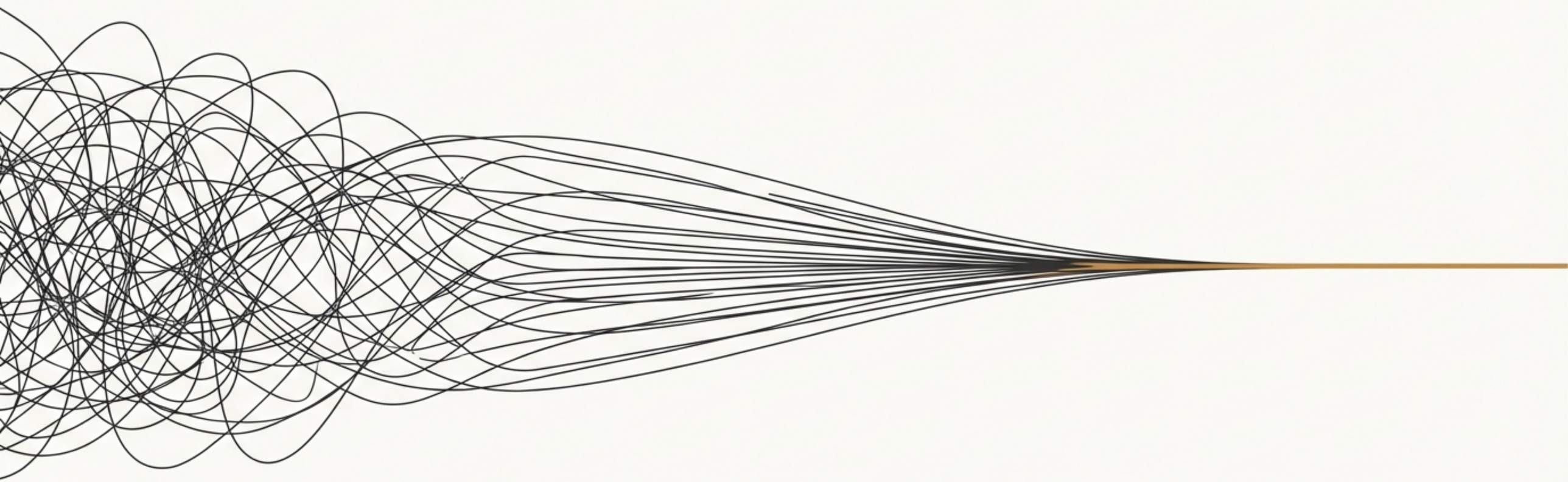
From Messy Markup to Ordered Text

A Deep Dive into the HTML Graph Node Flattening Transformation



The Challenge: Extracting Pure Content from Complex HTML

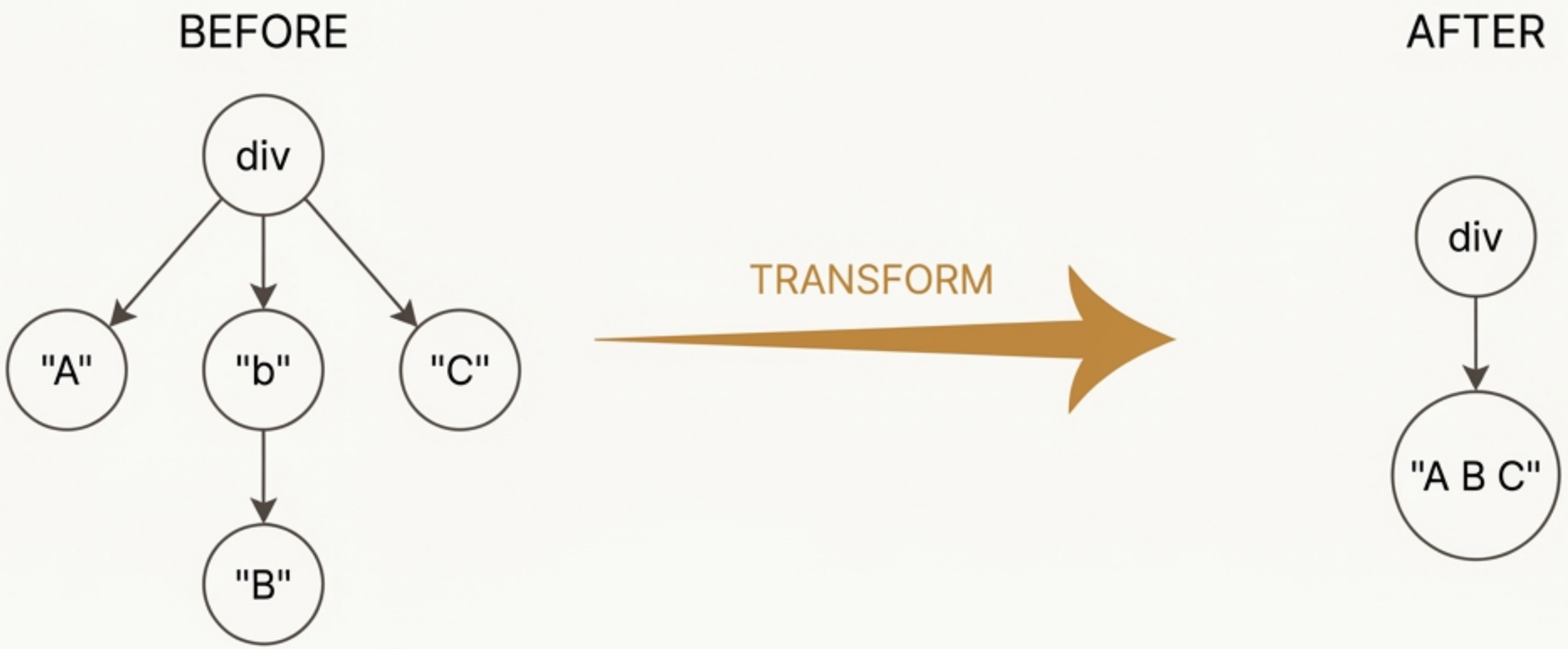
HTML is designed for rendering, not for semantic analysis. Its nested structure of formatting and content tags creates significant noise when the goal is to extract a simple, readable sequence of text. The implicit order of content is easily lost.

```
<div>
  <p>
     Paragraph 1, with <b>bold text</b>
     and <i>some italics</i>.
  </p>
</div>
```

Paragraph 1, with **bold** text and some *italics*.

Our Solution: A Graph-Based Pipeline that Preserves Order

