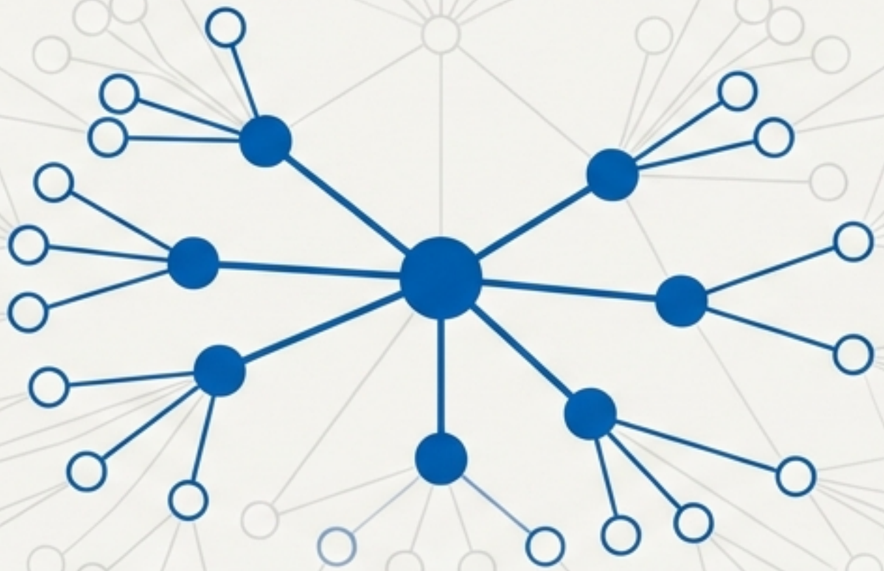
Optimising Object Creation in `osbot_utils`

Introducing `Type_Safe__On_Demand`



`osbot_utils` | Version v3.60.2 | December 2025

The Problem: The Hidden Cost of Deeply Nested Objects

Creating large `Type_Safe` object hierarchies can be a performance bottleneck. In some workflows, we pay the full construction cost for objects that are never even used.

