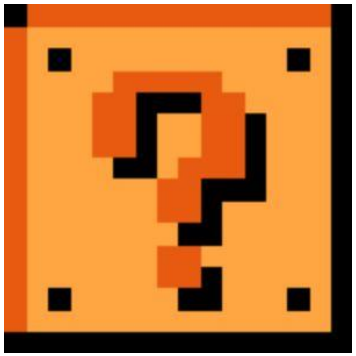
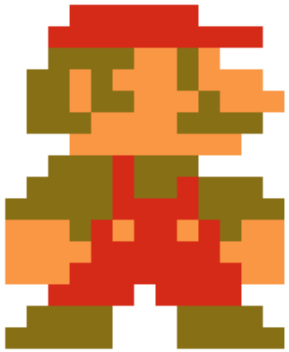
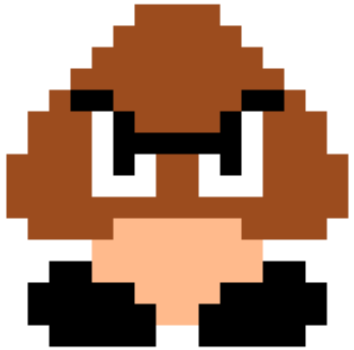
# Entity Component System

**HMIN317**

Source Alla Sheffer

# Qu’est ce qu’une entité ?

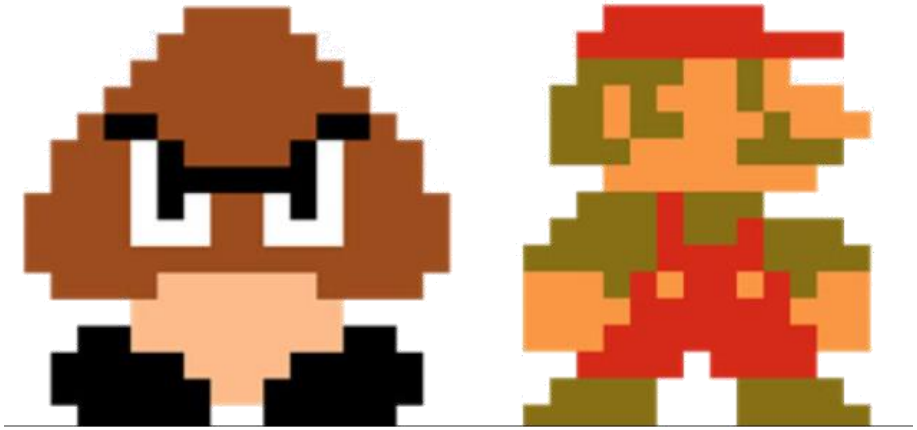
- **Entités** : choses qui existent dans le monde du jeu

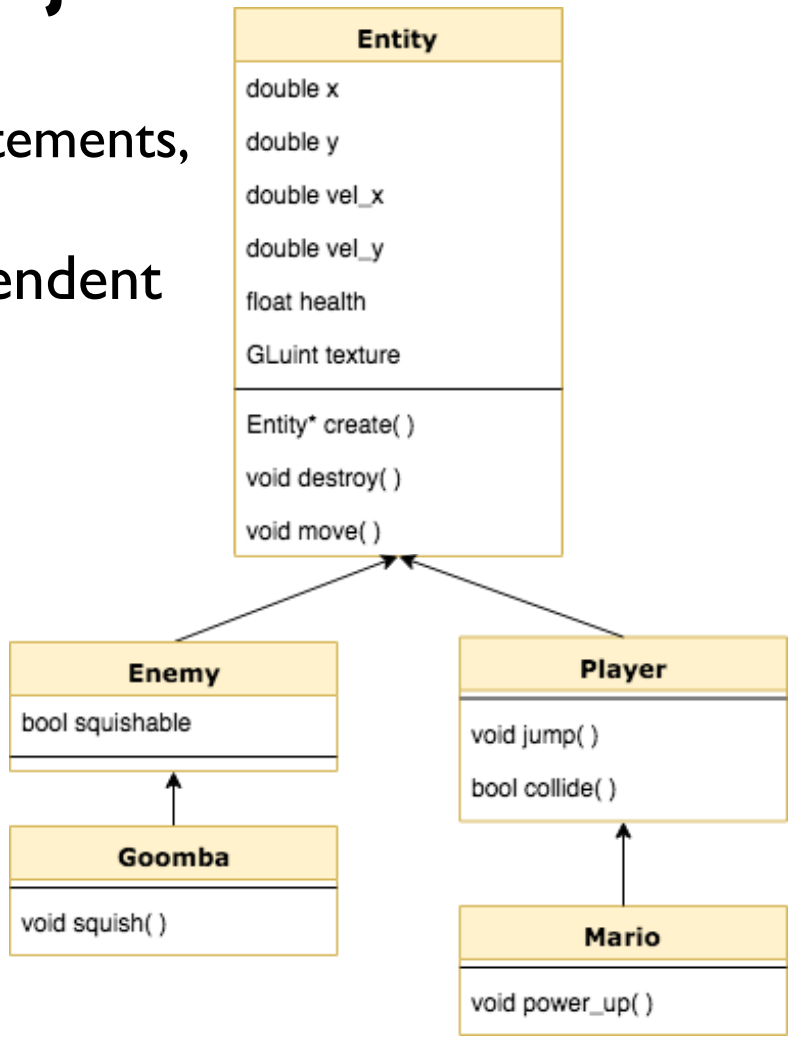





BGM

# Entités en programmation jeu classique

- **Programmation Orientée-Object**
  - Entités comme des objets
    - Contiennent des données, comportements, etc…
  - Hiérarchie d'entités : entités qui étendent d'autres entités



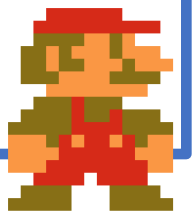


# Hiérarchie d'entités

```
class Entity {

public:
    void create();
    void destroy();
    void move();

private:
    double x;
    double y;
    double vel_x;
    double vel_y;
    vec2 bbox;
    float health;
    GLuint texture;
}
```

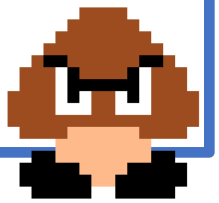
```
class Player : public Entity {

public:
    void jump();
    bool collide();
}
```

```
class Mario : public Player {

public:
    void power_up();
}
```



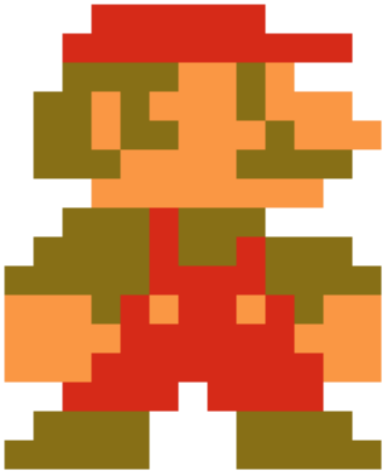
```
class Enemy : public Entity {

private:
    bool squishable;
}
```

