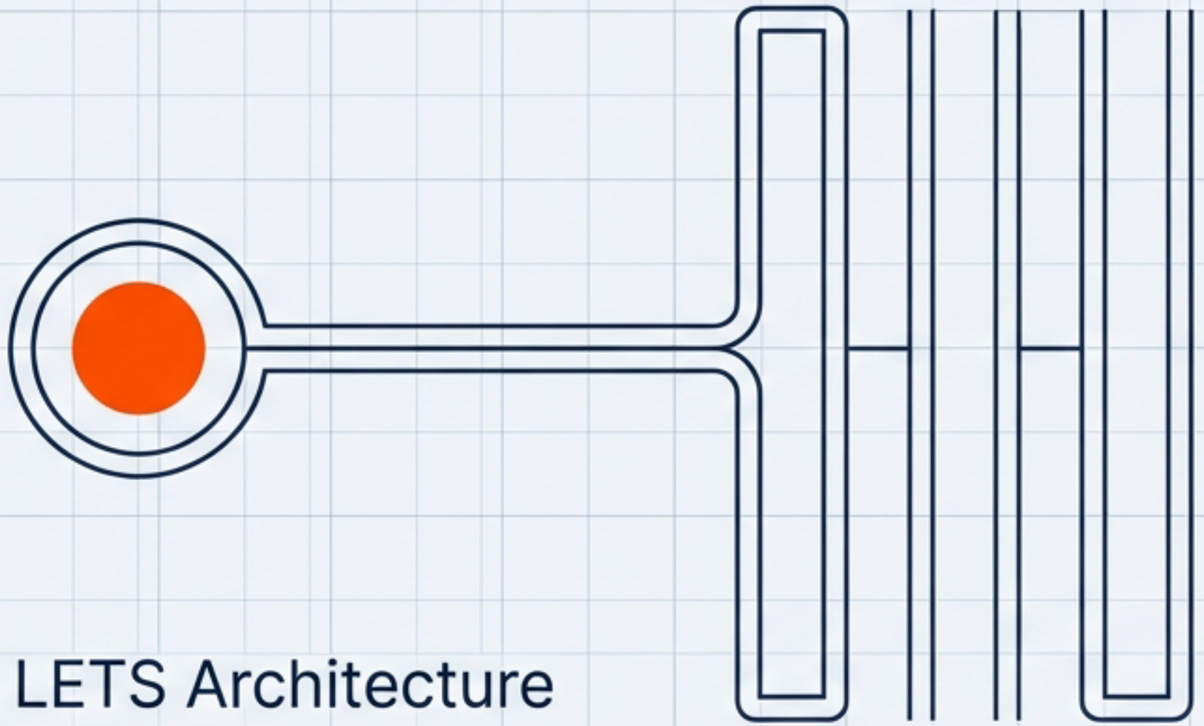
MGraph-AI Html Graph Service



Cache Storage & LETS Architecture

[redacted]

Implementation Brief v1.4.42

Status: Architecture Design - Ready for Implementation

Date: January 2026

CLASSIFICATION: TECHNICAL SPECIFICATION

Executive Summary

Strategic Context

The Mission: To build a cached HTML transformation pipeline that enables multi-stage transformations with full provenance and explainability.

The Driver: We are shifting from a legacy model of "recomputing everything" to a modern architecture of "intelligent reuse." This eliminates wasteful processing and enables granular inspection of every data state.

The Solution

What: A system handling Multi-stage HTML transformations where every step is cached.

How: Implementing two complementary systems:

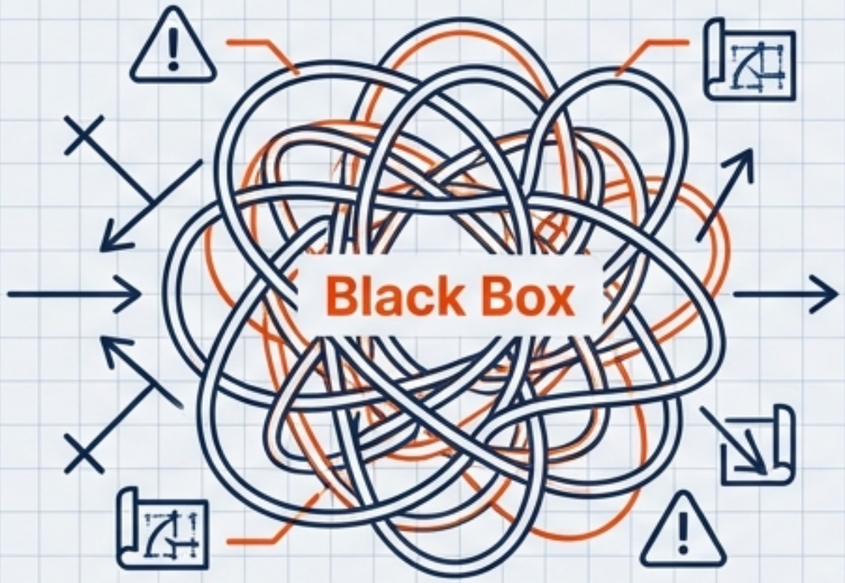
1. **LETS Framework**: A composable pipeline pattern (`Load` → `Extract` → `Transform` → Save).
2. **Cache Storage Layer**: A two-tier storage system using `MGraph-AI Cache Service`.

