NOKIA

DESIGNING DATA FOR PERFORMANCE AND MAINTAINABILITY

Laurent Carlier
Nokia
May 13th 2026

AGENDA

- Importance of data structure organization
- Problems with using SoA
- Solution with zero-cost abstractions
- Improvement using C++ 26's reflection
- Take-aways

IMPORTANCE OF DATA STRUCTURE ORGANIZATION

IMPORTANCE OF DATA STRUCTURE ORGANIZATION

- Maintainability

IMPORTANCE OF DATA STRUCTURE ORGANIZATION

- Maintainability
- Scalability

IMPORTANCE OF DATA STRUCTURE ORGANIZATION

- Maintainability
- Scalability
- 🚀🚀🚀 Performance 🚀🚀🚀

OUR EXAMPLE

A PARTICLE SIMULATOR

Goal

OUR EXAMPLE

A PARTICLE SIMULATOR

Goal

- Simulate the movement of particles

OUR EXAMPLE

A PARTICLE SIMULATOR

Goal

- Simulate the movement of particles
- Basic linear movement with velocity

OUR EXAMPLE

A PARTICLE SIMULATOR

Goal

- Simulate the movement of particles
- Basic linear movement with velocity
- Big amount of particles to simulate: 1.000.000.000

OUR EXAMPLE

A PARTICLE SIMULATOR

One particle has the following attribute

OUR EXAMPLE

A PARTICLE SIMULATOR

One particle has the following attribute

- A position (x, y)

OUR EXAMPLE

A PARTICLE SIMULATOR

One particle has the following attribute

- A position (x, y)
- A velocity (dx, dy)

OUR EXAMPLE

A PARTICLE SIMULATOR

One particle has the following attribute

- A position (x, y)
- A velocity (dx, dy)
- Auxiliary attributes (color, lifetime)

A PARTICLE SIMULATOR

THE PARTICLE STRUCT

```
1 struct Particle
2 {
3   float       x, y;
4   float       dx, dy;
5   uint64_t    lifetime;
6   Color       color;
7 };
8
9 std::vector<Particle> particles;
```

A PARTICLE SIMULATOR

THE PARTICLE STRUCT

```
1 struct Particle
2 {
3   float       x, y;
4   float       dx, dy;
5   uint64_t    lifetime;
6   Color       color;
7 };
8
9 std::vector<Particle> particles;
```

