

Entity Component System

VS

Classic OOP in C++

Кирилл Гейзеров – BlackHub Games



BLACKHUB
games

Структура доклада

- **The "World" – GameEngine и C++**
 - на примере простой симуляции
- **Варианты реализации, код, рассуждения, замеры производительности**
 - ООП, Entity-Centric, Наследование
 - ООП, Entity-Component, Агрегирование
 - DoD, Entity Component System
- **Неутешительные выводы**

The "World"

World – это множество **Entities**

- **Bat** Постоянно в движении
- **Obstacle** Статично

Механика

- При столкновении **Bat** с **Obstacle**
 - Меняем направление
 - Меняем цвет



World size: 1'000'000 Bats и 20 Obstacles

The "World"

Главный цикл:

```
// Main loop
while (appIsRunning()) {
    world.progress();

    render();

    /* ... Poll events, Swap buffers, ... */
}
```

- For each frame (60 FPS, e.g.)
 - Update the **World**
 - Update the **Entities**
 - Update the **Properties**



Run on: AMD Ryzen 5 5600G with Radeon Graphics 3.90 GHz

The "World"

Базовые свойства объектов:

```
struct Position {  
    float x, y;  
};  
  
struct Velocity {  
    float x, y;  
};  
  
struct Color {  
    float r, g, b;  
};  
  
struct Sprite {  
    enum Model : uint8_t { Bat, Tree };  
  
    Model model;  
    Color color;  
};
```



-02 opt level, OpenGL, map data directly CPU -> GPU

Engine

- Перебираем объекты каждый кадр
- Объектов много
- Динамический мир
- Динамические свойства
- Контекстно-зависимые свойства
- Контекстно-зависимое поведение
- **Multiple dispatch** — Взаимодействие всех со всеми

Неизбежно

- Усложнение дизайна
- Падение производительности

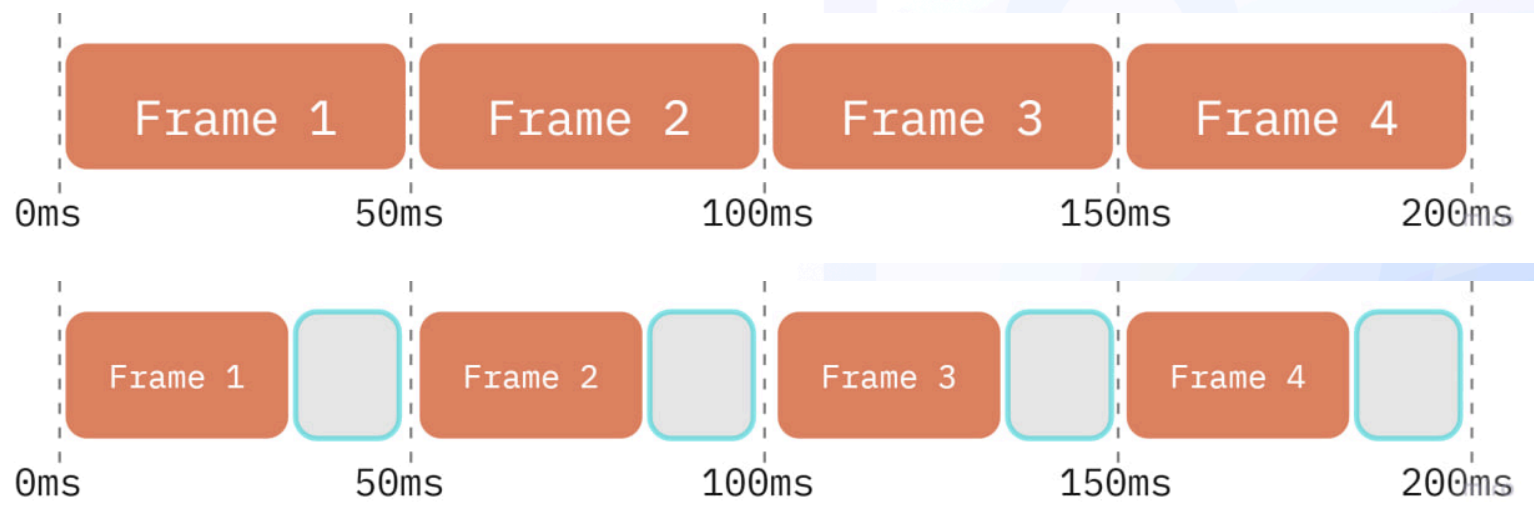
Что первично ?

Дизайн

- Понятный
- Гибкий

Быстродействие

- Быстрые кадры
 - Качество картинки: детализация, эффекты, большой FPS
 - Энергоэффективность: "спать" как можно чаще

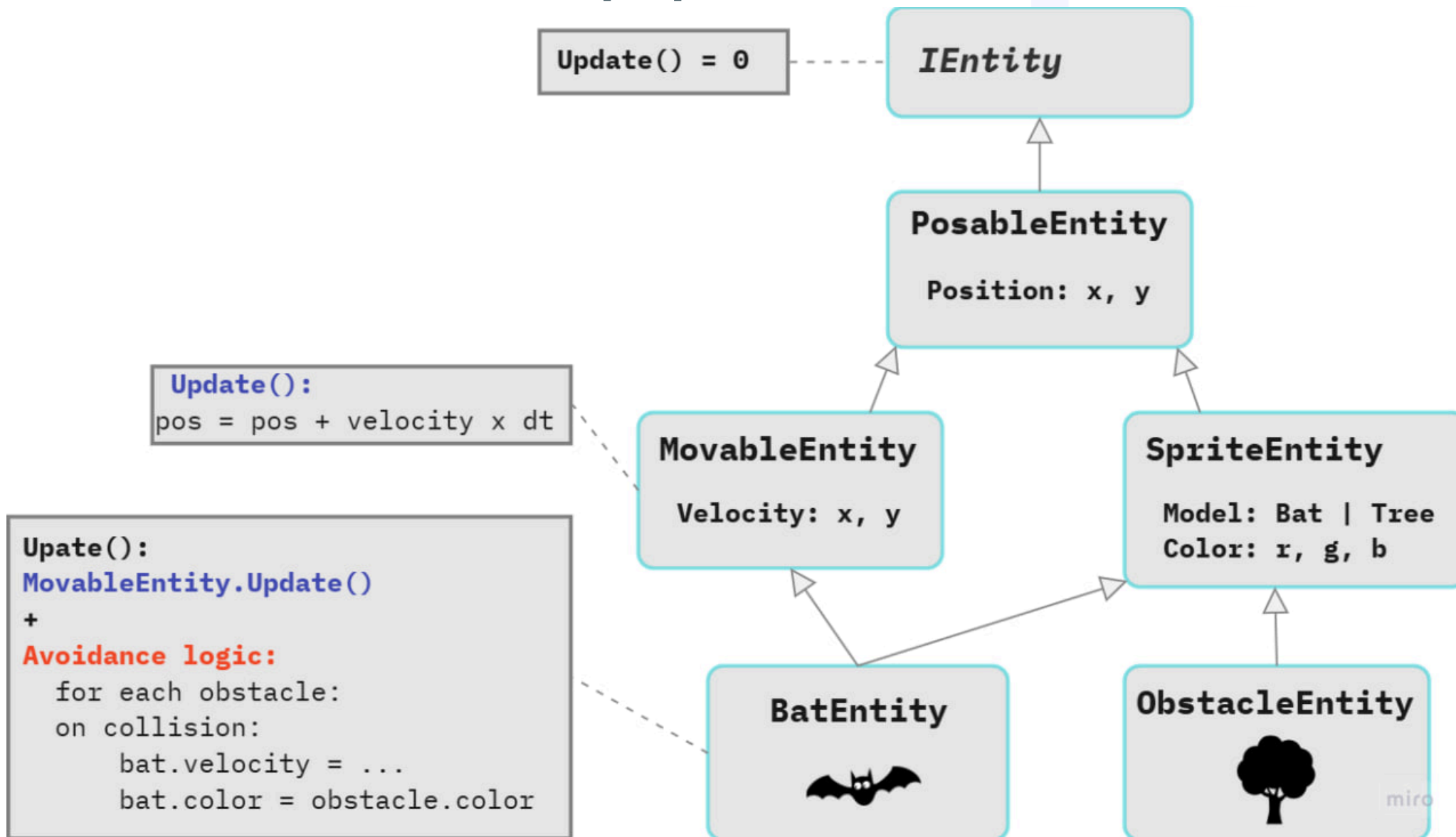


Варианты реализации

- **Object-oriented. Entity-Centric**
 - Свойства и логика сгруппированы в Entity
 - Наследование
- **Object-oriented. Entity-Component**
 - Свойства и логика сгруппированы в Entity
 - Агрегирование
- **Data-oriented. Entity Component System**
 - Свойства и логика независимы



ООП, Entity-centric: Иерархия



ООП, Entity-Centric: Иерархия

- Всё есть Entity
- Наследуем Entity

```
class IEntity {  
public:  
    virtual ~IEntity() = default;  
  
    virtual void update(World&) {}  
};
```

```
struct PosableEntity : IEntity {  
    Position pos;  
};  
  
struct MovableEntity : virtual PosableEntity {  
    Velocity vel;  
  
    void update(World& world) override;  
};  
  
struct SpriteEntity : virtual PosableEntity {  
    Sprite sprite;  
};  
  
struct ObstacleEntity : SpriteEntity { };  
  
struct BatEntity : SpriteEntity, MovableEntity {  
    void update(World& world) override;  
};
```

ООП, Entity-centric: Логика

```
void MovableEntity::update(World& world) override {
    PosableEntity::update(world);

    const auto dt = world.getTimeDelta();
    pos.x += vel.x * dt;
    pos.y += vel.y * dt;

    if (vel.x < 0.0f && pos.x < -1.f || vel.x > 0.0f && pos.x > 1.f) vel.x = -vel.x;
    if (vel.y < 0.0f && pos.y < -1.f || vel.y > 0.0f && pos.y > 1.f) vel.y = -vel.y;
}
```

```
void BatEntity::update(World& world) {
    SpriteEntity::update(world);
    MovableEntity::update(world);

    for (const auto& entt : world.getEntities()) {
        if (auto obstacle = dynamic_cast<const ObstacleEntity*>(entt)) {
            // process collision here ...
        }
    }
}
```