We transform the HTML Document Object Model (DOM) into a property graph. By applying a series of targeted transformations, we can collapse formatting elements while rigorously preserving the original content order.



Why This Transformation is Invaluable

This technique is a foundational component for a wide range of data processing tasks.

Text Extraction

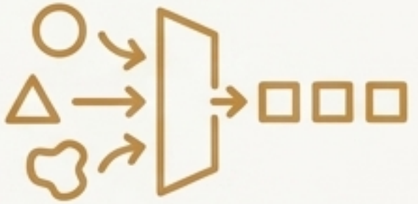
Strip formatting tags to produce clean, readable content for display or downstream processing.

Content Indexing

Generate clean, searchable text from web pages and documents for search engines.

Semantic Analysis

Reduce HTML to its essential text structure to feed natural language processing models.



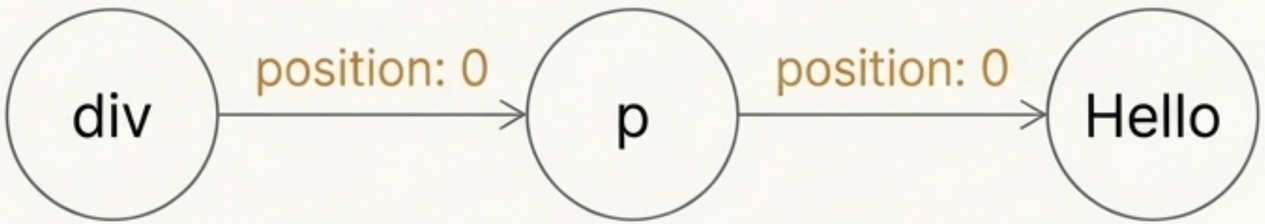
Data Normalisation

Standardise varied and inconsistent HTML structures into a uniform text representation.

