技术分享

ECS的存储结构和内存管理

赵辉

2023.3.31

目录

一、名词术语

1. Cache/Cache Miss

Cache指的是CPU Cache
CPU要访问的数据在Cache中有缓存，称为“命中”（Hit），反之则称为“缺失”（Miss）

2. ECS

降低耦合，使代码逐渐庞大的过程中保持条理清晰，减少CPU Cache Miss

E：Entity 实体 ID 代表身份
C：Component 组件 数据 不包含任何的逻辑 只有数据
S：System 系统 逻辑 行为

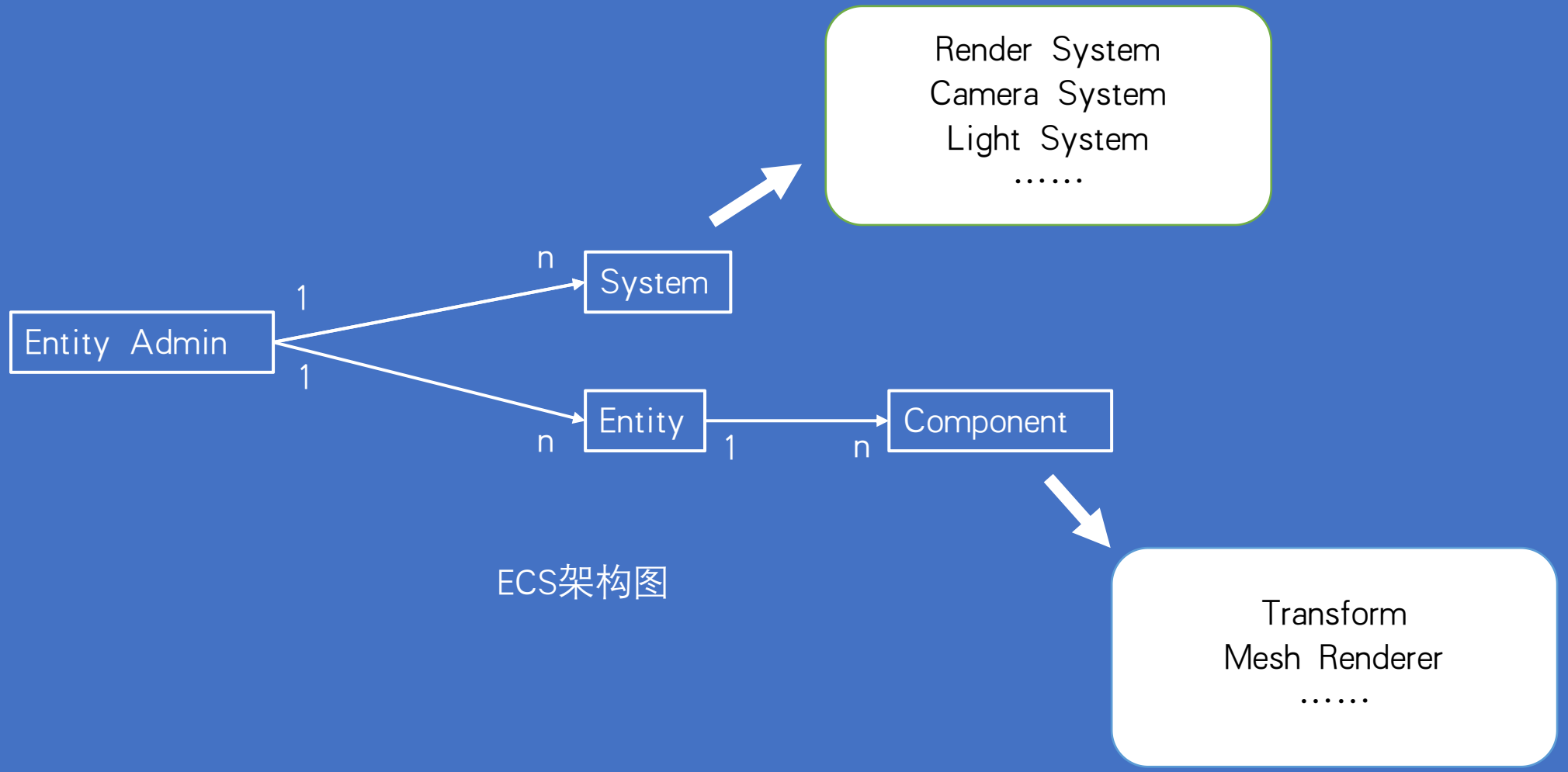
3. Archetype

原型：由多个Component组件描述的一种实体类型，一系列Component的组合就是Archetype

4. Chunk

一个固定大小的内存块，存储着Archetype类型的数据，一个块只能存一种类型

一、名词术语



ECS架构图

二、存储结构

1. 游戏循环

ECS中的游戏循环就是将各个System按顺序执行Update，在Update中遍历Entity，Entity应该带有各自System所关注的Component。