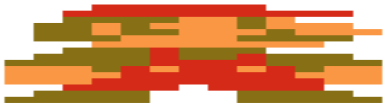
```
class Goomba : public Goomba {

public:
    void squish();
}
```

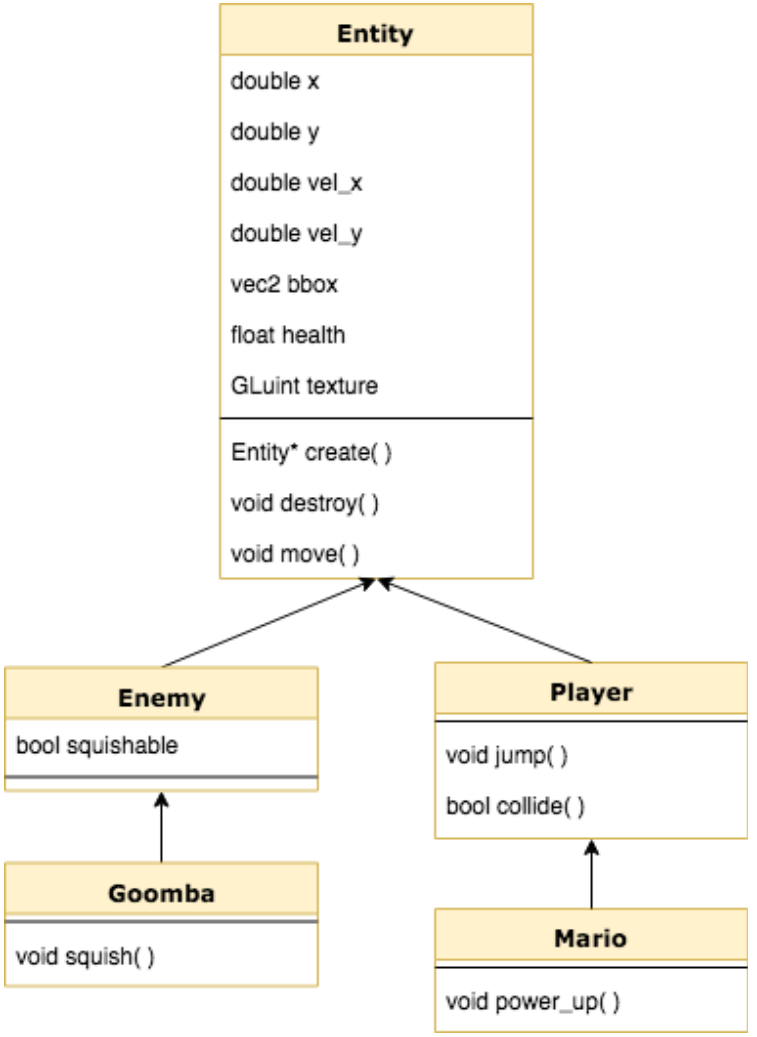


# Problème avec l'approche OO

Comment fait-on pour squisher Mario ?