This way of organizing the data is known as `Array of structures` aka AoS

A PARTICLE SIMULATOR

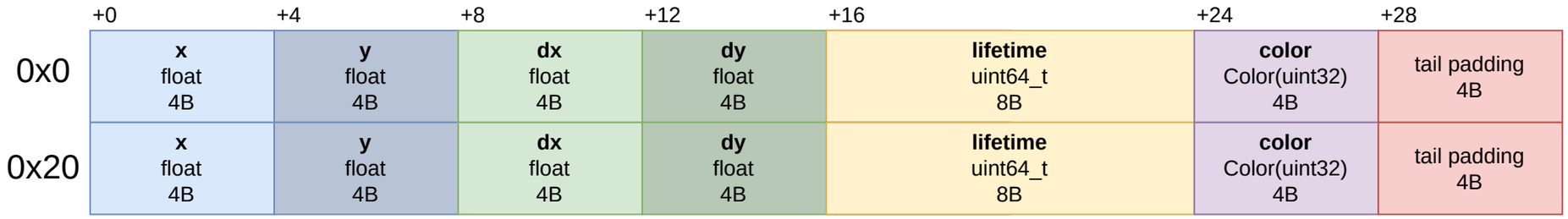
UPDATING THE PARTICLES

```
 1 double UpdateParticles(float dt)
 2 {
 3   auto t0 = Clock::now();
 4   for (auto& p : particles)
 5   {
 6     p.x += p.dx * dt;
 7     p.y += p.dy * dt;
 8
 9     if      (p.x < 0)                p.x += ScreenWidth();
10     else if (p.x > ScreenWidth())    p.x -= ScreenWidth();
11     if      (p.y < 0)                p.y += ScreenHeight();
12     else if (p.y > ScreenHeight())   p.y -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

MEMORY PROFILING

MEMORY PROFILING

MEMORY USAGE

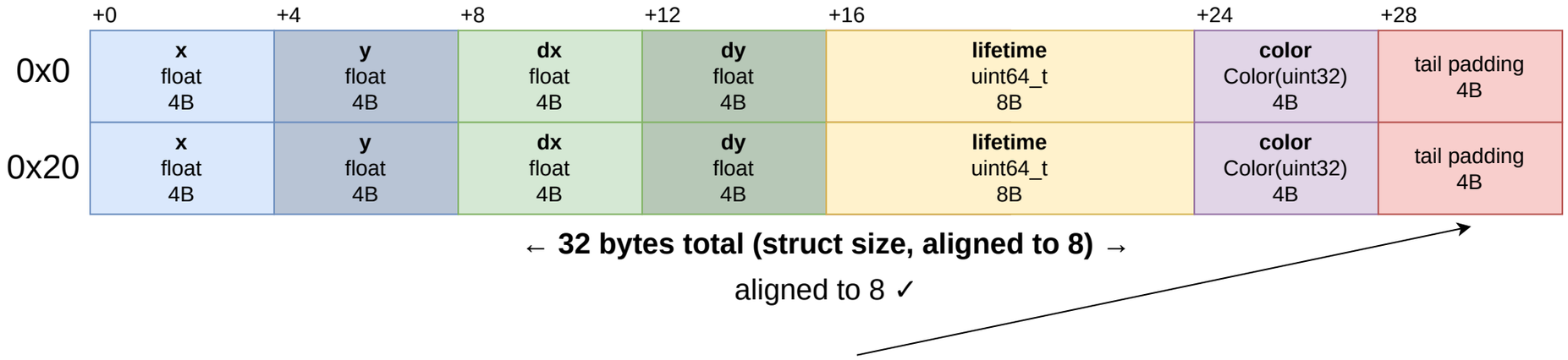


← 32 bytes total (struct size, aligned to 8) →

aligned to 8 ✓

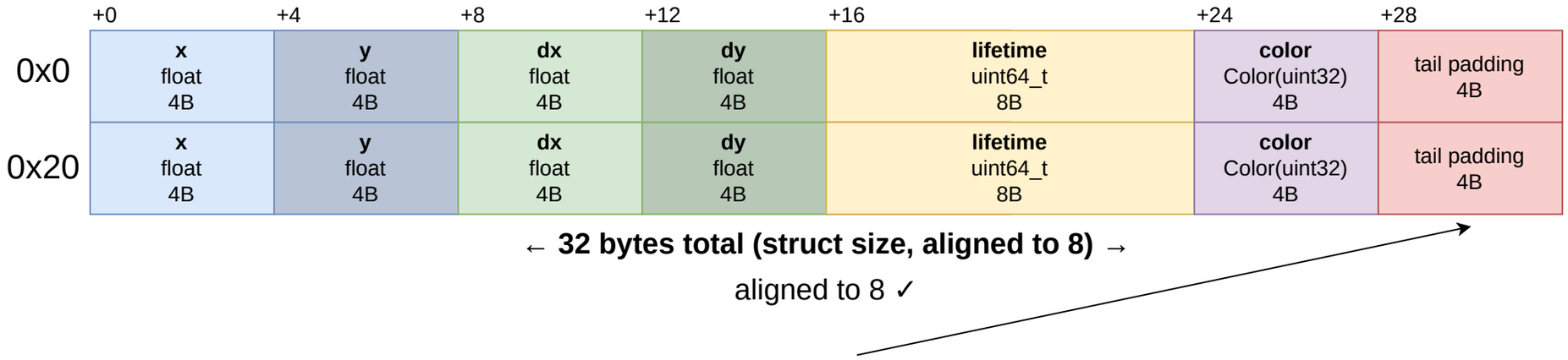
MEMORY PROFILING

MEMORY USAGE



MEMORY PROFILING

MEMORY USAGE



🙀Wasted memory🙀

4 * 1.000.000.000 = 4 GB

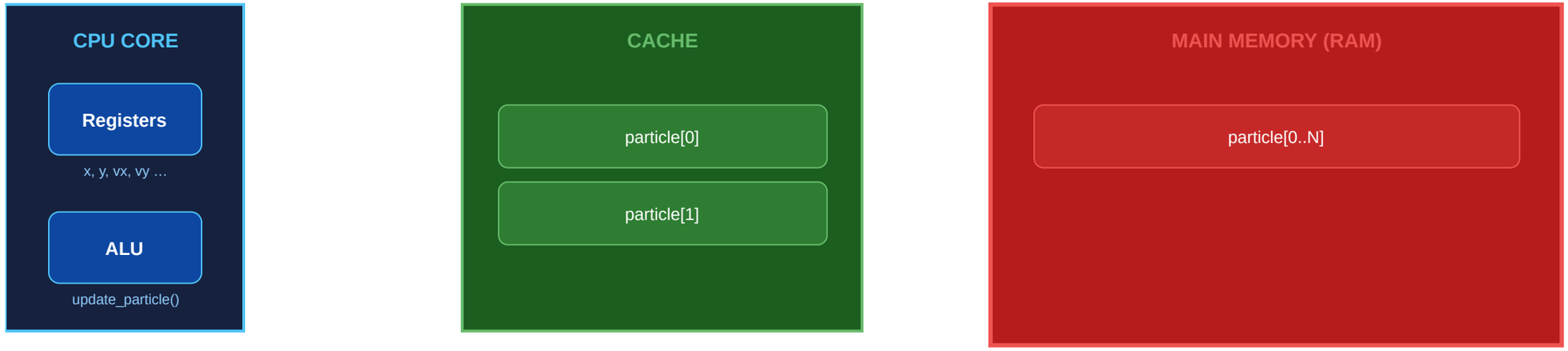
MEMORY PROFILING

CPU CACHE EFFICIENCY

MEMORY PROFILING

CPU CACHE EFFICIENCY

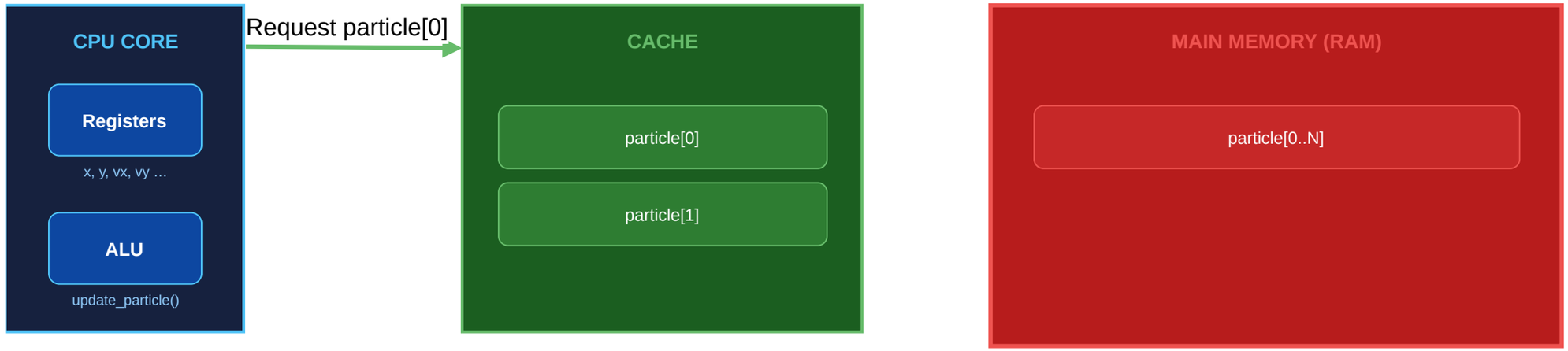
- When the CPU finds the data it needs in cache (a “cache hit”), it avoids costly memory fetches.



MEMORY PROFILING

CPU CACHE EFFICIENCY

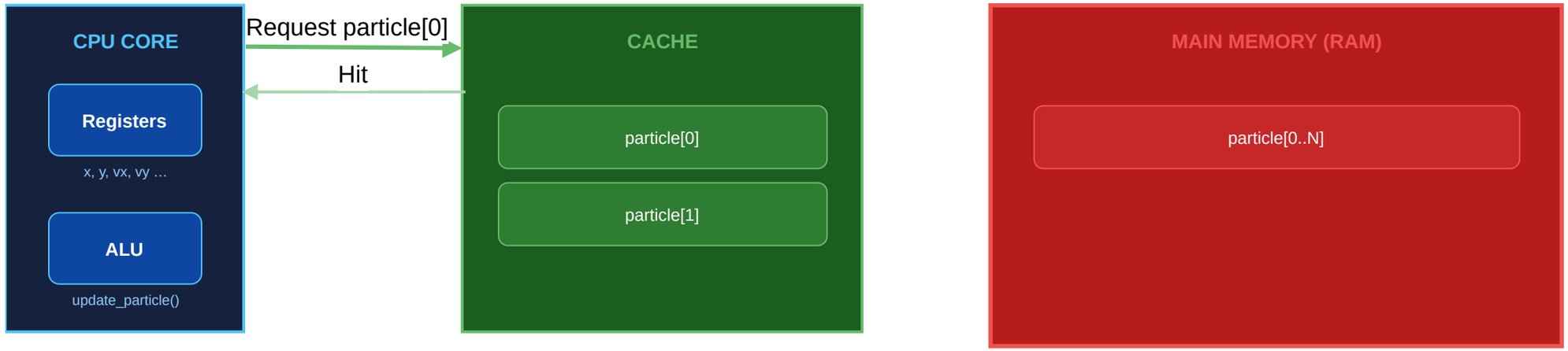
- When the CPU finds the data it needs in cache (a “cache hit”), it avoids costly memory fetches.



MEMORY PROFILING

CPU CACHE EFFICIENCY

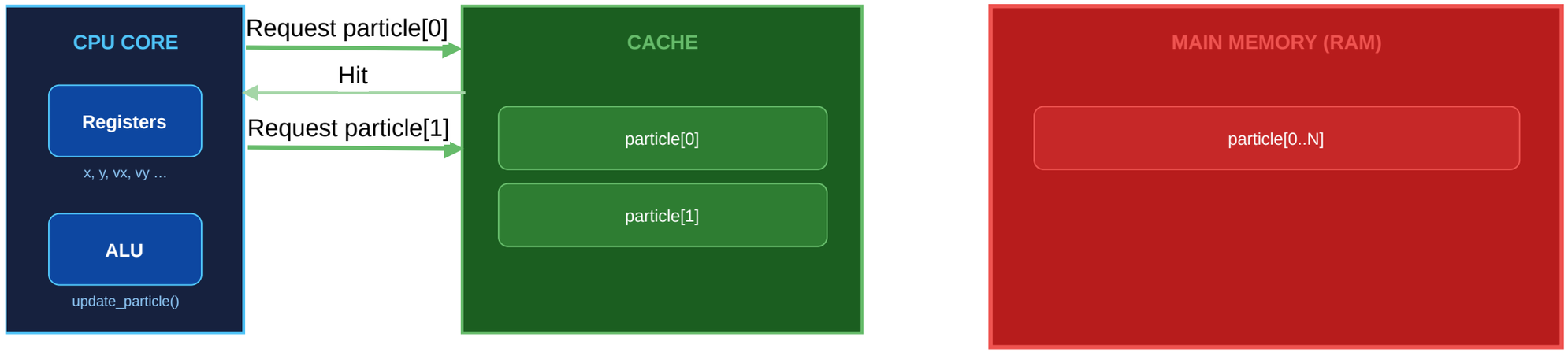
- When the CPU finds the data it needs in cache (a “cache hit”), it avoids costly memory fetches.



MEMORY PROFILING

CPU CACHE EFFICIENCY

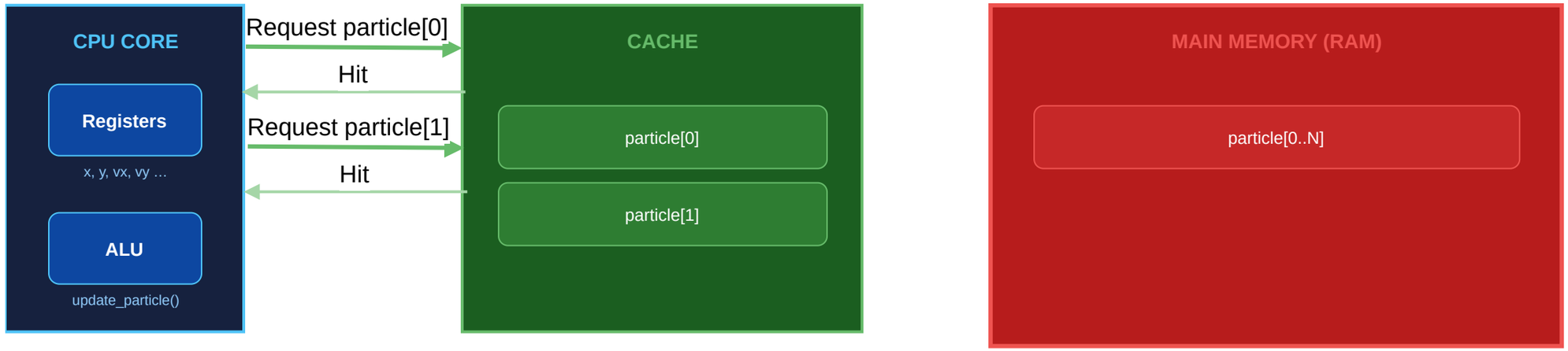
- When the CPU finds the data it needs in cache (a “cache hit”), it avoids costly memory fetches.



