

Conception Architectures Distribuées

- | -

Conception de l'architecture

Exemples dans le cours

- Software Engineering, 9th ed. - Ian Sommerville
- Design Patterns in Java - Steven John Metsker,
William C. Wake

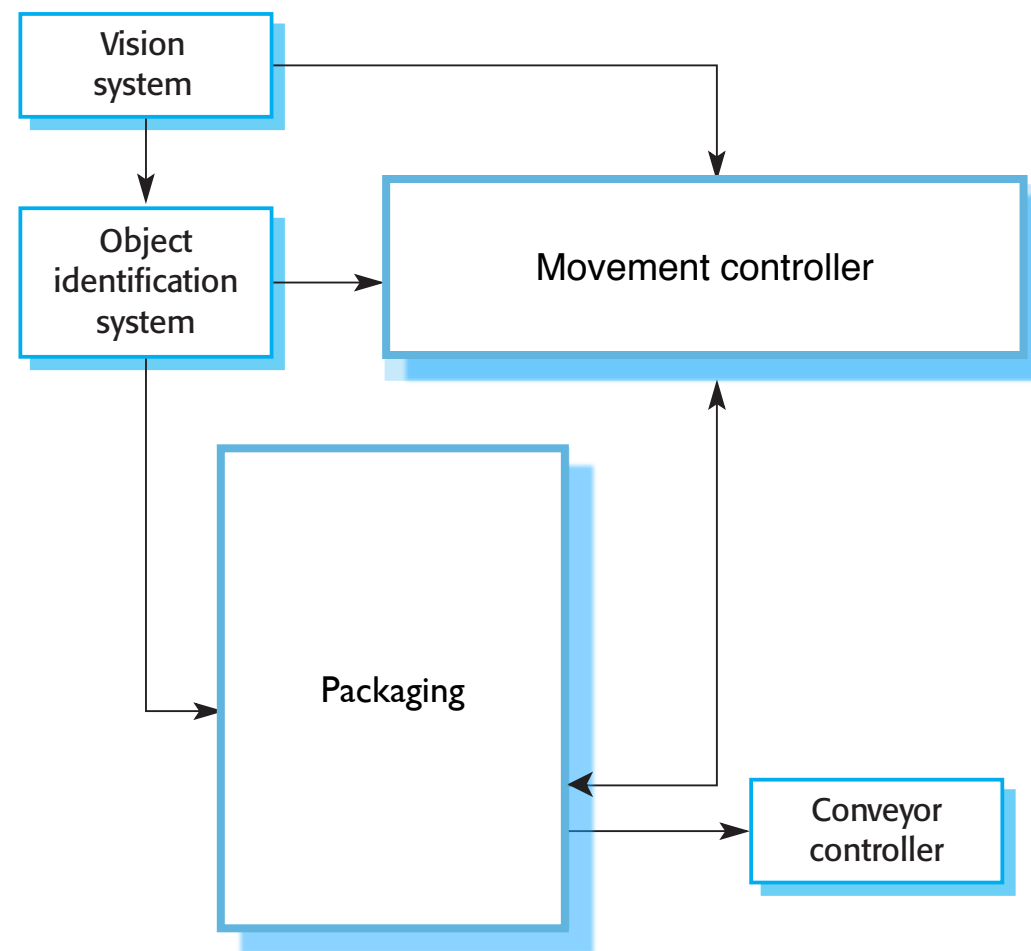
Conception de l'architecture

- le processus permettant d'identifier **les sous-systèmes** d'un système et la plateforme de **contrôle et communication** (des sous-systèmes)
 - ➔ ***architecture logicielle***

Conception de l'architecture

- représente le lien entre la spécification et la conception
- souvent réalisée en parallèle avec des activités de spécification
- identification de principaux composants et de leurs interactions

Architecture d'un système de contrôle



Granularité

- Grande échelle
 - ▶ architecture des systèmes d'entreprise complexes
- Petite échelle
 - ▶ architecture des systèmes/programmes indépendants

Utilisation

- Communication entre les différents acteurs
- Analyse système
- Application à d'autres systèmes
- Documentation de systèmes existants

Décisions

- Architecture générique applicable ?
- Comment est distribué le système ?
- Quelle approche pour structurer le système ?
- Comment le décomposer en modules ?
- Quelle stratégie de contrôle utiliser ?

Décisions

- Architecture générique applicable ?
- Comment est distribué le système ?
- Quels **patterns architecturaux** utiliser ?

Décisions

- Architecture générique applicable ?
- Comment est distribué le système ?
- Quels patterns architecturaux utiliser ?
- Comment évaluer l'architecture ?

Critères de choix

- **Performance**

- ▶ localiser les opérations critiques et minimiser les communications; utiliser une composition gros-grain

- **Sécurité**

- ▶ utiliser une architecture à couches avec les entités critiques dans les couches basses.

Critères de choix

- **Sûreté**
 - ▶ localiser les fonctionnalités critiques dans un nombre restreint de sous-systèmes
- **Disponibilité**
 - ▶ inclure des composants redondants et des mécanismes de tolérance aux fautes
- **Maintenabilité**
 - ▶ utiliser une composition fine

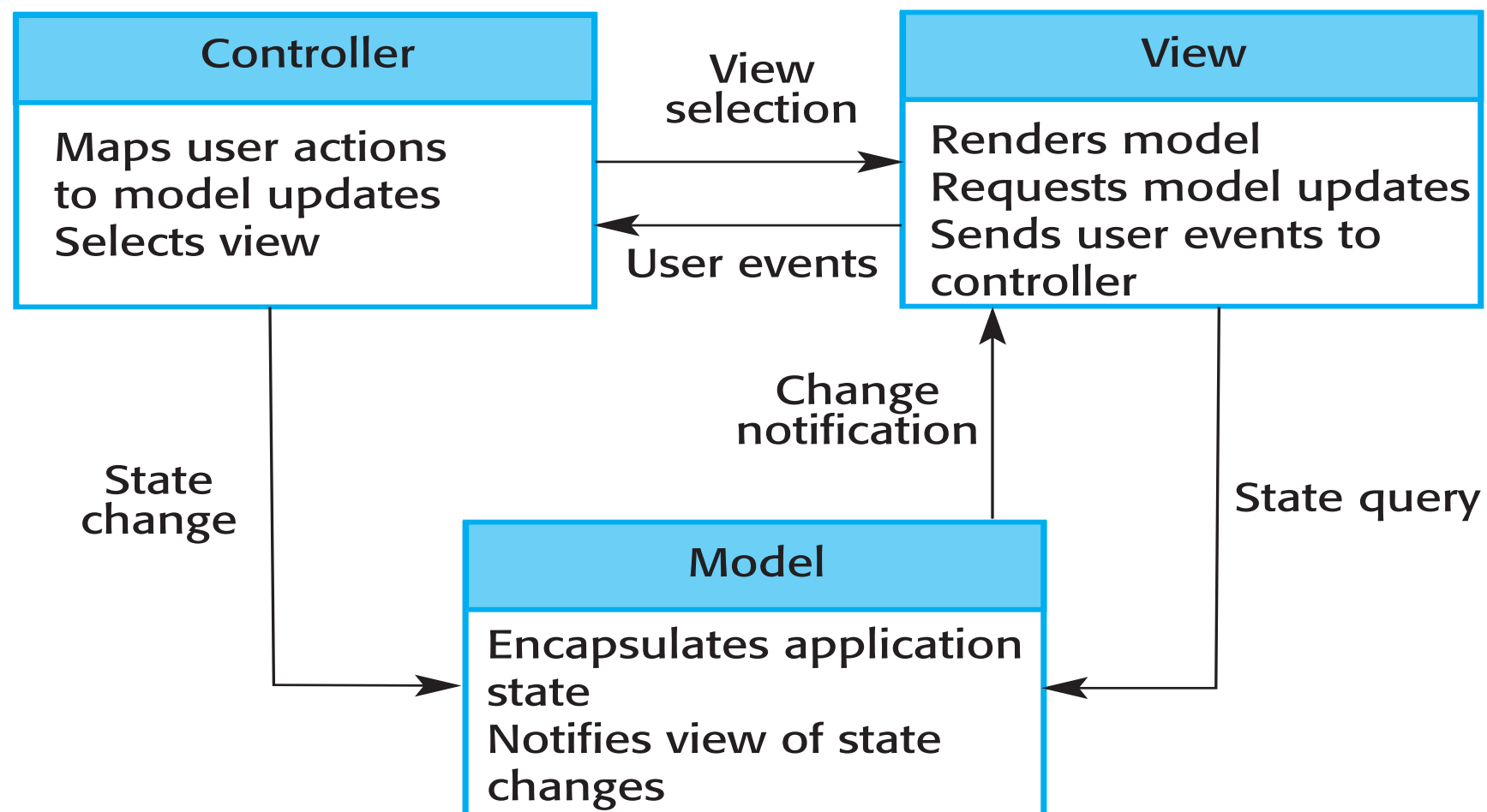
Patterns architecturaux

- un moyen de **représenter**, **partager**, **réutiliser** la connaissance/expérience
- description d'une bonne pratique appliquée dans différents environnements
- indications/contre-indications

