CS 525 - ASD

Advanced Software Development

MS.CS Program

Department of Computer Science

Rene de Jong, MsC.

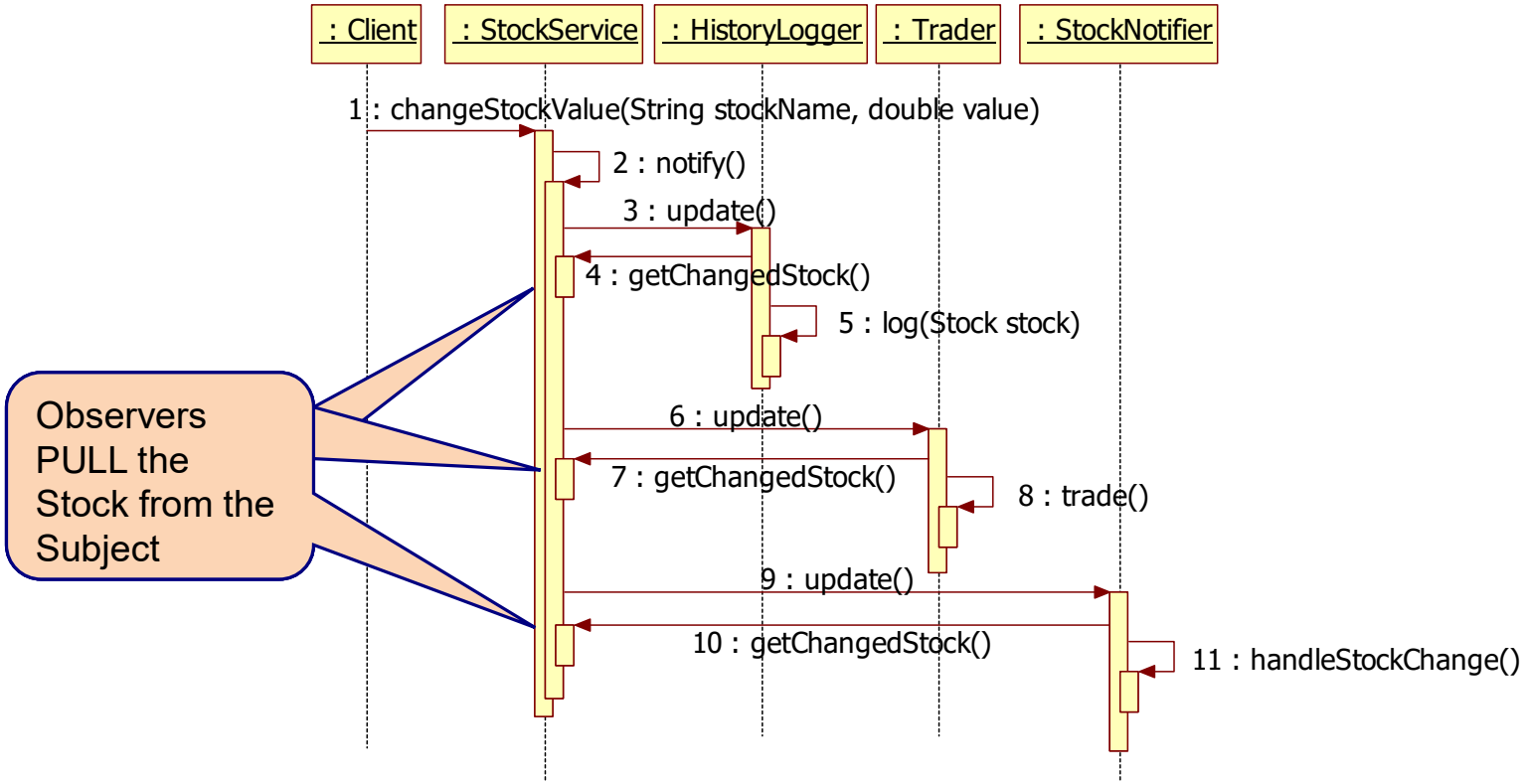
Maharishi University
OF MANAGEMENT

CS 525 - ASD

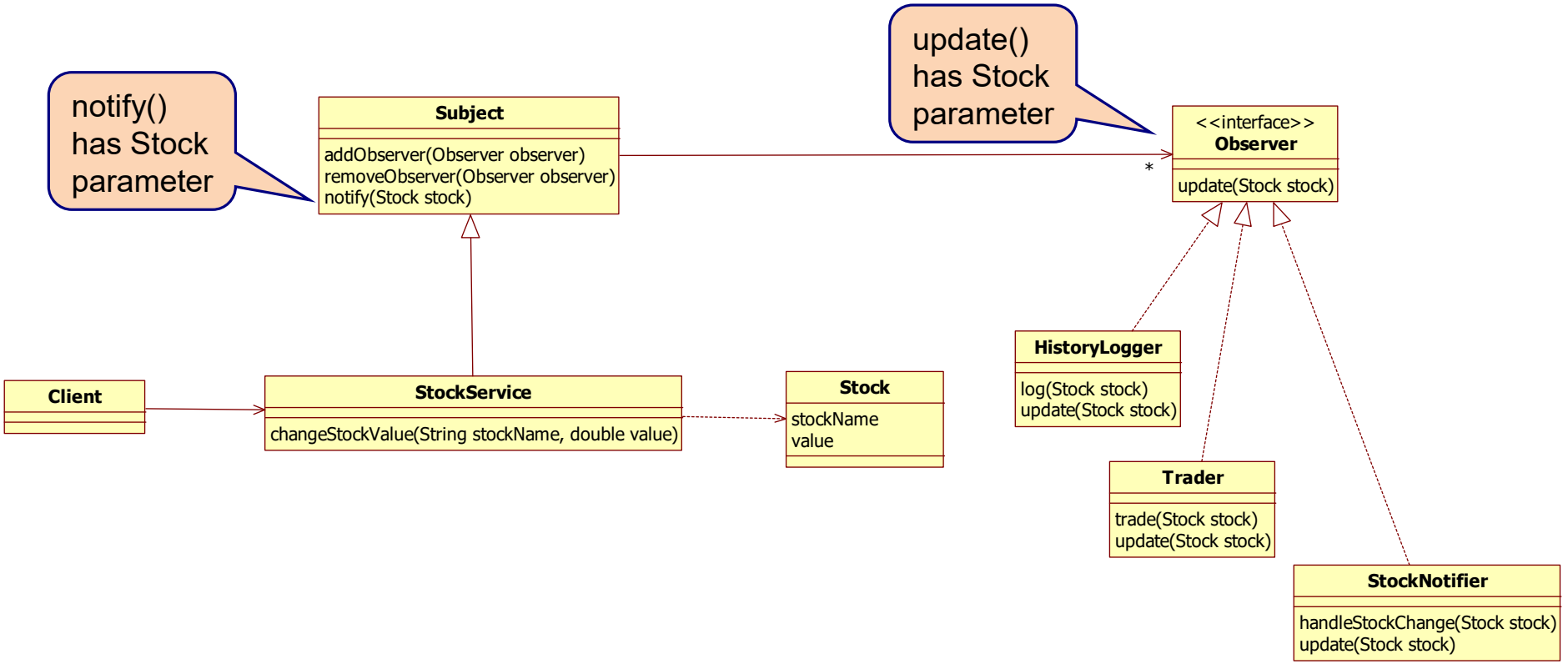
Advanced Software Development

Maharishi University
OF MANAGEMENT

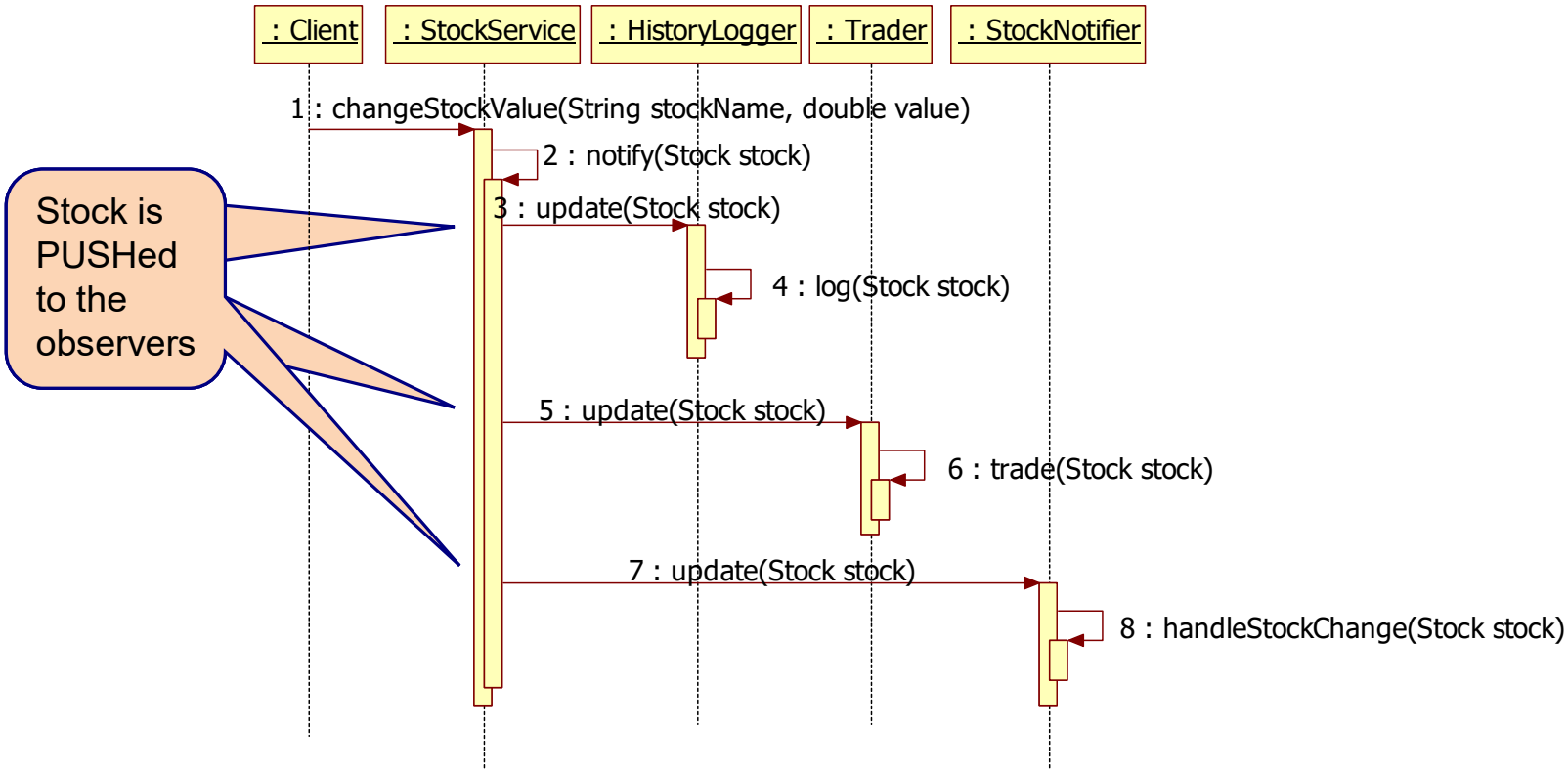
Pull model



Push model



Push model



Subject, IObserver and Stock

```
public class Subject  {
  private Collection<IObserver> observerlist = new ArrayList<IObserver>();

  public void addObserver(IObserver observer){
    observerlist.add(observer);
  }

  public void donotify(Stock stock){
    for (IObserver observer: observerlist){
      observer.update(stock);
    }
  }
}
```

```
public interface IObserver {
  public void update(Stock stock);
}
```

```
public class Stock {
  private String stockName;
  private double value;
  ...
}
```

HistoryLogger & Trader

```
public class HistoryLogger implements IObserver {

  public void log(Stock stock) {
    System.out.println("HistoryLogger log stock :" + stock);
  }

  @Override
  public void update(Stock stock) {
    log(stock);
  }
}
```

```
public class Trader implements IObserver{

  public void trade(Stock stock) {
    System.out.println("Trader trade stock :" + stock);
  }

  @Override
  public void update(Stock stock) {
    trade(stock);
  }
}
```

StockNotifier

```
public class StockNotifier implements IObserver {

  public void handleStockChange(Stock stock) {
    System.out.println("StockNotifier handle stock :" + stock);
  }

  @Override
  public void update(Stock stock) {
    handleStockChange(stock);
  }
}
```

StockService and Application

```
public class StockService extends Subject{

  public void changeStockValue(String stockName, double value) {
    Stock stock = new Stock(stockName, value);
    donotify(stock);
  }
}
```

