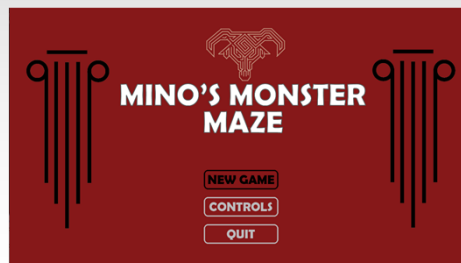


# CPSC 427

## Video Game Programming



### Entity Component System

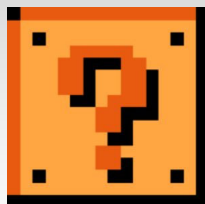


© Alla Sheffer

## What are Entities?



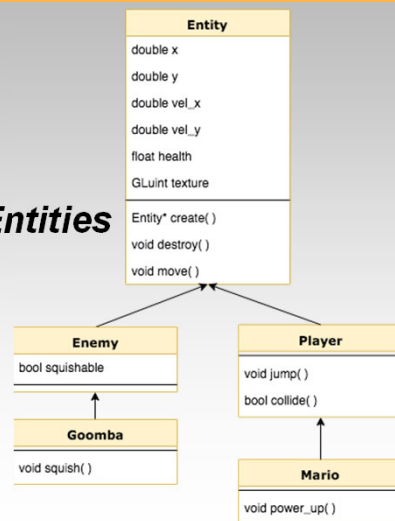
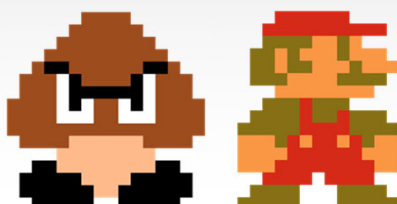
- **Entities:** things that exist in your game world



© Alla Sheffer

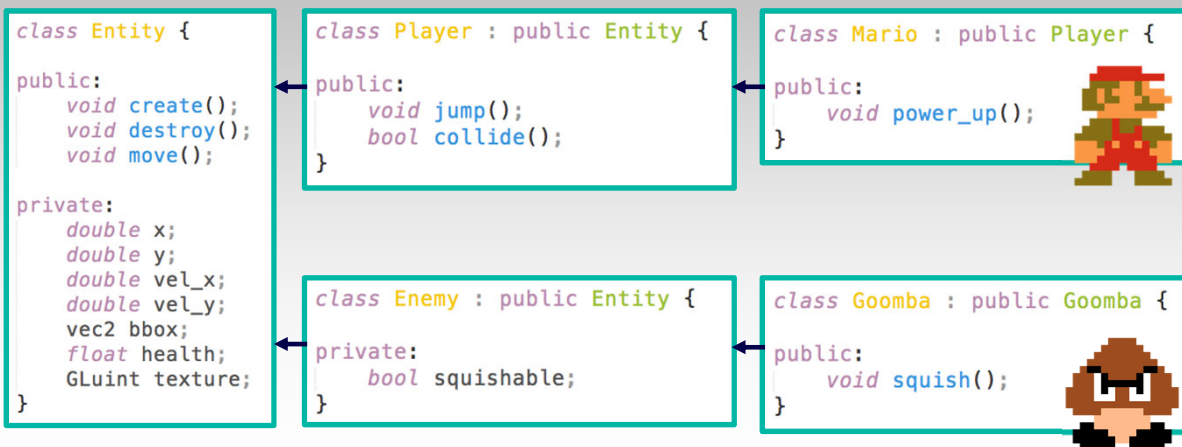
# Entities in Traditional Game Programming

- **Object-Oriented Programming**
  - **Entities as objects**
    - ▶ Contains data, behaviors, etc.
  - **Entity Hierarchy: Entities extend other Entities**



© Alla Sheffer

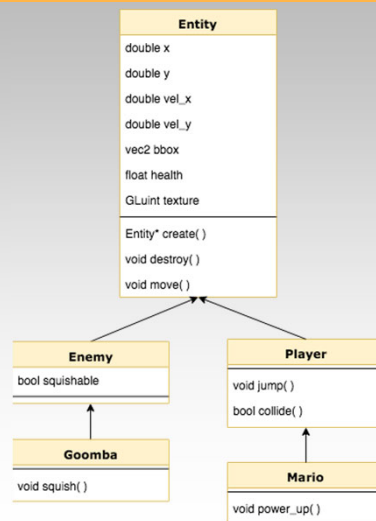
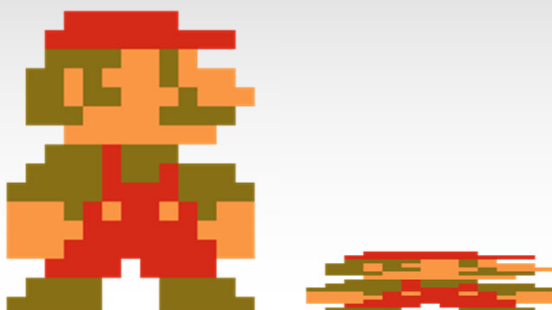
# Entity Hierarchy