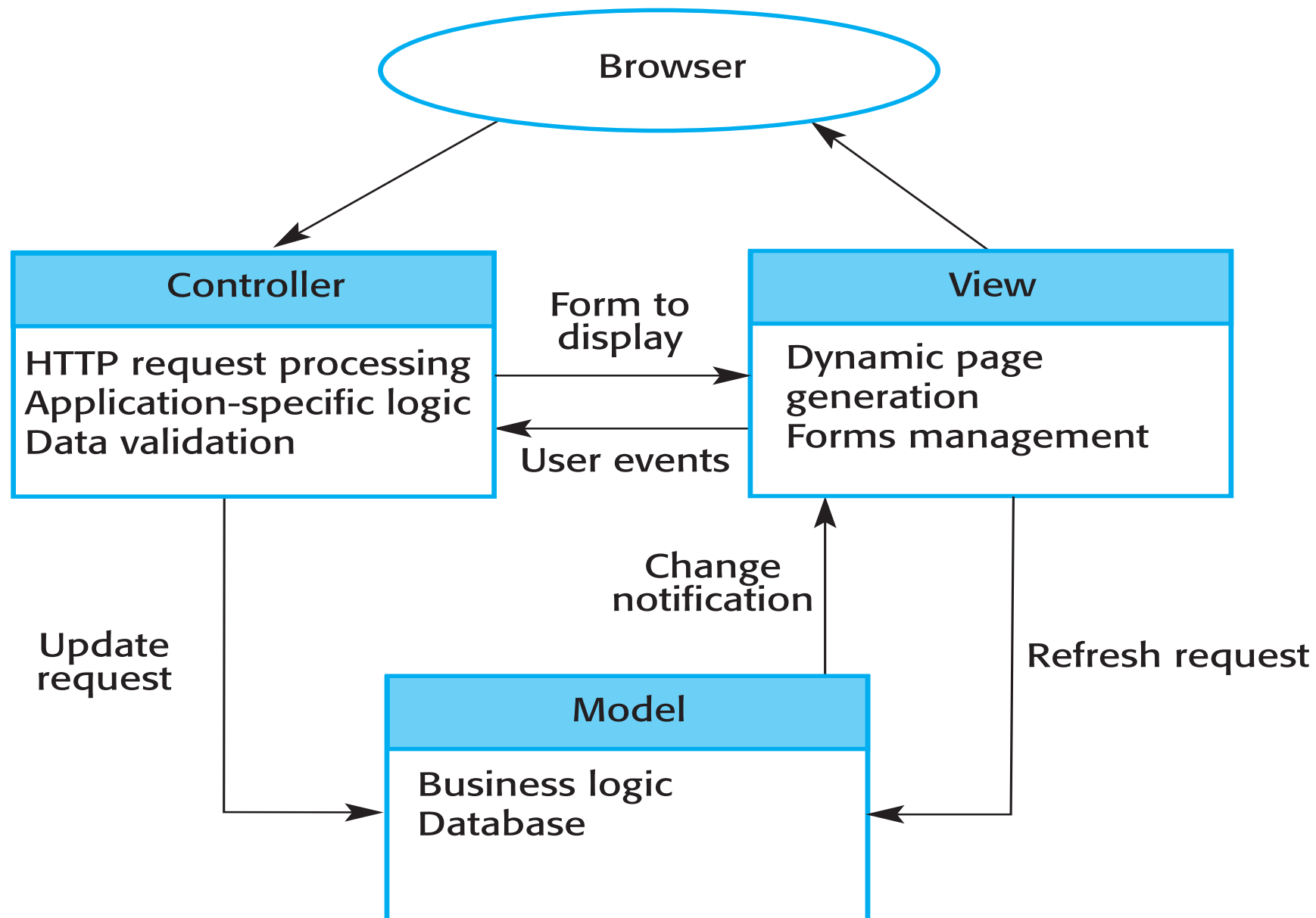
Model-View-Controller (MVC)

- utilisé pour séparer les aspects liés à la **présentation** et à **l'interaction** des **traitements** des données
- **quand ?**
 - ▶ plusieurs manières (connues ou non) de présenter ou d'interagir avec les données

Organisation MVC



Exemple MVC



MVC

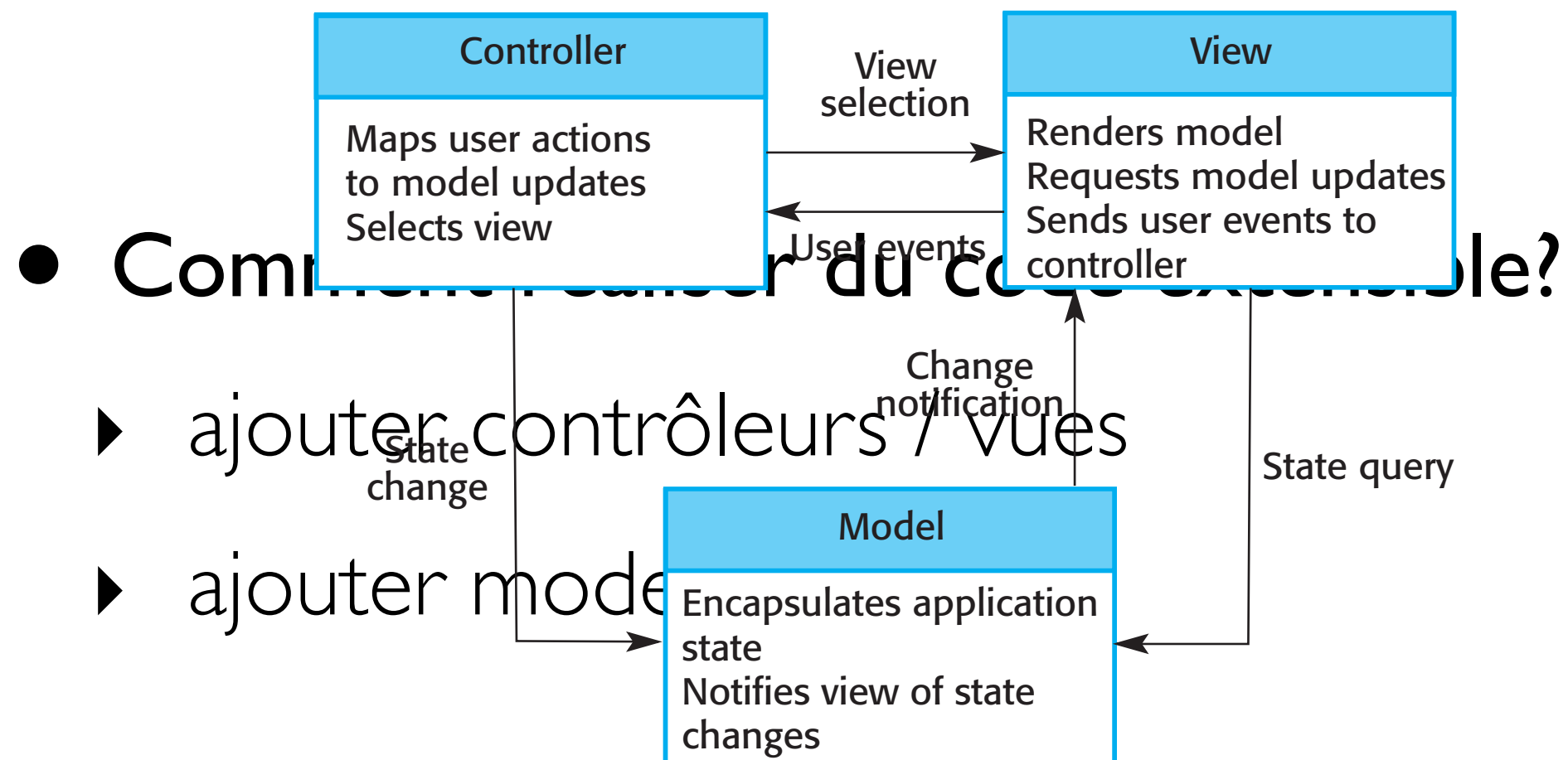
- **Avantages**

- ▶ les données peuvent évoluer indépendamment de leur présentation
- ▶ plusieurs présentations différentes pour les mêmes données

- **Inconvénients**

- ▶ ajouter du code (complexe) même pour des données/interactions simples

Implementation



```
public class NuclearPlantModel{  
    private double temperature;
```

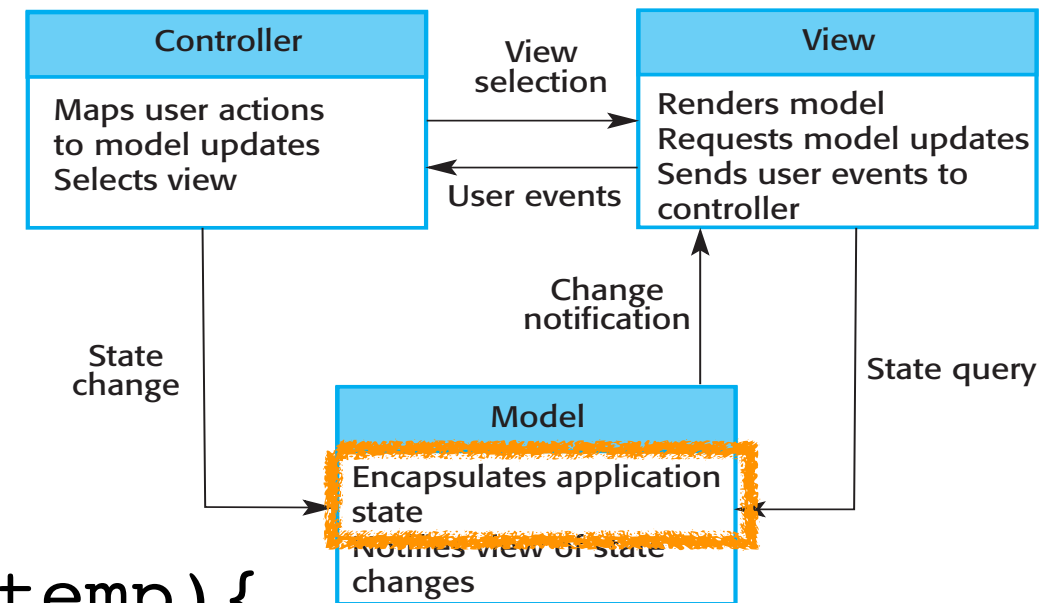
```
    public NuclearPlantModel(double temp){  
        this.temperature = temperature;  
    }
```

```
    public double getCelsius() {  
        return temperature;  
    }
```

```
    public double getFahrenheit() {  
        return temperature*9.0/5.0 + 32.0;  
    }
```

```
    public void setCelsius(double temperature) {  
        this.temperature = temperature;  
    }
```