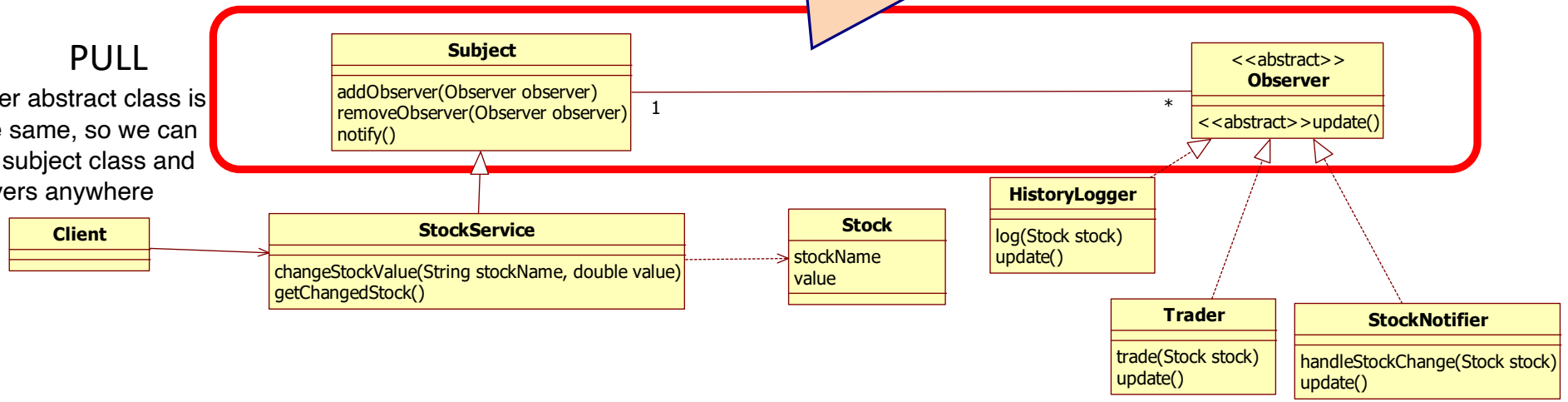
```
public class Application {

  public static void main(String[] args) {
    StockService stockService = new StockService();
    HistoryLogger historyLogger= new HistoryLogger();
    Trader trader = new Trader();
    StockNotifier stockNotifier = new StockNotifier();

    stockService.addObserver(historyLogger);
    stockService.addObserver(trader);
    stockService.addObserver(stockNotifier);

    stockService.changeStockValue("AMZN", 2310.80);
    stockService.changeStockValue("MSFT", 890.45);
  }
}
```

Difference push and pull

Generic for any observer

PULL

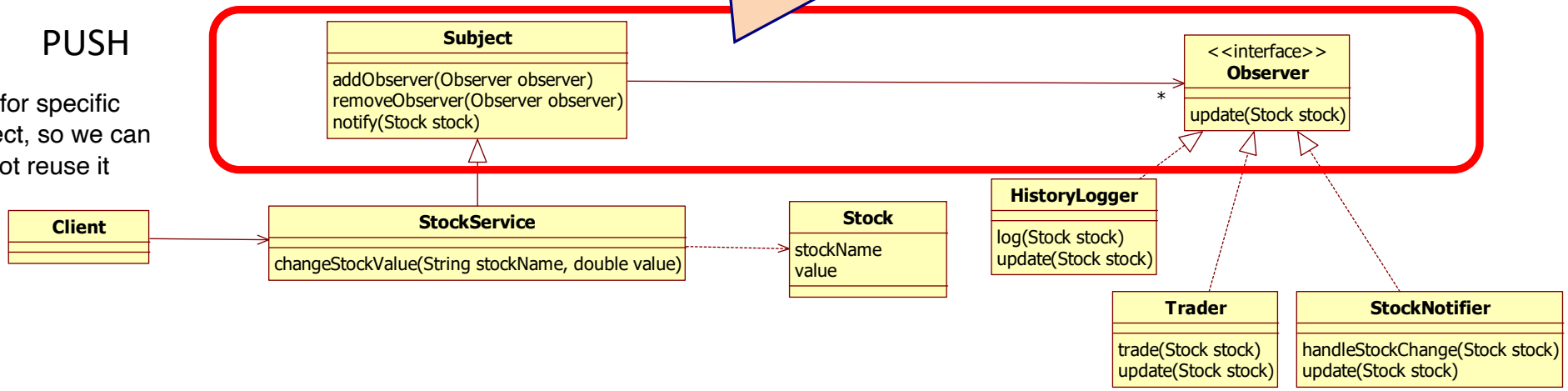
any observer abstract class is always the same, so we can reuse this subject class and observers anywhere



