# The Future of Scene Description on God of War

Koray Hagen

Santa Monica Studio

Hello everyone, my name is Koray Hagen …

## Background

Senior Programmer

Tools and Core Technology

Santa Monica Studio

Santa Monica Studio

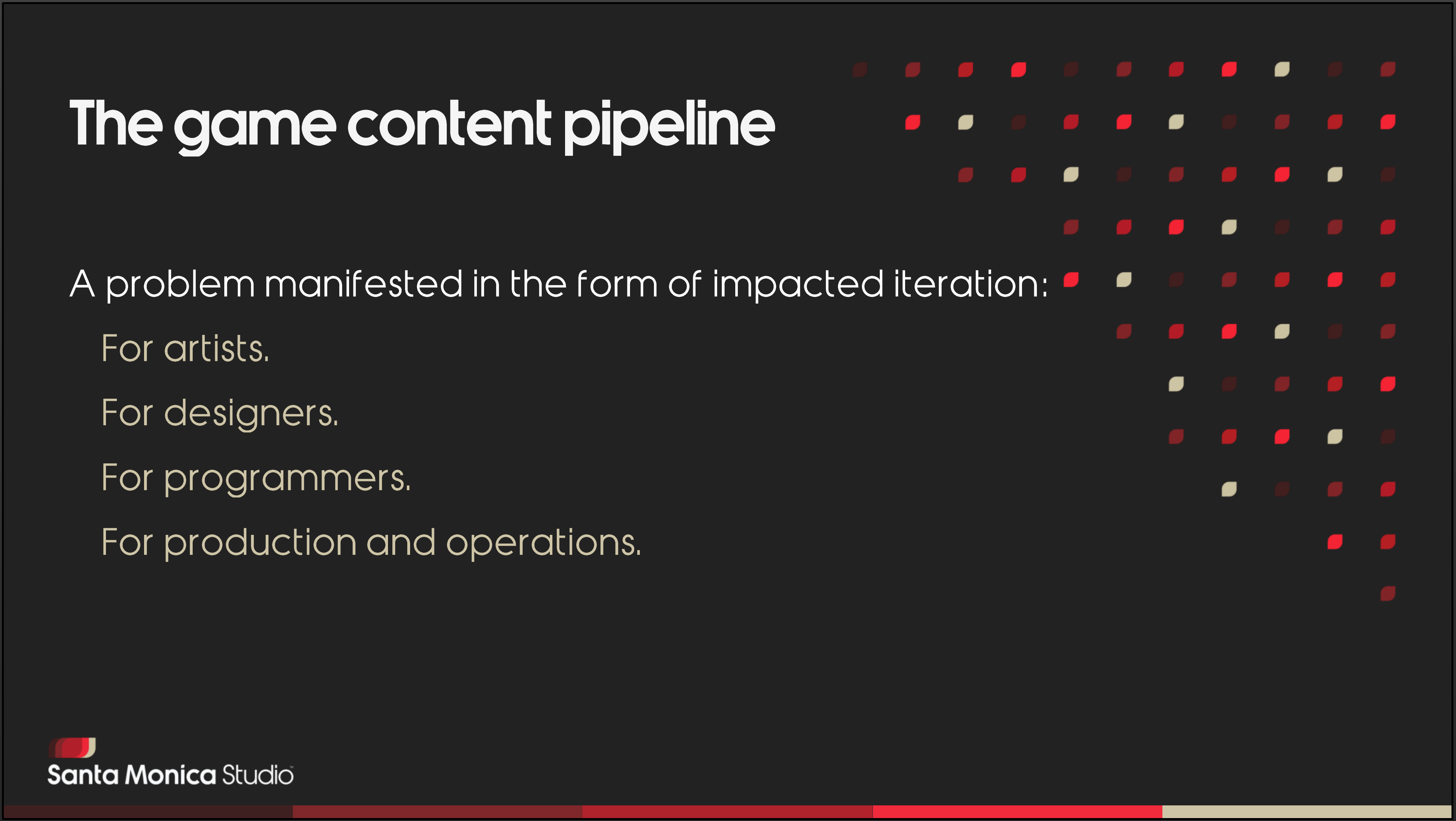
And I work at Santa Monica Studio on the development team for God of War contributing as a senior programmer on many of our core technology efforts. Today we're going to be talking our studio's current thinking and philosophy surrounding the design of data in games.

Santa Monica Studio

Historically, God of War games have dealt with much less data than our most recent title.

## The final game required an extraordinary amount of data.

In fact, having worked on the game for the past five years of my life I can tell you that we had huge scalability problems across the board when it came to managing the sheer amount of content that was required to produce the final game.



## The game content pipeline

A problem manifested in the form of impacted iteration:

- For artists.
- For designers.
- For programmers.
- For production and operations.

Santa Monica Studio

This scalability problem manifested itself in the form of impacted iteration times for everyone who was actively contributing to the game's development.

## The game content pipeline

Domain of game data:

- Models.
- Textures.
- Materials.
- Animation.
- Audio.
- … It's unending.

MAYA

ZBRUSH

Wwise
mpowers the audio community

SUBSTANCE

Custom Editor

Santa Monica Studio

So what's the cause of this problem? As a programmer contributing to the pipelining of this data around our studio, I can tell you that content in games is quite heterogenous – meaning that it originates from a variety of tools and workflows for different purposes.

## The game content pipeline

Data characteristics:

- Intrinsic differences:
  - Format, disk-space impact, quantity.
- Relationships:
  - Hierarchical, referential, dependencies.

Santa Monica Studio

But the differences don't end there. If you were to look at any particular piece of data in isolation, it has unique intrinsic characteristics such as its format, disk-space impact, even quantity of usage. In games our data can also often be hierarchical, or at least contain referencing semantics to other data – which further complicates their role in the pipeline.

## The game content pipeline

Game data duality:

Disparate requirements for source data and run-time data.

$\neq$

Santa Monica Studio

Games also have this strange data duality where the source data we ask our content creators to make isn't what we end up shipping to our players. We have to transform that data into something useable by our engines and hardware, which has the cost of time and complexity in the overall pipeline.

## The game content pipeline

Game data duality:

Disparate requirements for source data and run-time data.

God of War data breakdown:

A Dozen shipped PlayGos, comprised of …

Hundreds of Wads (custom binary format), comprised of …

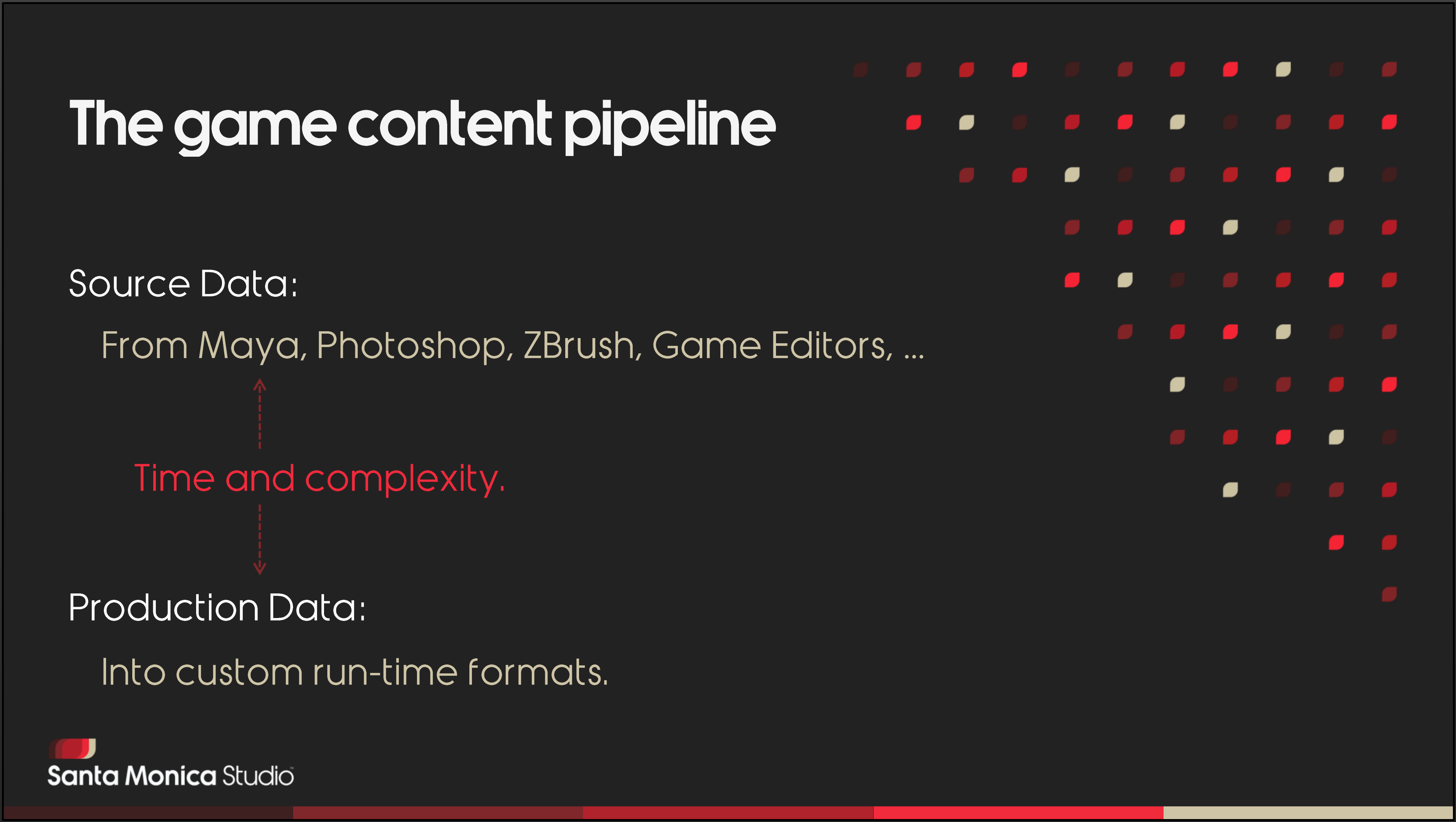
A million source assets.

Santa Monica Studio

Historically for our own studio, the source data and run-time data have nothing in common, especially their quantity . As an example, on God of War we shipped maybe a dozen PlayStation PlayGos which were comprised of hundreds wads -- our custom binary run-time format – and all of this originating from a million source assets.

## So where does that leave us?

So where does that leave us? Enumerating the effects of data in our games is a complex topic. And the role of our game’s pipeline is in marshalling that data from a developer’s workstation to a player’s hands.



## The game content pipeline

Source Data:

From Maya, Photoshop, ZBrush, Game Editors, …

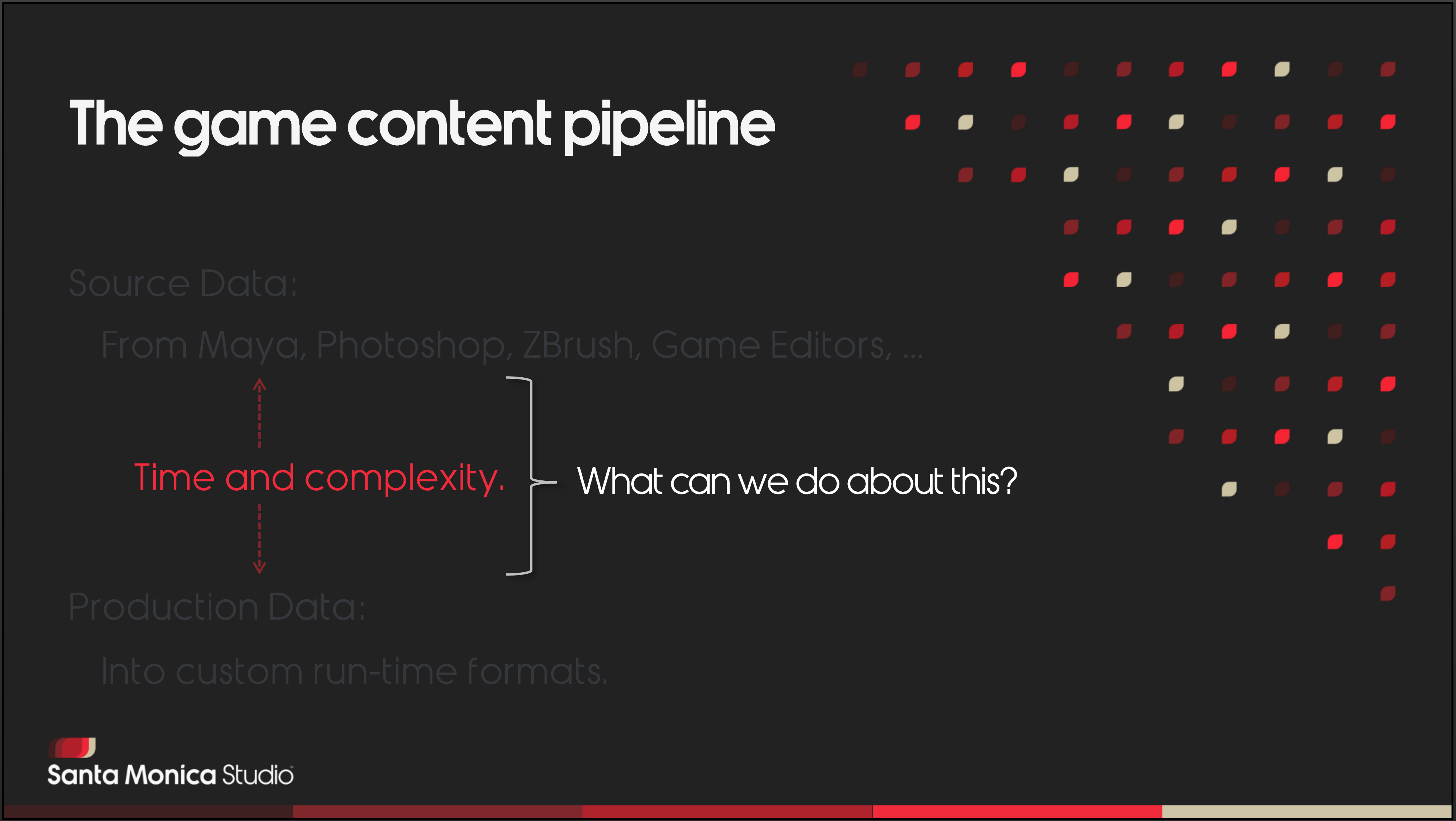
Time and complexity.

Production Data:

Into custom run-time formats.

Santa Monica Studio

Another way to look at it, is that the problem domain is about transforming our source data (that isn't meaningful to our players) into production data (which is). What we have seen in our own experience is that as our source data and production data diverge further and further away, the time and complexity gap to transform it continues to increase dramatically – ultimately impacting our ability to create the games we want.



## The game content pipeline

Source Data:

From Maya, Photoshop, ZBrush, Game Editors, …

Time and complexity.

What can we do about this?

Production Data:

Into custom run-time formats.

Santa Monica Studio

Like all large engineering challenges there are a lot of dimensions of optimization that can be applied to a problem space like this. But about 4-5 years ago when God of War's production began, we started to think about this problem in a new way.

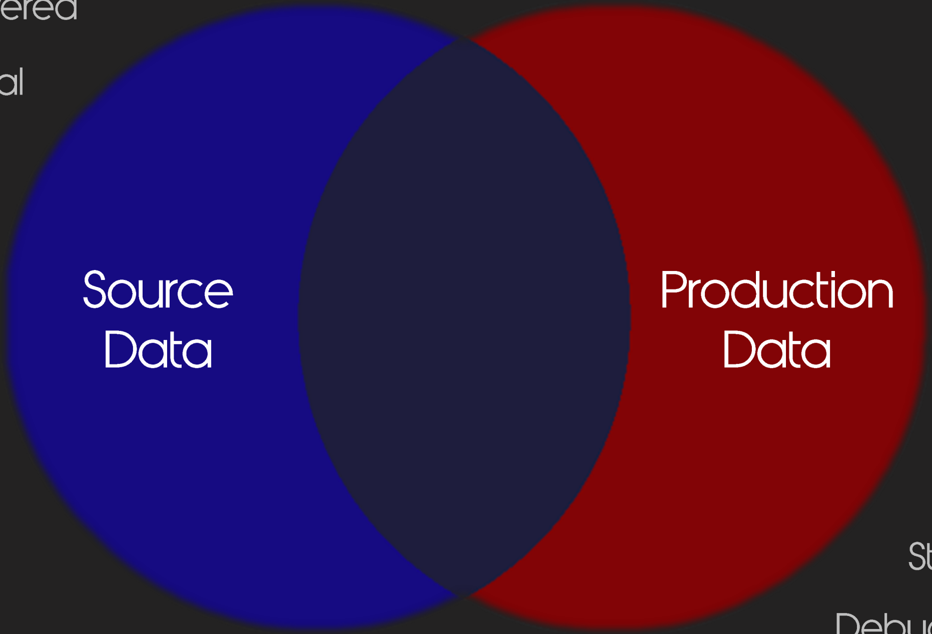
## The game content pipeline

Layered

Hierarchical

Persistence

Authoring



Strict layout

Linearization

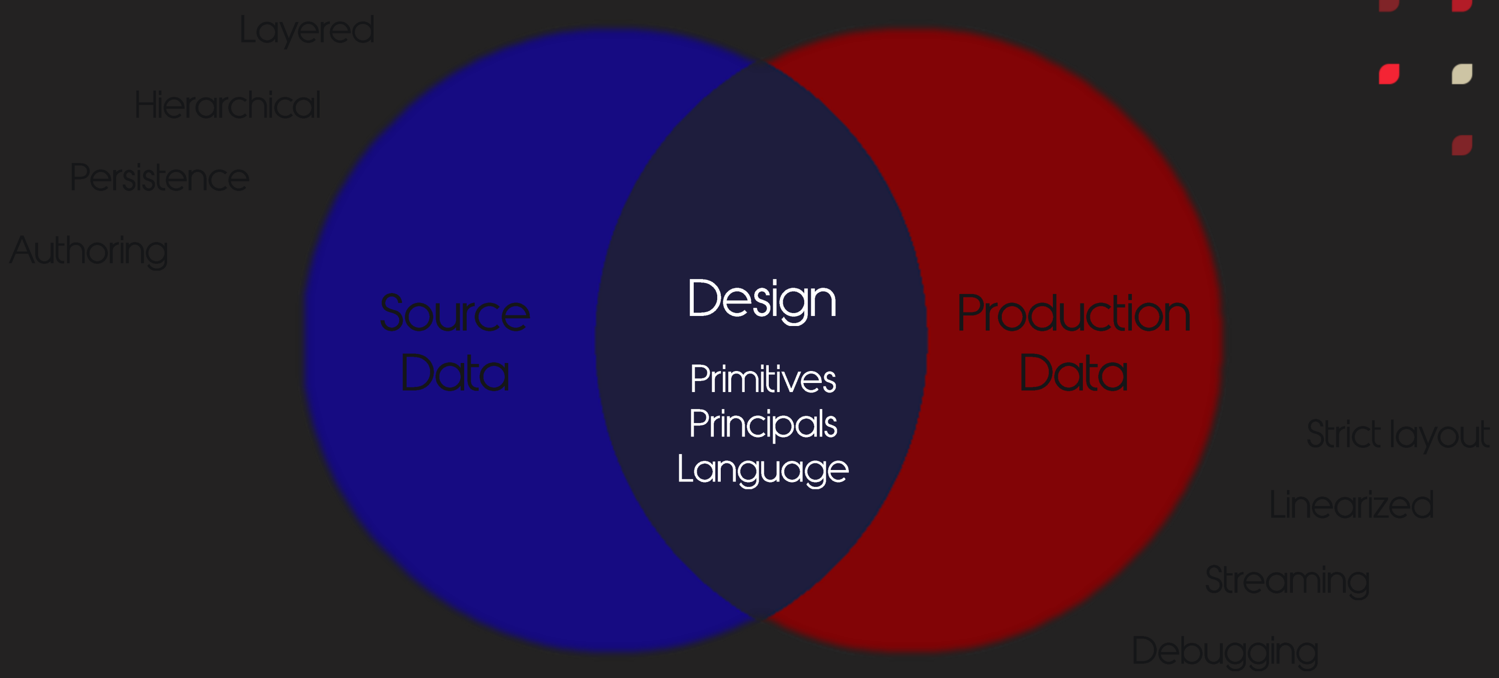
Streaming

Debugging

Santa Monica Studio

We started to think that our source data and production data didn't have to be a radically different in characteristics than we had thought in the past. In fact, to tackle this larger scalability problem we would have leverage the one characteristic they both had in common – they both required an explicit and sound design.

## The game content pipeline



Santa Monica Studio

And because they both required a sound design, that suggested that perhaps there was not only a common design language between them (in how we describe their structure and behavior), but also a common set of design primitives and principles from which they could be built. And this set of commonality could be tailored for the requirements of data in real-time games.

## An avenue of inspiration

Inspired by the Universal Scene Description philosophy:

- Content creators author USD assets.
- Hydra framework renders USD directly.

Over the years we have been particularly inspired by efforts that are occurring inside the computer graphics industry for film to tackle a similar problem, which has been coined as "scene description." In particular, the philosophy of Universal Scene Description resonated with us.

## An avenue of inspiration

Inspired by the Universal Scene Description philosophy:

- Content creators author USD assets.
- Hydra framework renders USD directly.

Author in USD. Render in USD.

## An avenue of inspiration

Inspired by the Universal Scene Description philosophy:

- Content creators author USD assets.
- Hydra framework renders USD directly.

Can this philosophy be applied to games?

It powerfully suggests USD as a core ecosystem, meant to address the full scope of the end-to-end pipeline. So there's a simple question – could there be a similar philosophy and ecosystem for real-time games? What would be the requirements? And what problems do we care about?

Towards that end, today we will explore the story of our thought process alongside a technology and design showcase. My goal is to present how we reconciled the familiar development constraints of creating a large game, while staying true to our vision for improving God of War's pipeline and technology.

With that said, let's get started.

## Three stories

Past:

Determinism and patching.

Present:

A view after years of investment.

Future:

The opportunities that lie ahead.

Santa Monica Studio

We’re going to cover just three stories. The first story is about our past, and how the everyday stresses of delivering a high quality game product influenced our thinking and desire to tackle an interesting and important problem with early experiments.

## Three stories

Past:

Determinism and patching.

Present:

A view after years of investment.

Future:

The opportunities that lie ahead.

Santa Monica Studio

The second story is about the present, and where our design and engineering philosophy have taken us.

## Three stories

Past:

Determinism and patching.

Present:

A view after years of investment.

Future:

The opportunities that lie ahead.

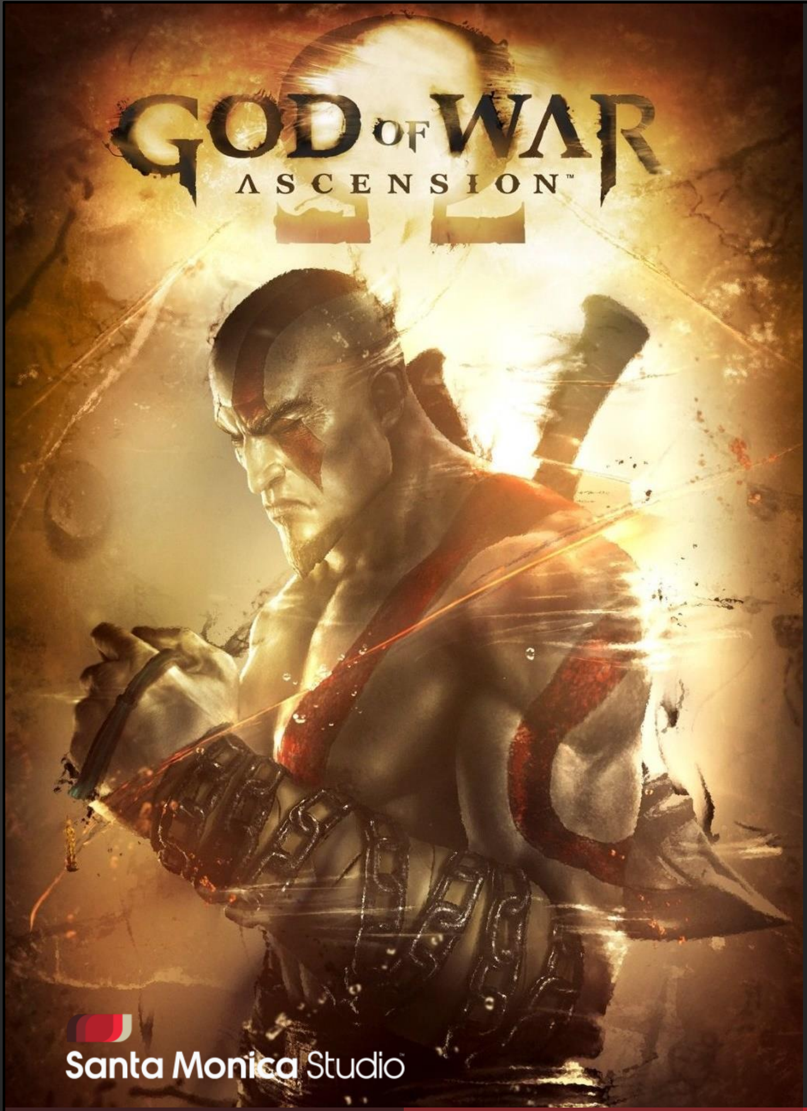
Santa Monica Studio

And the last story, is about our future. Our message is that there is so much opportunity in this space to explore, and our own lived experience tells us that we're just at the beginning.

## Exploring the past.

Santa Monica Studio

So let’s start at the beginning.

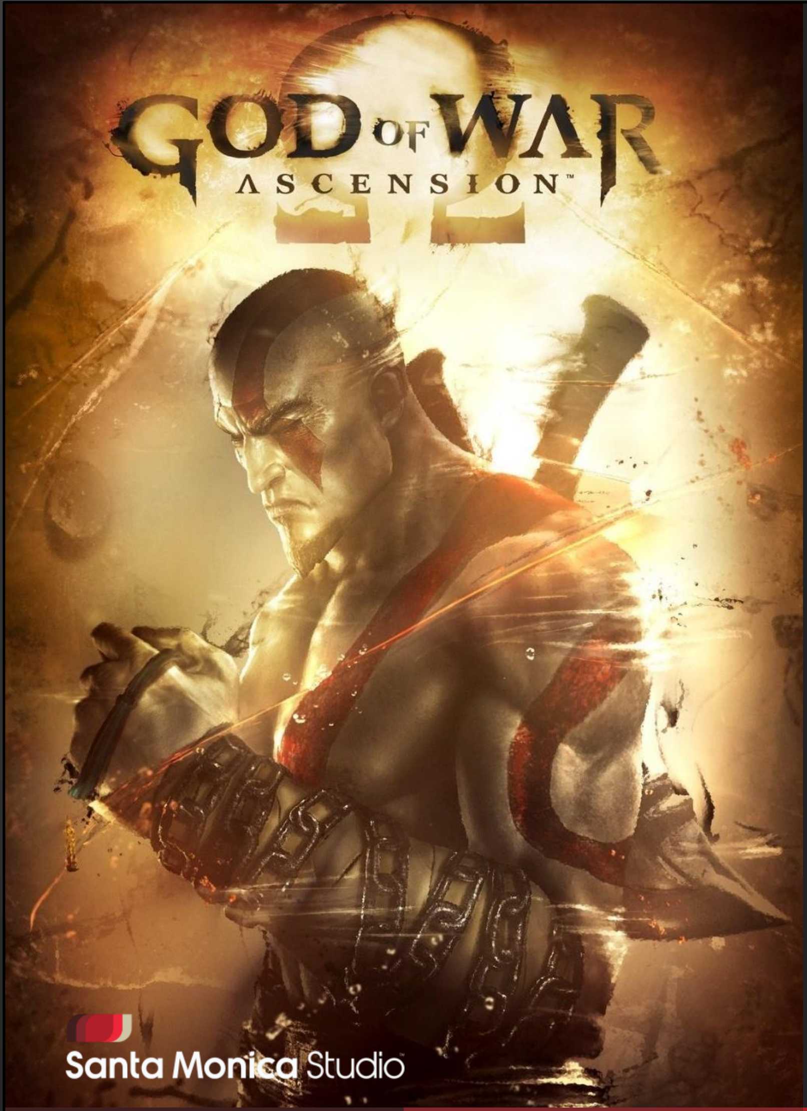


## A common enemy

God of War Ascension was ready to ship:

- An engineer was tasked with creating a patch for the game.

When God of War: Ascension was close to finishing development, an engineer on our team was tasked with testing our patching infrastructure for the live game.  The test was simple, make a small, inconsequential change; create a patch; measure the delta.



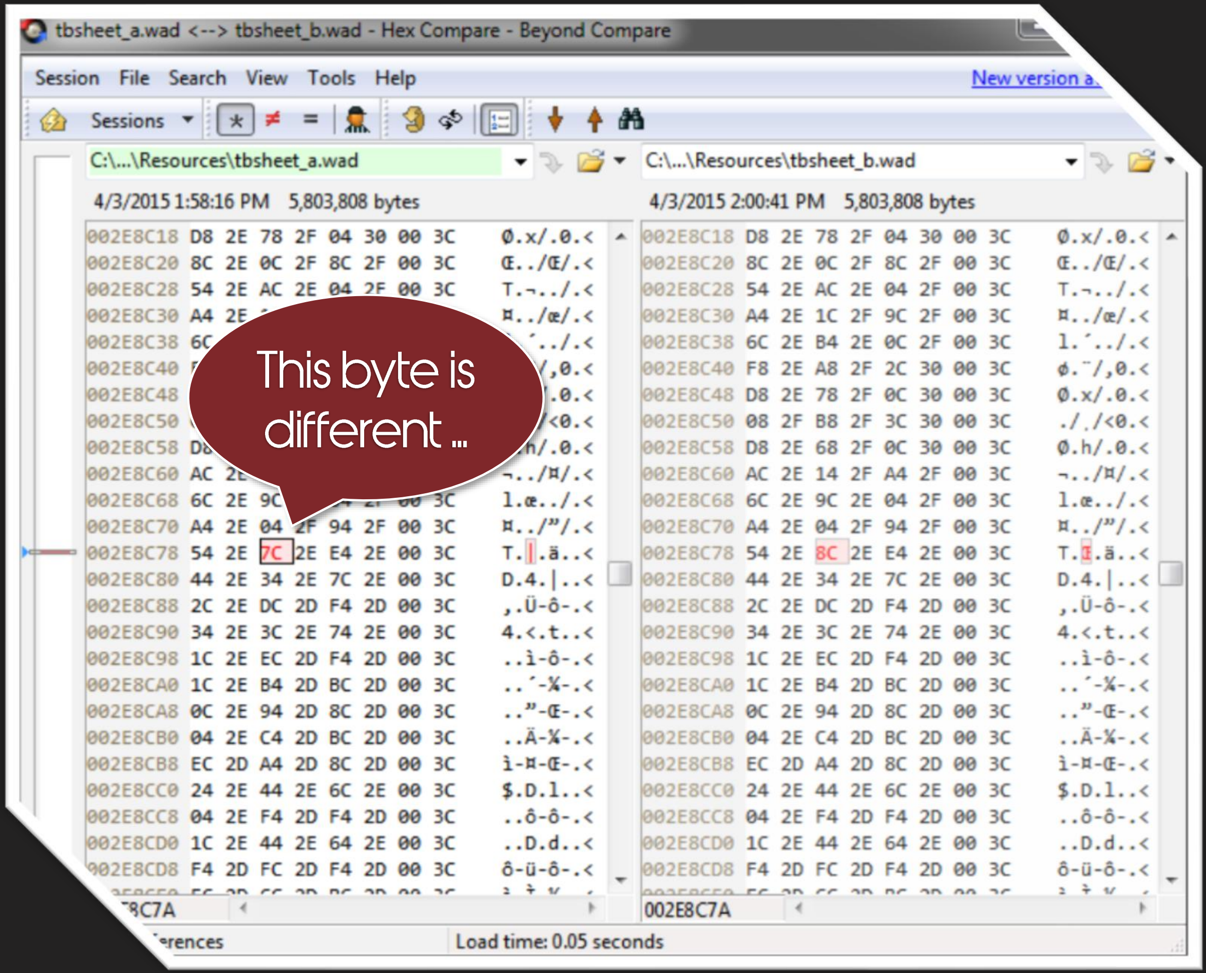
## A common enemy

God of War Ascension was ready to ship:

- An engineer was tasked with creating a patch for the game.

- The patch was the size of the entire disk.

We were surprised to find out that the patch was the size of the entire game disk.



tbsheet_a.wad <--> tbsheet_b.wad - Hex Compare - Beyond Compare

Session File Search View Tools Help — New version a

Sessions

C:\...\Resources\tbsheet_a.wad — 4/3/2015 1:58:16 PM 5,803,808 bytes

```
002E8C18 D8 2E 78 2F 04 30 00 3C   Ø.x/.0.<
002E8C20 8C 2E 0C 2F 8C 2F 00 3C   Œ../Œ/.<
002E8C28 54 2E AC 2E 04 2F 00 3C   T.¬../.<
002E8C30 A4 2E                     ¤../œ/.<
002E8C38 6C                        ´../.<
002E8C40                           /,0.<
002E8C48                           .0.<
002E8C50                           <0.<
002E8C58 D8                        h/.0.<
002E8C60 AC 2E                     ¬../¤/.<
002E8C68 6C 2E 9C             3C   l.œ../.<
002E8C70 A4 2E 04 2F 94 2F 00 3C   ¤../”/.<
002E8C78 54 2E 7C 2E E4 2E 00 3C   T.|.ä..<
002E8C80 44 2E 34 2E 7C 2E 00 3C   D.4.|..<
002E8C88 2C 2E DC 2D F4 2D 00 3C   ,.Ü-ô-.<
002E8C90 34 2E 3C 2E 74 2E 00 3C   4.<.t..<
002E8C98 1C 2E EC 2D F4 2D 00 3C   ..ì-ô-.<
002E8CA0 1C 2E B4 2D BC 2D 00 3C   ..´-¼-.<
002E8CA8 0C 2E 94 2D 8C 2D 00 3C   ..”-Œ-.<
002E8CB0 04 2E C4 2D BC 2D 00 3C   ..Ä-¼-.<
002E8CB8 EC 2D A4 2D 8C 2D 00 3C   ì-¤-Œ-.<
002E8CC0 24 2E 44 2E 6C 2E 00 3C   $.D.l..<
002E8CC8 04 2E F4 2D F4 2D 00 3C   ..ô-ô-.<
002E8CD0 1C 2E 44 2E 64 2E 00 3C   ..D.d..<
002E8CD8 F4 2D FC 2D F4 2D 00 3C   ô-ü-ô-.<
```

8C7A

erences — Load time: 0.05 seconds

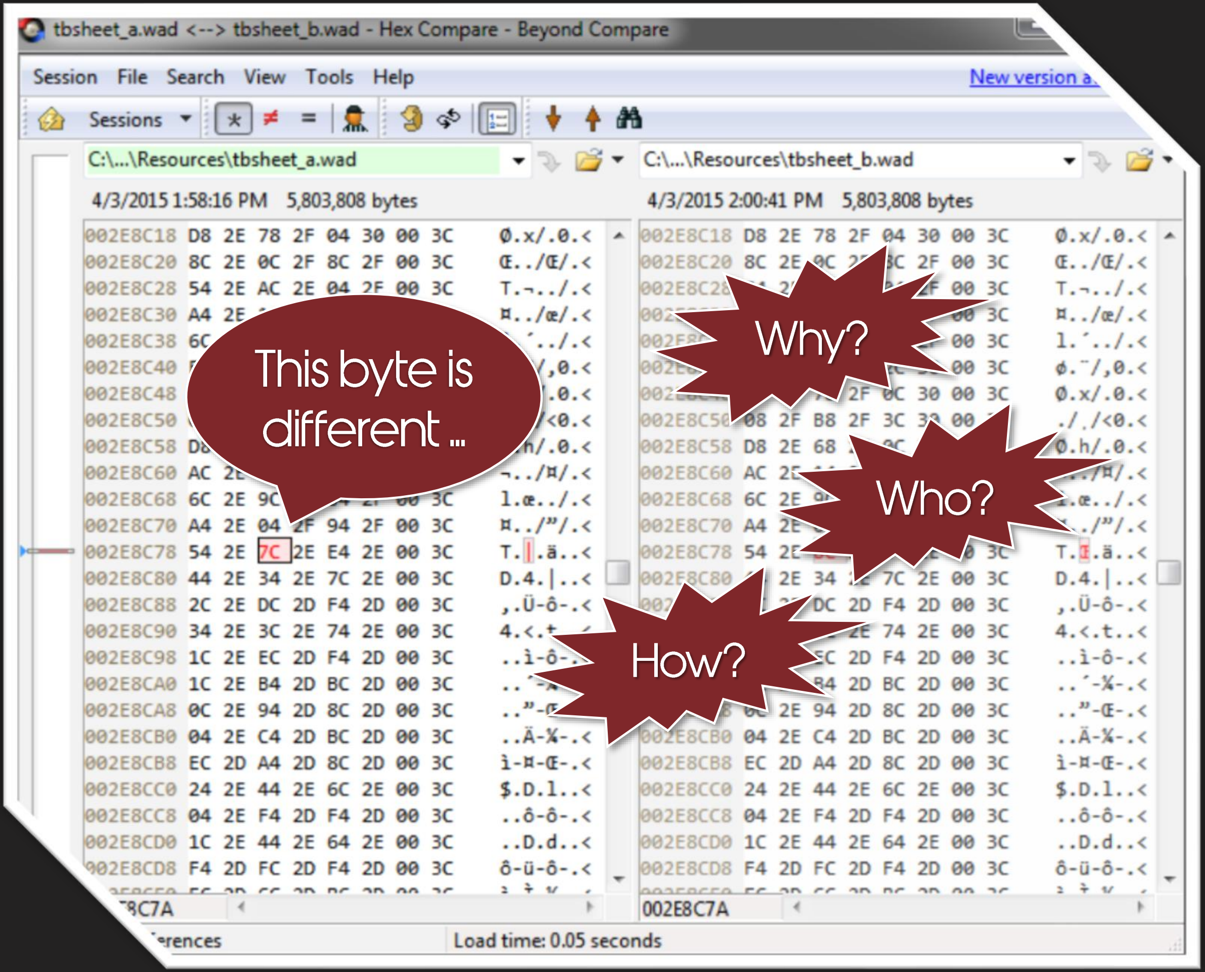
C:\...\Resources\tbsheet_b.wad — 4/3/2015 2:00:41 PM 5,803,808 bytes

```
002E8C18 D8 2E 78 2F 04 30 00 3C   Ø.x/.0.<
002E8C20 8C 2E 0C 2F 8C 2F 00 3C   Œ../Œ/.<
002E8C28 54 2E AC 2E 04 2F 00 3C   T.¬../.<
002E8C30 A4 2E 1C 2F 9C 2F 00 3C   ¤../œ/.<
002E8C38 6C 2E B4 2E 0C 2F 00 3C   l.´../.<
002E8C40 F8 2E A8 2F 2C 30 00 3C   ø.¨/,0.<
002E8C48 D8 2E 78 2F 0C 30 00 3C   Ø.x/.0.<
002E8C50 08 2F B8 2F 3C 30 00 3C   ./¸/<0.<
002E8C58 D8 2E 68 2F 0C 30 00 3C   Ø.h/.0.<
002E8C60 AC 2E 14 2F A4 2F 00 3C   ¬../¤/.<
002E8C68 6C 2E 9C 2E 04 2F 00 3C   l.œ../.<
002E8C70 A4 2E 04 2F 94 2F 00 3C   ¤../”/.<
002E8C78 54 2E 8C 2E E4 2E 00 3C   T.Œ.ä..<
002E8C80 44 2E 34 2E 7C 2E 00 3C   D.4.|..<
002E8C88 2C 2E DC 2D F4 2D 00 3C   ,.Ü-ô-.<
002E8C90 34 2E 3C 2E 74 2E 00 3C   4.<.t..<
002E8C98 1C 2E EC 2D F4 2D 00 3C   ..ì-ô-.<
002E8CA0 1C 2E B4 2D BC 2D 00 3C   ..´-¼-.<
002E8CA8 0C 2E 94 2D 8C 2D 00 3C   ..”-Œ-.<
002E8CB0 04 2E C4 2D BC 2D 00 3C   ..Ä-¼-.<
002E8CB8 EC 2D A4 2D 8C 2D 00 3C   ì-¤-Œ-.<
002E8CC0 24 2E 44 2E 6C 2E 00 3C   $.D.l..<
002E8CC8 04 2E F4 2D F4 2D 00 3C   ..ô-ô-.<
002E8CD0 1C 2E 44 2E 64 2E 00 3C   ..D.d..<
002E8CD8 F4 2D FC 2D F4 2D 00 3C   ô-ü-ô-.<
```

