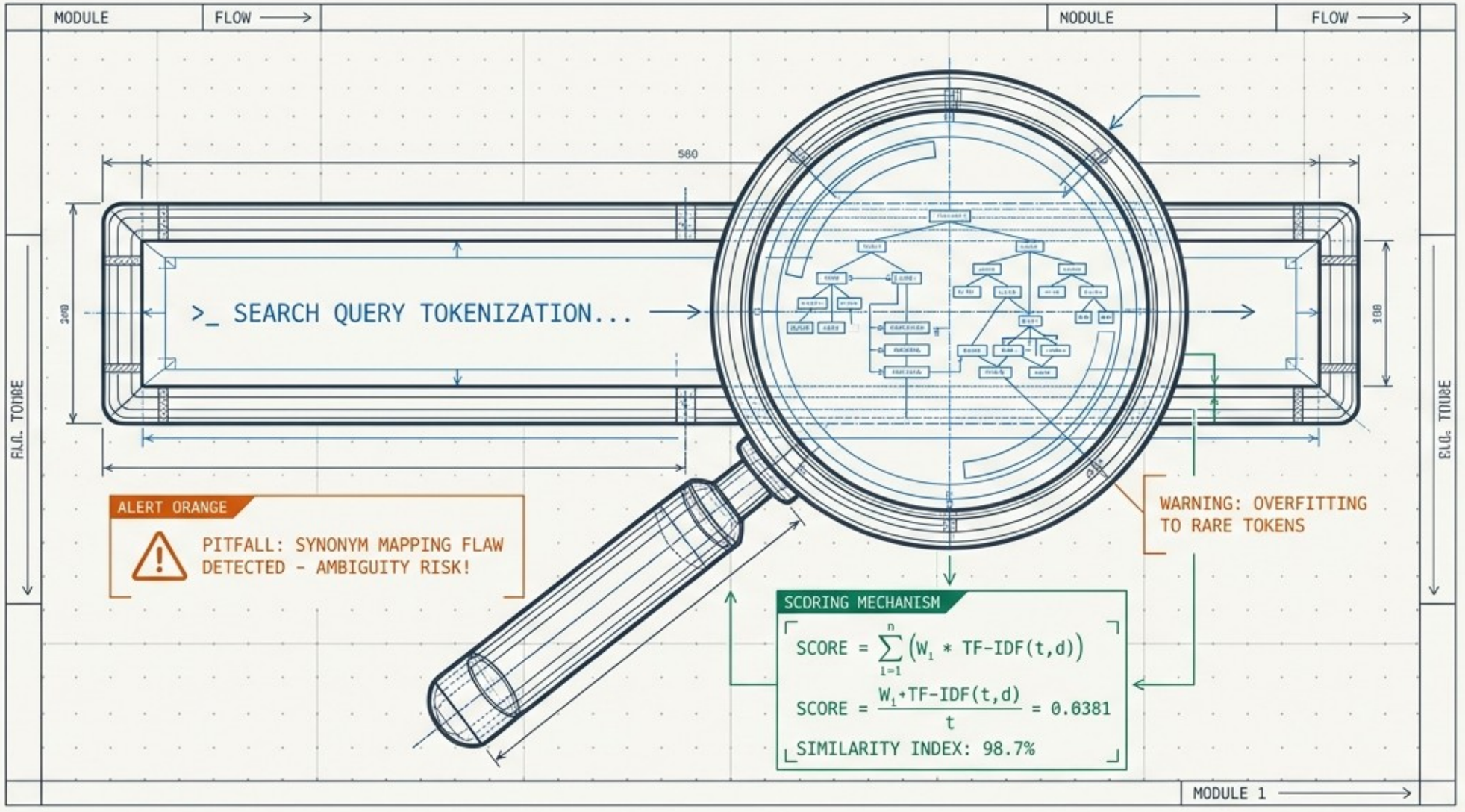
