Collections

A. Résumé de l'épisode précédent

Motivation

Manipulation de collections de données

Exemple (Fil conducteur)

Implantation d'un annuaire

A. Résumé de l'épisode précédent

Motivation

Manipulation de collections de données

Exemple (Fil conducteur)

Implantation d'un annuaire

Tableau

- Un *tableau* est une valeur *composite* formée de plusieurs valeurs du même type
- Construction :
  1. Déclaration
  2. Allocation
  3. Initialisation

A. Résumé de l'épisode précédent

Motivation

Manipulation de collections de données

Exemple (Fil conducteur)

Implantation d'un annuaire

Tableau

- Un *tableau* est une valeur *composite* formée de plusieurs valeurs du même type
- Construction :
  1. Déclaration
  2. Allocation
  3. Initialisation

Fonctionnement ? Sémantique ?

B. Tableaux et allocation mémoire

Modèle de mémoire

L'espace mémoire d'un programme est partagé en deux zones :

- La **pile** : variables locales des fonctions
- Le **tas** : le reste

Exemple

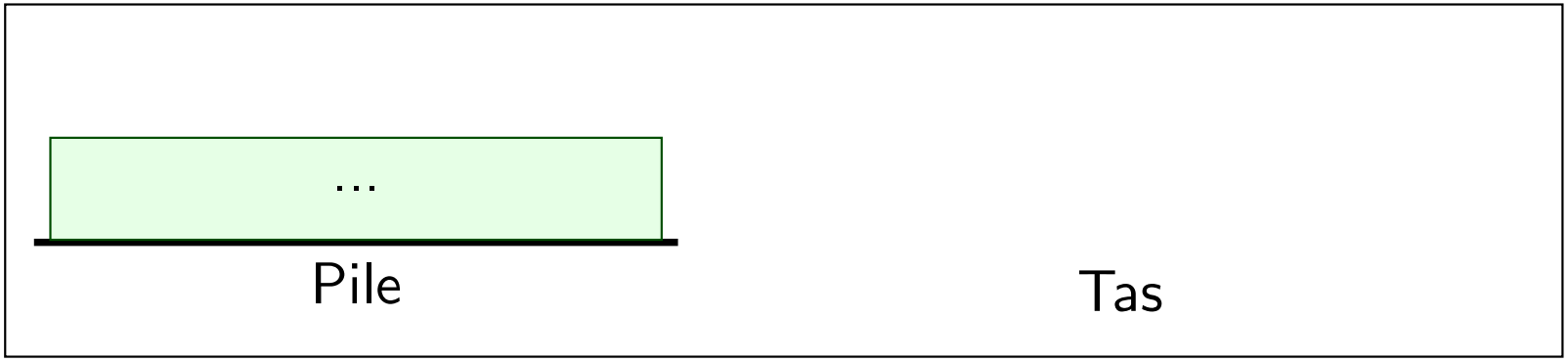
tableaux.cpp

```
vector<int> t;
t = vector<int>(8);
t[0] = 1;
t[1] = 4;
t[2] = 1;
t[3] = 5;
t[4] = 9;
t[5] = 2;
```

Exemple de construction d'un tableau

Étapes de la construction en mémoire

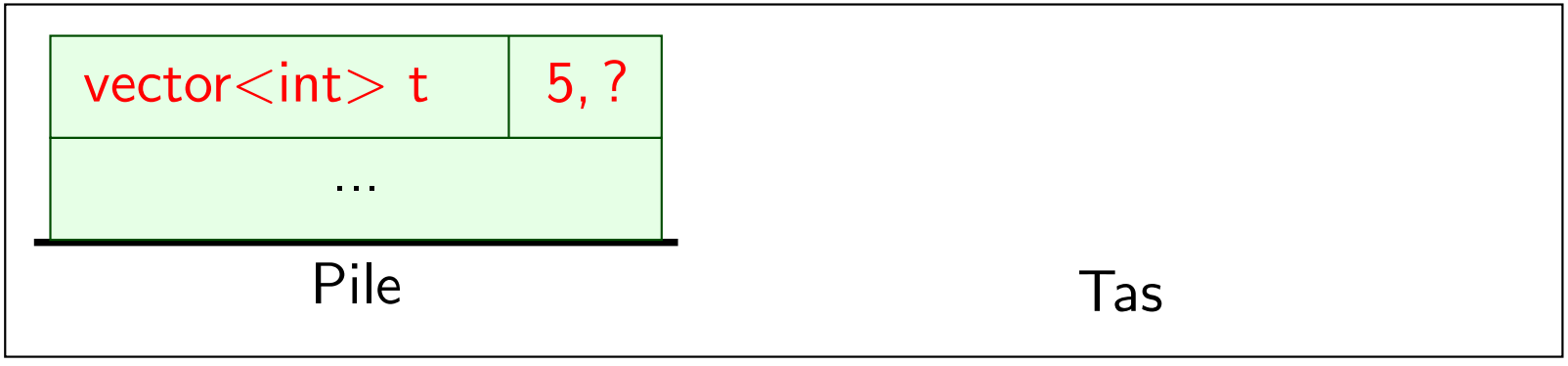
1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Exemple de construction d'un tableau

Étapes de la construction en mémoire

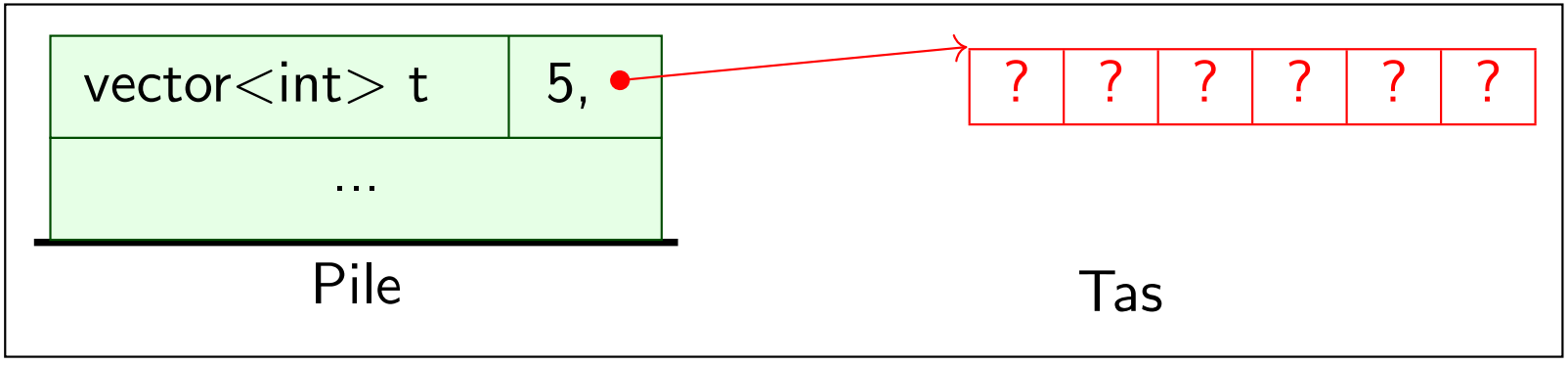
1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Exemple de construction d'un tableau

Étapes de la construction en mémoire

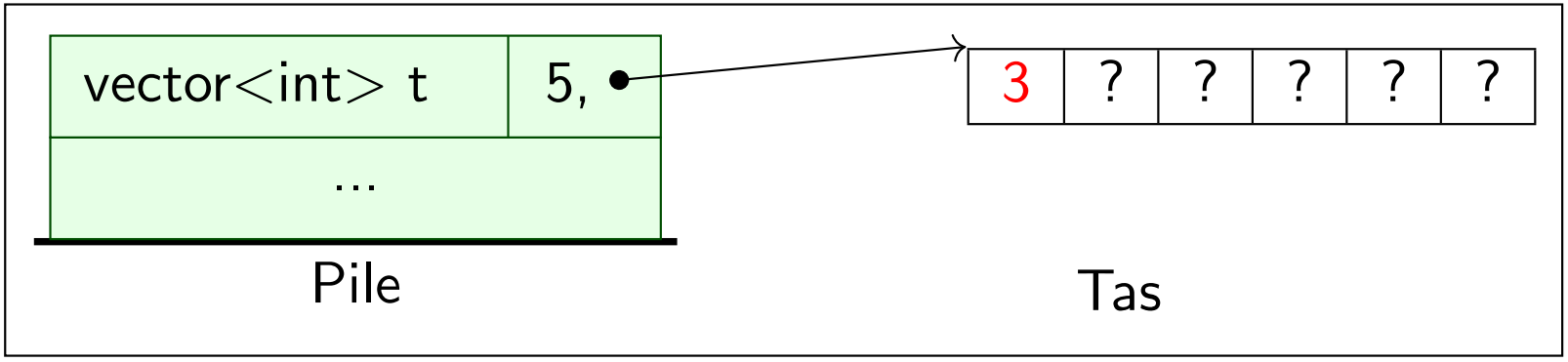
1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Exemple de construction d'un tableau

Étapes de la construction en mémoire

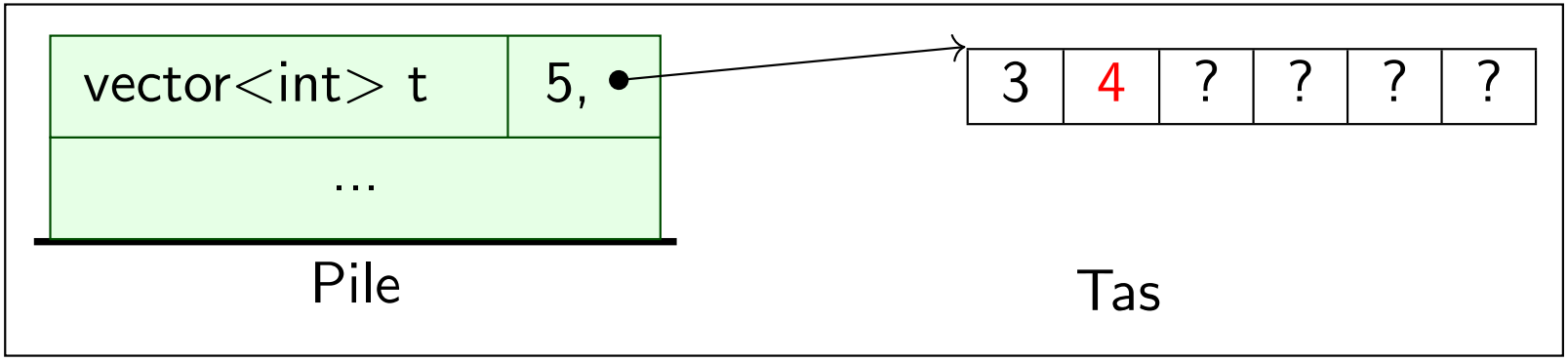
1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Exemple de construction d'un tableau

Étapes de la construction en mémoire

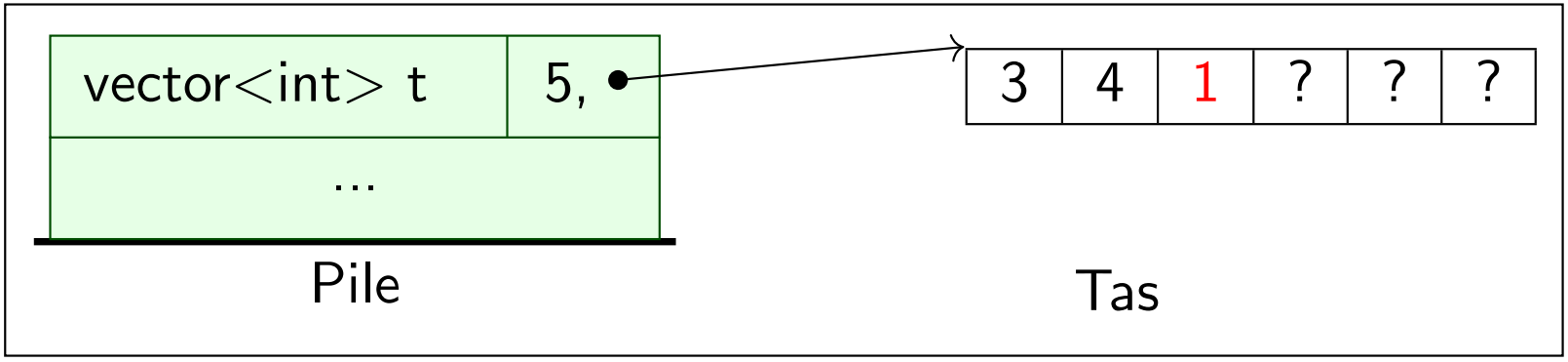
1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Exemple de construction d'un tableau