System怎么知道要遍历哪些Entity呢？
怎么知道这个Entity所带的组件里有此System所需要的组件呢？
或者说如果为Entity和System做标记？

二、存储结构

1.1 使用Bit数组

ID_COM_A 9	ID_COM_B 8	ID_COM_C 7	ID_COM_D 6	ID_COM_E 5	ID_COM_F 4	ID_COM_G 3	ID_COM_H 2	ID_COM_I 1	ID_COM_J 0

ID_COM_A ———— COM_A

ID_COM_B ———— COM_B

ID_COM_C ———— COM_C

…… ……

组件ID ——一对应 组件

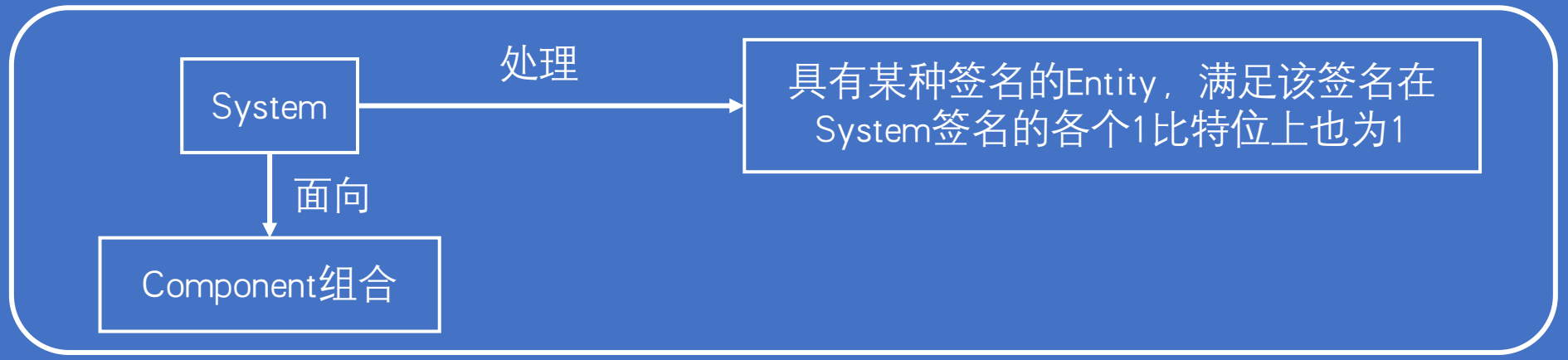
例：如果一个Entity有组件ACDG，那么他的签名 就是

10011001000

如果一个System关注的组件是CD，那么这个系统的签名就是

0011000000

将System的签名和Entity的签名比对就知道Entity符不符合所需要的组件列表啦。



二、存储结构

1.2 使用组件哈希

HASH_COM_A ———— COM_A

HASH_COM_B ———— COM_B

HASH_COM_C ———— COM_C