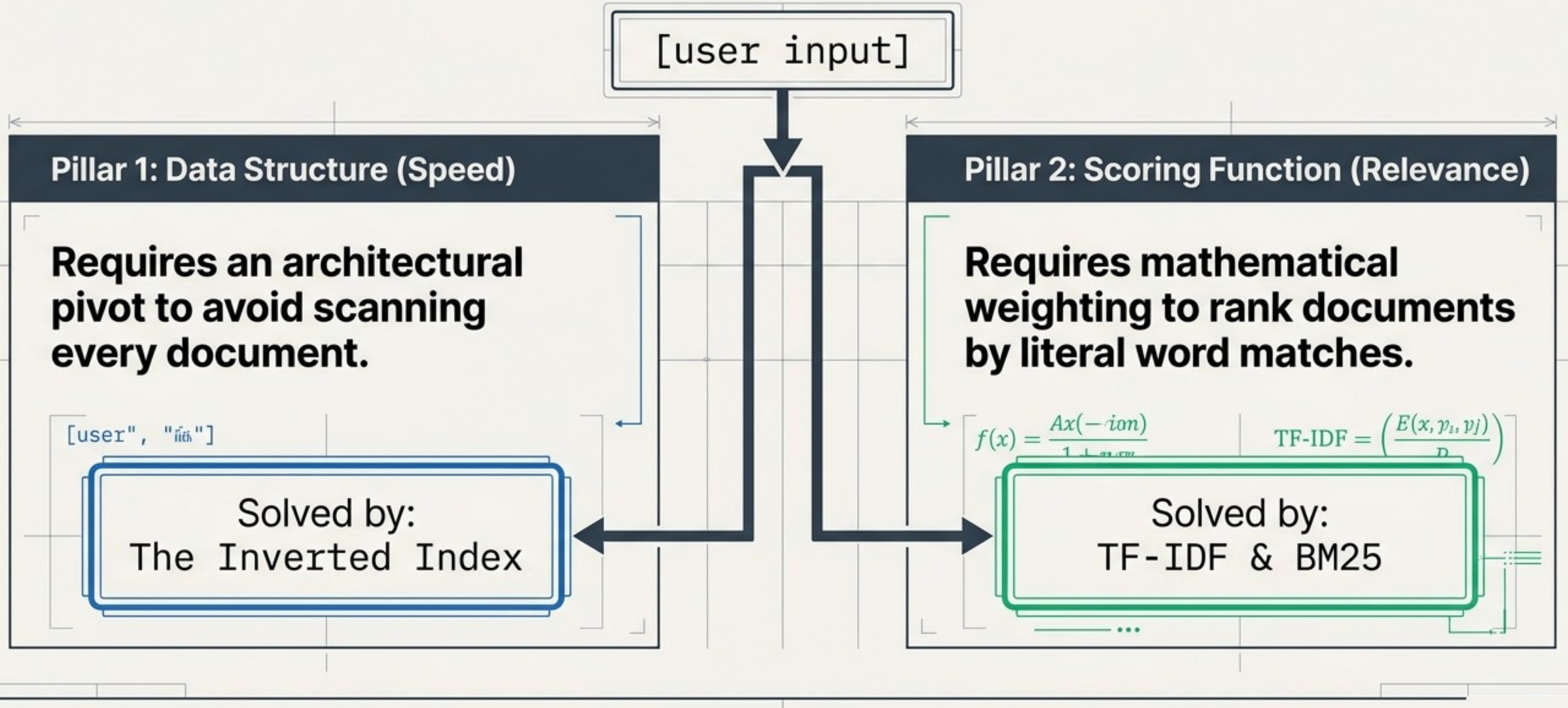



