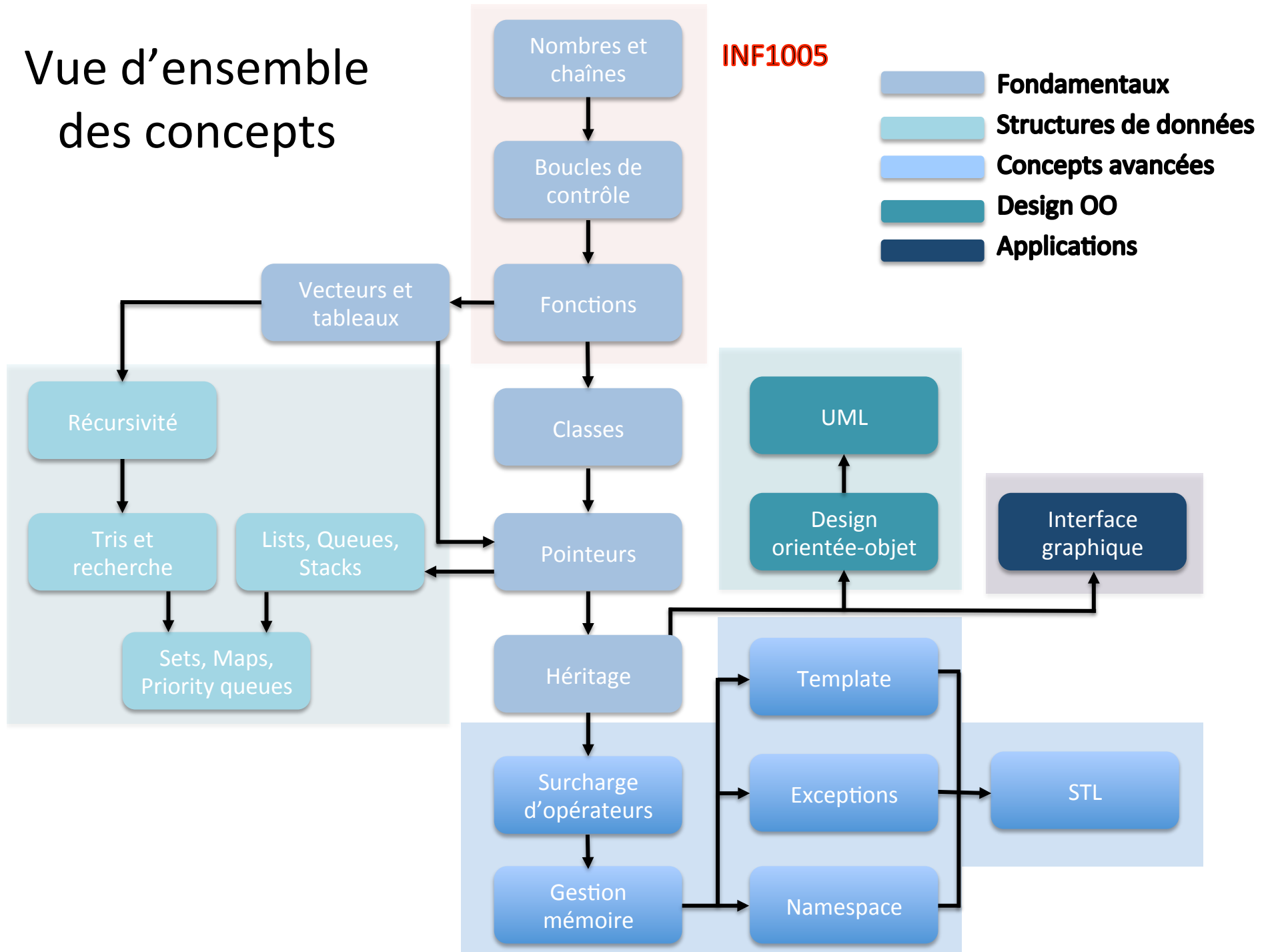


# ***Programmation orientée objet***

## **Vecteurs**

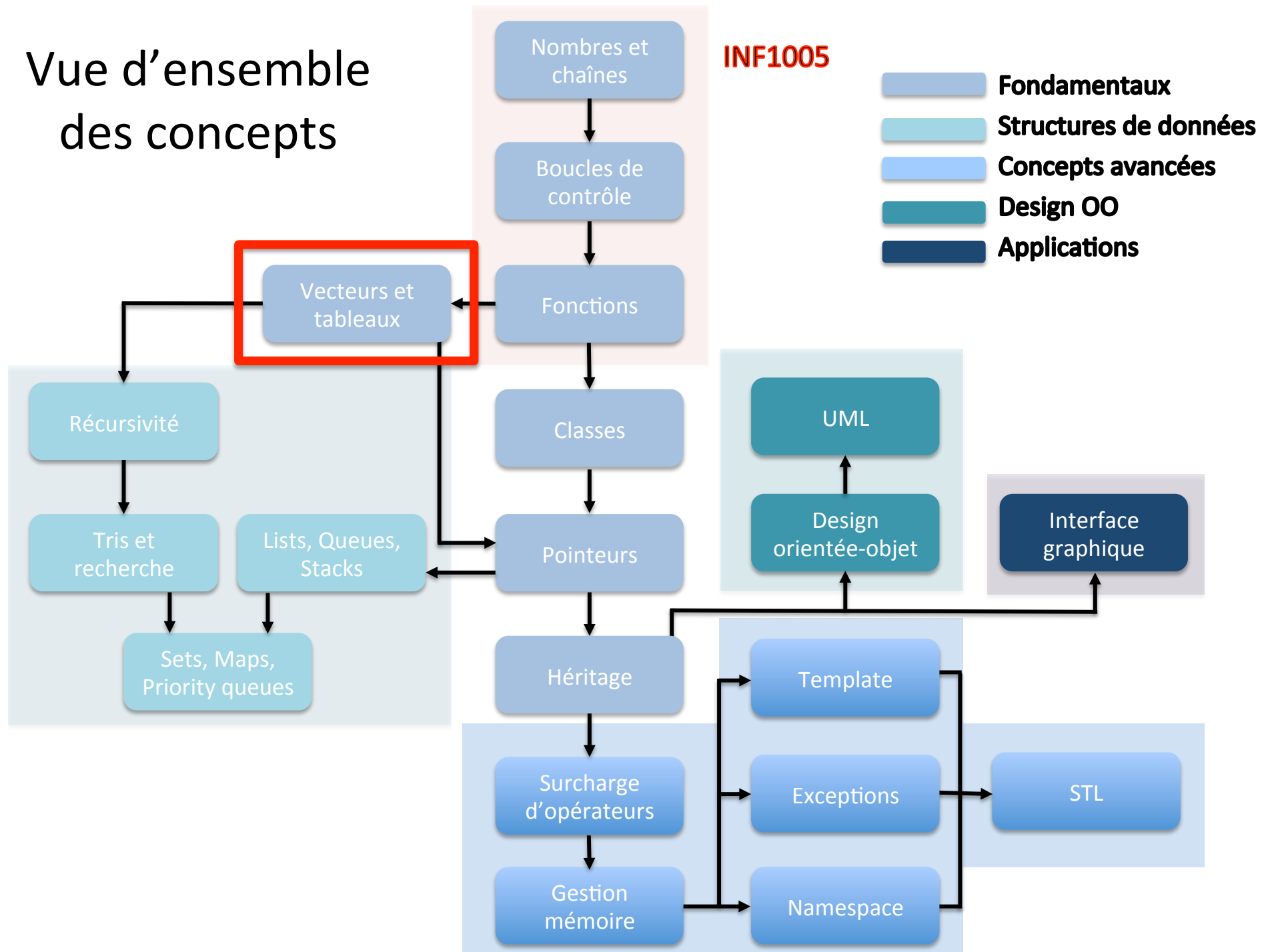
# Vue d'ensemble des concepts

INF1005



# Vue d'ensemble des concepts

# INF1005



# Vecteur

---

- Un vecteur est une **collection séquentielle d'items du même type**
- On l'utilise comme si c'était un tableau
- *Particularité intéressante:* si on insère un nouvel item dans un vecteur déjà plein, sa **taille sera automatiquement augmentée** afin de pouvoir recevoir ce nouvel item

## Vecteur (suite)

---

- La classe **vector** fait partie de la STL (Standard Template Library) de C++ et est une classe **générique**
- Quand on déclare un vecteur, il faut donc toujours spécifier le type des éléments qu'il contiendra:

```
#include <vector>

...

vector< double > salaires;
```