Étapes de la construction en mémoire

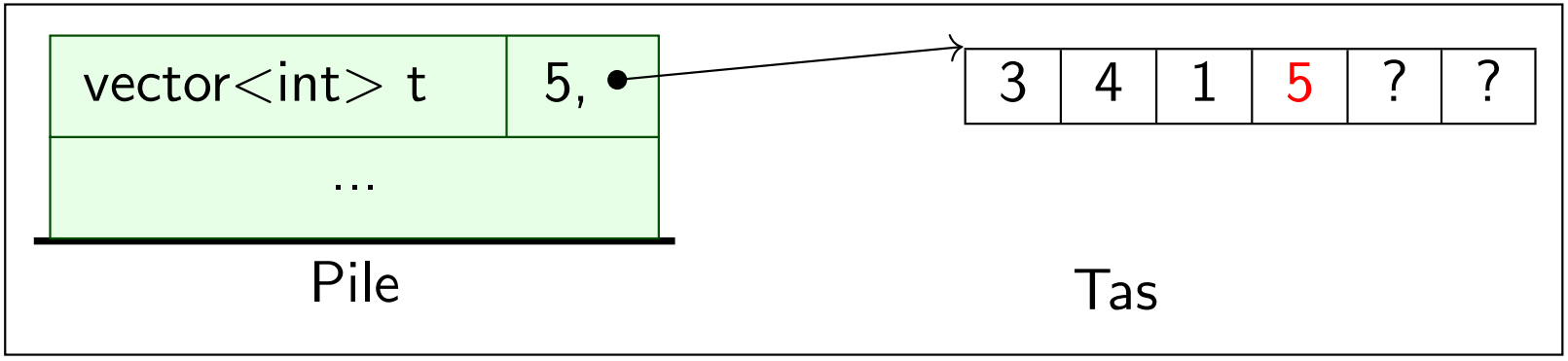
1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Exemple de construction d'un tableau

Étapes de la construction en mémoire

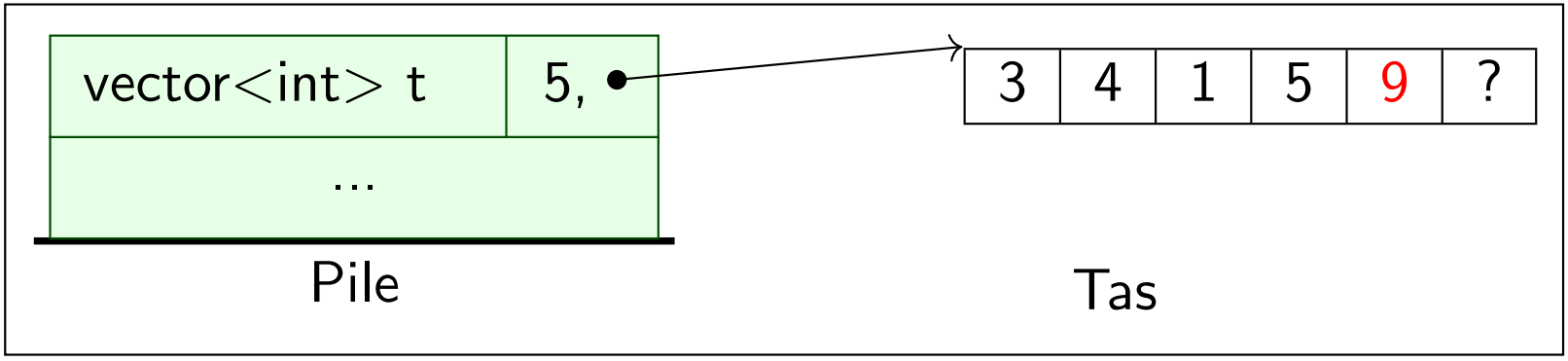
1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Exemple de construction d'un tableau

Étapes de la construction en mémoire

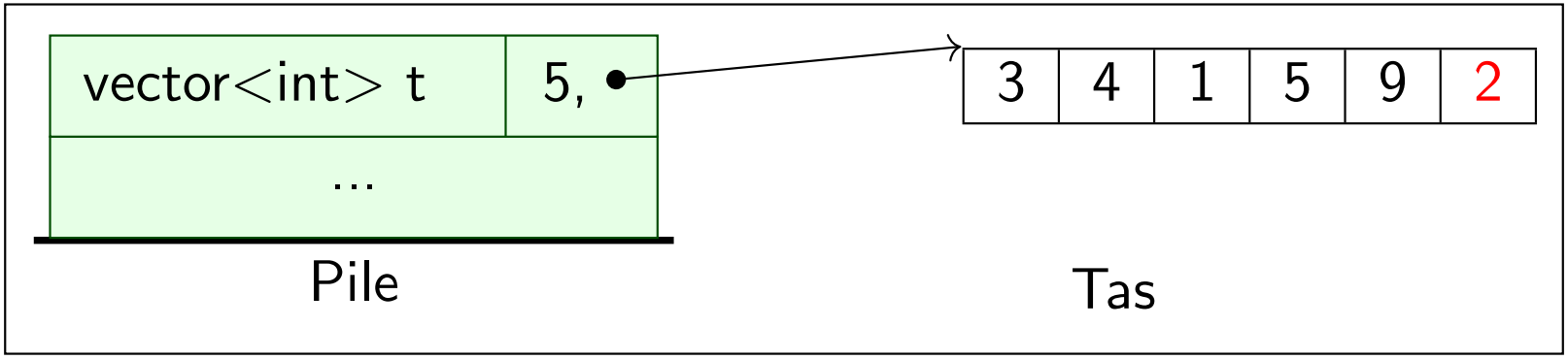
1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Exemple de construction d'un tableau

Étapes de la construction en mémoire

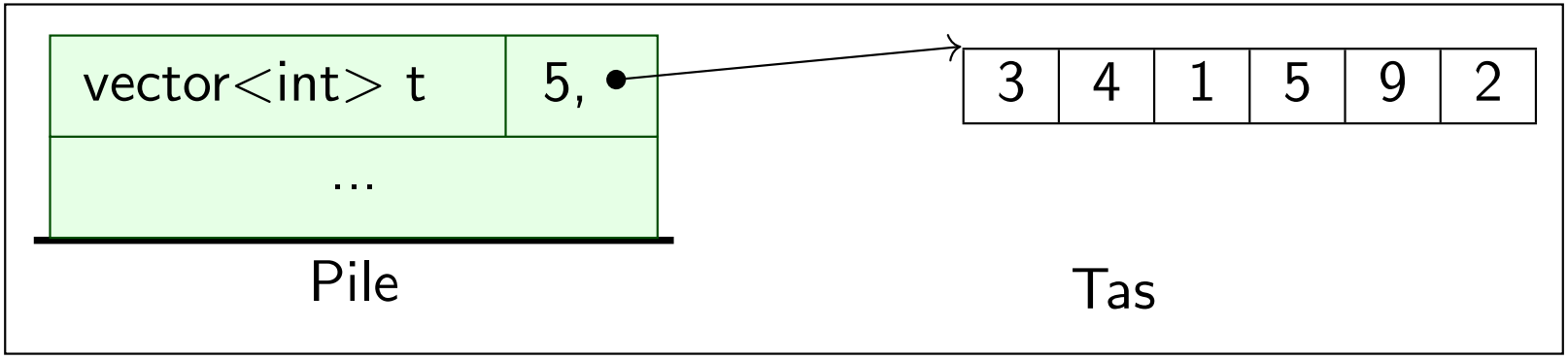
1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Exemple de construction d'un tableau

Étapes de la construction en mémoire

1. Déclaration du tableau
2. Allocation du tableau
3. Initialisation



Sémantique (Allocation d'un tableau)

tableaux.cpp

```
t = vector<int>(8);
```

1. Une suite contiguë de cases est allouée sur le tas
2. Une référence vers la première de ces cases est stockée dans `t`

Sémantique (Lecture et écriture dans un tableau)

```
t[i]
```

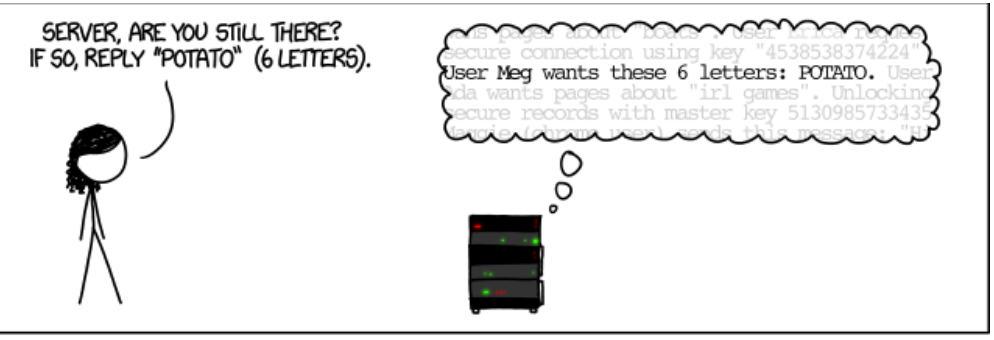
– Donne la i-ème case du tableau
– Obtenue en suivant la référence et se décalant de i cases
– Rappel : **pas de vérifications ! ! !**

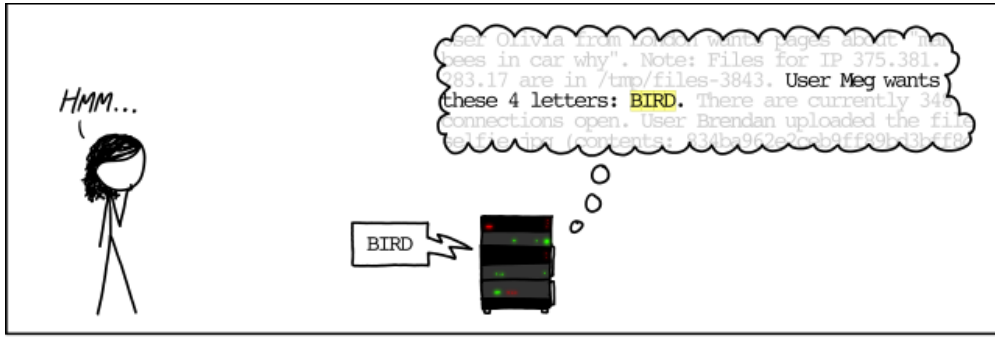
Exemple de piratage par débordement ♣

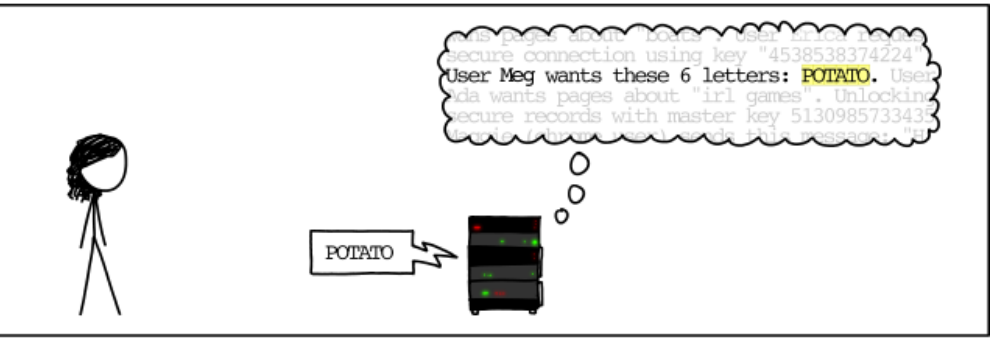
login.cpp

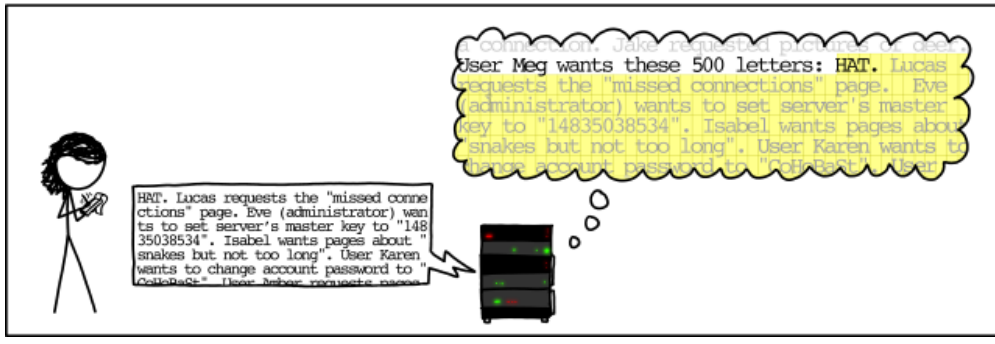
```
int main() {
    char motDePasseSecret[] = "s3*iA3";
    char motDePasse      [] = "XXXXXX";
    cout << motDePasseSecret - motDePasse << endl;
    do {
        cout << "Entrez le mot de passe: " << endl;
        cin >> motDePasse;
    } while ( string(motDePasse)  != string(motDePasseSecret) );

    cout << "Connection réussie. Bienvenue chef !" << endl;
    return 0;
}
```

Heartblead expliqué (http://xkcd.com/1354/)









Tableaux et allocation mémoire

À retenir

- *Une valeur de type tableau ne contient pas directement les cases du tableau, mais la localisation en mémoire de celles-ci (référence) et la taille du tableau.*

Tableaux et allocation mémoire

À retenir

- *Une valeur de type tableau ne contient pas directement les cases du tableau, mais la localisation en mémoire de celles-ci (référence) et la taille du tableau.*

- *Une variable de type tableau se construit en trois étapes :*
  1. *Déclaration*
  2. *Allocation*
     Sans cela : **faute de segmentation** *(au mieux !)*
  3. *Initialisation*
     Sans cela : même problème qu'avec les variables usuelles