ООП, Entity-centric: World

ObstacleEntity



Position: x, y

Sprite.Model: Tree

Sprite.Color: r, g, b

BatEntity

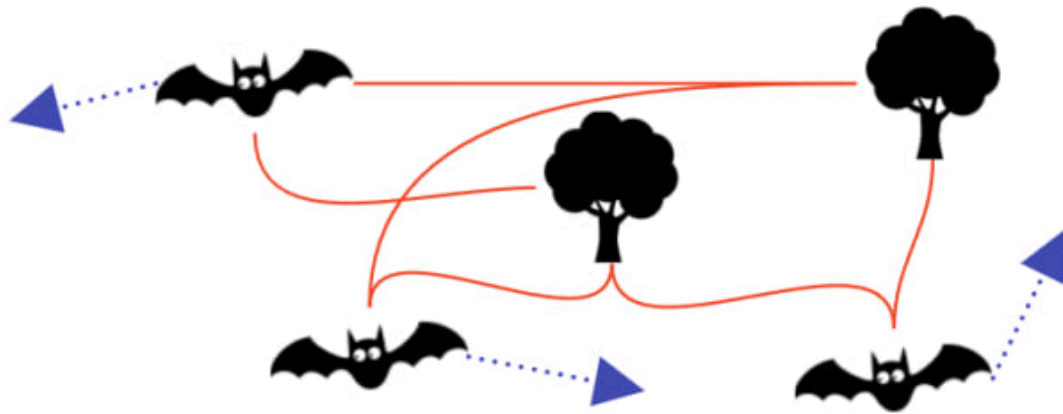


Position: x, y

Velocity: x, y

Sprite.Model: Bat

Sprite.Color: r, g, b



Update():

MovableEntity.Update()

+

Avoidance logic:

for each obstacle:

on collision:

bat.velocity = ...

bat.color = obstacle.color

ООП, Entity-centric: World

```
class World {
public:
    using Entities = std::vector<std::unique_ptr<IEntity>>;

    template <class I>
    T& addEntity() {
        T* entt = new T();
        _entities.emplace_back(entt);
        return *entt;
    }

    Entities& getEntities() { return _entities; }

    float getTimeDelta() const { return _time_delta.count(); }

    size_t count() const { return _entities.size(); }

    template <class EntT>
    size_t count() const {
        size_t cnt = 0;
        for (const auto& pe : _entities) {
            if (nullptr != dynamic_cast<const EntT*>(&pe))
                ++cnt;
        }
        return cnt;
    }
}
```

```
void progress() {
    // clock update here ...

    // Update all entities
    for (auto&& entt : _entities)
        entt->update(*this);
}

private:
    Entities _entities;
    TimeDelta _time_delta{};
};
```

ООП, Entity-centric: Тесты

```
World world;

for (int i = 0; i < 20; ++i) {
    auto& entt = world.addEntity<ObstacleEntity>();

    entt.sprite = { Sprite::Tree, randColor() };
    entt.pos = { frand(), frand() };
}

for (int i = 0; i < 1'000'000; ++i) {
    auto& entt = world.addEntity<BatEntity>();

    entt.sprite = { Sprite::Bat, randColor() };
    entt.pos = { frand(), frand() };
    entt.vel = { frand() * 0.5f, frand() * 0.5f };
}
```

Результаты

Без кэширования Obstacles

- Frame delta: **400+ ms** ❌
on **3'000 x 10**

Кэшируем Obstacles

- Frame delta: **175 ms** ✅
on **1'000'000 x 20**

ООП, Entity-centric: А что если...

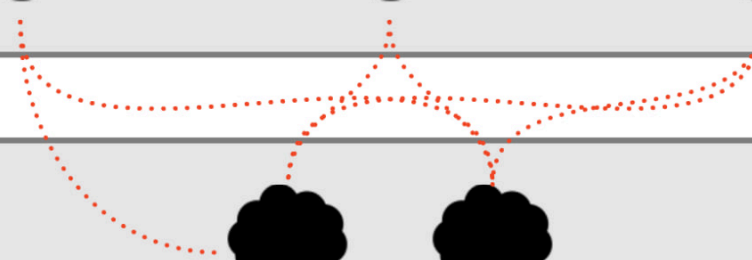
```
void* ObstacleEntity::operator new(std::size_t count) {  
    return Allocator<ObstacleEntity>::get().alloc_new();  
}  
  
void* BatEntity::operator new(std::size_t count) {  
    return Allocator<BatEntity>::get().alloc_new();  
}
```

... распределить Entity по типам
для улучшения локальности?

Bat memory pool



Obstacle memory pool



ООП, Entity-centric: ...

```
void* ObstacleEntity::operator new(std::size_t count) {  
    return Allocator<ObstacleEntity>::get().alloc_new();  
}  
  
void* BatEntity::operator new(std::size_t count) {  
    return Allocator<BatEntity>::get().alloc_new();  
}
```

Результаты

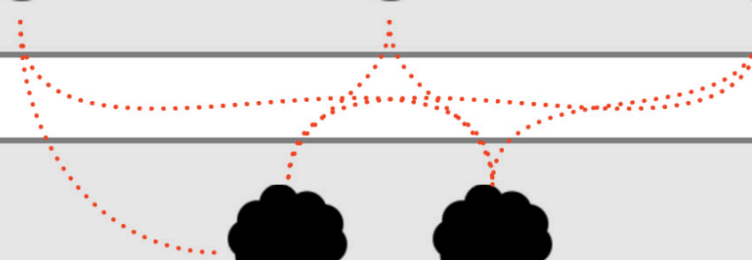
- Frame delta: **175 ms -> 120 ms**

Локальность важна!

Bat memory pool



Obstacle memory pool



Что не так с ООП в С++ ?

Дизайн

- Где данные?
- Где логика?
- Как расширяются данные?
- Как расширяется логика?
- Как связаны объекты?

Быстродействие

- Data locality
- Temporal locality
- Compile-time and run-time optimizations

Что не так с ООП в С++ ? Дизайн: данные

- Каждый объект — это набор разнородных данных (включая vptr)

```
struct PosableEntity : IEntity {
    Position pos;
};

struct MovableEntity : virtual PosableEntity {
    Velocity vel;
    // ...
};

struct BatEntity : SpriteEntity, MovableEntity {
    // ...
};
```

- Кто от кого наследуется? Если крякает, как утка — это утка? 🦆

Что не так с ООП в С++ ? Дизайн: логика

- Не очевидно, где реализовывать логику

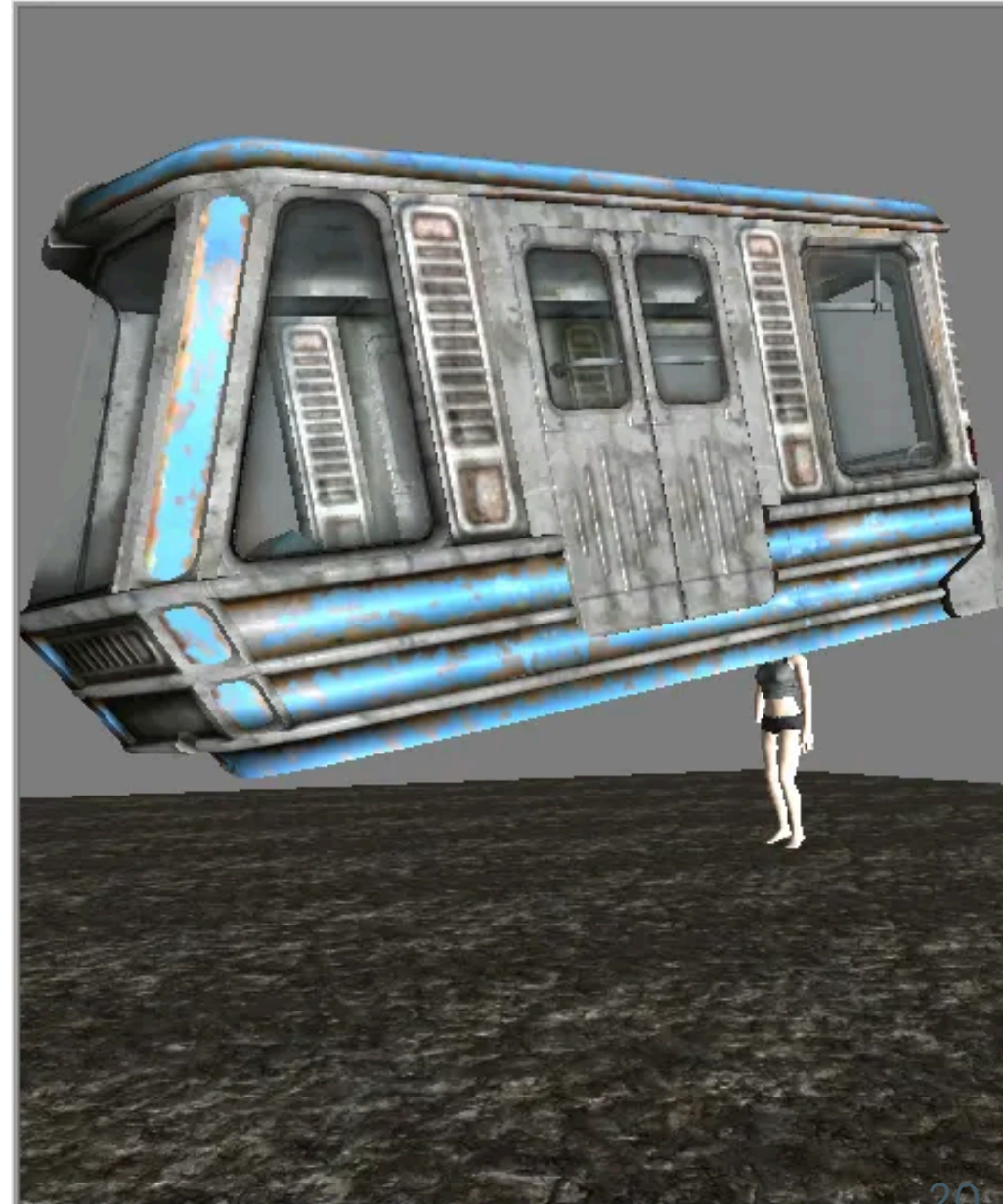
```
// Why not ObstacleEntity::update() ?
void BatEntity::update(World& world) {
    SpriteEntity::update(world);
    MovableEntity::update(world);

    for (const auto& entt : world.getEntities()) {
        if (auto obstacle = dynamic_cast<const ObstacleEntity*>(&entt)) {
            // process collision here ...
        }
    }
}
```

Что не так с ООП в С++ ?