# Keyword Search Mechanics

A schematic view of lexical matching, indexing, and scoring.

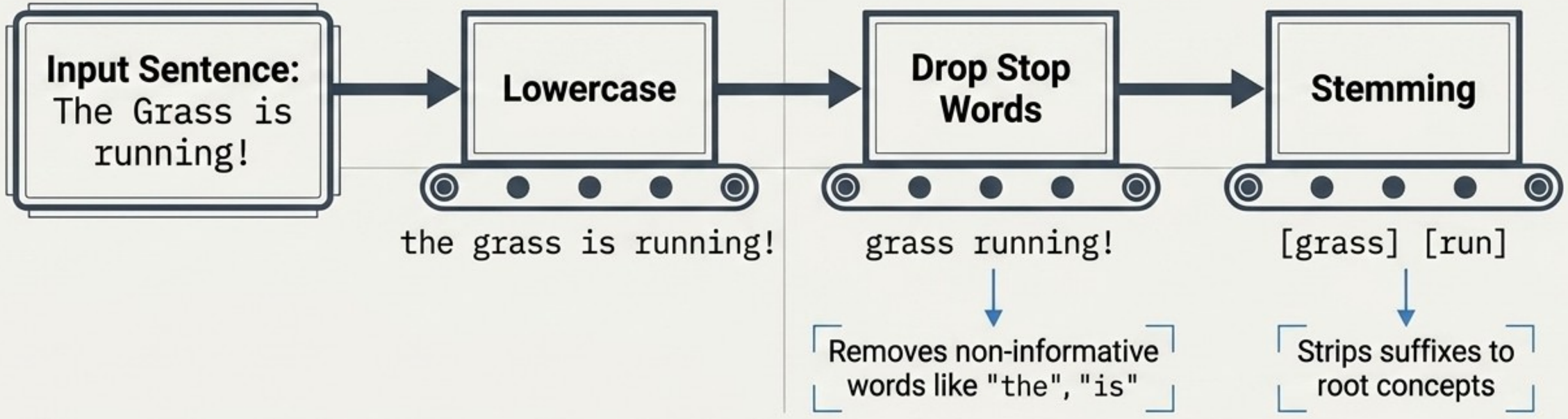
# The Millisecond Mandate

A search engine must return the best matches from millions of documents in milliseconds. Keyword search solves this with two mechanisms:



# Step 1: Tokenization Pipeline

Before any matching occurs, raw text is normalized into uniform tokens.




**Crucial Rule:** The exact same pipeline must be applied to both the query and the documents. If tokenization mismatches, literal matches silently vanish.

# Step 2: The Inverted Index Paradigm Shift

Search is instant because the database structure is inverted ahead of time.

## Forward Index (Slow Scan)

Maps Document -> Words

```
D0: [the, grass, is, green]
```

Querying requires scanning millions of documents sequentially.

## Inverted Index (Instant Lookup)

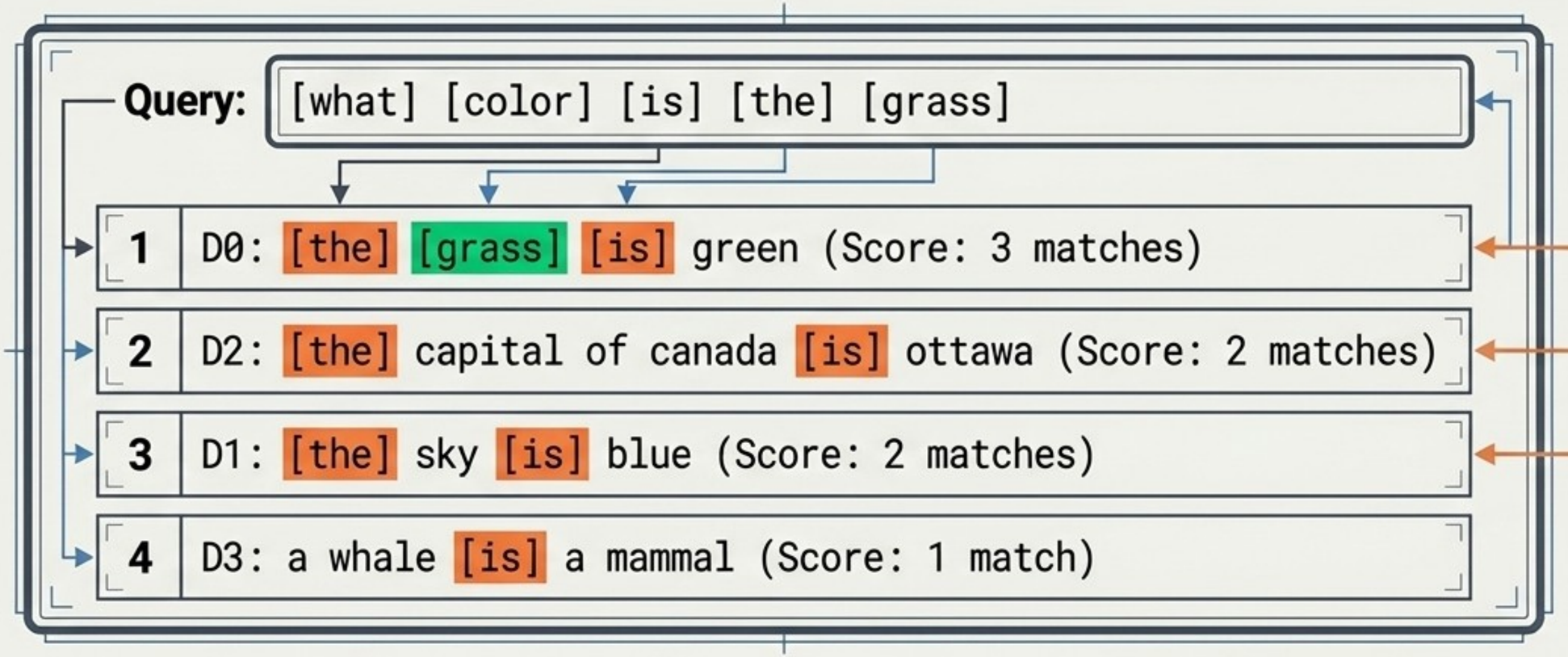
Maps Word -> Document Postings

```
[grass] -> {D0}
[blue]  -> {D1, D4}
[whale] -> {D4}
```