MEMORY PROFILING

CPU CACHE EFFICIENCY

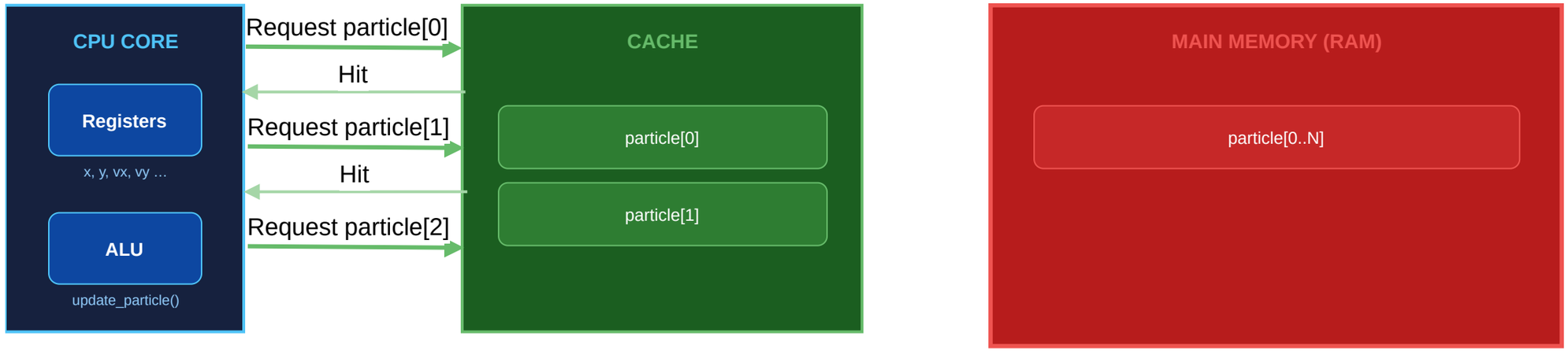
- When the CPU finds the data it needs in cache (a “cache hit”), it avoids costly memory fetches.



MEMORY PROFILING

CPU CACHE EFFICIENCY

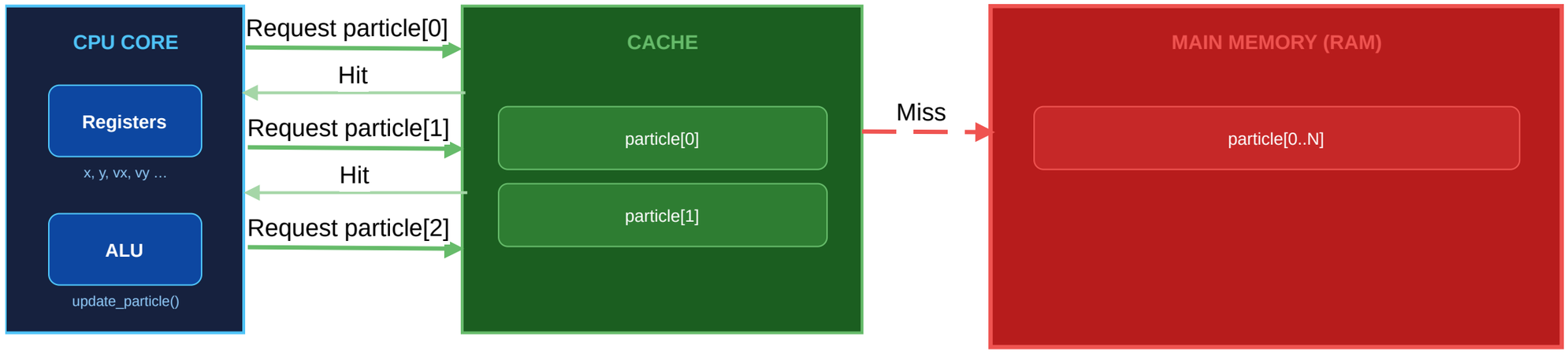
- When the CPU finds the data it needs in cache (a “cache hit”), it avoids costly memory fetches.



MEMORY PROFILING

CPU CACHE EFFICIENCY

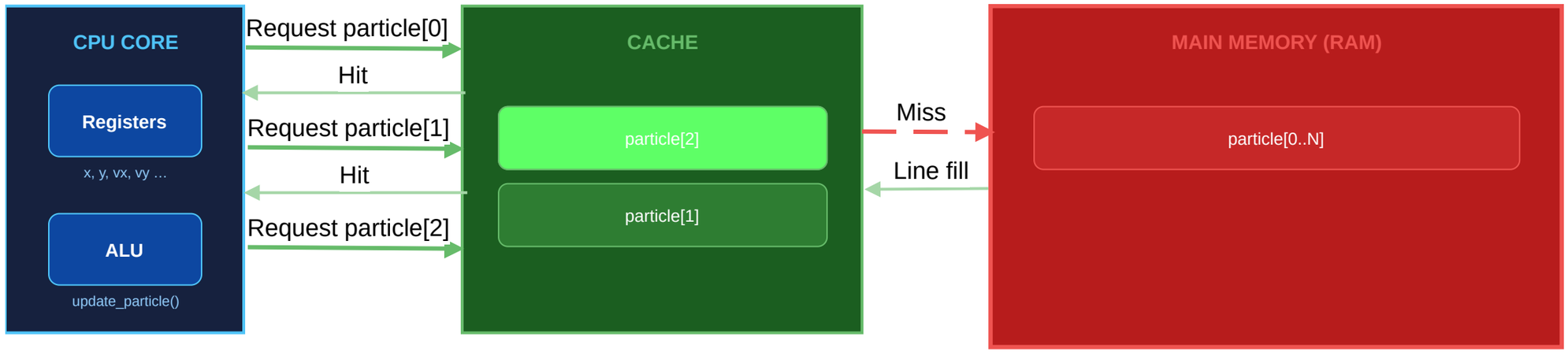
- When the CPU finds the data it needs in cache (a “cache hit”), it avoids costly memory fetches.



MEMORY PROFILING

CPU CACHE EFFICIENCY

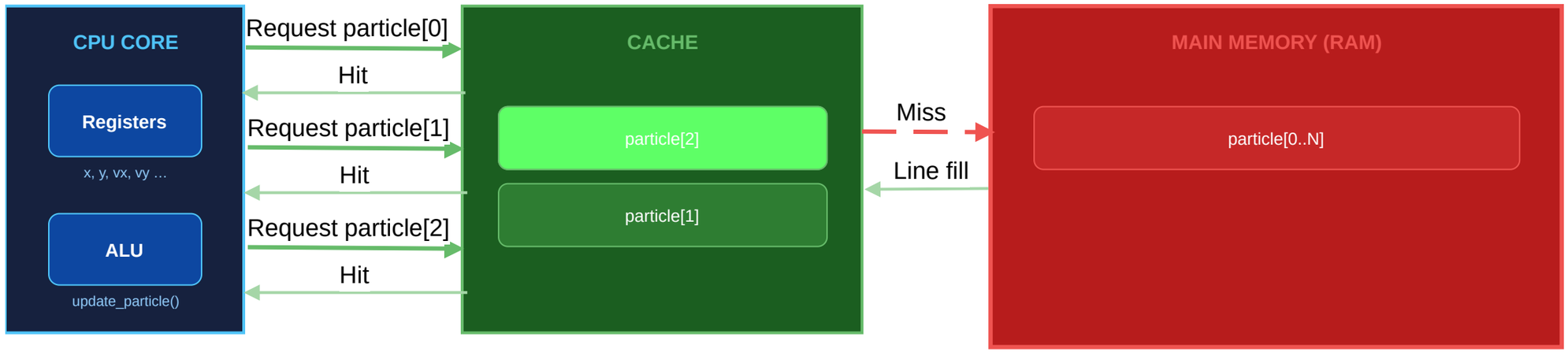
- When the CPU finds the data it needs in cache (a “cache hit”), it avoids costly memory fetches.



MEMORY PROFILING

CPU CACHE EFFICIENCY

- When the CPU finds the data it needs in cache (a “cache hit”), it avoids costly memory fetches.



MEMORY PROFILING

CPU CACHE EFFICIENCY

Typical access time comparison

	CPU cache (L1/L2/L3)	RAM	Disk (SSD)
Access time	~1 ns	~100 ns	~100 us

MEMORY PROFILING

CPU CACHE EFFICIENCY

Typical access time comparison

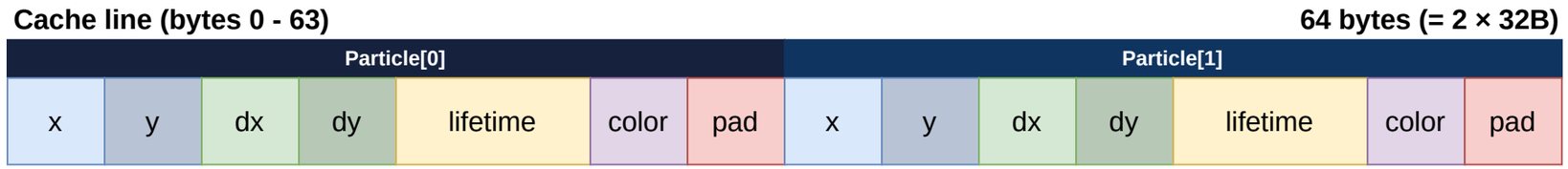
	CPU cache (L1/L2/L3)	RAM	Disk (SSD)
Access time	~1 ns	~100 ns	~100 us

Refreshing accessing the RAM ~100 times slower than then CPU cache

MEMORY PROFILING

CACHE USAGE

The CPU fetches instructions in fixed-size cache lines (typically 64 bytes), loading consecutive data into cache in a single access.



MEMORY PROFILING

CACHE USAGE

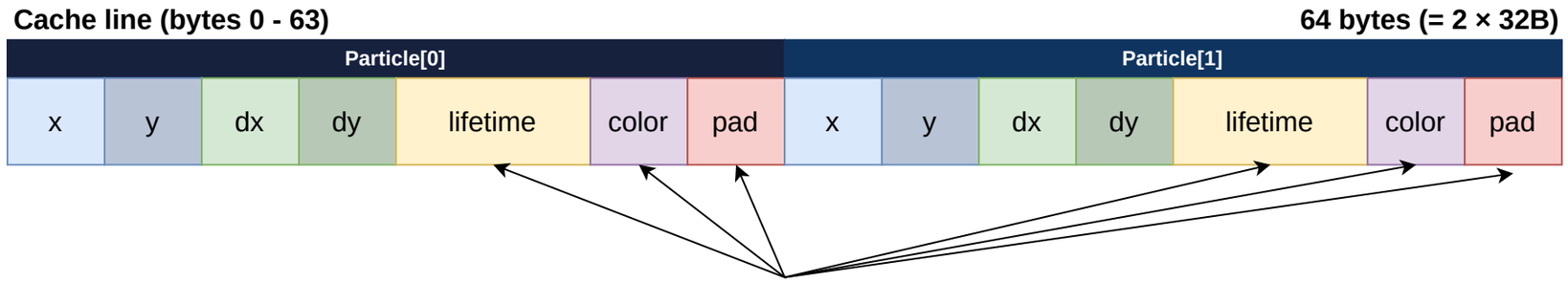
The CPU fetches instructions in fixed-size cache lines (typically 64 bytes), loading consecutive data into cache in a single access.

