Solving the Game Content Problem

Pipeline theory and practice

Koray Hagen

Background

Senior Programmer

Tools and Core Technology

SIEA, Santa Monica Studio

Focus:

Animation systems

Content build system

Content compiler tools

Game industry has a big problem

The problem is data

- The time required to develop games is increasing

- The time required to develop games is increasing
- The manpower and computational cost of developing games is increasing

- The time required to develop games is increasing
- The manpower and computational cost of developing games is increasing
- The unit price of a game has remained relatively stable

Meanwhile …

- Player expectations for graphical fidelity are increasing

- Player expectations for graphical fidelity are increasing
- Player expectations for the scale of the user experience are increasing

- Player expectations for graphical fidelity are increasing
- Player expectations for the scale of the user experience are increasing
- Developer expectations for our tools to meet this demand are increasing

- This problem has manifested itself in the form of impacted iteration times:
  - For Artists
  - For Designers
  - For Programmers
  - For Production
  - For Operations

- This problem has manifested itself in the form of impacted iteration times:

- Questions:
  - How long until a content author sees the result of changing ____ ?
  - How long until a content author can commit a change to ____ ?

- This problem has manifested itself in the form of impacted iteration times:
- Questions:
- Answers:
  - Desired? Instantly
  - Reality? Possibly minutes to hours

This problem is the domain of this lecture

It is also the domain of your game’s pipeline

The game content pipeline

- Purpose:
  - Transform unmeaningful data into meaningful data

The game content pipeline

- Purpose:
  - Transform unmeaningful data into meaningful data
  - Often, that means source data into production data

The game content pipeline

- Purpose:
  - Transform unmeaningful data into meaningful data
  - Often, that means source data into production data
  - Production data is content that ends up in a player's hands

The game content pipeline

- Purpose:

- Game content:
  - Meshes
  - Textures
  - Animations
  - Shaders
  - Materials
  - Physics
  - Scripts
  - Narrative
  - Audio

The game content pipeline

- Purpose:

- Game content:
  - Heterogeneous:
    - Has different intrinsic characteristics (formats, disk-space impact)
    - Requires different applied transformations

The game content pipeline

- Purpose:

- Game content:
  - Heterogeneous:
    - Has different intrinsic characteristics (formats, disk-space impact)
    - Requires different applied transformations
  - Hierarchical relationships
    - Dependencies
    - Content identity

The game content pipeline

- Philosophy:
  - Create a unified content pipeline system and architecture at our studio

The game content pipeline

- Philosophy:
  - Create a unified content pipeline system and architecture at our studio
  - Properties:
    - Correct
    - Expressive
    - Explicit
    - Homogenous
    - Scalable
    - Debuggable

This is a lecture on how we think and reason about achieving these desired properties