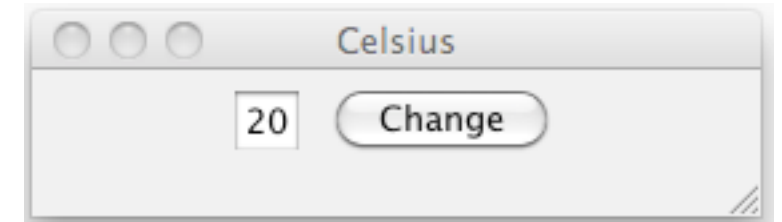
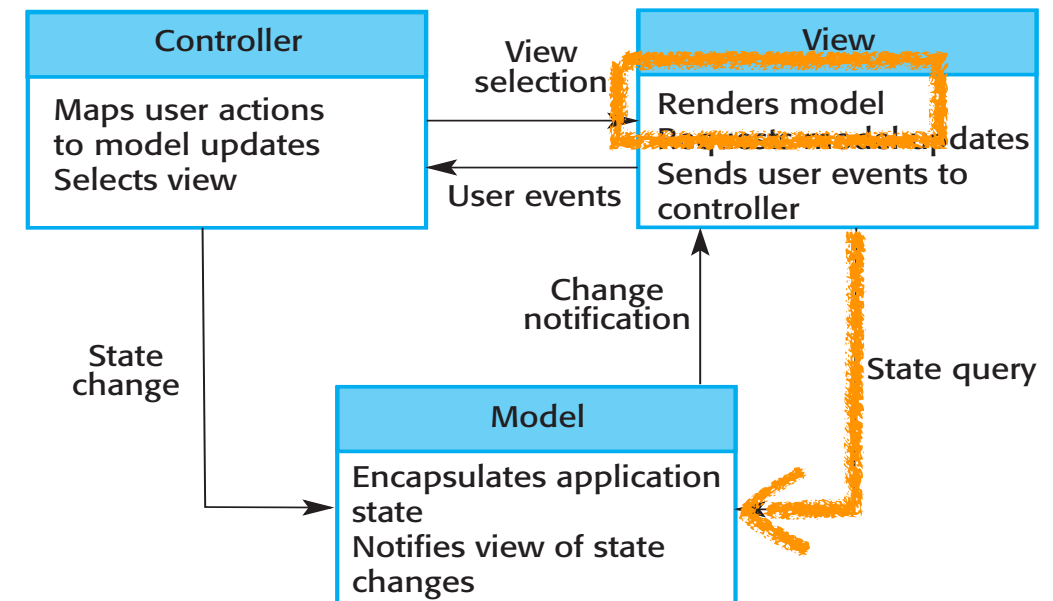
```
}
```



```

public class CelsiusView{
    private NuclearPlantModel model;
    ...
    private void render() {
        frame = new JFrame();
        contentPane = new JPanel();
        ...
        field.setValue(model.getCelsius());
        ...
    }
    ...
}

```



```
public class CelsiusView{
```

```
    private TemperatureController controller;
```

```
    private NuclearPlantModel model;
```

```
    private void render() {
```

```
        ...
```

```
        button = new JButton("Change");
```

```
        button.addActionListener(new ChangeListener() );
```

```
        ...
```

```
    }
```

```
    ...
```

```
    class ChangeListener implements ActionListener {
```

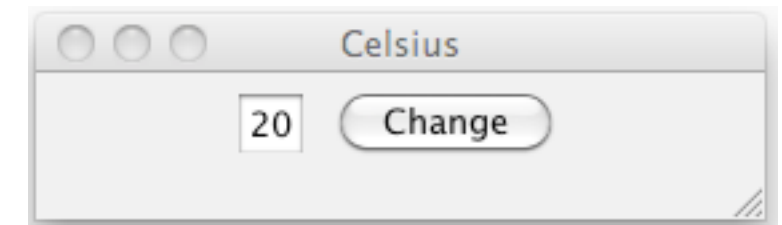
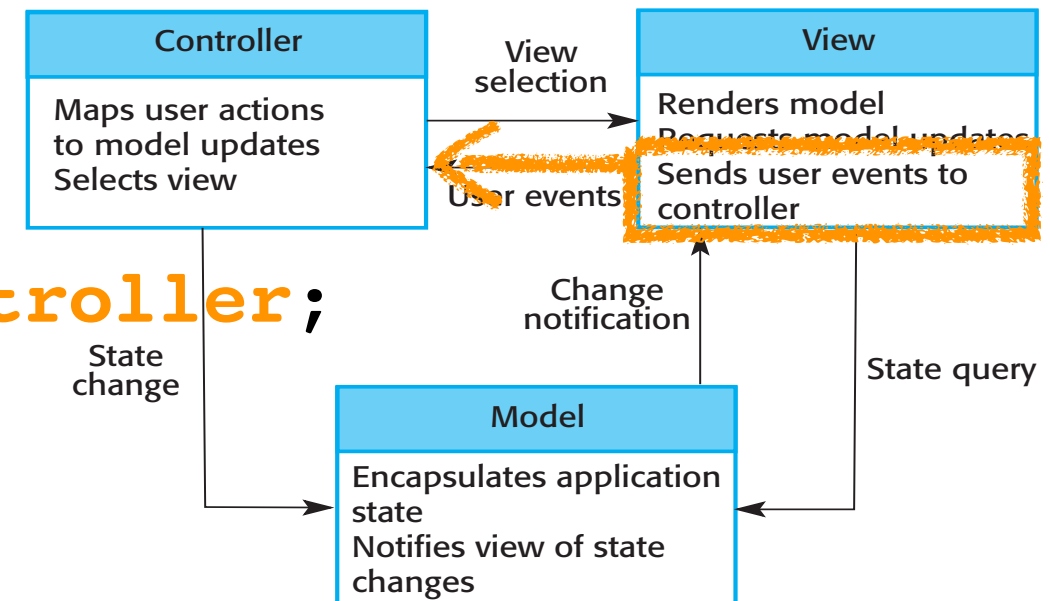
```
        public void actionPerformed(ActionEvent e){
```

```
            controller.notifyCelsiusChanged(field.getValue());
```

```
        }
```

```
    }
```

```
}
```



```

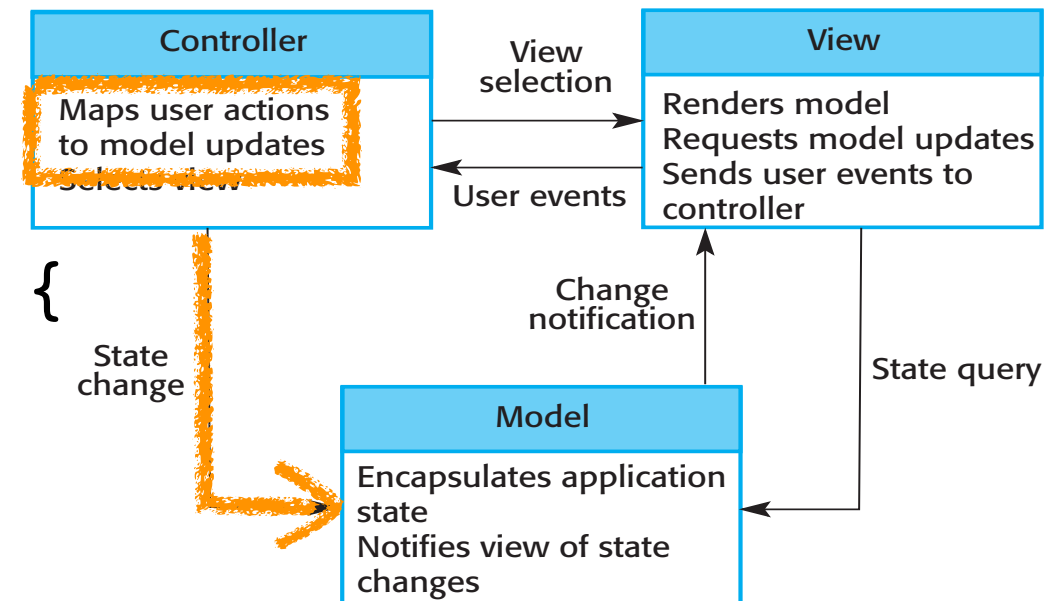
public class TemperatureController {
    private NuclearPlant model;
    ...

    public TemperatureController(NuclearPlant model){
        this.model = model;
        ...
    }

    //called from View
    public void notifyCelsiusChanged(double temp){
        model.setCelsius(temp);
    }

    public void notifyFahrenheitChanged(double temp){
        model.setFahrenheit(temp);
    }
}

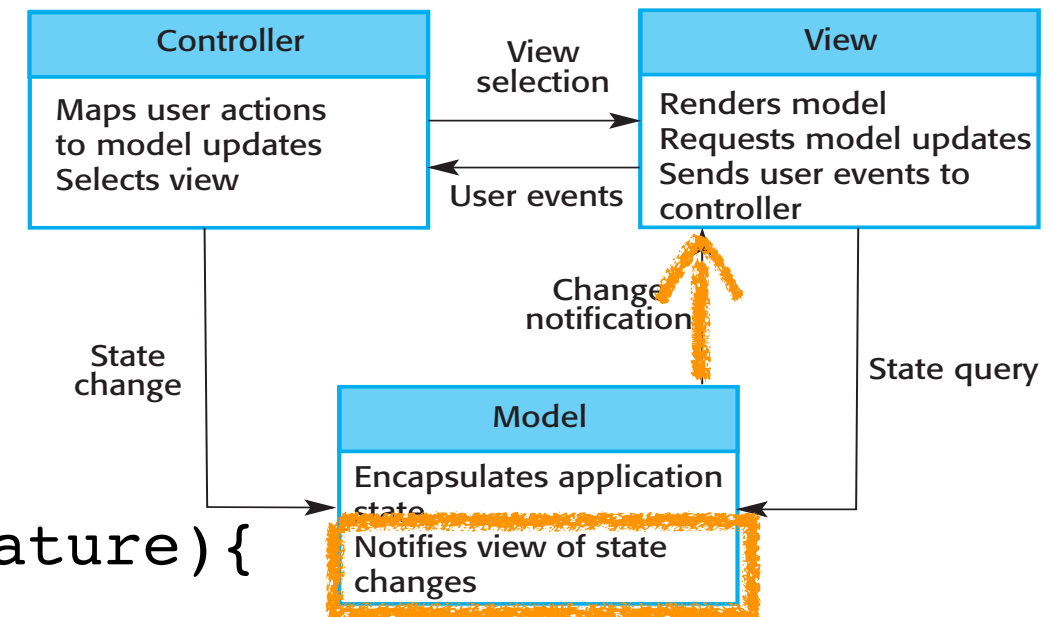
```



```
public class NuclearPlantModel{
    private List<View> listViews;
    private double temperature;
```

```
    public NuclearPlantModel(double temperature){
        listViews = new ArrayList<View>();
        this.temperature = temperature;
    }
    public void setCelsius(double temperature) {
        this.temperature = temperature;
        notifyViews();
    }
}
```

```
    public void addView(View view) {
        listViews.add(view);
    }
    public void notifyViews() {
        for (View view : listViews) {
            view.update();
        }
    }
}
```