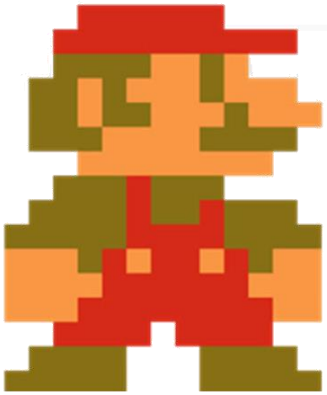


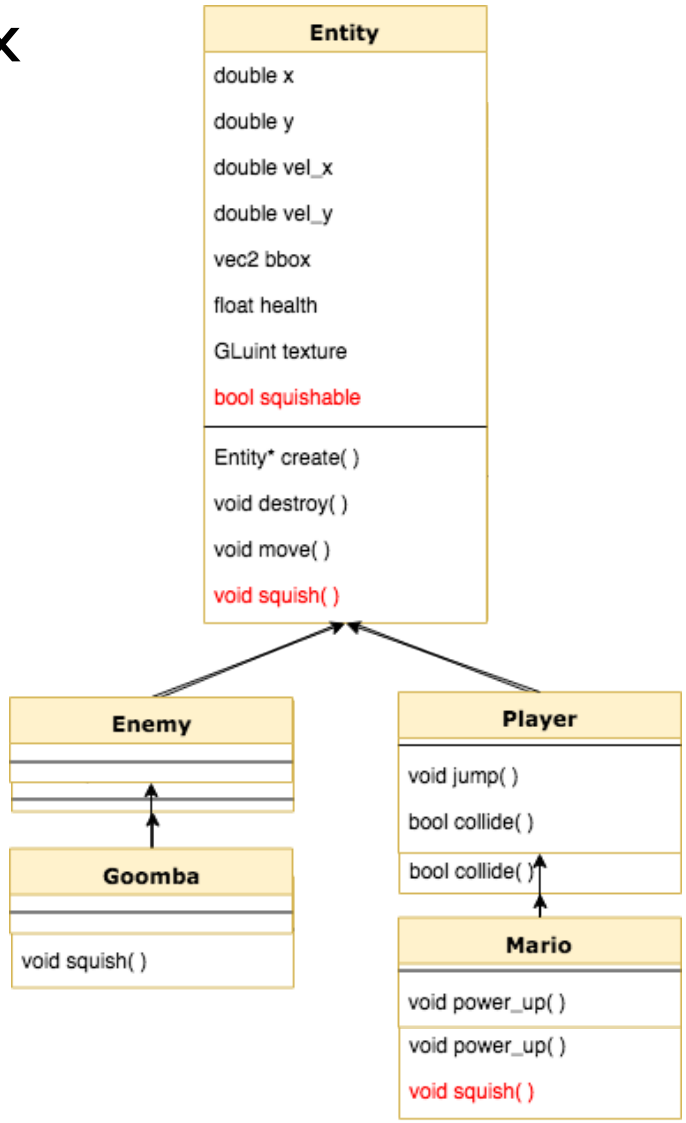


# Problème avec l'approche OO

- Difficile d'ajouter de nouveaux comportements
  - **Répliquer du code**

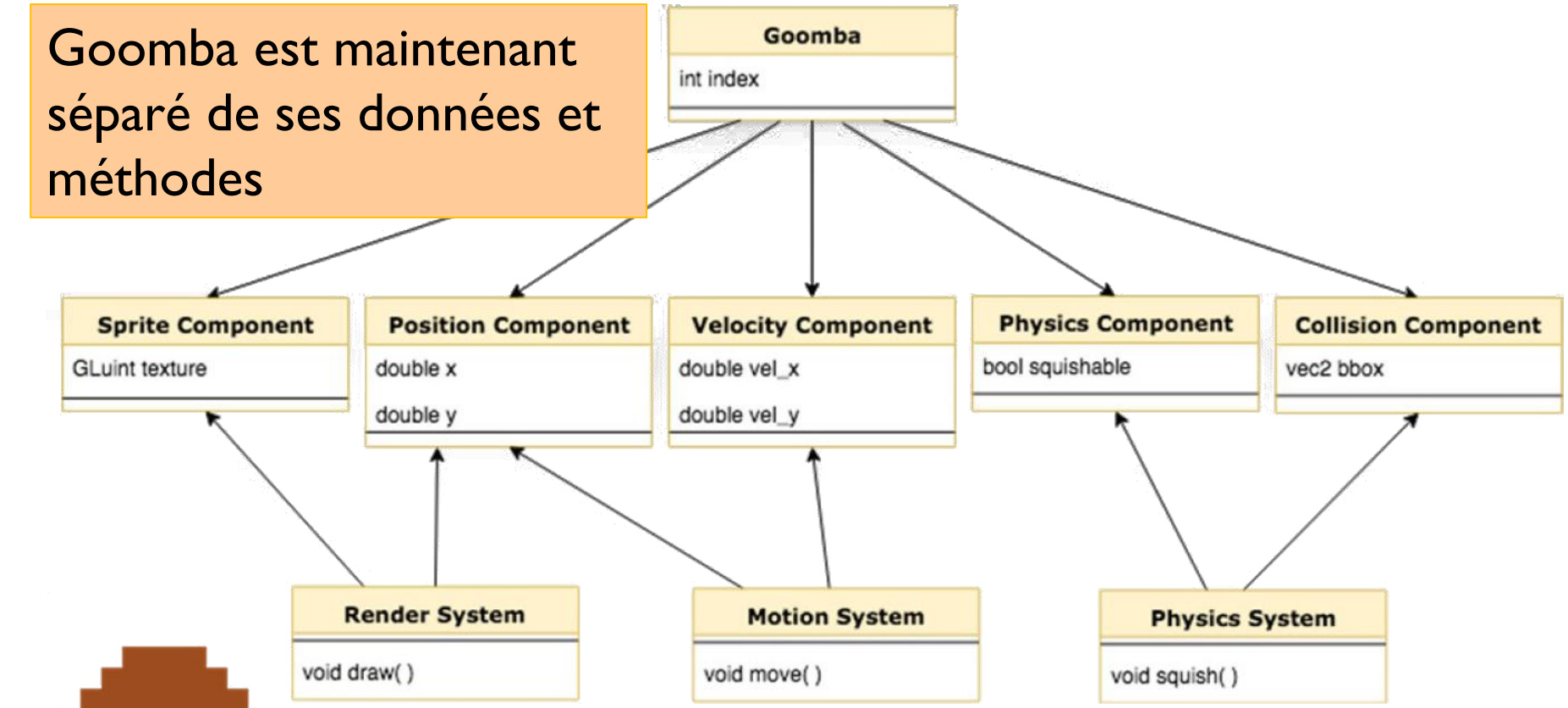
  ou
  - Classes parentes de **taille gigantesque**





