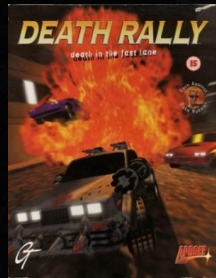


USD AT REMEDY

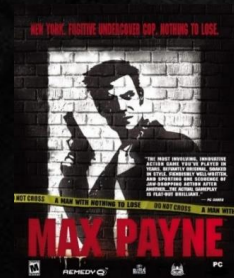
How Remedy uses USD in its next generation
game development pipelines



REMEDY HISTORY



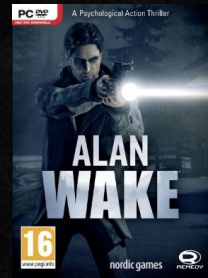
1996



2001



2003



2010



2012



2016



2019



2020



2021



2022

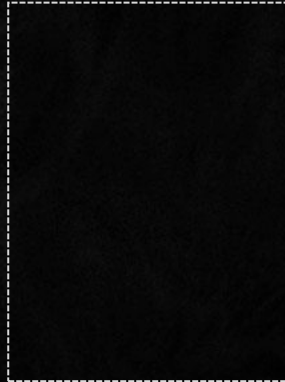
REMEDY TODAY



CONTROL



CONDOR



Heron*



ALAN WAKE II



Vanguard*



Max Payne
1-2 Remake

WITH

505GAMES

WITH



WITH

Tencent

WITH



* project codename

REMEDY'S TECHNOLOGY

n^orthlight[®]
REMEDY STORYTELLING TECHNOLOGY

PROPRIETARY ENGINE AND TOOLING (NORTHLIGHT)

- World Editor, Game Engine, Data Pipelines, etc.

PROPRIETARY DATA FORMATS

- Level/World data, Markup, Materials, etc.
- Interop with DCCs through import/export

NEW EDITOR

- Built from the ground up
- USD based

n^o

THE EDITOR



WHY USD

Scalability

- Growing teams and content
 - Composition arcs
 - Flexible VCS workflows
- DCC interoperability
- Data portability

Consolidation

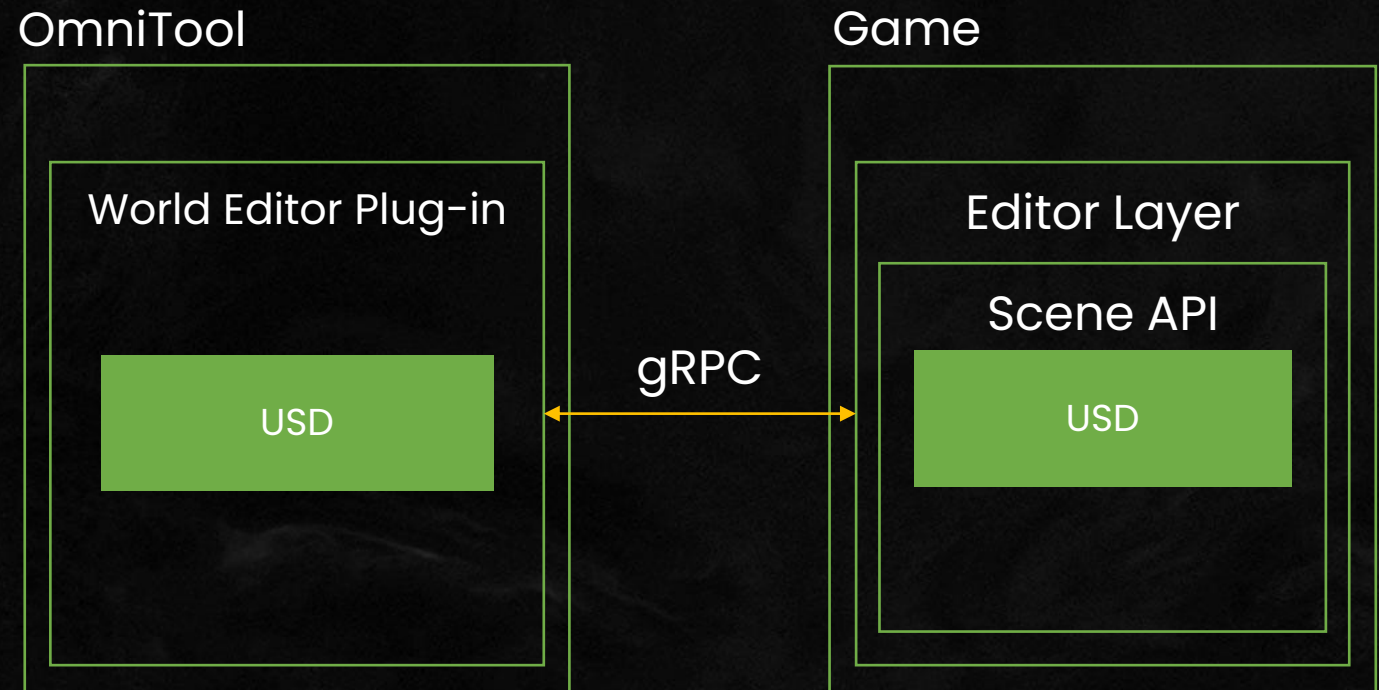
- Unify similar concepts: levels, prefabs, archetypes, presets

