Entity Component Systems & Data Oriented Design

Unity Training Academy 2018-2019, #3
Aras Pranckevičius

ADAM – Made with Unity

Outline

- *All this will **not** be Unity specific!*

- A rant on Object Oriented Design
- Data Oriented Design
- Entity Component Systems
- Practical Example

Problem

Object Oriented Design/ Programming

Typical Implementation of OO

- Class hierarchies
- Virtual functions
- Encapsulation often violated since stuff Needs To Know
- “One Thing At A Time” approach
- Late decisions

This is going to be…
OOP party like it’s 1999

Simple OO component system: Component

```
// Component base class. Knows about the parent game object, and has some virtual methods.
class Component
{
public:
    Component() : m_GameObject(nullptr) {}
    virtual ~Component() {}

    virtual void Start() {}
    virtual void Update(double time, float deltaTime) {}

    const GameObject& GetGameObject() const { return *m_GameObject; }
    GameObject& GetGameObject() { return *m_GameObject; }
    void SetGameObject(GameObject& go) { m_GameObject = &go; }

private:
    GameObject* m_GameObject;
};
```

Simple OO component system: GameObject

```
// Game object class. Has an array of components.
class GameObject
{
public:
    GameObject(const std::string&& name) : m_Name(name) { }
    ~GameObject() { for (auto c : m_Components) delete c; }

    // get a component of type T, or null if it does not exist on this game object
    template<typename T>
    T* GetComponent()
    {
        for (auto i : m_Components) { T* c = dynamic_cast<T*>(i); if (c != nullptr) return c; }
        return nullptr;
    }

    // add a new component to this game object
    void AddComponent(Component* c)
    {
        c->SetGameObject(*this); m_Components.emplace_back(c);
    }

    void Start() { for (auto c : m_Components) c->Start(); }
    void Update(double time, float deltaTime) { for (auto c : m_Components) c->Update(time, deltaTime); }

private:
    std::string m_Name;
    ComponentVector m_Components;
};
```

Simple OO component system: Utilities

```cpp
// Finds all components of given type in the whole scene
template<typename T>
static ComponentVector FindAllComponentsOfType()
{
    ComponentVector res;
    for (auto go : s_Objects)
    {
        T* c = go->GetComponent<T>();
        if (c != nullptr) res.emplace_back(c);
    }
    return res;
}

// Find one component of given type in the scene (returns first found one)
template<typename T>
static T* FindOfType()
{
    for (auto go : s_Objects)
    {
        T* c = go->GetComponent<T>();
        if (c != nullptr) return c;
    }
    return nullptr;
}
```

Simple OO component system: various components

```
// 2D position: just x,y coordinates
struct PositionComponent : public Component
{
    float x, y;
};


// Sprite: color, sprite index (in the sprite atlas), and scale for rendering it
struct SpriteComponent : public Component
{
    float colorR, colorG, colorB;
    int spriteIndex;
    float scale;
};
```

Simple OO component system: various components

```
// Move around with constant velocity. When reached world bounds, reflect back from them.
struct MoveComponent : public Component
{
    float velx, vely;
    WorldBoundsComponent* bounds;

    MoveComponent(float minSpeed, float maxSpeed)
    {
        /* … */
    }

    virtual void Start() override
    {
        bounds = FindOfType<WorldBoundsComponent>();
    }

    virtual void Update(double time, float deltaTime) override
    {
        /* … */
    }
};
```

Simple OO component system: components logic

```
virtual void Update(double time, float deltaTime) override
{
    // get Position component on our game object
    PositionComponent* pos = GetGameObject().GetComponent<PositionComponent>();

    // update position based on movement velocity & delta time
    pos->x += velx * deltaTime;
    pos->y += vely * deltaTime;

    // check against world bounds; put back onto bounds and mirror
    // the velocity component to "bounce" back
    if (pos->x < bounds->xMin) { velx = -velx; pos->x = bounds->xMin; }
    if (pos->x > bounds->xMax) { velx = -velx; pos->x = bounds->xMax; }
    if (pos->y < bounds->yMin) { vely = -vely; pos->y = bounds->yMin; }
    if (pos->y > bounds->yMax) { vely = -vely; pos->y = bounds->yMax; }
}
```

Simple OO component system: game update loop

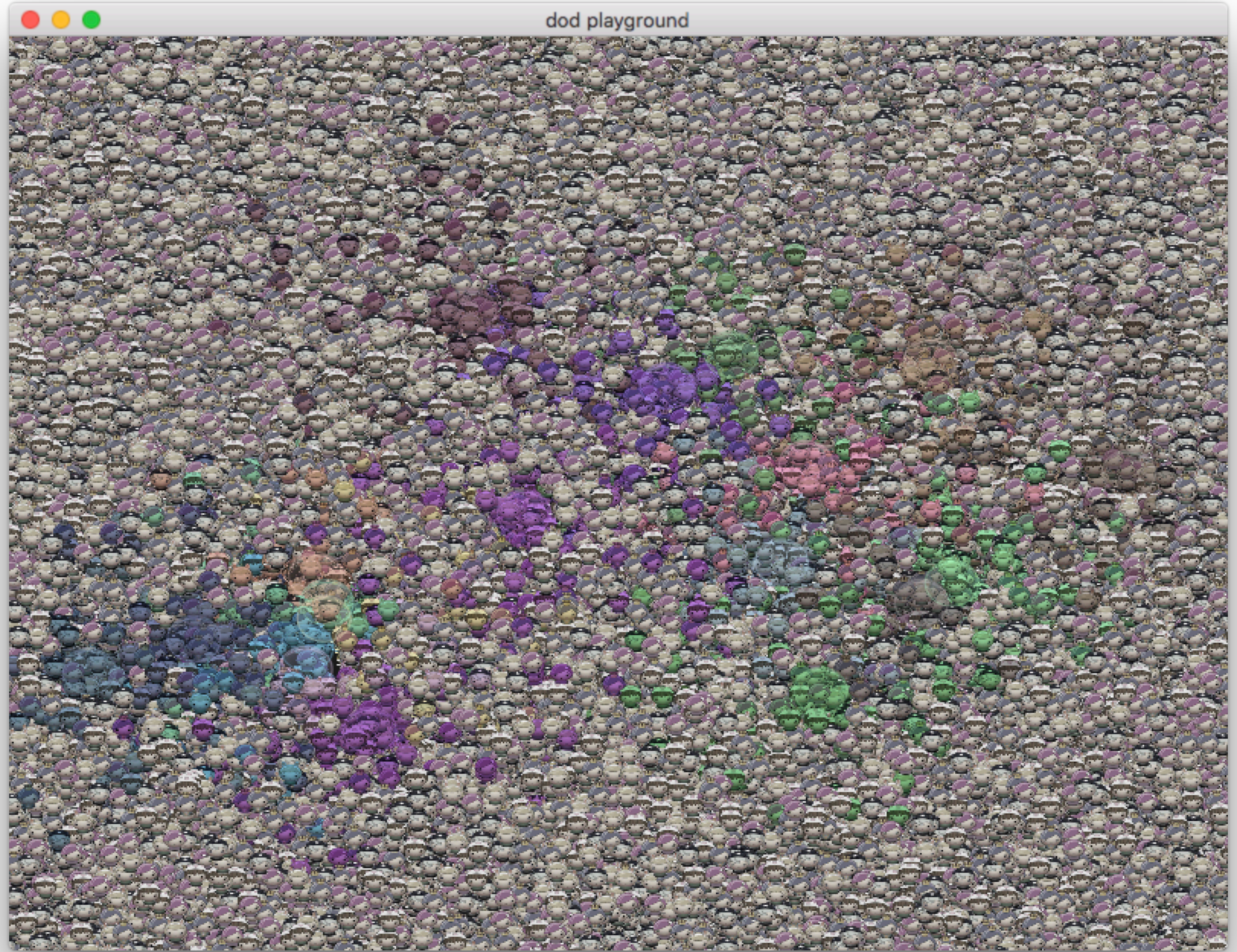
```
void GameUpdate(sprite_data_t* data, double time, float deltaTime)
{
    // go through all objects
    for (auto go : s_Objects)
    {
        // Update all their components
        go->Update(time, deltaTime);

        // For objects that have a Position & Sprite on them: write out
        // their data into destination buffer that will be rendered later on.
        PositionComponent* pos = go->GetComponent<PositionComponent>();
        SpriteComponent* sprite = go->GetComponent<SpriteComponent>();
        if (pos != nullptr && sprite != nullptr)
        {
            /* … emit data for sprite rendering … */
       }
    }
}
```

Let’s make a simple “game” with this!

- Sprites that move around & bounce from world edges
- Bubbles, move around slowly
- Sprites bounce from bubbles, and get their color

Let’s make a simple “game” with this!



Issues with OO design: where to put code?

- Many systems in games do not belong to “one object”
  - e.g. Collision, Damage, AI: work on 2+ objects
- “Sprites avoid Bubbles” in our game:
  - put avoidance logic onto thing that avoids something?
  - put avoidance logic onto thing that should be avoided?
  - somewhere else?

Issues with OO design: where to put code?

- Many languages are “single dispatch”
  - there are Objects, and Methods that work with them
- But what we need is “multiple dispatch”
  - Avoidance system works on two sets of objects

Issues with OO design: hard to know what does what

- Ever opened a Unity project and tried to figure out how it works?
  - …yeah, that :)
  - “game logic” scattered around in million components, with no overview

Issues with OO design: “messy base class” problem

```
EntityType entityType() const override;

void init(World* world, EntityId entityId, EntityMode mode) override;
void uninit() override;

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;

// relative to current position
RectF metaBoundBox() const override;

// relative to current position
RectF collisionArea() const override;
// … continued …
```

Pasted from “How many accessors could you possibly need?”, Catherine West
https://kyren.github.io/rustconf_2018_slides/index.html

Issues with OO design: “messy base class” problem

```
// … continued …
void hitOther(EntityId targetEntityId, DamageRequest const& damageRequest) override;
void damagedOther(DamageNotification const& damage) override;

List<DamageSource> damageSources() const override;

bool shouldDestroy() const override;
void destroy(RenderCallback* renderCallback) override;

Maybe<EntityAnchorState> loungingIn() const override;
bool lounge(EntityId loungeableEntityId, size_t anchorIndex);
void stopLounging();
// … continued …
```

Issues with OO design: “messy base class” problem

```
// … continued …
float health() const override;
float maxHealth() const override;
DamageBarType damageBar() const override;
float healthPercentage() const;

float energy() const override;
float maxEnergy() const;
float energyPercentage() const;
float energyRegenBlockPercent() const;

bool energyLocked() const override;
bool fullEnergy() const override;
bool consumeEnergy(float energy) override;

float foodPercentage() const;

float breath() const;
float maxBreath() const;
// … continued …
```

Issues with OO design: “messy base class” problem

```
// … continued …
void playEmote(HumanoidEmote emote) override;

bool canUseTool() const;

void beginPrimaryFire();
void beginAltFire();

void endPrimaryFire();
void endAltFire();

void beginTrigger();
void endTrigger();

ItemPtr primaryHandItem() const;
ItemPtr altHandItem() const;
// … etc.
```

This is not the best OO design, and it certainly is possible to make a better one.
But also, often code ends up being like this, even if no one wanted it that way.

