Bohemia Interactive

BOHEMIA INTERACTIVE

Datově orientovaný model

Xeniya Vondrášková, Filip Vondrášek

15. 5. 2019, MFF UK

Kdo jsme?

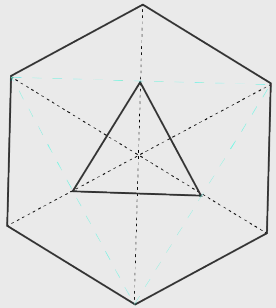
Pracujeme v Bohemia Interactive jako gameplay programátoři.

Náš projekt je Ylands -- sandboxová hra a platforma pro vytváření vašich vlastních her.

V současné době na Steamu v Early Accessu.

PLÁN

1. Rozložení paměti struktur
2. CPU cache
3. Datově orientovaný design
4. Unity DOTS (Entity Component System)



ROZLOŽENÍ PAMĚTI STRUKTUR

Rozložení paměti struktur

```
public struct NPCData
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```

Rozložení paměti struktur

```
unsafe
{
      Debug.Log(sizeof(NPCData));
}

68
```

Rozložení paměti struktur

```
public struct NPCData
{
	public Quaternion BodyOrientation;
	public Quaternion HeadOrientation;
	public Vector3 Position;
	public int Health;
	public int Damage;
	public byte IsPositionCurrent;
	public byte IsOrientationCurrent;
	public byte HasEverTakeDamage;
	public byte HasEverShot;
}
```

Rozložení paměti struktur

```
unsafe
{
      Debug.Log(sizeof(NPCData));
}

56
```

Rozložení paměti struktur

12 bytů méně - proč?

Odpověď je **rozložení paměti**

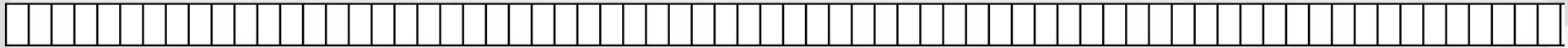
C# (stejně jako C/C++) jde **shora dolů** a skládá prvky struktur do paměti v tomto pořadí

Každý datový typ má **přirozené zarovnání**, které musí být dodrženo, aby CPU mohl číst a zapisovat paměť efektivně.

Pokud není paměť zarovnána, namísto jednoduchého čtení/zápisu musí CPU přečíst více bloků paměti, aplikovat masku, bit shift a OR operaci.

Rozložení paměti struktur

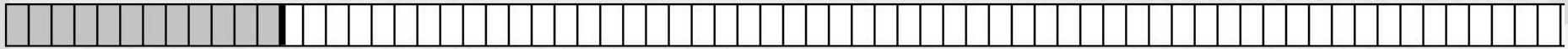
```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```



Rozložení paměti struktur

```
public struct WrongLayout
{
      → public Vector3 Position;
      public byte IsPositionCurrent;
      public Quaternion BodyOrientation;
      public byte IsOrientationCurrent;
      public int Health;
      public byte HasEverTakenDamage;
      public int Damage;
      public byte HasEverShot;
      public Quaternion HeadOrientation;
}
```

Vector 3
(3 floats)

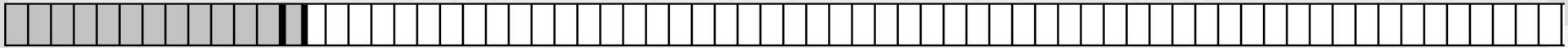


Rozložení paměti struktur

```
public struct WrongLayout
{
      public Vector3 Position;
      → public byte IsPositionCurrent;
      public Quaternion BodyOrientation;
      public byte IsOrientationCurrent;
      public int Health;
      public byte HasEverTakenDamage;
      public int Damage;
      public byte HasEverShot;
      public Quaternion HeadOrientation;
}
```

Vector 3
(3 floats)

byte



Rozložení paměti struktur

```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	→ public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```

Vector 3
(3 floats)

byte

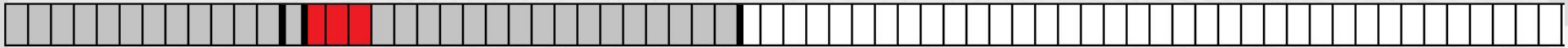
Rozložení paměti struktur

```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	→ public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```

Vector 3 (3 floats)

