Implementation of a **component-based entity system** in modern C++

https://github.com/SuperV1234/cppcon2015

cppcon the c++ conference

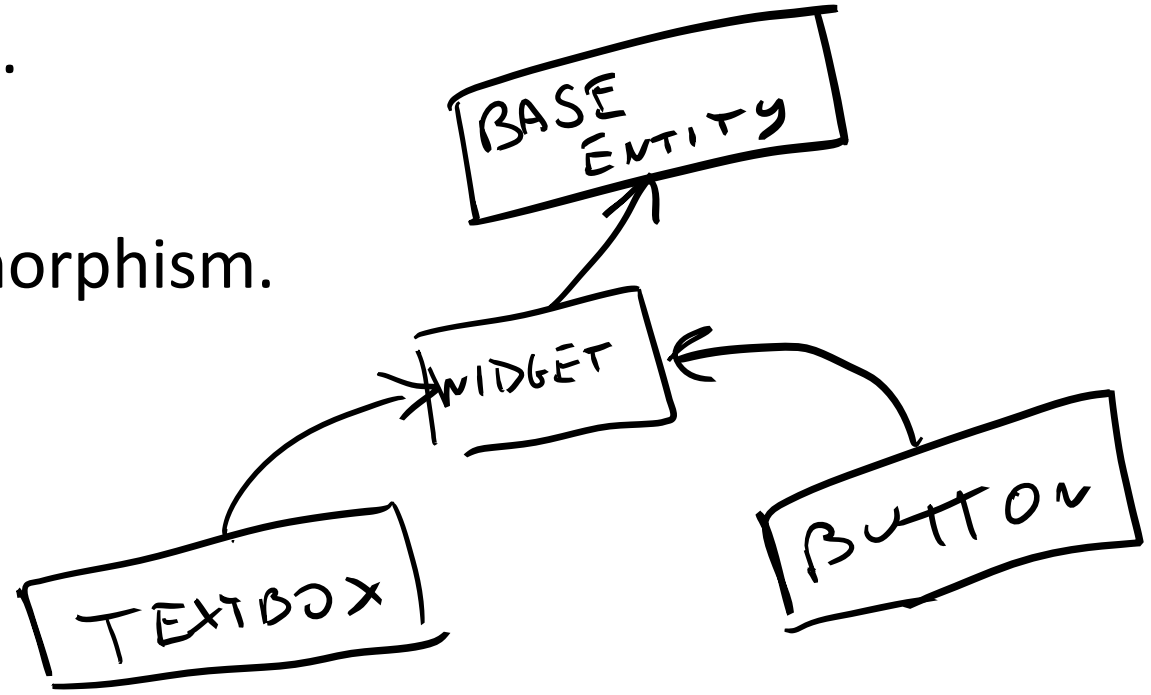
http://vittorioromeo.info
vittorio.romeo@outlook.com

What is an **entity**?