Key Insight: `LETS` is independent from caching. The cache is simply a target for the `Load`/Save phases, allowing for interchangeable backends.

Primary Value Proposition: **Decoupling Logic (LETS) from Memory (Cache) allows for testability, flexibility, and massive performance gains.**

From Black Box to Glass Box

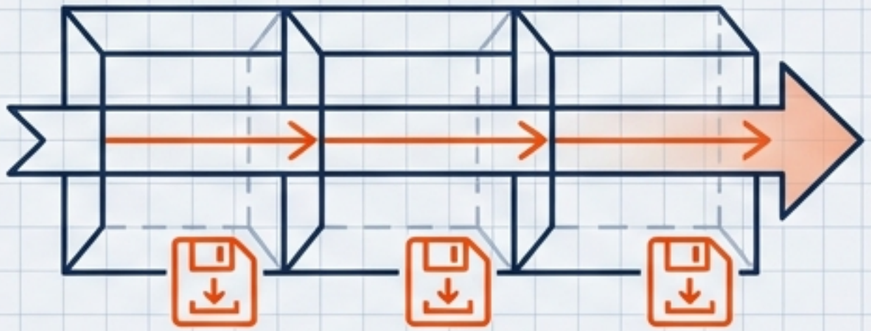
Current State (Friction)



- **No Caching:** Identical requests trigger expensive re-computations.
- **Tight Coupling:** Transformations hard-coded in handlers.
- **No Provenance:** Invisible intermediate states.

Architectural Evolution

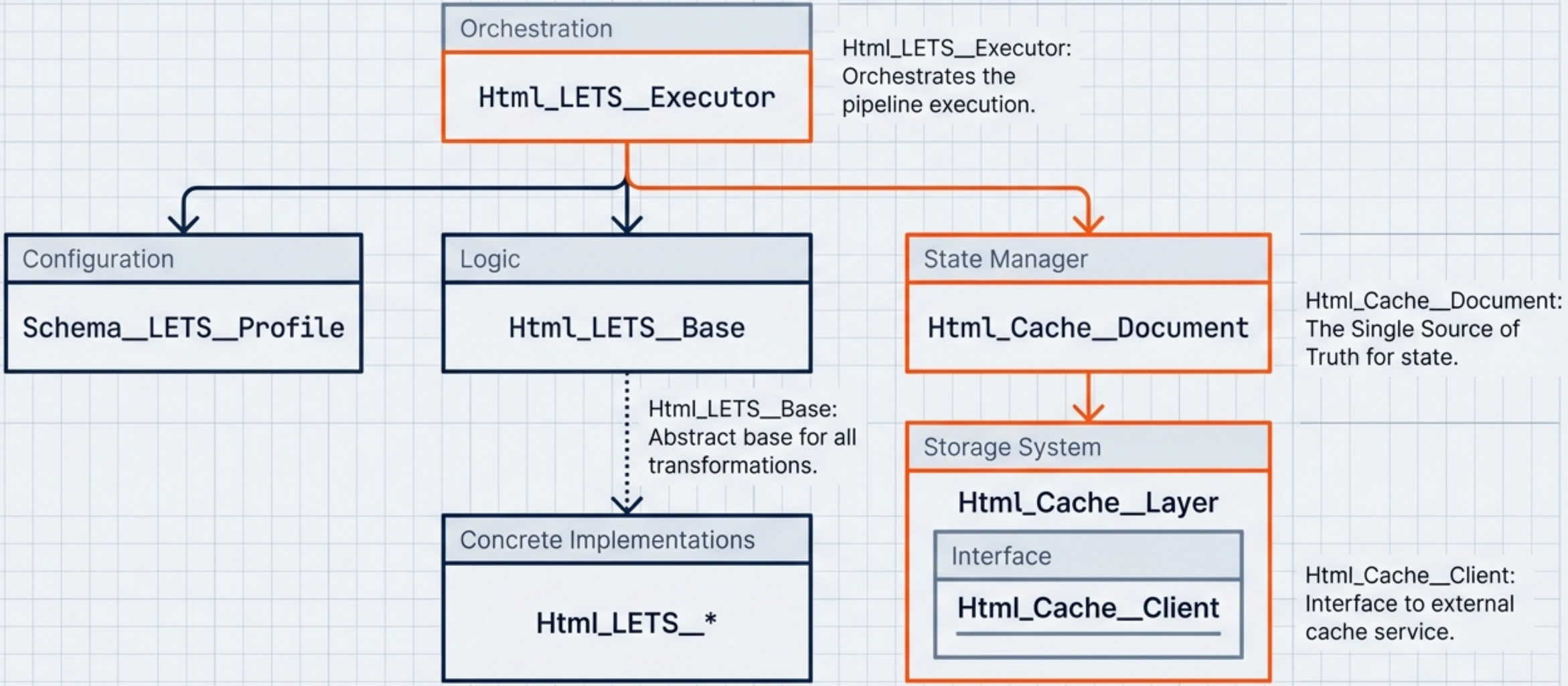
Desired State (Flow)