Дизайн: "классы-кентавры"

- Множество контекстов для Entity
 - Всемогущие "Классы-кентавры"
 - много наследования
 - много сеттеров/геттеров
 - много ненужной условной логики



Что не так с ООП в C++ ?

Дизайн: "классы-кентавры"

```
class Player :
public virtual ToolUserEntity,
public virtual LoungingEntity,
public virtual ChattyEntity,
public virtual DamageBarEntity,
public virtual PortraitEntity,
public virtual NametagEntity,
public virtual PhysicsEntity,
public virtual EmoteEntity {
public:
Vec2F position() const override;
Vec2F velocity() const override;

Vec2F mouthPosition() const override;
Vec2F mouthOffset() const;
Vec2F feetOffset() const;
```

```
Vec2F headArmorOffset() const;
Vec2F chestArmorOffset() const;
Vec2F legsArmorOffset() const;
Vec2F backArmorOffset() const;

void moveLeft();
void moveRight();
void moveUp();
void moveDown();
void jump();

void hitOther(EntityId target, DamageRequest const& damageRequest) override;
void interactWithEntity(InteractiveEntityPtr entity);
void setCameraFocusEntity(Maybe<EntityId> const& cameraFocusEntity) override;

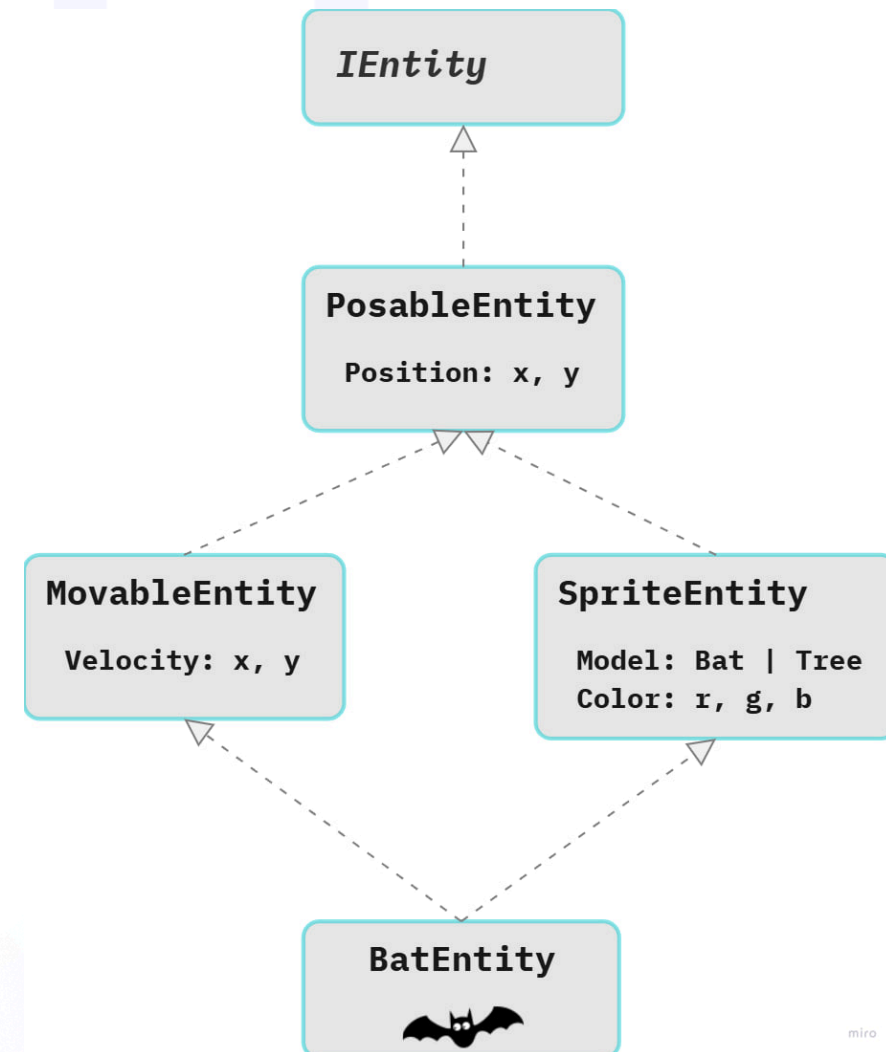
// ... more and more ...
};
```

Source: https://kyren.github.io/rustconf_2018_slides/index.html

Что не так с ООП в С++ ? Быстродействие

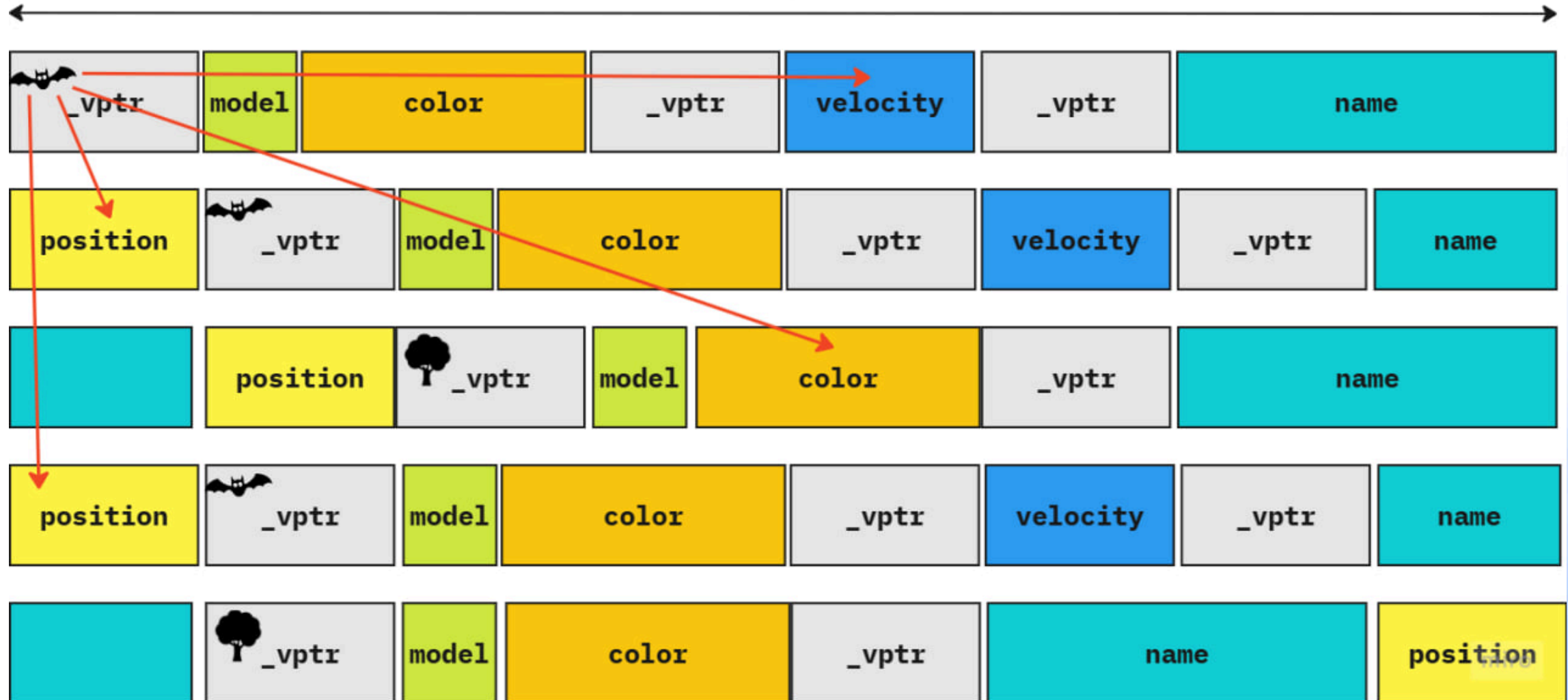
- Разнородные данные для множества Entity
 - Большие объекты в кэше
 - Нарушение локальности

```
struct BatEntity {  
    _vptr_SpriteEntity;           // 8 bytes  
    Model          Sprite.model;  // 4 bytes (with padding)  
    Color          Sprite.color;  // 12 bytes  
  
    _vptr_MovableEntity;         // 8 bytes  
    Velocity       Movable.vel;  // 8 bytes  
  
    _vptr_PosableEntity;        // 8 bytes  
    String         IEntity.name; // 16 bytes  
    Position       pos;         // 8 bytes  
};                               // 72 bytes
```