```
 1 double UpdateParticles(float dt)
 2 {
 3   auto t0 = Clock::now();
 4   for (auto& p : particles)
 5   {
 6     p.x += p.dx * dt;
 7     p.y += p.dy * dt;
 8
 9     if      (p.x < 0)                p.x += ScreenWidth();
10     else if (p.x > ScreenWidth())    p.x -= ScreenWidth();
11     if      (p.y < 0)                p.y += ScreenHeight();
12     else if (p.y > ScreenHeight())   p.y -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

MEMORY PROFILING

CACHE USAGE

The CPU fetches instructions in fixed-size cache lines (typically 64 bytes), loading consecutive data into cache in a single access.



🙀Unnecessary data in cache🙀

HOW DO WE FIX THIS? 

FIXING THE DATA SPACE LOCALIZATION

Data organization (old)

0x0	**x** float 4B	**y** float 4B	**dx** float 4B	**dy** float 4B	**lifetime** uint64_t 8B	**color** Color(uint32) 4B	tail padding 4B
0x20	**x** float 4B	**y** float 4B	**dx** float 4B	**dy** float 4B	**lifetime** uint64_t 8B	**color** Color(uint32) 4B	tail padding 4B
0x40	**x** float 4B	**y** float 4B	**dx** float 4B	**dy** float 4B	**lifetime** uint64_t 8B	**color** Color(uint32) 4B	tail padding 4B
0x60	**x** float 4B	**y** float 4B	**dx** float 4B	**dy** float 4B	**lifetime** uint64_t 8B	**color** Color(uint32) 4B	tail padding 4B
0x80	**x** float 4B	**y** float 4B	**dx** float 4B	**dy** float 4B	**lifetime** uint64_t 8B	**color** Color(uint32) 4B	tail padding 4B
0xa0	**x** float 4B	**y** float 4B	**dx** float 4B	**dy** float 4B	**lifetime** uint64_t 8B	**color** Color(uint32) 4B	tail padding 4B
0xc0	**x** float 4B	**y** float 4B	**dx** float 4B	**dy** float 4B	**lifetime** uint64_t 8B	**color** Color(uint32) 4B	tail padding 4B
0x100	**x** float 4B	**y** float 4B	**dx** float 4B	**dy** float 4B	**lifetime** uint64_t 8B	**color** Color(uint32) 4B	tail padding 4B

FIXING THE DATA SPACE LOCALIZATION

Data organization (new)

0x0	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B
0x20	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B
0x40	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B
0x60	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B
0x80	**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B	
0xa0	**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B	
0xc0	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B

FIXING THE DATA SPACE LOCALIZATION

Data organization (new)

0x0	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B
0x20	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B
0x40	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B
0x60	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B
0x80	**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B	
0xa0	**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B	
0xc0	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B
0x100	😸**No more space wasted for padding**😸							

FIXING THE DATA SPACE LOCALIZATION

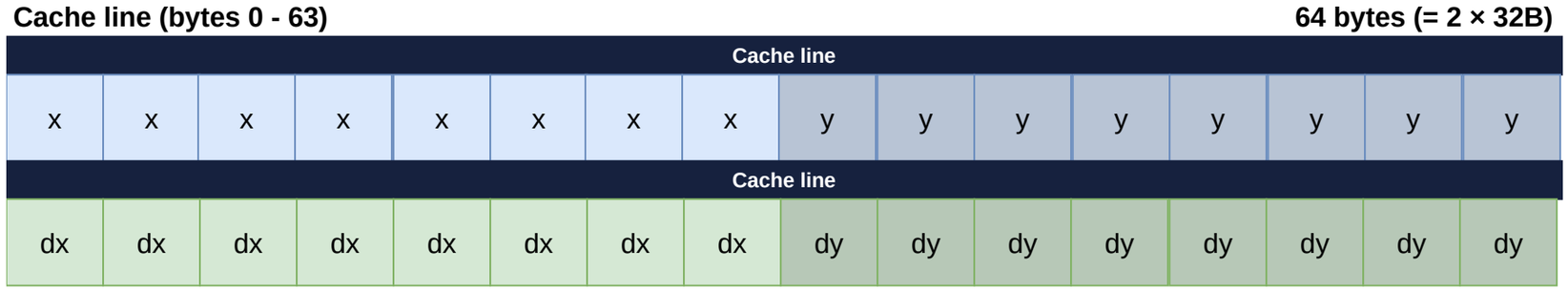
Data organization (new)

0x0	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B	**x** float 4B
0x20	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B	**y** float 4B
0x40	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B	**dx** float 4B
0x60	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B	**dy** float 4B
0x80	**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B	
0xa0	**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B		**lifetime** uint64_t 8B	
0xc0	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B	**color** Color(uint32) 4B
0x100	😸**No more space wasted for padding**😸							

This is also the correct alignment to allow the compiler to use SIMD instructions in our loop

FIXING THE DATA SPACE LOCALIZATION

New CPU cache organization



🐱No more unnecessary data in the cache🐱

FIXING THE DATA SPACE LOCALIZATION

The particle struct (revisited)

```
1 struct Particles
2 {
3   std::vector<float> x;  std::vector<float> y;
4   std::vector<float> dx; std::vector<float> dy;
5   std::vector<uint64_t> lifetime;
6   std::vector<Color>    color;
7 };
8
9 Particles particles;
```

FIXING THE DATA SPACE LOCALIZATION

The particle struct (revisited)

```
1 struct Particles
2 {
3   std::vector<float> x;  std::vector<float> y;
4   std::vector<float> dx; std::vector<float> dy;
5   std::vector<uint64_t> lifetime;
6   std::vector<Color>    color;
7 };
8
9 Particles particles;
```

This way of organizing the data is known as `Structure of Arrays` aka SoA

SOA IMPACTS

What about our UpdateParticles function? 🤔

```
 1 double UpdateParticles(float dt)
 2 {
 3   auto t0 = Clock::now();
 4   for (auto& p : particles)
 5   {
 6     p.x += p.dx * dt;
 7     p.y += p.dy * dt;
 8
 9     if      (p.x < 0)                p.x += ScreenWidth();
10     else if (p.x > ScreenWidth())    p.x -= ScreenWidth();
11     if      (p.y < 0)                p.y += ScreenHeight();
12     else if (p.y > ScreenHeight())   p.y -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

SOA IMPACTS

What about our UpdateParticles function? 🤔

