CPSC 427
Video Game Programming

Entity Component System (ECS)

PART 2

ECS is used in Minecraft and many other commercial games

What will you be doing next week?

Office hours

Monday 10-11 am, ICCSX141, Camilo

Wednesday 9:30 – 10:30, X653 & zoom, Helge

Thursday 2-3 pm, zoom (same as lecture), Tim or Andrew

This week

Monday:

- ***Guest lecture***
- ***more on ECS***

Wednesday

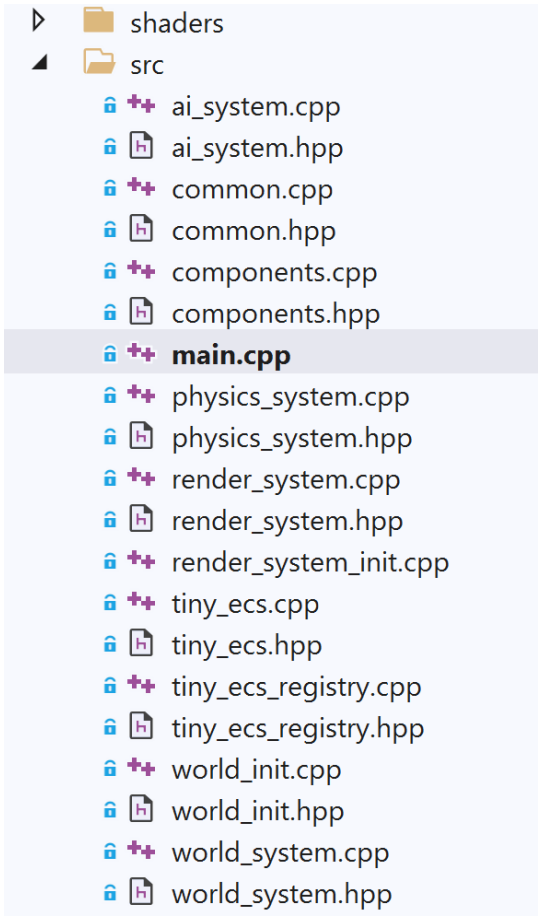
- ***Oral pitches***
- ***C++ Tutorial***

A0 deadline (Wednesday)

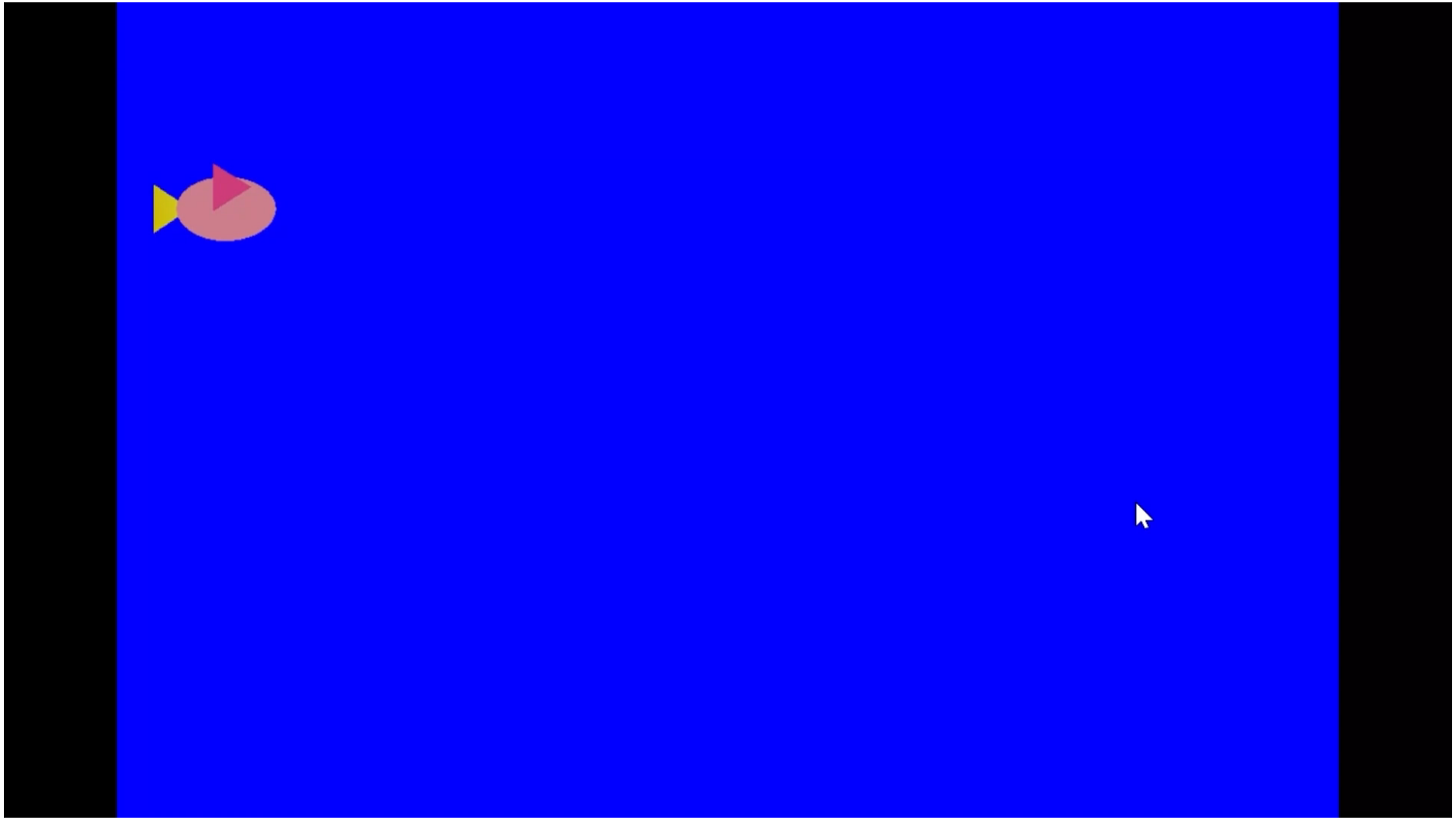
Written pitch deadline (Friday)

