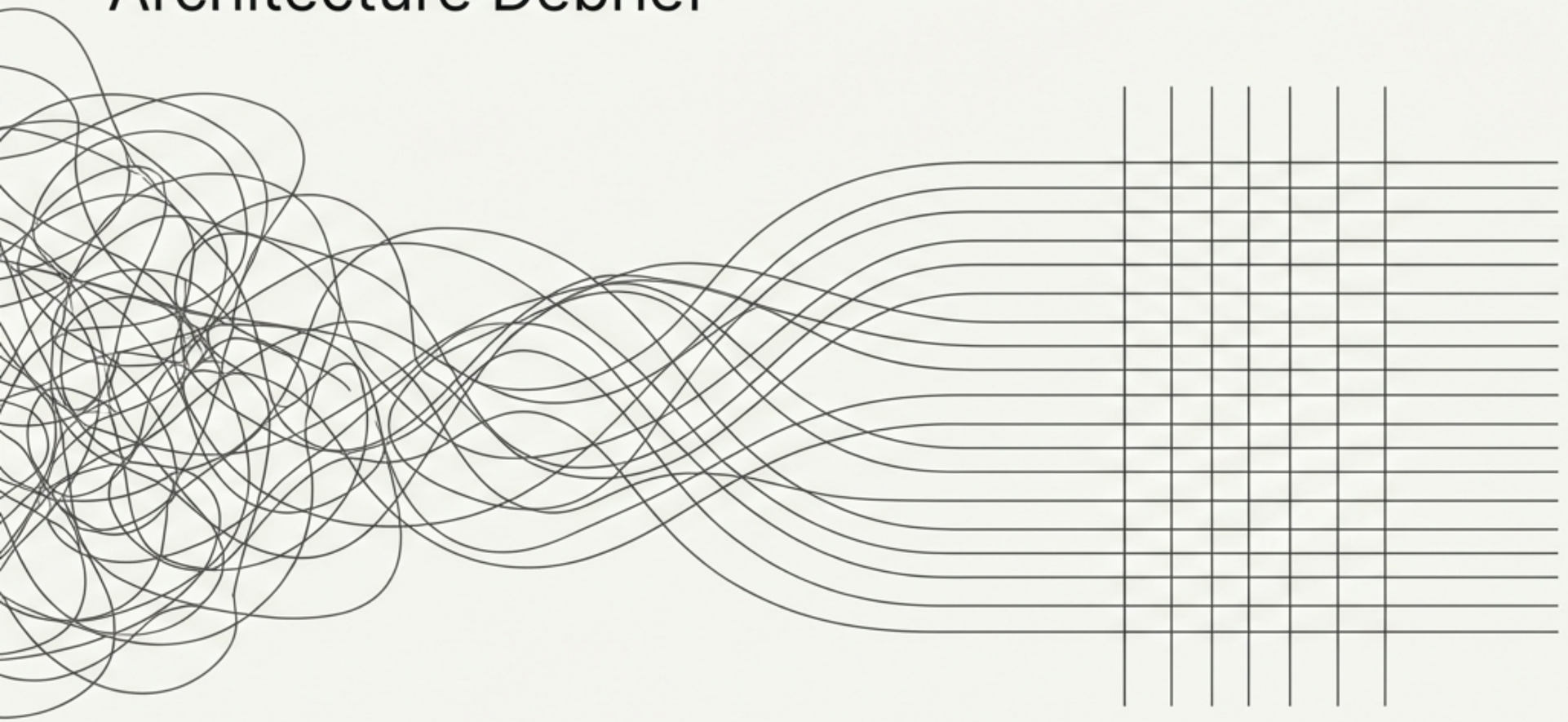
From Chaos to Clarity

The Html_MGraph Multi-Graph Architecture Debrief

50+

Files Refactored



978

Passing Tests

100%

Test Coverage

Project Version: v1.3.0 | Status: ✅ Deployed to AWS Lambda

We Transformed a Monolithic HTML Graph into a Clean, Specialised Architecture

The Html_MGraph refactoring project successfully converted a single, all-purpose graph into a sophisticated 5+1 multi-graph system. This initiative touched over 50 files, significantly improved the transformation pipeline, and achieved full test coverage, all while maintaining complete backward compatibility for existing APIs.

