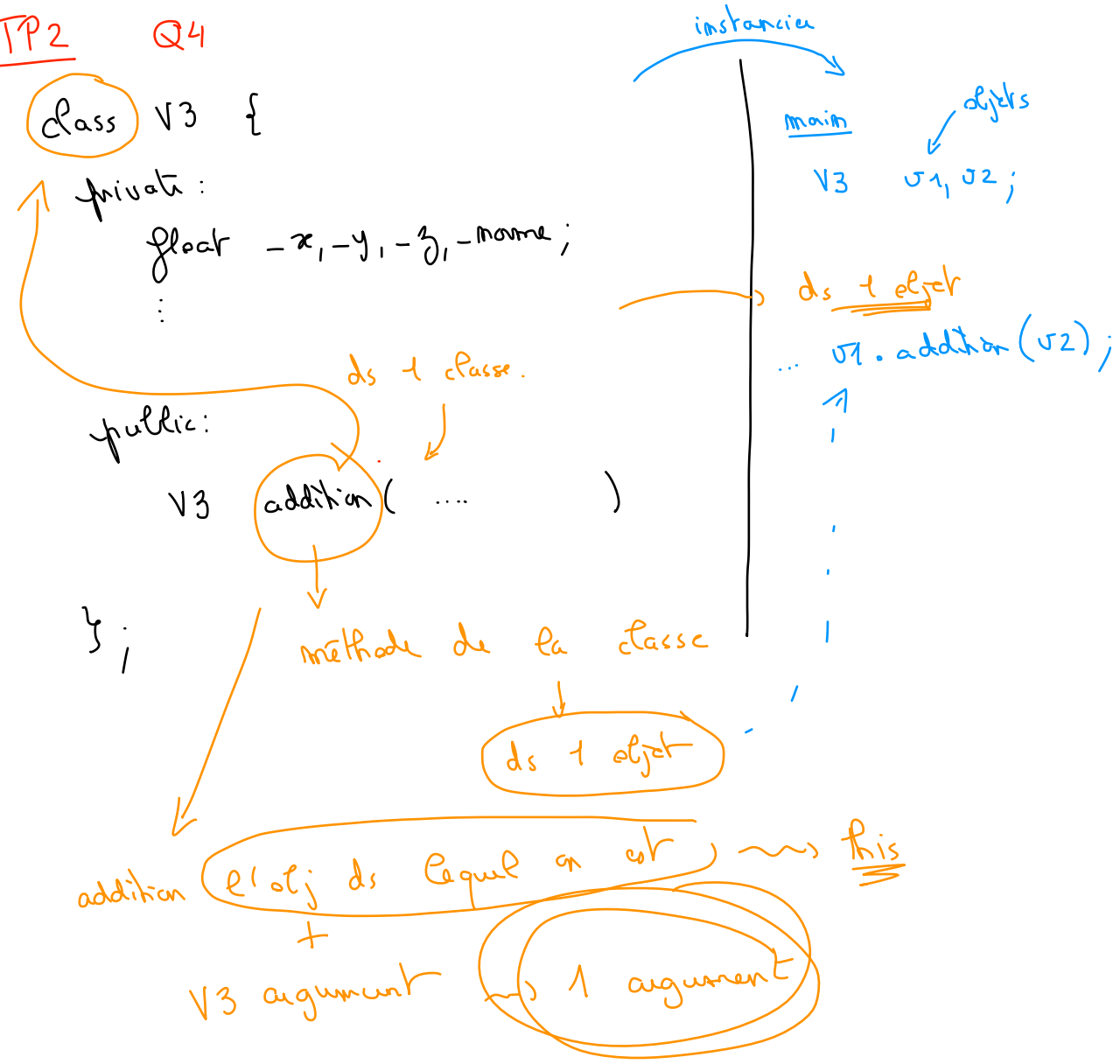
TP2 Q4



addition : this + arg

retourné.

class V3 {

:

V3 addition (V3 v)

(const V3 & v)

passage par réf. constante.

4 float ... + utile.

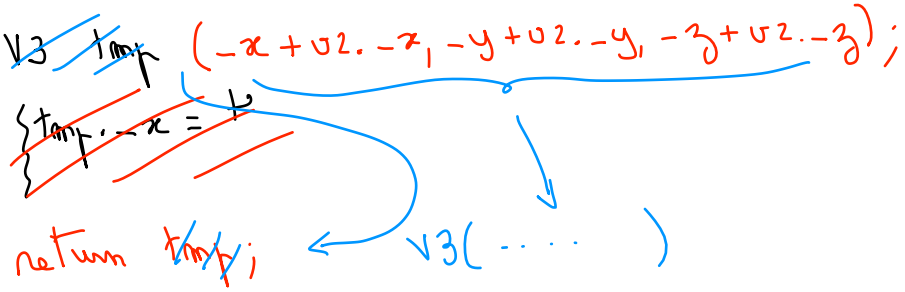
passage par val

copie .....

objets

impl.

```
V3   V3::addition (const V3 &v2)
{    // retourner "this + v2"
```



.h

V3(float x=0, float y=0, float z=0)

main ↓

V3 v1(1,2,3);

V3 v2; ← défaut

Constr par copie : (≠ affectation !)