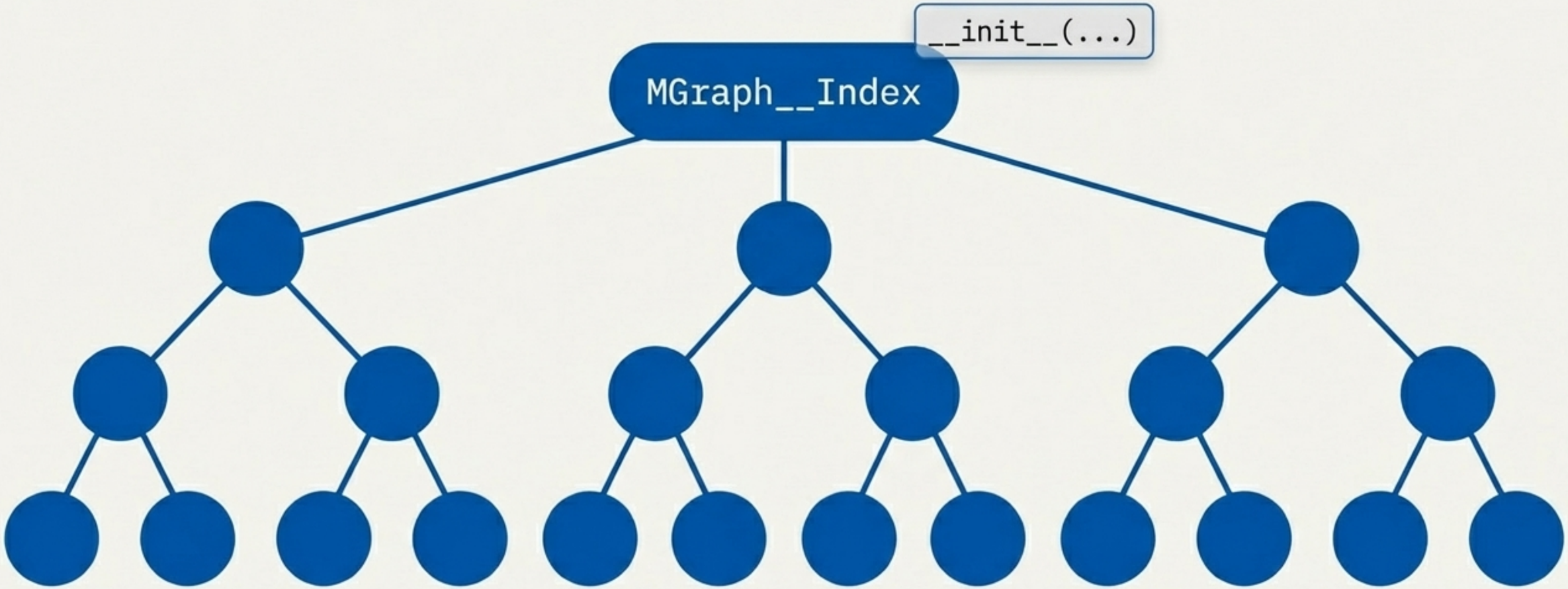
The `Html_MGraph` Use Case

- During document conversion, `Html_MGraph` eagerly creates 6 `MGraph__Index` instances.
- Each `MGraph__Index` is a complex `Type_Safe` class with many nested attributes.

This upfront creation process introduces a measurable overhead of

~11ms per document

Visualising Eager Creation



With standard Type_Safe, the entire object graph is created in memory immediately upon initialisation, even if only the root object is needed at first.

The Solution: Deferring Creation with **Type_Safe__On_Demand**

A new, performance-optimised variant of `Type_Safe` that defers the creation of nested `Type_Safe` objects until they are first accessed.

On-Demand

Nested objects are created only when you use them.

High-Performance

Drastically reduces initialisation time and memory allocation.

Drop-in Replacement

Maintains full API compatibility with the base Type_Safe class.