Glass Box

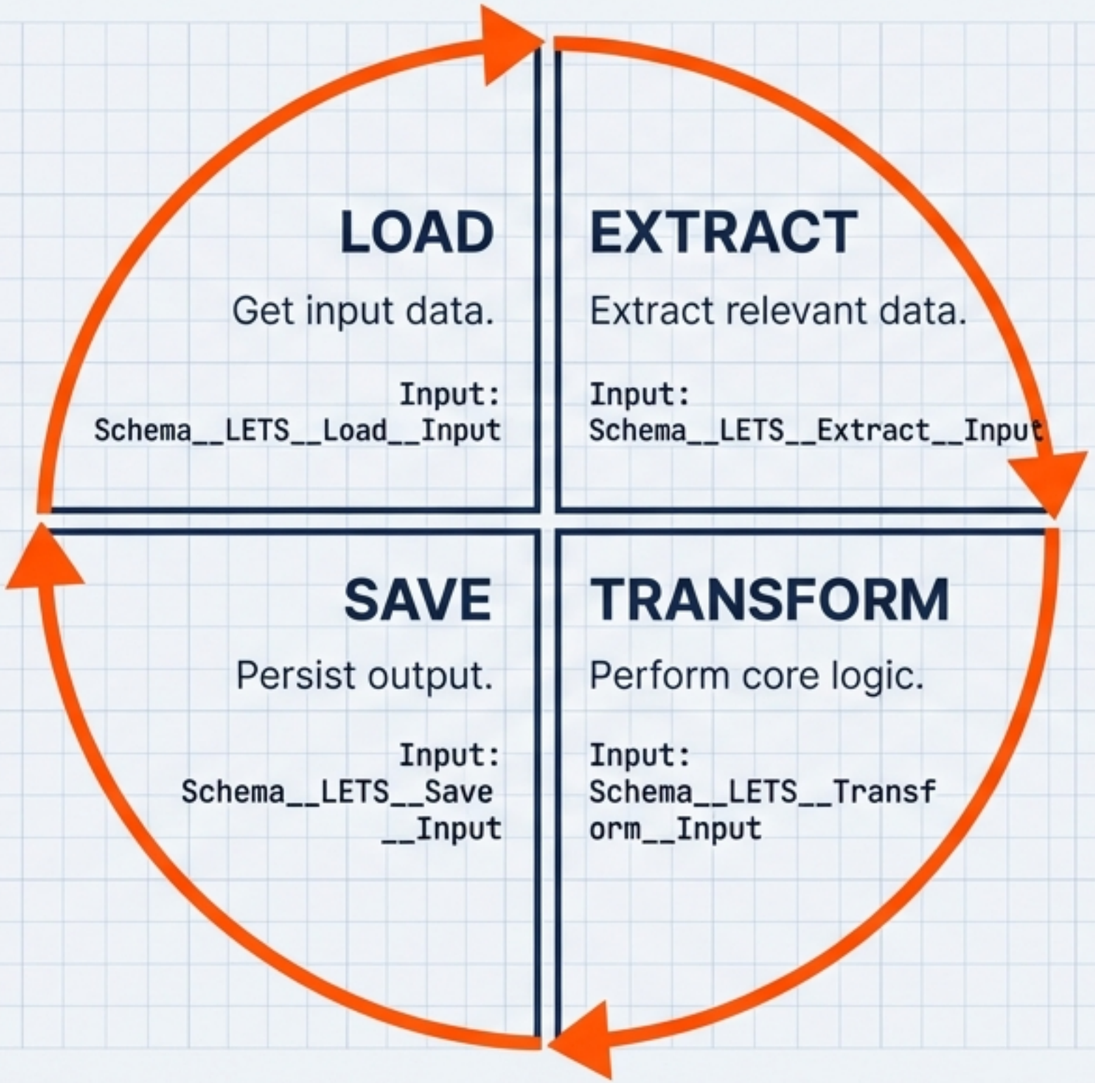
- **Traceability:** Inspect any step.
- **Optimization:** Zero-cost retrieval for hits.
- **Composable:** Test transformations in isolation.

