CPSC 427
Video Game Programming

Game Programming Basics: Event Driven Programming & Entity Component System (ECS)

Procedural Programming

Sequential control flow

- Program performs a sequence of tasks & terminates

Event-Driven Programming

Main loop not under your control

- vs. procedural

Control flow through event callbacks

- redraw the window now
- key was pressed
- mouse moved

Callback functions called from main loop when events occur

- mouse/keyboard

Minimal Main (openGL)

```
int main(int argc, char* argv[]) {
  if (!world.init(..)){
    return EXIT_FAILURE;
  }
  while (!world.is_over()) {
      glfwPollEvents(); // process events
      world.update(); // update game state based on events + timer
      world.draw(); // render
  }
  world.destroy();
  return EXIT_SUCCESS;
}
```

Update (responds to user input + timer)

1. ***Collision detection***
2. ***Game AI***
3. ***Physics***
   - Update positions, velocities, etc
   - Resolve collisions

5

Our game loop (A1-A3 Template, main.cpp)

```
// Entry point
int main()
{
    // Global systems
    WorldSystem world;
    RenderSystem renderer;
    PhysicsSystem physics;
    AISystem ai;

    // Initializing window
    GLFWwindow* window = world.create_window();
    if (!window) {
        // Time to read the error message
        printf("Press any key to exit");
        getchar();
        return EXIT_FAILURE;
    }

    // initialize the main systems
    renderer.init(window);
    world.init(&renderer);
```

```
    // variable timestep loop
    auto t = Clock::now();
    while (!world.is_over()) {
        // Processes system messages, if this wasn't present the window would become unresponsive
        glfwPollEvents();

        // Calculating elapsed times in milliseconds from the previous iteration
        auto now = Clock::now();
        float elapsed_ms =
            (float)(std::chrono::duration_cast<std::chrono::microseconds>(now - t)).count() / 1000;
        t = now;

        world.step(elapsed_ms);
        ai.step(elapsed_ms);
        physics.step(elapsed_ms);
        world.handle_collisions();

        renderer.draw();

        // TODO A2: you can implement the debug freeze here but other places are possible too.
    }

    return EXIT_SUCCESS;
}
```

6

openGL

- Low-level graphics API
- C Interface accessed from C++
- Shaders – graphics
  - *A LOT more details later*

Even Callbacks

Set at start – in our template in world.init()

```
auto key_redirect = [](GLFWwindow* wnd, int _0, int _1, int _2, int _3) {
    ((World*)glfwGetWindowUserPointer(wnd))->on_key(wnd, _0, _1, _2, _3); };
auto cursor_pos_redirect = [](GLFWwindow* wnd, double _0, double _1) {
    ((World*)glfwGetWindowUserPointer(wnd))->on_mouse_move(wnd, _0, _1); };
glfwSetKeyCallback(m_window, key_redirect);
glfwSetCursorPosCallback(m_window, cursor_pos_redirect);
```

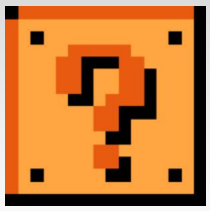
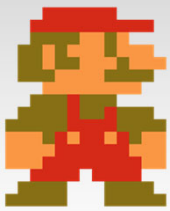
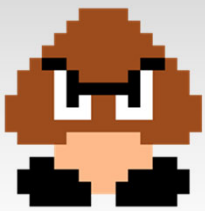
Another example would be a mouse click (same format)

Callback Actions

```
void World::on_key(GLFWwindow*, int key, int, int action, int mod){
if (action == GLFW_RELEASE && key == GLFW_KEY_R){
    …
}
if (action == GLFW_RELEASE && (mod & GLFW_MOD_SHIFT) &&  key ==
GLFW_KEY_COMMA){
 …
}
void World::on_mouse_move(GLFWwindow* window, double xpos, double
ypos){
}
```


What are Entities?

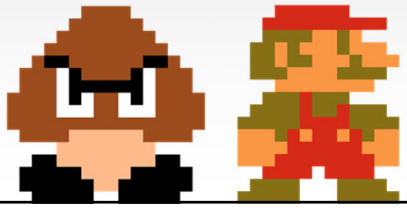
- **Entities:** things that exist in your game world

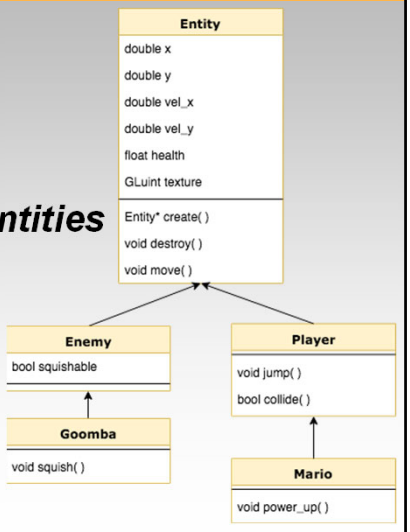





1

Entities in Traditional Game Programming

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





Entity Hierarchy (object oriented design)

```
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();
}
```