© Alla Sheffer

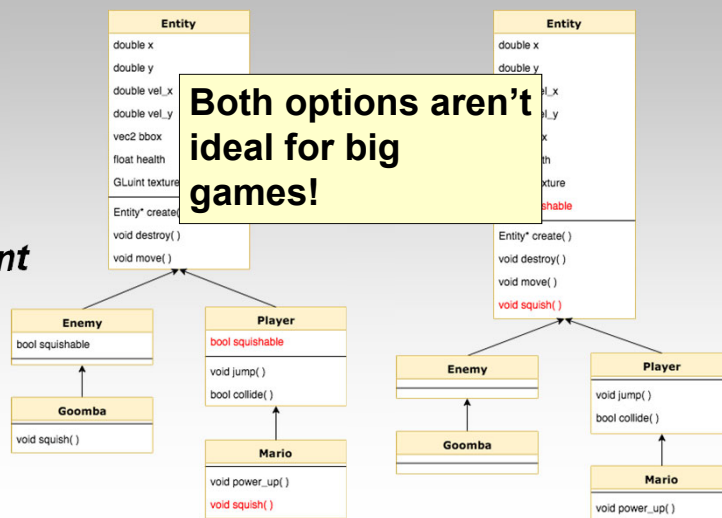
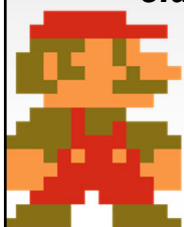
# Issues with Object-Oriented Approach

What if we want Mario to be able to be squished?



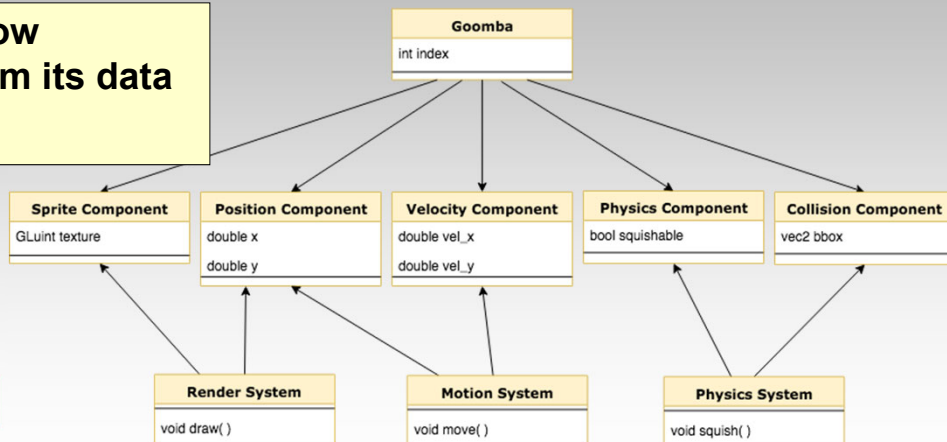
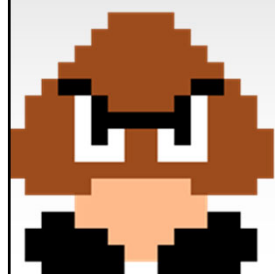
# Issues with Object-Oriented Approach

- Difficult to add **new** behaviors
  - Choice between **replicating code** or
  - **MONSTER SIZE** parent classes



