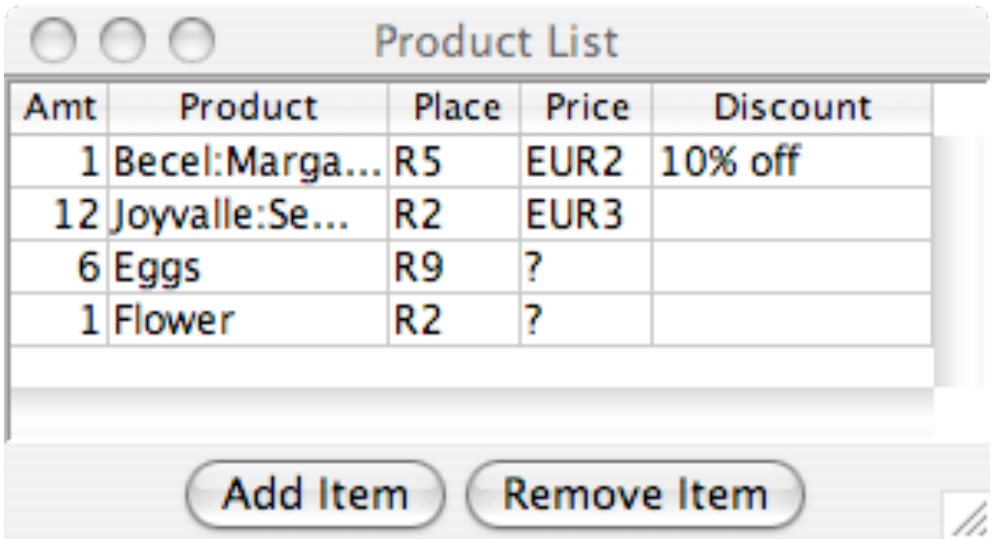
Abstracting Connection Volatility Through Tagged Futures

Johan Fabry - PLEIAD Lab - DCC - UChile
jfabry @ dcc

Pleiad

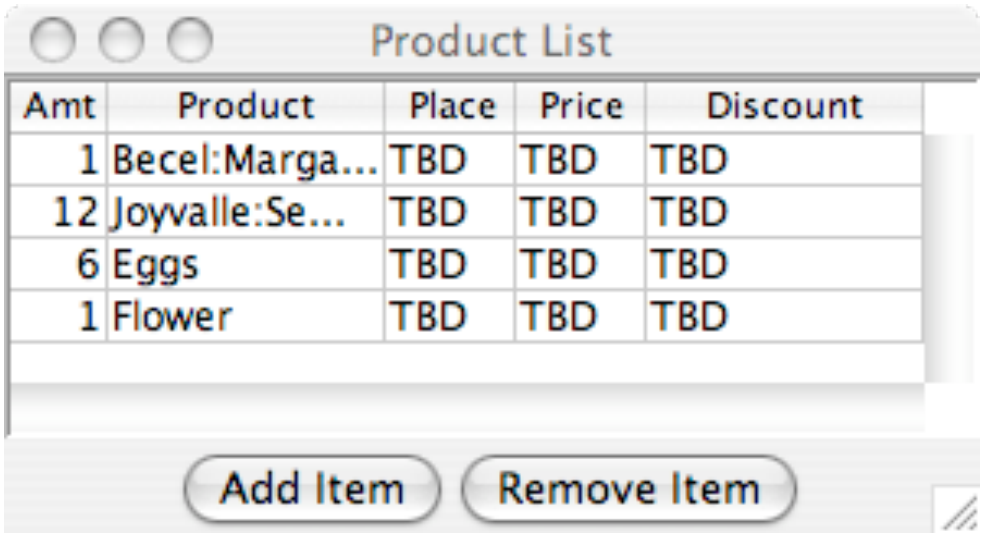
Departamento de Ciencias de la Computación
UNIVERSIDAD DE CHILE

What do we want?