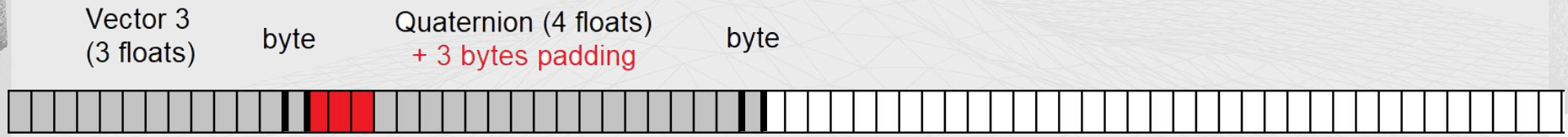
byte

Quaternion (4 floats) + 3 bytes padding



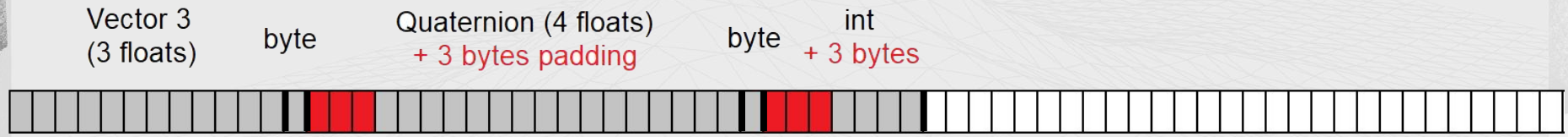
Rozložení paměti struktur

```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	→ public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```



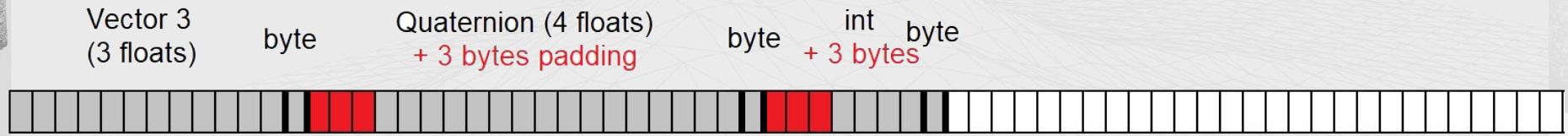
Rozložení paměti struktur

```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	→ public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```



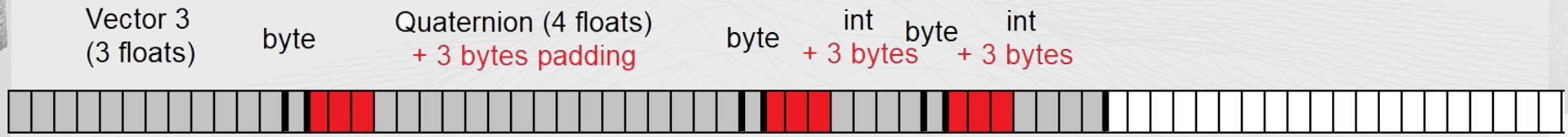
Rozložení paměti struktur

```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	→ public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```



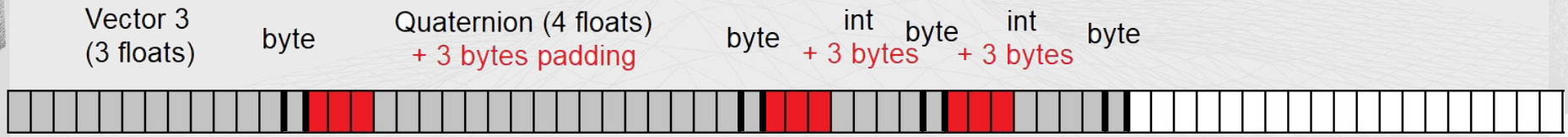
Rozložení paměti struktur

```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	→ public int Damage;
	public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```