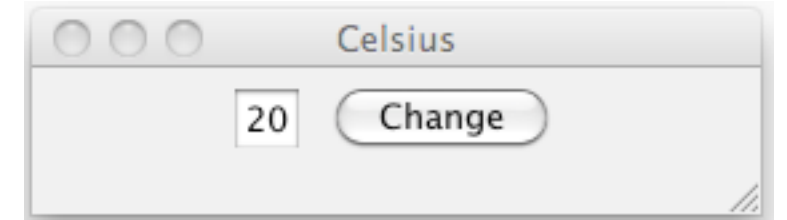
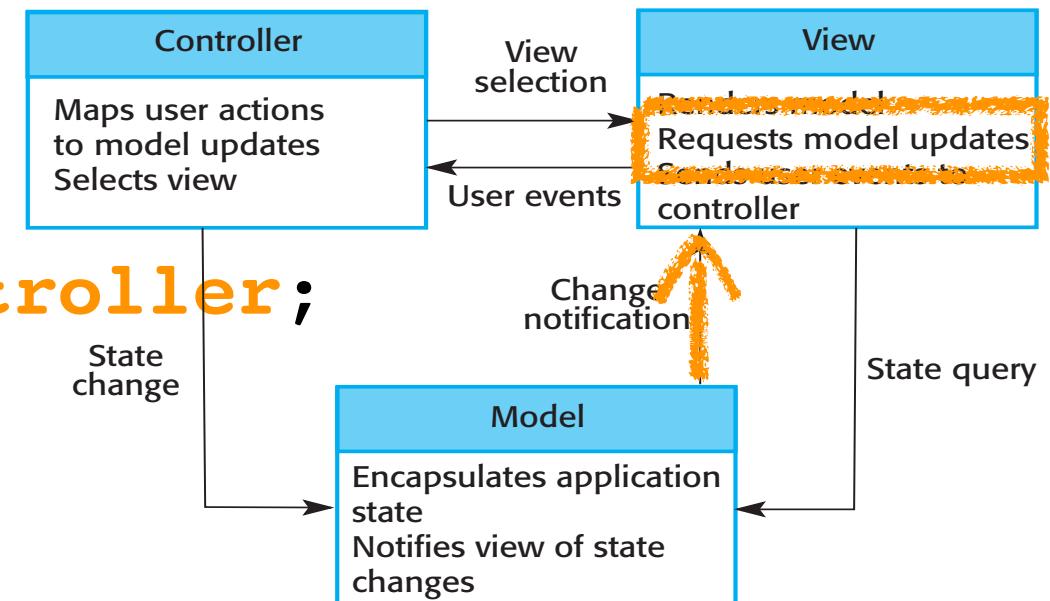
```
public interface View {
    void update();
}
```

```

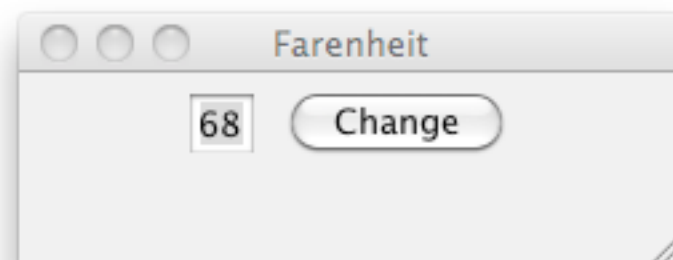
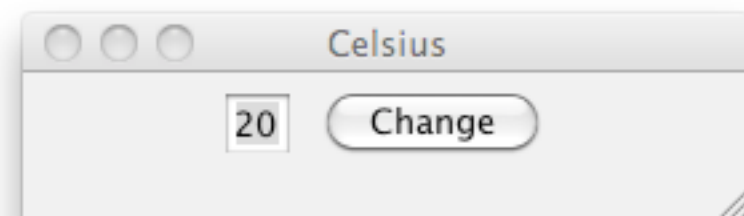
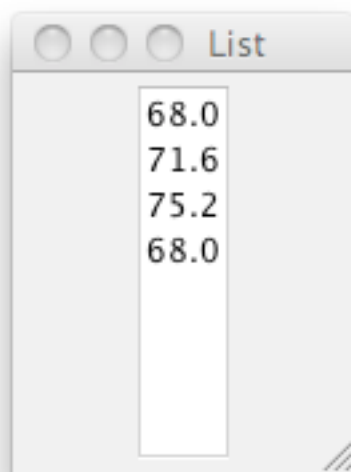
public class CelsiusView{
    private TemperatureController controller;
    private NuclearPlantModel model;

    private void render() { ... }
    ...
    // Called from the Model
    public void update() {
        field.setText(model.getCelsius());
    }
    public CelsiusView(TemperatureController controller,
                       NuclearPlantModel model) {
        this.controller = controller;
        this.model = model;
        model.addView(this);
        controller.addView(this);
        render();
    }
}

```



```
public class NuclearPlant {  
    public static void main(String[] args) {  
        NuclearPlantModel model = new NuclearPlantModel(20);  
        TemperatureController controller =  
            new TemperatureController(model);  
        new CelsiusView(controller, model);  
        new FarenheitView(controller, model);  
        new ListView(controller, model);  
        controller.start();  
    }  
}
```



Architecture en couches

- utilisée pour modéliser l'interfaçage entre sous-systèmes
- organise le système en plusieurs couches, chacune fournissant différents services au niveau supérieur
- supporte le développement incremental des sous-systèmes dans les différentes couches

Architecture en couches

- Quand ?
 - ▶ construction nouvelles fonctionnalités à partir de systèmes existants
 - ▶ développement réparti entre plusieurs équipes
 - ▶ demandes pour une sécurité multi-niveaux
 - ▶ ...

Layered architecture

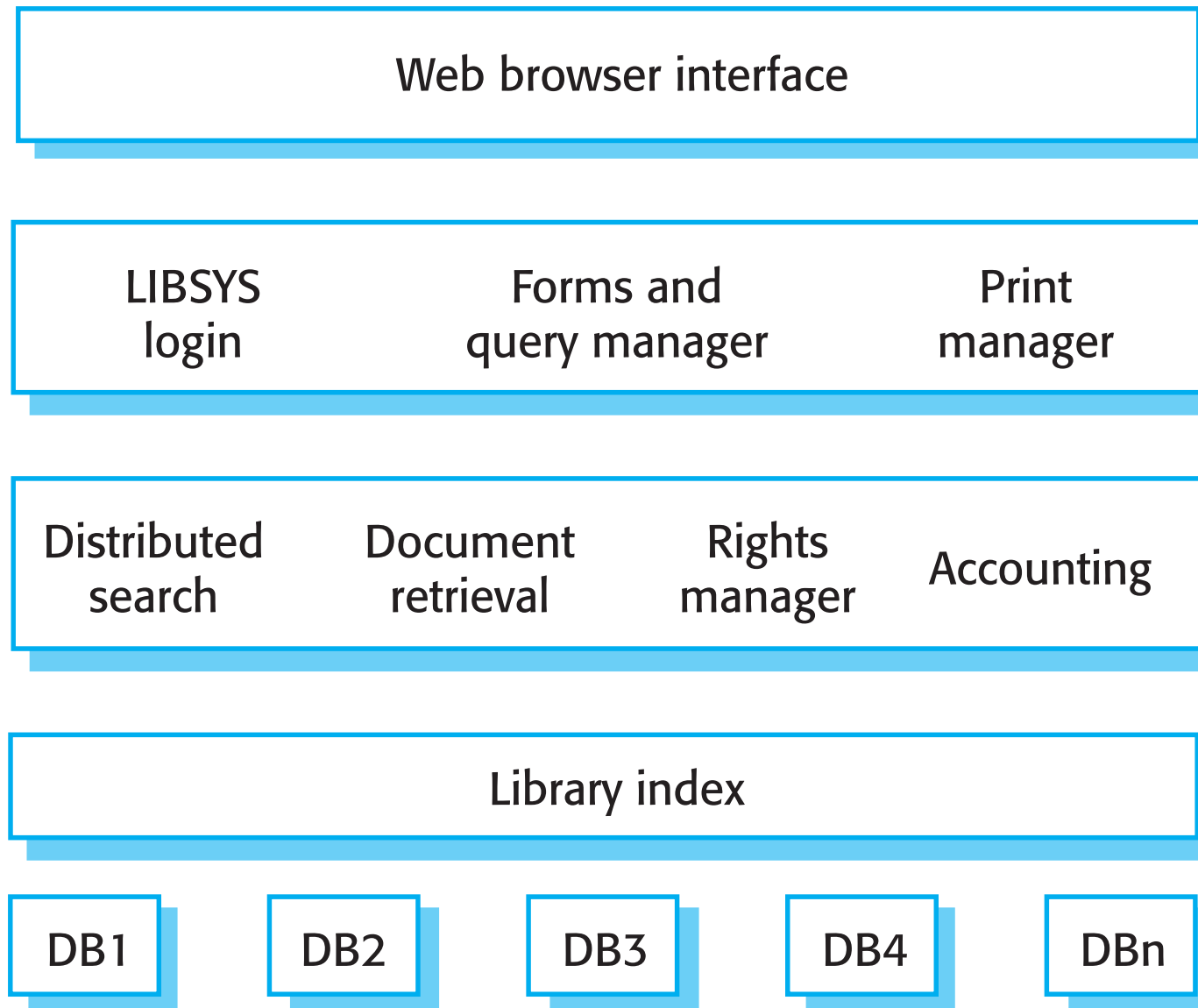
User interface

User interface management
Authentication and authorization

Core business logic/application functionality
System utilities

System support (OS, database etc.)

