Réutilisation de code

420-1B2-SW

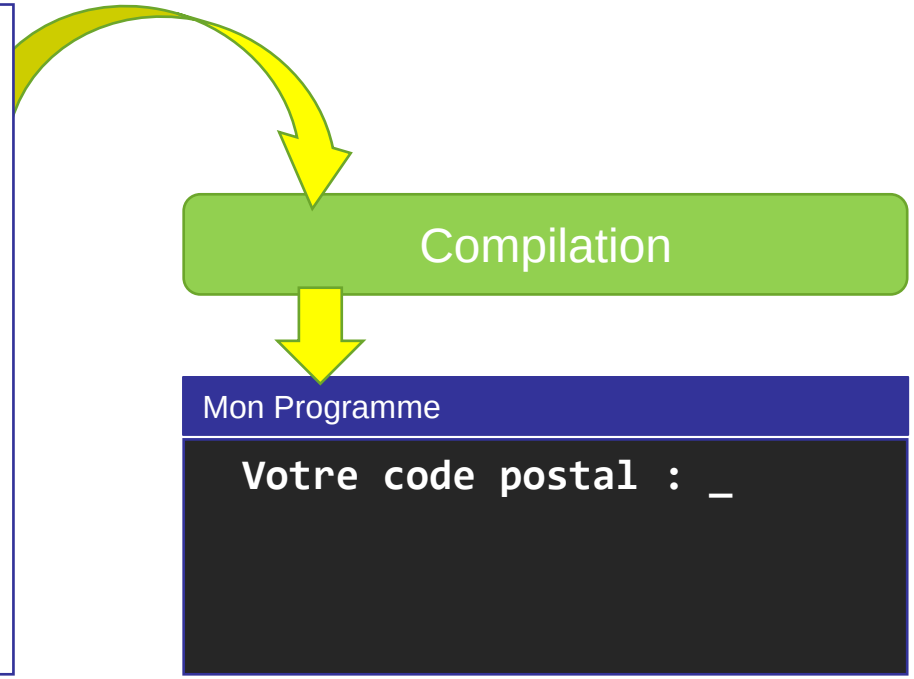
Développement avec base de données

Réutilisation de code

- Un premier programme informatique présente une architecture assez simple :

Code

```
01 – Afficher « Votre code postal : »

02 – Saisir code postal de l’utilisateur

03 – Vérifier si code postal est valide

04 – Boucler à l’étape 02 si invalide

05 – Afficher « bravo! »
```

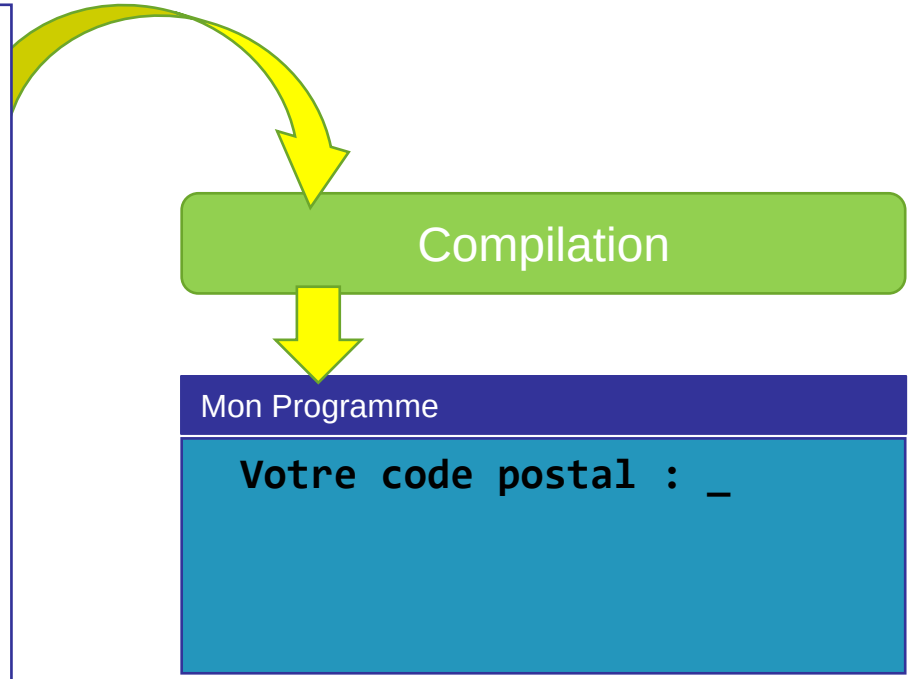


Réutilisation de code

- À vouloir améliorer un programme, son concepteur crée des fonctions et trucs *cools*…

Code

```
00 – Écran bleu avec texte noir

01 – Afficher « Votre code postal : »

02 – Saisir code postal de l'utilisateur

03 – Vérifier si code postal est valide

04 – Boucler à l'étape 02 si invalide

05 – Afficher « bravo! avec feux
     d'artifice »
```



Réutilisation de code

- …fonctions et trucs *cools* que son concepteur voudrait réutiliser pour plusieurs raisons :
  - Perte de temps que celle de coder à nouveau ce qui a déjà été fait

    Pourquoi ne pas prendre ce temps si précieux à coder de nouveaux trucs *cools*!

