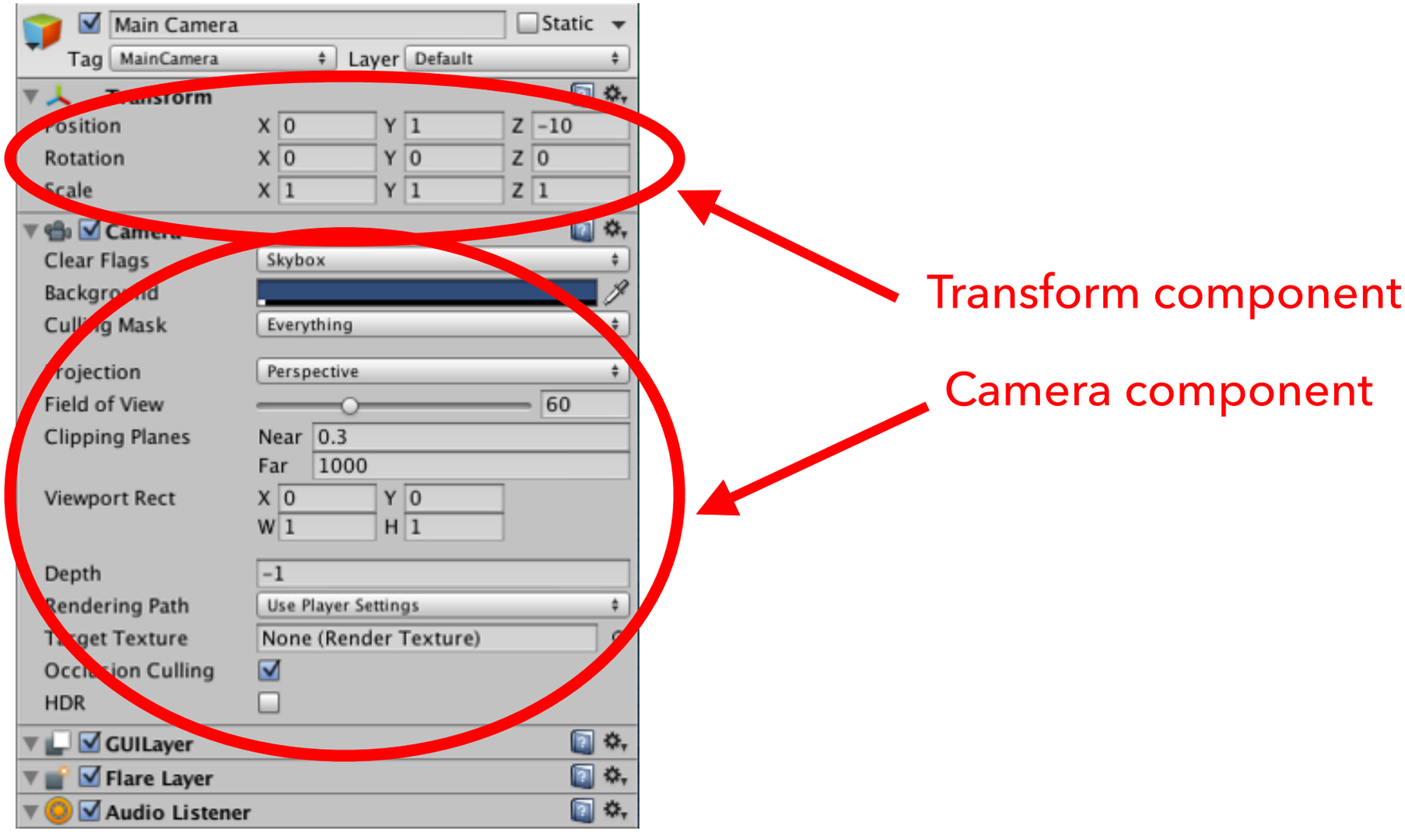
CS354P

DR SARAH ABRAHAM

ENTITY COMPONENT SYSTEMS

COMPONENTS IN UNITY

Example: Main Camera Game Object associated with each scene



UNITY COMPONENT PROPERTIES

- Primary form of interacting with the system
  - Functionality on components themselves
  - Easy to create redundant or inter-dependent systems
- Still object-oriented and inheritance-based
  - Memory management handled by GameObject
  - Not fully data-driven