```
1  double UpdateParticles(float dt)
2  {
3    auto t0 = Clock::now();
4    for (auto& p : particles)
5    {
6      p.x += p.dx * dt;
7      p.y += p.dy * dt;
8
9      if      (p.x < 0)            p.x += ScreenWidth();
10     else if (p.x > ScreenWidth())  p.x -= ScreenWidth();
11     if      (p.y < 0)            p.y += ScreenHeight();
12     else if (p.y > ScreenHeight())  p.y -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

It doesn't compile anymore

Demo: UpdateParti

```
in.cpp: In member function 'double
in.cpp:155:24: error: 'begin' was not declared in t
  155 |     for (auto& p : particles)
      |                    ^~~~~~~~~
```

SOA IMPACTS

What about our UpdateParticles function? 🤔

```
 1 double UpdateParticles(float dt)
 2 {
 3   auto t0 = Clock::now();
 4   for (uint32_t i; i < NB_PARTICLES; ++i)
 5   {
 6     particles.x[i] += particles.dx[i] * dt;
 7     particles.y[i] += particles.dy[i] * dt;
 8
 9     if      (particles.x[i] < 0)              particles.x[i] += ScreenWidth();
10     else if (particles.x[i] > ScreenWidth())  particles.x[i] -= ScreenWidth();
11     if      (particles.y[i] < 0)              particles.y[i] += ScreenHeight();
12     else if (particles.y[i] > ScreenHeight()) particles.y[i] -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

SOA IMPACTS

What about our UpdateParticles function? 🤔

```
 1 double UpdateParticles(float dt)
 2 {
 3   auto t0 = Clock::now();
 4   for (uint32_t i; i < NB_PARTICLES; ++i)
 5   {
 6 ❌  particles.x[i] += particles.dx[i] * dt;
 7 ❌  particles.y[i] += particles.dy[i] * dt;
 8
 9 ❌  if      (particles.x[i] < 0)                particles.x[i] += ScreenWidth();
10 ❌❌ else if (particles.x[i] > ScreenWidth())   particles.x[i] -= ScreenWidth();
11 ❌❌ if      (particles.y[i] < 0)                particles.y[i] += ScreenHeight();
12 ❌  else if (particles.y[i] > ScreenHeight())  particles.y[i] -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

SOA IMPACTS

What about our UpdateParticles function? 🤔

```
 1 double UpdateParticles(float dt)
 2 {
 3   auto t0 = Clock::now();
 4   for (uint32_t i; i < NB_PARTICLES; ++i)
 5   {
 6 ❌  particles.x[i] += particles.dx[i] * dt;
 7 ❌  particles.y[i] += particles.dy[i] * dt;
 8
 9 ❌  if      (particles.x[i] < 0)                particles.x[i] += ScreenWidth();
10 ❌  else if (particles.x[i] > ScreenWidth())    particles.x[i] -= ScreenWidth();
11 ❌  if      (particles.y[i] < 0)                particles.y[i] += ScreenHeight();
12 ❌  else if (particles.y[i] > ScreenHeight())   particles.y[i] -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

🙀That's a lot of impact🙀

SOA IMPACTS

MORE IMPACT

Passing a Particle to a function becomes difficult

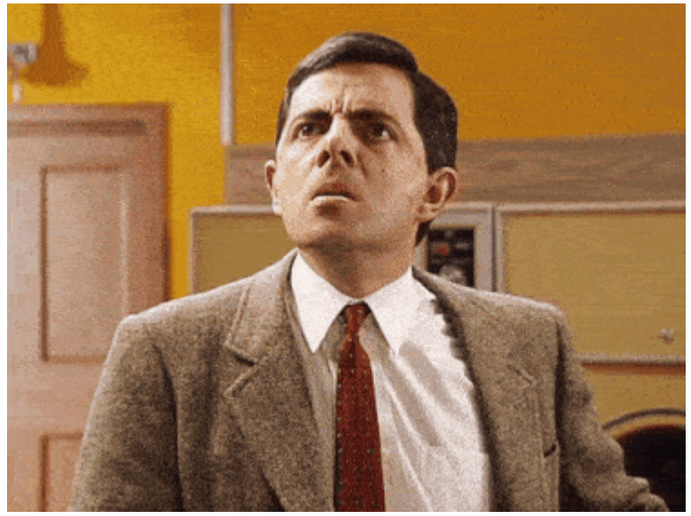
```
1 void printParticle(const Particle& particle)
```

becomes