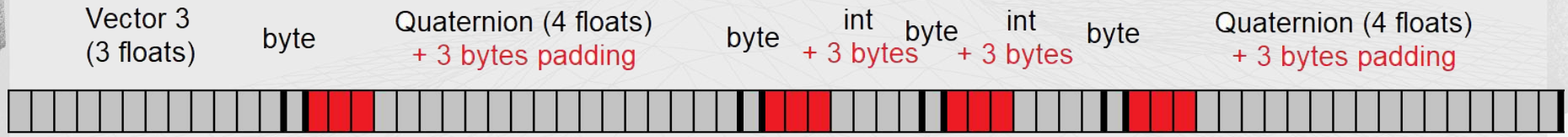
Rozložení paměti struktur

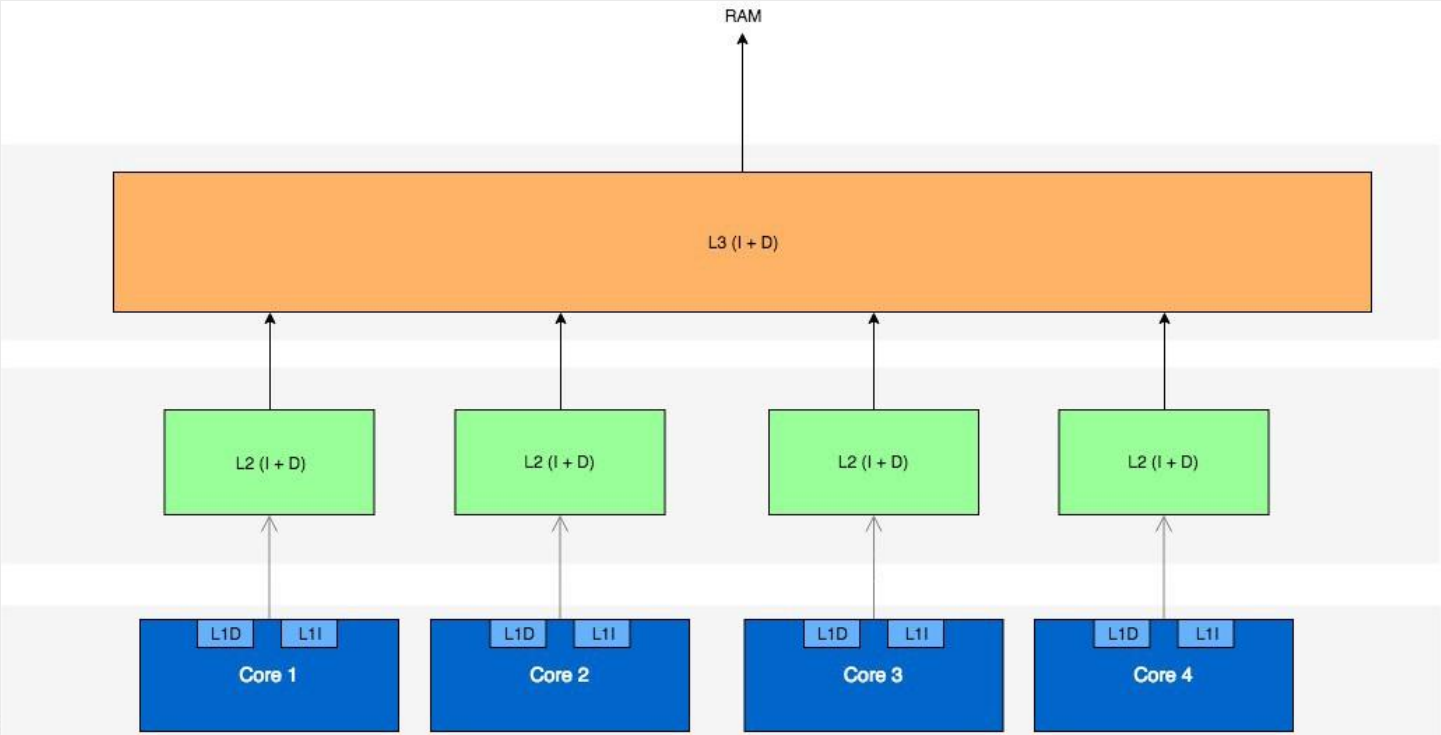
```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	→ public byte HasEverShot;
	public Quaternion HeadOrientation;
}
```



Rozložení paměti struktur

```
public struct WrongLayout
{
	public Vector3 Position;
	public byte IsPositionCurrent;
	public Quaternion BodyOrientation;
	public byte IsOrientationCurrent;
	public int Health;
	public byte HasEverTakenDamage;
	public int Damage;
	public byte HasEverShot;
	→ public Quaternion HeadOrientation;
}
```





https://codeburst.io/understanding-hardware-to-push-performance-to-the-max-and-then-some-part-1-c048021114e3

CPU CACHE

Instrukční cache (L1i, L2, L3...)

Datové cache (L1d, L2, L3...)

L1 a L2 většinou samostatná pro každé jádro procesoru

L3 sdílená mezi jádry

L1, L2, L3 typicky SRAM

L4 ve specializovaných procesorech, DRAM, mimo CPU kostku

NAČÍTÁNÍ DAT DO CPU

Procesor dostane instrukci “přečti data z adresy XYZ”

Data už jsou v cache -> **cache hit**

Data nejsou v cache -> **cache miss**

Cache miss: Musíme **naalokovat místo** v cache, **přečíst data z RAM**, pak teprve přečíst do registrů.