Querying requires a single dictionary lookup. The engine jumps directly to the exact documents containing the term.

# Step 3: The Naive Count (And Its Fatal Flaw)

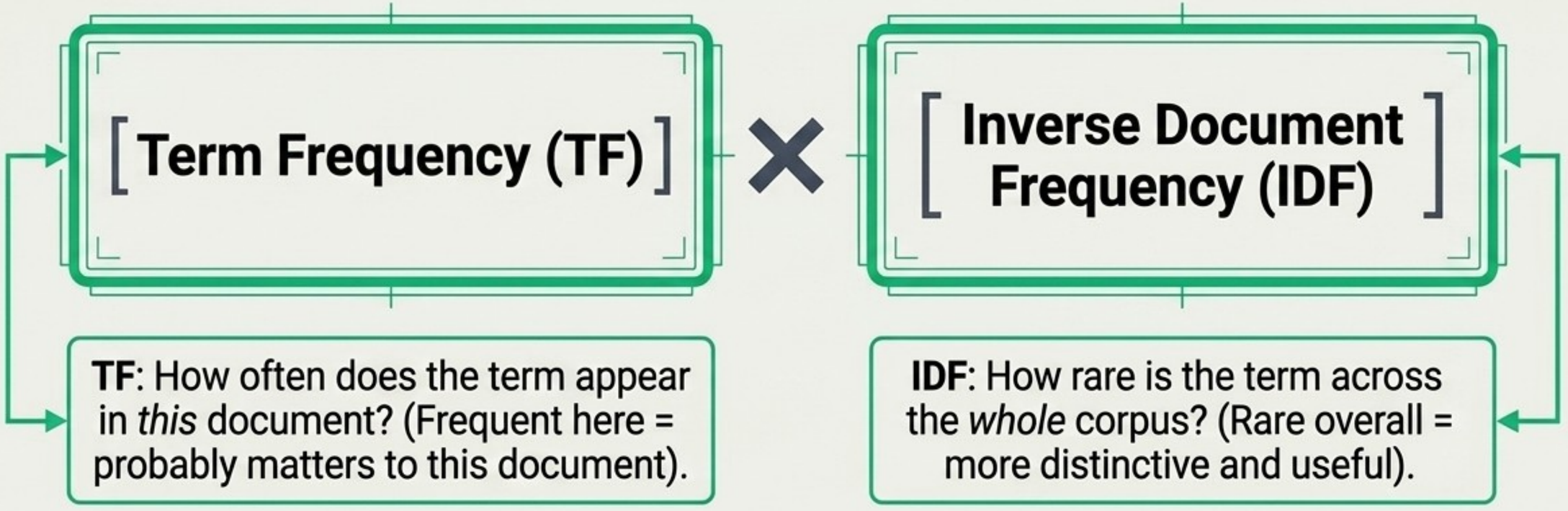
Intuition suggests we simply count the shared words.




