A SIMPLE LINE OF HTML CAUSED A SYSTEM CRASH

```
<input type="text" name="username" required>
```

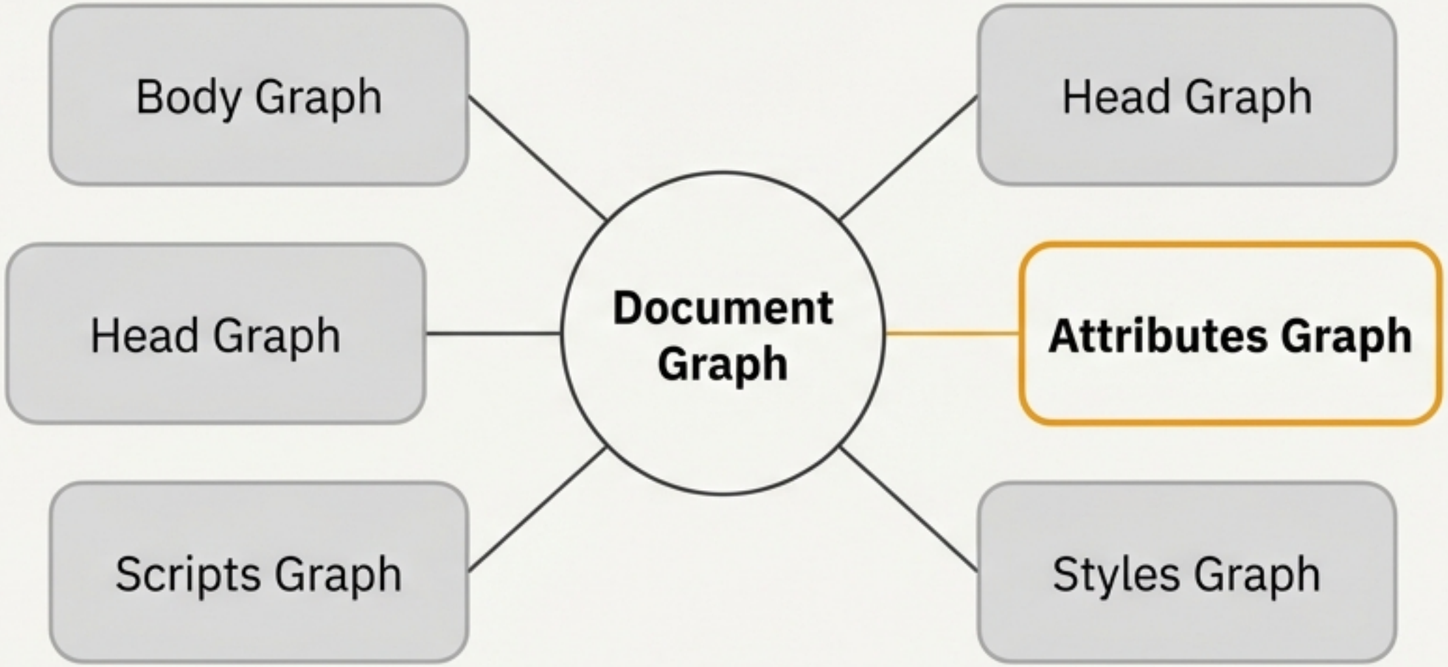
When processing HTML containing boolean attributes (attributes without a value), the `Html_MGraph` system would consistently fail.

The bug was discovered when a user attempted to load a standard HTML form.

This single attribute, `required` , exposed a fundamental flaw in how we model and store HTML attributes.

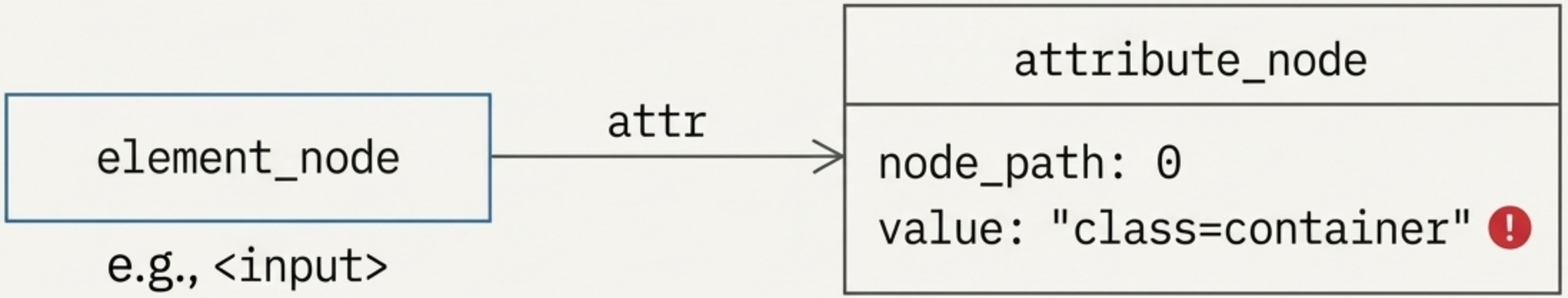
The initial investigation pointed to the `Html-Dict -> Html_MGraph` conversion layer as the point of failure.