Issues with OO design: performance

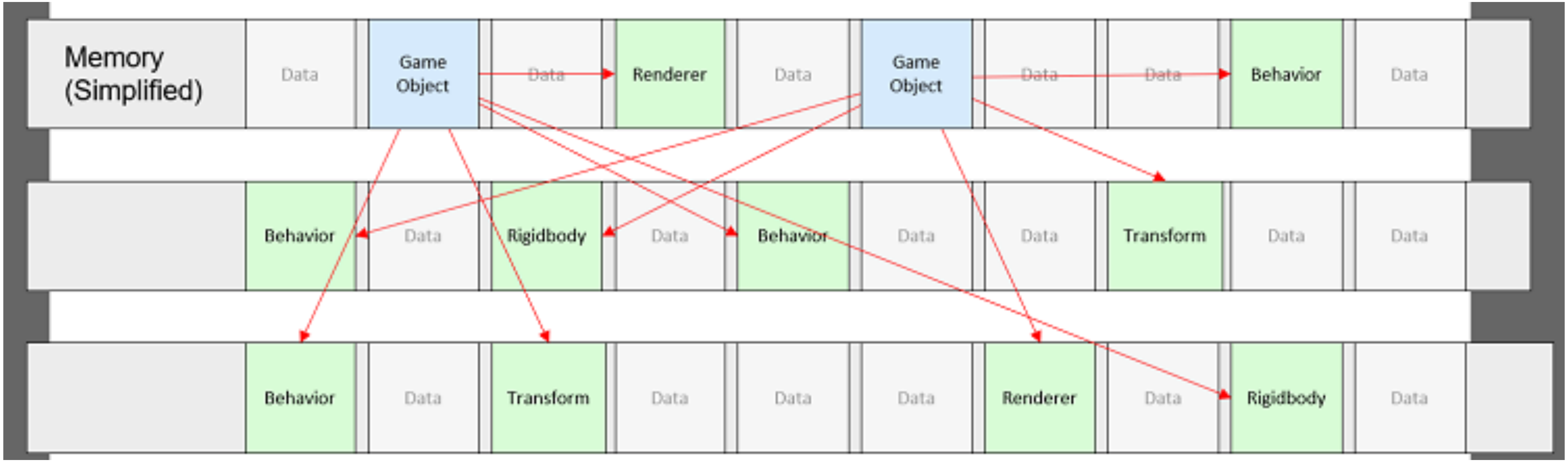
- 1 million sprites, 20 bubbles:
  - **330ms** game update
  - **470ms** startup time
- Low-hanging fruit stupidities
- Data scattered around in memory
- Virtual function calls

Timings on 2018 MacBookPro (2.9GHz Core i9), Xcode, Release build.
Code: https://github.com/aras-p/dod-playground/tree/3529f232

Issues with OO design: memory usage

- 1 million sprites, 20 bubbles:
  - **310MB** RAM usage
- Every Component has pointer to GameObject, but very few need it
- Every Component has a pointer to virtual function table
- Each GameObject/Component allocated individually

Issues with OO design: typical memory view



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

Issues with OO design: optimizability

- How would you multi-thread it?
- Or make it run on a GPU?

- In many OO designs doing that is **very hard**
    - Not clear who **reads** which data, and who **writes** which data

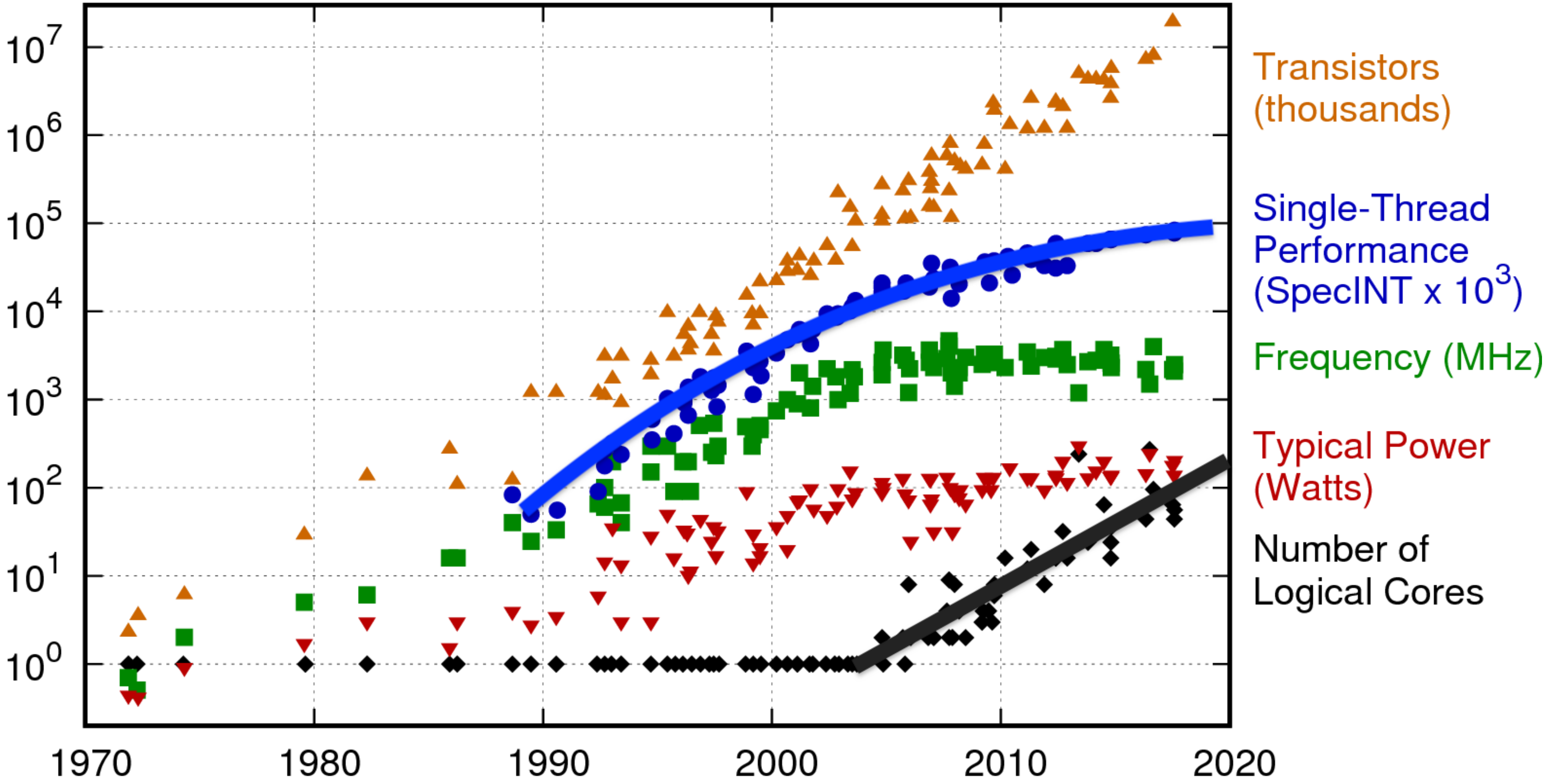
Issues with OO design: testability

- How would you write tests for this?
- OO designs often need **a lot** of setup/mocking/faking to test.
  - Create object hierarchies, managers, adapters, singletons, …

Intermission

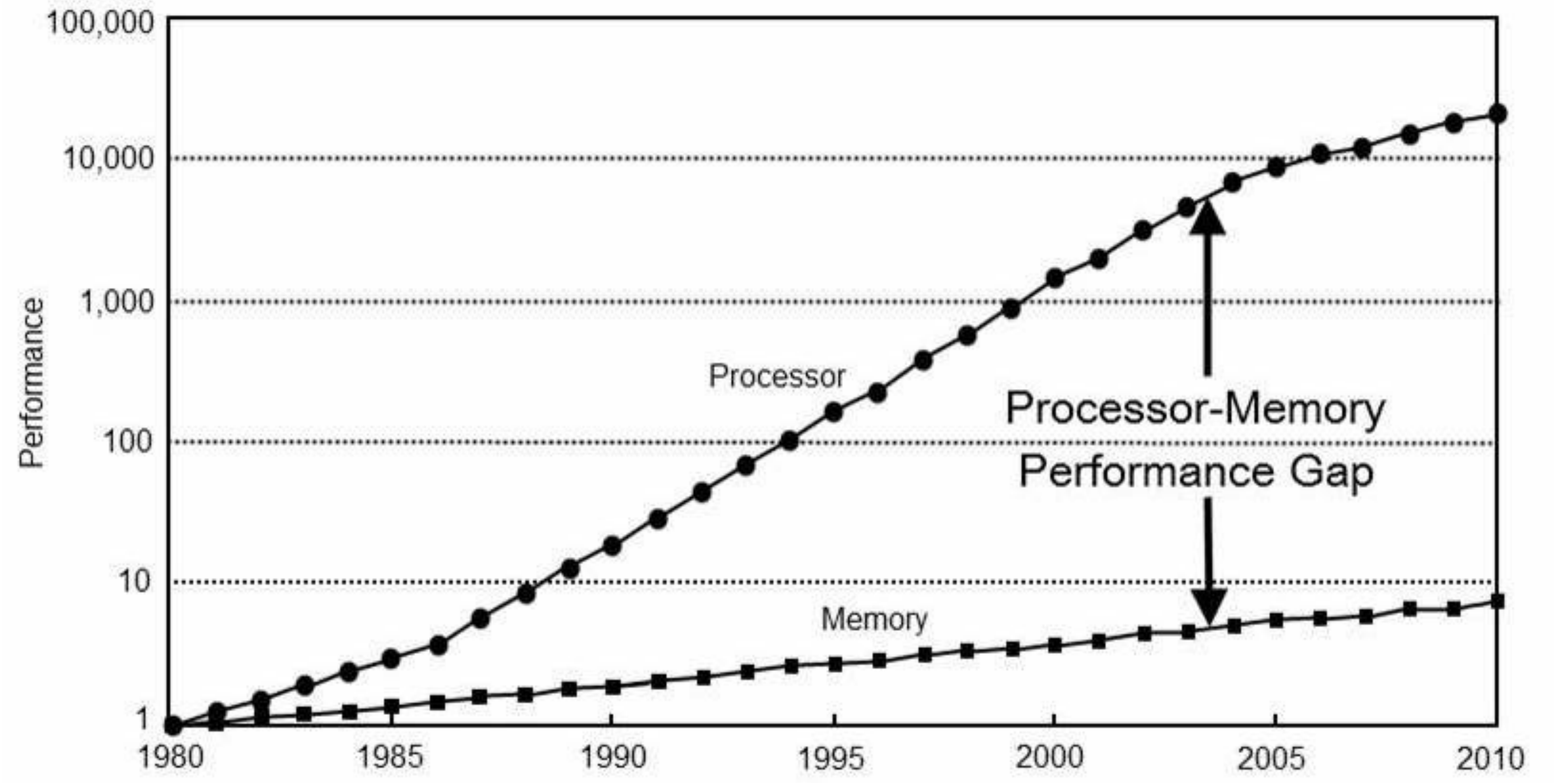
A Bit About Computer Hardware…

CPU performance trends*



* from https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

CPU-RAM performance gap*



* from Computer Architecture: A Quantitative Approach

Latency Numbers in Computers*

- Read from **CPU L1 cache: 0.5**ns
- Branch mispredict: 5ns
- Read from CPU L2 cache: 7ns
- Read from **RAM: 100**ns
- Read from SSD: 150’000ns
- Read **1MB from RAM: 250’000**ns
- Send network packet CA->NL->CA: 150’000’000ns

* from https://gist.github.com/hellerbarde/2843375 as of 2012
today some numbers slightly different, but rough ballpark similar

Latency Numbers in Computers, humanized*

- Read from **CPU L1 cache**: 0.5s - **one heart beat**
- Branch mispredict: 5s - yawn
- Read from CPU L2 cache: 7s - long yawn
- Read from **RAM**: 100s - **brushing teeth**
- Read from SSD: 1.7 days - a weekend
- Read **1MB from RAM**: 2.9 days - **a long weekend**
- Send network packet CA->NL->CA: 4.9 years - University with some slack

* multiply by a billion!

Alternatives to Traditional OO

Does Code and Data need to go together?

- Typical OO puts both Code and Data together in one class
- **Why**, though?
- Recall problem of “where to put code”:

```
// this?
class ThingThatAvoids
{
    void AvoidOtherThing(ThingToAvoid* thing);
};

// or this?
class ThingToAvoid
{
    void MakeAvoidMe(ThingThatAvoids* who);
};
```

```
// why not this instead? does not even need to be in a class
void DoAvoidStuff(ThingThatAvoids* who, ThingToAvoid* whom);
```

Data First

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

“If you don’t **understand the data**, you don’t understand the problem.”