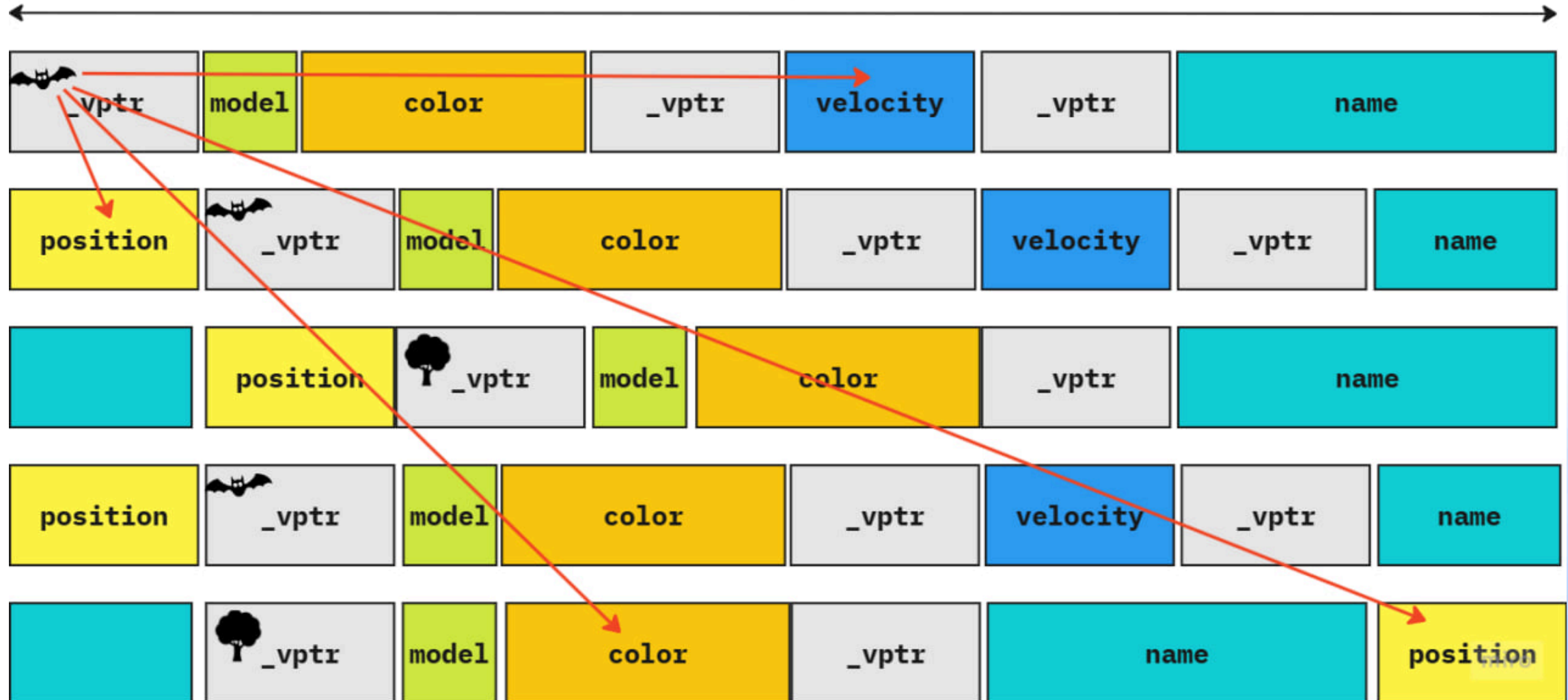
Что не так с ООП в С++ ? Быстродействие

64 bytes (typical CPU cache line)



Что не так с ООП в С++ ? Быстродействие

64 bytes (typical CPU cache line)



Что не так с ООП в C++ ?

Быстродействие: Оптимизации

- Объектно-центричная логика ломает явные и неявные оптимизации

```
// На самом деле здесь неявный M x N цикл
void BatEntity::update(World& world) {
    // Which order? Time locality ?
    SpriteEntity::update(world);
    MovableEntity::update(world);

    // Обход ВСЕХ entity. Как явно кэшировать только ObstacleEntity?
    // Как улучшить предсказания?
    // loop unrolling, SIMD ?
    for (const auto& entt : world.getEntities()) {
        if (auto obstacle = dynamic_cast<const ObstacleEntity*>(&entt)) {
            // process collision here ...
        }
    }
}
```

Что не так с ООП в C++ ?

Быстродействие: RTTI и циклы

- Полиморфизм провоцирует бездумно использовать RTTI

```
void BatEntity::update(World& world) {
    SpriteEntity::update(world);
    MovableEntity::update(world);

    for (const auto& entt : world.getEntities()) {
        // Если бы у нас не было RTTI – пришлось бы подумать над дизайном лучше
        if (auto obstacle = dynamic_cast<const ObstacleEntity*>(&entt)) {
            // process collision here ...
        }
    }
}
```

- Pools по типам Entity сильно быстрее



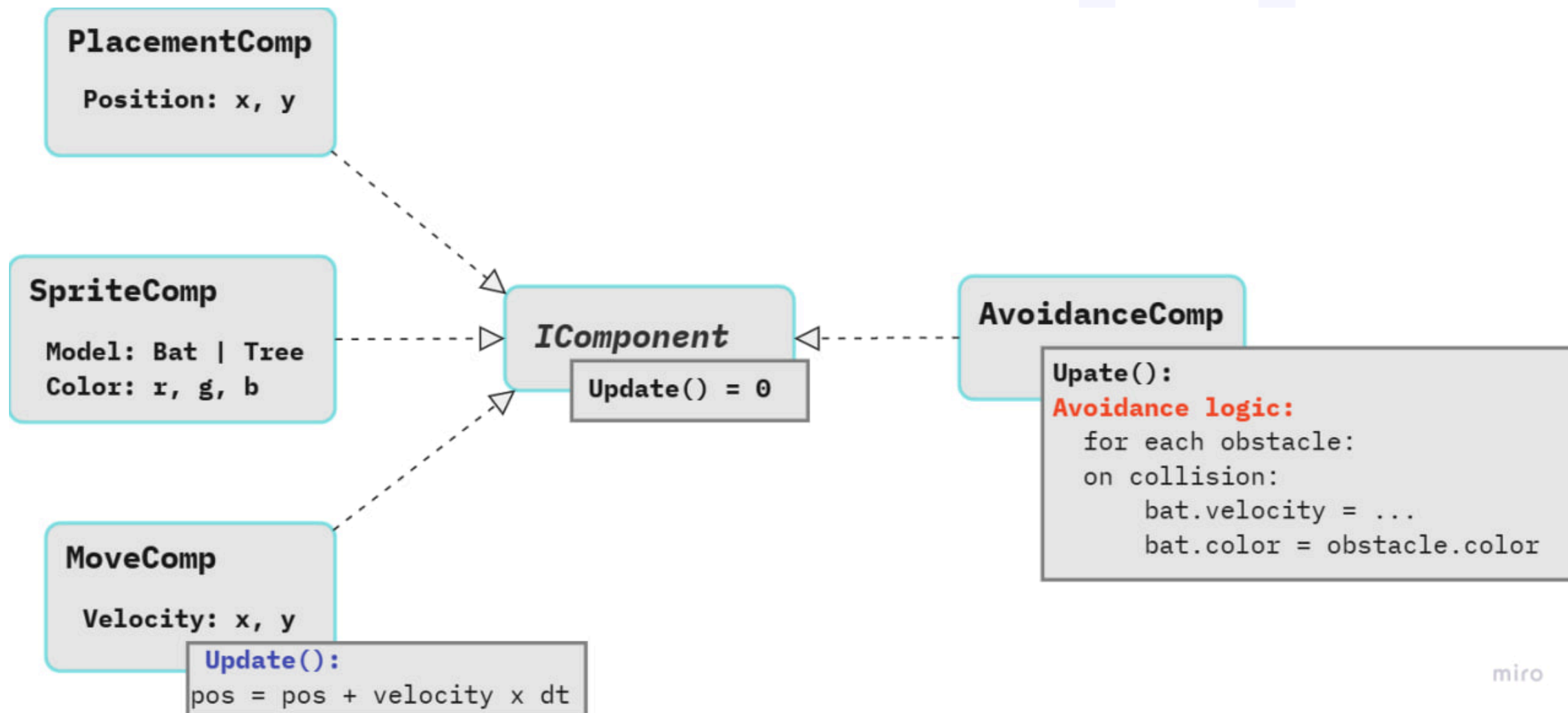
Escape the Inheritance

Варианты реализации

- **Object-oriented. Entity-Centric**
 - Свойства и логика сгруппированы в Entity
 - Наследование
- **Object-oriented. Entity-Component**
 - Свойства и логика сгруппированы в Entity
 - Агрегирование
- **Data-oriented. Entity Component System**
 - Свойства и логика независимы



ООП, Entity-Component: Иерархия



ООП, Entity-Component: Иерархия

```
class IComponent {  
public:  
    virtual ~IComponent() = default;  
  
    virtual void update(World&, Entity&) {}  
};
```

```
struct PlacementComponent : IComponent {  
    Position pos;  
};  
  
struct MoveComponent : IComponent {  
    Velocity vel;  
  
    void update(World&, Entity&) override;  
};  
  
struct SpriteComponent : IComponent {  
    Sprite sprite;  
};  
  
struct AvoidanceComponent : IComponent {  
    void update(World&, Entity&) override;  
};
```

ООП, Entity-Component: World

```
class Entity {  
public:  
    using Components =  
        std::vector<std::unique_ptr<IComponent>>;  
  
    void update(World&);  
  
    template <class CompT>  
    CompT* GetComponent() {  
        for (auto&& comp : _components) {  
            if (auto tcomp = dynamic_cast<CompT*>(&comp))  
                return tcomp;  
        }  
        return nullptr;  
    }  
  
    template <class CompT>  
    CompT& addComponent() {  
        auto* comp = new CompT();  
        _components.emplace_back(comp);  
        return *comp;  
    }  
  
private:  
    Components _components;  
};
```

Entity

IComponent x N

Update():

For each comp:
 comp.Update()

ObstacleEntity



PlacementComp

SpriteComp (model = Tree)

BatEntity



PlacementComp

MoveComp

SpriteComp (model = Bat)

AvoidanceComp

miro

```
void Entity::update(World& world) {  
    for (auto&& comp : _components) {  
        comp->update(world, *this);  
    }  
}
```

ООП, Entity-Component: Логика

Всё еще Entity-centric

```
// Who should include this component?
void AvoidanceComponent::update(World& world, Entity& entt) {
    auto bat_place = entt.getComponent<PlacementComponent>();
    auto bat_move = entt.getComponent<MoveComponent>();
    auto bat_sprite = entt.getComponent<SpriteComponent>();

    // Still iterate over ALL entities
    for (auto& entt : world.getEntities()) {
        // Yes, it's duck-typing
        if (entt.getComponent<ObstacleComponent>()) {
            auto obst_place = obst.getComponent<PlacementComponent>();
            auto obst_sprite = obst.getComponent<SpriteComponent>();

            // process collision here ...
        }
    }
}
```

ООП, Entity-Component: Тесты

```
for (int i = 0; i < 20; ++i) {
    auto& e = world.addEntity();

    auto& sprite = e.addComponent<SpriteComponent>();
    sprite.sprite = { Sprite::Tree, randColor() };

    auto& place = e.addComponent<PlacementComponent>();
    place.pos = Position{ frand(), frand() };
}

for (int i = 0; i < 1'000'000; ++i) {
    auto& e = world.addEntity();

    auto& sprite = e.addComponent<SpriteComponent>();
    sprite.sprite = { Sprite::Bat, randColor() };

    auto& place = e.addComponent<PlacementComponent>();
    place.pos = Position{ frand(), frand() };

    auto& move = e.addComponent<MoveComponent>();
    move.vel = { frand() * 0.5f, frand() * 0.5f };

    e.addComponent<AvoidanceComponent>();
}
```

Результаты

- Frame delta: **175 ms -> 269 ms ?**

ООП, Entity-Component: ~~Default~~ RTTI

```
struct PlacementComponent : IComponent {  
    static constexpr int id = ID_COMP_PLACEMENT; // <-  
    /* ... */  
};
```

```
std::unique_ptr<IComponent> _components[POOL_SIZE];  
  
template <class CompT>  
CompT* GetComponent() {  
    // for (auto&& comp : _components) {  
    //     if (auto tcomp = dynamic_cast<CompT*>(&comp))  
    //         return tcomp;  
    // }  
    return static_cast<CompT*>(_components[CompT::id].get());  
}
```

Результаты

- 175 ms -> 269 ms -> 90 ms ✓

А что насчёт промахов по кэшам?