Assignments

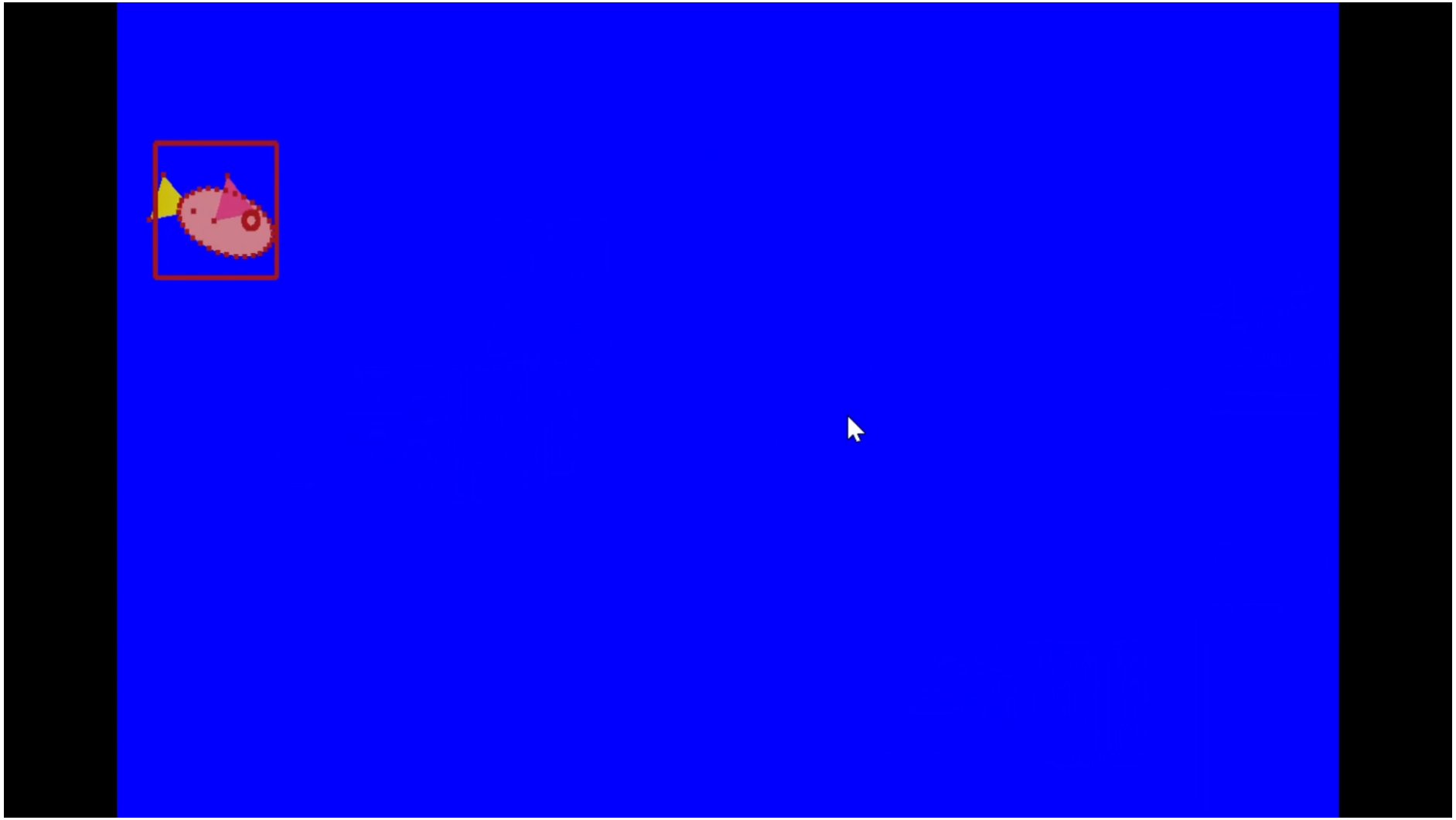
Template framework



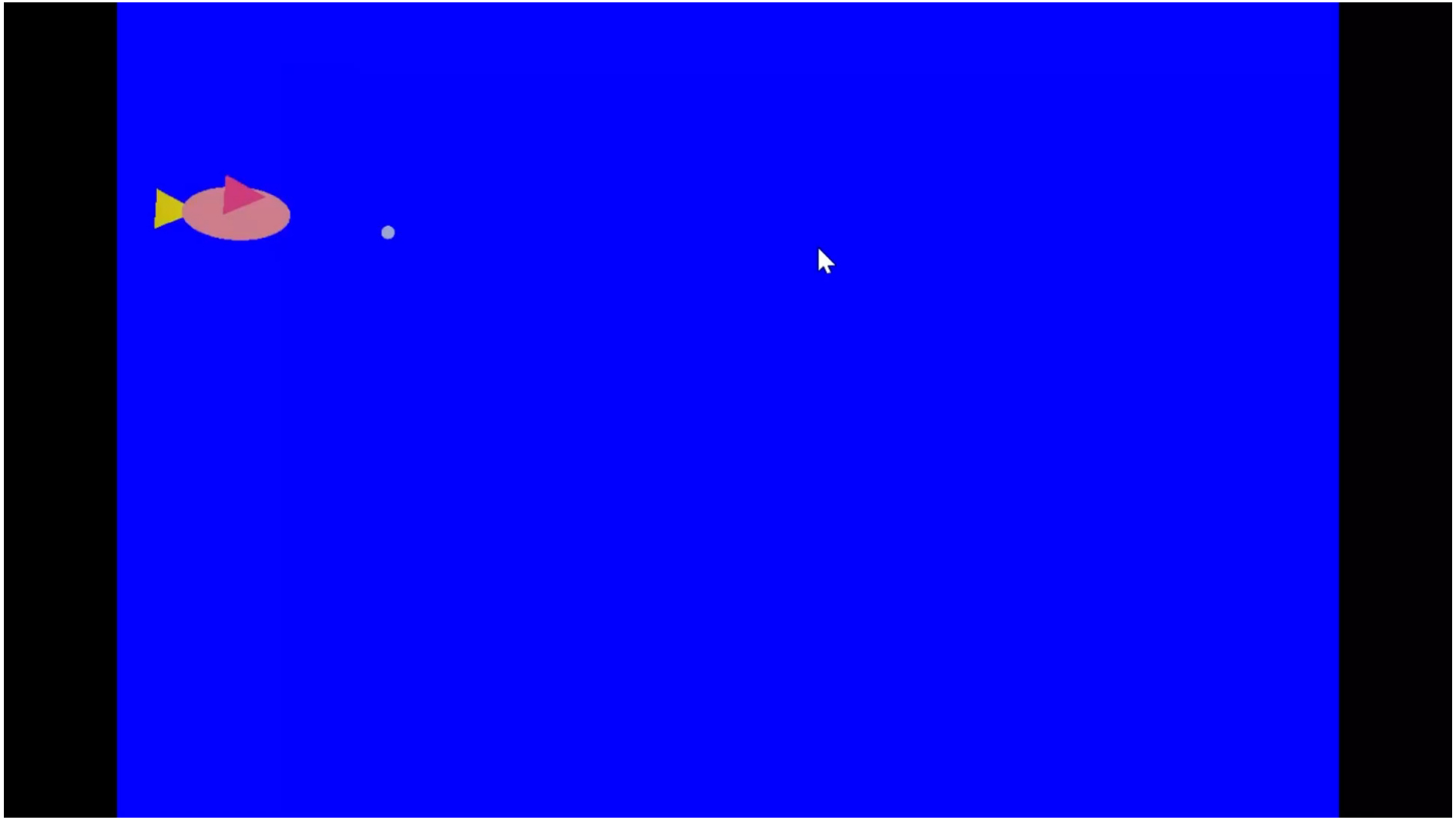
A1 – Game Graphics



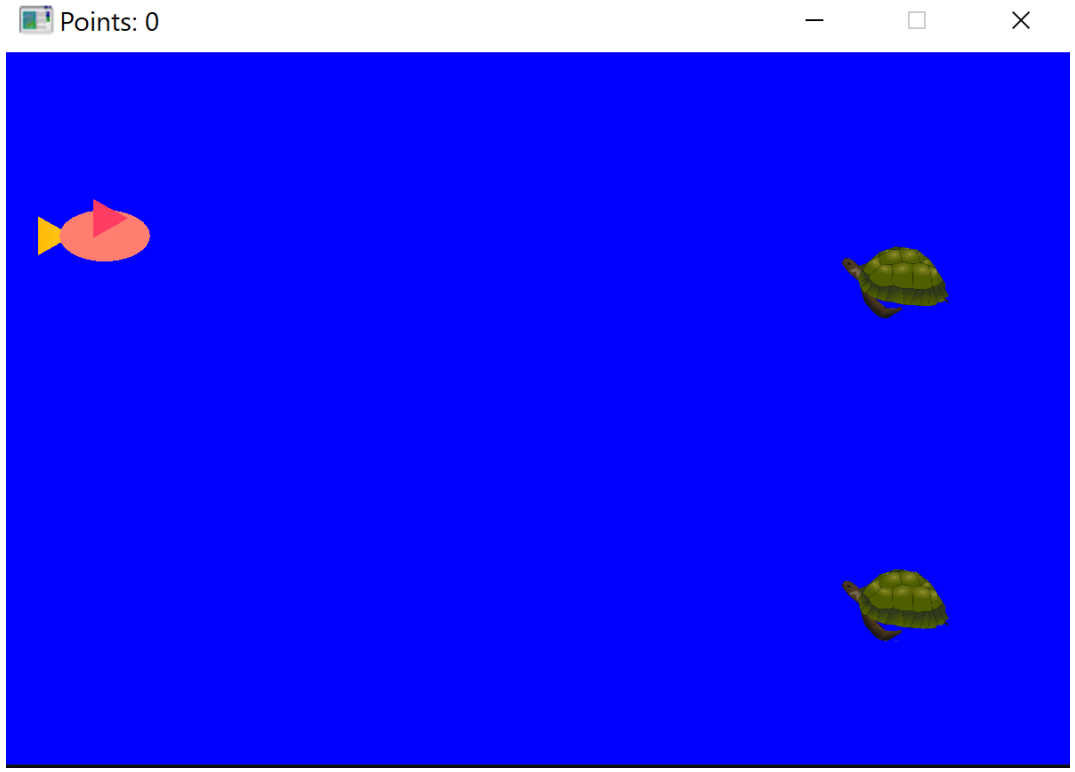
A2 – Game AI and Collision Processing

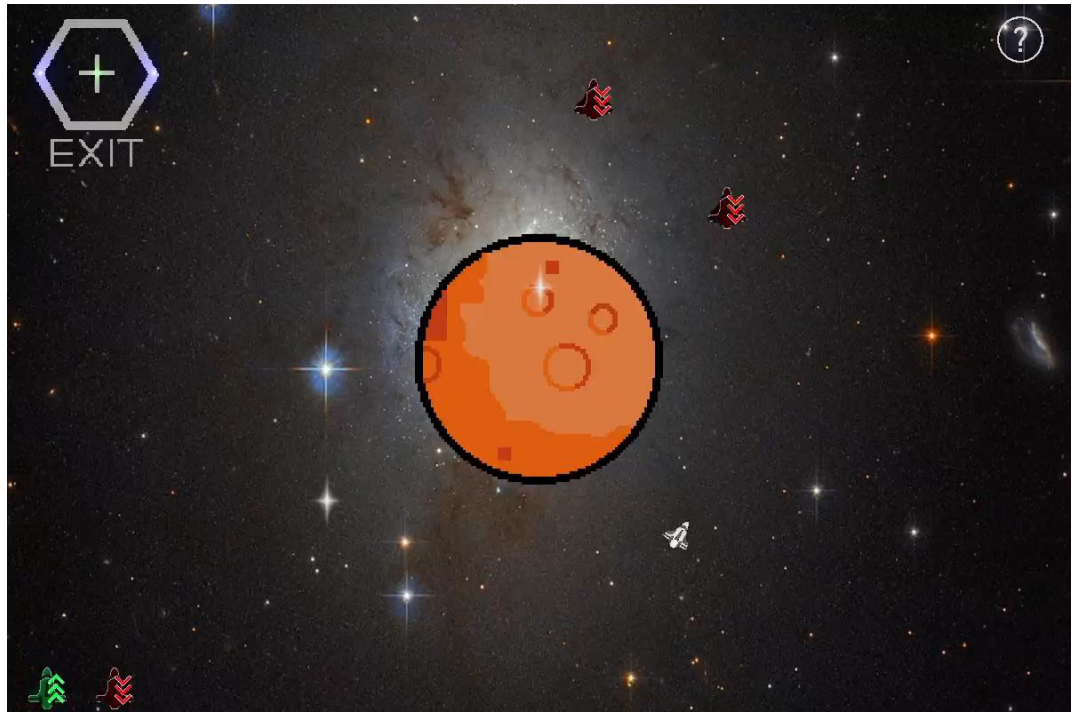


A3 – Animation and Physics



Your project





Register your Team!

Even if incomplete, please register

-> Canvas -> People -> Groups -> Team

Project templates

1. *Design (Oral Pitch -> Written Pitch -> Proposal/Development Plan)*
