# Phase E Performance Analysis: HTML → MGraph Pipeline

## Bottleneck Identification & Optimisation Strategy

---

DATE: JAN 2026

STATUS: ANALYSIS COMPLETE

SOURCE: Perf__Phase_E__Conversion

# Three Critical Findings Define the Pipeline Performance

## The Bottleneck

**97%**

of processing time is consumed by 'Dict->MGraph' conversion. Optimisation elsewhere is ineffective.

## The Win

**2.3x**

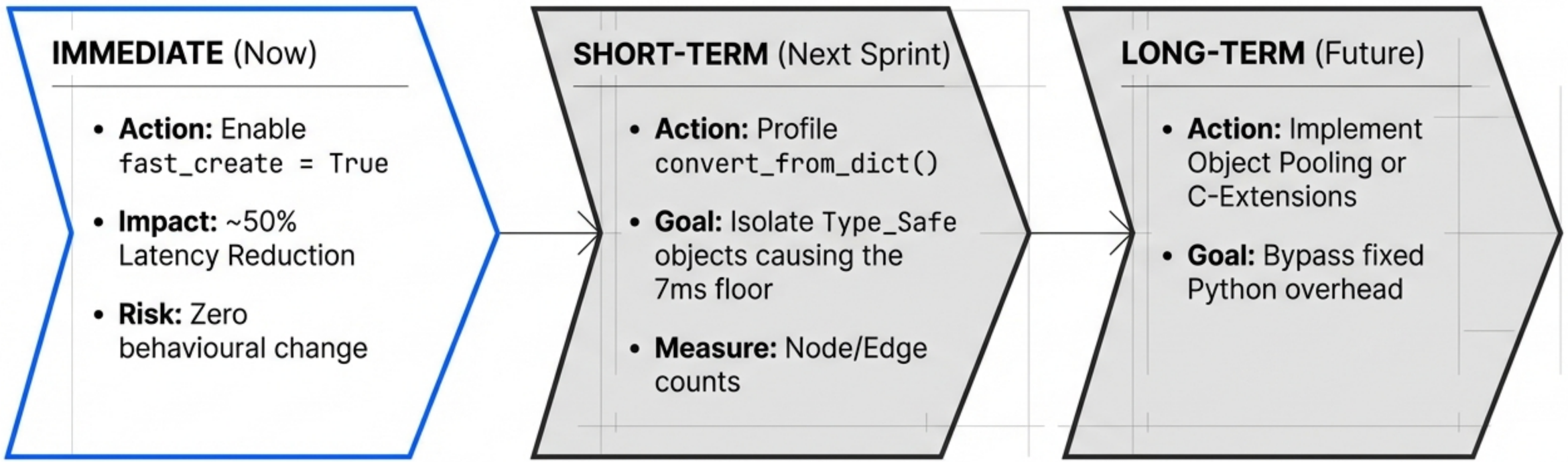
Speedup confirmed by enabling 'fast_create'. Immediate implementation recommended.

## The Mystery

**7ms**

Fixed architectural baseline exists regardless of input size. Suggests heavy object infrastructure.

# Strategic Roadmap: From Immediate Gains to Architectural Refactoring



# Methodology & Integrity

## Protocol

Metrics derived from `'measure_fast'` protocol using **87 Fibonacci** iterations to ensure stability and eliminate outliers.

## Environment

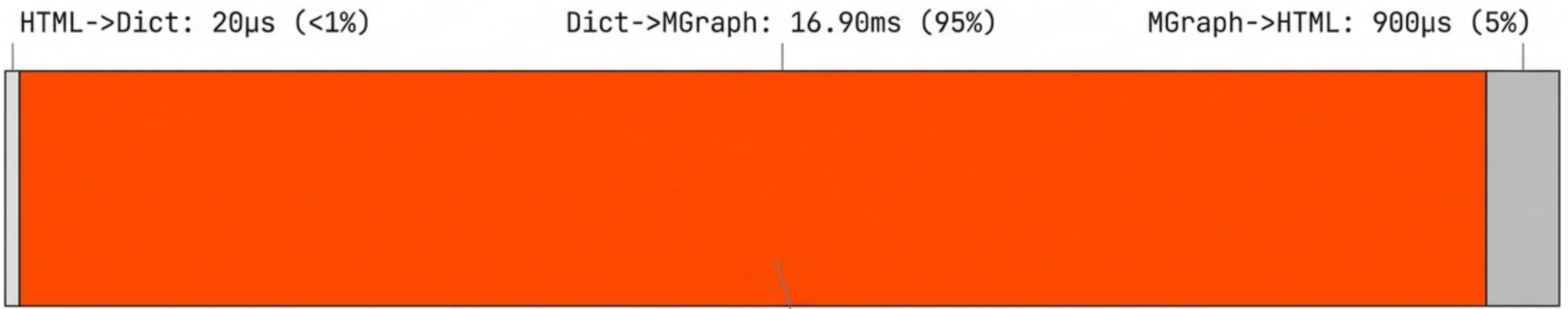
Test machine under normal load. Results consistent across multiple runs.