Key Achievements

Metric	Before	After
Architecture	Single monolithic graph	5+1 specialized graphs
Test Count	~800	978
Test Coverage	Partial	100%
Graph Transformations	9 (legacy)	6 (clean)
Export Formats	7	7 (enhanced)

The Old Monolithic Graph Was Creating Noise and Complexity

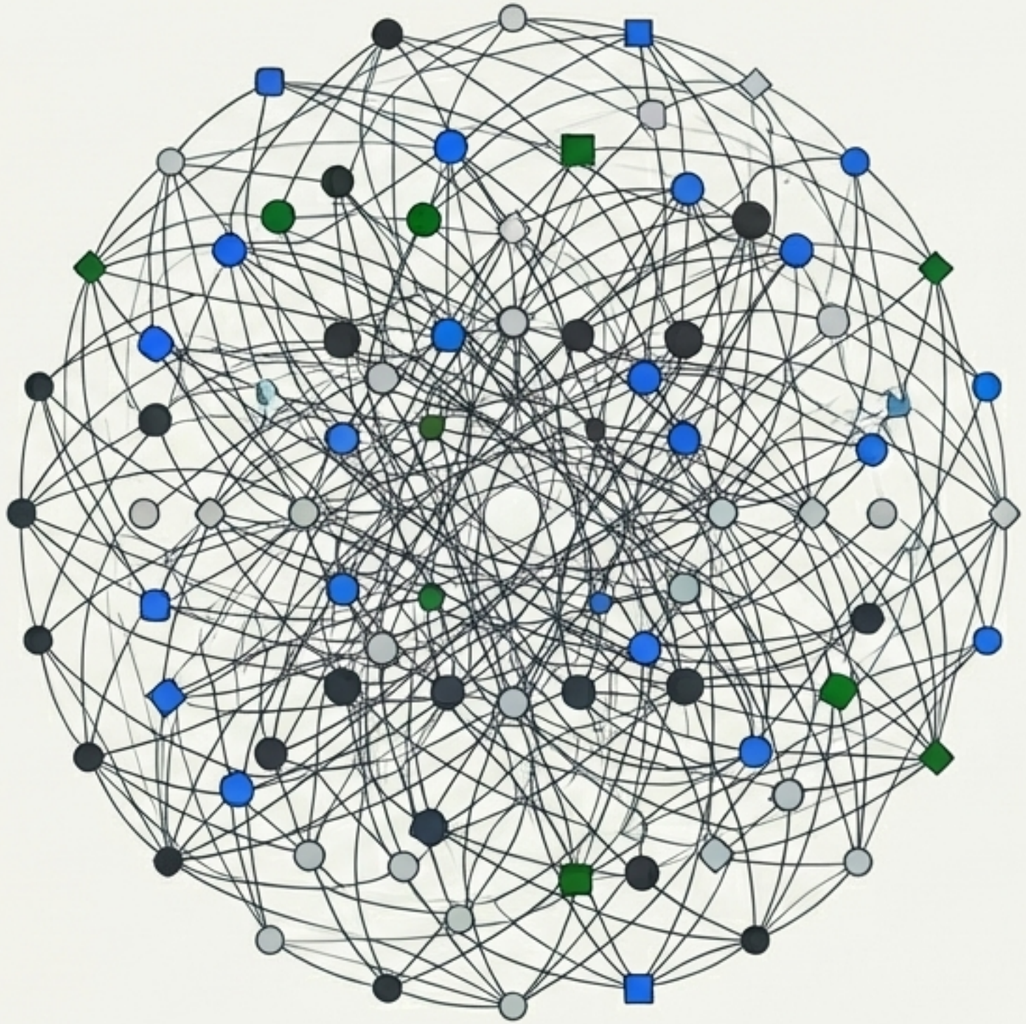
1. **Visualization Noise*: Every export was cluttered with both tag and attribute nodes.

2. **Complex Transformations*: Filters were forced to operate on dictionary structures before graph creation.