Performance

- Asset loading
- Large worlds

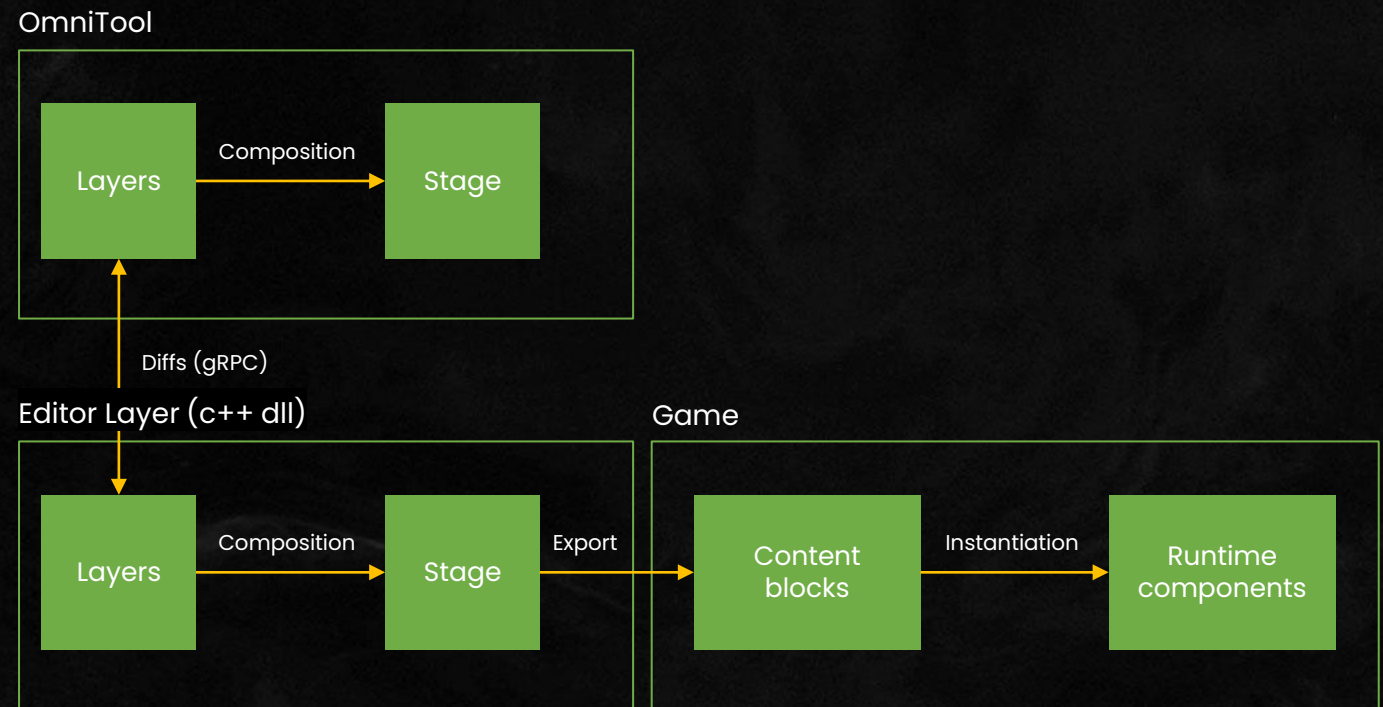
USD + NORTHLIGHT ARCHITECTURE

- Tools framework (OmniTool) in .NET/C#
- Game (runtime engine) in C++
- USD stages synced between processes



