

# Hacking a minimum ECS in Scala 3

張瑋修 Walter Chang

@weihsiu / [weihsiu@gmail.com](mailto:weihsiu@gmail.com)

<https://github.com/weihsiu/secs>

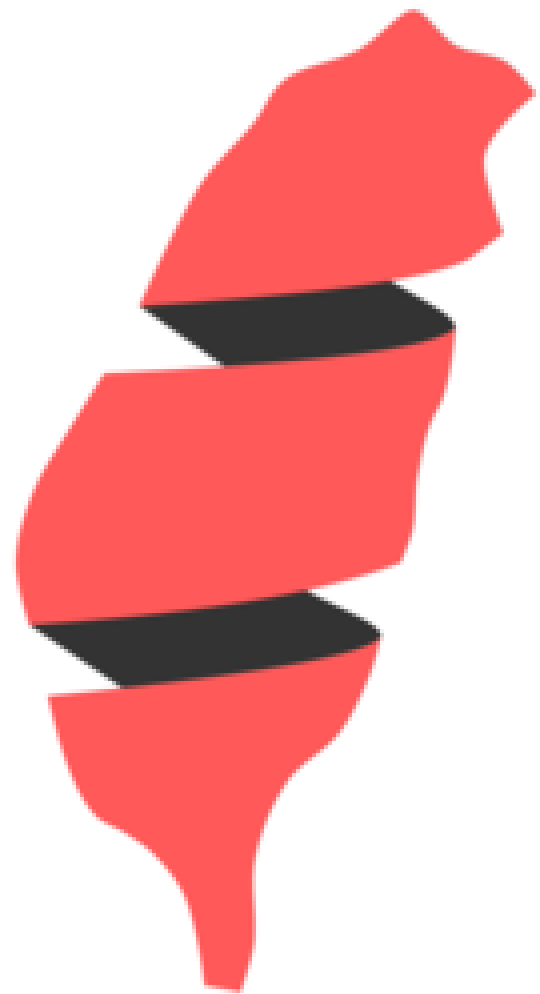


**Scala Taiwan**

**Scala Taiwan gitter channel**

**Scala Taiwan FB group**

**Scala Taiwan meetup**



# Agenda

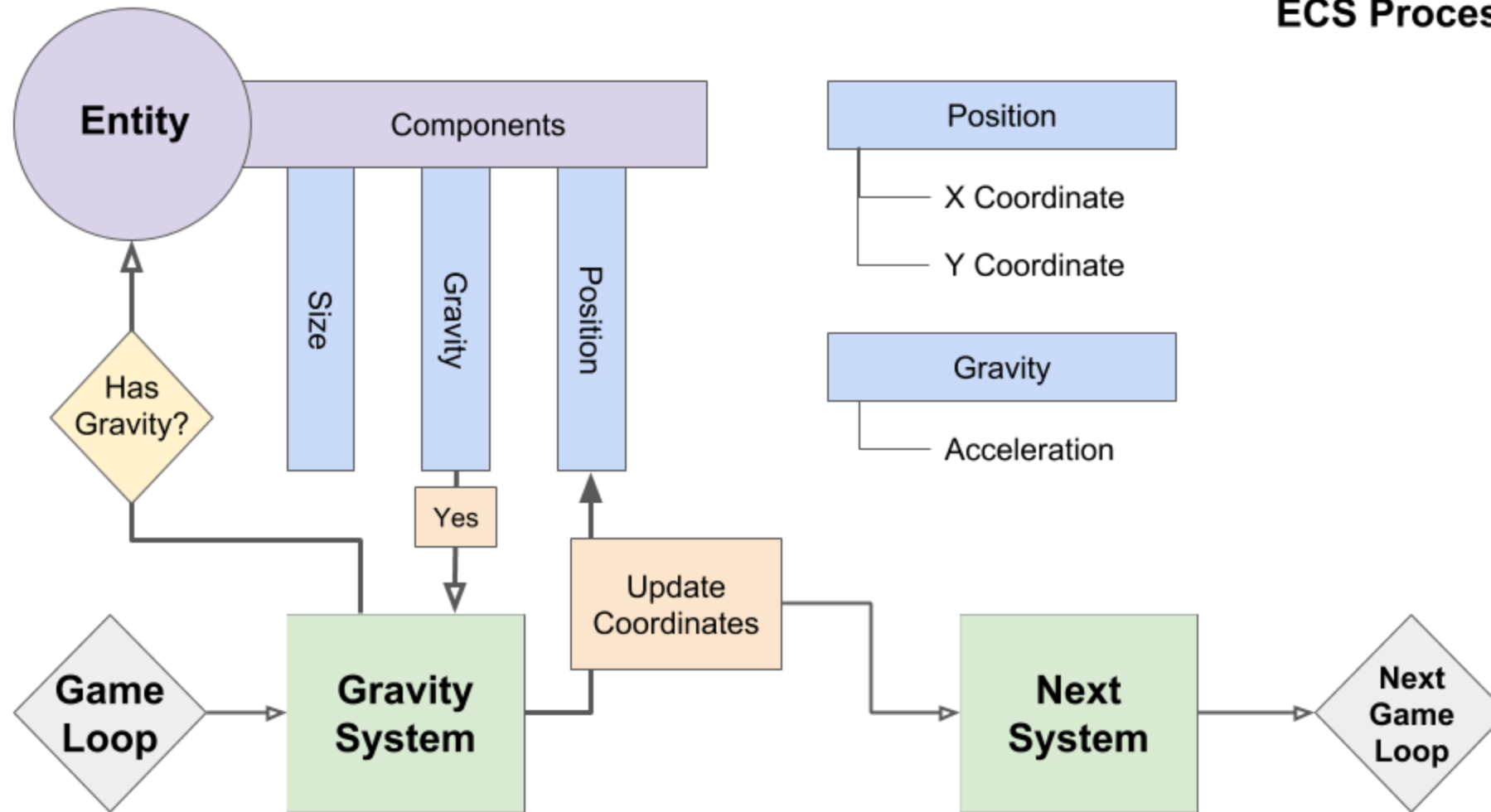
- What is ECS?
- Overview
- Entity
- Component
- System
- Command
- Query
- Filter
- Event
- Secs
- Examples
- Q&A