HASH_COM_D ———— COM_D

…… ……

组件HASH 一一对应 组件

例：如果一个Entity有组件ACDG，那么他的签名 就是

Set<HASH_COM_A \C \D \G>

如果一个System关注的组件是CD，那么这个系统的签名就是

Set<HASH_COM_C \D>

将System的Set集合进行遍历，判断Entity有没有相应的组件Hash就知道Entity符不符合所需要的组件列表了。

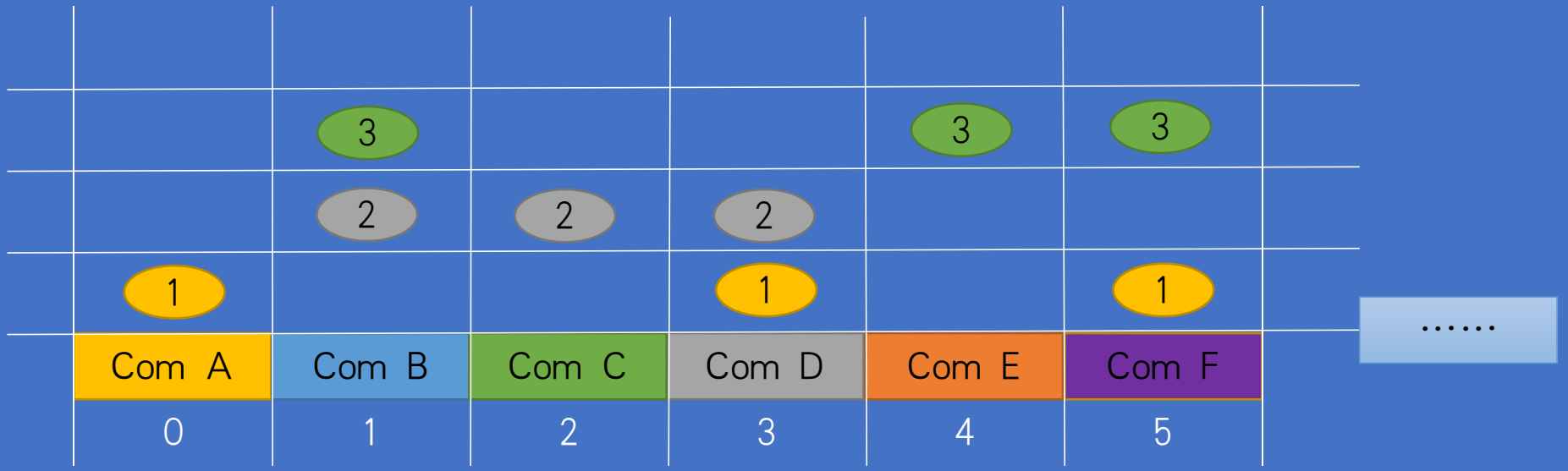
优点：可以不受制于Component的数量

缺点：需要更多的存储空间

二、存储结构

2. 数据存储

2.1 按Component存储



如图所示，假设有ABCDEF6种组件，有1、2、3Entity，1有A、D、F Component，
2有B、C、D Component，3有B、E、F Component
他们在内存中的存储就是图中的网格，网格的宽度为组件的数量，高度为Entity的数量

问题1. 内存浪费，网格里的空缺就是申请了但没有使用的内存
2. 组件过多的时候一个Entity所拥有的组件跨度太大超过了CPU Cache的大小，就会造成Cache Miss