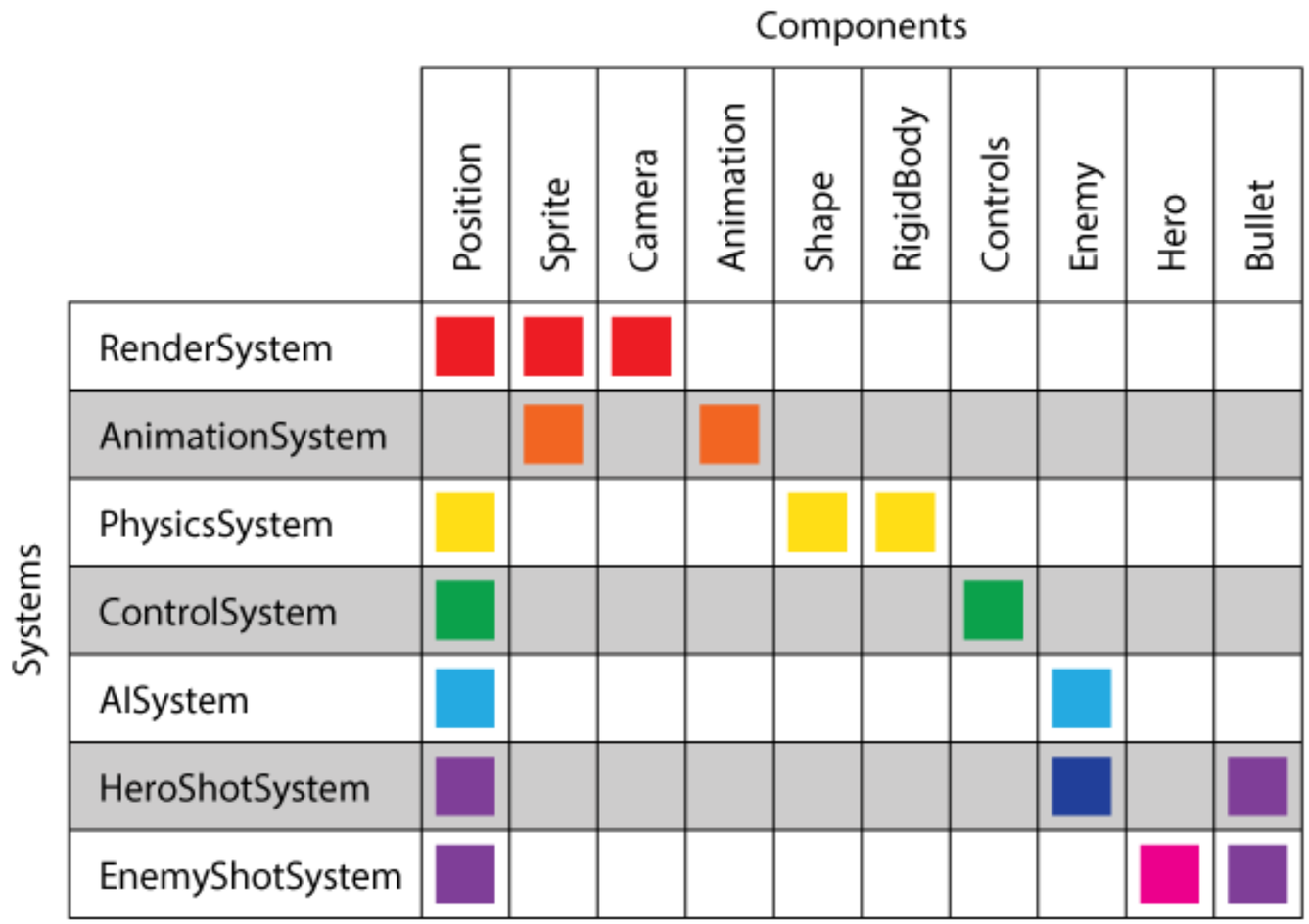
WHAT DOES DATA-DRIVEN MEAN?

