# .NET Garbage Collection

## *A GC Primer*

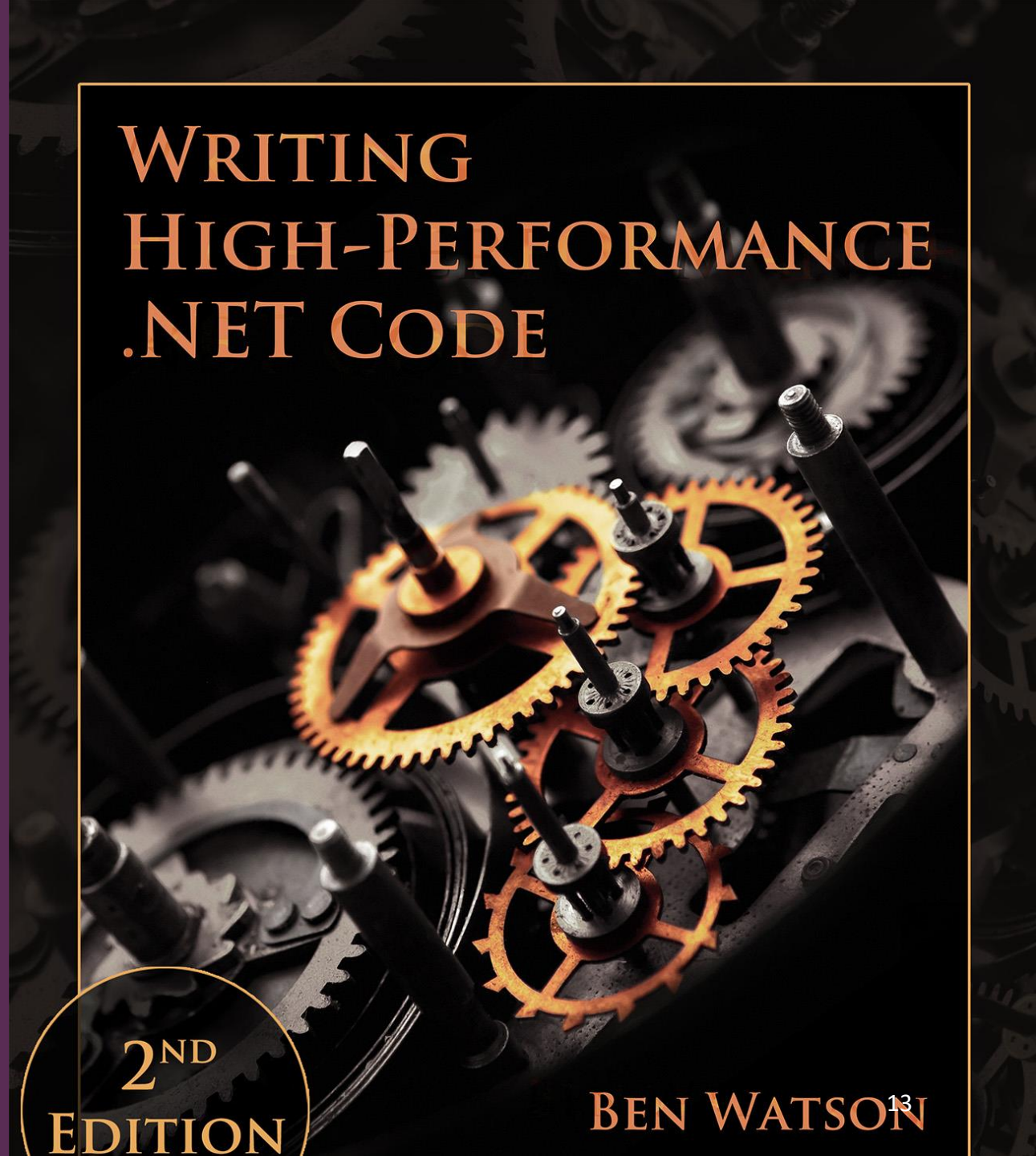
**Maximilian Meffert**

BRICKMAKERS

# .NET Performance

**"In .NET, you need to think of memory performance at least as much as CPU performance.**
It is so fundamental to smooth .NET operation that the most significant chunk of this book's content is dedicated to just this topic."

**Ben Watson**
***Writing High-Performance .NET Code***



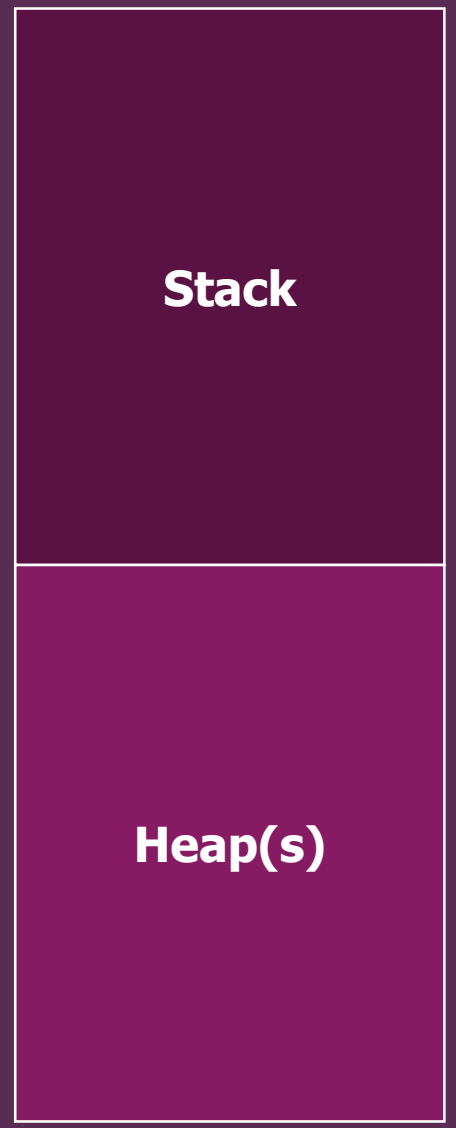
# .NET CLR Memory

**Stack**

*Used for storing function calls and local variables which are not classical objects.*

**Heap(s)**

*Used for storing objects.*



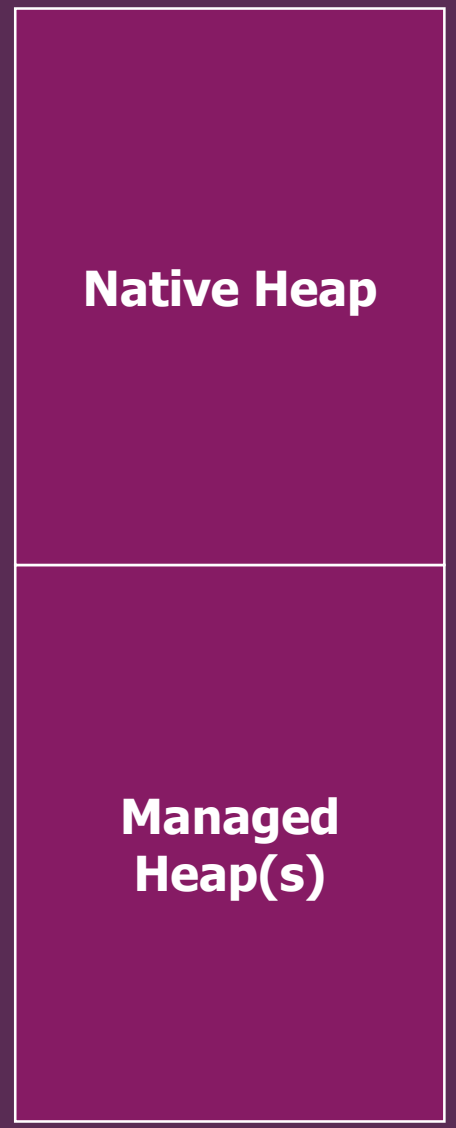
# .NET CLR Heaps

**Native Heap**

*Used by Windows and the CLR for unmanaged memory, e.g.: Windows API, OS data structures, the CLR itself, etc.*