System Architecture Overview



The LETS Framework

Logic Engine Transformation System



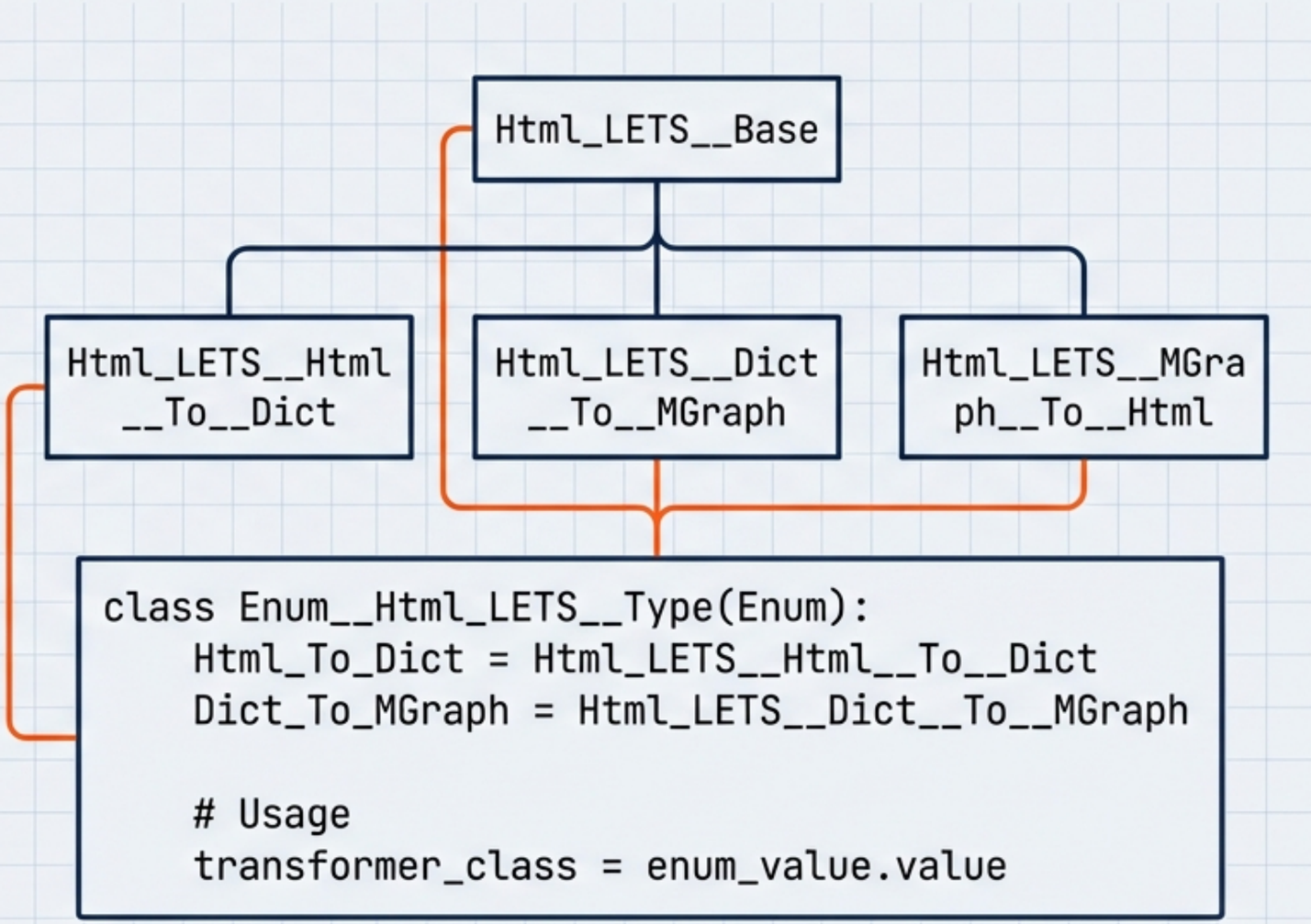
Design Principle: Cache Independence

LETS does not care ***where* it saves**, only ***that*** it saves. This allows swapping memory, disk, disk, or remote storage without changing business logic.