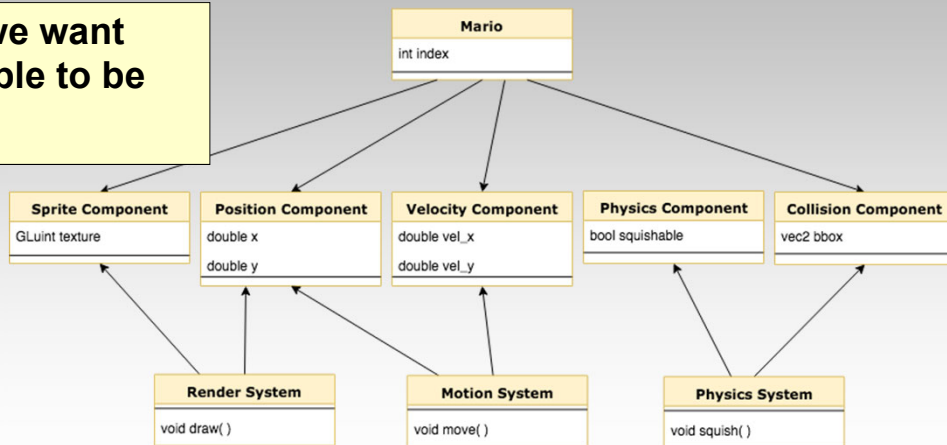
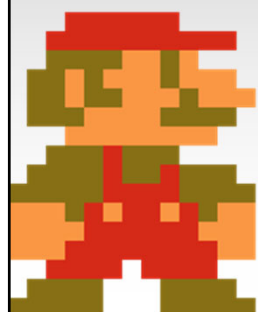
## Example ECS Diagram

Goomba is now separated from its data & methods



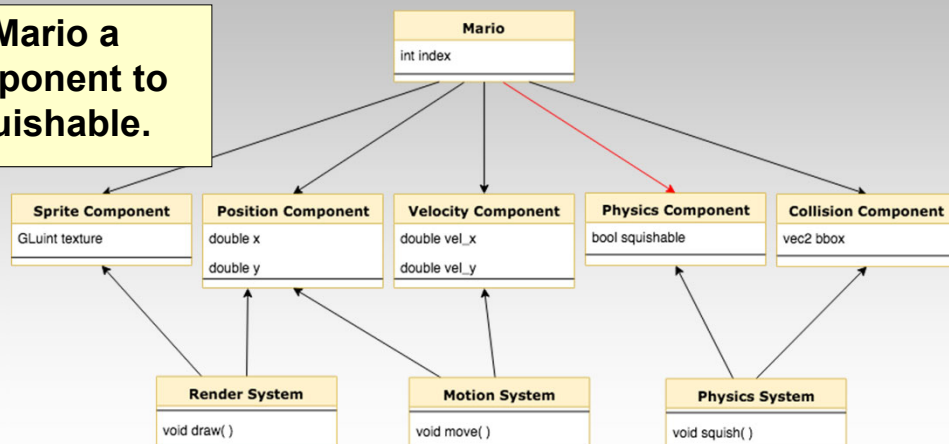
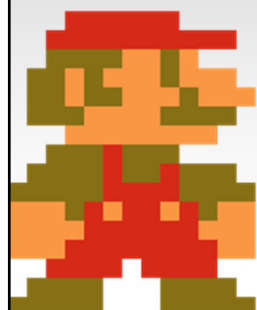
## Example ECS Diagram

Now what if we want Mario to be able to be squished?



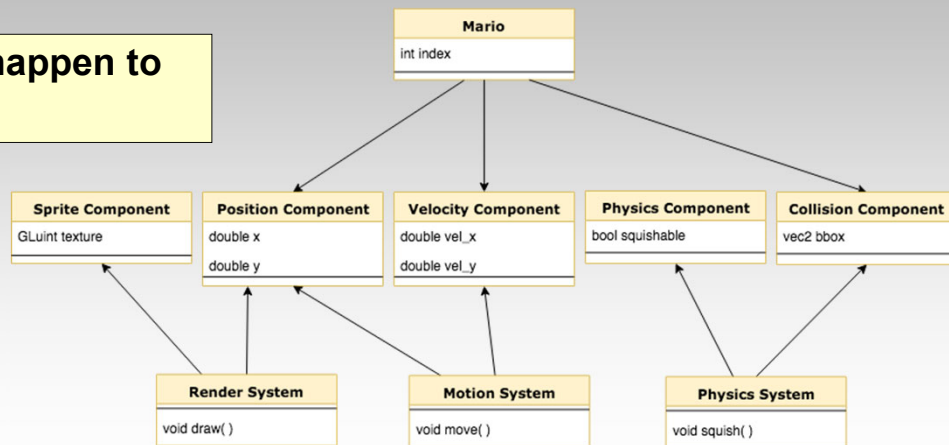
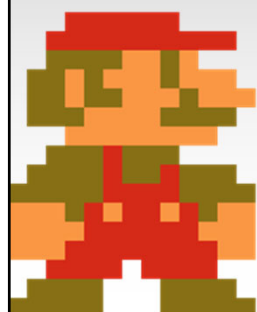
## Example ECS Diagram

We can give Mario a Physics Component to make him squishable.