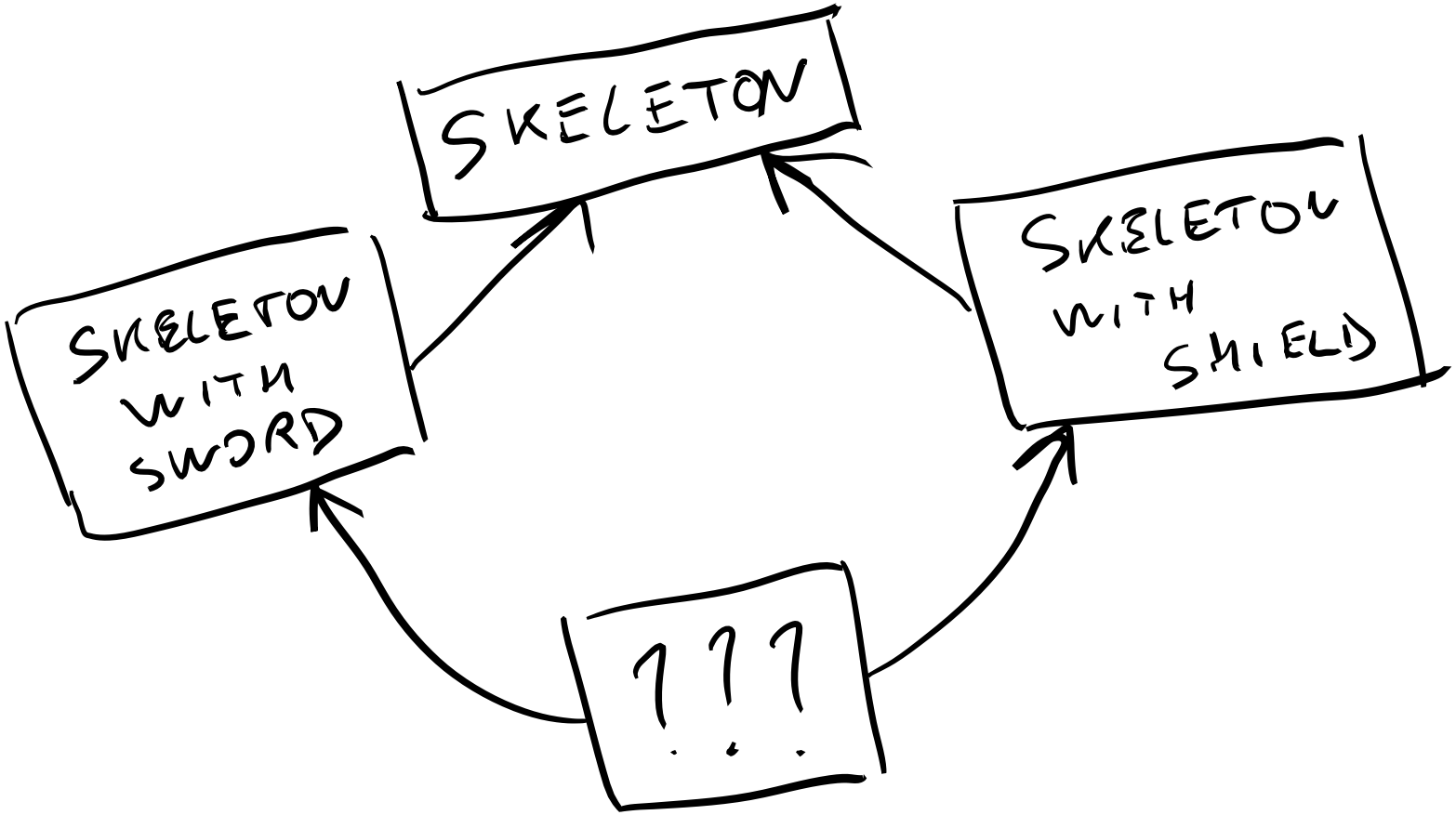
- Something **tied** to a **concept**.
- Has related **data** and/or **logic**.
- We may want to **track** a particular entity.
- Can be **created** and **destroyed.**
- Examples:
  - **Game objects:** *player, bullet, car.*
  - **GUI widgets:** *window, textbox, button.*

Encoding entities – OOP inheritance

- An **entity type** is a **polymorphic class**.
- **Data** is stored inside the class.
- **Logic** is handled using **runtime** polymorphism.
- **Very easy** to implement.
- **Cache-unfriendly**.
- **Runtime overhead**.
- **Lack of flexibility**.