The agenda

- Foundations
  - Creating the building blocks of a content pipeline

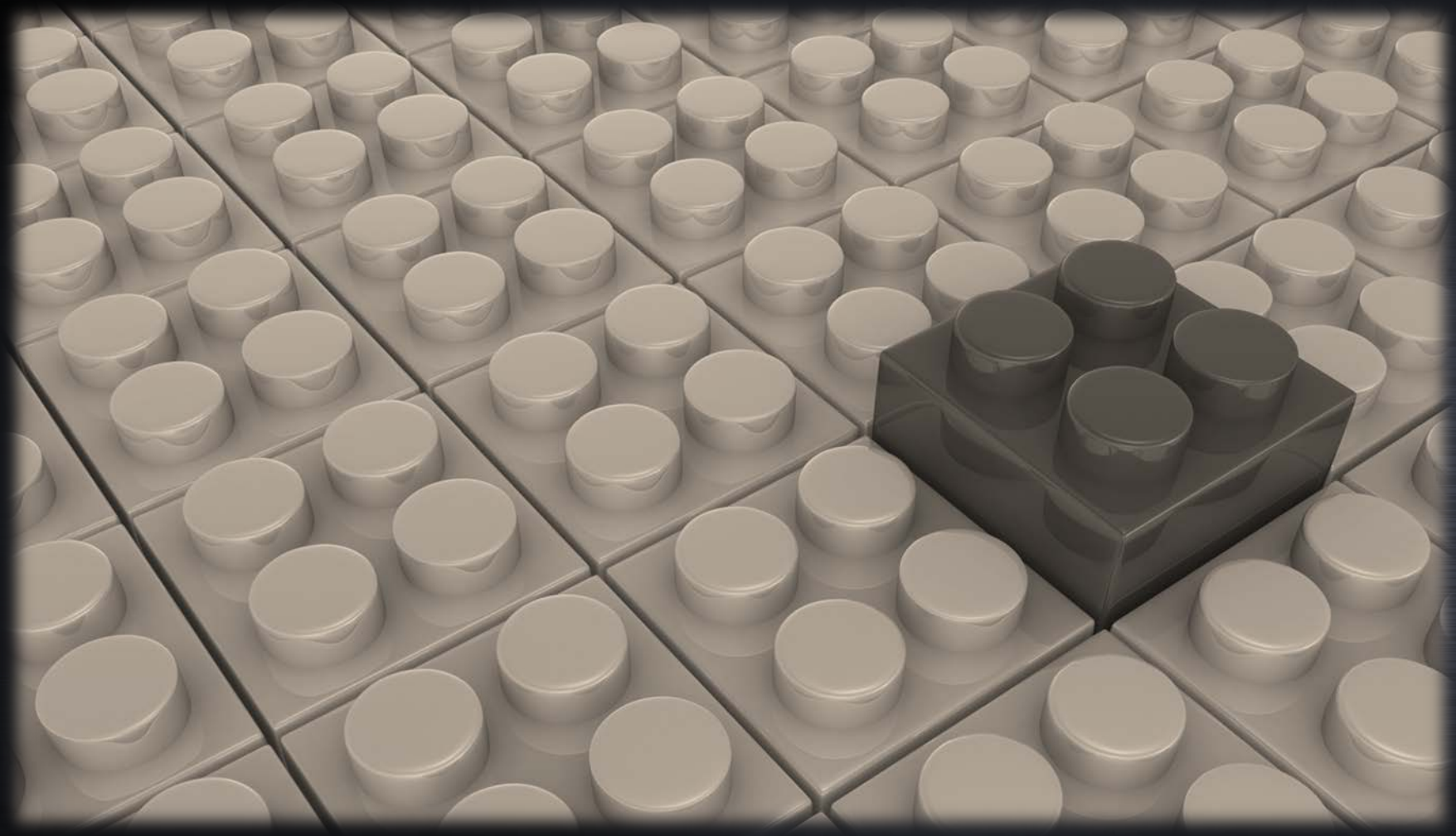
The agenda

- Foundations

- Theory and architecture
  - Considerations for building a pipeline
  - High level view of design

The agenda

- Foundations
- Theory and architecture
- Lessons and current research
  - Concrete technology examples
  - Questions for the future



Foundations

Each piece represents a conceptual building block

Data transformation functions

- Purpose:
  - Core operator of a game's pipeline
  - Transforms a unit of data from one state to another state

Data transformation functions

- Purpose:

- Example:

```
1  struct mesh { /* members */ };
2  struct mesh_opt { /* members */ };
3
4  bool mesh_to_optimized_mesh(const mesh* src, mesh_opt* dst);
5  bool merge_optimized_mesh(const mesh_opt** src, mesh_opt* dst);
```

Data transformation functions

- Purpose:
- Example:
- Invariants:
  - Serializable
  - Transferrable
  - Side-effect free
  - Discrete
  - Atomic

Data transformation functions

- Purpose:
- Example:
- Invariants:
  - Serializable
    - Both inputs and outputs are representable in different contexts
      - Memory mapped
      - File system
      - Networked

Data transformation functions

- Purpose:
- Example:
- Invariants:
  - Serializable
  - Transferrable
    - Usable within a game’s pipeline or run-time

Data transformation functions

- Purpose:
- Example:
- Invariants:
  - Serializable
  - Transferrable
  - Side-effect free
    - Safely usable in a concurrent system
    - Knowable state and always debuggable

Data transformation functions

- Purpose:
- Example:
- Invariants:
  - Serializable
  - Transferrable
  - Side-effect free
  - Discrete
    - Conceptually simple to reason about
    - Represents a single unit of transformation

Data transformation functions

- Purpose:
- Example:
- Invariants:
  - Serializable
  - Transferrable
  - Side-effect free
  - Discrete
  - Atomic
    - Transformation either did or did not occur, there is no half-way

Data transformation functions

- Goals:
  - Develop a library of transformations
  - Define an appropriate granularity
  - Feverously respect invariants

Data transformation functions

- Goals:
  - Develop a library of transformations
    - Needed to represent the full scope of game data
  - Define an appropriate granularity
  - Feverously respect invariants