**Managed/GC Heap(s)**

*Used by the CLR to allocate objects subjected to garbage collection.*



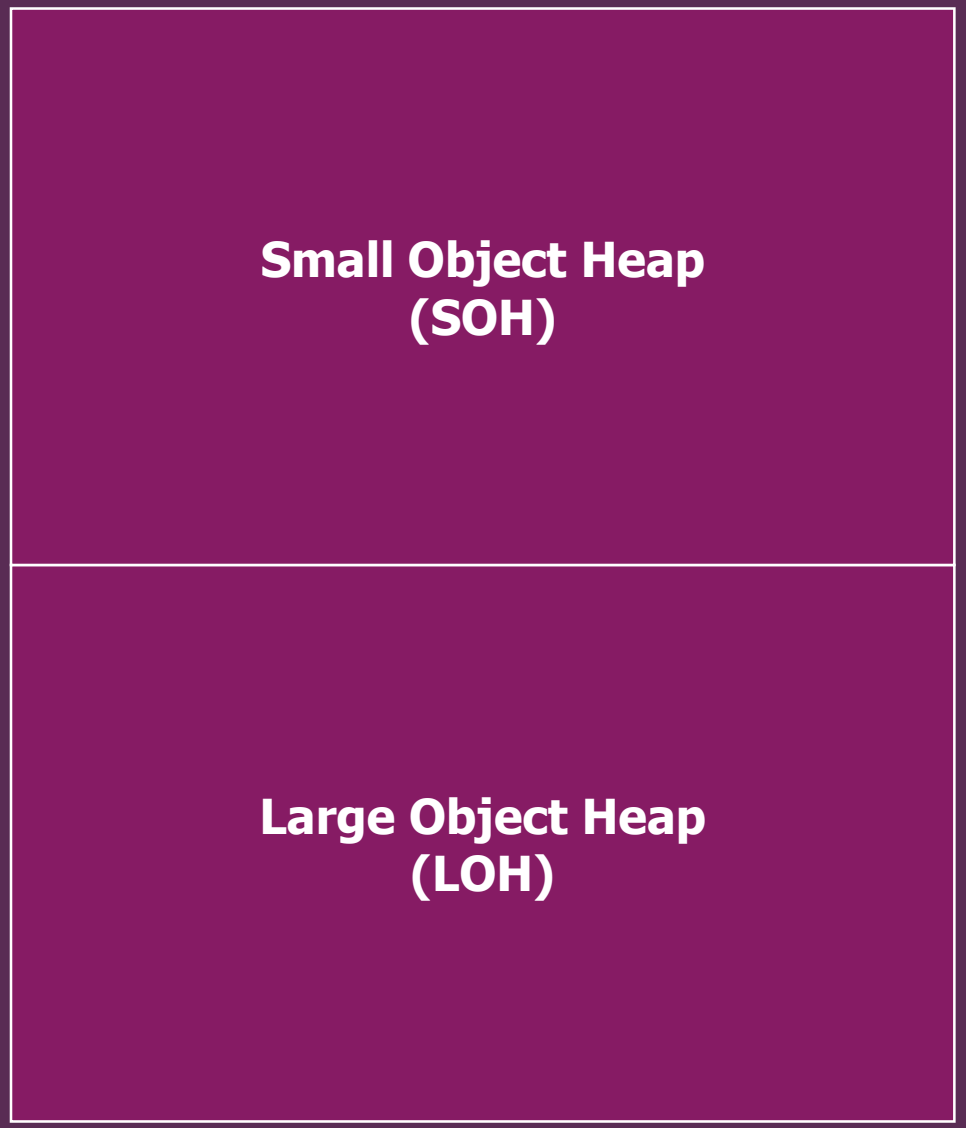
# The Managed Heap(s)

**Small Object Heap**

*for objects < 85000 bytes*

**Large Object Heap**

*for objects ≥ 85000 bytes*



# The Managed Heap(s)

| Example | Memory Size | Heap |
|---|---|---|
| `class A {}`<br>`var obj = new A();` | 12 bytes (x86)<br>24 bytes (x64) | SOH |
| `var arr = new byte[84988];` | 85000 bytes (x86)<br>85016 bytes (x64) | LOH |
| `var str = new string('a', 42492);` | 85000 bytes (x86)<br>85016 bytes (x64) | LOH |
| `var str = "";` | 0 bytes | ? |
| `var str = "String with 42492 chars …";` | 0 bytes | ? |

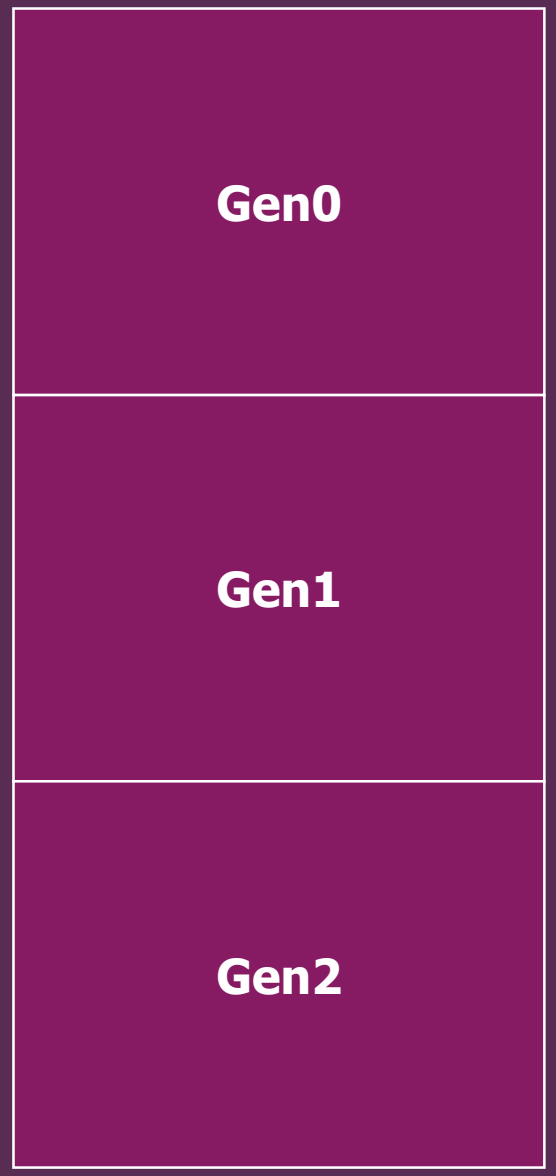
# The Managed Heap(s)

```
class A { }
class Program
{
    static void Main(string[] args)
    {
        var initialMemory = GC.GetTotalMemory(true);

        //var obj = new A();                                    //    12 bytes (x86) |    24 bytes (x64)
        //var obj = new object();                               //    12 bytes (x86) |    24 bytes (x64)
        //var arr = new byte[0];                                //    12 bytes (x86) |    24 bytes (x64)
        //var arr = new byte[1];                                //    16 bytes (x86) |    32 bytes (x64)
        //var arr = new byte[5];                                //    20 bytes (x86) |    32 bytes (x64)
        //var arr = new byte[84988];                            // 85000 bytes (x86) | 85016 bytes (x64)
        //var str = "";                                         //     0 bytes (x86) |     0 bytes (x64)
        //var str = "string with 42492 characters ...";         //     0 bytes (x86) |     0 bytes (x64)
        //var str = new string('a', 42492);                     // 85000 bytes (x86) | 85016 bytes (x64)
        //var num = 42;                                         //     0 bytes (x86) |     0 bytes (x64)
        //var nums = new int[0];                                //    12 bytes (x86) |    24 bytes (x64)
        //var nums = new int[1];                                //    16 bytes (x86) |    32 bytes (x64)
        //var nums = new int[5];                                //    32 bytes (x86) |    48 bytes (x64)

        var finalMemory = GC.GetTotalMemory(true);

        Console.WriteLine(finalMemory - initialMemory);
        Console.ReadKey();
    }
}
```