Exemple



Architecture en couches

- **Avantages**

- ▶ permet le remplacement d'un niveau si les interfaces sont respectées
- ▶ des fonctionnalités redondantes peuvent être présentes dans les couches successives

- **Inconvénients**

- ▶ séparation pas toujours naturelle
- ▶ accès à des niveaux inférieurs parfois nécessaire
- ▶ performance

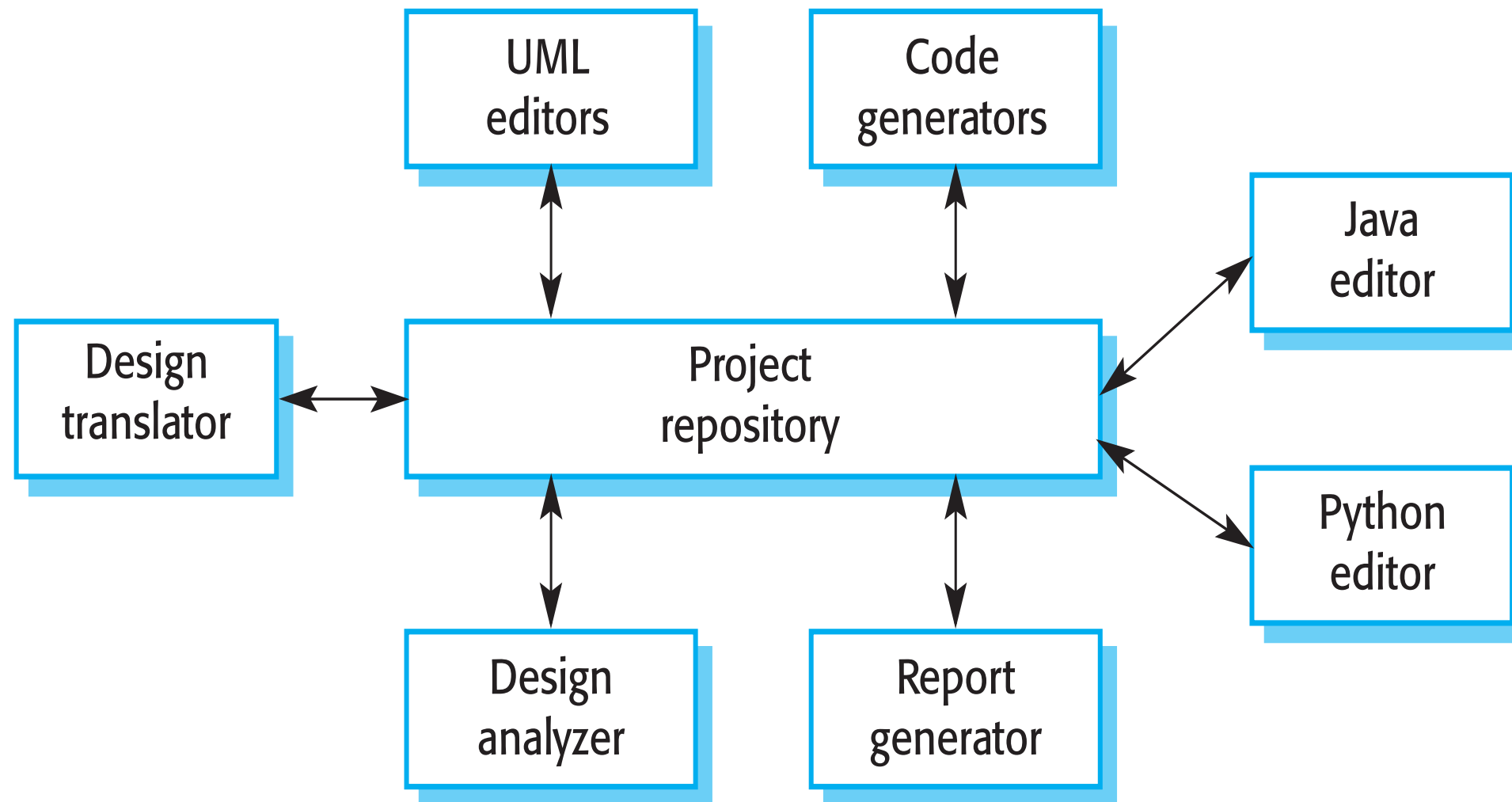
Architecture « Repository »

- modéliser un dépôt de données central partagé par plusieurs sous-systèmes

Architecture repository

- Quand ?
 - ▶ des volumes importants de données générés doivent être stockés pour de longues périodes
 - ▶ systèmes dirigés par les données : la modification du repository peut déclencher des événements

Exemple - IDE



Architecture repository

- **Avantages**

- ▶ composants indépendants
- ▶ modifications répercutées sur tous les autres
- ▶ donnée gérées d'une manière consistante

- **Inconvénients**

- ▶ le repository est un point sensible
- ▶ distribuer le repository peut s'avérer difficile
- ▶ performance : communications via le repository

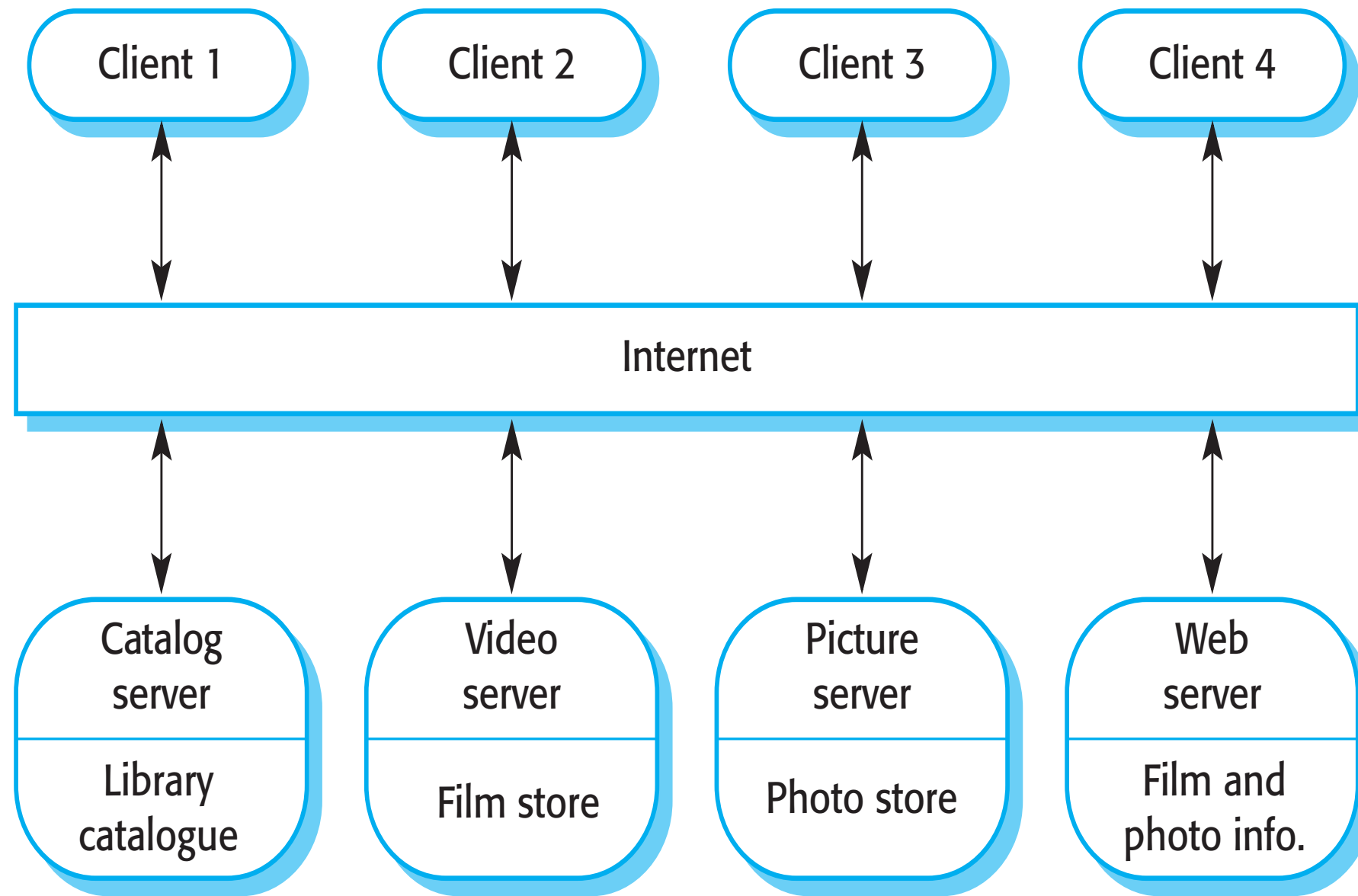
Architecture Client-serveur

- utilisé pour représenter un ensemble de services distribués sur un ensemble de serveurs
- les services sont appelés par (plusieurs instances d') un client (via un réseau)

Client-serveur

- Quand ?
 - ▶ des données partagées doivent être accédées depuis plusieurs machines
 - ▶ quand la charge d'un système est importante et les services doivent être dupliqués