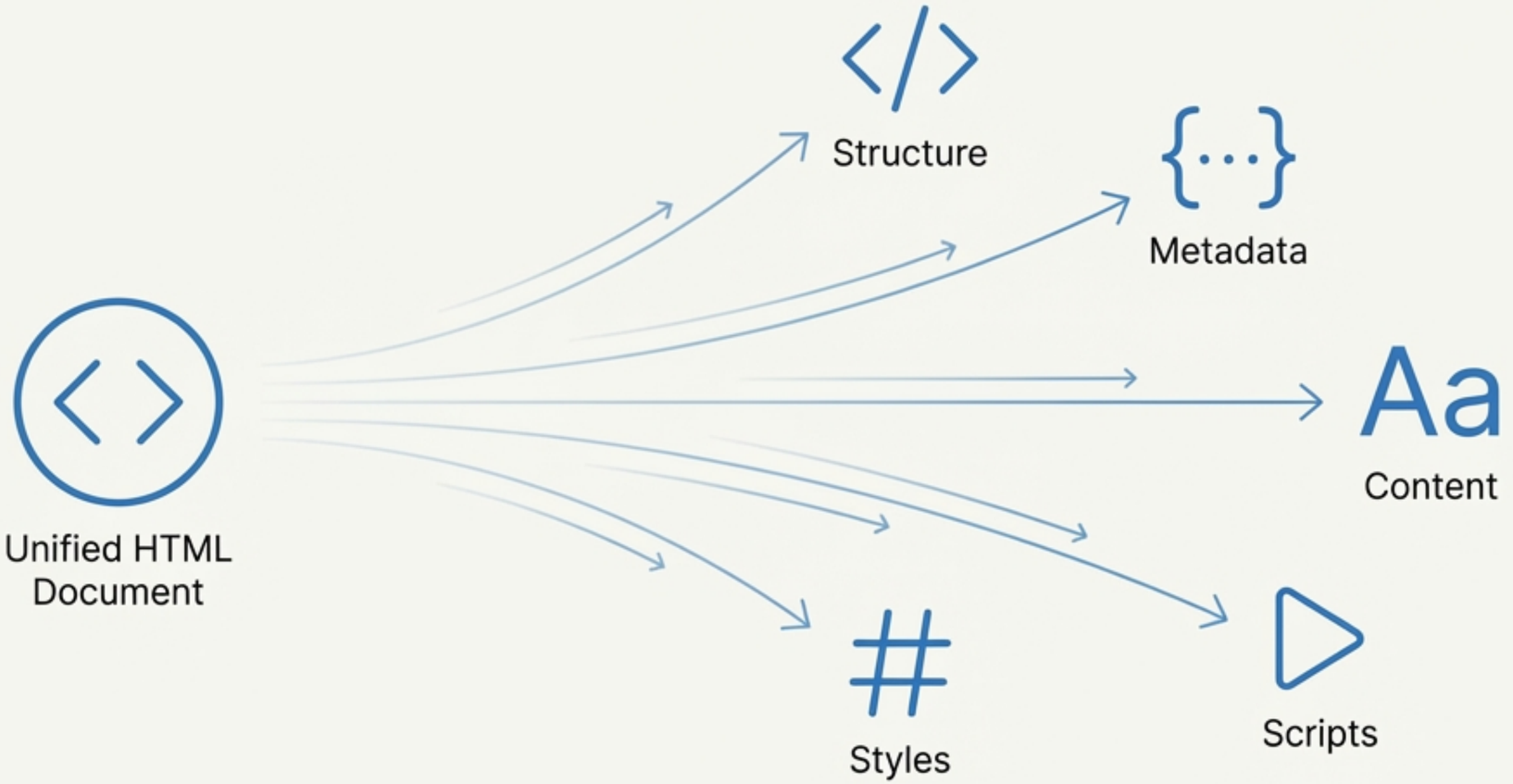
3. **Tight Coupling*: An adapter layer (`Html_Dict__OSBot__To__Html_Dict`) created an unnecessary conversion step.

4. **Head/Body Mixing*: No clean separation existed between document metadata and its content.