2. *Implement (Three milestones)*

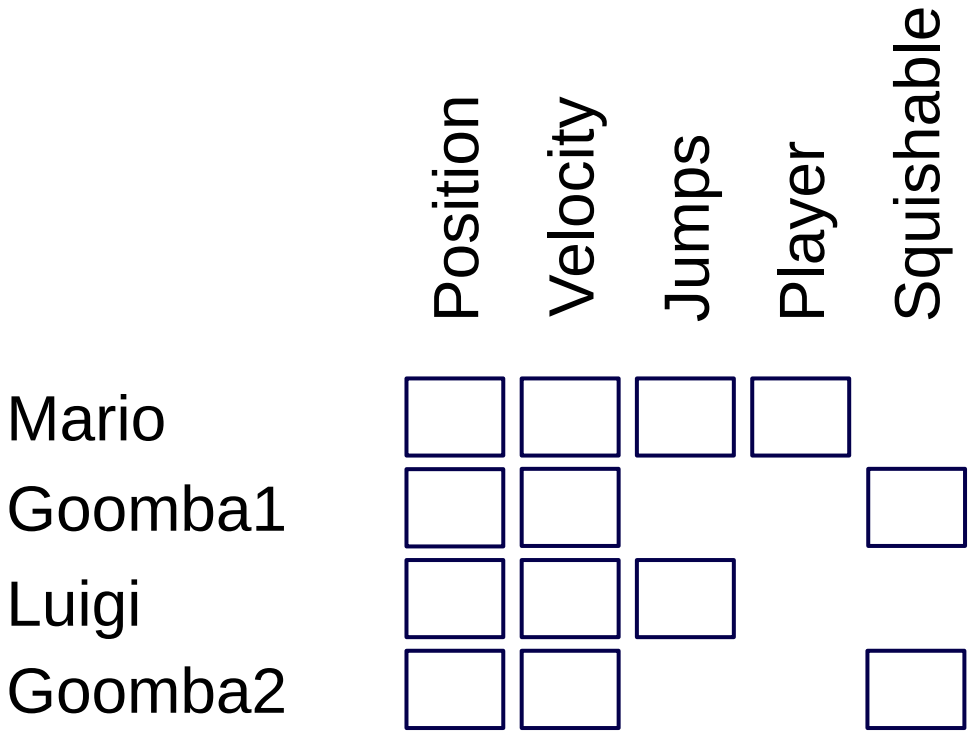
ECS implementations

Memory & ECS

Where do we store our Components?

- **RAM, harddrive, or chache?**
- **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

Problem: associating entities and components



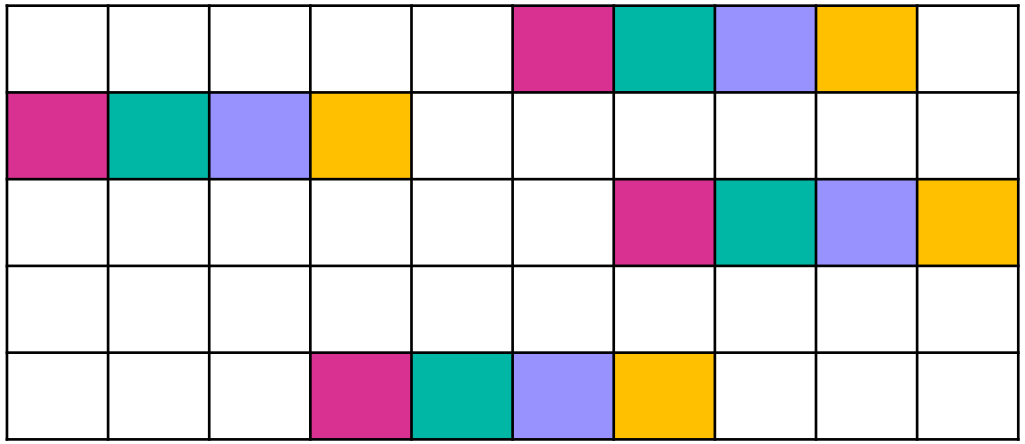
Object-oriented-programming (OOP)?

ECS = containers of components?

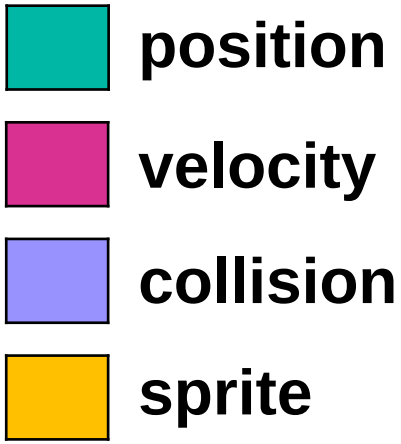
Memory & ECS

Where do we store our Components?