Online

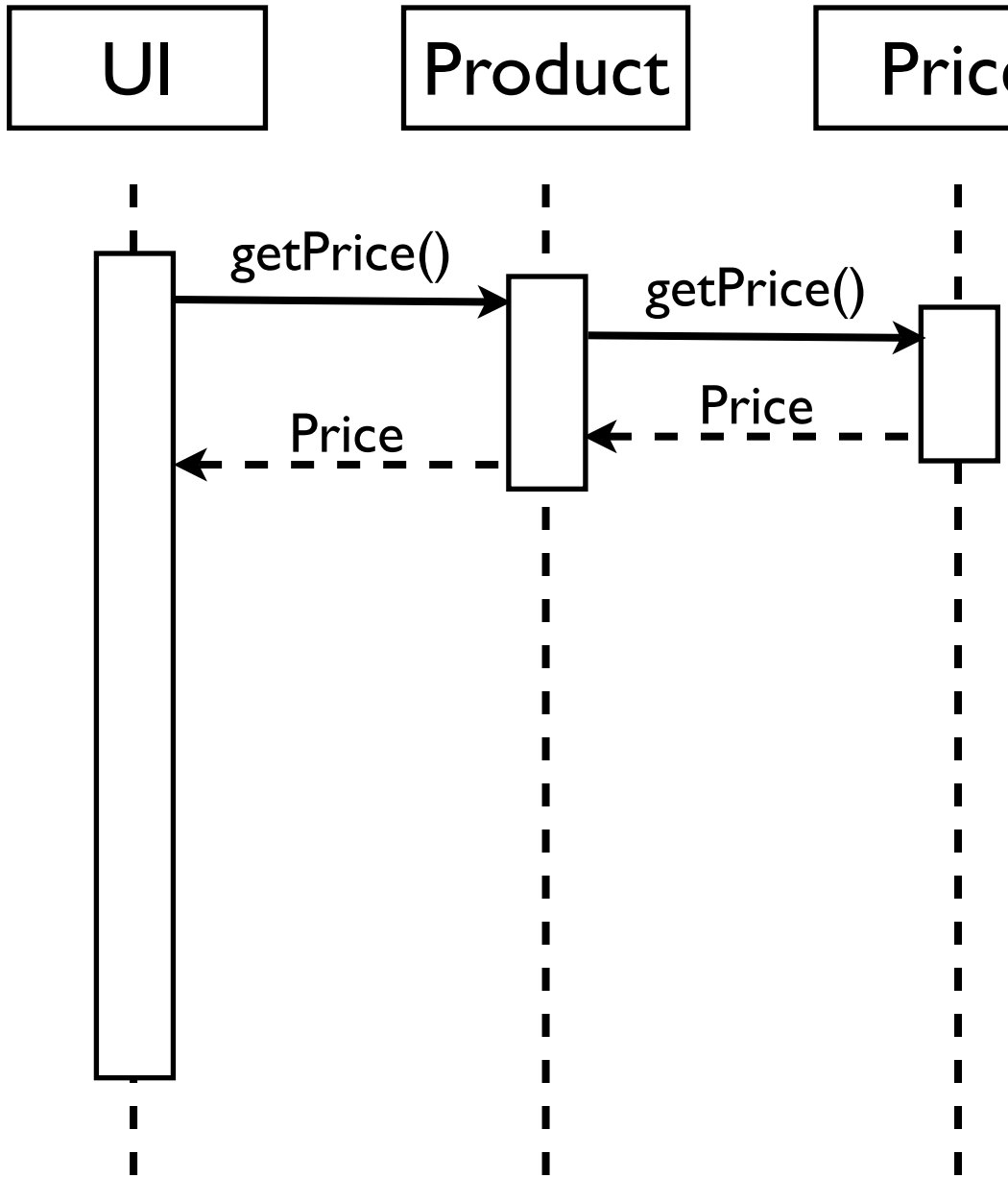
Offline



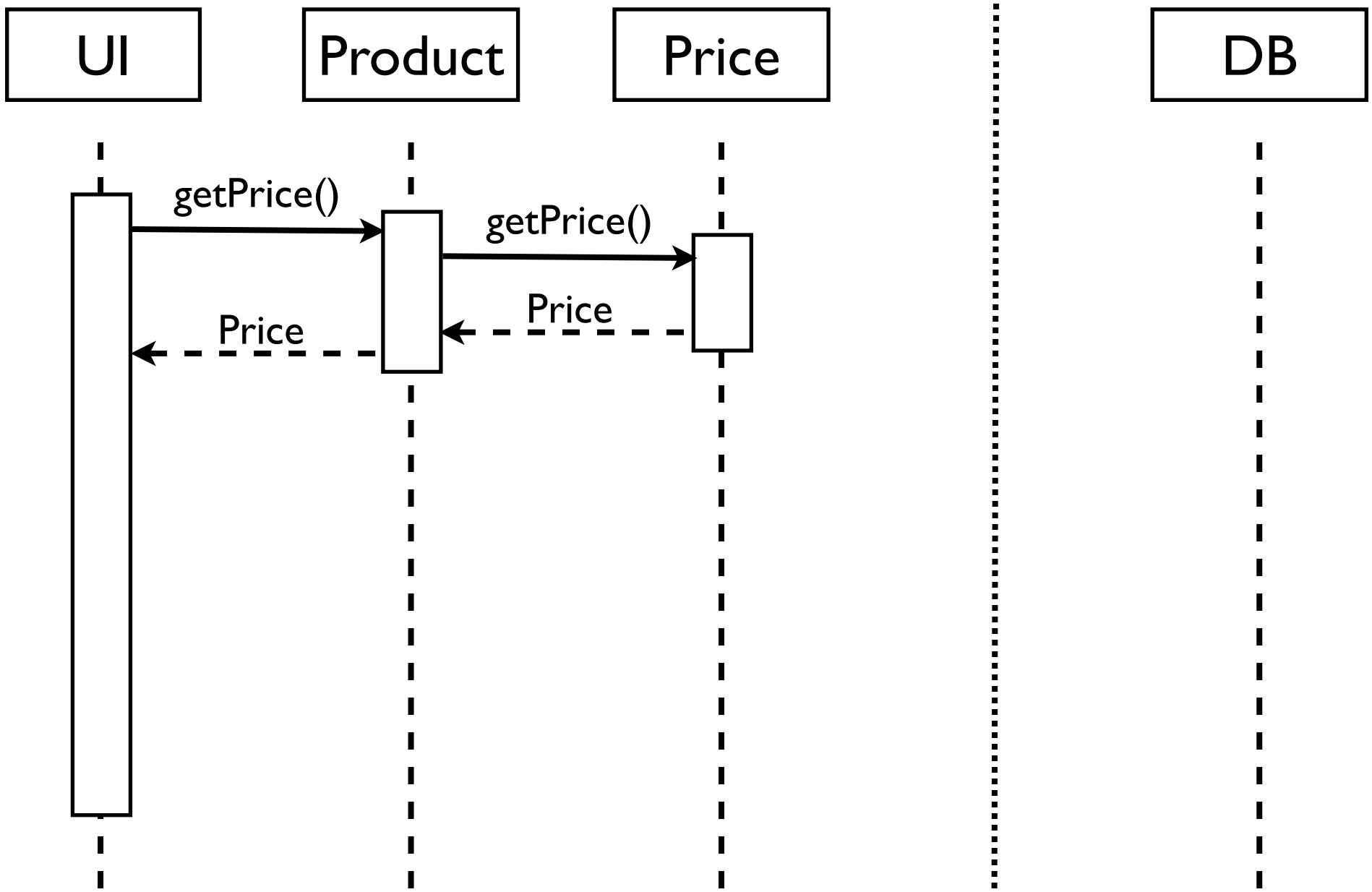
Abstractions for Connection Volatility

- At the middleware level
  - Scattered throughout application code!
- At the language level
  - Higher degree of (syntactic) obliviousness
  - Existing proposal: Futures