Data transformation functions

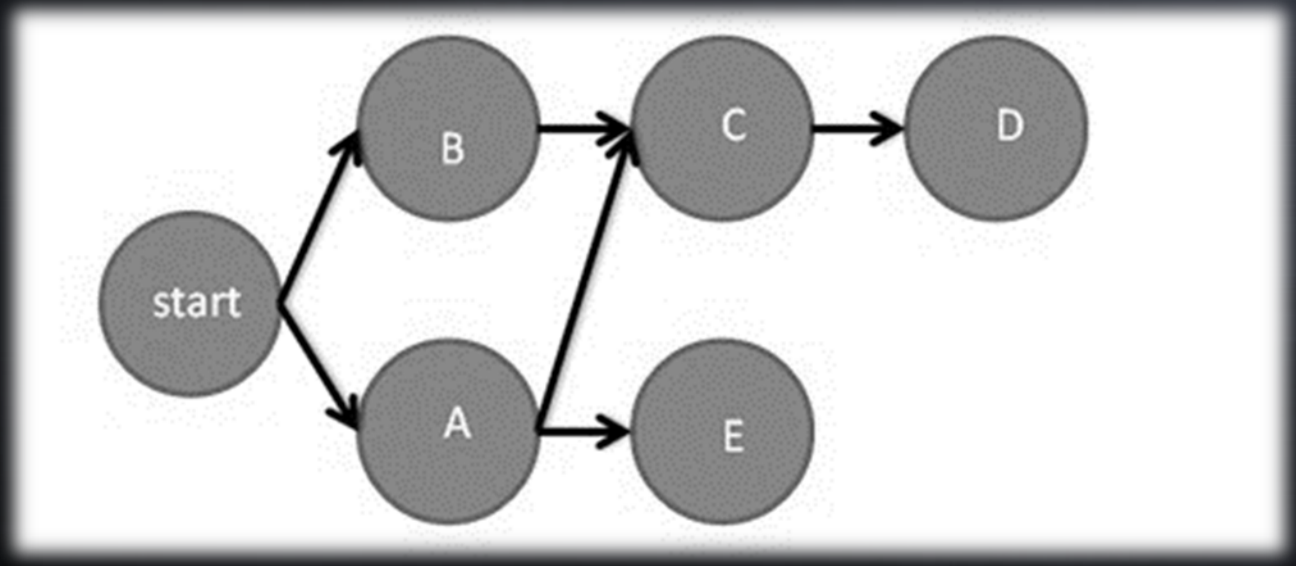
- Goals:
  - Develop a library of transformations
  - Define an appropriate granularity
    - Base judgment on time and resources required for the transform
  - Feverously respect invariants

Data transformation functions

- Goals:
  - Develop a library of transformations
  - Define an appropriate granularity
  - Feverously respect invariants
    - This will allow powerful architectural opportunities later

Dependency graph

- Purpose:
  - Track the global state of data and relationships
  - Serves as an interface to both data and transformations



Dependency graph

- Purpose:
- Data dependency graph
- Transformation dependency graph

Dependency graph

- Purpose:

- Data dependency graph
  - The relationship network of data files
  - Example:
    - Skybox
      - Cubemap
        - 1D Texture (six)
        - Cubemap Texture

- Transformation dependency graph

Dependency graph

- Purpose:

- Data dependency graph
  - The relationship network of data files
  - Example:
  - Properties:
    - File identity (as a path or GUID)
    - Connectivity to and from other files
    - Reachability to and from other files

- Transformation dependency graph

Dependency graph

- Purpose:
- Data dependency graph
- Transformation dependency graph
  - The relationship network of data transformation functions
  - Example:
    - `bool cubemap_to_skybox(const cubemap* src, skybox* dst);`
      - `bool textures_to_cubemap(const texture** src, cubemap* dst);`
        - `bool tga_to_texture(const tga* src, texture* dst);`

Dependency graph

- Purpose:
- Data dependency graph
- Transformation dependency graph
  - The relationship network of data transformation functions
  - Example:
  - Properties and differences from data dependency graph:
    - Requires critical path evaluation for a desired result
    - Requires deciding if and when a transformation occurs
    - Performance critical

Storage

- Purpose:
  - Provide permanent or transient contexts for information and data

Storage

- Purpose:

- Considerations:
  - Often a tradeoff between latency, capacity, and longevity
    - Example: low latency, in-memory key-value database
    - Example: high latency, networked file system