## Test Conditions


Generated HTML files compared across:

- Default 'Type_Safe' Mode
- **‘fast_create=True’** / **‘skip_validation=True’**

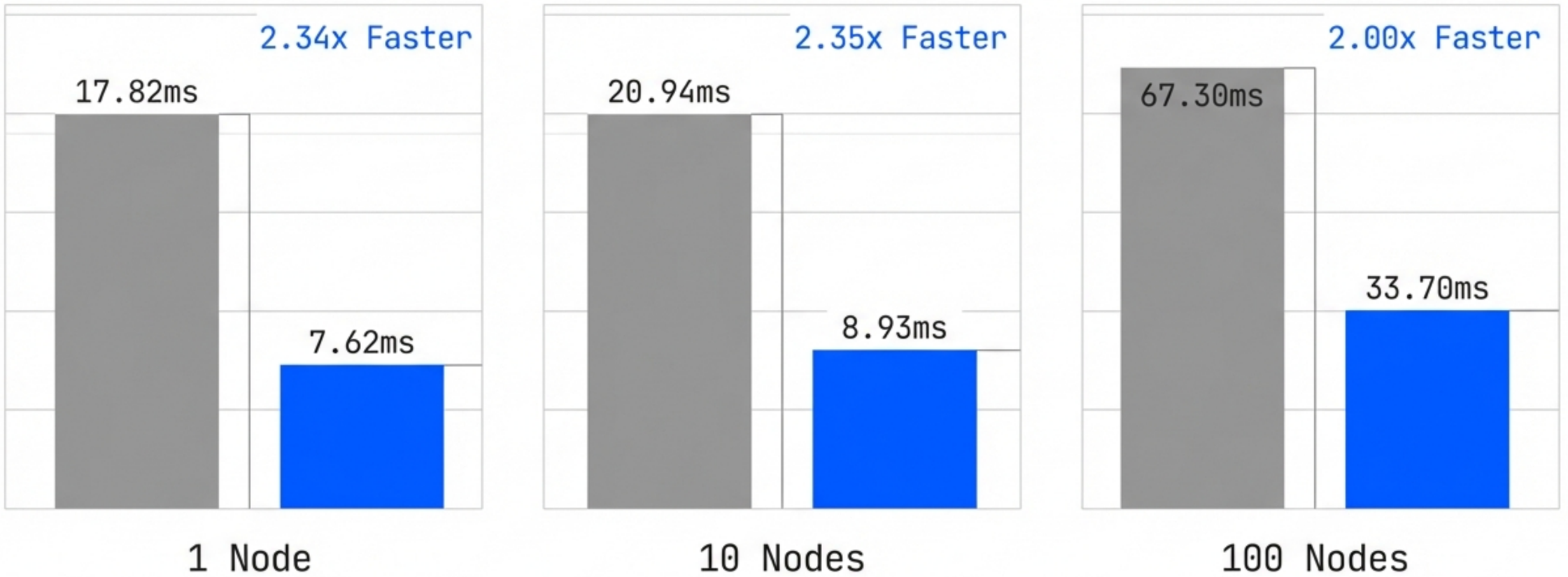
# The ‘Dict->MGraph’ Stage Dominates Execution Time