Encoding entities – OOP inheritance



http://vittorioromeo.info
vittorio.romeo@outlook.com

Encoding entities – OOP inheritance

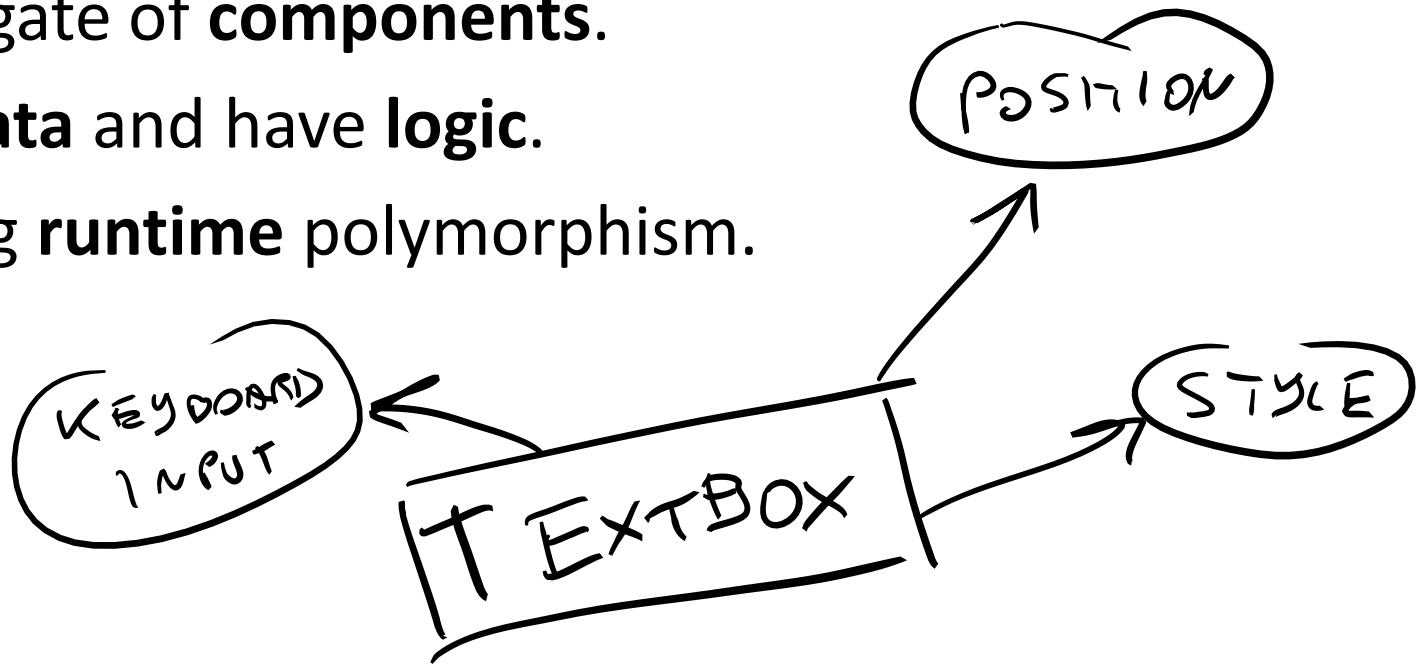
```
struct Entity
{
    virtual ~Entity() { }
    virtual void update() { }
    virtual void draw() { }
};
```

```
struct Skeleton : Entity
{
    std::vector<Bone> bones;
    void update() override
    {
        // do things skeletons do
    }
};
```