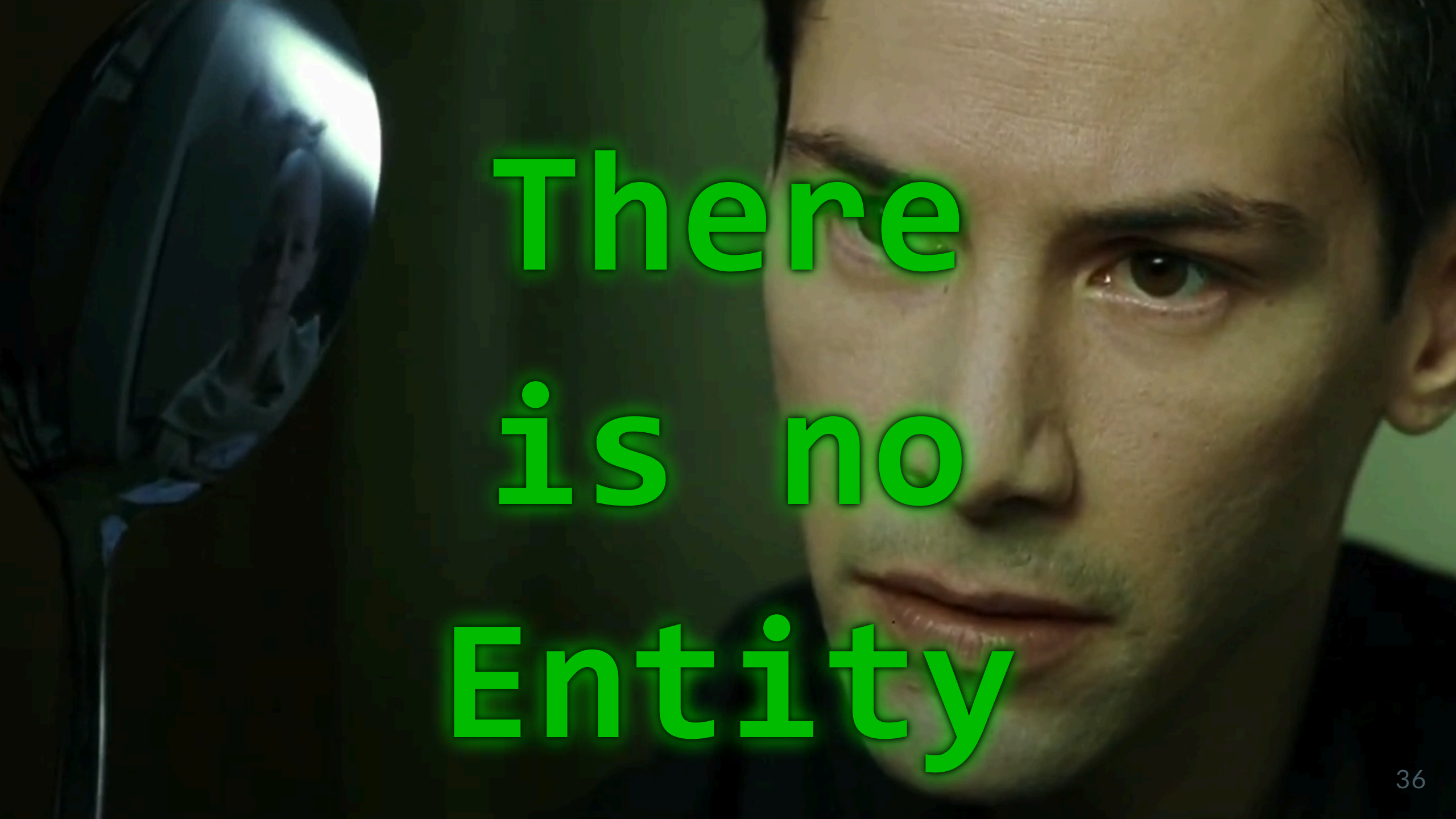
ООП, Entity-Component: Per-component pools

```
void* PlacementComponent::operator new(std::size_t count) {  
    return Allocator<PlacementComponent>::get().alloc_new();  
}  
  
void* MoveComponent::operator new(std::size_t count);  
  
void* SpriteComponent::operator new(std::size_t count);  
  
void* AvoidanceComponent::operator new(std::size_t count);
```

Looks good. But
Всё ещё Entity-Centric...

Результаты

- 90 ms -> 62 ms ✓



There
is no
Entity

Варианты реализации

- **Object-oriented. Entity-Centric**
 - Свойства и логика сгруппированы в Entity
 - Наследование
- **Object-oriented. Entity-Component**
 - Свойства и логика сгруппированы в Entity
 - Агрегирование
- **Data-oriented. Entity Component System**
 - Свойства и логика независимы

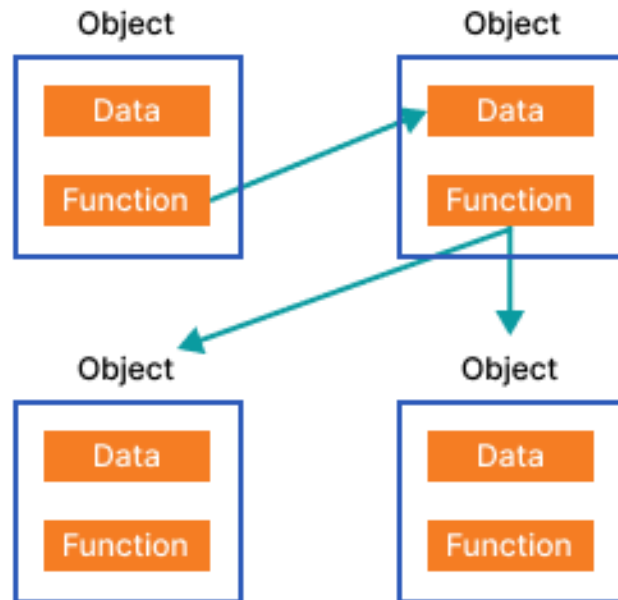


ООП vs. Data-oriented Design

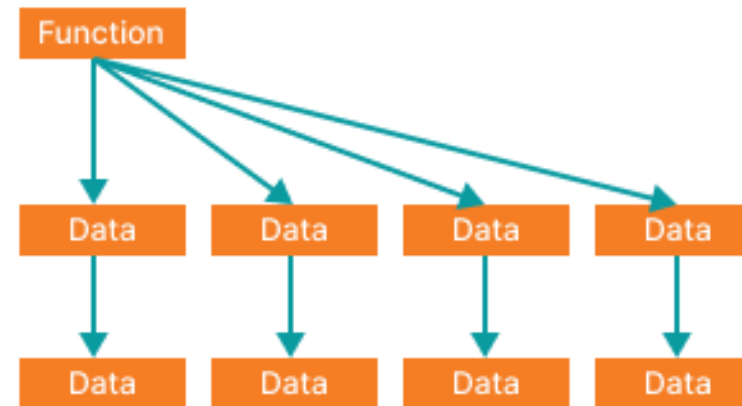
The purpose of all programs, and all parts of those programs, is to **transform data from one form to another.**

– Mike Acton

Object-oriented Programming (OOP)



Data-oriented Design (DOD)