Tableaux et allocation mémoire

À retenir

- *Une valeur de type tableau ne contient pas directement les cases du tableau, mais la localisation en mémoire de celles-ci (référence) et la taille du tableau.*

- *Une variable de type tableau se construit en trois étapes :*
  1. *Déclaration*
  2. *Allocation*
     Sans cela : **faute de segmentation** *(au mieux !)*
  3. *Initialisation*
     Sans cela : même problème qu'avec les variables usuelles

- *Lors de l'accès à une case* `i` *d'un tableau* `t`*, on vérifie toujours les bornes :* `0 <= i` *et* `i < t.size()`
  Sans cela : **faute de segmentation** *(au mieux !)*

Affectation de tableaux (sémantique)

Exemple

Qu'affiche le programme suivant ?

tableaux-copie.cpp

```
vector<int> t = { 1, 4, 1, 5, 9, 2, 6, 5, 3, 5 };
cout << "t[0]: " << t[0] << endl;
vector<int> t2;
t2 = t;                 // Affectation
t2[0] = 0;
cout << "t[0]: " << t[0] << ", t2[0]: " << t2[0] << endl;
```

Affectation de tableaux (sémantique)

Exemple

Qu'affiche le programme suivant ?

tableaux-copie.cpp

```
vector<int> t = { 1, 4, 1, 5, 9, 2, 6, 5, 3, 5 };
cout << "t[0]: " << t[0] << endl;
vector<int> t2;
t2 = t;                 // Affectation
t2[0] = 0;
cout << "t[0]: " << t[0] << ", t2[0]: " << t2[0] << endl;
```

À retenir

- *En C++, lors d'une affectation, un `vector` est* **copié** *!*
- *On dit que `vector` a une sémantique de copie*
- *Différent de Java, Python, ou des `array` en C !*

Affectation de tableaux (sémantique ♣)

Exemple

tableau-fonction-valeur.cpp

```
void modifie(vector<int> tableau) {
    tableau[0] = 42;
}

int main() {
    vector<int> tableau = { 1, 2, 3, 4 };
    modifie(tableau);
    cout << tableau[0] << endl;
}
```

Affectation de tableaux (sémantique ♣)

Exemple

tableau-fonction-valeur.cpp

```
void modifie(vector<int> tableau) {
    tableau[0] = 42;
}

int main() {
    vector<int> tableau = { 1, 2, 3, 4 };
    modifie(tableau);
    cout << tableau[0] << endl;
}
```

Fonctions et tableaux

- Affectation des paramètres $\Longrightarrow$ copie
- Donc les `vector` sont passés *par valeur* aux fonctions

Affectation de tableaux (sémantique ♣)

Exemple

tableau-fonction-valeur.cpp

```
void modifie(vector<int> tableau) {
    tableau[0] = 42;
}

int main() {
    vector<int> tableau = { 1, 2, 3, 4 };
    modifie(tableau);
    cout << tableau[0] << endl;
}
```

Fonctions et tableaux

- Affectation des paramètres $\Longrightarrow$ copie
- Donc les `vector` sont passés *par valeur* aux fonctions
- Mais la fonction peut renvoyer le tableau modifié !

Affectation de tableaux (sémantique ♣)

Exemple

tableaux.cpp

```
t = vector<int>(8);
```

Affectation de tableaux (sémantique ♣)

Exemple

tableaux.cpp

```
t = vector<int>(8);
```

Sémantique ≠ fonctionnement interne exact

– Pour préserver les performances, `vector` suit le patron de conception de *copie-en-écriture* : la copie n'a lieu que lorsqu'elle est nécessaire

C. Tableaux, collections et vecteurs en C++

– La bibliothèque standard C++ fournit de nombreuses autres structures de données pour représenter des *collections* :
`array`, `list`, `queue`, `stack`, `set`, `multiset`, ...

C. Tableaux, collections et vecteurs en C++

- La bibliothèque standard C++ fournit de nombreuses autres structures de données pour représenter des *collections* :
  `array`, `list`, `queue`, `stack`, `set`, `multiset`, ...
- Chacune a ses spécificités en terme de **sémantique**, d'**opérations disponibles** et de **performances**

C. Tableaux, collections et vecteurs en C++

- La bibliothèque standard C++ fournit de nombreuses autres structures de données pour représenter des *collections* :
  `array`, `list`, `queue`, `stack`, `set`, `multiset`, ...
- Chacune a ses spécificités en terme de **sémantique**, d'**opérations disponibles** et de **performances**
- On se contentera dans ce cours de `vector`
  On en verra plus en S2 !

C. Tableaux, collections et vecteurs en C++

- La bibliothèque standard C++ fournit de nombreuses autres structures de données pour représenter des *collections* :
  `array`, `list`, `queue`, `stack`, `set`, `multiset`, ...
- Chacune a ses spécificités en terme de **sémantique**, d'**opérations disponibles** et de **performances**
- On se contentera dans ce cours de `vector`
  On en verra plus en S2 !

- Les chaînes de caractères (`string`) se comportent en gros comme des tableaux de caractères

C. Tableaux, collections et vecteurs en C++

- La bibliothèque standard C++ fournit de nombreuses autres structures de données pour représenter des *collections* :
  `array`, `list`, `queue`, `stack`, `set`, `multiset`, ...
- Chacune a ses spécificités en terme de **sémantique**, d'**opérations disponibles** et de **performances**
- On se contentera dans ce cours de `vector`
  On en verra plus en S2 !

- Les chaînes de caractères (`string`) se comportent en gros comme des tableaux de caractères
- Les `vector` de C++ ne sont pas des vecteurs au sens mathématique : pas d'opération d'addition, ...

La boucle *for each* (C++ 2011) ♣

Exemple

tableau-foreach.cpp

```
vector<int> tableau = { 3, 1, 7, 4, 6, 2, 5 };

for ( int i=0; i < tableau.size(); i++ ) { // Boucle for
    cout << tableau[i] << " ";
}
cout << endl;

for ( int valeur: tableau ) { // Boucle for each
    cout << valeur << " ";
}
cout << endl;
```

La boucle *for each* (C++ 2011) ♣

Exemple

tableau-foreach.cpp

```
vector<int> tableau = { 3, 1, 7, 4, 6, 2, 5 };

for ( int i=0; i < tableau.size(); i++ ) { // Boucle for
    cout << tableau[i] << " ";
}
cout << endl;

for ( int valeur: tableau ) { // Boucle for each
    cout << valeur << " ";
}
cout << endl;
```

Avantages

- Pas de risque d'erreur de manipulation d'indice !
- Fonctionne avec n'importe quelle collection !

D. Tableaux à deux dimensions

Motivation

Jeu de morpion

D. Tableaux à deux dimensions

Motivation

Jeu de morpion

Remarque

- On représente un tableau à deux dimensions par un tableau de tableaux : `vector<vector<int>>`
- $t_{i,j}$: `t[i][j]`

D. Tableaux à deux dimensions

Motivation

Jeu de morpion

Remarque

- On représente un tableau à deux dimensions par un tableau de tableaux : `vector<vector<int>>`
- $t_{i,j}$: `t[i][j]`

À retenir

Un tableau à deux dimension se construit en quatre étapes :

1. *Déclaration du tableau*
2. *Allocation du tableau*
3. *Allocation des sous-tableaux*
4. *Initialisation*

Tableaux à deux dimensions : exemple (grille de morpion)

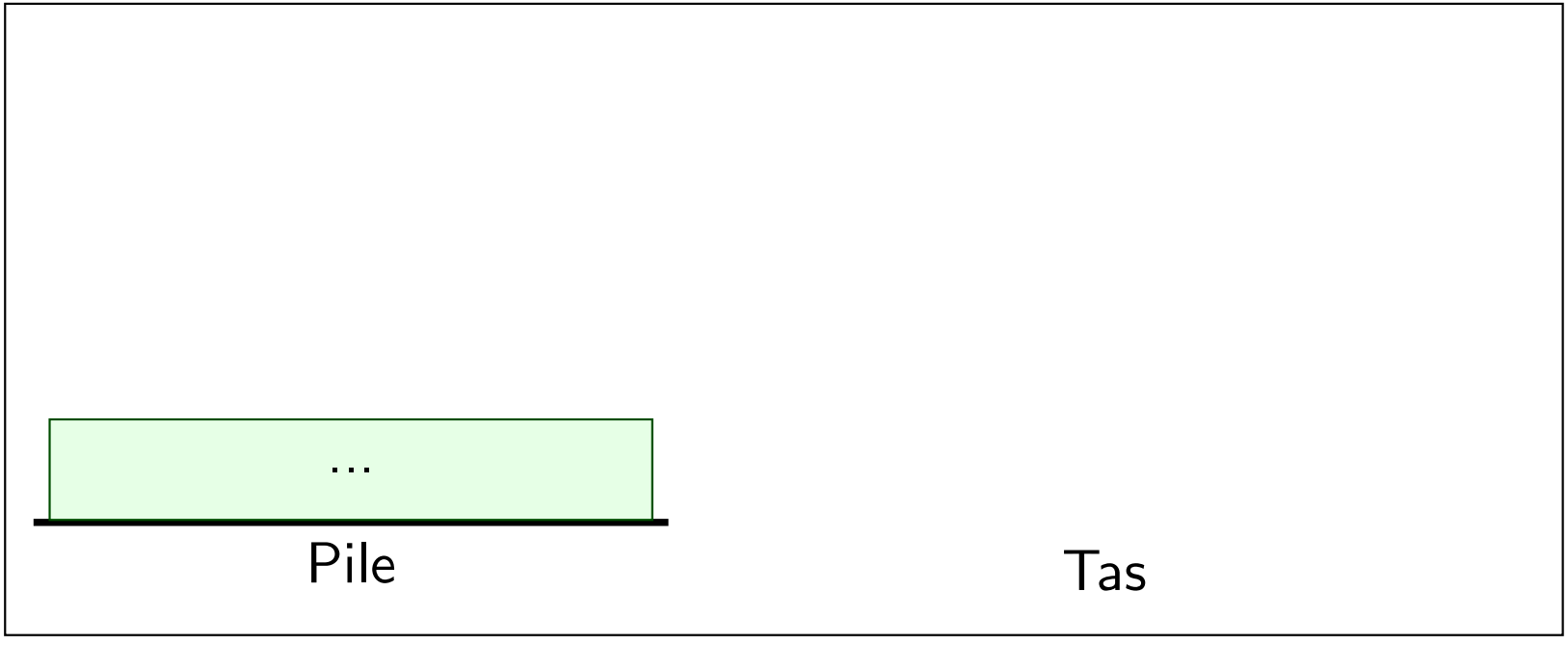
tableaux2D.cpp

```
// Déclaration
vector<vector<int>> t;

// Allocation
t = vector<vector<int>>(3);

// Allocation des sous-tableaux
for ( int i = 0; i < t.size(); i++ )
    t[i] = vector<int>(3);

// Initialization
t[0][0] = 1; t[0][1] = 2; t[0][2] = 0;
t[1][0] = 1; t[1][1] = 1; t[1][2] = 1;
t[2][0] = 0; t[2][1] = 2; t[2][2] = 2;
```

Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

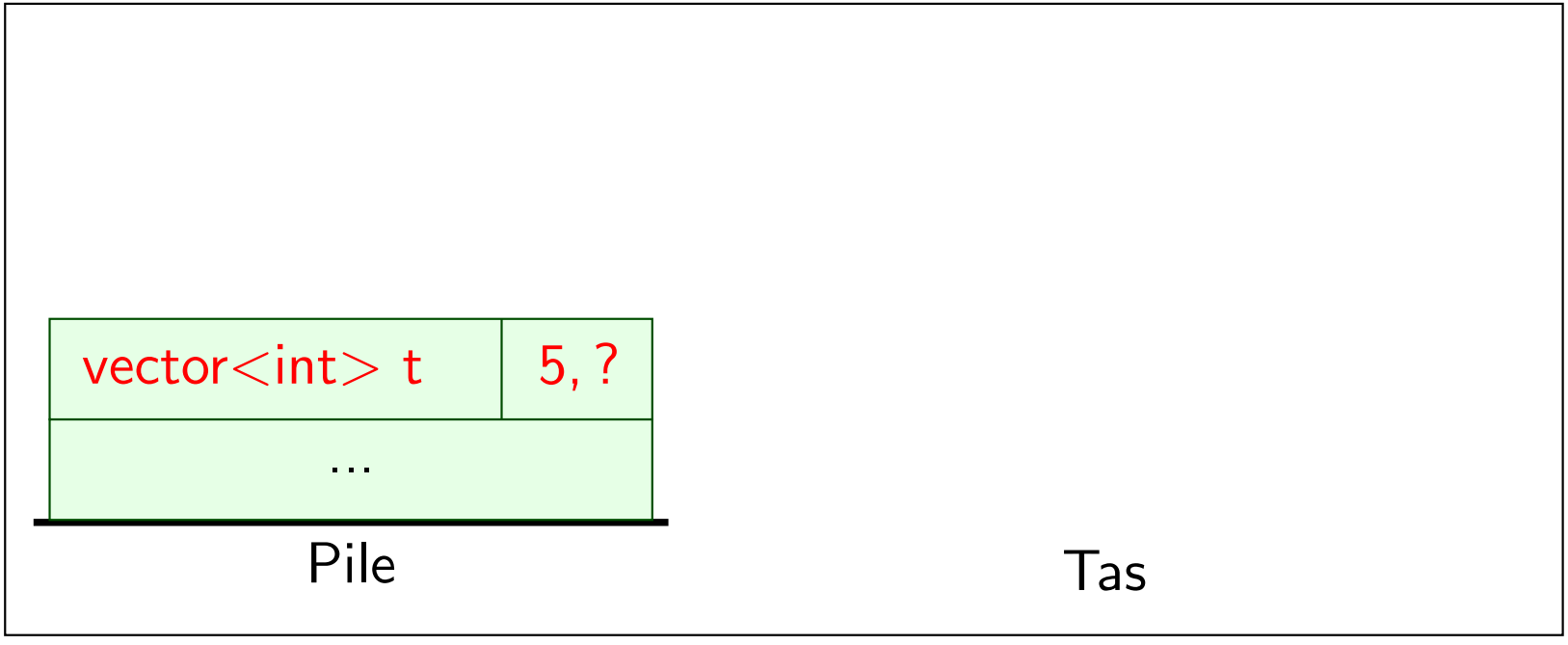
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