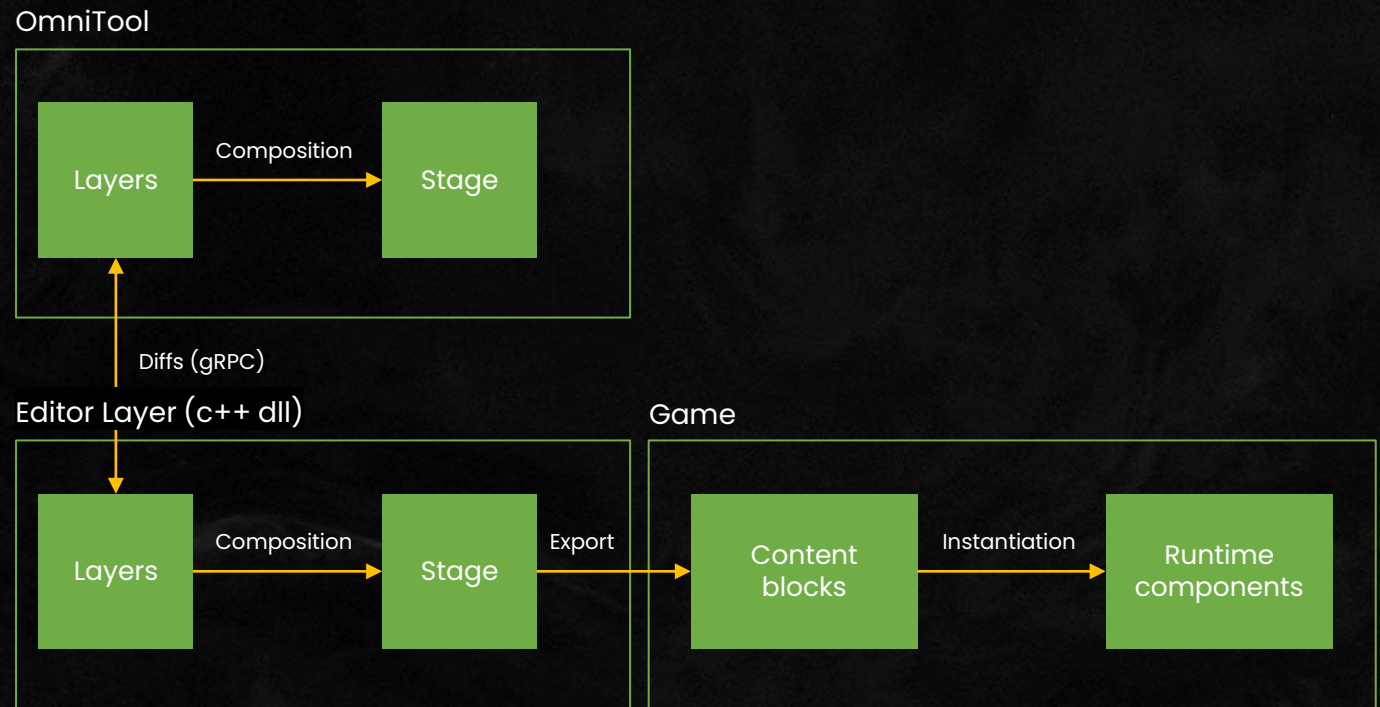
USD + NORTHLIGHT ARCHITECTURE

- Tools framework (OmniTool) in .NET/C#
- Game (runtime engine) in C++
- USD stages synced between processes
 - Layer diffs applied as live-edits
 - Uses `SdfLayerStateDelegate`



USD + NORTHLIGHT ARCHITECTURE

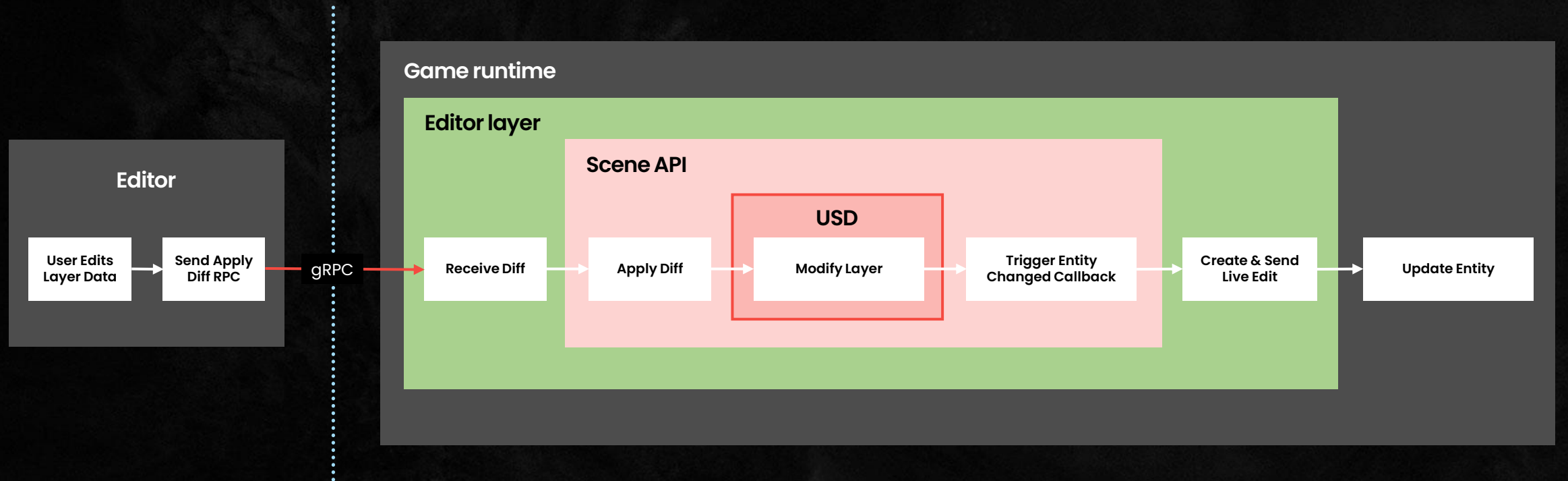
- Tools framework (OmniTool) in .NET/C#
- Game (runtime engine) in C++
- USD stages synced between processes
 - Layer diffs applied as live-edits
 - Uses `SdfLayerStateDelegate`
- USD not included in final build
 - Used only at edit-time