UNDERSTANDING THE `Html_MGraph` ECOSYSTEM



The **`Html_MGraph`** system represents a complete HTML document as a collection of interconnected, specialised graphs. Each graph manages a specific aspect of the document, from the DOM tree to styles and attributes. This presentation focuses on a critical refactor within the **"Attributes Graph"**, which stores the relationship between HTML tags and their attributes.

THE ORIGINAL MODEL: ONE NODE FOR NAME, VALUE, AND POSITION



> In the original implementation, a single node was responsible for representing three distinct concepts: the attribute's name, its value, and its positional order. **This conflation** was the source of the system's fragility.

THE INVESTIGATION: RULING OUT THE SIMPLE FIXES

ATTEMPT 1

Tactic: Make `attr_value` optional and use an empty string.

Problem: The attribute name is lost. The graph shows a node with value `"0"` instead of representing `required`.

ATTEMPT 2

Tactic: Store `attr_name` in `node_path` for boolean attributes.

Problem: This breaks roundtrip processing. `get_attributes()` expects `node_path` to be an integer (position), not a string (name).

`get_attributes()` ! TypeError

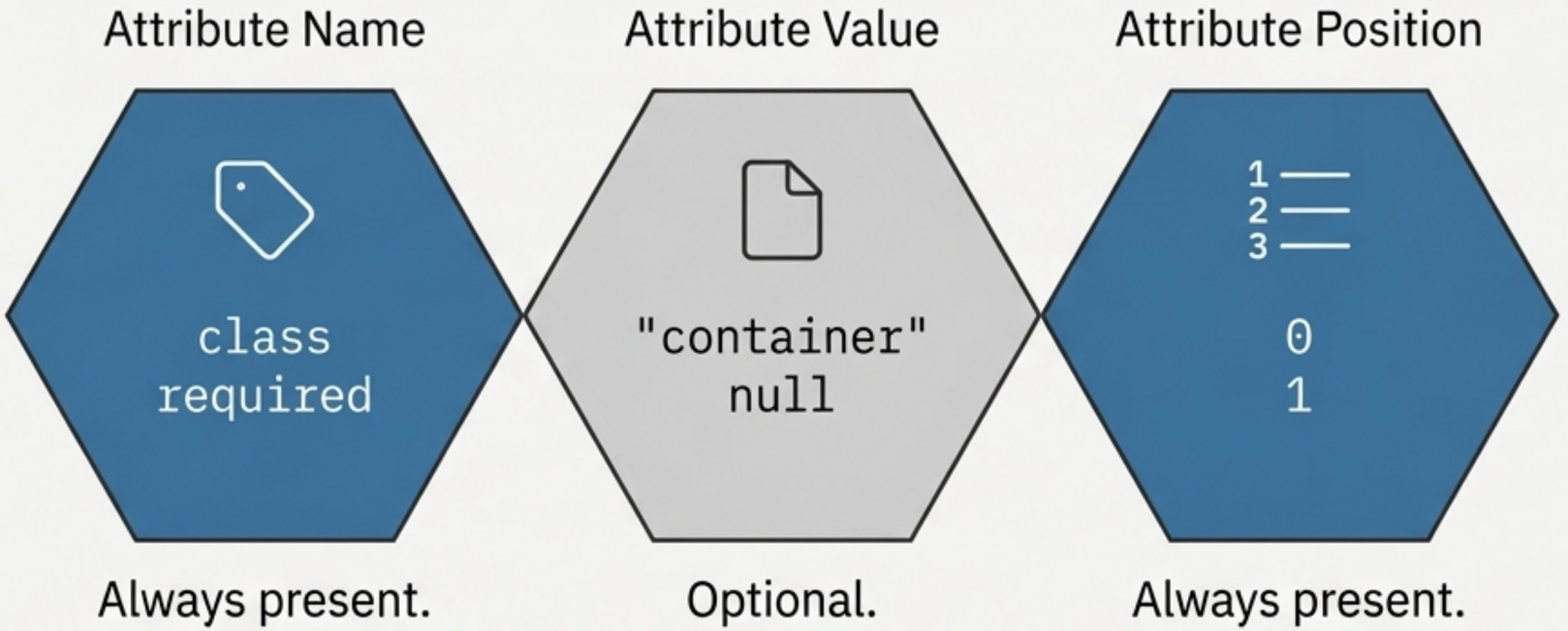
ATTEMPT 3

Tactic: Always use position, but accept `""` as a value.