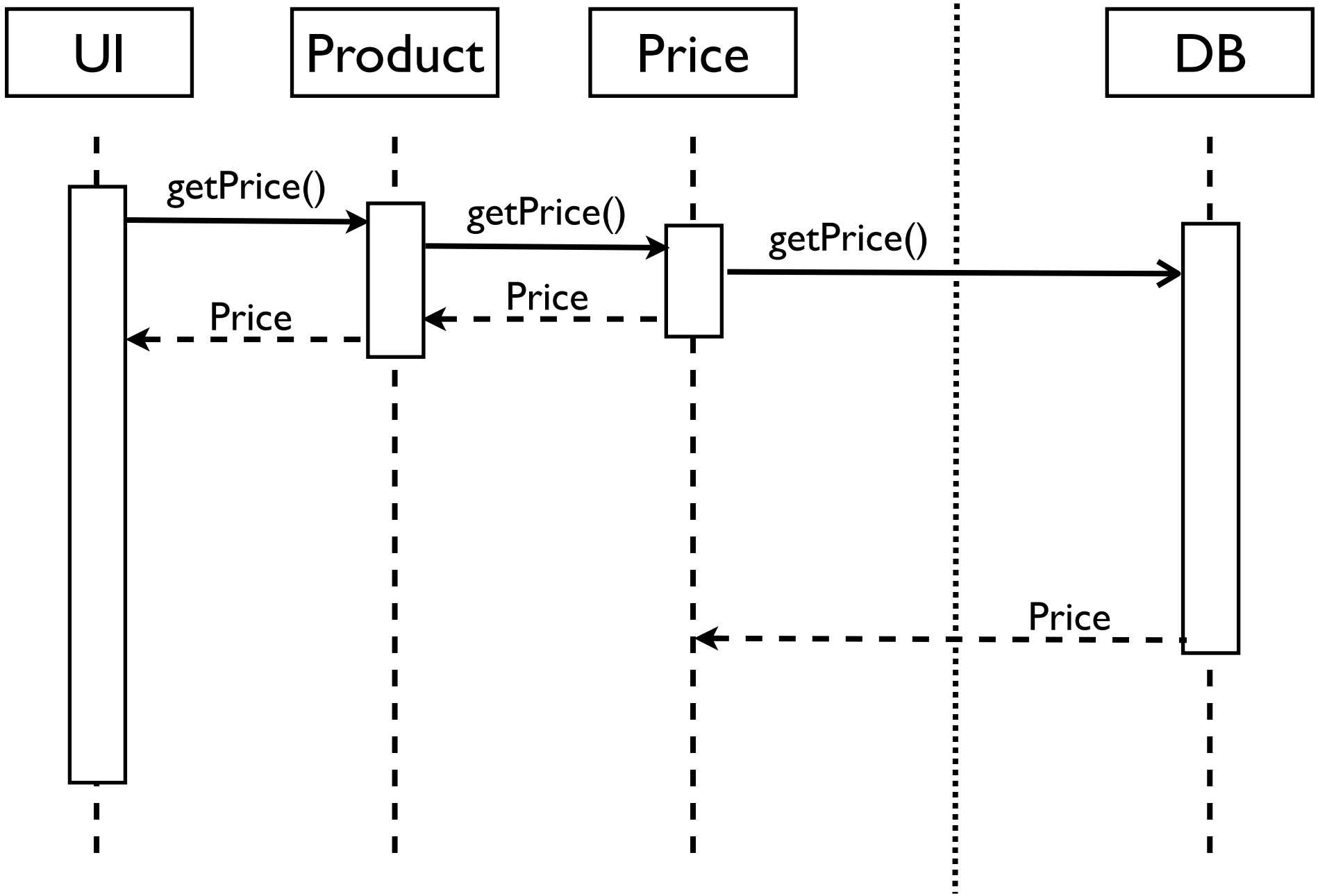
Using Futures



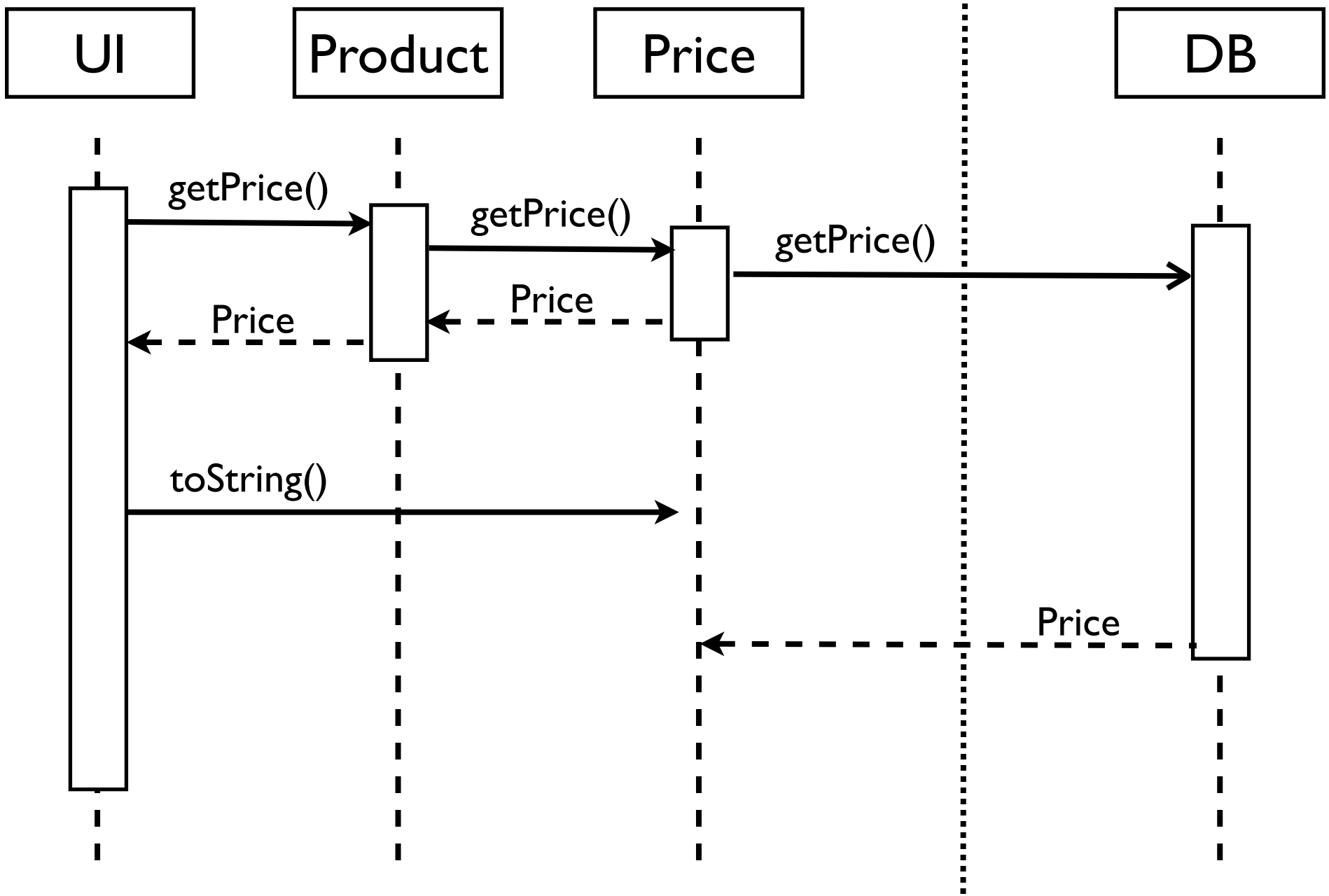
Using Futures



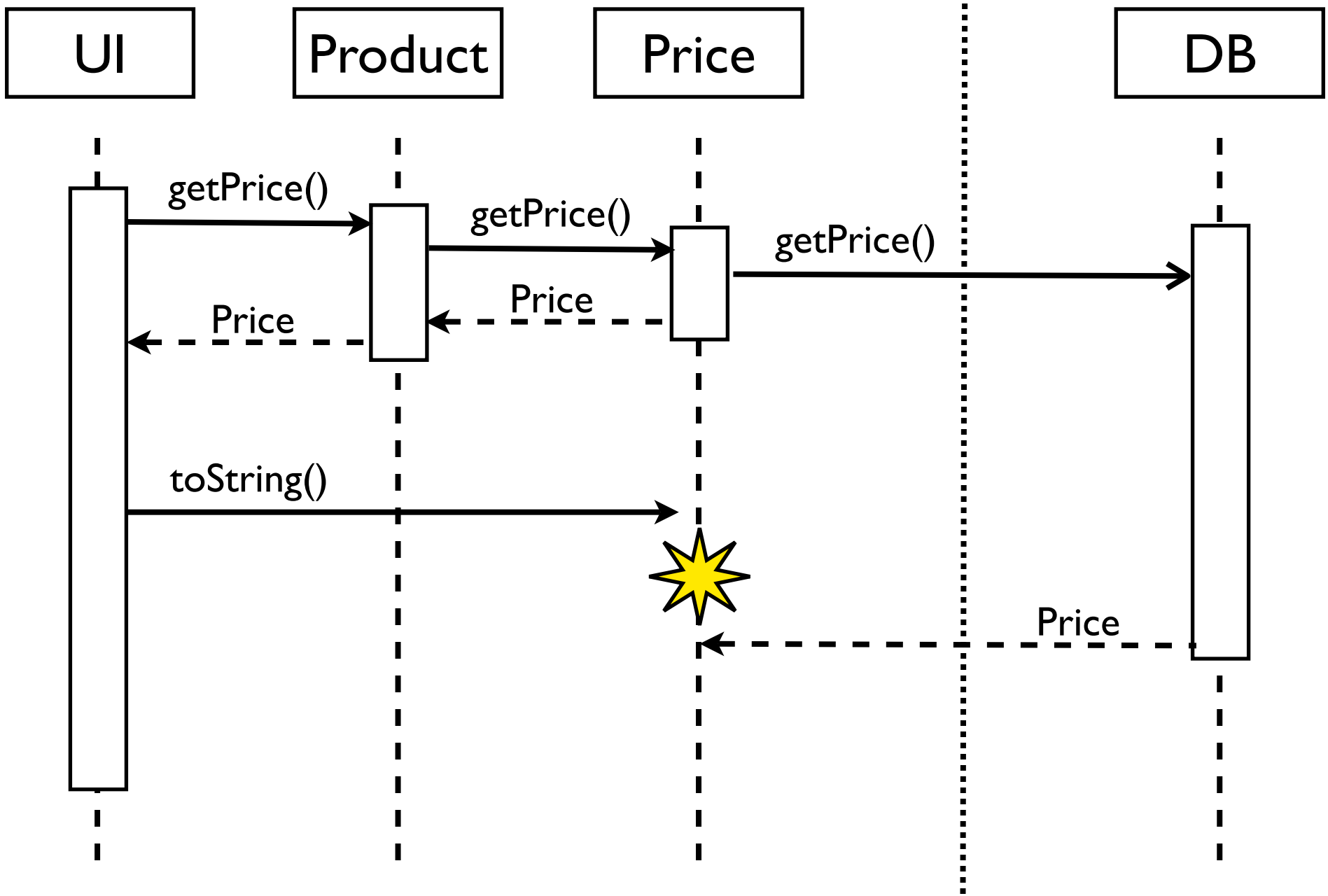
Using Futures



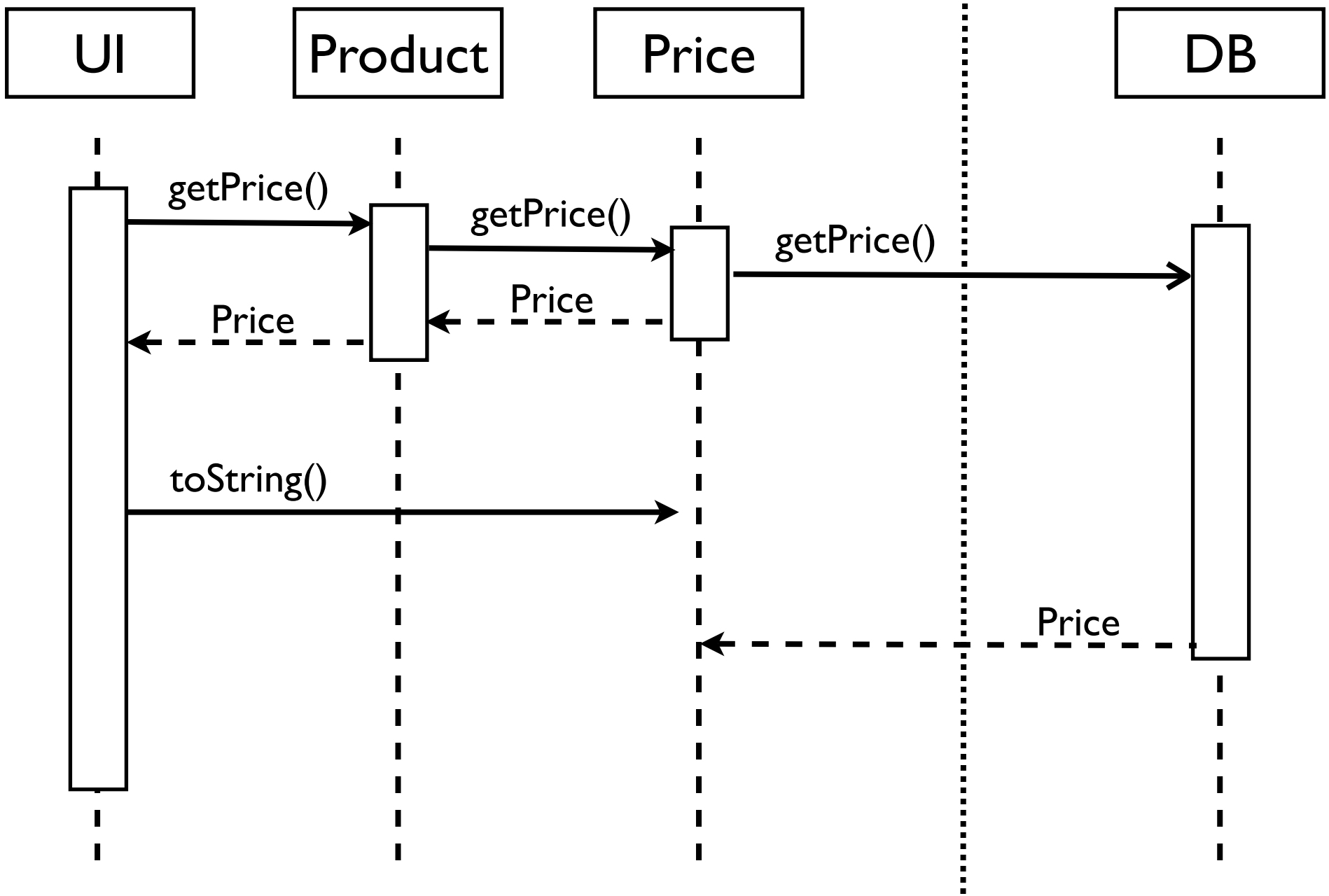
Using Futures



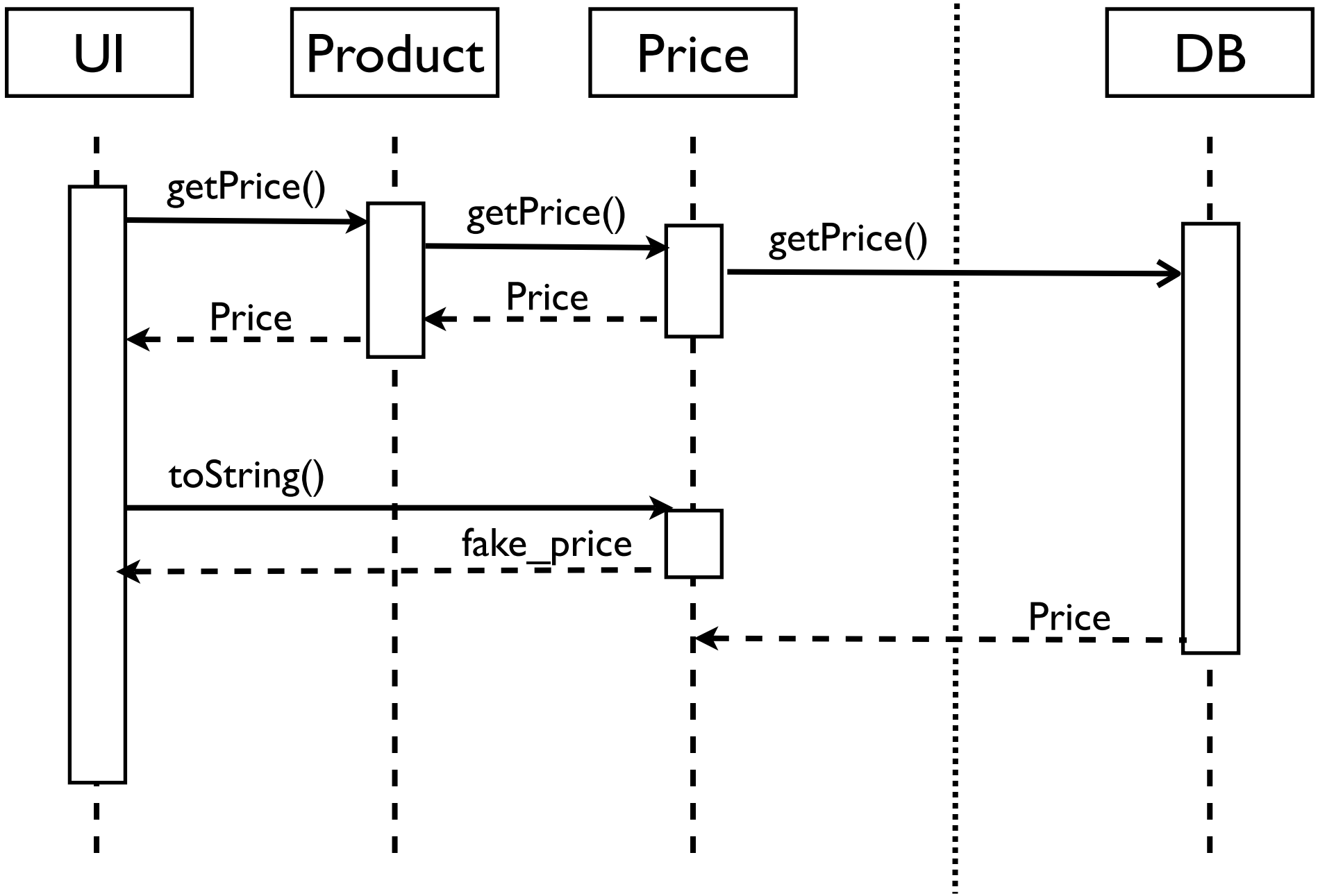
Using Futures



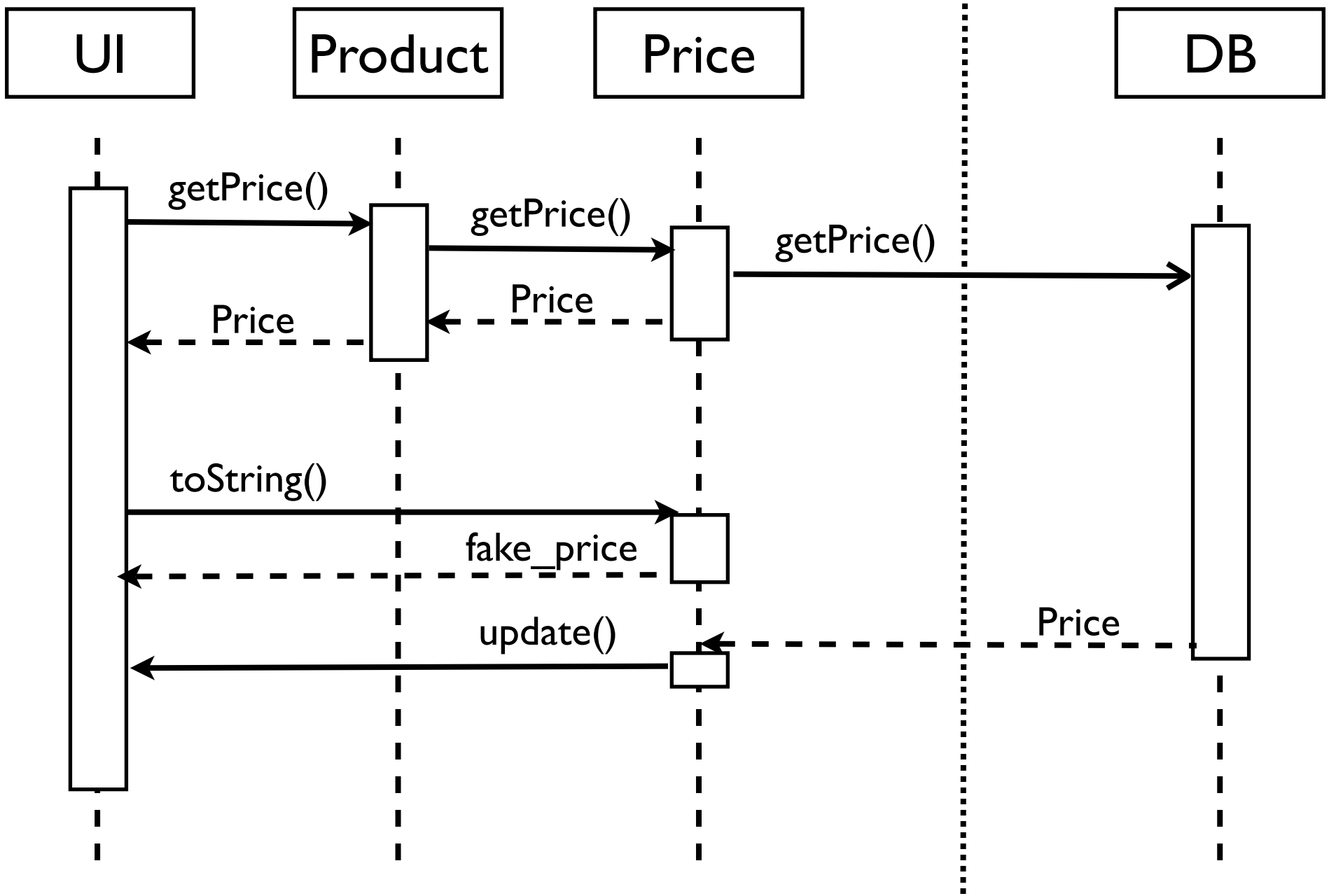
Using Futures



Using Futures



Using Futures



Tagged Futures

- Attach metadata to the future (Tag)
  - Offline value (“TBD”)
  - ...
- Update and Invalidation mechanism

Tagged Futures as Java 5 Annotations

- Future = placeholder
- Passive futures: no update
- @Connect: update mech
- @ObservedFuture = change subscription
- Online/Offline = change notification

Passive Futures:
@Future(expression)

Futures:
@Future(expression)
@Connect

Future Observers:
@ObservedFuture
@Online(type)
@Offline(type)

Tagged Futures as Java 5 Annotations

Price

- Future = placeholder
- Passive futures: no update
- @Connect: update mech

- @ObservedFuture = change subscription
- Online/Offline = change notification