— Mike Acton

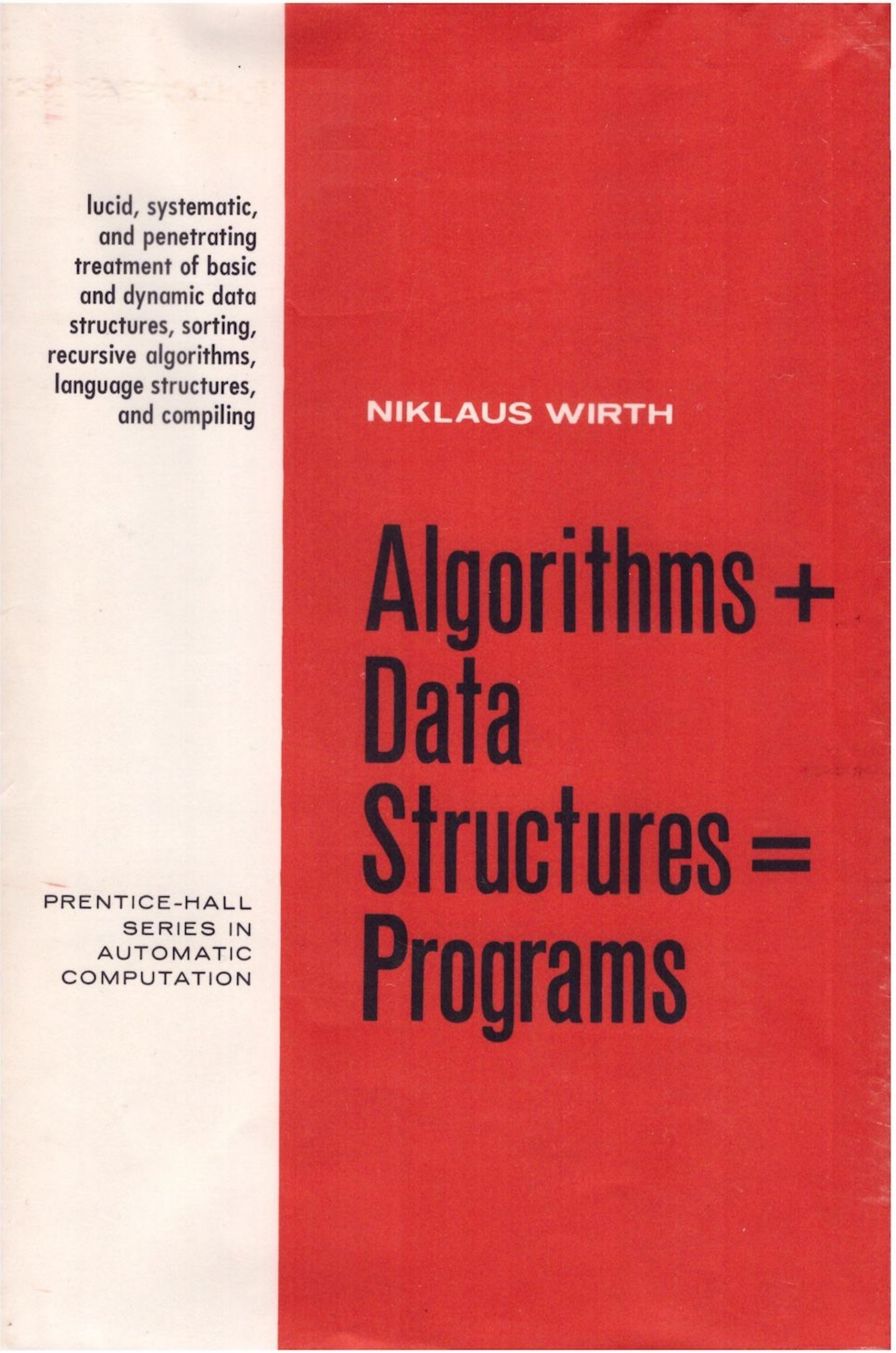
"Data-Oriented Design and C++”, CppCon 2014 https://www.youtube.com/watch?v=rX0ItVEVjHc

Data First

Here’s a 1976 classic book by Niklaus Wirth.

One could argue that “data structures” maybe should be first.

Notice how it does not talk about “objects” at all!



lucid, systematic,
and penetrating
treatment of basic
and dynamic data
structures, sorting,
recursive algorithms,
language structures,
and compiling

NIKLAUS WIRTH

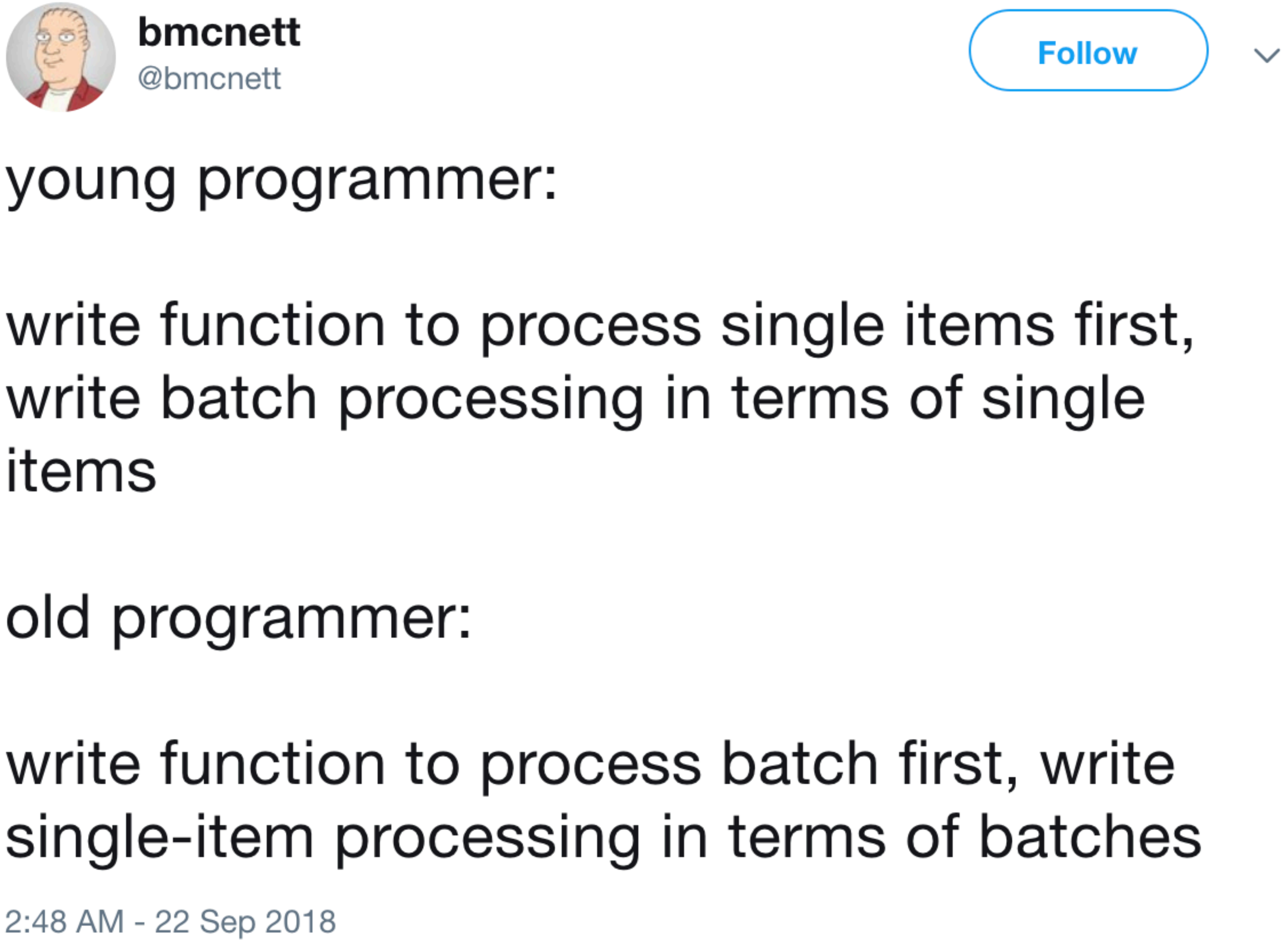
Algorithms +
Data
Structures =
Programs

PRENTICE-HALL
SERIES IN
AUTOMATIC
COMPUTATION

When there is One, there is Many

- How often do you have **one** of a particular thing?
- In games, most common cases are:
    - There’s **a handful** of things. Any code will work here.
    - There’s **way too many** things. Have to be careful with performance.

When there is One, there is Many



bmcnett
@bmcnett

young programmer:

write function to process single items first, write batch processing in terms of single items

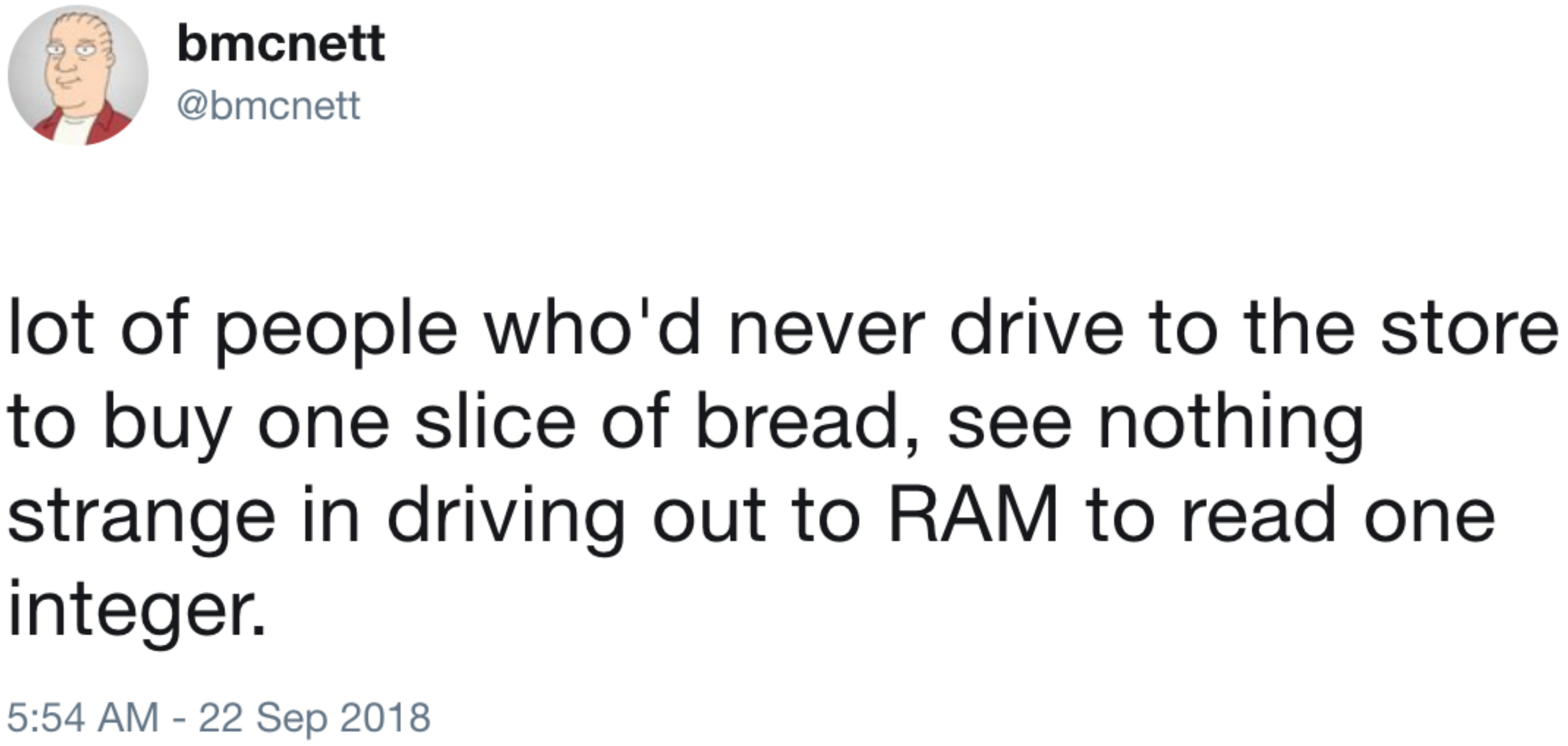
old programmer:

write function to process batch first, write single-item processing in terms of batches

2:48 AM - 22 Sep 2018

https://twitter.com/bmcnett/status/1043285997998432256

When there is One, there is Many



bmcnett
@bmcnett

lot of people who'd never drive to the store to buy one slice of bread, see nothing strange in driving out to RAM to read one integer.