LETS Class Hierarchy & Enums

Code as Truth Strategy

- We map Enum names directly to class types.
- The Insight: Enum values ARE the class types.
- Benefit: Removes lookup logic errors. Enables Type[X] references to be JSON serialized.



The Cache Storage Layer

State Manager

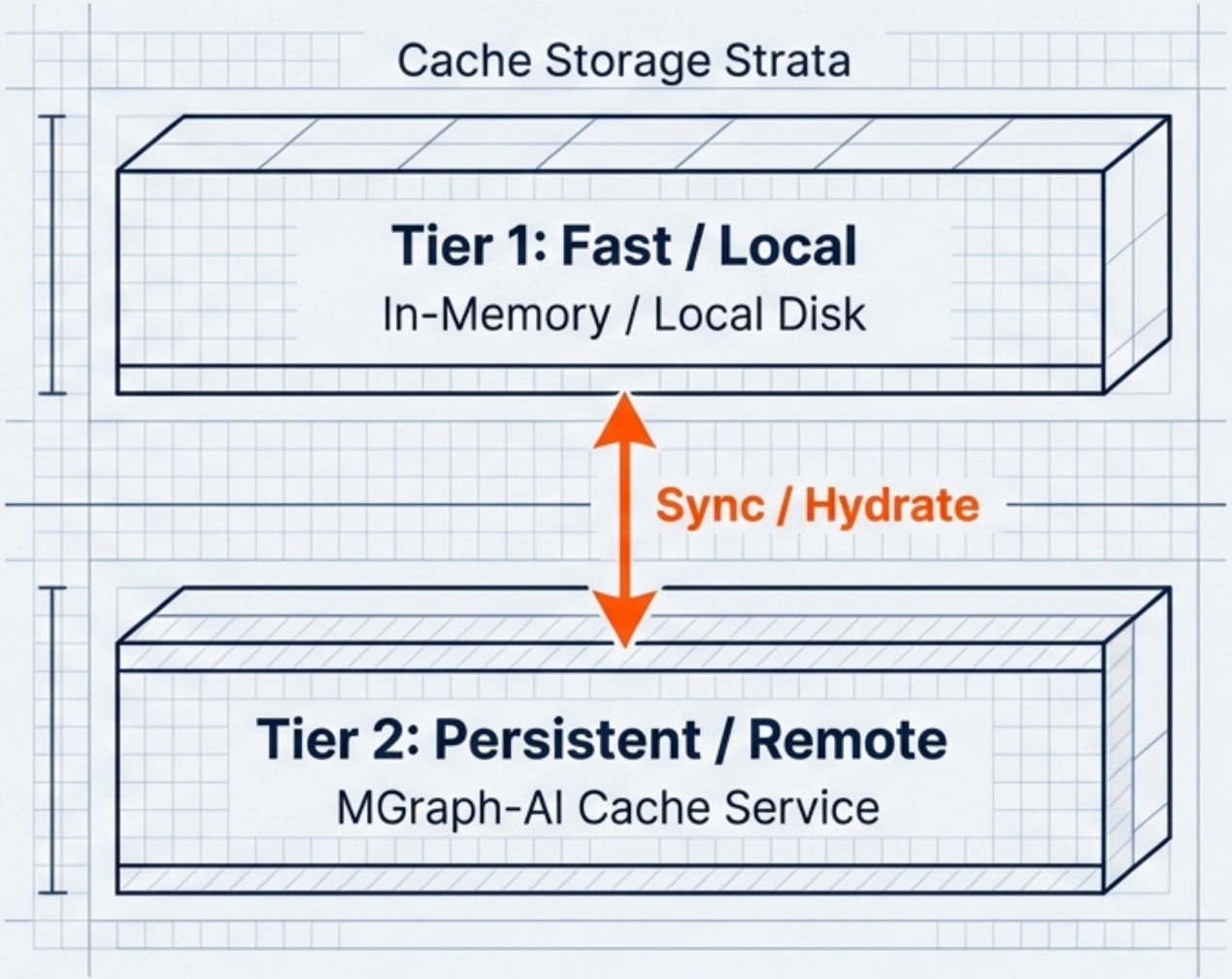
`Root Document (html-entry.json)`

Acts as the single source of truth.

Tracks: What has been built.