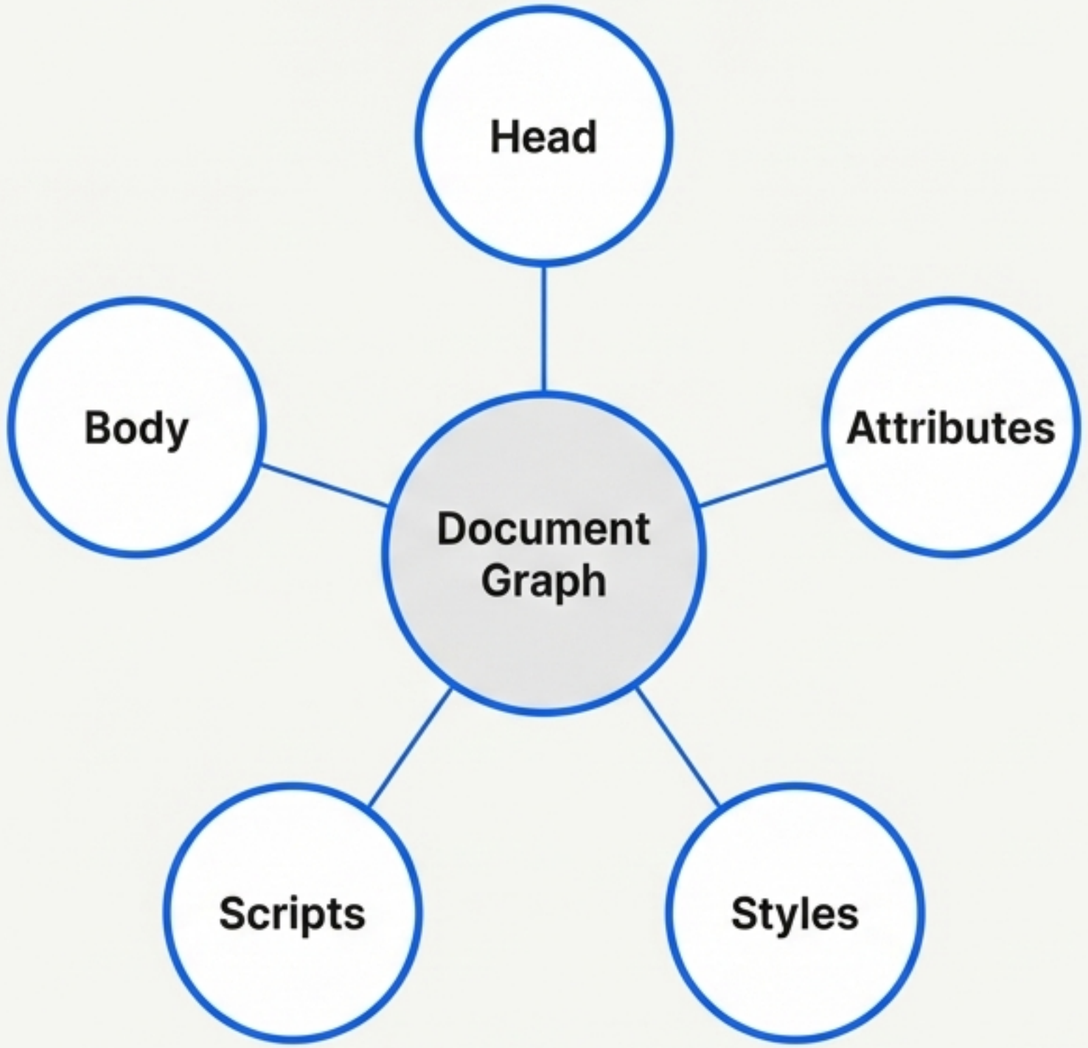
5. **Difficult Analysis*: Extracting simple data like “just the text” required multiple, inefficient passes.



The Core Insight: HTML Has Naturally Separate Concerns



Instead of forcing everything into one graph, we could create a system of specialised graphs, each responsible for a single concern. This would allow for cleaner data models, simpler transformations, and more powerful, targeted analysis.