5:54 AM - 22 Sep 2018

https://twitter.com/bmcnett/status/1043332565308923904

When there is One, there is Many

```
virtual void Update(double time, float deltaTime) override
{
    /* move one thing */
}
```

↓

```
void UpdateAllMoves(size_t n, GameObject* objects, double time, float deltaTime)
{
    /* move all of them */
}
```

The Grand Unveil

Data Oriented Design

unity

Data Oriented Design (DOD)

- *… the previous ideas basically already are DOD:*
- **Understand The Data**
  - What is the ideal data needed to solve the problem?
  - How is it laid out?
  - Who reads what and who writes what?
  - What are the patterns in the data?
- **Design For Common Case**
  - Very rarely there is “one” of something
  - Why is your code working on “one” thing at a time?

DOD Resources

- Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP) blog post, Noel Llopis
- Practical Examples in Data Oriented Design slides, Niklas Gray
- Data-Oriented Design and C++ video, Mike Acton
- Typical C++ Bullshit slide gallery, Mike Acton
- Data-Oriented Design blog post & links, Adam Sawicki

The Grand Unveil, Act II

Entity Component Systems

Is traditional Unity GO/Component setup ECS?

- Tradionaly Unity setup uses Components, but not ECS.
- Components solve part of *“Base Class From Hell”* problem, but not others:
  - Hard to reason about logic, data & code flow,
  - Logic (Update etc.) performed on one thing at a time,
  - Inside one type/class (“where to put code” problem),
  - Memory/data locality is not great,
  - A bunch of virtual calls & pointers

Entity-Component-System (ECS)

- Entity: just an **identifier**.
  - Kinda like “primary key” from database? Yes!
- Component: **data**.
- System: **code** that works on entities having certain set(s) of Components.

https://en.wikipedia.org/wiki/Entity-component-system

ECS Resources

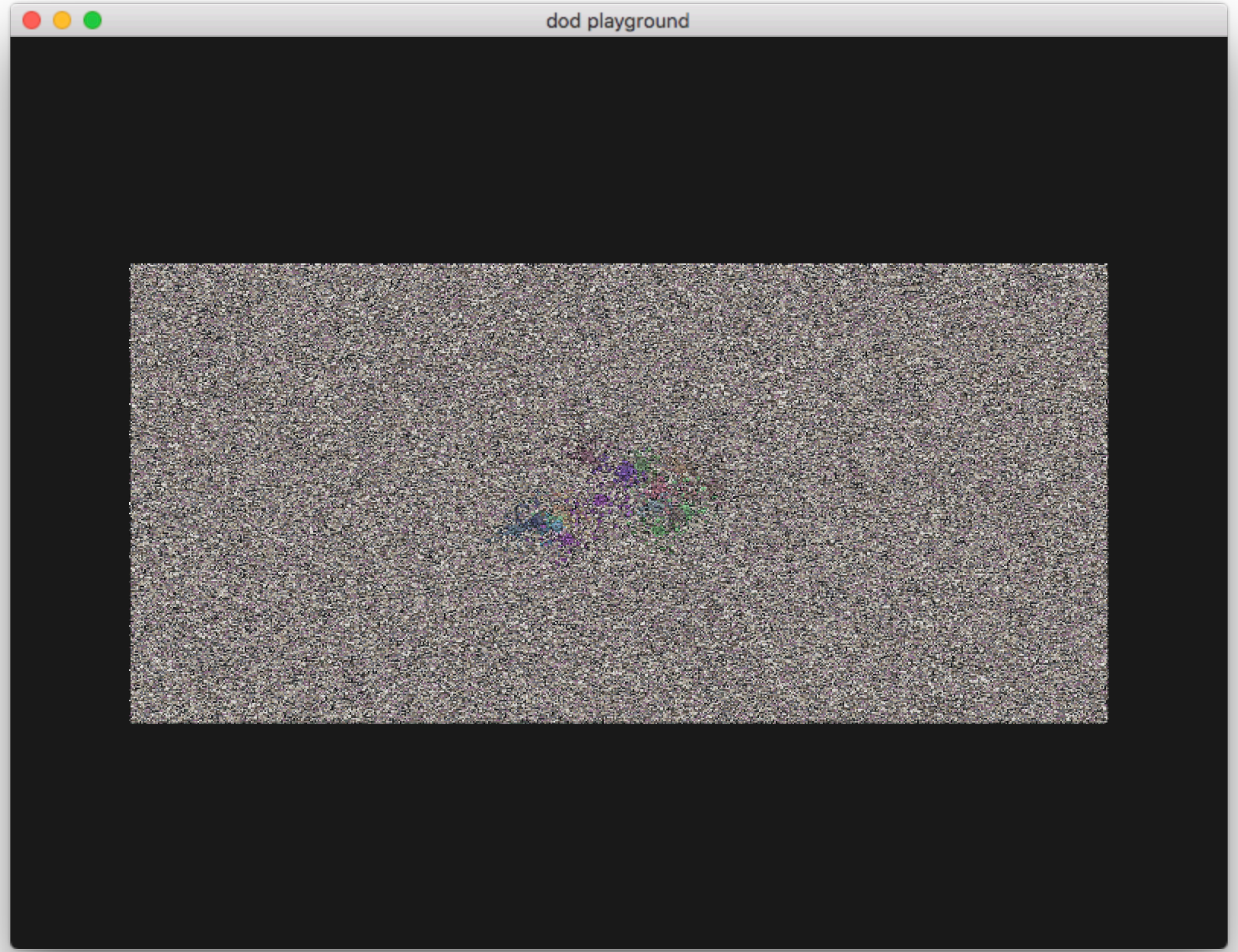
- "Using Rust For Game Development", Catherine West
  - You can just ignore Rust parts, the ECS part is great!
  - Blog, Slides, Video.
- Unity ECS specific:
  - **https://unity3d.com/unity/features/job-system-ECS**: ECS/JobSystem/Burst
  - ECS in Unity Tutorial, Sondre Agledahl
  - Get Started with the Unity ECS, Job System, and Burst, Cristiano Ferreira & Mike Geig

Yeah I’ve no idea what to write here by now

ECS/DOD Example

Recall our simple "game"

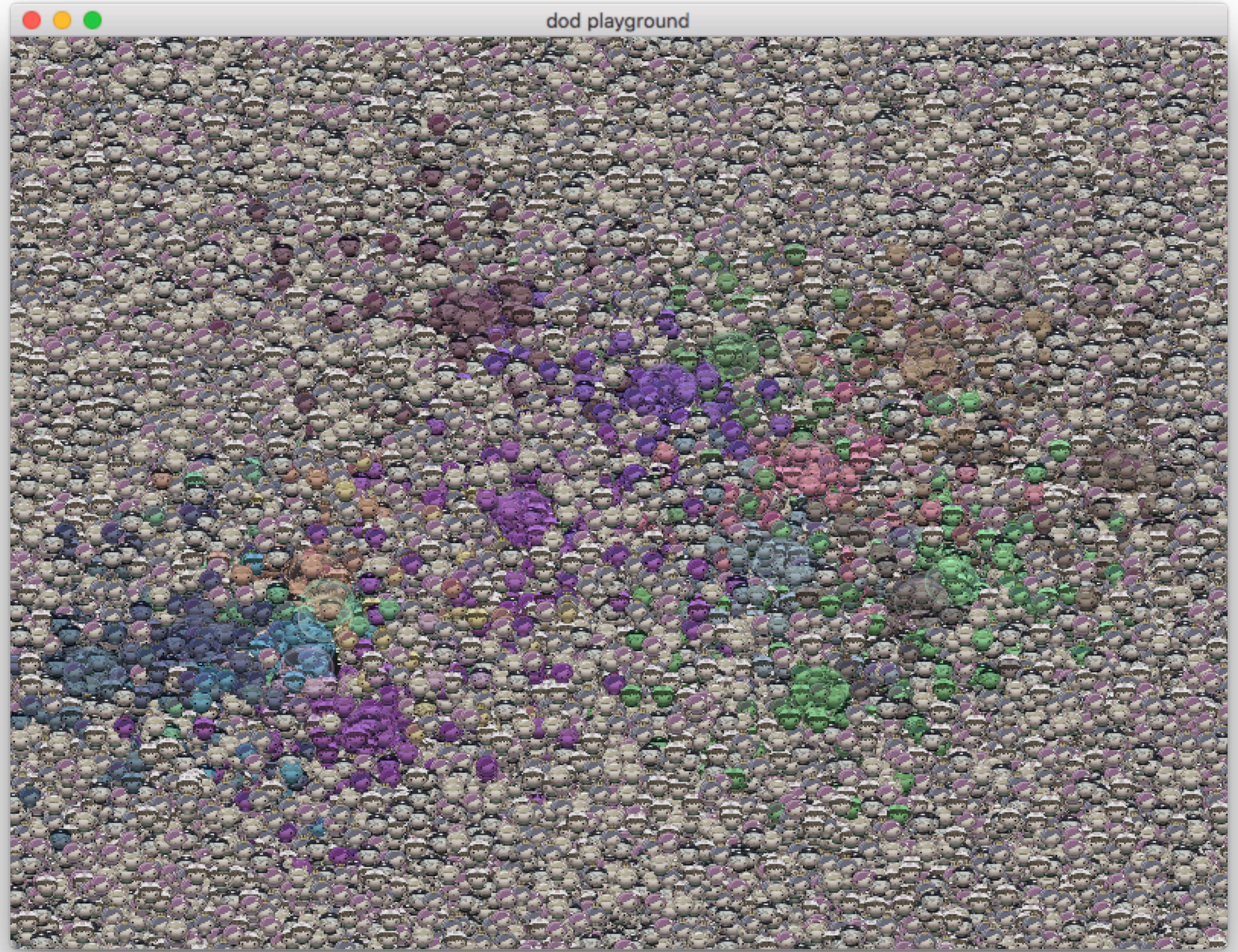
- **400 lines** of code
- 1 million sprites, 20 bubbles:
  - **330ms** update time
  - **470ms** startup time
  - **310MB** memory usage



Sprites from Dan Cook's SpaceCute prototyping challenge,
http://www.lostgarden.com/2007/03/spacecute-prototyping-challenge.html

Recall our simple “game”

- **400 lines** of code
- 1 million sprites, 20 bubbles:
  - **330ms** update time
  - **470ms** startup time
  - **310MB** memory usage



Sprites from Dan Cook’s SpaceCute prototyping challenge, http://www.lostgarden.com/2007/03/spacecute-prototyping-challenge.html

Recall our simple “game”

- **400 lines** of code
- 1 million sprites, 20 bubbles:
  - **330ms** update time
  - **470ms** startup time
  - **310MB** memory usage

Sprites from Dan Cook’s SpaceCute prototyping challenge,
http://www.lostgarden.com/2007/03/spacecute-prototyping-challenge.html

First: Fix Stupidities

- GetComponent searches for component in GO *each. and. every. time.*
- We could find them once and store it! *(common opt. in Unity too)*
- 330ms → **309**ms (commit)