1. Project ID (Scope)
2. MGraph ID (Instance)
3. File ID (Asset)

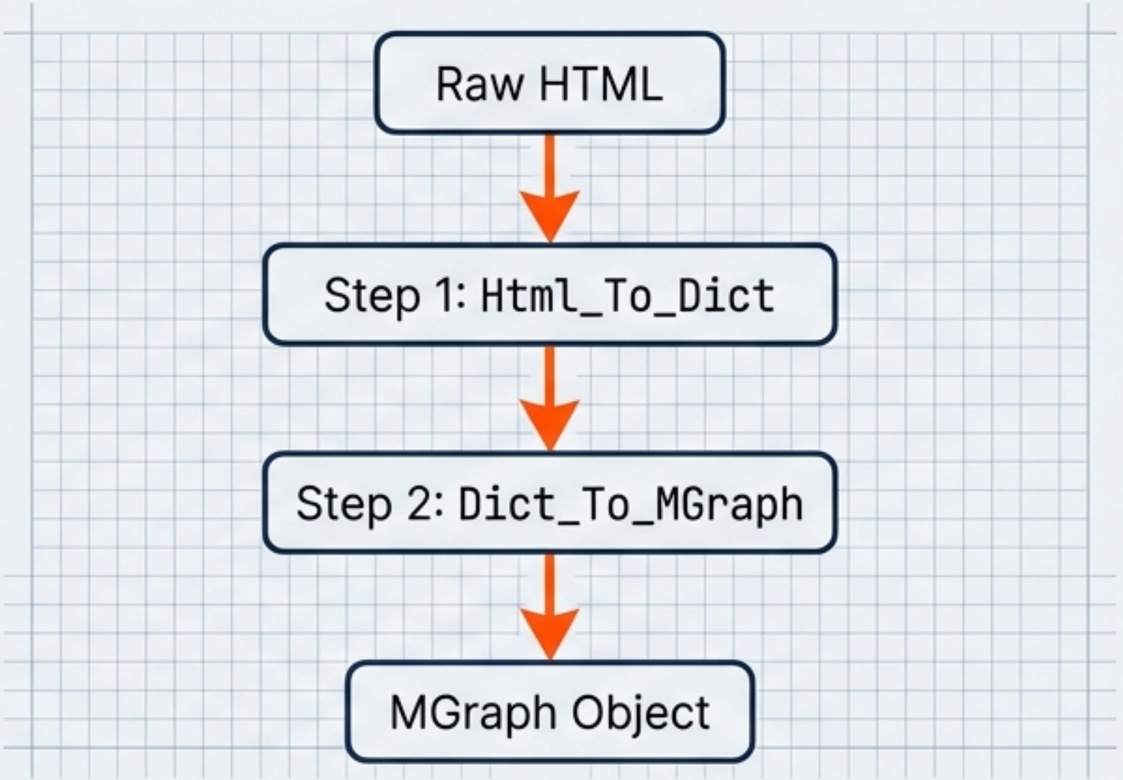
Access Pattern: Check Root Document -> Verify Layer Existence -> Fetch Payload.



Profiles System: Orchestration

Defining the Recipe

```
// Schema__LETS__Profile
{
  'profile_name': 'standard_extraction',
  'steps': [
    'Html_To_Dict',
    'Dict_To_MGraph'
  ]
}
```



Profiles separate configuration from implementation. Changing the JSON changes the pipeline behavior without code deployment.

Type System & Safety

The Non-Negotiables

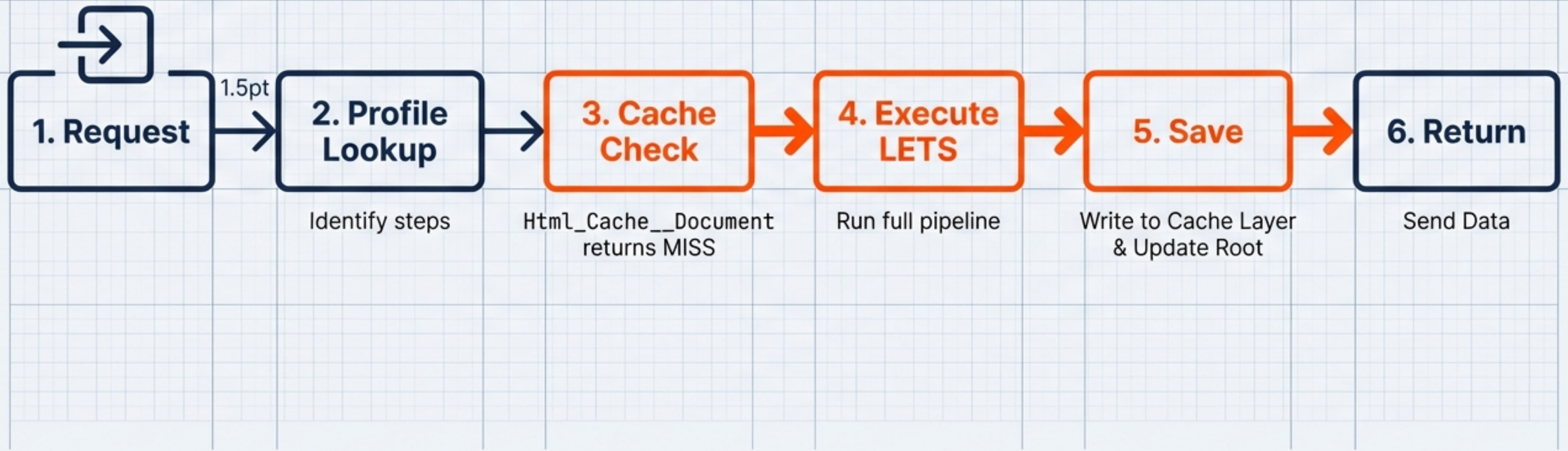
Unsafe Primitives (Banned)	Type Safe Wrappers (Required)
`str`	`Safe_Str__LETS__Name`, `Safe_Str__Cache_Key`
`dict`	`Schema__LETS__Config`, `Schema__LETS__Result`
`list`	`List[Schema__LETS__Step]`

