

Arquitetura e Gerenciamento de Recursos

Slides por:
Gustavo Ferreira Ceccon (TEDJE - FoG - ICMC, 2017)





Este material é uma criação do
Time de Ensino de Desenvolvimento de Jogos
Eletrônicos (TEDJE)

Filiado ao grupo de cultura e extensão
Fellowship of the Game (FoG), vinculado ao
ICMC - USP

Este material possui licença CC By-NC-SA. Mais informações em:
<https://creativecommons.org/licenses/by-nc-sa/4.0/>



Objetivos

- Representação e acesso de objetos a memória
- Aumentar performance de um jogo



Índice

1. Gerenciamento de Memória
2. Gerenciamento de Processo



1. Gerenciamento de Memória



1. Gerenciamento de Memória

- Como jogos são sistema de tempo real, é necessário aproveitar cada milésimo de segundo possível
 - ◆ Isso só é possível paralelizando as tarefas e otimizando o uso de memória e processamento
- Gerenciamento de memória tem um papel importante, já que o custo de alocação e desalocação é alto
- Também implica no aproveitamento da cache



1. Gerenciamento de Memória

- Alinhamento é como os dados de uma estrutura estão organizados na memória
 - ◆ Em geral eles estão na ordem da declaração, o problema é aproveitamento de espaço
 - ◆ Quando temos variáveis pequenas que não podem ser agrupadas com variáveis vizinhas, criamos um buraco
 - ◆ Alguns compiladores têm otimização



1. Gerenciamento de Memória

```
struct InefficientPacking {  
    U32 mU1; // 32 bits  
    F32 mF2; // 32 bits  
    U8  mB3; // 8 bits  
    I32 ml4; // 32 bits  
    bool mB5; // 8 bits  
    char* mP6; // 32 bits  
};
```



0x0	mU1	
0x4	mF2	
0x8	mB3	
0xC	mI4	
0x10	mB5	
0x14	mP6	

Alinhamento do InefficientPacking



1. Gerenciamento de Memória

```
struct MoreEfficientPacking {  
    U32 mU1; // 32 bits (4-byte aligned)  
    F32 mF2; // 32 bits (4-byte aligned)  
    I32 mI4; // 32 bits (4-byte aligned)  
    char* mP6; // 32 bits (4-byte aligned)  
    U8 mB3; // 8 bits (1-byte aligned)  
    bool mB5; // 8 bits (1-byte aligned)  
};
```



0x0	mU1		
0x4	mF2		
0x8	mI4		
0xC	mP6		
0x10	mB3	mB5	

Alinhamento do MoreEfficientPacking

1. Gerenciamento de Memória

- Alocação de memória é custoso, o sistema operacional faz o que é possível para tornar menos custoso e aproveitar melhor a memória
- Nem sempre ele faz um trabalho bom (obrigado Windows), então o programador fica encarregado de implementar sua própria classe de gerenciamento de memória

1. Gerenciamento de Memória

- Object-Centric Architecture (AOS - Array Of Structures)
 - ◆ Instâncias de objetos contém as propriedades e temos uma coleção de objetos
 - ◆ Mais legível e amigável com programador
 - ◆ `GameObject array[] = new GameObject[100];`
- Property-Centric Architecture (SOA - Structure Of Arrays)
 - ◆ Temos uma coleção de propriedades e cada objeto tem um id para as propriedades
 - ◆ Menos legível e amigável com a máquina
 - ◆ `Transform transforms[] = new Transform[100]`