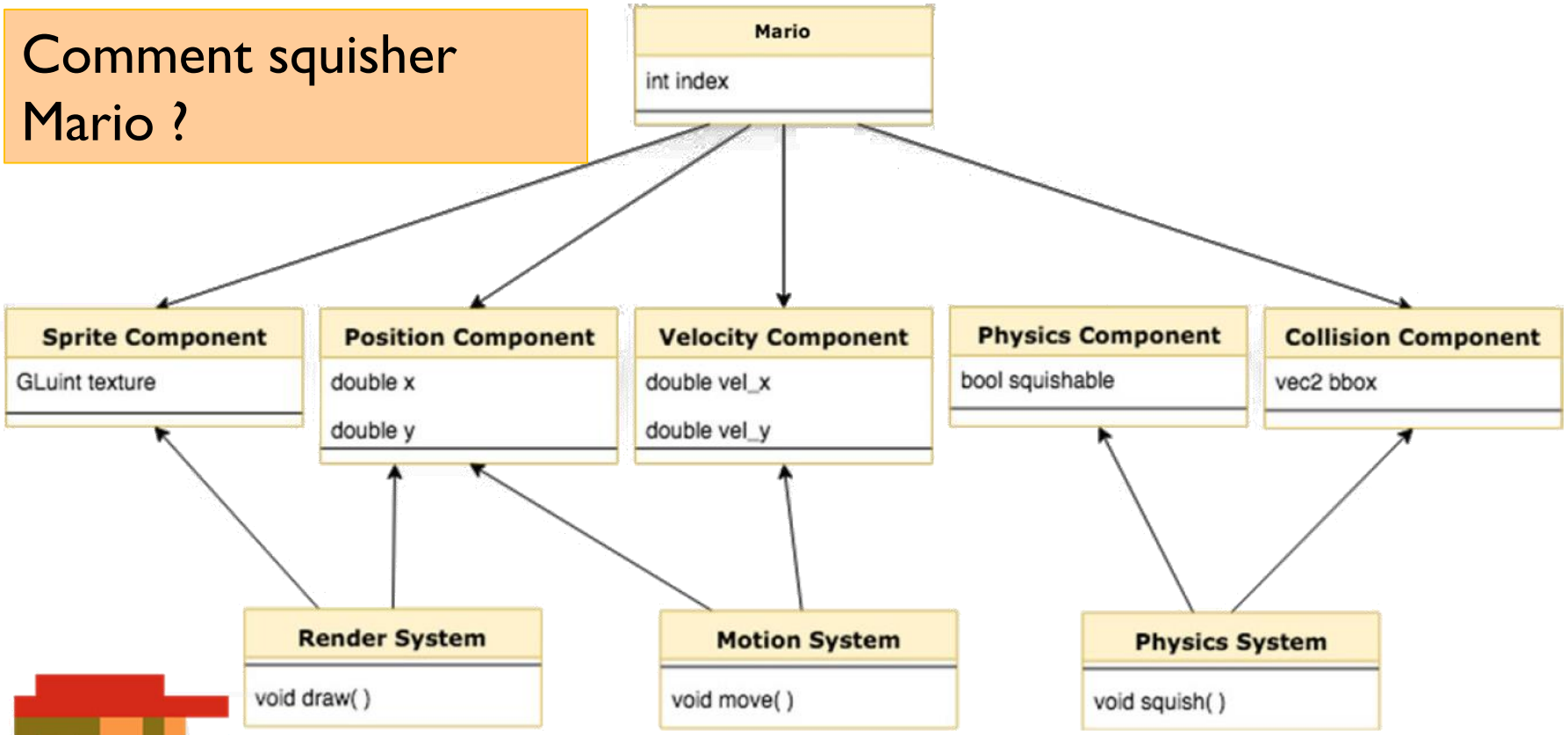
# Exemple de diagramme ECS

Goomba est maintenant séparé de ses données et méthodes



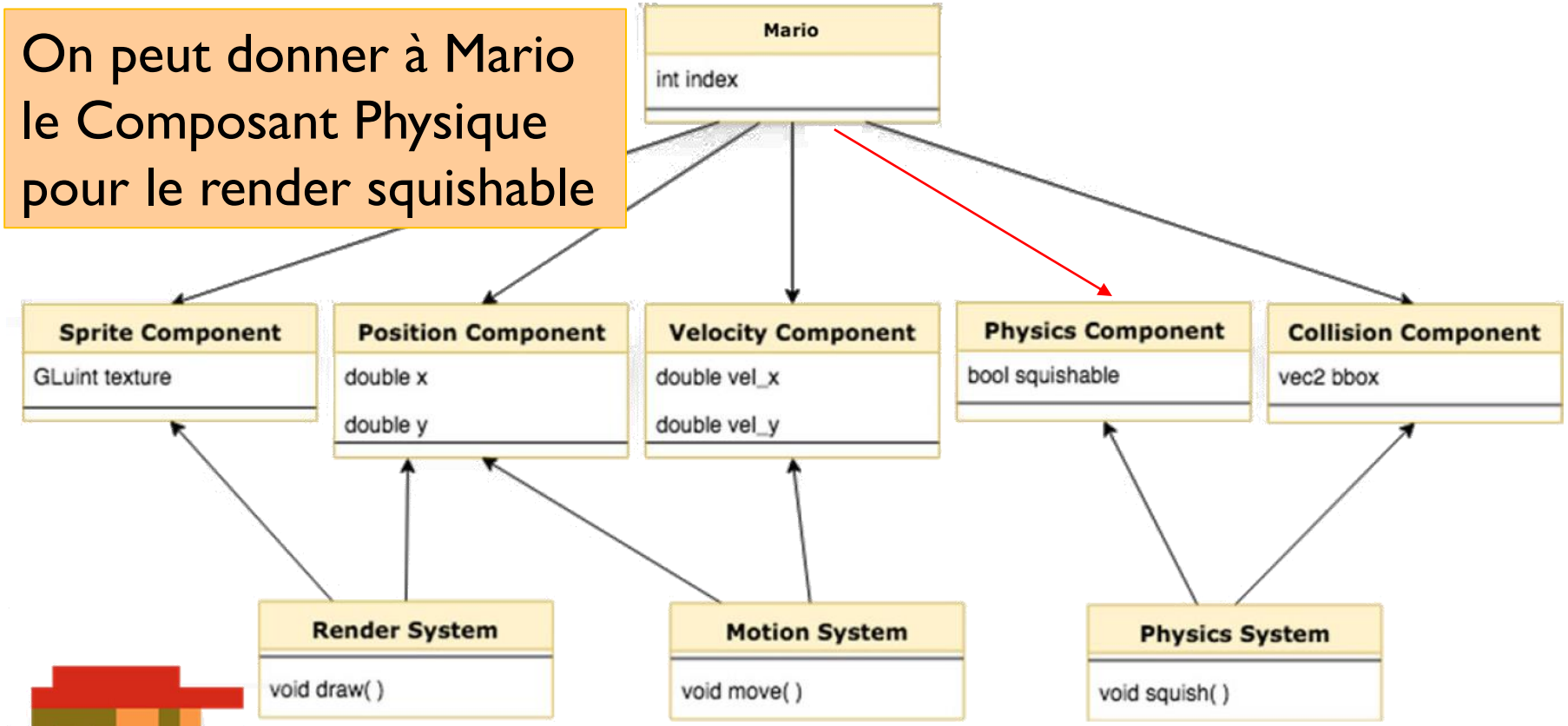

# Exemple de diagramme ECS

Comment squisher Mario ?




# Exemple de diagramme ECS

On peut donner à Mario le Composant Physique pour le render squishable




# Qu’est ce que l’ECS ?

- Alternative à la programmation orientée objet

- La donnée est **indépendante** et **modulaire**
  - Concept similaire pour construire des blocks
  - Les Entités ne possèdent plus leurs données
  - Les Entités choisissent

# Qu’est ce que l’ECS ?

- Les actions des entités **seulement** déterminées par **leurs données**
  - Boucle de mise à jour n’a pas besoin des références aux Entités
  - Les systèmes cherchent les entités avec les bonnes parties (données) et mettent à jour
    - Pour bouger Mario a besoin d’une position et d’une vitesse

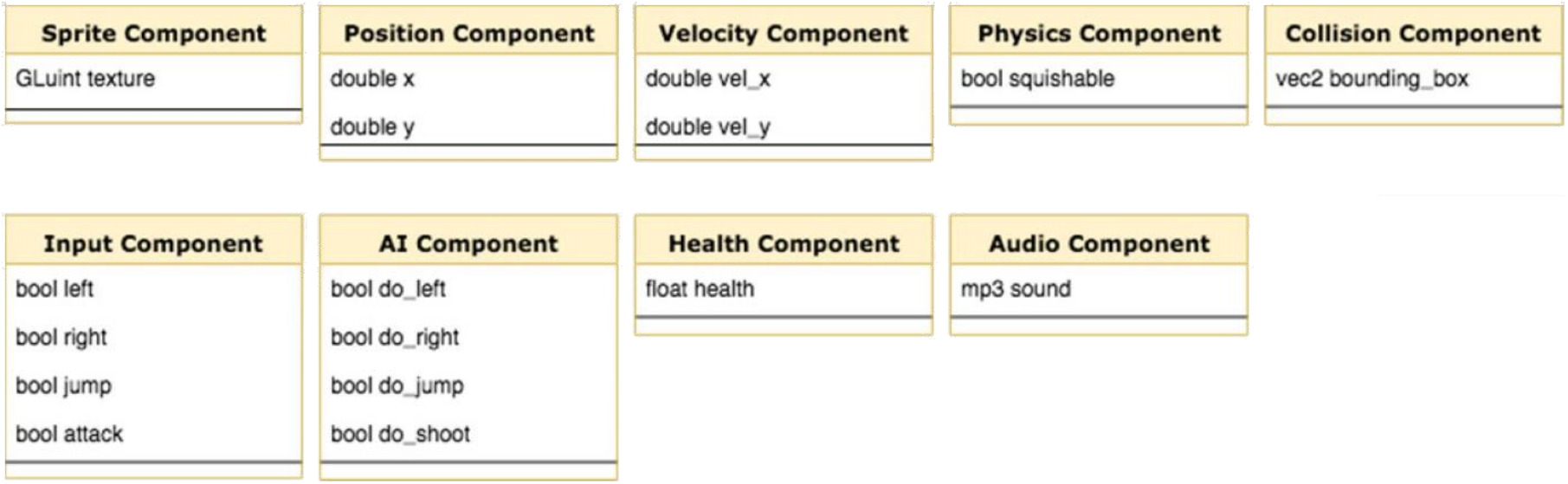
# Qu’est ce que l’ECS ?