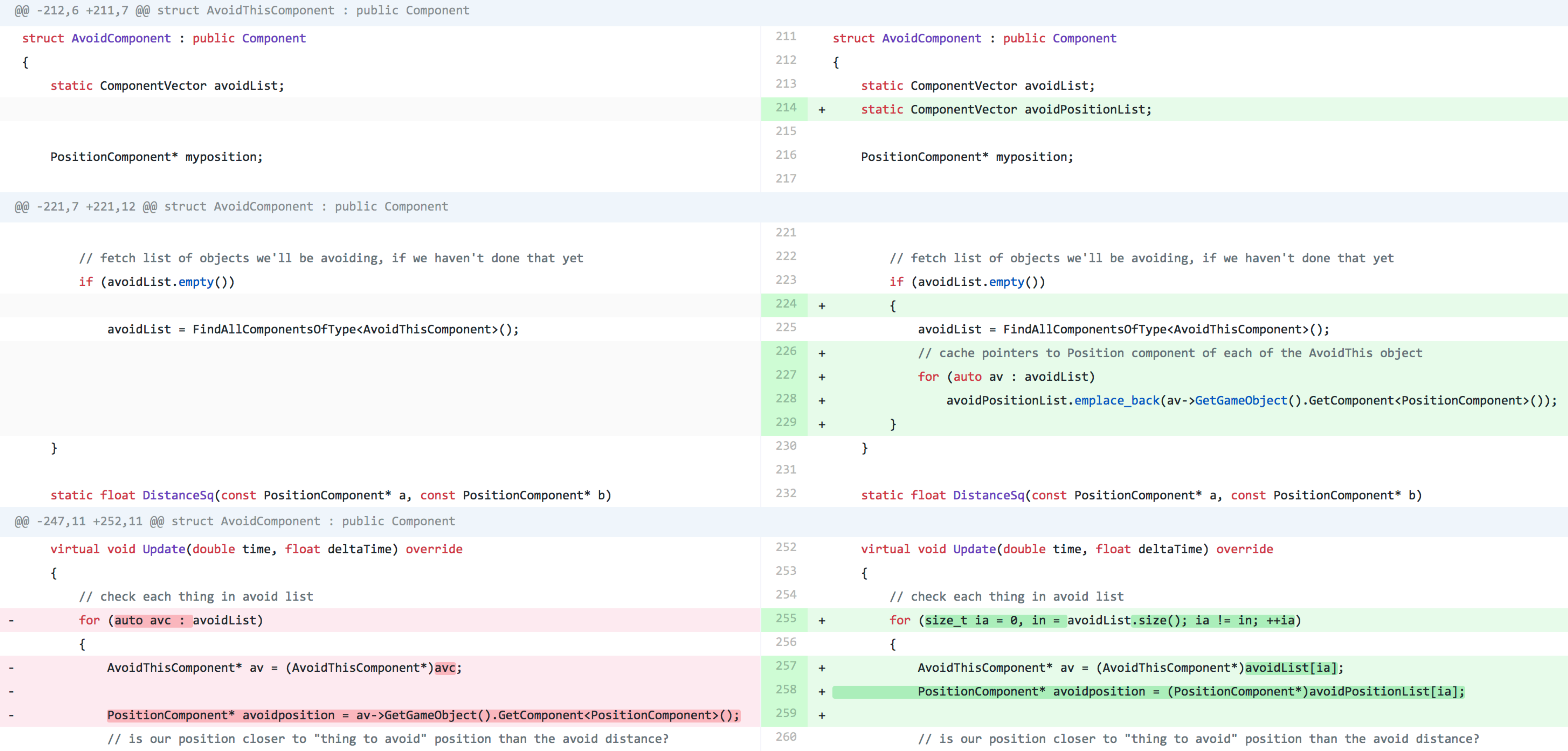
```diff
@@ -148,7 +148,8 @@ struct MoveComponent : public Component
 {
     float velx, vely;
     WorldBoundsComponent* bounds;
-
+    PositionComponent* pos;
+

     MoveComponent(float minSpeed, float maxSpeed)
     {
         // random angle
@@ -163,13 +164,12 @@ struct MoveComponent : public Component
     virtual void Start() override
     {
         bounds = FindOfType<WorldBoundsComponent>();
+        // get Position component on our game object
+        pos = GetGameObject().GetComponent<PositionComponent>();
     }

     virtual void Update(double time, float deltaTime) override
     {
-        // get Position component on our game object
-        PositionComponent* pos = GetGameObject().GetComponent<PositionComponent>();
-
         // update position based on movement velocity & delta time
```

First: Fix Stupidities, take 2

- GetComponent inside inner loop of Avoid component, cache that too.
- 309ms → **78**ms! (commit)



```
@@ -212,6 +211,7 @@ struct AvoidThisComponent : public Component
```

```
 struct AvoidComponent : public Component
 {
     static ComponentVector avoidList;

     PositionComponent* myposition;
```

```
 struct AvoidComponent : public Component
 {
     static ComponentVector avoidList;
+    static ComponentVector avoidPositionList;

     PositionComponent* myposition;

```

```
@@ -221,7 +221,12 @@ struct AvoidComponent : public Component
```

```

         // fetch list of objects we'll be avoiding, if we haven't done that yet
         if (avoidList.empty())

             avoidList = FindAllComponentsOfType<AvoidThisComponent>();




     }

     static float DistanceSq(const PositionComponent* a, const PositionComponent* b)
```

```

         // fetch list of objects we'll be avoiding, if we haven't done that yet
         if (avoidList.empty())
+        {
             avoidList = FindAllComponentsOfType<AvoidThisComponent>();
+            // cache pointers to Position component of each of the AvoidThis object
+            for (auto av : avoidList)
+                avoidPositionList.emplace_back(av->GetGameObject().GetComponent<PositionComponent>());
+        }
     }

     static float DistanceSq(const PositionComponent* a, const PositionComponent* b)
```

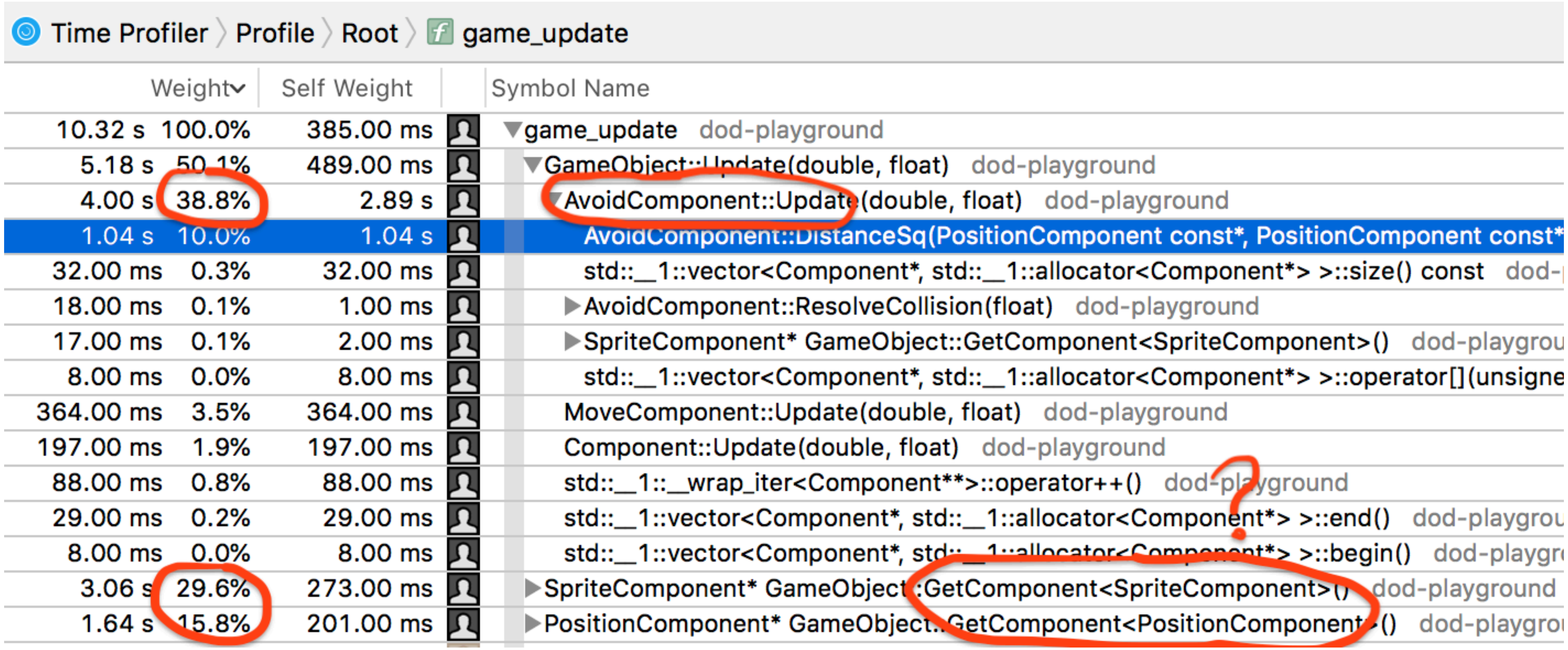
```
@@ -247,11 +252,11 @@ struct AvoidComponent : public Component
```

```
     virtual void Update(double time, float deltaTime) override
     {
         // check each thing in avoid list
-        for (auto avc : avoidList)
         {
-            AvoidThisComponent* av = (AvoidThisComponent*)avc;
-
-            PositionComponent* avoidposition = av->GetGameObject().GetComponent<PositionComponent>();
             // is our position closer to "thing to avoid" position than the avoid distance?
```

```
     virtual void Update(double time, float deltaTime) override
     {
         // check each thing in avoid list
+        for (size_t ia = 0, in = avoidList.size(); ia != in; ++ia)
         {
+            AvoidThisComponent* av = (AvoidThisComponent*)avoidList[ia];
+            PositionComponent* avoidposition = (PositionComponent*)avoidPositionList[ia];
+
             // is our position closer to "thing to avoid" position than the avoid distance?
```

Where time is spent now?

- Let’s use a Profiler.
- I’m on Mac, so Xcode Instruments.



Time Profiler 〉 Profile 〉 Root 〉 game_update

Weight		Self Weight	Symbol Name
10.32 s	100.0%	385.00 ms	▼game_update dod-playground
5.18 s	50.1%	489.00 ms	▼GameObject::Update(double, float) dod-playground
4.00 s	38.8%	2.89 s	▼AvoidComponent::Update(double, float) dod-playground
1.04 s	10.0%	1.04 s	AvoidComponent::DistanceSq(PositionComponent const*, PositionComponent const*
32.00 ms	0.3%	32.00 ms	std::__1::vector<Component*, std::__1::allocator<Component*> >::size() const dod-
18.00 ms	0.1%	1.00 ms	▶AvoidComponent::ResolveCollision(float) dod-playground
17.00 ms	0.1%	2.00 ms	▶SpriteComponent* GameObject::GetComponent<SpriteComponent>() dod-playgrou
8.00 ms	0.0%	8.00 ms	std::__1::vector<Component*, std::__1::allocator<Component*> >::operator[](unsigne
364.00 ms	3.5%	364.00 ms	MoveComponent::Update(double, float) dod-playground
197.00 ms	1.9%	197.00 ms	Component::Update(double, float) dod-playground
88.00 ms	0.8%	88.00 ms	std::__1::__wrap_iter<Component**>::operator++() dod-playground
29.00 ms	0.2%	29.00 ms	std::__1::vector<Component*, std::__1::allocator<Component*> >::end() dod-playgrou
8.00 ms	0.0%	8.00 ms	std::__1::vector<Component*, std::__1::allocator<Component*> >::begin() dod-playgr
3.06 s	29.6%	273.00 ms	▶SpriteComponent* GameObject::GetComponent<SpriteComponent>() dod-playground
1.64 s	15.8%	201.00 ms	▶PositionComponent* GameObject::GetComponent<PositionComponent>() dod-playgro

?

Let’s make some Systems: AvoidanceSystem

- Avoid & AvoidThis components are almost only data now,
- System knows all things it will operate on

```
// When present, tells things that have Avoid component to avoid this object
struct AvoidThisComponent : public Component
{
    float distance;
};

// Objects with this component "avoid" objects with AvoidThis component.
struct AvoidComponent : public Component
{
    virtual void Start() override;
};

// "Avoidance system" works out interactions between objects that have AvoidThis and Avoid
// components. Objects with Avoid component:
// - when they get closer to AvoidThis than AvoidThis::distance, they bounce back,
// - also they take sprite color from the object they just bumped into
struct AvoidanceSystem
{
    // things to be avoided: distances to them, and their position components
    std::vector<float> avoidDistanceList;
    std::vector<PositionComponent*> avoidPositionList;

    // objects that avoid: their position components
    std::vector<PositionComponent*> objectList;
    // …
```

Let’s make some Systems: AvoidanceSystem

- Here’s the logic code of the system
- 78ms → **69**ms (commit)

```
void UpdateSystem(double time, float deltaTime)
{
    // go through all the objects
    for (size_t io = 0, no = objectList.size(); io != no; ++io)
    {
        PositionComponent* myposition = objectList[io];

        // check each thing in avoid list
        for (size_t ia = 0, na = avoidPositionList.size(); ia != na; ++ia)
        {
            float avDistance = avoidDistanceList[ia];
            PositionComponent* avoidposition = avoidPositionList[ia];

            // is our position closer to "thing to avoid" position than the avoid distance?
            if (DistanceSq(myposition, avoidposition) < avDistance * avDistance)
            {
                /* … */
            }
        }
    }
}
```

Let’s make some Systems: MoveSystem

- Similar, let’s make a MoveSystem

```
// Move around with constant velocity. When reached world bounds, reflect back from them.
struct MoveComponent : public Component
{
    float velx, vely;
};

struct MoveSystem
{
    WorldBoundsComponent* bounds;
    std::vector<PositionComponent*> positionList;
    std::vector<MoveComponent*> moveList;
    /* … */
```