  - Risques d'erreurs

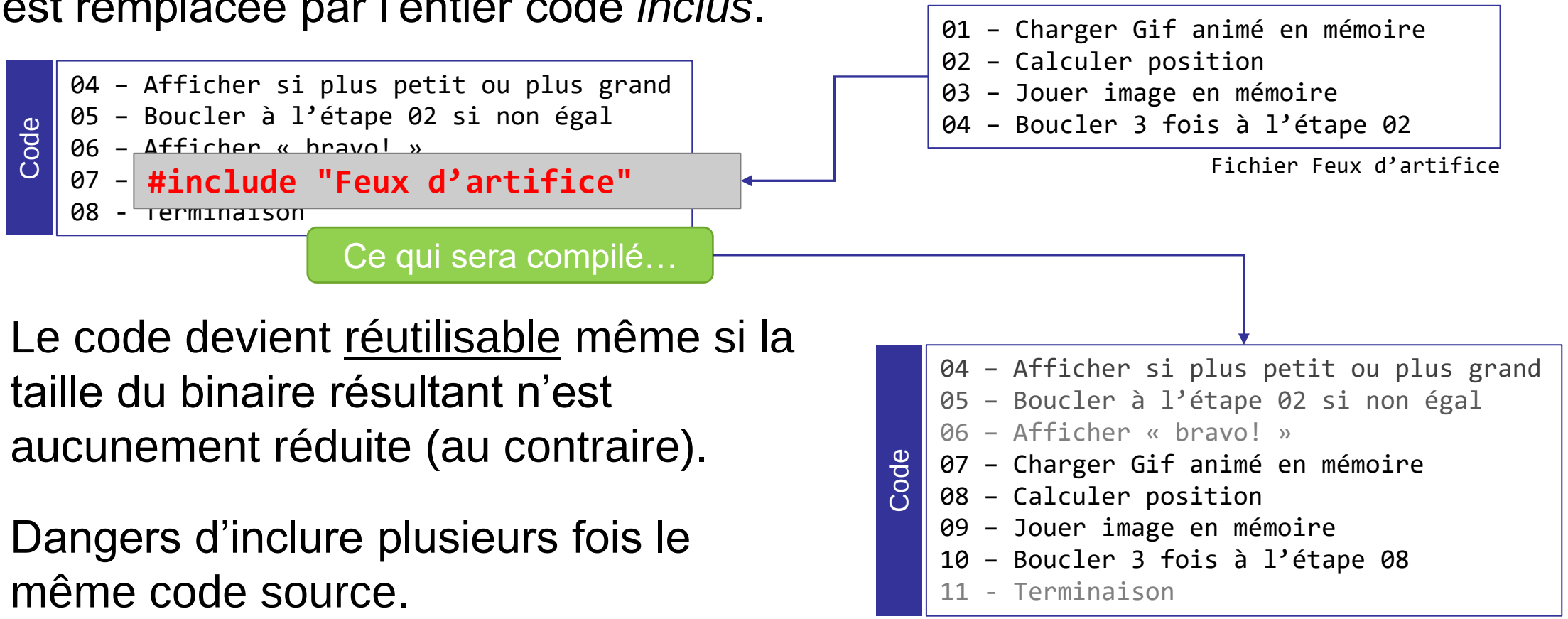
    Risques d'introduire des comportements différents entre chaque occurrence (répétition) du code.

  - Maintenance difficile et risquée

    Dans le cas où le comportement du code doit être corrigé, remplacé, etc. on va s'adonner à une contre-productive séance de *« Find and replace »* dans le code ☹

Réutilisation de code

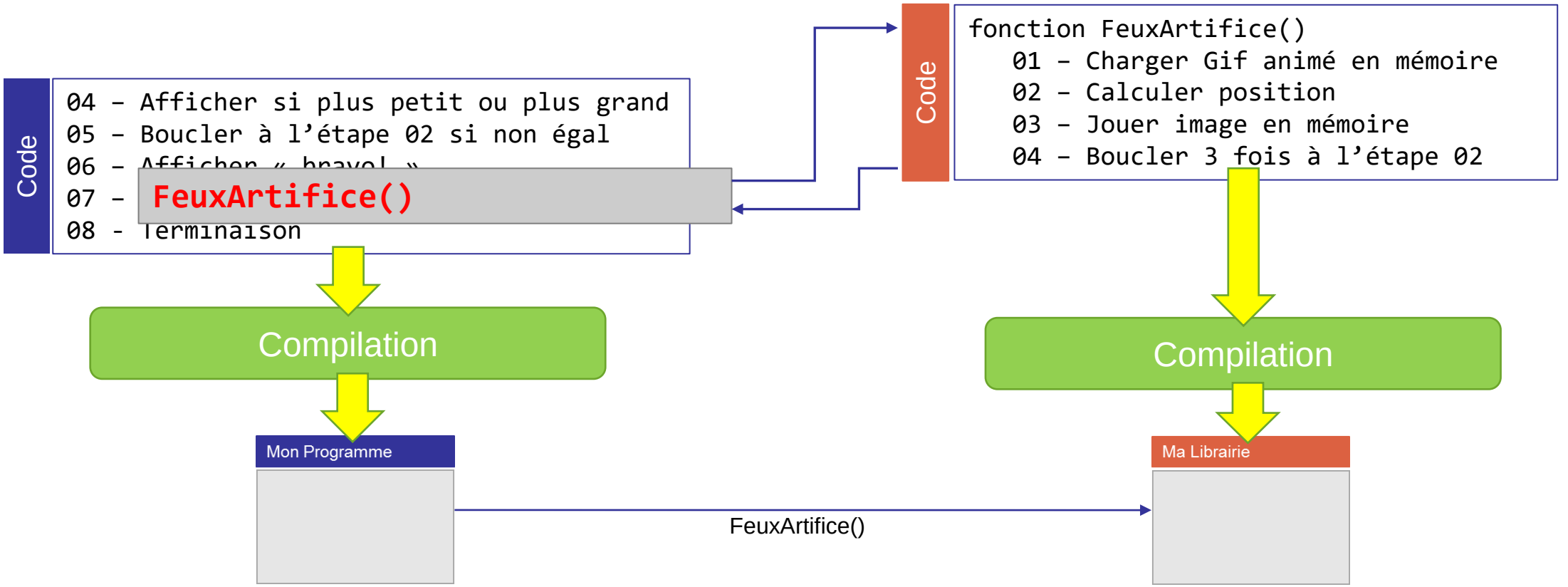
- Deux approches pour réutiliser du code :
  - Une technique est d'**inclure le code** provenant d'un tiers fichier source. Lors de la compilation/interprétation, la directive du code *incluant* est remplacée par l'entier code *inclus*.