002E8C7A

Santa Monica Studio

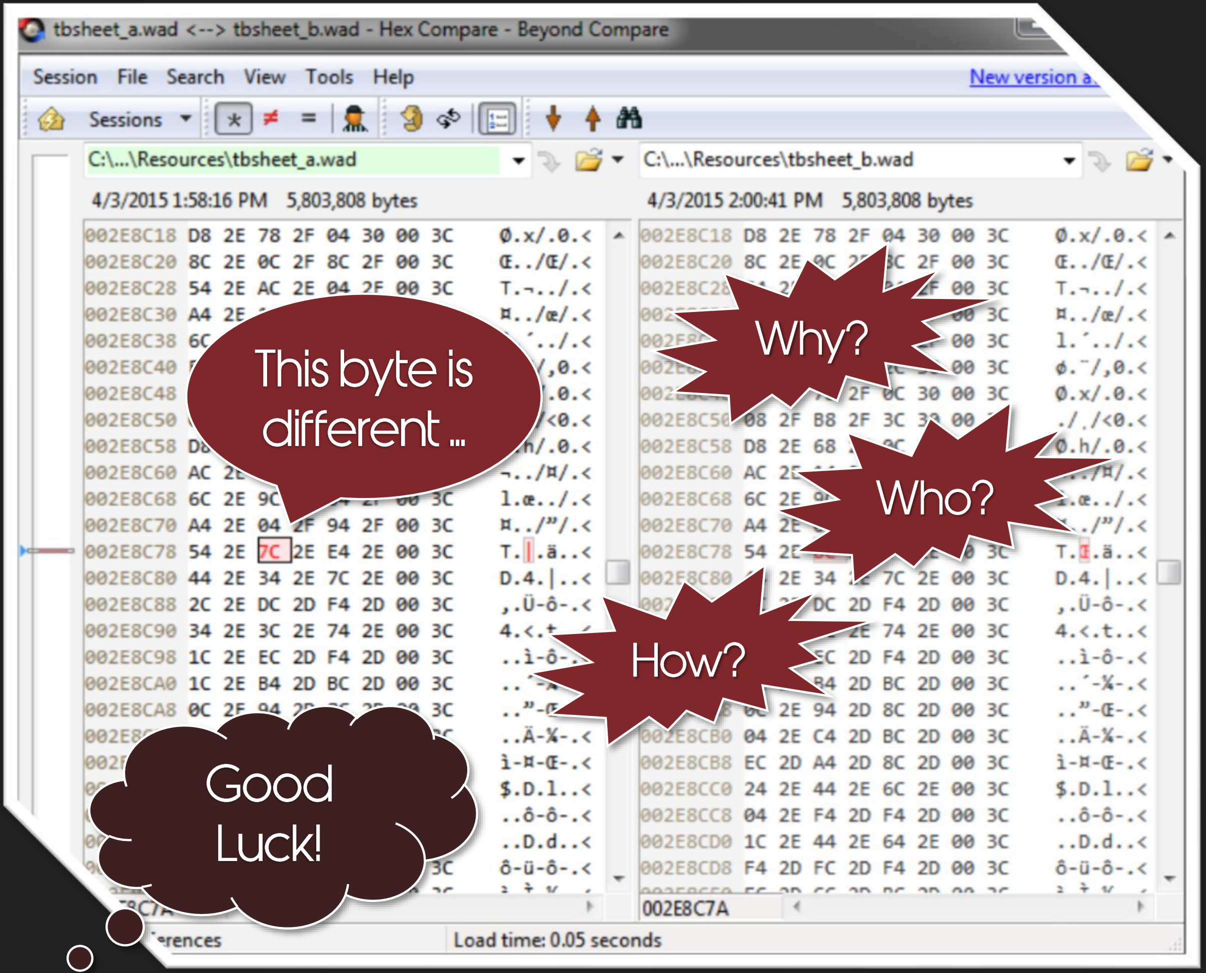
With that extreme of a result there was only one question:


tbsheet_a.wad <--> tbsheet_b.wad - Hex Compare - Beyond Compare
Session File Search View Tools Help
New version a
Sessions
This byte is different …
Why?
Who?
How?

```
C:\...\Resources\tbsheet_a.wad
4/3/2015 1:58:16 PM   5,803,808 bytes
002E8C18 D8 2E 78 2F 04 30 00 3C   Ø.x/.0.<
002E8C20 8C 2E 0C 2F 8C 2F 00 3C   Œ../Œ/.<
002E8C28 54 2E AC 2E 04 2F 00 3C   T.¬../.<
002E8C30 A4 2E                     ¤../æ/.<
002E8C38 6C                        ´../.<
002E8C40                           ,0.<
002E8C48                           .0.<
002E8C50                           <0.<
002E8C58 D8                        h/.0.<
002E8C60 AC 2E                     ¬../¤/.<
002E8C68 6C 2E 9C             3C   l.œ../.<
002E8C70 A4 2E 04 2F 94 2F 00 3C   ¤../”/.<
002E8C78 54 2E 7C 2E E4 2E 00 3C   T.|.ä..<
002E8C80 44 2E 34 2E 7C 2E 00 3C   D.4.|..<
002E8C88 2C 2E DC 2D F4 2D 00 3C   ,.Ü-ô-.<
002E8C90 34 2E 3C 2E 74 2E 00 3C   4.<.t..<
002E8C98 1C 2E EC 2D F4 2D 00 3C   ..ì-ô-.<
002E8CA0 1C 2E B4 2D BC 2D 00 3C   ..
002E8CA8 0C 2E 94 2D 8C 2D 00 3C   ..”-Œ
002E8CB0 04 2E C4 2D BC 2D 00 3C   ..Ä-¼-.<
002E8CB8 EC 2D A4 2D 8C 2D 00 3C   ì-¤-Œ-.<
002E8CC0 24 2E 44 2E 6C 2E 00 3C   $.D.l..<
002E8CC8 04 2E F4 2D F4 2D 00 3C   ..ô-ô-.<
002E8CD0 1C 2E 44 2E 64 2E 00 3C   ..D.d..<
002E8CD8 F4 2D FC 2D F4 2D 00 3C   ô-ü-ô-.<
8C7A
Load time: 0.05 seconds
```

```
C:\...\Resources\tbsheet_b.wad
4/3/2015 2:00:41 PM   5,803,808 bytes
002E8C18 D8 2E 78 2F 04 30 00 3C   Ø.x/.0.<
002E8C20 8C 2E 0C       2F 00 3C   Œ../Œ/.<
002E8C28                2F 00 3C   T.¬../.<
002E8C30                   00 3C   ¤../æ/.<
002E8C38                   00 3C   l.´../.<
002E8C40                   00 3C   ¢.¨/,0.<
002E8C48          2F 0C 30 00 3C   Ø.x/.0.<
002E8C50 08 2F B8 2F 3C 30 00      ./¸/<0.<
002E8C58 D8 2E 68                  Ø.h/.0.<
002E8C60 AC 2E                     ../¤/.<
002E8C68 6C 2E                     .œ../.<
002E8C70 A4 2E                     ../”/.<
002E8C78 54 2E                3C   T.Œ.ä..<
002E8C80    2E 34    7C 2E 00 3C   D.4.|..<
            DC 2D F4 2D 00 3C      ,.Ü-ô-.<
               74 2E 00 3C         4.<.t..<
            EC 2D F4 2D 00 3C      ..ì-ô-.<
            B4 2D BC 2D 00 3C      ..´-¼-.<
               2E 94 2D 8C 2D 00 3C   ..”-Œ-.<
002E8CB0 04 2E C4 2D BC 2D 00 3C   ..Ä-¼-.<
002E8CB8 EC 2D A4 2D 8C 2D 00 3C   ì-¤-Œ-.<
002E8CC0 24 2E 44 2E 6C 2E 00 3C   $.D.l..<
002E8CC8 04 2E F4 2D F4 2D 00 3C   ..ô-ô-.<
002E8CD0 1C 2E 44 2E 64 2E 00 3C   ..D.d..<
002E8CD8 F4 2D FC 2D F4 2D 00 3C   ô-ü-ô-.<
002E8C7A
```

What factors could have possibly led to this situation?



tbsheet_a.wad <--> tbsheet_b.wad - Hex Compare - Beyond Compare

Session File Search View Tools Help

C:\...\Resources\tbsheet_a.wad — 4/3/2015 1:58:16 PM 5,803,808 bytes

C:\...\Resources\tbsheet_b.wad — 4/3/2015 2:00:41 PM 5,803,808 bytes

```
002E8C18 D8 2E 78 2F 04 30 00 3C   Ø.x/.0.<
002E8C20 8C 2E 0C 2F 8C 2F 00 3C   Œ../Œ/.<
002E8C28 54 2E AC 2E 04 2F 00 3C   T.¬../.<
002E8C30 A4 2E                     ¤../æ/.<
002E8C70 A4 2E 04 2F 94 2F 00 3C   ¤../”/.<
002E8C78 54 2E 7C 2E E4 2E 00 3C   T.|.ä..<
002E8C80 44 2E 34 2E 7C 2E 00 3C   D.4.|..<
002E8C88 2C 2E DC 2D F4 2D 00 3C   ,.Ü-ô-.<
002E8C90 34 2E 3C 2E 74 2E 00 3C   4.<.t..<
002E8C98 1C 2E EC 2D F4 2D 00 3C   ..ì-ô-.<
002E8CA0 1C 2E B4 2D BC 2D 00 3C
002E8CA8 0C 2E 94
```

```
002E8C18 D8 2E 78 2F 04 30 00 3C   Ø.x/.0.<
002E8C50 08 2F B8 2F 3C 30 00      ./¸/<0.<
002E8C58 D8 2E 68                  Ø.h/.0.<
002E8CB0 04 2E C4 2D BC 2D 00 3C   ..Ä-¼-.<
002E8CB8 EC 2D A4 2D 8C 2D 00 3C   ì-¤-Œ-.<
002E8CC0 24 2E 44 2E 6C 2E 00 3C   $.D.l..<
002E8CC8 04 2E F4 2D F4 2D 00 3C   ..ô-ô-.<
002E8CD0 1C 2E 44 2E 64 2E 00 3C   ..D.d..<
002E8CD8 F4 2D FC 2D F4 2D 00 3C   ô-ü-ô-.<
```

002E8C7A

Load time: 0.05 seconds

Santa Monica Studio

Well, we found that there were two heavy hitters.

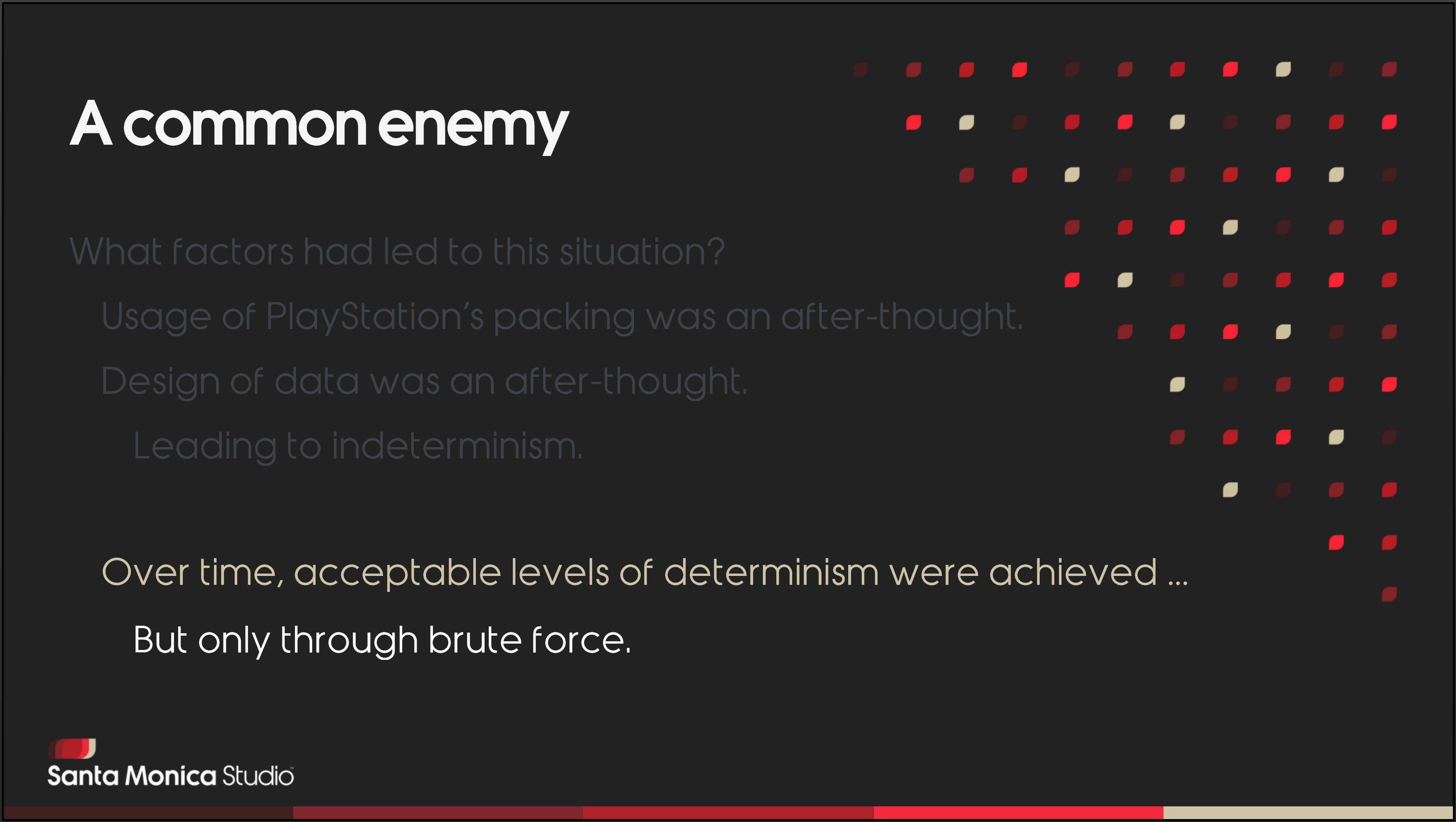
## A common enemy

What factors had led to this situation?

- Usage of PlayStation's packaging was an after-thought.
- Problems in the design of run-time data.
  - Leading to indeterminism.

Santa Monica Studio

For one, our usage of the Playstation packaging system for the live product was an after-thought – something that was solved only when it had to be. But some aspects regarding the design of the run-time data had been an after-thought as well. While we did have a common binary chunk format for the run-time structures of the engine, the contents of any particular chunk were far more loose – leading to sporadic cases of indeterminism in the data.



## A common enemy

What factors had led to this situation?

- Usage of PlayStation's packing was an after-thought.
- Design of data was an after-thought.
  - Leading to indeterminism.

Over time, acceptable levels of determinism were achieved …

But only through brute force.

Santa Monica Studio

With Ascension we were able to achieve acceptable levels of data determinism for the final game – it wasn't perfect, but it was shippable.

## Examples

Unitialized padding:

Time stamps:

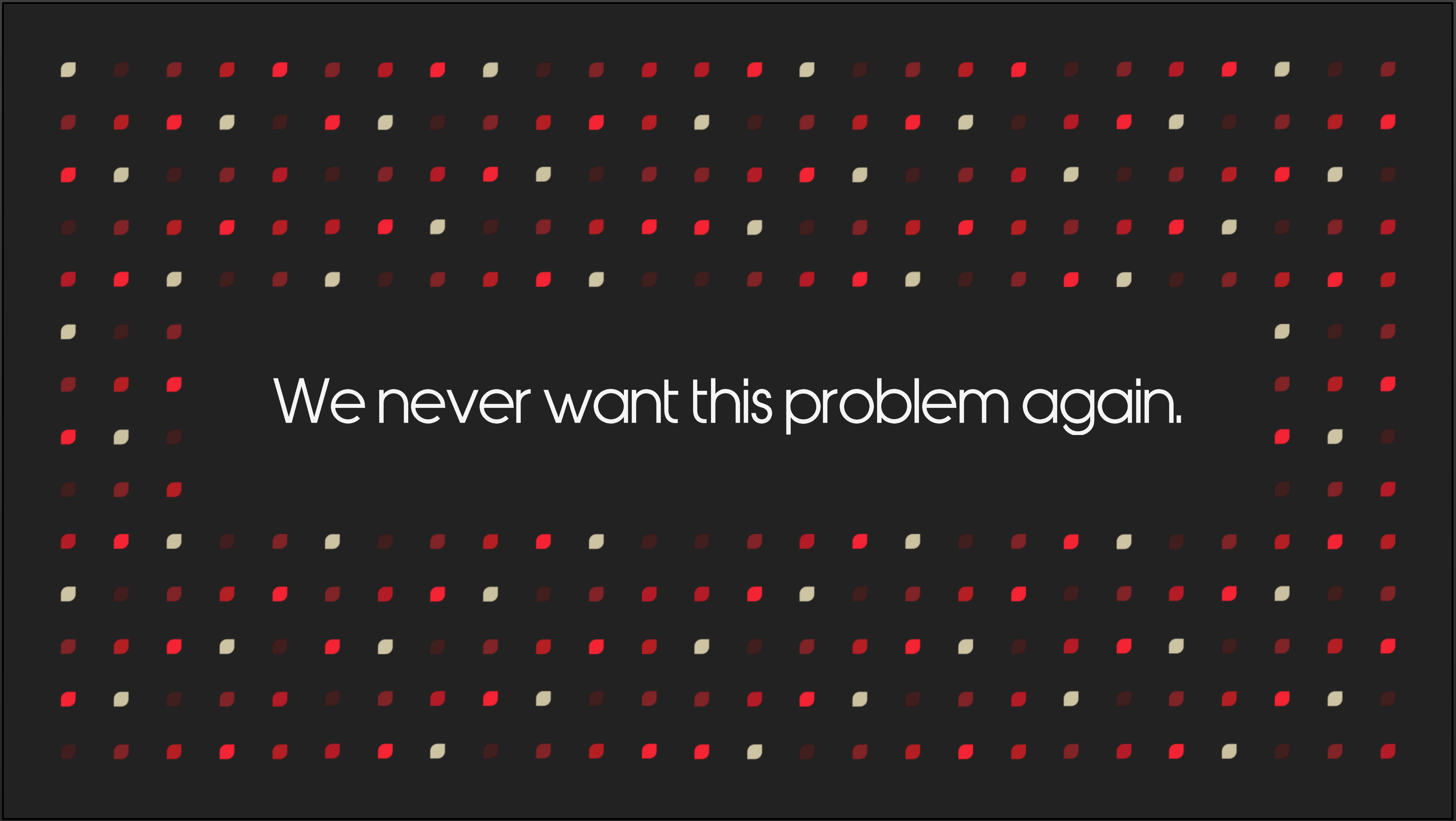
Maya behavior:

Indeterminism when traversing transform siblings.

```
struct Chunk
{
  uint64_t id;
  uint32_t data;
  // uint8_t oops[4];
};

struct Payload
{
  uint64_t id;
  uint64_t timestamp;
  Chunk buffer[64];
};
```

Santa Monica Studio

Indeterminism can be caused by a wide-range of issues, even shockingly easy mistakes. A very common mistake is uninitialized padding in serialized data – and this can also be a nightmare to triage depending on where it occurs in your pipeline. I have been personally shocked by what information programmers inadvertently ship with production data – for example, local timestamps, usernames, the name of a Jenkins node – there seems to be no shortage of opportunity.



## We never want this problem again.

From that experience, two truths were clear. We needed to get a handle on our data, and we never wanted to work on these types of problems ever again.

## Find a better way

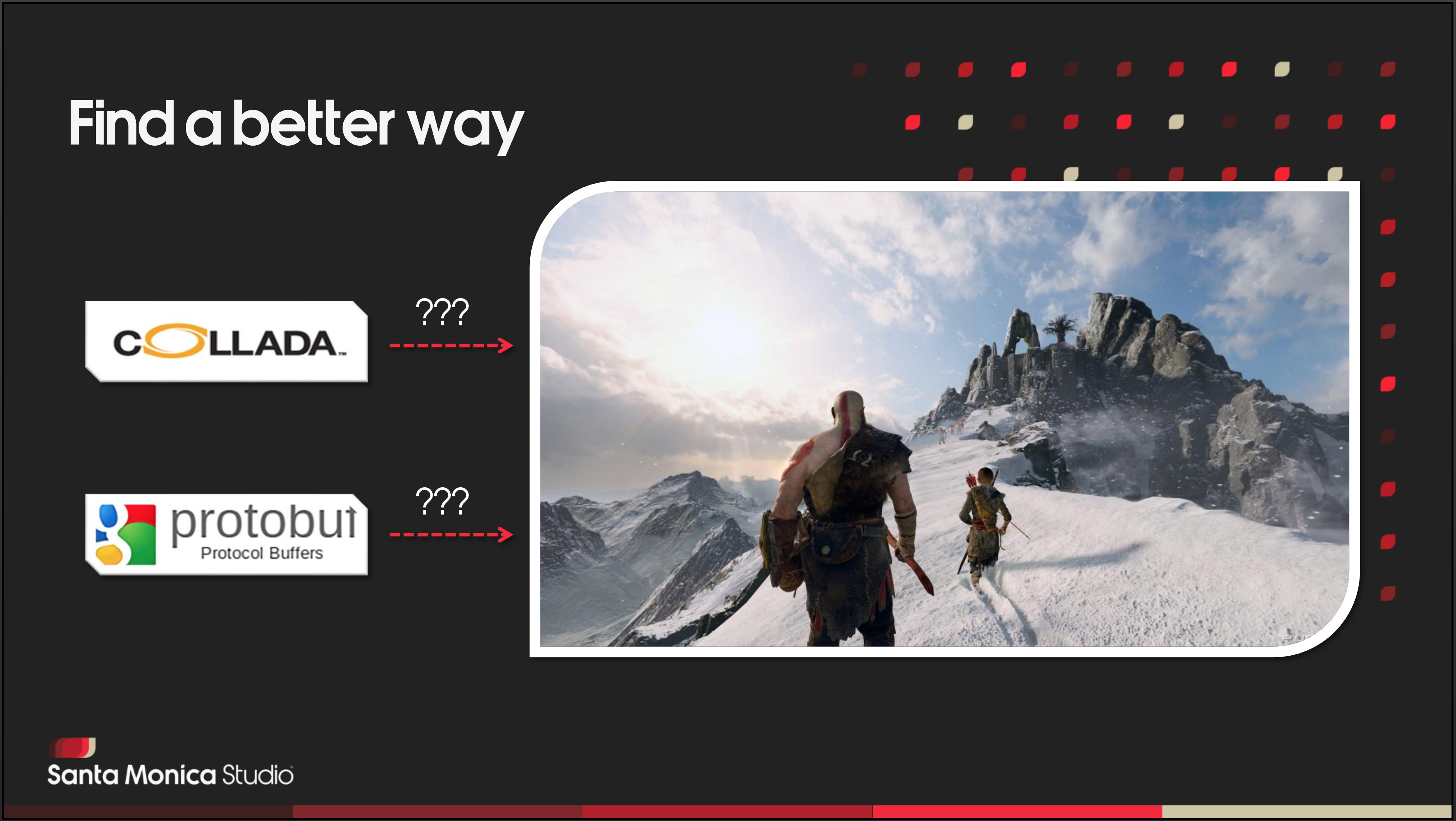
Began exploring explicit data formats.

- We discovered a lot of solutions existed:
  - COLLADA and USD.
  - FlatBuffers and ProtocolBuffers.

None covered the full range of problems we faced.

Santa Monica Studio

So we started looking into what solutions existed for explicit data formats. What we found was that there were two broad categories of technologies to choose from. There were low-level libraries (such as FlatBuffers and ProtocolBuffers) that gave a desirable level of granularity for control and abstraction. And then larger systems such as USD and COLLADA that had clear paths of integration into our Maya pipeline and content authoring. Unfortunately none of these solutions fulfilled all the requirements that we had.



## Find a better way

COLLADA ??? →

protobuf Protocol Buffers ??? →

Santa Monica Studio

For example, COLLADA made sense as a source or intermediate format, but it was less clear how it would integrate into our run-time technology. On the opposite spectrum, Protocol Buffers gave us the granularity we wanted for data definition but it was less clear how tightly we could control properties such as the in-memory layout or allocation scheme. These were very different categories of solutions.

## SmSchema experiment

Data Definition Language ( DDL ) defined through a JSON-based format:

- Served as a proof of concept.
- C++ code generation through Jinja templating.
- Serialization of user-defined types.

Santa Monica Studio

Over time we saw that there could be a lot of advantages to controlling data definition in-house if we were willing to make the investment. So we began an early experiment to create a schema-based data definition language called SmSchema – or Santa Monica Schema. With a restricted set of functionality, it would serve as a proof of concept for the features we thought we might need in the future.

## SmSchema experiment

Native language support of:

- Subset of C language types.
- Variety of data structures.
- Mathematical primitives.

```
int8
int16
int32
int64

uint8
uint16
uint32
uint64

float16
float32
float64

hash32
hash64
hash128

Vector3
Vector4
Matrix4x4
Quaternion

String
Array
Set
Map
```

```
Chunk={
  members={
    id={
      type="uint64"
    },
    data={
      type="uint32"
    }
  }
},
Payload={
  members={
    id={
      type="uint64"
    },
    chunks={
      type={
        typename="Array",
        valuetype="Chunk"
      }
    }
  }
}
```

Santa Monica Studio

The schema of any given type was defined in a simplified JSON notation called SJSON. We supported a small range of native types, containers, and primitives that were appropriate yet restricted for the domain of data that we wanted to represent. For example we had native support for mathematical types such as Vectors and Quaternions, alongside our most commonly encountered data structures such as arrays, sets, and maps. Again the goal of the type system and early experiments was to be pragmatic regarding anything's inclusion.

## SmSchema experiment

Schema definition

```
Chunk={
   members={
      id={
         type="uint64"
      },
      data={
         type="uint32"
      }
   }
}
```

```
Payload={
   members={
      id={
         type="uint64"
      },
      chunks={
         type={
            typename="Array",
            valuetype="Chunk"
         }
      }
   }
}
```

Jinja Code generation

```
struct Chunk
{
   uint64_t id;
   uint32_t data;
};

struct Payload
{
   uint64_t id;
   core::Array<Chunk> chunks;
};
```

Serialization

```
{
   "id": 1,
   "chunks": [{
      "id": 1,
      "data": 100
   },{
      "id": 2,
      "data": 200
   }]
}
```

Santa Monica Studio

The pipeline for this system was implemented in Python and leveraged a templating engine called Jinja that could render generated C++ code for user defined types. It was an easy system to understand – types were defined, the code generator created C++ code consistent with what you would expect, and at the time we only supported JSON as a serialization format.

## SmSchema experiment

The experiment was a success:

- Using JSON as a language meant no parser or grammar.
  - Allowed for quick iteration on the system.
  - Introspecting types with JSON-Jinja interop was trivial.

Santa Monica Studio

As a proof of concept the experiment was a big success. Using JSON as a language directly had a lot of advantages despite the verbosity. For one, it meant that the system didn't really require a parser or well-defined grammar, which allowed for quick iteration and experimentation on the system itself. Writing algorithms to use Jinja for introspecting the JSON schema directly during code generation was trivial due to Python's object and dictionary semantics.

## We began to see a glimpse of SmSchema's overall value.

Even in it's prototype state, SmSchema eliminated an entire class of determinism problems, since we could encode whatever rules we wanted in the code generation directly. We had a better understanding of SmSchema's properties and we were ready to test it at scale on a larger part of the pipeline.

## The next stage

Santa Monica Studio is a Maya workshop.

The entry point to most workflows.

Including world-building.

Santa Monica Studio

Santa Monica Studio is a Maya-shop. What that means is that most of the workflows for creating game content have Maya as an entry point. A historically painful Maya-based workflow has been our world-building, so it seemed like a good place to start applying ourselves.

## The next stage

Santa Monica Studio is a Maya workshop.

The entry point to most workflows.

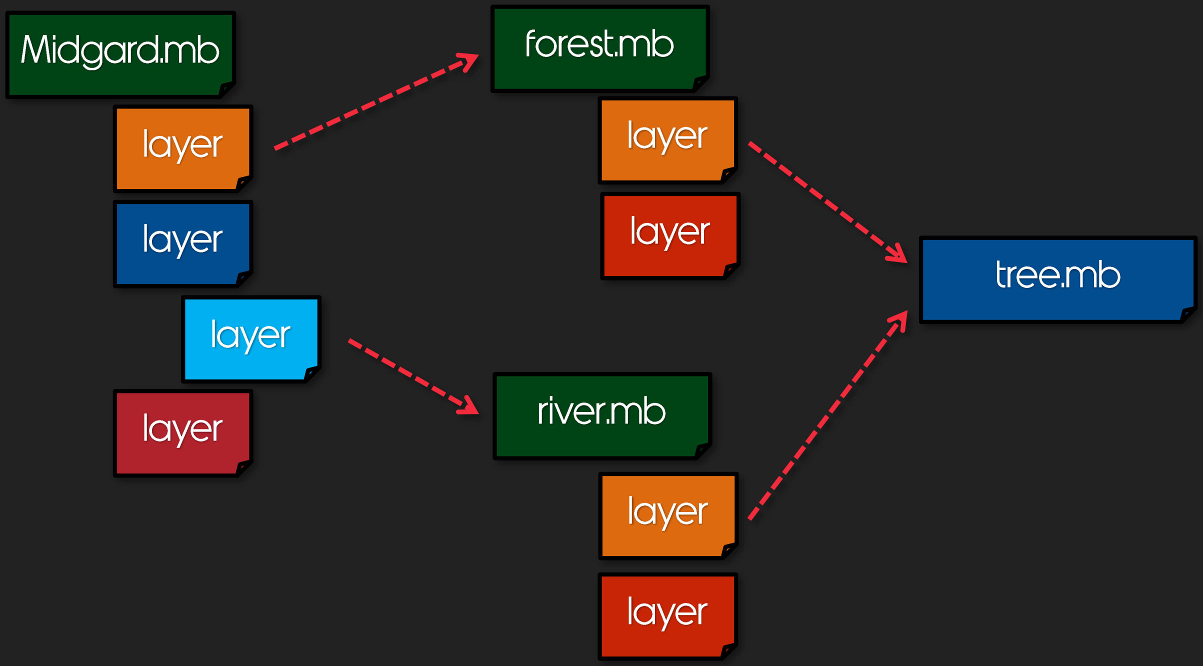
Including world-building.

SmSchema DDL → X3D Scene

Santa Monica Studio

As a first use case, we wanted to use the SmSchema DDL as a means to represent our own proxy scene format inside of Maya.

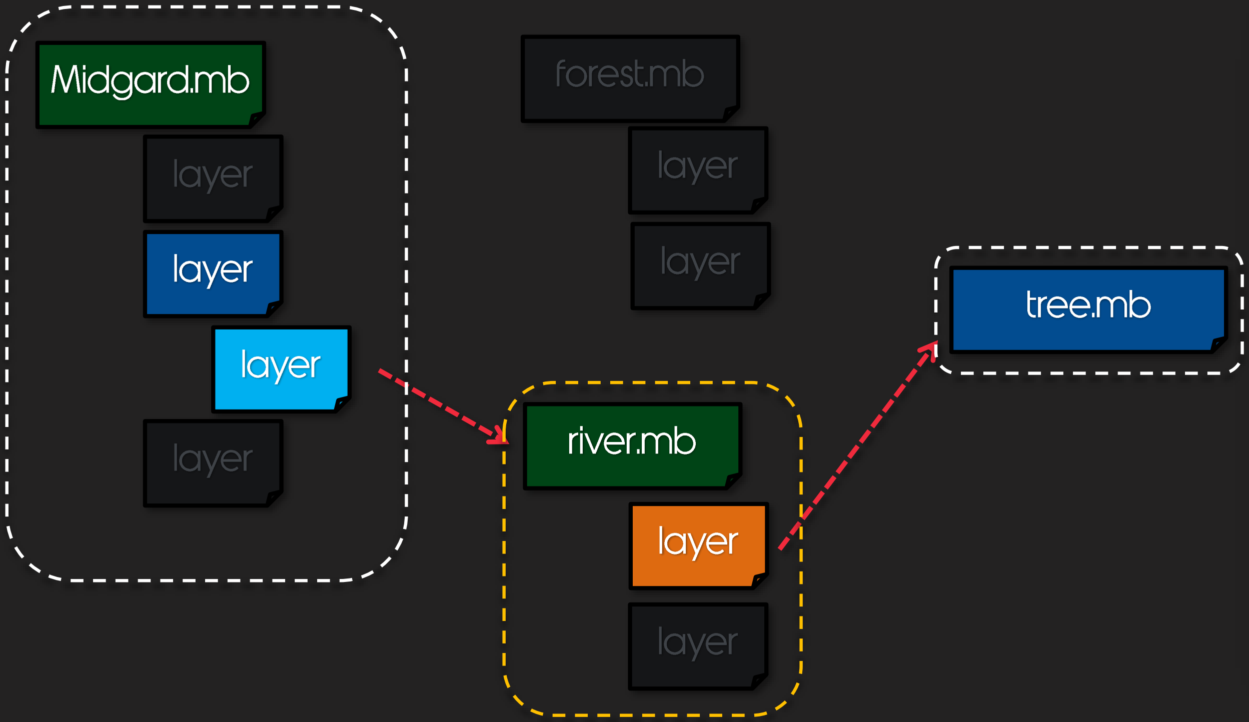
## Segments of the world are defined as a network of layered Maya binary files.



Santa Monica Studio

God of War's world building workflow segments the entire game world into layered network of Maya binary scenes.

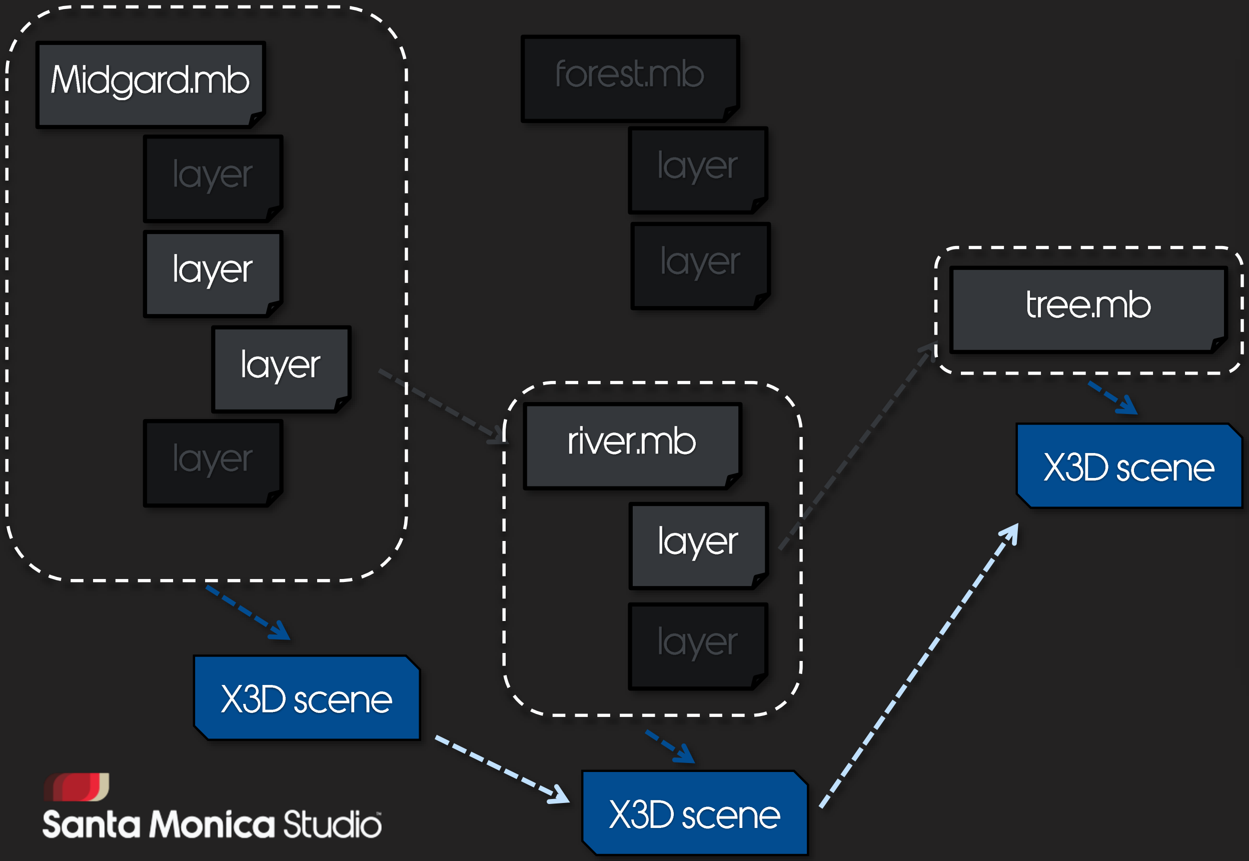
## Artists can work in a section of that world with context.



Santa Monica Studio

Within this network, an artist has the opportunity to work in a particular section of the game with context. As an example, if you decided to work in the highlighted river layer, our custom plugin will render higher layers and lower layers as unchangeable proxy geometry.

## The proxy representation of a Maya scene is known as the X3D scene.



Santa Monica Studio

The in-memory proxy representation of any particular Maya file is known as the X3D Scene.

## Defining the X3D scene

X3D scene was particularly problematic:

- Fully custom serialization.
- Source of a lot data indeterminism

Opportunities:

- An intermediate file format.
- High impact and visibility.