Let’s make some Systems: MoveSystem

- Here’s the logic of the MoveSystem
- 69ms → **83**ms (commit).
- **What?!**

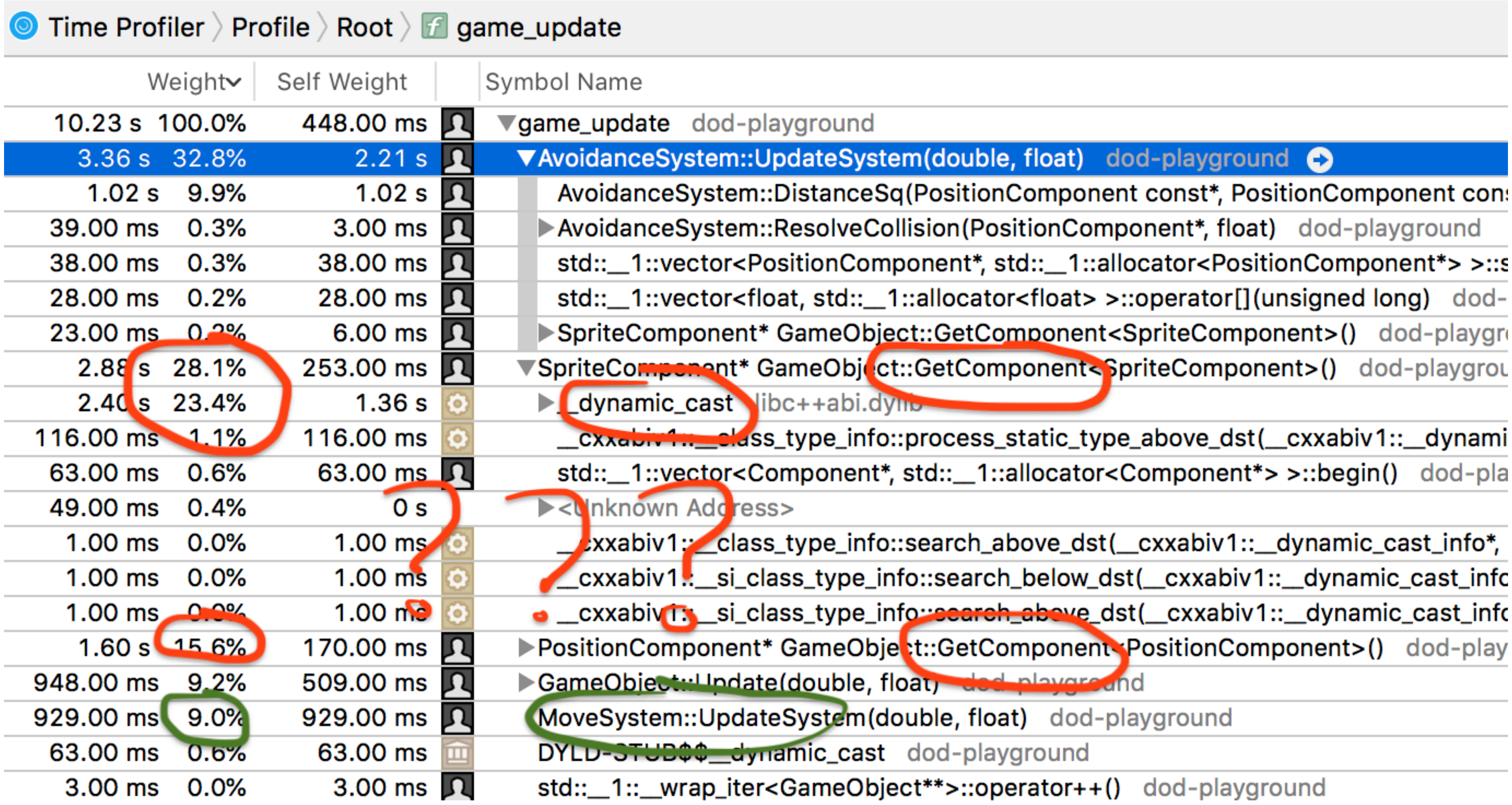
```
void UpdateSystem(double time, float deltaTime)
{
    // go through all the objects
    for (size_t io = 0, no = positionList.size(); io != no; ++io)
    {
        PositionComponent* pos = positionList[io];
        MoveComponent* move = moveList[io];

        // update position based on movement velocity & delta time
        pos->x += move->velx * deltaTime;
        pos->y += move->vely * deltaTime;

        // check against world bounds; put back onto bounds and mirror the velocity component to "bounce" back
        if (pos->x < bounds->xMin) { move->velx = -move->velx; pos->x = bounds->xMin; }
        if (pos->x > bounds->xMax) { move->velx = -move->velx; pos->x = bounds->xMax; }
        if (pos->y < bounds->yMin) { move->vely = -move->vely; pos->y = bounds->yMin; }
        if (pos->y > bounds->yMax) { move->vely = -move->vely; pos->y = bounds->yMax; }
    }
}
```

Ok what is going on?

- Profiler again:



Time Profiler 〉 Profile 〉 Root 〉 game_update

Weight		Self Weight	Symbol Name
10.23 s	100.0%	448.00 ms	▼game_update dod-playground
3.36 s	32.8%	2.21 s	▼AvoidanceSystem::UpdateSystem(double, float) dod-playground
1.02 s	9.9%	1.02 s	AvoidanceSystem::DistanceSq(PositionComponent const*, PositionComponent con
39.00 ms	0.3%	3.00 ms	▶AvoidanceSystem::ResolveCollision(PositionComponent*, float) dod-playground
38.00 ms	0.3%	38.00 ms	std::__1::vector<PositionComponent*, std::__1::allocator<PositionComponent*> >::s
28.00 ms	0.2%	28.00 ms	std::__1::vector<float, std::__1::allocator<float> >::operator[](unsigned long) dod-
23.00 ms	0.2%	6.00 ms	▶SpriteComponent* GameObject::GetComponent<SpriteComponent>() dod-playgr
2.88 s	28.1%	253.00 ms	▼SpriteComponent* GameObject::GetComponent<SpriteComponent>() dod-playgrou
2.40 s	23.4%	1.36 s	▶dynamic_cast libc++abi.dylib
116.00 ms	1.1%	116.00 ms	__cxxabiv1::__class_type_info::process_static_type_above_dst(__cxxabiv1::__dynami
63.00 ms	0.6%	63.00 ms	std::__1::vector<Component*, std::__1::allocator<Component*> >::begin() dod-pla
49.00 ms	0.4%	0 s	▶<Unknown Address>
1.00 ms	0.0%	1.00 ms	__cxxabiv1::__class_type_info::search_above_dst(__cxxabiv1::__dynamic_cast_info*,
1.00 ms	0.0%	1.00 ms	__cxxabiv1::__si_class_type_info::search_below_dst(__cxxabiv1::__dynamic_cast_info
1.00 ms	0.0%	1.00 ms	__cxxabiv1::__si_class_type_info::search_above_dst(__cxxabiv1::__dynamic_cast_info
1.60 s	15.6%	170.00 ms	▶PositionComponent* GameObject::GetComponent<PositionComponent>() dod-play
948.00 ms	9.2%	509.00 ms	▶GameObject::Update(double, float) dod-playground
929.00 ms	9.0%	929.00 ms	MoveSystem::UpdateSystem(double, float) dod-playground
63.00 ms	0.6%	63.00 ms	DYLD-STUB$$dynamic_cast dod-playground
3.00 ms	0.0%	3.00 ms	std::__1::__wrap_iter<GameObject**>::operator++() dod-playground

?
?

Lessons so far

- Optimizing one place can make things slower for unexpected reasons.
  - Out-of-order CPUs, caches, prefetching, … maybe? I did not dig in here :/
- C++ RTTI (`dynamic_cast`) can be *really slow*.
  - We use it in `GameObject::GetComponent`.

```
// get a component of type T, or null if it does not exist on this game object
template<typename T>
T* GetComponent()
{
    for (auto i : m_Components) { T* c = dynamic_cast<T*>(i); if (c != nullptr) return c; }
    return nullptr;
}
```

Let’s stop using C++ RTTI then

- If we had a “Type” enum, and each Component stored the Type…
- 83ms → **54**ms (commit), yay.

```
enum ComponentType
{
    kCompPosition,
    kCompSprite,
    kCompWorldBounds,
    kCompMove,
    kCompAvoid,
    kCompAvoidThis,
};
// ...
ComponentType m_Type;

// was: T* c = dynamic_cast<T*>(i); if (c != nullptr) return c;
if (c->GetType() == T::kTypeId) return (T*)c;
```

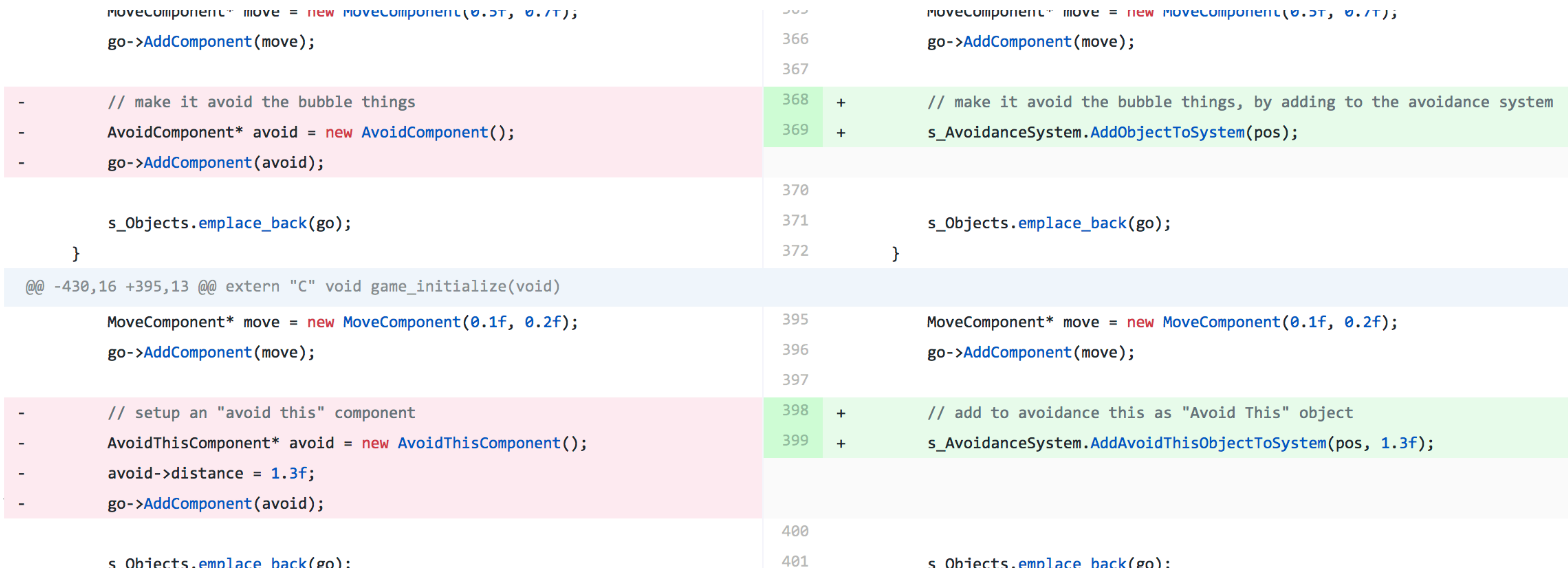
So far:

- Update performance: **6x faster** (330ms→54ms), yay!
- Memory usage: **increased** 310MB→363MB
  - Component pointer caches, type IDs in each component, …
- Lines of code: **more** 400→500

- Let’s try to remove some things!

Avoid & AvoidThis Components, who needs them?