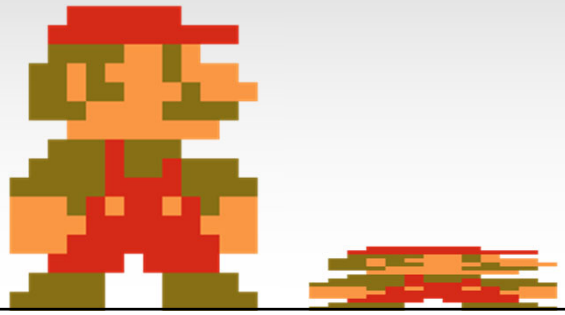
```
class Mario : public Player {

public:
    void power_up();
}
```

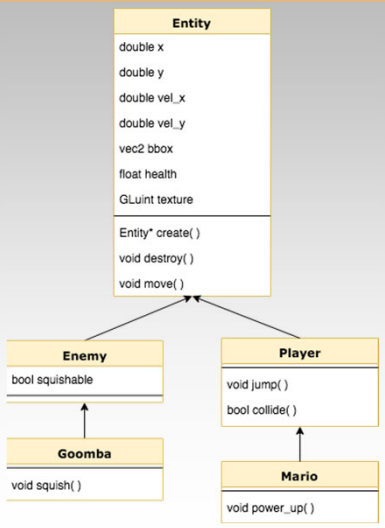
```
class Enemy : public Entity {

private:
    bool squishable;
}
```

```
class Goomba : public Goomba {

public:
    void squish();
}
```

Issues with Object-Oriented Approach

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

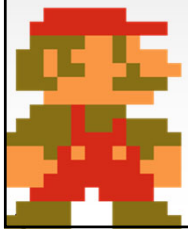


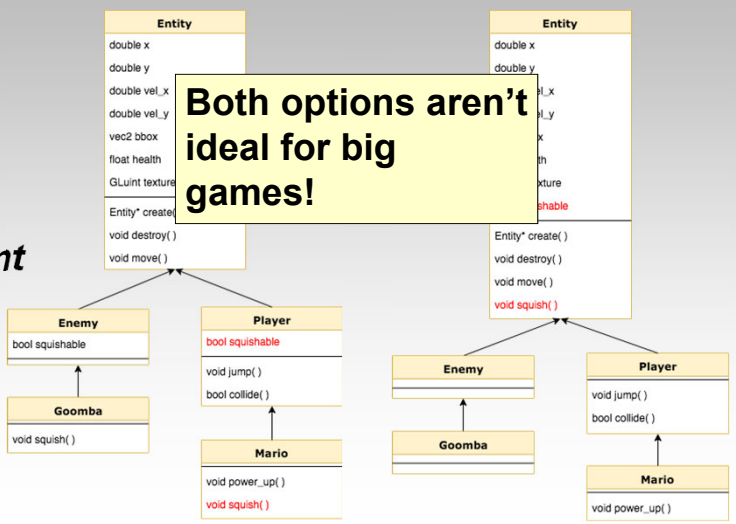


Issues with Object-Oriented Approach

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

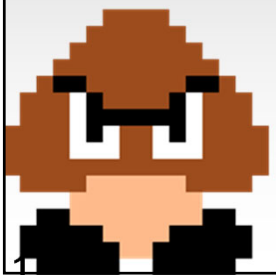
Both options aren't ideal for big games!

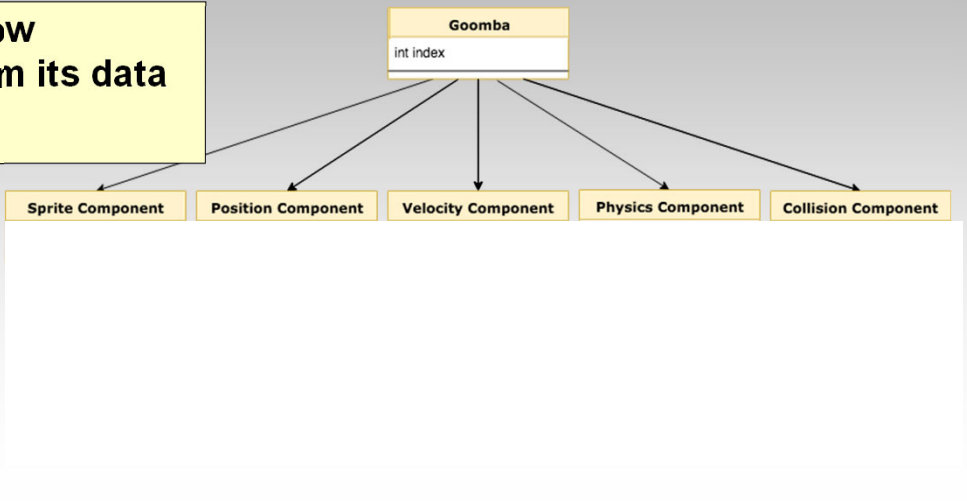




Example ECS Diagram

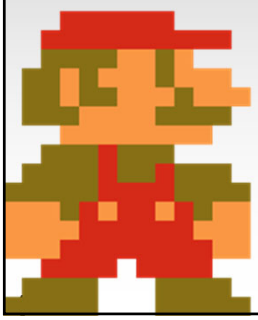
Goomba is now separated from its data & methods