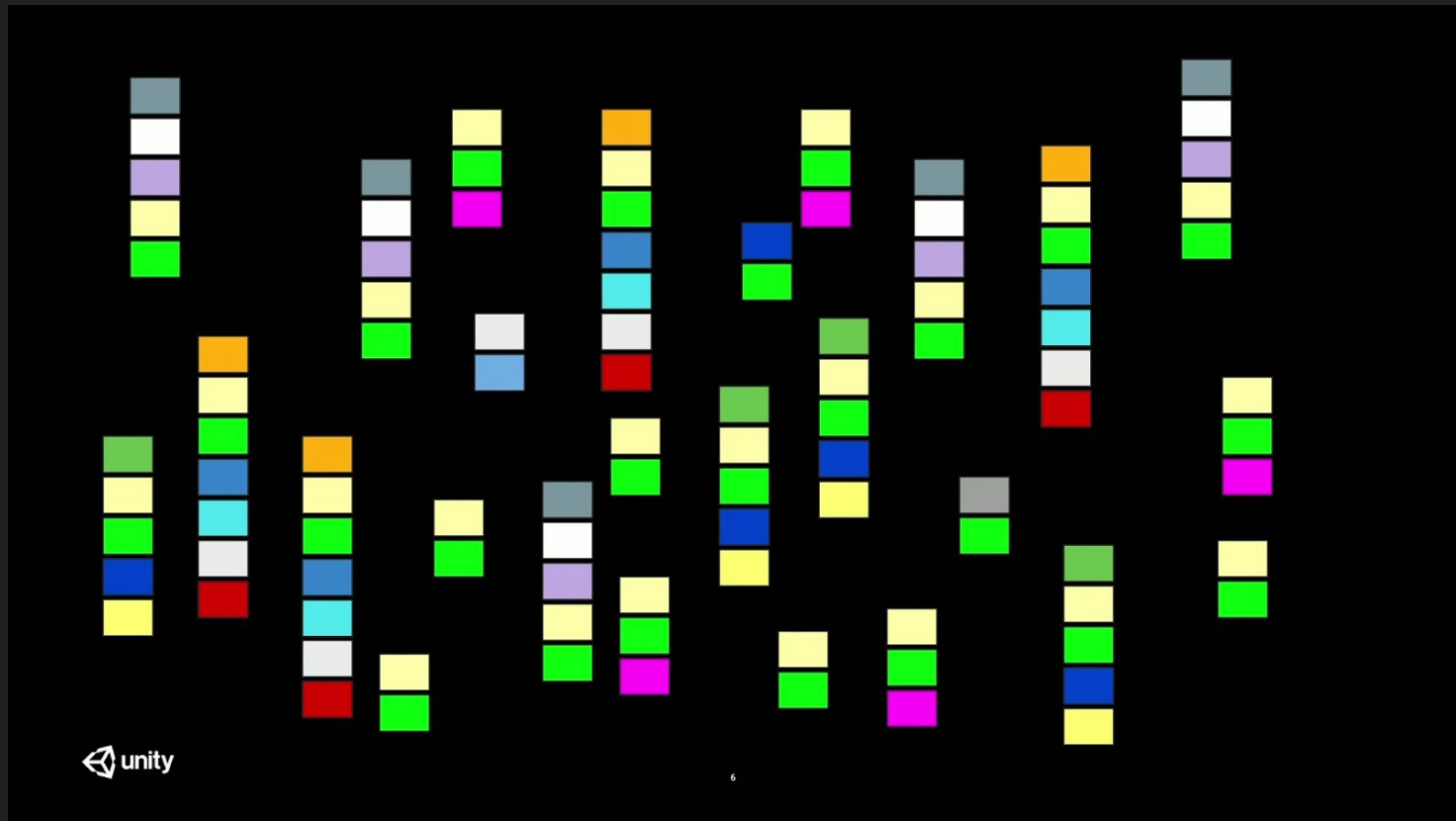


Object-Centric Architecture