- **Composition** plutôt que **hiérarchie**

- **E**ntités sont une collection de **Composants**
- **C**omposants contiennent les **données du jeu**
  - Position, vitesse, entrées…
- **S**ystèmes sont une collection d’**actions**
  - Système de rendu, système de mouvement, …

# Composant

- Contient **seulement** des données du jeu
- Décrit un unique aspect d'une entité
  - Ex une entité trompette aura un composant audio



# Composant

- Typiquement implémenté avec des structs

```
struct SpriteComponent {
    GLuint texture;
}
```

```
struct PositionComponent {
    double x;
    double y;
}
```

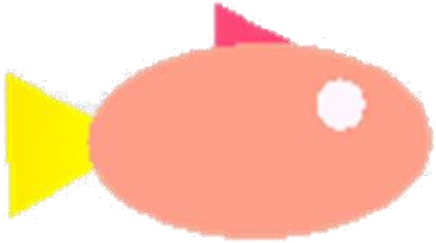
```
struct VelocityComponent {
    double vel_x;
    double vel_y;
}
```

```
struct PhysicsComponent {
    bool squishable;
}
```

```
struct CollisionComponent {
    vec2 bbox;
}
```

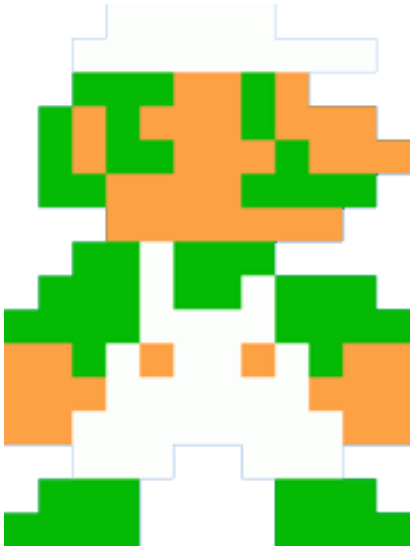
# Quels composants créer ?

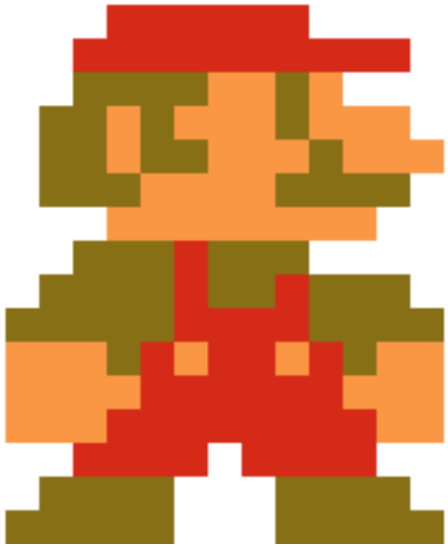
- Quels composants donneriez-vous à ces entités ?



# Composants

- Simplicité d'ajout d'une nouvelle caractéristique à l'Entité
  - Créer le Composant désiré et la donner à l'Entité

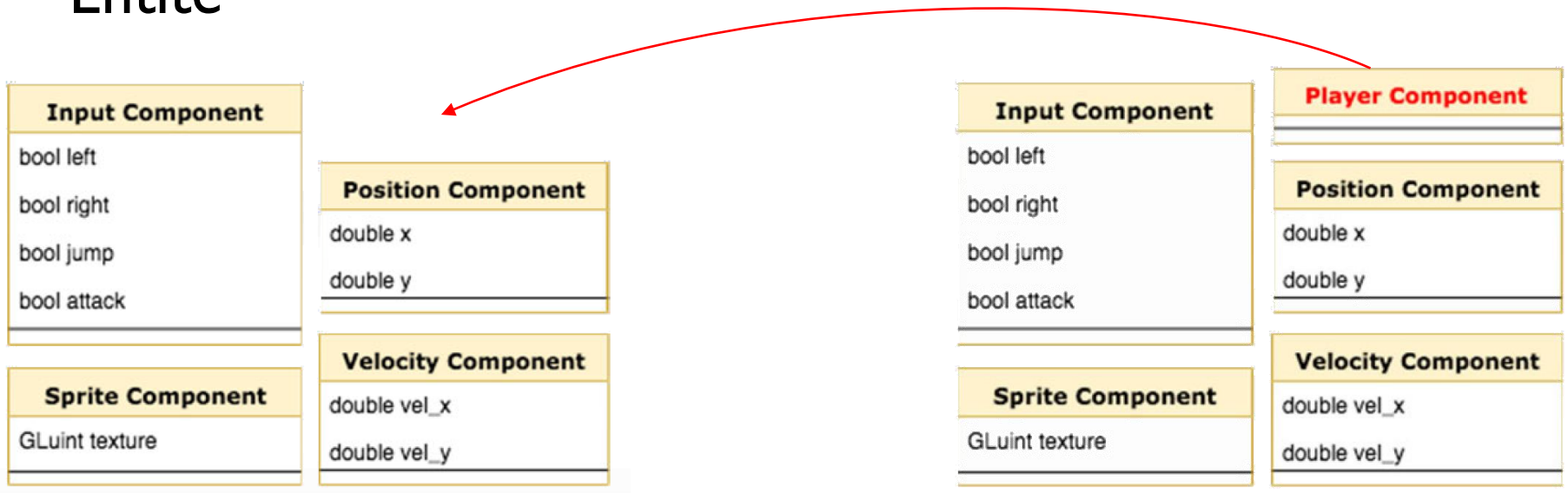


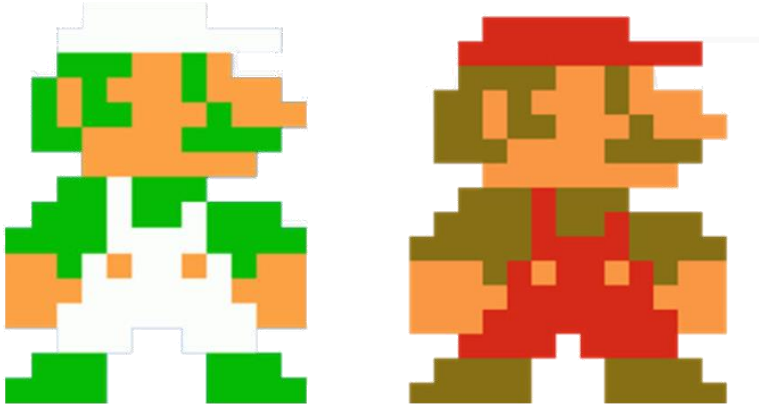


Comment changer le personage jouable de Mario à Luigi ?

# Composants

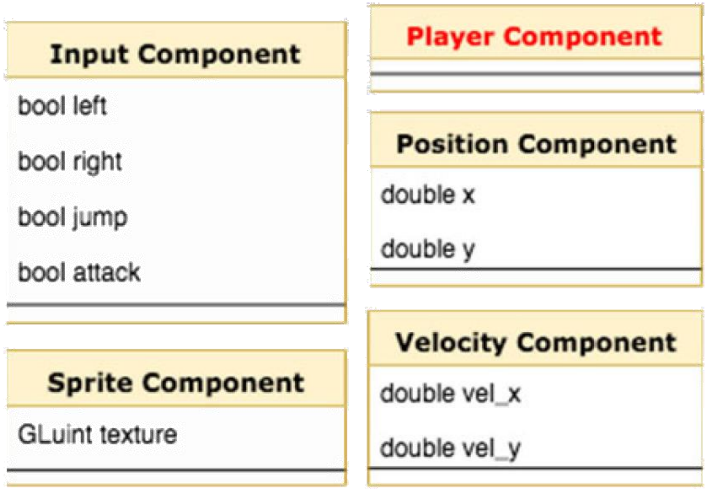
- Un composant vide peut être utilisé pour marquer une Entité