- **Inside Entities?**



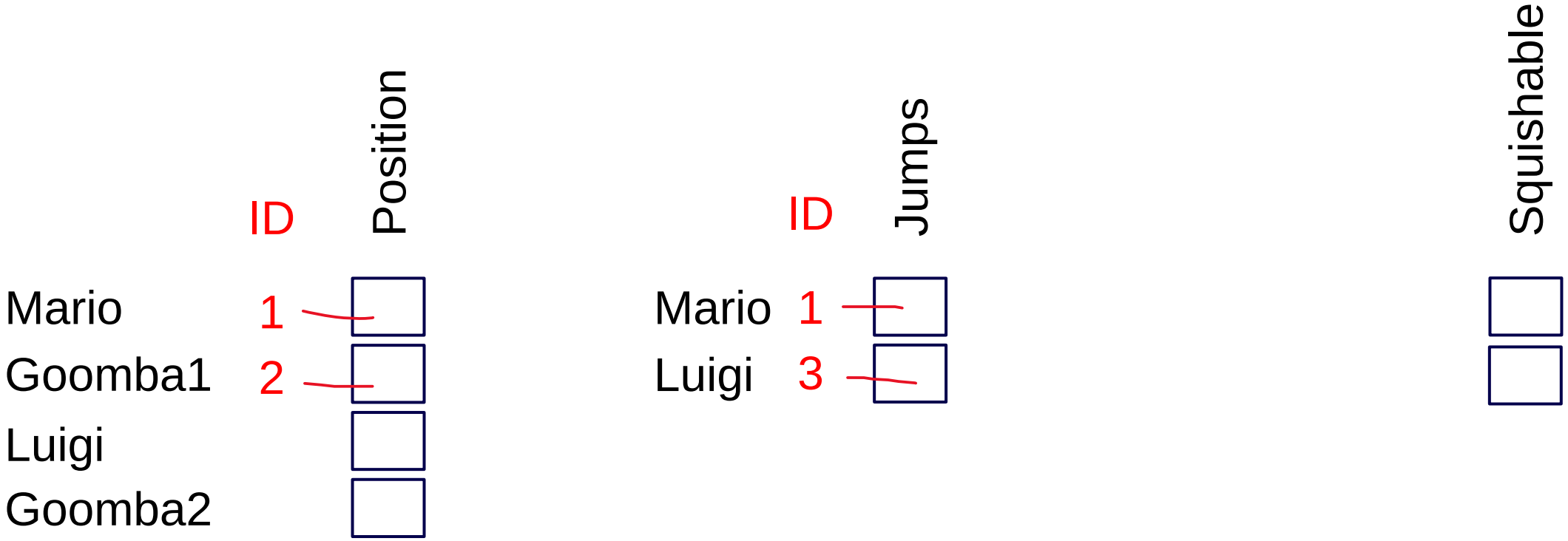
Memory Blocks



Update loop has to access non-contiguous memory repeatedly!

Slow memory access!

The Map Approach (entity ID to component address)

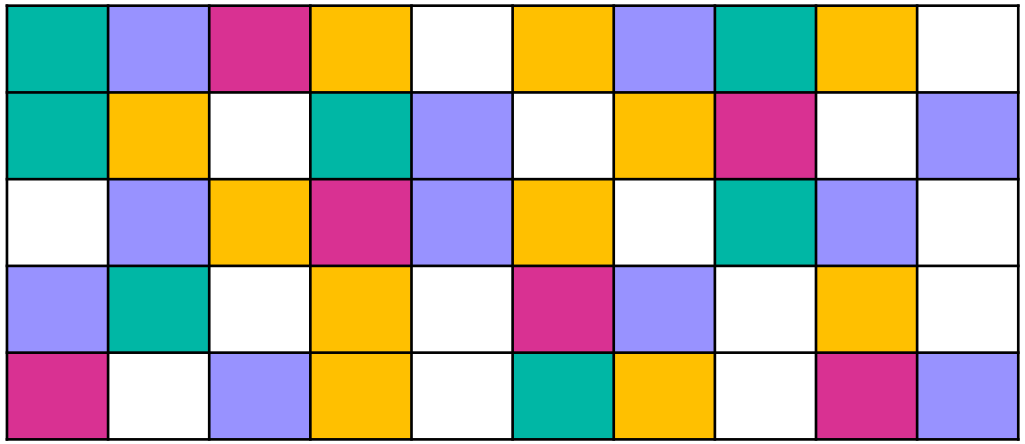


Concept: A (hierarchical) acceleration structure to lookup components
Implementation: std:map<Entity,Position>

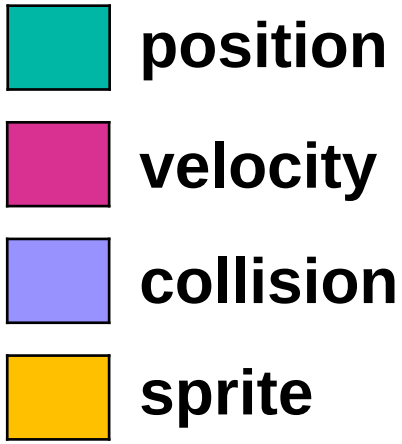
Memory & ECS

Where do we store our Components?

- In a map?



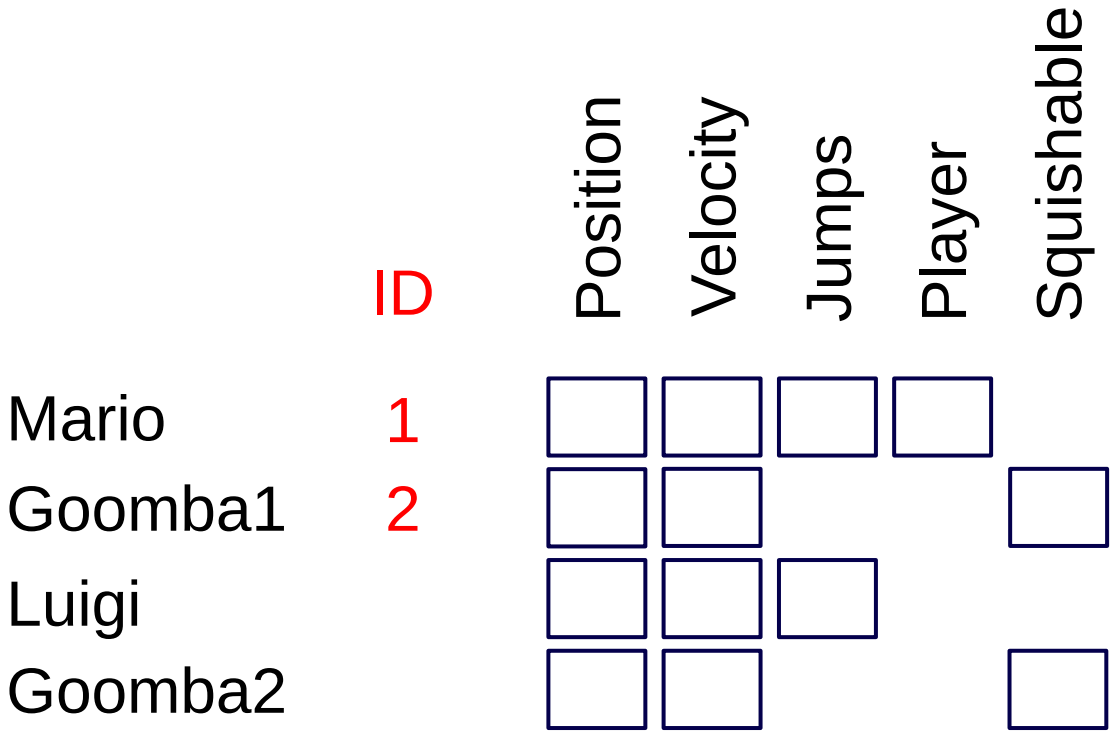
Memory Blocks



Update loop has to access non-contiguous memory repeatedly!

Slow memory access!

The (giant) Sparse Array



Issues?

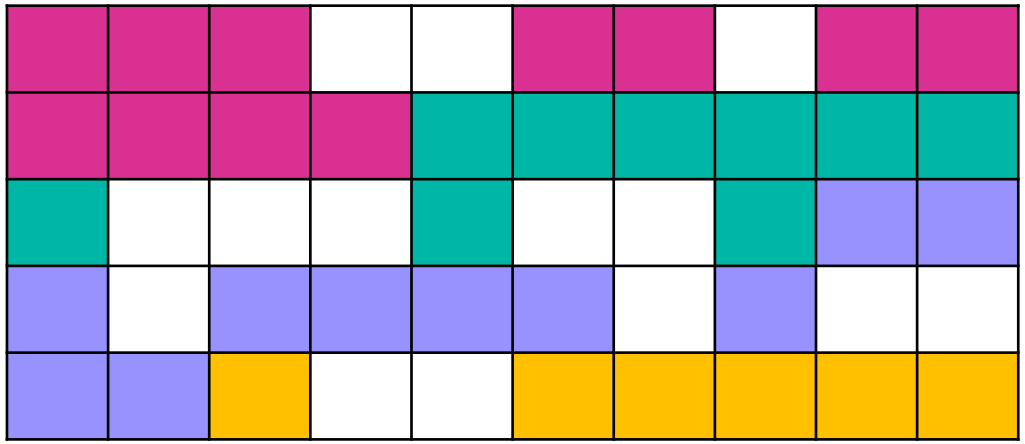
Concept: A huge data matrix of size Nr. Entities x Nr. components
Implementation: std:vector<Position>; std:vector<Velocity>

Memory & ECS

Where do we store our Components?