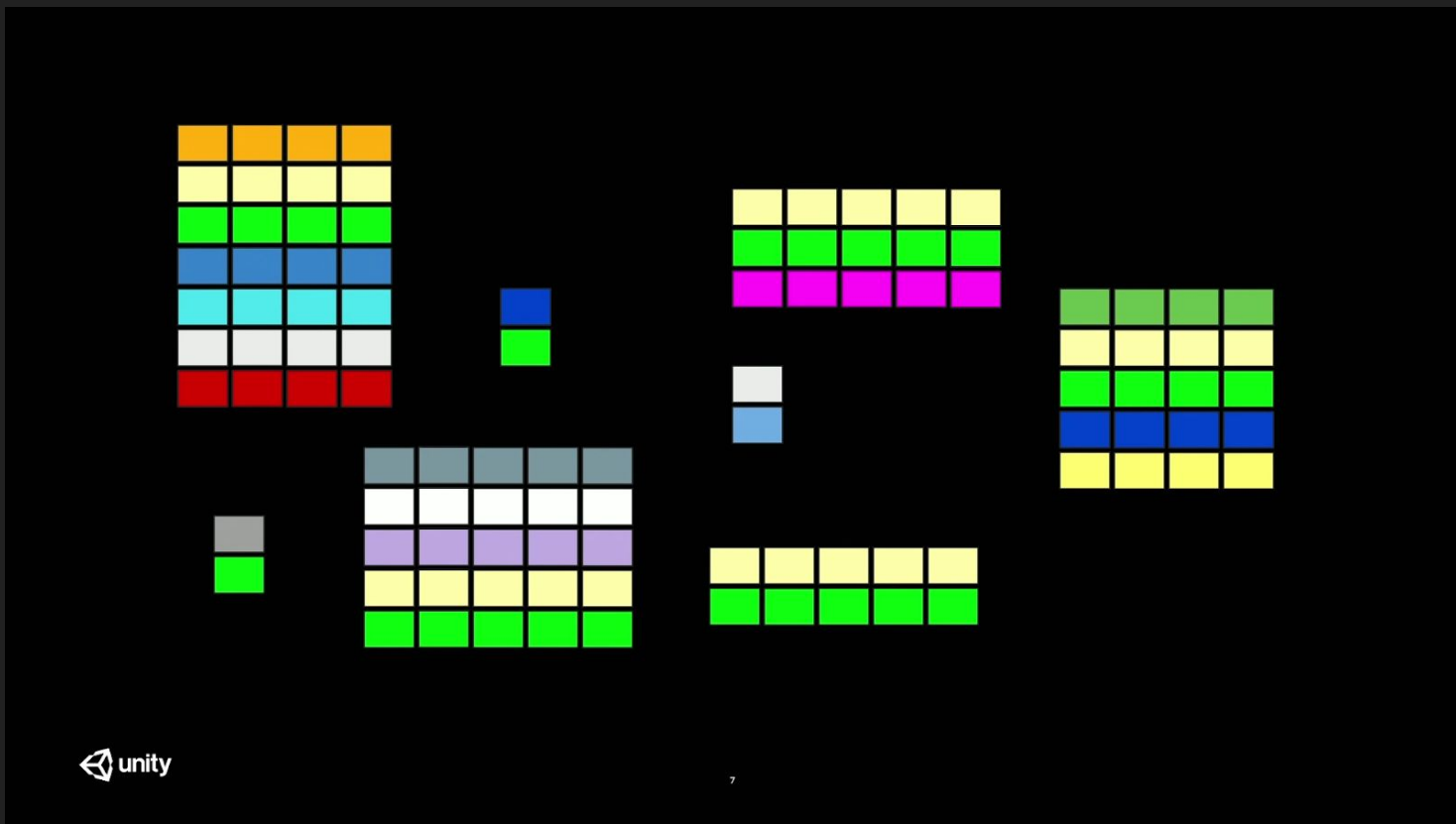
Objeto 1		Objeto 2		
HP	Mana	HP	Mana	Mago
100	10	50	200	Object