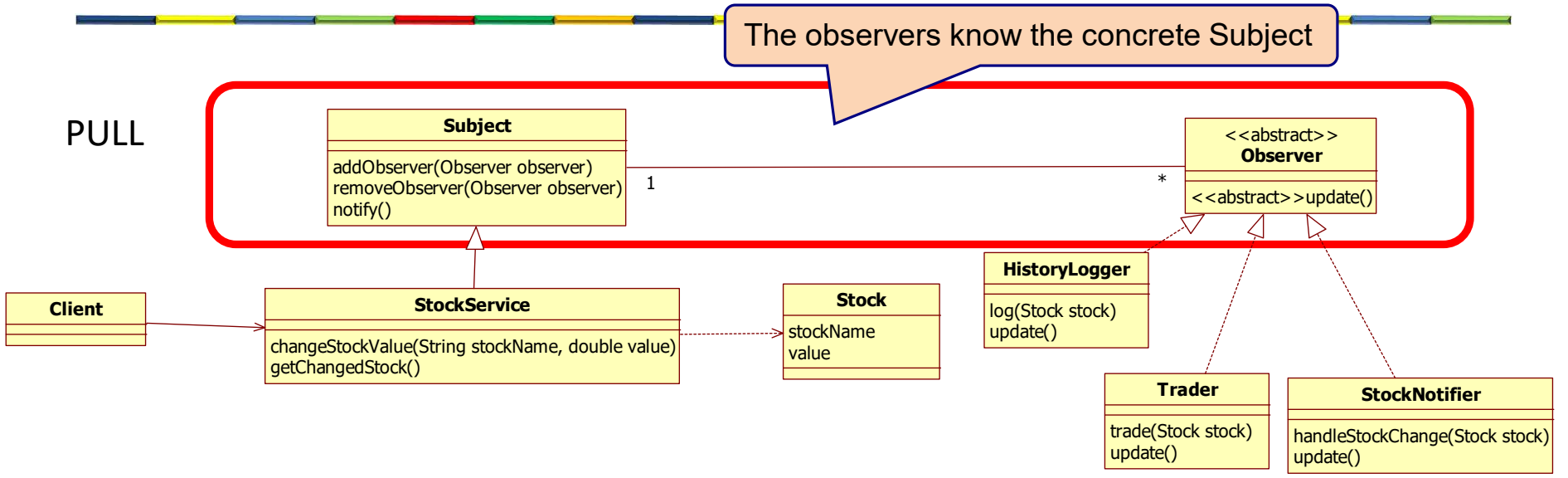
Specific for Stock observers

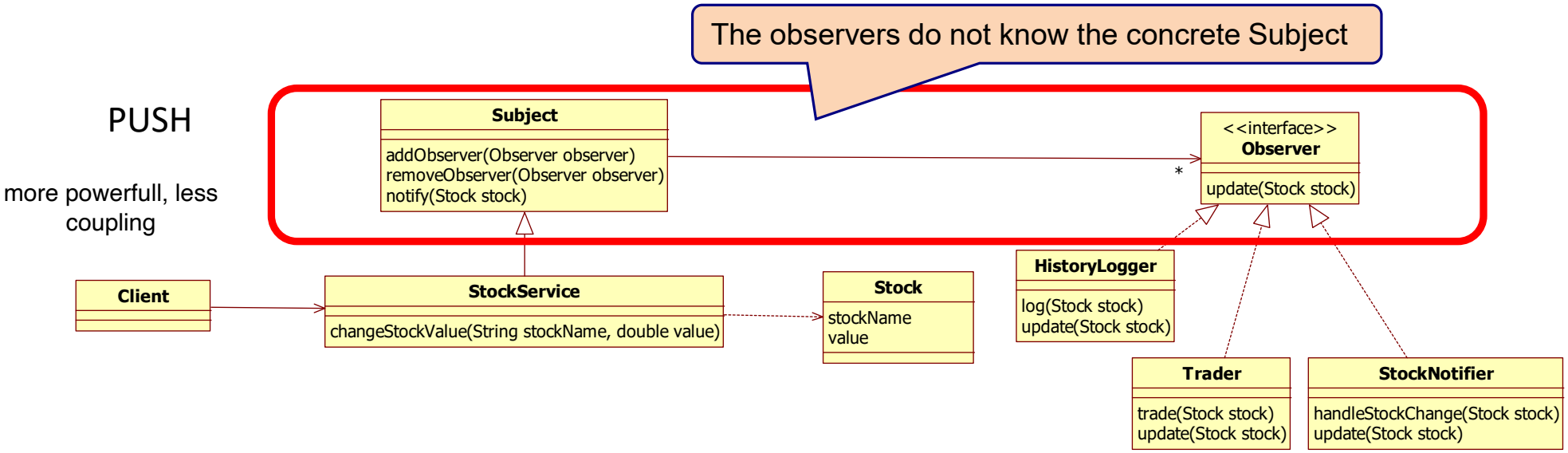
PUSH

is for specific subject, so we can not reuse it

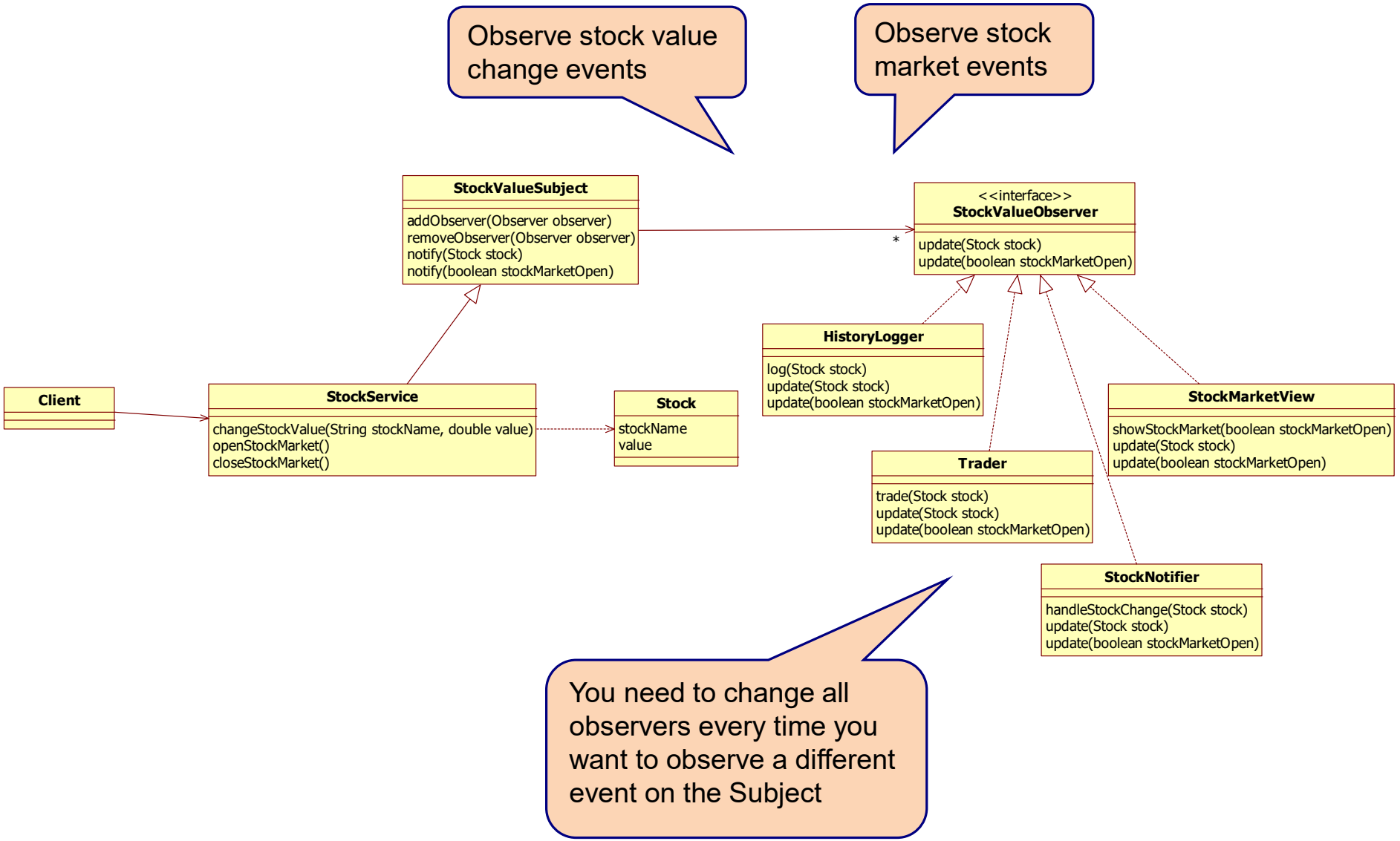


Difference push and pull

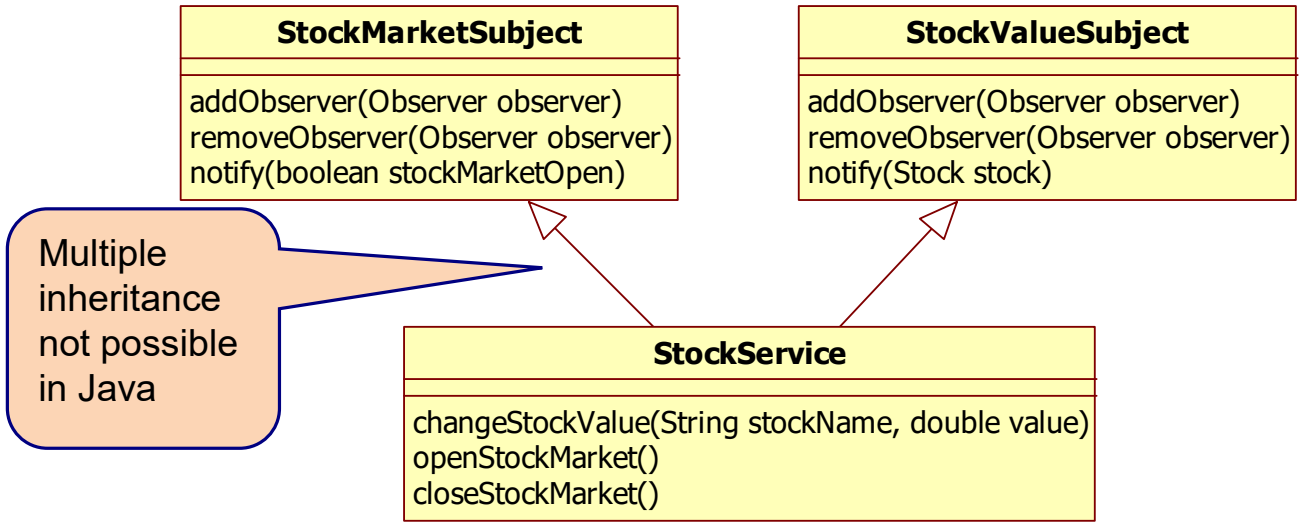




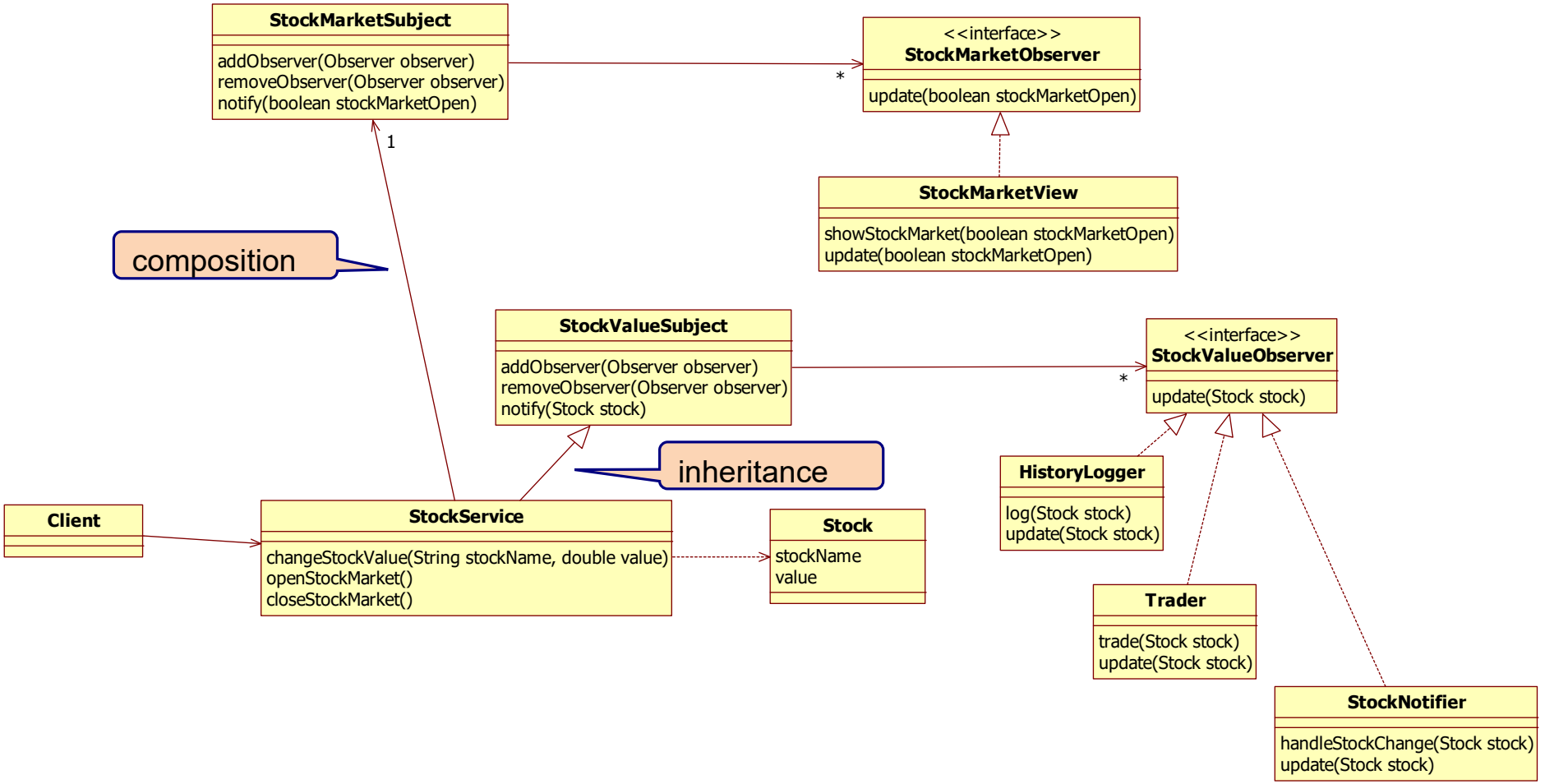
Observing multiple events



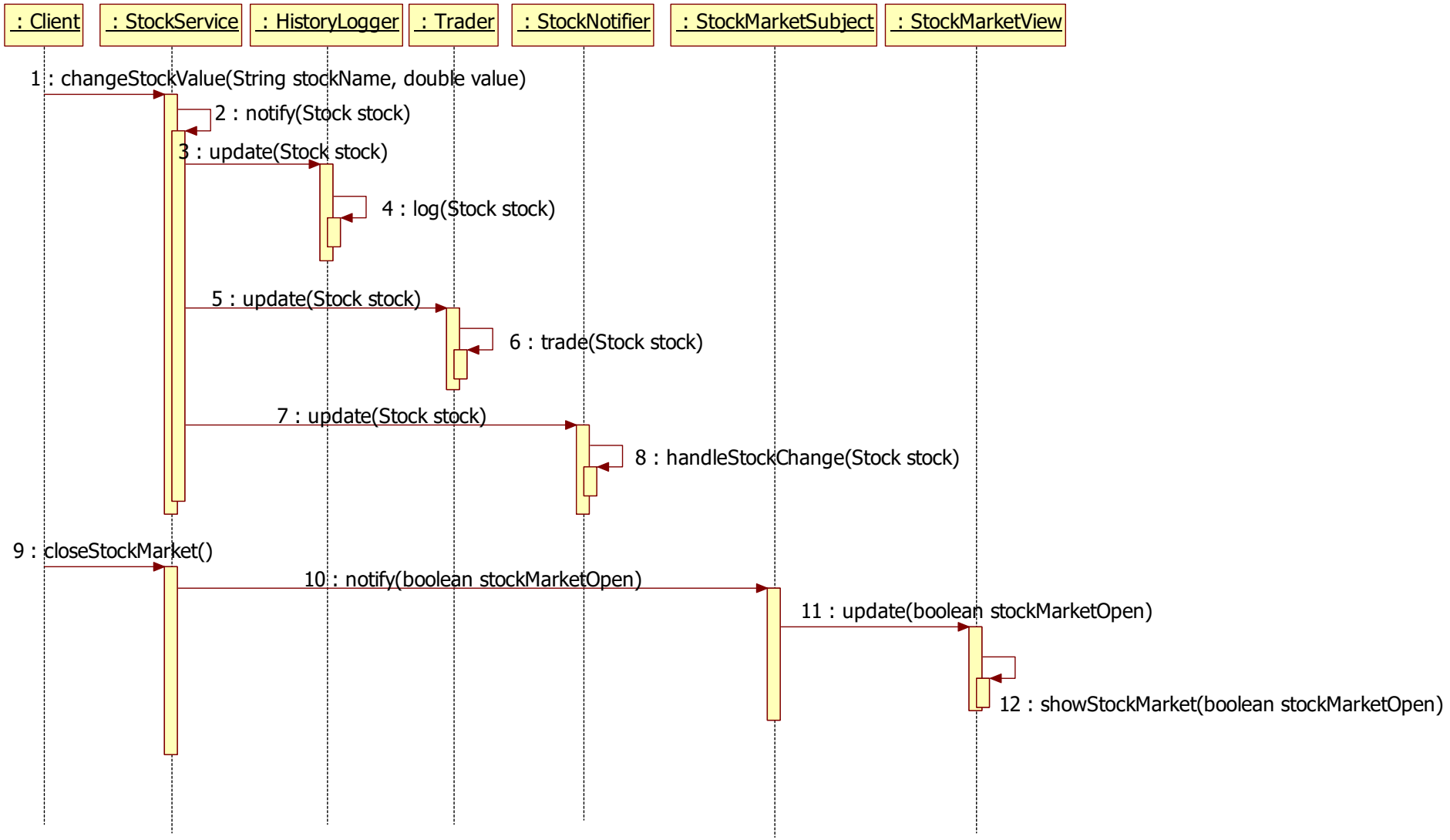
Multiple subjects



Multiple Subjects



Multiple Subjects



StockService

```
public class StockService extends StockValueSubject{
  private boolean stockMarketOpen=false;
  private StockMarketSubject stockMarketSubject;

  public void changeStockValue(String stockName, double value) {
    Stock stock = new Stock(stockName, value);
    donotify(stock);
  }

  public void openStockMarket() {
    stockMarketOpen=true;
    stockMarketSubject.donotify(stockMarketOpen);
  }

  public void closeStockMarket() {
    stockMarketOpen=false;
    stockMarketSubject.donotify(stockMarketOpen);