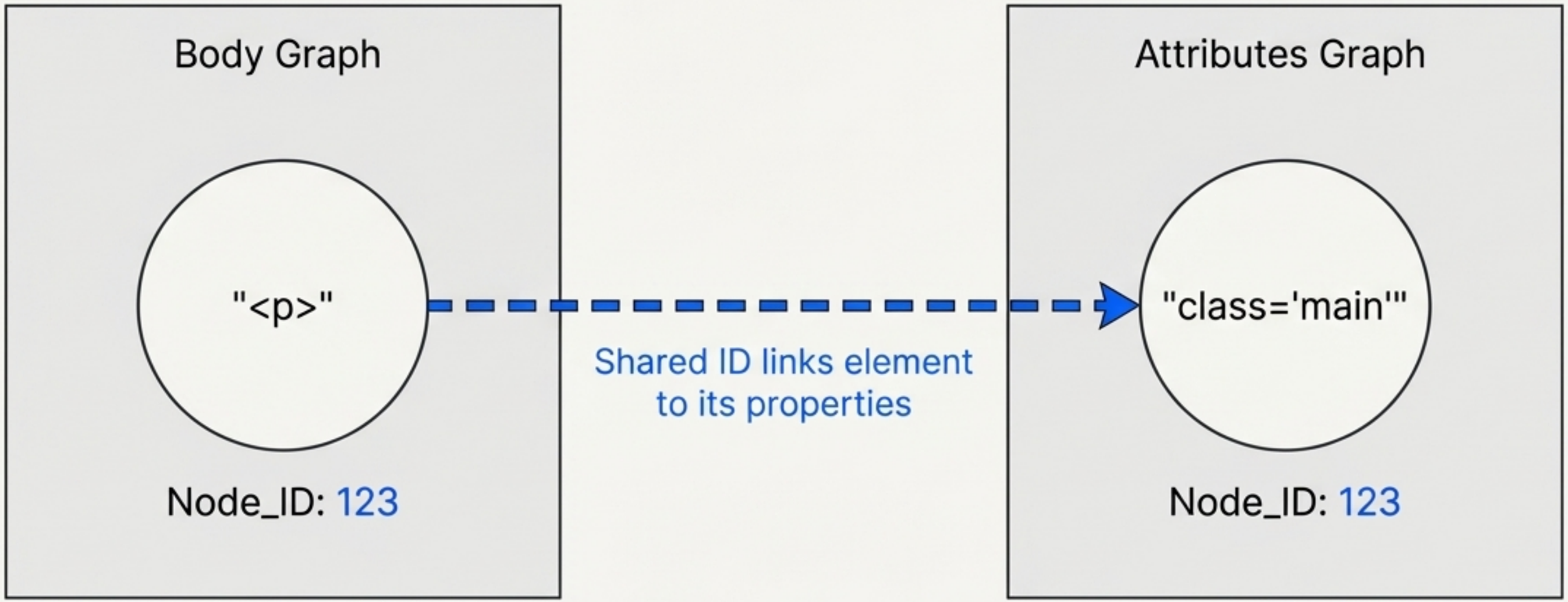


Introducing the 5+1 Multi-Graph Architecture

Graph Responsibilities		
Graph	**Purpose**	**Key Contents**
Document	Root container	**<html>** element, orchestrates sub-graphs
Head	Document metadata	**<meta>**, **<title>**, **<link>**
Body	Visible content	** **, **<p>**, text nodes
Attributes	Element properties	**`class`**, **`id`**, **`href`**, **`src`**
Scripts	JavaScript	**<script>** elements, **`src`** refs
Styles	CSS	**<style>** elements, **`link`** refs

