A large, faint, circular watermark logo is centered in the background. It features a central emblem surrounded by text in a circular arrangement, likely representing a conference or organization. The text is illegible due to its low opacity.

OOP is dead, long live Data-oriented design

Stoyan Nikolov

@stoyannk

Who am I?

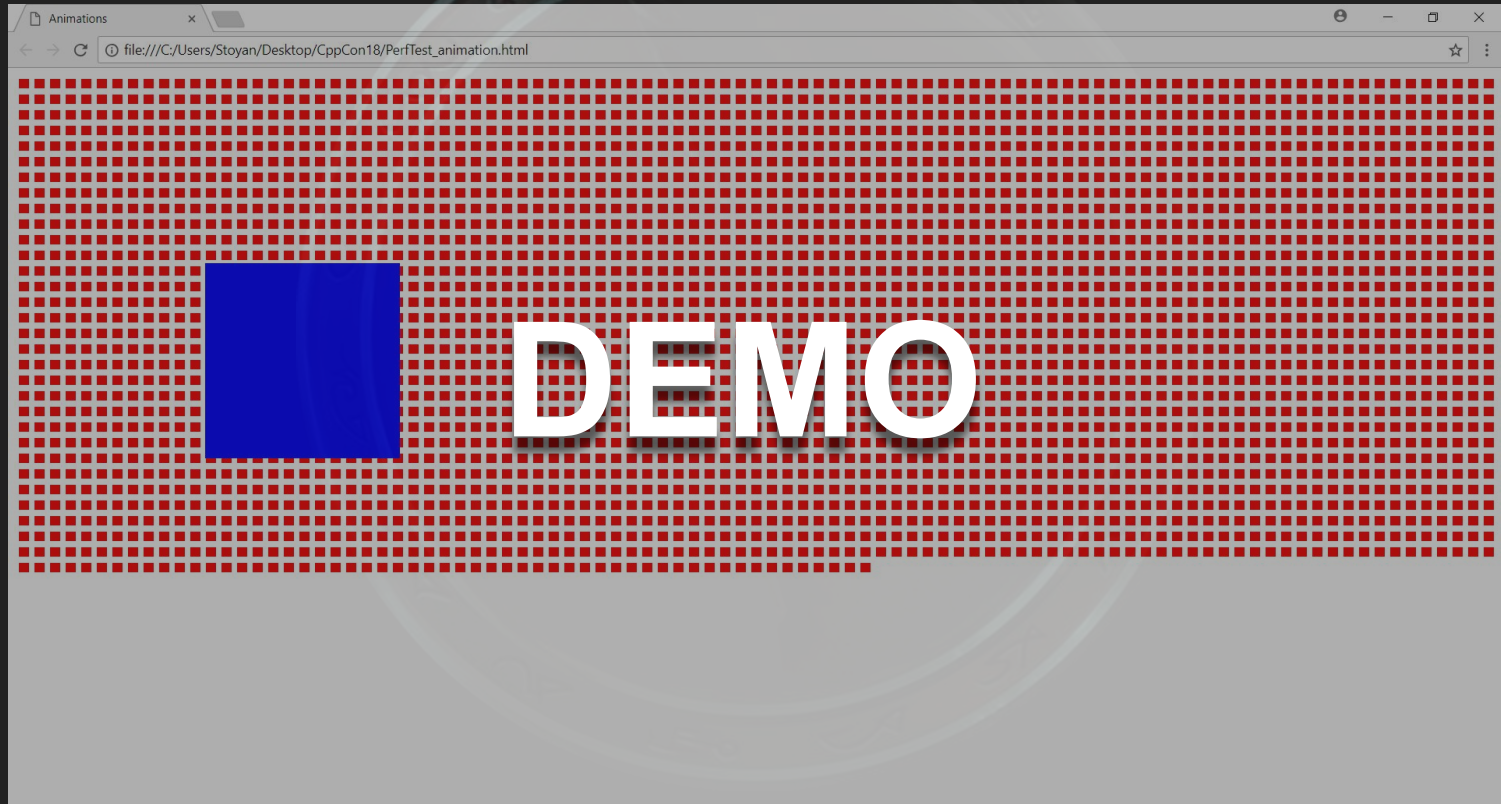
- In the video games industry for 10+ years
- Software Architect at [Coherent Labs](#)
- Working on game development technology
- Last 6.5 years working on
 - chromium
 - WebKit
 - Hummingbird - in-house game UI & browser engine
- High-performance maintainable C++