  }

  public StockMarketSubject getStockMarketSubject() {
    return stockMarketSubject;
  }

  public void setStockMarketSubject(StockMarketSubject stockMarketSubject) {
    this.stockMarketSubject = stockMarketSubject;
  }
}
```

The Subjects and observer interfaces

```
public class StockValueSubject  {
  private Collection<StockValueObserver> observerlist = new ArrayList<StockValueObserver>();

  public void addObserver(StockValueObserver observer){
    observerlist.add(observer);
  }

  public void donotify(Stock stock){
    for (StockValueObserver observer: observerlist){
      observer.update(stock);
    }
  }
}
```

```
public interface StockValueObserver {
  public void update(Stock stock);
}
```

```
public class StockMarketSubject  {
  private Collection<StockMarketObserver> observerlist = new ArrayList<StockMarketObserver>();

  public void addObserver(StockMarketObserver observer){
    observerlist.add(observer);
  }

  public void donotify(boolean stockMarketOpen){
    for (StockMarketObserver observer: observerlist){
      observer.update(stockMarketOpen);
    }
  }
}
```

```
public interface StockMarketObserver {
  public void update(boolean stockMarketOpen);
}
```

The concrete observers

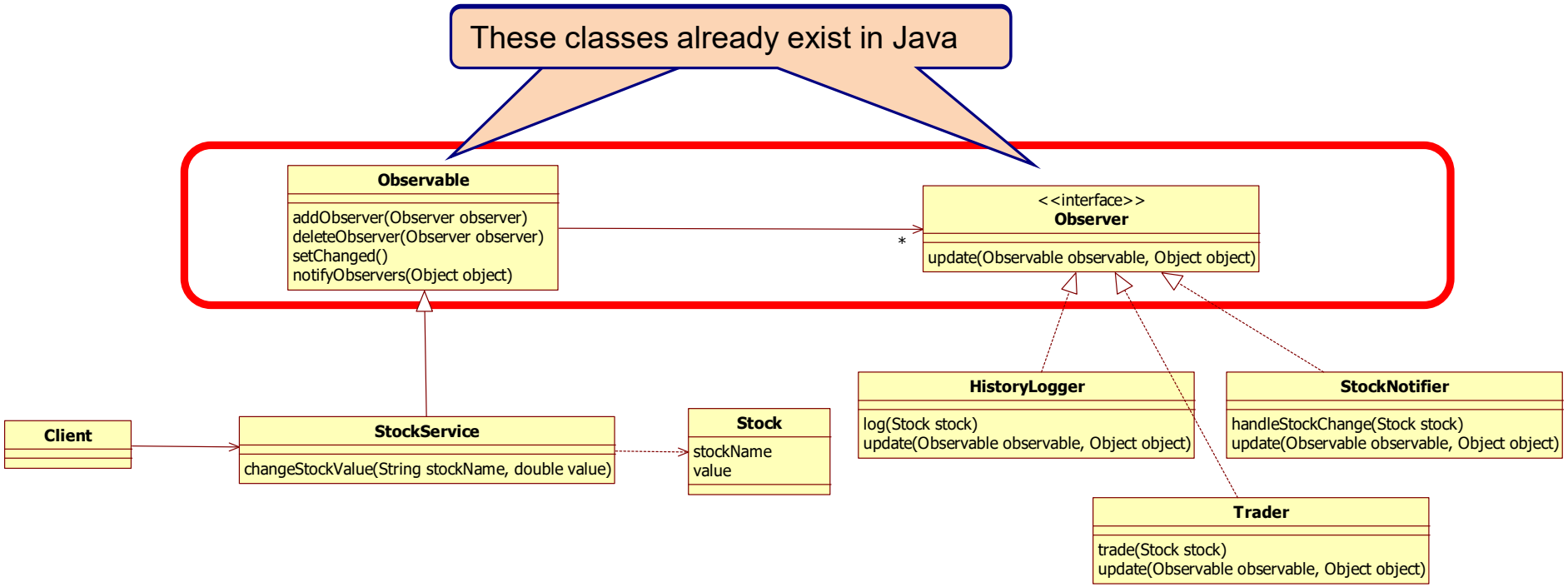
```
public class HistoryLogger implements IObserver {

  public void log(Stock stock) {
    System.out.println("HistoryLogger log stock :" + stock);
  }

  @Override
  public void update(Stock stock) {
    log(stock);
  }
}
```

```
public class StockMarketView implements StockMarketObserver{

  @Override
  public void update(boolean stockMarketOpen) {
    showStockMarket( stockMarketOpen);
  }

  public void showStockMarket(boolean stockMarketOpen) {
    if (stockMarketOpen) {
      System.out.println("The stock market is open");
    }
    else {
      System.out.println("The stock market is closed");
    }
  }
}
```