Výpočetně velmi náročné (zvlášť u write-back systémů)

Cache line - data jsou načítaná po blocích fixní velikosti

ČASOVÁ NÁROČNOST TYPICKÝCH OPERACÍ

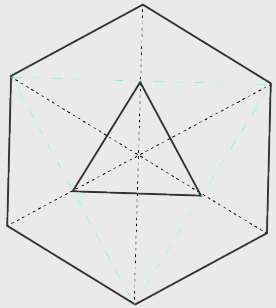
Operace	Čas
Čtení z L1 cache	0,5 ns
Provedení instrukce	1 ns
Branch misprediction	5 ns
Čtení z L2 cache	7 ns
Čtení z RAM	100 ns (200x pomalejší než L1 cache!)
Přečtení 1 MB RAM sekvenčně	250 000 ns
Přečtení 1 MB SSD sekvenčně	1 000 000 ns
Přečtení 1 MB HDD sekvenčně	20 000 000 ns

PROBLÉMY RYCHLOSTI PAMĚTÍ

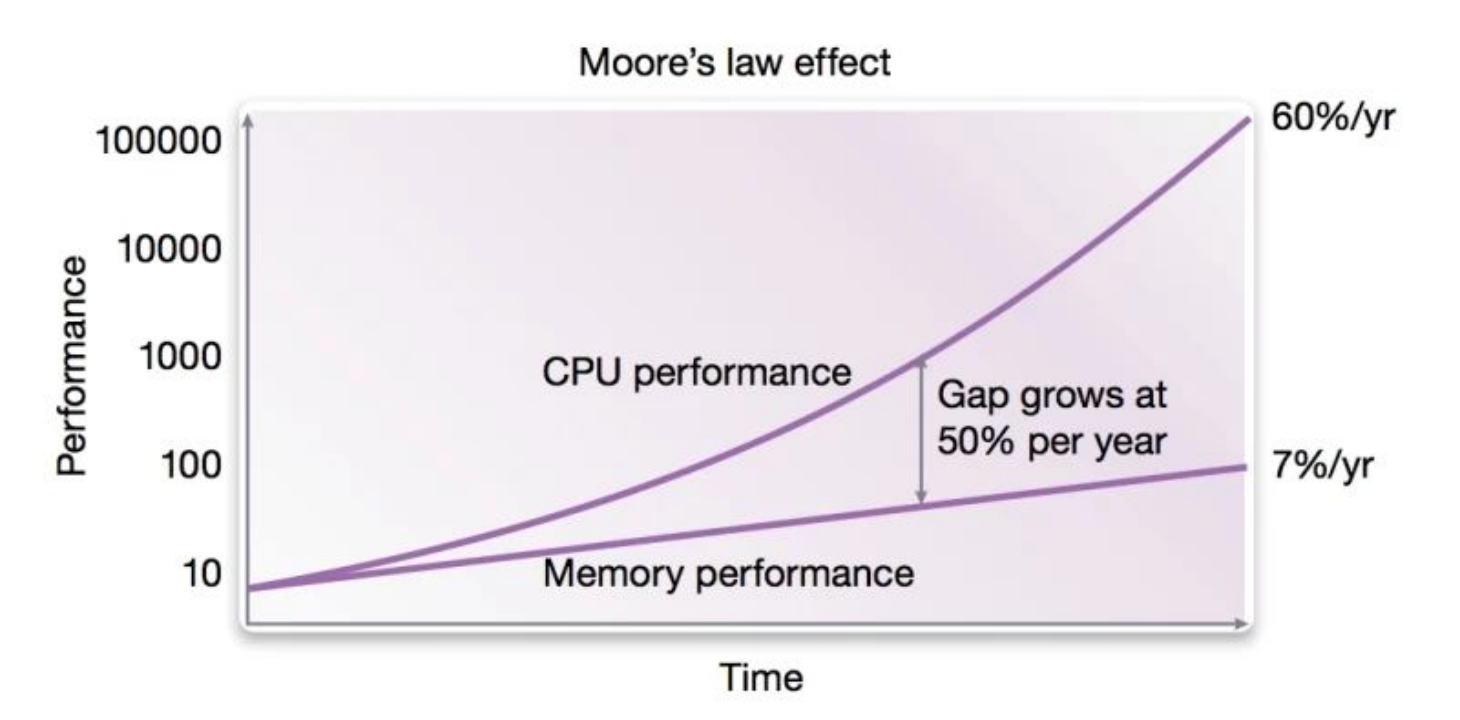
CPU stall - Při čekání na cache line nemá CPU nic co na práci. Za čas, co CPU čeká na data, by jinak zvládl zpracovat **stovky instrukcí - bottleneck**

Mitigace: **out-of-order execution** - provádění následujících operací, které nejsou závislé na očekávaných datech.

Dále **simultánní multithreading** (např. Intel Hyper-Threading) - zatímco aktivní vlákno v CPU čeká na zdroje, po tu dobu může pracovat jiné vlákno.



DATOVĚ ORIENTOVANÝ DESIGN



https://motherboard.vice.com/en_us/article/3dkkx8/memory-is-holding-up-the-moores-law-progression-of-processing-power

DRUHY PROGRAMOVACÍCH PŘÍSTUPŮ

Objektově orientované programování - C++, C#, Java...

Funkcionální programování - Prolog, Lisp...

Procedurální programování - C

Datově orientované programování - ?