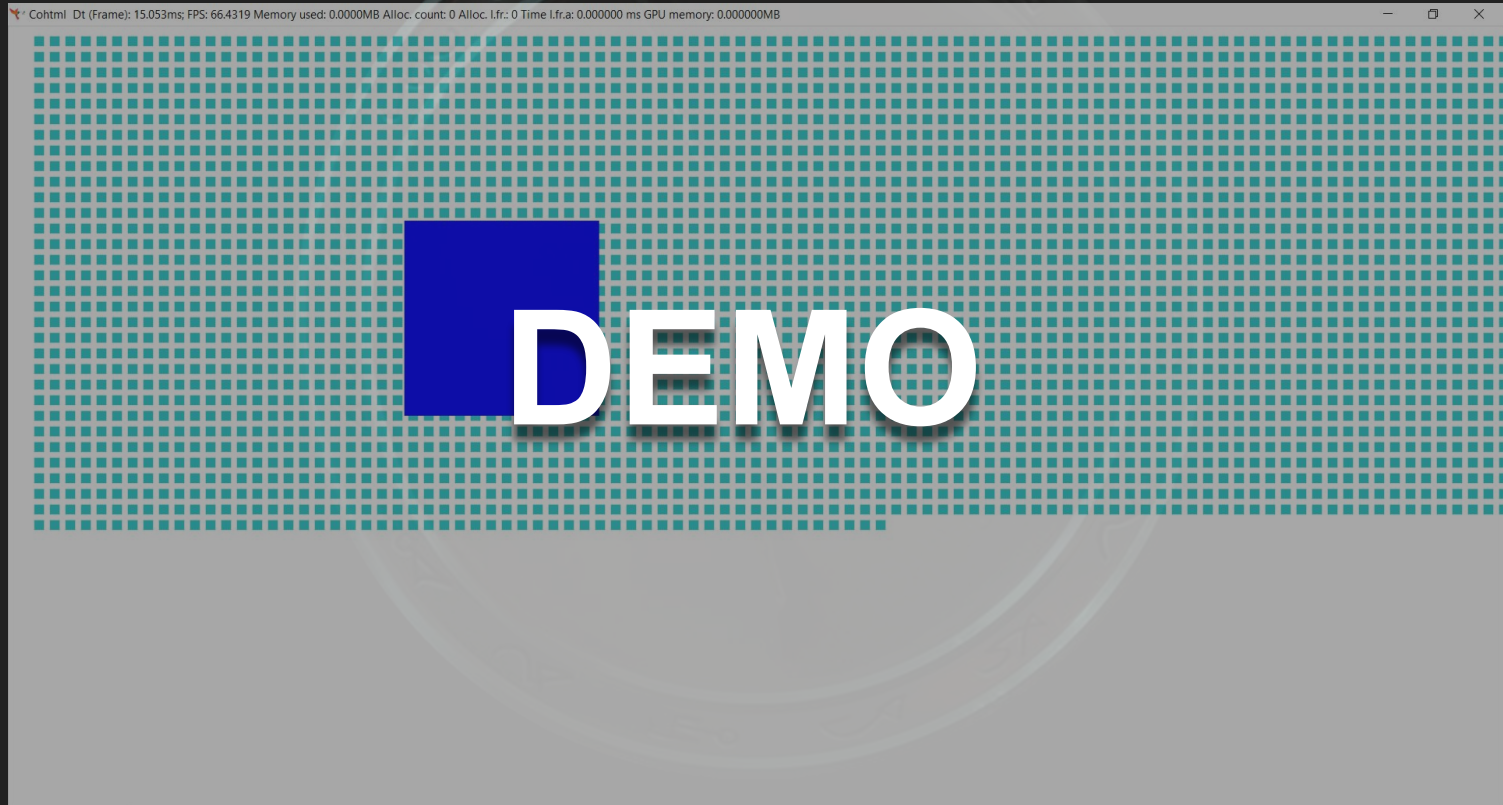
*Games using Coherent Labs technology
Images courtesy of Rare Ltd., PUBG Corporation*

Can a browser engine be successful with
data-oriented design?

CSS Animations with chromium (OOP)



CSS Animations with Hummingbird (DoD)



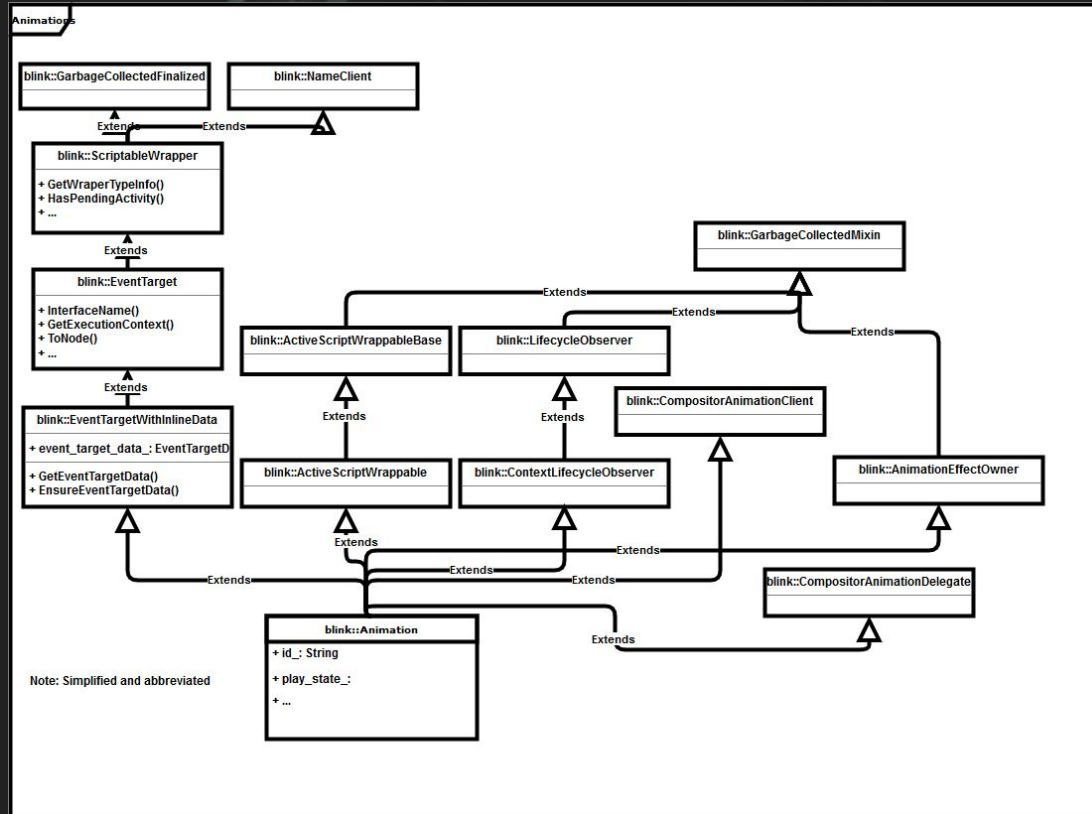
Yes

Agenda

- Basic issue with Object-oriented programming (OOP)
- Basics of Data-oriented design (DoD)
- Problem definition
- Object-oriented programming approach
- Data-oriented design approach
- Results & Analysis



What is so wrong with OOP?

