Intro to Entity-Component-System

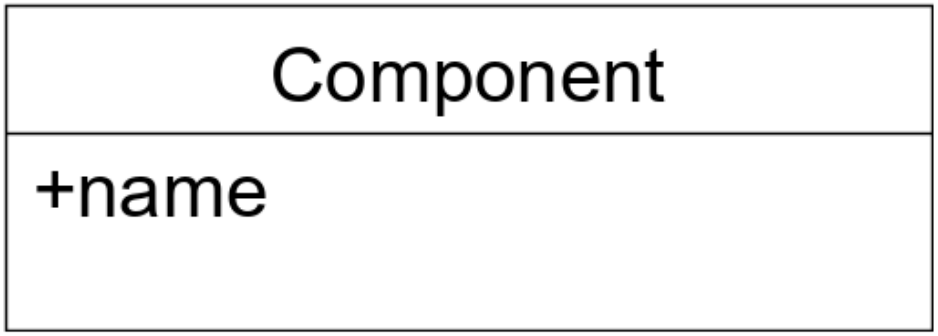
Entity-Component-System

An architectural pattern that favors composition over inheritance

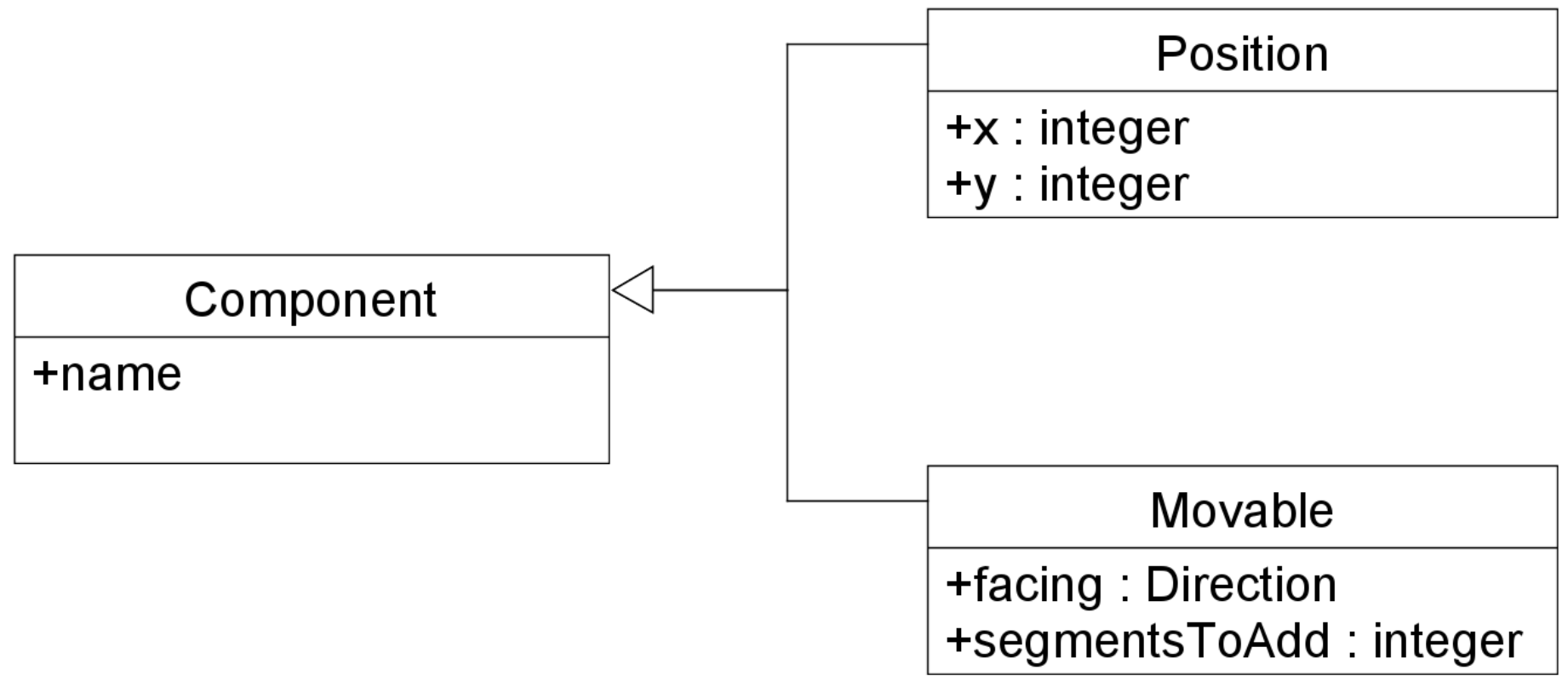
ECS - Defined

- **Entity**: Container that has only a **unique identifier** and a list of components for an object.
  - Only addComponent/removeComponent behaviors