  + Le code devient réutilisable même si la taille du binaire résultant n'est aucunement réduite (au contraire).
  – Dangers d'inclure plusieurs fois le même code source.

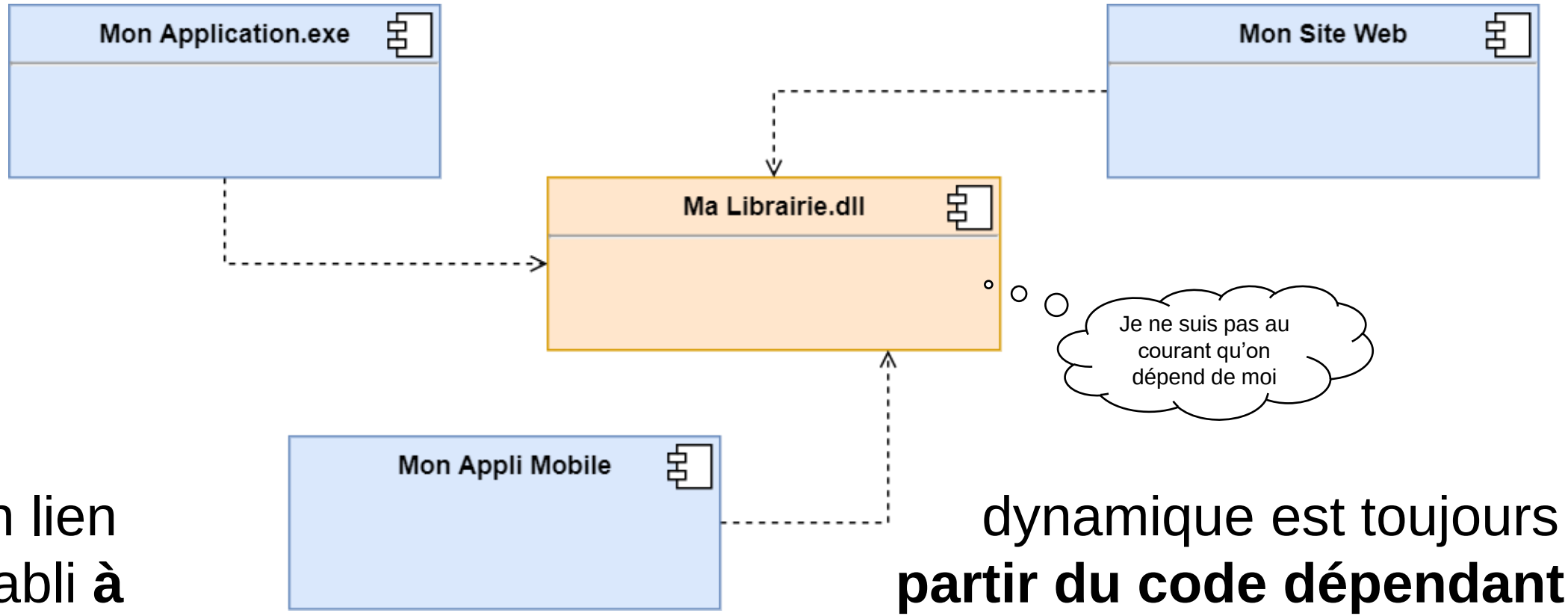
Réutilisation de code

- Deux approches pour réutiliser du code :
  - Une autre technique permet d'établir une référence binaire, un **lien dynamique** vers un binaire résultant de la compilation d'un autre fichier.



Réutilisation de code

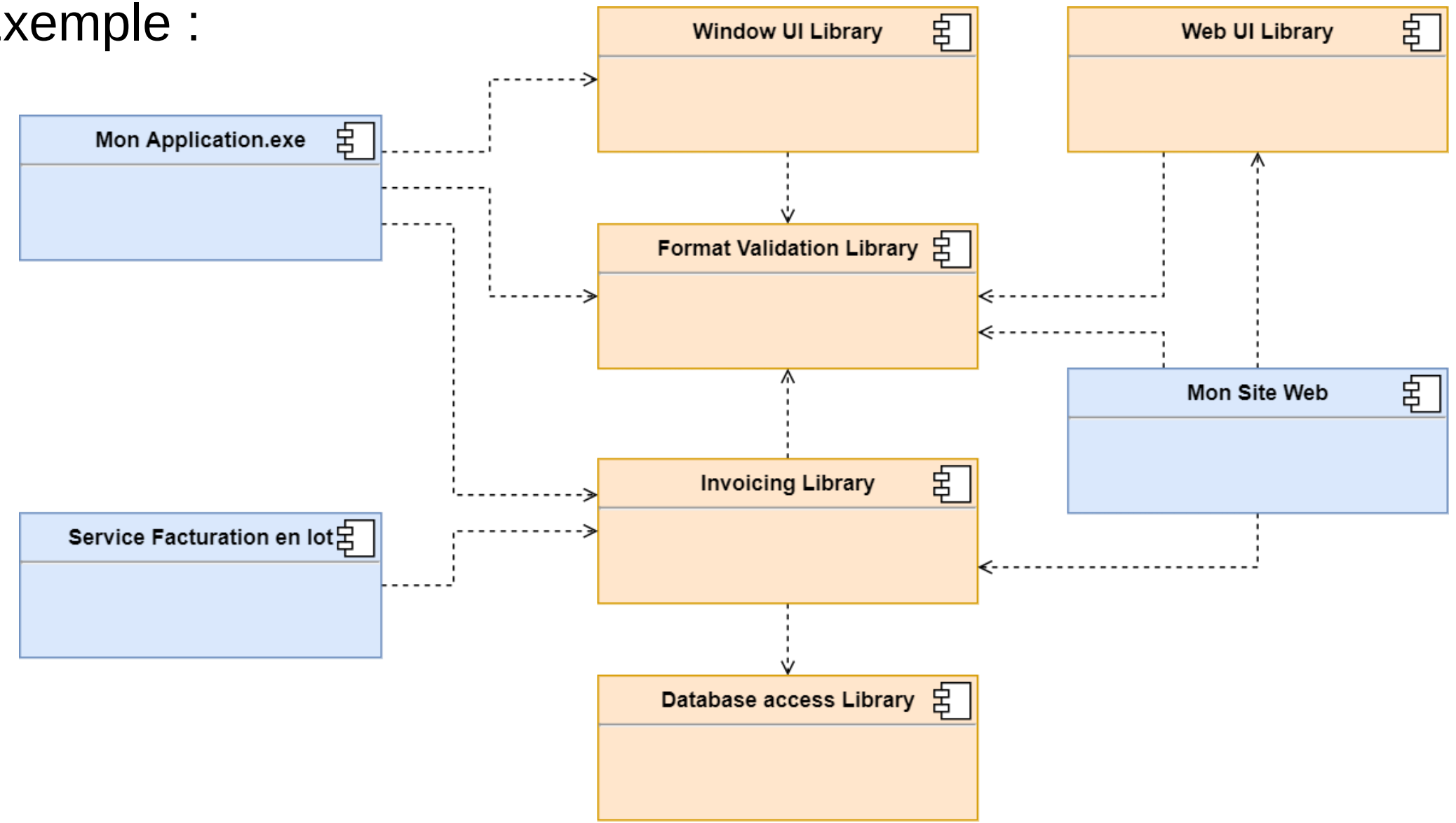
- Le **lien dynamique** permet à un code d'être pleinement réutilisé en réduisant la taille totale des binaires résultants.