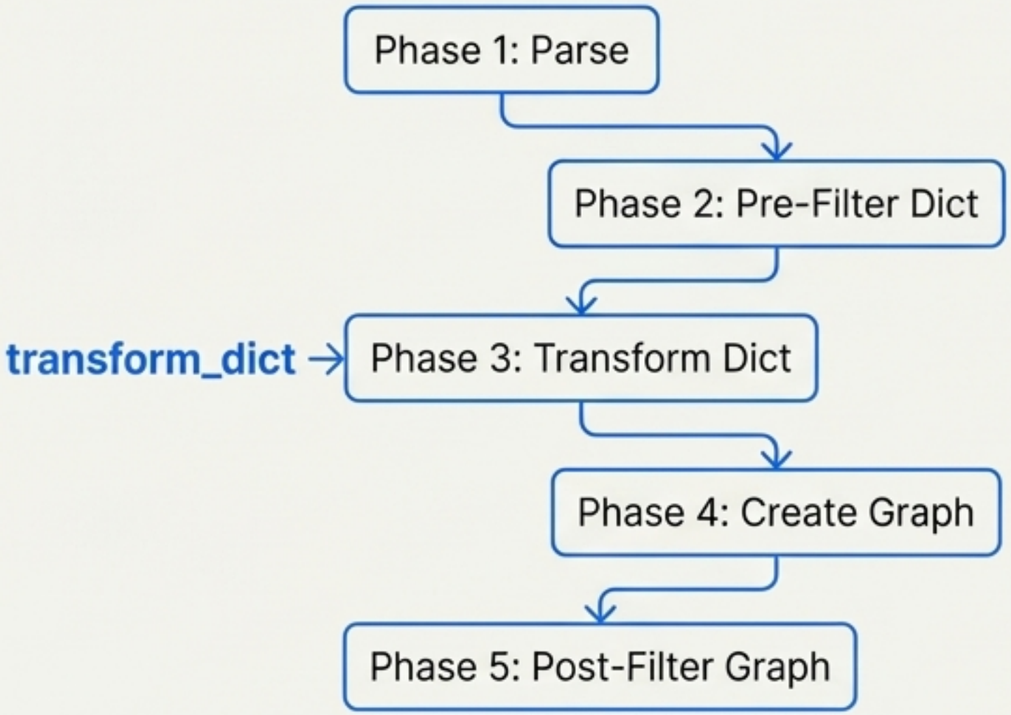
Shared Node_IDs: The Key to Cohesion and Round-Trip Conversion

The breakthrough that connects the specialised graphs is a shared Node_ID pattern. An element in one graph (e.g., the `Body` graph) uses the same unique ID as its corresponding properties in another (e.g., the `Attributes` graph).

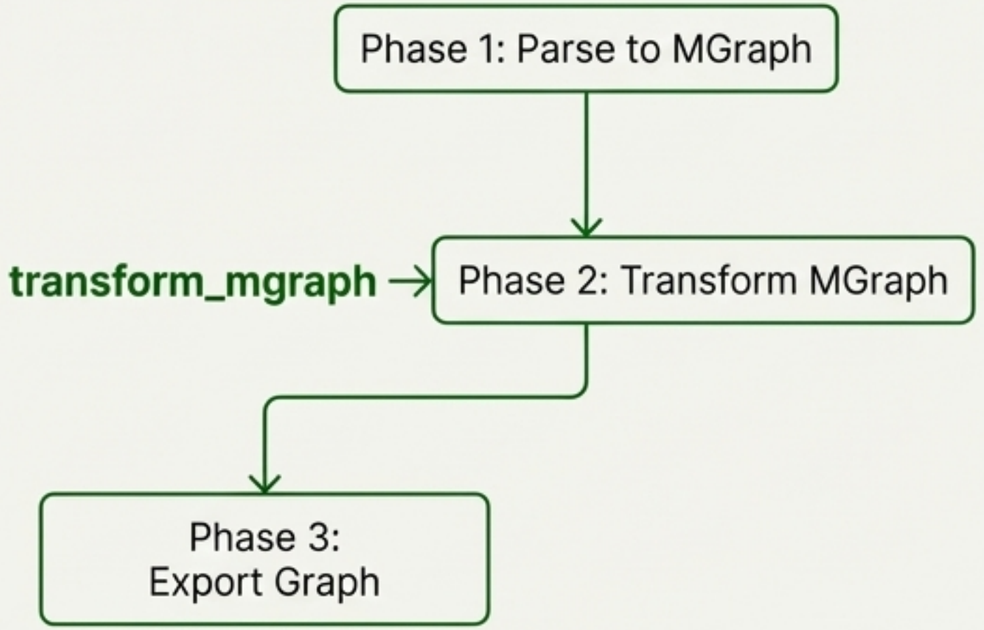


From a 5-Phase Dictionary Pipeline to a Streamlined 3-Phase Graph Pipeline

Before: 5-Phase, Dict-Based



After: 3-Phase, Graph-Based



Rationalising the Transformation Zoo: From 9 Legacy Tools to 6 Focused Ones

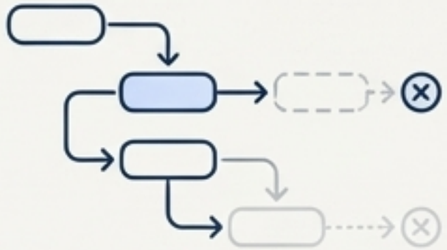
A key part of the refactoring was auditing the existing transformation library. Instead of migrating all nine legacy transforms, we evaluated their utility, merging, rewriting, or removing them as needed.