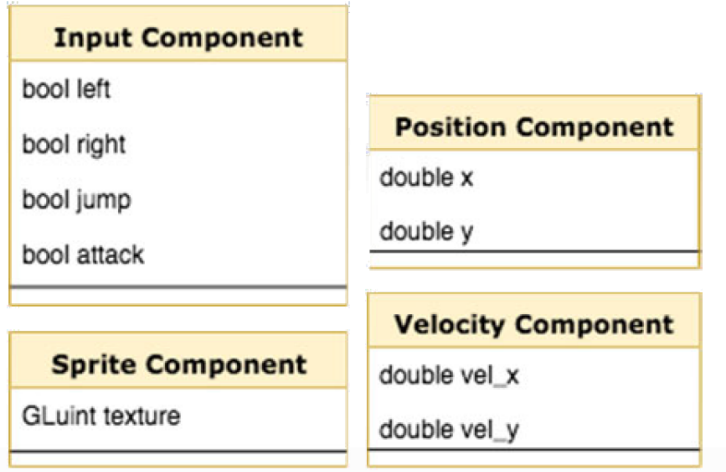


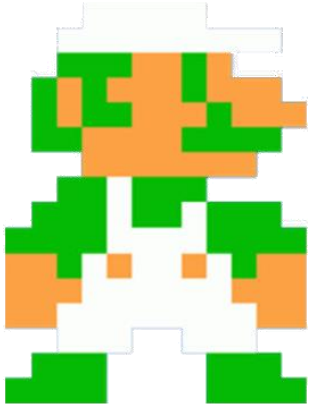
# Composants

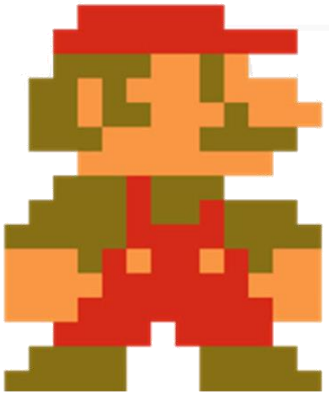
- Un composant vide peut être utilisé pour marquer une Entité




Luigi est identifié comme le joueur actif



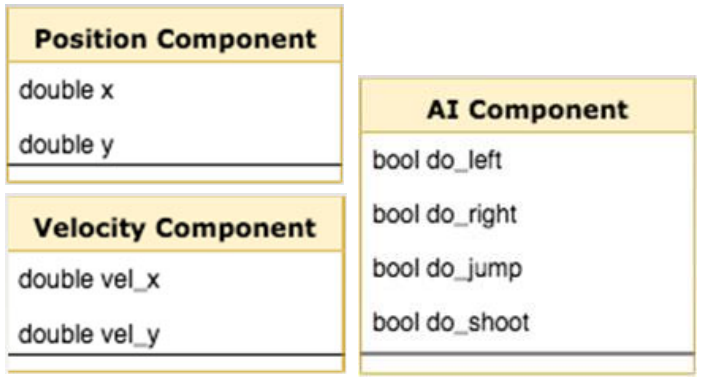


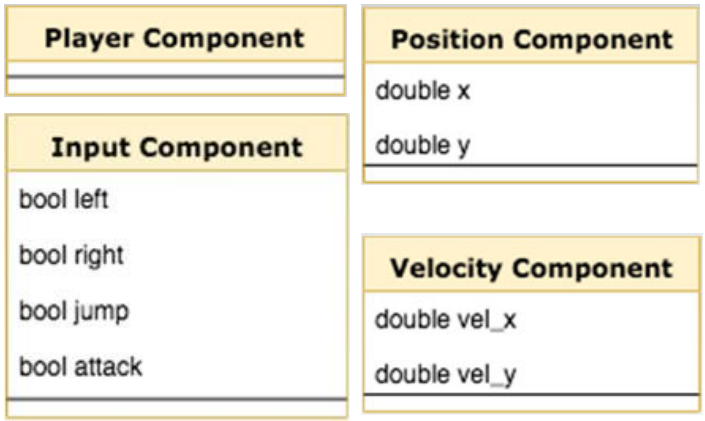
# Systèmes

- Les groupes de Composant décrivent **un comportement/une action**
  - Ex : boite englobante, position et vitesse décrivent les collisions
- Les Systèmes codent un **comportement/une action**
- Opèrent sur des Entités avec des **groupes de Composants liés**
  - Liés : décrivent **le même (type de)** comportement/action
  - Ex : rendre toutes les Entités avec un sprite et une position
- Le comportement des Entités peut être **dynamique**
  - Ajouté/supprimé à la volée

# Exemple de Système

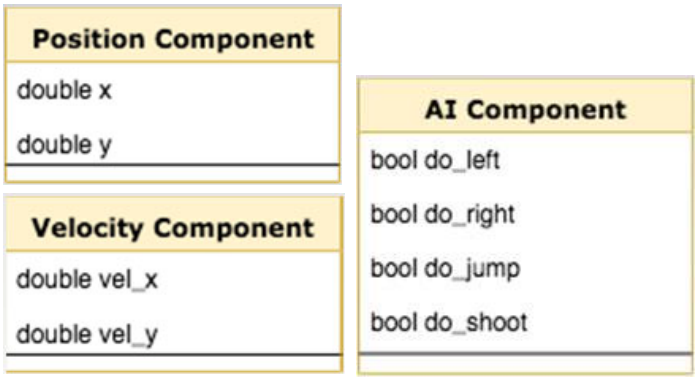
- Quels systèmes sont décrits par ce groupe de Composants ?



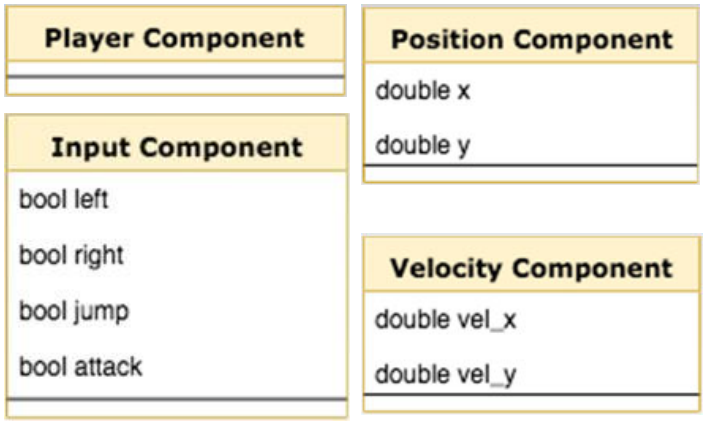



# Exemple de Système

- Quels systèmes sont décrits par ce groupe de Composants ?




**Système de déplacement de l'ennemi**




**Système de déplacement du joueur**

# Comment un système trouve ses entités ?

- Méthode Bitmap
  - Chaque Entité a une séquence de bits représentant ses Composants
  - Chaque Système a une séquence de bits représentant les Composants qui l'intéresse

# Comment un système trouve ses entités ?

- Méthode Bitmap
  - Exemple de code

```
function RenderSystem.process(entityList) {
    // Loops through all entities
    foreach(entity in entityList) {
        // Let's say that position 1 and 3 represent the position and sprite components.
        if ((entity.componentBitMap & '1010') == '1010') {
            graphicsContext.render(
                entity.componentList['PositionComponent'].x,
                entity.componentList['PositionComponent'].y,
                entity.componentList['SpriteComponent'].image
            );
        }
    }
}
```

# Comment un système trouve ses entités ?

- Méthode Bitmap
  - Couteux pour chaque boucle de mise à jour

```
function RenderSystem.process(entityList) {
    // Loops through all entities
    foreach(entity in entityList) {
        // Let's say that position 1 and 3 represent the position and sprite components.
        if ((entity.componentBitMap & '1010') == '1010') {
            graphicsContext.render(
                entity.componentList['PositionComponent'].x,
                entity.componentList['PositionComponent'].y,
                entity.componentList['SpriteComponent'].image
            );
        }
    }
}
```

  - Peut-ou faire mieux ?