```
1 void printParticle(float x, float y, float dx, float dy, ...)
```

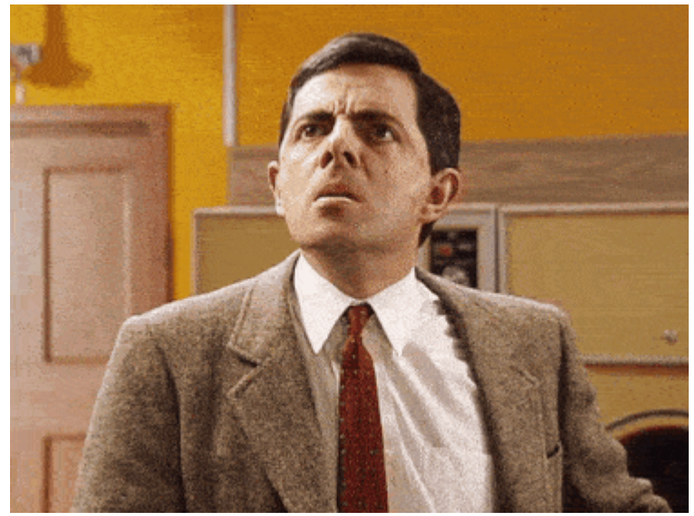
SOA IMPACTS

SERIOUSLY?



SOA IMPACTS

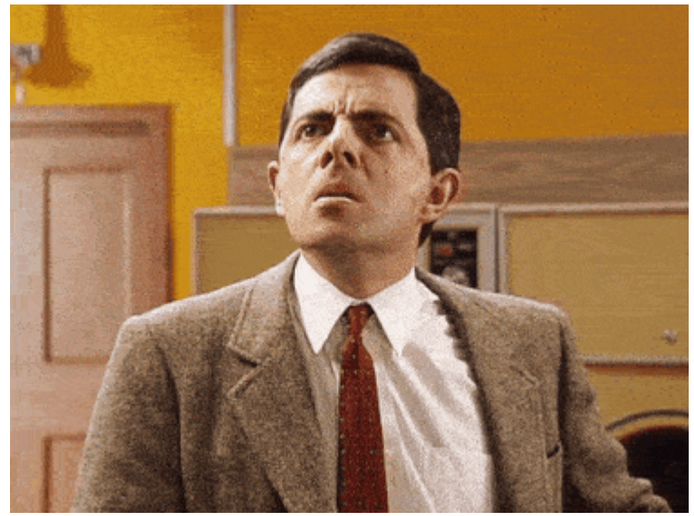
SERIOUSLY?



Do we need to adapt all of our code base? 🙀

SOA IMPACTS

SERIOUSLY?

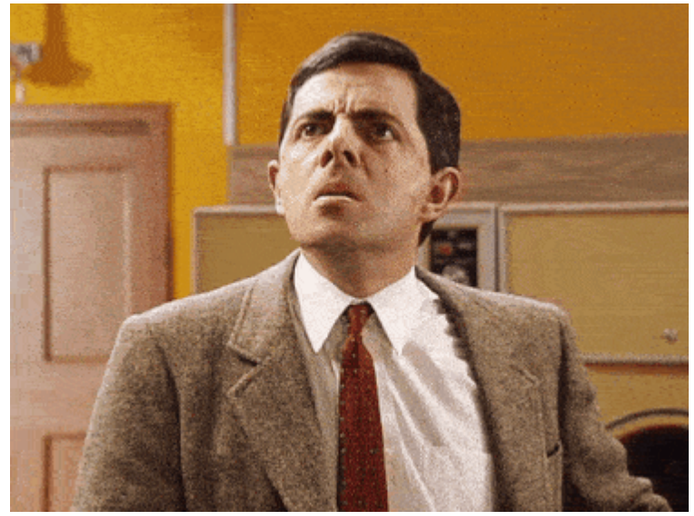


Do we need to adapt all of our code base? 🙀

Make our code ugly? 🙀

SOA IMPACTS

SERIOUSLY?



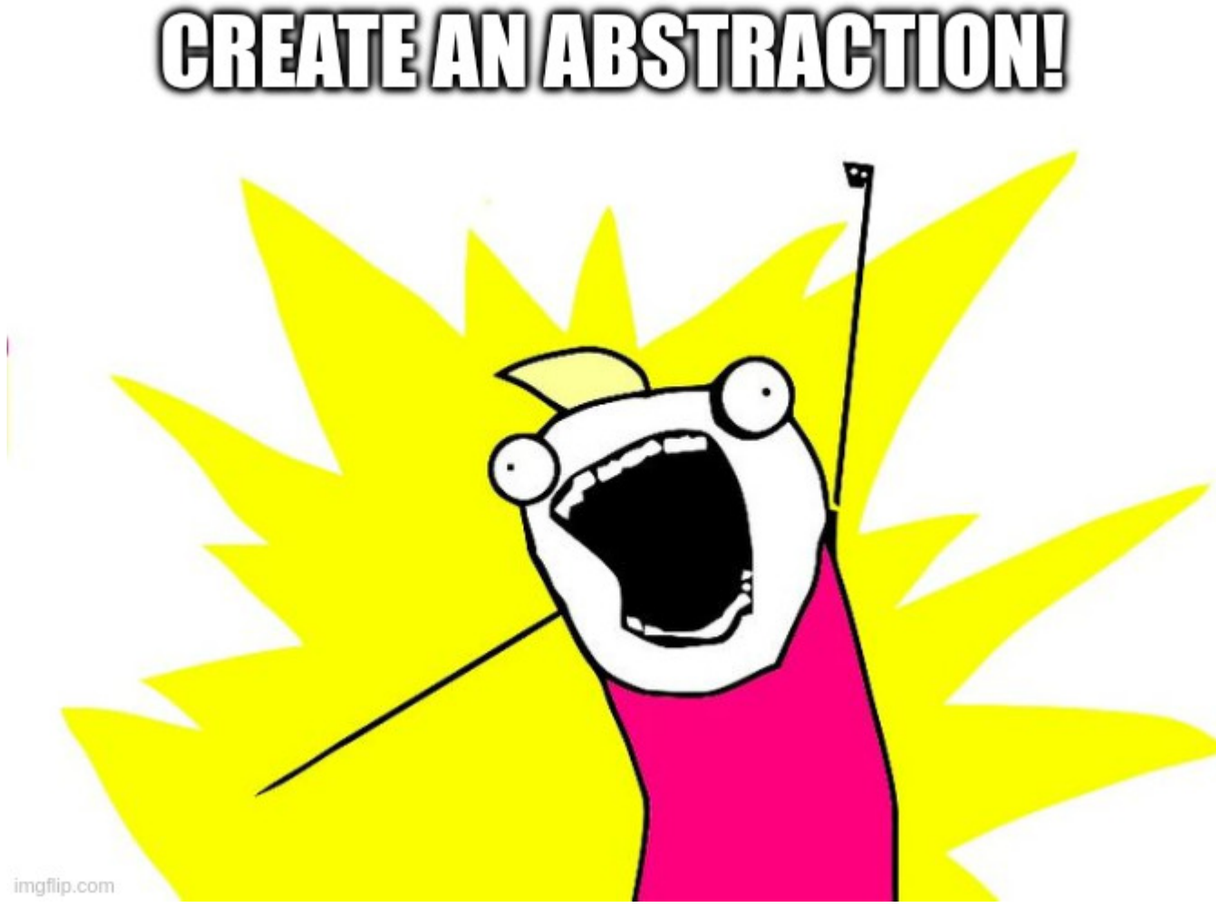
Do we need to adapt all of our code base? 🙀

Make our code ugly? 🙀

For performance reason? 🤔

What solution can we have? 🤔

What solution can we have? 🤔

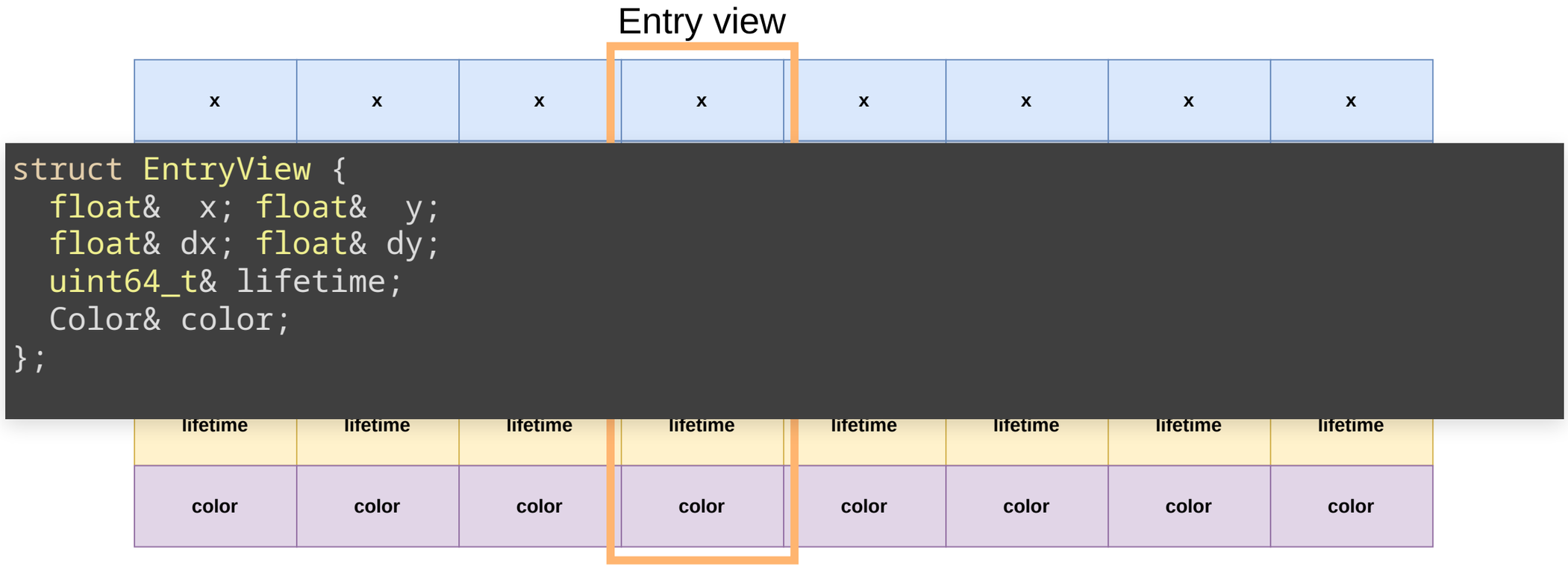


THE ENTRY VIEW ABSTRACTION

Entry view

x	x	x	x	x	x	x	x
y	y	y	y	y	y	y	y
dx	dx	dx	dx	dx	dx	dx	dx
dy	dy	dy	dy	dy	dy	dy	dy
lifetime	lifetime	lifetime	lifetime	lifetime	lifetime	lifetime	lifetime
color	color	color	color	color	color	color	color

THE ENTRY VIEW ABSTRACTION




```
struct EntryView {
  float&  x; float&  y;
  float& dx; float& dy;
  uint64_t& lifetime;
  Color& color;
};
```

THE DECLTYPE SPECIFIER

THE DECLTYPE SPECIFIER

- Inspects the declared type of an entity or the type and value category of an expression

THE DECLTYPE SPECIFIER

- Inspects the declared type of an entity or the type and value category of an expression
- Simply said, it retrieves the type and its CV qualifier of an expression

```
1 uint32_t a{0};
2 static_assert(std::is_same_v<decltype(a), uint32_t>);
```

THE DECLTYPE SPECIFIER

- Inspects the declared type of an entity or the type and value category of an expression
- Simply said, it retrieves the type and its CV qualifier of an expression

```
1 uint32_t a{0};
2 static_assert(std::is_same_v<decltype(a), uint32_t>);
```

- decltype can be used to declare a variable of the same type of another

```
1 decltype(a) b{0};
2 // b has the same type as a i.e. uint32_t
3 static_assert(std::is_same_v<decltype(a), decltype(b)>);
```

THE DECLTYPE SPECIFIER

- Also works on functions

```
1 int& fun();
2 static_assert(std::is_same_v<decltype(fun()), int&>);
```

BUILDING THE ENTRY VIEW

```
1 struct Particles
2 {
3   std::vector<float> x;  std::vector<float> y;
4   std::vector<float> dx; std::vector<float> dy;
5   std::vector<uint64_t> lifetime;
6   std::vector<Color>    color;
7 };
```

BUILDING THE ENTRY VIEW

```
1 struct Particles
2 {
3 private:
4   std::vector<float> x;  std::vector<float> y;
5   std::vector<float> dx; std::vector<float> dy;
6   std::vector<uint64_t> lifetime;
7   std::vector<Color>    color;
8 };
```

BUILDING THE ENTRY VIEW

```
 1 struct Particles
 2 {
 3 private:
 4   std::vector<float> x;  std::vector<float> y;
 5   std::vector<float> dx; std::vector<float> dy;
 6   std::vector<uint64_t> lifetime;
 7   std::vector<Color>    color;
 8
 9   template <typename T>
10   struct EntryViewT
11   {
12   };
13 };
```

BUILDING THE ENTRY VIEW