- That’s right. No one!
- Just register objects directly with AvoidanceSystem.
- 54ms → **46**ms, 363MB→325MB, 500→455lines ([commit](#))



```
        MoveComponent* move = new MoveComponent(0.5f, 0.7f);
        go->AddComponent(move);

-       // make it avoid the bubble things
-       AvoidComponent* avoid = new AvoidComponent();
-       go->AddComponent(avoid);

        s_Objects.emplace_back(go);
    }
@@ -430,16 +395,13 @@ extern "C" void game_initialize(void)
        MoveComponent* move = new MoveComponent(0.1f, 0.2f);
        go->AddComponent(move);

-       // setup an "avoid this" component
-       AvoidThisComponent* avoid = new AvoidThisComponent();
-       avoid->distance = 1.3f;
-       go->AddComponent(avoid);

        s_Objects.emplace_back(go);
```

```
        MoveComponent* move = new MoveComponent(0.5f, 0.7f);
        go->AddComponent(move);

+       // make it avoid the bubble things, by adding to the avoidance system
+       s_AvoidanceSystem.AddObjectToSystem(pos);

        s_Objects.emplace_back(go);
    }

        MoveComponent* move = new MoveComponent(0.1f, 0.2f);
        go->AddComponent(move);

+       // add to avoidance this as "Avoid This" object
+       s_AvoidanceSystem.AddAvoidThisObjectToSystem(pos, 1.3f);

        s_Objects.emplace_back(go);
```

Actually, who needs Component hierarchy?

- Just have component fields in GameObject
- 46ms→43ms update, 398→**112ms** startup, 325MB→**218MB**, 455→**350**lines (commit)

```
// each object has data for all possible components,
// as well as flags indicating which ones are actually present.
struct GameObject
{
    GameObject(const std::string&& name)
      : m_Name(name), m_HasPosition(0), m_HasSprite(0), m_HasWorldBounds(0), m_HasMove(0) { }
    ~GameObject() {}

    std::string m_Name;
    // data for all components
    PositionComponent m_Position;
    SpriteComponent m_Sprite;
    WorldBoundsComponent m_WorldBounds;
    MoveComponent m_Move;
    // flags for every component, indicating whether this object "has it"
    int m_HasPosition : 1;
    int m_HasSprite : 1;
    int m_HasWorldBounds : 1;
    int m_HasMove : 1;
};
```

Stop allocating individual GameObjects

- `vector<GameObject*>` → `vector<GameObject>`
- 43ms update, 112→**99ms** startup, 218MB→**203MB** (commit)

```
@@ -84,7 +84,7 @@ struct GameObject
```

```
  // The "scene": array of game objects.
  // "ID" of a game object is just an index into the scene array.
  typedef size_t EntityID;
- typedef std::vector<GameObject*> GameObjectVector;
  static GameObjectVector s_Objects;
```

```
  // The "scene": array of game objects.
  // "ID" of a game object is just an index into the scene array.
  typedef size_t EntityID;
+ typedef std::vector<GameObject> GameObjectVector;
  static GameObjectVector s_Objects;
```

```
@@ -109,13 +109,13 @@ struct MoveSystem
```

```

      void UpdateSystem(double time, float deltaTime)
      {
-         const WorldBoundsComponent* bounds = &s_Objects[boundsID]->m_WorldBounds;

          // go through all the objects
          for (size_t io = 0, no = entities.size(); io != no; ++io)
          {
-             PositionComponent* pos = &s_Objects[io]->m_Position;
-             MoveComponent* move = &s_Objects[io]->m_Move;

              // update position based on movement velocity & delta time
```

```

      void UpdateSystem(double time, float deltaTime)
      {
+         const WorldBoundsComponent* bounds = &s_Objects[boundsID].m_WorldBounds;

          // go through all the objects
          for (size_t io = 0, no = entities.size(); io != no; ++io)
          {
+             PositionComponent* pos = &s_Objects[io].m_Position;
+             MoveComponent* move = &s_Objects[io].m_Move;

              // update position based on movement velocity & delta time
```

Geez how many intermissions you plan to have here?!

Structure-of-Arrays (SoA) data layout

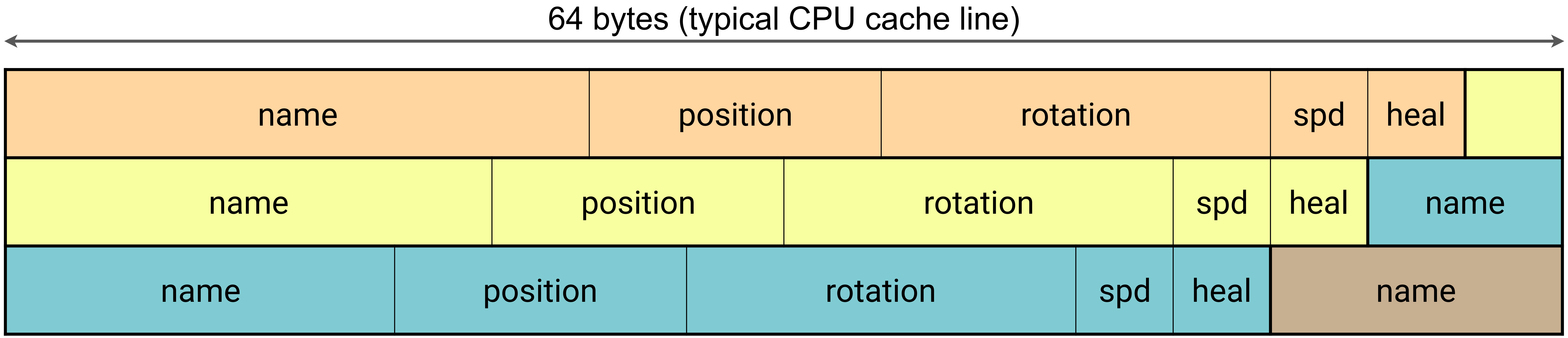
Typical layout: Array-of-Structures (AoS)

- Some objects, and arrays of them.
- Simple to understand and manage.
- Great… ***iff*** we need *all* the data from each object.

```
// structure
struct Object
{
    string name;
    Vector3 position;
    Quaternion rotation;
    float speed;
    float health;
};
// array of structures
vector<Object> allObjects;
```

How does data look like in memory?

```
struct Object                   // 60 bytes:
{
    string name;                // 24 bytes
    Vector3 position;           // 12 bytes
    Quaternion rotation;        // 16 bytes
    float speed;                // 4 bytes
    float health;               // 4 bytes
};
```



64 bytes (typical CPU cache line)

name position rotation spd heal

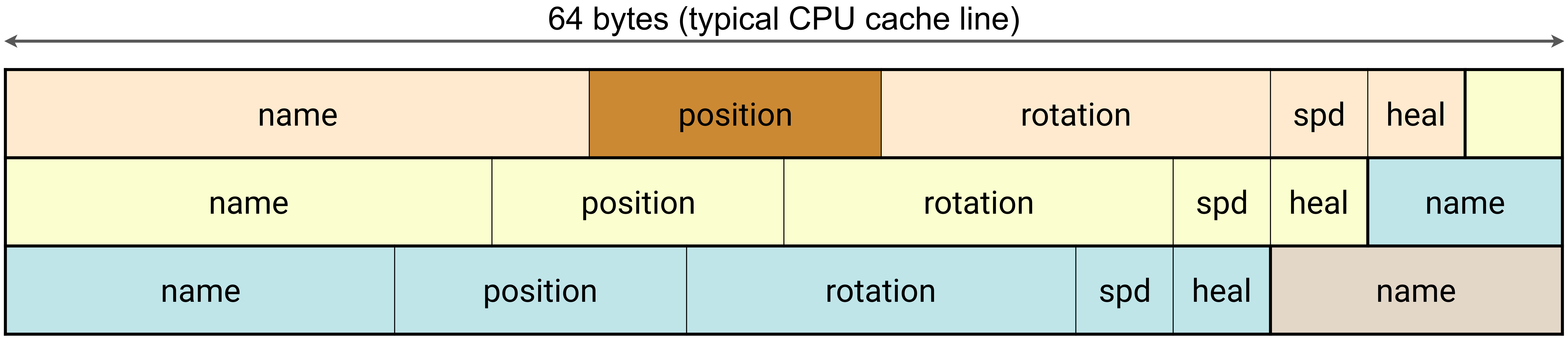
name position rotation spd heal name

name position rotation spd heal name

What if we don’t need all data?

- If we have a system that only needs object `position` & `speed`…

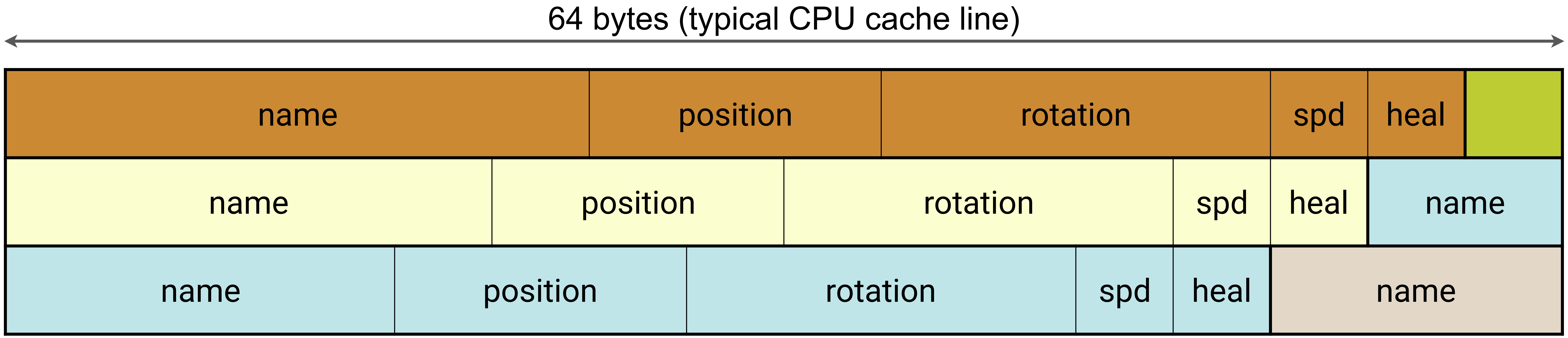
- Hey CPU, read me position of first object!
- Sure, it’s right here…


64 bytes (typical CPU cache line)

What if we don’t need all data?

- If we have a system that only needs object `position` & `speed`…

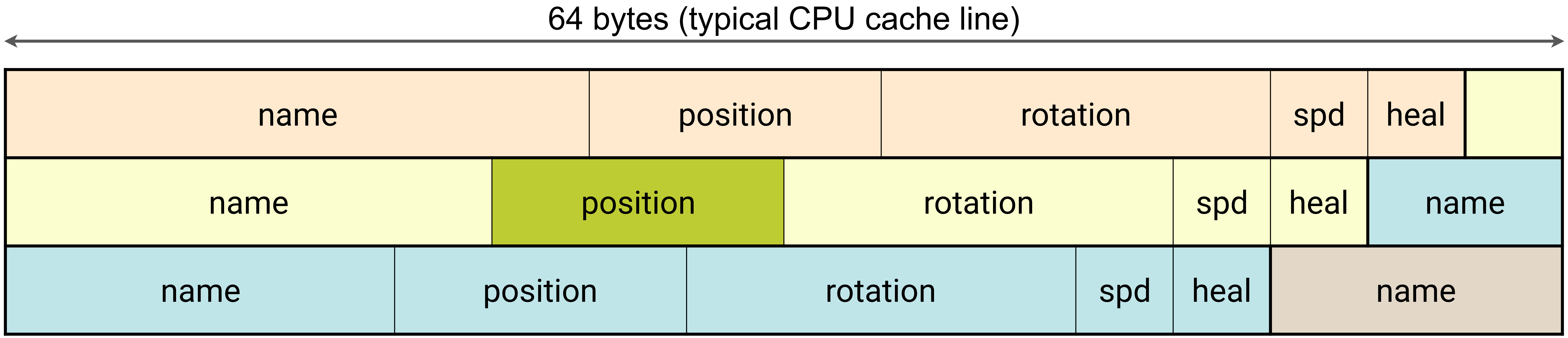
- Hey CPU, read me position of first object!
- Sure, it’s right here… lemme read the whole cache line from memory for you!



What if we don’t need all data?