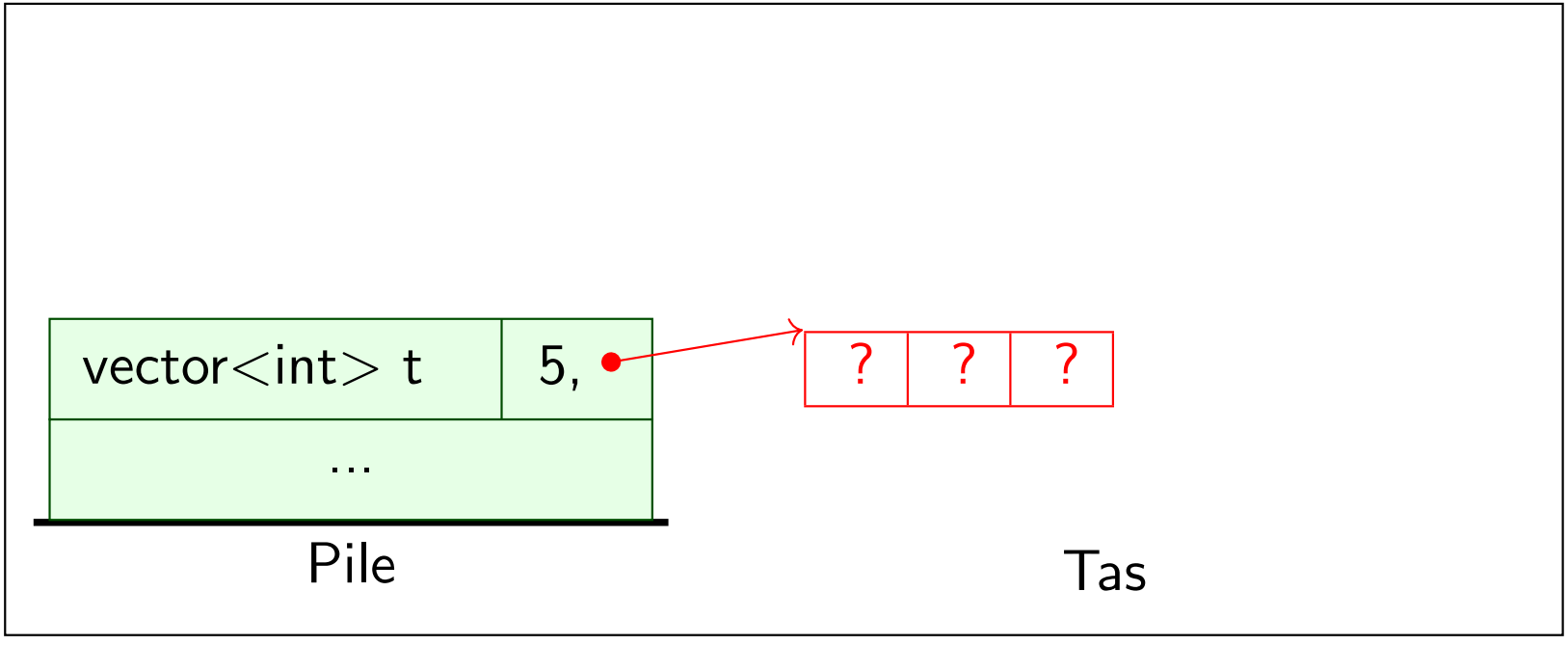
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

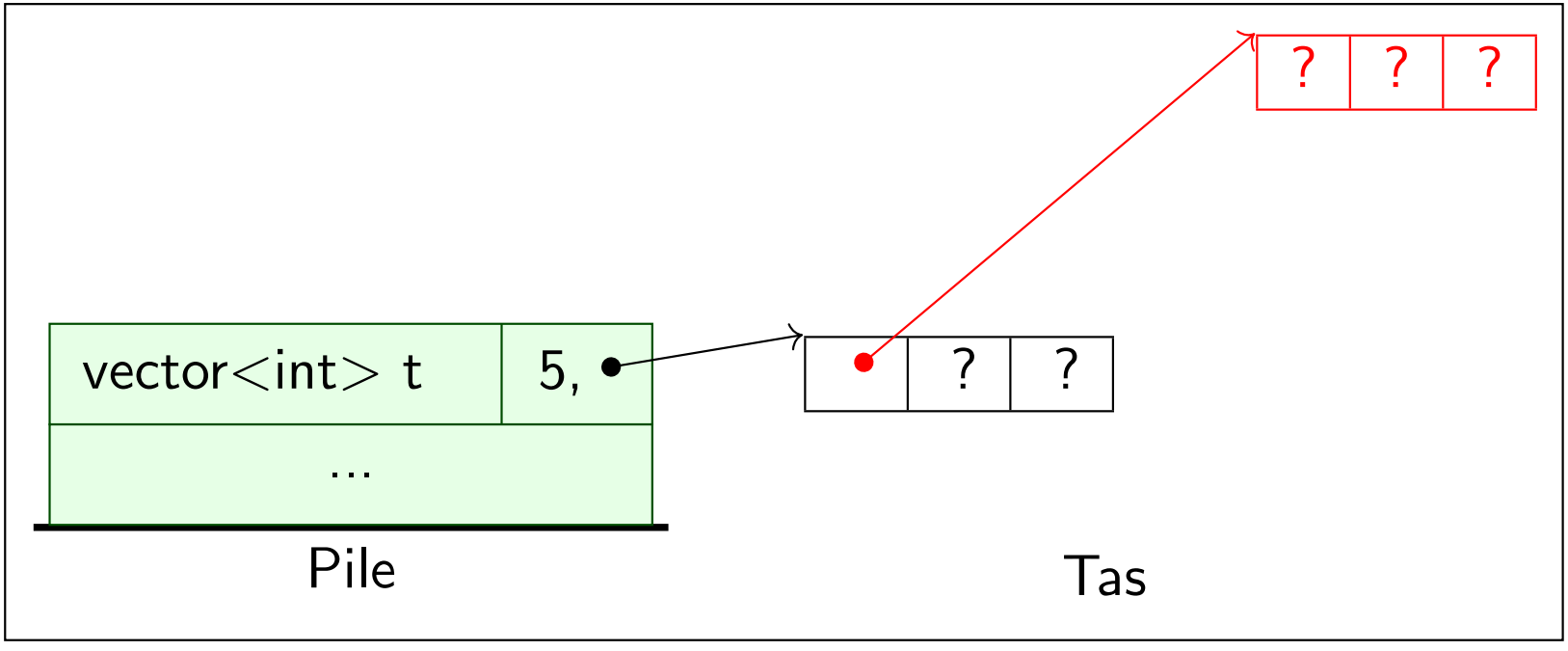
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

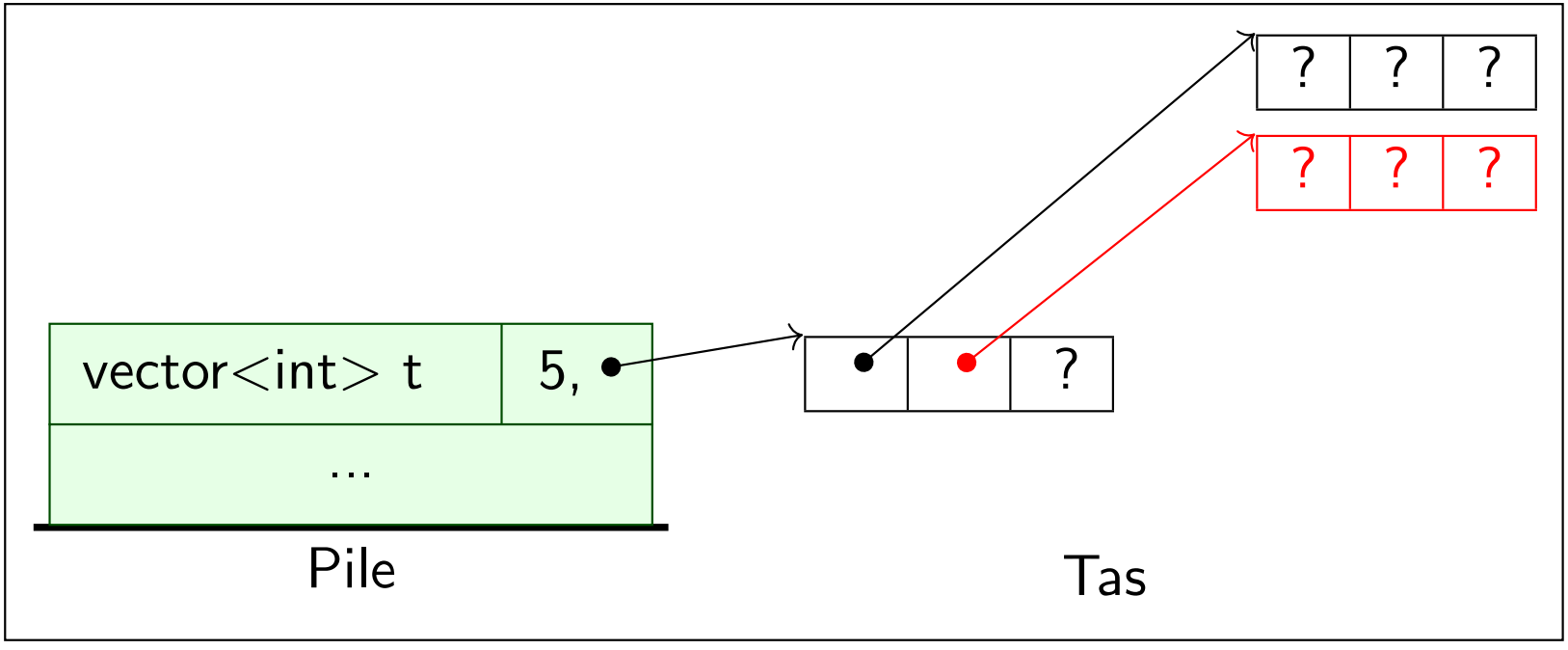
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

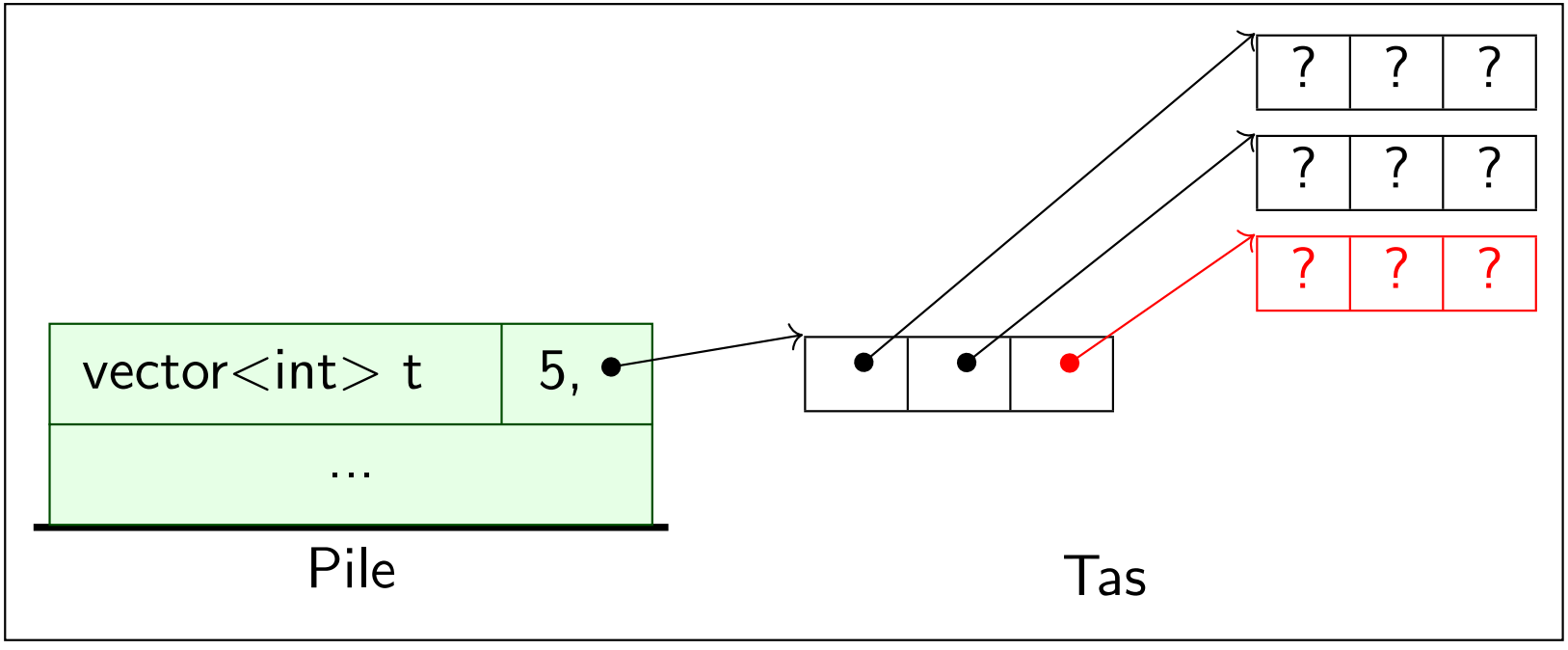
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