The Mechanism: How We Represent and Transform HTML

The process begins by parsing HTML into a property graph. Every element and piece of text becomes a node. The relationships between them become edges.

```
<div>
  <p>Hello</p>
</div>
```



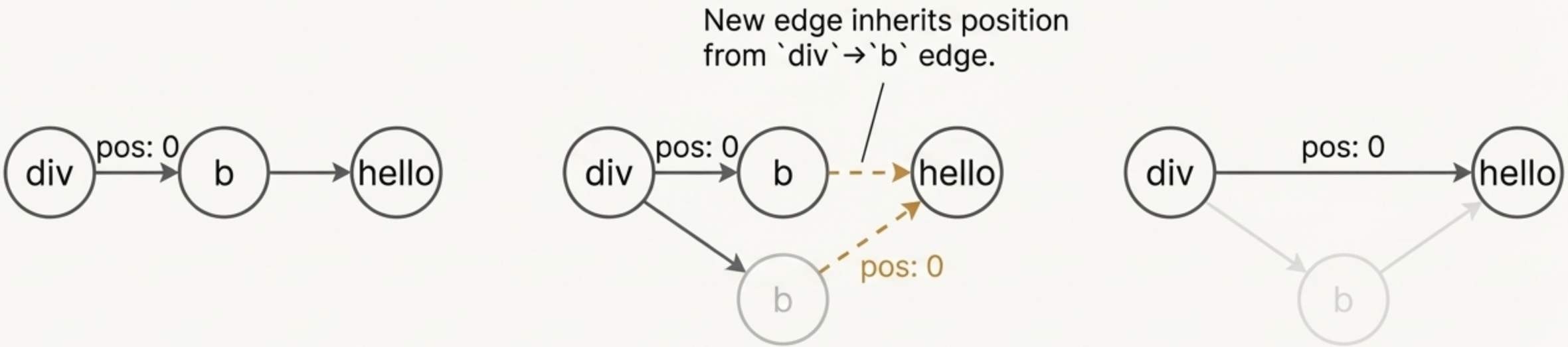
Edge Position is Crucial.

To preserve the original reading order, we number the edges based on a child's position within its parent. This metadata is the key to reassembling the text correctly after transformation.

Example 1: Collapsing a Simple Wrapper

```
Initial HTML: <div><b>hello</b></div>
```

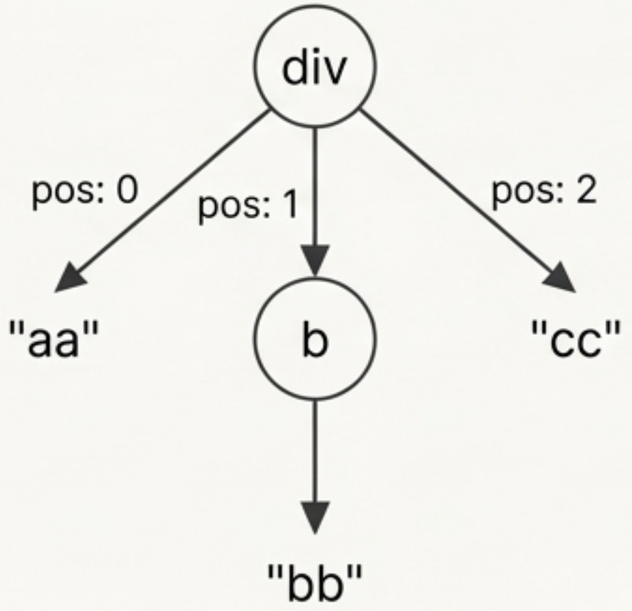
Step 1: Parse to Graph ⟶ Step 2: Add Shortcut Edge ⟶ Step 3: Remove Wrapper