Storage

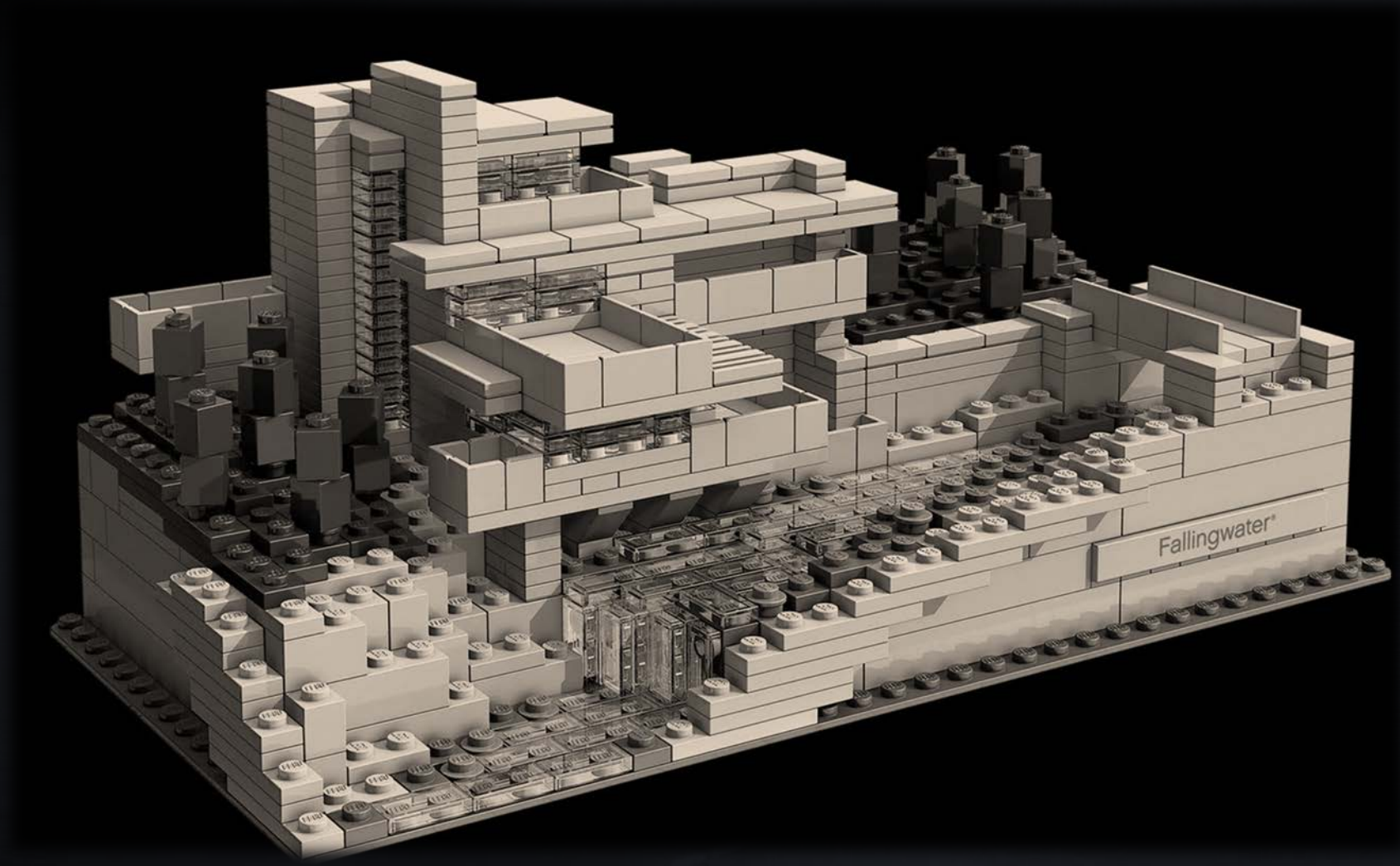
- Purpose:

- Considerations:
  - Often a tradeoff between latency, capacity, and longevity
  - Needed for many areas in a pipeline:
    - Source data
    - Intermediate data
    - Inter/Intraprocess communication
    - Production data

Storage

- Purpose:

- Considerations:
  - Often a tradeoff between latency, capacity, and longevity
  - Needed for many areas in a pipeline:
  - Should never be directly interacted with by transformation functions:
    - Fundamentally breaks invariants
    - Should be orchestrated at a higher level
      - During evaluation of the transformation dependency graph



Theory and architecture

Computational work

- Iteration times:
  - For content authors, improving iteration is purely a metric of time
    - The amount of computation required is not relevant

Computational work

- Iteration times:
  - For content authors, improving iteration is purely a metric of time
    - The amount of computation required is not relevant
  - Often, more computation results in more time required

Computational work

- Iteration times:

- Mitigation strategies:
  - Data model
  - Concurrency
  - Avoidance

Computational work

- Iteration times:

- Mitigation strategies:
  - Data model
    - Reduction of transformation steps required to see result
      - Transformations are unnecessary or optional
      - Direct optimization or removal of transformations

  - Concurrency
  - Avoidance

Computational work

- Iteration times:

- Mitigation strategies:
  - Data model
    - Reduction of transformation steps required to see result
    - Ideal to address, the best general optimization is to do less work
  - Concurrency
  - Avoidance