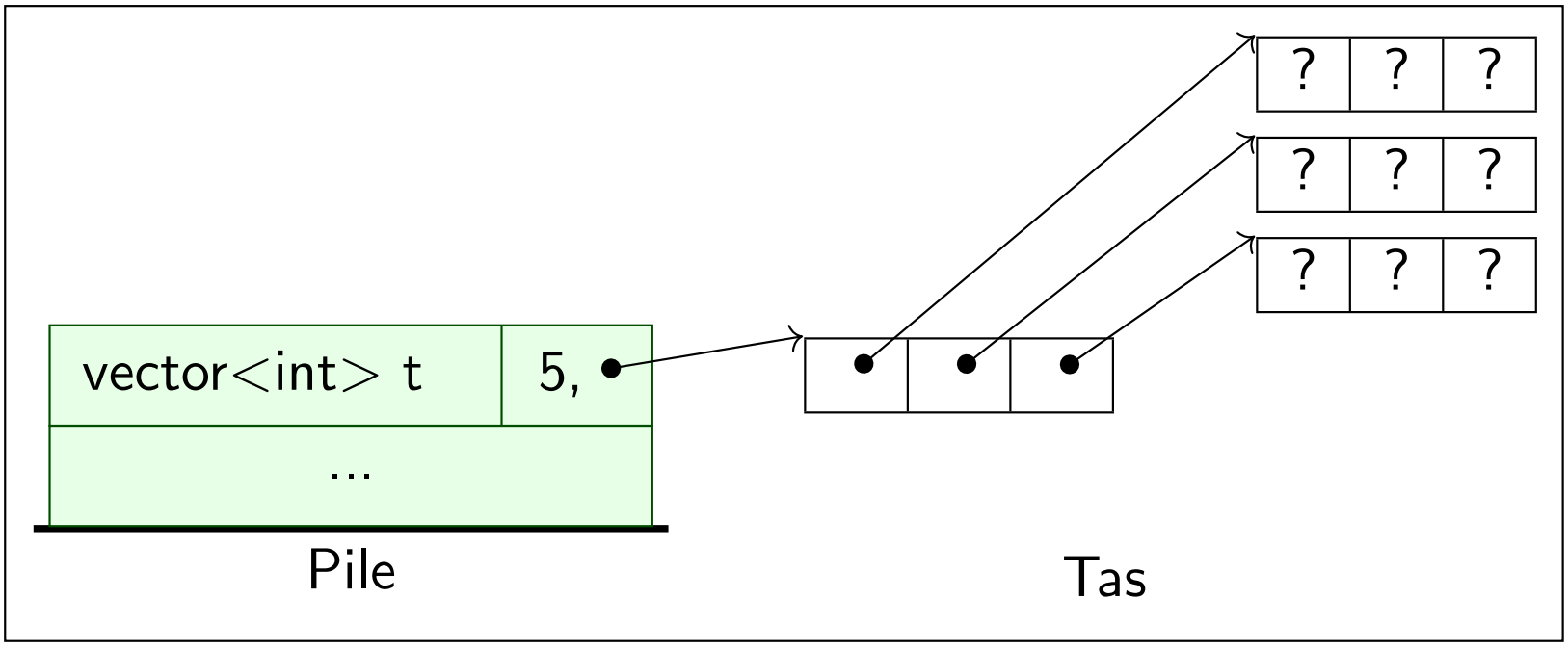
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

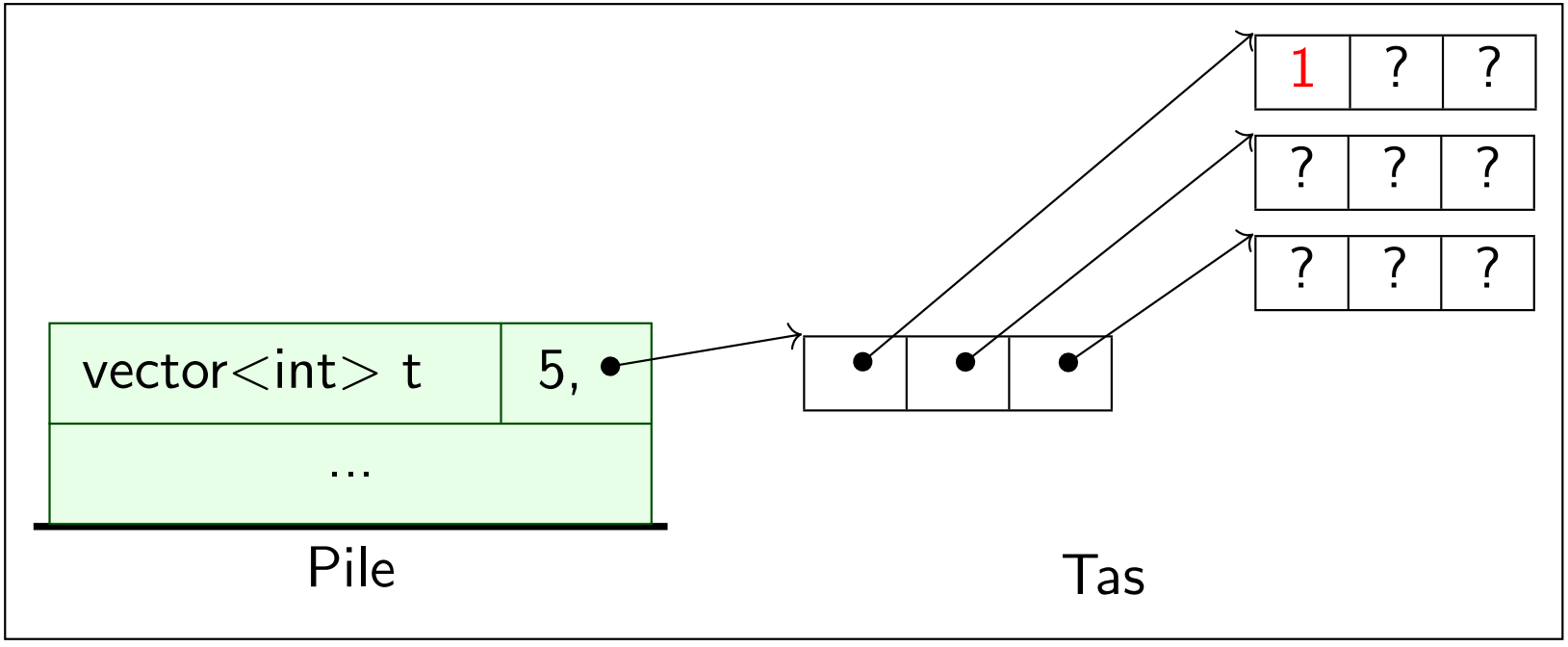
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

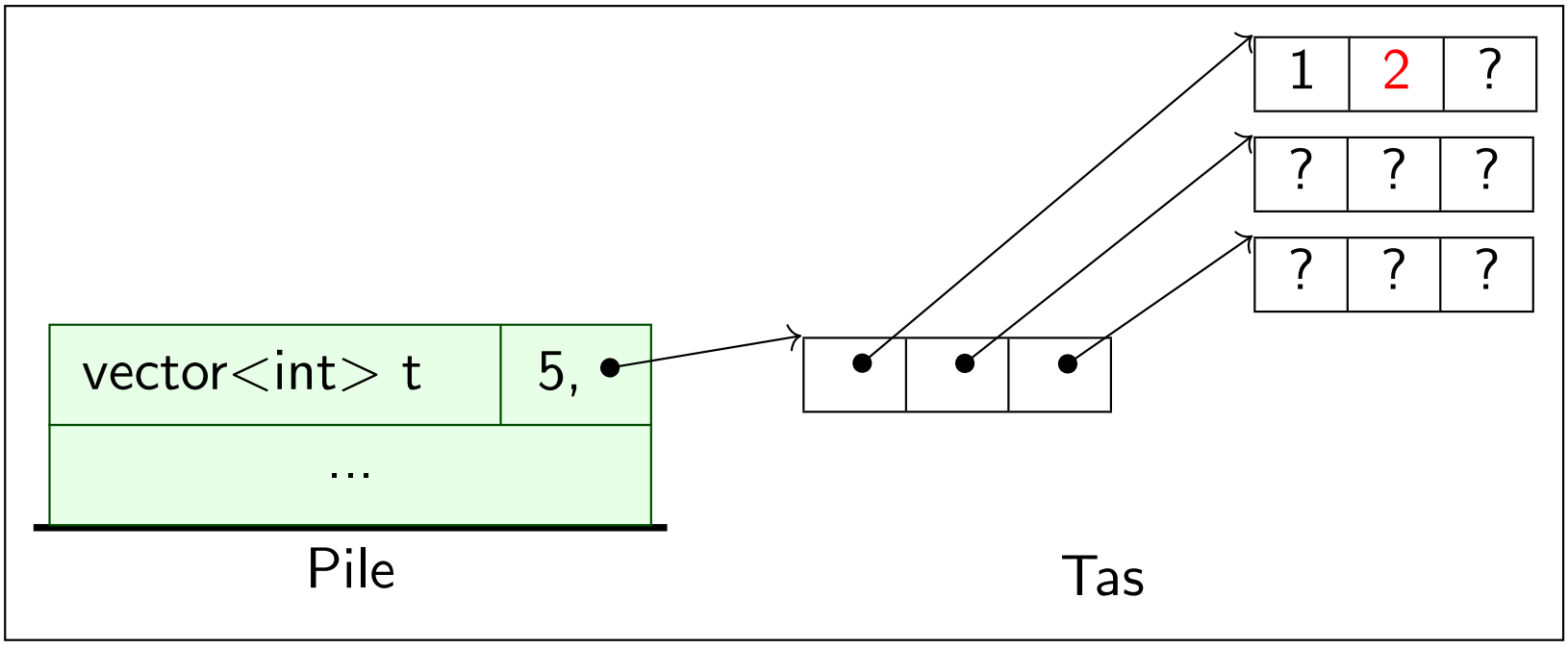
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

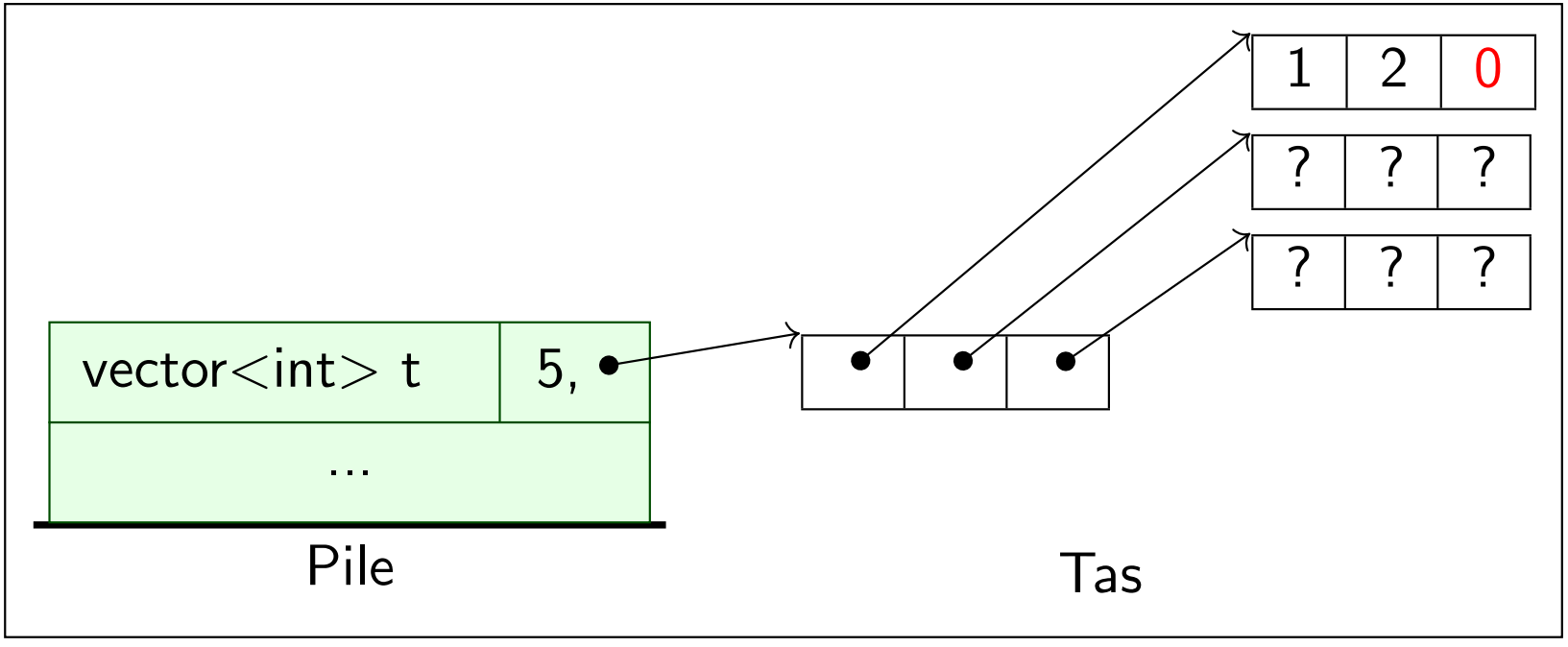
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