Last slide

- Array with holes?



Memory Blocks

position

velocity

collision

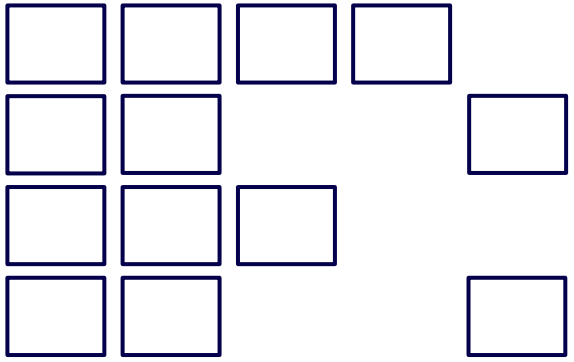
sprite

Better cache utilization!

Not memory efficient!

Bitset / Bitmap

	ID	Bitset/bitmap
Mario	1	11110
Goomba1	2	11001
Luigi	3	
Goomba2	4	



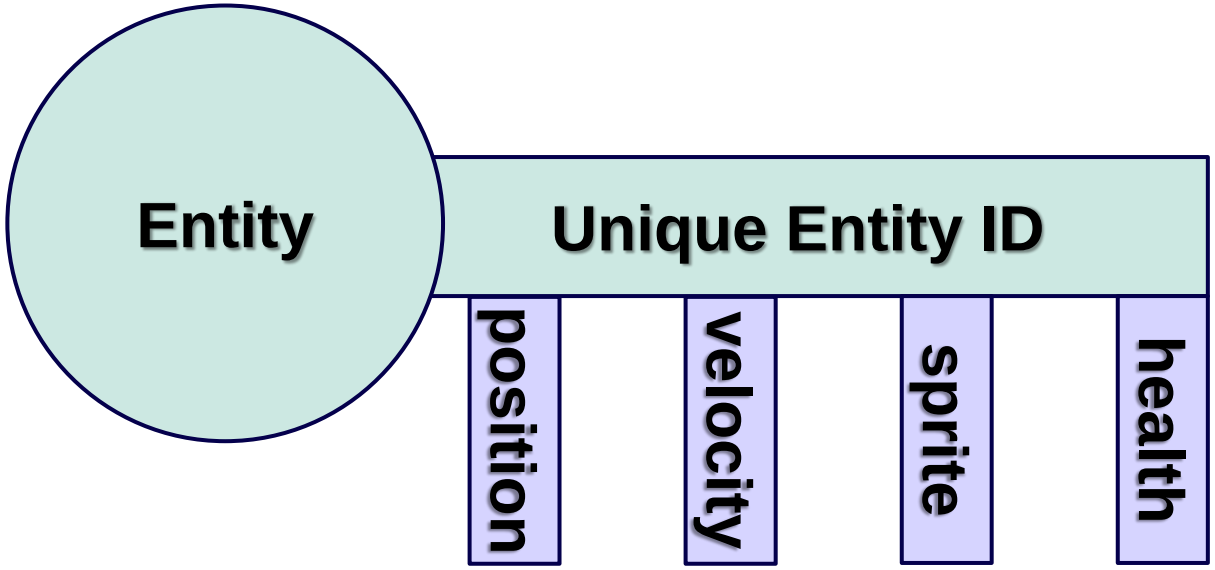
Issues?

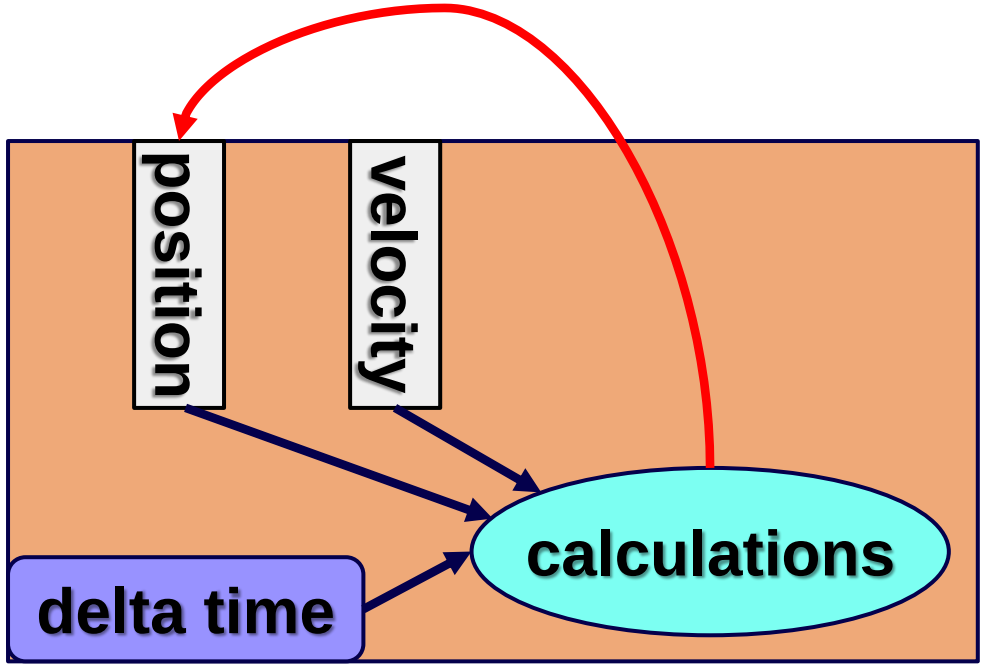
Concept: Each entity has a bitset that is true for its 'owned' components

Implementation: long bitset; // how many components can we support?

If(bitset & query == query) // has the entity all query components?

Key & Lock Metaphor



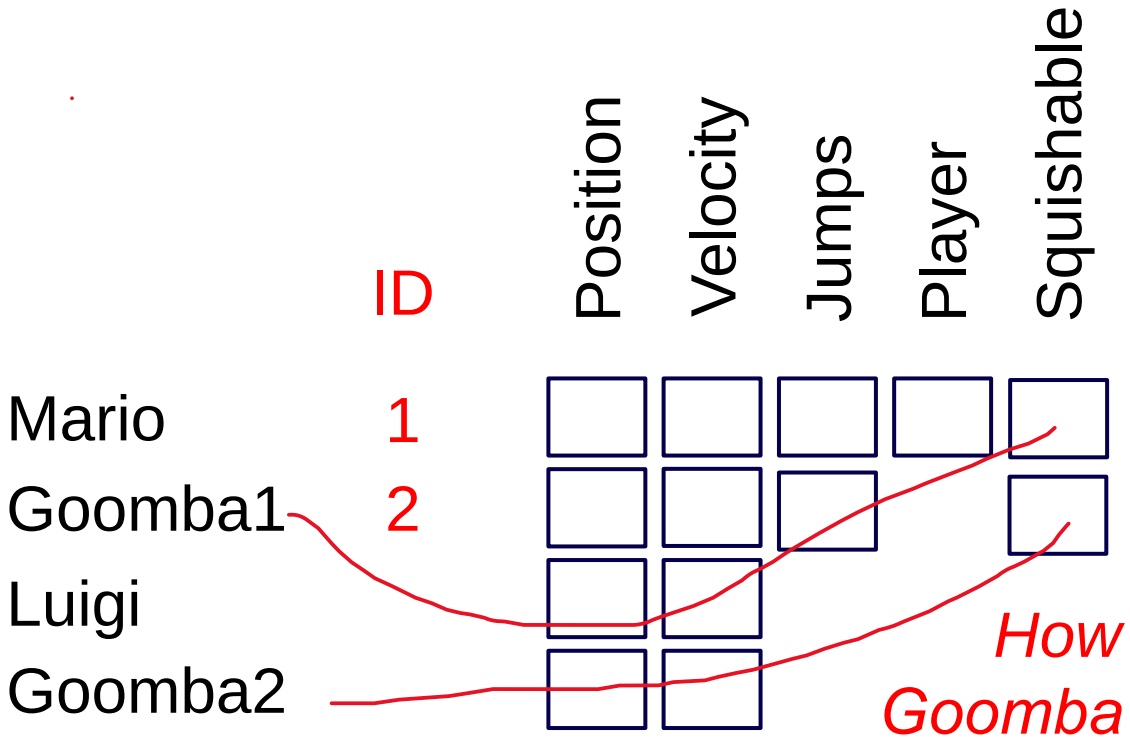


Motion System

Systems will only operate on Entities with the required Components

Further Improvements

Dense Component Vectors (an attempt, needs more)