# The Small Object Heap

## Object Lifetime

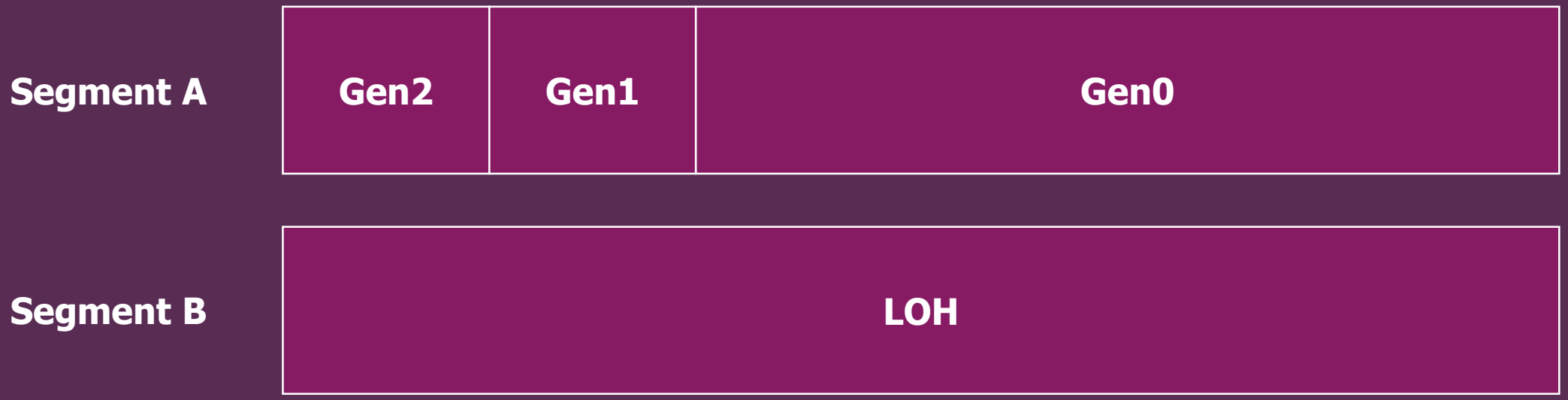
*When objects survive garbage collection they get promoted to a higher generation starting from Gen0 and ending in Gen2.*



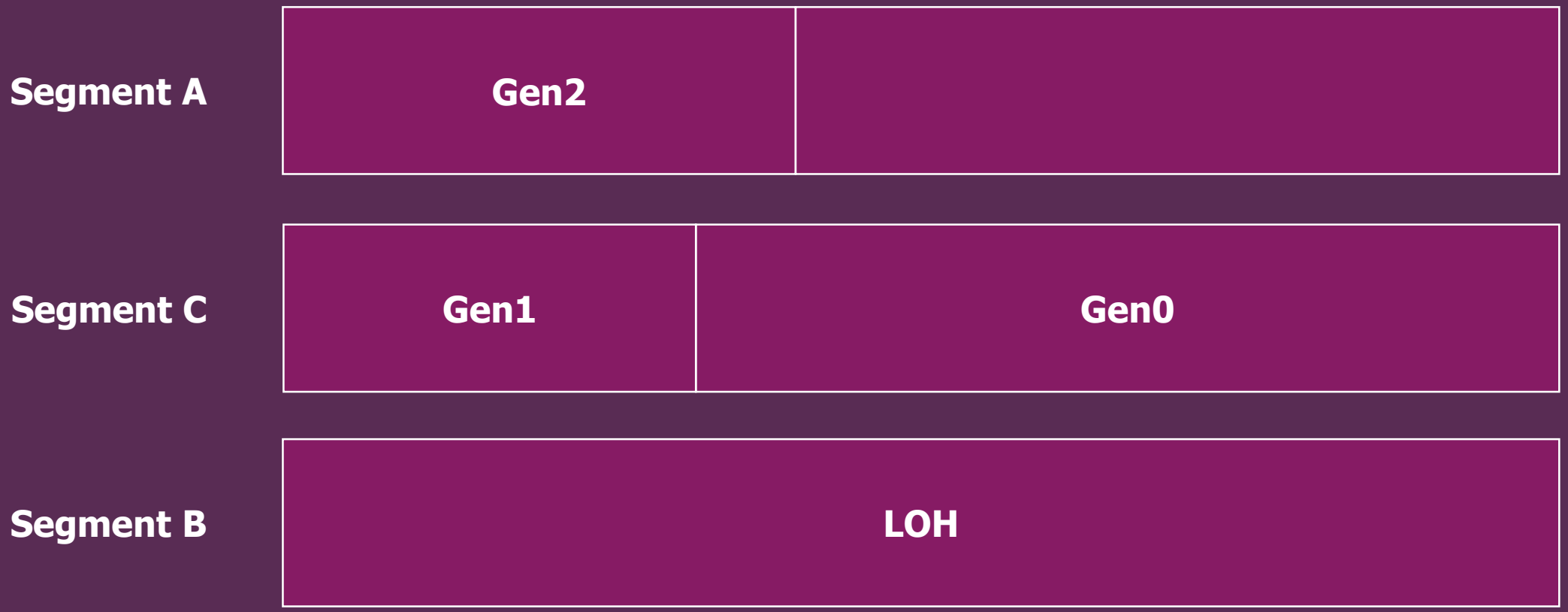
# Memory Segmentation

## Rules

1. *All segments have the same size.*
2. *LOH and Gen2 can span multiple segments.*
3. *Gen0 and Gen1 always reside in the same segment.*