Oui !

# Comment un système trouve ses entités ?

- **Gestionnaire d'Entités**
  - Chaque Système a une liste **d'identifiants d'Entité** l'intéressant
  - Les Systèmes enregistrent leurs bitmaps auprès du gestionnaire d'Entités
  - Quand une entité est ajoutée :
    - Evaluer quels Systèmes sont intéressés et mettre à jour leur liste d'identifiants

# Comment un système trouve ses entités ?

- **Gestionnaire d'Entités**
  - Exemple de code

```
arraylist entityList;

bitmap renderSysBM = '1010';

function EntityManager.addEntity(entity) {
    if (entity.componentBitMap & renderSysBM == renderSysBM) {
        RenderSystem.addEntity(entity.ID)
    }
    ...
    entityList.add(entity);
}
```

# Comment un système trouve ses entités ?

- **Gestionnaire d'Entités**
  - Composant est ajouté/supprimé d'une Entité
    - Ré-évaluer quels systèmes sont intéressés et mettre à jour leur liste d'identifiants

```
function EntityManager.reevaluate(entity) {
    if (entity.componentBitMap & renderSysBM == renderSysBM) {
        if (!RenderSystem.contains(entity))
            RenderSystem.addEntity(entity.ID);
    }
    else if (RenderSystem.contains(entity))
        RenderSystem.removeEntity(entity.ID);
    ...
}
```

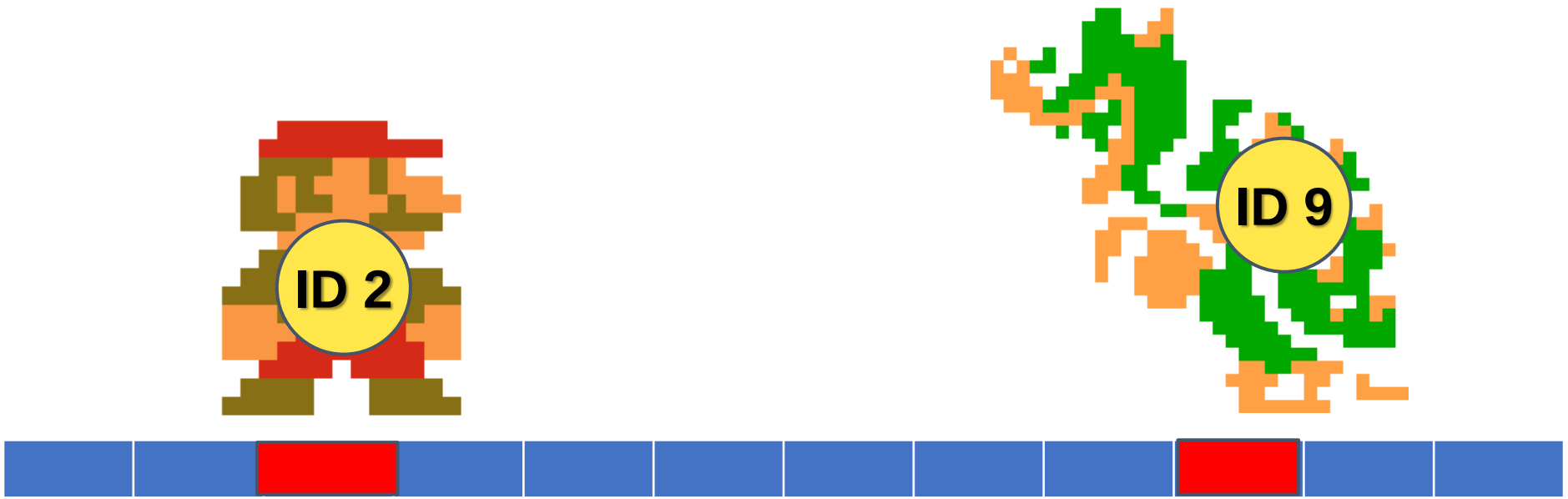
# Comment un système trouve ses entités ?

- Gestionnaire d'Entités
  - Les Systèmes doivent seulement boucler sur leur liste d'identifiants

```
arraylist entityIDs;

function MotionSystem.update() {
    foreach (id in entityIDs) {
        EntityManager.positionComponents[id] += EntityManager.velocityComponents[id];
    }
}
```

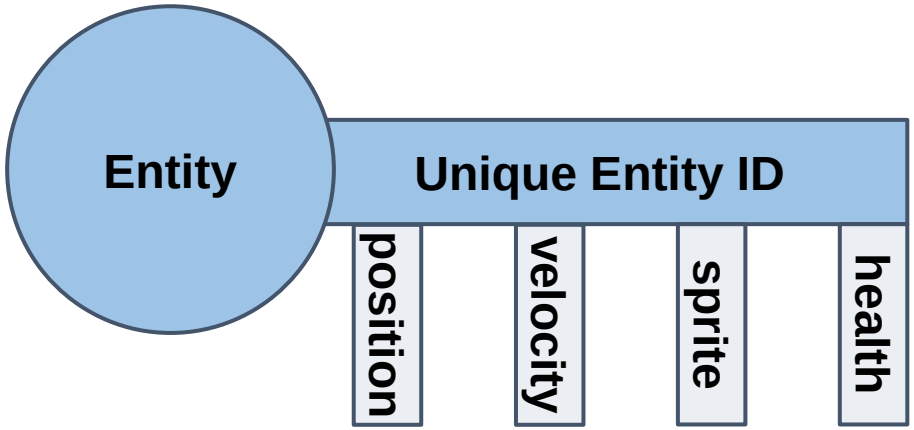
# Entité

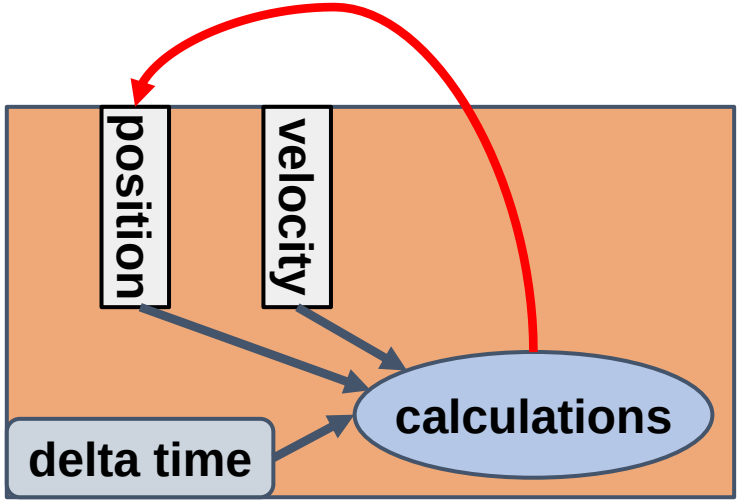
- Chaque Entité est typiquement juste un **identifiant unique** pour ses **Composants**
- Stocker les Entités dans une liste statique dans le gestionnaire d'Entité