PŘÍKLAD DATOVÉ CACHE

```
class Character {
    float HP;      // 4
    float Stamina; // 4
    Vector3 Position; // 3 floats -> 12
    Quaternion Rotation; // 4 floats -> 16
    bool IsAlive; // 1
};
class NPC: Character
{
    Color HairColor; // 4 floats -> 16
};
```

PŘÍKLAD DATOVÉ CACHE

```
class Character {
    float HP;      // 4
    float Stamina; // 4
    Vector3 Position; // 3 floats -> 12
    Quaternion Rotation; // 4 floats -> 16
    bool IsAlive; // 1
};
class NPC: Character
{
    Color HairColor; // 4 floats -> 16
};
```

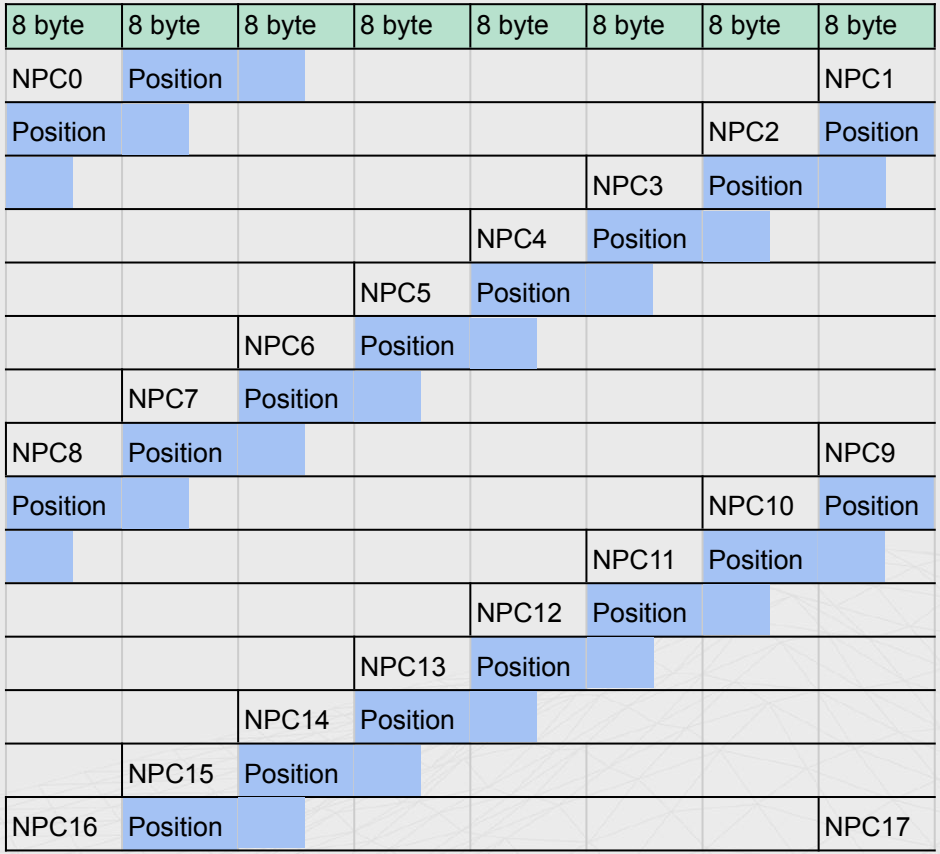
```
class NPC    size(56):
      +---
0     | +--- (base class Character)
0     | | HP
4     | | Stamina
8     | | Vector3 Position
20    | | Quaternion Rotation
36    | | IsAlive
      | | <alignment member> (size=3)
      | +---
40    | Color HairColor
      +---
```

(generated on https://godbolt.org using x64 msvc v19.15 with the */d1reportSingleClassLayoutNPC* compiler option)

PŘÍKLAD DATOVÉ CACHE

```
for (int i=0; i< NPCCount; i++) {
  NPCs[i].Move(deltaTime);
}

void NPC::Move(float deltaTime) {
  Position.x += AI.xDir * deltaTime;
  Position.y += AI.yDir * deltaTime;
  Position.z += AI.MoveUp ? deltaTime : 0f;
}
```