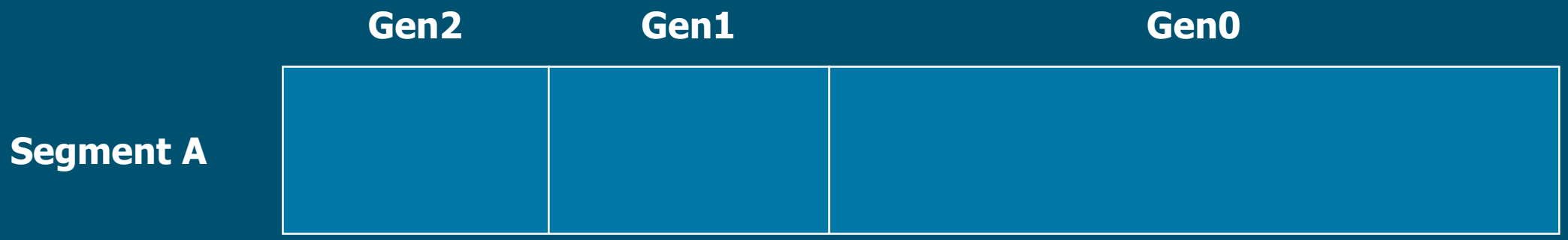


# Memory Segmentation



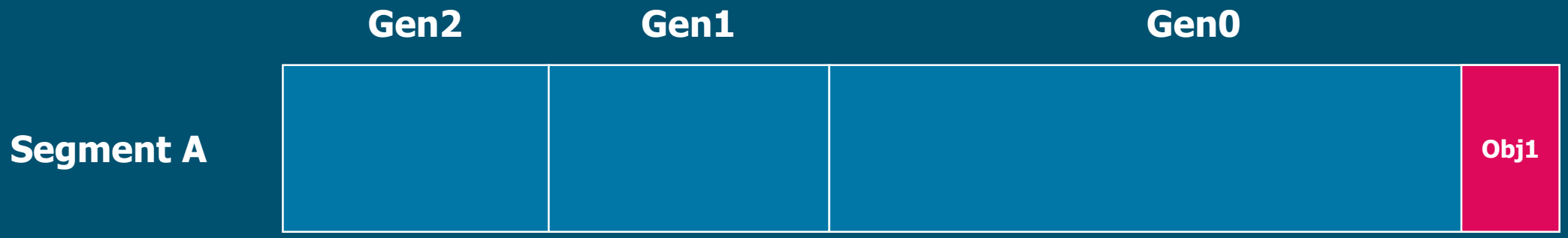
# Allocation

```
Obj1 = new A();
```



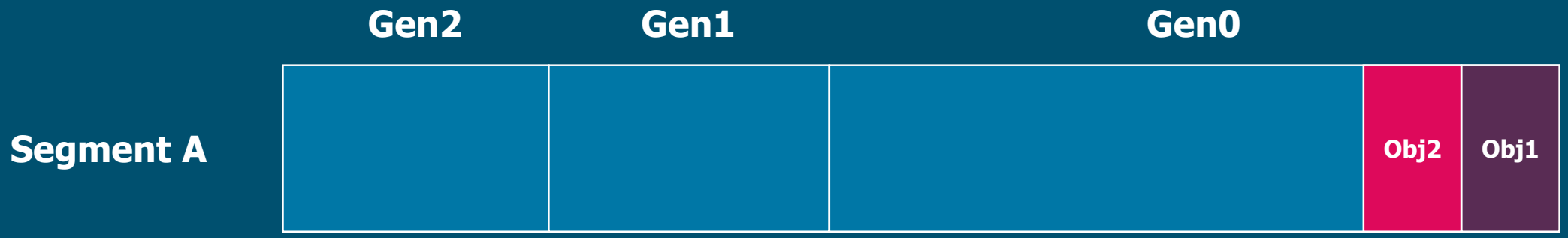
# Fast Allocation

## Try to allocate at the end of Gen0.



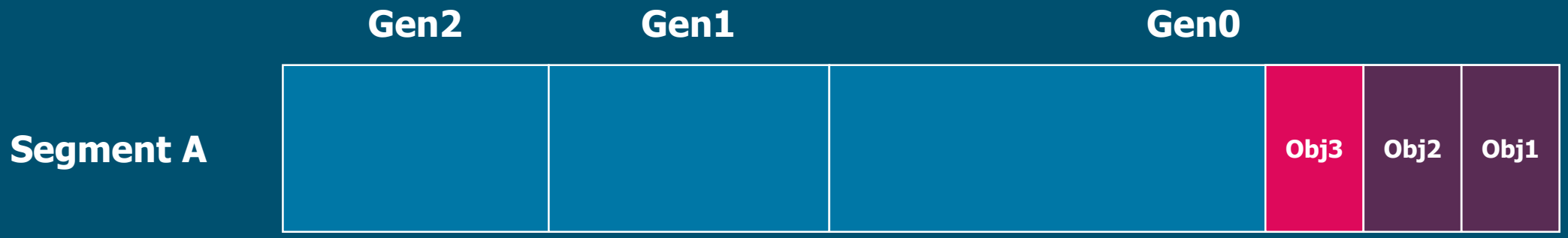
# Fast Allocation

```
Obj2 = new A();
```



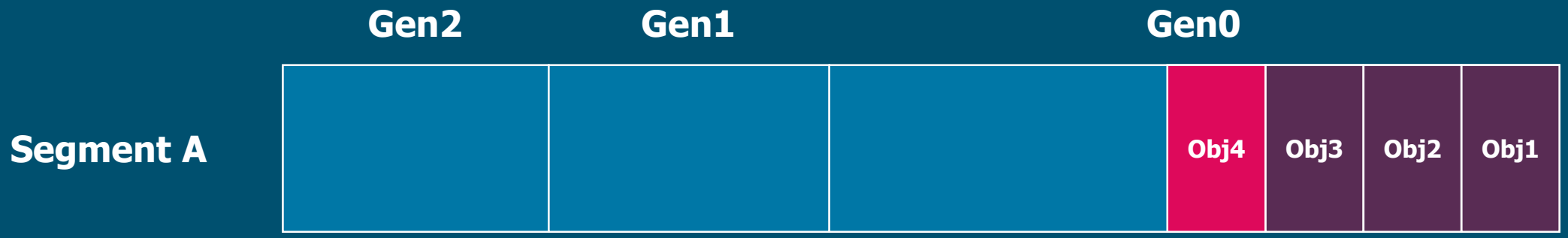
# Fast Allocation

```
Obj3 = new A();
```



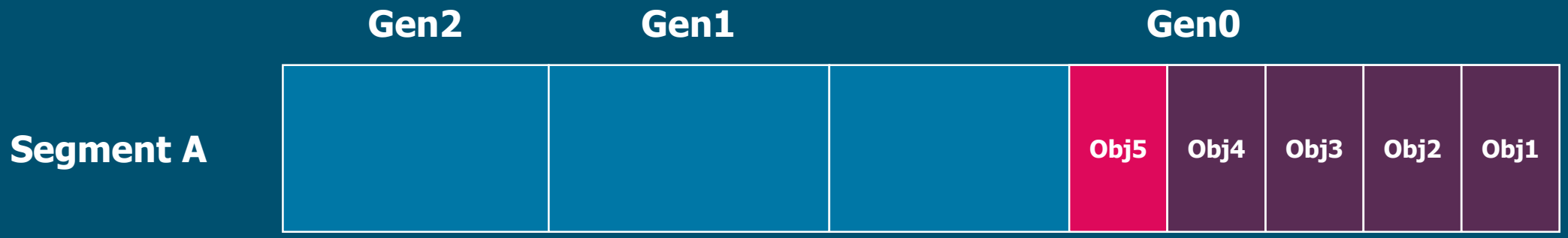
# Fast Allocation

```
Obj4 = new A();
```



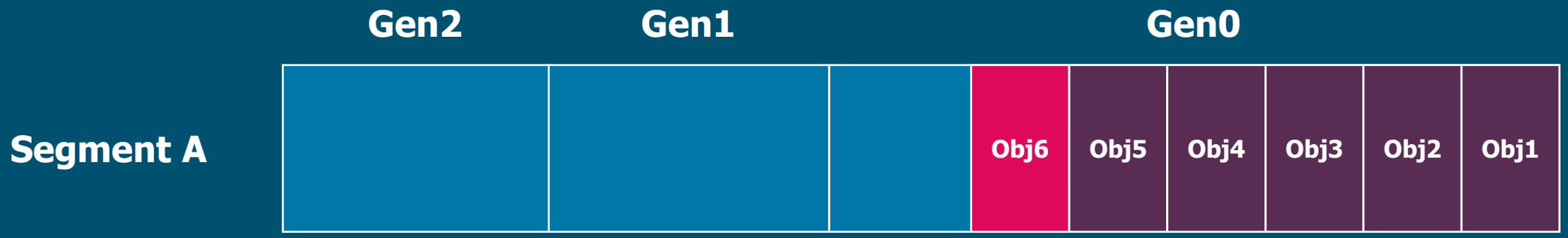
# Fast Allocation

```
Obj5 = new A();
```



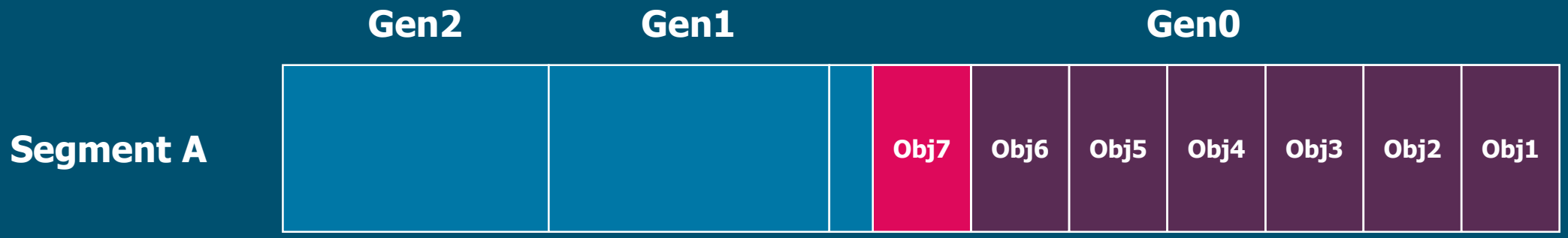
# Fast Allocation

```
Obj6 = new A();
```