ENTITY COMPONENT SYSTEMS (ECS)

- A form of component-based architecture where all functionality comes from the components
- Entity is an id
- Entity data stored in components
- Systems modify related components

ECS VS OOP

- Key difference: entity does not control or organize components in ECS
- Objects have properties and behaviors
  - Model resembles the real-world concept of objects
- Entities are purely a container class
  - Model resembles a relational database



(http://www.alecmce.com/)

UNITY DOTS

- Data-oriented Technology Stack (DOTS) intended to make Unity competitive in the Triple A space
  - Better architecture for managing large-scale projects
  - Better support for multi-threading
  - Better compilation
- Move from MonoBehaviour-based system to an ECS system

DOTS ENTITIES

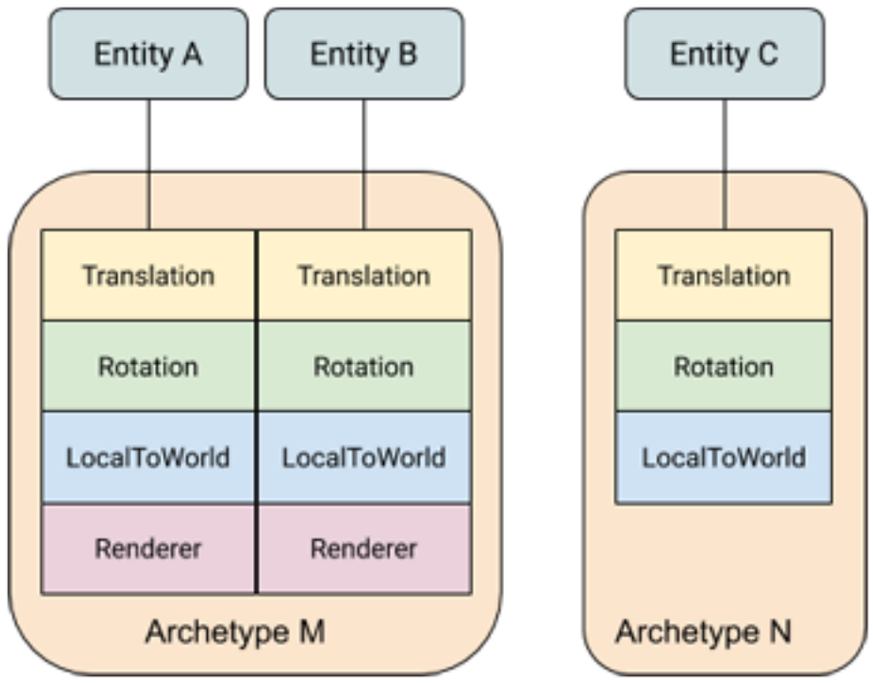
- Entities are IDs that are stored per-world by an EntityManager
- EntityManager maintains list of all entities
  - Determines how to process entities to optimize performance
    - Creates **EntityArchetypes** based on components associates with entities
- EntityArchetype structs allow the creation and reuse of particular combinations of components
- GameObjects and Prefabs converted to entities at runtime
- Possible to also create entities directly using `Instantiate()` and `CreateEntity()`

DOTS COMPONENTS

- Components contain data related to a particular behavior
- Implemented as a struct with variable data
  - Still a data container -- behaviors exists within Systems only
- Component structs use interfaces based on type of data and data needs:
  - IComponentData, IBufferElementData, ISharedComponentData, ISystemStateComponentData, ISharedStateComponentData