Problem: This loses the semantics of a boolean attribute. The roundtrip produces incorrect HTML.

```
<input required="">
```

THE ROOT CAUSE: A FLAWED ASSUMPTION

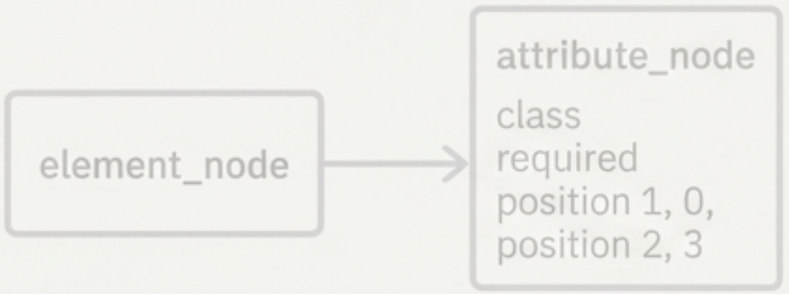


The single-node model fundamentally cannot represent these three distinct and independent concepts cleanly.

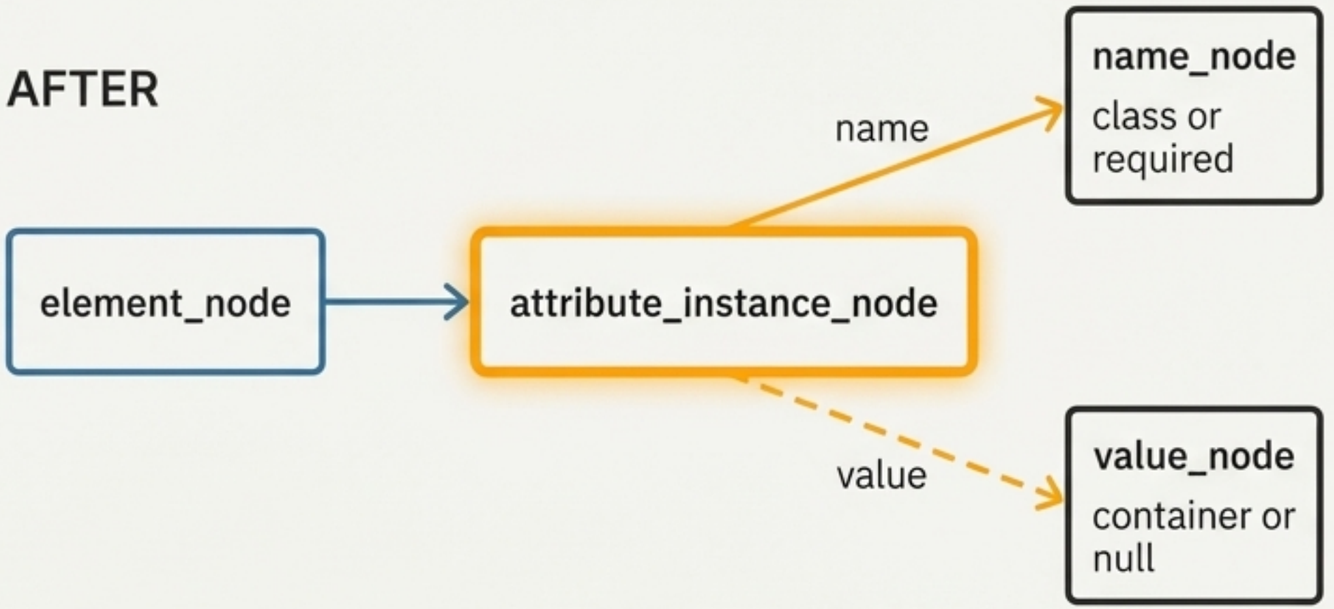
Furthermore, the old model prevented any form of data reuse. Every single attribute instance, even for common names like `class` or `id`, created entirely new nodes, leading to graph bloat.