Computational work

- Iteration times:
- Mitigation strategies:
  - Data model
  - Concurrency
    - Leveraging the invariants of our data transformation functions
    - Available contexts:
      - Threads
      - Processes
      - Network
  - Avoidance

Computational work

- Iteration times:
- Mitigation strategies:
  - Data model
  - Concurrency
  - Avoidance
    - Hierarchal caching contexts (more detail later):
      - In-memory
      - File system
      - Network

Computational work

- Iteration times:

- Mitigation strategies:
  - Data model
  - Concurrency
  - Avoidance
    - Hierarchal caching contexts:
    - Pruning action based on criteria:
      - Unneeded for desired result
      - Redundant

Conceptual architectures

- Live iteration
- Asset baking

Conceptual architectures

- Live iteration
  - Working directly in the game to see changes
  - Streaming data to the game through an external tool
- Asset baking

Conceptual architectures

- Live iteration
  - Working directly in the game to see changes
  - Streaming data to the game through an external tool
  - Properties:
    - Has the benefit of user seeing instant results
    - Has the problem of data transience
      - Edit state could become corrupt
      - Edit state could encounter an irrecoverable error (crashes)
    - Has the problem of needing to reflect changes into source data
      - Other assets may have dependence on changes
- Asset baking

Conceptual architectures

- Live iteration
- Asset baking
  - Using an offline toolset to “cook” or “bake” assets for the game

Conceptual architectures

- Live iteration
- Asset baking
  - Using an offline toolset to “cook” or “bake” assets for the game
  - Properties:
    - Has the benefit of permanence
    - Has the benefit of simple directional flow of changes
      - Source → intermediate → production
    - Has the problem, often, of increased computational cost
    - Has the problem, often, of non-uniformity
      - Example: Different teams having different file formats
      - Resulting in a more complex unified pipeline

Conceptual architectures

- Live iteration
- Asset baking

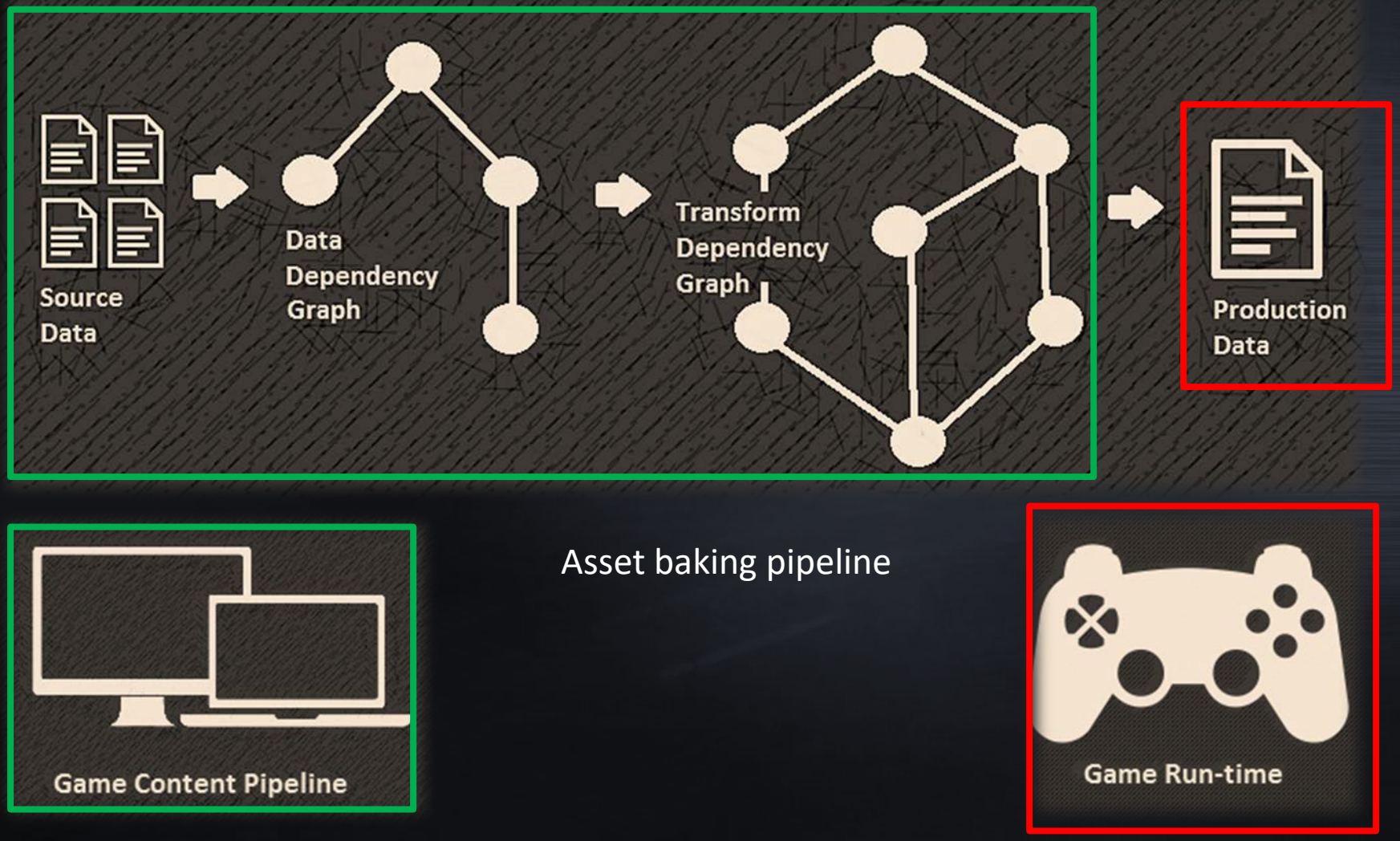
- Philosophy:
  - Live iteration workflows and asset baking can be reconciled
  - Both concepts are shades of the same architectural principles