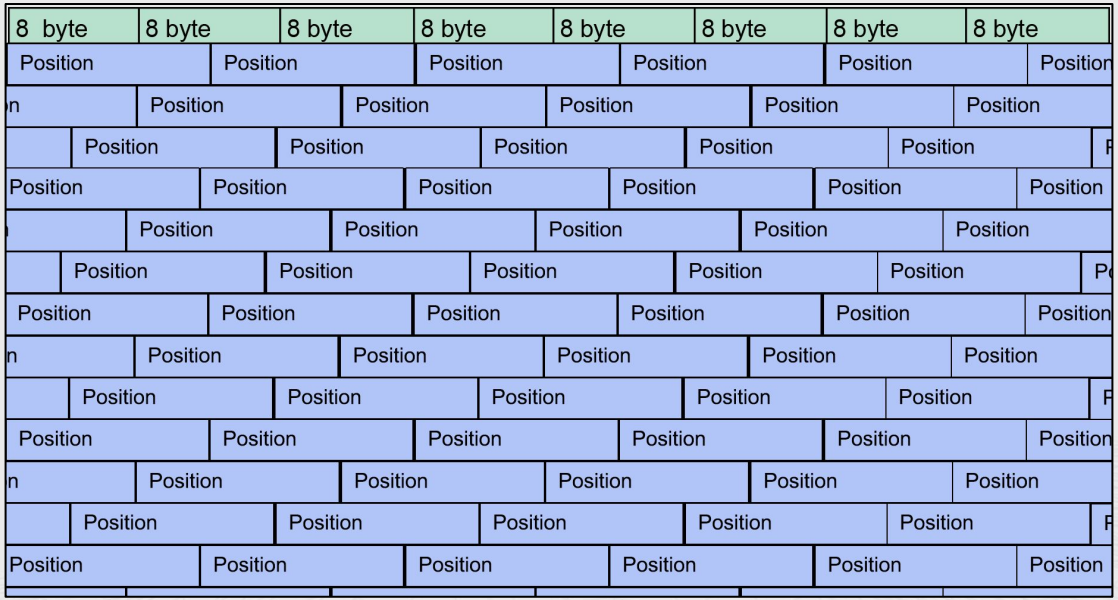


<- 64 byte cache line

PŘÍKLAD DATOVÉ CACHE

```
for (int i=0; i< NPCCount; i++) {
  NPCs[i].Move(deltaTime);
}

void NPC::Move(float deltaTime) {
  Position.x += AI.xDir * deltaTime;
  Position.y += AI.yDir * deltaTime;
  Position.z += AI.MoveUp ? deltaTime : 0f;
}
```


8 byte
8 byte
8 byte
8 byte
8 byte
8 byte
8 byte
8 byte
Position

OOP

Zapouzdření (Encapsulation):
Schovávání informací
V podstatě jen read/write

Dědičnost (Inheritance):
Třídy dědí ze svých předků
Rozšiřování, případně přepisování vlastností předků

Polymorfismus (Polymorphism):
Schopnost zpracovávat objekty různými způsoby v závislosti na jejich typu nebo třídě.

Třídy (Classes):
Data + chování v jedné “obálce”

DOD

Nejčastěji využíván při vývoji her

Důraz na **uspořádání dat v paměti**

Není o struktuře kódu, ale o struktuře dat: seskupení dat **podle využití**, ne podle logického modelu

Minimum objektů, většinou jen **systémy**, které manipulují s daty

Redukce cache missů - mitigace pomalé paměti oproti CPU

Příklad : OOP vs DOD

```
struct Ball {
  Point  position;
  Color  color;
  double radius;

  void draw();
};

vector<Ball> balls;
```

Array of Structs (AoS):

| position | color | radius | position | color | radius | position | color | ...

Příklad : OOP vs DOD

```
struct Balls {
  vector<Point>  position;
  vector<Color>  color;
  vector<double> radius;

  void draw();
};
```

Struct of Arrays (SoA):

| position | position | position | color | color | color | radius | radius | ...

VÝHODY DOD

Efektivní využití cache CPU

Snadná paralelizace

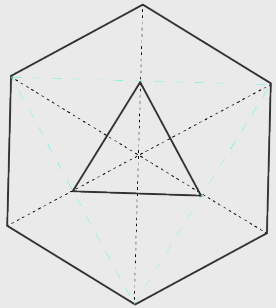
Modularita

NEVÝHODY DOD

Zcela jiný programovací přístup než OOP

Nízké využití vlastností jazyka

Netriviální propojení s již existujícím OOP/procedurálním kódem



DOD V PRAXI

Unity DOTS (Data Oriented Technology Stack)

V Unity poprvé představen ve verzi 2018.1 jako **experimentální balíček**.

Obsahuje tři hlavní aspekty, které by měly výrazným způsobem zlepšit výkon výsledného produktu:

- **Entity-Component-System (ECS)**: stará se o rozložení paměti.
- **C# Job System**: multi-threading.
- **Burst compiler**: vysoce optimalizovaný strojový kód.

Unity: Tradiční přístup

ECS je nový architektonický návrh. Zakládá se na myšlence, že vývojáři začnou používat DOD namísto klasického OOP.

Tradiční přístup:

- GameObjects + Components + MonoBehaviour

Entity Component System

Rozložení objektů na různé **komponenty** (pozice, rotace, počet životů...)