Santa Monica Studio

The X3D Scene seemed like an ideal candidate to stress SmSchema's capabilities for many reasons. It was the source of a lot of technical debt such as an isolated and fully custom serialization system, numerous performance problems, indeterminism, and abandoned asset referencing capabilities amongst used ones.

But it also presented a restricted set of problems that we had to solve. It had this nice property of being an intermediate file which was regenerated any time a source Maya file was changed or created. And it matched our mental model of where we thought SmSchema could provide value.

## Defining the X3D scene

Committed to rewriting the format:

- SmSchema-defined structure:
  - Represent all necessary Maya primitives.
  - Generate Maya import code.
  - Well-defined referencing.

Santa Monica Studio

We could see the path … all we had to do was represent all the necessary Maya primitives, automatically generate the Maya import code, formalize a referencing system amongst the scenes to define the scene graph, improve rendering and loading performance for some political buy-in from the artists.

## Defining the X3D scene

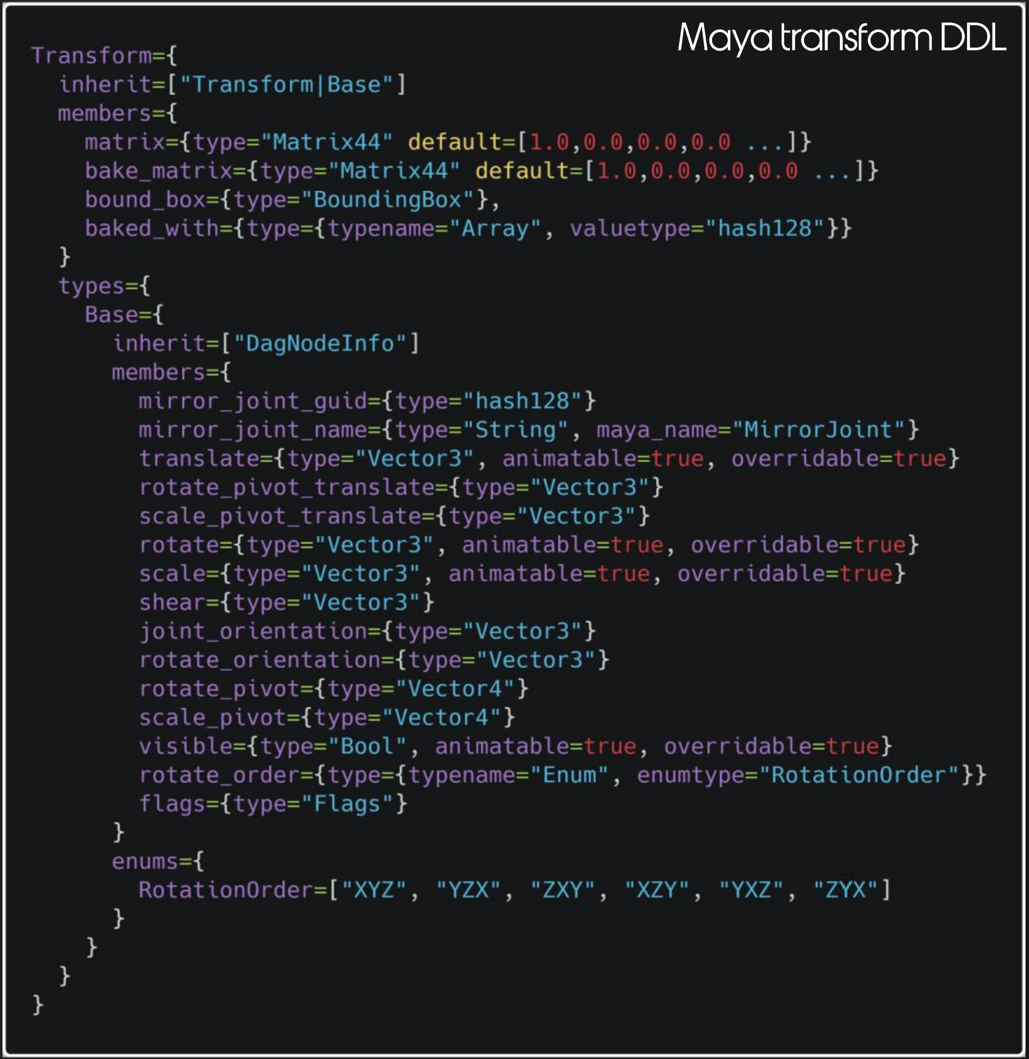
Committed to rewriting the format:

- SmSchema-defined structure:
  - Represent all necessary Maya primitives.
  - Well-defined referencing.
  - Improve overall performance.

... "We'll fix all the problems!"

Santa Monica Studio

In fact with SmSchema we can potentially solve all the problems. Simple enough!

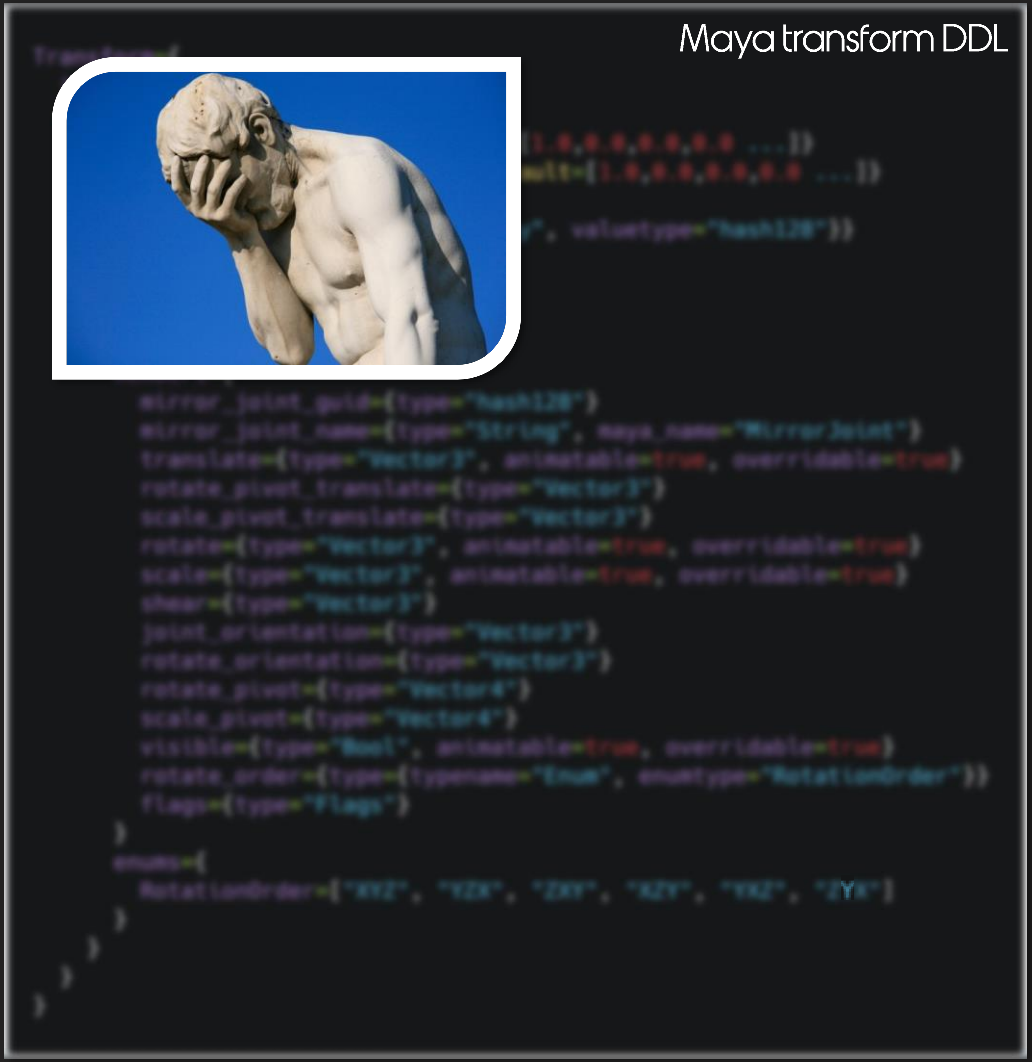

Maya transform DDL

```
Transform={
  inherit=["Transform|Base"]
  members={
    matrix={type="Matrix44" default=[1.0,0.0,0.0,0.0 ...]}
    bake_matrix={type="Matrix44" default=[1.0,0.0,0.0,0.0 ...]}
    bound_box={type="BoundingBox"},
    baked_with={type={typename="Array", valuetype="hash128"}}
  }
  types={
    Base={
      inherit=["DagNodeInfo"]
      members={
        mirror_joint_guid={type="hash128"}
        mirror_joint_name={type="String", maya_name="MirrorJoint"}
        translate={type="Vector3", animatable=true, overridable=true}
        rotate_pivot_translate={type="Vector3"}
        scale_pivot_translate={type="Vector3"}
        rotate={type="Vector3", animatable=true, overridable=true}
        scale={type="Vector3", animatable=true, overridable=true}
        shear={type="Vector3"}
        joint_orientation={type="Vector3"}
        rotate_orientation={type="Vector3"}
        rotate_pivot={type="Vector4"}
        scale_pivot={type="Vector4"}
        visible={type="Bool", animatable=true, overridable=true}
        rotate_order={type={typename="Enum", enumtype="RotationOrder"}}
        flags={type="Flags"}
      }
      enums={
        RotationOrder=["XYZ", "YZX", "ZXY", "XZY", "YXZ", "ZYX"]
      }
    }
  }
}
```

## Defining the X3D scene

SmSchema was not ready.

That’s not what happened. Reality began to sink in – SmSchema was not ready. Here’s an example of the monstrosity it took to define a partial schema for the Maya transform.

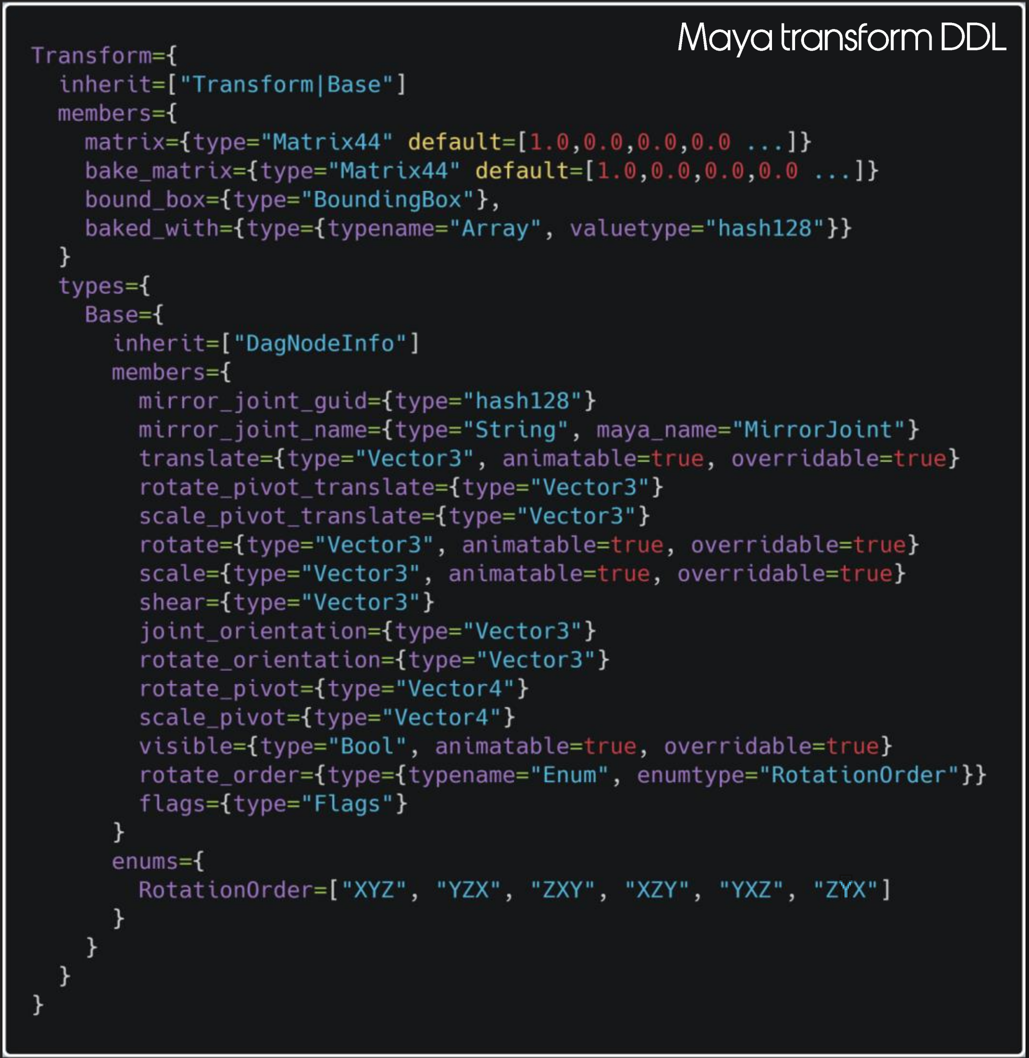


## Defining the X3D scene

SmSchema was not ready.

- DDL syntax was too verbose.
- Sparse language features.
- Serialization was too inefficient:
  - 2 gigabyte JSON scenes.

The previous prototype JSON syntax for the schema language was now far too verbose and practically unauthorable. The serialization was also way too inefficient to support the amount of content in any production level. And SmSchema as a language was far too sparse in features to represent all of the concepts we needed. Features like type inheritance and enumerations were too cumbersome and didn't make sense.


Maya transform DDL

```
Transform={
  inherit=["Transform|Base"]
  members={
    matrix={type="Matrix44" default=[1.0,0.0,0.0,0.0 ...]}
    bake_matrix={type="Matrix44" default=[1.0,0.0,0.0,0.0 ...]}
    bound_box={type="BoundingBox"},
    baked_with={type={typename="Array", valuetype="hash128"}}
  }
  types={
    Base={
      inherit=["DagNodeInfo"]
      members={
        mirror_joint_guid={type="hash128"}
        mirror_joint_name={type="String", maya_name="MirrorJoint"}
        translate={type="Vector3", animatable=true, overridable=true}
        rotate_pivot_translate={type="Vector3"}
        scale_pivot_translate={type="Vector3"}
        rotate={type="Vector3", animatable=true, overridable=true}
        scale={type="Vector3", animatable=true, overridable=true}
        shear={type="Vector3"}
        joint_orientation={type="Vector3"}
        rotate_orientation={type="Vector3"}
        rotate_pivot={type="Vector4"}
        scale_pivot={type="Vector4"}
        visible={type="Bool", animatable=true, overridable=true}
        rotate_order={type={typename="Enum", enumtype="RotationOrder"}}
        flags={type="Flags"}
      }
      enums={
        RotationOrder=["XYZ", "YZX", "ZXY", "XZY", "YXZ", "ZYX"]
      }
    }
  }
}
```

## Defining the X3D scene

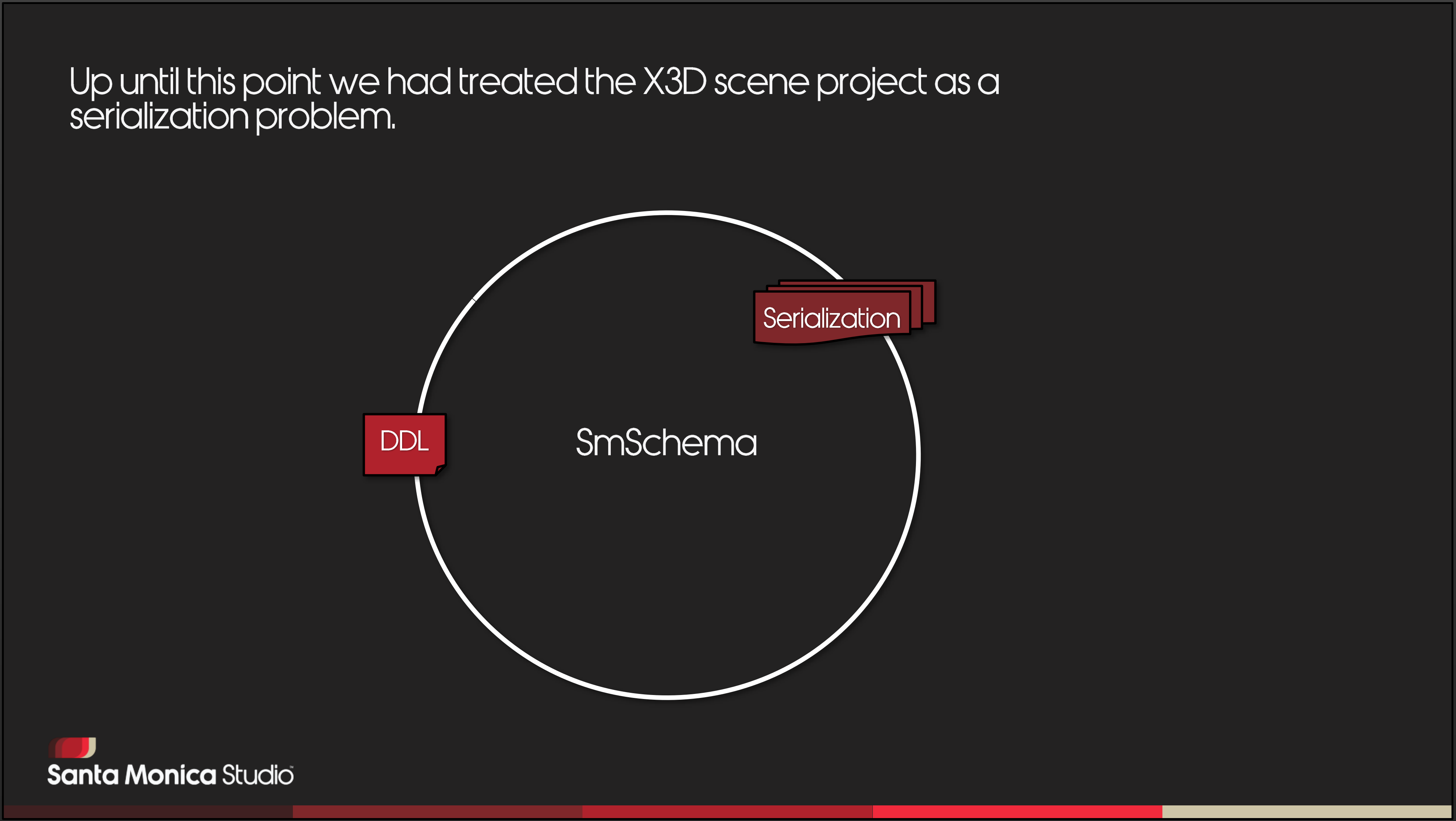
SmSchema was not ready.

- DDL syntax was too verbose.
- Sparse language features.
- Serialization was too inefficient.

Primitive referencing options:

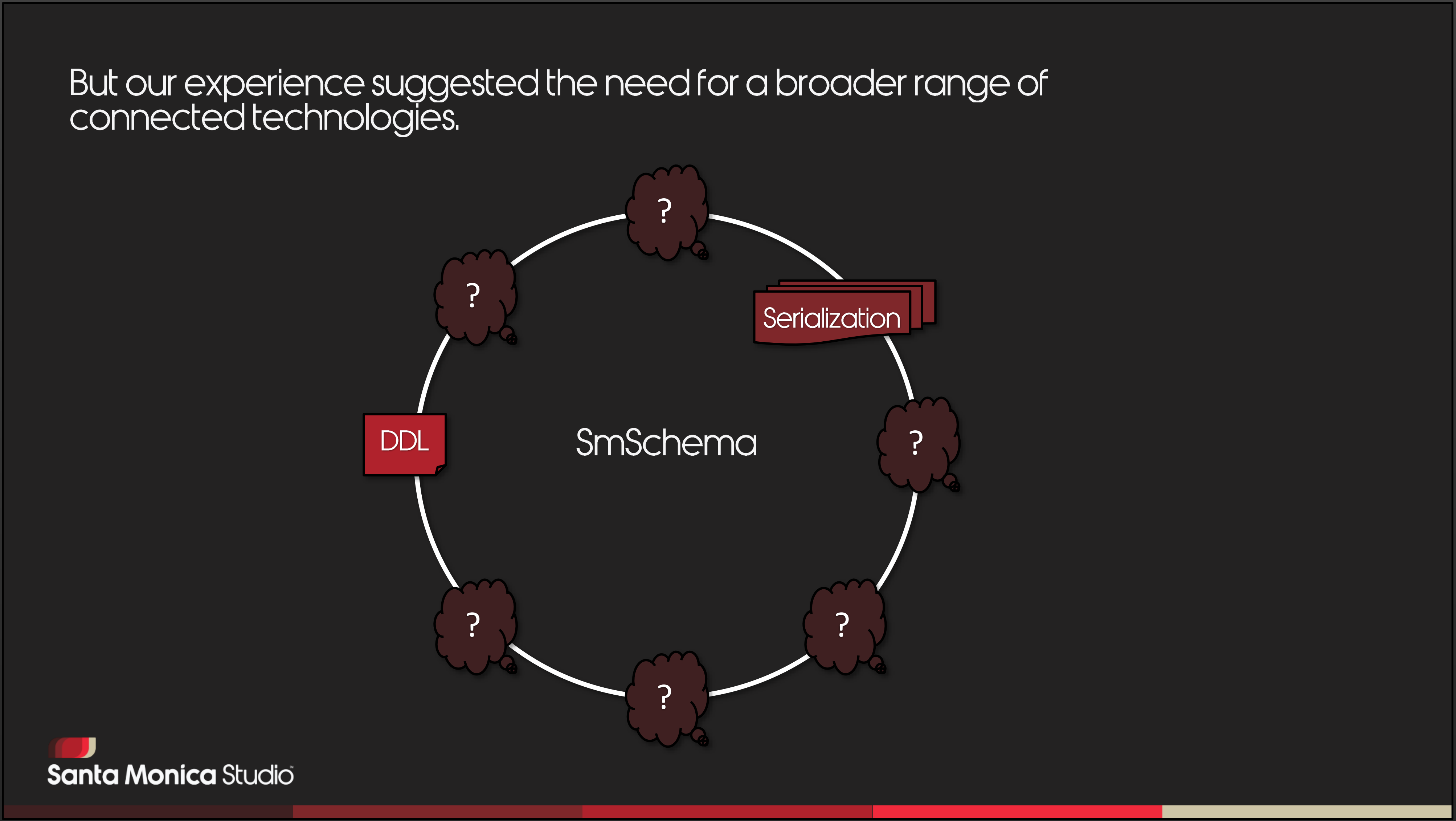
- String file path? GUID?

To make matters worse, we struggled for a long time about how to represent the scene graph itself. How would we express a scene reference? Should we use a string file path? Maybe a GUID? It wasn’t clear.

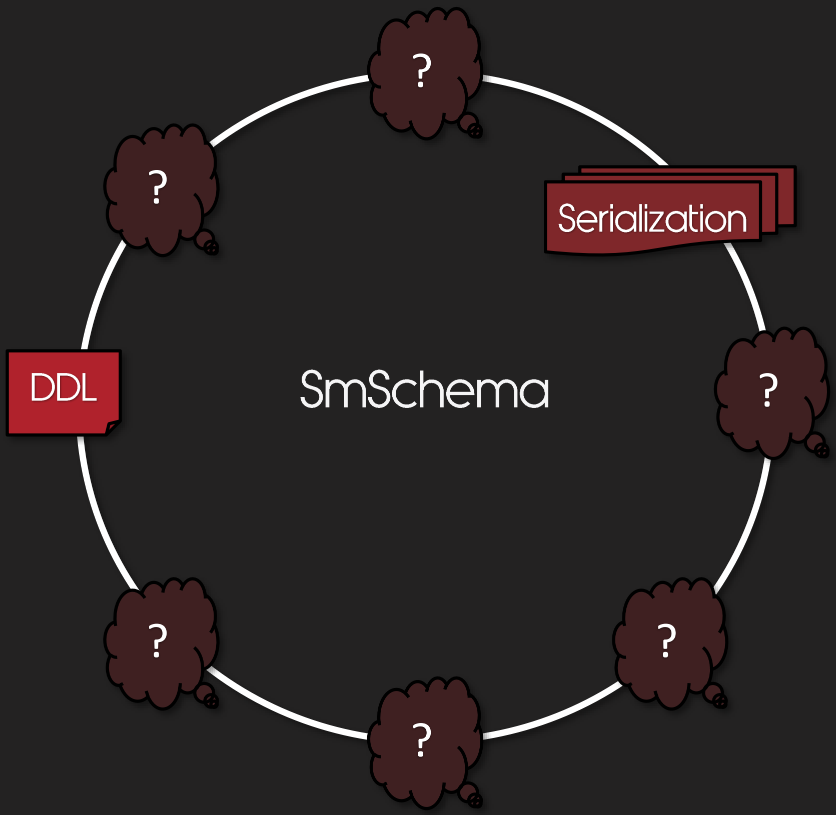


## Up until this point we had treated the X3D scene project as a serialization problem.

Through all of the struggles and questions it began to dawn on us that our mental model of SmSchema was far more incomplete than we had first thought. Up until this point we had treated the X3D scene as a serialization problem, requiring only a robust set of technology around data definition and data formats.



## But our experience suggested the need for a broader range of connected technologies.



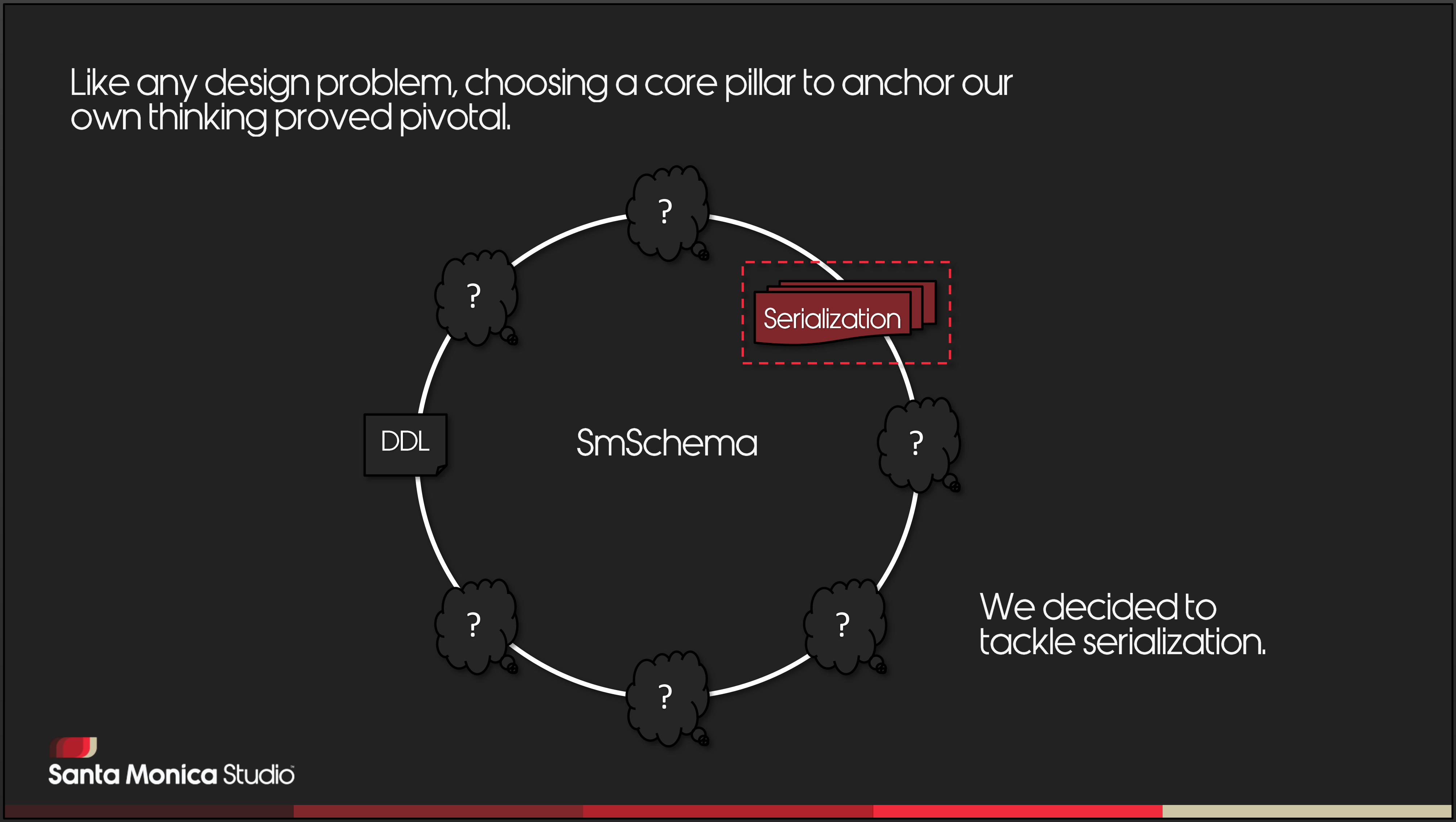
Santa Monica Studio

But the necessity for referencing and expanded code generation was directing us in a different way. The X3D scene project suggested the need for a broader set of interconnected technologies. And while the DDL and the serialization were an important aspect of that system, they seemed to be only components of a larger vision. And while we had been spending time thinking about the design of X3D itself …

## Defining SmSchema.

Santa Monica Studio

With this new experience, it was now time to think about the design of SmSchema.



## Like any design problem, choosing a core pillar to anchor our own thinking proved pivotal.

We decided to tackle serialization.

Santa Monica Studio

As engineers, designing can sometimes be a painful process. With SmSchema we knew there was a lot of ground to cover and so we decided to pick the design of serialization as an anchor point for our thinking. Plus, serialization had been particularly painful with X3D scene project, so it was a good starting point.

## Defining SmSchema

First-principles requirements:

- Automatic serialization.
- Native binary and text format.
- Random access.
- Binary memory mappability.
- Strict, rich DDL.

Santa Monica Studio

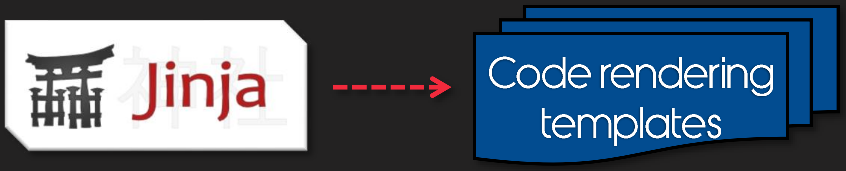
With our newfound experience we established a set of first principles regarding the design of serialization – all of these were must-haves. If SmSchema was to be successful we needed automatic serialization, a native binary and text data format, full random access of fields within the serialized data, and complete binary memory mappability. In addition to these serialization goals, we would need to invest in redesigning and expanding our schema language.

## Defining SmSchema

Automatic serialization:

Data structures would be defined only once …

Push the complexity of serialization to generated code.



Santa Monica Studio

In our previous experiments, we had seen the value of the Jinja code rendering templates in pushing serialization complexity away from any user-defined code and we wanted to keep this.

## Defining SmSchema

Native binary and text format:

- Flexibility to apply the technology in various domains:
  - Binary files for the run-time …
  - Supporting automatic conversion to and from a text format.
    - Such as JSON and more.

Santa Monica Studio

Supporting different serialization formats meant that we could potentially use SmSchema in various capacities. Our previous experience with JSON serialization initially led us to believe that JSON was a poor serialization format. However it was clear that text-based formats had a lot of advantages for human interaction, such as diffing and triaging of data issues. While a packed binary format had the advantage of speed and other properties such as information density.

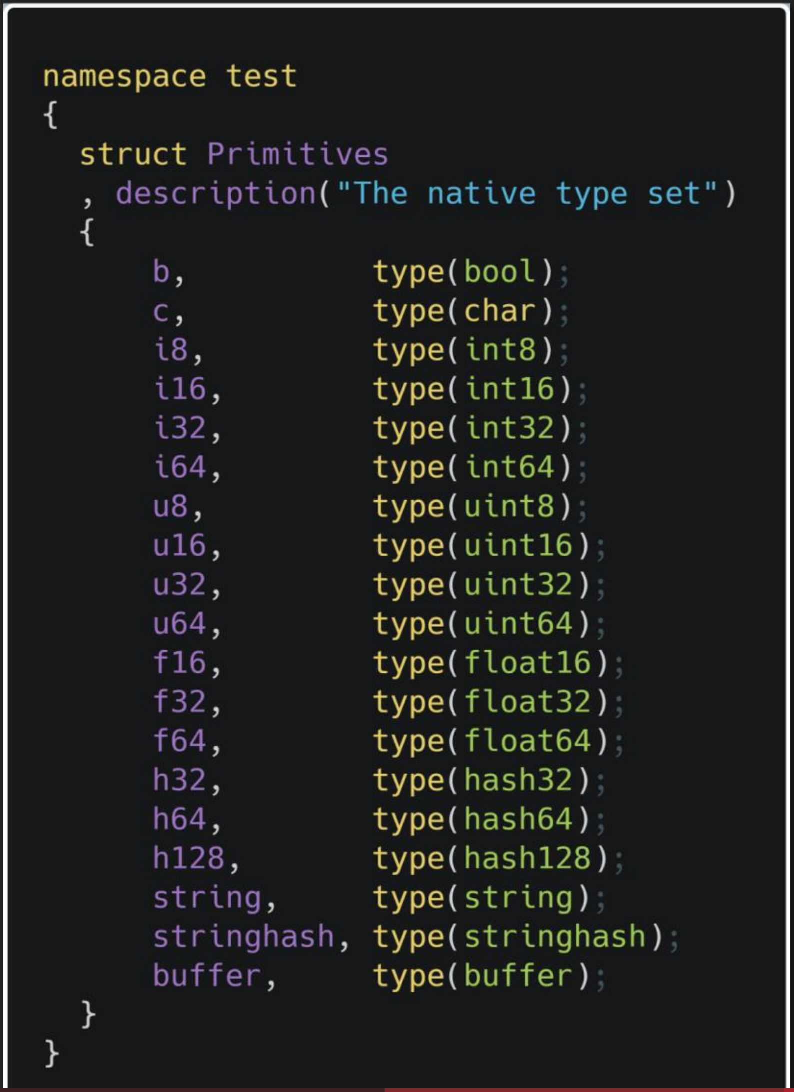
## Defining SmSchema

Random access, and binary memory-mappability:

- This meant no deserialization step for binary data.
  - Loading should be as simple as mmap or memcpy.
- Pointer patching had to be optional.

Santa Monica Studio

In our research of binary formats, we concluded that striving for full binary memory mappability had lot of performance and design advantages. Loading SmSchema binary files would be as simple as calling mmap or memcpy, with no deserialization step. We knew that this would have a profound effect on the design of pointers, but it seemed feasible and a worthy goal.



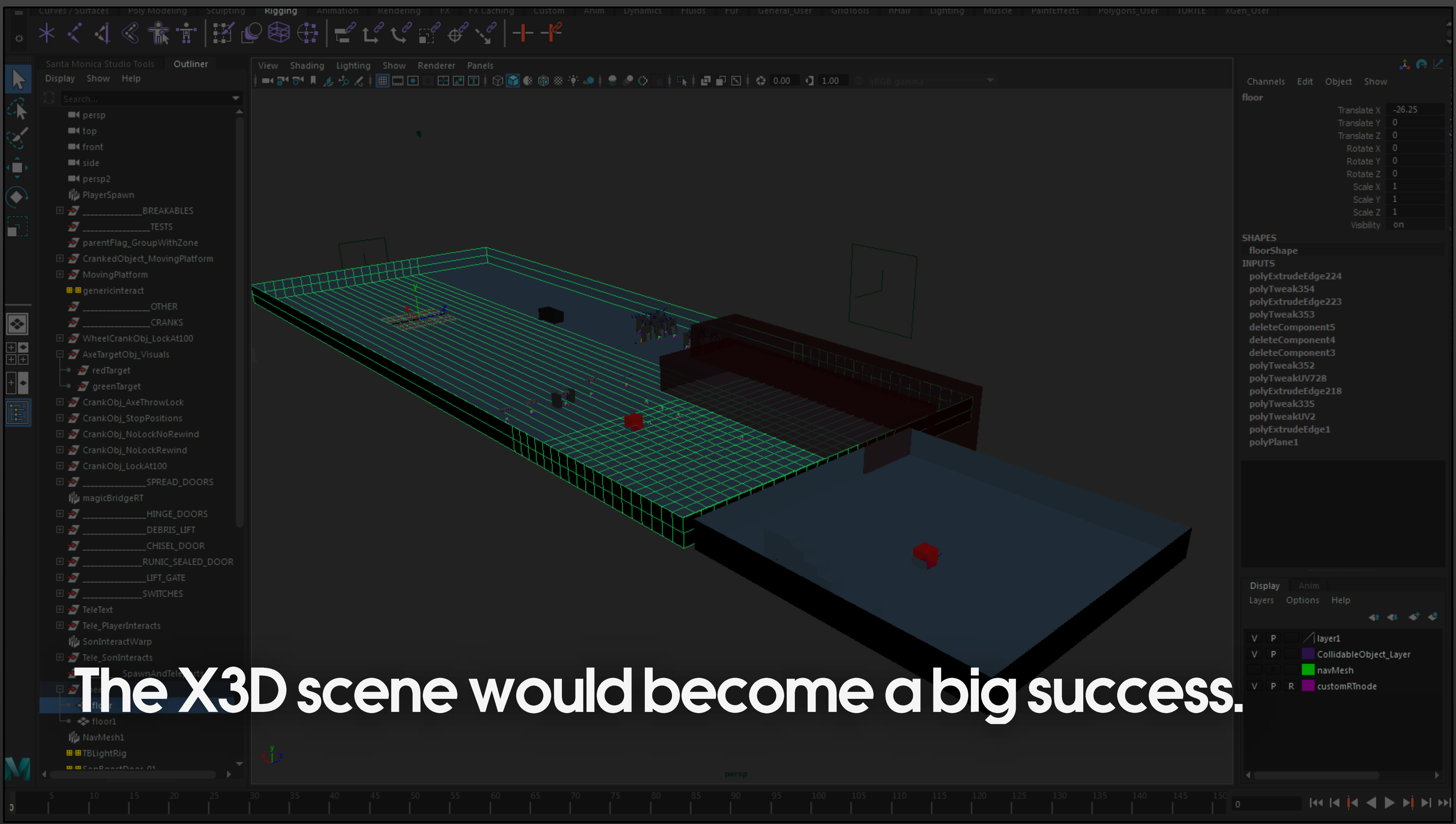
```
namespace test
{
  struct Primitives
  , description("The native type set")
  {
      b,          type(bool);
      c,          type(char);
      i8,         type(int8);
      i16,        type(int16);
      i32,        type(int32);
      i64,        type(int64);
      u8,         type(uint8);
      u16,        type(uint16);
      u32,        type(uint32);
      u64,        type(uint64);
      f16,        type(float16);
      f32,        type(float32);
      f64,        type(float64);
      h32,        type(hash32);
      h64,        type(hash64);
      h128,       type(hash128);
      string,     type(string);
      stringhash, type(stringhash);
      buffer,     type(buffer);
  }
}
```

## Defining SmSchema

Strict, rich definition:

- Get rid of the JSON-based language.
- Formalize our language:
  - The dream was to author data in a restricted but modernized syntax similar to C for ease of use, and approachability.