Conceptual architectures

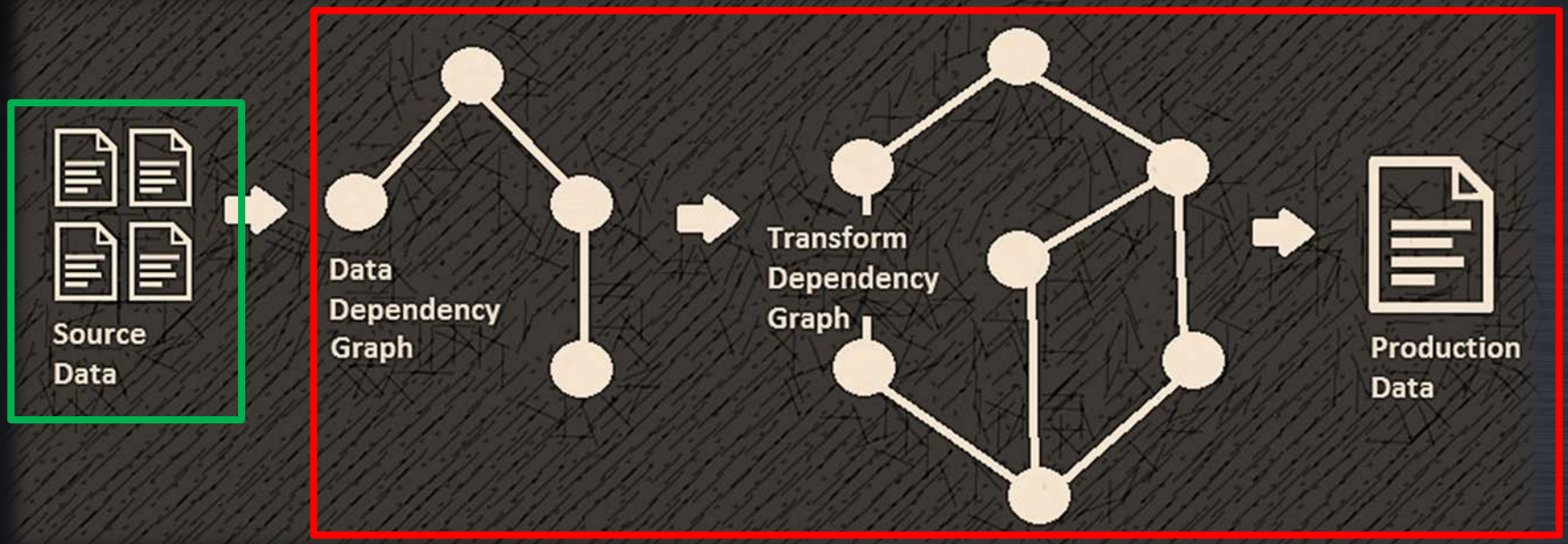
- Live iteration
- Asset baking
- Philosophy:

- Desired pipeline architecture:
  - Transferrable library of available data transformations
  - Full data dependency graph of all game content
  - Computable transformation dependency graphs
  - Generalized hierarchical caching model