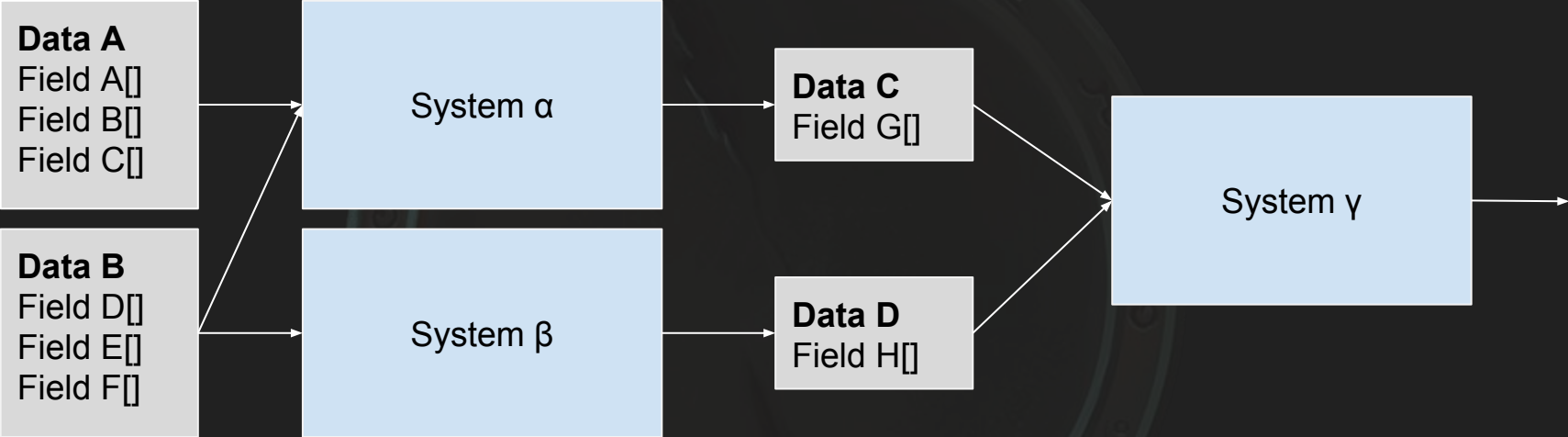


OOP marries data with operations...

- ...it's not a happy marriage
- Heterogeneous data is brought together by a “logical” **black box** object
- The object is used in vastly different contexts
- Hides **state** all over the place
- Impact on
 - **Performance**
 - **Scalability**
 - **Modifiability**
 - **Testability**



Data-oriented design



Logical Entity 0		Logical Entity 1	
Field A[0]	Field D[0]	Field A[1]	Field D[1]
Field B[0]	Field E[0]	Field B[1]	Field E[1]
Field C[0]	Field F[0]	Field C[1]	Field F[1]

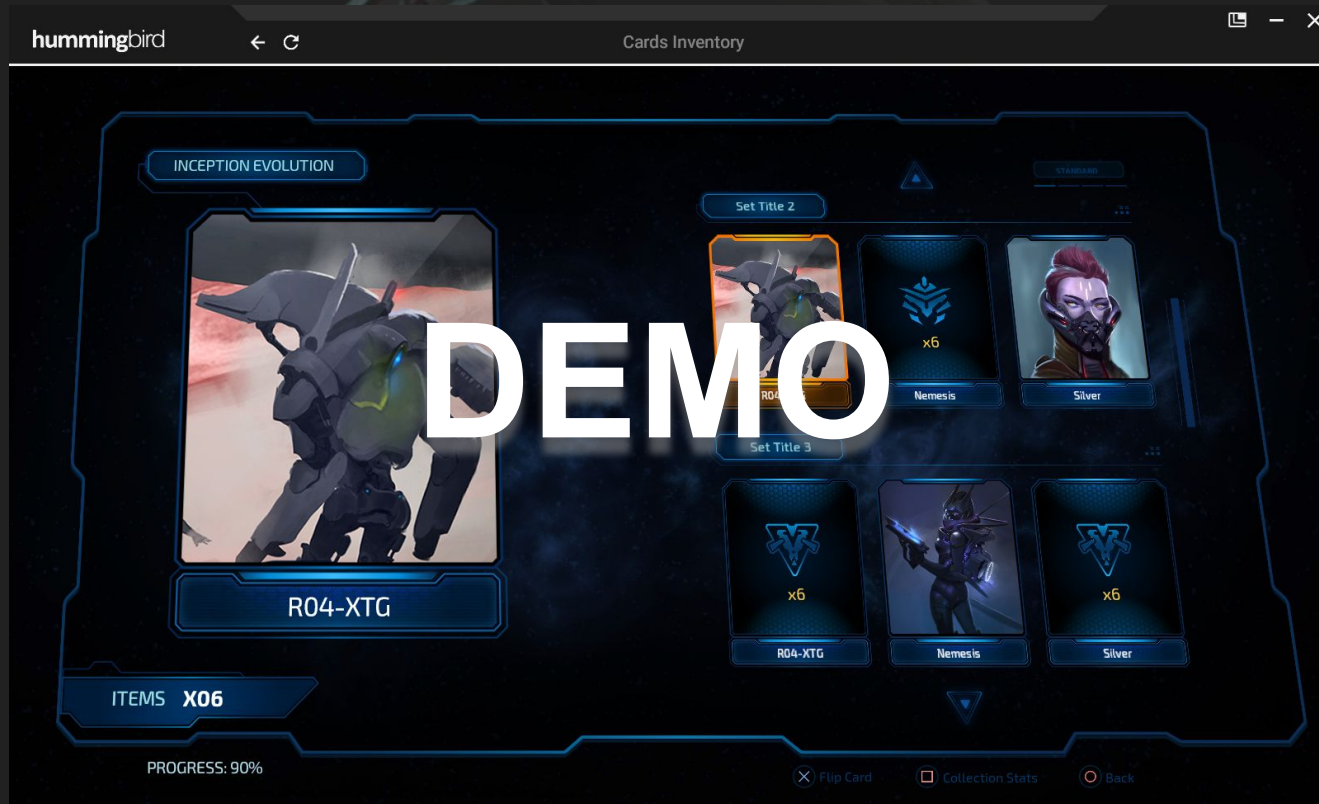
...