## Interface de vector

---

- L'interface de la classe vector contient deux méthodes principales:
  - `push_back(item)` pour ajouter un nouvel item à la fin du vecteur
  - `size()` pour connaître le nombre d'items contenus dans vecteur
- Le vecteur est un *conteneur séquentiel*: normalement, on le remplit en **ajoutant les éléments à la fin**

# Accès aux éléments d'un vecteur

---

- Pour accéder aux éléments d'un vecteur on utilise des indices entre crochets [ ], comme pour un tableau:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
cout << salaires[2]
```



# Accès aux éléments d'un vecteur

---

- Pour accéder aux éléments d'un vecteur on utilise des indices entre crochets [ ], comme pour un tableau:


```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
cout << salaires[2] // affiche 78000
```



# Exemple simple

---

```
int main()
{
    vector<int> unTableau;

    for (int i = 1; i < 10; ++i) {
        unTableau.push_back(i);
    }
    for (int j = 0; j < unTableau.size(); ++j) {
        cout << unTableau[j] << ' '; 
    }
    cout << endl;
    return(0);
}
```

# Exemple simple

---

```
int main()
{
    vector<int> unTableau;

    for (int i = 1; i < 10; ++i) {
        unTableau.push_back(i);
    }
    for (int j = 0; j < unTableau.size(); ++j) {
        cout << unTableau[j] << ' ';
    }
    cout << endl;
    return(0);
} //1 2 3 4 5 6 7 8 9
```

# Taille et capacité

---

- Il est important de distinguer la taille et la capacité d'un vecteur
- Un vecteur maintient à l'interne un tableau qui n'est pas nécessairement rempli
- *Taille*: nombre d'éléments effectivement **contenus** dans le tableau interne
- *Capacité*: taille du tableau **interne**. Elle est augmentée lorsqu'on veut insérer un nouvel item et que le tableau est plein

# Capacité

---

- Par défaut, la capacité d'un vecteur est 0
- On peut **initialiser la taille** d'un vecteur lors de la déclaration (dans ce cas, il sera rempli par des valeurs par défaut ou en appelant le **constructeur par défaut** dans le cas d'un vecteur d'objets):

```
vector<int> unTableau(10) ;
```

- Si on sait qu'un indice est inférieur à la capacité d'un vecteur, on peut modifier la valeur à cette position comme dans un tableau (soyez prudent!):

```
unTableau[6] = 64 ;
```

## Capacité (suite)

---

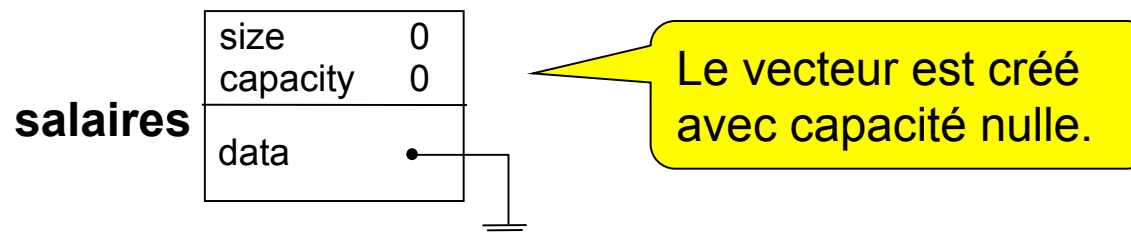
- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```

## Capacité (suite)

- Soit par exemple le programme suivant:

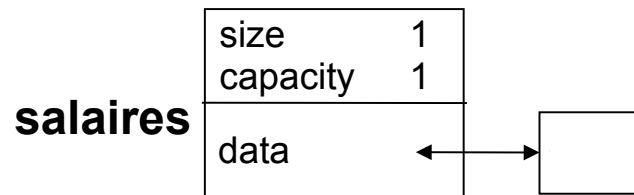
```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```



## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```

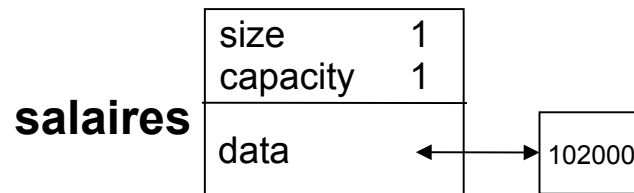


On alloue un tableau de taille 1 pour contenir le nouvel élément.

## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```



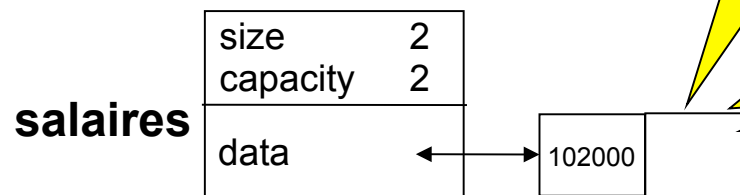
On alloue un tableau de taille 1 pour contenir le nouvel élément.



## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```



Comme le tableau est plein, on “double” sa taille pour pouvoir ajouter le nouvel élément.

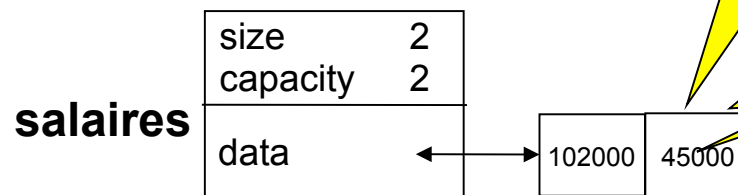
“double” ne veut PAS dire: étendre l'espace actuelle (à l'intérieur, le vector doit faire beaucoup plus!)

“double” veut dire: (1) allouer l'espace double sur le tas, (2) copier tout vers cette espace, et (3) détruire l'espace originale

## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```



Comme le tableau est plein, on “double” sa taille pour pouvoir ajouter le nouvel élément.

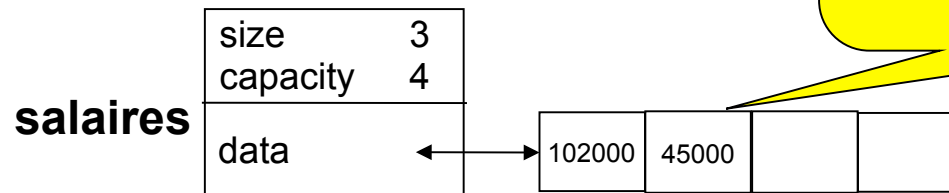
“double” ne veut PAS dire: étendre l'espace actuelle (à l'intérieur, le vector doit faire beaucoup plus!)

“double” veut dire: (1) allouer l'espace double sur le tas, (2) copier tout vers cette espace, et (3) détruire l'espace originale

## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```

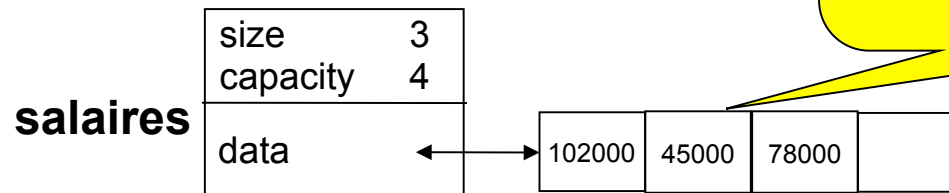


Encore une fois, comme le tableau est plein, on “double” sa taille pour pouvoir ajouter le nouvel élément.

## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```

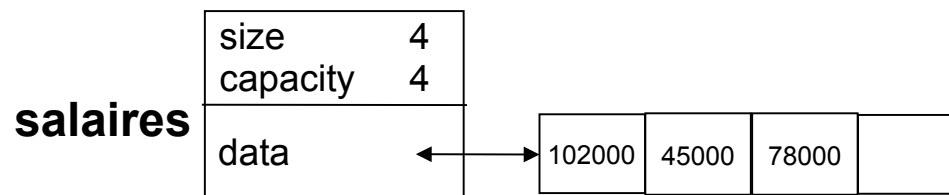


Encore une fois, comme le tableau est plein, on “double” sa taille pour pouvoir ajouter le nouvel élément.

## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```

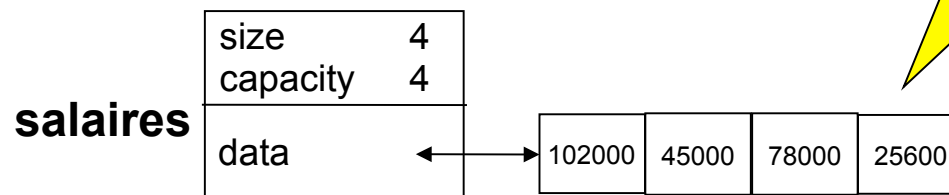


Remarquez qu'on n'a pas atteint la capacité maximale du tableau.

## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```



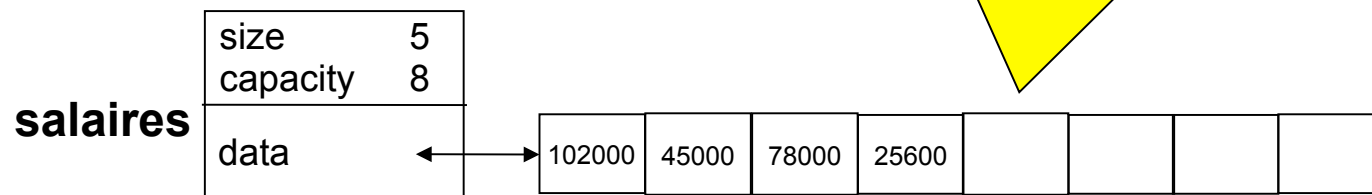
On ajoute le nouvel élément.

Remarquez qu'on n'a pas atteint la capacité maximale du tableau.

## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```

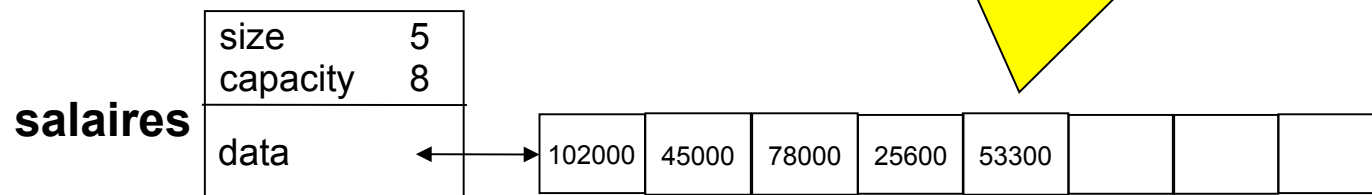


Encore une fois, comme le tableau est plein, on “double” sa taille pour pouvoir ajouter le nouvel élément.

## Capacité (suite)

- Soit par exemple le programme suivant:

```
#include <vector>
...
vector< double > salaires;
salaires.push_back(102000.0);
salaires.push_back(45000.0);
salaires.push_back(78000.0);
salaires.push_back(25600.0);
salaires.push_back(53300.0);
```



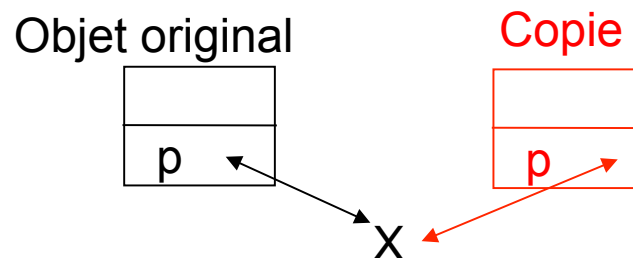
Encore une fois, comme le tableau est plein, on “double” sa taille pour pouvoir ajouter le nouvel élément.



# Passage de vecteur en paramètre

---

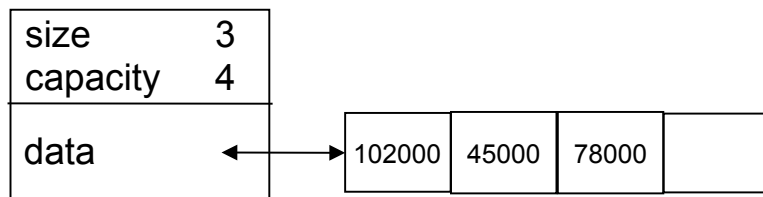
- Jusqu' à maintenant, nous avons toujours considéré que lorsqu' on copie un objet contenant un pointeur, c' est le pointeur qui est copié (“**shallow copy**”)
- Ainsi, on aura deux objets qui pointent au même endroit



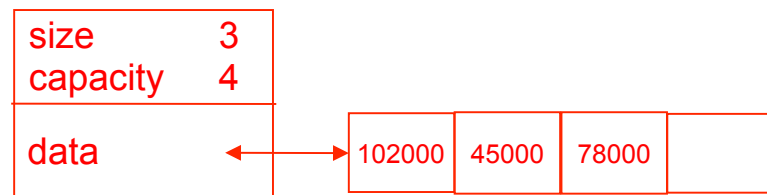
# Passage de vecteur en paramètre (suite)