THE SOLUTION: SEPARATING CONCERNS WITH AN INSTANCE NODE

BEFORE



AFTER

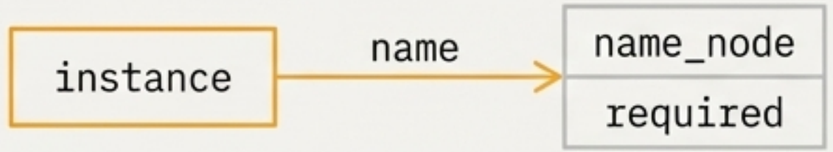


The new model introduces an **attribute instance node**. This node acts as an intermediary, cleanly separating the attribute's identity (its name and value) from its specific use (its position on an element).

KEY DESIGN PRINCIPLES OF THE NEW MODEL

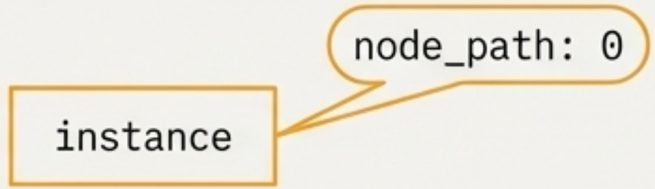
1. Native Boolean Support

A boolean attribute is represented by an `attribute_instance` with a `name` edge but **no** `value` edge. The presence or absence of the value edge provides clean, unambiguous semantics.



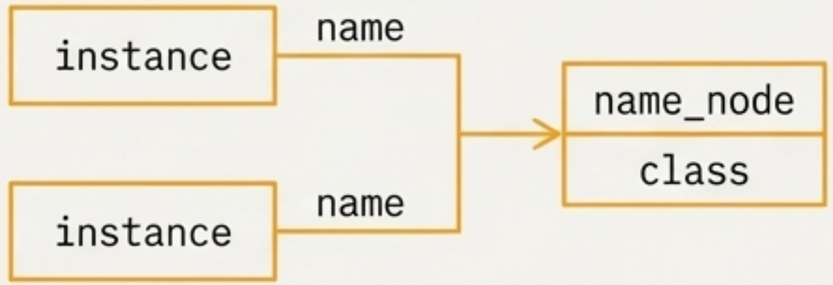
2. Position is Preserved

The attribute's order is stored on the `attribute_instance_node.node_path`, maintaining its original responsibility.



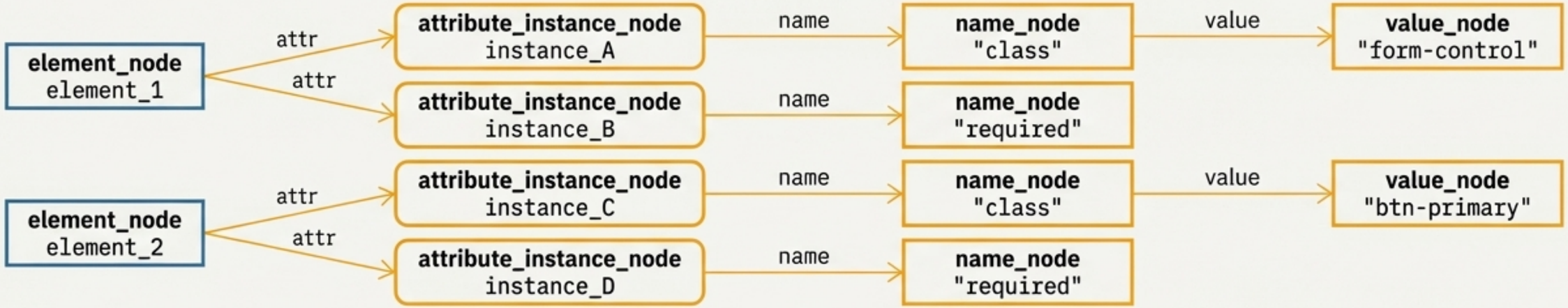
3. Radical Deduplication

Name and Value nodes are now reusable. All `class` attributes point to a single `name_node("class")`. All `"container"` values point to a single `value_node("container")`.