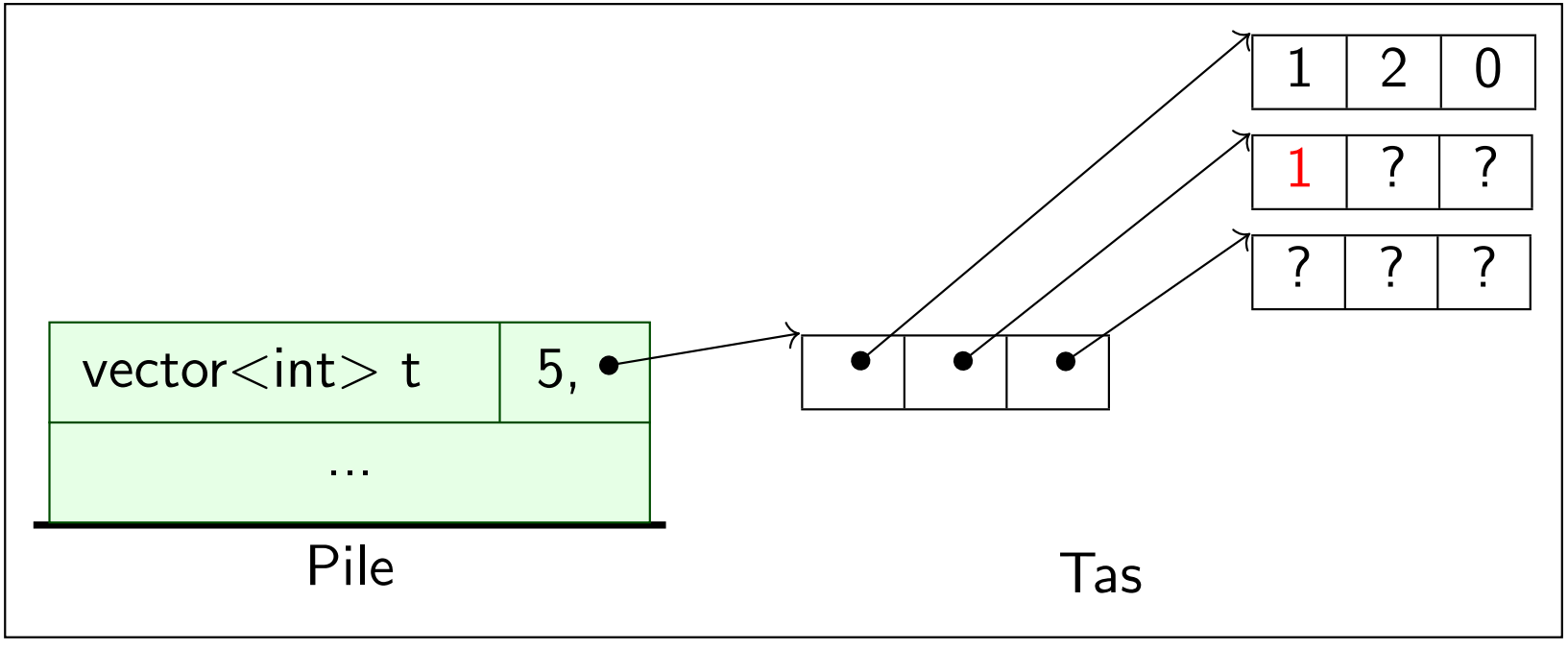
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

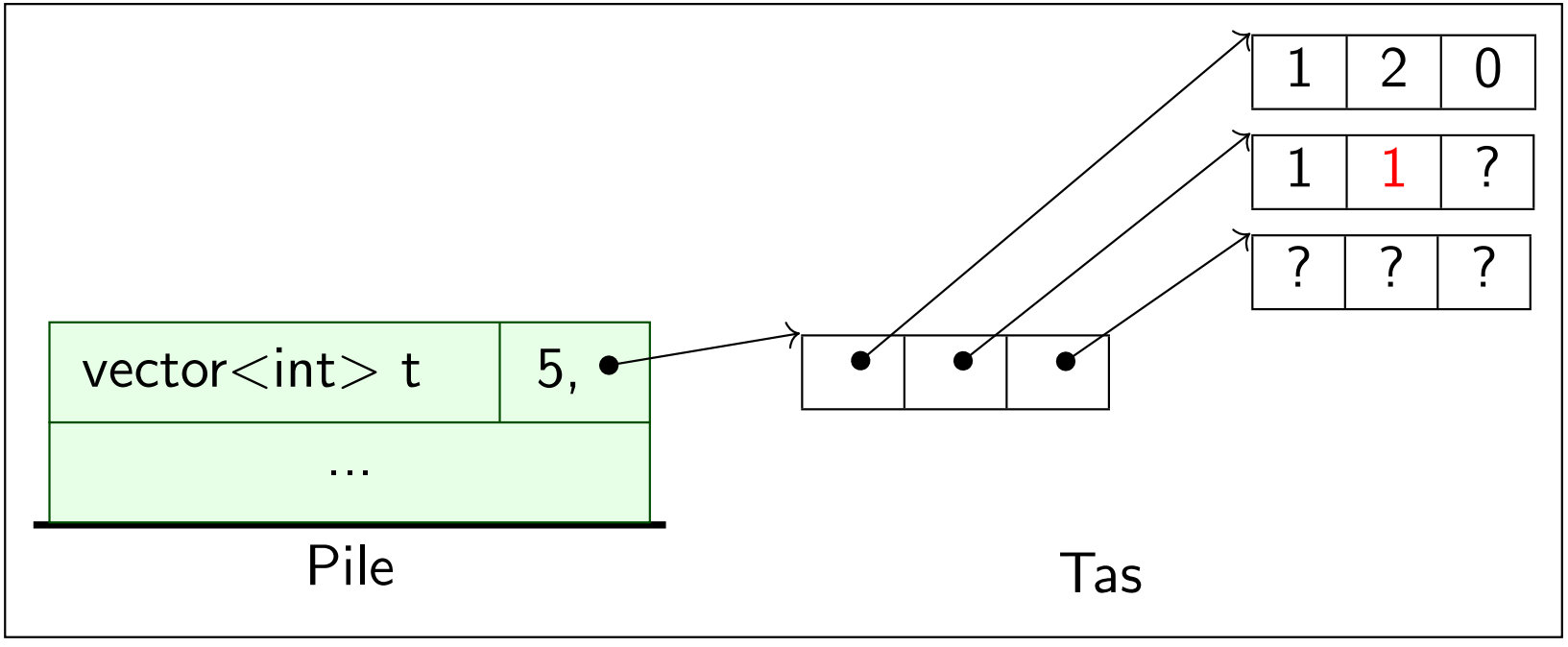
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

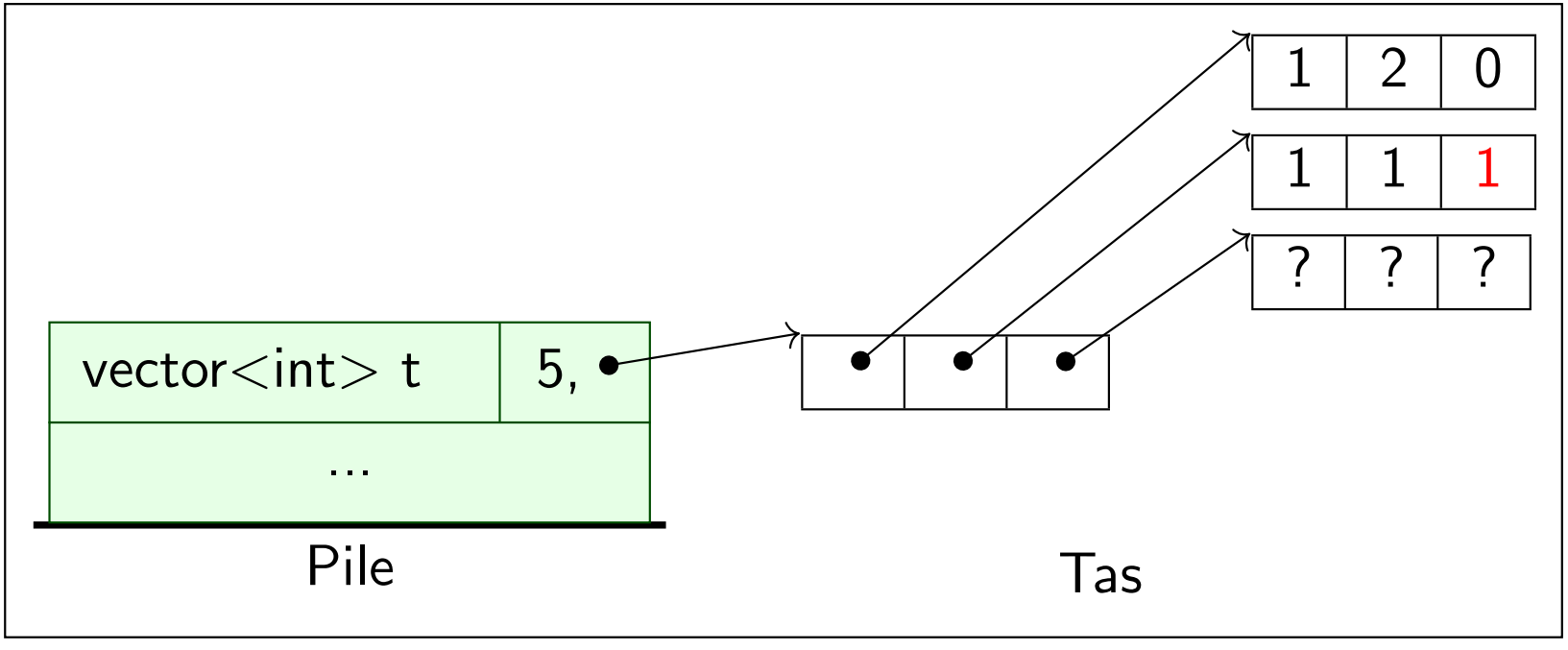
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

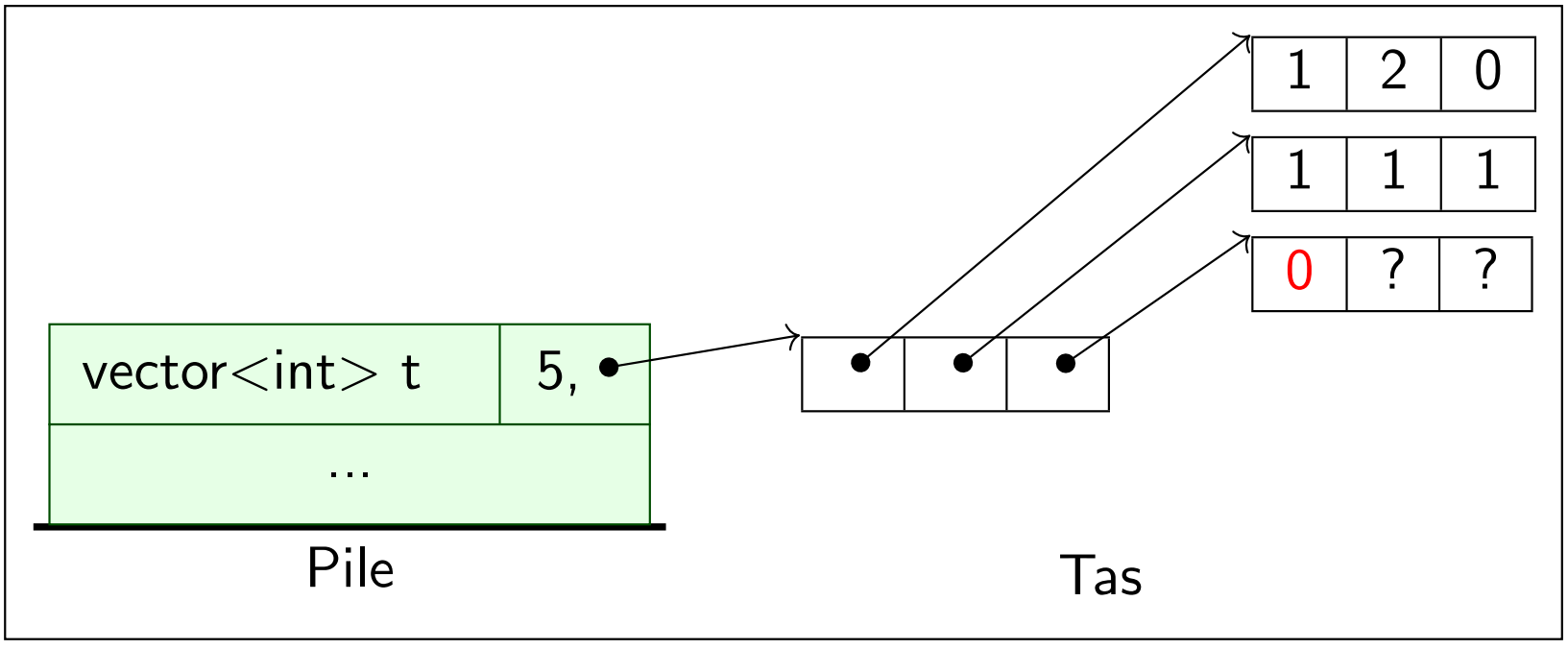
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