- Un lien dynamique est toujours établi **à partir du code dépendant vers le code duquel il dépend**. Ce dernier n'a pas à connaître l'existence de la relation de dépendance.

Réutilisation de code

- Exemple :



Accès aux données

420-1B2-SW

Développement avec base de données

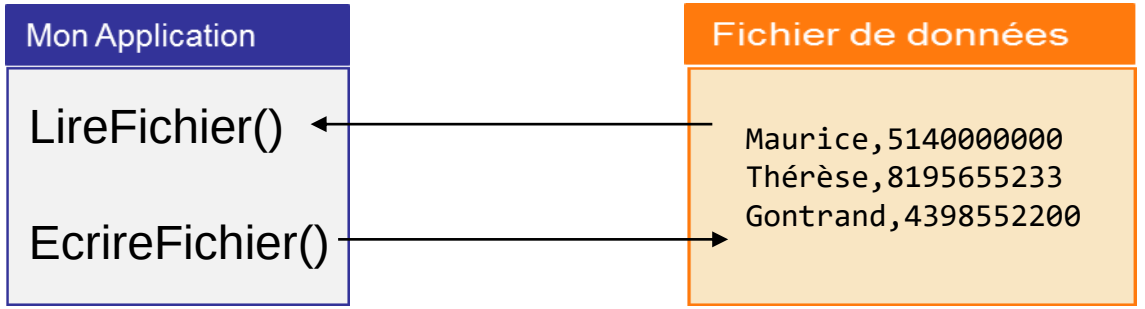
Accès aux données

- Un premier programme informatique présente une architecture assez simple :

Code

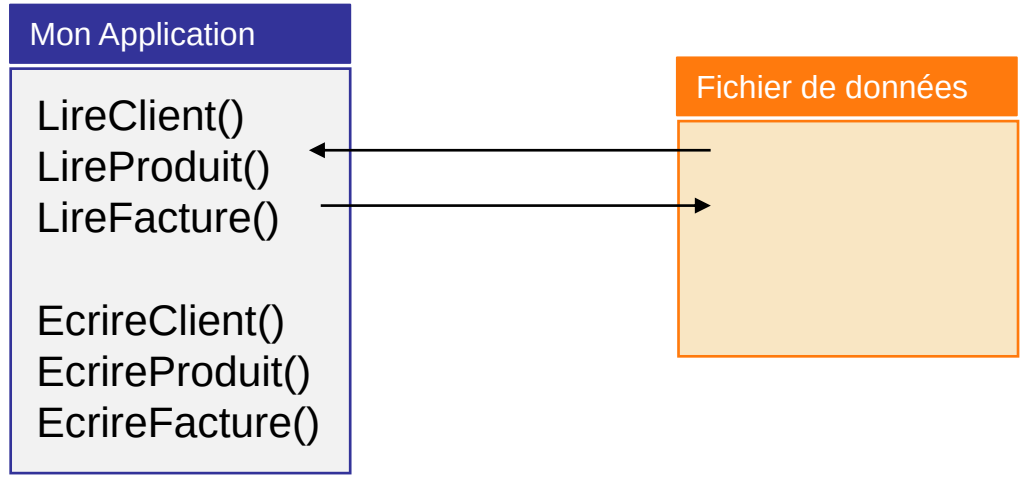
```
01 – Entrez le nom du client/prospect

02 – Entrez son téléphone

03 – Boucler à l'étape 01
```

- Mais arrive le jour où il est nécessaire de stocker, persister les données (nom et coordonnées du client, dans le précédent exemple)



Accès aux données

- Un code permettant d'accéder aux données peut rapidement s'avérer très répétitif et/ou très désordonné.