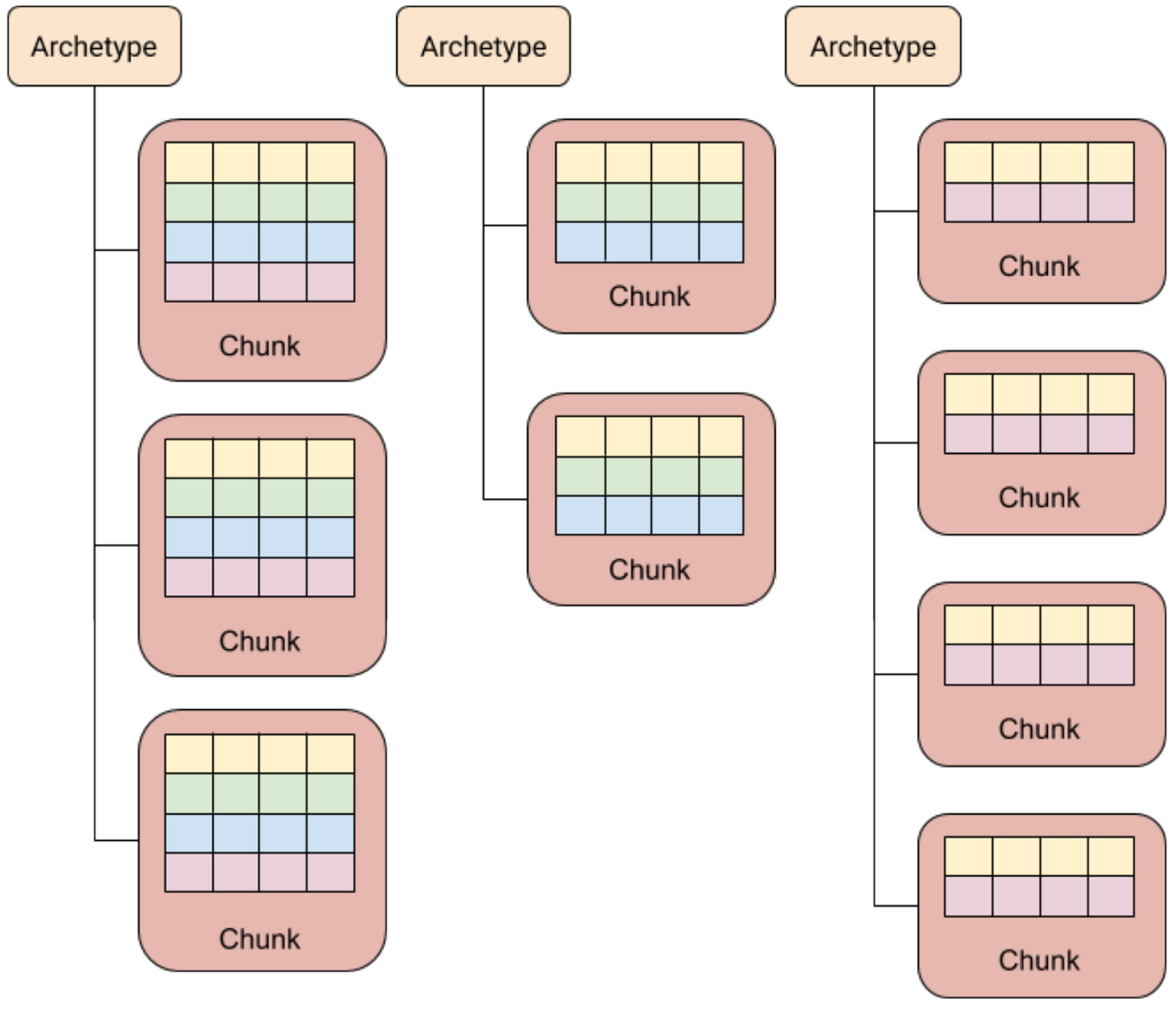
ARCHETYPES AND MEMORY LAYOUT

- Archetypes group components based on entities
- Entities with same archetype stored in chunks
- Changes to components at runtime changes an entity's archetype and where it is stored



CHUNKS AND MEMORY LAYOUT

- Chunks always contain entities of single archetype
- Chunk memory allocated dynamically
- Allows for one-to-many relationship for querying
- ECS generally requires flat, cache-friendly data layout to get good performance



SOME ADDITIONAL TYPES OF COMPONENTS

- Chunk components contain data applied to all entities in a chunk
- Shared components allow entities to be stored with other entities that have the same value
- Both chunk and shared component data stored outside of chunk
  - Allows the reuse of a single component instance across the chunk

DOTS SYSTEMS

- Systems that perform actual logic on component data
- Systems automatically discovered and instantiated at runtime
  - Organized into groups within the world
- Two basic types of systems provided depending on the intended functionality:
  - Component System and Job Component System