ООП vs. Data-oriented Design

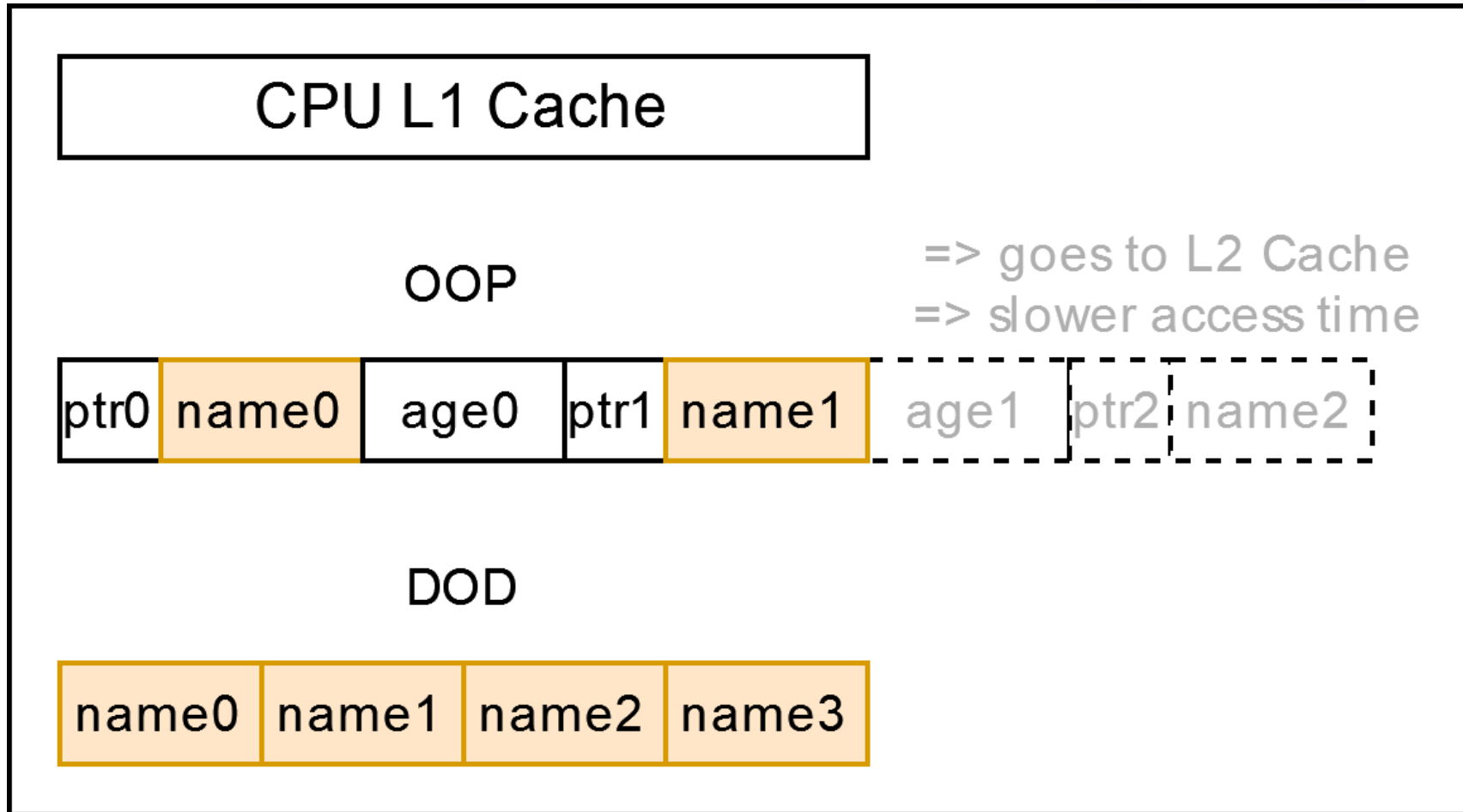
ООП

- **AoS** Array of Structures
- Данные разбросаны по памяти
- Логика привязана к объектам
 - Абстракции
- Наследование
- Динамический полиморфизм
- Удобный дизайн?
- Привычно

DoD

- **SoA** Structure of Arrays
- Данные последовательны
- Логика привязана к потокам данных
 - Данные и функции
- Агрегирование
- Duck-typing
- Скорость выполнения
- Необычно

ООП vs. Data-oriented Design



ECS (Entity Component System)

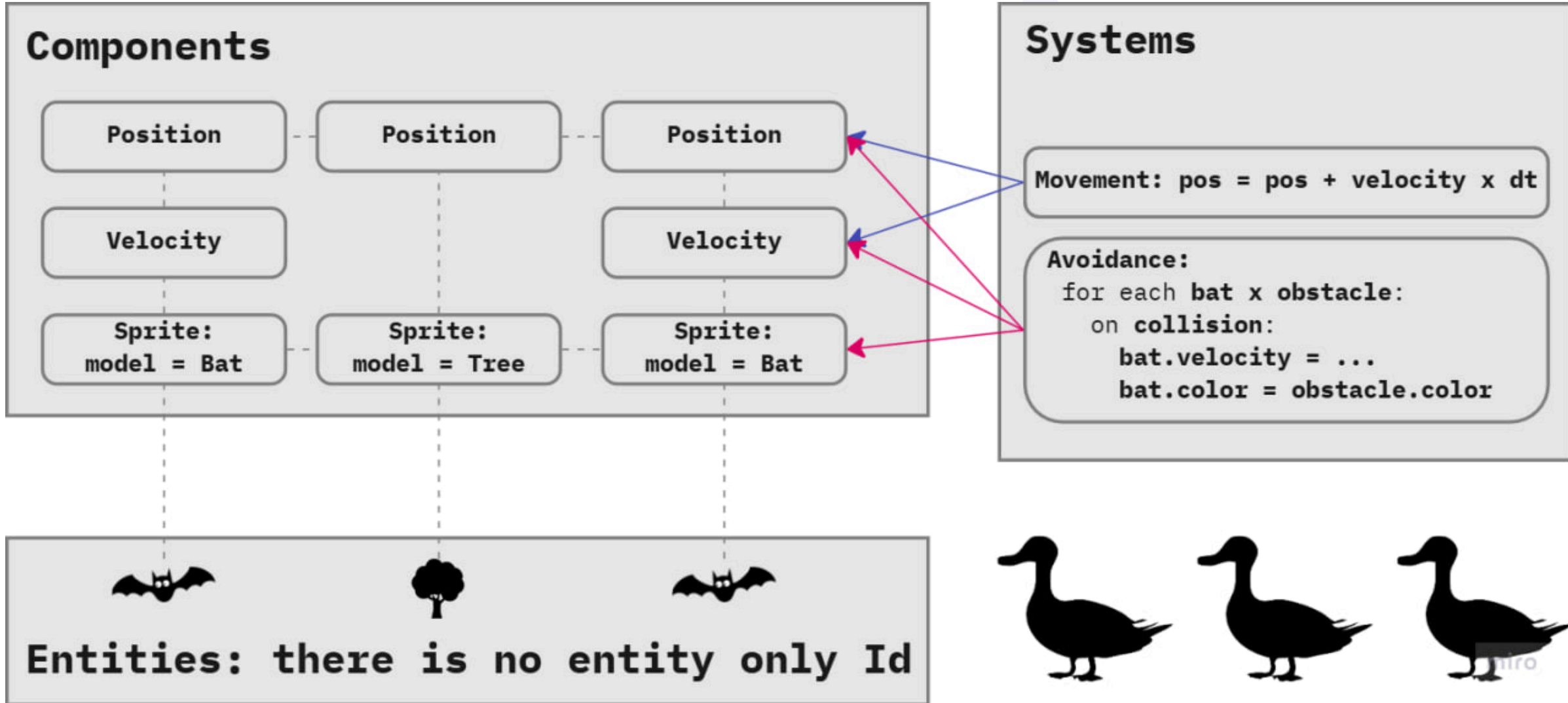
ECS — разновидность **Data-oriented Design**

- **Component** — Данные
- **System** — Логика
- **Entity** — ???

Типичный flow:

- Для каждой системы
 - Делаем выборку нужных компонентов (чем меньше типов в выборке — тем лучше)
 - Кэшируем явно (если нужно) выбранные компоненты
 - Производим трансформации компонентов

ECS (Entity Component System)



ECS: World on Flecs

```
struct Bat {}; // Just a tag
struct Obstacle {}; // Just a tag

flecs::world world;

world.component<Position>();
world.component<Velocity>();
world.component<Sprite>();
world.component<Bat>();
world.component<Obstacle>();

world.system<Position, Velocity>("MoveSystem")
    .each([&](const flecs::iter& it, size_t row, Position& p, Velocity& v) {
        p.x += v.x * it.delta_time();
        p.y += v.y * it.delta_time();

        if (v.x < 0.0f && p.x < bounds.x0 || v.x > 0.0f && p.x > bounds.x1) v.x = -v.x;
        if (v.y < 0.0f && p.y < bounds.y0 || v.y > 0.0f && p.y > bounds.y1) v.y = -v.y;
    });

world.system("AvoidanceSystem").run(avoidanceSystem);
```

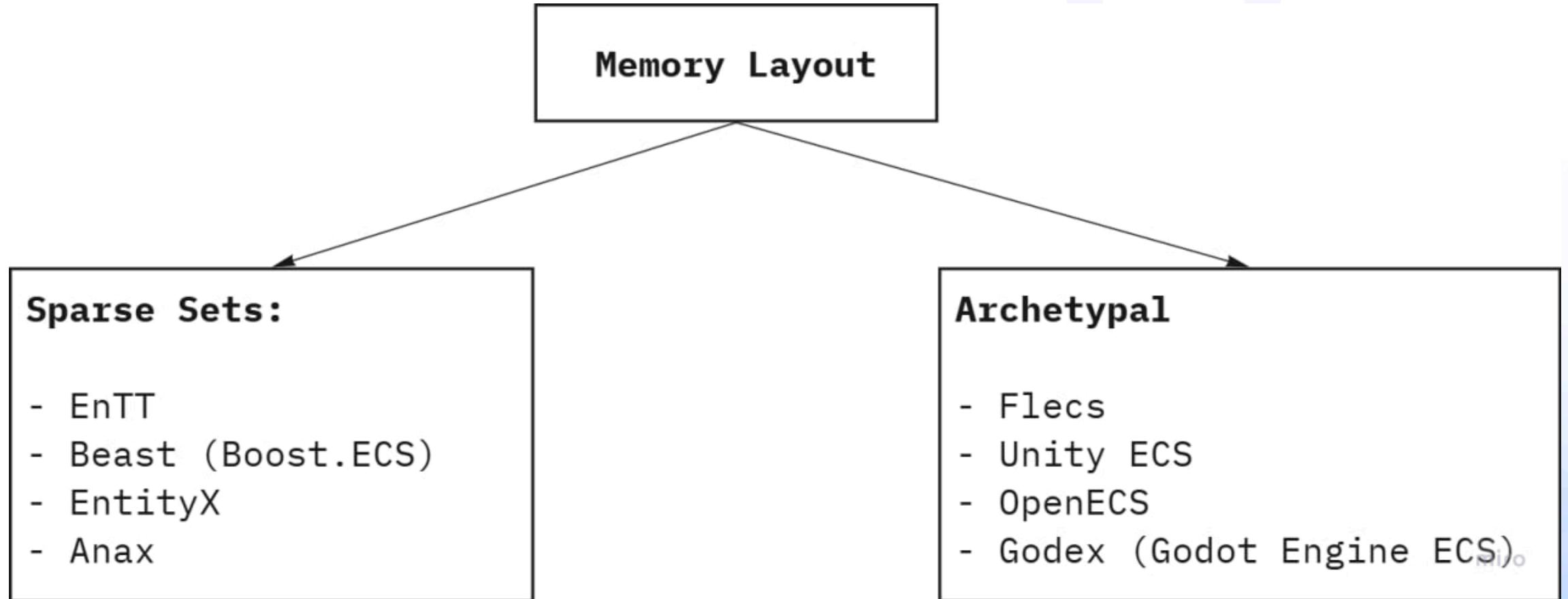
ECS: Тесты

```
for (int i = 0; i < 20; ++i) {
    world.entity()
        .add<Obstacle>()
        .emplace<Sprite>(Sprite::Tree, randColor())
        .emplace<Position>(frand(), frand());
}

for (int i = 0; i < 1'000'000; ++i) {
    world.entity()
        .add<Bat>()
        .emplace<Sprite>(Sprite::Bat, randColor())
        .emplace<Position>(frand(), frand())
        .emplace<Velocity>(frand() * 0.5f, frand() * 0.5f);
}
```