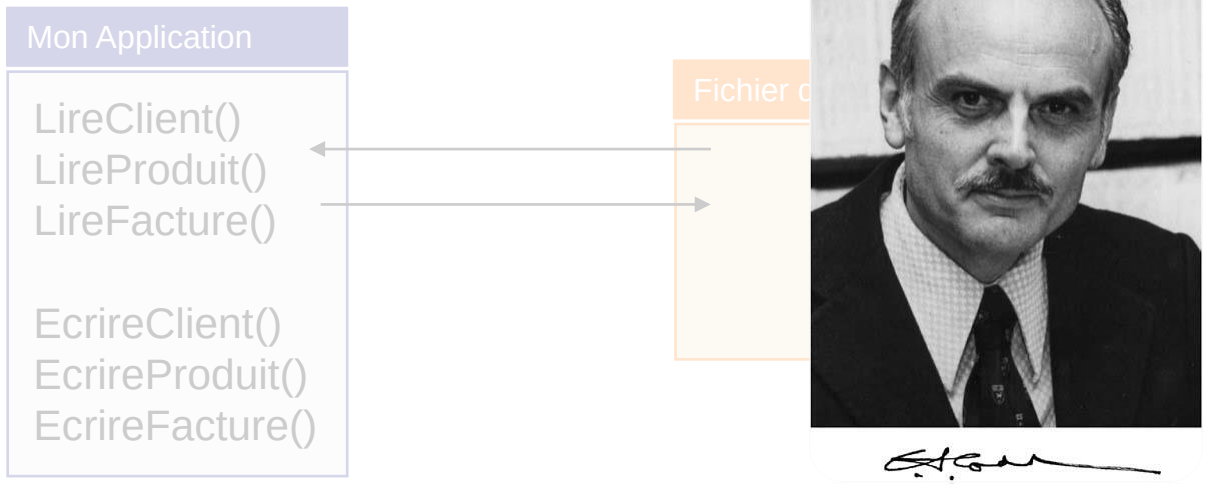
Code

```
01 – Numéro = lirePosition(0, 7)
02 – Sexe = lirePosition(8, 8)
03 – Nom = lirePosition(9, 20)
04 – Prenom = lirePosition(30, 40)
05 – DateNaissance = lirePosition(50,58)
```

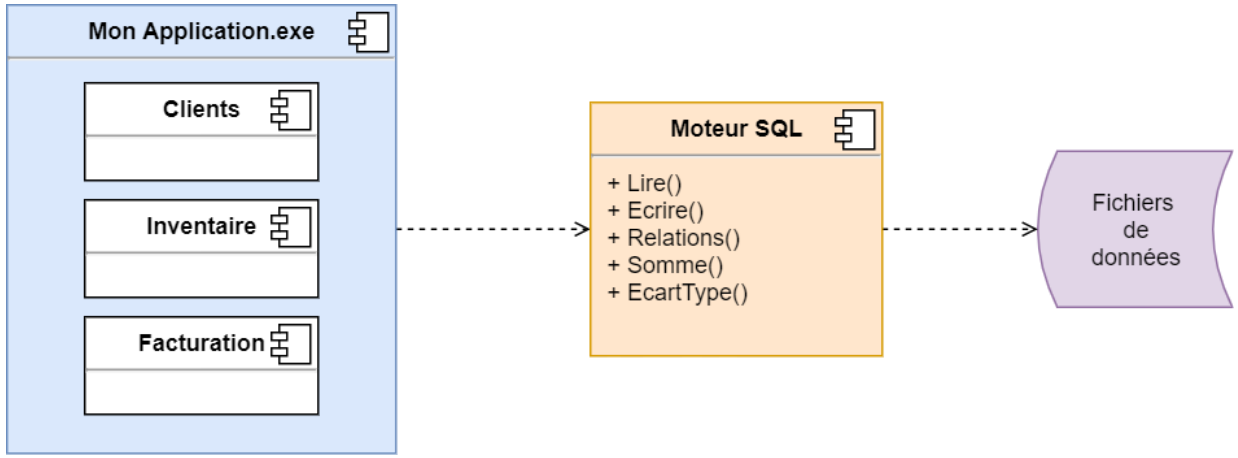
0 10 20	30 40	50 60
00000101MAYKROYD	DAN	19520701CAOTTAWA
00001548MMORANIS	RICK	19530418CATORONTO
00000195MMURRAY	BILL	19500921USWILMETTE
00000601MRAMIS	HAROLD	19441121USCHICAGO
00000244FWEAVER	SIGOURNEY	19491008USNEW-YORK

Accès aux données

- Un code permettant d'accéder aux données peut rapidement s'avérer très répétitif et/ou très désordonné.



- Pour cette raison, le **modèle relationnel** inventé en 1970 par Edgar Codd travaillant pour IBM permet de diminuer la charge de travail nécessaire aux développeurs pour accéder aux données de manière structurée et efficace.



Accès aux données

- Le SQL (Structured Query Language) établi une couche d'abstraction entre le code de l'application et les données.

Code

```
01 – execSQL("SELECT * FROM Clients WHERE Numero=1548")
02 – Nom = lireChamp("Nom")
03 – Prenom = lireChamp("Prenom")
```

SQL

```
01 – Retourne lirePosition(9, 20)
```

SQL

```
01 – Retourne lirePosition(30, 40)
```

0	10	20	30	40	50	60
00000101MAYKROYD			DAN		19520701CAOTTAWA	
00001548MMORANIS			RICK		19530418CATORONTO	
00000195MMURRAY			BILL		19500921USWILMETTE	
00000601MRAMIS			HAROLD		19441121USCHICAGO	
00000244FWEAVER			SIGOURNEY		19491008USNEW-YORK	

Le code n'a plus à se soucier des technicalités permettant d'accéder aux données (position, longueur, etc).

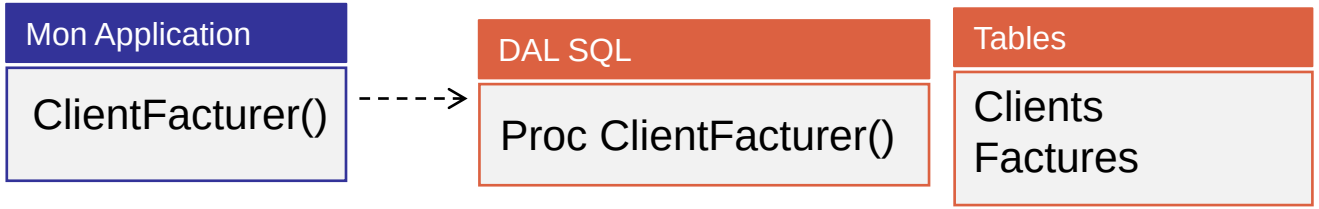
Accès aux données

- Un code permettant d'accéder à des données, même à l'aide de SQL, peut rapidement s'avérer très répétitif et quand même difficile à structurer adéquatement.
  - Le problème est que SQL donne à l'application accès à une quantité de données (client, produit, facture, *etc*) sans pour autant restreindre la façon d'y accéder.