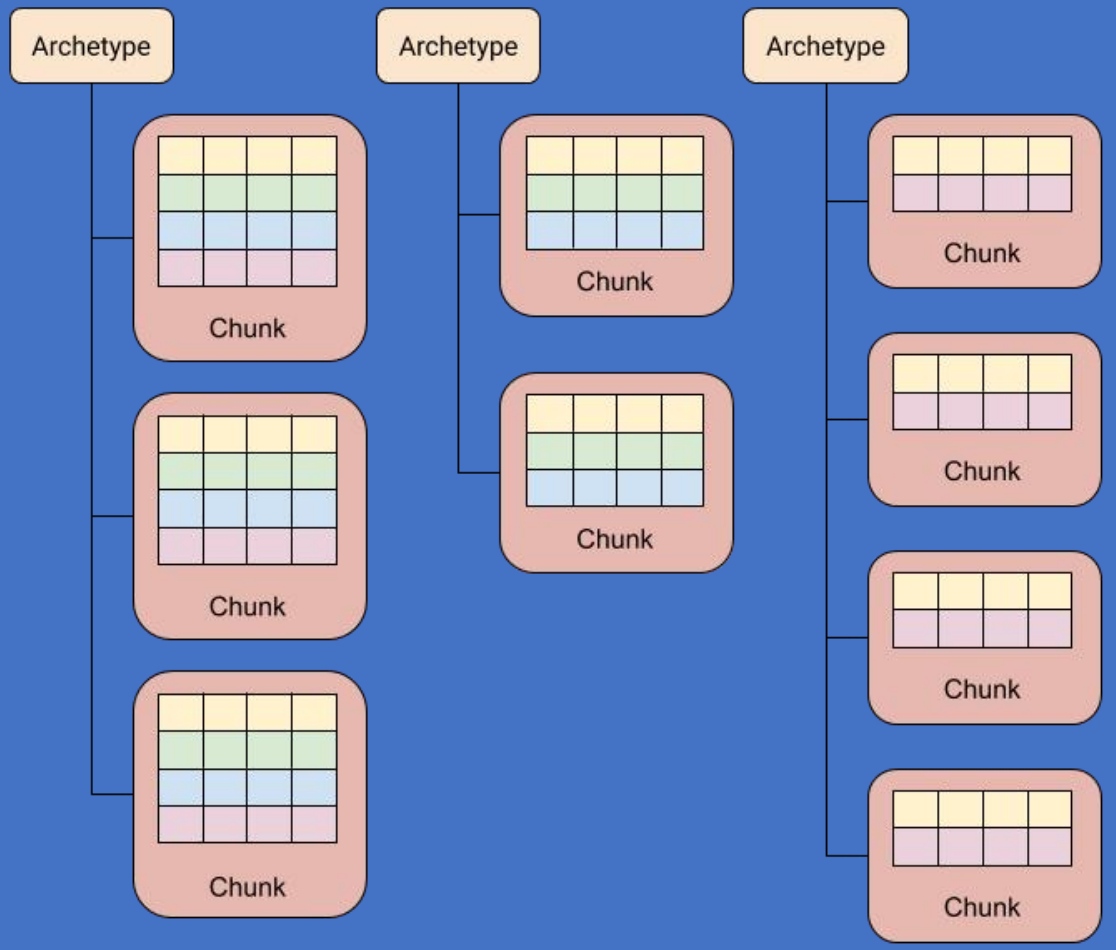
二、存储结构

2. 数据存储

2.1 按Archetype存储

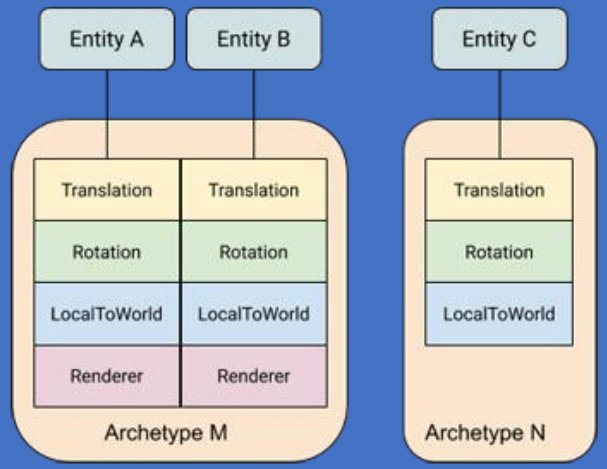
Unity中ECS的数据存储

同一种颜色的是同一种Component
将内存分为多个大小相同的Chunk
每个Chunk存储的Archetype的最大数量相同



Unity中的存储结构

每一竖列是一个Entity所有的Component



二、存储结构

2. 数据存储

2.3 总结

Component的存储结构，既要满足同一Archetype的数据不要分散，又要满足同种Component存在一起顺序排列
所以~~按Component存储~~，采用Unity中ECS的数据存储方法，按Archetype存储

即AAAABBBB式存储，而非ABABABAB式
因为System面向的是Component而不是Archetype，一个Archetype里的组件只有部分用得到，比如ABABABAB的例子，如果System只关注A那么ABABABAB方式存的就内存不连续了。
存储上每一个Archetype都需要一个内存池，所有内存池采用同一种内存分配策略

三、内存分配器

内存分配器用于给新创建的Entity分配内存

1.1 内存分配的功能
（1）创建Entity（2）获取Entity的某个Component（3）释放

1.2 接口设计

（1）初始化，预申请一个Chunk
```
template<typename TypeList>
bool Init()
```

（2）申请一个Entity的内存
```
ChunkHandle* Allocate()
```

（3）在handle上创建T类型的Component
```
template<typename T, typename ...Args>
T* Create(ChunkHandle* handle, Args&&... args)
```

(4)在handle上获取T类型的Component
```
template<typename T>
T* Get(ChunkHandle* handle)
```