Komponenty stejného typu (nezávisle na entitě, která je vlastní) chceme vyrovnat za sebe **lineárně do paměti** -> ideální na zpracování velkého množství objektů

O zpracování dat se starají **systémy**, které pracují nad množinou komponent

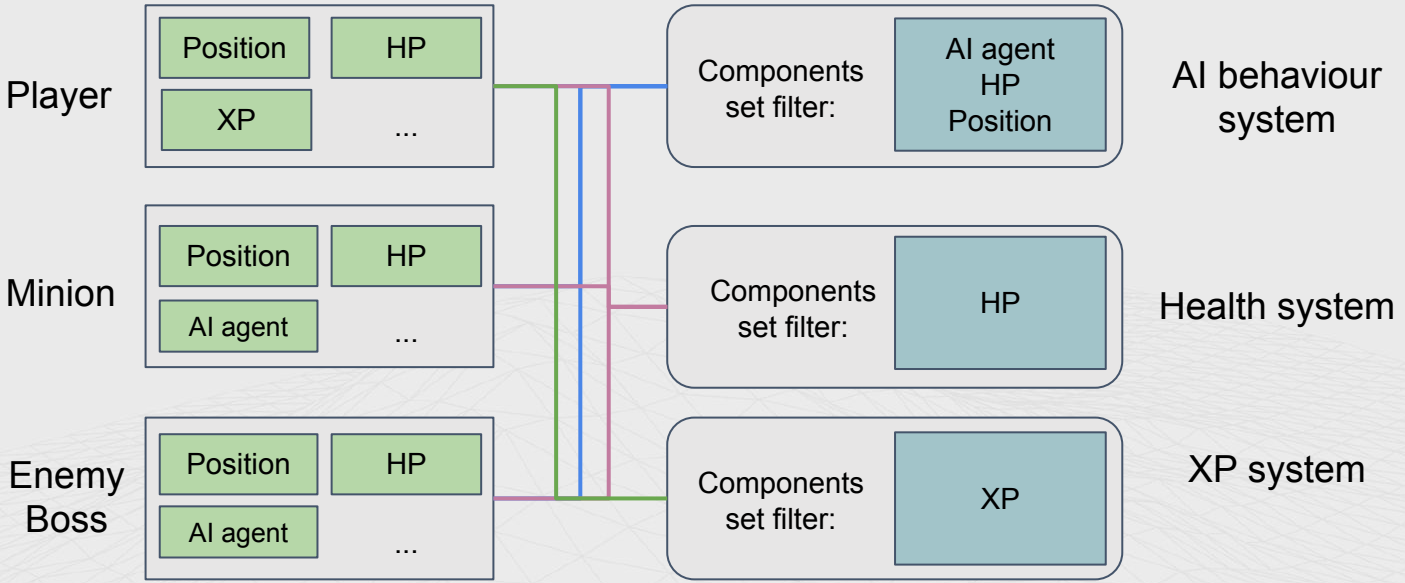
Unity: Entity Component System

V ECS rozdělujeme celou strukturu hry na:

- **Entity**: “ID”
- **Komponenty**: Struktury obsahující pouze instance dat pro entitu. Neobsahuje metody.
- **Systémy**: Obsahují funkcionalitu/logiku. Zodpovědné za update všech entit s odpovídající množinou komponent.

ECS garantuje lineární datový layout - rychlejší management dat.
Také separuje data od logiky - snadnější aplikování instrukcí na velké množství entit paralelně.

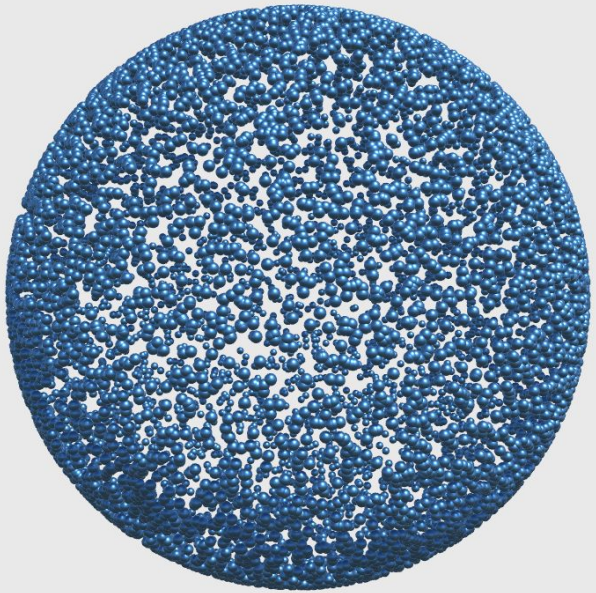
Unity: Entity Component System



https://software.intel.com/en-us/articles/get-started-with-the-unity-entity-component-system-ecs-c-sharp-job-system-and-burst-compiler