- **Component**: Data for one aspect of an object.
  - Only state, no behaviors
- **System**: Logic that updates the state of related components for all Entities that contain those components.
  - Only behaviors, no states

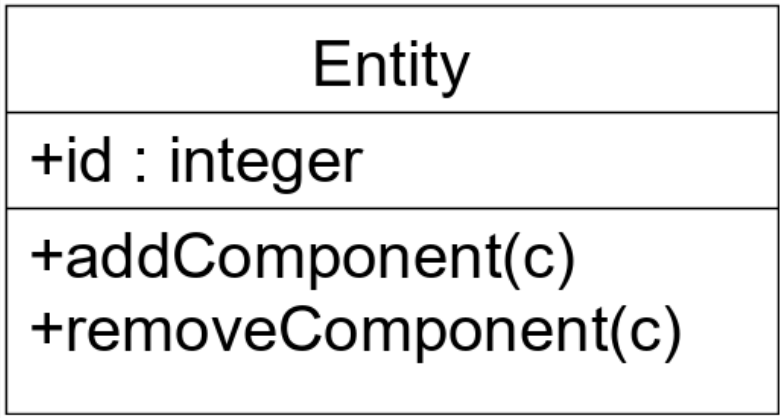
Component



Component


Position
+x : integer
+y : integer
Component
+name
Movable
+facing : Direction
+segmentsToAdd : integer