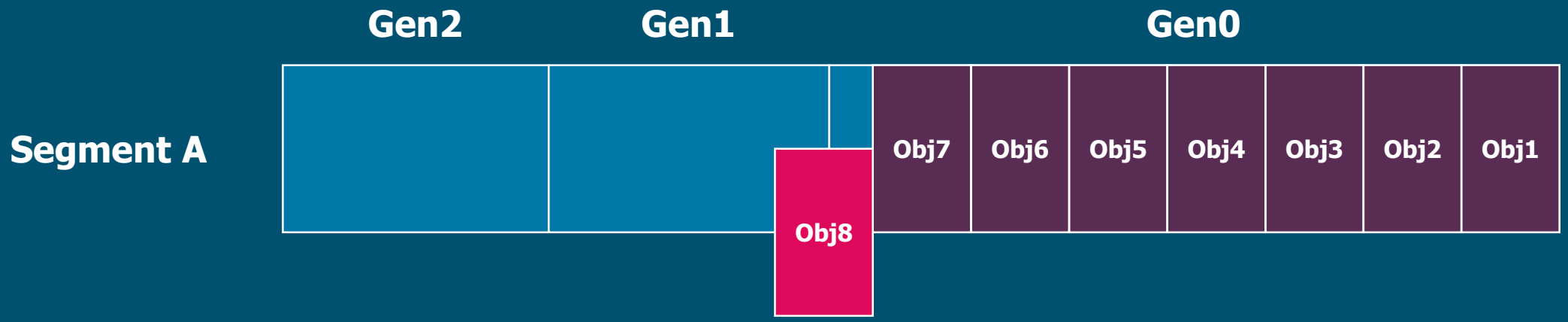
# Fast Allocation

```
Obj7 = new A();
```



# Fast Allocation

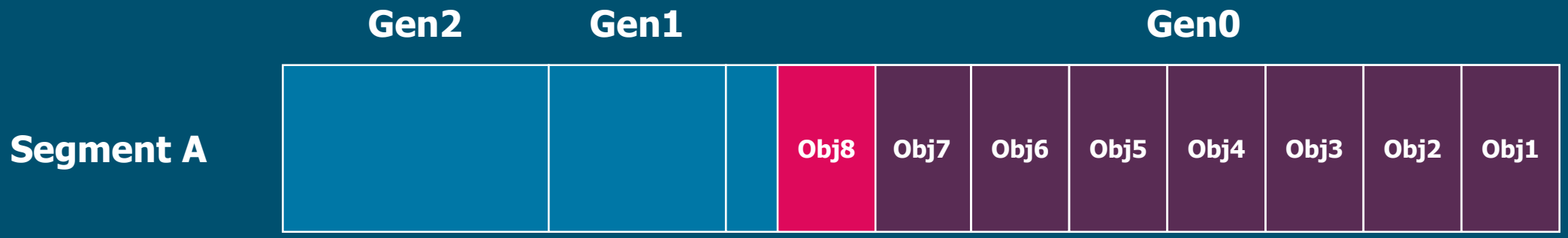

```
Obj8 = new A();
```



# Slow Allocation (Case 1)

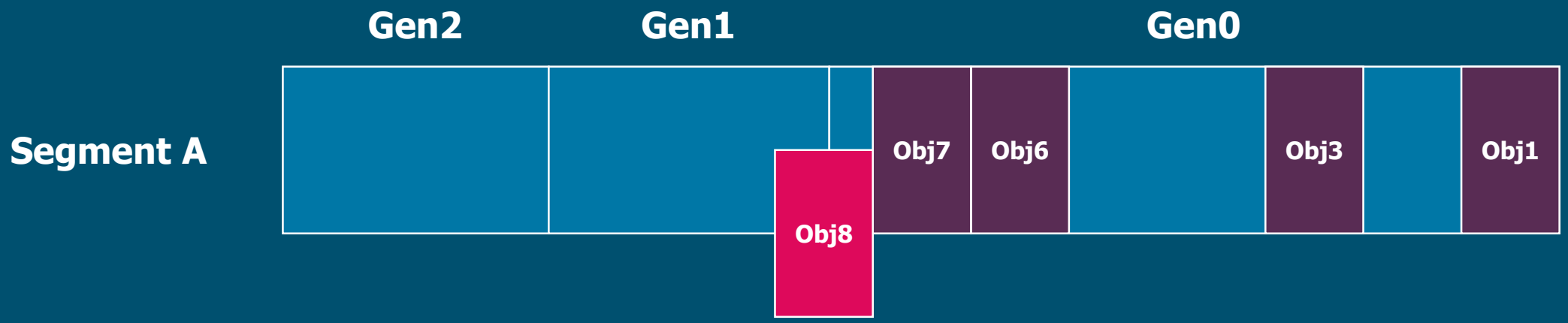
## Try to expand Gen0.

**(Possible if other generations are still empty.)**



# Slow Allocation (Case 2)

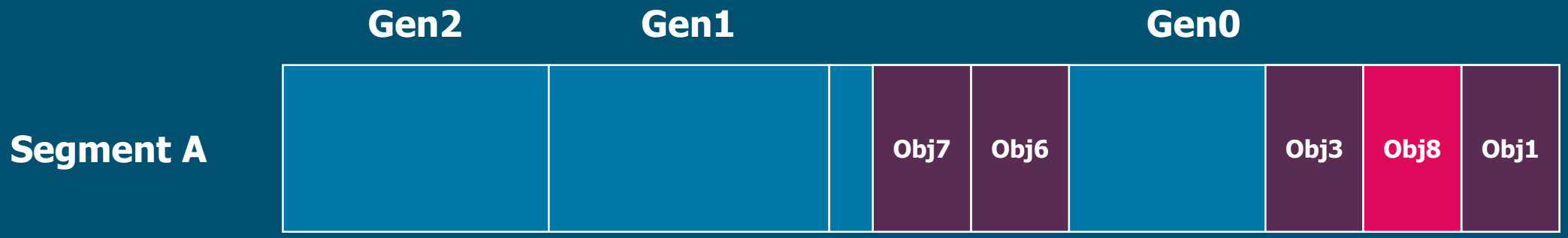

```
Obj8 = new A();
```



# Slow Allocation (Case 2)

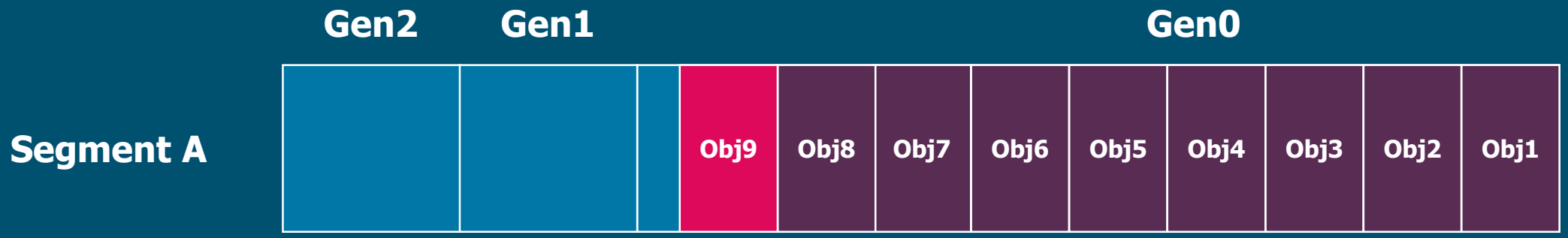
## Try to fit anywhere in Gen0.