COMPONENT SYSTEMS VS JOB COMPONENT SYSTEM

- Component Systems perform work on the main thread and not specifically optimized for ECS
  - Behaves similarly to old-style Unity Component (but only contains methods)
- Job Component Systems perform work on components in parallel
  - Behaves in expected ECS way

TYPES OF JOBS

- Systems kick off jobs to iterate over entities/components
- Job types provided based on usage and performance requirements
  - IJobForEach, IJobForEachWithEntity, IJobChunk, IJobParallelFor, etc...
- Possible to access specific data using EntityQueries
  - Allow running jobs specifically for those entities/components

HANDLING JOB DEPENDENCIES

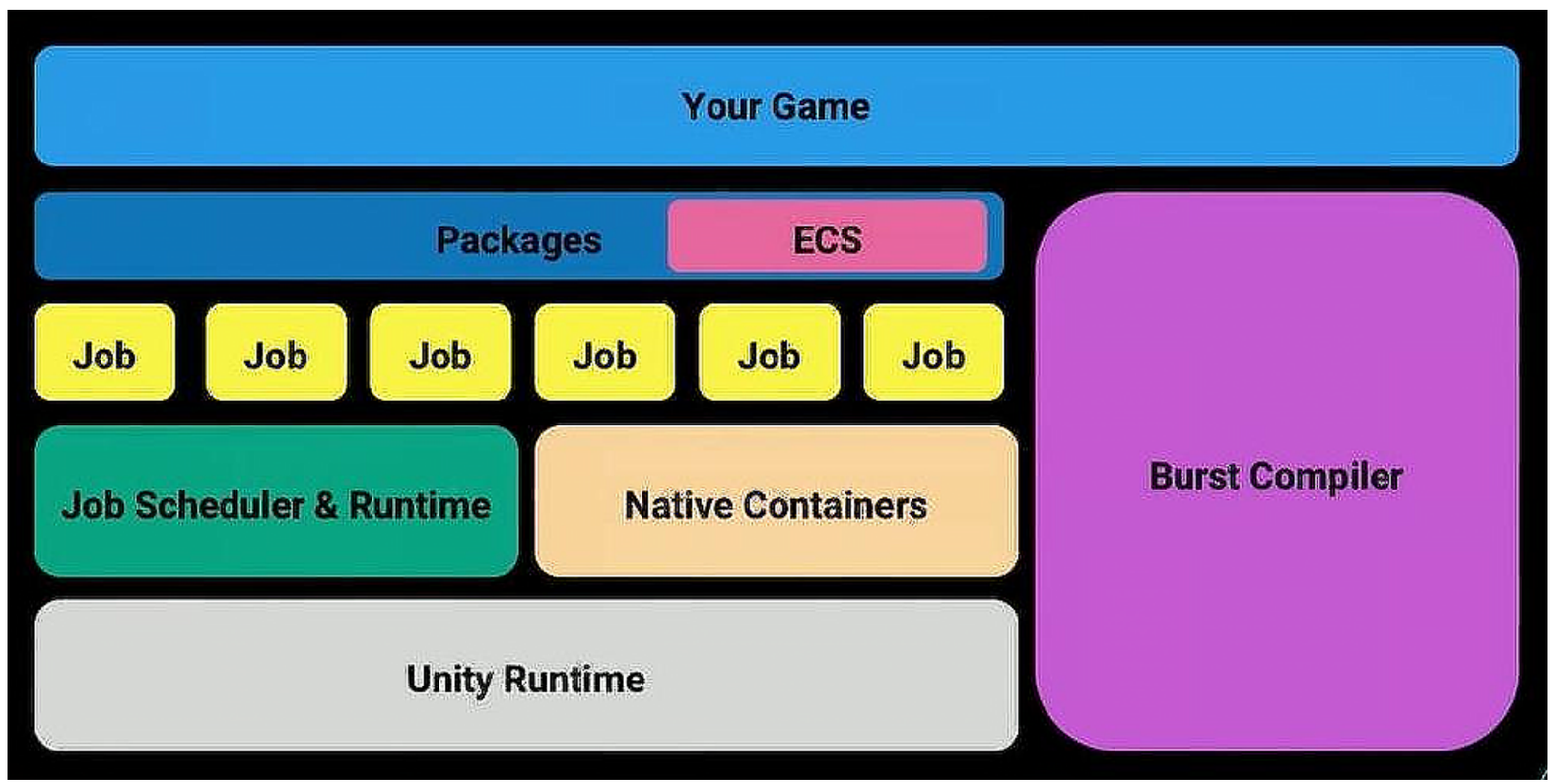
- When data is read-only, system jobs are embarrassingly parallel, but writes require dependencies
- Job Handle created per-world to schedule jobs based on read/write dependencies within data
- Dependency graph created automatically for any Job Component System