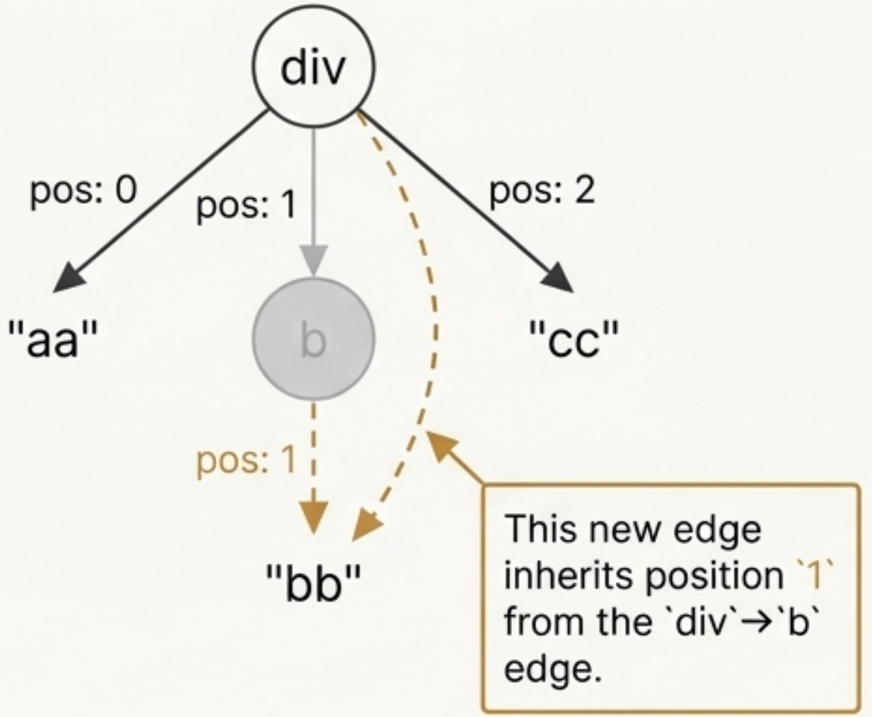
Example 2: Handling Mixed Text and Formatting

```
Initial HTML: `div>aa<b>bb</b>cc</div>`
```

1. Initial Graph



2. Add Shortcut & Remove Wrapper



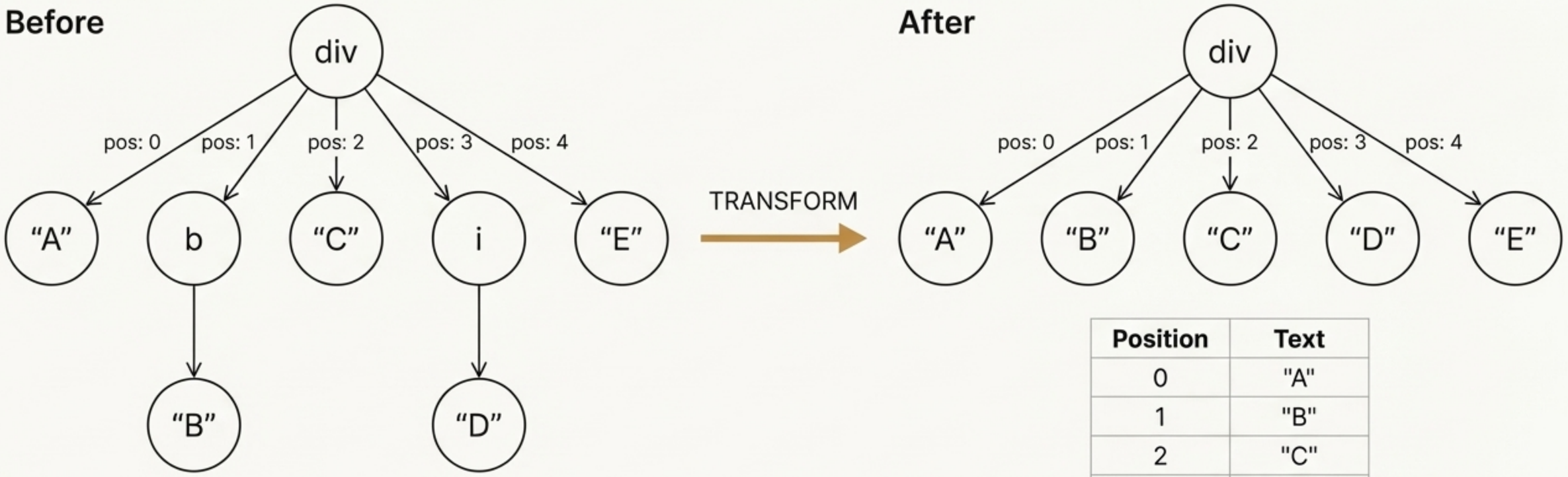
3. Merge by Position

Position	Text
0	"aa"
1	"bb"
2	"cc"