Passive Futures:
@Future(expression)

Futures:
@Future(expression)
@Connect

Future Observers:
@ObservedFuture
@Online(type)
@Offline(type)

Tagged Futures as Java 5 Annotations

Price

- Future = placeholder
- Passive futures: no update
- @Connect: update mech

User Interface

- @ObservedFuture = change subscription
- Online/Offline = change notification

Passive Futures:
@Future(expression)

Futures:
@Future(expression)
@Connect

Future Observers:
@ObservedFuture
@Online(type)
@Offline(type)

- Extension of Spoon annotation processor
- Transform client-server app to ambient app
- Syntactic obliviousness

SG Abstracts

- Without SG:

```
public class ShopProductInfo implements Serializable ,
ConnectionObserver {
    protected Product target_product;
    protected String location;

    public ShopProductInfo(Product target_product ,String location) {
        this.target_product = target_product;
        this.location = location;
    }

    public String getDiscount() {
        try {
            if (ConnectionManager.IS_CONNECTED)
                try {
                    return "";
                } catch (UnmarshalException ex) {
                    throw new ConnectException("Server Disconnect" ,
ex);
                }
            else
                return "TBD";

        } catch (ConnectException ex) {
            ConnectionManager.setConnected(false);
            return this.getDiscount();
        }
    }

  ..................................................................

    public void graffitiConnectUpdate(Object arg) {
        if (arg.equals("Connected")) {
            become();
            observing_me.graffitiOnlineUpdate(this);
        } else
            observing_me.graffitiOfflineUpdate(this);

    }

    public void become() {
        ShopProductInfo realSPI =
spoon.graffiti.examples.shoppingApp.Shop.getShopProductInfo
(target_product);
        this.location = realSPI.location;
    }
  ..................................................................
```

- With SG:

```
public class ShopProductInfo implements Serializable{
        protected Product target_product;
        protected String location;

        public ShopProductInfo(Product target_product, String location) {
                this.target_product = target_product;
                this.location = location;
        }

        @Future("\"TBD\"")
        public String getDiscount() {
                return "";
        }

        @Future("\"TBD\"")
        public String getPrice() {
                return "?";
        }

        @Future("\"TBD\"")
        public String getPlace() {
                return location;
        }

        @Connect
        public void become(){
                ShopProductInfo realSPI = Shop.getShopProductInfo(target_product);
                this.location = realSPI.location;
        }

        public static ShopProductInfo createEmptySPI(Product product) {
                if(product instanceof SpecificProduct)
                        return ShopSpecificProductInfo.createEmptySPI(product);
                return new ShopProductInfo(product,null);
        }
}
```