Example



Client-serveur

- **Avantages**

- ▶ séparation et indépendance
- ▶ les serveurs peuvent être distribués

- **Inconvénients**

- ▶ les serveurs sont des points sensibles
- ▶ performance dépendante du réseau

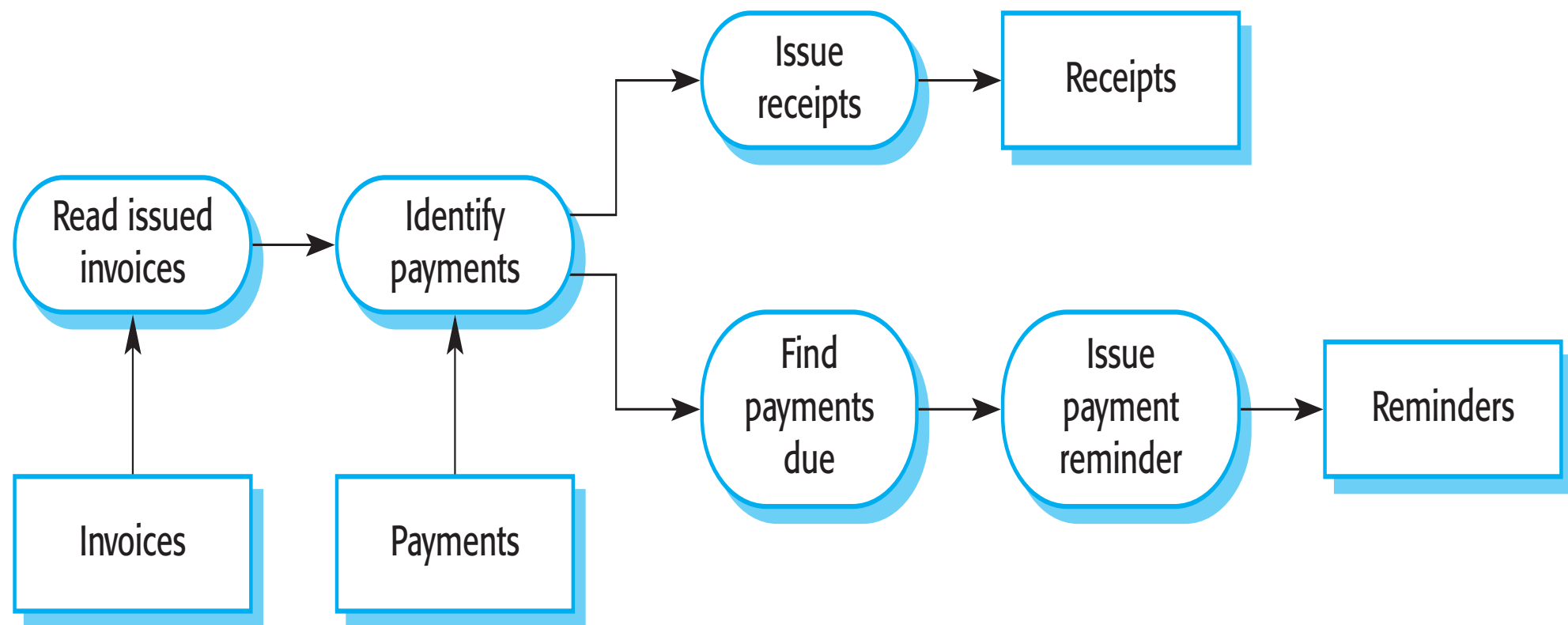
Pipe/Filter

- enchaînement de transformations fonctionnelles qui utilisent des entrées pour produire des sorties
- les transformations sont exécutées d'une manière séquentielle ou parallèle

Pipe/Filter

- Quand ?
 - ▶ applications de traitement (transactionnel) de données (par lots) en plusieurs étapes

Example



Pipe/Filter

- **Avantages**

- ▶ correspond à la structure de beaucoup de processus métier
- ▶ permet la réutilisation/ajout de transformations

- **Inconvénients**

- ▶ le format de données établi entre composants
- ▶ performance : parsing/output

Architectures d'application

- les applications d'un domaine partagent souvent une architecture commune qui reflète les besoins du domaine
- architectures génériques d'application
 - ▶ architecture pour un type de logiciel qui peut être adaptée et configurée pour correspondre à des besoins spécifiques

Utilisation

- point de départ pour la conception de l'architecture d'un système
- checklist pour une conception déjà réalisée
- organisation d'une équipe de développement
- identifier les composants réutilisables
- vocabulaire pour communiquer

Exemples d'applications

- ***Data processing***

- ▶ traitement de données par lots (facturation)

- ***Transaction processing***

- ▶ requêtes/mises à jour d'une BDD (e-commerce)

- ***Event processing***

- ▶ les actions dépendent d'événements (embarqué)

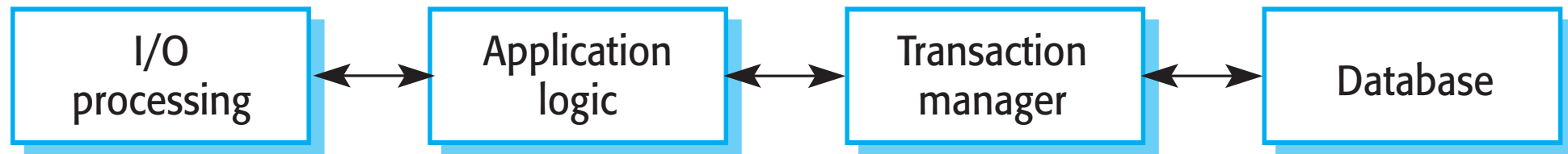
- ***Language processing***

- ▶ les actions obtenues en interprétant un langage

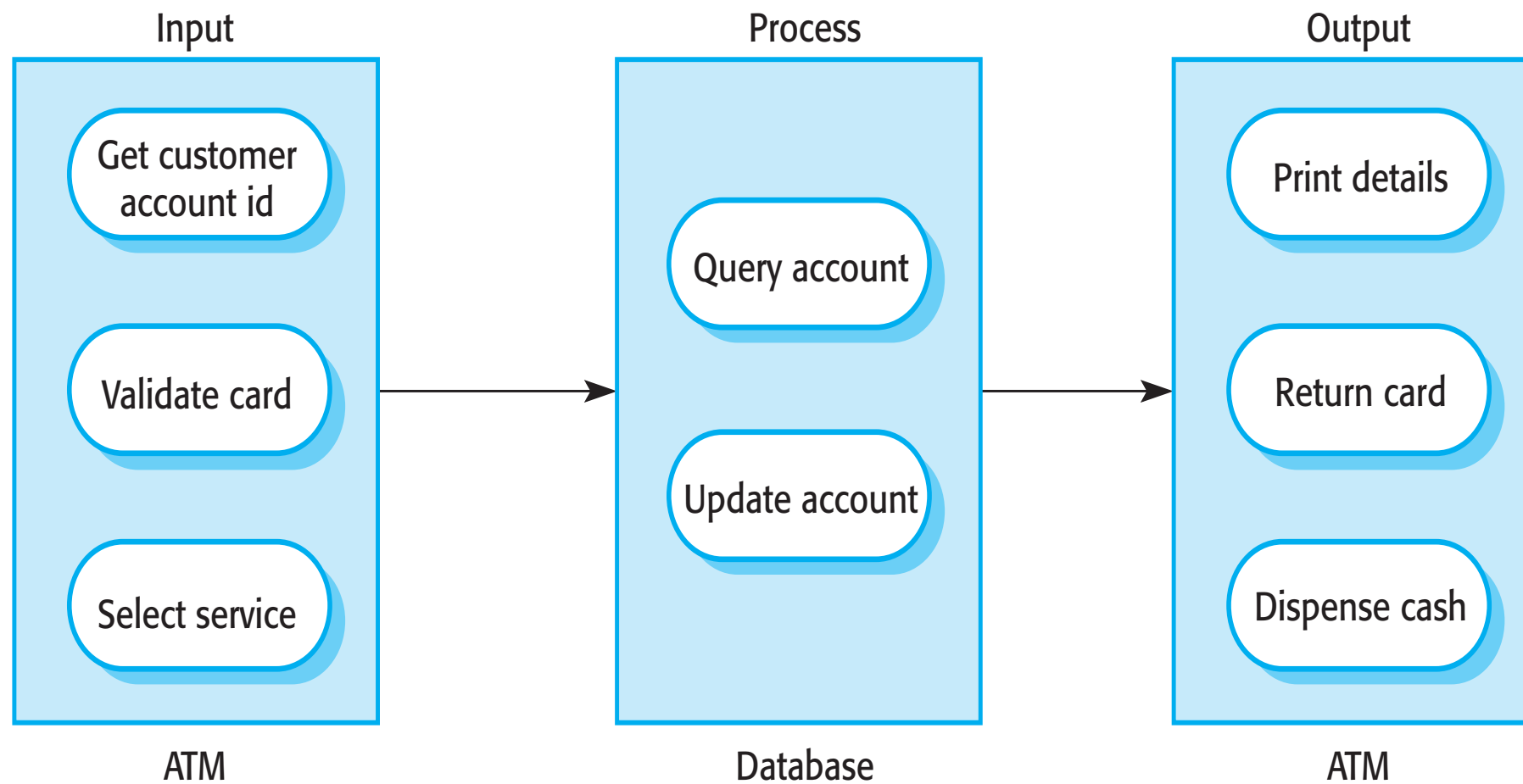
Transaction processing

- traitement de requêtes d'information ou de mise à jour par rapport à une BDD
- *transaction* : séquence d'opérations pour accomplir un objectif
- l'utilisateur fait des requêtes (asynchrones) traitées par un manager de transactions