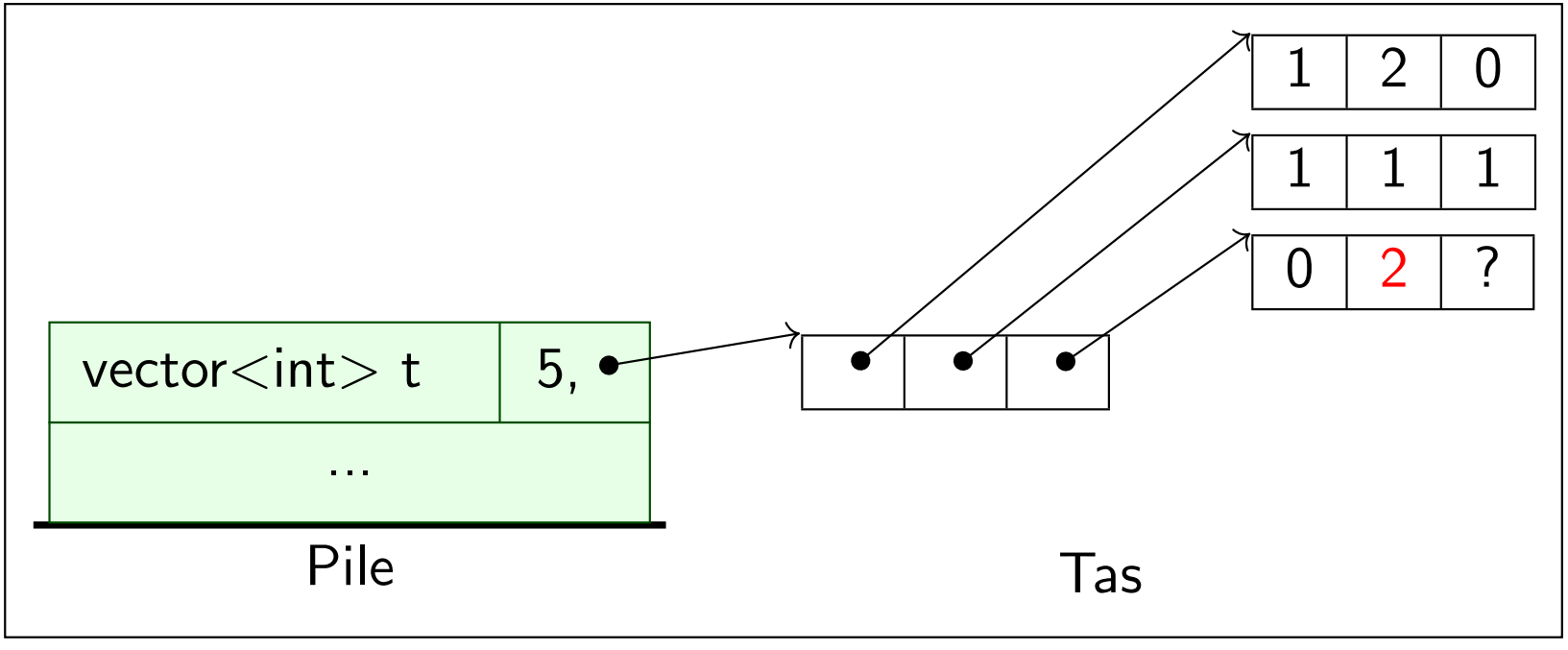
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

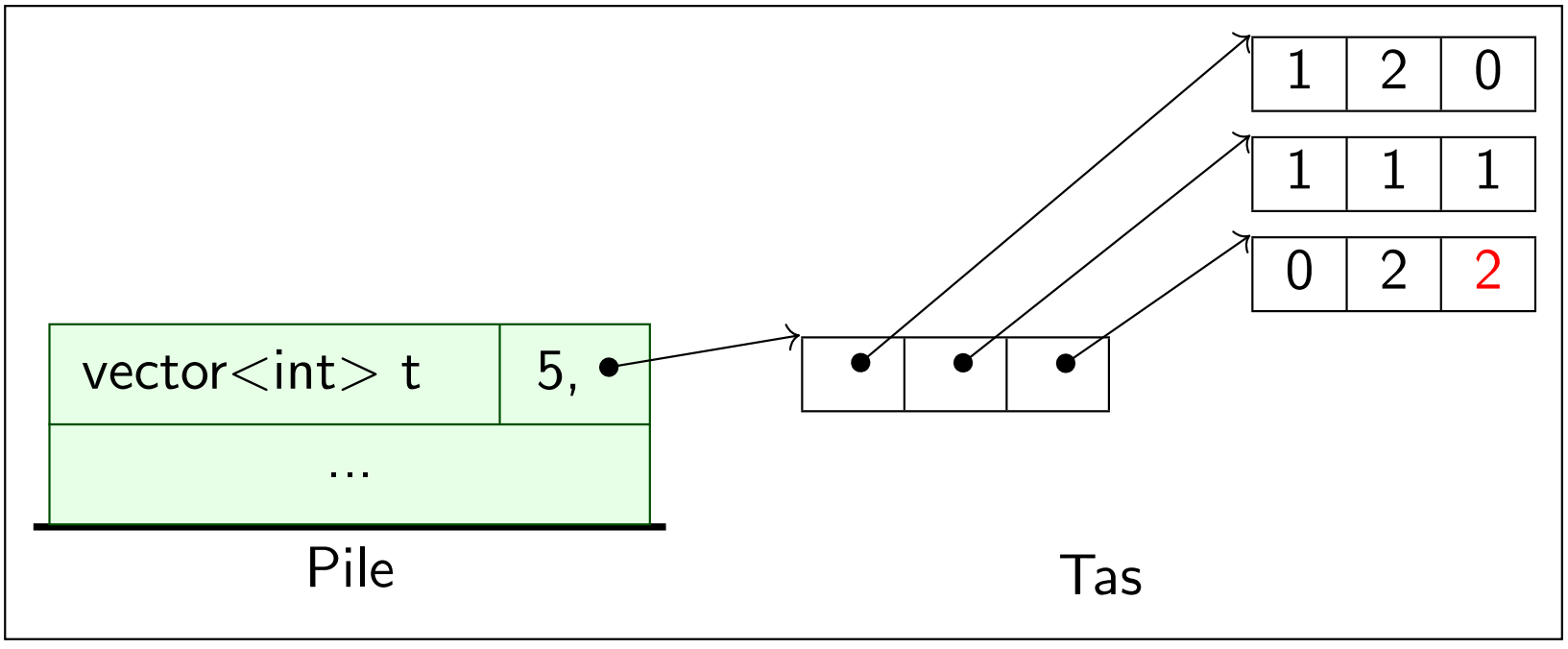
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

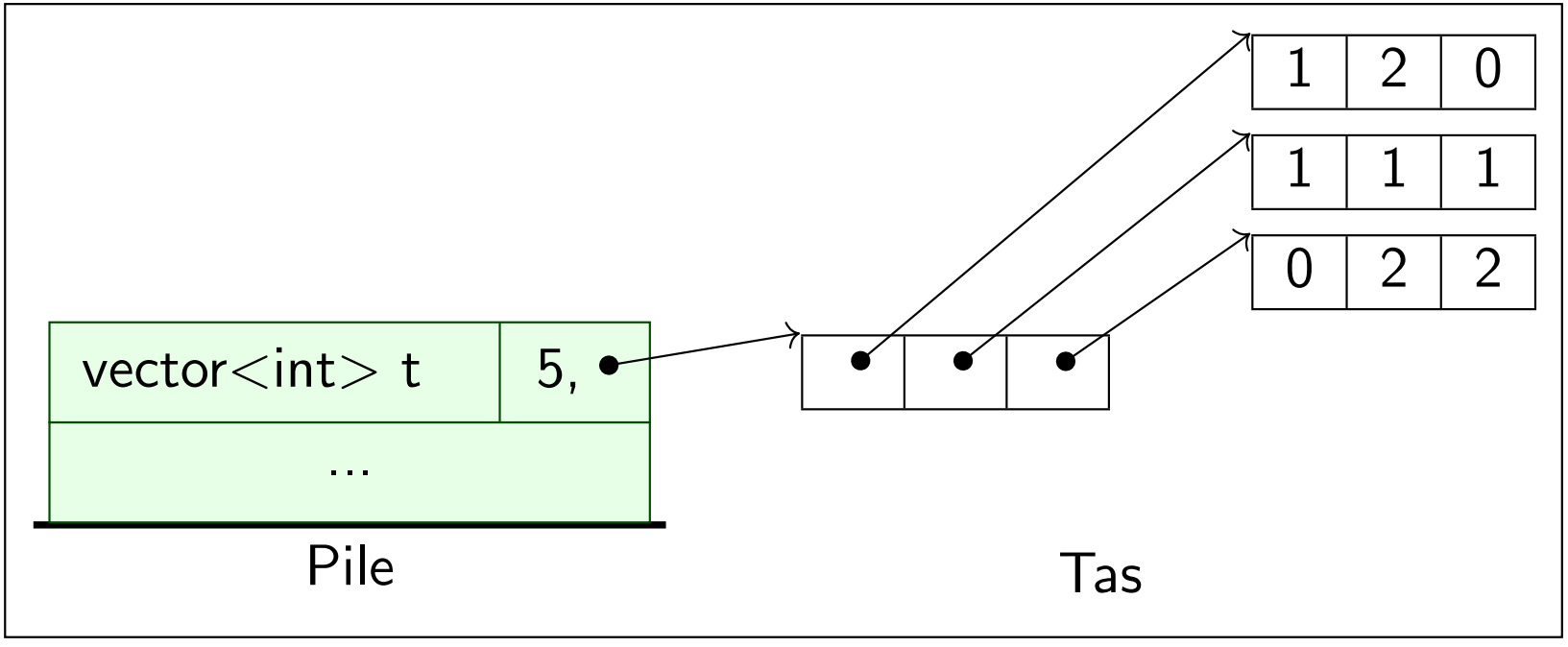
1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

Étapes de la construction en mémoire

1. Déclaration du tableau
2. Allocation du tableau
3. Allocation des sous-tableaux
4. Initialisation



Tableaux à deux dimensions : exemple (grille de morpion)

tableaux2D.cpp

```
// Utilisation
for ( int i=0; i < t.size(); i++ ) {
    for ( int j=0; j < t[i].size(); j++ ) {
        if      ( t[i][j] == 1 ) cout << "X ";
        else if ( t[i][j] == 2 ) cout << "O ";
        else                     cout << "  ";
    }
    cout << endl;
}
```

Tableaux à deux dimensions : exemple (grille de morpion)

tableaux2D-raccourci.cpp

```
// Déclaration, allocation, allocation des sous-tableaux
// et initialisation en une seule instruction
vector<vector<int>> t = {
    { 1,2,0 },
    { 1,1,1 },
    { 0,2,2 },
};

// Utilisation
for ( int i = 0; i < t.size(); i++ ) {
    for ( int j = 0; j < t[i].size(); j++ ) {
        if      ( t[i][j] == 1 ) cout << "X ";
        else if ( t[i][j] == 2 ) cout << "O ";
        else                     cout << "  ";
    }
    cout << endl;
}
```

E. Types de base et représentation des données

Rappels

– Premier modèle de mémoire : Une suite contiguë de 0 et de 1 (les *bits*, groupés par *octets*)

E. Types de base et représentation des données

Rappels

- Premier modèle de mémoire : Une suite contiguë de 0 et de 1 (les *bits*, groupés par *octets*)
- Une variable est un segment nommé de cette mémoire

E. Types de base et représentation des données

Rappels

- Premier modèle de mémoire : Une suite contiguë de 0 et de 1 (les *bits*, groupés par *octets*)
- Une variable est un segment nommé de cette mémoire

Question

Comment représenter des informations avec juste des bits ?