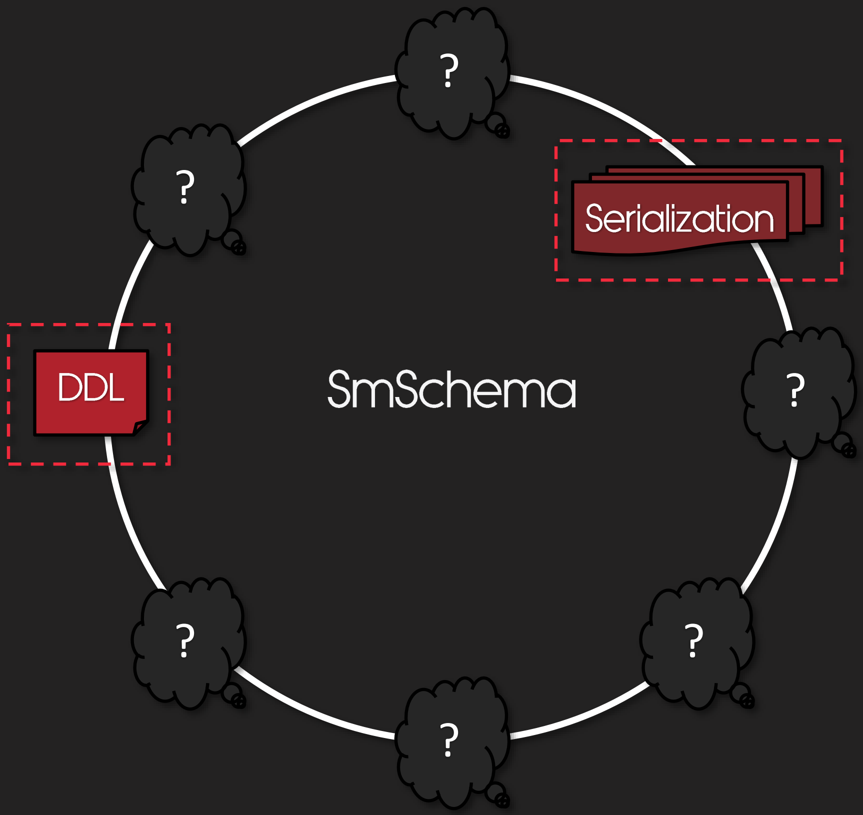
JSON wasn't working as a schema language – we needed to get rid of it. Our philosophy was simple, it should be a joy to use the schema but also feel familiar. Our pipe-dream was either to design or find something that felt like a modernized, C-style language – but with an expanded set of types and primitives that contextually made sense when designing data for games.



## The X3D scene would become a big success.

Focusing on the DDL and serialization of SmSchema was a big of part of what made the X3D proxy a big success. Investing in a binary serialization format eventually did pay dividends in terms of performance and stability. But the existing JSON format also allowed us to triage the data of the proxy much more easily than in the past.

## While the bigger picture still eluded us, the DDL and serialization backends were becoming solid.



Santa Monica Studio

At this point in time, the DDL and serialization formats were becoming much more robust. We understood their requirements quite well, and the proxy was now in use. But this bigger picture of what SmSchema could be still eluded us. For example, we still didn't understand how features such as referencing could be modeled in this system beyond string paths to other Maya files.

## New challenges

As God of War progressed:

It became clear that having Maya files as the authored source of most game assets was very problematic.

As our production progressed, we began to run into new issues. Maya is a very powerful ecosystem to invest in and has a lot of advantages for smaller engineering teams. But having Maya files as the authored source for most game content was proving to be a very problematic historical artifact.

## New challenges

As God of War progressed:

It became clear that having Maya files as the authored source of most game assets was very problematic.

Poor scalability, information density.

Editing outside of Maya was fragile.

Extracting dependency information was difficult and expensive.

By imposing too much responsibility on the format, what could our expectation be other than this?

## New challenges

Maya was a huge constraint on the project:

- The studio content teams were productive with Maya.
- Even to this day, Maya is still in use.
  - It's a typical tricky situation:
    - How to fix a plane that is in mid-flight?

Santa Monica Studio

It was a tricky situation. Despite all these problems the game was in production and our studio was acclimated to Maya as a workflow. It's a classic scenario where you begin to ask yourself "How do you fix the plane that is mid-flight?" But there is a silver lining …

## New insight

Question: In time, could we just replace the Maya format entirely?

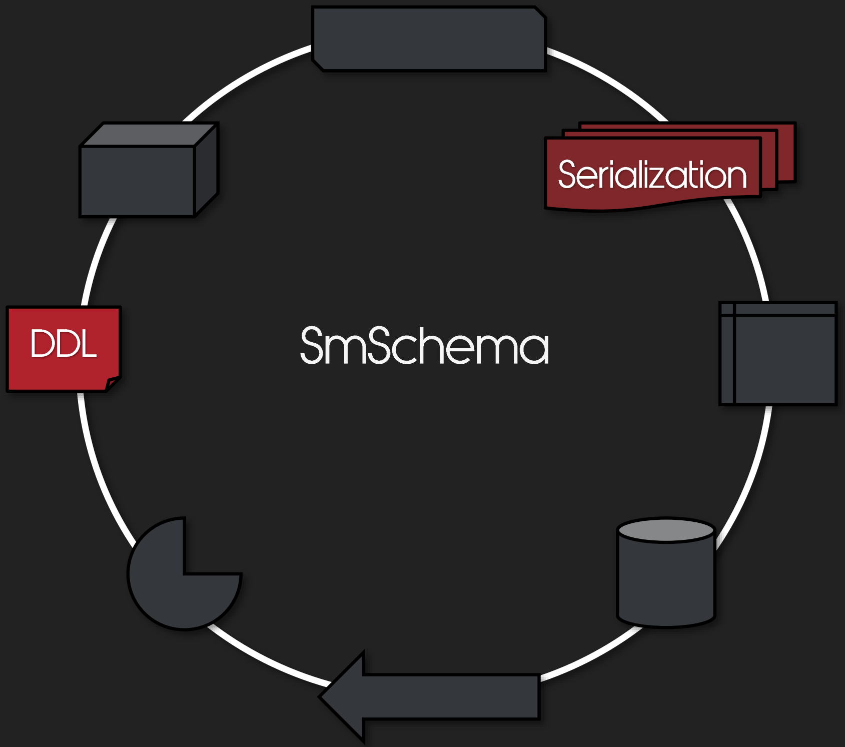
Question: Could that format be built with SmSchema?

Question: What requirements would that pose on the system?

Santa Monica Studio

Having the experience of stretching Maya to its limits gave us the acumen and experience to refine our own thinking and design with important questions.

## The bigger picture was now coming into full focus. The architecture and design was taking shape.



Santa Monica Studio

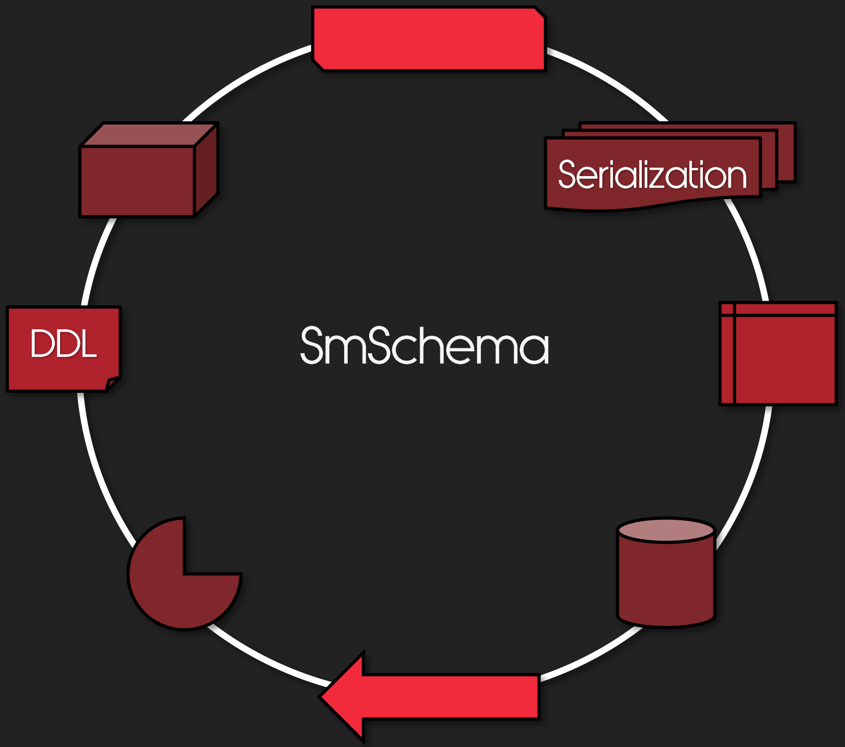
Questions that allowed us to see the bigger picture of what a technology like this could become.

## Into the present.

Santa Monica Studio

So with that, let’s fast forward to the present and explore where our experience has taken us.

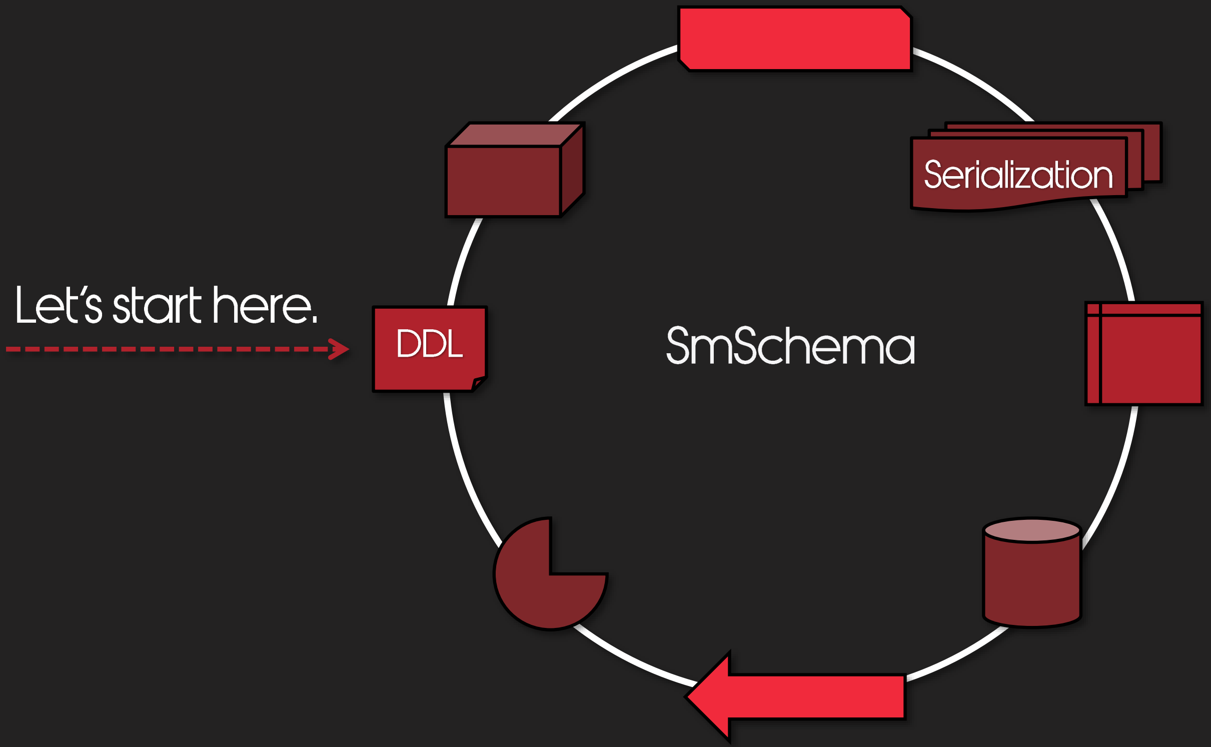
## SmSchema is a set of interconnected technologies surrounding data description.



Santa Monica Studio

SmSchema is a set of interconnected technologies surrounding the domain of data description. What we've chosen to include as part of our design is equally as important as the things we've left out.

## SmSchema is a set of interconnected technologies surrounding data description.



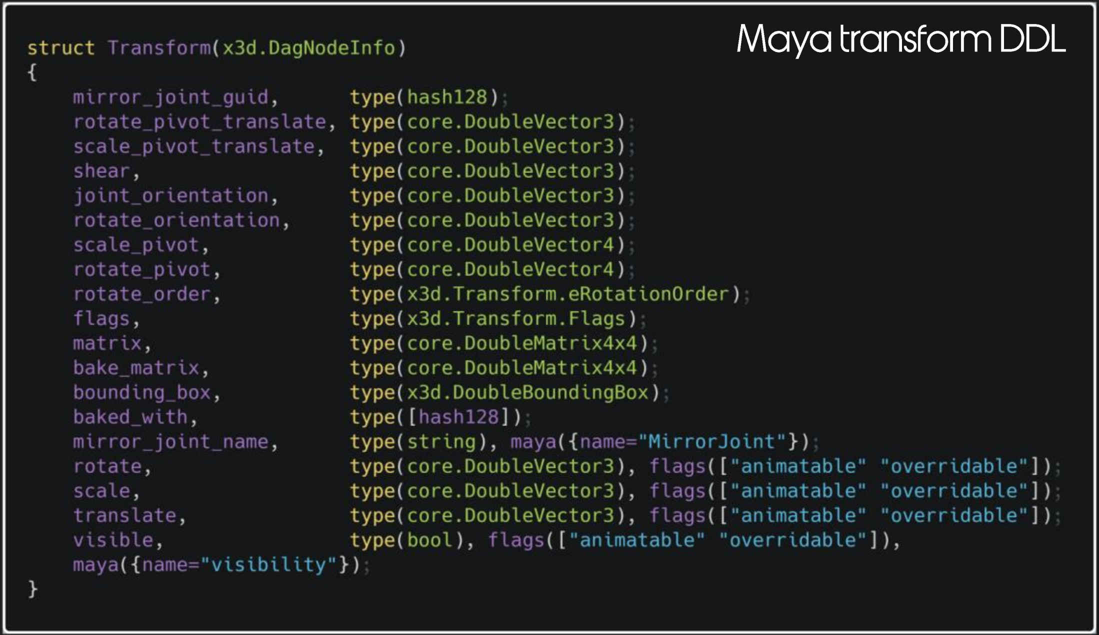
Santa Monica Studio

Let's start here at the DDL.

## Data Definition Language

Inspired by the Insomniac DDL:

Language defined by ANTLR.



Maya transform DDL

```
struct Transform(x3d.DagNodeInfo)
{
    mirror_joint_guid,       type(hash128);
    rotate_pivot_translate, type(core.DoubleVector3);
    scale_pivot_translate,  type(core.DoubleVector3);
    shear,                  type(core.DoubleVector3);
    joint_orientation,      type(core.DoubleVector3);
    rotate_orientation,     type(core.DoubleVector3);
    scale_pivot,            type(core.DoubleVector4);
    rotate_pivot,           type(core.DoubleVector4);
    rotate_order,           type(x3d.Transform.eRotationOrder);
    flags,                  type(x3d.Transform.Flags);
    matrix,                 type(core.DoubleMatrix4x4);
    bake_matrix,            type(core.DoubleMatrix4x4);
    bounding_box,           type(x3d.DoubleBoundingBox);
    baked_with,             type([hash128]);
    mirror_joint_name,      type(string), maya({name="MirrorJoint"});
    rotate,                 type(core.DoubleVector3), flags(["animatable" "overridable"]);
    scale,                  type(core.DoubleVector3), flags(["animatable" "overridable"]);
    translate,              type(core.DoubleVector3), flags(["animatable" "overridable"]);
    visible,                type(bool), flags(["animatable" "overridable"]),
    maya({name="visibility"});
}
```

Santa Monica Studio

Our current data definition language was heavily inspired by the Insomniac DDL due to its clarity and aesthetic. The DDL's grammar and parser are implemented using a language technology stack known as ANTLR.

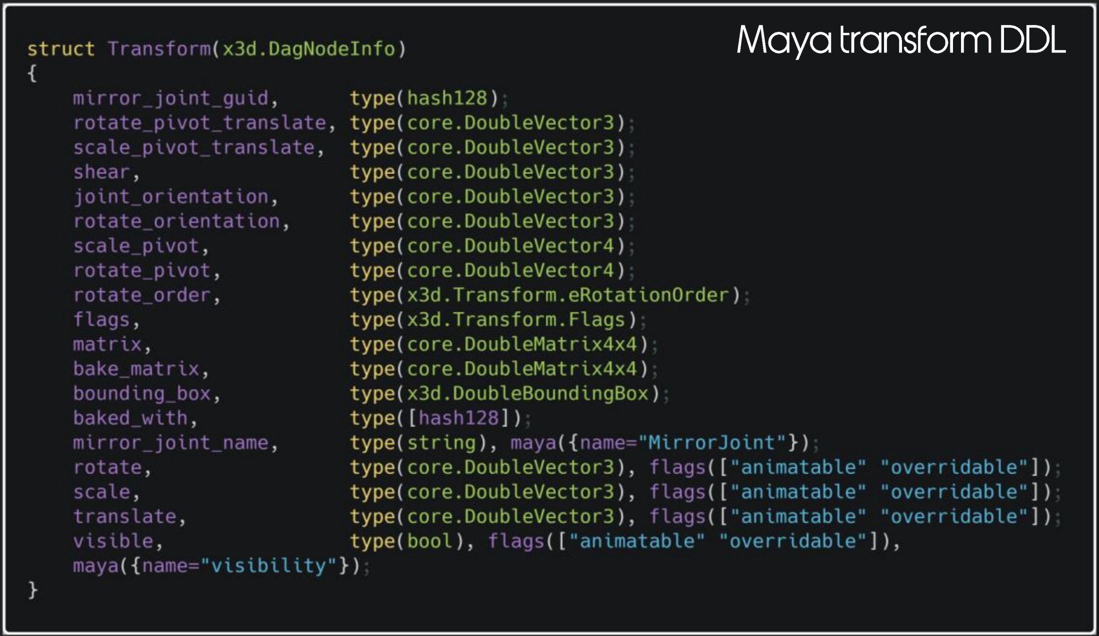
## Data Definition Language

Inspired by the Insomniac DDL:

- Language defined by ANTLR.

Strict syntax and support for:

- Native and custom types.
- Foreign type aliasing.
- Code generation overrides.

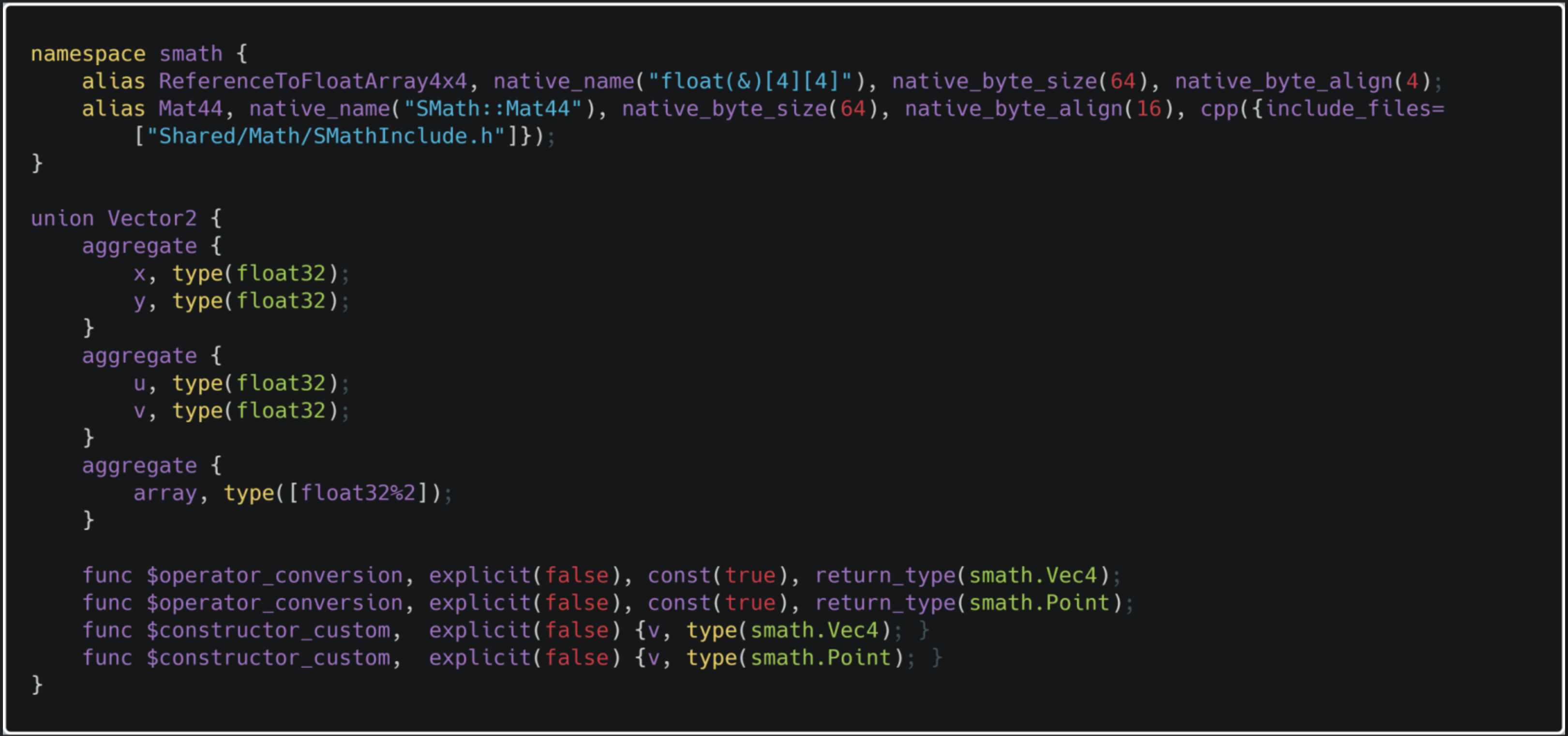

Maya transform DDL

```
struct Transform(x3d.DagNodeInfo)
{
    mirror_joint_guid,      type(hash128);
    rotate_pivot_translate, type(core.DoubleVector3);
    scale_pivot_translate,  type(core.DoubleVector3);
    shear,                  type(core.DoubleVector3);
    joint_orientation,      type(core.DoubleVector3);
    rotate_orientation,     type(core.DoubleVector3);
    scale_pivot,            type(core.DoubleVector4);
    rotate_pivot,           type(core.DoubleVector4);
    rotate_order,           type(x3d.Transform.eRotationOrder);
    flags,                  type(x3d.Transform.Flags);
    matrix,                 type(core.DoubleMatrix4x4);
    bake_matrix,            type(core.DoubleMatrix4x4);
    bounding_box,           type(x3d.DoubleBoundingBox);
    baked_with,             type([hash128]);
    mirror_joint_name,      type(string), maya({name="MirrorJoint"});
    rotate,                 type(core.DoubleVector3), flags(["animatable" "overridable"]);
    scale,                  type(core.DoubleVector3), flags(["animatable" "overridable"]);
    translate,              type(core.DoubleVector3), flags(["animatable" "overridable"]);
    visible,                type(bool), flags(["animatable" "overridable"]),
    maya({name="visibility"});
}
```

Santa Monica Studio

While we support many of the features that we would all come to expect from a data definition system, I'd like to focus our time on a particular aspect that has proven invaluable. As game programmers often we are at the mercy of interfacing with third party formats and data structures that are outside of our control. With SmSchema we didn't want to ignore this problem, so features such as foreign type aliasing, raw buffers, and code generation overrides are designed to allow this interop natively.

## Data Definition Language

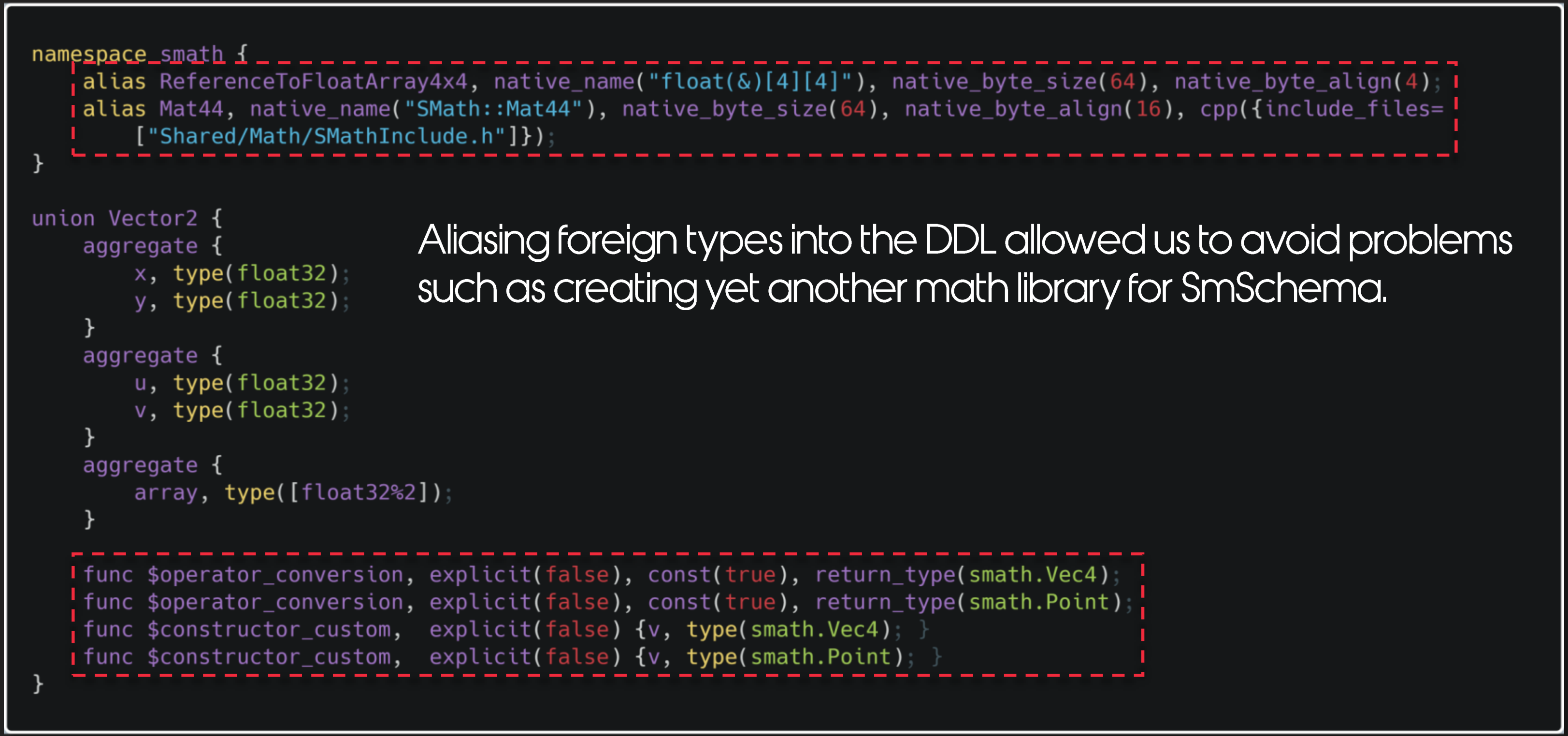


```
namespace smath {
    alias ReferenceToFloatArray4x4, native_name("float(&)[4][4]"), native_byte_size(64), native_byte_align(4);
    alias Mat44, native_name("SMath::Mat44"), native_byte_size(64), native_byte_align(16), cpp({include_files=
        ["Shared/Math/SMathInclude.h"]});
}

union Vector2 {
    aggregate {
        x, type(float32);
        y, type(float32);
    }
    aggregate {
        u, type(float32);
        v, type(float32);
    }
    aggregate {
        array, type([float32%2]);
    }

    func $operator_conversion, explicit(false), const(true), return_type(smath.Vec4);
    func $operator_conversion, explicit(false), const(true), return_type(smath.Point);
    func $constructor_custom,  explicit(false) {v, type(smath.Vec4); }
    func $constructor_custom,  explicit(false) {v, type(smath.Point); }
}
```

Santa Monica Studio

Here’s an example where we have an existing math library – SMath – that we want to use directly in SmSchema.

## Data Definition Language



```
namespace smath {
    alias ReferenceToFloatArray4x4, native_name("float(&)[4][4]"), native_byte_size(64), native_byte_align(4);
    alias Mat44, native_name("SMath::Mat44"), native_byte_size(64), native_byte_align(16), cpp({include_files=
        ["Shared/Math/SMathInclude.h"]});
}

union Vector2 {
    aggregate {
        x, type(float32);
        y, type(float32);
    }
    aggregate {
        u, type(float32);
        v, type(float32);
    }
    aggregate {
        array, type([float32%2]);
    }

    func $operator_conversion, explicit(false), const(true), return_type(smath.Vec4);
    func $operator_conversion, explicit(false), const(true), return_type(smath.Point);
    func $constructor_custom,  explicit(false) {v, type(smath.Vec4); }
    func $constructor_custom,  explicit(false) {v, type(smath.Point); }
}
```

Aliasing foreign types into the DDL allowed us to avoid problems such as creating yet another math library for SmSchema.

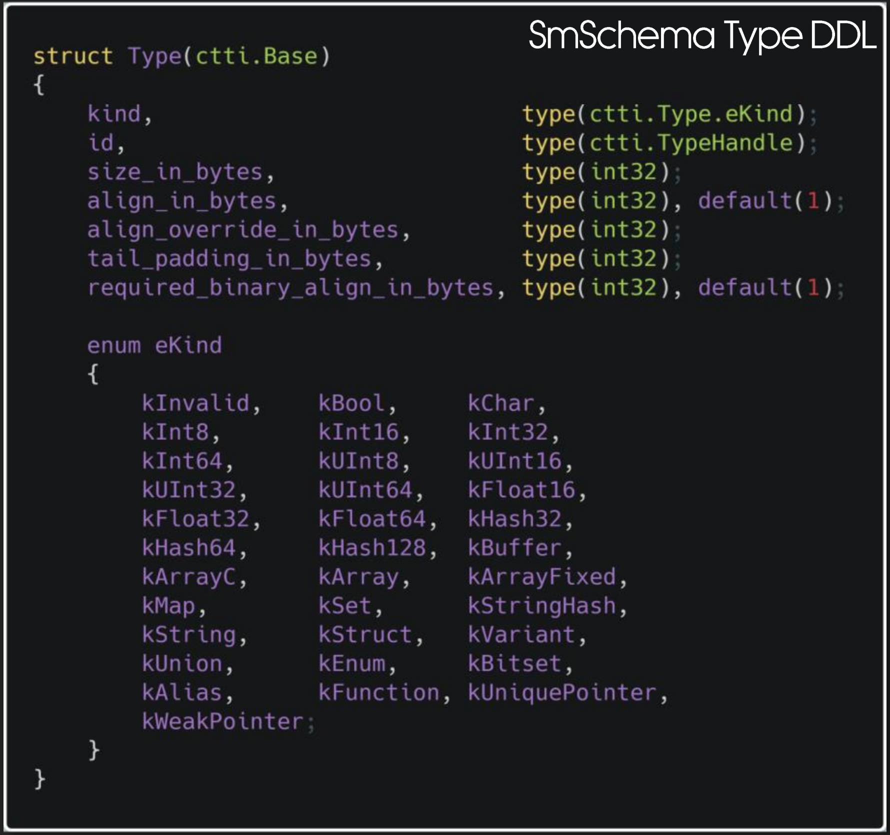
Santa Monica Studio

With foreign type aliasing, we can annotate and define the type meta-data needed by the system directly in the schema, in order to produce the correct serialization behavior. This type of solution is applicable to so many different scenarios, such as introducing SDK texture formats and animation formats.

## Compile-time Type Information

CTTI is the schema of the language.

It defines the code-generation-time and compile-time reflection of all SmSchema concepts.

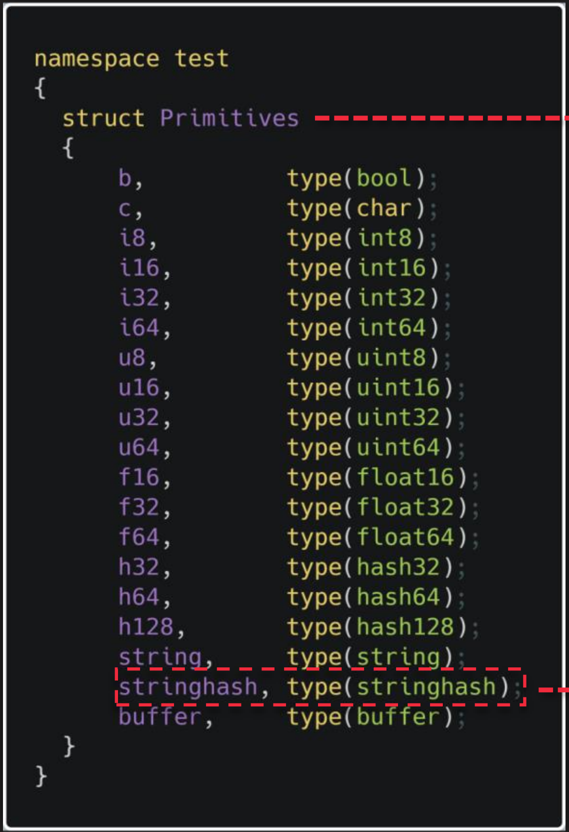

SmSchema Type DDL

```
struct Type(ctti.Base)
{
    kind,                               type(ctti.Type.eKind);
    id,                                 type(ctti.TypeHandle);
    size_in_bytes,                      type(int32);
    align_in_bytes,                     type(int32), default(1);
    align_override_in_bytes,            type(int32);
    tail_padding_in_bytes,              type(int32);
    required_binary_align_in_bytes,     type(int32), default(1);

    enum eKind
    {
        kInvalid,     kBool,      kChar,
        kInt8,        kInt16,     kInt32,
        kInt64,       kUInt8,     kUInt16,
        kUInt32,      kUInt64,    kFloat16,
        kFloat32,     kFloat64,   kHash32,
        kHash64,      kHash128,   kBuffer,
        kArrayC,      kArray,     kArrayFixed,
        kMap,         kSet,       kStringHash,
        kString,      kStruct,    kVariant,
        kUnion,       kEnum,      kBitset,
        kAlias,       kFunction,  kUniquePointer,
        kWeakPointer;
    }
}
```

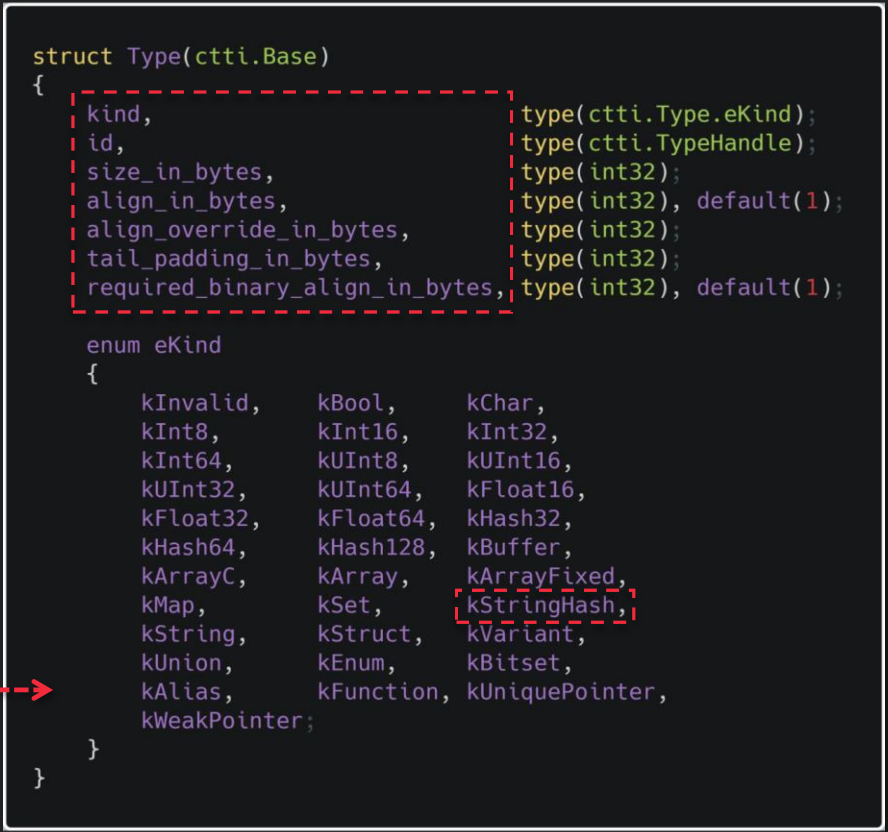
Santa Monica Studio

Following the DDL we have the CTTI, or the compile-time type information. The CTTI is the core of SmSchema, as it is the schema definition of the language itself. It defines all of the meta-data properties associated with our type and declaration system, and is reflect-able at a few points in the SmSchema ecosystem.

## A user-defined type will result in a CTTI Type instance.



```
namespace test
{
  struct Primitives
  {
      b,          type(bool);
      c,          type(char);
      i8,         type(int8);
      i16,        type(int16);
      i32,        type(int32);
      i64,        type(int64);
      u8,         type(uint8);
      u16,        type(uint16);
      u32,        type(uint32);
      u64,        type(uint64);
      f16,        type(float16);
      f32,        type(float32);
      f64,        type(float64);
      h32,        type(hash32);
      h64,        type(hash64);
      h128,       type(hash128);
      string,     type(string);
      stringhash, type(stringhash);
      buffer,     type(buffer);
  }
}
```

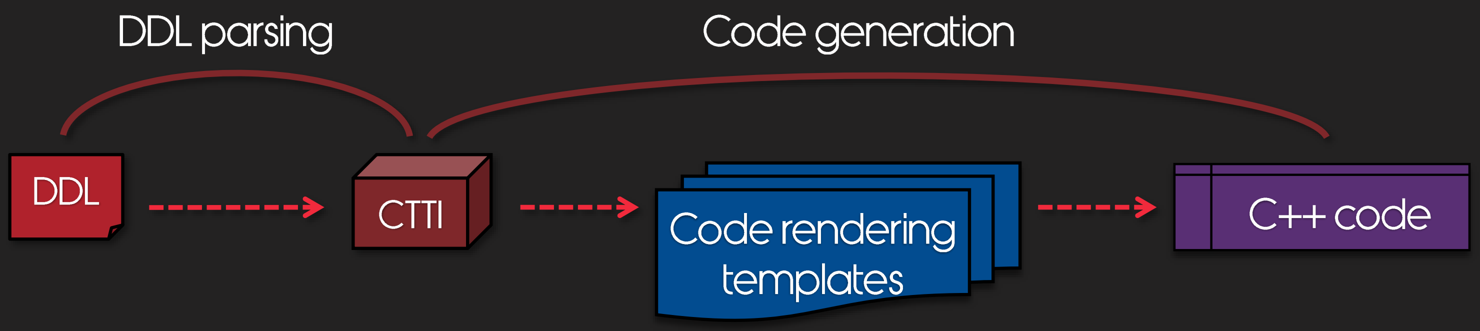


```
struct Type(ctti.Base)
{
    kind,                             type(ctti.Type.eKind);
    id,                               type(ctti.TypeHandle);
    size_in_bytes,                    type(int32);
    align_in_bytes,                   type(int32), default(1);
    align_override_in_bytes,          type(int32);
    tail_padding_in_bytes,            type(int32);
    required_binary_align_in_bytes,   type(int32), default(1);

    enum eKind
    {
        kInvalid,     kBool,       kChar,
        kInt8,        kInt16,      kInt32,
        kInt64,       kUInt8,      kUInt16,
        kUInt32,      kUInt64,     kFloat16,
        kFloat32,     kFloat64,    kHash32,
        kHash64,      kHash128,    kBuffer,
        kArrayC,      kArray,      kArrayFixed,
        kMap,         kSet,        kStringHash,
        kString,      kStruct,     kVariant,
        kUnion,       kEnum,       kBitset,
        kAlias,       kFunction,   kUniquePointer,
        kWeakPointer;
    }
}
```

Santa Monica Studio

Conceptually, a CTTI type instance is associated with any native or user-defined type. Within the CTTI definition we can specify whatever attributes matter to us, but it's dependent on our use case. While having the CTTI opens up the opportunity to implement a wide range features and analytics such as RTTI,  it currently has a very specific function.