**(Possible if Gen0 is already fragmented.)**



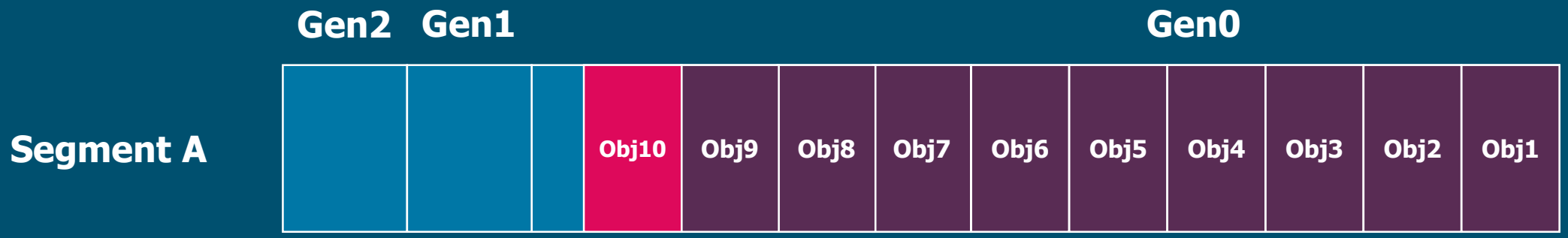
# Allocate More Objects

```
Obj9 = new A();
```



# Allocate More Objects

```
Obj10 = new A();
```



# Allocate More Objects

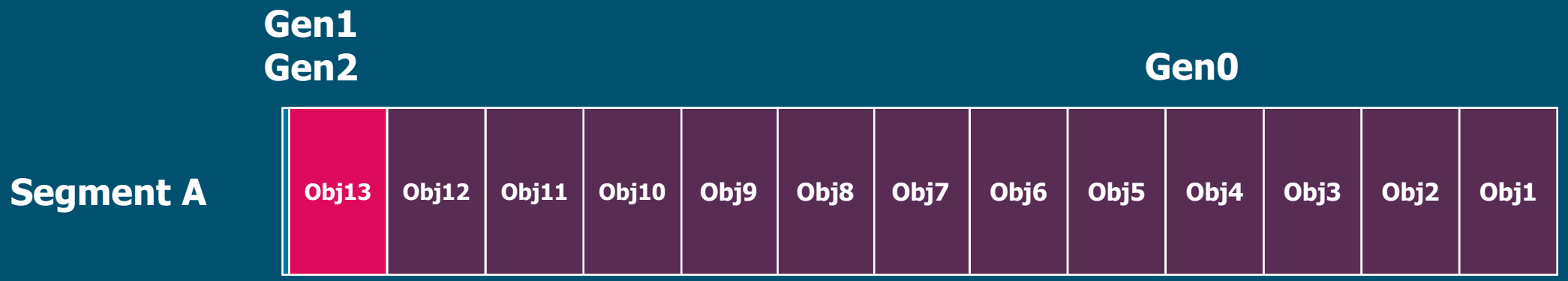
```
Obj11 = new A();
```

# Allocate More Objects

```
Obj12 = new A();
```

# Allocate More Objects

```
Obj13 = new A();
```



# Allocate More Objects

```
Obj14 = new A();
```

# Allocate More Objects

## Allocation will exceed segement size.

# Allocate More Objects

**Allocati… …ent size.**

**A**
***Garbage Collection***
**is triggered!**

# Recap

*The CLR tries 3 times to allocate new objects.*

**Fast Allocation**

1. *Try to allocate at the end of Gen0.*

**Slow Allocation**

2. *Try to expand Gen0 and allocate at the new end.*
3. *Try to allocate anywhere whithin Gen0 if possible.*

# Recap

## If allocation exceeds the segment size a garbage collection occurs.

*There are other reasons for garbage collections to occur depending on internal metrics and dynamic thresholds of the GC.*