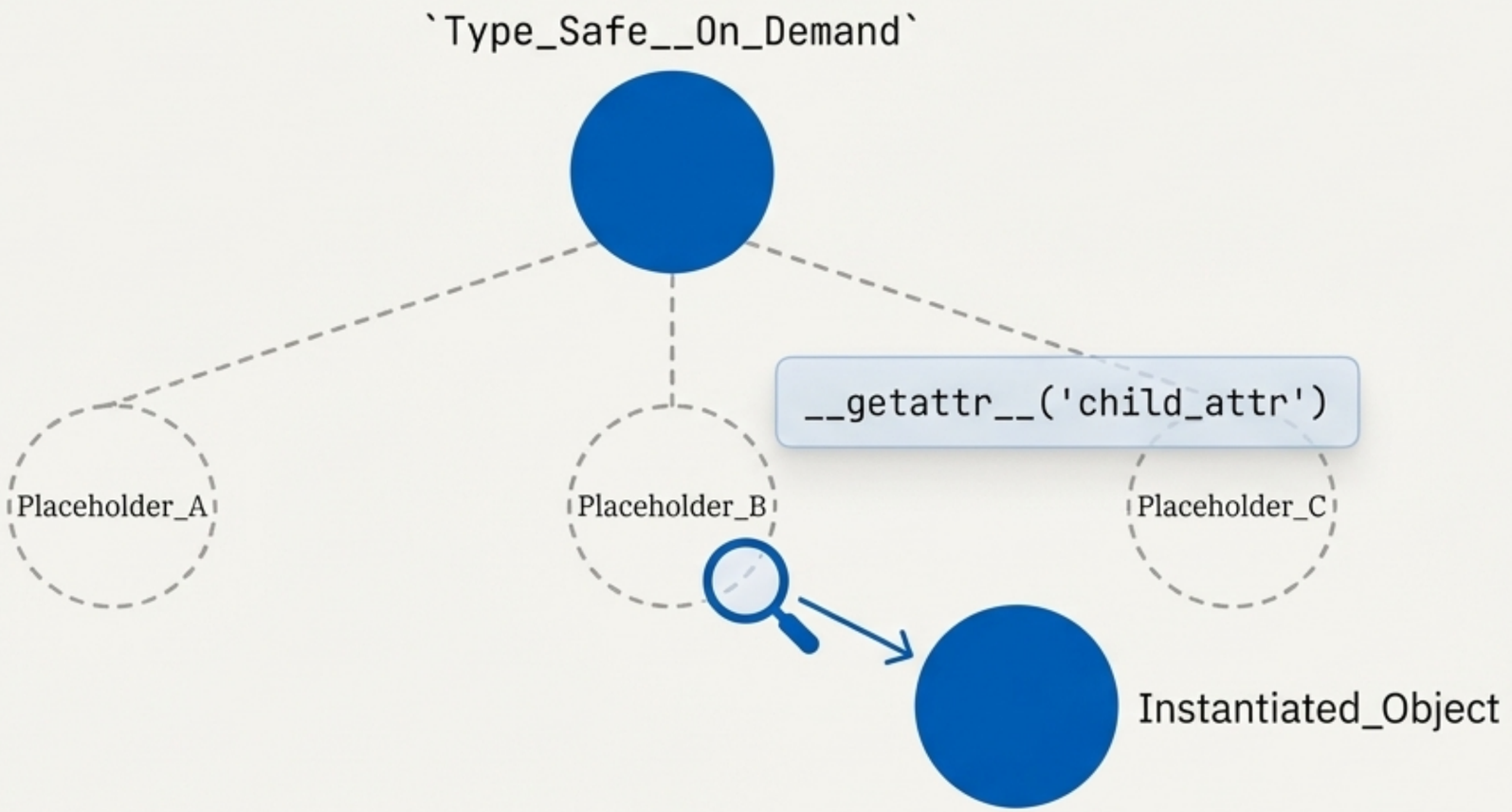
The Result: A 20x Performance Leap

Benchmark based on a single `MGraph__Index` object initialisation.

Metric	Type_Safe (Eager)	Type_Safe__On_Demand	Improvement
Construction Time	~1,800 µs	~90 µs	**20x faster**
Objects Created	47	1	**98% reduction**
Memory Allocation	All Upfront	On-Demand	**Deferred**

By deferring the creation of 46 child objects, initialisation becomes virtually instantaneous.

How It Works: The On-Demand Mechanism



The root object is created instantly. Nested `Type_Safe` attributes are stored as types and are only instantiated into full objects upon their first access.

What Gets Deferred? A Clear Breakdown

Deferral is targeted specifically at expensive object creations. Cheap, simple containers and primitives are initialised as normal to ensure predictable behaviour.

Type	Deferred?	Reason
`Type_Safe` subclasses	✅ Yes	These are the primary targets; expensive to create.
`Type_Safe__On_Demand` subclasses	✅ Yes	The same principle applies recursively.
`Dict[K, V]` (via `Type_Safe__Dict`)	❌ No	Initialised as empty dicts. Cheap to create.
`List[T]` (via `Type_Safe__List`)	❌ No	Initialised as empty lists. Cheap to create.
`Type_Safe__Primitive` (`Safe_Str`, etc.)	❌ No	Immutable and often require upfront validation.
Native primitives (`str`, `int`, `bool`)	❌ No	Zero overhead; treated as regular attributes.

Migration is a Single-Line Change

	Before		After
	`from osbot_utils.type_safe.Type_Safe import Type_Safe`		`from osbot_utils.type_safe.Type_Safe__On_Demand import Type_Safe__On_Demand`
-	`class MGraph__Index(Type_Safe):`	+	`class MGraph__Index(Type_Safe__On_Demand):`
	`    # ... attributes`		`    # ... attributes`

Simply change the parent class. All existing type hints and attributes work without modification.

When Should You Use `Type_Safe__On_Demand`?

✅ Use When...

- Your class has **multiple nested `Type_Safe` attributes**.
- Many of those attributes are **not accessed in typical workflows**.
- Construction performance is a **known or suspected bottleneck**. (Use `Performance_Measure__Session` to verify).