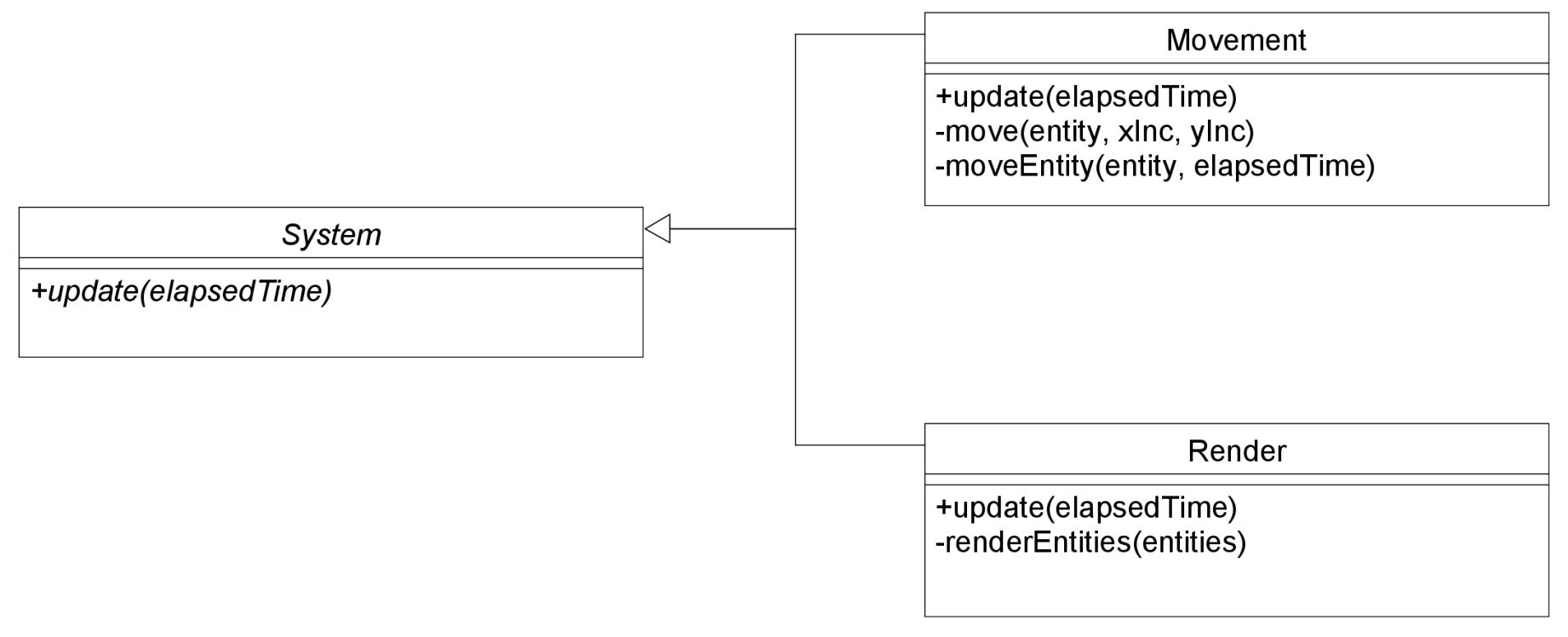
Entity



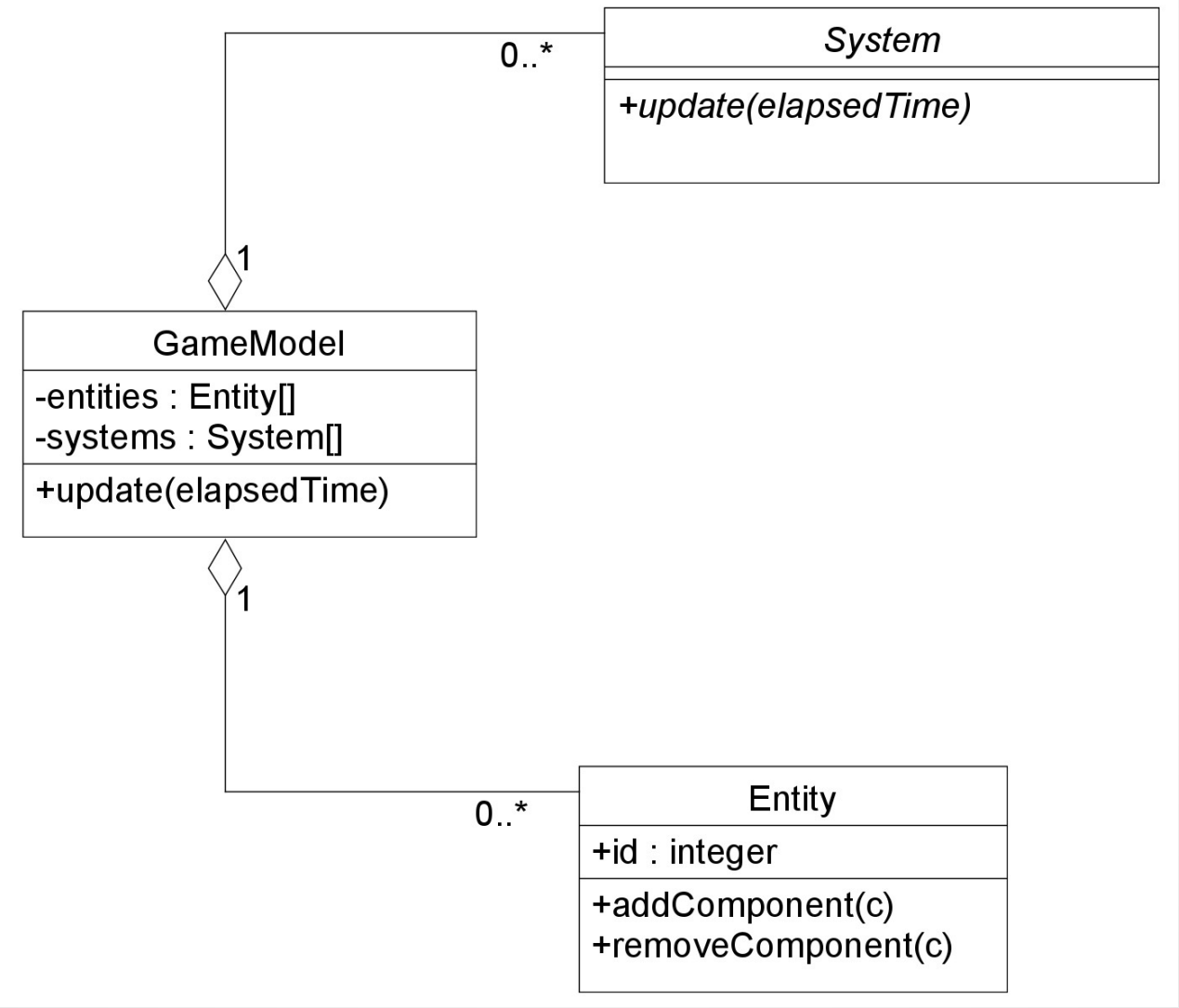
System

System
+*update(elapsedTime)*

System


Movement
+update(elapsedTime)
-move(entity, xInc, yInc)
-moveEntity(entity, elapsedTime)
System
+update(elapsedTime)
Render
+update(elapsedTime)
-renderEntities(entities)