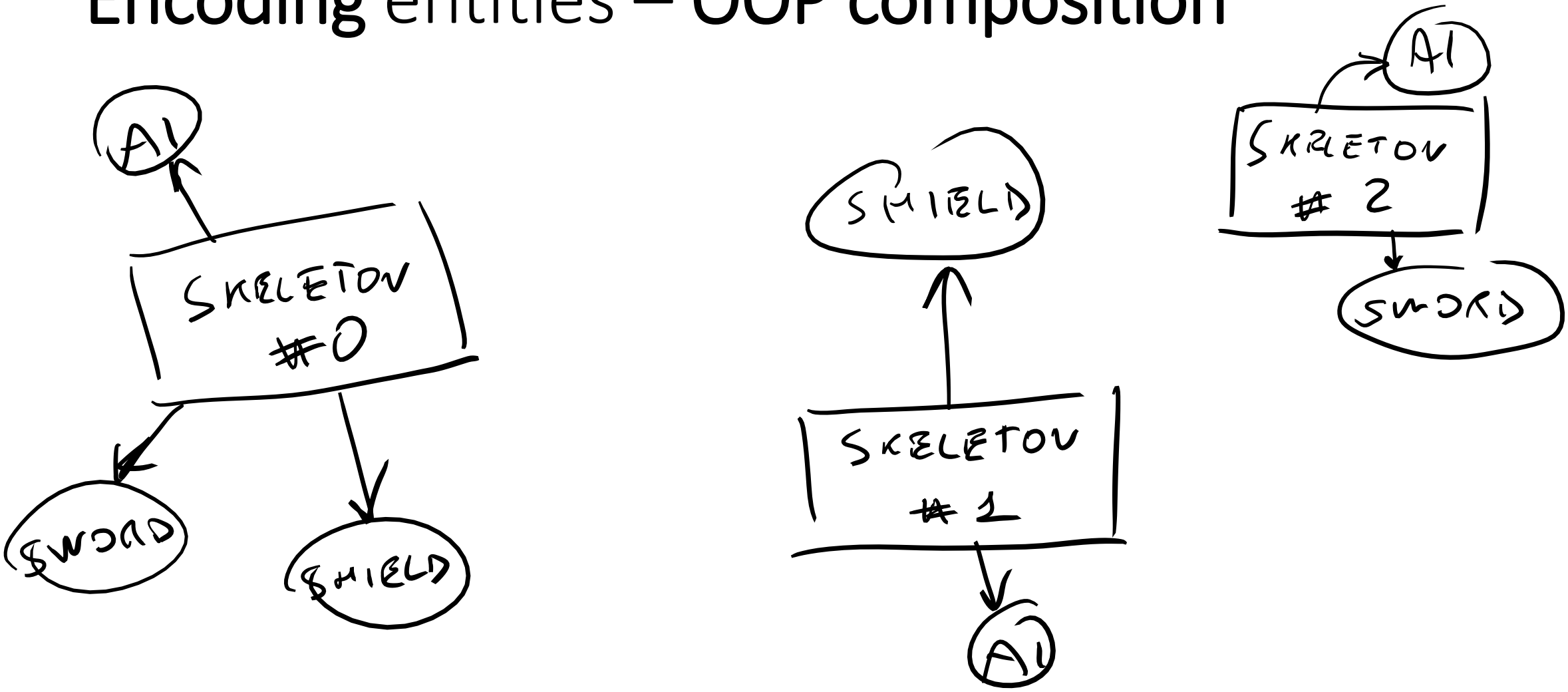

http://vittorioromeo.info
vittorio.romeo@outlook.com

Encoding entities – OOP composition

- An **entity** is an aggregate of **components**.
- Components **store data** and have **logic**.
- **Logic** is handled using **runtime** polymorphism.
- **Easy** to implement.
- More **flexible**.
- **Cache-unfriendly**.
- **Runtime overhead**.



Encoding entities – OOP composition



Encoding entities – OOP composition

```
struct Component
{
    virtual ~Component() { }
    virtual void update() { }
    virtual void draw() { }
};

struct Entity
{
    std::vector<std::unique_ptr<Component>> components;

    void update() { for(auto& c : components) c->update(); }
    void draw() { for(auto& c : components) c->draw(); }
};
```


http://vittorioromeo.info
vittorio.romeo@outlook.com

Encoding entities – OOP composition

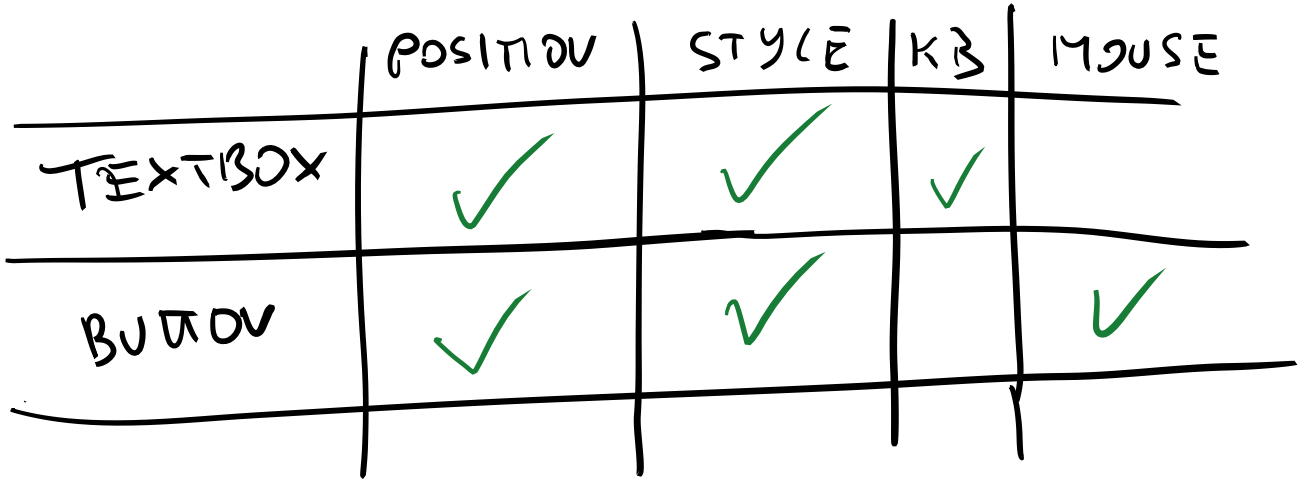
```
struct BonesComponent : Component
{
    std::vector<Bone> bones;
    void update() override
    {
        // do things skeletons do
    }
};

auto makeSkeleton()
{
    Entity e;
    e.components.emplace_back(std::make_unique<BonesComponent>());
    return e;
}
```