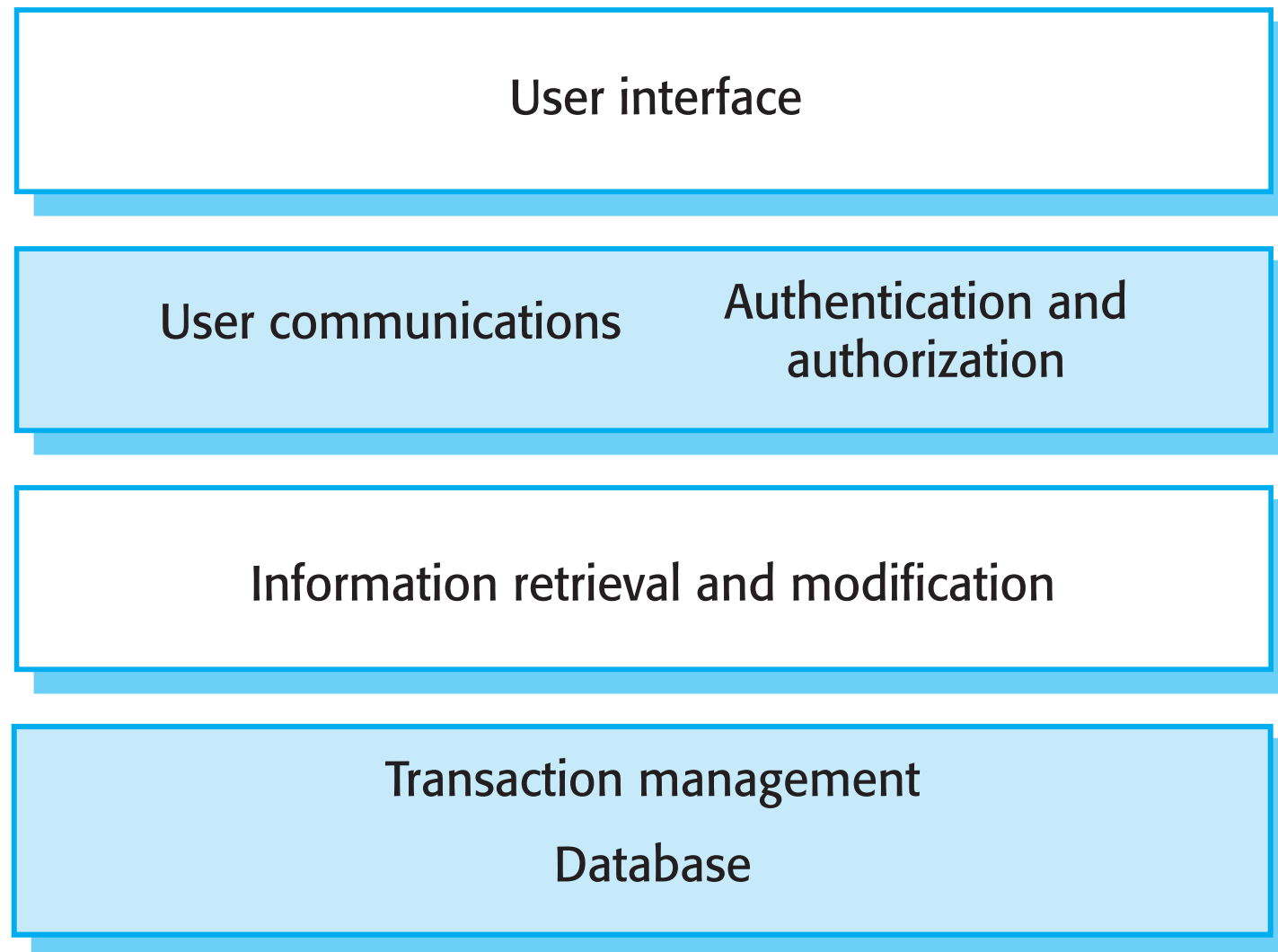
Structure



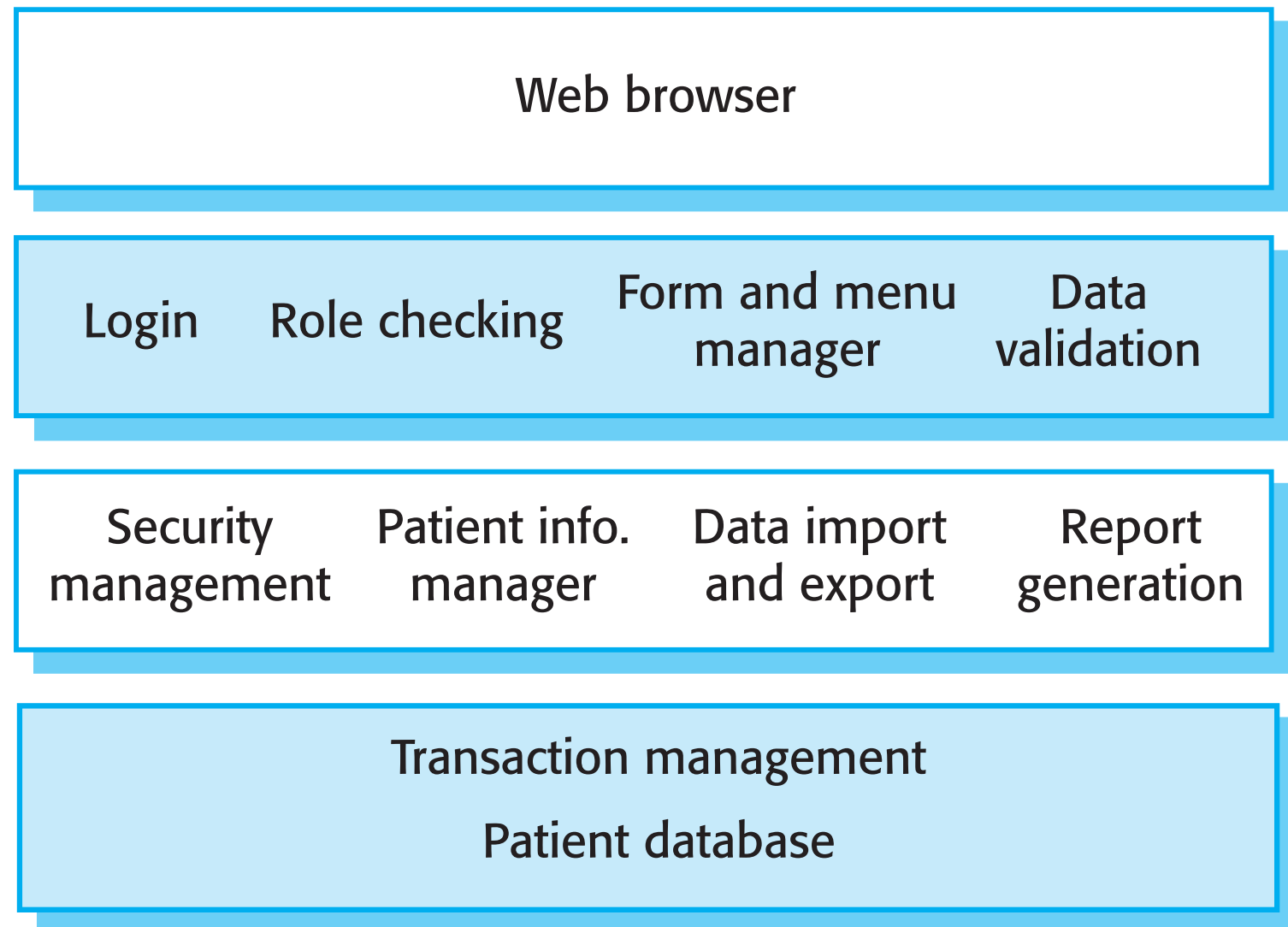
Exemple : DAB



Systemes d'information

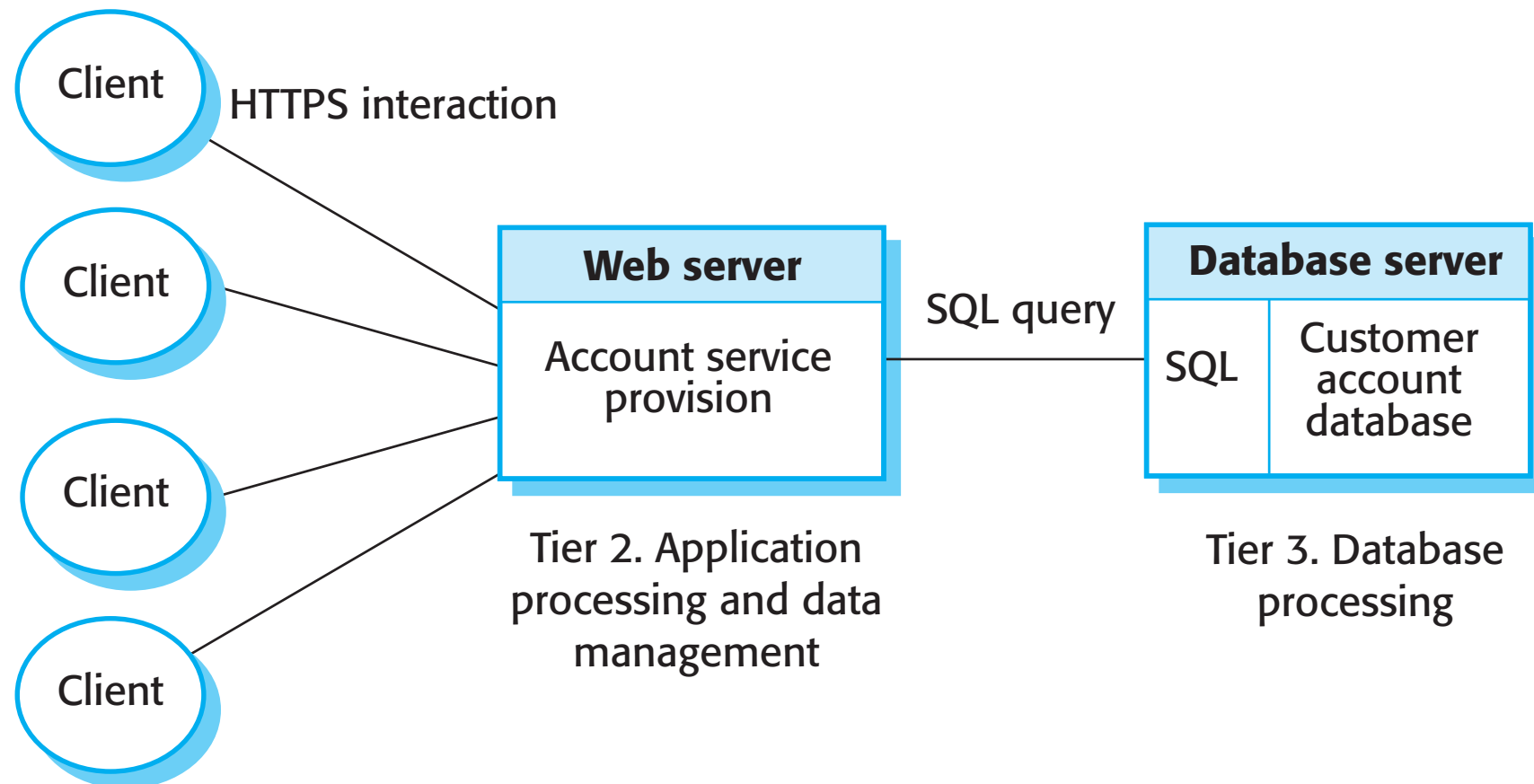


Exemple: dossier médical



Exemple : banque à distance

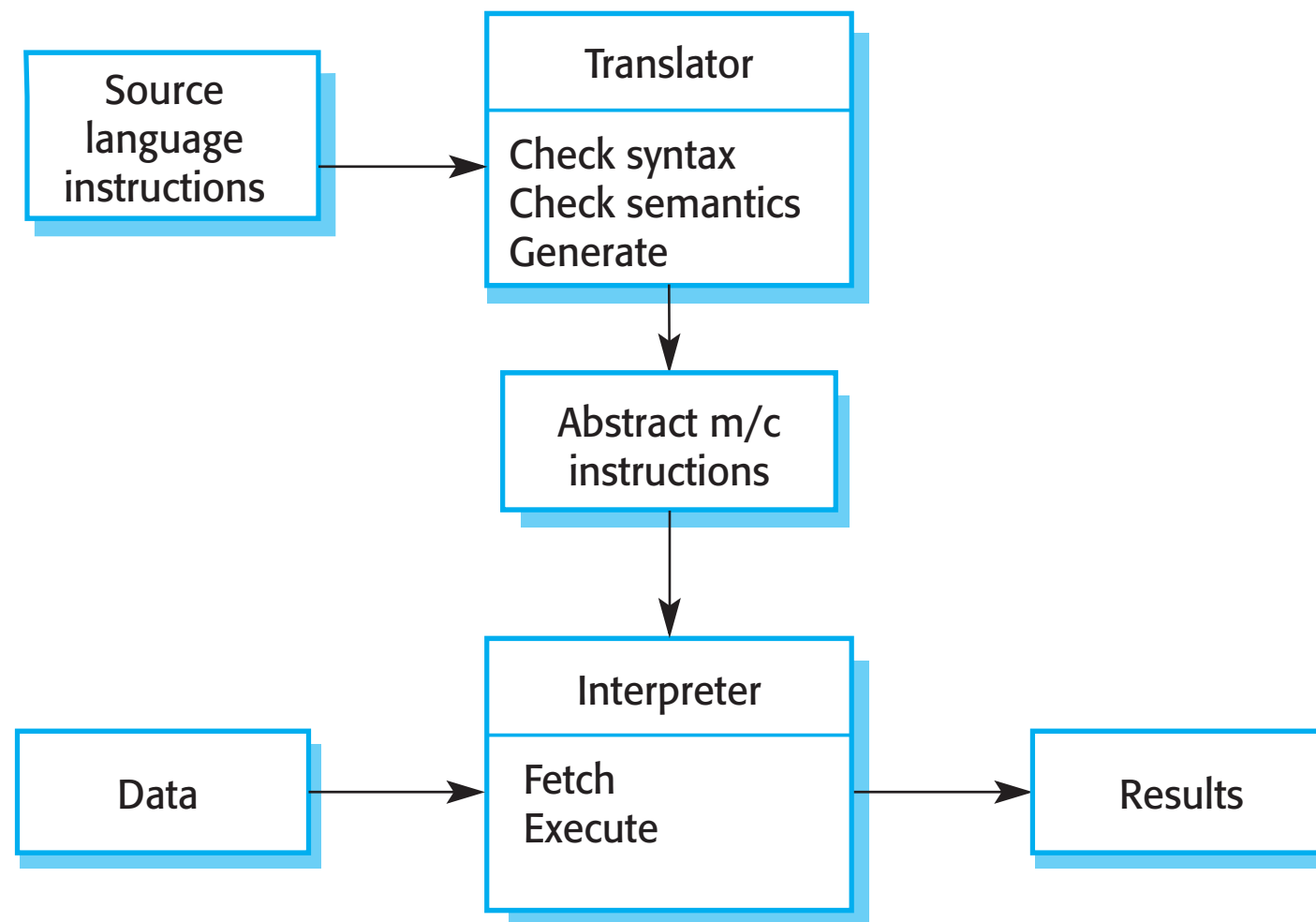
Tier 1. Presentation



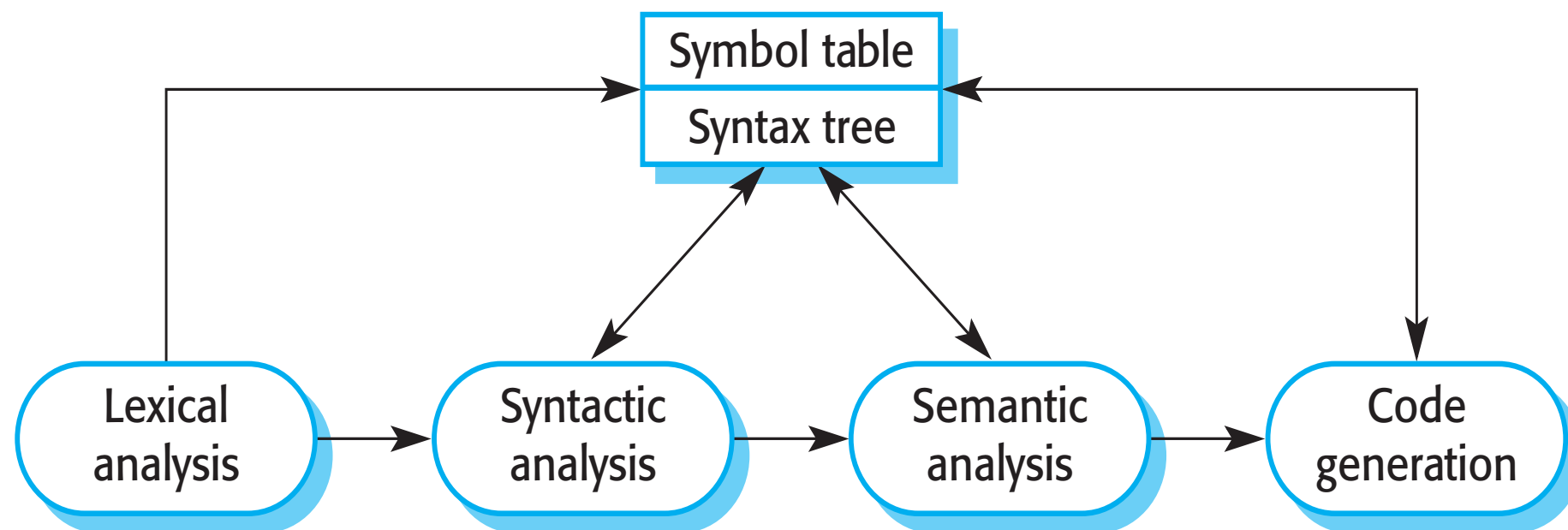
Language processing

- accepte langage (naturel ou machine) pour générer une autre représentation
- peut disposer d'un interpréteur pour exécuter les instructions obtenues

Structure