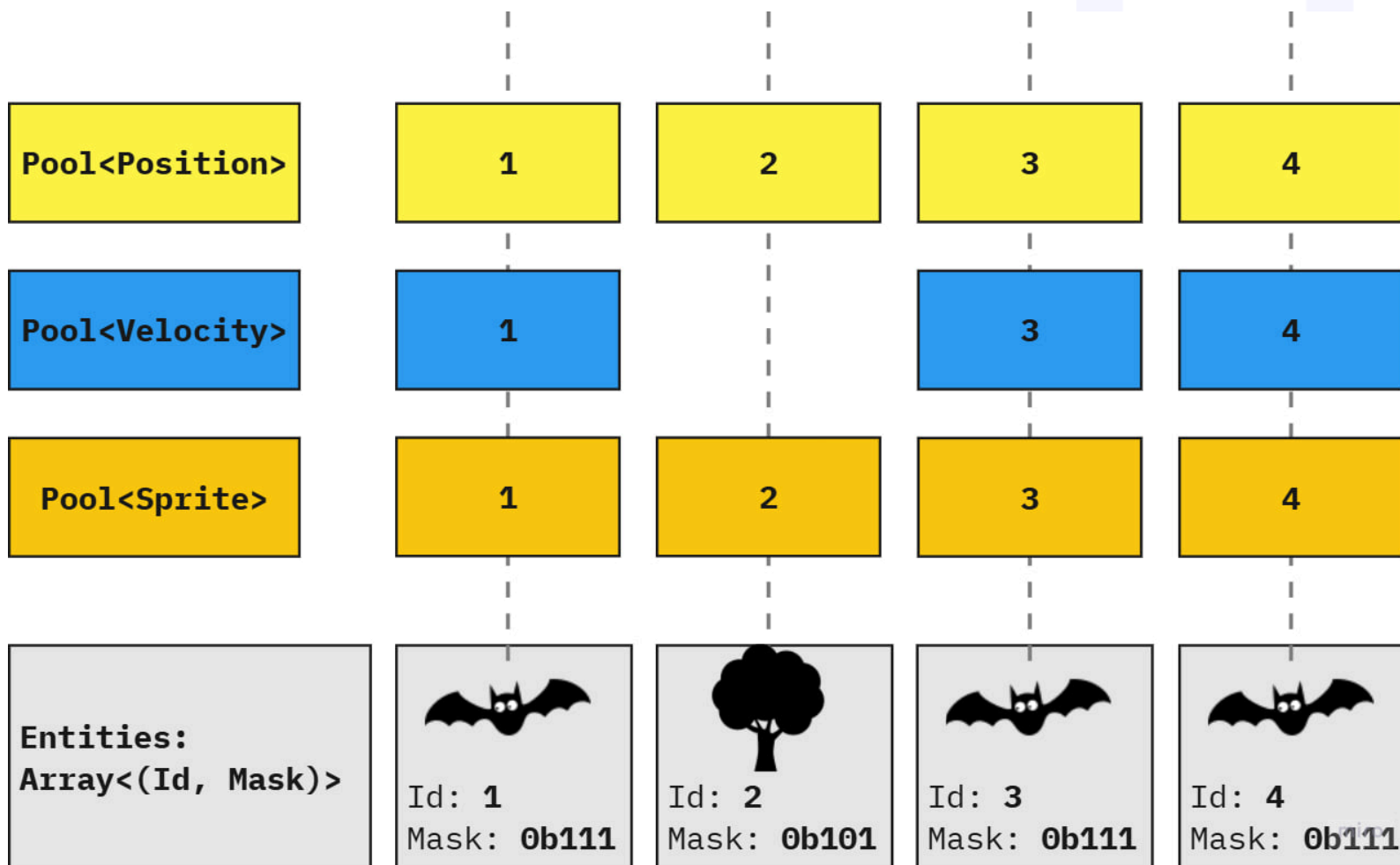
• **62 ms -> 47 ms** ✓

ECS: Memory layout



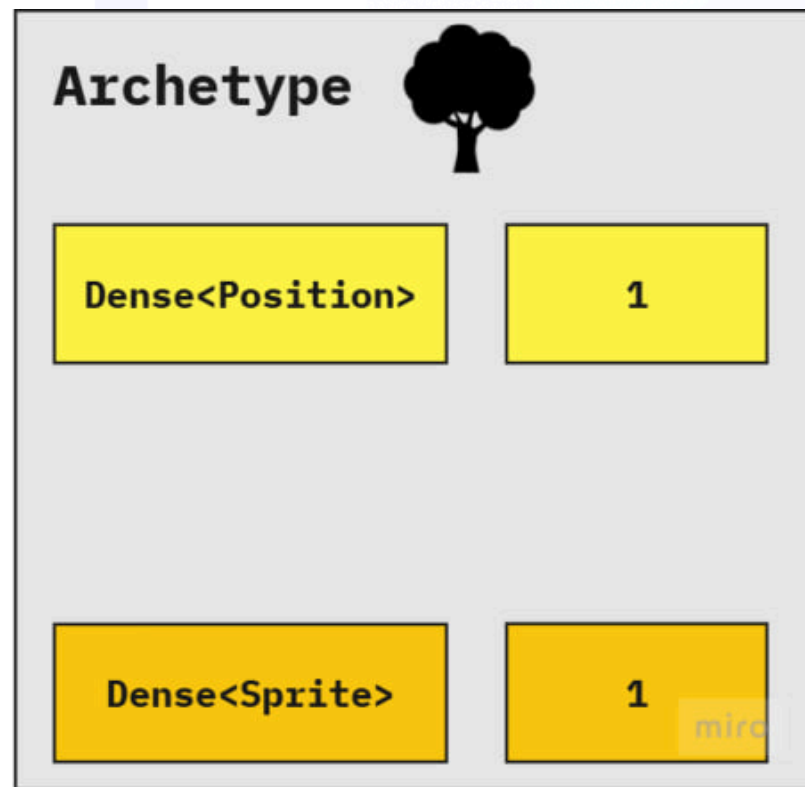
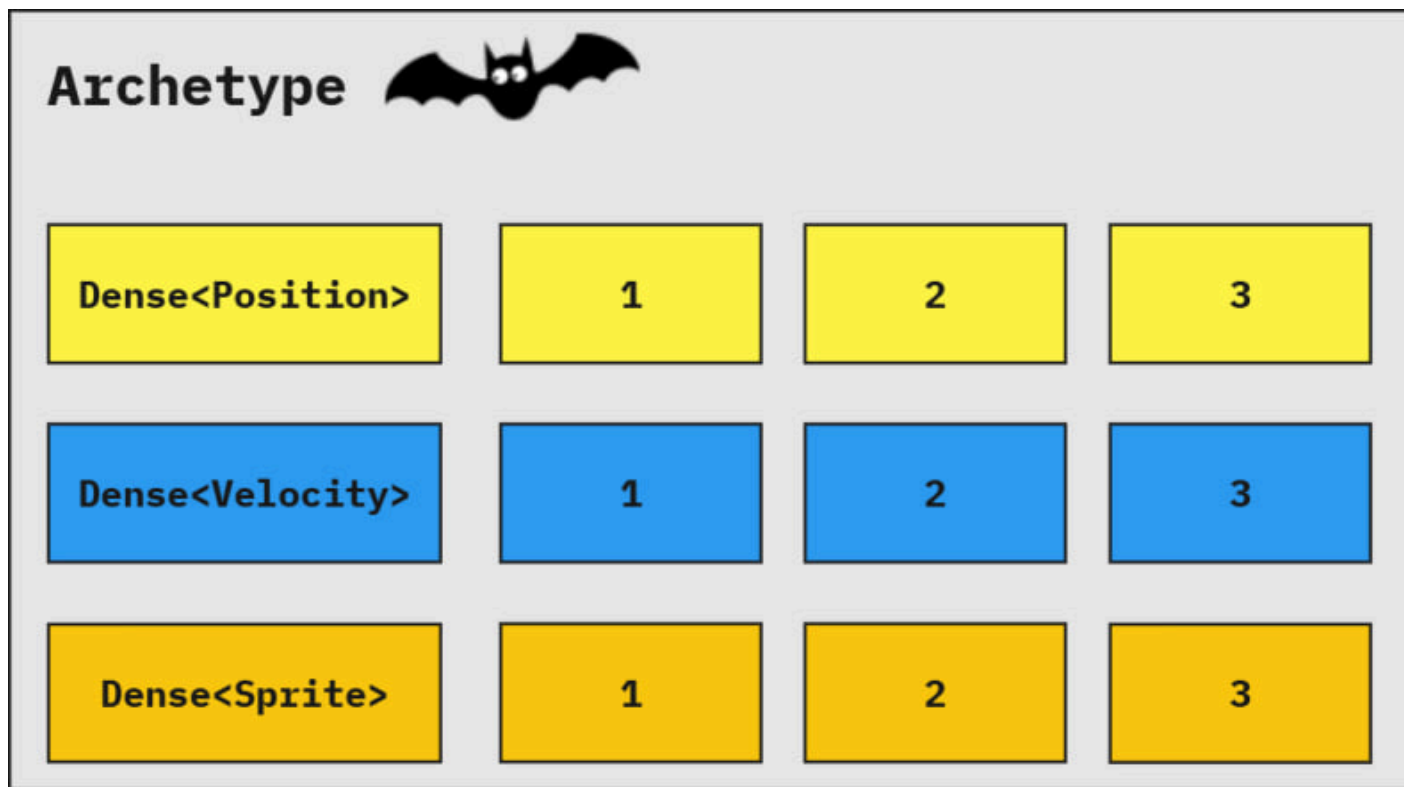
ECS: Memory layout, Sparse Sets