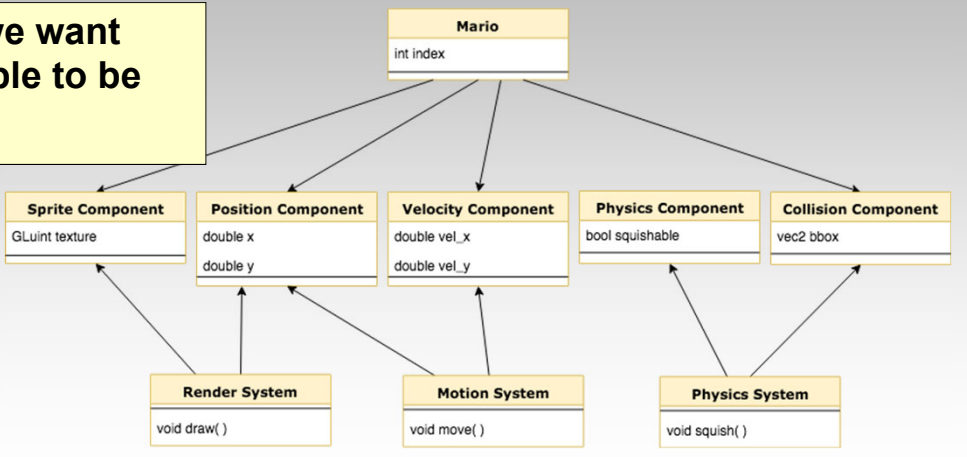




Example ECS Diagram

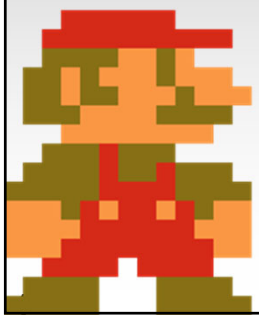
Now what if we want Mario to be able to be squished?

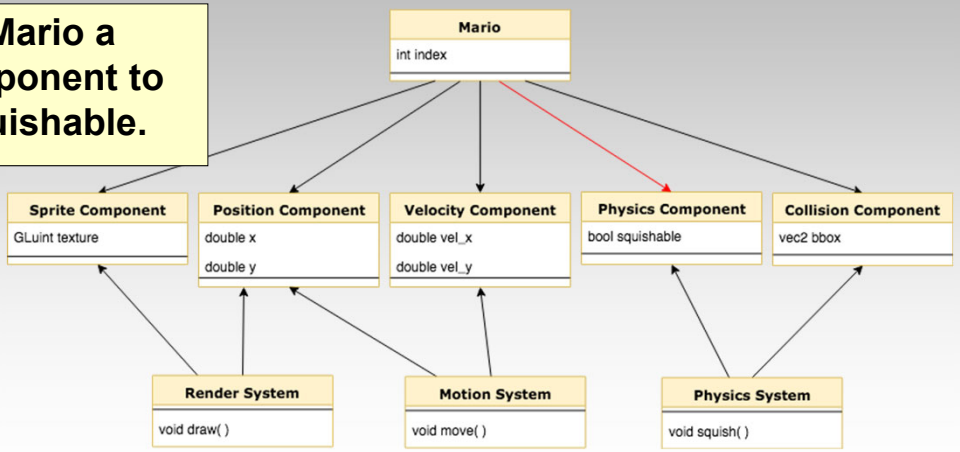




Example ECS Diagram

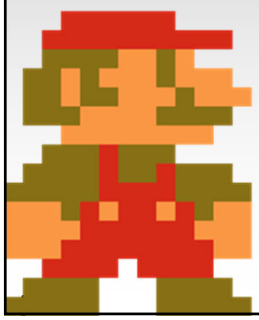
We can give Mario a Physics Component to make him squishable.