# What is ECS?

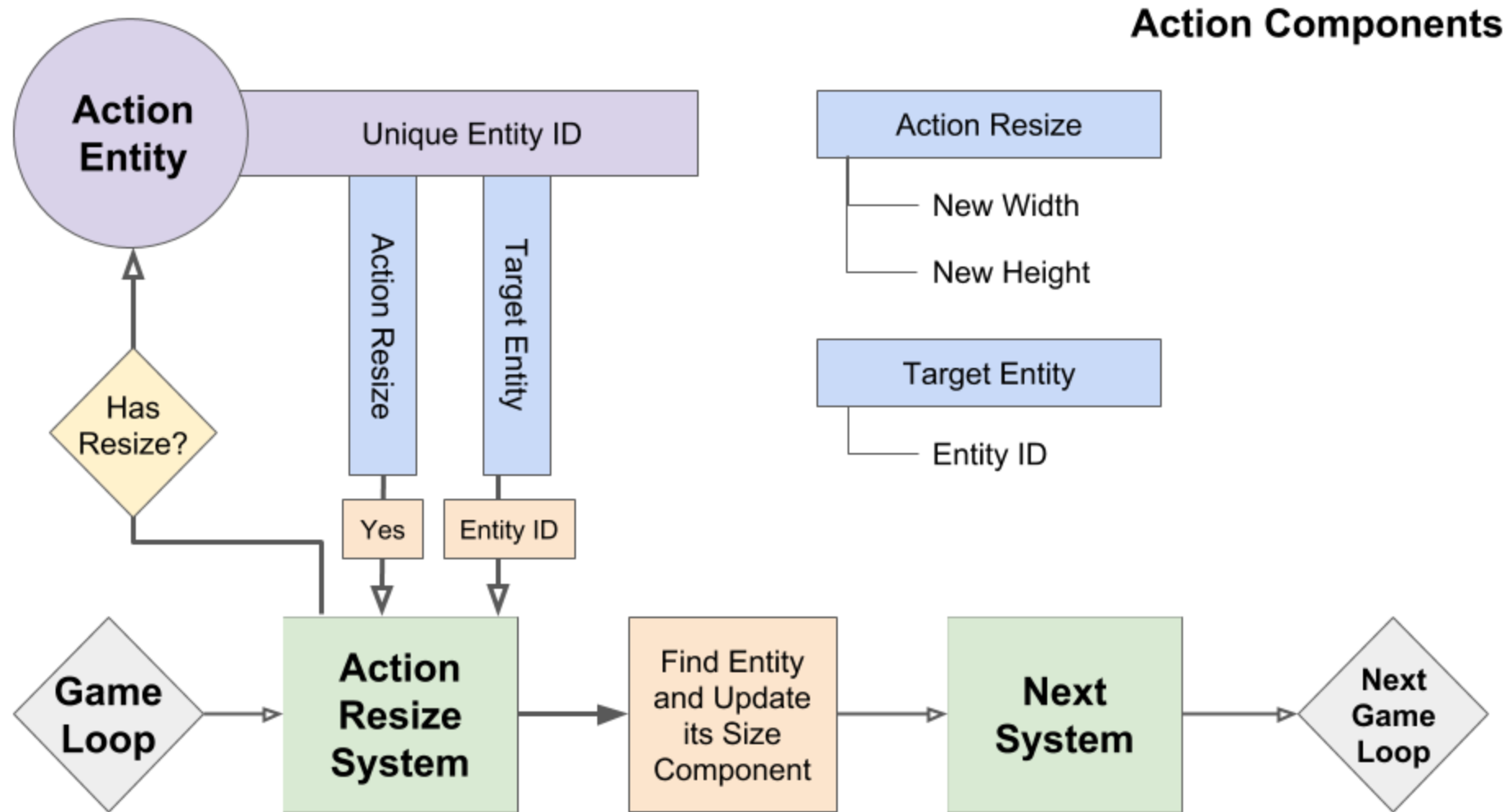
- "Entity–component–system (ECS) is a software architectural pattern that is mostly used in video game development." -- Wikipeida
- It makes manipulating entities with the same types of components (attributes) very simple.