SG Abstracts

- Without SG:

```
public class ShopProductInfo implements Serializable ,
ConnectionObserver {
    protected Product target_product;
    protected String location;

    public ShopProductInfo(Product target_product ,String location) {
        this.target_product = target_product;
        this.location = location;
    }

    public String getDiscount() {
        try {
            if (ConnectionManager.IS_CONNECTED)
                try {
                    return "";
                } catch (UnmarshalException ex) {
                    throw new ConnectException("Server Disconnect" ,
ex);
                }
            else
                return "TBD";

        } catch (ConnectException ex) {
            ConnectionManager.setConnected(false);
            return this.getDiscount();
        }
    }

  ..................................................................

    public void graffitiConnectUpdate(Object arg) {
        if (arg.equals("Connected")) {
            become();
            observing_me.graffitiOnlineUpdate(this);
        } else
            observing_me.graffitiOfflineUpdate(this);

    }

    public void become() {
        ShopProductInfo realSPI =
spoon.graffiti.examples.shoppingApp.Shop.getShopProductInfo
(target_product);
        this.location = realSPI.location;
    }
  ..................................................................
```

- With SG:

```
public class ShopProductInfo implements Serializable{
        protected Product target_product;
        protected String location;

        public ShopProductInfo(Product target_product, String location) {
                this.target_product = target_product;
                this.location = location;
        }

        @Future("\"TBD\"")
        public String getDiscount() {
                return "";
        }

        @Future("\"TBD\"")
        public String getPrice() {
                return "?";
        }

        @Future("\"TBD\"")
        public String getPlace() {
                return location;
        }

        @Connect
        public void become(){
                ShopProductInfo realSPI = Shop.getShopProductInfo(target_product);
                this.location = realSPI.location;
        }

        public static ShopProductInfo createEmptySPI(Product product) {
                if(product instanceof SpecificProduct)
                        return ShopSpecificProductInfo.createEmptySPI(product);
                return new ShopProductInfo(product,null);
        }
}
```

@Future("TBD")

@Connect

SG Abstracts

- Without SG:

```
public class ShoppingList implements TableModel ,
FutureOfflineObserver , FutureOfflineOnlineObserver {
    private Vector<ShopProductInfo>  products;
    private Vector<java.lang.Integer>  prod_amounts;
    private Vector<javax.swing.event.TableModelListener>  tml;

 ..............................................................

    public void addProduct(ShopProductInfo prod, Integer amount) {
        ShopProductInfo p2;
        {
            p2 = prod;
            spoon.graffiti.infrastructure.ConnectionManager.addObserver
(p2,this);
        }
        products.add(prod);
        prod_amounts.add(amount);
        changed(prod);
    }
  ..............................................................
    public void graffitiOfflineUpdate(Object arg) {
        try{this.changed
((spoon.graffiti.examples.shoppingApp.ShopProductInfo)arg);return;}
catch(ClassCastException ex){};
    }

        private void changedOn(ShopProductInfo arg){
                //System.out.println("Online GP"+arg);
                for(TableModelListener listener : tml)
                        listener.tableChanged(new TableModelEvent
(this));
        }

    public void graffitiOnlineUpdate(Object arg) {
        try{this.changedOn
((spoon.graffiti.examples.shoppingApp.ShopProductInfo)arg);return;}
catch(ClassCastException ex){};
    }

        private void changed(ShopProductInfo arg){
                for(TableModelListener listener : tml)
                        listener.tableChanged(new TableModelEvent
(this));

 ..............................................................
}
```

- With SG:

```
public class ShoppingList implements TableModel {

        private Vector<ShopProductInfo> products;
        private Vector<Integer> prod_amounts;
        private Vector<TableModelListener> tml;

        public ShoppingList(){
                products = new Vector<ShopProductInfo>();
                prod_amounts = new Vector<Integer>();
                tml = new Vector<TableModelListener>();
        }

        public void addProduct(ShopProductInfo prod, Integer amount){
                @ObservedFuture ShopProductInfo p2 = prod;
                products.add(prod);
                prod_amounts.add(amount);
                this.changed(prod);
        }
  ..............................................................

        @Online
        private void changedOn(ShopProductInfo arg){
                //System.out.println("Online GP"+arg);
                for(TableModelListener listener : tml)
                        listener.tableChanged(new TableModelEvent(this));
        }

        @Offline
        private void changed(ShopProductInfo arg){
                for(TableModelListener listener : tml)
                        listener.tableChanged(new TableModelEvent(this));
        }
  ..............................................................
}
```

SG Abstracts

- Without SG:

```
public class ShoppingList implements TableModel ,
FutureOfflineObserver , FutureOfflineOnlineObserver {
    private Vector<ShopProductInfo>  products;
    private Vector<java.lang.Integer>  prod_amounts;
    private Vector<javax.swing.event.TableModelListener>  tml;

 ..................................................................

    public void addProduct(ShopProductInfo prod, Integer amount) {
        ShopProductInfo p2;
        {
            p2 = prod;
            spoon.graffiti.infrastructure.ConnectionManager.addObserver
(p2,this);
        }
        products.add(prod);
        prod_amounts.add(amount);
        changed(prod);
    }
  ..................................................................
    public void graffitiOfflineUpdate(Object arg) {
        try{this.changed
((spoon.graffiti.examples.shoppingApp.ShopProductInfo)arg);return;}
catch(ClassCastException ex){};
    }

        private void changedOn(ShopProductInfo arg){
                //System.out.println("Online GP"+arg);
                for(TableModelListener listener : tml)
                        listener.tableChanged(new TableModelEvent
(this));
        }

    public void graffitiOnlineUpdate(Object arg) {
        try{this.changedOn
((spoon.graffiti.examples.shoppingApp.ShopProductInfo)arg);return;}
catch(ClassCastException ex){};
    }

        private void changed(ShopProductInfo arg){
                for(TableModelListener listener : tml)
                        listener.tableChanged(new TableModelEvent
(this));

 ..................................................................
}
```