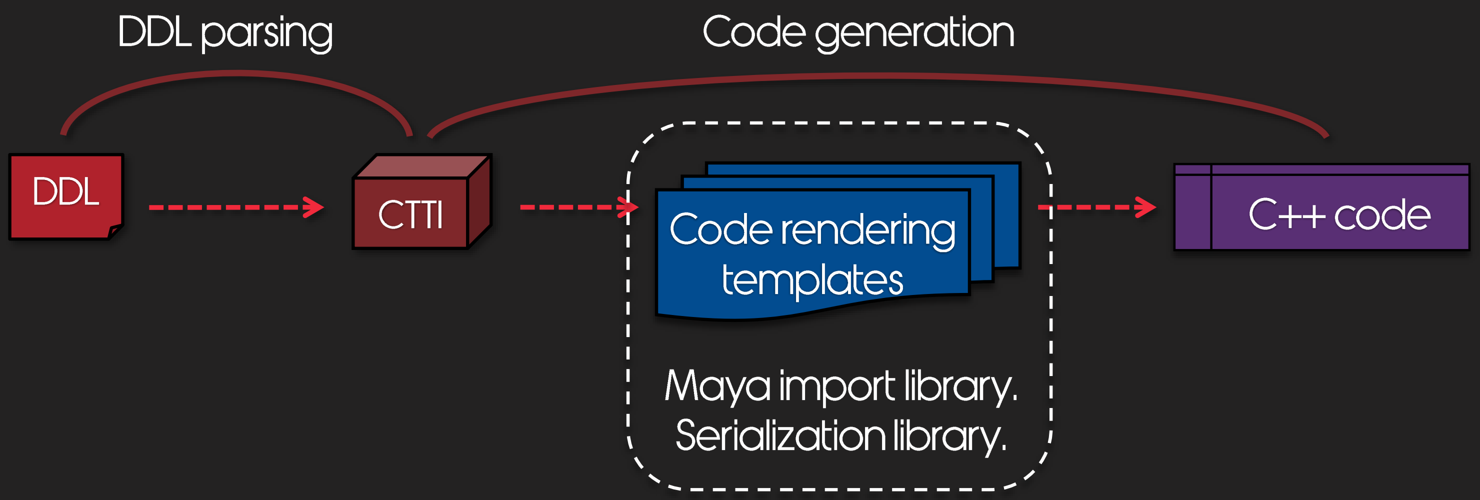
## CTTI is the primary input into the code generation templates.



Santa Monica Studio

The CTTI is the primary input into the code generation process. Given the schema of a set of types, we parse the DDL, generate the CTTI, and feed it into the code rendering templates in order to create the expected C++ behavior that we want.

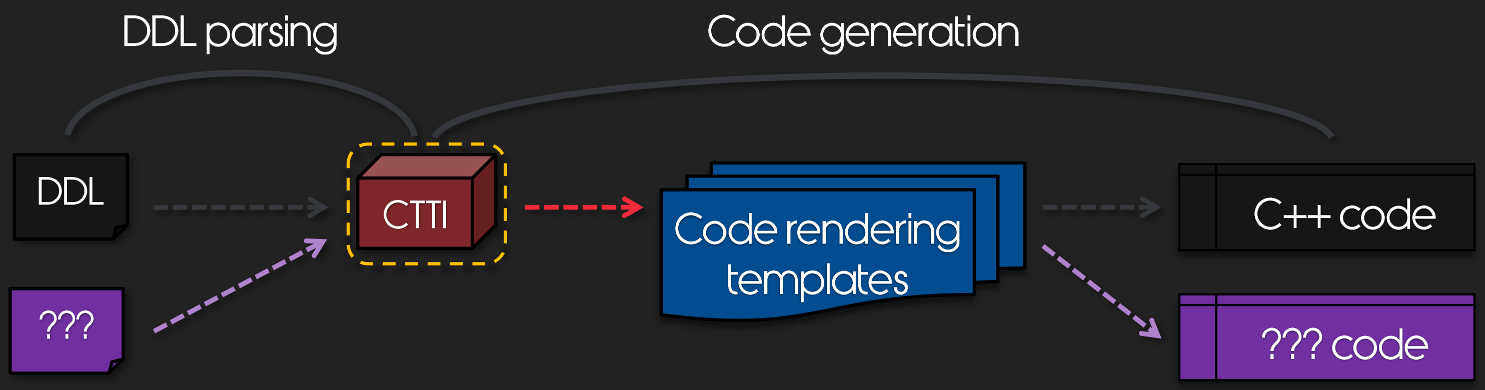
## CTTI is the primary input into the code generation templates.



Santa Monica Studio

By organizing our rendering templates into a set of segmented functionality, we can express behavior such as deciding between the generation of Maya import code or serialization code.

## CTTI is the primary input into the code generation templates.



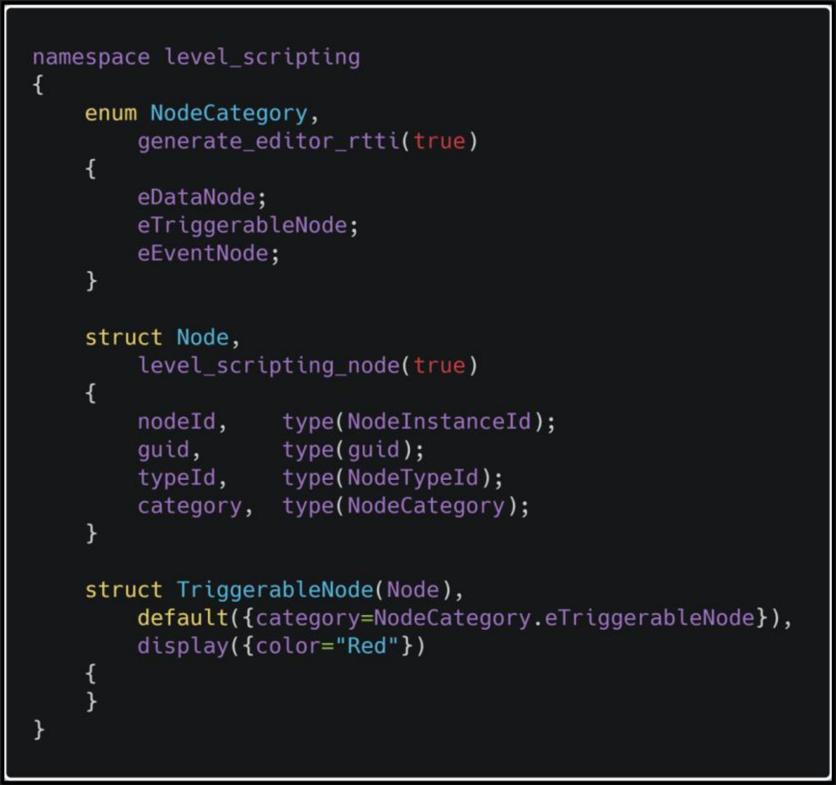
Santa Monica Studio

This type of staging architecture allows for a lot of flexibility at various points in the system. It would be accurate to say that the currently chosen DDL design and code generation capabilities are really just artifacts of our chosen problem domain. But the CTTI is the ground truth for the entire system.

## Annotations

User-driven code generation:

- Tagging types with arbitrary properties.
- Augment the generated code:
  - Without changing the parser.
  - Without adding cost to types.



```
namespace level_scripting
{
    enum NodeCategory,
        generate_editor_rtti(true)
    {
        eDataNode;
        eTriggerableNode;
        eEventNode;
    }

    struct Node,
        level_scripting_node(true)
    {
        nodeId,     type(NodeInstanceId);
        guid,       type(guid);
        typeId,     type(NodeTypeId);
        category,   type(NodeCategory);
    }

    struct TriggerableNode(Node),
        default({category=NodeCategory.eTriggerableNode}),
        display({color="Red"})
    {
    }
}
```

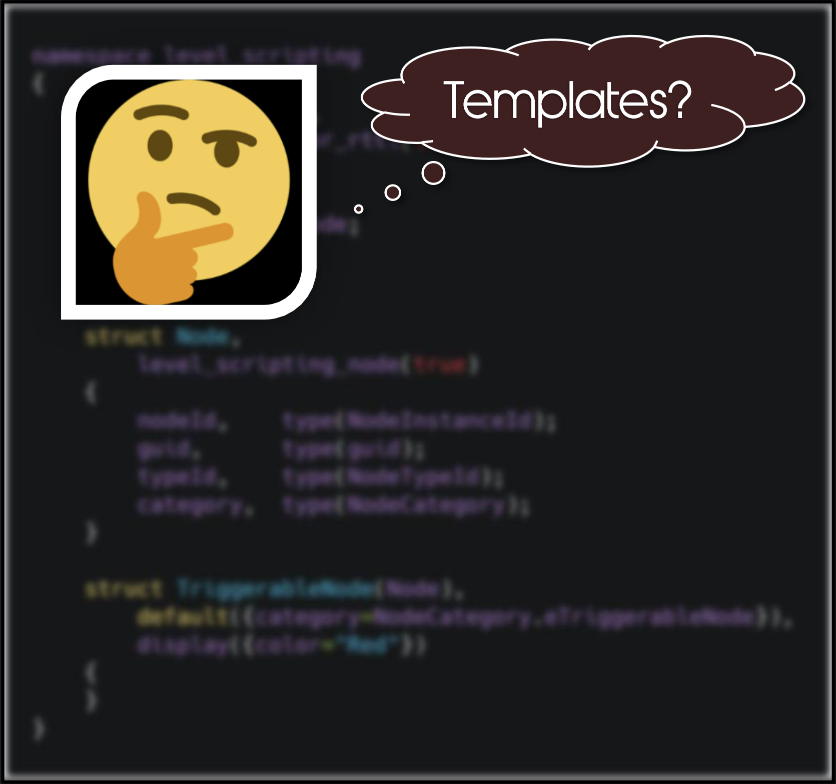
Santa Monica Studio

Having invested quite in a bit into our pipeline for code generation, with SmSchema we wanted to expose the ability to drive this behavior with an easier mechanism that didn’t require changing the parser or language.

## Annotations

User-driven code generation:

- Tagging types with arbitrary properties.
- Augment the generated code:
  - Without changing the parser.
  - Without adding cost to types.



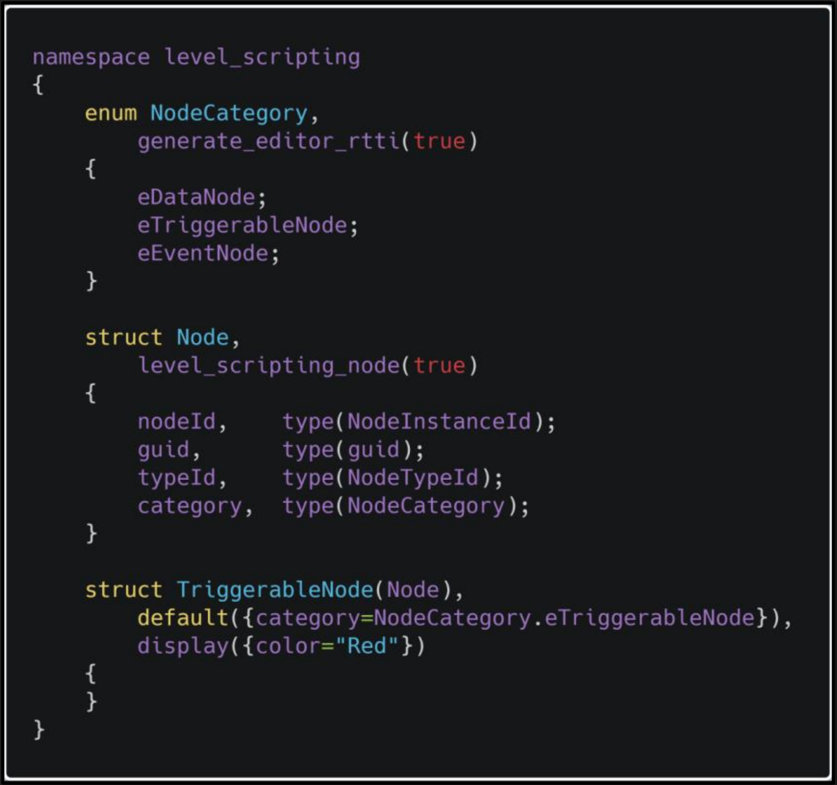
Santa Monica Studio

Some of your alarms might be going off in fear that we introduced C++ style templates, but don't worry, there would need to be a strong justification for doing so and we haven't found it yet.

## Annotations

User-driven code generation:

- Tagging types with arbitrary properties.
- Augment the generated code:
  - Without changing the parser.
  - Without adding cost to types.

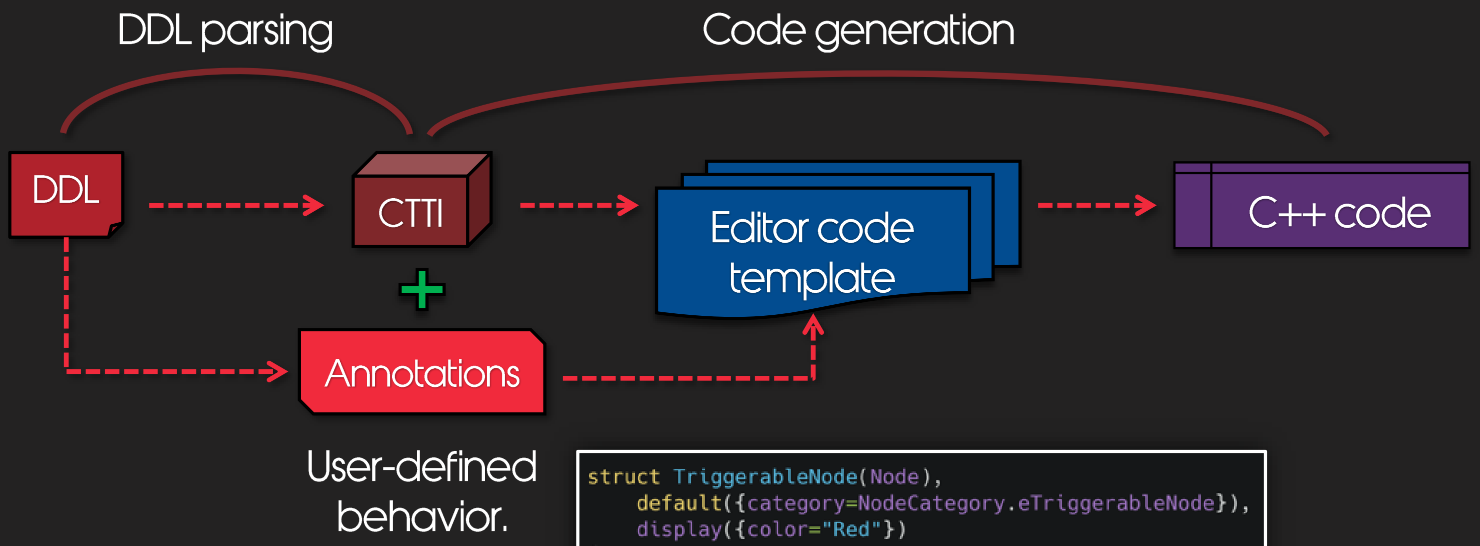


```
namespace level_scripting
{
    enum NodeCategory,
        generate_editor_rtti(true)
    {
        eDataNode;
        eTriggerableNode;
        eEventNode;
    }

    struct Node,
        level_scripting_node(true)
    {
        nodeId,    type(NodeInstanceId);
        guid,      type(guid);
        typeId,    type(NodeTypeId);
        category,  type(NodeCategory);
    }

    struct TriggerableNode(Node),
        default({category=NodeCategory.eTriggerableNode}),
        display({color="Red"})
    {
    }
}
```

Santa Monica Studio

Instead we decided to try out an appropriate and simple meta-programming construct called annotations, which augment and sit on top of the DDL and CTTI.

## Annotations provide a form of meta-programming during code generation.



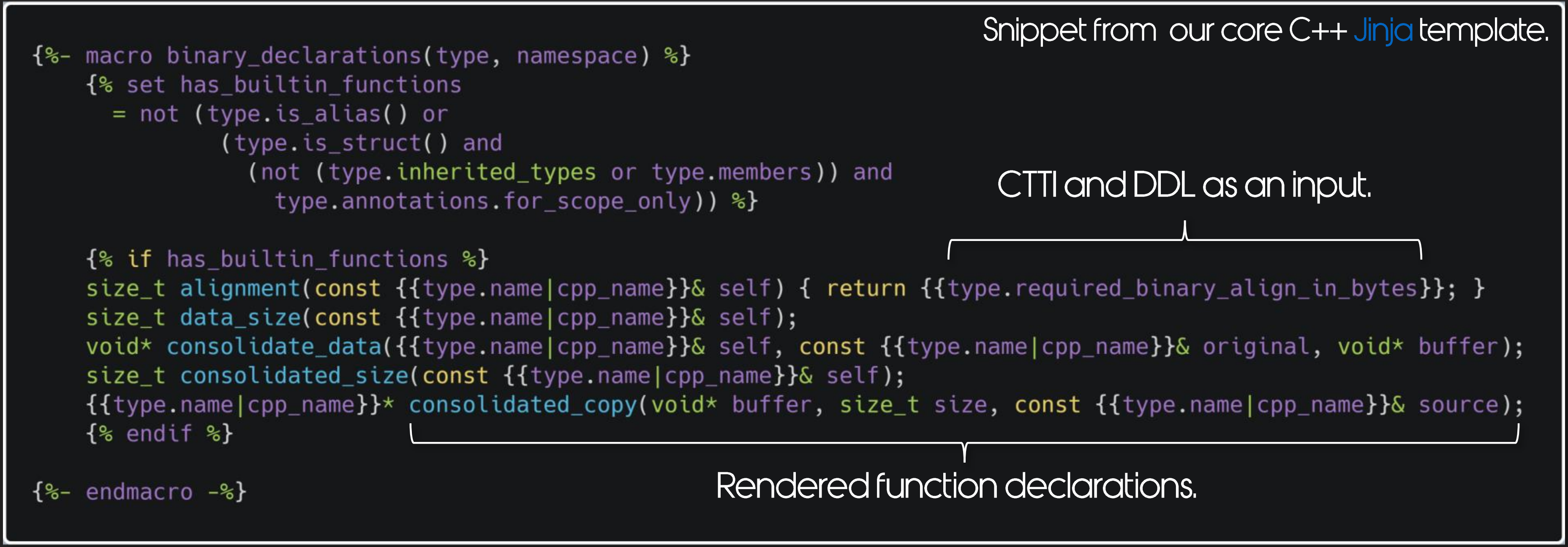
```
struct TriggerableNode(Node),
    default({category=NodeCategory.eTriggerableNode}),
    display({color="Red"})
{
}
```

Editor node color configuration.

Santa Monica Studio

Annotations are a fast path to tagging types with arbitrary properties that are only visible at the point of code generation and do not survive to the actual type definition. For example this TriggerableNode from our level scripting system has an annotation of a node display color, which is consumed by our editor's code template for establishing a visual color.

## Example Jinja Snippet


Snippet from our core C++ Jinja template.
CTTI and DDL as an input.
Rendered function declarations.

```
{%- macro binary_declarations(type, namespace) %}
    {% set has_builtin_functions
      = not (type.is_alias() or
            (type.is_struct() and
              (not (type.inherited_types or type.members)) and
                type.annotations.for_scope_only)) %}

    {% if has_builtin_functions %}
    size_t alignment(const {{type.name|cpp_name}}& self) { return {{type.required_binary_align_in_bytes}}; }
    size_t data_size(const {{type.name|cpp_name}}& self);
    void* consolidate_data({{type.name|cpp_name}}& self, const {{type.name|cpp_name}}& original, void* buffer);
    size_t consolidated_size(const {{type.name|cpp_name}}& self);
    {{type.name|cpp_name}}* consolidated_copy(void* buffer, size_t size, const {{type.name|cpp_name}}& source);
    {% endif %}

{%- endmacro -%}
```

Santa Monica Studio

Our code generation templates are quite powerful but not without their fair share of nuance and problems. For one, looking at a template for the first time it can be difficult to immediately discern the actual code being rendered versus the input incoming from the annotations and CTTI. The templates also have a very loose contract between what properties the CTTI provides and what is expected during rendering. This is still an area of research where we would like to improve usability.

## Serialization formats

Format choices driven by use case:

- JSON: diffing, merging, repairable.
- Binary: performance, compression.
- MsgPack: JSON encoded as binary.

And finally with serialization we support formats that each have a specific purpose. For example JSON serves an important function of being human readable, repairable, but also allows SmSchema to take advantage of the entire open-source ecosystem. We pursued memory-mappable binary specifically to address our requirements for the run-time.

## Serialization formats

Serialization / Deserialization benchmark:

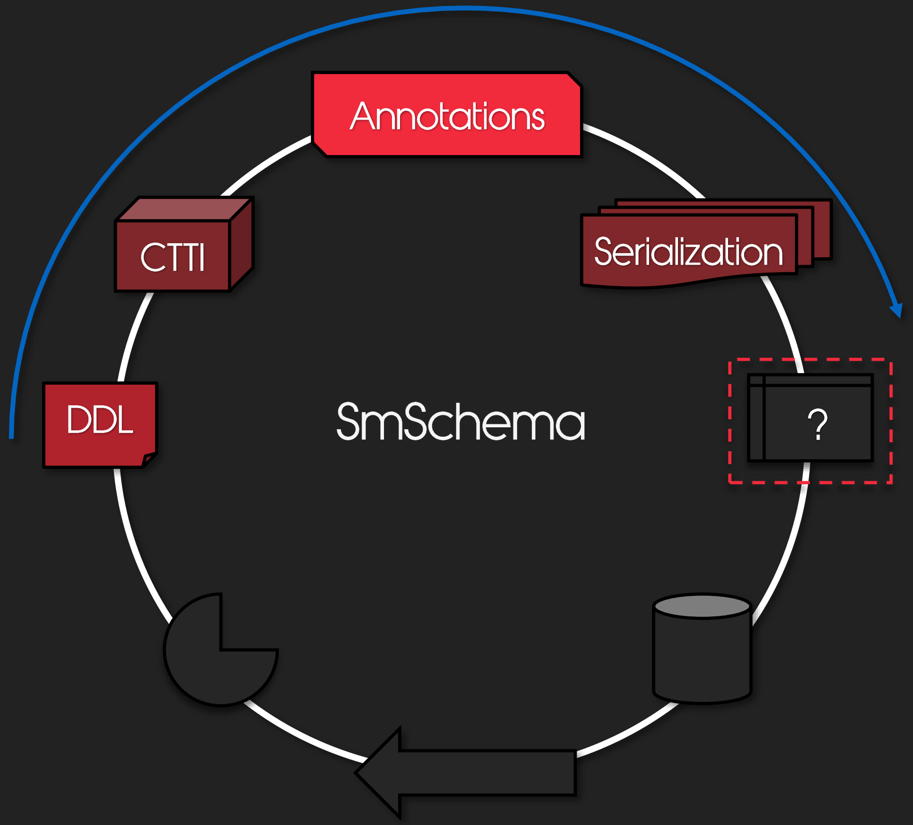
Timings for: Kratos source art, full-fidelity game rig.

| | Maya Binary | X3D Binary | X3D MsgPack | X3D JSON |
|---|---|---|---|---|
| File size (megabytes) | 331.3 | 316.1 | 583.1 | 1603.8 |
| Loading speed (seconds) | 44.4 | 6.2 | 7.1 | 19.2 |
| Saving speed (seconds) | 5.3 | 0.5 | 1.1 | 3.2 |

Santa Monica Studio

To show some numbers, our X3D scene loader used by Maya and the content build system increased performance of loading and saving of scenes by nearly an order of magnitude – even without capitalizing on known optimization techniques that we are still pursuing.

## Where does this lead?



Santa Monica Studio

We've talked a lot about serialization and data definition. But as stated before, it was clear even from our early experiments that challenges such as referencing went beyond this domain. So with the DDL, CTTI, annotations, and the serialization backend …

## How are these design pillars connected?

There’s a fair question – where does this path lead? What connects them?

## The Document model

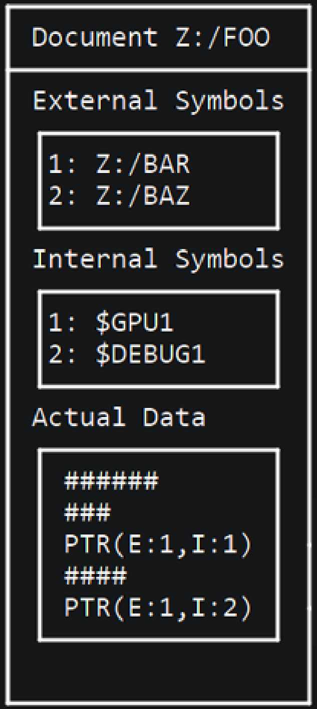
The defining data structure of SmSchema.

- Connects all other design primitives together.
- The fundamental unit of data:
  - Serialization doesn't occur at a finer granularity.

Santa Monica Studio

Together, all of the previous technologies manifest and are connected by the defining structure of SmSchema – known as the document. The document is the fundamental unit of data, and serialization does not happen a finer granularity.

## The Document model


```
Document Z:/FOO

External Symbols
1: Z:/BAR
2: Z:/BAZ

Internal Symbols
1: $GPU1
2: $DEBUG1

Actual Data
  ######
  ###
  PTR(E:1,I:1)
  ####
  PTR(E:1,I:2)
```

Document is not:

An asset.

A prefab.

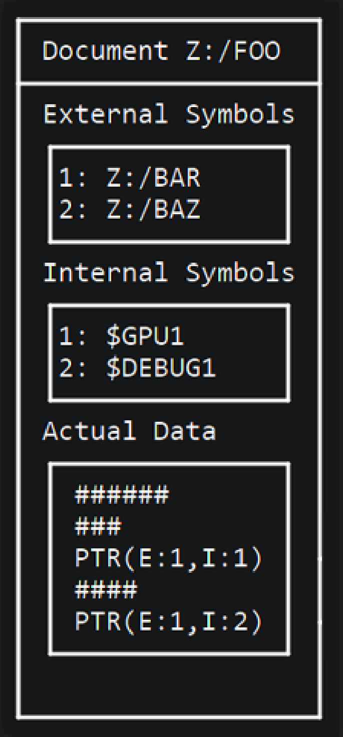
A scene.

… An abstraction.

Santa Monica Studio

The document is not an asset, it's not a prefab, it's not a scene. It's not an abstraction …

## The Document model



```
Document Z:/FOO

External Symbols
  1: Z:/BAR
  2: Z:/BAZ
Internal Symbols
  1: $GPU1
  2: $DEBUG1
Actual Data
   ######
   ###
   PTR(E:1,I:1)
   ####
   PTR(E:1,I:2)
```

Document is:

Sections of data with a header.

Starting root type:

```
struct Scene, storable(true) { /* ... */ }
```

Conceptually:

Provides a context to define the semantics of pointers.

Santa Monica Studio

The document is just sections of data with a header, and a starting root type driven by the storable annotation. But the most important property that the document provides is a context that defines the semantics of pointers – which serve as the basis of our referencing solution.

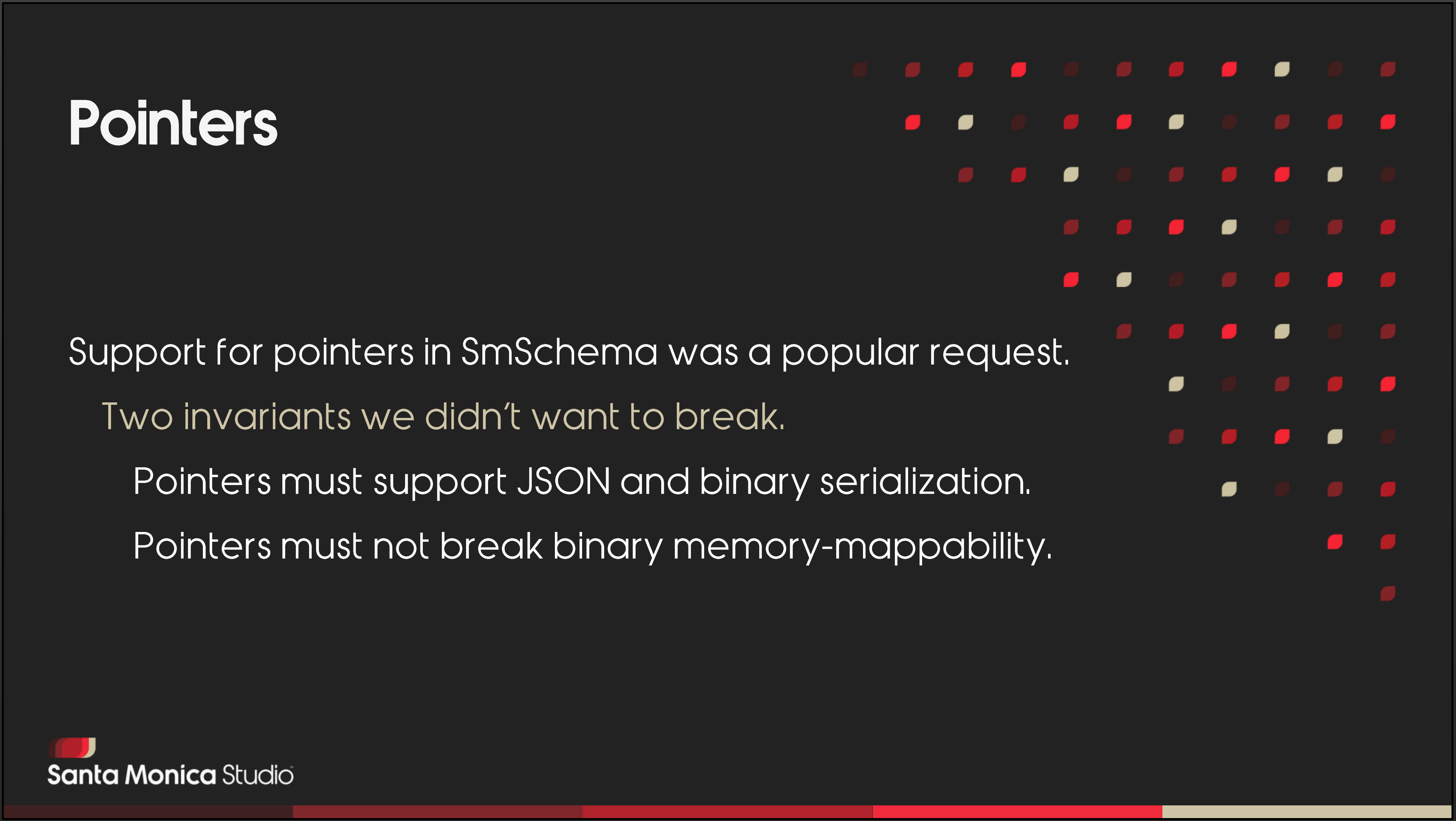
## Document insights

Documents had a profound effect on SmSchema's design.

- Referencing semantics could now be expressed coherently:
  - Documents may have internal pointers.
  - Documents externally "point" to other documents.

Santa Monica Studio

The document is a relatively simple data structure, but treating the document as a context of data that's connected and has placement in the outside world had a profound impact on the design of SmSchema. It made sense to be able to say statements such as "document's contain internal referencing" or "document's may externally reference – or point to – other documents".



## Pointers

Support for pointers in SmSchema was a popular request.

- Two invariants we didn't want to break.
  - Pointers must support JSON and binary serialization.
  - Pointers must not break binary memory-mappability.

Santa Monica Studio

Supporting pointers introduces some complexity to a system like this, so I'd like to switch gears and talk about how they works. Pointers were a very popular request, but if we wanted to tackle them we absolutely could not break the original invariants we had imposed. This meant that pointers had to fully support all of our serialization formats, alongside maintaining binary memory mappability.

## Pointers

Provide a set of orthogonal features:

- Ownership semantics: Unique, weak.
- Locality semantics: Local, Section, External.

| | Local | Section | External |
|---|---|---|---|
| Unique pointer | ✓ | ✓ | ✗ |
| Weak pointer | ✓ | ✓ | ✓ |

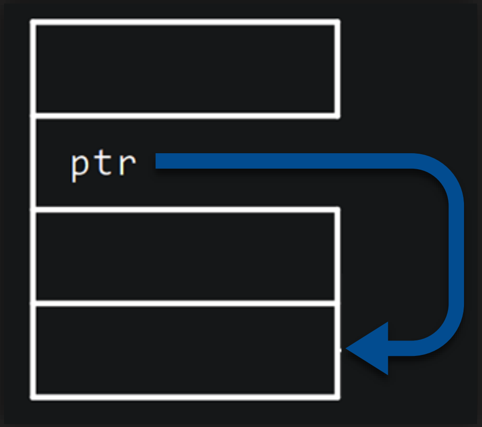
Santa Monica Studio

SmSchema pointers are defined as a space of orthogonal features that intersect between ownership and locality semantics.

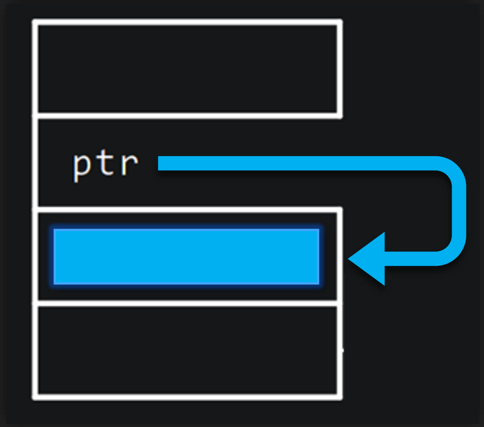
## Pointers

Ownership semantics are connected to serialization behavior:

Weak



Unique



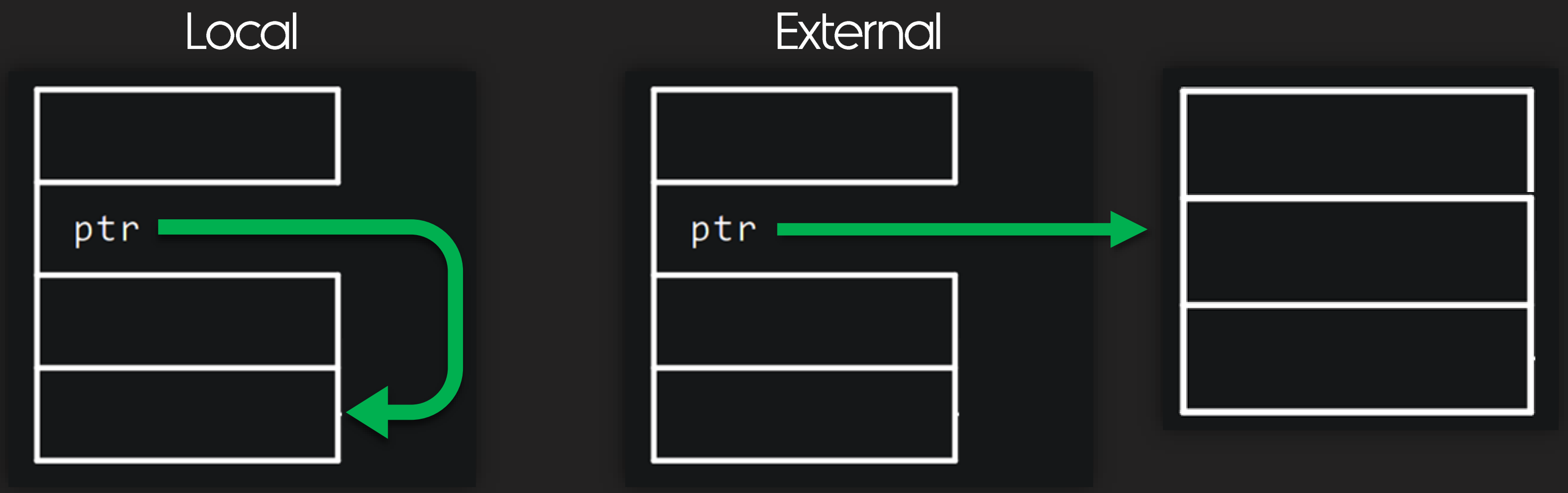
Contents are followed during serialization.

Santa Monica Studio

Ownership semantics are connected to serialization behavior. For example during serialization a unique pointer's content's will be serialized alongside the pointer itself. Whereas a weak pointer only defines a loose reference, relying on the actual content to serialize by some other means.

## Pointers

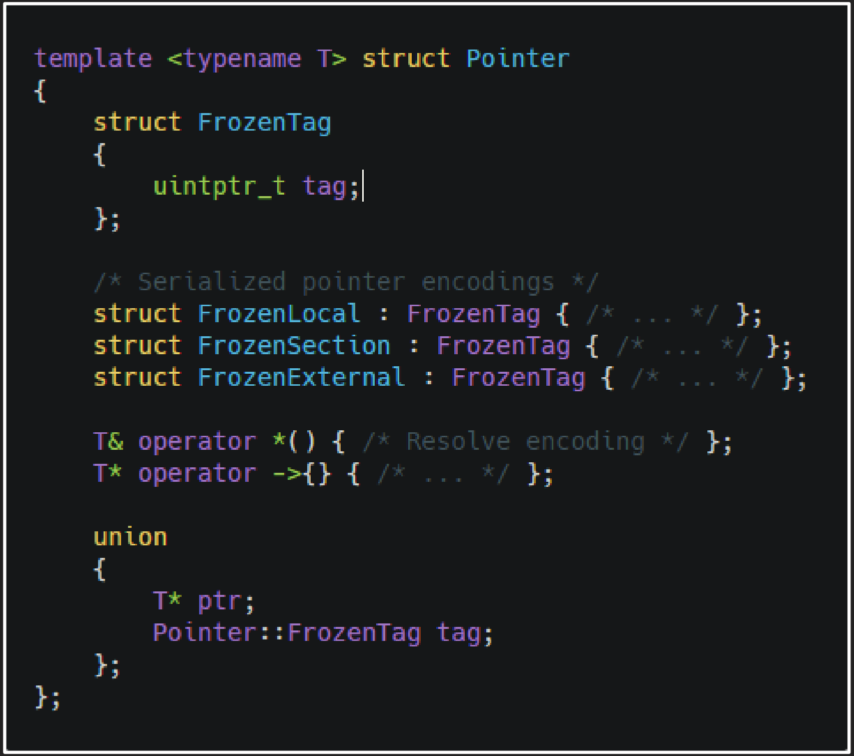
Locality semantics are connected to referencing behavior.