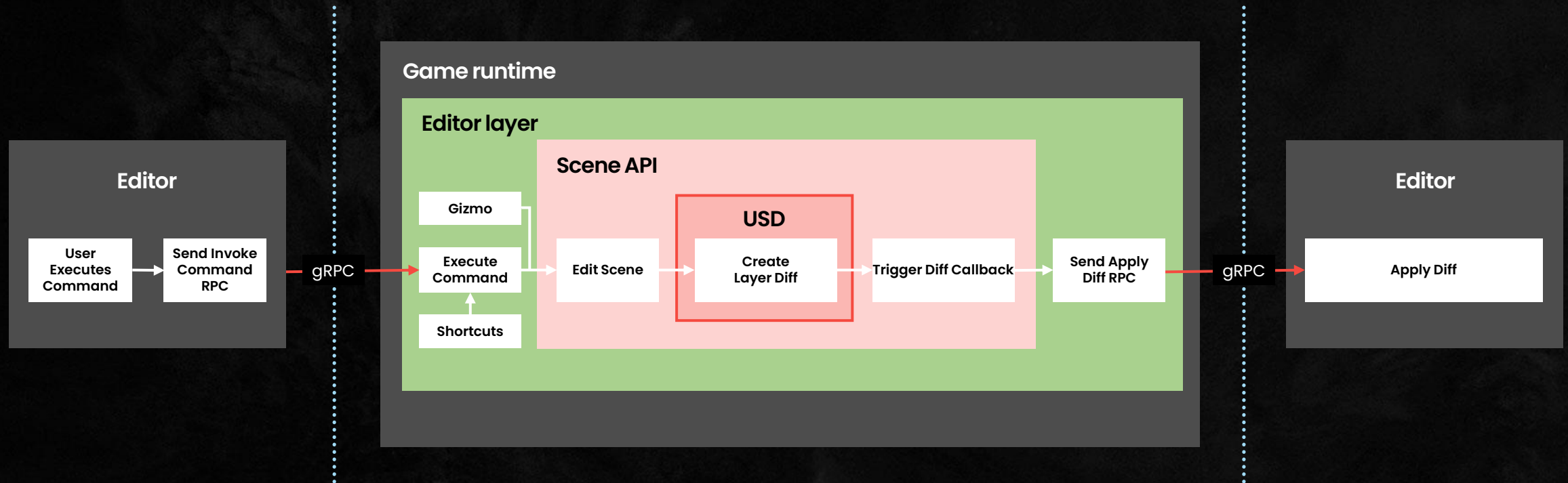
WHY NO RUNTIME USD AT REMEDY

- No support for non-desktop platforms
 - PlayStation 5, Xbox Series X|S, Switch, etc...
- USD should be optimized for content creation on desktop
 - Composition only at edit-time
 - Data transformation/baking for runtime
- Rewriting the runtime was a no-go zone
- Live-editing over import/export/stage reloading workflows
- Flattened "scene" at runtime, no hierarchy is needed (ECS vs Scene Graph)

COMMUNICATION FLOW – LAYER EDITS

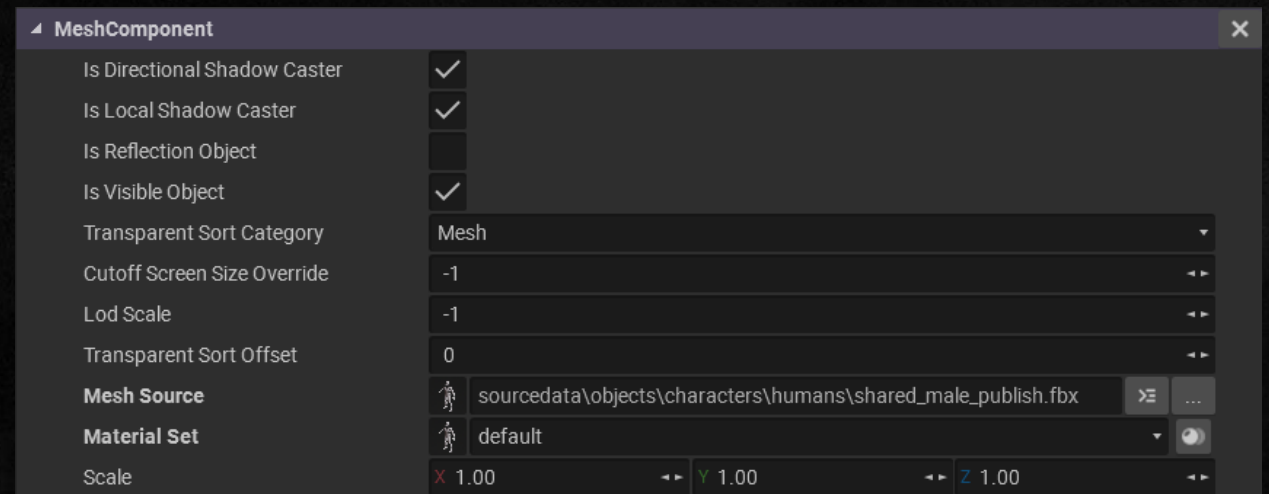


COMMUNICATION FLOW – COMMANDS



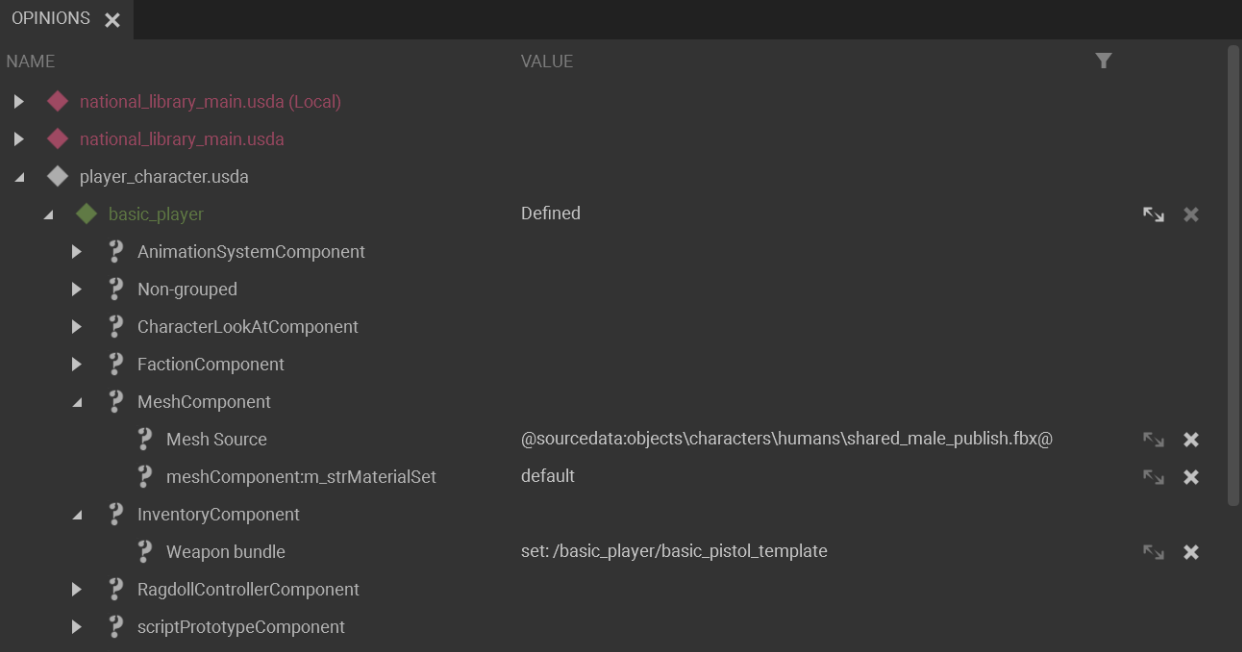
CONTENT - NOW

- Mesh and other “source” data are asset paths in applied schema attributes



CONTENT - NOW

- Mesh and other “source” data are asset paths in applied schema attributes
- Only World/Entity concepts can be composed
 - No opinions on meshes, material attributes, etc.