## Example ECS Diagram

What would happen to Mario here?





## What is ECS?

- Alternative to object-oriented programming
- Data is **self-contained & modular**
  - *Similar concept to building blocks*
  - *Entities no longer “own” data*
  - *Entities pick & choose*

© Alla Sheffer



## What is ECS?

- Entities actions determined **only by their data**
  - *Update loop doesn't need references to Entities*
  - *Systems search for Entities with right parts (data) & update*
    - ▶ For Mario to move he needs a position & velocity

© Alla Sheffer



## What is ECS?

- **Composition** over hierarchy
- **Entities** are collections of **Components**
- **Components** contain **game data**
  - *Position, velocity, input, etc.*
- **Systems** are collections of **actions**
  - *Render system, motion system, etc.*

© Alla Sheffer



## Component

- Contains **only** game data
- Describes **one** aspect of an Entity
  - *ex. a trumpet Entity will likely have an audio Component*

<b>Sprite Component</b> GLuint texture	<b>Position Component</b> double x double y	<b>Velocity Component</b> double vel_x double vel_y	<b>Physics Component</b> bool squishable	<b>Collision Component</b> vec2 bounding_box
<b>Input Component</b> bool left bool right bool jump bool attack	<b>AI Component</b> bool do_left bool do_right bool do_jump bool do_shoot	<b>Health Component</b> float health	<b>Audio Component</b> mp3 sound	

© Alla Sheffer

# Component

- Typically implemented with structs.

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

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

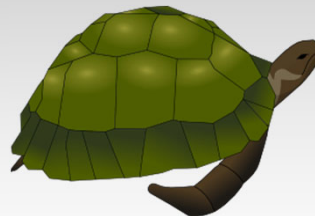
```
struct VelocityComponent {  
    double vel_x;  
    double vel_y;  
}
```

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

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

# What Components to Make?

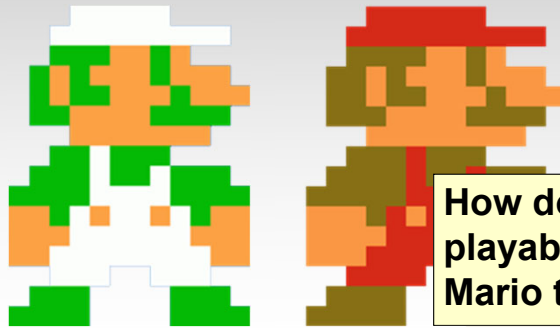
- What Components would we give to the following Entities?





## Components

- Easy to add new Entity characteristics
  - *Just create the desired Component & give to Entity*



How do we change our playable hero from Mario to Luigi?

## Components

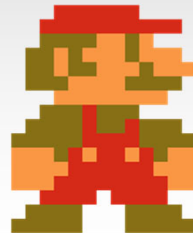
- Empty Components can be used to tag Entities

Input Component
bool left
bool right
bool jump
bool attack

Position Component
double x
double y

Velocity Component
double vel_x
double vel_y

Sprite Component
GLuint texture



Input Component
bool left
bool right
bool jump
bool attack

Sprite Component
GLuint texture

Player Component

Position Component
double x
double y

Velocity Component
double vel_x
double vel_y

## Components

- Empty Components can be used to tag Entities



© Alla Sheffer

## Systems

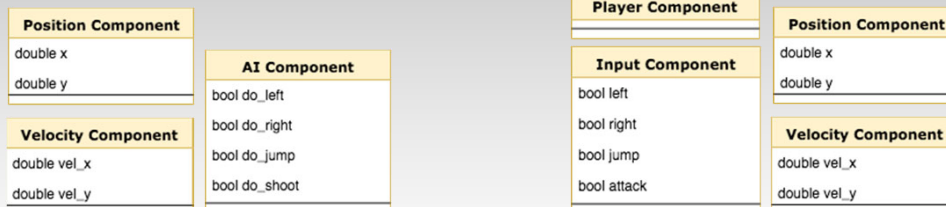
- Groups of Components **describe behavior/action**
  - *ex. bounding box, position & velocity describe collisions*
- Systems code **behaviors/actions**
- Operate on Entities with **related groups of components**
  - *Related: describe **same (type of)** behavior/action*
  - *ex. render all Entities with sprite & position*
- Entity behavior can be **dynamic**
  - *Add/remove components on the fly*

© Alla Sheffer



## System Example

- What systems might these related groups of components describe?