Data-oriented design - the gist

- **Separates** data from logic
 - Structs and functions live independent lives
 - Data is regarded as information that has to be transformed
- The logic embraces the data
 - Does not try to hide it
 - Leads to functions that work on arrays
- Reorganizes data according to it's usage
 - If we aren't going to use a piece of information, why pack it together?
- Avoids “hidden state”
- No virtual calls
 - There is no need for them
- Promotes deep **domain knowledge**
- **References** at the end for more detail

The system at hand

What is a CSS Animation?



Animation definition

```
@keyframes example {  
  from {left: 0px;}  
  to {left: 100px;}  
}  
div {  
  width: 100px;  
  height: 100px;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 1s;  
}
```

- Straightforward declaration
 - Interpolate some properties over a period of time
 - Apply the Animated property on the right Elements
- However at a second glance..
 - Different property types (i.e. a **number** and a **color**)
 - There is a DOM API (JavaScript) that requires the existence of some classes (Animation, KeyframeEffect etc.)

Let's try OOP

The OOP way (chromium 66)

- chromium has 2 Animation systems
 - We'll be looking at the Blink system
- Employs some classic OOP
 - Closely follows the HTML5 standard and IDL
 - Running Animation are **separate objects**
- Study chromium - it's an amazing piece of software, a lot to learn!

```
class CORE_EXPORT Animation final : public EventTargetWithInlineData,  
                                   public ActiveScriptWrappable<Animation>,  
                                   public ContextLifecycleObserver,  
                                   public CompositorAnimationDelegate,  
                                   public CompositorAnimationClient,  
                                   public AnimationEffectOwner {
```


The flow

- Unclear lifetime semantics

```
135 void DocumentTimeline::ServiceAnimations(TimingUpdateReason reason) {
136     TRACE_EVENT0("blink", "DocumentTimeline::serviceAnimations");
137
138     last_current_time_internal_ = CurrentTimeInternal();
139
140     HeapVector<Member<Animation>> animations;
141     animations.ReserveInitialCapacity(animations_needing_update_.size());
142     for (Animation* animation : animations_needing_update_) ▶
143         animations.push_back(animation);
144
145     std::sort(animations.begin(), animations.end(), Animation::HasLowerPriority);
146
147     for (Animation* animation : animations) {
148         if (!animation->Update(reason))
149             animations_needing_update_.erase(animation);
150     }
```

The state

- Hidden state
- Branch mispredictions

```
952 bool Animation::Update(TimingUpdateReason reason) {
953     if (!timeline_) { // 1ms elapsed
954         return false;
955     }
956     PlayStateUpdateScope update_scope(*this, reason, kDoNotSetCompositorPending);
957
958     ClearOutdated();
959     bool idle = PlayStateInternal() == kIdle;
960
961     if (content_) {
962         double inherited_time = idle || IsNull(timeline_>CurrentTimeInternal())
963             ? NullValue()
964             : CurrentTimeInternal();
965     }
966 }
```

The KeyframeEffect

```
Member<AnimationEffectReadOnly> content_;  
Member<DocumentTimeline> timeline_;
```

```
961  if (content_) {  
962      double inherited_time = idle || IsNull(timeline_>CurrentTimeInterval())  
963          ? NullValue()  
964          : CurrentTimeInterval();  
965  
966      // Special case for end-exclusivity when playing backwards.  
967      if (inherited_time == 0 && playback_rate_ < 0)  
968          inherited_time = -1;  
969      content_>UpdateInheritedTime(inherited_time, reason);  
970  }
```

- Cache misses

Updating time and values

- Jumping contexts
- Cache misses (data and instruction)
- Coupling between systems (animations and events)

```
242     if (reason == kTimingUpdateForAnimationFrame &&
243         (!owner_ || owner_ ->IsEventDispatchAllowed())) {
244         if (event_delegate_)
245             event_delegate_ ->OnEventCondition(*this);
246     }
247
248     if (needs_update) { ▶ 1ms elapsed
249         // FIXME: This probably shouldn't be recursive.
250         UpdateChildrenAndEffects();
251         calculated_.time_to_forwards_effect_change =
252             CalculateTimeToEffectChange(true, local_time, time_to_next_iteration);
253         calculated_.time_to_reverse_effect_change =
254             CalculateTimeToEffectChange(false, local_time, time_to_next_iteration);
255     }
```


Interpolate different types of values

```
15 void InvalidatableInterpolation::Interpolate(int, double fraction) {  
60 class CORE_EXPORT Interpolation : public RefCounted<Interpolation> {  
61     public:  
62         virtual ~Interpolation() = default;  
63  
64         virtual void Interpolate(int iteration, double fraction) = 0;  
--  
19 // The Interpolation class is an abstract class representing an animation effect  
20 // between two keyframe values for the same property (CSS property, SVG  
21 // attribute, etc), for example animating the CSS property 'left' from '100px' ▶  
22 // to '200px'.  
--  
..
```

- Dynamic type erasure - data and instruction cache misses
- Requires testing combinations of concrete classes