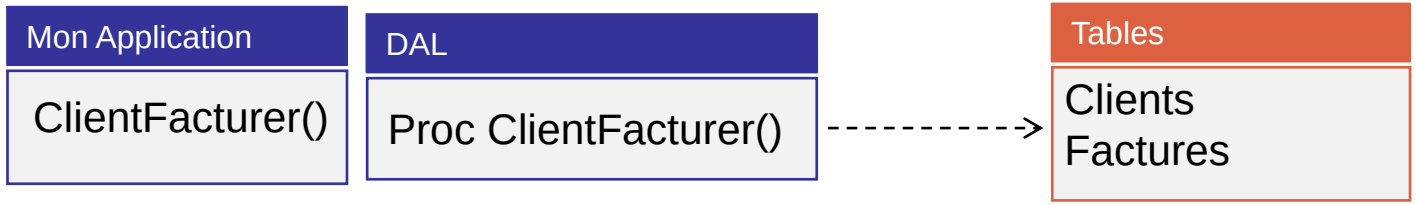
  - De plus, SQL de base n'offre pas une grande diversité de contraintes d'entreprise (permet d'ajouter une facture à un client sans en vérifier la marge de crédit, *etc*).

Accès aux données

- Une couche logicielle supplémentaire (DAL : Data Access Layer) permettra de structurer l'accès aux données en offrant des méthodes encapsulant la logique.
  - Certains manufacturiers SQL prévoient d'implémenter ces méthodes à même les objets de données (déclencheurs, procédures, vues, *etc*).



  - Sinon, cette couche est implémentée à même l'application :



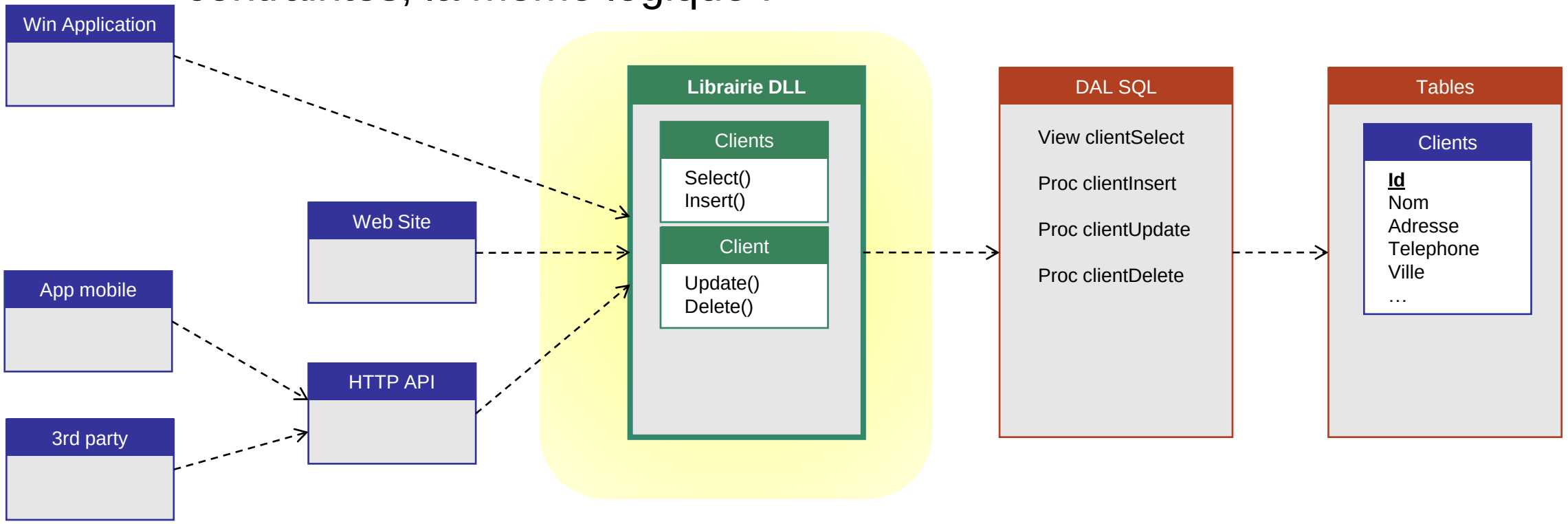
Architectures à tiers

420-1B2-SW

Développement avec base de données

Architectures à tiers

- Tel que vu précédemment, l'intégration de code au sein d'une **librairie** de **liens dynamiques** permet au code d'être réutilisé.
  - La centralisation de la logique d'entreprise permet à diverses applications de natures différentes d'appliquer les mêmes règles, les mêmes contraintes, la même logique :



Architectures à tiers

- On nomme ces architectures **à tiers** traditionnellement composées de trois constituants :
  1. Interface graphique (UI)

     Ensemble des moyens de communication avec l'utilisateur pour lui afficher des informations mais aussi pour saisir ses données (étiquette *(label)*, bouton, zone de texte, calendrier, grille, formulaire, rapport, *etc*.)
  2. Code

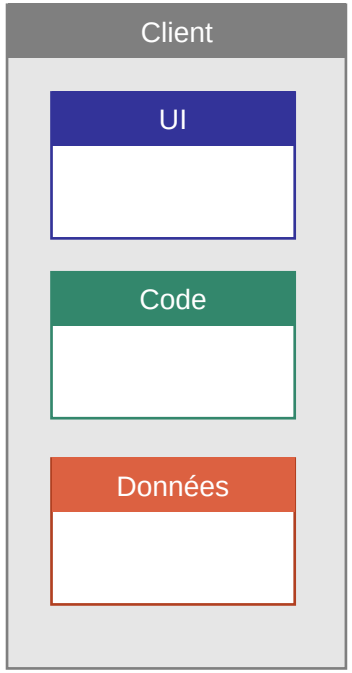
     Les classes, les méthodes, le code procédant à la validation des données saisies et à leur formatage pour des fins d'affichage, de rendu à l'utilisateur.
  3. Données