The screenshot shows a window titled "OPINIONS" with a close button. It contains a table with two columns: "NAME" and "VALUE". The table lists various schema attributes and their values, with some attributes expanded to show sub-attributes.

NAME	VALUE
▶ national_library_main.usda (Local)	
▶ national_library_main.usda	
▶ player_character.usda	
▶ basic_player	Defined
▶ AnimationSystemComponent	
▶ Non-grouped	
▶ CharacterLookAtComponent	
▶ FactionComponent	
▶ MeshComponent	
? Mesh Source	@sourcedata:objects\characters\humans\shared_male_publish.fbx@
? meshComponent:m_strMaterialSet	default
▶ InventoryComponent	
? Weapon bundle	set:/basic_player/basic_pistol_template
▶ RagdollControllerComponent	
▶ scriptPrototypeComponent	

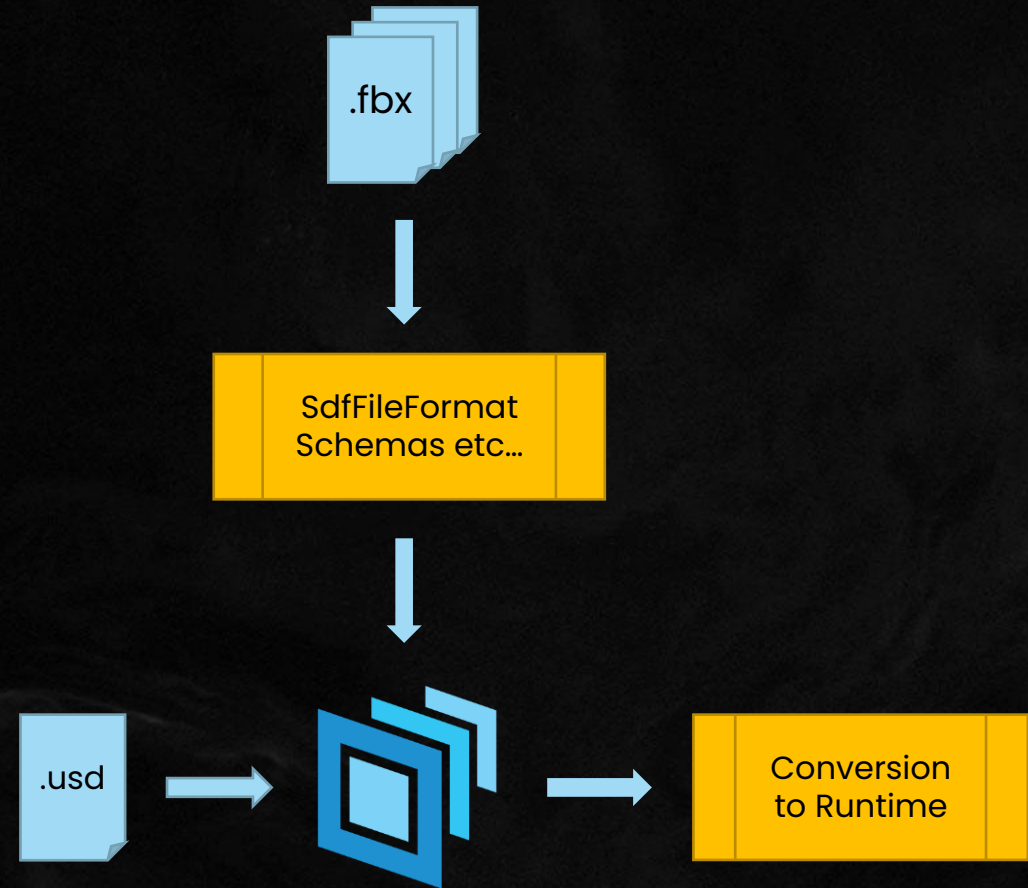
CONTENT - NOW

- Mesh and other “source” data are asset paths in applied schema attributes
- Only World/Entity concepts can be composed
 - No opinions on meshes, material attributes, etc.
- Monolithic Data
 - Character geo & skeletons, environment assets, etc.



CONTENT - MIGRATION

- Existing data via custom plugins
- Data transformation for USD layers
 - Ingests USD and current data
 - Easy to extend