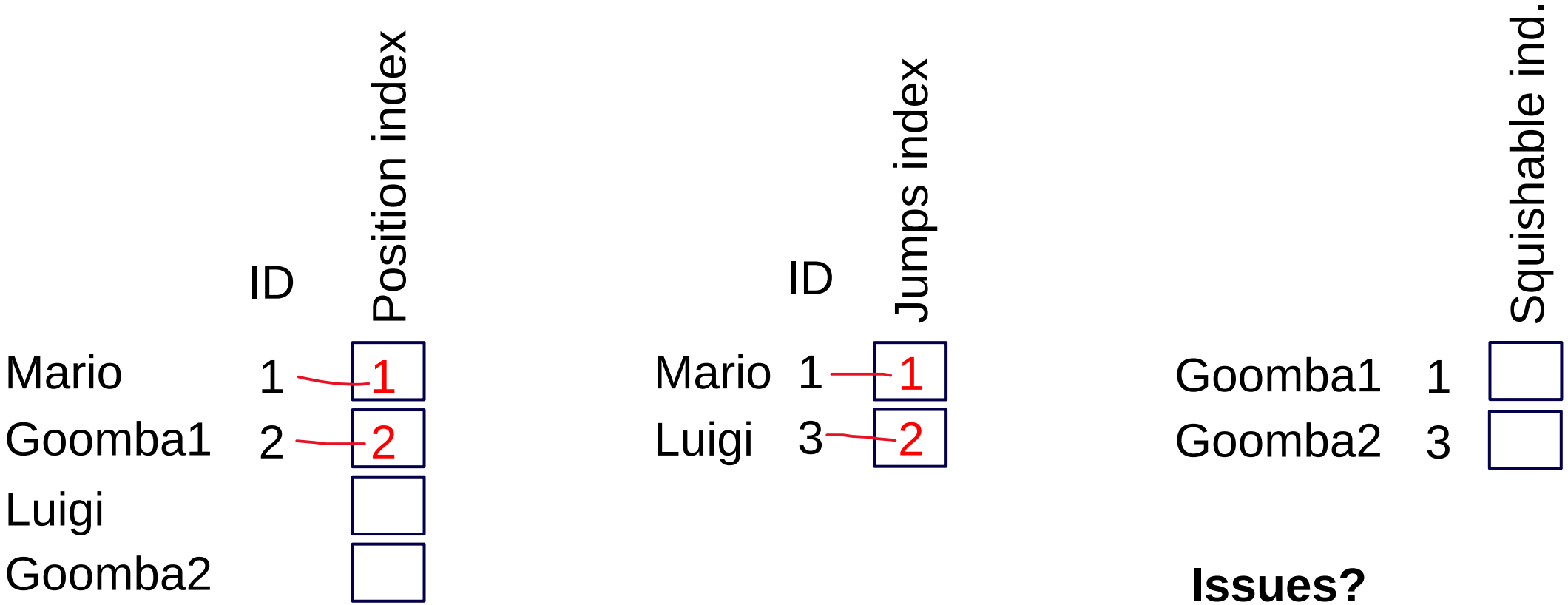
Issues?

How to find the position of Goomba's squishable component?

Concept: One array/vector per component, but how to associate?

Implementation: std:vector<Position>; std:vector<Velocity> + X?

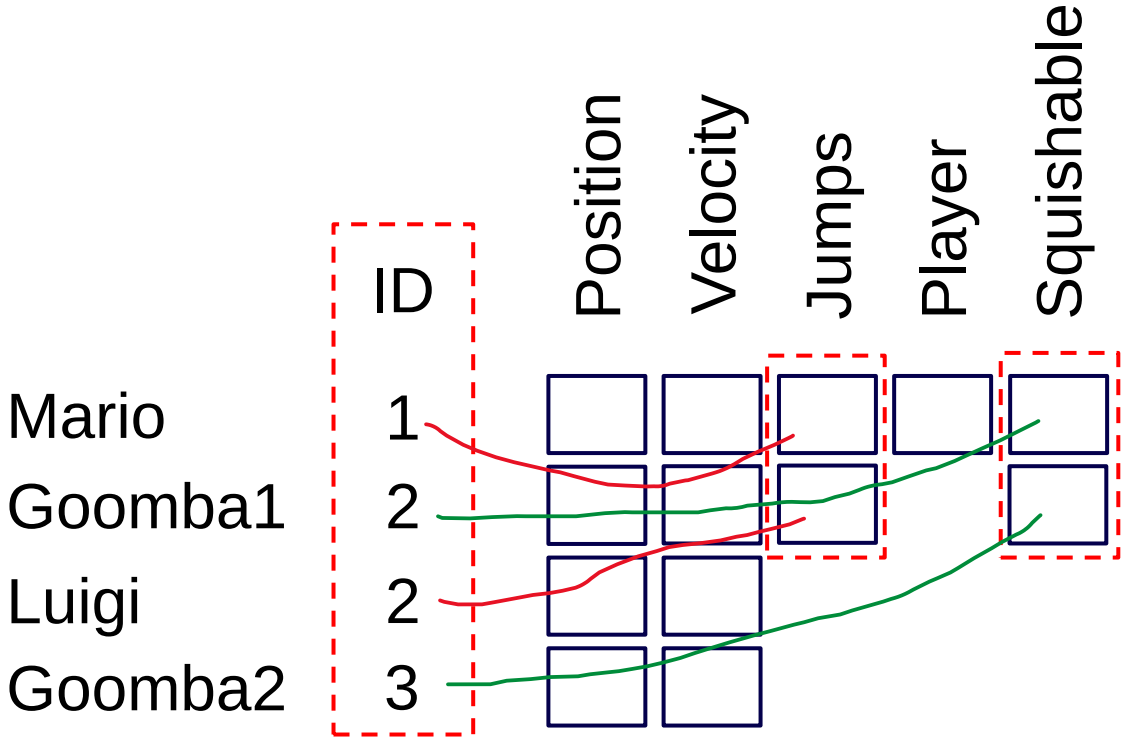
Map + Dense Component Vectors (entity ID to component ~~address~~ index)