Property-Centric Architecture

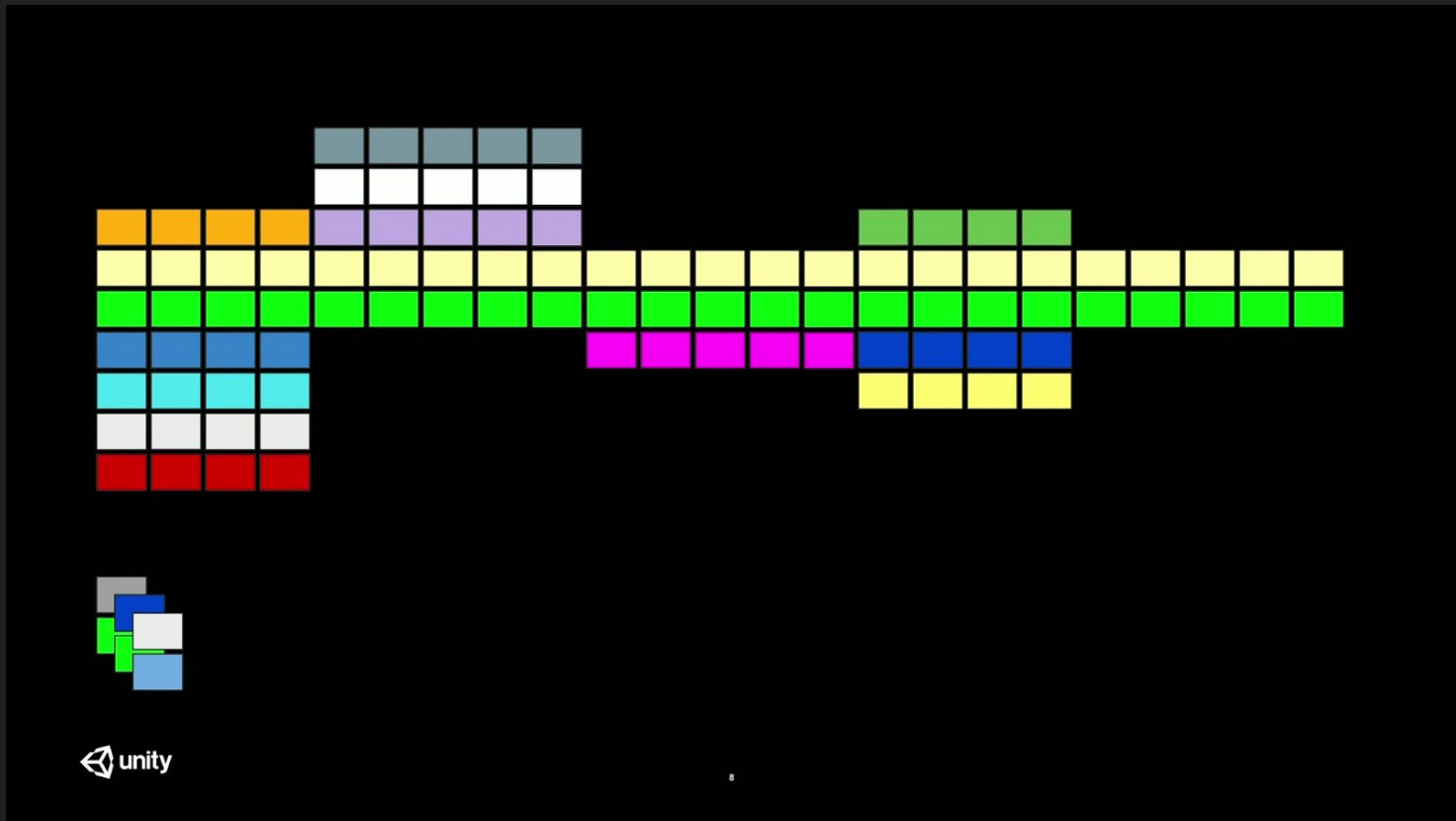
HP		Mana		Mago
Objeto 1	Objeto 2	Objeto 1	Objeto 2	Objeto 2
100	50	10	200	Object



Alocação Padrão de Memória no Unity



Alocação usando Unity DOTS



Processamento de Memória usando Unity DOTS

1. Gerenciamento de Memória

- O disco rígido também sofre seus problemas, por exemplo, fragmentação
 - ◆ Um jogo contém muitos arquivos de arte, música, som, texto etc., isso ocupa muito espaço
 - ◆ Muitos arquivos também podem acabar fragmentando o disco
 - ◆ Além da compressão de cada asset, compactamos todos eles em zip, por exemplo

1. Gerenciamento de Memória

- Assim diminuimos o desperdício de memória, porém aumentamos o custo de processamento, para descompactar esses assets
- Por isso organizamos em diferentes packs, podendo seguir a linha de game design e similaridade
 - ◆ Um level tem um zip próprio com os assets daquele level, por exemplo

2. Gerenciamento de Processo



2. Gerenciamento de Processo

- Além de melhorar o aproveitamento de memória, precisamos melhorar o aproveitamento de processamento
- Temos muitas tarefas para cumprir, num pequeno espaço de tempo, como vamos dividir essas tarefas?
 - ◆ Paralelismo
- Mais fácil falar do que fazer, existem diversos problemas de concorrência