Santa Monica Studio

Locality semantics are connected to referencing behavior. For example, a local pointer specifically means that this pointer references content in the current section of the current document. Whereas an external pointer means that we reference content in a different document.

## Pointers



```
template <typename T> struct Pointer
{
    struct FrozenTag
    {
        uintptr_t tag;
    };

    /* Serialized pointer encodings */
    struct FrozenLocal : FrozenTag { /* ... */ };
    struct FrozenSection : FrozenTag { /* ... */ };
    struct FrozenExternal : FrozenTag { /* ... */ };

    T& operator *() { /* Resolve encoding */ };
    T* operator ->{} { /* ... */ };

    union
    {
        T* ptr;
        Pointer::FrozenTag tag;
    };
};
```

Pointers are a tagged union:

- Frozen: 64 bit serialized encodings.
- Alive: in-process address.

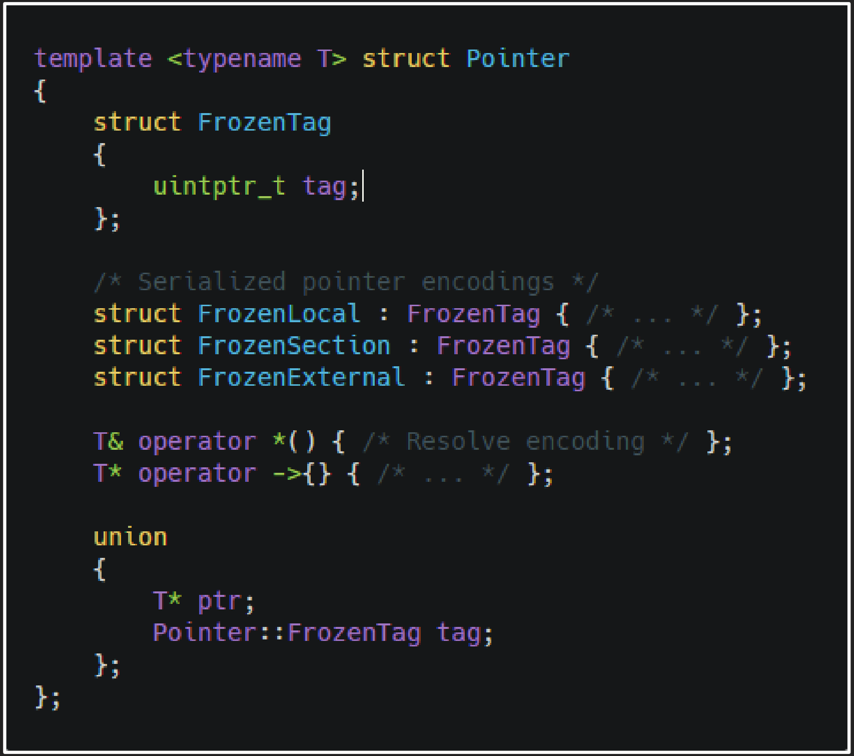
Dereference operator allows the potential for lazy unfreezing of the pointer:

- Allowing memory mappability.

Santa Monica Studio

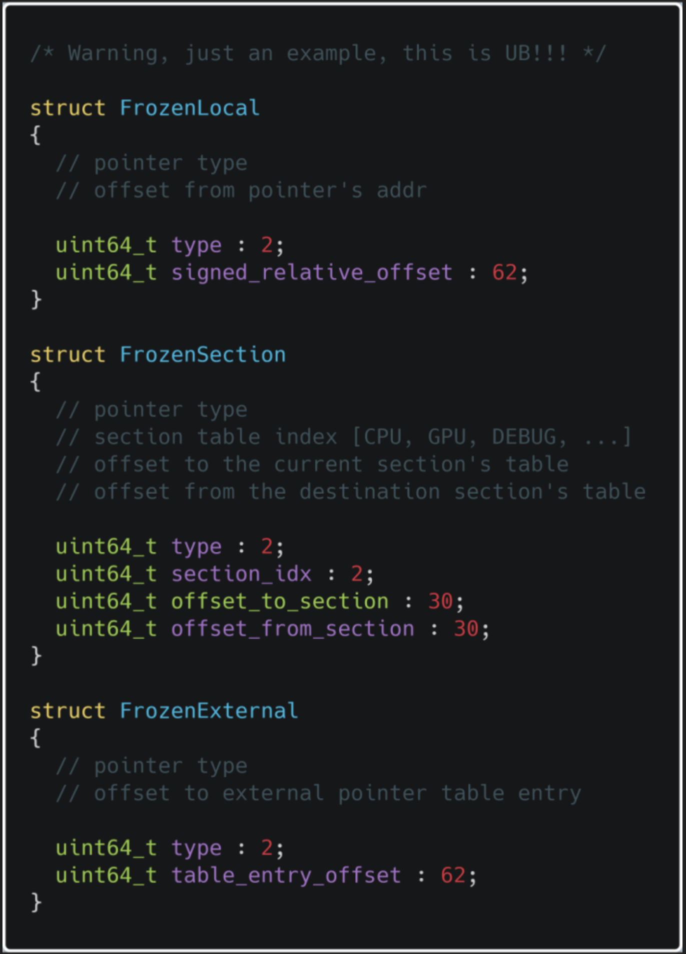
To encode these rules and achieve binary memory mappability, pointers are implemented as a 64 bit, tagged union containing an active alive or frozen state. The heavy-lifting occurs in the overloaded deference and arrow operators. By reading the high 2 bits of the pointer encoding, we can determine how the pointer should be resolved. For example the pointer may be alive in the current process address space. Alternatively it may be in a frozen, serialized state described by one of our three encodings. During deserialization, transforming a frozen pointer into an alive pointer is done through an optional process on the document known as unfreezing.

## Pointers



```
template <typename T> struct Pointer
{
    struct FrozenTag
    {
        uintptr_t tag;
    };

    /* Serialized pointer encodings */
    struct FrozenLocal : FrozenTag { /* ... */ };
    struct FrozenSection : FrozenTag { /* ... */ };
    struct FrozenExternal : FrozenTag { /* ... */ };

    T& operator *() { /* Resolve encoding */ };
    T* operator ->{} { /* ... */ };

    union
    {
        T* ptr;
        Pointer::FrozenTag tag;
    };
};
```

encoding

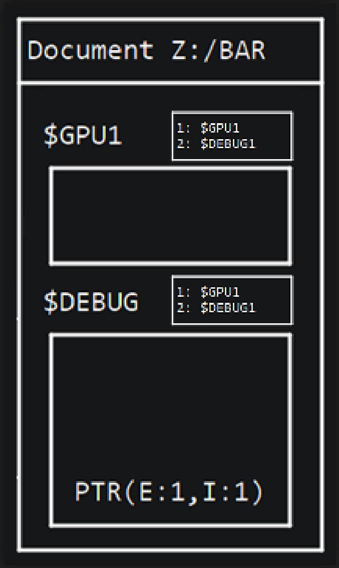


```
/* Warning, just an example, this is UB!!! */

struct FrozenLocal
{
  // pointer type
  // offset from pointer's addr

  uint64_t type : 2;
  uint64_t signed_relative_offset : 62;
}

struct FrozenSection
{
  // pointer type
  // section table index [CPU, GPU, DEBUG, ...]
  // offset to the current section's table
  // offset from the destination section's table

  uint64_t type : 2;
  uint64_t section_idx : 2;
  uint64_t offset_to_section : 30;
  uint64_t offset_from_section : 30;
}

struct FrozenExternal
{
  // pointer type
  // offset to external pointer table entry

  uint64_t type : 2;
  uint64_t table_entry_offset : 62;
}
```

Santa Monica Studio

Frozen encodings vary by the degree of outside information needed for resolution. The FrozenLocal pointer is completely self-contained in that it stores the relative signed offset from the pointer itself to the referenced content. But the FrozenExternal pointer needs the document since it contains an unsigned offset to a published external reference table that exists in the document's header. Our most complex encoding, is the FrozenSection pointer.

## Sections



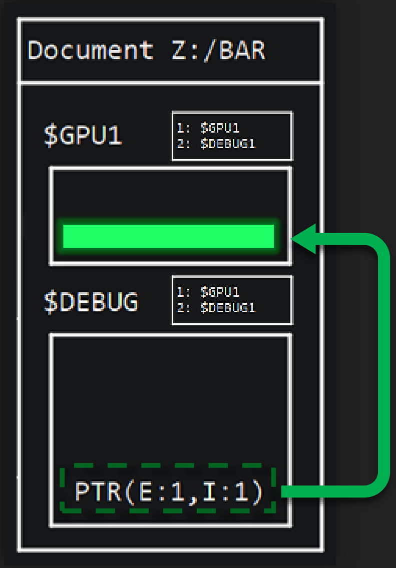
Fixed number of optional blocks:

- Main (required): CPU
- CPU Debug
- GPU
- GPU Debug

Santa Monica Studio

Document's are segmented into a fixed number of memory blocks known as sections. Each section is conceptually connected with a run-time allocator that mirrors it's use case. And all documents are required to have a main section – mapped to the general root address space – with subsequent optional sections for different purposes, such as the storage of development-only debug data or GPU data.

## Sections



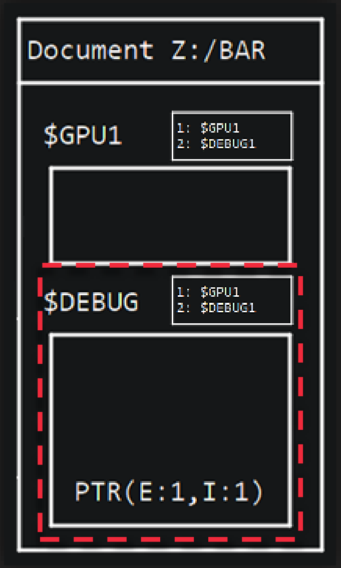
```
struct DebugTextureResource
{
    gnf, type(buffer), section(core.GPU);
}
```

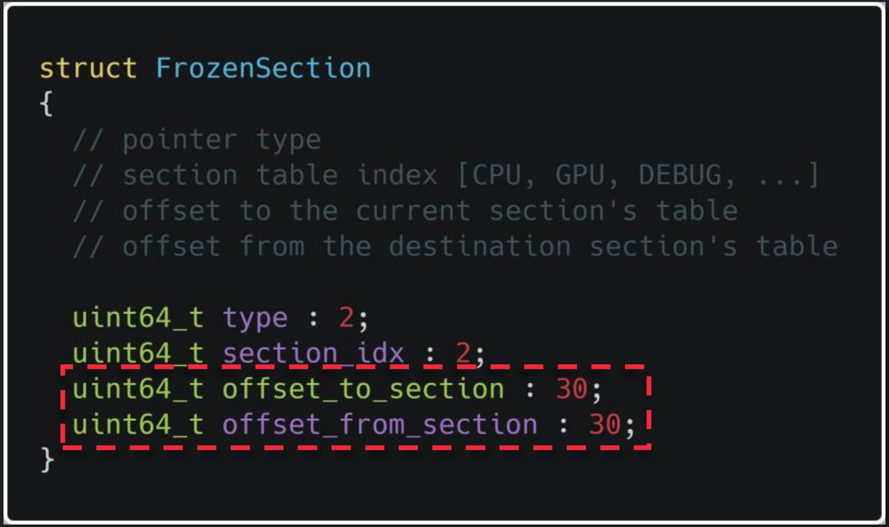
Section pointers allow for expressing specialized resources.

Santa Monica Studio

Allowing pointers to stride the document boundaries and reference resources in other sections, is a powerful mechanism for describing the layout of specialized run-time resources. But including sections as a concept in SmSchema, did impose some limitations in order to maintain binary memory mappability.

## Sections





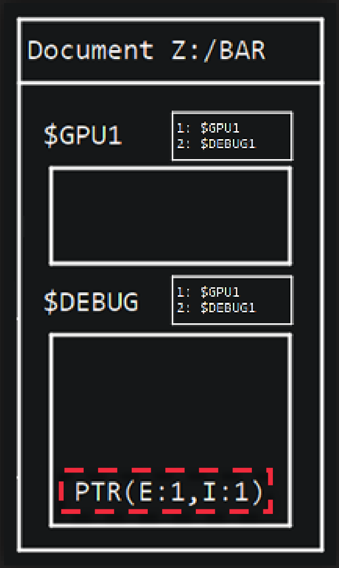
```
struct FrozenSection
{
  // pointer type
  // section table index [CPU, GPU, DEBUG, ...]
  // offset to the current section's table
  // offset from the destination section's table

  uint64_t type : 2;
  uint64_t section_idx : 2;
  uint64_t offset_to_section : 30;
  uint64_t offset_from_section : 30;
}
```

30 bit unsigned offsets
limited sections to 4GB.

Santa Monica Studio

Document sections had to be limited in size due to the strict space constraints of frozen pointers. Currently we support a maximum size of 4 GB per section, which is driven by the resolution behavior of the FrozenSection encoding. Section traversal is achieved as following:

## Sections



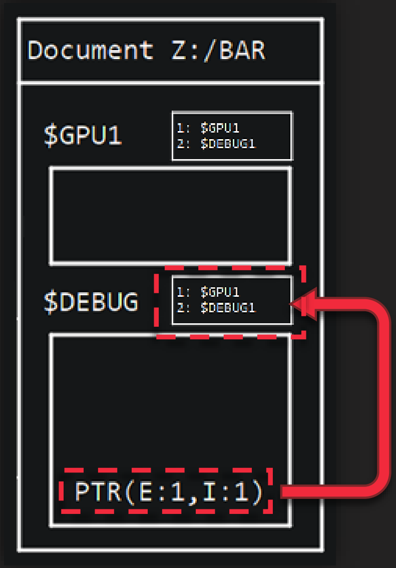
```
struct FrozenSection
{
   // pointer type
   // section table index [CPU, GPU, DEBUG, ...]
   // offset to the current section's table
   // offset from the destination section's table

   uint64_t type : 2;
   uint64_t section_idx : 2;
   uint64_t offset_to_section : 30;
   uint64_t offset_from_section : 30;
}
```

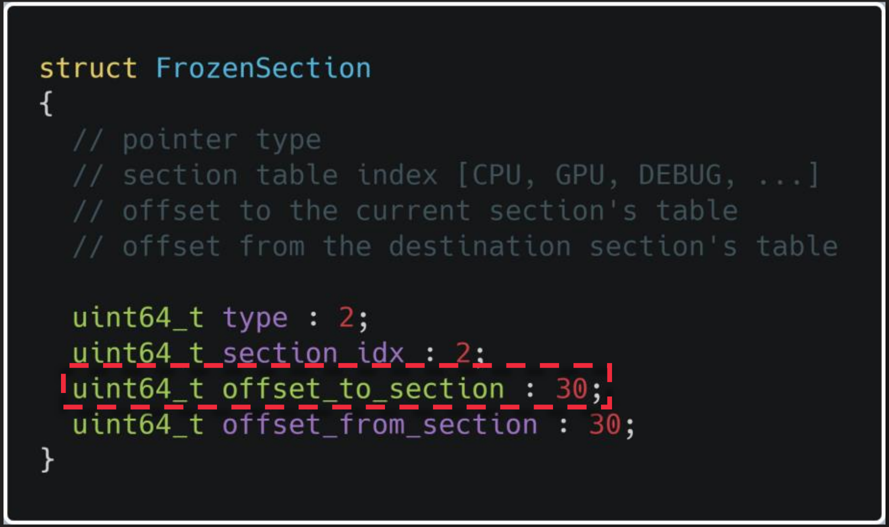
Given a frozen pointer is determined to be FrozenSection.

Santa Monica Studio

Given the discovery of a frozen section pointer defined in the type.

## Sections





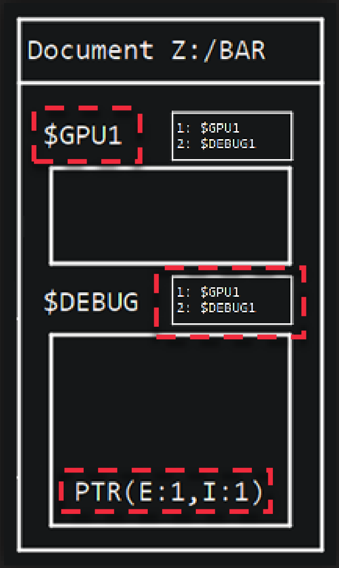
```
struct FrozenSection
{
  // pointer type
  // section table index [CPU, GPU, DEBUG, ...]
  // offset to the current section's table
  // offset from the destination section's table

  uint64_t type : 2;
  uint64_t section_idx : 2;
  uint64_t offset_to_section : 30;
  uint64_t offset_from_section : 30;
}
```

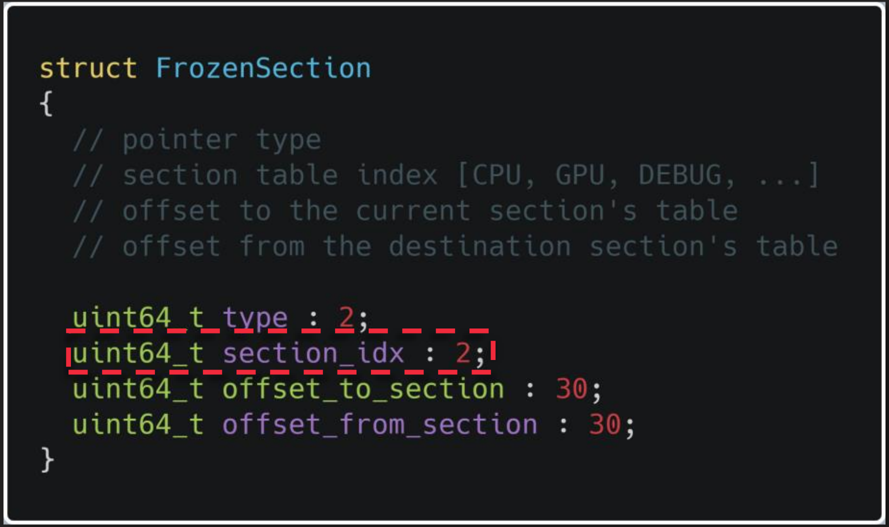
Follow the 30 bit unsigned offset to the section table.

Santa Monica Studio

We follow a 30 bit unsigned offset from the location of the current pointer, to our section's copy of the global section table.

## Sections





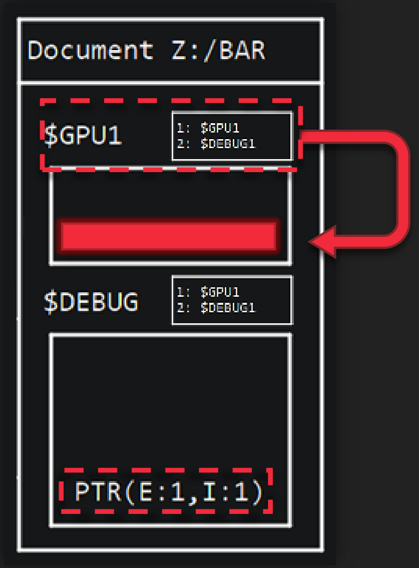
```
struct FrozenSection
{
  // pointer type
  // section table index [CPU, GPU, DEBUG, ...]
  // offset to the current section's table
  // offset from the destination section's table

  uint64_t type : 2;
  uint64_t section_idx : 2;
  uint64_t offset_to_section : 30;
  uint64_t offset_from_section : 30;
}
```

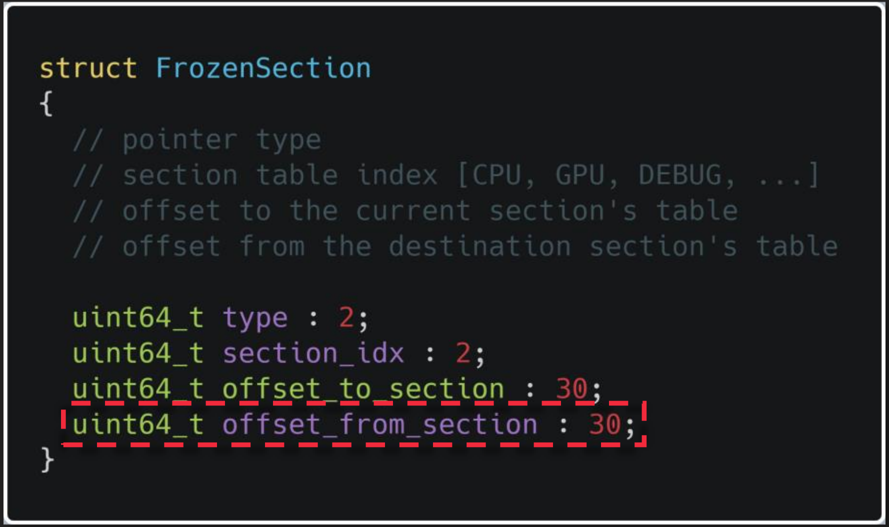
Determine the location of the destination section.

Santa Monica Studio

In this table, the pointer's section index will map to a document-space offset corresponding to the location of the destination section.

## Sections





```
struct FrozenSection
{
  // pointer type
  // section table index [CPU, GPU, DEBUG, ...]
  // offset to the current section's table
  // offset from the destination section's table

  uint64_t type : 2;
  uint64_t section_idx : 2;
  uint64_t offset_to_section : 30;
  uint64_t offset_from_section : 30;
}
```

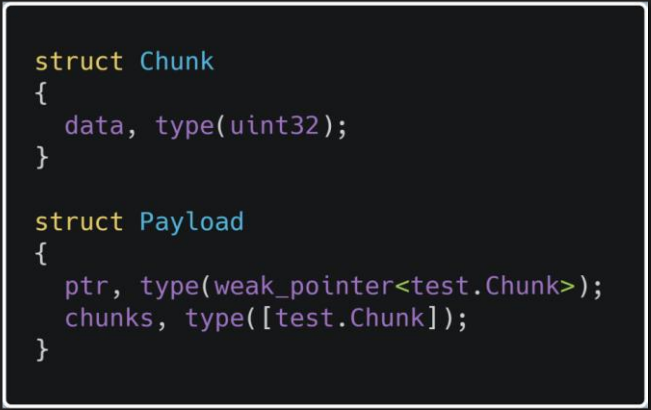
Follow the 30 bit unsigned offset from the destination section table.

Santa Monica Studio

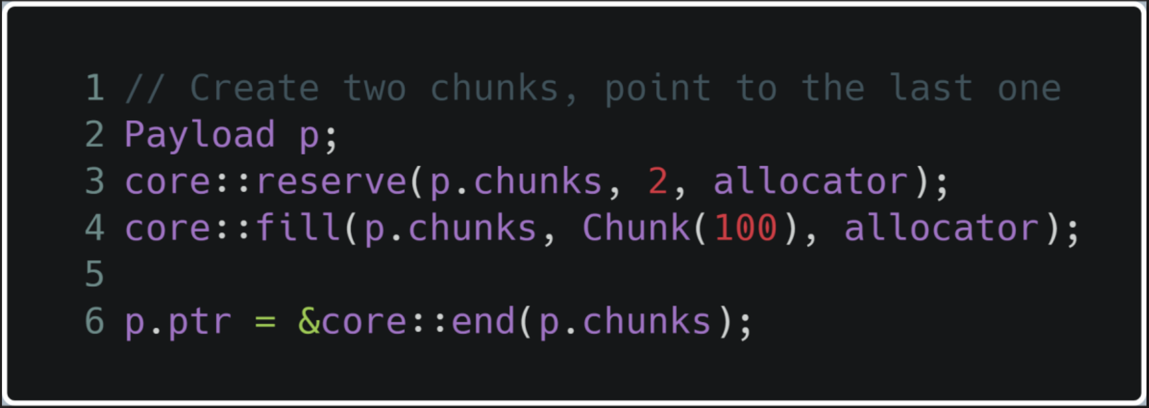
And finally, we resolve the pointer by following the unsigned offset leading from the entry of the destination section.

# Pointers

Began to present new issues in the serialization, such as the order of events.



```
struct Chunk
{
  data, type(uint32);
}

struct Payload
{
  ptr, type(weak_pointer<test.Chunk>);
  chunks, type([test.Chunk]);
}
```

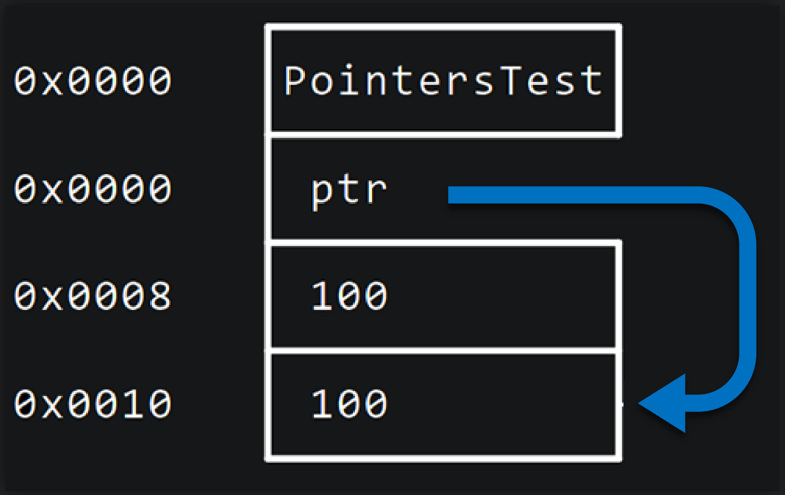


```
// Create two chunks, point to the last one
Payload p;
core::reserve(p.chunks, 2, allocator);
core::fill(p.chunks, Chunk(100), allocator);

p.ptr = &core::end(p.chunks);
```

Santa Monica Studio

Pointers are powerful, but they certainly come with tradeoffs. Their existence complicates the serialization model quite a bit. I call this the "Hello World" of pointer serialization. In this schema, a pointer in Payload to an arbitrary chunk is defined before the array of chunks itself.

# Pointers



Serialization occurs in the order of the schema definition:

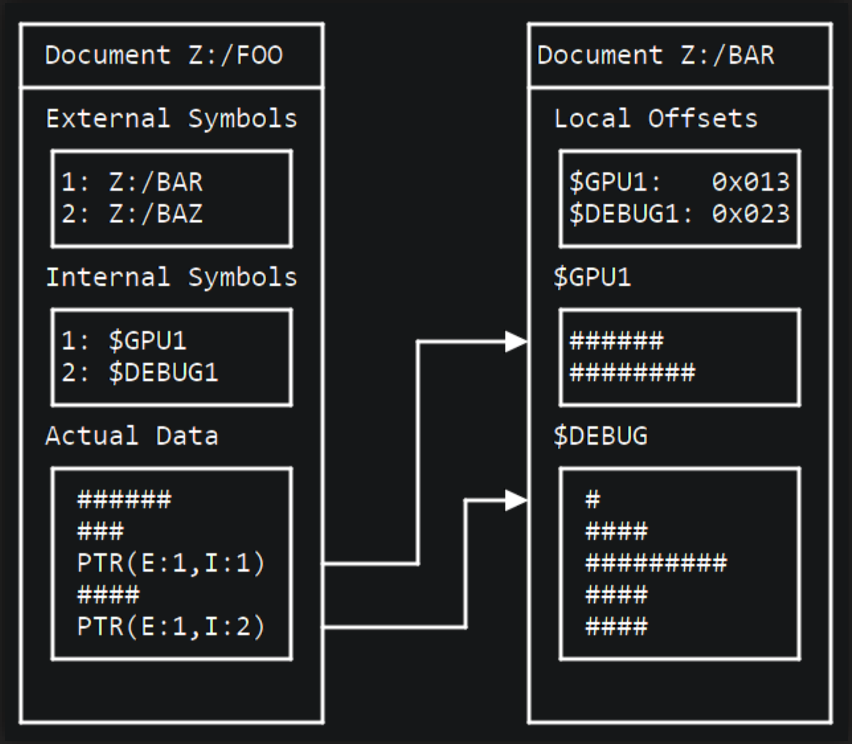
Ptr serializes to the document before the last chunk – which it is pointing to.

Difficult to resolve what the baked binary offset should be in one pass.

Santa Monica Studio

But during serialization, we will encounter the pointer before the array. This creates a problem, because it would be useful to know the physical address of where the array will be serialized before we establish the pointer's frozen encoding. It's true that problems like this complicate the serialization of the document -- due to requiring multiple passes – for our purposes it was the correct tradeoff to make; because it allowed for a zero-cost deserialization step in the run-time due to binary memory mappability.

## The Document model

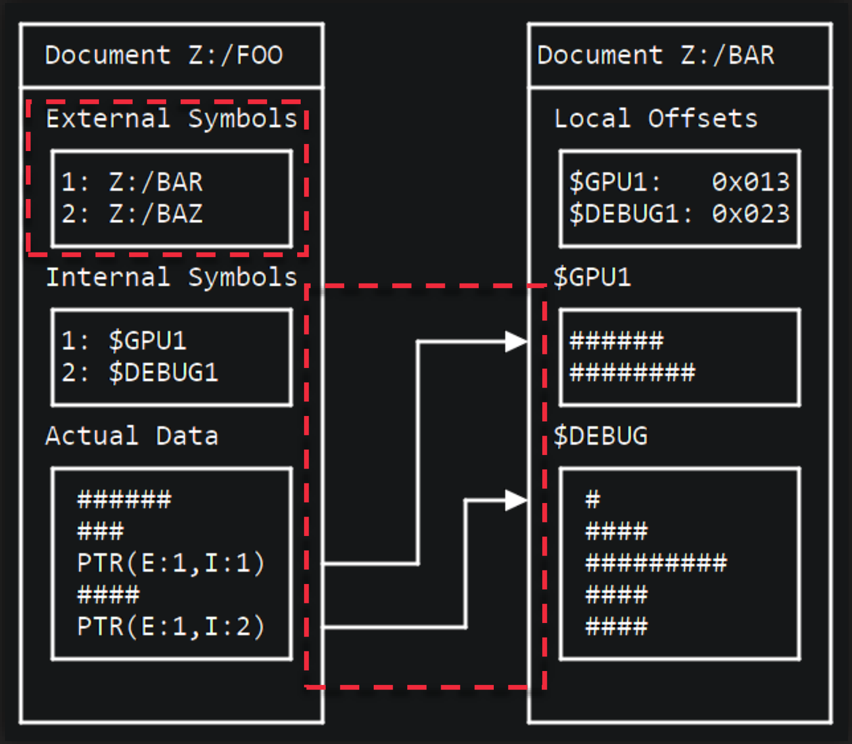


With the advent of pointers, documents became a powerful tool.

Santa Monica Studio

With pointers, documents are a very powerful way to represent general networks of interconnected data.

# The Document model

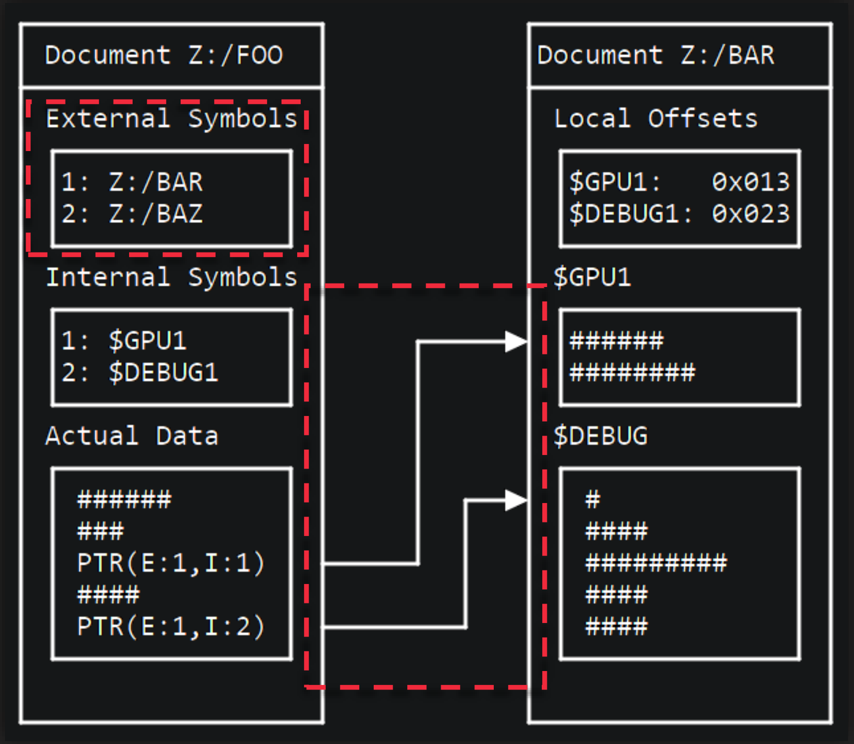


External pointers, allow documents to define networks of dependencies.

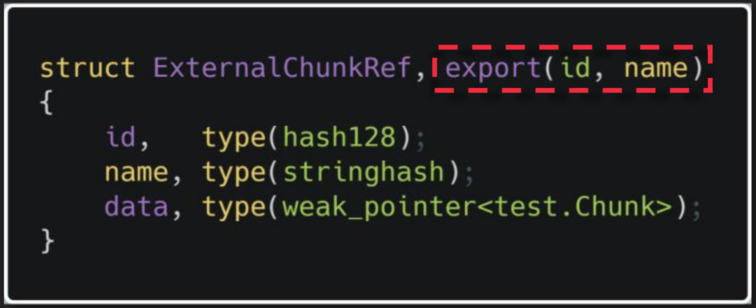
Santa Monica Studio

Longstanding problems such as content dependency resolution and discovery are a natural part of the data model, since external pointers are globally published.

## The Document model



Through the DDL, we could push programmers to create contracts that were enforceable.

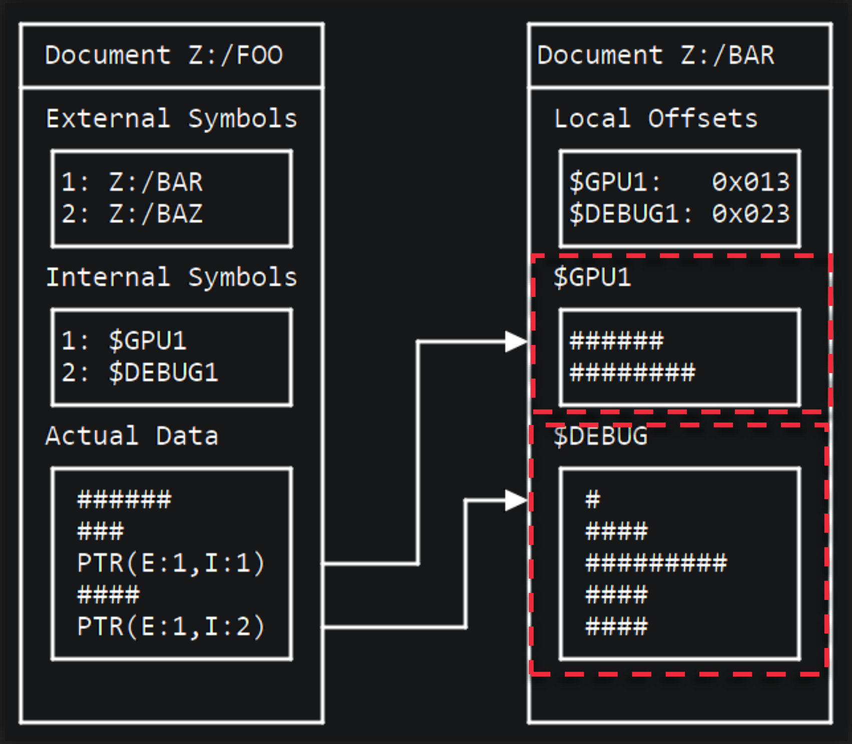


```
struct ExternalChunkRef, export(id, name)
{
    id,   type(hash128);
    name, type(stringhash);
    data, type(weak_pointer<test.Chunk>);
}
```

Santa Monica Studio

By leveraging investments made in annotations and the DDL, we can now push programmers to establish referencing contracts that are enforceable.

## The Document model



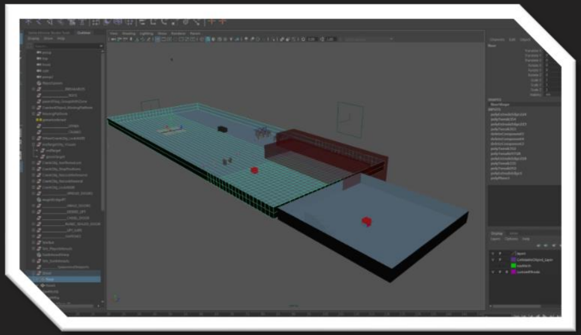
Sectioning allows segmentation of the document into arenas:

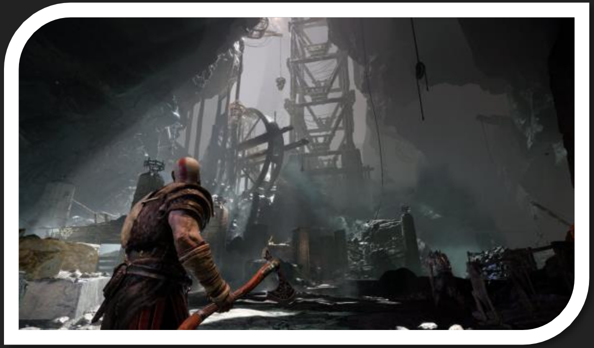
- Connected with certain allocators.
  - GPU data.
  - Non-shipping debug data.

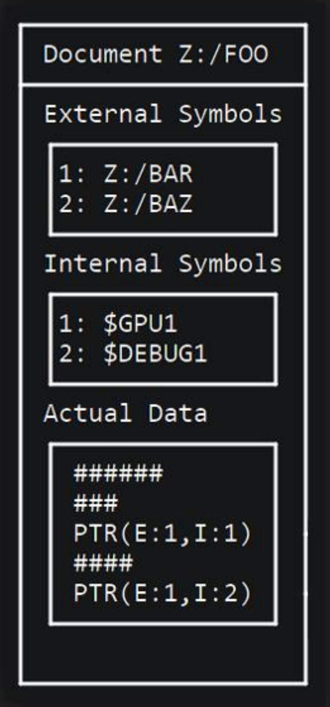
Santa Monica Studio

By segmenting the data into discreet sections, we have a place to put development only information so we can reason about it's cost and impact, and connect it easily to allocation behavior in our run-time.