Concept: Combine dense vectors with a map

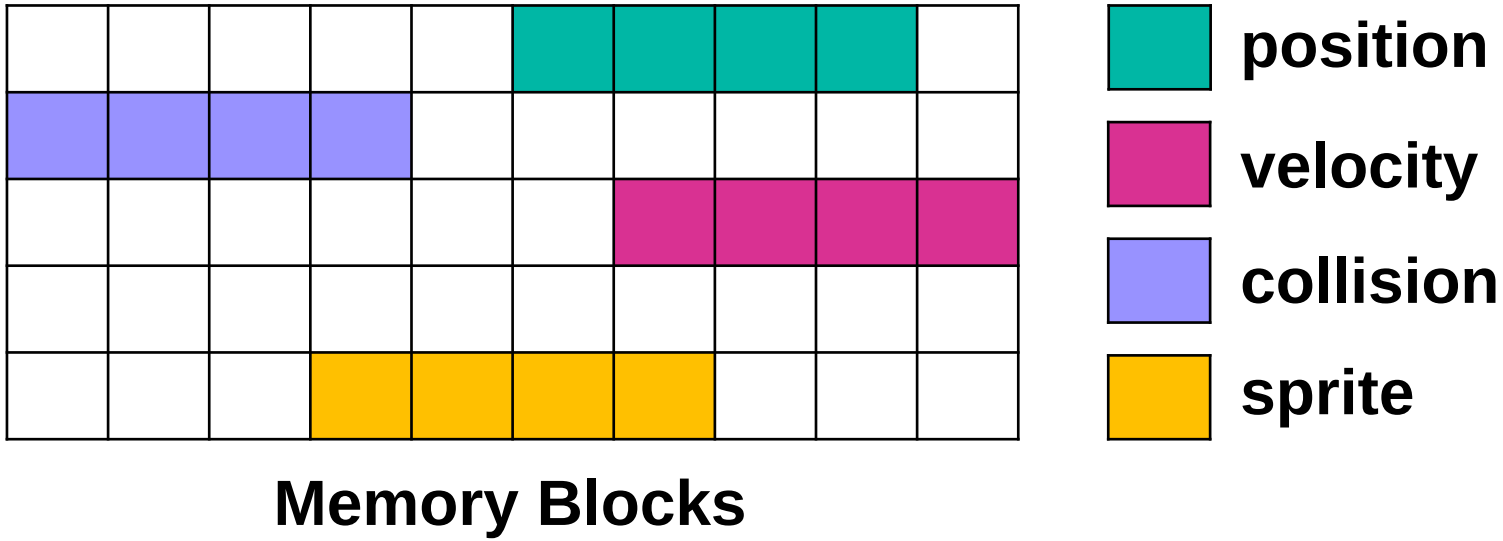
Implementation: std::vector<Component>; std::map<Entity,unsigned int>

Map + Dense Vector (different visualization)

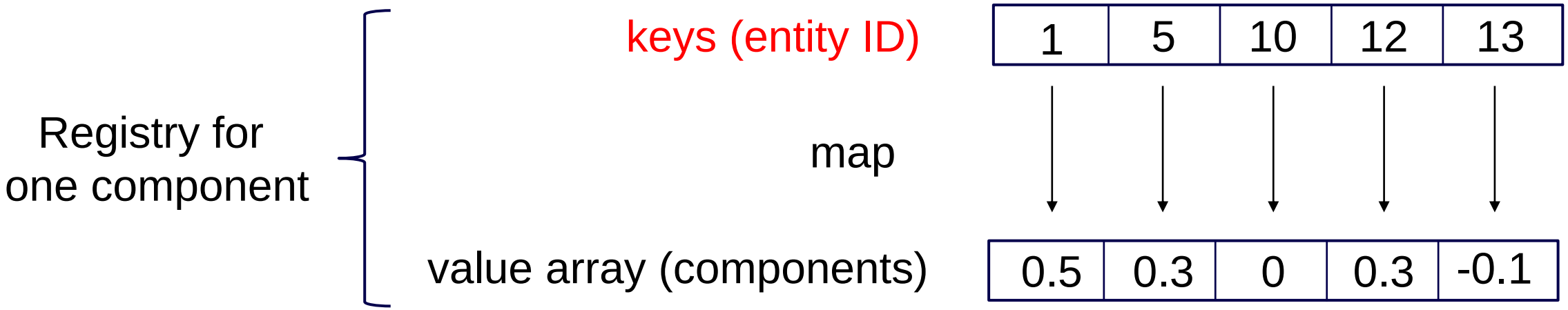


Cache is Key

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



Map + Component Vector + Entity Vector



Concept: Add a dense vector of entities to facilitate quick iteration over entities
Implementation: std::vector<Entities>; std::vector<Component>; std::map<Entity,unsigned int>

Easy to iterate over all velocity components that belong to an entity with a position

```
for(int entity : velocity_entities) // using the entities array
    if (position_entity_map.has(entity)) // using the map
        position_entity_map.get(entity)+= velocity_entity_map.get(entity); // using component array
```

Faster iteration via entity and component array

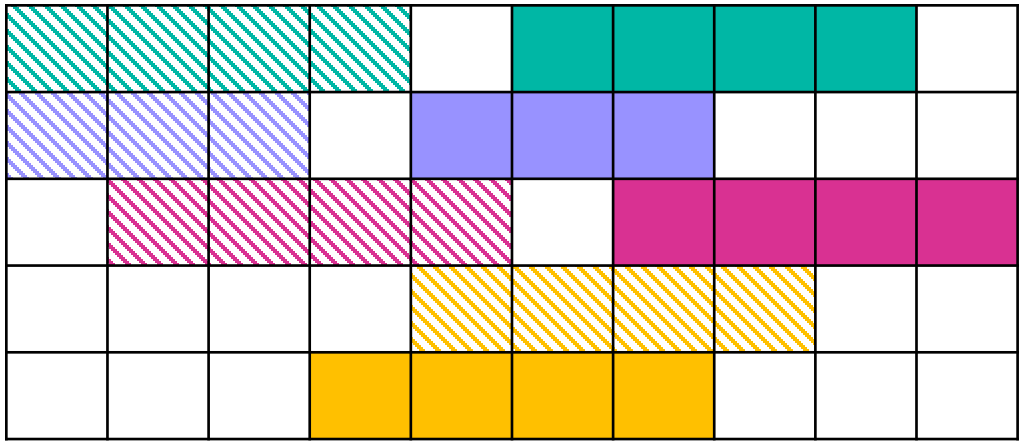
Accessing the velocity map (reg_velocity.map) is an unnecessary indirection

```
for(int entity : velocity_entities) // efficient
    if (position_entity_map.has(entity)) // inefficient lookup
            position_entity_map.get(entity)+= velocity_entity_map.get(entity); // 2x inefficient lookup
```

We can access the velocity components in linear fashion

```
for(int vel_i = 0; vel_i < velocity_entities.size(); vel_i++) // efficient
    Entity entity : velocity_entities[vel_i]; // efficient
    int pos_i = position_entity_map.getIndex(entity); // inefficient lookup
    if (pos_i)
            position_components[pos_i]+= reg_velocity_components[vel_i]; // efficient
```

Map + Component Vectors + Entity Vector Cache is Key



Memory Blocks

position

velocity

collision

sprite

position entity IDs

velocity entity IDs

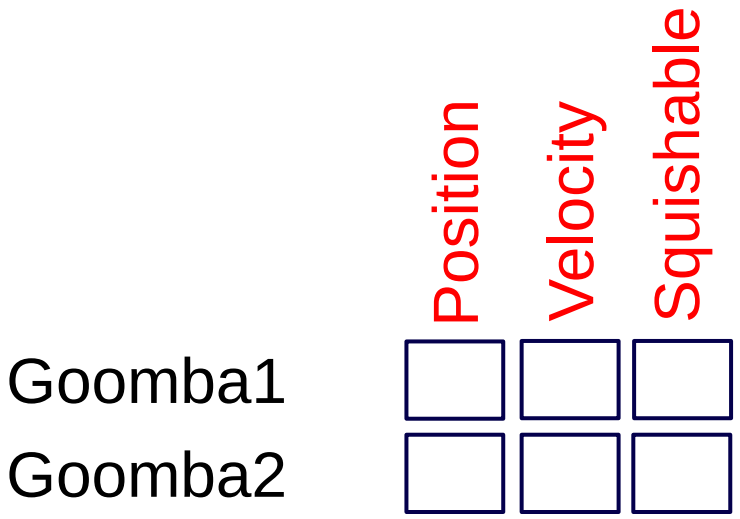
collision entity IDs

sprite entity IDs

Update loop accesses contiguous memory IDEAL!