V3(const V3 & v) ⇝ " this ← v "

main ↓

① V3 v3 = V3(v1);

① : constr v3 par défaut

② : affectation

③ : on construit un V3 par copie de v1

V3 v3(v1);

---

v1.~~add~~(v2) ⇝ v1 + v2

operator+ / V3.

class V3 {

V3 operator+ (const V3 & v2)

↓

this + v2

1er arg implicite : this

v1 + v2

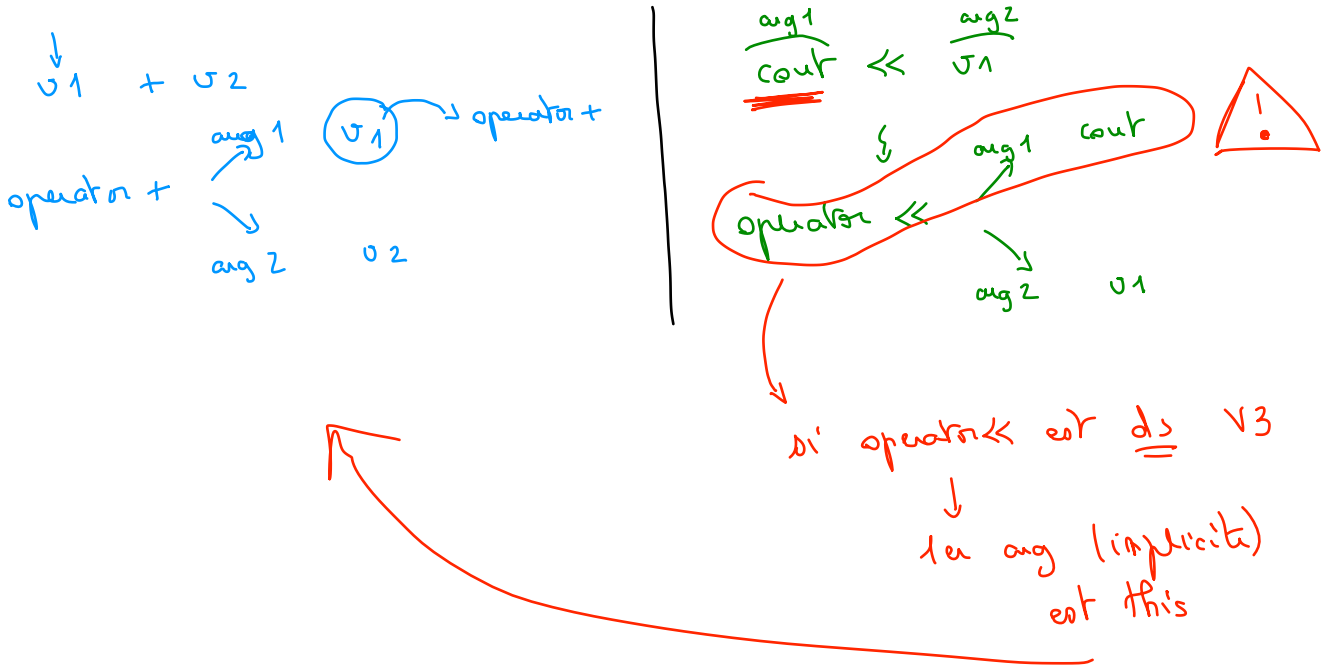
v1.operator+(v2)

operator<< ⇝

(cout << v1) << endl.

↓ ↓

cout cout



Pour que << doit hors de la classe:

→ friend → hors classe

sol 1 → accès aux données privées.

sol 2 → défini hors de la classe

↓

appelle afficher ! ← ostream &

ostream &

class V3

private:

float _x, _y, _z;

public:

inline float get_x () { return _x; }

main

V3 v(1,2,3);

v.get_x();

```
class V3 {
V3 (float x=0, float y=0, float z=0);
V3 operator+ (const V3 & v);
```

!

**

V3 v1 (1,2,3), v2 ;

v2 = v1 + 2 ;

V3

int ⇝ V3 ?

int → float → on peut construire un V3 à partir d'1 seul float (**)

V3

$\begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$

v1.operator+(2) ;

2 + 2.5

int → float

, float

→ +float

~~v2 = 2 + v1~~

⇝ surcharges symétriques opérateurs

↗ friend

↘ hors classe ←

v1 + 1 / 1 + v1 ← ok.

⇝ const pour une meth.

la meth. de modifie pas l'arg.

V3 addition (const V3 & v) const

la méthode ne modifie pas this.

v non mod

V3 fct_bidon (const V3 & v)

addition ne doit pas modifier v ...

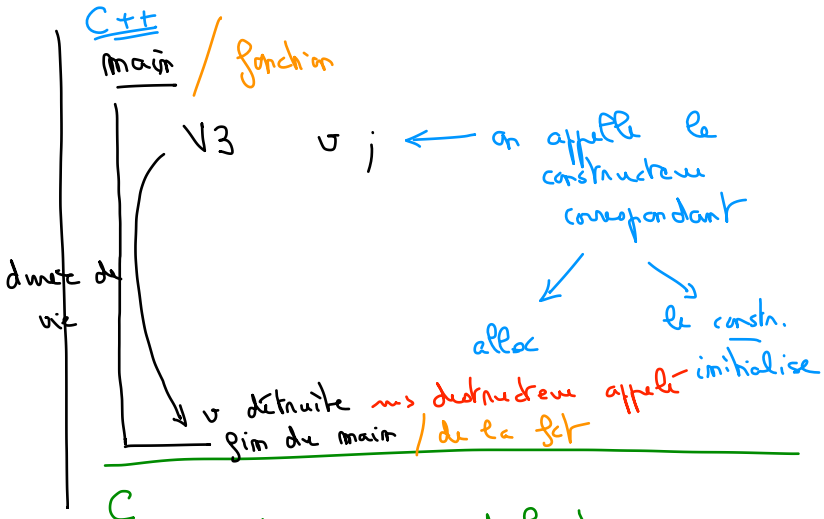
{ v.addition (V3(1,1,1)); ... }

Destructeur

```
class V3 {
      .
      .
      .
}
```

destructeur

inline ~V3 ()
{cout << "destructeur" <<endl; }



C

int i ;

déclaration
⇓
allocation
~~init~~