http://vittorioromeo.info
vittorio.romeo@outlook.com

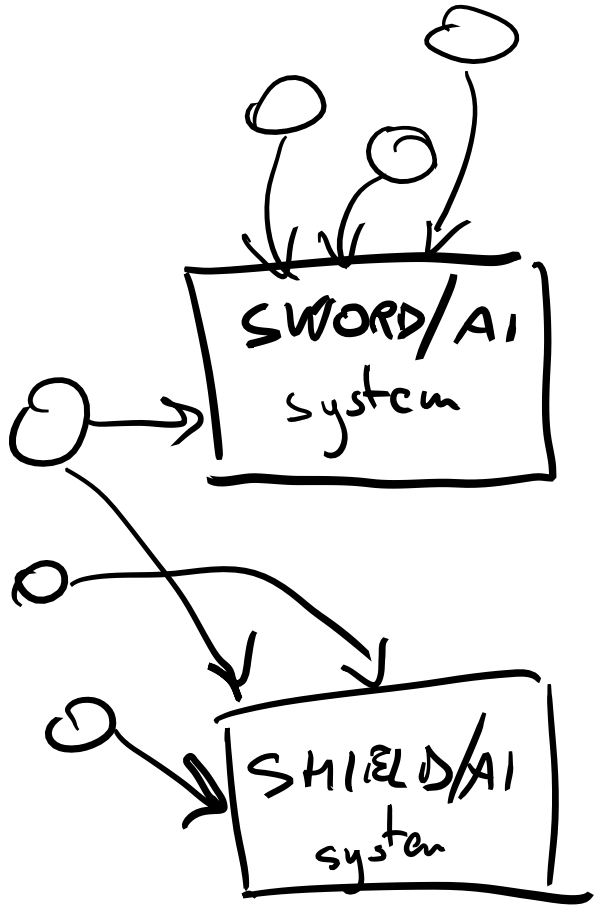
Encoding entities – DOD composition

- An **entity** is an aggregate of **components**.
- Components only store **data** (logicless).
- **Logic** is handled using **systems**.
- Potentially **cache-friendly**.
- Minimal **runtime overhead.**
- Great **flexibility.**
- **Very hard** to implement.