## Offline and run-time can now leverage the same technology.





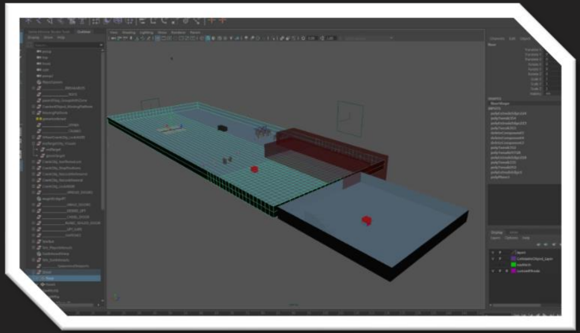


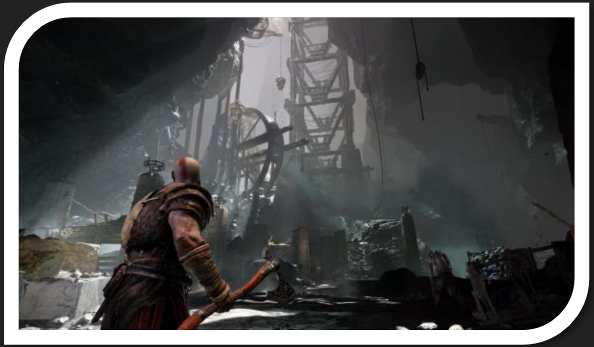
```
Document Z:/FOO

External Symbols
  1: Z:/BAR
  2: Z:/BAZ

Internal Symbols
  1: $GPU1
  2: $DEBUG1

Actual Data
   ######
   ###
   PTR(E:1,I:1)
   ####
   PTR(E:1,I:2)
```

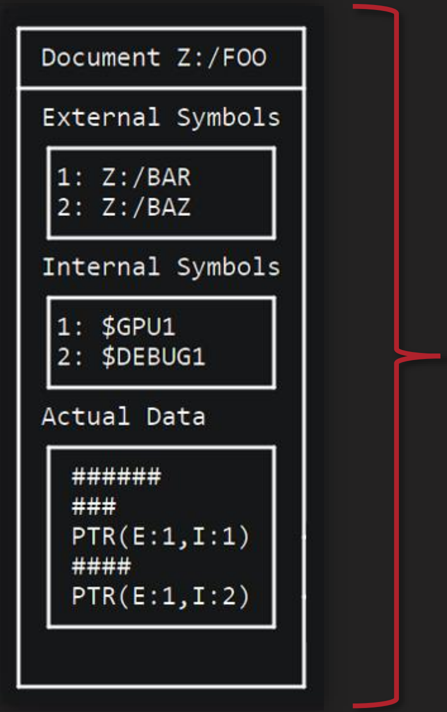
Santa Monica Studio

With the document, we’re now designing workflows where our source data and run-time data are almost the same thing. Often only requiring a minimum subset of transformations we needed before such as …

## Offline and run-time can now leverage the same technology.







```
Document Z:/FOO

External Symbols
  1: Z:/BAR
  2: Z:/BAZ

Internal Symbols
  1: $GPU1
  2: $DEBUG1

Actual Data
  ######
  ###
  PTR(E:1,I:1)
  ####
  PTR(E:1,I:2)
```

Transforms:

Json «-» Binary.

Data lineariziation.

Data stripping.

Santa Monica Studio

Automatically converting from source json to run-time binary as a format. Or potentially rearranging the data into structures that make sense for our run-time behavior. Or stripping fields and entire sections that don't have use in our shipping product.

## Document insights

If the document is not:

- An asset?
- A prefab?
- A scene?

Santa Monica Studio

The document gives us all of these capabilities, but if it’s not an asset, a prefab or scene …

## Document insights

If the document is not:

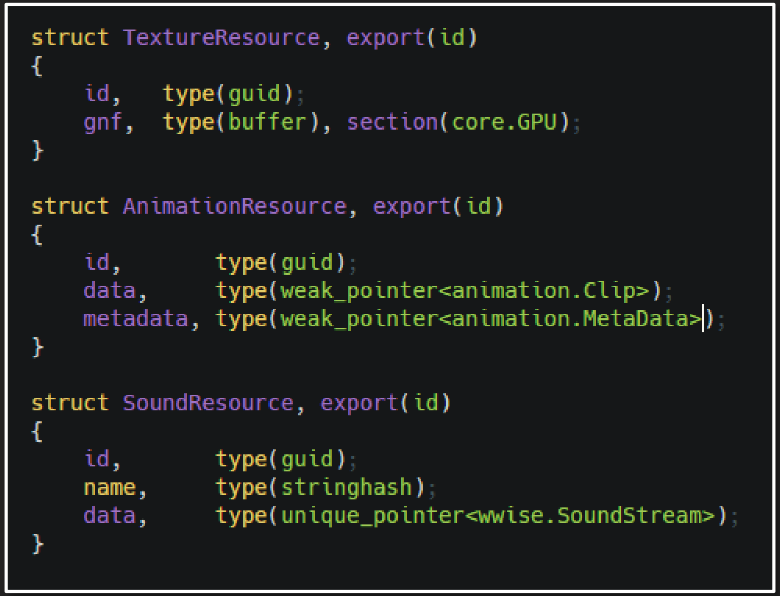
- An asset?
- A prefab?
- A scene?

Then what is it?

Santa Monica Studio

Then what is it? Well, that’s the deeper point.

## A set of streaming resources for the engine.



```
struct TextureResource, export(id)
{
    id,   type(guid);
    gnf,  type(buffer), section(core.GPU);
}

struct AnimationResource, export(id)
{
    id,       type(guid);
    data,     type(weak_pointer<animation.Clip>);
    metadata, type(weak_pointer<animation.MetaData>);
}

struct SoundResource, export(id)
{
    id,       type(guid);
    name,     type(stringhash);
    data,     type(unique_pointer<wwise.SoundStream>);
}
```

Santa Monica Studio

The document can be a set of shipping streaming resources for the game.

## It's a string hash reverse-lookup table.

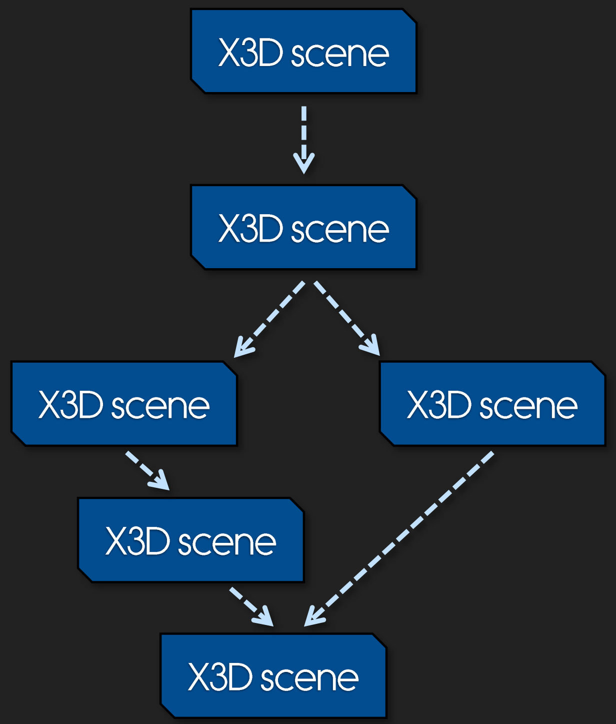
```
struct StringHashDatabase
{
    id,      type(guid);
    mapping, type(unique_pointer<{hash64: string}>), section(core.Debug);
}
```

Santa Monica Studio

It can be a string hash look-up table only used in our development workflows.

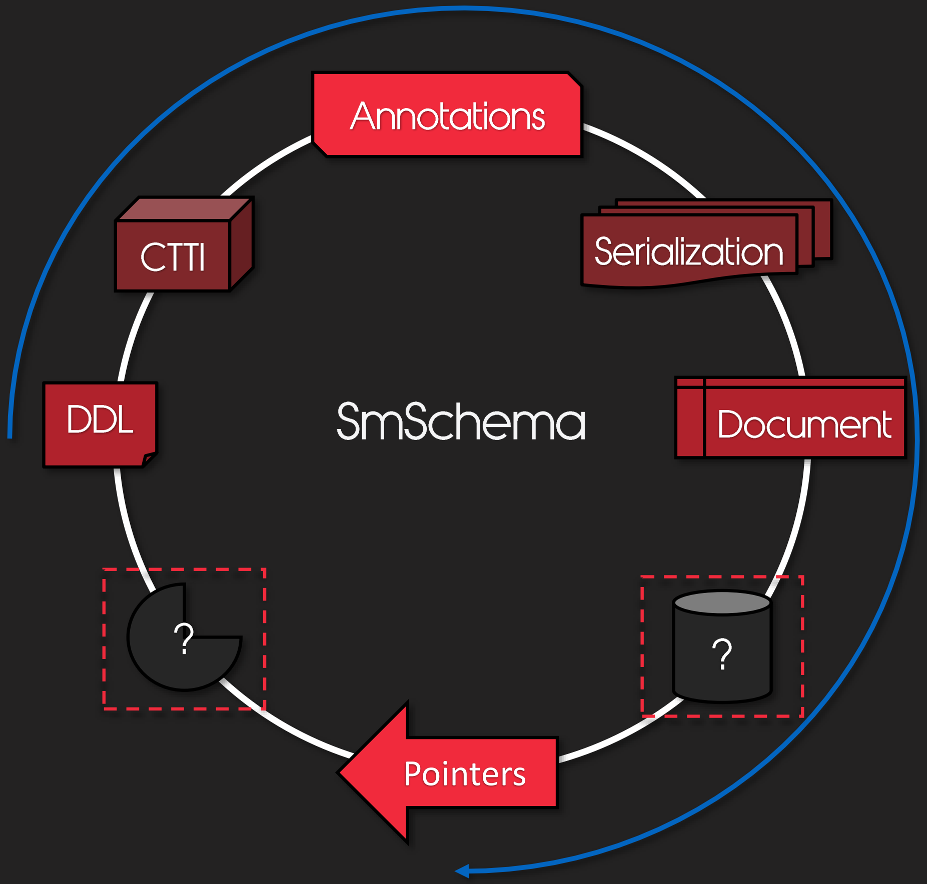
```
struct Scene
{
    original_file_path,     type(string);
    hierarchy,              type(x3d.TransformHierarchy);
    transforms,             type({hash128: x3d.Transform});
    transform_constraints,  type({hash128: x3d.TransformConstraint});
    anim_curves,            type({hash128: x3d.AnimCurve});
    cameras,                type({hash128: x3d.Camera});
    visual_meshes,          type({hash128: x3d.Visual.Mesh});
    visual_materials,       type({hash128: x3d.Visual.Material});
    rigid_bodies,           type({hash128: x3d.RigidBody});
    zones,                  type({hash128: x3d.Zone});
    collision_shapes,       type({hash128: x3d.Collision.Shape});
    collision_materials,    type({hash128: x3d.Collision.Material});
    nurbs_curves,           type({hash128: x3d.NurbsCurve});
    poly_lines,             type({hash128: x3d.PolyLine});
    clip_areas,             type({hash128: x3d.ClipArea});
    nav_obstacles,          type({hash128: x3d.NavObstacle});
    nav_meshes,             type({hash128: x3d.NavMesh});
    textures,               type({hash128: x3d.Texture});
    ref_nodes,              type({hash128: x3d.RefNode});
    game_objects_instances, type({hash128: x3d.GameObjectInstance});
    lod_groups,             type({hash128: x3d.LodGroup});
    entities,               type({hash128: x3d.Entity});
    lua_tables,             type({hash128: x3d.LuaTable});
    emitters,               type({hash128: x3d.Emitter});
    fields,                 type({hash128: x3d.Field});
    particle_systems,       type({hash128: x3d.ParticleSystem});
    lights,                 type({hash128: x3d.Light});
    light_maps,             type({hash128: x3d.NovaLightMap});
    effect_geometries,      type({hash128: x3d.EffectGeometry});
    editor_locators,        type({hash128: x3d.EditorLocator});
    sound_emitters,         type({hash128: x3d.SoundEmitter});
    sound_portals,          type({hash128: x3d.SoundPortal});
    text_objects,           type({hash128: x3d.TextObject});
    text_boxes,             type({hash128: x3d.TextBox});
    hideproxy_sets,         type([x3d.HideProxySet]);
    transform_instances,    type({hash128: x3d.TransformInstancesSet});
    maya_only_meshes,       type({hash128: x3d.Visual.Mesh});
}
```

## It's a source X3D scene graph.



Or it can even be the X3D Scene, carrying all the semantics we would need for a source data format.

## Despite present progress, opportunities remain.



Santa Monica Studio

SmSchema's first step into God of War was the X3D Scene project. And through that experience we defined a restricted set of technology that is currently powering the design of our data for many different systems, including AI behaviors, level scripting, animation, and beyond for our future products …

## Looking to the future.

Santa Monica Studio

But there are still many opportunities that remain in this space, as we look towards our current and future research.

## In progress …

As use-cases expand, so do opportunities.

⋮

External pointer resolution.

Versioning and schema invalidation.

Run-time type information.

Content build-system architecture.

⋮

In the space of an hour it isn't possible for me to go through all of the current efforts we are tackling with SmSchema – as much as I'd love to.

## In progress …

As use-cases expand, so do opportunities.

⋮

External pointer resolution.

Versioning and schema invalidation.

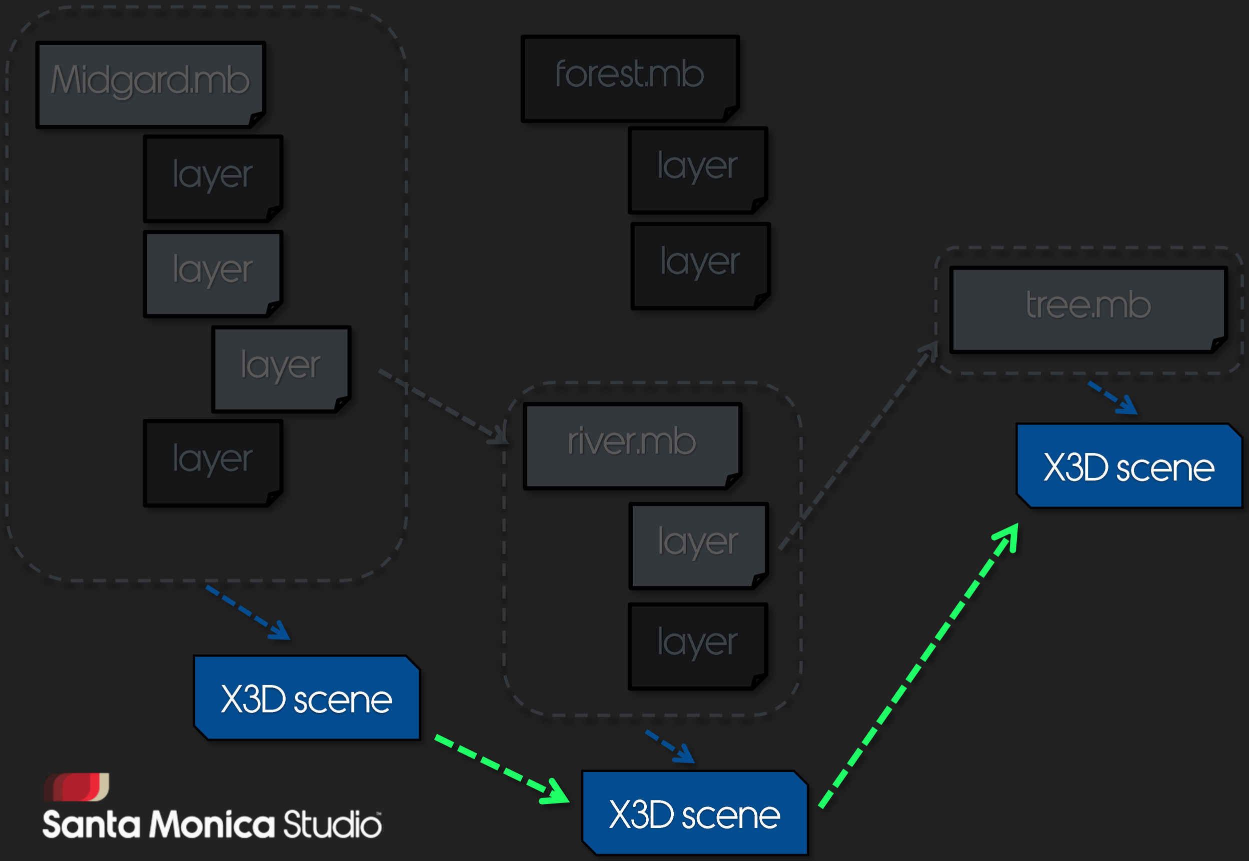
Run-time type information.

Content build-system architecture.

⋮

Instead, I'd like to focus on one particular topic – external pointer resolution – as an anecdote to showcase where we are headed.

## While external pointers expanded referencing semantics, they introduced new problems.

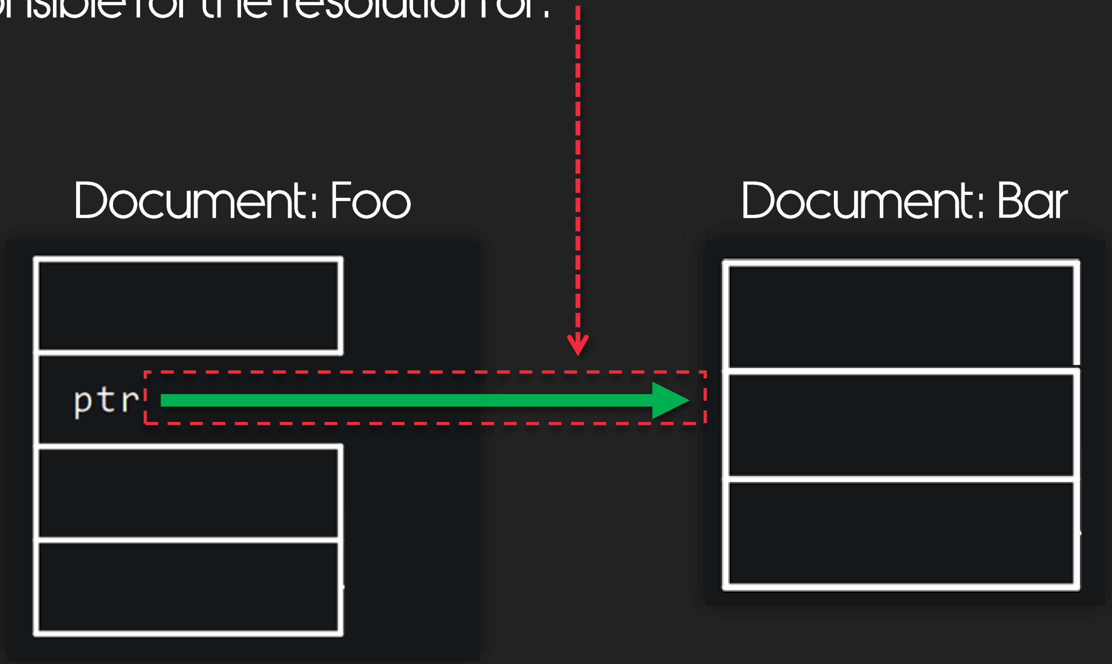


Santa Monica Studio

As a reminder, God of War's world-building workflow has an in-memory representation as a directed acyclic graph of X3D Scenes. But if the connections between scenes are defined by external pointers …

## External Pointers

Who is responsible for the resolution of:

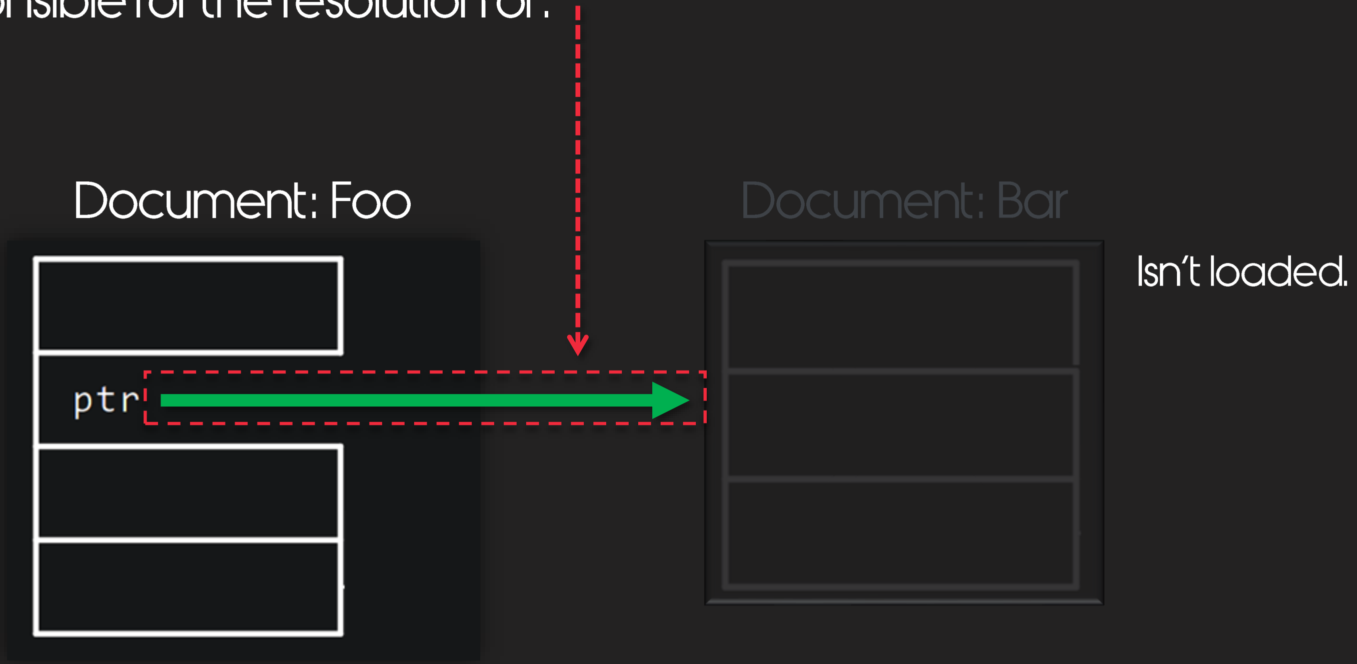


Santa Monica Studio

Who is responsible for the resolution of that pointer? Should it be document Foo? Document Bar? It's not clear, because an external pointer closely resembles both a reference and contract between the two documents. Their states are connected to one another, for example …

## External Pointers

Who is responsible for the resolution of:

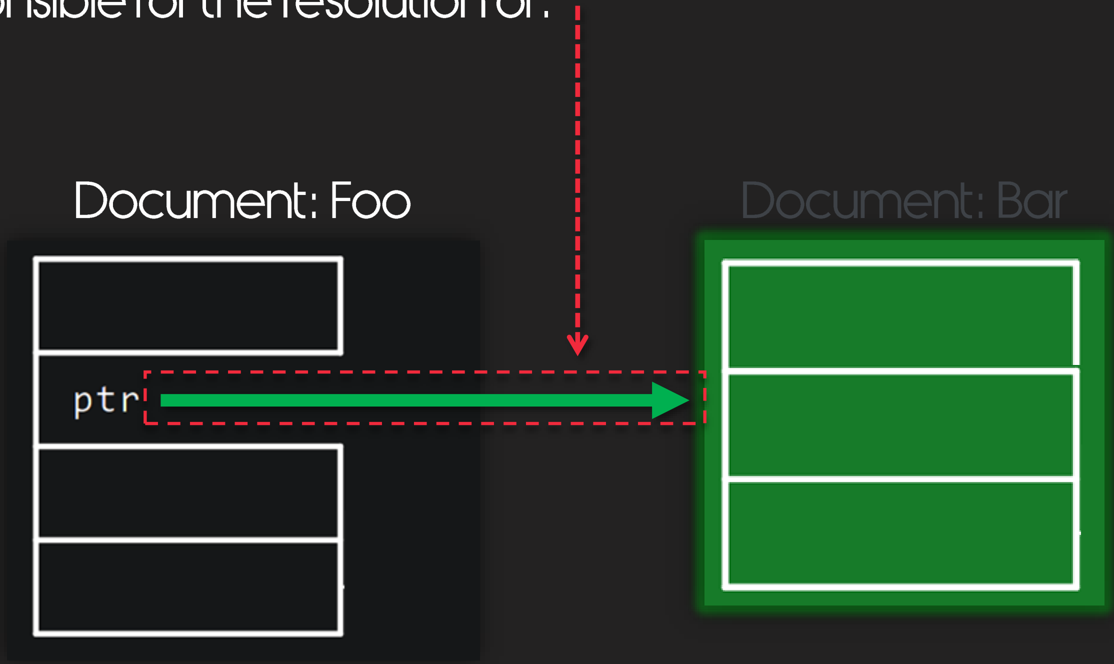


Santa Monica Studio

Foo may be present in memory, but Bar may still be frozen and currently unloaded.

## External Pointers

Who is responsible for the resolution of:

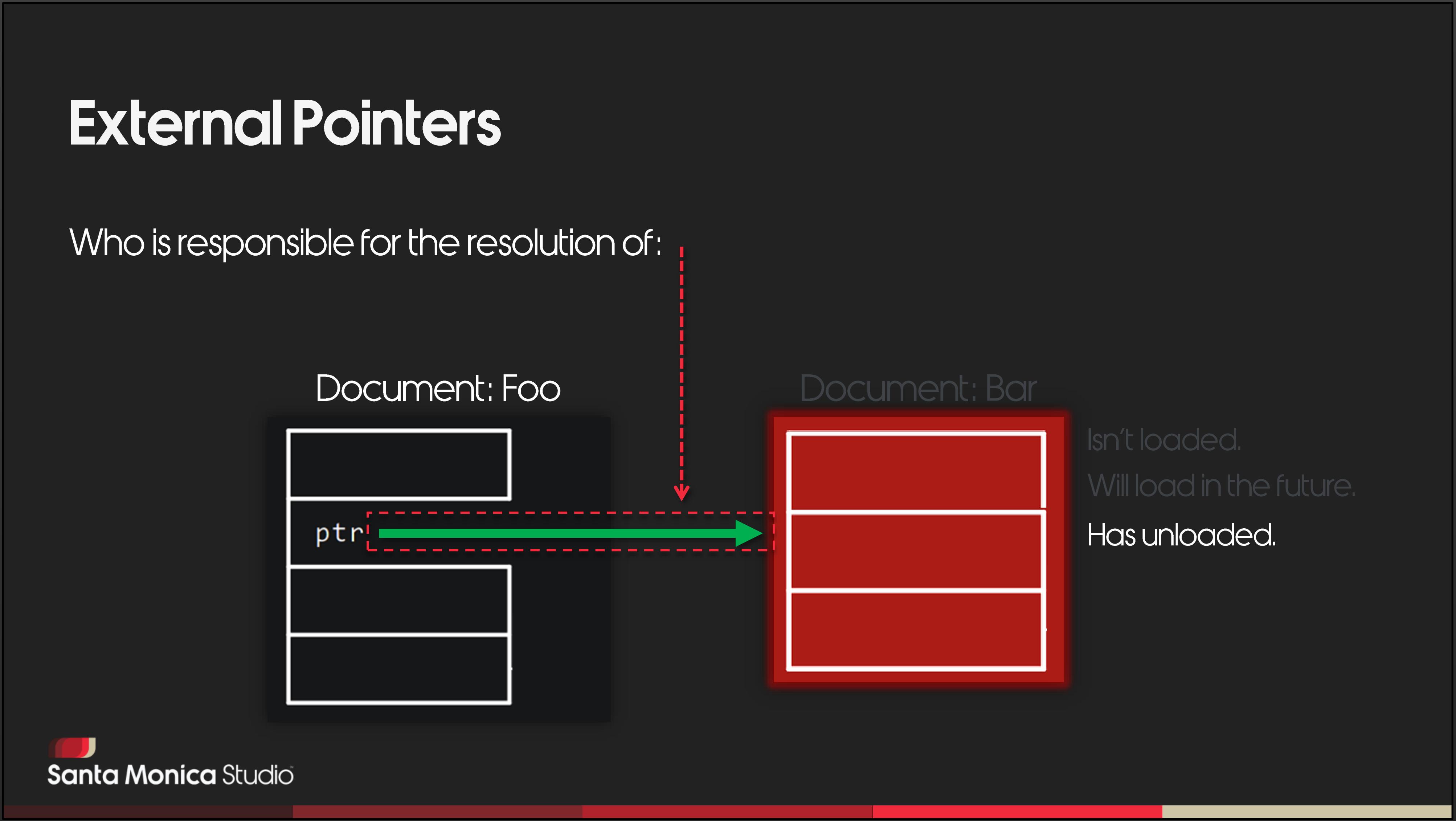


Isn't loaded.

Will load in the future.

Santa Monica Studio

Alternatively, while Bar may not be currently loaded, it will be in the future – requiring a deferred unfreezing of Foo's external pointer.



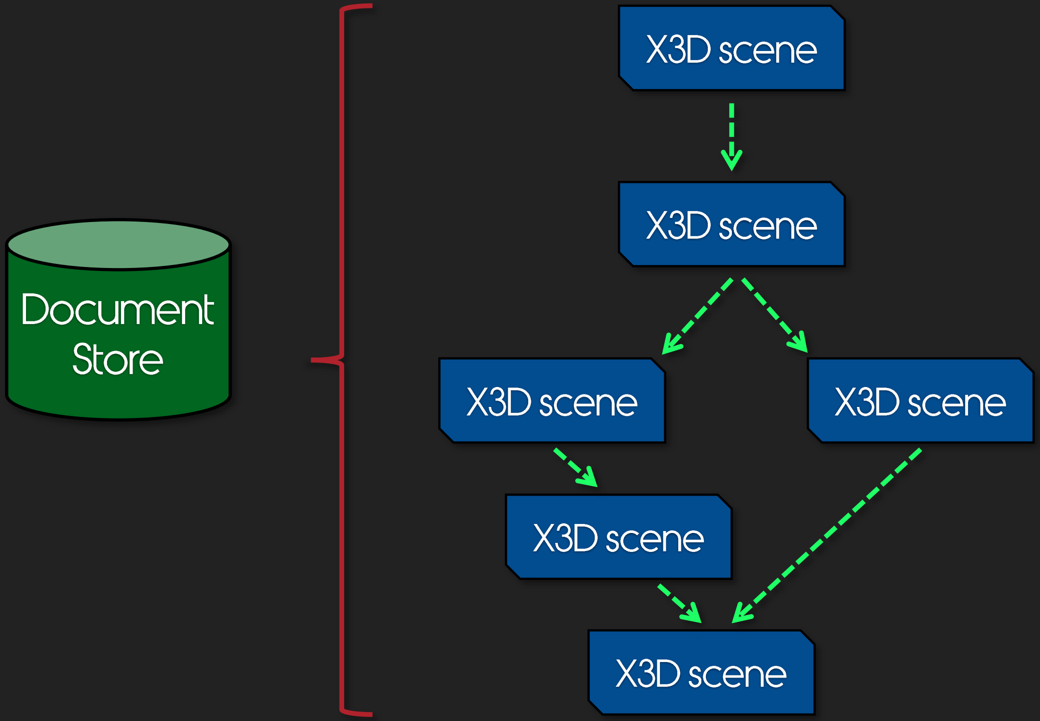
## External Pointers

Who is responsible for the resolution of:

Santa Monica Studio

The resolution of Foo's external pointer is connected to Bar's state – which suggests that we need a way to express this as a transaction.

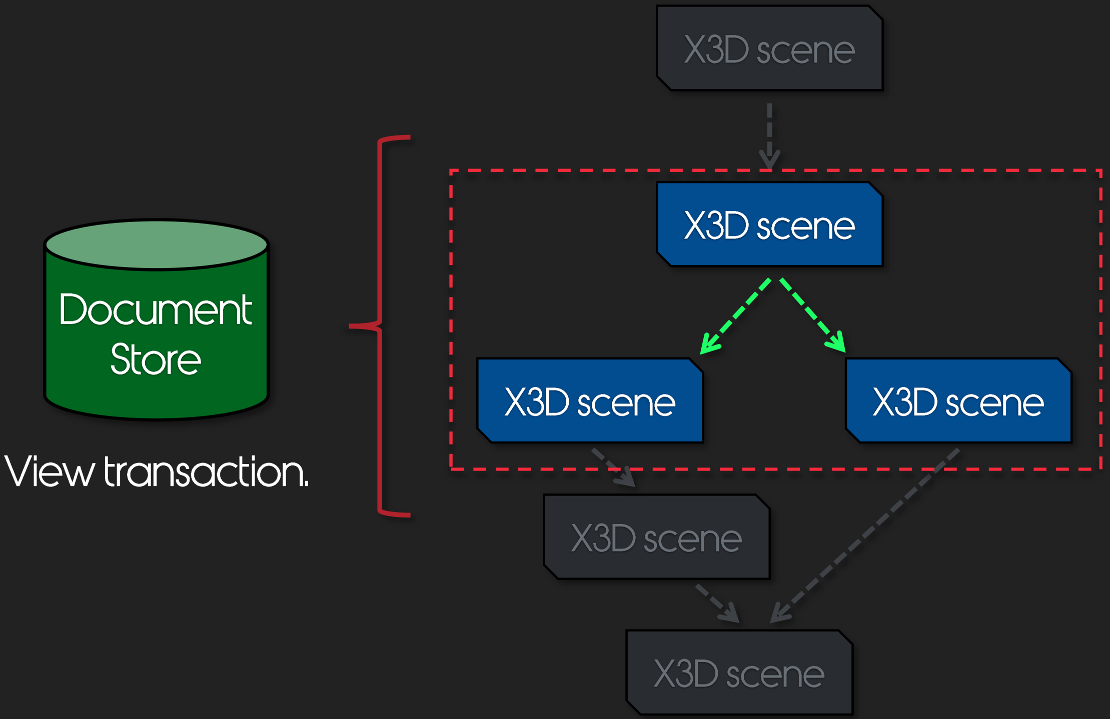
## The existence of external pointers suggested the necessity for a data structure to arbitrate documents.



Santa Monica Studio

This necessity to arbitrate across a range of documents led to the creation of the document store. The document store is a data structure that maintains a set of in-memory document images, and specifically has the authority to establish and resolve external pointers given a resolution policy by the user.

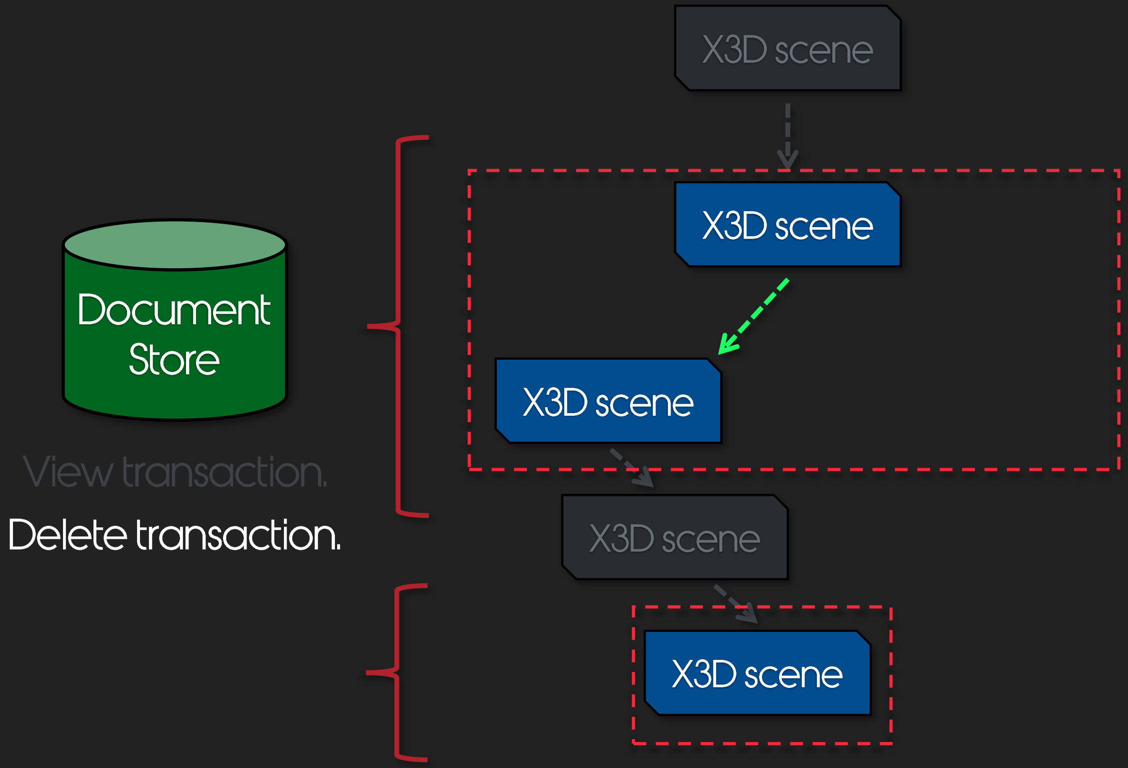
## Example: viewing a portion of the scene graph in isolation.



Santa Monica Studio

In our current usage and experiments, we've modeled document arbitration as a set of transactions that can be applied. For example we may want to unload and view a sub-graph of the X3D Scene graph during a level artist's workflow.

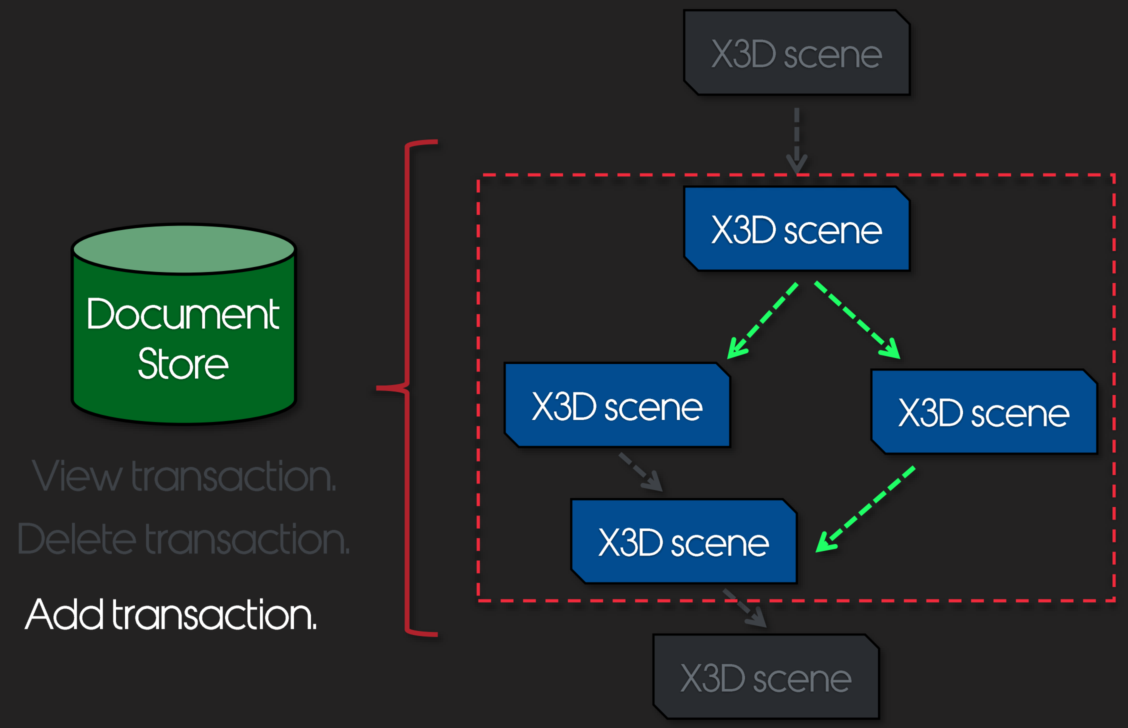
## Example: deleting a scene graph node, while maintaining referencing coherence.