Map access slow

Advanced ECS: Archetypes / prototypes / pools



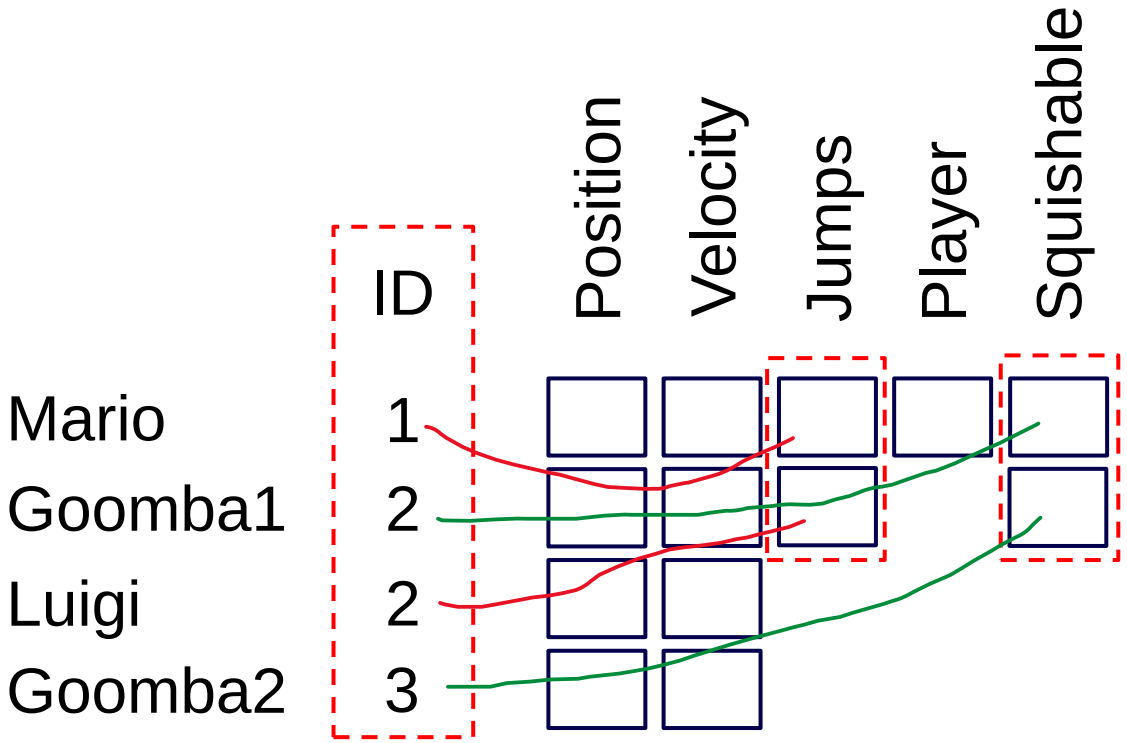
- **Concept:** store all types with the same components in dense arrays
- Used by the Unity ECS system
- Difficult to implement

How Does a System Find its Entities?

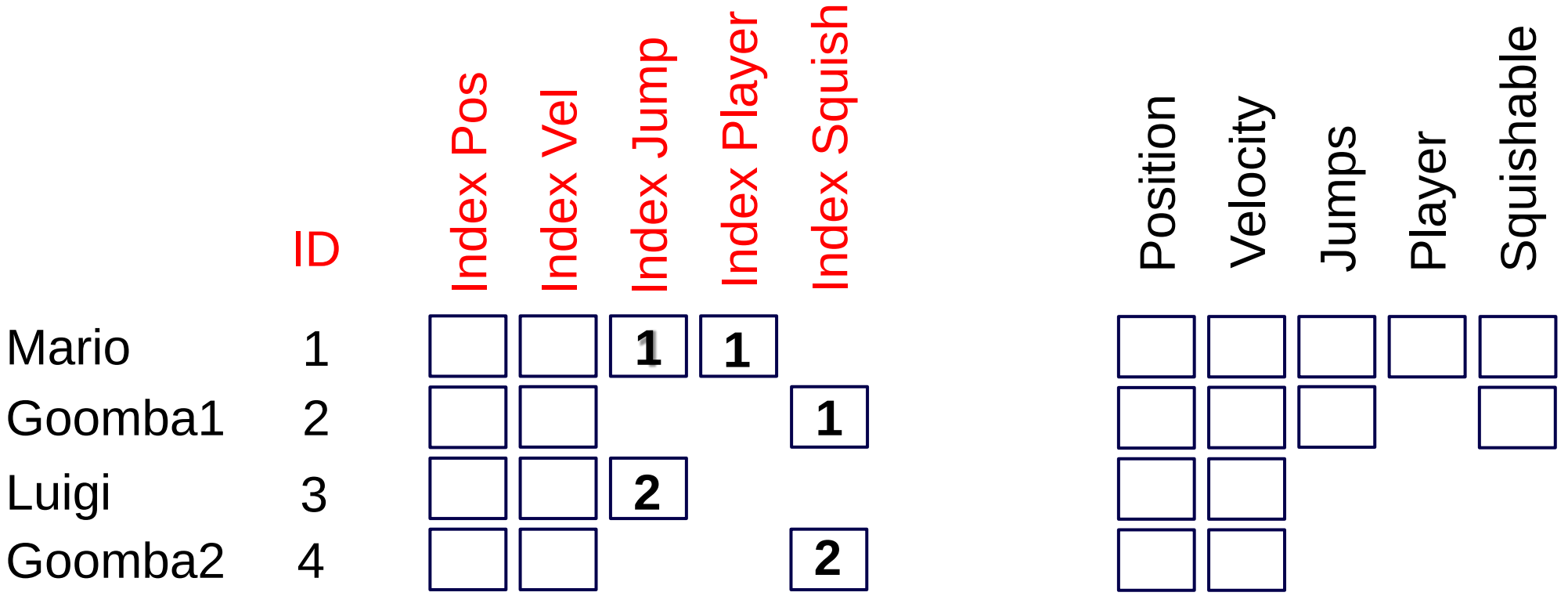
Extension: Entity Manager

- **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***

Self-study: A special map approach



Self-study: The ‘Sparse Set’



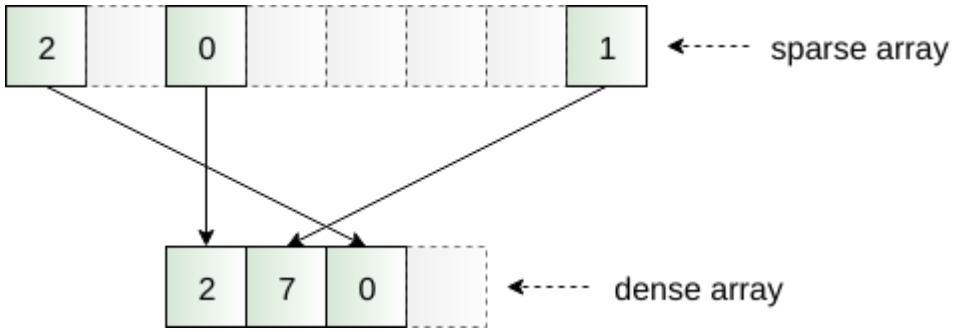
Issues?

Concept: Sparse array + dense array

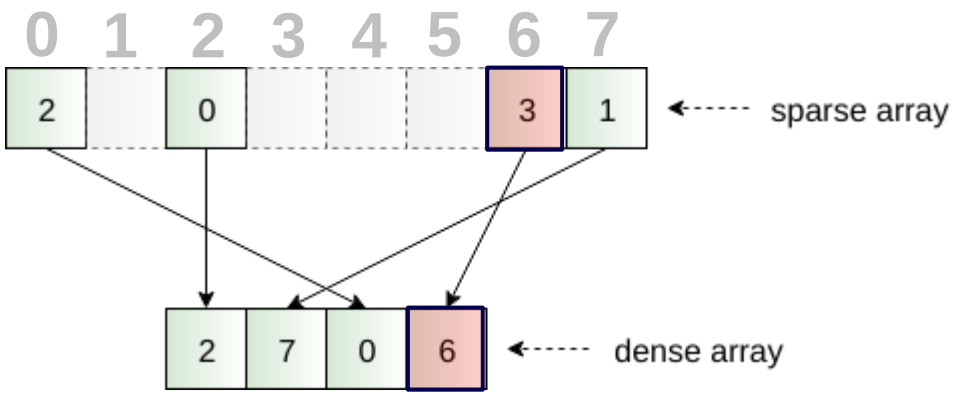
Implementation: std:vector<Entity> entities; std:vector<unsigned int> indices; std:vector<Components> components;

Self-study: Faster Lookup with Sparse Sets

Lookup:



Insert:



[https://skypjack.github.io/2020-08-02-ecs-baf-part-9/]

The map lookup (map.get(entity)) is costly

- A hashmap is O(1), but that 1 is big

Sparse set:

- An array as large as the number of entities in the game
  - Crazy waste of memory?!
  - 32 bit integer -> ???
  - a sparsely filled array
- A small dense array of all entities in sequence (as before)
- Extremely fast lookup, insert, & clear