- If we have a system that only needs object `position` & `speed`…

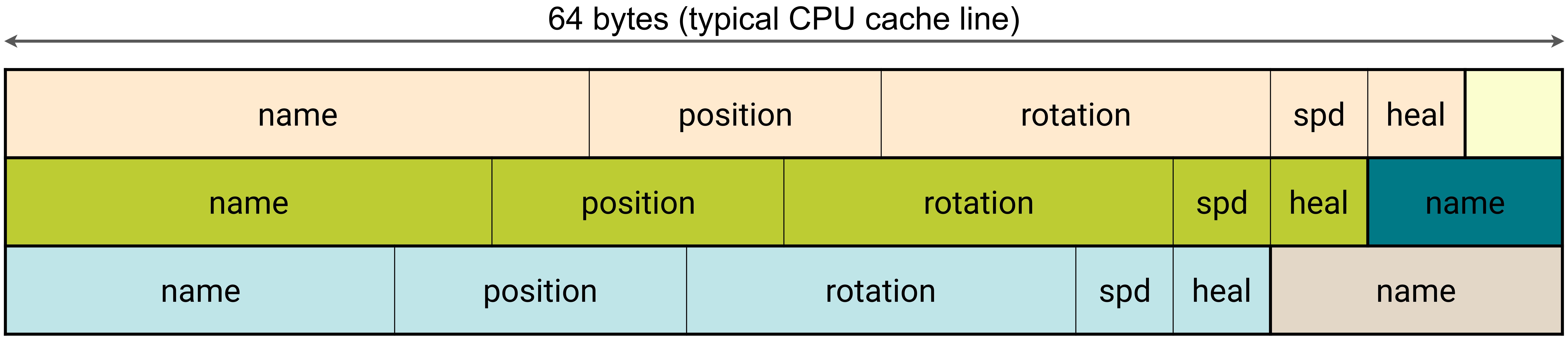
- Uh ok, get me position of second object then
- Will do!



What if we don’t need all data?

- If we have a system that only needs object `position` & `speed`…

- Uh ok, get me position of second object then
- Will do! Here’s the whole cache line for you again!



64 bytes (typical CPU cache line)

name | position | rotation | spd | heal

name | position | rotation | spd | heal | name

name | position | rotation | spd | heal | name

What if we don’t need all data?

- If we have a system that only needs object `position` & `speed`…
- We end up reading **everything** from memory,
- But we only needed **16 bytes** out of **60** in every object.
- **74%** of all memory traffic we *did not even need*!

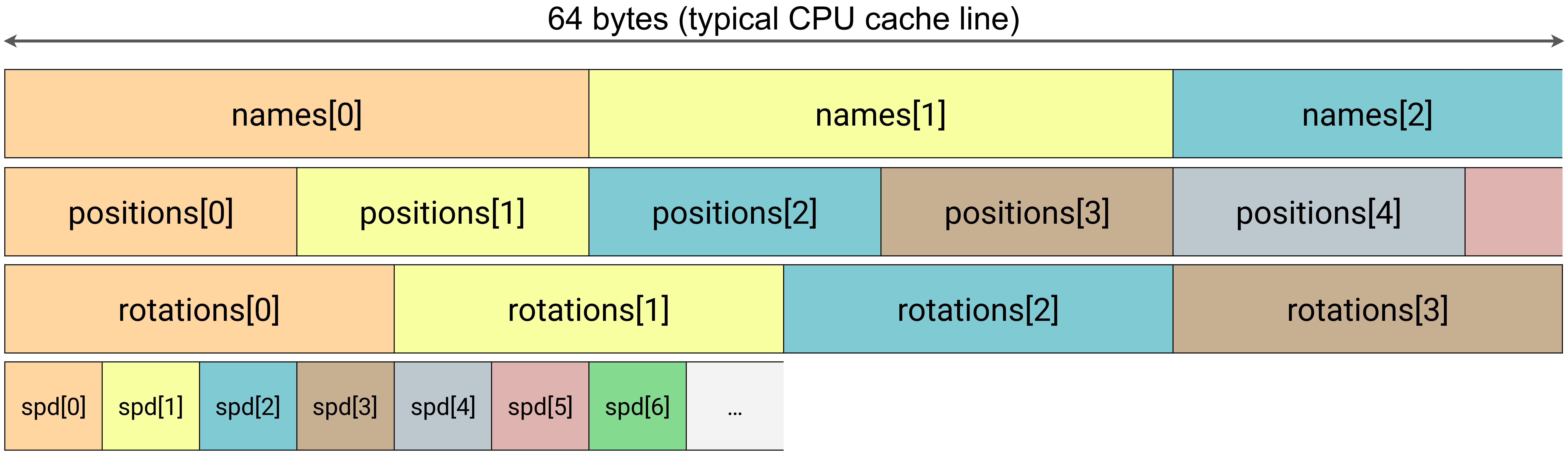
Flip it: Structure-of-Arrays (SoA)

- Separate arrays for each data member.
- Arrays need to be kept in sync.
- “The object” no longer exists; data accessed through an index.

```
// structure of arrays
struct Objects
{
    vector<string> names;         // 24 bytes each
    vector<Vector3> positions;    // 12 bytes each
    vector<Quaternion> rotations; // 16 bytes each
    vector<float> speeds;         // 4 bytes each
    vector<float> healths;        // 4 bytes each
};
```

How does data look like in memory?

```
struct Objects
{
    vector<string> names;          // 24 bytes each
    vector<Vector3> positions;     // 12 bytes each
    vector<Quaternion> rotations;  // 16 bytes each
    vector<float> speeds;          // 4 bytes each
    vector<float> healths;         // 4 bytes each
};
```



64 bytes (typical CPU cache line)

names[0] names[1] names[2]

positions[0] positions[1] positions[2] positions[3] positions[4]

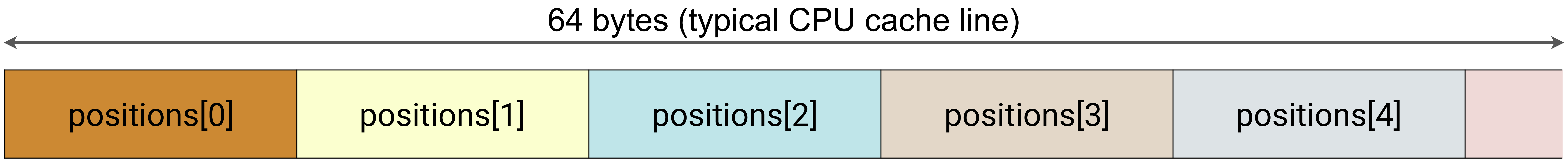
rotations[0] rotations[1] rotations[2] rotations[3]

spd[0] spd[1] spd[2] spd[3] spd[4] spd[5] spd[6] ...

Reading partial data in SoA

- If we have a system that only needs object `position` & `speed`…

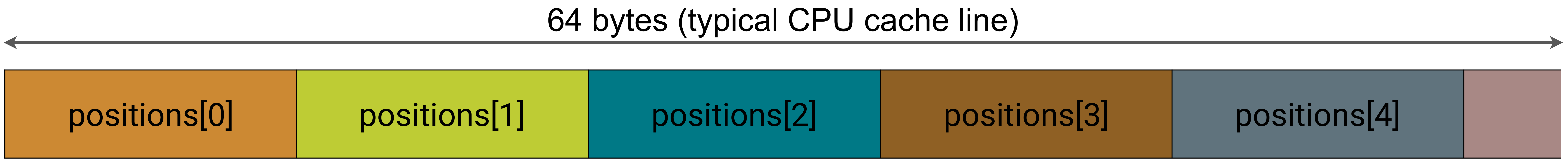
- Hey CPU, read me position of first object!
- Sure, it’s right here…



Reading partial data in SoA

- If we have a system that only needs object `position` & `speed`…

- Hey CPU, read me position of first object!
- Sure, it’s right here… lemme read the whole cache line from memory for you!
- *(narrator) and so positions for next 4 objects got read into CPU cache too*



SoA data layout transformation

- Is fairly common
- Careful to not overdo it though!
  - At some point the # of individual arrays can get counterproductive
  - Structure-of-Arrays-of-Structures (SoAoS), etc. :)

Back to us: SoA layout for component data

- No longer a `GameObject` class, just an `EntityID`
- 43ms→**31ms** update, 99→94ms startup, 350→375 lines (commit)

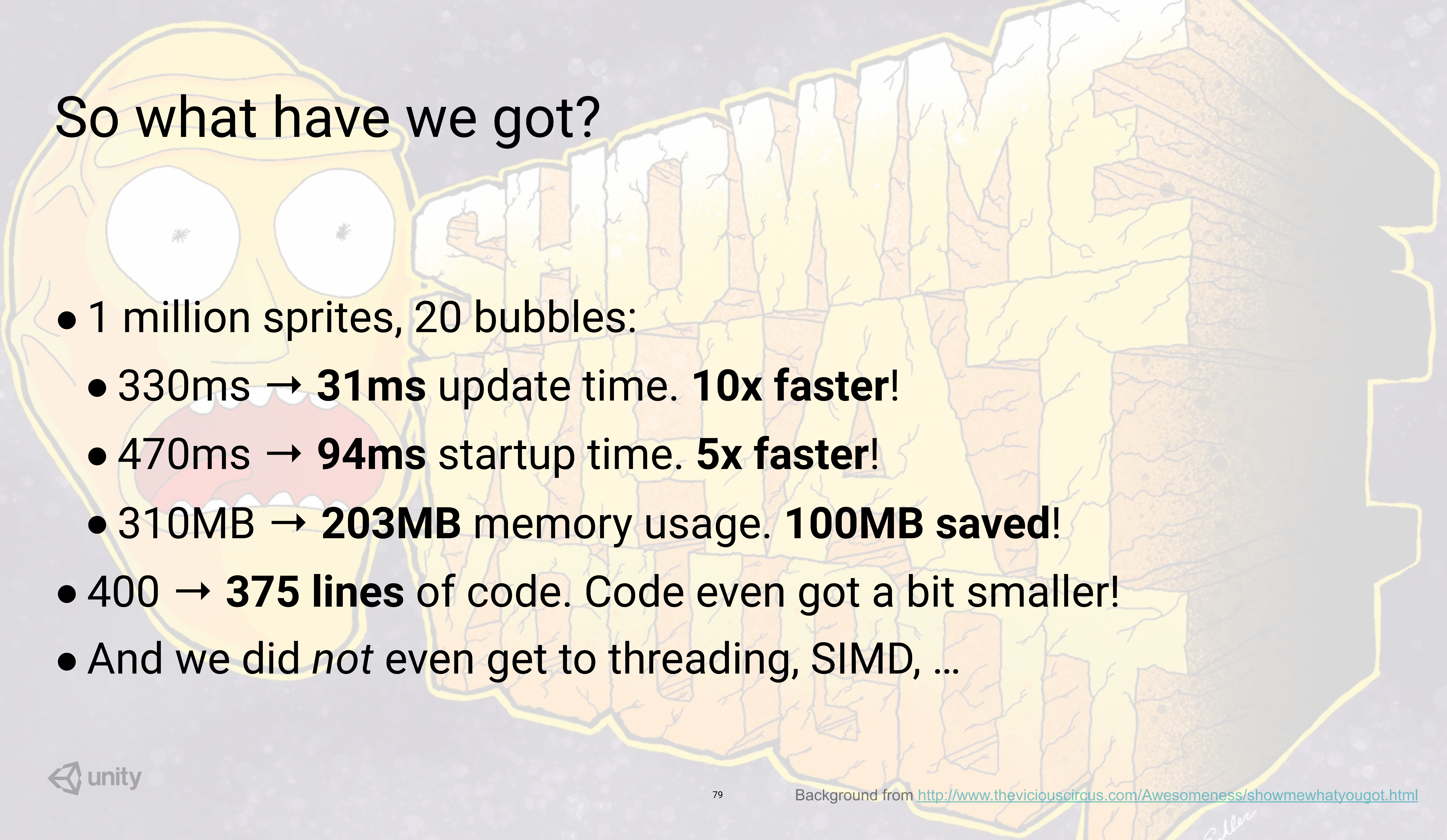
```
// "ID" of a game object is just an index into the scene array.
typedef size_t EntityID;

// /* … */

// names of each object
vector<string> m_Names;
// data for all components
vector<PositionComponent> m_Positions;
vector<SpriteComponent> m_Sprites;
vector<WorldBoundsComponent> m_WorldBounds;
vector<MoveComponent> m_Moves;
// bit flags for every component, indicating whether this object "has it"
vector<int> m_Flags;
```

So what have we got?

- 1 million sprites, 20 bubbles:
    - 330ms → **31ms** update time. **10x faster**!
    - 470ms → **94ms** startup time. **5x faster**!
    - 310MB → **203MB** memory usage. **100MB saved**!
- 400 → **375 lines** of code. Code even got a bit smaller!
- And we did *not* even get to threading, SIMD, …



Background from http://www.theviciouscircus.com/Awesomeness/showmewhatyougot.html

Ask me questions

Question & Homework time!

Zero Days by Scatter — Made with Unity