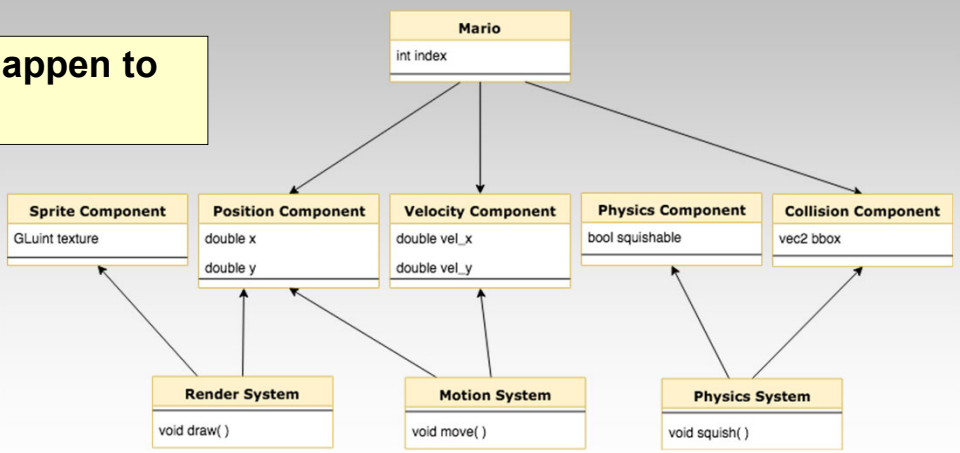




Example ECS Diagram

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*

1

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

2

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.*

Component

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

Sprite Component
GLuint texture

Position Component
double x
double y

Velocity Component
double vel_x
double vel_y

Physics Component
bool squishable

Collision Component
vec2 bounding_box

Input Component
bool left
bool right
bool jump
bool attack

AI Component
bool do_left
bool do_right
bool do_jump
bool do_shoot

Health Component
float health

Audio Component
mp3 sound

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;
}
```

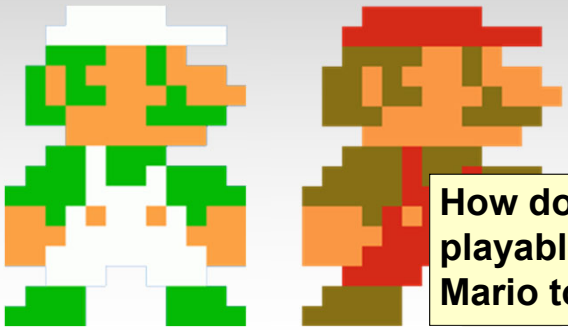
2

What Components to Make?

- What Components would we give to the following Entities?

Components

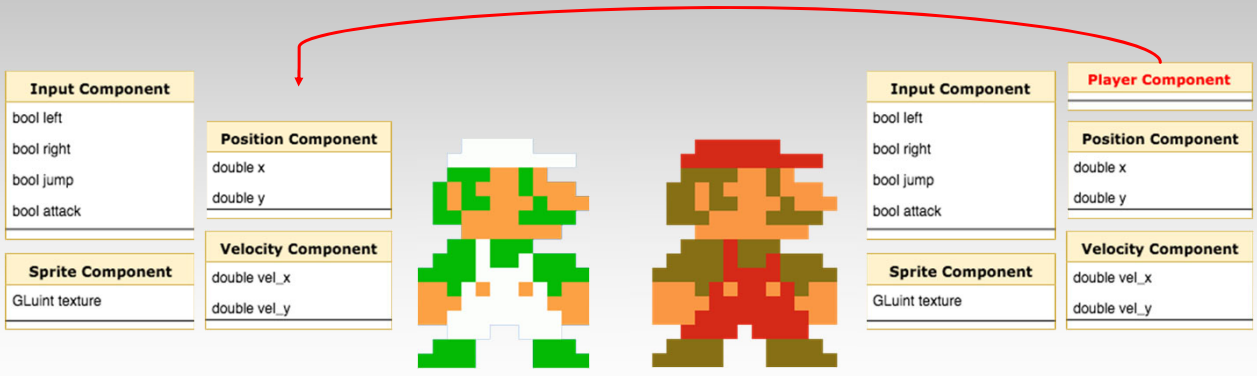
- Easy to add new Entity characteristics
  - *Just create the desired Component & give to Entity*



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

Components

- Empty Components can be used to tag Entities



Empty components are useful, a flag indicating an ability!

Components

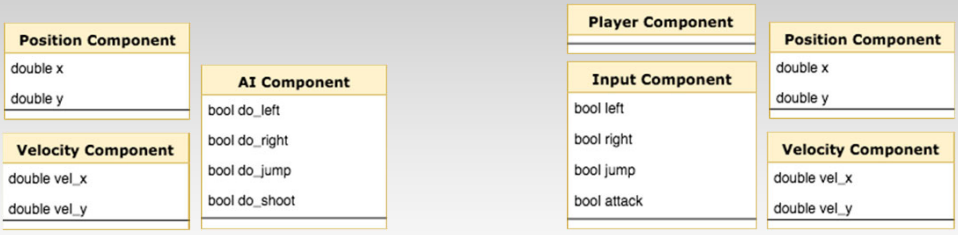
- Empty Components can be used to tag Entities

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*

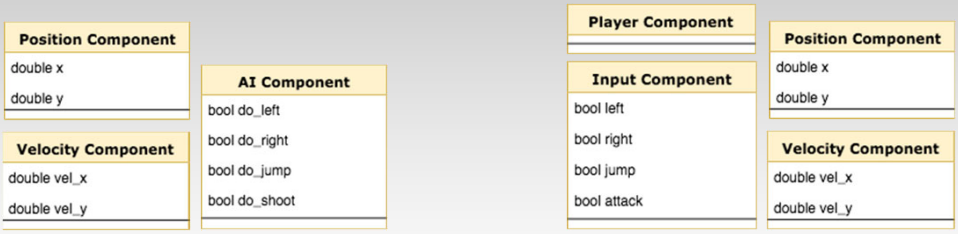
System Example

- What systems might these related groups of components describe?



System Example

- What systems might these related groups of components describe?



Enemy Motion System **Player Motion System**

System Examples

Physics System … iterates over all components of type velocity