**CRITICAL FOCUS AREA**

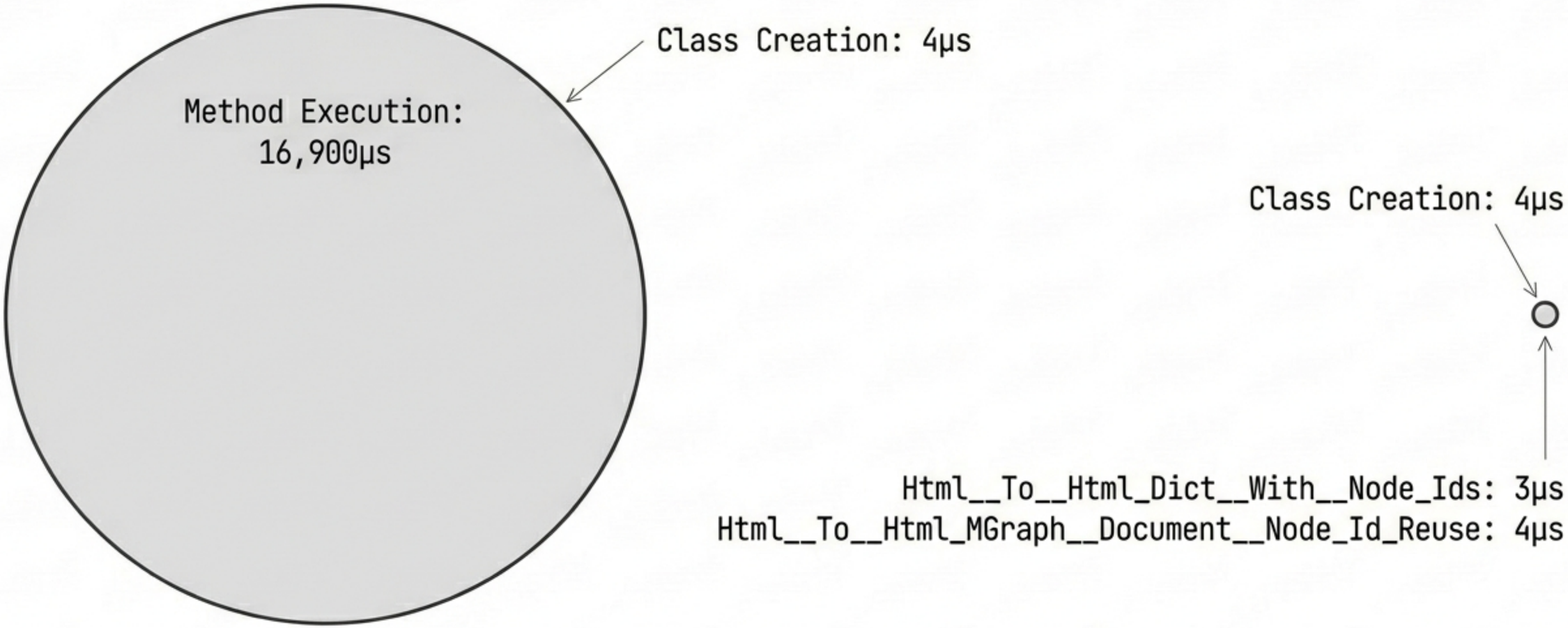
Optimization focus must be exclusive to the “convert_from_dict” method. This 16.90ms block dwarfs the input stage.

# 'fast_create' Delivers a Consistent 2x Speedup




The speedup is consistent (~50-57% reduction) across all scales.