**The Problem:** D0 wins, but for the wrong reason. Ultra-common words like 'the' and 'is' dominate the count while carrying zero meaning.

# Evolution 1: TF-IDF

We must mathematically weight words by how informative they are.




**The Formula:** `IDF(t) = ln(N / df(t))`

(Where N = total documents, and `df` = documents containing the term).

# TF-IDF in Action

Calculating the query against a 5-document corpus.

| | Term | Occurrences | IDF Calculation |
|---|---|---|---|
| 2 | [is] | Appears in 5 docs | IDF = ln(5/5) = 0.00 |
| 3 | [the] | Appears in 3 docs | IDF = ln(5/3) ≈ 0.51 |
| 4 | [grass] | Appears in 1 doc | IDF = ln(5/1) ≈ 1.61 |

Scoring D0 (‘the grass is green’):

$$\text{score}(D0) \approx (1 \times 1.61) + (1 \times 0.51) + (1 \times 0.00) = 2.12$$

**The Result:** The word ‘grass’ drives the entire score. Because ‘is’ appears in every document, its IDF drops to absolute zero. Meaning drives the ranking.

# Evolution 2: BM25 (The Workhorse)

TF-IDF has two vulnerabilities. BM25 applies common-sense fixes, making it the default first-stage ranking function for decades.

## Fix 1: Term Saturation (Parameter $k_1$)

**TF-IDF Bug:** Scores grow infinitely if a word is repeated.

**BM25 Fix:** Diminishing returns. Seeing a word 10 times is not 10x more relevant than seeing it once. The score flattens out.

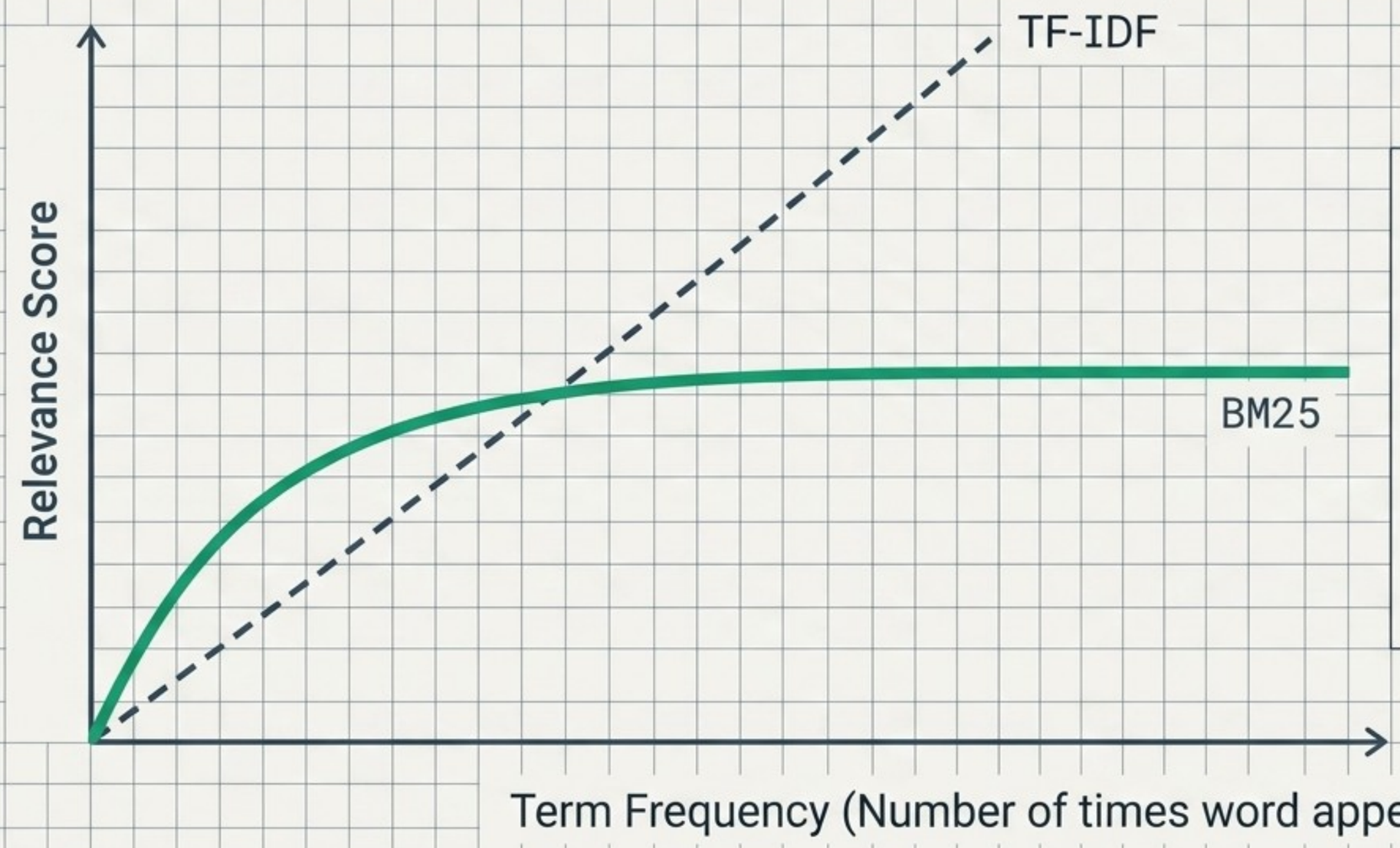
## Fix 2: Length Normalization (Parameter $b$)