```
 1 struct Particles
 2 {
 3 private:
 4   std::vector<float> x;  std::vector<float> y;
 5   std::vector<float> dx; std::vector<float> dy;
 6   std::vector<uint64_t> lifetime;
 7   std::vector<Color>    color;
 8
 9   template <typename T>
10   struct EntryViewT
11   {
12     private:
13       T& parent_ref;
14     public:
15       decltype(parent_ref.x[0]) x; // A reference to x at a particular index
```

BUILDING THE ENTRY VIEW

```
 4   std::vector<float> x;  std::vector<float> y;
 5   std::vector<float> dx; std::vector<float> dy;
 6   std::vector<uint64_t> lifetime;
 7   std::vector<Color>    color;
 8
 9   template <typename T>
10   struct EntryViewT
11   {
12     private:
13       T& parent_ref;
14     public:
15       decltype(parent_ref.x[0]) x; // A reference to x at a particular index
16       static_assert(std::is_lvalue_reference_v<decltype(x)>);
17   };
18 };
```

BUILDING THE ENTRY VIEW

```
 1 struct Particles
 2 {
 3 private:
 4   std::vector<float> x;  std::vector<float> y;
```

std::vector operator[] declaration

```
reference operator[]( size_type pos );
```

```
11   {
12     private:
13       T& parent_ref;
14     public:
15       decltype(parent_ref.x[0]) x; // A reference to x at a particular index
```

BUILDING THE ENTRY VIEW

```
 7 struct EntryViewT
 8   {
 9     private:
10       T& parent_ref;
11     public:
12       decltype(parent_ref.x[0])      x;
13       static_assert(std::is_lvalue_reference_v<decltype(x)>);
14       decltype(parent_ref.y[0])      y;      static_assert(std::is_lvalue_refere
15       decltype(parent_ref.dx[0])     dx;     static_assert(std::is_lvalue_refere
16       decltype(parent_ref.dy[0])     dy;     static_assert(std::is_lvalue_refere
17       decltype(parent_ref.radius[0]) radius; static_assert(std::is_lvalue_refere
18       decltype(parent_ref.mass[0])   mass;   static_assert(std::is_lvalue_refere
19       decltype(parent_ref.color[0])  color;  static_assert(std::is_lvalue_refere
20   };
```

BUILDING THE ENTRY VIEW

```
23 template <typename T>
24   struct EntryViewT
25   {
26     EntryViewT(T& parent, std::size_t index) :
27       parent_ref(parent),
28       x          (parent.x     [index]),
29       y          (parent.y     [index]),
30       dx         (parent.dx    [index]),
31       dy         (parent.dy    [index]),
32       radius     (parent.radius[index]),
33       mass       (parent.mass  [index]),
34       color      (parent.color [index])
35     {}
36   };
```

BUILDING THE ENTRY VIEW

```
35    using Entry = EntryViewT<Particles>;
36    Entry operator[](std::size_t index) { return Entry(*this, index); }
```

BUILDING THE ENTRY VIEW

```
38   using ConstEntry = EntryViewT<const Particles>;
39   ConstEntry operator[](std::size_t index) const
40     { return ConstEntry(*this, index); }
```

BUILDING THE ENTRY VIEW

```
38   using ConstEntry = EntryViewT<const Particles>;
39   ConstEntry operator[](std::size_t index) const
40     { return ConstEntry(*this, index); }
```

The use of decltype in EntryViewT is important to make sure CV qualifier are properly applied to the references

SUPPORT FOR RANGE-BASED FOR LOOP

```
41   template<typename T>
42   struct IteratorT
43   {
44     T&  parent;
45     std::size_t index;
46
47     EntryViewT<T> operator*()  { return EntryViewT<T>(parent, index); }
48     IteratorT&    operator++() { ++index; return *this; }
49     bool operator!=(const IteratorT& other) const
50       { return index != other.index; }
51   };
```

SUPPORT FOR RANGE-BASED FOR LOOP

```
53    using Iterator = IteratorT<Particles>;
54    using ConstIterator = IteratorT<const Particles>;
55
56    Iterator begin() { return Iterator{ *this, 0 }; }
57    Iterator end()   { return Iterator{ *this, size() }; }
58
59    ConstIterator begin() const { return ConstIterator{ *this, 0 }; }
60    ConstIterator end()   const { return ConstIterator{ *this, size() }; }
```

BACK TO UPDATE PARTICLES

```
 1 double UpdateParticles(float dt)
 2 {
 3   auto t0 = Clock::now();
 4   for (auto&& p : particles)
 5   {
 6     p.x += p.dx * dt;
 7     p.y += p.dy * dt;
 8
 9     if      (p.x < 0)                p.x += ScreenWidth();
10     else if (p.x > ScreenWidth())    p.x -= ScreenWidth();
11     if      (p.y < 0)                p.y += ScreenHeight();
12     else if (p.y > ScreenHeight())   p.y -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

BACK TO UPDATE PARTICLES

```
 1 double UpdateParticles(float dt)
 2 {
 3   auto t0 = Clock::now();
 4   for (auto&& p : particles)
 5   {
 6     p.x += p.dx * dt;
 7     p.y += p.dy * dt;
 8
 9     if      (p.x < 0)                p.x += ScreenWidth();
10     else if (p.x > ScreenWidth())    p.x -= ScreenWidth();
11     if      (p.y < 0)                p.y += ScreenHeight();
12     else if (p.y > ScreenHeight())   p.y -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

BACK TO UPDATE PARTICLES

```
1  double UpdateParticles(float dt)
2  {
3    auto t0 = Clock::now();
4    for (auto&& p : particles)
5    {
6      p.x += p.dx * dt;
7      p.y += p.dy * dt;
8
9      if      (p.x < 0)                p.x += ScreenWidth();
10     else if (p.x > ScreenWidth())    p.x -= ScreenWidth();
11     if      (p.y < 0)                p.y += ScreenHeight();
12     else if (p.y > ScreenHeight())   p.y -= ScreenHeight();
13   }
14   return measure_ms(t0, Clock::now());
15 }
```

BACK TO UPDATE PARTICLES

```
double UpdateParticles(float dt)
{
  auto t0 = Clock::now();
  for (auto&& p : particles)
  {
    p.x += p.dx * dt;
    p.y += p.dy * dt;

    if      (p.x < 0)                p.x += ScreenWidth();
    else if (p.x > ScreenWidth())    p.x -= ScreenWidth();
    if      (p.y < 0)                p.y += ScreenHeight();
    else if (p.y > ScreenHeight())   p.y -= ScreenHeight();
  }
  return measure_ms(t0, Clock::now());
}
```

😺 Very limited impact 😺

DEMO

```
/*****************************************************************************
*
*   SoA vs AoS cache demonstration - Particle System (no collisions)
*   Compile with: -O3
*****************************************************************************/
#include <type_traits>
#include <random>
#include <vector>
#include <cmath>
#include <chrono>
#include <string>
#include <memory>

#if !defined(AoS) && !defined(SoA)
#error "Use -DAoS or -DSoA to compile"
```

```
 8 #include <vector>
 9 #include <cmath>
10 #include <chrono>
11 #include <string>
12 #include <memory>
13
14 #if !defined(AoS) && !defined(SoA)
15 #error "Use -DAoS or -DSoA to compile"
16 #endif
17
18 #define OLC_PGE_APPLICATION
19 #include "olcPixelGameEngine.h"
20
21 static constexpr int   NUM_PARTICLES      = 4000000;
```

```
29
30 // ─── AoS ──────────────────────────────────────────────────────────────
31
32 #if defined(AoS)
33
34 struct Particle
35 {
36     float      x, y;
37     float      dx, dy;
38     float      radius;
39     float      mass;
40     olc::Pixel color;
41 };
42
43 using ParticlesContainer = std::vector<Particle>;
```

```
45 // --- SoA ---
46
47 #elif defined(SoA)
48
49 struct Particles
50 {
51 private:
52     std::vector<float>      x, y, dx, dy;
53     std::vector<float>      radius, mass;
54     std::vector<olc::Pixel> color;
55
56     template <typename T>
57     struct EntryT
58     {
59         T& parent_ref;
```

```
159     {
160         auto t0 = Clock::now();
161
162         // This loop is identical for AoS and SoA.
163         // AoS: each iteration loads a full struct — cold fields waste cache ba
164         // SoA: each iteration touches only references into contiguous float ar
165         //      cold arrays (radius, mass, color) are never loaded into cache a
166         for (auto&& p : particles)
167         {
168             p.x += p.dx * dt;
169             p.y += p.dy * dt;
170
171             if      (p.x < 0)                p.x += (float)ScreenWidth();
172             else if (p.x > ScreenWidth())     p.x -= (float)ScreenWidth();
```