Apply the new value

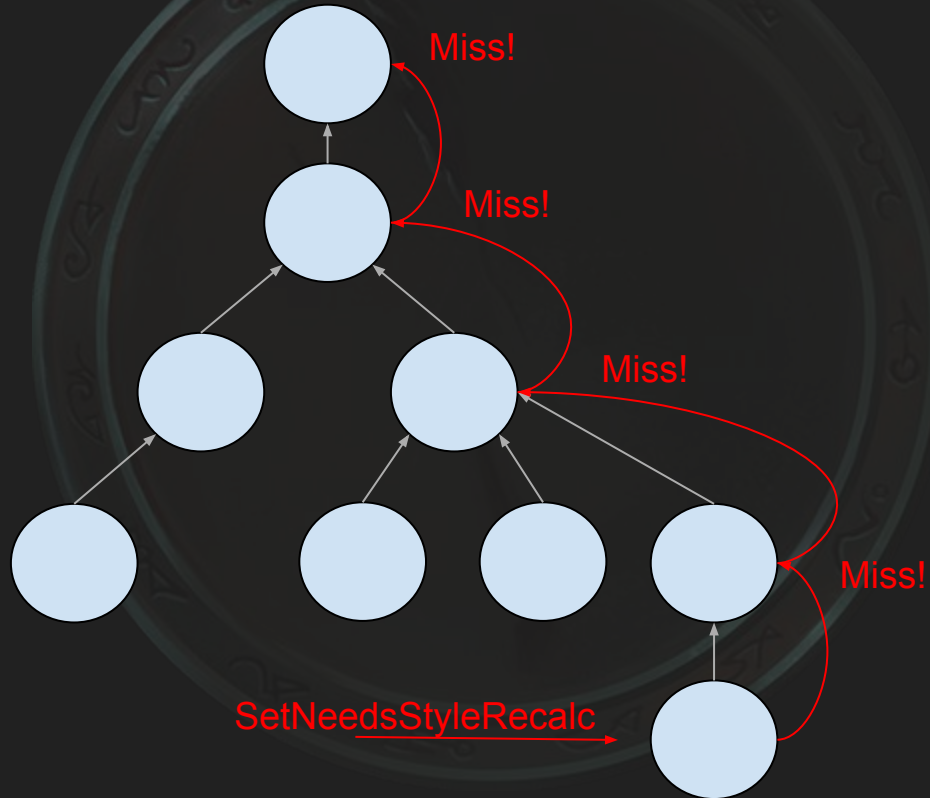
- Coupling systems - Animations and Style solving
- Unclear lifetime - who “owns” the Element
- Guaranteed cache misses

```
207     if (changed) {
208         target_ ->SetNeedsAnimationStyleRecalc(); ≤ 1ms elapsed
209         if (RuntimeEnabledFeatures::WebAnimationsSVGEnabled() &&
210             target_ ->IsSVGElement())
211             ToSVGElement(*target_).SetWebAnimationsPending();
212     }

2448 void Element::SetNeedsAnimationStyleRecalc() {
2449     if (GetStyleChangeType() != kNoStyleChange)
2450         return;
2451     SetNeedsStyleRecalc(kLocalStyleChange, StyleChangeReasonForTracing::Create(
2452         StyleChangeReason::kAnimation));
2453     SetAnimationStyleChange(true);
2454 }
2455 }
```

Walks up the DOM tree!

SetNeedsStyleRecalc



Recap

- We used more than **6** non-trivial classes
- Objects contain smart **pointers** to other objects
- Interpolation uses **abstract classes** to handle different property types
- CSS Animations directly **reach out** to other systems - coupling
 - Calling events
 - Setting the value in the DOM Element
 - How is the lifetime of Elements synchronized?