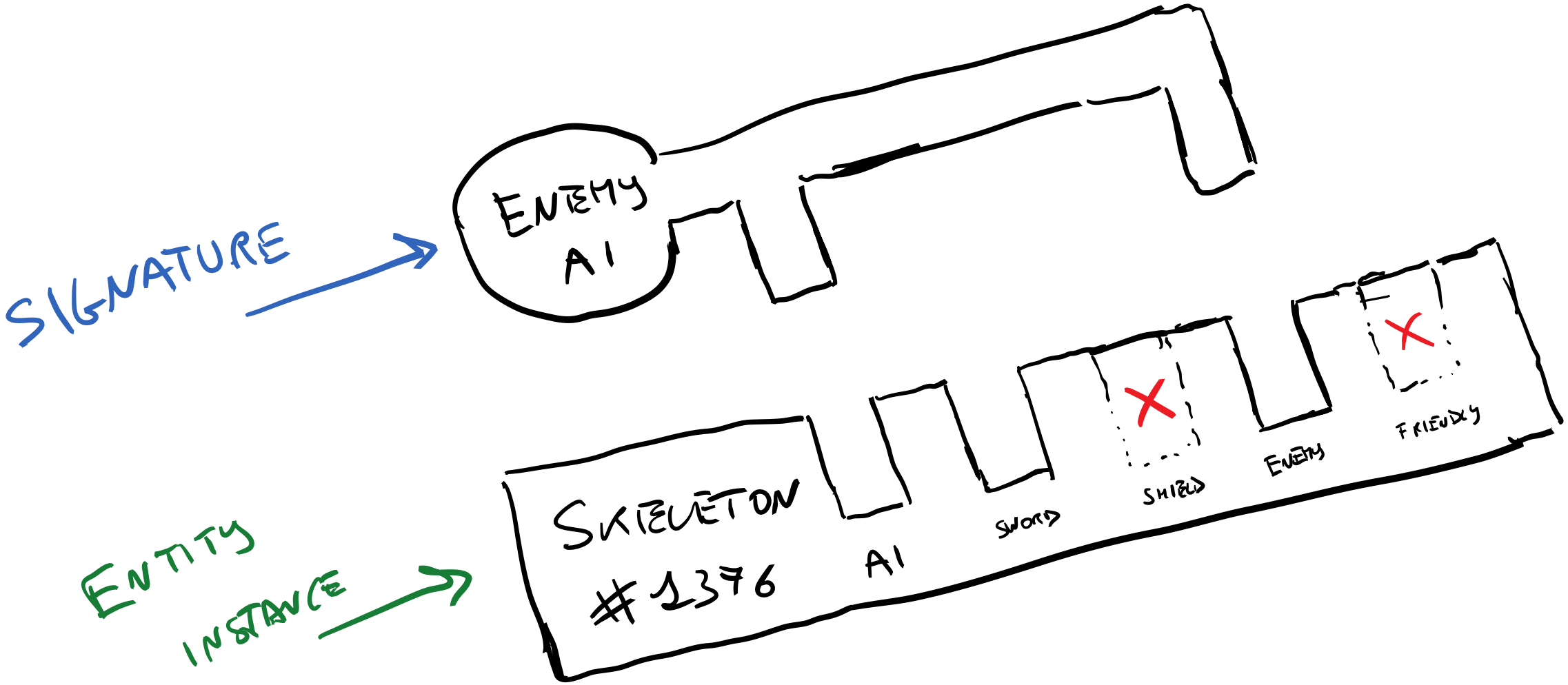


Encoding entities – DOD composition

	AI	SWORD	SHIELD
SKELETON T0	✓	✓	✓
SKELETON T1	✓	✓	
SKELETON T2	✓		✓



Encoding entities – DOD composition



http://vittorioromeo.info
vittorio.romeo@outlook.com

Encoding entities – DOD composition

```
using Entity = std::size_t;
constexpr std::size_t maxEntities{1000};

struct BonesComponent { /* ... */ };
struct AIComponent { /* ... */ };
struct SpriteComponent { /* ... */ };

struct Manager
{
    std::array<BonesComponent, maxEntities> bonesComponents;
    std::array<AIComponent, maxEntities> aiComponents;
    std::array<SpriteComponent, maxEntities> spritesComponents;

    // ...
};
```