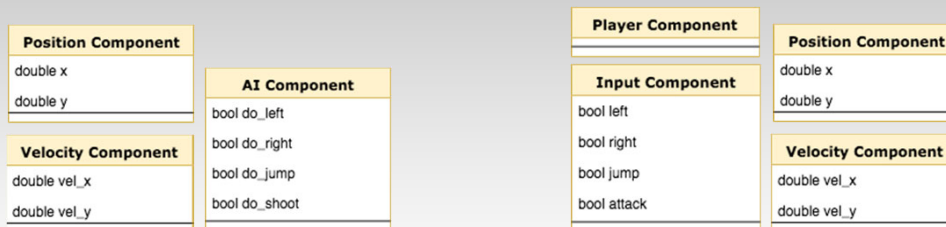


© Alla Sheffer



## System Example

- What systems might these related groups of components describe?



**Enemy Motion System**

**Player Motion System**

© Alla Sheffer



## How Does a System Find its Entities?

### Bitmap Method

- Each Entity has a sequence of bits that represent its Components
- Each System has a sequence of bits that represent the Components it is interested in

© Alla Sheffer



## How Does a System Find its Entities?

### Bitmap Method

- Code example

```
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  
            );  
        }  
    }  
}
```

© Alla Sheffer



## How Does a System Find its Entities?

### Bitmap Method

- This can be **costly** if we do it for every update loop

```
Can we do better?  
function process(entityList) {  
  // Loops through all entities  
  foreach(entity in entityList) {  
    // Let's say that position and sprite 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  
      );  
    }  
  }  
}
```

**YES!**

© Alla Sheffer



## How Does a System Find its Entities?

### Entity Manager

- Each system has a list of **entity IDs** it is interested in
- Systems register their bitmaps with the Entity Manager
- Whenever an Entity is added...
  - *Evaluate which systems are interested & update their ID lists*

© Alla Sheffer



## How Does a System Find its Entities?

### Entity Manager

- Code example

```
arraylist entityList;

bitmap renderSysBM = '1010';

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

© Alla Sheffer



## How Does a System Find its Entities?

### Entity Manager

- Whenever a Component is added/removed from an Entity...
  - *Re-evaluate which systems are interested & update ID lists*