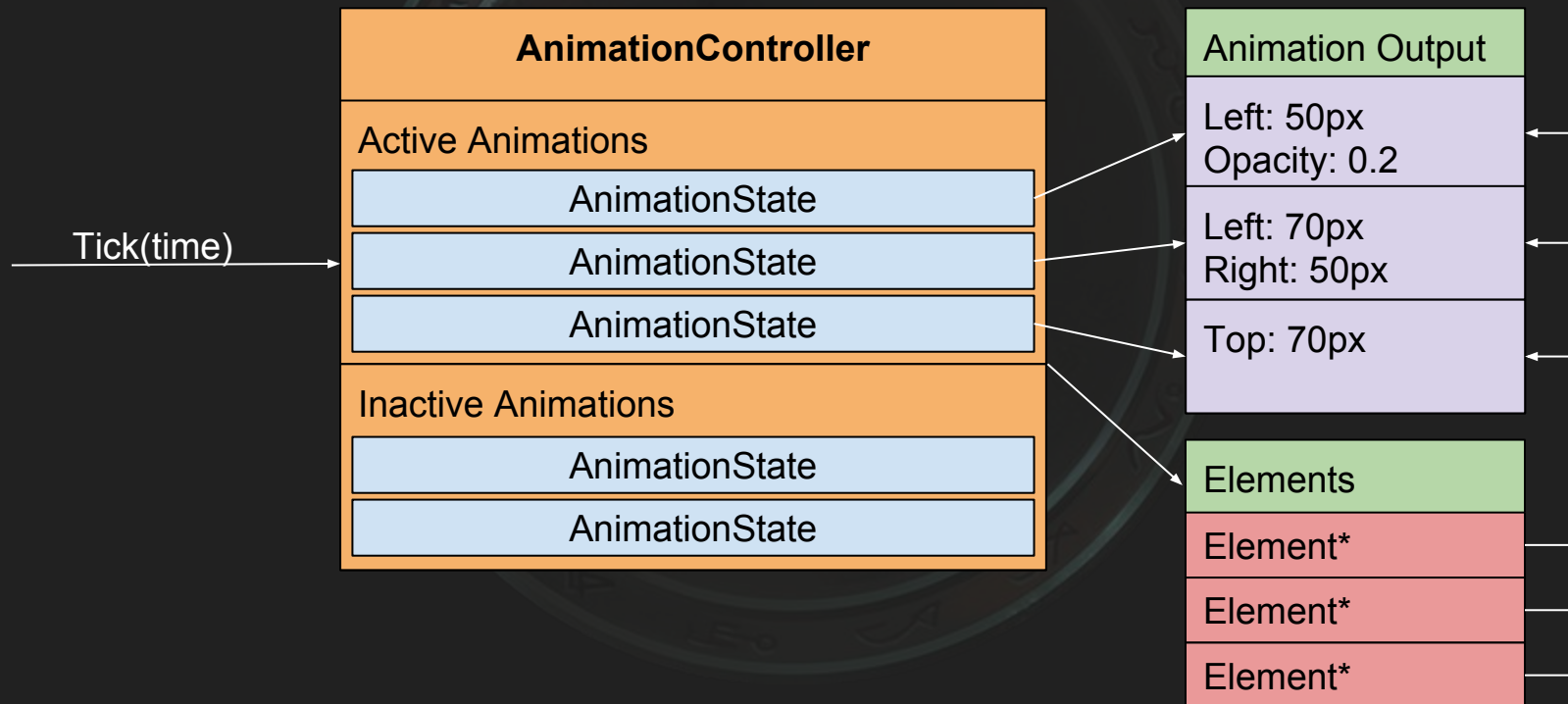


Let's try data-oriented design

Back to the drawing board

- Animation data operations
 - Tick (Update) -> **99.9%**
 - Add
 - Remove
 - Pause
 - ...
- Animation Tick Input
 - Animation definition
 - Time
- Animation Tick Output
 - Changed properties
 - New property values
 - Who owns the new values
- Design for **many animations**

The AnimationController



Go flat!

```
43 struct AnimationStateCommon
44 {
45     AnimationId Id;
46     mono_clock::time_point::seconds StartTime;
47     mono_clock::time_point::seconds PauseTime;
48     mono_clock::duration::seconds Duration;
49     mono_clock::duration::seconds Delay;
50     Optional<mono_clock::time_point::seconds> ScheduledPauseTime;
51     AnimationIterationCount::Value Iterations;
52     AnimationFillMode::Type FillMode;
53     AnimationDirection::Type Direction;
54     AnimationTimingFunction::Timing Timing;
55     AnimationPlayState::Type PlayState;
56     float IterationsPassed = 0.f;
57     float PlaybackRate = 1.0f;
58 };
```

Note: Some read-only data gets **duplicated** across multiple instances

Avoid type erasure

```
67  template<typename T>  
68  struct AnimationStateProperty : public AnimationState  
69  {  
70      AnimatedDefinitionFrames<T> Keyframes;  
71  };
```

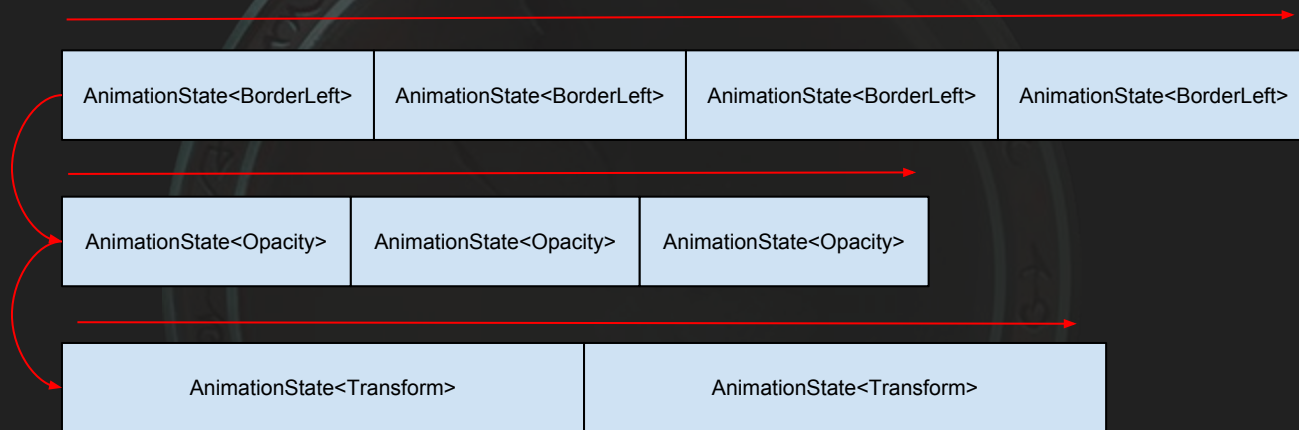
Per-property vector for every Animation type!

```
// -- Auto-generated -- /  
CSSVector<AnimationStateProperty<BorderWidth>> m_BorderTopWidthActiveAnimState;  
CSSVector<AnimationStateProperty<BorderWidth>> m_BorderLeftWidthActiveAnimState;  
// ... //  
CSSVector<AnimationStateProperty<ZIndex>> m_ZIndexActiveAnimState
```

Note: We know every needed type at compile time, the vector declarations are auto-generated

Ticking animations

- Iterate over all vectors



- Use **implementation-level** templates (in the .cpp file)

```
670     template<css::PropertyTypes PropType>
671     AnimationRunningState TickAnimation(mono_clock::time_point::seconds now,
672     AnimationStateProperty<typename css::PropertyValue<PropType>::type_t>& state)
673     {
```

Avoiding branches

- Keep lists per-boolean “flag”
 - Similar to database tables - sometimes called that way in DoD literature
- Separate **Active** and **Inactive** animations
 - Active are currently running
 - But can be stopped from API
 - Inactive are finished
 - But can start from API
- Avoid “*if (isActive)*” !
- Tough to do for every bool, prioritize according to branch predictor chance