```
for(Velocity& velocity : velocity_components)
   velocity += 9.81 * dt
```

The physics system does not care about entities at all!

Game loop

```
Entity player;
   if(! alive_entities.has(player) ) exit();
```

Single boolean check

Motion System … iterates over all entities that have velocity and position

```
for(int entity : velocity_entities)
   if (position_entities.has(entity))
         position_components.get(entity)+= velocity_components.get(entity);
```

Need to know all entities that have component X
Need to retrieve a component X from an entity

3

ECS implementation: So how do we code this?

3

Where/How do we store components?

- **Inside Entities?**

	Position	Velocity	Jumps	Player	Squishable
Mario	☐	☐	☐	☐	
Goomba1	☐	☐			☐
Luigi	☐	☐	☐		
Goomba2	☐	☐			☐

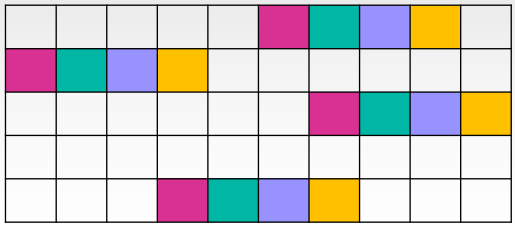
3

Where/How do we store components?

- **Inside Entities?**
  - ***NO!***

Slow memory access!

Update loop has to access non-contiguous memory repeatedly!



Memory Blocks

position
velocity
collision
sprite

3

Where/How do we store components?

- Inside Systems?
  - *NO!*
    - Component may be used by different systems

3

Where/How do we store components?

Where do we store our Components?

- ***Inside Component Managers!!***
  - *Entity doesn't need to know what components exist.*
  - *Easy to add new components.*
  - *Components are encapsulated.*

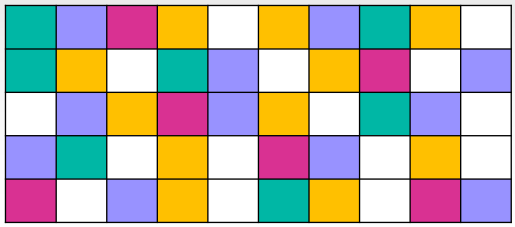
- Okay, good. How?

Where/How do we store components?

- In a map (hash table)?
  - *NO!*

Slow memory access!

Update loop STILL has to access non-contiguous memory repeatedly!