Application

```
public class Application {

  public static void main(String[] args) {
    StockService stockService = new StockService();
    HistoryLogger historyLogger= new HistoryLogger();
    Trader trader = new Trader();
    StockNotifier stockNotifier = new StockNotifier();

    stockService.addObserver(historyLogger);
    stockService.addObserver(trader);
    stockService.addObserver(stockNotifier);

    StockMarketSubject stockMarketSubject = new StockMarketSubject();
    StockMarketView stockMarketView = new StockMarketView();
    stockMarketSubject.addObserver(stockMarketView);
    stockService.setStockMarketSubject(stockMarketSubject);

    stockService.openStockMarket();
    stockService.changeStockValue("AMZN", 2310.80);
    stockService.changeStockValue("MSFT", 890.45);
    stockService.closeStockMarket();
  }
}
```

Observer in Java



HistoryLogger and Trader

```
import java.util.Observable;
import java.util.Observer;

public class HistoryLogger implements Observer {

  public void log(Stock stock) {
    System.out.println("HistoryLogger log stock :" + stock);
  }

  public void update(Observable observable, Object stock) {
    log((Stock) stock);
  }
}
```

```
import java.util.Observable;
import java.util.Observer;

public class Trader implements Observer{

  public void trade(Stock stock) {
    System.out.println("Trader trade stock :" + stock);
  }


  public void update(Observable observable, Object stock) {
    trade((Stock) stock);
  }
}
```

StockNotifier and StockService

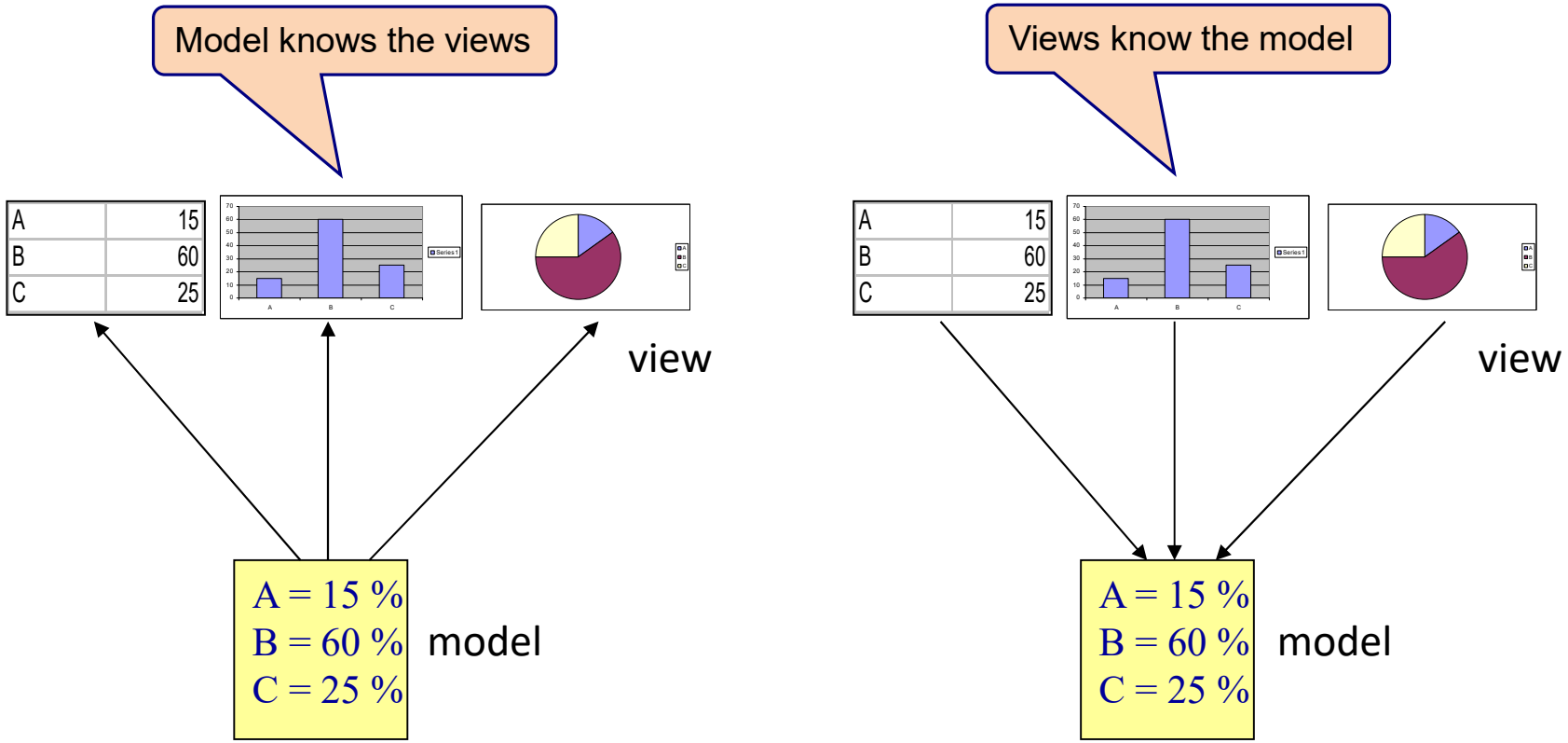
```
import java.util.Observable;
import java.util.Observer;

public class StockNotifier implements Observer {

  public void handleStockChange(Stock stock) {
    System.out.println("StockNotifier handle stock :" + stock);
  }

  public void update(Observable observable, Object stock) {
    handleStockChange((Stock) stock);
  }
}
```

```
import java.util.Observable;

public class StockService extends Observable{

  public void changeStockValue(String stockName, double value) {
    Stock stock = new Stock(stockName, value);
    setChanged();
    notifyObservers(stock);
  }
}
```