**TF-IDF Bug:** Long documents naturally repeat words and win unfairly.

**BM25 Fix:** Compares a document to the average document length in the corpus, penalizing overly long texts.

# Visualizing Term Saturation

Why **keyword stuffing** fails against BM25.

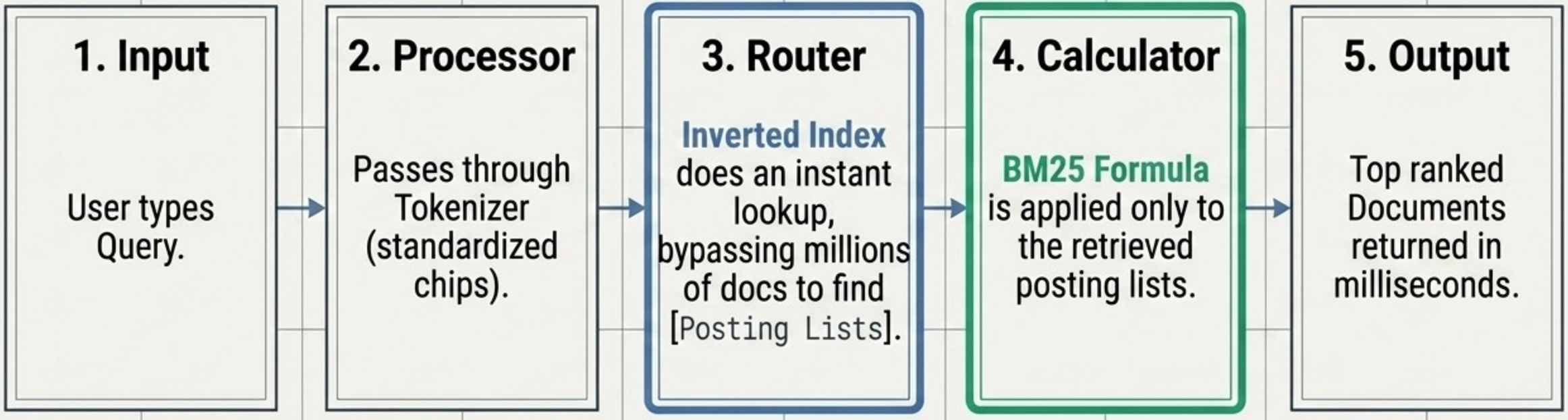



**As a document repeats the exact same term, plain `TF-IDF` scales relevance infinitely.**
BM25 **applies an** `asymptotic curve`.

**Once a term appears a few times, its marginal contribution to the score approaches zero.**

# The Lexical Search Engine

The **complete architecture**. An exceptionally fast, robust baseline that requires no model training.