```
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);
    ...
}
```

© Alla Sheffer

## How Does a System Find its Entities?

### Entity Manager

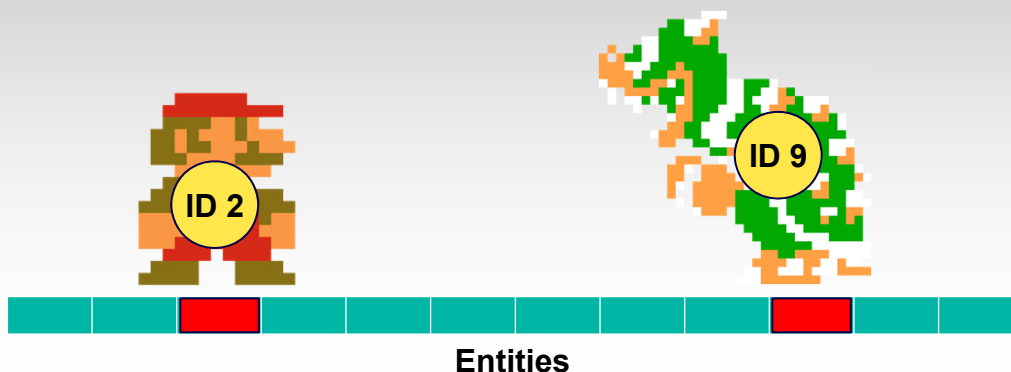
- Now Systems only need to loop through their ID lists

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

© Alla Sheffer

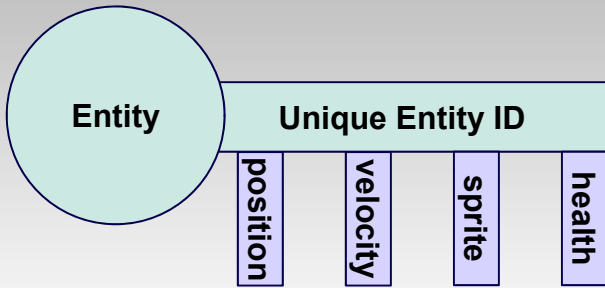
## Entity Overview

- Each Entity is typically just a **unique identifier** to its **components**
- Store Entities in a big static array in the Entity Manager

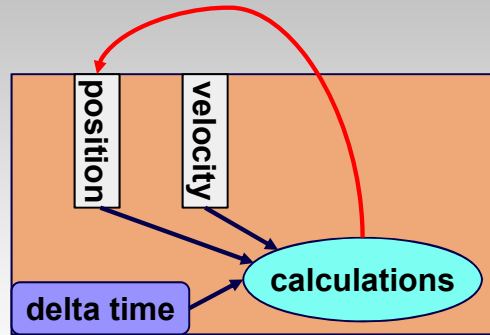


© Alla Sheffer

# Key & Lock Metaphor



Systems will only operate on Entities with the required Components



**Motion System**

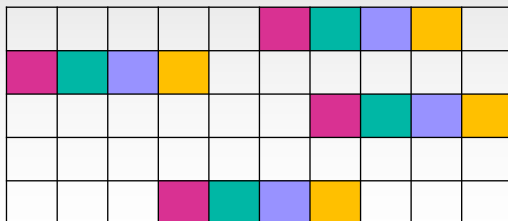
# Memory & ECS

## Where do we store our Components?

- Inside Entities?

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

Update loop has to access non-contiguous memory repeatedly!



- teal position
- pink velocity
- purple collision
- yellow sprite

**Not memory efficient!**

Memory Blocks





## Memory & ECS

### Where do we store our Components?

- Inside Systems?
  - *Better, but could be improved*
  - *Different Systems may need the **same** Component types*
    - ▶ How do we decide **who owns what**?
    - ▶ Messaging can get overly complex between systems

© Alla Sheffer



## Memory & ECS

### Where do we store our Components?

- Inside the Entity Manager?
  - *Systems don't own components*
  - *One big array for each Component type*
  - *Takes advantage of modular architecture of ECS*

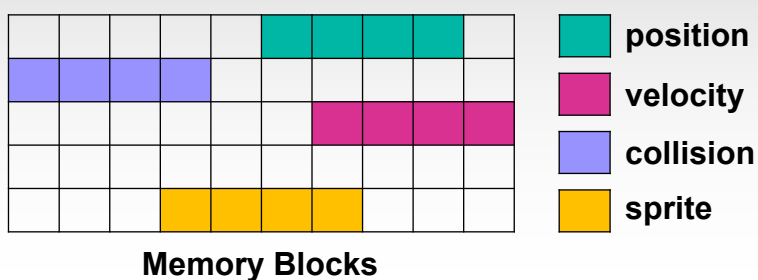
**YES!**

© Alla Sheffer



## Cache is Key

- Each Component type has a **statically** allocated array
- Minimizes costly cache misses
  - *Keeps components we access around the same time **close to each other***



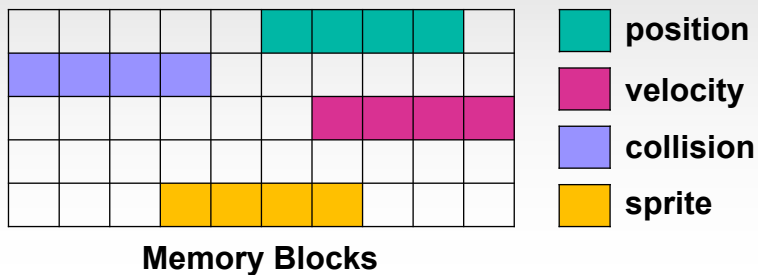
© Alla Sheffer



## Cache is Key

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

Update loop  
accesses contiguous  
memory



**IDEAL!**

© Alla Sheffer



## Cache is Key

- When we “delete” an entity we must delete **corresponding components** to.
- Different approaches to this,
  - *Fill deleted components in arrays with the **last entities data***
    - ▶ Extra care must be taken when managing indices
  - *Mark spots in arrays as **rewritable***
    - ▶ Big systems will suffer from poor memory management

© Alla Sheffer



## Entity Component Systems: Benefits

- Complexity
  - *Game code tends to **grow** exponentially*
  - *Complexity of ECS architecture does not grow with it*
  - **Easy to maintain**
- Customization
  - Games have a lot of **dynamic** operations
  - ***Add/remove components** to change Entity behavior*
  - *ECS is **highly modular***
- Can be very memory efficient!

© Alla Sheffer