（5）释放handle
```
void Free(ChunkHandle* handle)
```

注：Type List：等同于Archetype，是Component的类型集

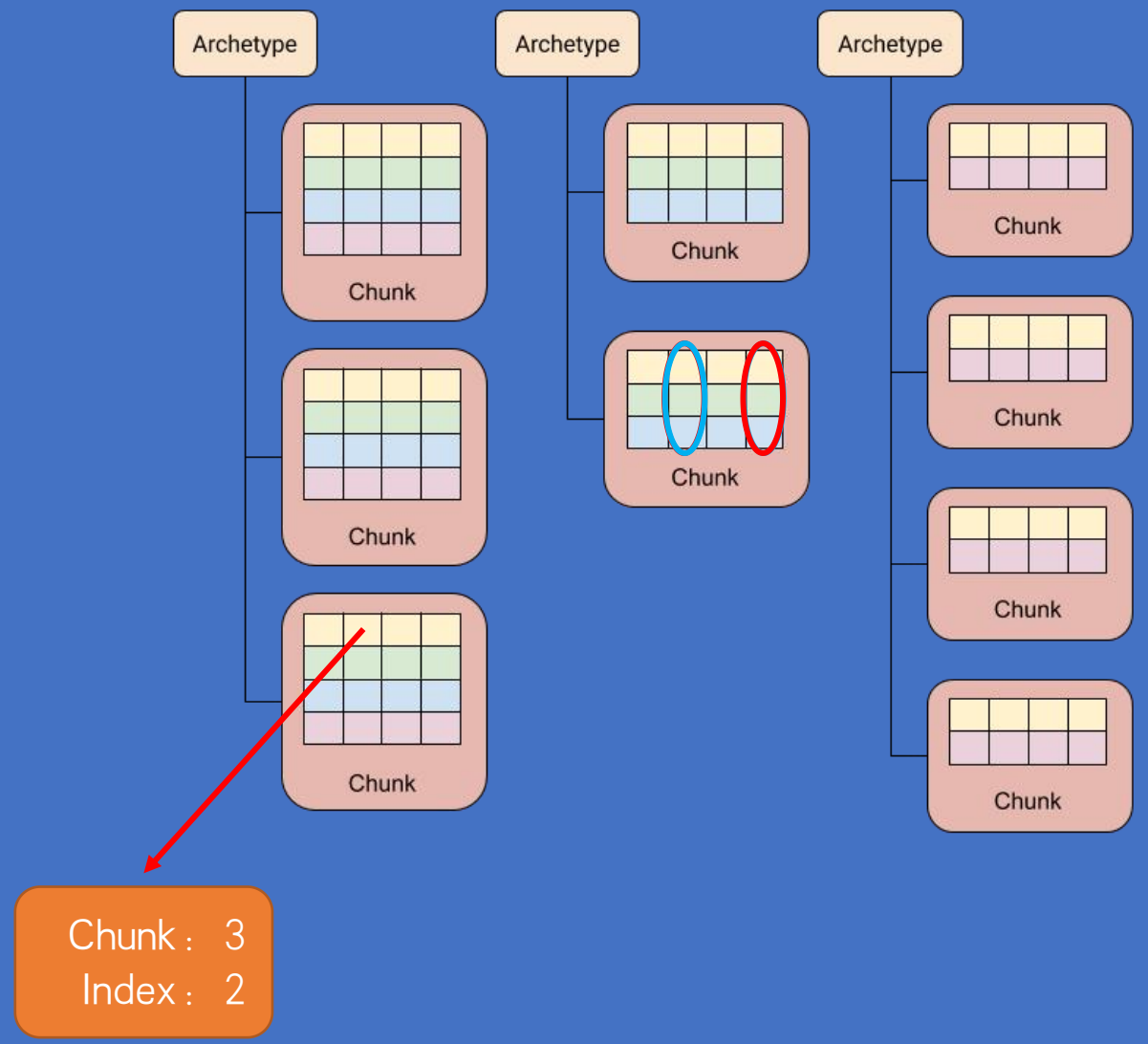
三、内存分配器

1.3 内存分配器可以分为分配器和增长器两部分

1.3.1 增长器
增长器负责给到分配器两个数值
（1）预分配Chunk大小
（2）在Chunk不够放下一个Entity的时候，新Chunk的大小
通常这两个值是相等的

1.3.2 分配器
（1）Pop Pop出一块内存给Entity
返回一个Handle，Handle记录了是哪个Chunk，在这个Chunk的第几个格子里，*Handle永远是新Chunk的第一个或者不满的Chunk的第一个空闲位置*

（2）Push 把Pop出去的内存再返还
需要将这个Chunk最后面的Entity替换到需要Pop的Entity的位置然后释放最后一个的内存



https://github.com/TobeyChao/LaughingEngine

https://docs.unity3d.com/Packages/com.unity.entities@1.0/manual/index.html

游戏编程精粹 7 1.2 高性能堆分配器
游戏编程精粹 5 1.11 用基于Policy的设计改进FreeList
游戏编程精粹 4 1.5 利用模板化的空闲块列表克服内存碎片问题
游戏编程精粹 3 1.6 定制STL分配器
游戏编程精粹 2 1.10 一个插入式内存管理器

谢谢