Entités

# Métaphore de la clé et la serrure




Les Systèmes opèreront seulement sur les Entités avec les Composants requis

**Système de déplacement**

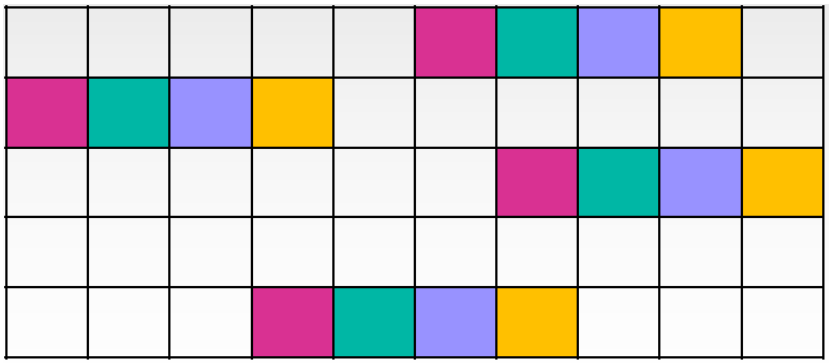
# Mémoire et ECS

- Où stocke-t-on les composants ?
  - Dans les Entités ?

```
function MotionSystem.update() {
    foreach (entity with position and velocity) {
        entity.getPosition() += entity.getVelocity();
    }
}
```

La boucle de mise à jour accède fréquemment à des parties non-contigues en mémoire

**Gestion de la mémoire inefficace**



Blocks mémoire

# Mémoire et ECS

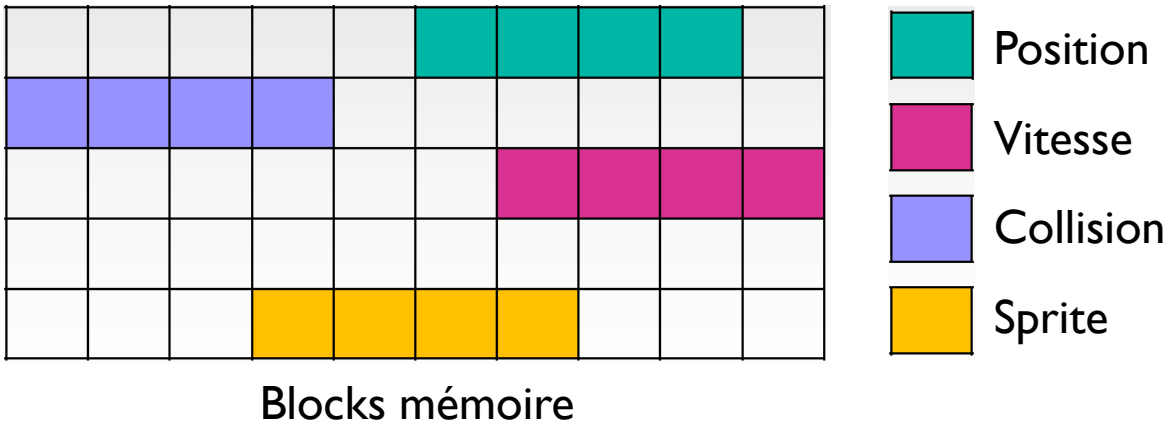
- **Où stocke-t-on les composants ?**
  - Dans les Systèmes ?
    - Mieux mais pourrait être amélioré
    - Différents Systèmes peuvent avoir besoin **des mêmes types** de Composants
      - Comment décide-t-on **qui possède quoi** ?
      - La communication peut se compliquer entre les Systèmes

# Mémoire et ECS

- **Où stocke-t-on les composants ?**
  - Dans le gestionnaire d'Entités ?
    - Systèmes ne possèdent pas de Composants
    - Une grande liste pour chaque type de Composants
    - Profite de l'architecture modulaire de l'ECS

# Cache est la clé

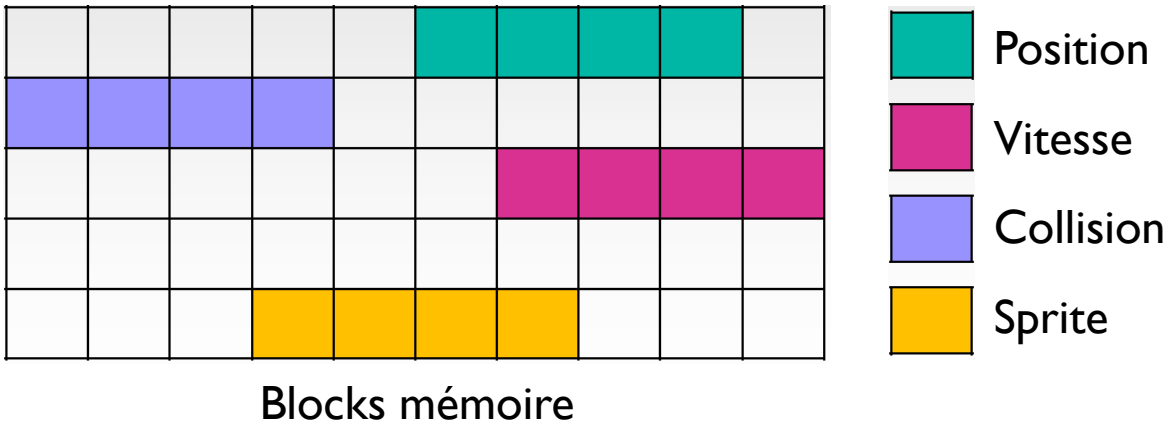
- Chaque type de Composant a une liste **statiquement** allouée
- Minimise les ratés de cache
  - Garde les Composant auxquels ont accèdera en même temps **proches en mémoire**



# Cache est la clé

```
function MotionSystem.update() {
    foreach (entity with position and velocity) {
        entity.getPosition() += entity.getVelocity();
    }
}
```

La boucle de mise à jour accède à des parties contigues en mémoire




**Idéal**

# Cache est la clé

- Quand on **« supprime »** une Entité il faut supprimer les **Composants correspondants**
- Approches :
  - Remplir avec les données des **dernières entités de la liste**
    - Faire très attention à la gestion d'indices
  - Marquer des parties de la liste comme **réinscriptibles**

# Avantages de l'ECS

- Complexité
  - Le code des jeux tend à **augmenter** de façon exponentielle
  - La complexité de l'ECS n'augmente pas avec
  - **Facile à maintenir**
- Customisation
  - Les jeux nécessitent beaucoup d'opérations **dynamiques**
  - **Ajouter/supprimer des Composants** pour changer les comportements des Entités
  - ECS est **très modulaire**
- Peut être efficace coté gestion mémoire