From <https://medium.com/@clevyr/entity-component-system-for-react-js-e3ab6e9be776>

## ECS Process



From <https://medium.com/@clevyr/entity-component-system-for-react-js-e3ab6e9be776>



# Overview

- Scala 3 meta-programming is utilized to make ECS programming simpler and less error-prone
- Meta-programming in Scala 3 is made out of the following features:
  - Typelevel programming
    - Match types
    - `scala.compiletime` package
  - Inline functions
  - Macros (not used in this project)

# Entity

- Just an identifier
- Each entity can have multiple components associated with it

```
opaque type Entity = Int
```



# Component

- Just Scala case classes that extend `Component` (marker trait) and derives `ComponentMeta`.
- Builtin components:
  - `EntityC`
  - `Label`

```
case class Dimension(width: Double, height: Double) extends Component derives ComponentMeta
```

# System

- Just Scala inline functions
- Used to insert/modify/delete components each entity is associated with

```
inline def updateDimensions(using  
    command: Command,  
    query: Query1[(EntityC, Dimension)])  
): Unit = ???
```

# Command

- Used to insert/modify/delete components
- obtained via `using` with system functions

```
trait Command:
  def spawnEntity(): EntityCommand
  def entity(entity: Entity): EntityCommand
  def despawnEntity(entity: Entity): Unit

trait EntityCommand:
  def entity: Entity
  def insertComponent[C <: Component](component: C)(using CM: ComponentMeta[C]): EntityCommand
  def updateComponent[C <: Component](update: C => C)(using CM: ComponentMeta[C]): EntityCommand
  def removeComponent[C <: Component]()(using CM: ComponentMeta[C]): EntityCommand
```

# Query

- Used to select entities with required components
- obtained via `using` with system functions

```
trait Query[CS <: Tuple, OS <: BoolOps]:  
  inline def result: List[CS]
```

```
inline def updateSpaceship(time: Double)(using  
  C: Command,  
  Q: Query1[(EntityC, Direction, Movement, Option[CoolOff])])  
): Unit =  
  Q.result.foreach((e, d, m, c0) =>  
    // e: EntityC  
    // d: Direction  
    // m: Movement  
    // c0: Option[CoolOff]  
    ???  
  )
```

# Filter

- Used to select entities that meet the filtering criteria
- Specified in the second generic parameter of Query

```
inline def system(using Q: Query[EntityC *: EmptyTuple, Dimension  $\wedge$  Added[Rotation]]): Unit = ???
```

# Event

- Builtin components with functions to send and receive events
- Events are produced and consumed all in a single frame

```
case class EventSender[E]() extends Component:  
  def send(event: E)(using CM: ComponentMeta[EventSender[E]], W: World): Unit = W.sendEvent(event)  
  
case class EventReceiver[E]() extends Component:  
  def receive(using CM: ComponentMeta[EventSender[E]], W: World): Iterable[E] = W.receiveEvents
```

# Secs

- To tie everything together, implement `Secs`
- In `init()`, call system functions that will only be called in the very beginning
- In `tick()`, call system functions that will be invoke on every rendering frame

```
enum EntityState:  
    case Spawned, SpawnedAndAlive, Alive, AliveAndChanged, Despawned  
  
trait Secs[SS <: Tuple]:  
    type Worldly = World ?=> Unit  
    def init(): Worldly  
    def tick(time: Double): Worldly  
    def beforeRender(): Unit  
    def renderEntity(  
        entity: Entity,  
        status: EntityState,  
        components: Components,  
        previousComponents: => Components  
    ): Unit  
    def afterRender(): Unit
```

# Examples

- Astroids
  - As an Java app
  - As a browser page
- Retained
  - retained mode operations



## Q&A

That's all and thank you for your attention

<https://github.com/weihsiu/secs>