SYSTEM EVENTS

- Events supported per-system for entity life-cycle management
  - OnCreate(), OnStartRunning(), OnUpdate(), OnStopRunning(), OnDestroy()
- All events executed on the main thread
  - Can schedule jobs on background threads from OnUpdate()

DOTS STACK

- ECS is only one part of the greater DOTS framework
  - Job scheduler essential
  - Burst compiler allows for further optimizations



C# JOB SYSTEM

- C# Job System allows for the writing of safe, multi-threaded code
- Integrated into Unity's native job system for better pooling
- Jobs sent copy of data rather than reference to data to prevent race conditions
  - Uses "blittable" types to avoid conversion overhead
  - Blittable types can be safely copied using memcpy

WORKING WITH JOBS

- Jobs scheduled on the main thread using `Schedule()`
  - Once called, job cannot be interrupted
- Results of job should be stored in a `NativeContainer` so that it is accessible by both job thread and main thread
- `Complete()` called when results are needed
  - If job is not completed when this function is called, will block
- `Dispose()` frees memory allocated by result

BURST COMPILER

- Recall: C# is compiled by .NET's CLR (Common Language Runtime) VM
  - First compiled into IL (Intermediate Language) then JIT (Just-in-time) compiled into machine code at runtime
- Burst compiler uses LLVM to translate IL into machine code for greater efficiency
  - Works well with ECS job model

LLVM

- Infrastructure for cross-platform compilation and toolchain technologies
- Written in C++ but designed to be language-independent
  - Supports compiling of Rust, Ada, Haskell, Swift, Lua, Fortran etc...
- LLVM Intermediate Representation (IR) is low-level language
  - Strongly-typed RISC (Reduced Instruction Set) language
  - Three equivalent forms of IR: in-memory compiler, on-disk bitcode, human-readable
- Highly optimizable, flexible, and powerful system for compilation

ECS PROS

- Can be more memory-efficient
  - Only store properties in use, no unused data members in objects
- Easier to construct in a data-driven way
  - Define new attributes with scripts, less recoding of class definitions
- Can be more cache-friendly
  - Data tables loaded into contiguous locations in cache
  - Struct of arrays (rather than array of structs) principle

ECS CONS

- Hard to enforce relationships among properties
- Harder to implement large-scale behaviors
  - Composed of scattered pieces of fine-grained behavior
- Harder to debug
  - Can’t just put a game object into a watch window in the debugger and see what happens to it

WHAT IS THE RIGHT MODEL?

- ECS is a very low-level model
- Requires thinking extensively about memory management
  - Job dependencies, caching properties, etc
- Works well if the engine mostly handles this for game developers
  - Trade-off between convenience and performance otherwise

DOMAIN-SPECIFIC LANGUAGES AND PROGRAMS

- Domain-specific languages (DSLs) are languages intended for specific use-cases rather than general-purpose programming
  - Examples: Matlab, Mathematica, YACC, SQL, etc
- Domain-specific programs are the same concept applied to a greater system
  - Provide high-level interface with potential for low-level optimizations
  - But not intuitive to non-expert users...

FURTHER READING

- Unity ECS Overview <https://docs.unity3d.com/Packages/com.unity.entities@0.1/manual/index.html>
- Unity at GDC <https://www.youtube.com/watch?v=kwnb9Clh2Is&t=1s>
- Unity Burst Compiler <https://docs.unity3d.com/Packages/com.unity.burst@1.2/manual/index.html>