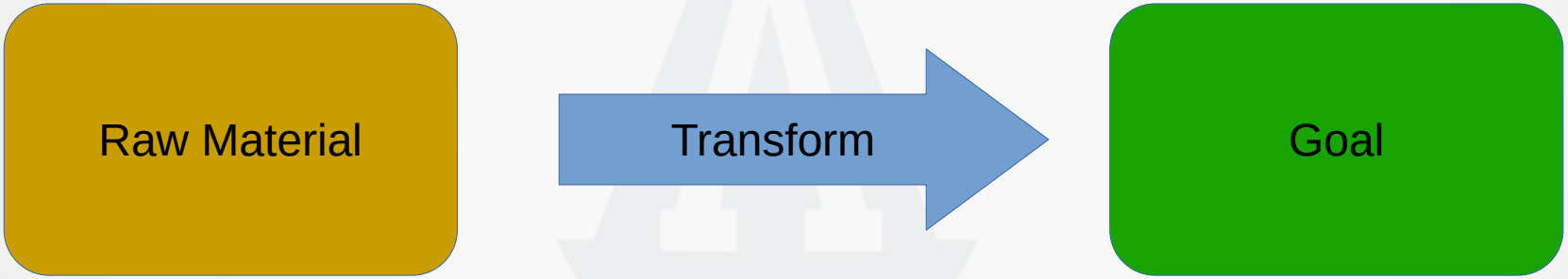
Putting Them Together



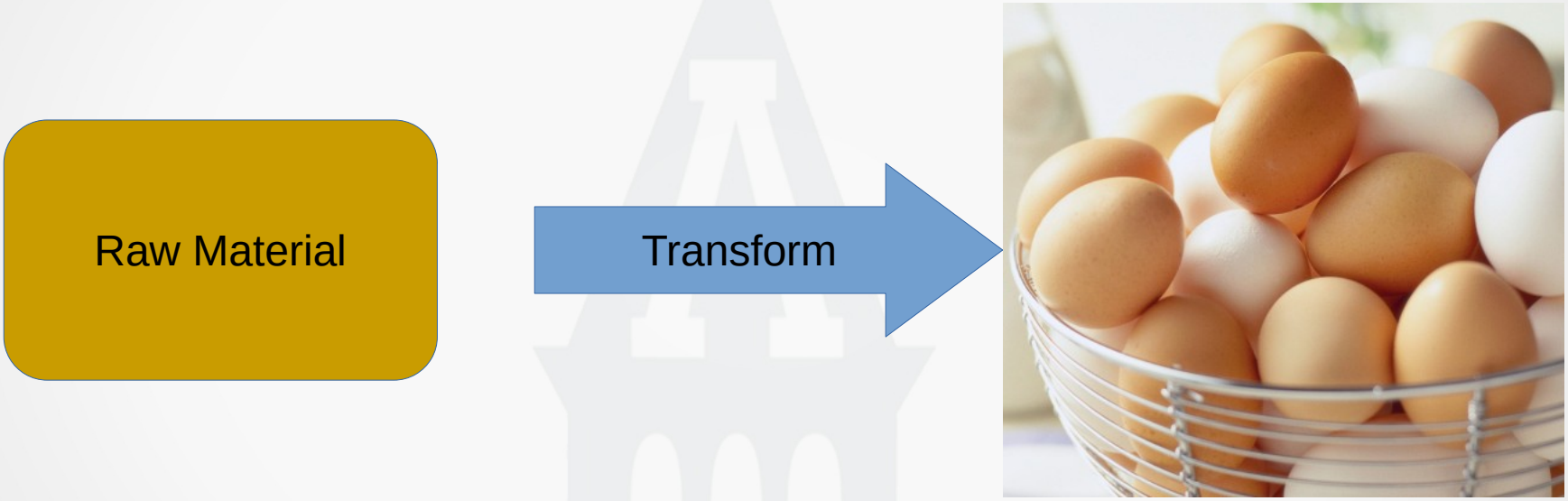
Suggested Benefits of ECS

- Data Oriented Design (next few slides)
  - Improved memory layout, resulting in improved performance
  - Improved potential for task-based parallelism
- Eliminate complications from inheritance hierarchies
- Improve flexibility in creating game objects
  - Can design/modify object at runtime, code not required
  - Designers can experiment with object design
- Faster compile/test/debug because of fewer code dependencies

Data Oriented Design



Data Oriented Design



Data Oriented Design

Transform

Data Oriented Design

Transformation – Movement

position: { x: 0, y: 0},
momentum: { x: 1, y: 0}
facing: { x: -1, y: 0}

(elapsedTime: 1)

Transform

position: { x: 1, y: 0},
momentum: { x: 1, y: 0}
facing: {x: -1, y: 0}

Transformation – Thrust

position: { x: 0, y: 0},
momentum: { x: 1, y: 0}
facing: { x: -1, y: 0}

(elapsedTime: 1)

Transform

position: { x: 0, y: 0},
momentum: { x: 0, y: 0}
facing: {x: -1, y: 0}

Transformation – Render

position: { x: 0, y: 0},
momentum: { x: 1, y: 0}
facing: { x: -1, y: 0}

Transform

ECS Example : JavaScript - Snake Mini-Game

ECS Example : C#/MonoGame - Simple Game

Additional Items

- Defining Dependencies
- Rather than iterating through all components…
  - Organize by type
  - Give all systems a chance to be “interested” in entities

References

- Wikipedia
  - https://en.wikipedia.org/wiki/Entity_component_system
- Professor Porkins – Game Techniques
  - https://github.com/ProfPorkins/GameTech
- ES Wiki
  - http://entity-systems.wikidot.com/es-articles
- T-Machine
  - http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/
- Ash
  - https://www.richardlord.net/ash/
- Erik Hazzard
  - http://www.vasir.net/blog/game-development/how-to-build-entity-component-system-in-javascript
- Gamedev.net
  - https://www.gamedev.net/articles/programming/general-and-gameplay-programming/understanding-component-entity-systems-r3013