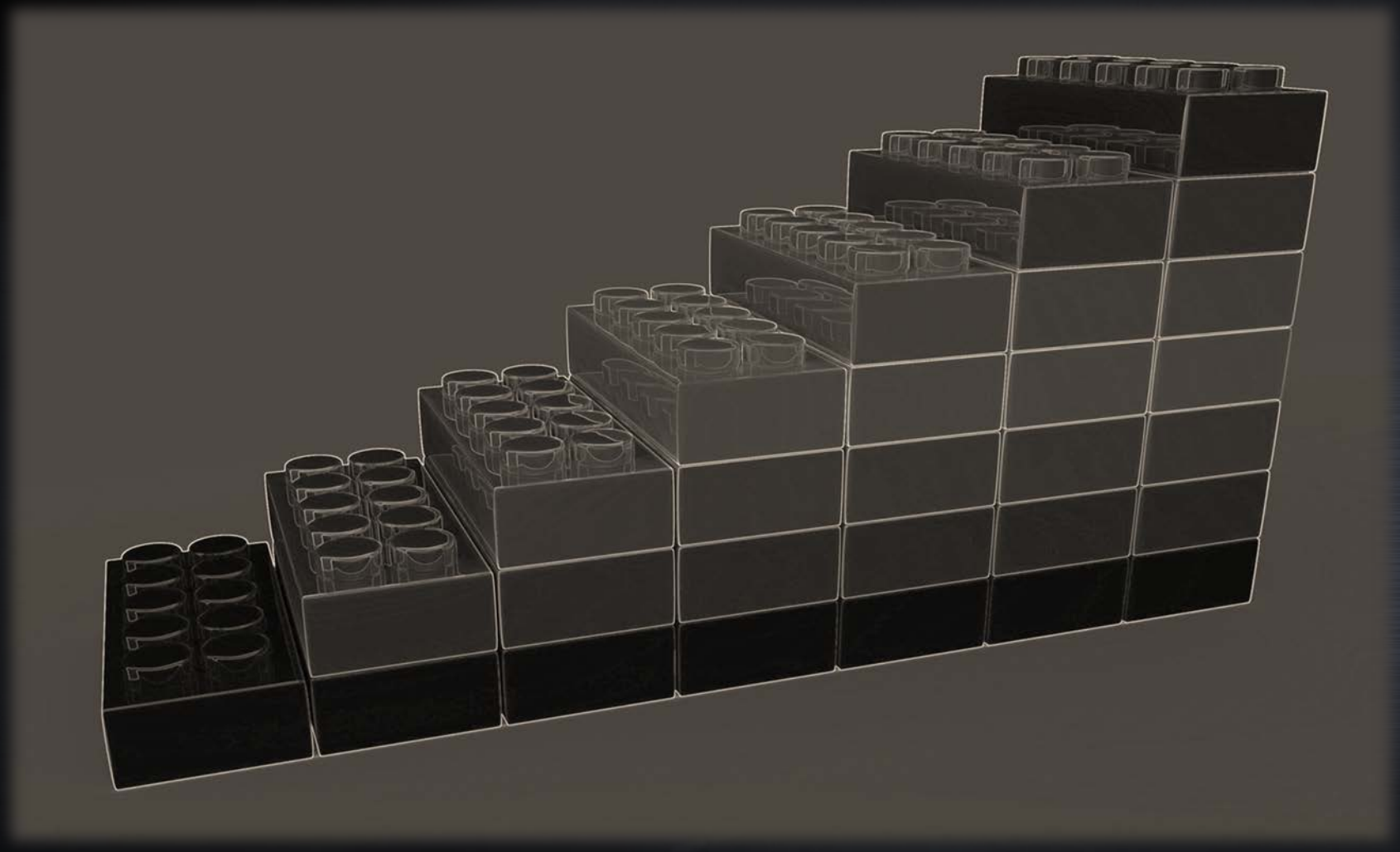
Visualization of a content build



Visualization of a content build



Live iteration pipeline



Lessons and current research

Data transformation functions

- Lesson: Define the data formats
- Lesson: Discipline in respecting invariants
- Lesson: Transferability is crucial

Data transformation functions

- Lesson: Define the data formats
  - Developed an in-house object serialization library
    - Similar to Google Flat Buffers and Protocol Buffers

- Lesson: Discipline in respecting invariants
- Lesson: Transferability is crucial

Data transformation functions

- Lesson: Define the data formats
  - Developed an in-house object serialization library
    - Similar to Google Flat Buffers and Protocol Buffers
    - Designed for:
      - Memory-mappability
      - Binary format for speed, Json format for debugging
      - Data layout optimization
      - Versioning and Identification
      - Automatic API generation
      - Custom allocators
- Lesson: Discipline in respecting invariants
- Lesson: Transferability is crucial