     Stockage des données pour en assurer la persistance

- L'architecture sera qualifiée selon l'emplacement physique d'**où** s'exécute chacun de ces composants…

Architectures à tiers

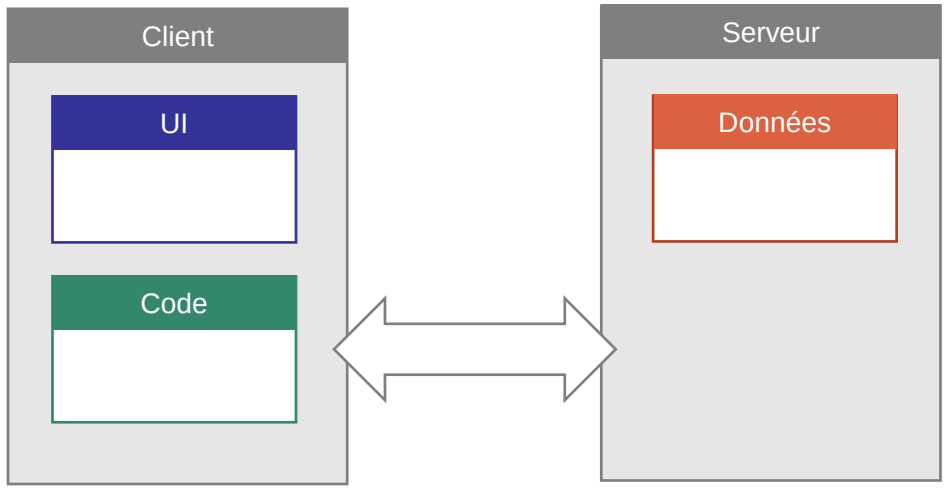
- Une **architecture 1-tier** est la plus simple intégration des 3 composants au sein d'un même entité situés à un emplacement unique.



- Les données peuvent être stockées au sein d'un fichier situé à l'extérieur de l'exécutable, le code peut être stocké au sein d'une DLL externe à l'EXE mais on parle tout de même d'architecture 1-tier puisque ces composants sont tous situés sur le même poste client.

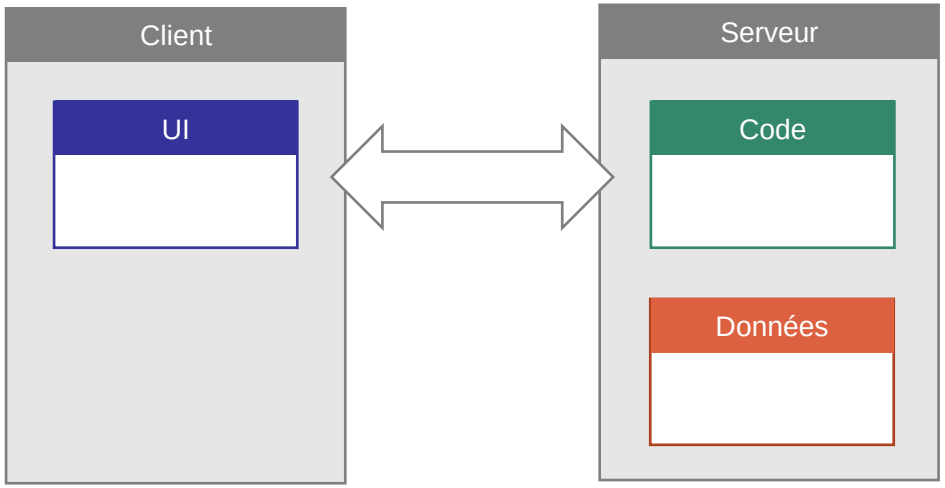
Architectures à tiers

- Une **architecture 2-tiers** va centraliser les données en un emplacement partagé à l'ensemble des clients.



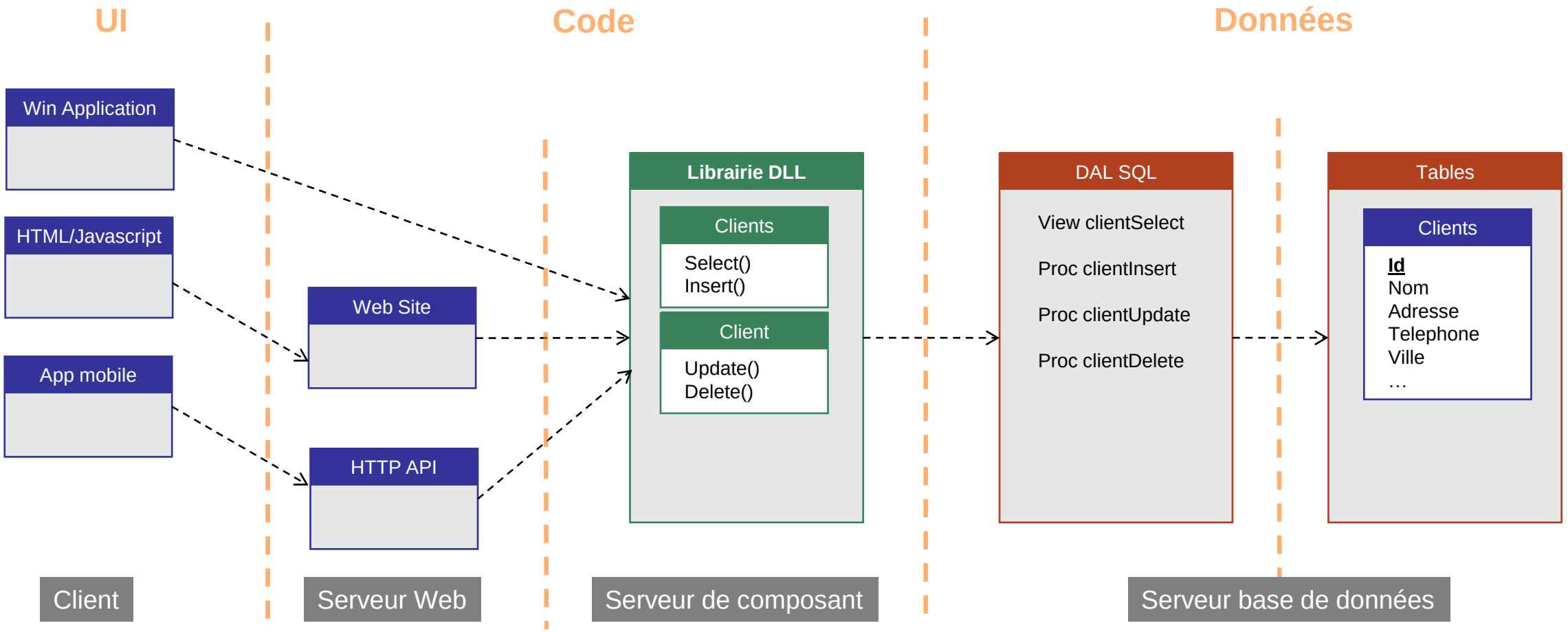
Le code peut demeurer sur le client *(dans l'EXE ou une DLL, c'est sans importance)*.
On parle alors d'un **client lourd** *(Thick client ou Fat client)*.