❌ Avoid When...

- Your class has few or no nested `Type_Safe` attributes.
- All attributes are guaranteed to be accessed immediately after construction.
- The class primarily contains primitives or collections (`list`, `dict`).

Proven Integration Patterns

Index Classes

The primary use case. Ideal for classes that act as a container for multiple, independent sub-indexes.

```
class MGraph__Index(Type_Safe__On
    nodes: Nodes_Index
    edges: Edges_Index
```

Mixed Hierarchies

Combine on-demand and eager classes. Use `Type_Safe__On_Demand` at the top level and standard `Type_Safe` for children that are always needed.

```
class Config(Type_Safe__On_Demand
    database: Db_Config  # Eager
    cache: Cache_Config  # Eager
```

Factory Classes

Useful for factories that can construct many different types of complex objects, but only one is chosen per instance.

```
class Report_Factory(Type_Safe__O
    pdf_builder: Pdf_Report
    csv_builder: Csv_Report
```

Understanding the API and Internal State

`Type_Safe__On_Demand` inherits all methods from `Type_Safe`. The standard interface (`.json()`, `.reset()`, etc.) is fully preserved.

New Internal Attributes

- `_on_demand__types: dict[str, type]`: Stores pending attribute names and their types.
- `_on_demand__init_complete: bool`: A flag that becomes `True` after `__init__` completes, enabling the on-demand mechanism.

Modified Behaviour

- **`__repr__()`**: The representation now indicates pending attributes for easier debugging.

```
<MGraph__Index (10 attrs pending)>
```

Common Pitfalls and How to Avoid Them (1/2)

Expecting .json() to Trigger Creation

Problem: Calling `.json()` or `.obj()` on a new instance will not create pending objects. The serialised output will only contain attributes that have already been explicitly accessed.

Solution: If you need a fully populated object for serialisation, you must access each required attribute first.

```
index = MGraph__Index()
index.json()         # Returns: {}
index.nodes          # Triggers creation of 'nodes'
index.json()         # Returns: {'nodes': {...}}
```

Assignment to Pending Attributes

Problem: Assigning a value to an attribute that is still pending may be overwritten when that attribute is accessed for the first time (triggering its default creation).

Solution: Access the attribute to create it before you modify it.

Common Pitfalls and How to Avoid Them (2/2)

Assuming All `Type_Safe` Attributes Are Deferred

Remember that collections (`Type_Safe__List`, `Type_Safe__Dict`) and primitives (`Safe_Str`) are **not** deferred. They are created immediately as empty containers or default values.

```
class My_Class(Type_Safe__On_Demand):
    complex_child: Complex_Object  # DEFERRED
    tags: list[str]                # NOT deferred, created as []
    name: str                      # NOT deferred, created as ''

instance = My_Class()
# instance._on_demand__types == {'complex_child': <class 'Complex_Object'>}
# instance.__locals__() == {'tags': [], 'name': ''}
```

Compatibility and Guarantees

Feature	Compatible	Notes
`Type_Safe` inheritance	✅	Acts as a direct subclass.
JSON serialisation (`.json()`)	✅	Serialises accessed attributes only.
Deserialisation (`from_json()`)	✅	Works as normal, populating all fields.
Object comparison (`.obj()`)	✅	For a full comparison, access all attributes first.
`Type_Safe__Dict` / `__List`	✅	Works as normal; these attributes are not deferred.
`Type_Safe__Primitive`	✅	Works as normal; not deferred.
Context Managers	✅	Supported without any changes.
`.reset()` method	✅	Resets the object to its initial pending state.

Summary: Faster, Smarter Object Initialisation

20x Faster Construction: For complex, nested hierarchies.

Incremental Memory: Allocate resources only when needed.

Full API Compatibility: A safe and familiar interface.

Trivial Migration: A single-line inheritance change.

For the `Html_MGraph` project, this single change saves **~10ms of overhead per document**, leading to a significant throughput increase in batch processing.