E. Types de base et représentation des données

E. Types de base et représentation des données

Questions

– Quelle information peut-on représenter sur 1 bit ?

E. Types de base et représentation des données

Questions

- Quelle information peut-on représenter sur 1 bit ?
- Sur deux bits ?

E. Types de base et représentation des données

Questions

- Quelle information peut-on représenter sur 1 bit ?
- Sur deux bits ?
- Sur quatre bits ?

E. Types de base et représentation des données

Questions

- Quelle information peut-on représenter sur 1 bit ?
- Sur deux bits ?
- Sur quatre bits ?
- Sur huit bits ?
- Sur n bits ?

E. Types de base et représentation des données

Questions

- Quelle information peut-on représenter sur 1 bit ?
- Sur deux bits ?
- Sur quatre bits ?
- Sur huit bits ?
- Sur n bits ?

À retenir

Le type d'une variable décrit la **structure de donnée** *:*
Comment l'information est **représentée** *par une suite de bits*

E. 1. Les entiers : types `int`, `short`, `long`, ...

R

appel : codage des entiers en base B

Exemple

Le nombre qui s'exprime en base B par les quatre chiffres 1101 vaut :

$$1 \times B^3 + 1 \times B^2 + 0 \times B^1 + 1 \times B^0$$

- En base $B = 10$:
  $1 \times 10^3 + 1 \times 10^2 + 0 \times 10^1 + 1 \times 10^0 = 1000 + 100 + 1 = 1101$
- En base $B = 8$: $1 \times 8^3 + 1 \times 8^2 + 0 \times 8^1 + 1 \times 8^0 = 512 + 64 + 1 = 577$
- En base $B = 2$: $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 1 = 13$

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

– Non signé : $7 + 1 =$

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$
- Signé : $3 + 1 =$

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$
- Signé : $3 + 1 = -4$

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$
- Signé : $3 + 1 = -4$

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$
- Signé : $3 + 1 = -4$
- On calcule modulo $2^3 = 8$!

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$
- Signé : $3 + 1 = -4$
- On calcule modulo $2^3 = 8$!
- Calcul de $-x$ en signé : 1 + complément à 2

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$
- Signé : $3 + 1 = -4$
- On calcule modulo $2^3 = 8$!
- Calcul de $-x$ en signé : 1 + complément à 2
- $-(-4) =$

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$
- Signé : $3 + 1 = -4$
- On calcule modulo $2^3 = 8$!
- Calcul de $-x$ en signé : 1 + complément à 2
- $-(-4) = -4$!

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$
- Signé : $3 + 1 = -4$
- On calcule modulo $2^3 = 8$!
- Calcul de $-x$ en signé : 1 + complément à 2
- $-(-4) = -4$!
- abs$(-4) =$

Entiers non signés et signés sur 3 bits

Exemple

code binaire	entiers non signés	entiers signés
000	0	-4
001	1	-3
010	2	-2
011	3	-1
100	4	0
101	5	1
110	6	2
111	7	3

Remarques

- Non signé : $7 + 1 = 0$
- Signé : $3 + 1 = -4$
- On calcule modulo $2^3 = 8$!
- Calcul de $-x$ en signé : 1 + complément à 2
- $-(-4) = -4$!
- abs$(-4) = -4$!

Entiers machine vs entiers mathématiques

Entiers machine

– Représentés sur un mot machine

Entiers machine vs entiers mathématiques

Entiers machine

- Représentés sur un mot machine
- Sur une machine à n bits on peut représenter 2^n entiers, soit les entiers compris entre -2^{n-1} et $2^{n-1} - 1$
- Les bornes sont données par `-INT_MAX-1` et `INT_MAX`
- Voir `Exemples/int.cpp`

Entiers machine vs entiers mathématiques

Entiers machine

- Représentés sur un mot machine
- Sur une machine à n bits on peut représenter 2^n entiers, soit les entiers compris entre -2^{n-1} et $2^{n-1} - 1$
- Les bornes sont données par `-INT_MAX-1` et `INT_MAX`
- Voir `Exemples/int.cpp`

Attention !

- Les entiers machine sont des **approximations** des entiers mathématiques !

Entiers machine vs entiers mathématiques

Entiers machine

- Représentés sur un mot machine
- Sur une machine à n bits on peut représenter 2^n entiers, soit les entiers compris entre -2^{n-1} et $2^{n-1} - 1$
- Les bornes sont données par `-INT_MAX-1` et `INT_MAX`
- Voir `Exemples/int.cpp`

Attention !

- Les entiers machine sont des **approximations** des entiers mathématiques !
- Pour de grosses valeurs : risque de **débordement**

Entiers machine vs entiers mathématiques

Entiers machine

- Représentés sur un mot machine
- Sur une machine à n bits on peut représenter 2^n entiers, soit les entiers compris entre -2^{n-1} et $2^{n-1} - 1$
- Les bornes sont données par `-INT_MAX-1` et `INT_MAX`
- Voir `Exemples/int.cpp`

Attention !

- Les entiers machine sont des **approximations** des entiers mathématiques !
- Pour de grosses valeurs : risque de **débordement**
- Il est aussi possible de calculer avec des « vrais » entiers !

Quelques variantes

- Entiers longs : `long` ; voir `Exemples/long.cpp`
- Entiers courts : `short` ; voir `Exemples/long.cpp`
- Entiers non signés : `unsigned int`, `unsigned long`, ...

Remarque

Le nombre de bits utilisés (et donc les bornes) peuvent dépendre du compilateur, du système d'exploitation, du processeur, ...

E. 2. Les réels : types `float`, `double`

Motivation

– Représenter des nombres réels ?

E. 2. Les réels : types `float`, `double`

Motivation

- Représenter des nombres réels ?
- Nombres approchés (chiffres significatifs)
- Grande variations d'ordres de grandeur

E. 2. Les réels : types `float`, `double`

Motivation

- Représenter des nombres réels ?
- Nombres approchés (chiffres significatifs)
- Grande variations d'ordres de grandeur

Nombres à virgule flottante

- Représentation par *mantisse* et *exposant* : *3.423420*e+*05*

E. 2. Les réels : types `float`, `double`

Motivation

- Représenter des nombres réels ?
- Nombres approchés (chiffres significatifs)
- Grande variations d'ordres de grandeur

Nombres à virgule flottante

- Représentation par *mantisse* et *exposant* : *3.423420*e+*05*
- Voir : `Exemples/float.cpp` et `Exemples/double.cpp`.

E. 2. Les réels : types `float`, `double`

Motivation

- Représenter des nombres réels ?
- Nombres approchés (chiffres significatifs)
- Grande variations d'ordres de grandeur

Nombres à virgule flottante

- Représentation par *mantisse* et *exposant* : *3.423420*e+*05*
- Voir : `Exemples/float.cpp` et `Exemples/double.cpp`.
- Un certain nombre de bits pour la mantisse
- Les bits restant pour l'exposant
- Les détails de la représentation varient suivant les langages de programmation, les machines et les normes utilisées
- Normes IEEE très précises sur les règles d'arrondis

E. 3. Les caractères : type `char`

- Permettent de stocker un seul caractère :
  - Une lettre de l'alphabet (sans accent) : 'a', ..., 'z', 'A', ..., 'Z'
  - Un chiffre '0', ..., '9'
  - Un caractères du clavier ('@', '+', '/', ' ')
  - Quelques caractères spéciaux

E. 3. Les caractères : type `char`

- Permettent de stocker un seul caractère :
  - Une lettre de l'alphabet (sans accent) : 'a', ..., 'z', 'A', ..., 'Z'
  - Un chiffre '0', ..., '9'
  - Un caractères du clavier ('@', '+', '/', ' ')
  - Quelques caractères spéciaux
- Notés entre apostrophes (exemple : `'A'`) pour distinguer le caractère `'A'` de la variable `A`

E. 3. Les caractères : type `char`

- Permettent de stocker un seul caractère :
  - Une lettre de l'alphabet (sans accent) : 'a', ..., 'z', 'A', ..., 'Z'
  - Un chiffre '0', ..., '9'
  - Un caractères du clavier ('@', '+', '/', ' ')
  - Quelques caractères spéciaux
- Notés entre apostrophes (exemple : `'A'`) pour distinguer le caractère `'A'` de la variable `A`
- La table ASCII associe un numéro unique entre 0 et 127 à chaque caractère, ce qui permet d'introduire un ordre

E. 3. Les caractères : type `char`

- Permettent de stocker un seul caractère :
  - Une lettre de l'alphabet (sans accent) : 'a', ..., 'z', 'A', ..., 'Z'
  - Un chiffre '0', ..., '9'
  - Un caractères du clavier ('@', '+', '/', ' ')
  - Quelques caractères spéciaux
- Notés entre apostrophes (exemple : `'A'`) pour distinguer le caractère `'A'` de la variable `A`
- La table ASCII associe un numéro unique entre 0 et 127 à chaque caractère, ce qui permet d'introduire un ordre
- Et les lettres accentuées ? Les caractères chinois ? ...

E. 3. Les caractères : type `char`

- Permettent de stocker un seul caractère :
  - Une lettre de l'alphabet (sans accent) : 'a', ..., 'z', 'A', ..., 'Z'
  - Un chiffre '0', ..., '9'
  - Un caractères du clavier ('@', '+', '/', ' ')
  - Quelques caractères spéciaux
- Notés entre apostrophes (exemple : `'A'`) pour distinguer le caractère `'A'` de la variable `A`
- La table ASCII associe un numéro unique entre 0 et 127 à chaque caractère, ce qui permet d'introduire un ordre
- Et les lettres accentuées ? Les caractères chinois ? ...
  Voir : Unicode, UTF-8

Voir `Exemples/char.cpp`

E. 4. Les chaînes de caractères : type `string`

- Permettent de stocker une suite de caractères :
  un mot, une phrase, ...
- Notées entre guillemets doubles
  Exemple : `"Bonjour"`
- Se comportent essentiellement comme des tableaux de caractères

Opérations

opération	exemple	résultat
concaténation	`"bonjour"+ "toto"`	`"bonjourtoto"`
indexation	`"bonjour"[3]`	`'j'`
longueur	`"bonjour".length()`	7

E. 5. Les booléens : type `bool`

Notes

Les variables booléennes ne peuvent prendre que deux valeurs :

- vrai (mot clé `true`)
- faux (mot clé `false`)

Les opérations possibles sur les booléens sont :

- la négation (opération unaire, mot clé `not`)
- la conjonction (opération binaire, mot clé `and`)
- la disjonction (opération binaire, mot clé `or`)
- ...

Expressions booléennes : encadrements

Attention !

Les encadrements ne peuvent pas être écrits directement en C++
Ils doivent être réalisés à l'aide de deux comparaisons connectées par l'opérateur `and`

Exemple

L'encadrement mathématique :

$$0 \leq x \leq 15$$

se traduit en C++ par l'expression booléenne :

```
(0 <= x) and (x <= 15)
```

Évaluation paresseuse des expressions booléennes

Exemple

Quelle est la valeur des expressions suivantes :

- `false and ( 3*x + 1 >= 2 or 1/(1+x) < 42 )`
- `true or ( 3*x + 1 >= 2 or 1/(1+x) < 42 )`

Deux possibilités :

- l'**évaluation complète** : évaluer tous les opérandes des expressions booléennes

Évaluation paresseuse des expressions booléennes

Exemple

Quelle est la valeur des expressions suivantes :

- `false and ( 3*x + 1 >= 2 or 1/(1+x) < 42 )`
- `true or ( 3*x + 1 >= 2 or 1/(1+x) < 42 )`

Deux possibilités :

- l'**évaluation complète** : évaluer tous les opérandes des expressions booléennes
- l'**évaluation paresseuse** : stopper l'évaluation dès que l'on peut :
  - Pour une conjonction `a and b` on peut s'arrêter si `a` est faux
  - Pour une disjonction `a or b` on peut s'arrêter si `a` est vrai

Résumé

Collections

- Un modèle de mémoire raffiné avec pile et tas
- L'allocation des tableaux, sur le tas
- Les tableaux à deux dimensions
  Construction en quatre étapes !
- D'autres collections

Résumé

Collections

- Un modèle de mémoire raffiné avec pile et tas
- L'allocation des tableaux, sur le tas
- Les tableaux à deux dimensions
  Construction en quatre étapes !
- D'autres collections

Représentation des données en mémoire

- Types de base : int, long, float, double, char, bool

Résumé

Collections

- Un modèle de mémoire raffiné avec pile et tas
- L'allocation des tableaux, sur le tas
- Les tableaux à deux dimensions
  Construction en quatre étapes !
- D'autres collections

Représentation des données en mémoire

- Types de base : int, long, float, double, char, bool
- Types composites : string, vector, struct

Résumé

Collections

- Un modèle de mémoire raffiné avec pile et tas
- L'allocation des tableaux, sur le tas
- Les tableaux à deux dimensions
  Construction en quatre étapes !
- D'autres collections

Représentation des données en mémoire

- Types de base : int, long, float, double, char, bool
- Types composites : string, vector, struct
- À partir de ceux-ci, on peut représenter tout type d'information : Texte, images, sons, ...
- Le *type* d'une variable décrit la **structure de donnée** :
  Comment l'information est **représentée** par une suite de bits