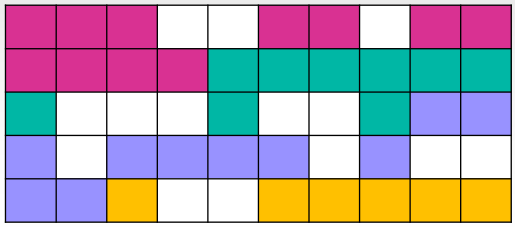


position
velocity
collision
sprite

Memory Blocks

3

The (giant) Sparse Array (MATRIX)

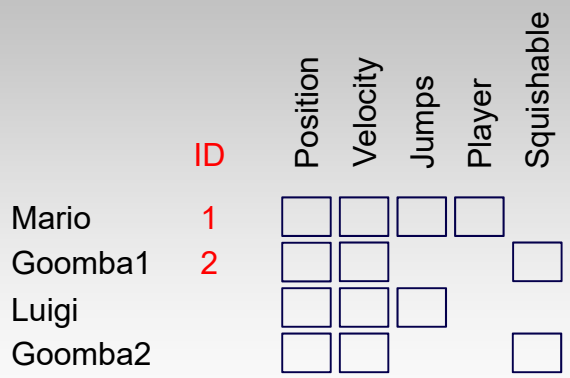


position
velocity
collision
sprite

Memory Blocks

3

The (giant) Sparse Array (MATRIX)

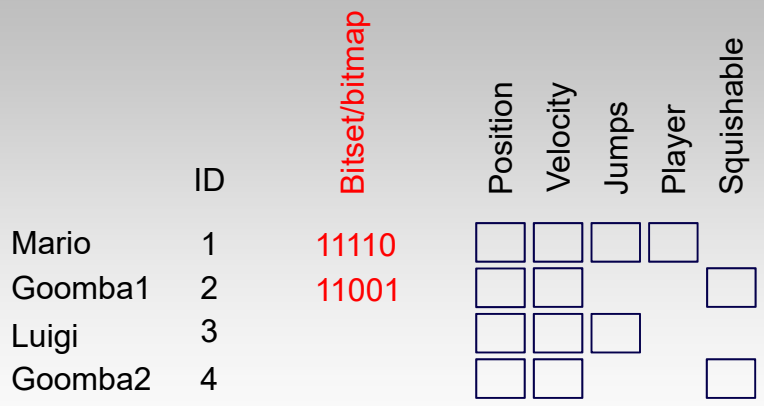


Issues?

Concept: A huge data matrix of size Nr. Entities x Nr. components
Implementation: std:vector<Position>; std:vector<Velocity>

3

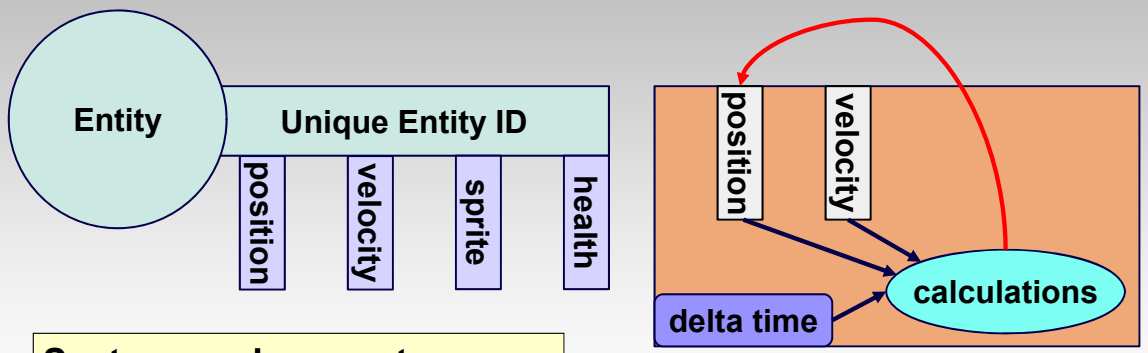
Bitset / Bitmap



Concept: Each entity has a bitset that is true for its 'owned' components
Implementation: long bitset; // how many components can we support?
If(bitset & query == query) // has the entity all query components?

4

Key & Lock Metaphor

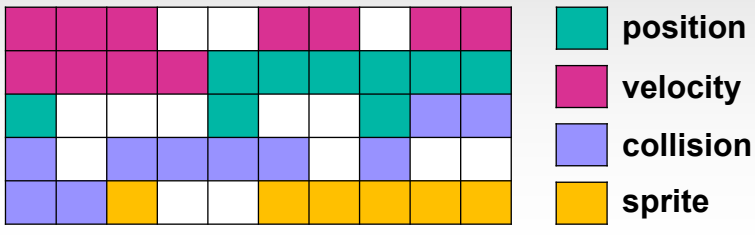


Systems only operate on Entities with required Components

Motion System

The (giant) Sparse Array (MATRIX)

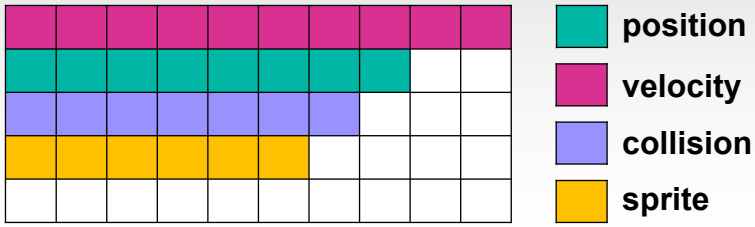
- Good: O(1) map from entity to component
- Bad: Large Holes
  - *not memory efficient; fragmentation, cache utilization is wasted on empty space ☹*



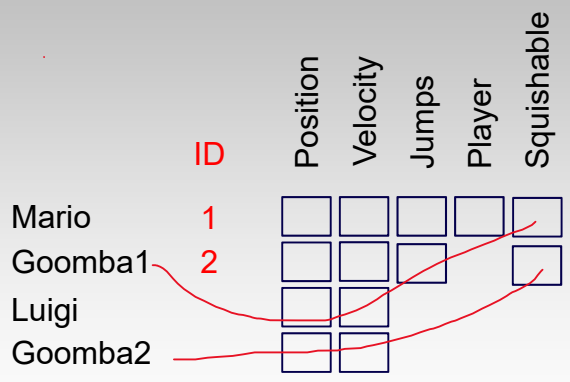
Dense Component Vectors

How do we store our Components?

- Dense Component Vectors **!**
  - ***Less fragmentation; good cache usage*** ☺



Dense Component Vectors

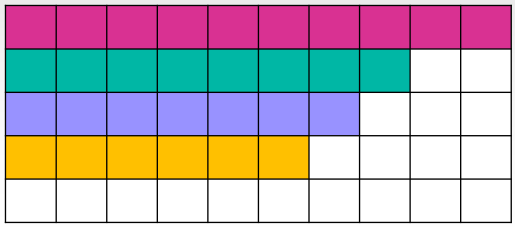


Concept: One array/vector per component, **Implementation:** std:vector<Position>; std:vector<Velocity>

Dense Component Vectors

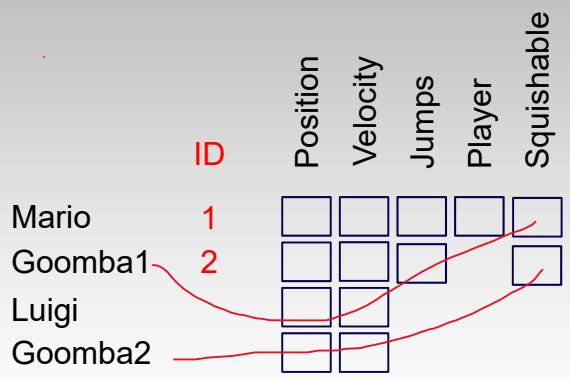
How do we store our Components?

- Dense Component Vectors**!**