A little bit of code

```
template<css::PropertyTypes_PropType>
AnimationRunningState TickAnimation(mono_clock::time_point::seconds now,
] AnimationStateProperty<typename css::PropertyValue<PropType>::type_t>& state)
{
    using Type = typename css::PropertyValue<PropType>::type_t;

    AnimationRunningState transition;
    const auto t = CalculateAnimationPoint(now, state, transition);
    assert(!std::isnan(t));

    const typename AnimatedDefinitionFrames<Type>::Frame* from = nullptr;
    const typename AnimatedDefinitionFrames<Type>::Frame* to = nullptr;

    size_t firstFrameIndex;
    auto interpolator = DetermineKeyFrameInterval(t, state, from, to, firstFrameIndex);

    interpolator = ApplyEase(interpolator, state.Timing, state.Duration);

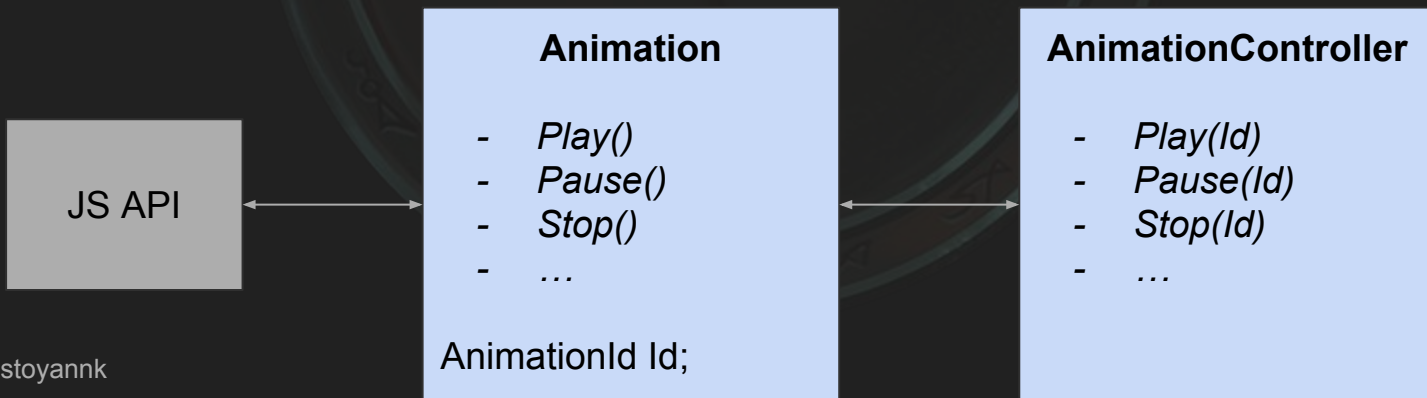
    const auto newValue = GetInterpolatedValue(state,
        firstFrameIndex,
        interpolator,
        from->Value,
        to->Value);

    state.Output->template SetValue<Type, PropType>(newValue);

    return transition;
}
```


Adding an API - Controlling Animations

- The API requires having an “Animation” object
 - `play()`
 - `pause()`
 - `playbackRate()`
- But we have no “Animation” object?!
- An Animation is simply a **handle** to a bunch of data!
- **AnimationId** (unsigned int) wrapped in a JS-accessible C++ object



Implementing the DOM API cont.

- AnimationController implements all the data modifications
- “Animation” uses the AnimationId as a simple handle

```
135 void PauseAnimation(AnimationId animationId);
136 void PlayAnimation(AnimationId animationId);
137 void PlayFromTo(AnimationId animationId,
138               mono_clock::duration::milliseconds playTime,
139               mono_clock::duration::milliseconds pauseTime);
140 void SetAnimationSeekTime(AnimationId animationId, mono_clock::duration::milliseconds seekTime);
141 mono_clock::duration::milliseconds GetAnimationSeekTime(AnimationId animationId);
142 void SetAnimationPlaybackRate(AnimationId animationId, float playbackRate);
143 float GetAnimationPlaybackRate(AnimationId animationId);
144 void ReverseAnimation(AnimationId animationId);
```

Analogous concepts between OOP and DoD

OOP	DoD
<code>blink::Animation</code> inheriting 6 classes	<code>AnimationState</code> templated struct
References to <code>Keyframe</code> data	Read-only duplicates of the <code>Keyframe</code> data
List of dynamically allocated <code>Interpolations</code>	Vectors per-property
Boolean flags for “activeness”	Different tables (vectors) according to flag
Inherit <code>blink::ActiveScriptWrappable</code>	Animation interface with <code>Id</code> handle
Output new property value to <code>Element</code>	Output to tables of new values
Mark <code>Element</code> hierarchy (DOM sub-trees) for styling	List of modified <code>Elements</code>

Key points

- Keep data flat
 - Maximise cache usage
 - No RTTI
 - Amortized dynamic allocations
 - Some read-only duplication improves performance and readability
- Existence-based predication
 - Reduce branching
 - Apply the same operation on a whole table
- Id-based handles
 - No pointers
 - Allow us to rearrange internal memory
- Table-based output
 - No external dependencies
 - Easy to reason about the flow



Analysis

Performance analysis

	OOP	DoD
Animation Tick time average	6.833 ms	1.116 ms

DoD Animations are **6.12x** faster



Scalability

- Issues multithreading OOP chromium Animations
 - Collections getting modified during iteration
 - Event delegates
 - Marking Nodes for re-style
- Solutions for the OOP case
 - Carefully re-work each data dependency
- Issues multithreading DoD Animations
 - Moving AnimationStates to “inactive” (table modification from multiple threads)
 - Building list of modified Nodes (vector push_back across multiple threads)
- Solutions in the DoD case
 - Each task/job/thread keeps a private table of modified nodes & new inactive anims
 - Join merges the tables
 - Classic fork-join