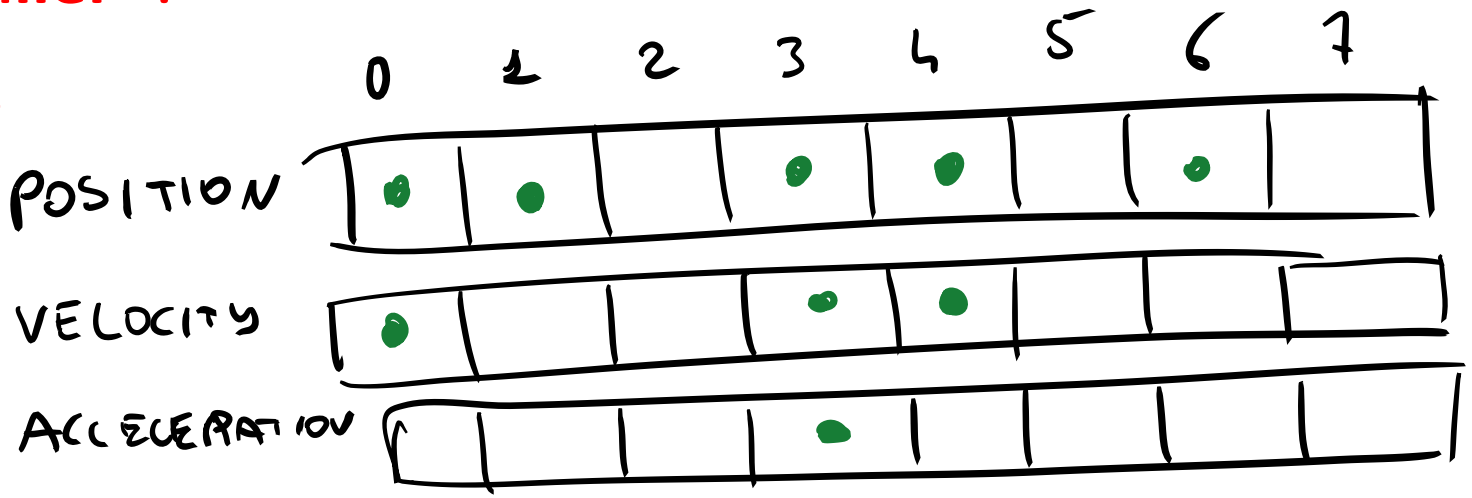

http://vittorioromeo.info
vittorio.romeo@outlook.com

Implementation details – DOD composition

- Component **types** are known at **compile-time**.
  - They are simple logicless **classes**.
- Entities are **lightweight** objects.
  - They contain a **bitset** of their available components.
  - They also store **metadata** for handles and memory reclamation.
- Systems are **«implicit»**.
  - **Signatures** are used instead. A signature is a set of required components types.
  - A signature is a **bitset**. Entities can be **queried by signature**.
- How do we **store components**?