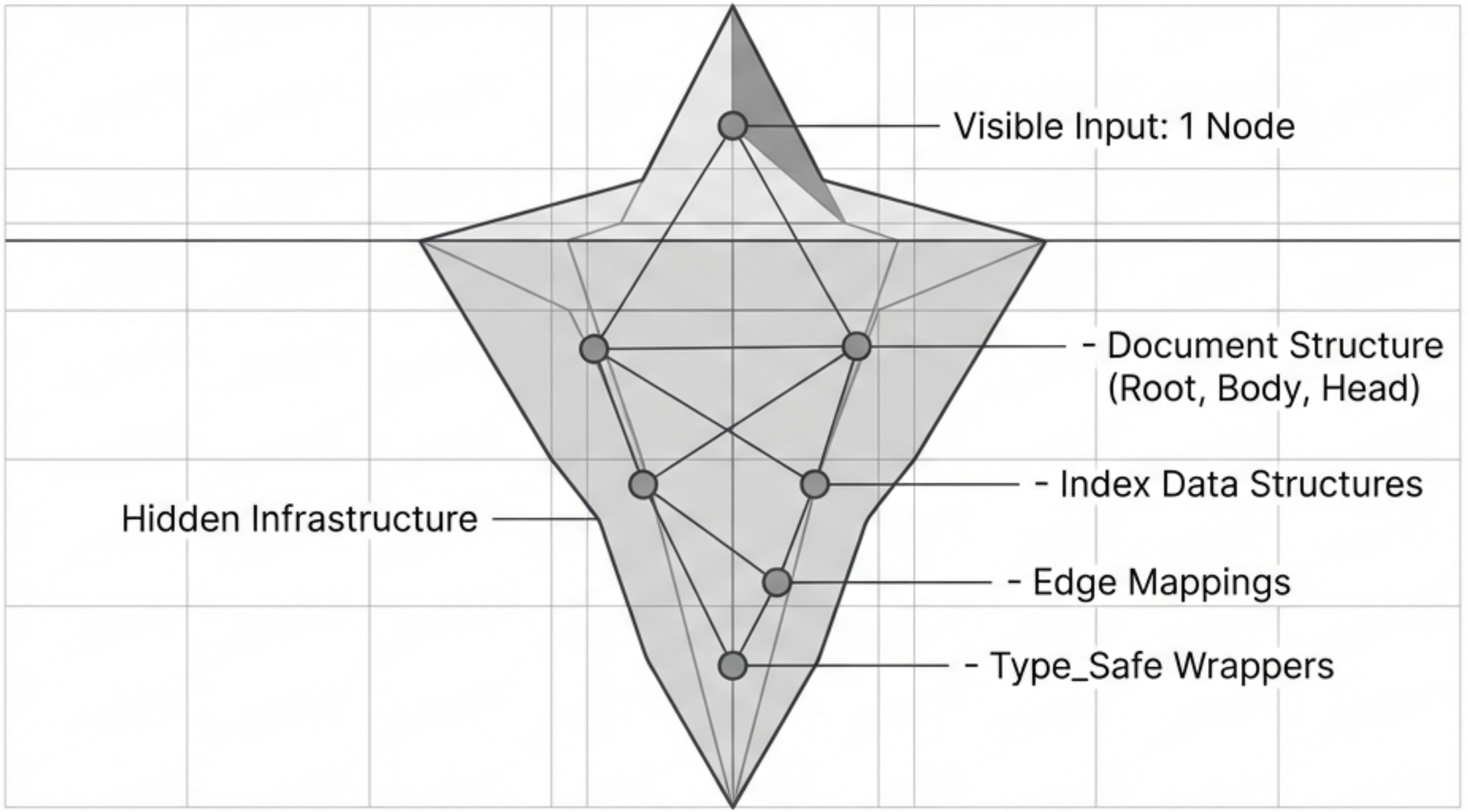
# De-risking: Converter Instantiation is Negligible