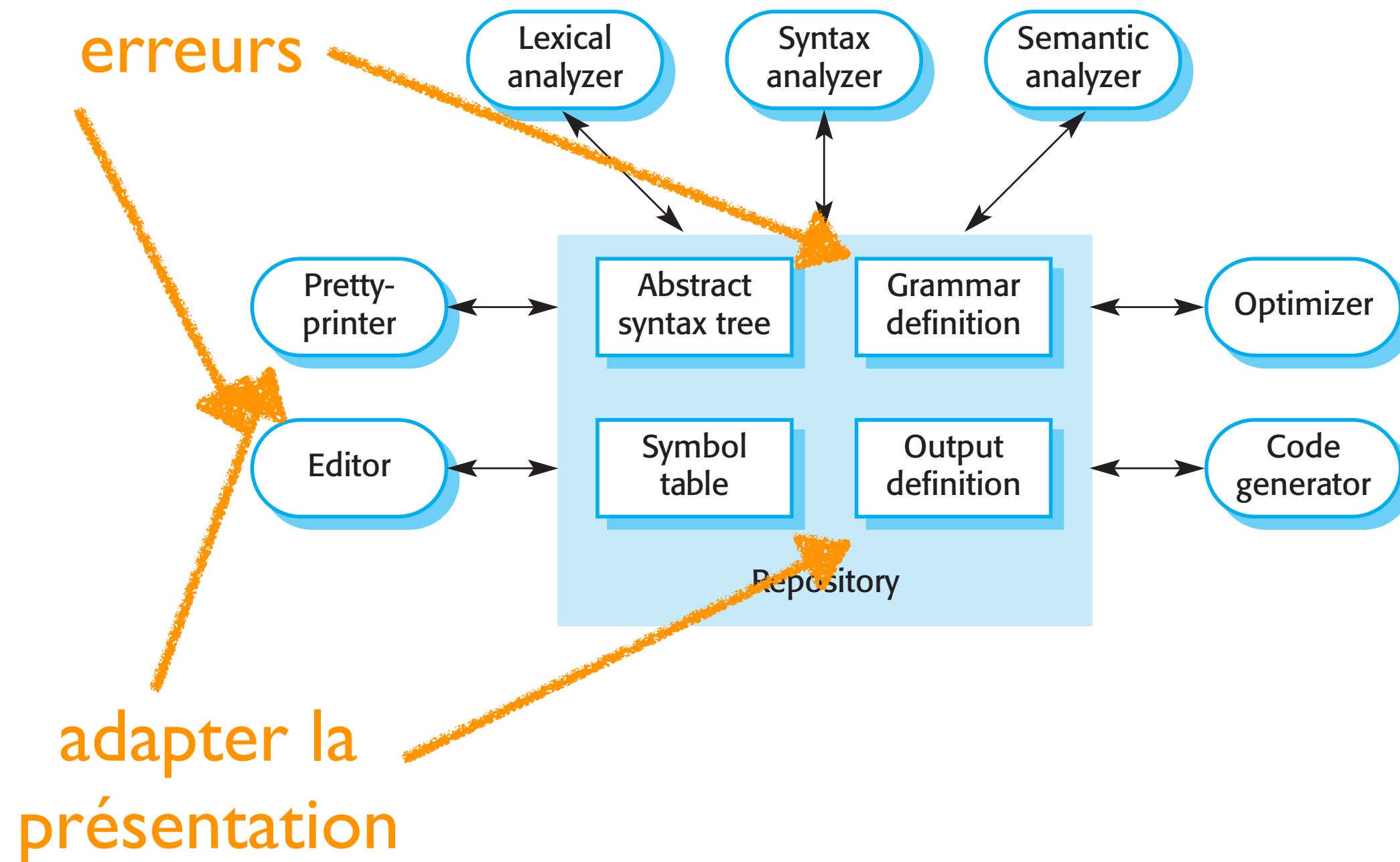


Exemple : compilateur



Exemple : compilateur

détecter les
erreurs



Key points

- **software architecture**  description de l'organisation d'un logiciel
- les choix architecturaux dépendent
 - ▶ du type de l'application,
 - ▶ de la distribution du système
 - ▶ des styles architecturaux choisis

Key points

- **perspectives possibles** : conceptuelle, logique, processus , développement
- **patterns architecturaux**
 - ▶ description de l'architecture
 - ▶ cas d'utilisation
 - ▶ avantages
 - ▶ inconvénients