Testability analysis

- The OOP case
 - Needs mocking the main input - animation definitions
 - Needs mocking at least a dozen classes
 - Needs building a complete mock DOM tree - to test the “needs re-style from animation logic”
 - Combinatorial explosion of internal state and code-paths
 - Asserting correct state is difficult - multiple output points
- The DoD case
 - Needs mocking the input - animation definitions
 - Needs mocking a list of Nodes, complete DOM tree is not needed
 - AnimationController is self-contained
 - Asserting correct state is easy - walk over the output tables and check

Modifiability analysis

- OOP
 - Very tough to change base classes
 - Very hard to reason about the consequences
 - Data tends to “harden”
 - Hassle to move fields around becomes too big
 - Nonoptimal data layouts stick around
 - Shared object lifetime management issues
 - Hidden and often fragile order of destruction
 - Easy to do “quick” changes
- DoD
 - Change input/output -> requires change in System “before”/”after” in pipeline
 - Implementation changes - local
 - Can experiment with data layout
 - Handles mitigate potential lifetime issues

Downsides of DoD

- Correct data separation can be hard
 - Especially before you know the problem very well
- Existence-based predication is not always feasible (or easy)
 - Think adding a bool to a class VS moving data across arrays
 - Too many booleans is a symptom - think again about the problem
- “Quick” modifications can be tough
 - OOP allows to “just add” a member, accessor, call
 - More discipline is needed to keep the benefits of DoD
- You might have to unlearn a thing or two
 - The beginning is tough
- The language is not always your friend

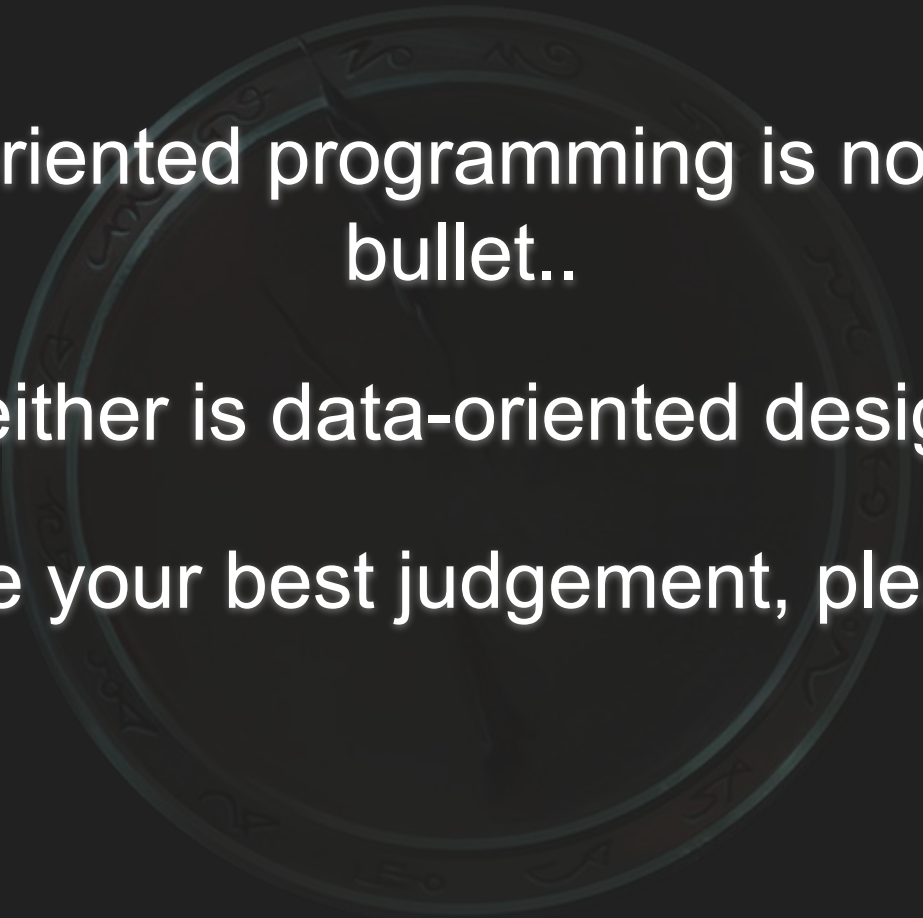


What to keep from OOP?

- Sometimes we have no choice
 - Third-party libraries
 - IDL requirements
- Simple structs with simple methods are perfectly fine
- Polymorphism & Interfaces have to be kept under control
 - Client-facing APIs
 - Component high-level interface
 - IMO more convenient than C function pointer structs
- Remember - C++ has great facilities for static polymorphism
 - Can be done through templates
 - .. or simply include the right “impl” according to platform/build options

Changes in C++ to better support DoD

- Allow new memory layout schemes for object arrays
 - Structure Of Arrays (SOA) / Array Of Structures (AOS)
 - Components layout, preserving classic C++ object access semantics
 - **Kinda doable** now, but requires a lot of custom code
 - We do it to awesome effect, but sooooo tough
 - Alas tough to get in core
- Ranges look really exciting
- Relax requirements for `unordered_map/unordered_set` (or define new ones)
 - Internal linked list does too many allocations & potential cache misses
 - Standard hashmap/set with **open addressing**



Object-oriented programming is not a silver
bullet..

..neither is data-oriented design..

..use your best judgement, please.

References

- [“Data-Oriented Design and C++”](#), Mike Acton, CppCon 2014
- [“Pitfalls of Object Oriented Programming”](#), Tony Albrecht
- [“Introduction to Data-Oriented Design”](#), Daniel Collin
- [“Data-Oriented Design”](#), Richard Fabian
- [“Data-Oriented Design \(Or Why You Might Be Shooting Yourself in The Foot With OOP\)”](#), Noel Llopis
- [“OOP != classes, but may == DOD”](#), roathe.com
- [“Data Oriented Design Resources”](#), Daniele Bartolini
- <https://stoyannk.wordpress.com/>