Legacy Transform	Decision	Rationale / Replacement
collapse_text	Merged	Into `semantic` transform
elements_only	Renamed	Now `structure_only` for clarity
body_only	Rewrote	As a modern export filter
consolidate_text	Removed	Redundant with `semantic`
strip_inline_tags	Removed	Handled by modern parsers
collapse_single_child	Merged	Into `semantic` transform
text_blocks_only	Removed	Rarely used edge case

Overcoming Hurdles: Engineering Principles in Practice

Maintaining Backward Compatibility

Implemented `Html_MGraph` as a Facade. It provides an identical API to the old single-graph system, hiding the new internal complexity from existing consumers.



Incompatible Transformation Pipeline

Rewrote valuable legacy transformations to use new methods (`transform_mgraph`) and deprecated redundant ones. This ensured a clean, modern, and non-repetitive toolset.

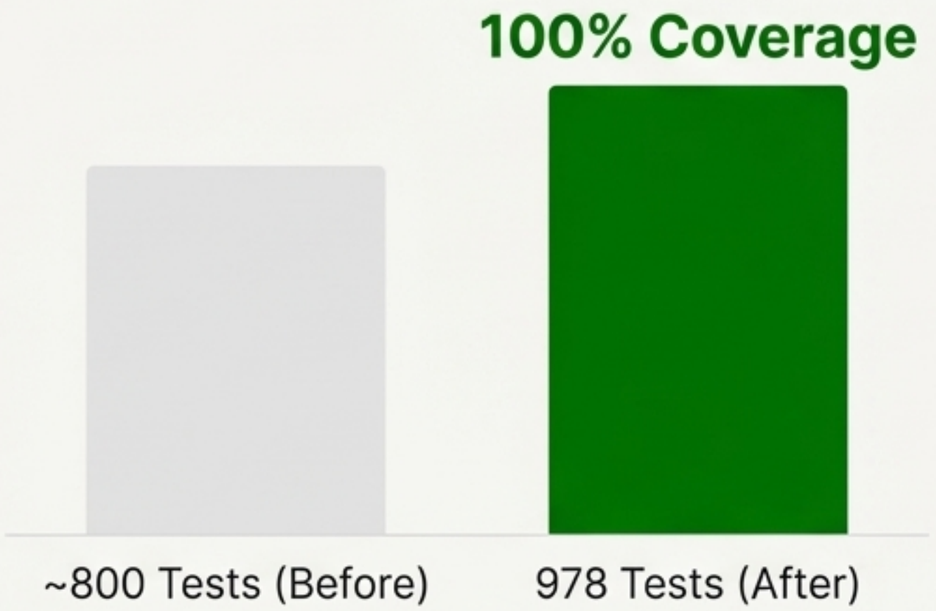
Closing the Test Coverage Gap

Implemented a comprehensive testing strategy for the new multi-graph code paths. This increased the test suite from **~800 to 978**, achieving **100% coverage**.

A Quantifiable Leap Forward in Architecture and Quality

Aspect	Before	After
Architecture	Single monolithic graph	**5+1 specialized graphs**
Separation of Concerns	Mixed	**Clean (head/body/attrs/scripts/styles)**
Transformation Pipeline	5-phase (dict-based)	**3-phase (graph-based)**
Transformations	9 (many legacy)	**6 (focused, useful)**
Test Coverage	Partial	**100%**
Round-trip Support	Complex & manual	**Built-in via shared Node_IDs**

Rock-Solid Confidence: 978 Tests Providing 100% Coverage



This comprehensive test suite is integrated into our CI/CD pipeline, giving us the confidence to make aggressive changes to the system while ensuring stability and correctness.