FPS: 28

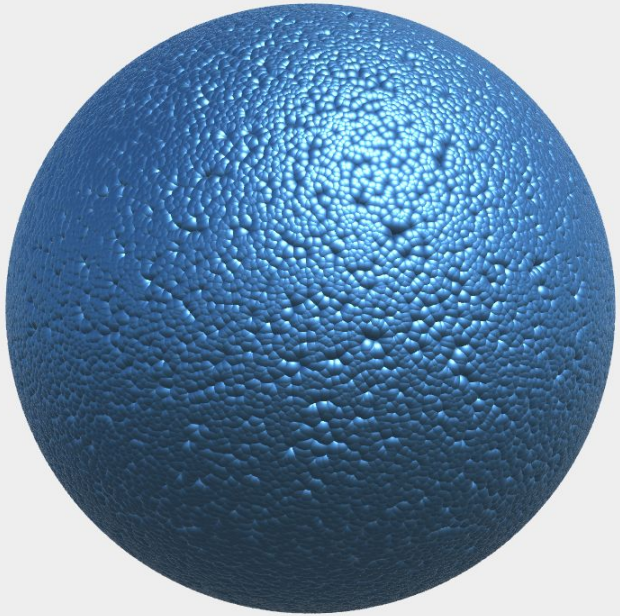
Object count: 9000



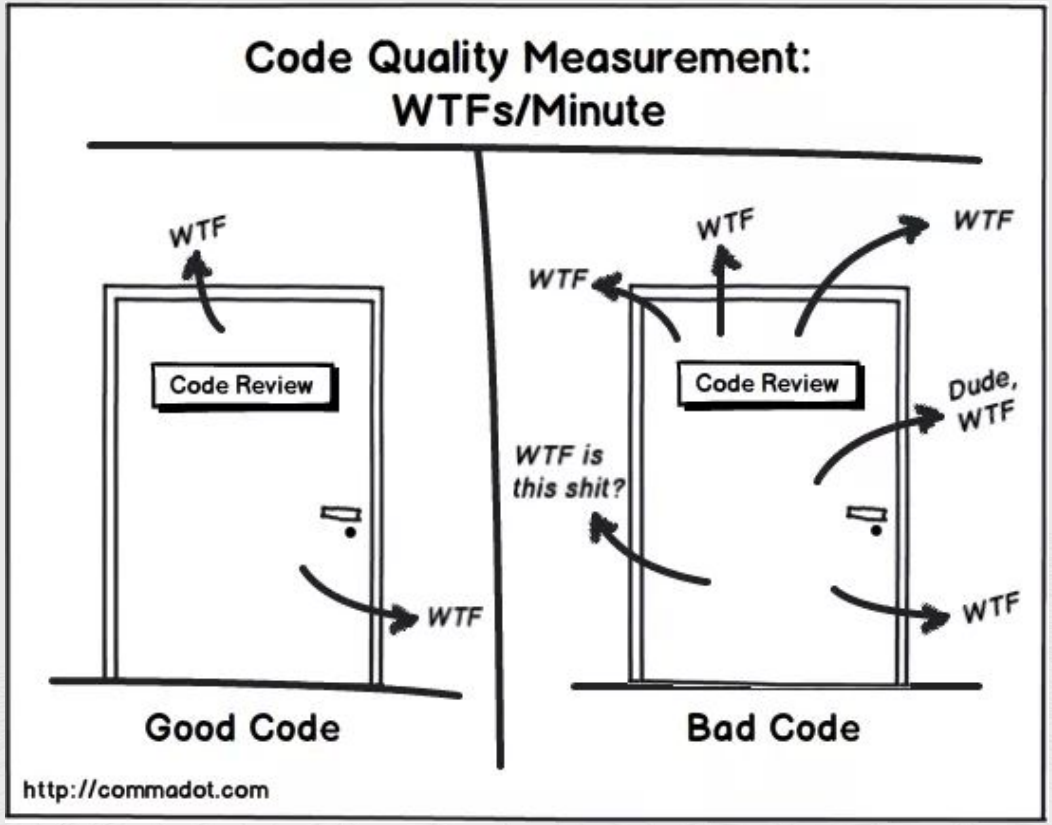
Tradiční přístup

FPS: 31

Object count: 43000



ECS (+Burst)


Code Quality Measurement:
WTFs/Minute
WTF
Code Review
WTF
Good Code
WTF
WTF
WTF
WTF
Code Review
Dude,
WTF
WTF is
this shit?
WTF
Bad Code
http://commadot.com

DĚKUJEME ZA POZORNOST

xeniya.valentova@bistudio.com
filip.vondrasek@bistudio.com

Follow us on:


@bohemiainteract


facebook.com/BohemiaInteractive/


linkedin.com/company/bohemia-interactive/