Critical Rule: Never use raw str, int, dict, or list in Type_Safe classes.

Rationale: We trade initial development speed for long-term system reliability and self-documentation.

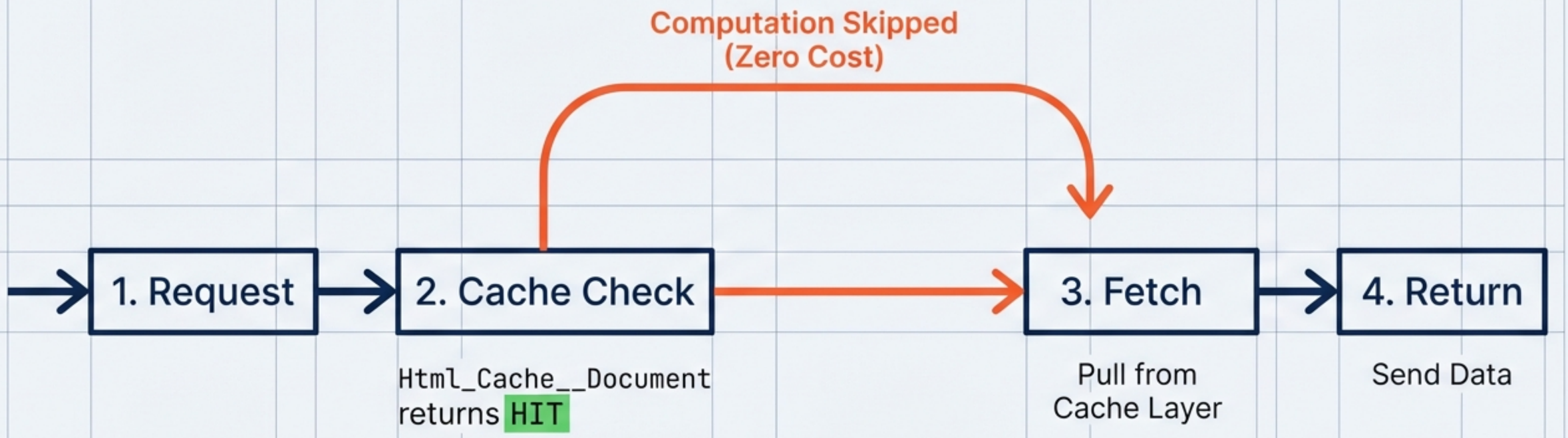
Data Flow: The Cache Miss

Workflow 1: First Request Execution



Data Flow: The Cache Hit

Workflow 2: Subsequent Request Optimization



Workflow 3 (Force Rebuild): Specific flag can force the system to ignore the cache hit.

Critical Schema Definitions

The Data Dictionary

LETS Schemas (Pure Data)

`Schema__LETS__Result`:
Standardized output container for any step.

`Schema__LETS__Config`:
Configuration parameters for transformations.

`Schema__LETS__Load__Input`:
Strict input definition for Load phase.

Cache Schemas (State)

`Schema__Html_Cache__Entry`:
Represents a single cached item's metadata.

`Schema__Html_Cache__Root`:
Structure of the 'html-entry.json' state file.

Design Rule:
Schemas contain NO methods. They are purely for validation and serialization boundaries.