Santa Monica Studio

Or we may want to disconnect referencing between certain scenes …

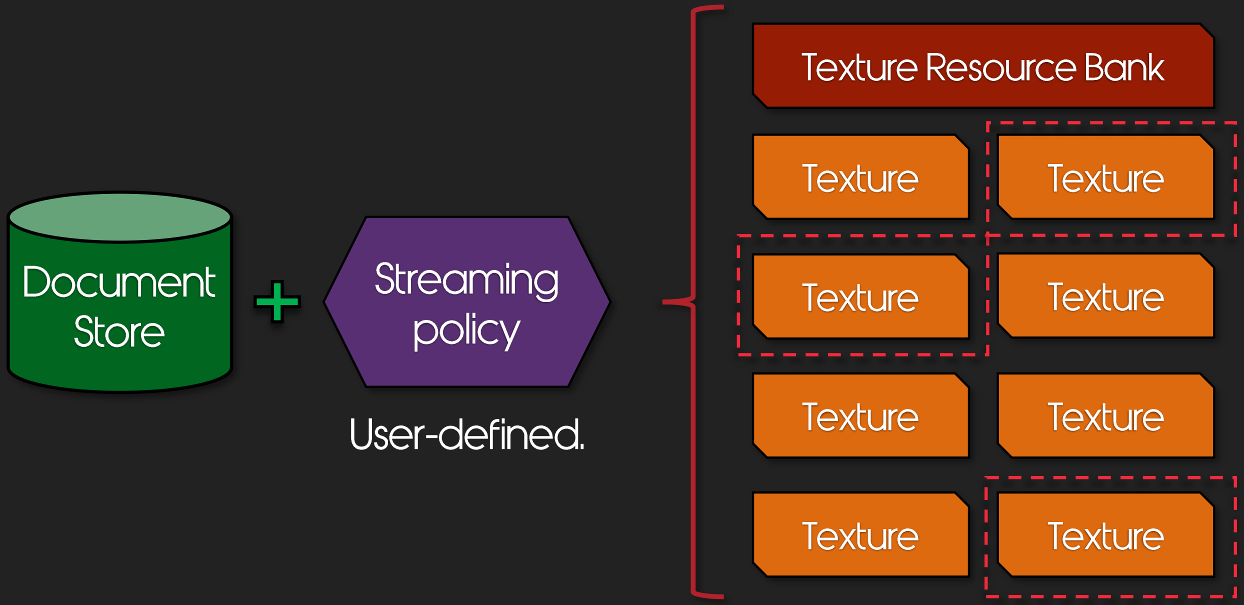
## Example: introducing a new node to the scene graph.



Santa Monica Studio

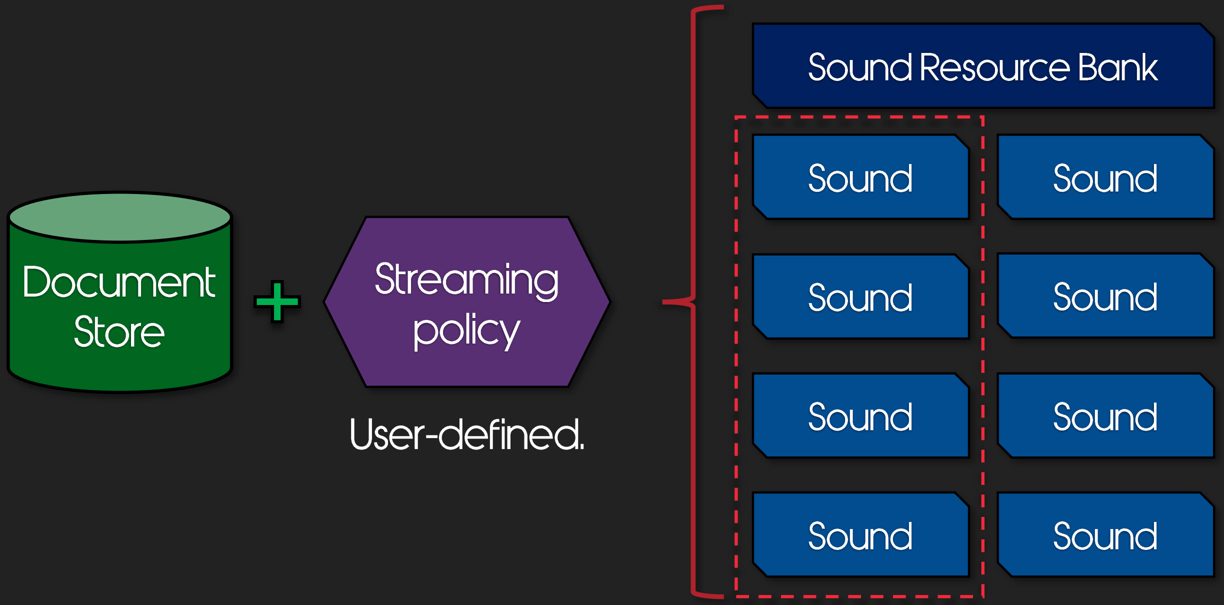
Alongside establishing new connections and views of the data.

## Example: texture streaming, driven by an engine policy.



Our documents don't necessarily need to be hierarchal or describe a sophisticated network such as a scene graph.

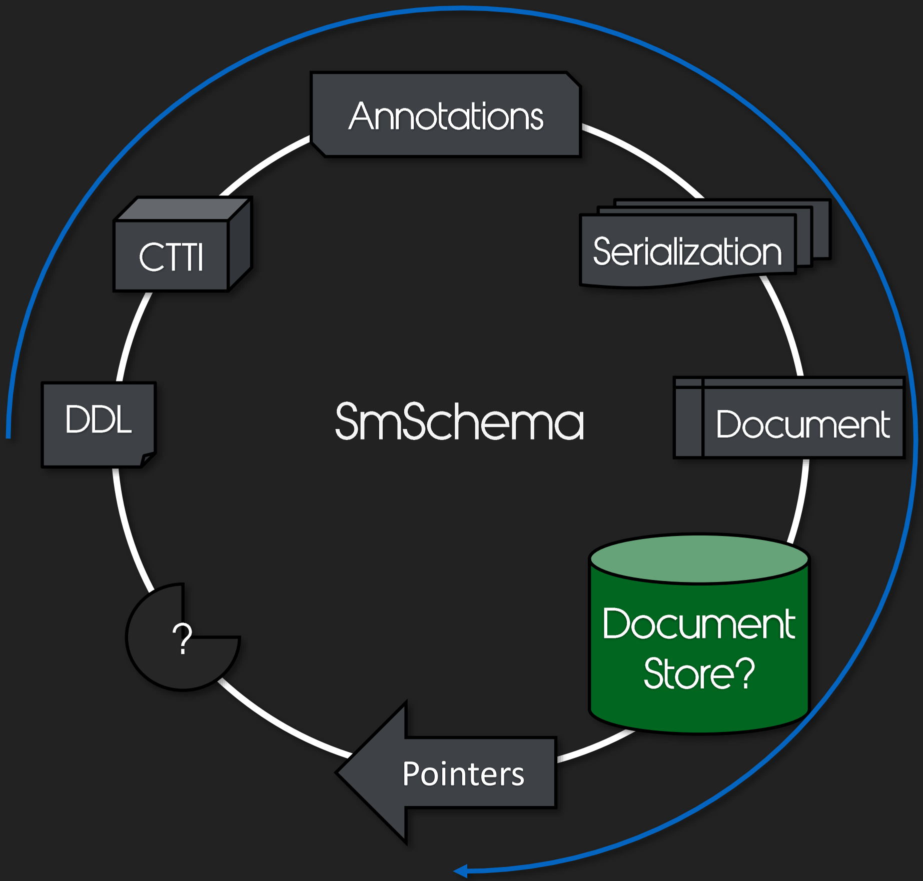
## Example: audio streaming, driven by an engine policy,



Santa Monica Studio

Policies may describe complex decision behavior, but arbitrate much simpler ranges of documents that are related only by type.

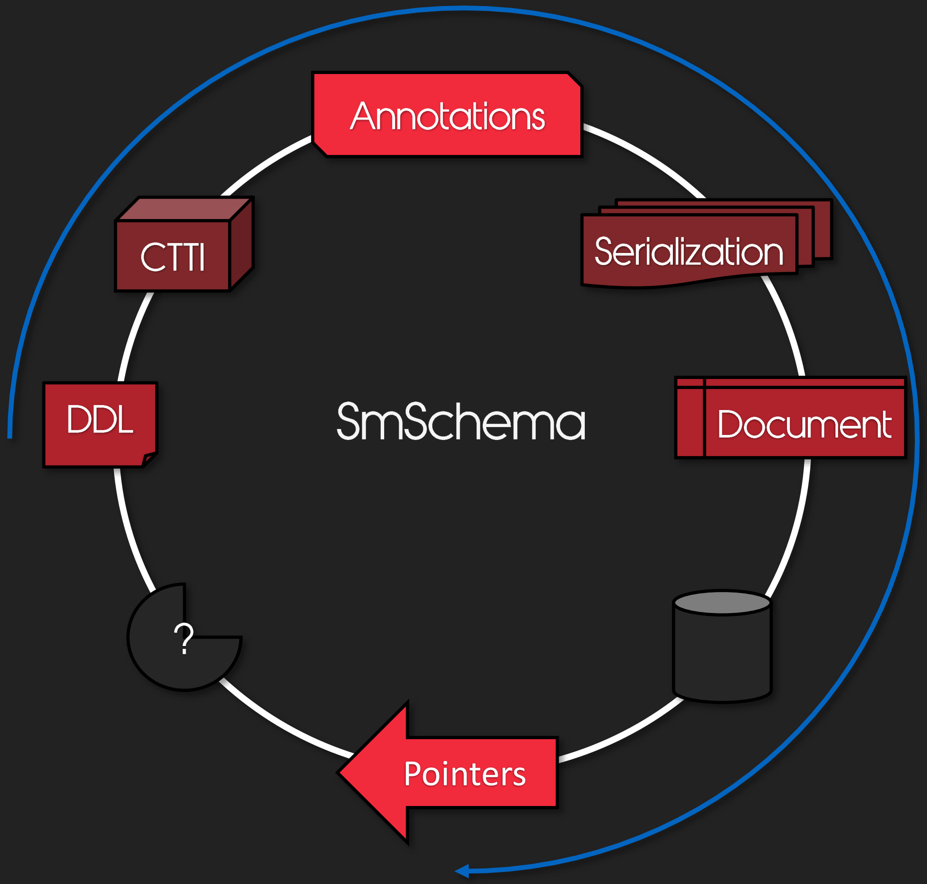
## … Does the document store makes sense to include in our new ecosystem?



Santa Monica Studio

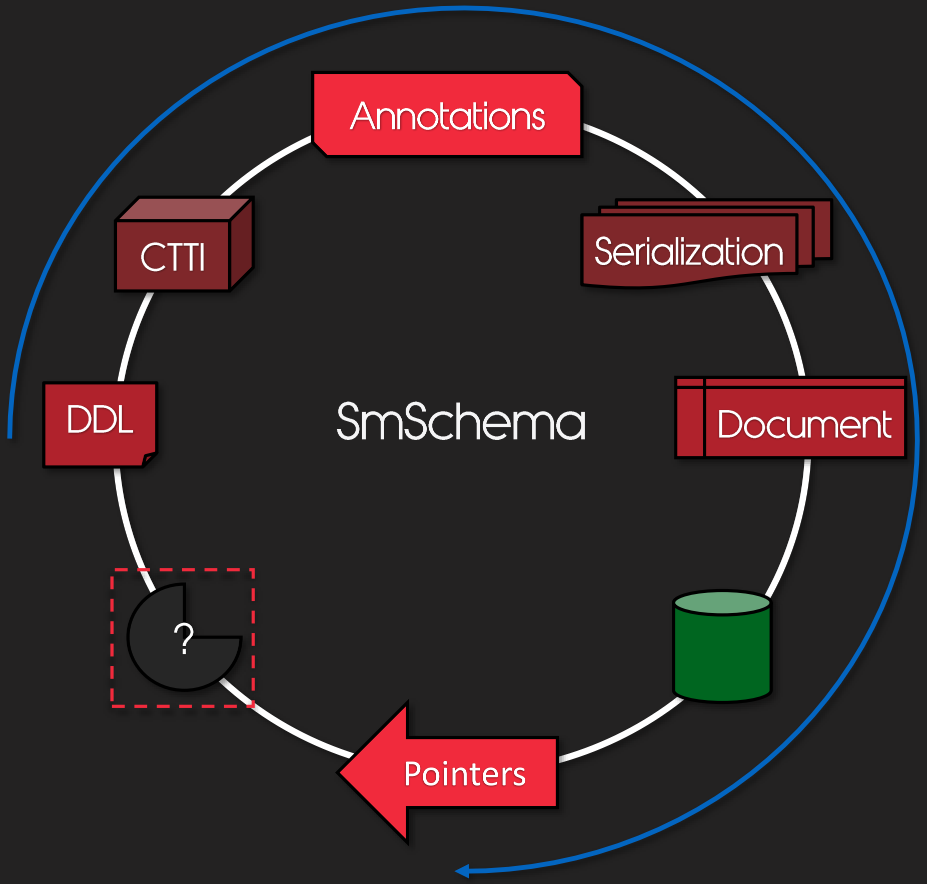
Pretty cool … but does the document store make sense to include in our new ecosystem? It’s hard to say … it speaks to a broader point regarding the design and behavior of source data, production data and any system in between.

## Our decision-making is driven by use-case, with the goal of making our technology purposeful.



Santa Monica Studio

Are solutions have to be driven by use case, introduced with scrutiny, and solve a domain of problems evidently and clearly. As we move into the future, content will only continue to increase in volume and complexity.



They serve as an anecdote of our thought process.

Santa Monica Studio

If we are to introduce sophisticated technologies such as the document store and beyond …

## Lessons we’ve learned.

We need to hold steadfast and rely on the hard lessons we’ve learned that have brought us to this point.

Question:

What is the future of scene description for God of War?

Santa Monica Studio

When we started on this journey and the X3D Scene project, there was a question that was posed early on, that in truth I haven't addressed directly – what is the future of scene description for God of War? In our attempts to answer that question …

Takeaway:

We encountered a different question.

Regarding the nature of modeling data in games.

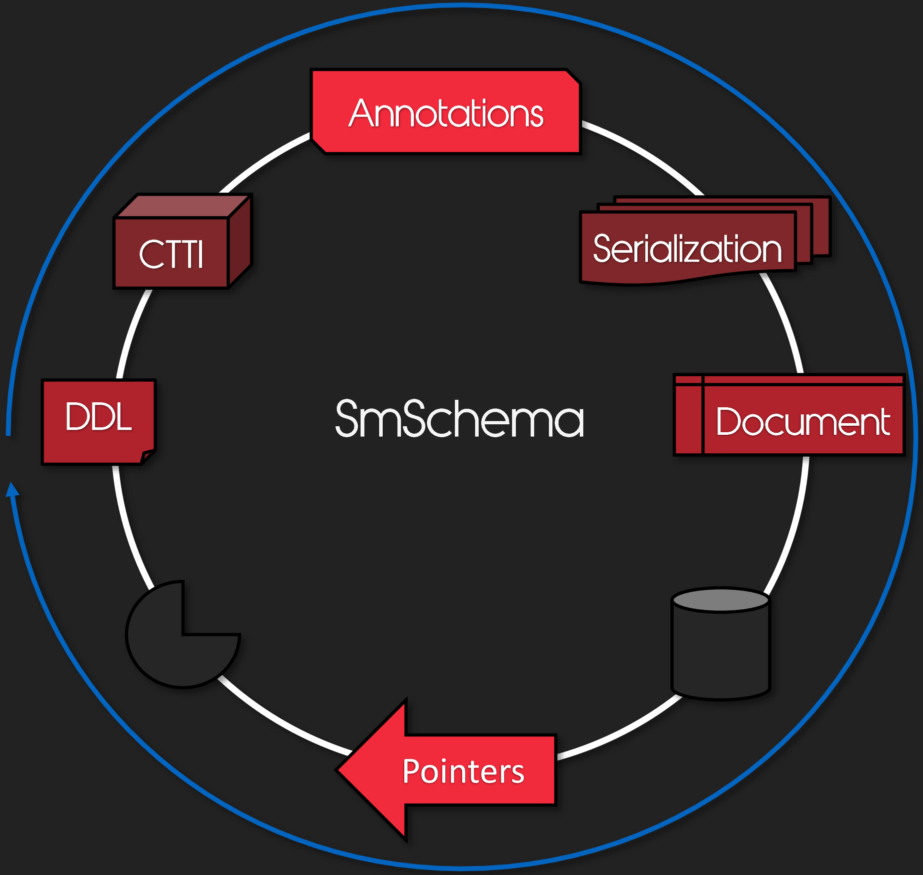
What are the set primitives that help us in our design?

For the next era of engines? workflows? formats?

Santa Monica Studio

We encountered a deeper question – one that spoke to the nature of modeling data itself for our games. What is the set of primitives that make sense for building our future products? How will those primitives make our technology simpler, coherent, and power the next generation of workflows, engines, and formats?

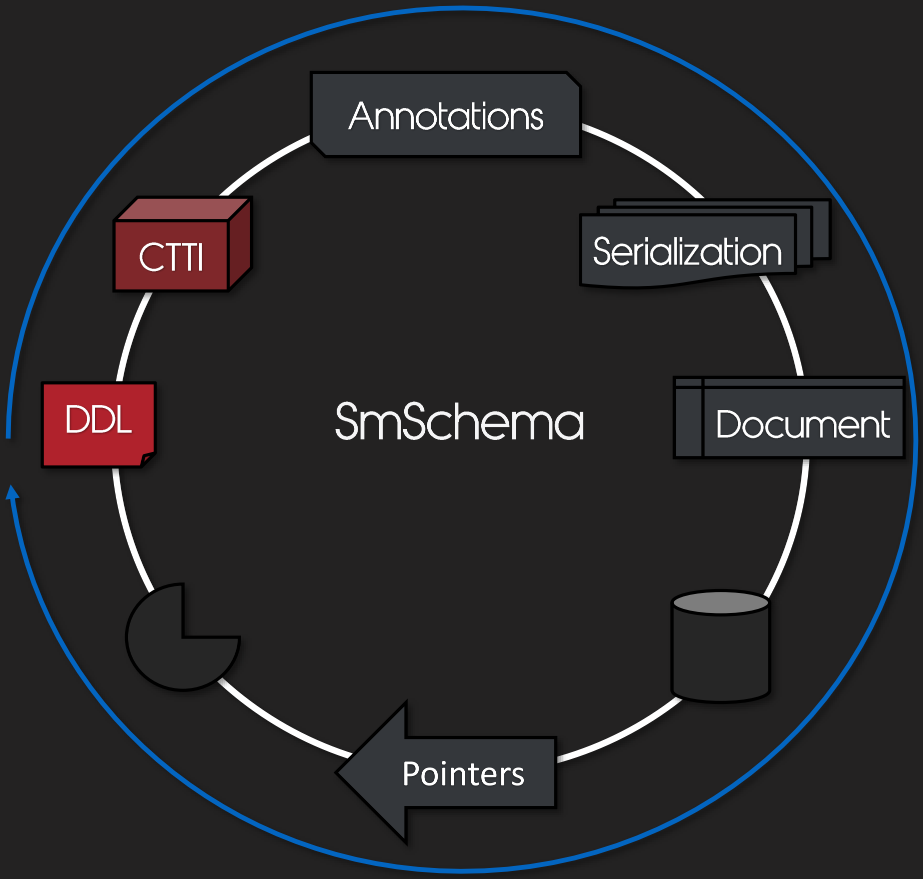
## The goal was not predispose ourselves to a particular design, and instead allow us to answer a set of questions.



Santa Monica Studio

Our goal was never to predispose ourselves to a particular design. The future of scene description is not a data structure, it's not a design pattern; it's a set questions that we need must be able to answer about our content. Questions such as …

What is the design?

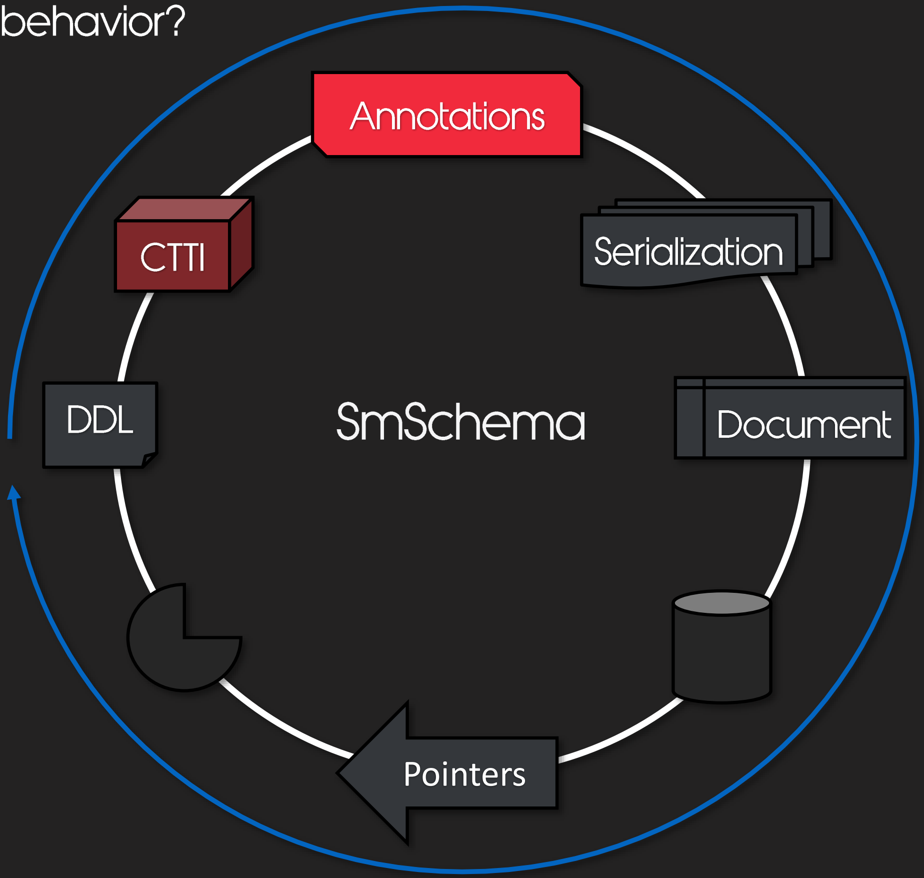


Santa Monica Studio

What is the design of the data, what tradeoffs make sense today versus tomorrow?

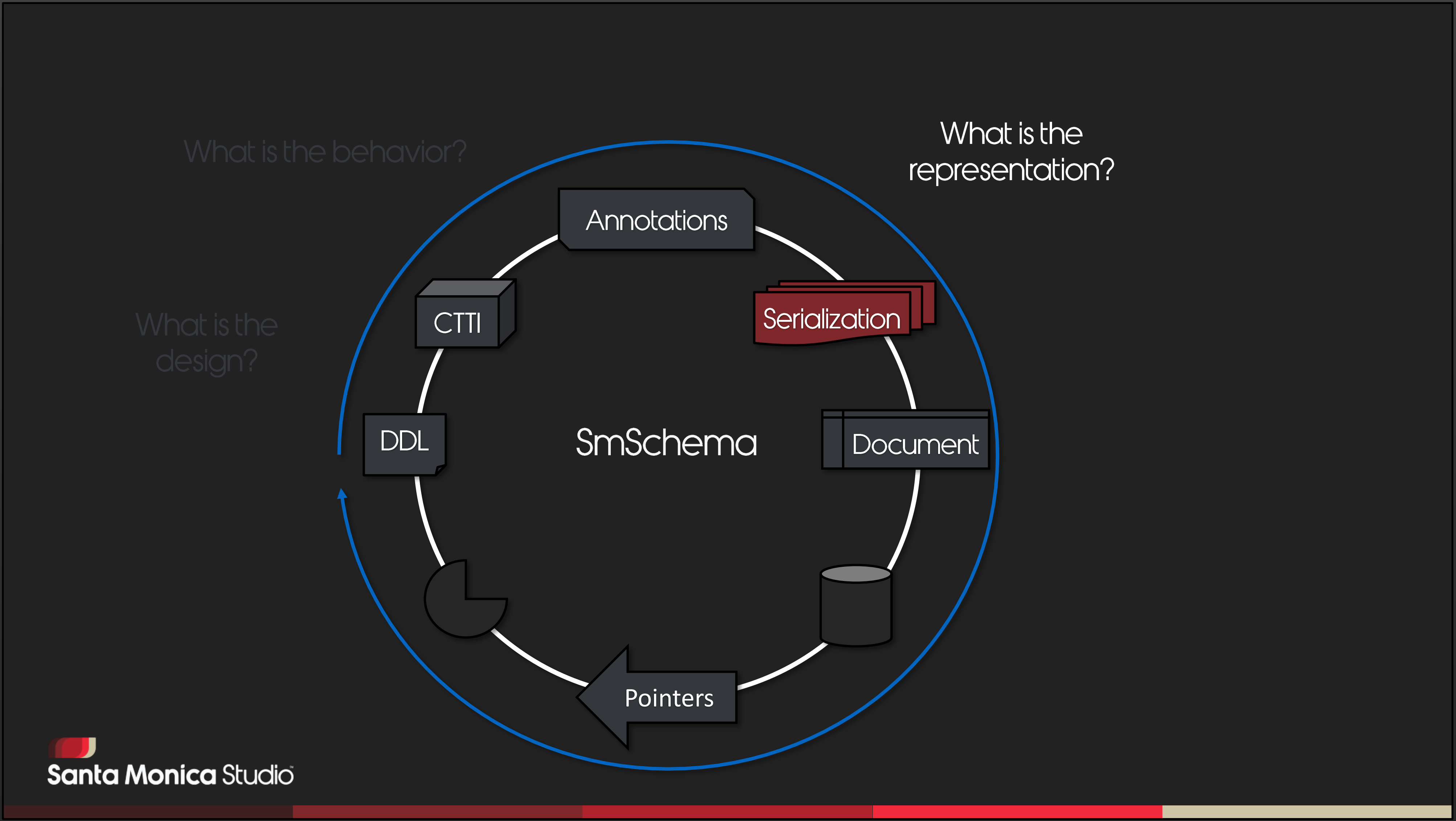
## What is the behavior?

What is the design?

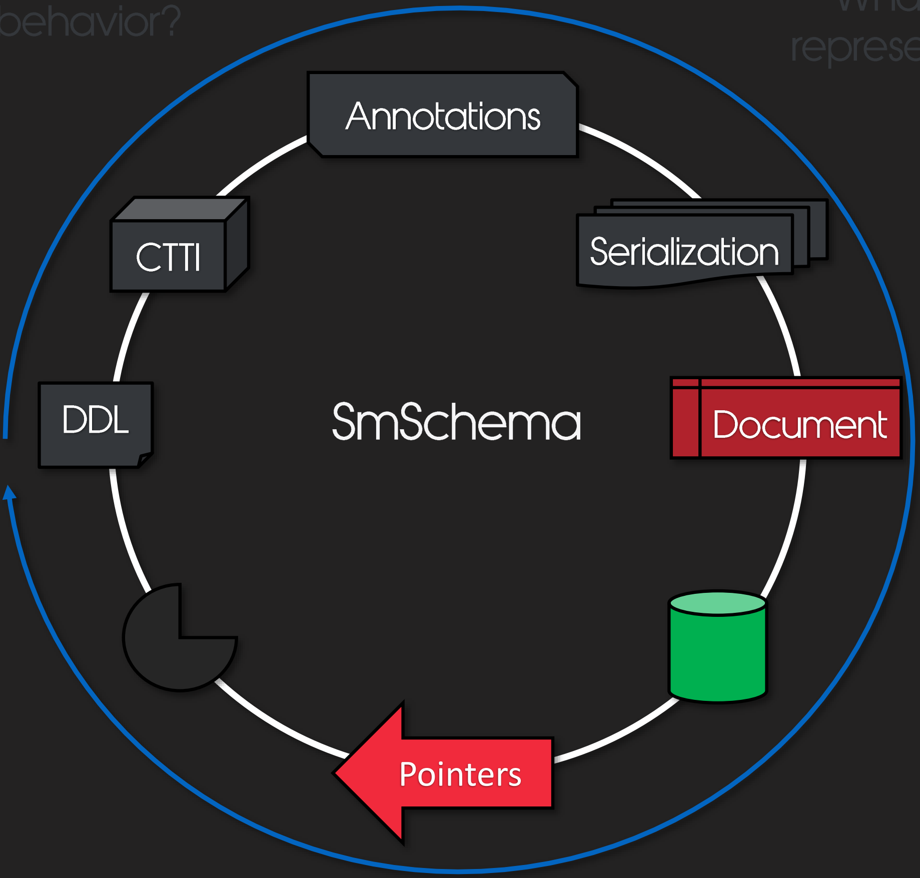


Santa Monica Studio

What behavior should the data have? Driven by context and usage.



How is the data represented, if it exists in Perforce or our player's home console?



What is the behavior?

What is the representation?

What is the design?

What are the relationships?

Santa Monica Studio

And how is the data connected, what are the relationships and granularity?

## Takeaway:

Prioritize flexibility and nimbleness.

Future-proofing any design is a trap.

Solve the right problem.

Santa Monica Studio

With SmSchema we prioritized flexibility and nimbleness with the technology, since it closely modeled our aspired method of decision making. In fact, future proofing any particular design seemed like a trap. And if we did invest in a technology solution, our goal has always been to try and solve the right problem.

## Credits

Federico Bianco Prevot

John Calsbeek

Paolo Costabel

Alexandre de Pereya

Bob Soper

Sam Willis

Santa Monica Studio

I feel fortunate to have worked on this project with so many incredible engineers who contributed their expertise to this very interesting and fundamental problem domain for games. Specifically I would like to mention John Calsbeek, Paolo Costabel, Alexandre de Pereya, Bob Soper, and Sam Willis for their work throughout the years on this project. And a very special thank you to my dear friend Federico Bianco Prevot who pioneered much of this work from its inception, and challenged all of us to dig deeper.

## Thank You!

koray.hagen@sony.com
@korayhagen

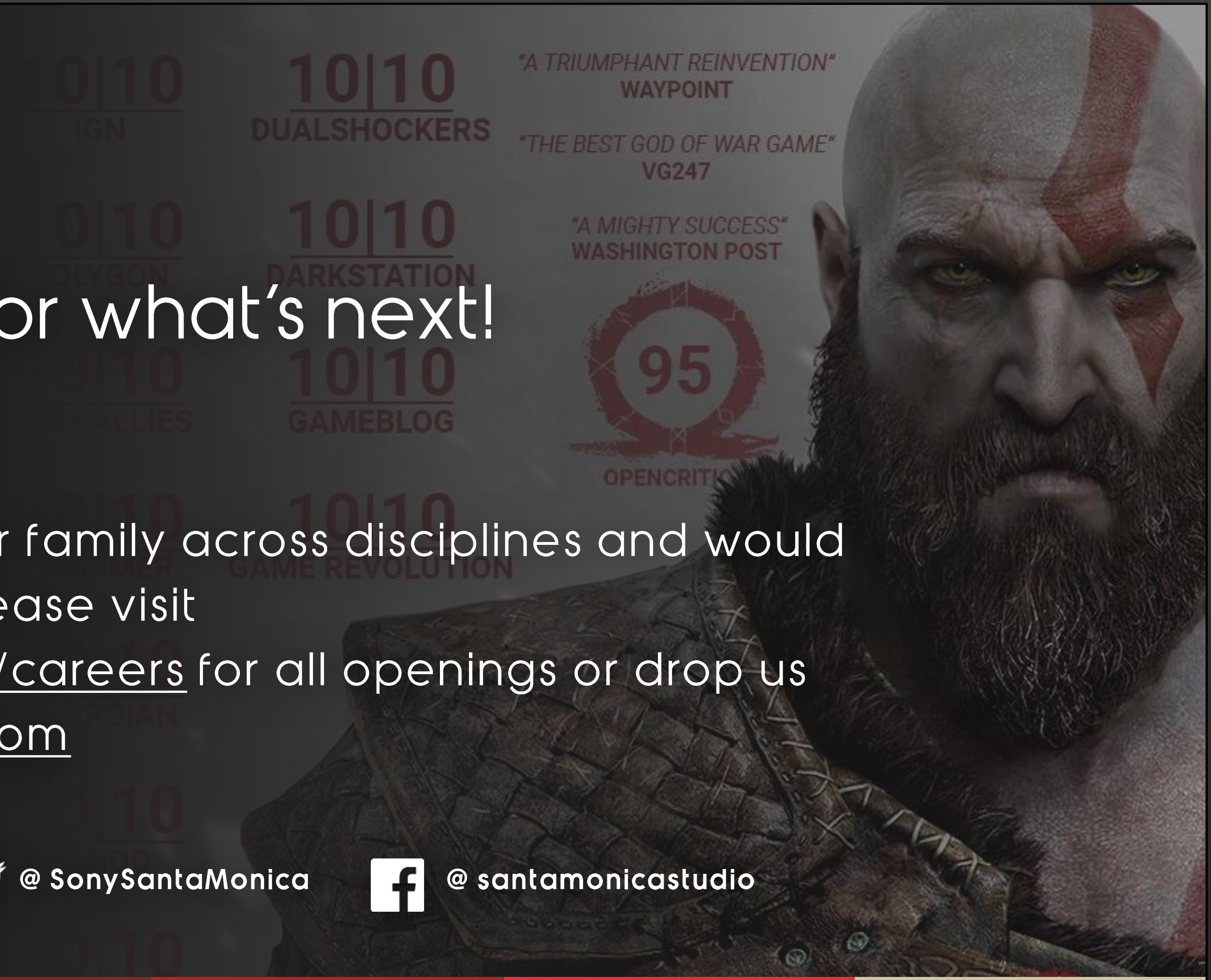
SIE Worldwide Studios

Thank you very much.

Santa Monica Studio

# Our journey
# Your story

## We're hiring for what's next!



We're expanding our family across disciplines and would love to meet you. Please visit sms.playstation.com/careers for all openings or drop us a line at sms@sony.com

@ santamonicastudio @ SonySantaMonica @ santamonicastudio

JOIN US AT GDC 2019

# GOD OF WAR

BUILD YOUR GOD OF WAR GDC AT:
SCHEDULE.GDCONF.COM

**BRUNO VELAZQUEZ** • Animation Director
God of War: Breathing New Life into a Hardened Spartan • ANIMATION BOOTCAMP
MONDAY, MARCH 18 • 10:00AM - 11:00AM • ROOM 2010, WEST HALL

**MELISSA SHIM** • Senior Animator
Animation Bootcamp: Animation Tricks of The Trade • ANIMATION BOOTCAMP
MONDAY, MARCH 18 • 4:40PM - 5:10PM • ROOM 2010, WEST HALL

**ROB DAVIS** • Lead Level Designer
Level Design Workshop: The Level Design of God of War • LD SUMMIT
TUESDAY, MARCH 19 • 11:20AM - 12:20PM • ROOM 301, SOUTH HALL

**ERICA PINTO** • Lead Narrative Animator
What They Don't Teach You in Art School: Lessons for First Time Leads • ART DIRECTION BOOTCAMP
TUESDAY, MARCH 19 • 1:20PM - 2:20PM • ROOM 2001, WEST HALL

**AXEL STANLEY-GROSSMAN** • Lead Technical Character Artist
Santa Monica Studio Presents: God of War (Presented by Autodesk) • VISUAL ARTS
TUESDAY, MARCH 19 • 2:40PM - 3:40PM • ROOM 3020, WEST HALL

**RUPERT RENARD** • Senior Programmer
Wind Simulation in God of War • PROGRAMMING
WEDNESDAY, MARCH 20 • 10:30AM - 11:00AM • ROOM 303, SOUTH HALL

**RUPERT RENARD** • Senior Programmer
Disintegrating Meshes with particles in God of War • PROGRAMMING
WEDNESDAY, MARCH 20 • 11:30AM - 12:00PM • ROOM 303, SOUTH HALL

**ED DEARIEN & JEET SHROFF** • Assistant Producer & Gameplay Director
Playtesting God of War • PRODUCTION
WEDNESDAY, MARCH 20 • 2:00PM - 3:00PM • ROOM 2001, WEST HALL

**KORAY HAGEN** • Senior Programmer
The Future of Scene Description on God of War • PROGRAMMING
WEDNESDAY, MARCH 20 • 2:00PM - 3:00PM • ROOM 302, SOUTH HALL

**DORI ARAZI** • Director of Photography
Creating a Deeper Emotional Connection: The cinematography of God of War • VISUAL ARTS
WEDNESDAY, MARCH 20 • 3:30PM - 4:30PM • ROOM 2005, WEST HALL

**JASON MCDONALD** • Design Director
Taking an Axe to God of War Gameplay • DESIGN
THURSDAY, MARCH 21 • 10:00AM - 11:00AM • ROOM 303, SOUTH HALL

**ERICA PINTO** • Lead Narrative Animator
Keyframes and Cardboard Props: The Cinematic Process Behind God of War • VISUAL ARTS
THURSDAY, MARCH 21 • 11:30AM - 12:30PM • ROOM 2001, WEST HALL

**MIKE NIEDERQUELL** • Lead Sound Designer
The Sound Design of God of War • AUDIO
THURSDAY, MARCH 21 • 2:00PM - 2:30PM • ROOM 3002, WEST HALL

**SHAYNA MOON** • Associate Producer
Shipping Greatness: Practical Lessons from Audio Production on God of War • PRODUCTION
THURSDAY, MARCH 21 • 3:00PM - 3:30PM • ROOM 3002, WEST HALL

**HAYATO YOSHIDOME** • Sr. Staff Technical Combat Designer
Raising Atreus for Battle in God of War • DESIGN
THURSDAY, MARCH 21 • 4:00PM - 5:00PM • ROOM 2005, WEST HALL

**JOSH HOBSON** • Lead Rendering Programmer
The Indirect Lighting Pipeline of God of War • PROGRAMMING
THURSDAY, MARCH 21 • 4:00PM - 5:00PM • ROOM 2010, WEST HALL

**SEAN FEELEY** • Sr Staff Technical Artist
Interactive Wind and Vegetation in God of War • DESIGN
THURSDAY, MARCH 21 • 5:30PM - 6:30PM • ROOM 2005, WEST HALL

**CORY BARLOG** • Creative Director
Reinventing God of War
FRIDAY, MARCH 22 • 10:00AM - 11:00AM • ROOM 303, SOUTH HALL

**MINIR SHETH & JEET SHROFF** • Lead Combat Designer & Gameplay Director
Evolving Combat in God of War for a New Perspective • DESIGN
FRIDAY, MARCH 22 • 1:30PM - 2:30PM • ROOM 2005, WEST HALL

Santa Monica Studio GDC