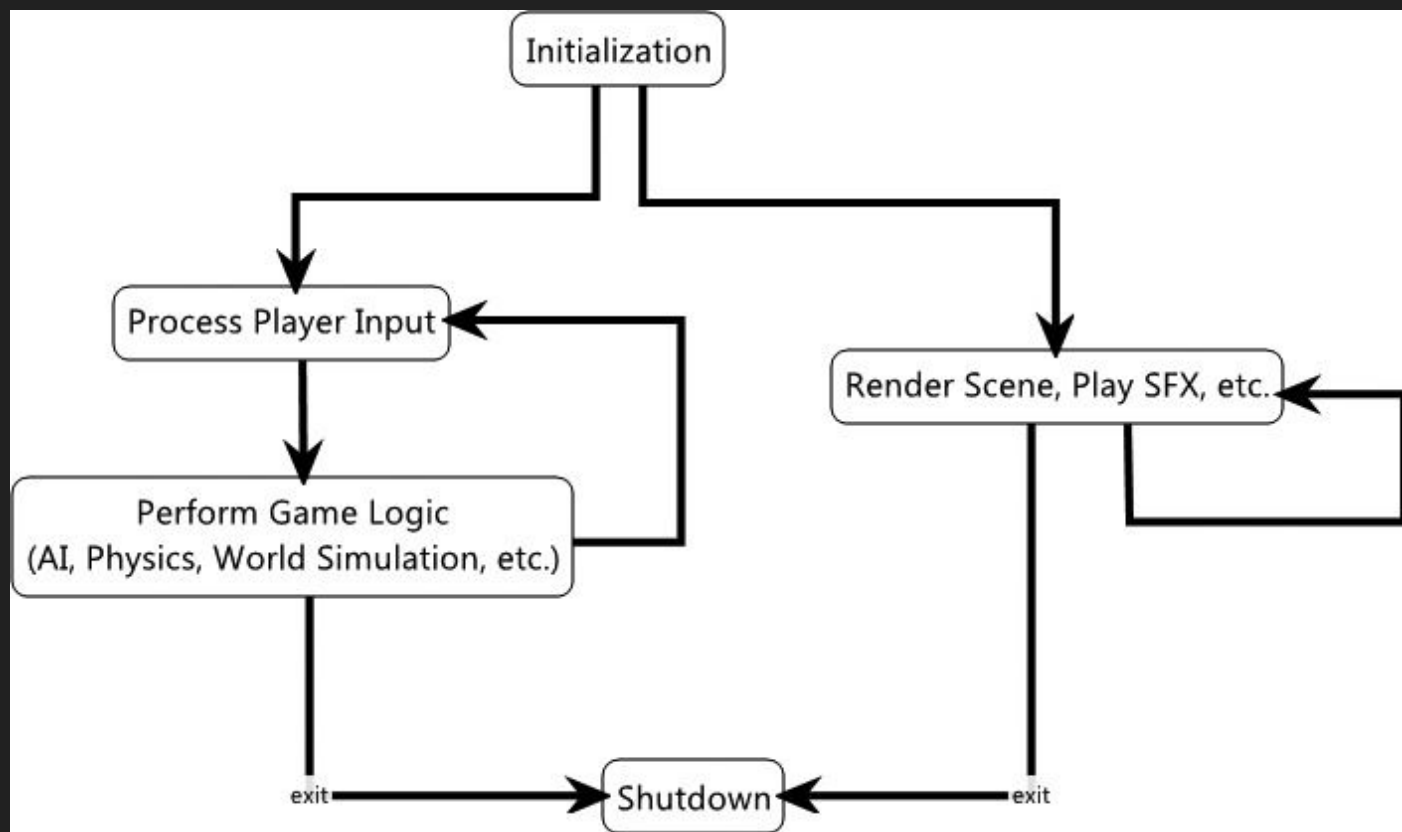
2. Gerenciamento de Processo

→ Opções:

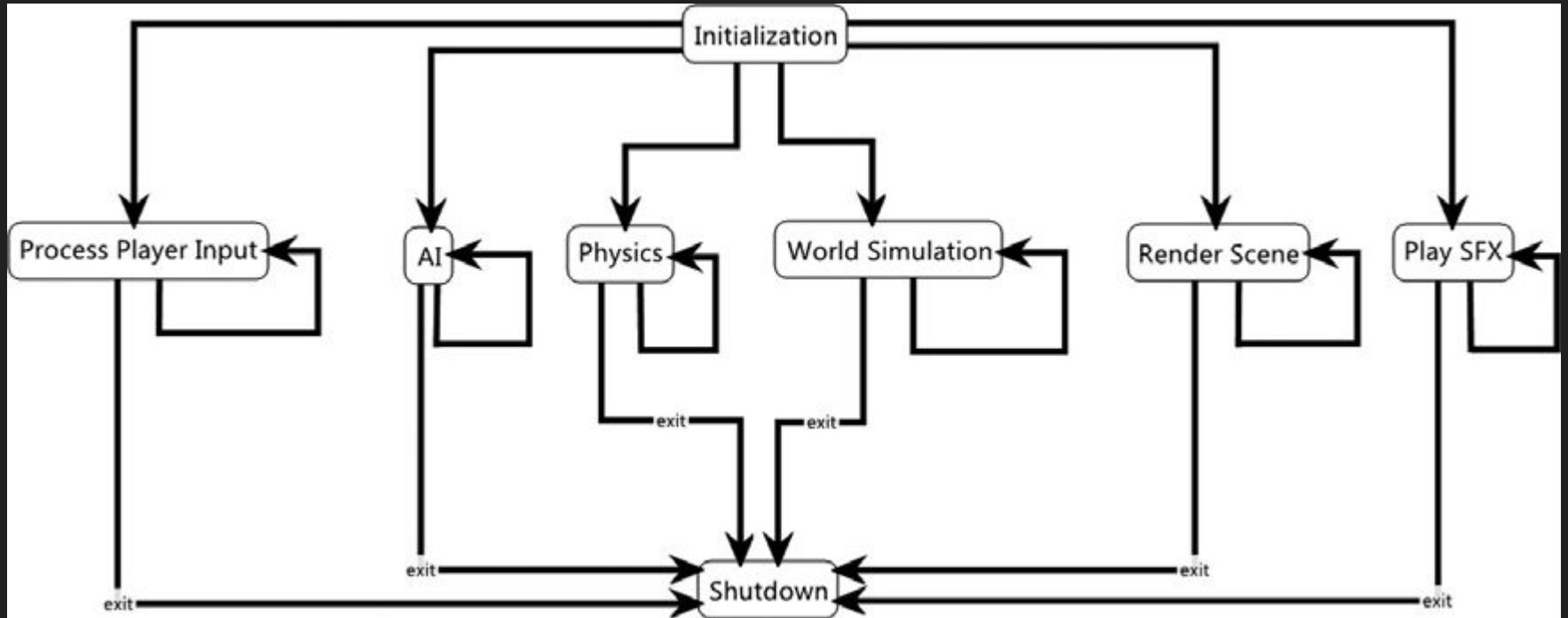
- ◆ Paralelizar cada pedaço da engine (áudio, gráficos, física, IA, entidades etc.)
- ◆ Paralelizar cada tarefa dentro de cada pedaço
- ◆ Paralelizar tudo

→ Problemas:

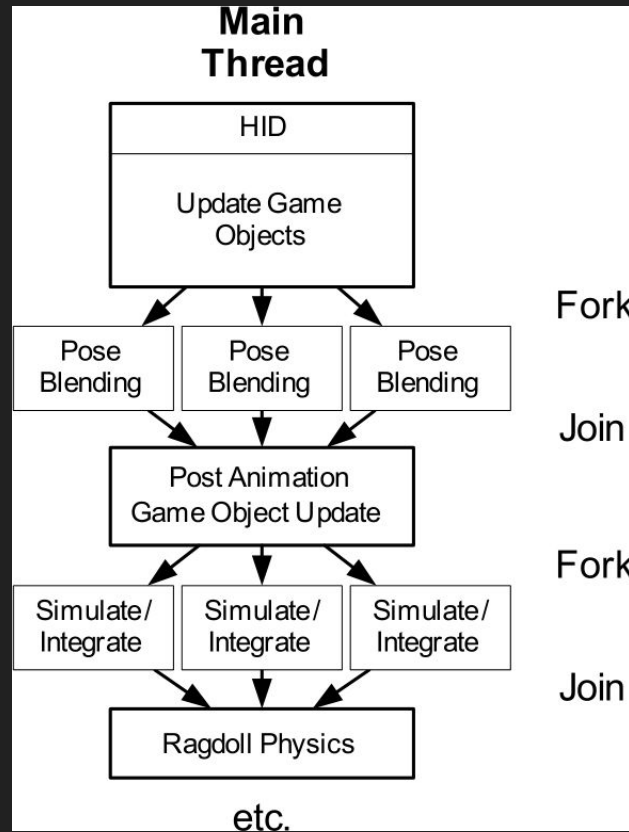
- ◆ Dependência de dados
- ◆ Memória compartilhada