Lessons Learned from the Refactoring Process

What Worked Well ✅

- **Facade Pattern**: The `Html_MGraph` entry point was critical for hiding complexity and ensuring compatibility.
- **Shared Node_IDs**: The elegant solution for cross-graph references without tight coupling.
- **Comprehensive Testing**: The high test count gave us the confidence to refactor aggressively.
- **Clear Briefing Document**: A detailed spec enabled consistent and focused implementation.

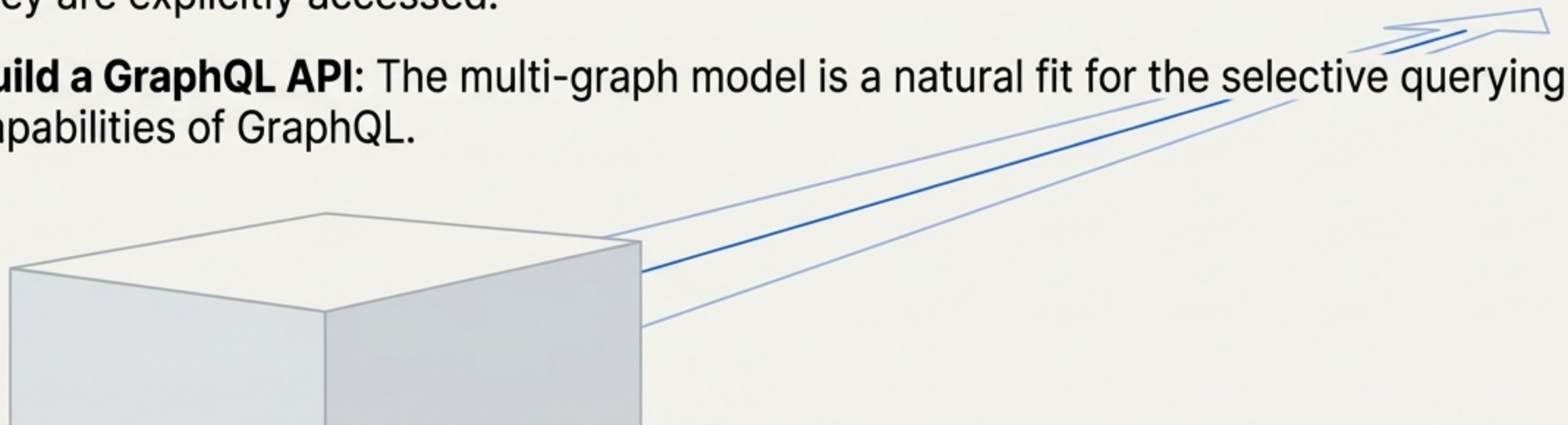
Areas for Improvement 🛠️

- **Earlier Test Refactoring**: Some test updates were delayed, causing friction late late in the project.
- **Performance Benchmarking**: We focused on functional correctness; performance tests should be added.

Building on a Strong Foundation: The Road Ahead

The new multi-graph architecture is not just a refactoring; it's an enabler. Its clean separation of concerns and robust design open up several opportunities for future development.

1. **Add Performance Benchmarks**: Quantify the performance of the multi-graph system versus the old monolith.
2. **Implement Streaming Support**: Process very large HTML documents without holding them all in memory.
3. **Explore Lazy Loading**: Only populate specific sub-graphs (like Scripts or Styles) when they are explicitly accessed.
4. **Build a GraphQL API**: The multi-graph model is a natural fit for the selective querying capabilities of GraphQL.



A New Standard for a Core System

The multi-graph refactoring has delivered a more robust, maintainable, and powerful system for HTML representation.

- **Separated concerns** into five purpose-specific graphs.
- **Simplified the data pipeline** with a clean, 3-phase, graph-based process.
- **Enabled powerful new use cases** through focused, uncluttered graph views.
- **Maintained 100% backward compatibility** with all existing APIs via a facade.
- **Achieved 100% test coverage** with 978 passing tests, ensuring future stability.

This robust, well-tested system was designed, implemented, and deployed to production on AWS Lambda in **approximately 48 hours** of focused development.

Implementation Team:
Claude + DinisCruz

Document generated: December 22, 2025