CONTENT - MIGRATION

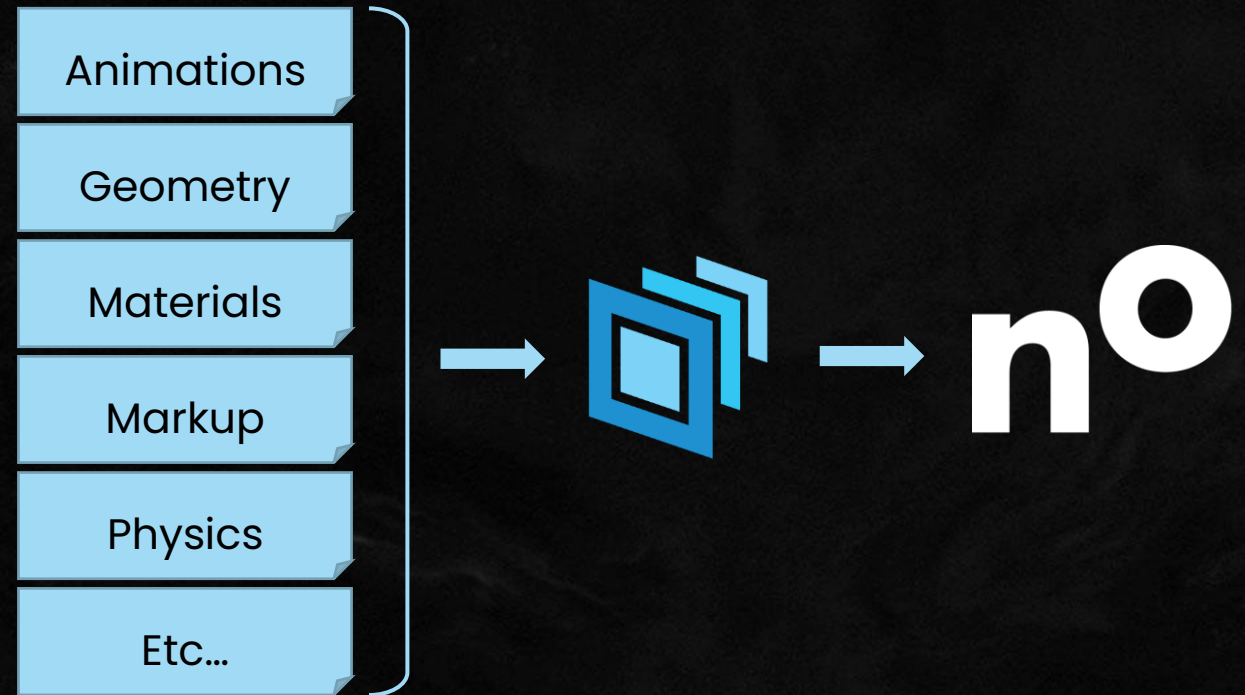
- Existing data via custom plugins
- Data transformation for USD layers
 - Ingests USD and current data
 - Easy to extend
- Composable source assets*
 - Modular chunks
 - Opinions on source data

```
def EntityNode "SomeAsset" (  
  prepend apiSchemas = ["MeshComponentAPI"]  
)  
{  
  rel meshComponent:mesh = </SomeAsset/mesh_root>  
  
  def Xform "mesh_root" (  
  )  
  {  
    def "foo" (  
      prepend references = @sourcedata:objects/foo.fbx@  
    )  
    {  
      over "foo_mesh"  
      {  
        double3 xformOp:translate = (10, 0, 0)  
      }  
    }  
  
    def "bar" (  
      prepend references = @sourcedata:objects/bar.fbx@  
    )  
    {  
      over "bar_mesh"  
      {  
        double3 xformOp:translate = (20, 0, 0)  
      }  
    }  
  }  
}
```

