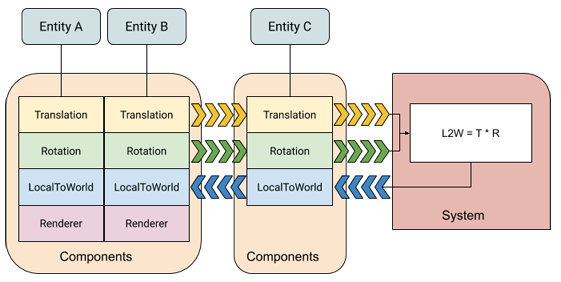
**ECS concepts**

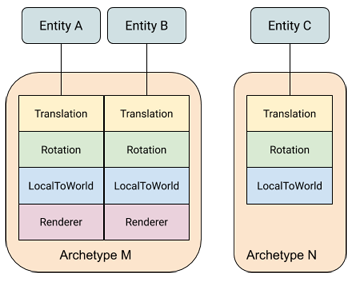
The following diagram illustrates how these three basic parts work together:



In this diagram, a system reads Translation and Rotation components, multiplies them and then updates the corresponding LocalToWorld components (L2W = T\*R).

**Archetypes**

A unique combination of component types is called an [Archetype](xref:Unity.Entities.Archetype).

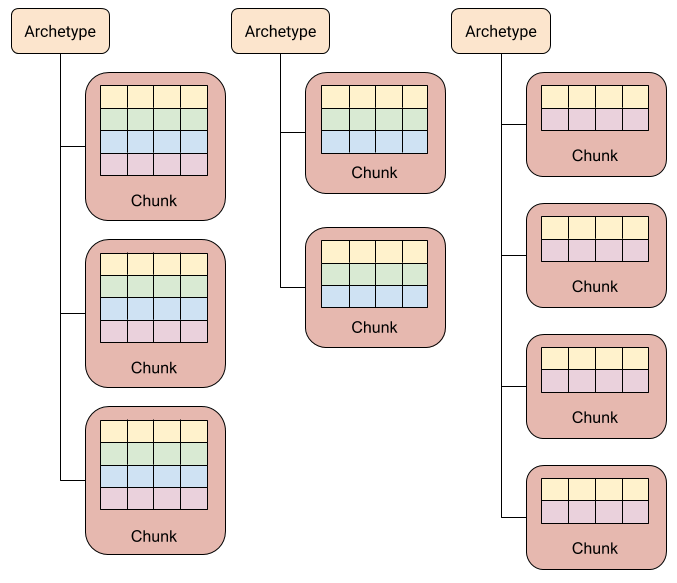


In this diagram, entities A and B share archetype M, while entity C has archetype N.

To smoothly change the archetype of an entity, you can add or remove components at runtime. For example, if you remove the Renderer component from entity B, it then moves to archetype N.

**Memory Chunks**

A chunk always contains entities of a single archetype. When a chunk of memory becomes full, ECS allocates a new chunk of memory for any new entities created with the same archetype. If you add or remove components, which then changes an entity archetype, ECS moves the components for that entity to a different chunk.



This organizational scheme provides a one-to-many relationship between archetypes and chunks. It also means that finding all the entities with a given set of components only requires searching through the existing archetypes, which are typically small in number, rather than all of the entities, which are typically much larger in number.

ECS does not store the entities that are in a chunk in a specific order. When an entity is created or changed to a new archetype, ECS puts it into the first chunk that stores the archetype, and that has space. Chunks remain tightly packed, however; when an entity is removed from an archetype, ECS moves the components of the last entity in the chunk into the newly vacated slots in the component arrays.

**Entity queries**

To identify which entities a system should process, use an [EntityQuery](https://docs.unity3d.com/Packages/com.unity.entities@0.9/api/Unity.Entities.EntityQuery.html). You can specify the following component requirements with a query:

* **All** — the archetype must contain all of the component types in the **All** category.
* **Any** — the archetype must contain at least one of the component types in the **Any** category.
* **None** — the archetype must not contain any of the component types in the **None** category.

An entity query provides a list of the chunks that contain the types of components the query requires. You can then iterate over the components in those chunks directly with [IJobChunk](https://docs.unity3d.com/Packages/com.unity.entities@0.9/manual/chunk_iteration_job.html).

**Jobs**

To take advantage of multiple threads, you can use the [C# Job system]. ECS provides the [SystemBase](xref:Unity.Entites.SystemBase) class, along with the Entities.ForEach and [IJobChunk](https://docs.unity3d.com/Packages/com.unity.entities@0.9/manual/chunk_iteration_job.html) Schedule() and ScheduleParallel() methods, to transform data outside the main thread. Entities.ForEach is the simplest to use and typically requires fewer lines of code to implement. You can use IJobChunk for more complex situations that Entities.ForEach does not handle.

ECS schedules jobs on the main thread in the [order that your systems are arranged](https://docs.unity3d.com/Packages/com.unity.entities@0.9/manual/ecs_core.html#system-organization). As jobs are scheduled, ECS keeps track of which jobs read and write which components. A job that reads a component is dependent on any prior scheduled job that writes to the same component and vice versa. The job scheduler uses job dependencies to determine which jobs it can run in parallel and which must run in sequence.

**System organization**

ECS organizes systems by [World](https://docs.unity3d.com/Packages/com.unity.entities@0.9/api/Unity.Entities.World.html) and then by [group](xref:Unity.Enties.ComponentSystemGroup). By default, ECS creates a default World with a predefined set of groups. It finds all available systems, instantiates them, and adds them to the predefined [simulation group](https://docs.unity3d.com/Packages/com.unity.entities@0.9/api/Unity.Entities.SimulationSystemGroup.html) in the default World.

You can specify the update order of systems within the same group. A group is a kind of system, so you can add a group to another group and specify its order just like any other system. All systems within a group update before the next system or group. If you do not specify an order, ECS inserts systems into the update order in a deterministic way that does not depend on creation order. In other words, the same set of systems always updates in the same order within their group even when you don't explicitly specify an order. [Entity component buffer systems](xref:Unity.Entities.EntityComponentBufferSystem)

System updates happen on the main thread. However, systems can use jobs to offload work to other threads. [SystemBase](xref:Unity.entities.SystemBase) provide a straightforward way to create and schedule Jobs.

**ECS authoring**

When you create your game or application in the Unity Editor, you can use GameObjects and MonoBehaviours to create a conversion system to map those UnityEngine objects and components to entities. For more information, see [Creating Gameplay](https://docs.unity3d.com/Packages/com.unity.entities@0.9/manual/gp_overview.html).

记忆：

A unique combination of component types is called an [Archetype](xref:Unity.Entities.Archetype).

A chunk always contains entities of a single archetype.

ECS organizes systems by [World](https://docs.unity3d.com/Packages/com.unity.entities@0.9/api/Unity.Entities.World.html) and then by [group](xref:Unity.Enties.ComponentSystemGroup).