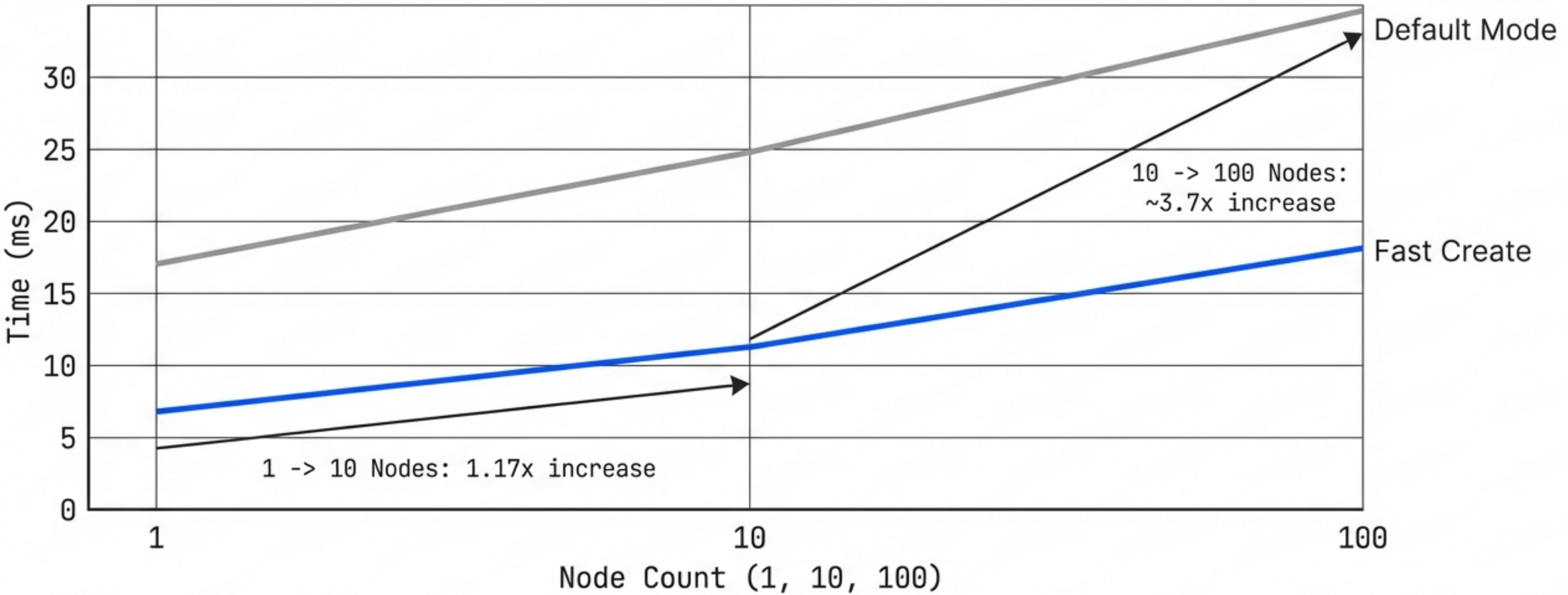

**We do NOT need to cache converter instances. The cost is entirely within the .convert() methods.**

# The 7ms Baseline Mystery

Even a single Hello World takes 7.40ms. Why?

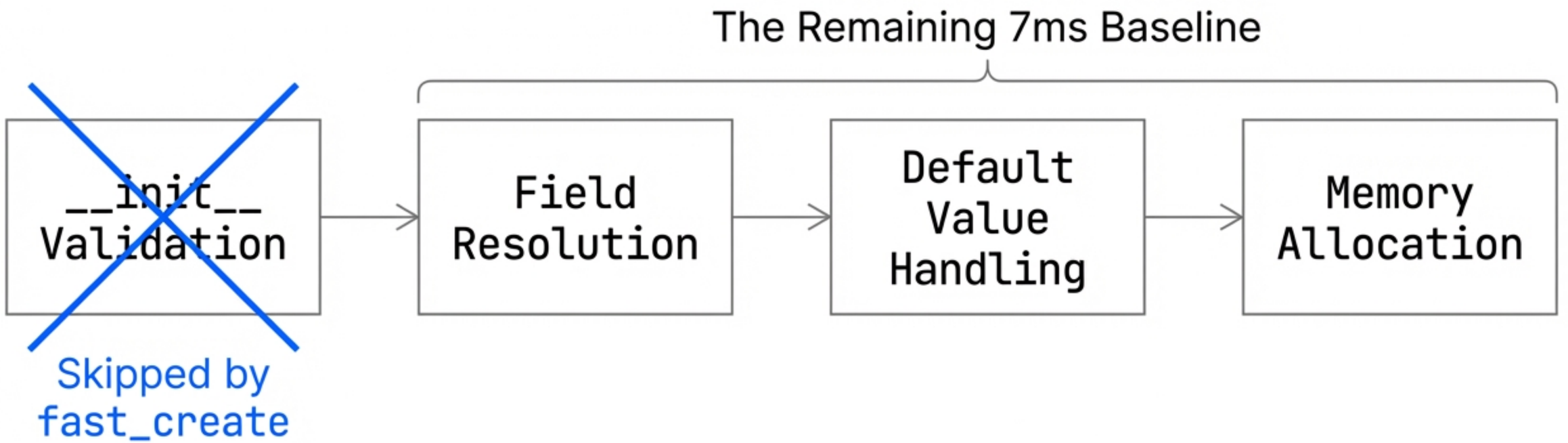


# Scaling Behaviour is Linear and Healthy




Conclusion: No O(n^2) behaviour. The algorithm is sound. The issue is the high fixed cost (‘the floor’), not the scaling ceiling.

# Root Cause: The Cost of “Type_Safe” Overhead




`fast_create` bypasses validation (**~56% savings**), but allocation costs remain. Further gains require `Object Pooling` or `C Extensions`.

# Immediate Action: Enable ‘fast_create’

**Target:** `Html__To__Html_MGraph__Document__Node_Id_Reuse.convert_from_dict()`

**Setting:** `fast_create = True`

**Validation:** `skip_validation = True`

Justification: Zero behavioural change. Immediate **50%** latency reduction. Confirmed **2x Speedup**.

# Future Strategy: Breaking the 7ms Barrier

## Investigation Targets

- **Metric Analysis:** Benchmark `MGraph` creation directly to isolate graph infrastructure.
- **Lazy Initialisation:** Assess if indexes must be built immediately.

## Architectural Shifts

- **Schema-Based Creation:** Pre-compute field layouts.
- **Object Pooling:** Reuse node/edge objects to avoid allocation costs.

# Summary & Conclusion

We have a sound linear algorithm with a high fixed cost. Enabling 'fast_create' buys us performance today while we solve the architectural overhead for tomorrow.

`TODAY: Enable 'fast_create' (2x Speedup)`

`TOMORROW: Optimise internal object allocation`

Phase E Analysis Complete. Ready for implementation.