  - ***Less fragmentation; good cache usage*** ☺
  - ***New problem: How to find out which entities have which components, and where?***



Memory Blocks

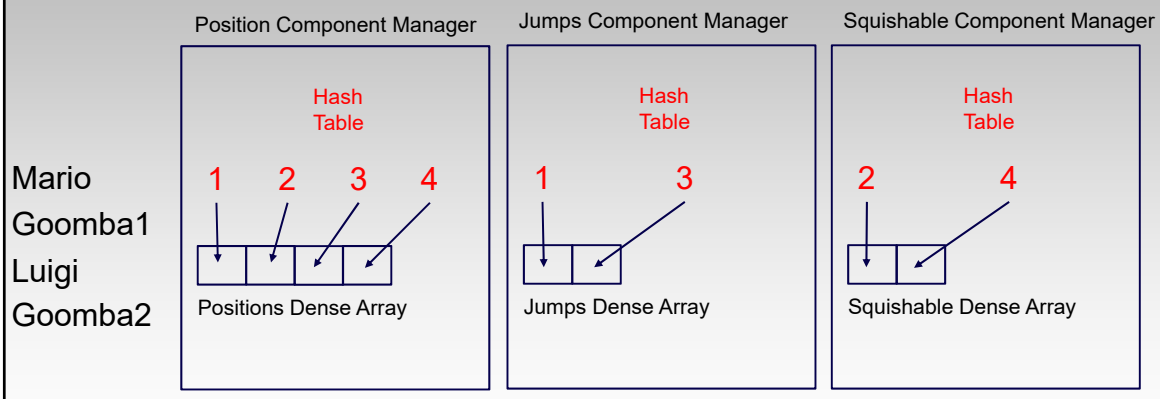
Dense Component Vectors



Need to:
- *Find position of Goomba's squishable component in component mgr*
- *Find parent entity of components*
- *Find out if an object is squishable*

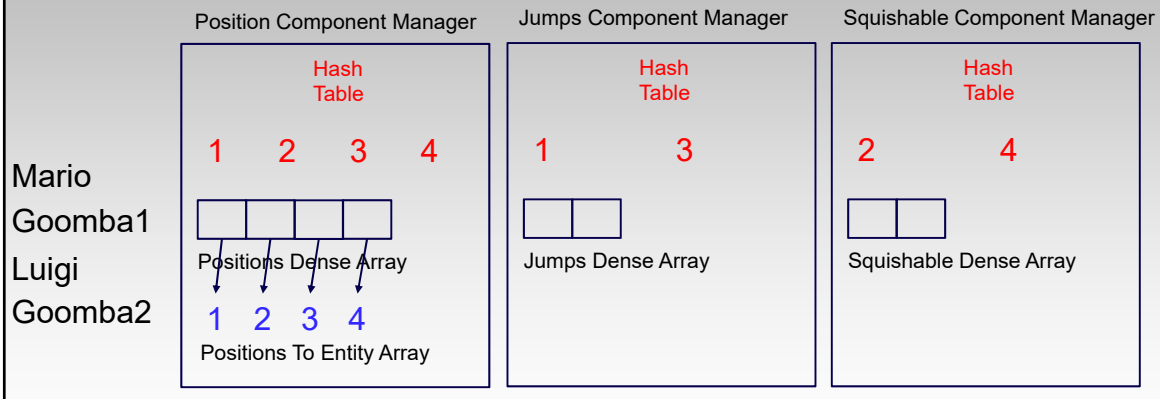
Concept: One array/vector per component, but how to associate?
Implementation: std:vector<Position>; std:vector<Velocity> + WHAT?

Option 1: Hash Tables + Component Managers



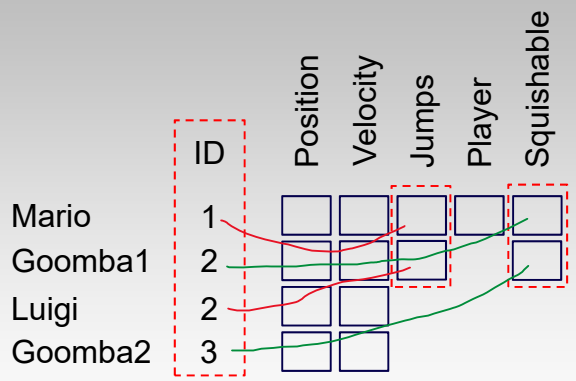
Concept: Use a hash table to look up, for each entity, its position in dense array
Implementation: std::unordered_map<Entity,int>; std::vector<Position>

Option 2: Hash Tables++



Concept: Add reference, for each component, to parent entity
Implementation: std::vector<Entity>

Map + Dense Vector (different visualization)

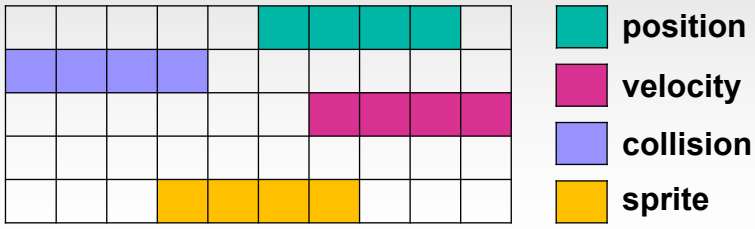


How Do I…

- **Find all my entity's components?**
  - ***Have registry of component managers; check all hash tables***
  - ***O(n) in # of component managers***
- **Find if an entity has a component and where?**
  - ***Check hash table; returns index or NULL***
  - ***O(1) look up on average***
- **Find if a component has a parent entity?**
  - ***Just look it up. O(1)***

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*



Memory Blocks

Faster iteration via entity and component array

Accessing the velocity map (`unjbyharflw|1p ds`) is an unnecessary indirection