Result: "aa bb cc"

Example 3: The Pattern Scales to Multiple Tags

```
Initial HTML: <div>A<b>B</b>C<i>D</i>E</div>
```

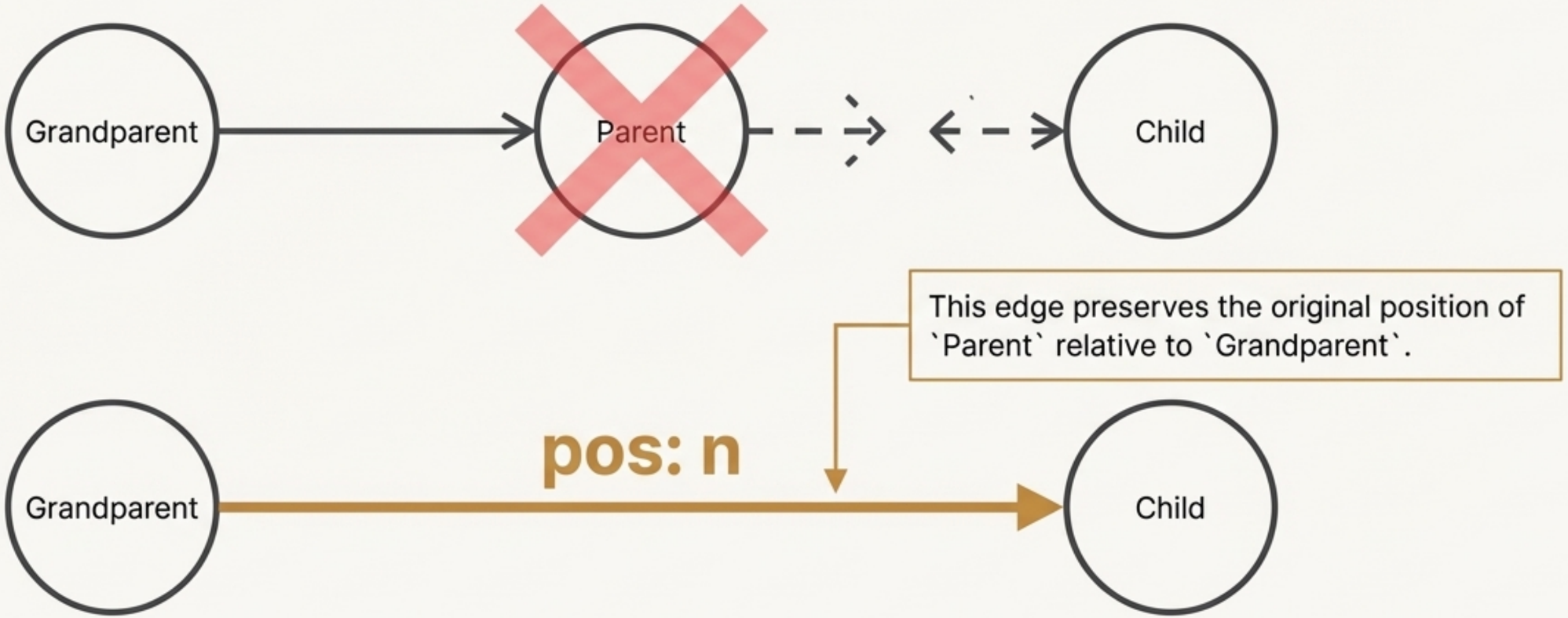


Position	Text
0	"A"
1	"B"
2	"C"
3	"D"
4	"E"

Result: "A B C D E"

Design Principle I: Edge Position Makes Implicit Order Explicit

HTML's content order is implicit in the DOM tree. When we flatten the tree by **removing** nodes, that structural ordering is lost. By encoding the position in edge metadata, we create a durable record that allows us to **reconstruct the correct reading order at any stage**.