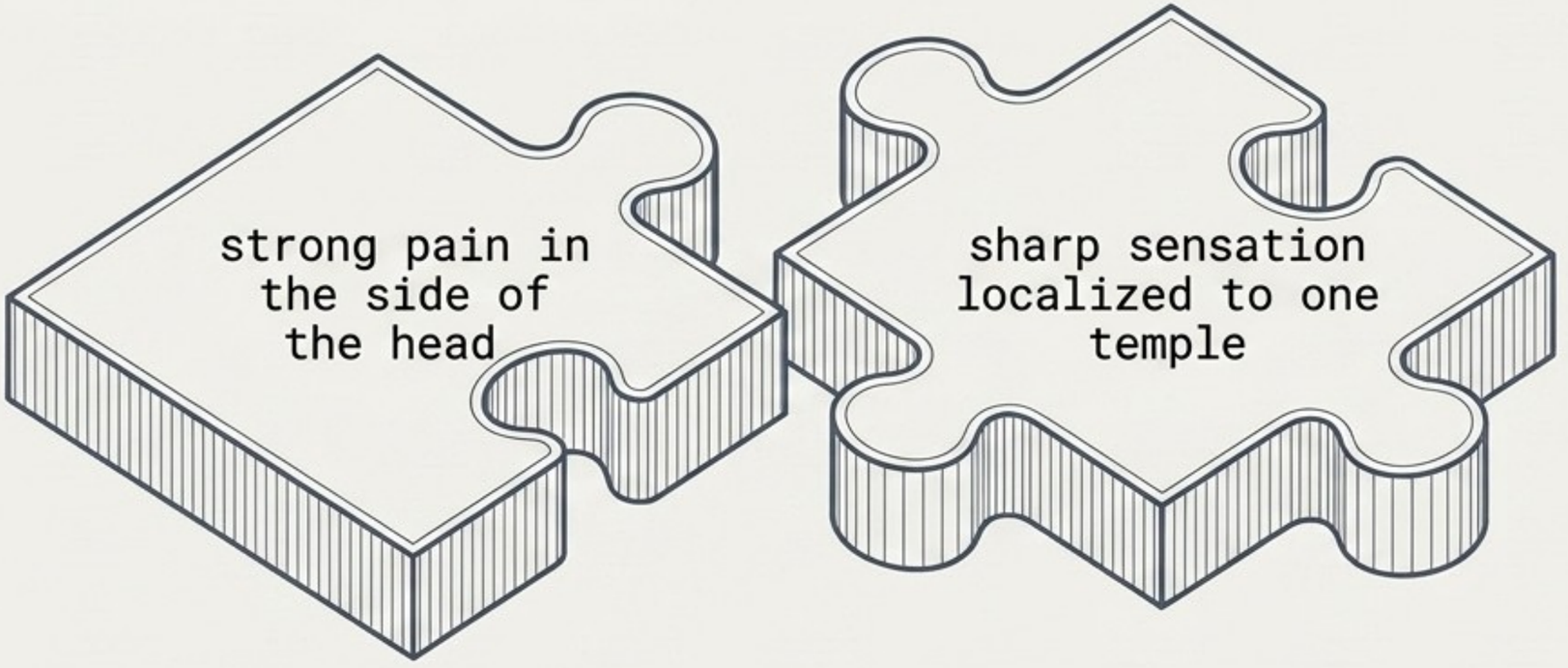
# System Diagnostic Matrix

Common pitfalls in lexical search implementations.

| Symptom | Cause | Rule Violated |
| --- | --- | --- |
| Exact word matches disappear entirely. | Mismatched tokenization. | Query and Index pipelines must be perfectly mirrored. |
| Queries with “not” or “no” return opposite meanings. | Over-aggressive stop-word removal. | Contextual modifiers must survive tokenization. |
| Long documents inexplicably dominate the top results. | Using raw counts without length normalization. | Always default to BM25 over basic TF. |

# The Fundamental Limitation

Lexical search matches words, not meaning.




If a relevant document uses synonyms or paraphrases, the literal tokens will not match. Even with the perfect inverted index and optimized BM25 parameters, keyword search will completely miss the document.

**This is the Vocabulary Mismatch Problem. To solve it, we must transition from literal tokens to mathematical meaning. (Next: From Text to Vectors)**