Multithread simples



Multithread cooperativo



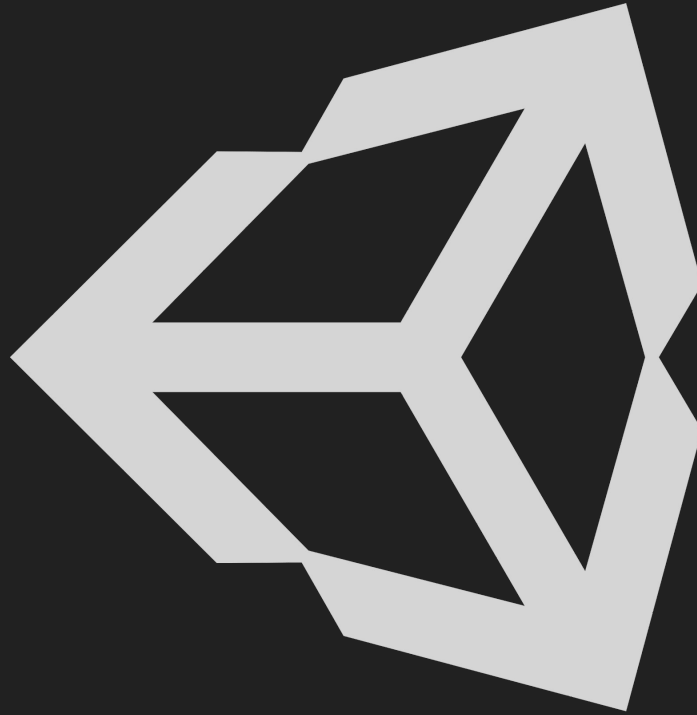
Fork e Join

2. Gerenciamento de Processo

→ Job System

- ◆ Sistema de threads onde é fácil criar e rodar processos simples (tarefas) em threads separadas
- ◆ Ele cuida de criar, manter (Thread Pool) e priorizar as threads

UNITY TIME !!!!



Dúvidas?



Referências

Referências

- [1] Jason Gregory-Game Engine Architecture-A K Peters (2009)
- [2] Game Coding Complete, Fourth Edition (2012) - Mike McShaffry, David Graham
- [3] David H. Eberly 3D Game Engine Architecture Engineering Real-Time Applications with Wild Magic The Morgan Kaufmann Series in Interactive 3D Technology 2004
- [4] <http://gameprogrammingpatterns.com/>
- [5] <http://gafferongames.com/>
- [6] <http://docs.unity3d.com/Manual/index.html>
- [7] <http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/>
- [8] https://en.wikipedia.org/wiki/Software_design_pattern
- [9] <https://www.youtube.com/user/BSVino/videos>
- [10] <https://www.youtube.com/user/thebennybox/videos>
- [11] <https://www.youtube.com/user/GameEngineArchitects/videos>
- [12] <https://www.youtube.com/user/Cercopithecian/videos>
- [13] http://www.glfw.org/docs/latest/input_guide.html
- [14] <http://lazyfoo.net/tutorials/SDL/index.php>
- [15] <https://docs.unity3d.com/Packages/com.unity.entities@0.0/manual/index.html>
- [16] <https://www.youtube.com/watch?v=ILfUuBLfzGI&list=PLzDRvYVwl53s40yP5ROXitbT--IRcHqba>
- [17] <https://www.youtube.com/watch?v=QbnVELXf5RQ>