- With SG:

```
public class ShoppingList implements TableModel {

        private Vector<ShopProductInfo> products;
        private Vector<Integer> prod_amounts;
        private Vector<TableModelListener> tml;

        public ShoppingList(){
                products = new Vector<ShopProductInfo>();
                prod_amounts = new Vector<Integer>();
                tml = new Vector<TableModelListener>();
        }

        public void addProduct(ShopProductInfo prod, Integer amount){
                @ObservedFuture ShopProductInfo p2 = prod;
                products.add(prod);
                prod_amounts.add(amount);
                this.changed(prod);
        }
  ..................................................................
        @Online
        private void changedOn(ShopProductInfo arg){
                //System.out.println("Online GP"+arg);
                for(TableModelListener listener : tml)
                        listener.tableChanged(new TableModelEvent(this));
        }

        @Offline
        private void changed(ShopProductInfo arg){
                for(TableModelListener listener : tml)
                        listener.tableChanged(new TableModelEvent(this));
        }
  ..................................................................
}
```

@ObservedFuture

@Online

@Offline

SG Untangles!

- Tagged Futures = Abstraction mechanism
- We have untangling
- Still scattered
  - Annotations
  - Code
- AOP : Scattering is bad

Unscattering SG Code

- `@Future( ... )`
- `@Connect`
- `@ObservedFuture`
- `@Online`
- `@Offline`

Unscattering SG Code

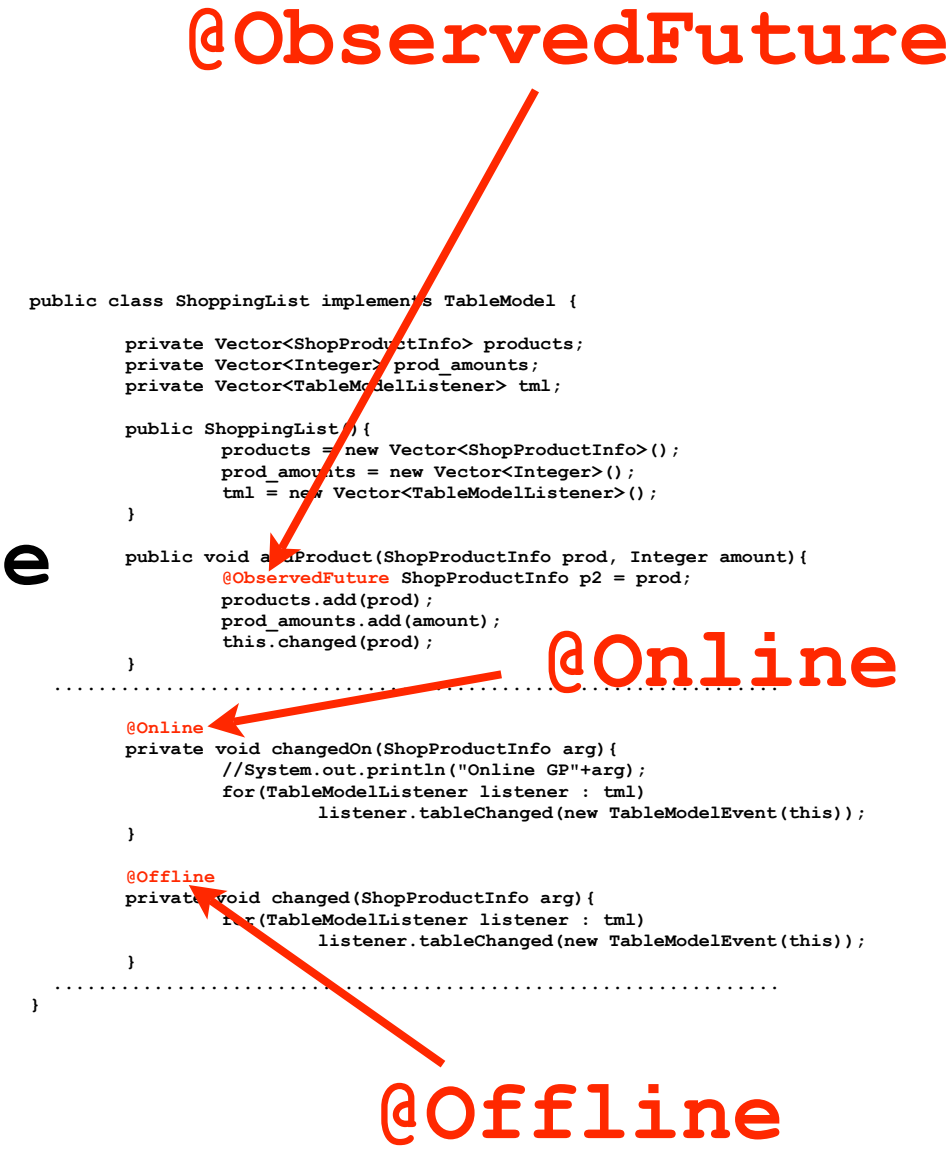
- `@Future( ... )`
- `@Connect`
- `@ObservedFuture`
- `@Online`
- `@Offline`

Unscattering SG Code

- `@Future( ... )`
- `@Connect`
- `@ObservedFuture`
- `@Online`
- `@Offline`

Unscattering SG Code

- `@Future( ... )`
- `@Connect`
- `@ObservedFuture`
- `@Online`
- `@Offline`

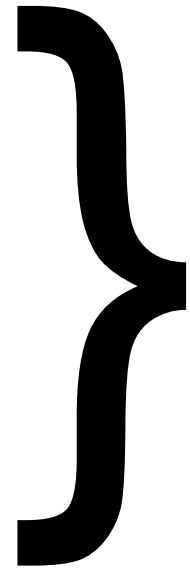


Unscattering SG Code

- `@Future( ... )`
- `@Connect`
- `@ObservedFuture`
- `@Online`
- `@Offline`

Unscattering SG Code

- `@Future( ... )`
- `@Connect`
- `@ObservedFuture`
- `@Online`
- `@Offline`



- No Quant
- Non-trivial!

A middle road ...

- Do you always want to go to the extreme?
- No:
  - Unscatter `@Future( ... )`
  - Leave the rest in peace
- How to eval trade-offs?
- But what is this? Is it AOP?