Data transformation functions

- Lesson: Define the data formats
- Lesson: Discipline in respecting invariants
  - Allows for concurrency guarantees
  - Allows for data, build, and game state guarantees

- Lesson: Transferability is crucial

Data transformation functions

- Lesson: Define the data formats
- Lesson: Discipline in respecting invariants
- Lesson: Transferability is crucial
  - Same library of functions can be used at bake time and run time
  - Opportunity for the transformation dependency graph to computationally decide where to execute transform:
    - A thread
    - A local process
    - A networked process

Dependency Graphs

- Lesson: Most research required here
- Lesson: Describing the flow of a network
- Lesson: Graph operators

Dependency Graphs

- Lesson: Most research required here
  - *Open problem*: Scaling computation to millions of nodes
  - *Open problem*: Cache coherent spatial data structures and processing
  - *Open Problem*: Extremely performant transformation dependency graph evaluation and iterative construction

- Lesson: Describing the flow of a network
- Lesson: Graph operators

Dependency Graphs

- Lesson: Most research required here
- Lesson: Describing the flow of a network
  - Transformation dependency graph
    - Required a language (DSL) to describe the transformation network
    - Transformations known as commands:
      - Examples of granularity:
        - » compress_texture
        - » create_animation_clip
    - DSL results in vast amount of automatic code generation
- Lesson: Graph operators

Dependency Graphs

- Lesson: Most research required here
- Lesson: Describing the flow of a network
- Lesson: Graph operators
  - Transformation dependency graph traversal:
    - Two operators could encapsulate our most complex operations
    - Reductions:
      - Compute a subgraph under a node as input to a command
    - Maps:
      - Direct input node to a command

Storage

- Lesson: File system by itself is very limiting
- Lesson: Creating a global asset library

Storage

- Lesson: File system by itself is very limiting
  - Butting heads with Windows file system rules is painful:
    - Path lengths (256 characters)
    - Path and file name uniqueness
      - Often results in insane naming conventions for files

- Lesson: Creating a global asset library

Storage

- Lesson: File system by itself is very limiting
- Lesson: Creating a global asset library
  - Many reasons for desirability:
    - Freedom for custom identification and file versioning
    - Freedom from file system path and naming conventions
    - Filenames on physical disk can be mangled, while the asset library can ensure human readability

Caching

- Lesson: Hierarchical caching is a powerful model
  - Goal: Never re-compute a function with the same parameters.
  - Achieved by decorating functions with appropriate caching strategies:
    - Match a function with the most appropriate caching semantic
    - Driven by the nature of input and output data
    - Example:
      - File-based parameters cached to the file system or network
      - Normal function parameters cached in memory

Caching

- Lesson: Hierarchical caching is a powerful model
  - Any piece of computation can be decorated with caching
    - Found it to be most beneficial with:
      - Path and string manipulation
      - Data transformation functions

Caching

- Lesson: Hierarchical caching is a powerful model
  - Any piece of computation can be decorated with caching
  - Hiding caching structure behind generalized interfaces and decorators allowed for experimentation.
    - Proved to be invaluable when we switched our file-system based key-value store from SQLite to LMDB

Caching

- Lesson: Hierarchical caching is a powerful model
  - Any piece of computation can be decorated with caching
  - Hiding caching structure behind generalized interfaces and decorators allowed for experimentation.
  - Our current implementations:
    - In-memory: achieved with function level memoization
    - File-system: achieved with LMDB
    - Networked: achieved with an in-house technology backed by CEPH (a networked file system object store)

Caching

- Lesson: Hierarchical caching is a powerful model
  - Example:

```
1  [global_cached]
2  void execute_command(command* func, const file_node* src, file_node* dst) {
3      // func merge_optimized_mesh
4      func(src.get(), dst.get())
5  }
6
7  [memoized]
8  bool expensive_operation(mesh* src) {
9      // compute something expensive
10     return true;
11 }
12
13 bool merge_optimized_mesh(const mesh_opt** src, mesh_opt* dst) {
14     return expensive_operation(dst)
15 }
```

Takeaways

Hard Problems

- Building the foundation
- Dependency graph scalability
- Dependency graph decision making

Hard Problems

- Building the foundation
  - Most studios, including ours have a long history of tool's development
  - Difficult to modernize and justify changing monolithic, stable tools
  - Large upfront cost for changing data formats
- Dependency graph scalability
- Dependency graph decision making

Hard Problems

- Building the foundation
- Dependency graph scalability
- Dependency graph decision making
  - Performant and meaningful computation against large graphs is difficult but not insurmountable.
  - Pushing automated decision making to cover:
    - Caching strategies
    - Appropriate contexts for data transformations

The problem is data

The solution is automation

Thank you

Questions?