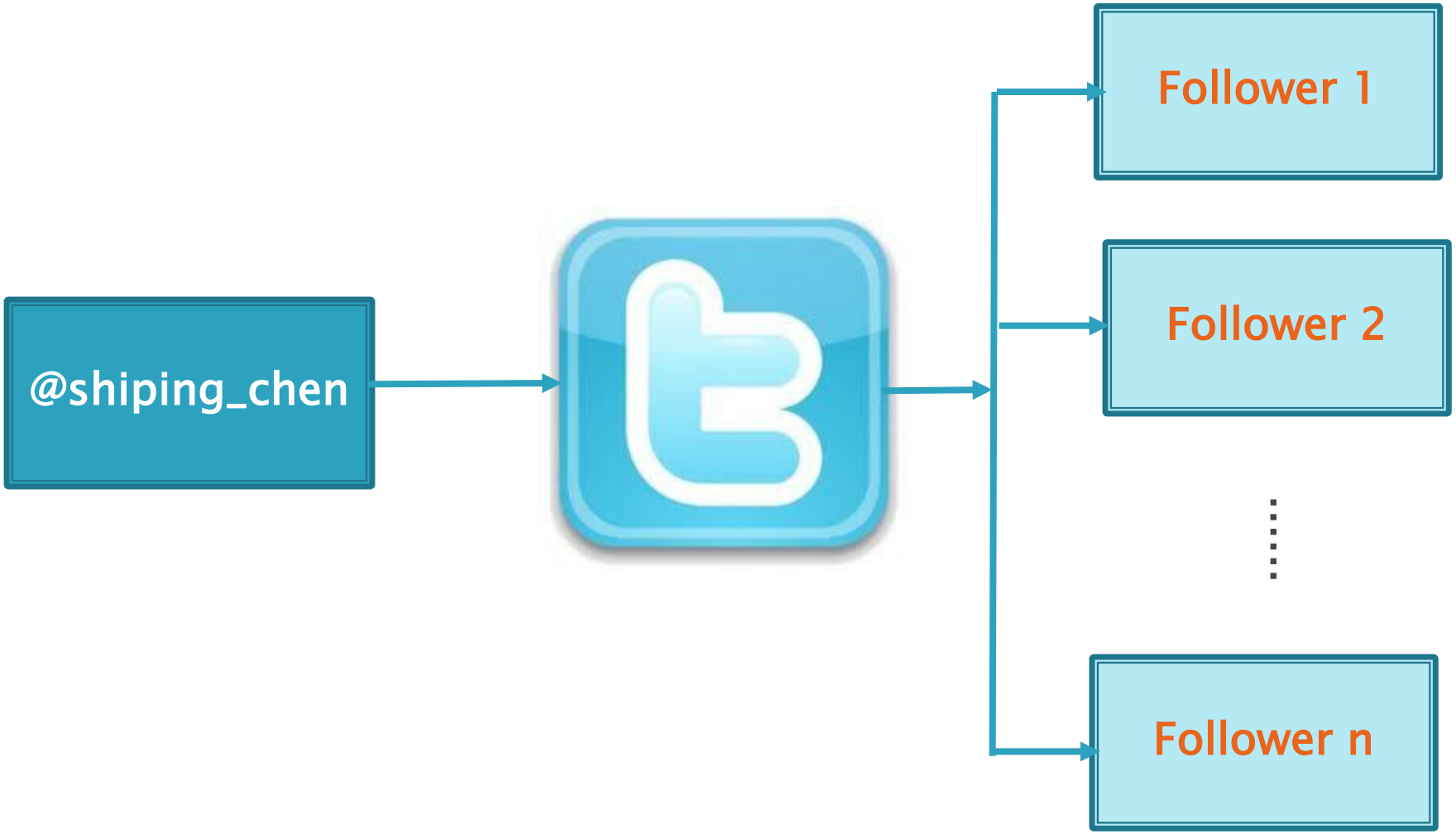
Pros	• Senders and receivers don't need to know each other
	• The sender can send and forget
	• Enable different topologies:
	  • One-to-Many
	  • Many-to-Many

Cons	• No response received at runtime • Cannot guarantee delivery of messages
When to use	• One-to-many or many-to-many communication • Decoupling the sender and receivers • High performance & scalability for massive messaging

Twitter: A Type Application of Pub/Sub



How Does it work?



Component Design Patterns

- Creational design patterns
  - How to create objects?
- Structural design patterns:
  - How to construct a system?
- Behavioural patterns?
  - How to control objects at runtime?

Creational design patterns

- Singleton
- Factory
- Factory Method
- Abstract Factory
- Builder
- Prototype
- Object Pool

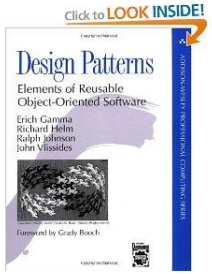
Structural design patterns

- Adapter
- Bridge
- Composite
- Decrorator
- Flyweight
- Proxy

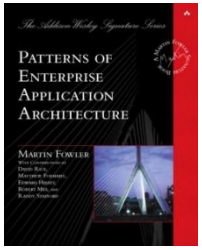
Behavioral Design Patterns

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Memento
- Observer
- Template Method
- Visitor
- Null Object

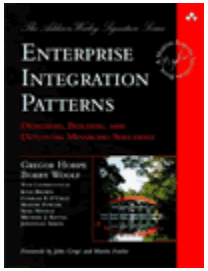
Further Reading



- "Design Patterns"
  - Erich Gamma, Richard Helm, Ralph Johnson and John Glissades (GOF)



- "Patterns of Enterprise Application Architecture"
  - Martin Fowler *et al.*



- "Enterprise Integration Patterns"
  - Gregor Hohpe and Bobby Woolf

Task 2 of Assignment

- Please design the car sale system based on the requirements developed in Task 1?

- Advices
  - Using UML if possible
  - Try to reuse some design patterns
  - But you don't have to use all design patterns learnt today!