THE NEW MODEL IN ACTION: A CONCRETE EXAMPLE

```
<input class="form-control" required>
<input class="btn-primary" required>
```



Node Table

Node ID	Type	node_path	value
element_1	element	body.input[0]	-
element_2	element	body.input[1]	-
instance_A	element	0	-
instance_B	element	1	-
instance_C	element	0	-
instance_D	element	1	-
name_class	value	name	"class"
name_required	value	name	"required"
val_form_control	value	value	"form-control"
val_btn_primary	value	value	"btn-primary"

Edge Table

From	To	Predicate	edge_path
element_1	instance_A	attr	-
element_1	instance_B	attr	-
element_2	instance_C	attr	-
element_2	instance_D	attr	-
instance_A	name_class	name	-
instance_A	val_form_control	value	-
instance_B	name_required	name	-
instance_C	name_class	name	-
instance_C	val_btn_primary	value	-
instance_D	name_required	name	-

A NECESSARY EVOLUTION: IMPACT AND MIGRATION

Breaking Changes

1. **Incompatible Graph Structure:** Existing serialized graphs are incompatible with the new model.

2. **Updated API Contract:** `get_attributes()` now returns `Dict[str, Optional[str]]` to support valueless attributes.

3. **Increased Node Count:** Each attribute now creates 2-3 nodes, though this is offset by significant node reuse.

Migration Strategy

A clean break was the most effective path forward for this system.

- Html_MGraph is ephemeral and regenerated from source HTML on each use.
- There is no persistent storage of graph structures at this time.
- All downstream consumers were updated simultaneously as part of the change.

This change warrants a minor version bump to v0.2.0 to signal the breaking change to all users.

THE VERDICT: A QUANTIFIABLY SUPERIOR SYSTEM

Aspect	Before	After
Boolean Attribute Support	✗ Crashes	✓ Native Support
Roundtrip Fidelity	✗ Produces `required=""`	✓ Preserves `required`
Name Deduplication	✗ None	✓ Full Reuse
Value Deduplication	✗ None	✓ Full Reuse
Graph Semantics	Conflated & Fragile	Clean Separation of Concerns
Nodes per Attribute	1	2-3 (with reuse)

The refactor successfully eliminated a critical bug while creating a more robust, efficient, and semantically correct data model for all HTML attributes.

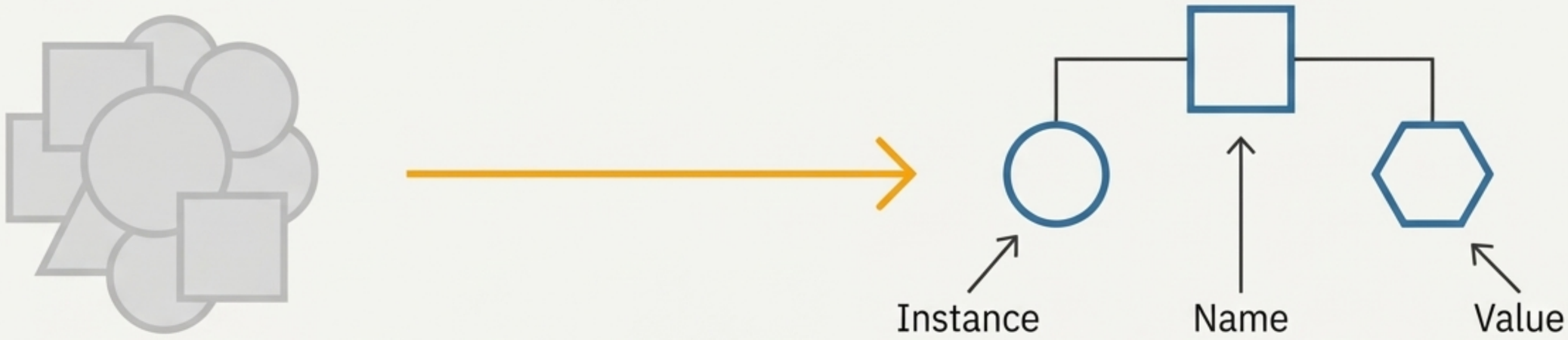
THIS BUG WAS A FEATURE: THE POWER OF VISUALISATION

The initial discovery of this bug was not accidental; it was a direct result of using our own graph visualisation tools.

Being able to see the graph structure made the flaw in the `Html-Dict -> Html_MGraph` conversion immediately obvious in a way that reading logs or code could not.

This process validates our investment in tooling: good visualisation is not a luxury, but a critical component of debugging and designing complex systems.

A LESSON IN 'THINKING IN GRAPHS'



Monolithic Node

Decomposed, Reusable Nodes

Key Lessons Learned

- Model the relationships, not just the data.
- Separate identity from instance.
- Use the visual representation to challenge your assumptions and discover edge cases.

This refactor wasn’t just about fixing a bug. It was about adopting a more precise and powerful data model that will serve the system’s long-term health and scalability.