Design Principle II: Collapse Formatting, Preserve Structure

The transformation's goal is to remove formatting noise, not to destroy the document's semantic structure. Our rule is simple but effective: elements with multiple children are treated as structural containers, while elements with a single child are considered formatting wrappers.

Action: COLLAPSE (Formatting Wrappers)

- <b> with a single text node
- <p> with just a single child

Action: KEEP (Structural Containers)

- with multiple <li> items
- with mixed text and element children

The Four Core Rules of the Transformation

1 **Collapse single-child parents only.**
Why? To distinguish formatting from structure.

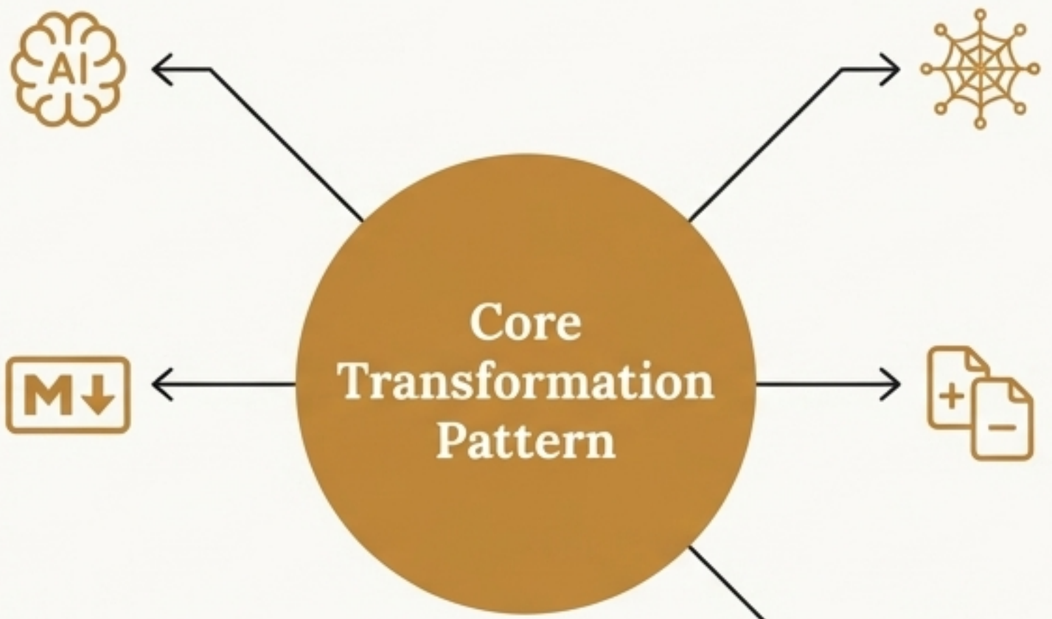
2 **Inherit edge position for shortcuts.**
Why? To preserve the original reading order.

3 **Sort by position before merging text.**
Why? To guarantee the correct final text sequence.

4 **Delete original nodes only after merging.**
Why? To prevent data loss during the transformation.

A Versatile Pattern for Modern Applications

This robust text extraction pattern is a key enabler for numerous advanced applications.



LLM Preprocessing: Simplify complex HTML to clean text for language model input.

Markdown Extraction: A potential extension to preserve heading structure while flattening inline formatting.

Web Scraping: Reliably extract article text without boilerplate formatting.

Content Comparison / Diffing: Compare document versions based on semantic content, not just markup changes.

Accessibility: Generate clean, plain-text versions of rich HTML for screen readers.

From Markup to Meaning: Three Key Takeaways

1

Graphs Tame HTML Complexity.

Representing HTML as a property graph provides a powerful and flexible foundation for complex document transformations.

2

Edge Position is the Key to Order.

Storing positional metadata on edges, not nodes, is the critical design choice that guarantees order is preserved when the structure is flattened.

3

A Foundational Pattern for Content Intelligence.

This transformation is more than a utility; it's a core pattern for building systems that can index, analyse, and understand content at scale.