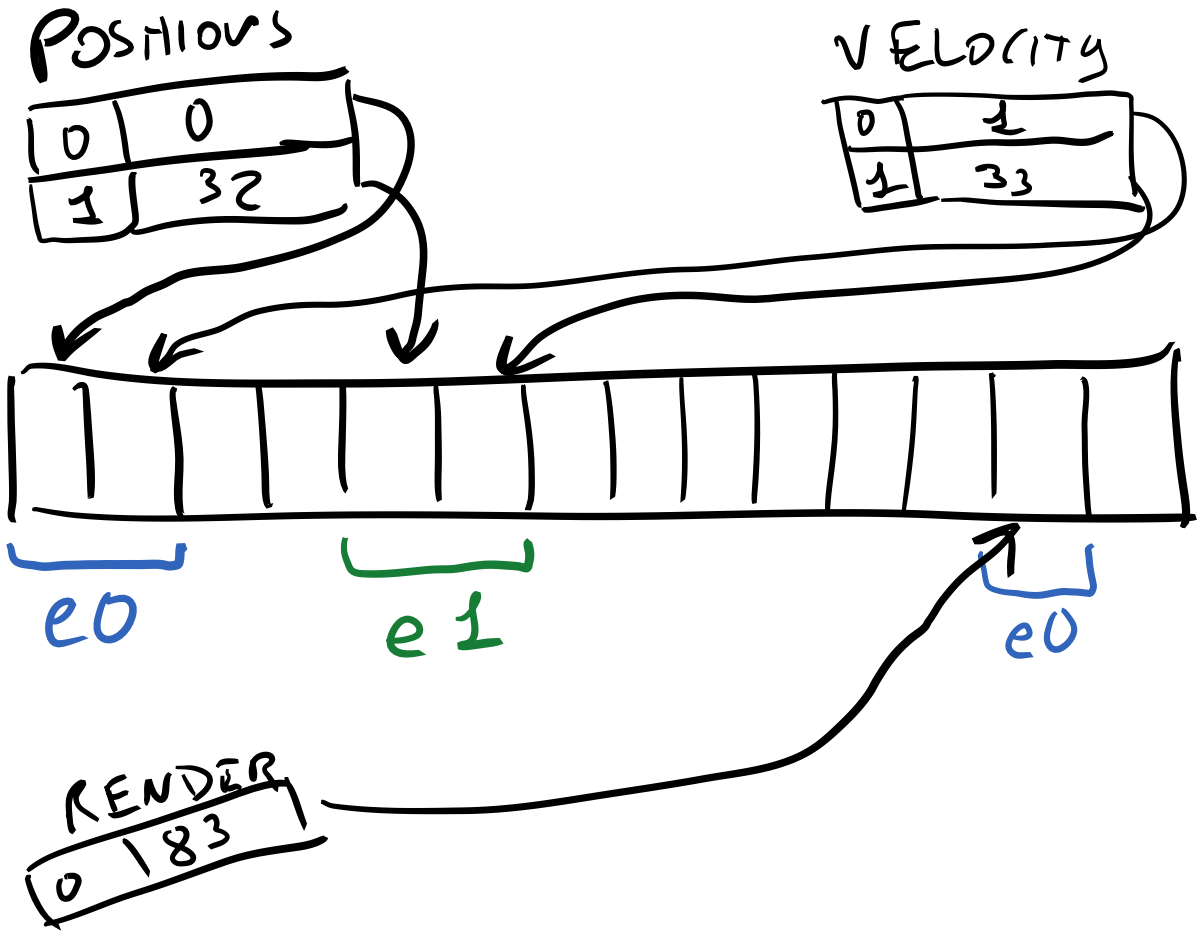
Storing components - **one array per type**

- **Very easy** to implement.
- Suitable for **most** projects.
- Easy to **add/remove** components at runtime.
- Can be «**cache-friendlier**».
- **Wasteful** of memory.



Storing components – **mega-array**

- **Cache-friendly**.
- **Minimizes** memory waste.
- **Very hard** to implement.
  - Requires **indexing tables**.
  - **Hard** to deal with component **addition/removal.**
  - **Very hard** to **keep track** of free «slots» in the mega-array.



Live example – **simple shoot'em'up game**

- Start from a **traditional inheritance-based** implementation.
  - We will **quickly skim through** the «OOP implementation».
- Gradually transition to a **component-based** implementation.
  - Components will be stored using the **«one array per type»** approach.
  - Additional features: **handles** and **tags**.
- Heavy usage of **modern C++** features.
  - Variadic template metaprogramming, tuples, auto, …
  - **«More functional»** programming style where possible.
- **SFML** will be used for input/window/rendering.

http://vittorioromeo.info
vittorio.romeo@outlook.com

Future ideas/improvements

- Benchmark **alternative data structures.**
- **Serialization/networking** capabilities.
  - Components are easy to serialize – calculating differences between two states would also be useful.
- **Events** for communication between unrelated entities.
- **Entity caching**.
  - Entities could get matched and stored during **refresh**.
- **Preventing invalidation** when creating entities during updates.
- **Multithreading** features.
- **Constexpr bitsets**.

http://vittorioromeo.info
vittorio.romeo@outlook.com

Resources

- http://t-machine.org
  - Articles on **data structures**, **multithreading** and **networking**.
  - Wiki with **ES approaches** and existing implementations.
- http://stackoverflow.com/questions/1901251
  - In-depth analysis of **component-based engine design**.
- http://bitsquid.blogpost.it
  - Articles on **contiguous component data** allocation strategies.
- http://gameprogrammingpatterns.com/component
  - Covers **component-based design** and **entity communication** techniques.
- http://randygaul.net
  - Articles on **component-based design**, covering **communication** and **allocation**.

Questions?

http://vittorioromeo.info

vittorio.romeo@outlook.com

http://github.com/SuperV1234

Thank you for attending!