```
iru+bwhqwlw|##yharflw|bhqwlwv,#2#niilfhqw
   li+srvlwrqbhqwlw|bp ds1kdv+hqwlw|,,#2#qhiilfhqw#brrnxs
         srvlwrqbhqwlw|bp dsljhw+hqwlw|,.@#yharflw|bhqwlw|bp dsljhw+hqwlw|,*#2#5{#qhiilfhqw#brrnxs
```

We can access the velocity components in linear fashion

```
iru+bwyhdo1@#3>yhdo1?#yharflw|bhqwlwv1vljh+,*yhdo1..,#2#niilfhqw
  Hqwlw|hqwlw|##yharflw|bhqwlwv^yhdo1`#2#niilfhqw
  bwsrvo1@#srvlwrqbhqwlw|bp dsljhwTqgh{+hqwlw|,*#2#qhiilfhqw#brrnxs
  li+srvo1,
        srvlwrqbfrp srqhqw^srvo1`.@#hjbyharflw|bfrp srqhqw^yhdo1`#2#niilfhqw
```

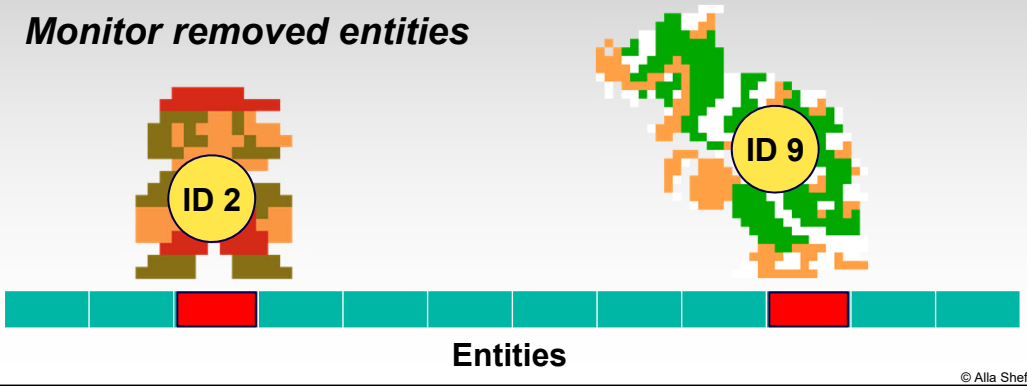
How Does a System Find its Entities?

Extension: Entity Manager

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

Entity Summary

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



Memory & ECS

Where do we store our Components?

- **Inside a registry!**
  - ***Systems don't own components***
  - ***One big array for each Component type***
  - ***Takes advantage of modular architecture of ECS***

YES!

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

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!

5