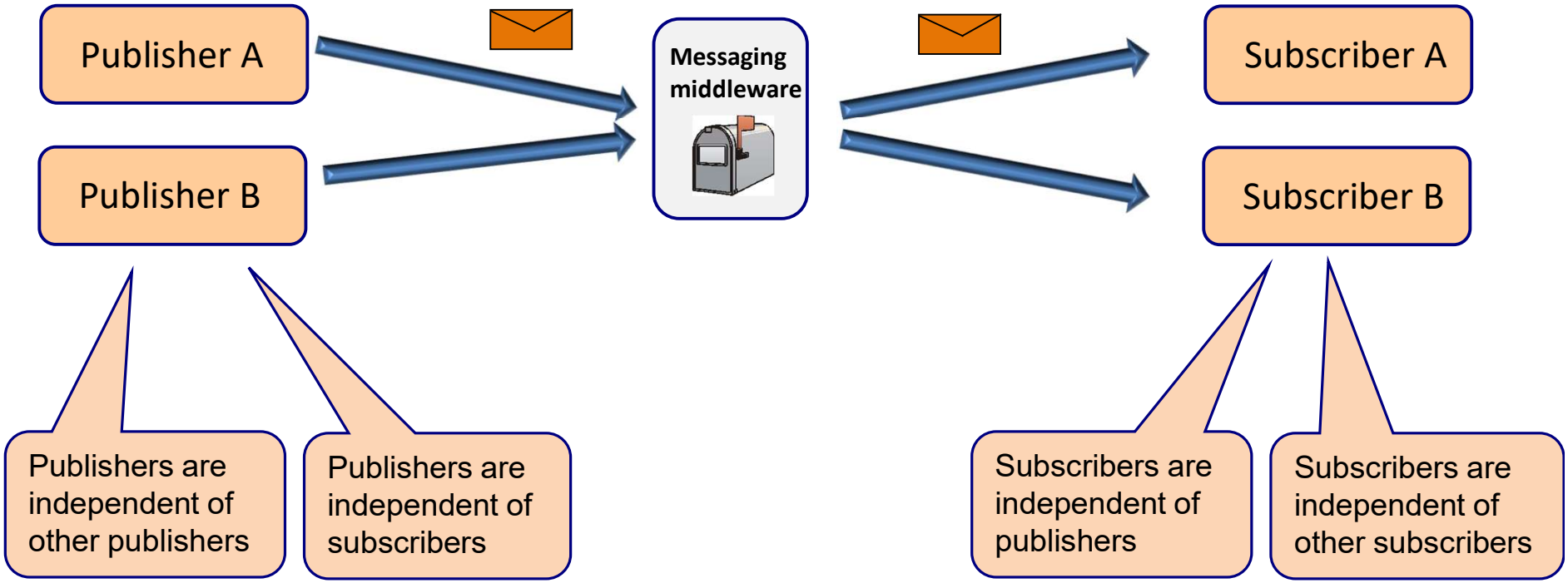
Application

```
public class Application {

  public static void main(String[] args) {
    StockService stockService = new StockService();
    HistoryLogger historyLogger= new HistoryLogger();
    Trader trader = new Trader();
    StockNotifier stockNotifier = new StockNotifier();

    stockService.addObserver(historyLogger);
    stockService.addObserver(trader);
    stockService.addObserver(stockNotifier);

    stockService.changeStockValue("AMZN", 2310.80);
    stockService.changeStockValue("MSFT", 890.45);
  }
}
```

Model View Controller (MVC)

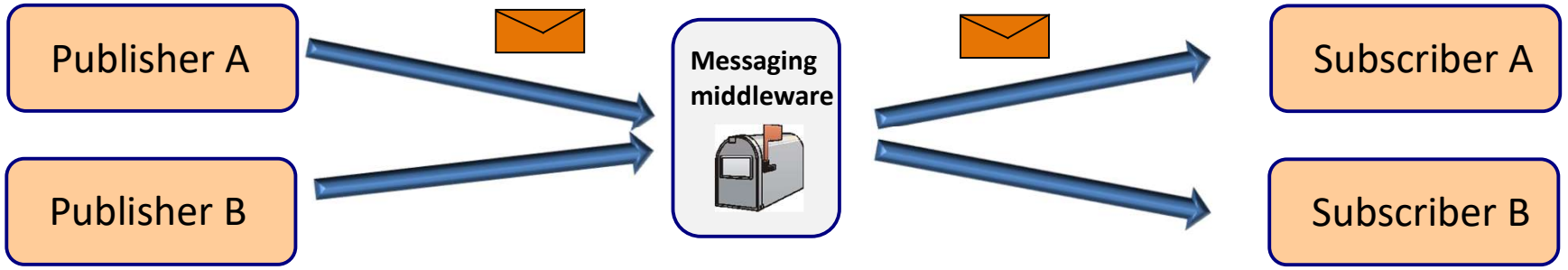


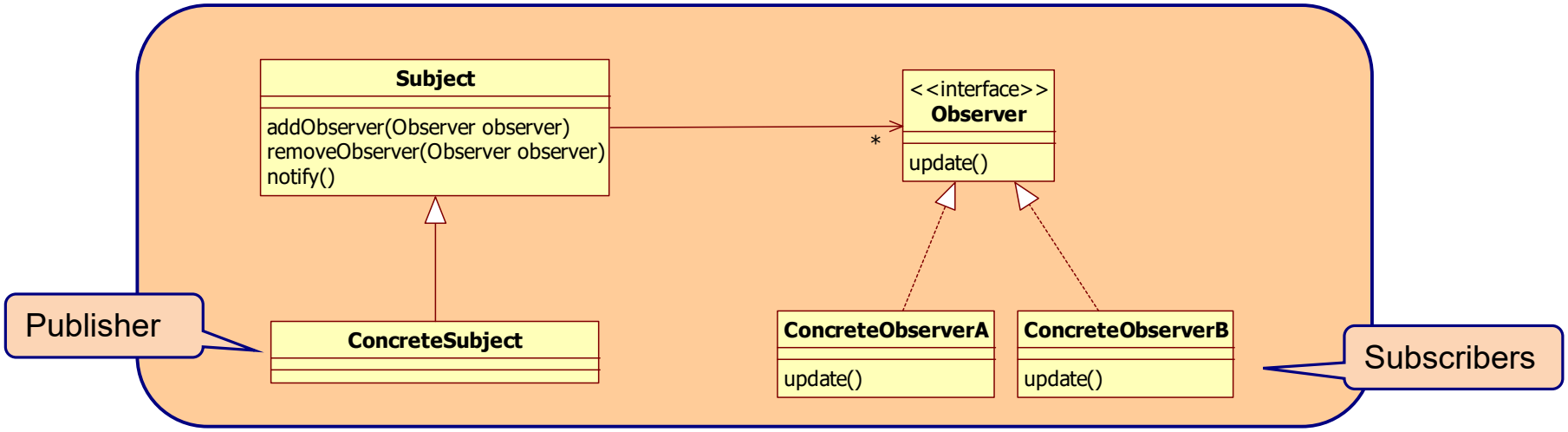
Publish-subscribe (pub-sub)



Loose coupling

Publish-subscribe (pub-sub)





Loose coupling

Observer pattern

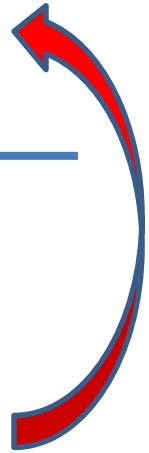
- What problem does it solve?
  - When a change to one object requires changing others, and you don't know how many objects need to be changed.
  - When an object should be able to notify other objects without making assumptions about who these objects are. In other words, you don't want these objects tightly coupled.

Main point

- The observer pattern is all about loose coupling between subject and observers. You can add new observers without changing the subject.
- By transcending one contacts the unbounded unified field which contains all intelligence of nature.

Connecting the parts of knowledge with the wholeness of knowledge

1. The observer pattern decouples the observers from the subject.
2. In the Model-View-Controller pattern, the model publish updates to the views without knowing anything about these views.



3. **Transcendental consciousness** is a field all possibilities.
4. **Wholeness moving within itself:** In unity consciousness the simple state of wholeness of the Self, is an everyday living reality, resulting in a life full of bliss, and maximum efficiency with least effort.