```
g++ main.cpp -o game -O3 -lX11 -lGL -lpng -pthread -std=c++17 -DAoS
```

```
g++ main.cpp -o game -O3 -lX11 -lGL -lpng -pthread -std=c++17 -DSoA
```

GOING FURTHER WITH C++26'S REFLECTION

GOING FURTHER WITH C++26'S REFLECTION

- The entry view is quite verbose

GOING FURTHER WITH C++26'S REFLECTION

- The entry view is quite verbose
- "code duplication" per members

GOING FURTHER WITH C++26'S REFLECTION

- The entry view is quite verbose
- "code duplication" per members
- With C++26 reflection, we can generate the entry view

GOING FURTHER WITH C++26'S REFLECTION

```
template <typename T>
struct SoaViewImpl {

  struct view;

  consteval {
    std::vector<std::meta::info> view_members = {};

    template for (constexpr std::meta::info member : std::define_static_array(
      nonstatic_data_members_of(^^T, std::meta::access_context::current()))) {
        constexpr std::meta::info elem_type =
          template_arguments_of(type_of(member))[0];

        view_members.push_back(
          data_member_spec(^^std::add_lvalue_reference_t<typename[:elem_type:]>
```

GOING FURTHER WITH C++26'S REFLECTION

```
template <typename T>
struct SoaViewImpl {

  struct view;

  consteval {
    std::vector<std::meta::info> view_members = {};

    template for (constexpr std::meta::info member : std::define_static_array(
      nonstatic_data_members_of(^^T, std::meta::access_context::current()))) {
        constexpr std::meta::info elem_type =
          template_arguments_of(type_of(member))[0];

        view_members.push_back(
```

GOING FURTHER WITH C++26'S REFLECTION

```
template <typename T>
struct SoaViewImpl {

  struct view;

  consteval {
    std::vector<std::meta::info> view_members = {};

    template for (constexpr std::meta::info member : std::define_static_array(
      nonstatic_data_members_of(^^T, std::meta::access_context::current()))) {
        constexpr std::meta::info elem_type =
          template_arguments_of(type_of(member))[0];

        view_members.push_back(
          data_member_spec(^^std::add_lvalue_reference_t<typename[:elem_type:]>
```

GOING FURTHER WITH C++26'S REFLECTION

```
template <typename T>
struct SoaViewImpl {

  struct view;

  consteval {
    std::vector<std::meta::info> view_members = {};

    template for (constexpr std::meta::info member : std::define_static_array(
      nonstatic_data_members_of(^^T, std::meta::access_context::current()))) {
        constexpr std::meta::info elem_type =
          template_arguments_of(type_of(member))[0];

        view_members.push_back(
          data_member_spec(^^std::add_lvalue_reference_t<typename[:elem_type:]>
```

GOING FURTHER WITH C++26'S REFLECTION

```
 2 struct SoaViewImpl {
 3
 4   struct view;
 5
 6   consteval {
 7     std::vector<std::meta::info> view_members = {};
 8
 9     template for (constexpr std::meta::info member : std::define_static_array(
10       nonstatic_data_members_of(^^T, std::meta::access_context::current()))) {
11         constexpr std::meta::info elem_type =
12           template_arguments_of(type_of(member))[0];
13
14         view_members.push_back(
15           data_member_spec(^^std::add_lvalue_reference_t<typename[:elem_type:]>,
16             {.name = identifier_of(member)}));
```

GOING FURTHER WITH C++26'S REFLECTION

```
struct SoaViewImpl {

  struct view;

  consteval {
    std::vector<std::meta::info> view_members = {};

          template_arguments_of(type_of(member))[0];

        view_members.push_back(
          data_member_spec(^^std::add_lvalue_reference_t<typename[:elem_type:]>,
            {.name = identifier_of(member)}));
```

member = `std::vector<float>`

GOING FURTHER WITH C++26'S REFLECTION

```
4   struct View;
5
6   consteval {
7     std::vector<std::meta::info> view_members = {};
8
9     template for (constexpr std::meta::info member : std::define_static_array(
10      nonstatic_data_members_of(^^T, std::meta::access_context::current()))) {
11        constexpr std::meta::info elem_type =
12          template_arguments_of(type_of(member))[0];
13
14        view_members.push_back(
15          data_member_spec(^^std::add_lvalue_reference_t<typename[:elem_type:]>,
16            {.name = identifier_of(member)}));
17    }
18  }
```

GOING FURTHER WITH C++26'S REFLECTION

```
4   struct View;
5
6   consteval {
7     std::vector<std::meta::info> view_members = {};
8
9     template for (constexpr std::meta::info member : std::define_static_array(
10
11
12
13
14        view_members.push_back(
15          data_member_spec(^^std::add_lvalue_reference_t<typename[:elem_type:]>,
16            {.name = identifier_of(member)}));
17    }
18  }
```

element_type = `float`

GOING FURTHER WITH C++26'S REFLECTION

```
 8
 9      template for (constexpr std::meta::info member : std::define_static_array(
10        nonstatic_data_members_of(^^T, std::meta::access_context::current()))) {
11          constexpr std::meta::info elem_type =
12            template_arguments_of(type_of(member))[0];
13
14          view_members.push_back(
15            data_member_spec(^^std::add_lvalue_reference_t<typename[:elem_type:]>,
16              {.name = identifier_of(member)}));
17      }
18    }
19
20    define_aggregate(^^view, view_members);
21
22    static constexpr std::size_t member_count =
```

GOING FURTHER WITH C++26'S REFLECTION

```
13
14          view_members.push_back(
15            data_member_spec(^^std::add_lvalue_reference_t<typename[:elem_type:]>,
16              {.name = identifier_of(member)}));
17      }
18    }
19
20    define_aggregate(^^view, view_members);
21
22    static constexpr std::size_t member_count =
23      nonstatic_data_members_of(^^T, std::meta::access_context::current()).size();
24
25    static auto make(T& src, std::size_t i) {
26      return [&]<std::size_t... Is>(std::index_sequence<Is...>) {
27        constexpr auto members = std::define_static_array(
```

GOING FURTHER WITH C++26'S REFLECTION

```
18    }
19
20    define_aggregate(^^view, view_members);
21
22    static constexpr std::size_t member_count =
23      nonstatic_data_members_of(^^T, std::meta::access_context::current()).size();
24
25    static auto make(T& src, std::size_t i) {
26      return [&]<std::size_t... Is>(std::index_sequence<Is...>) {
27        constexpr auto members = std::define_static_array(
28          nonstatic_data_members_of(^^T, std::meta::access_context::current()));
29        return typename SoaViewImpl<T>::view{src.[:members[Is]:][i]...};
30      }(std::make_index_sequence<member_count>{});
31    }
32  };
```

THE PARTICLES STRUCT (FINAL VERSION)

```
1 struct Particles {
2   std::vector<float> x;  std::vector<float> y;
3   std::vector<float> dx; std::vector<float> dy;
4   std::vector<uint64_t> lifetime;
5   std::vector<Color>    color;
6
7   using View = SoaViewImpl<Particles>
8   auto operator[](std::size_t i) { return View::make(*this, i); }
9 };
```

CONCLUSION

CONCLUSION

- Data layout is critical for performance
  Memory access patterns can matter more than algorithms

CONCLUSION

- Data layout is critical for performance
  Memory access patterns can matter more than algorithms
- SoA improves cache efficiency
  Better spatial locality → fewer cache misses → big speedups

CONCLUSION

- Data layout is critical for performance
  Memory access patterns can matter more than algorithms
- SoA improves cache efficiency
  Better spatial locality → fewer cache misses → big speedups
- SoA can hurt usability and maintainability
  Code becomes harder to write, read, and evolve

CONCLUSION

- Data layout is critical for performance
  Memory access patterns can matter more than algorithms
- SoA improves cache efficiency
  Better spatial locality → fewer cache misses → big speedups
- SoA can hurt usability and maintainability
  Code becomes harder to write, read, and evolve
- Abstractions can give you both
  Zero-cost abstraction (templates + decltype) restores clean APIs

CONCLUSION

- Data layout is critical for performance
  Memory access patterns can matter more than algorithms
- SoA improves cache efficiency
  Better spatial locality → fewer cache misses → big speedups
- SoA can hurt usability and maintainability
  Code becomes harder to write, read, and evolve
- Abstractions can give you both
  Zero-cost abstraction (templates + decltype) restores clean APIs
- Design data first, APIs second ?
  NO! Performance and maintainability are not mutually exclusive

CONCLUSION

- Data layout is critical for performance
  Memory access patterns can matter more than algorithms
- SoA improves cache efficiency
  Better spatial locality → fewer cache misses → big speedups
- SoA can hurt usability and maintainability
  Code becomes harder to write, read, and evolve
- Abstractions can give you both
  Zero-cost abstraction (templates + decltype) restores clean APIs
- Design data first, APIs second ?
  NO! Performance and maintainability are not mutually exclusive
- Reflection
  Generic way to generate abstraction at compile time

 QUESTION ?