API Integration & Package Structure

The Interface

Extended Request Schemas allow clients to specify:

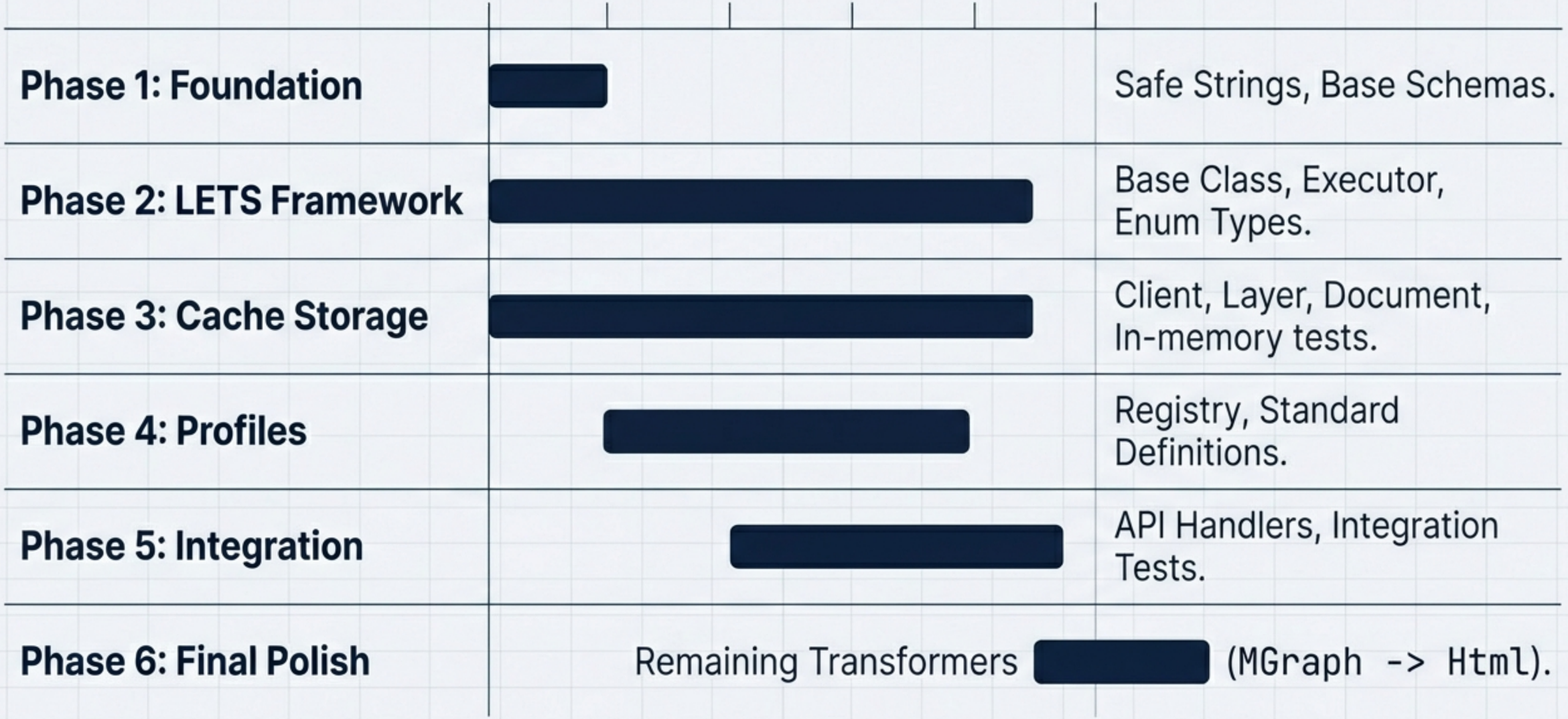
1. Target Profile
2. Force Rebuild Flags

Pattern: API Handlers are thin; they delegate complexity to `Html_LETS__Executor`

Package Structure

```
impl/
├── lets/              (Logic)
│   ├── base.py
│   ├── executor.py
│   └── transformers/
├── cache/             (Storage)
│   ├── client.py
│   └── document.py
└── profiles/          (Configuration)
    └── registry.py
```

Implementation Roadmap



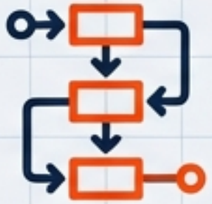
Summary & Key Takeaways

LETS Framework

Composable Logic.

Cache Storage

Two-Tier Provenance.



Profiles

Configurable Pipelines.

Type Safety

No Raw Primitives.

Independence

Decoupled Concerns.

We are building a system that is not just faster, but structurally smarter—enabling a transparent, observable, and evolution-ready pipeline.