- Просто, Добавление/удаление – дешево, обход всех Entity – **cache misses**



ECS: Memory layout, Archetypal

- Табличное представление данных (Таблица – Архетип, Поля – Компоненты)
- Обход всех Entity и компонентов более плотный —> меньше промахов
- При добавлении/удалении компонента —> перемещение в другой архетип



Многопоточность: ООП

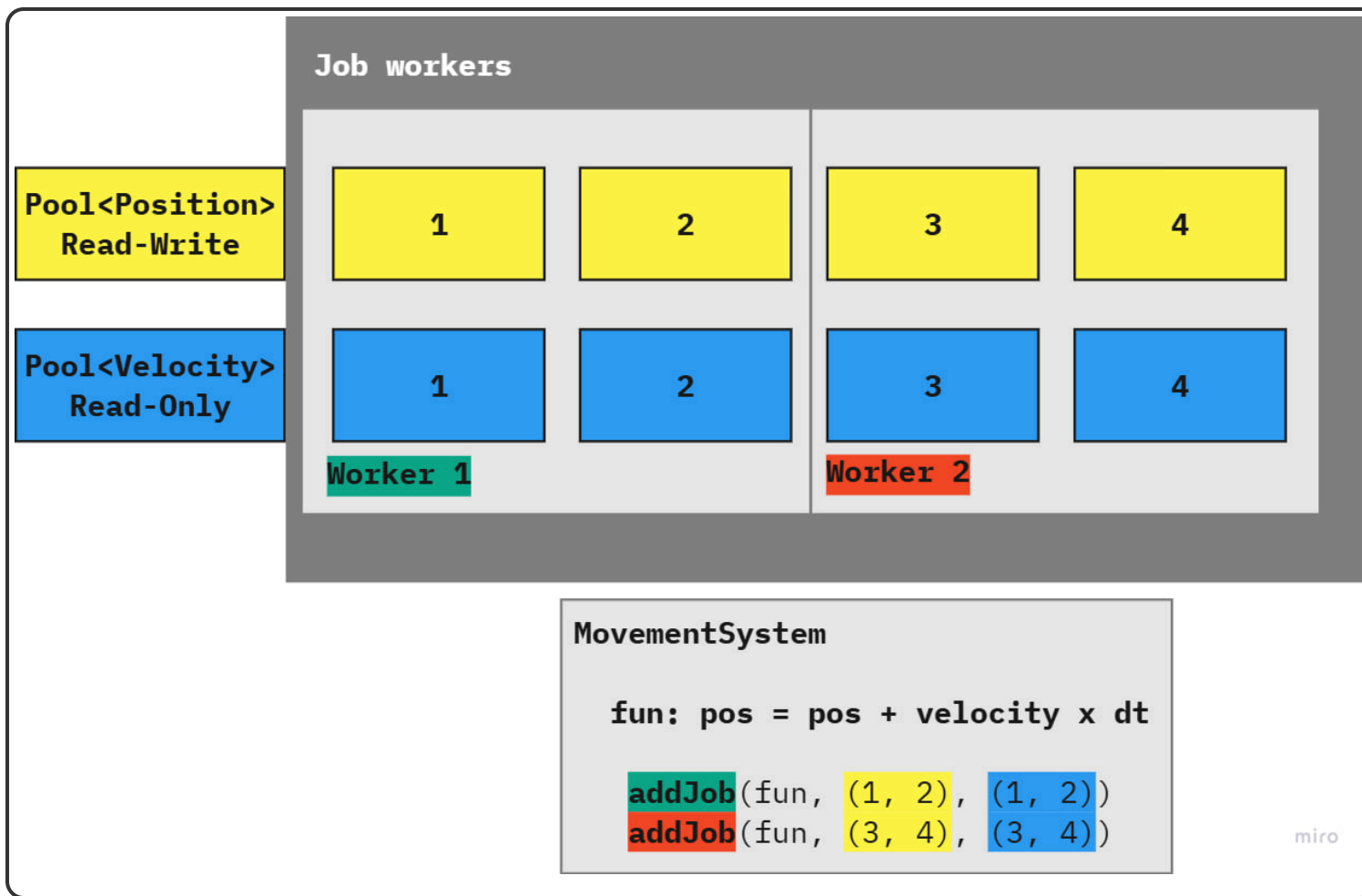
Основная проблема ООП и Shared State -

разнородные данные и логика связаны внутри объектов, что приводит к:

- Гонки при одновременном доступе из разных потоков
- Необходимость в блокировках — замедление
- Cache Coherency Overhead, False Sharing, Cache Trashing

ECS решает эти проблемы благодаря **Systems**, работающим отдельно от **Components**.

Многопоточность: ECS



Flecs:

```
world.system<Position, Velocity>()
    .multi_threaded(true)
    .each([](Position& p, Velocity& v) {
        p.x += v.x * dt;
        p.y += v.y * dt;
    });
```

Неутешительные выводы

ООП

- Frame delta: **175 ms**
 - Cache misses
 - Multithreading – Hard
- Понятный дизайн? Сомнительно
 - Данные и логика смешаны
 - "Классы-кентавры"
 - Наследование

ECS

- Frame delta: **47 ms** ✓
 - Cache friendly
 - Multithreading – Easy
- Понятный дизайн? Сомнительно
 - Данные и логика разделены
 - Независимые компоненты
 - Агрегирование

- Дизайн определяет производительность?

- ООП – наследование + Custom pool allocators = компромисс? **62 ms** ✓

Спасибо за внимание!

 [@ProgDevil](https://t.me/ProgDevil)

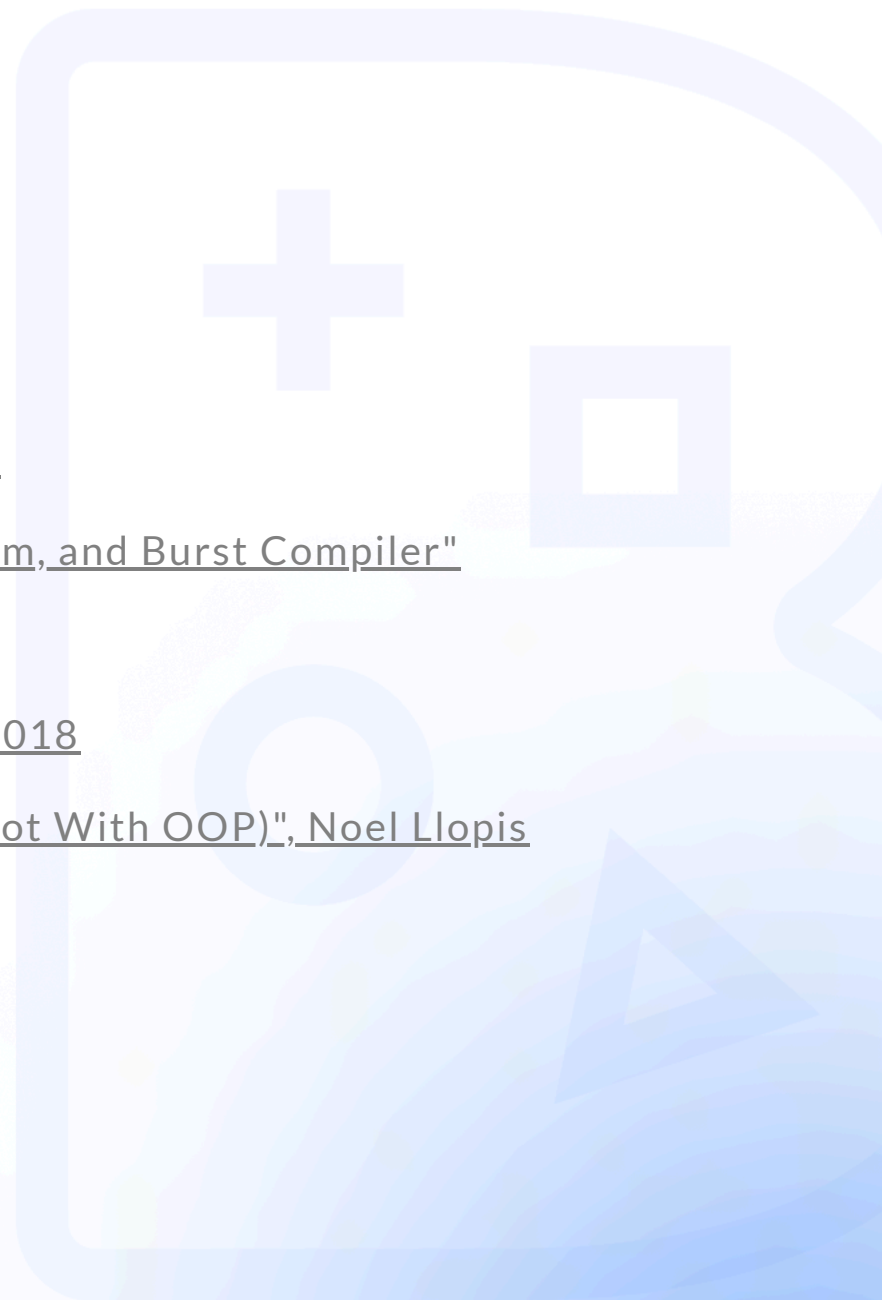
 t.me/BlackHubG



BLACKHUB
games

Материалы

- ["Entity Component Systems & Data Oriented Design", Aras Pranckevičius](#)
- ["Using Rust for Game Development", Catherine West \(@Kyrenite\)](#)
- ["Практика применения C++ в играх и игровых движках", Антон Яковлев](#)
- ["Get Started with the Unity Entity Component System \(ECS\), C# Job System, and Burst Compiler"](#)
- ["Data-Oriented Design and C++", Mike Acton, CppCon 2014](#)
- ["OOP Is Dead, Long Live Data-oriented Design", Stoyan Nikolov, CppCon 2018](#)
- ["Data-Oriented Design \(Or Why You Might Be Shooting Yourself in The Foot With OOP\)", Noel Llopis](#)
- ["Practical Examples in Data Oriented Design slides", Niklas Gray](#)
- ["Typical C++ Bullshit slide gallery", Mike Acton](#)
- ["Data-Oriented Design blog post & links", Adam Sawicki](#)



GitHub

- [Flecs](#)
- [EnTT](#)