* mockup layer, no indication for future

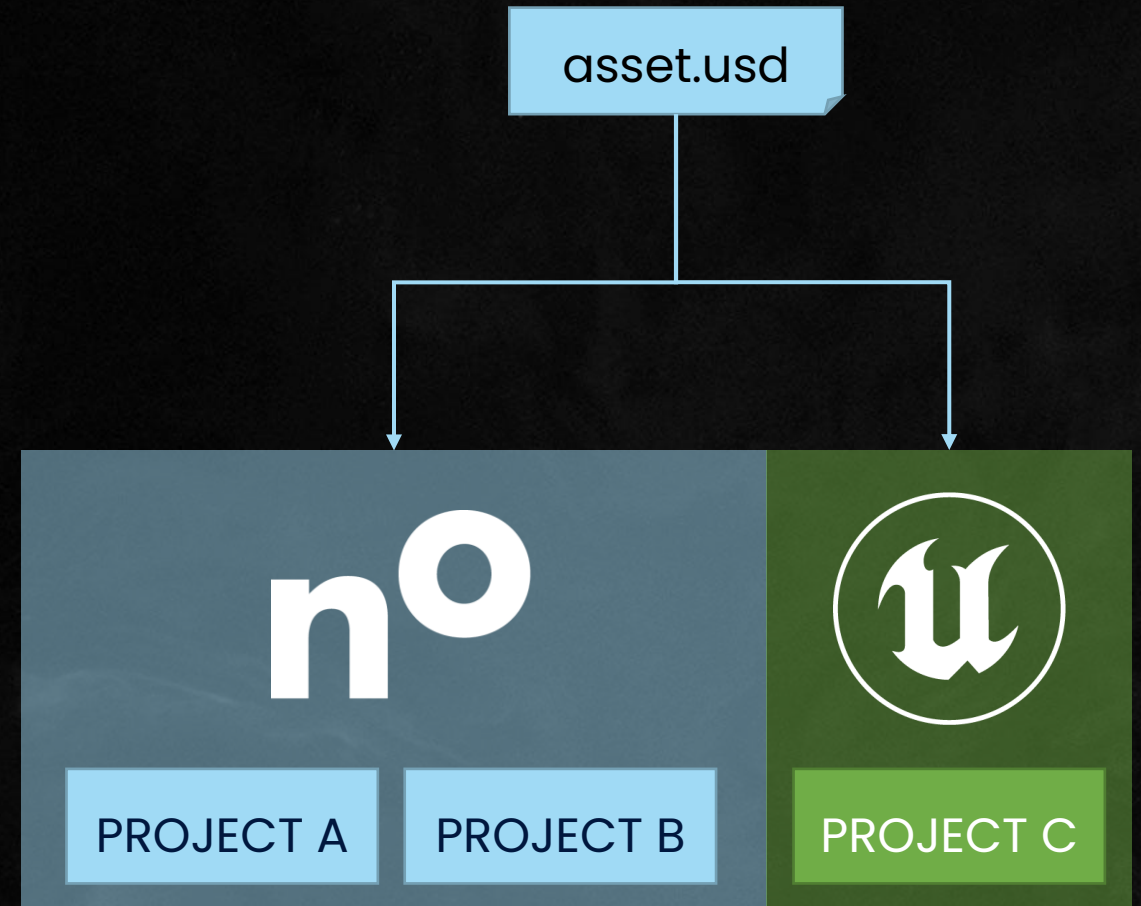
CONTENT - FUTURE

- Most data as USD



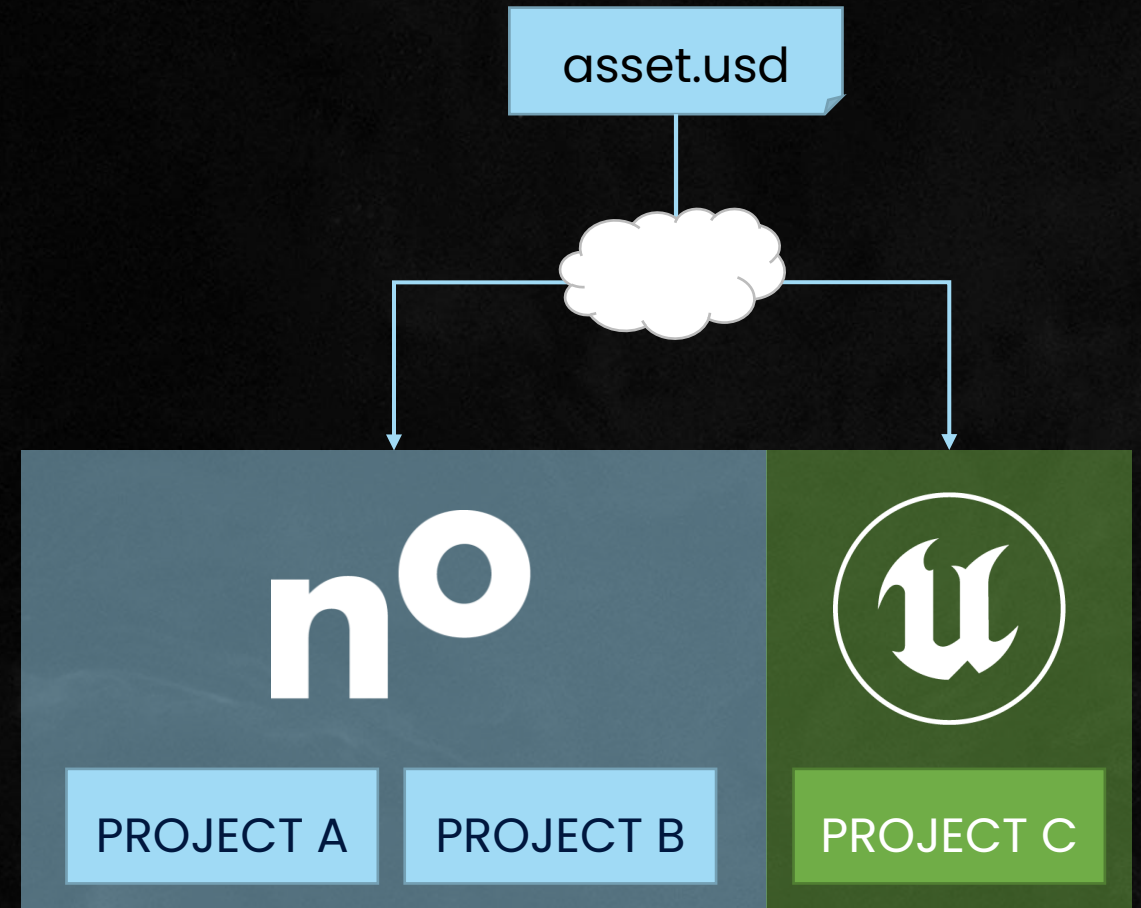
CONTENT - FUTURE

- Most data as USD
- Reusable assets across projects and engine



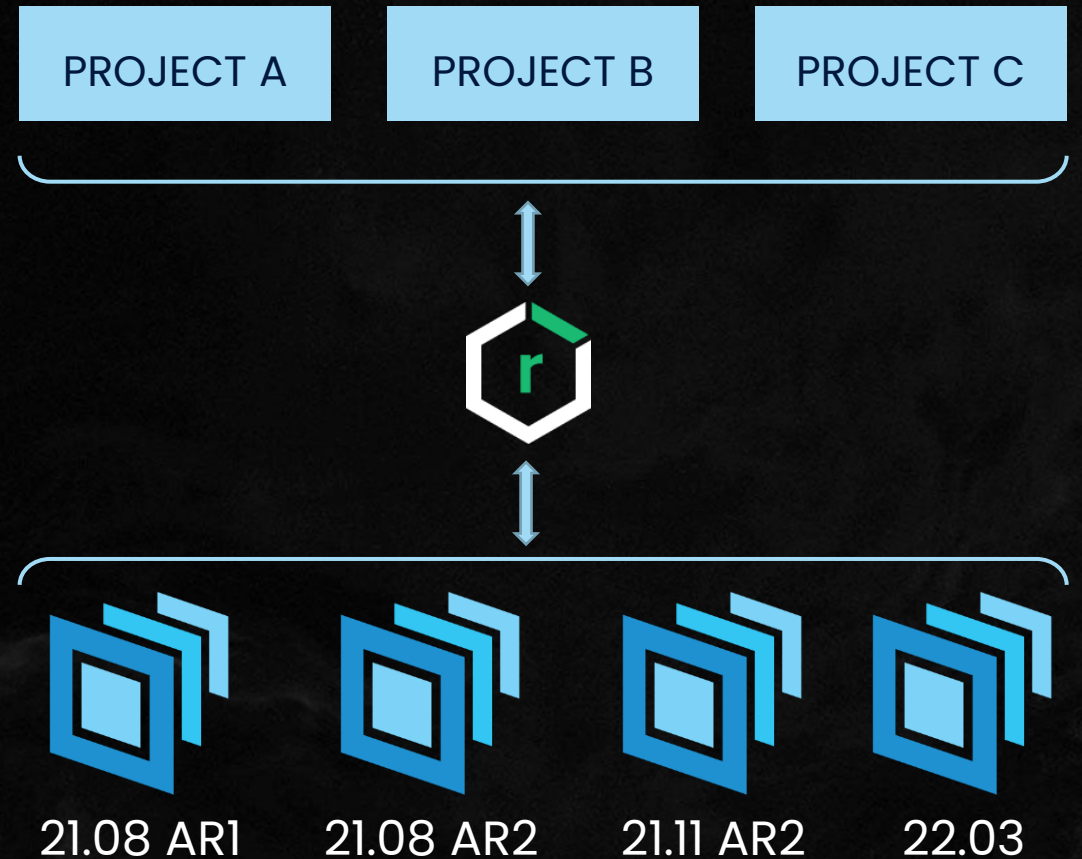
CONTENT - FUTURE

- Most data as USD
- Reusable assets across projects and engine
- No filesystem



CONTENT - FUTURE

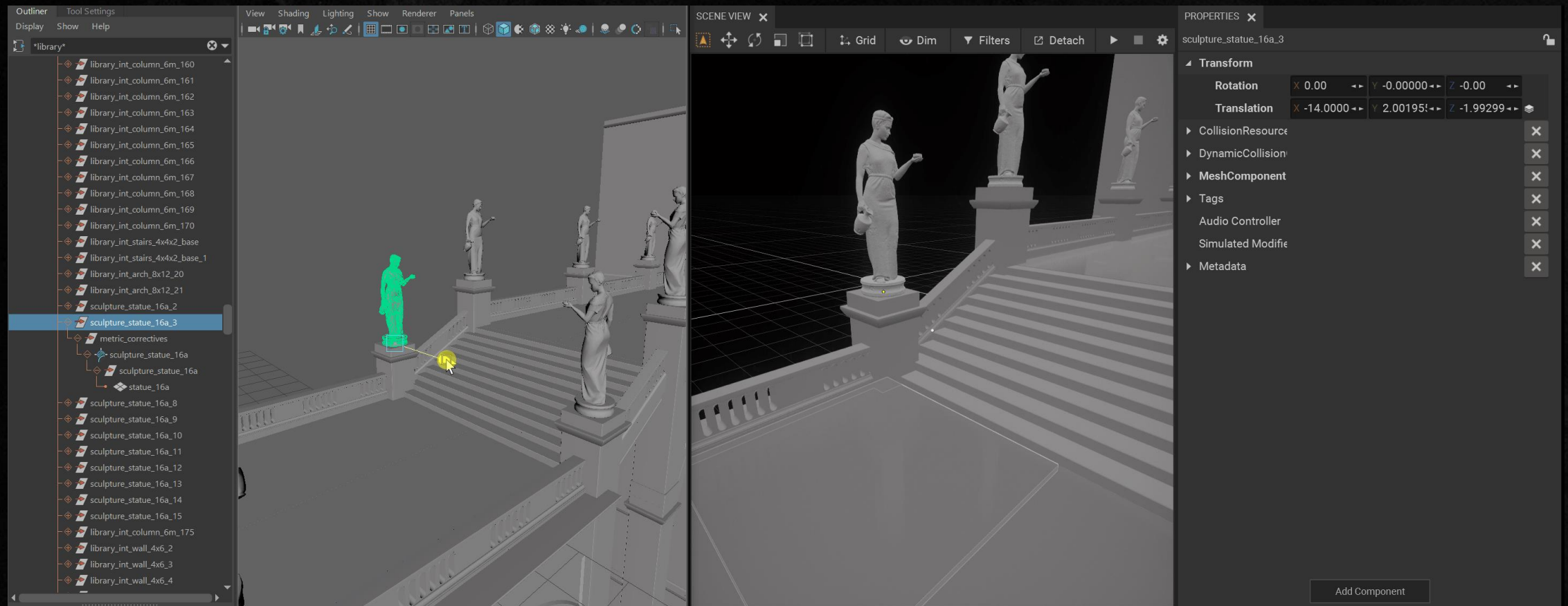
- Most data as USD
- Reusable assets across projects and engine
- No filesystem
- USD and DCC Plugins, Schemas,... installed on demand, per project



CONTENT - FUTURE

- Most data as USD
- Reusable assets across projects and engine
- No filesystem
- USD and DCC Plugins, Schemas,... installed on demand, per project
- Live Editing with DCCs

LIVE EDITING CONTENT



WORKING GROUP TOPICS

- Hierarchy iterative restructuring (deletion, moving, relationships, etc...)
- Schema changes/additions and hotloading
- Variant introspection
- Current Edit Target editable content introspection
- List-editing uniform properties?
 - E.g., composing multiple skeletons into one
- Missing schemas like Cloth/Destruction/Animation Curves



USD

For more details, ask us on slack!