Le code peut s'exécuter d'une DLL situé sur un serveur.
On parle alors d'un **client léger** *(Thin client)*.
Cette architecture est plus simple à maintenir qu'un client lourd puisque la mise à jour du composant de code sur le serveur impacte l'ensemble des clients.



Architectures à tiers

- Au-delà de 2-tiers, on nommera ces architectures ***n*-tiers**, chaque composant logiciel pouvant être explosé sur autant de niveaux logiques que physiques :



Architectures infonuagiques

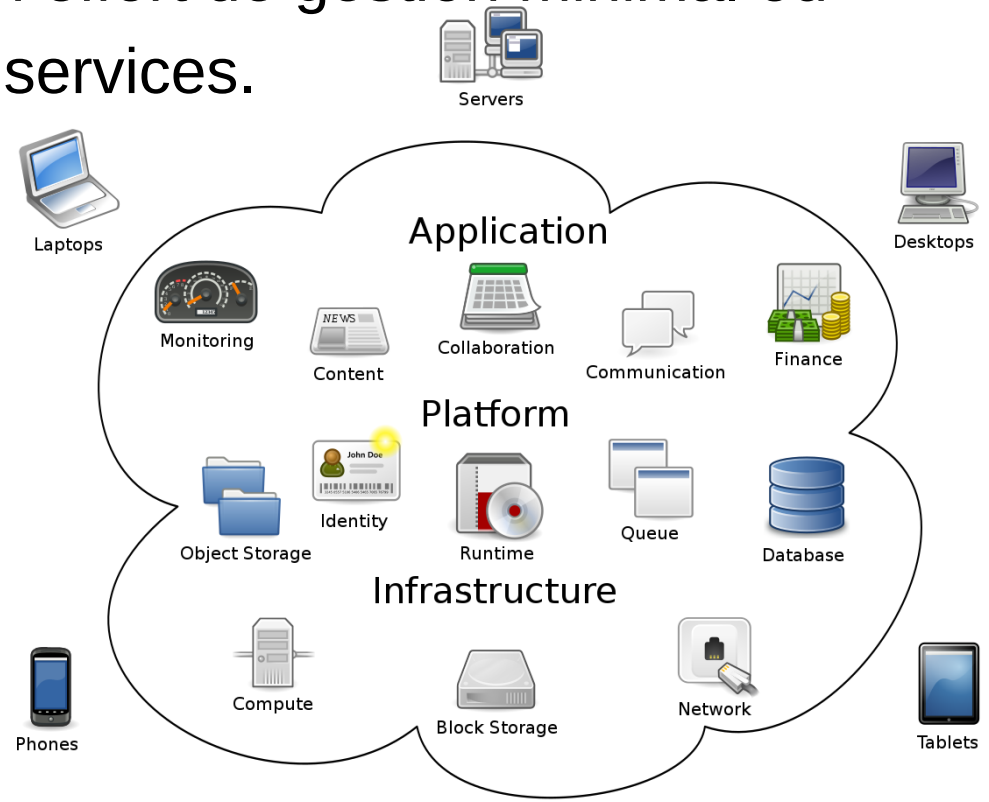
420-1B2-SW

Développement avec base de données

Architectures infonuagiques

- **Infonuagique** - *cloud computing*

  Modèle permettant un accès réseau omniprésent, pratique et à la demande à un *pool* partagé de ressources informatiques configurables (par exemple, réseaux, serveurs, stockage, applications et services) qui peuvent être rapidement provisionnés et libérés avec un effort de gestion minimal ou interaction minimal avec le fournisseur de services.



Architectures infonuagiques

- Il existe trois principales approches quant à la mise en œuvre de technologies par le *cloud*, définies selon l'emplacement physique d'**où** s'exécute chacun de ses composants:

1. Software as a service (SaaS)

   Le logiciel est installé sur les serveurs distants du fournisseur plutôt que sur les machines du client. Les clients ne paient pas de licence d'utilisation pour une version mais utilisent librement le service en ligne ou, plus généralement, payent un abonnement

2. Platform as a service (PaaS)

   Met à la disposition du client un environnement d'exécution rapidement disponible, en lui laissant la maîtrise des applications qu'il peut installer, configurer et utiliser lui-même.

3. Infrastructure as a service (IaaS)

   Le client dispose sur abonnement payant d'une infrastructure informatique (serveurs, stockage, sauvegarde, réseau) qui se trouve physiquement tous chez le fournisseur.