- Dans le cas d'un vecteur, ce n'est pas ce qui se passe
- En effet, lorsqu'une copie est réalisée, ce n'est pas le pointeur qu'on copie, mais tout le tableau dynamique ("**deep copy**")
- Ainsi, toute modification à la copie n'a aucun impact sur le vecteur original

Vecteur original



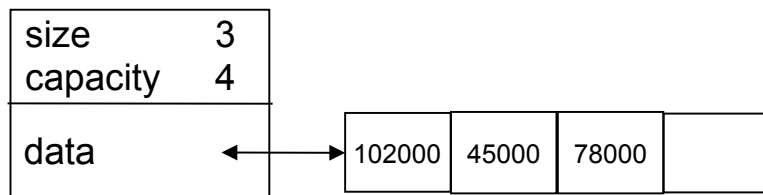
Copie



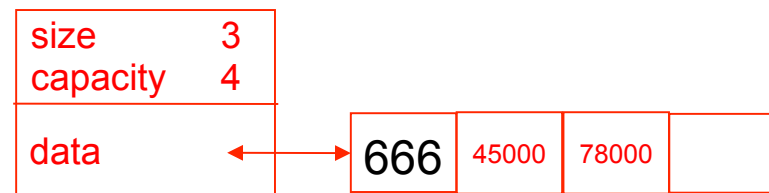
# Passage de vecteur en paramètre (suite)

- Dans le cas d'un vecteur, ce n'est pas ce qui se passe
- En effet, lorsqu'une copie est réalisée, ce n'est pas le pointeur qu'on copie, mais tout le tableau dynamique ("**deep copy**")
- Ainsi, toute modification à la copie n'a aucun impact sur le vecteur original

Vecteur original



Copie



## Passage de vecteur en paramètre (suite)

---

- Cela implique que lorsqu' on passe un vecteur par valeur à une fonction, une copie éventuellement coûteuse est réalisée
- Il est donc **généralement plus approprié de passer un vecteur par référence**
- Si on veut éviter qu' il soit modifié, on passe une référence constante:

```
f(const vector< int >& unVecteur)
{
    ...
}
```

## Retrait d'un élément au milieu

---

- Si l'ordre des éléments n'a pas d'importance, l'opération n'est pas trop coûteuse
- On copie d'abord le dernier élément à la position de l'élément retiré
- Puis on appelle `pop_back()` pour retirer le dernier élément

## Retrait d'un élément au milieu (suite)

---

- Par contre, si l'ordre des éléments doit être conservé, l'opération est coûteuse
- Il faut alors décaler d'une position tous les éléments qui suivent l'élément retiré
- Exemple:

```
void erase(vector< int >& v, int pos)
{
    unsigned int max = v.size() - 1;
    for (unsigned int i = pos; i < max; ++i)
        v[i] = v[i+1];
    v.pop_back();
}
```

## Insertion d'un élément au milieu (suite)

---

- On a le même problème pour l'insertion d'un élément au milieu:

```
void insert(vector<int>& v, unsigned int pos, int item)
{
    unsigned int last = v.size() - 1;
    v.push_back(v[last]); //espace pour élément additionnel
    for (unsigned int i = last; i > pos; --i)
        v[i] = v[i-1];
    v[pos] = item;
}
```



## Insertion et retrait au milieu

---

- Nous verrons plus tard que la bibliothèque des vecteurs contient les méthodes `insert()` et `erase()`.
- Mais rappelez-vous qu'il faut **éviter de faire appel à ces méthodes**, à cause de leur coût en temps d'exécution
- Si vous devez fréquemment les appeler, c'est que **l'utilisation d'un vecteur n'est pas appropriée**



## Remarques sur les vecteurs d'objets

---

- Si on a un vecteur d'objets, l'appel à `push_back()` fera une copie de l'objet dans le vecteur
- À la longue, cela peut devenir coûteux
- C'est pourquoi on a souvent tendance, en C++, à **utiliser des vecteurs de pointeurs sur des objets**

# Remarques sur les vecteurs d'objets (suite)

---

- Exemple:

```
...  
vector< Point* > listeDePoints;  
Point* p1 = new Point(3,2);  
Point* p2 = new Point(1,2);  
Point* p3 = new Point(4,5);  
...  
listeDePoints.push_back(p1);  
listeDePoints.push_back(p2);  
listeDePoints.push_back(p3);  
...
```

On obtient un vecteur de trois pointeurs sur des points.