# Garbage Collection Phases

**Suspend**

**Mark**

**Compact**

**Resume**

# Garbage Collection Phases

**Suspend**

*Suspend all managed threads.*

**Mark**

**Compact**

**Resume**

# Garbage Collection Phases

**Suspend**

***Mark***

*Mark all objects transitively referenced by any GC Root.*

**Compact**

**Resume**

# Garbage Collection Phases

**Suspend**

**Mark**

**Compact**

*Relocate and promote marked objects.*

**Resume**

# Garbage Collection Phases

**Suspend**

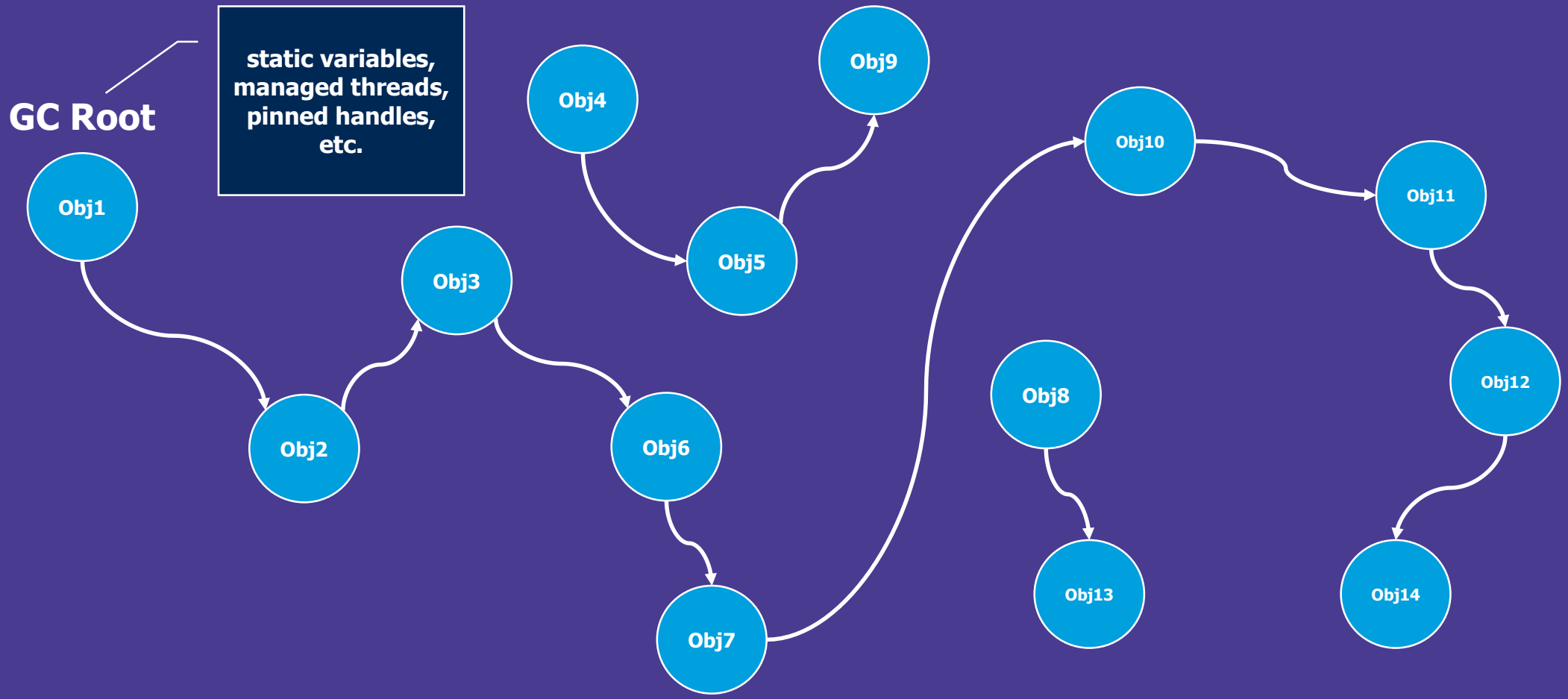
**Mark**

**Compact**

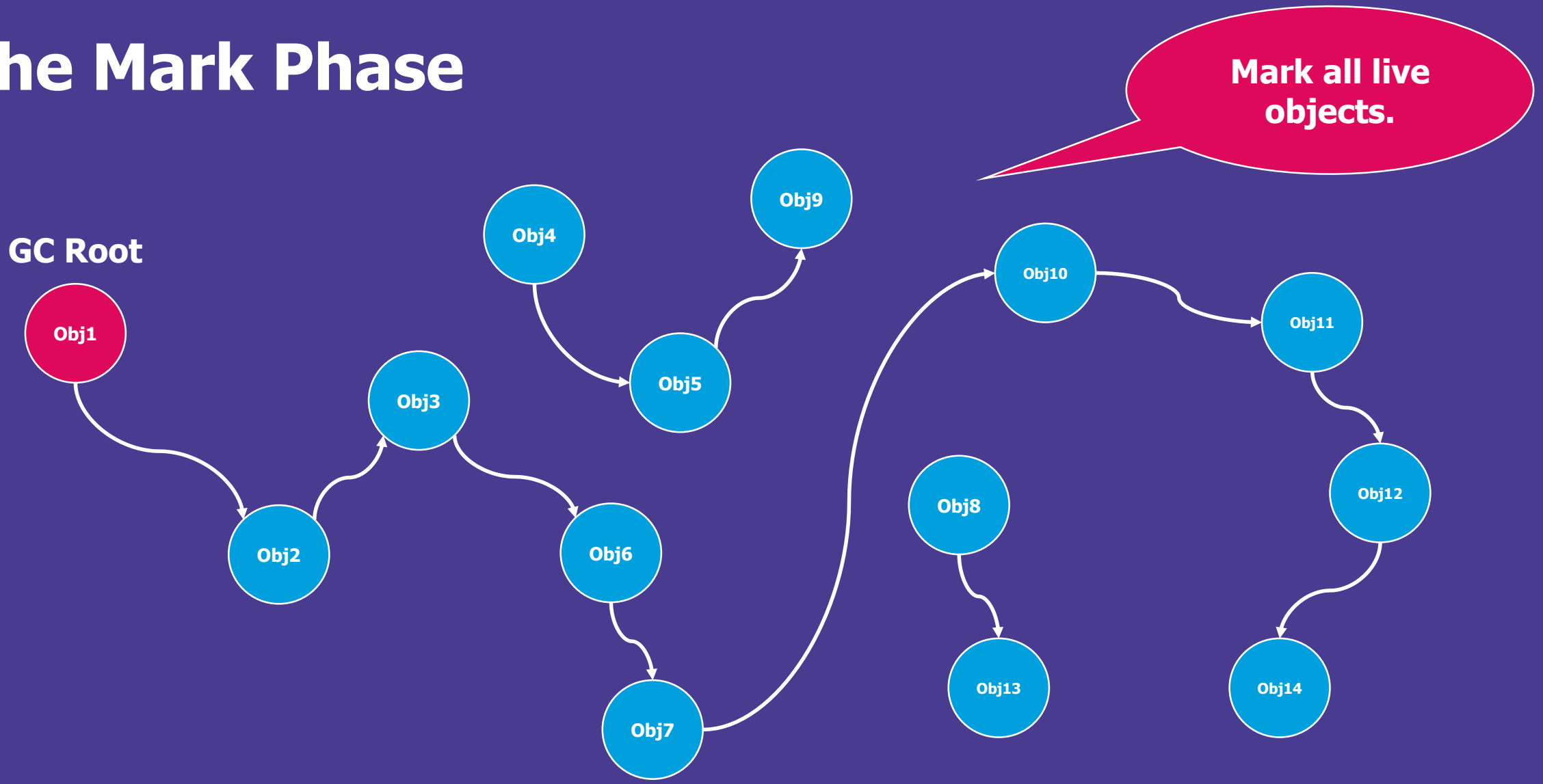
**Resume**

*Resume all suspended threads.*

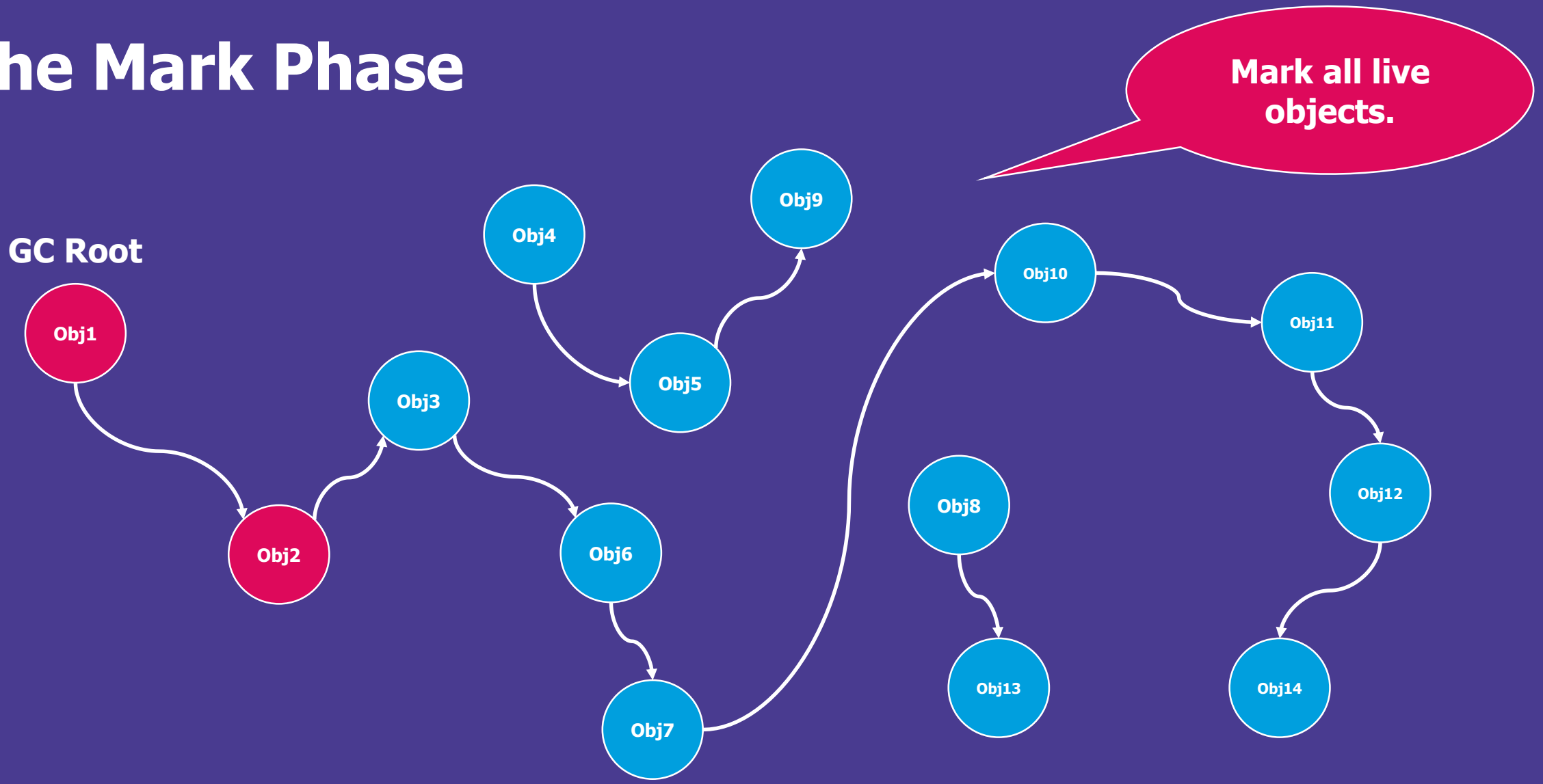
# The Mark Phase



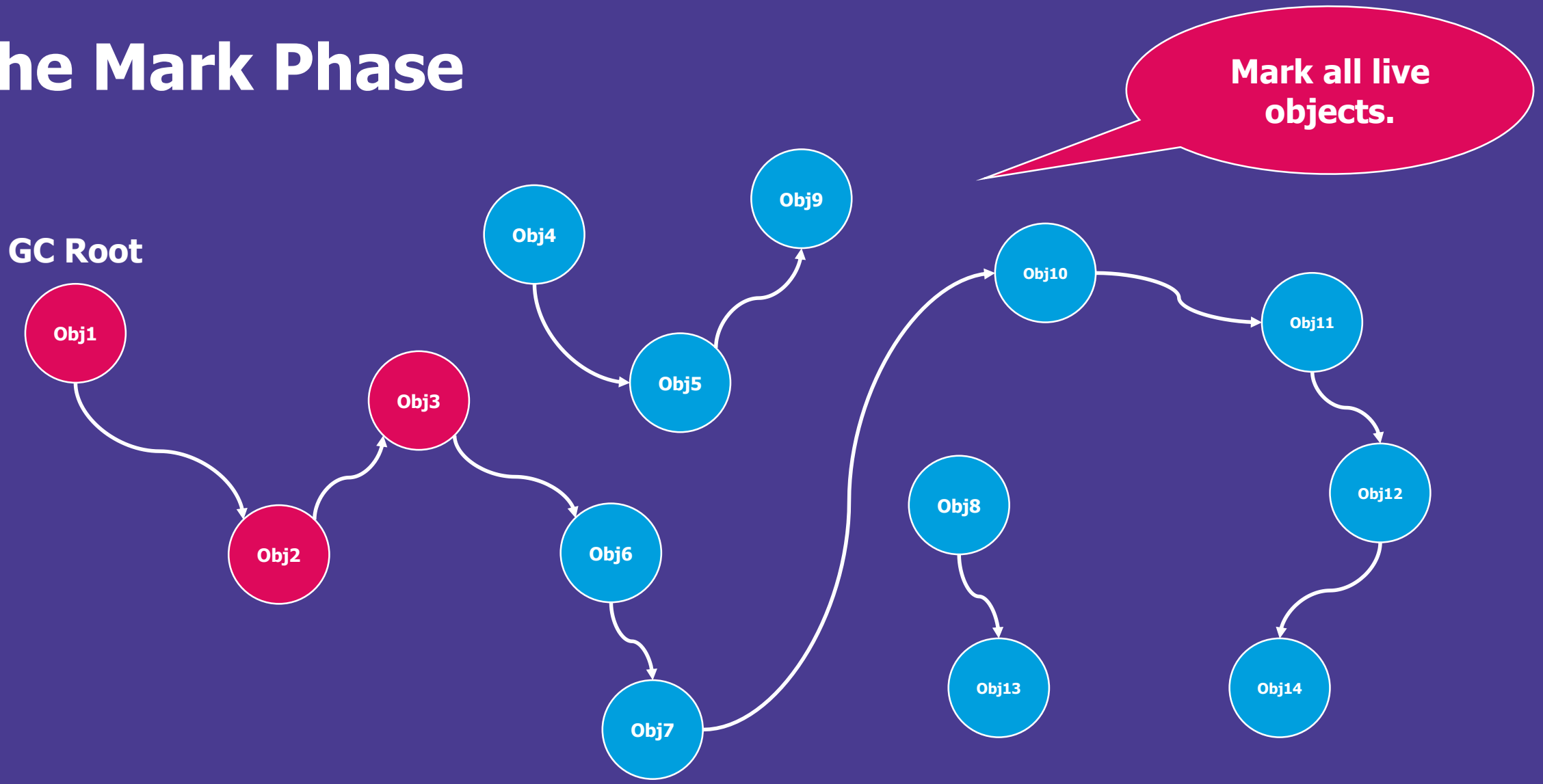
# The Mark Phase



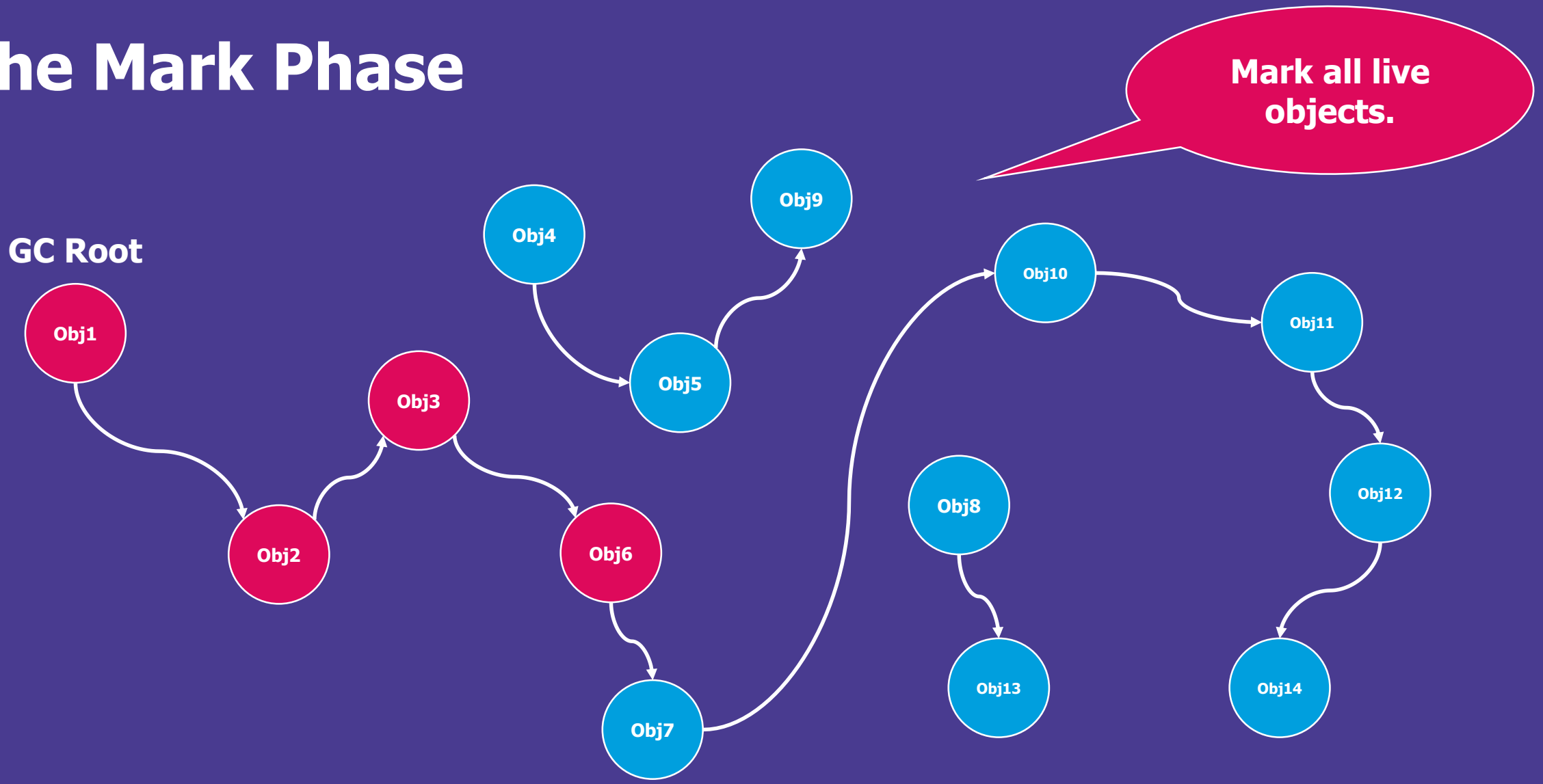
# The Mark Phase



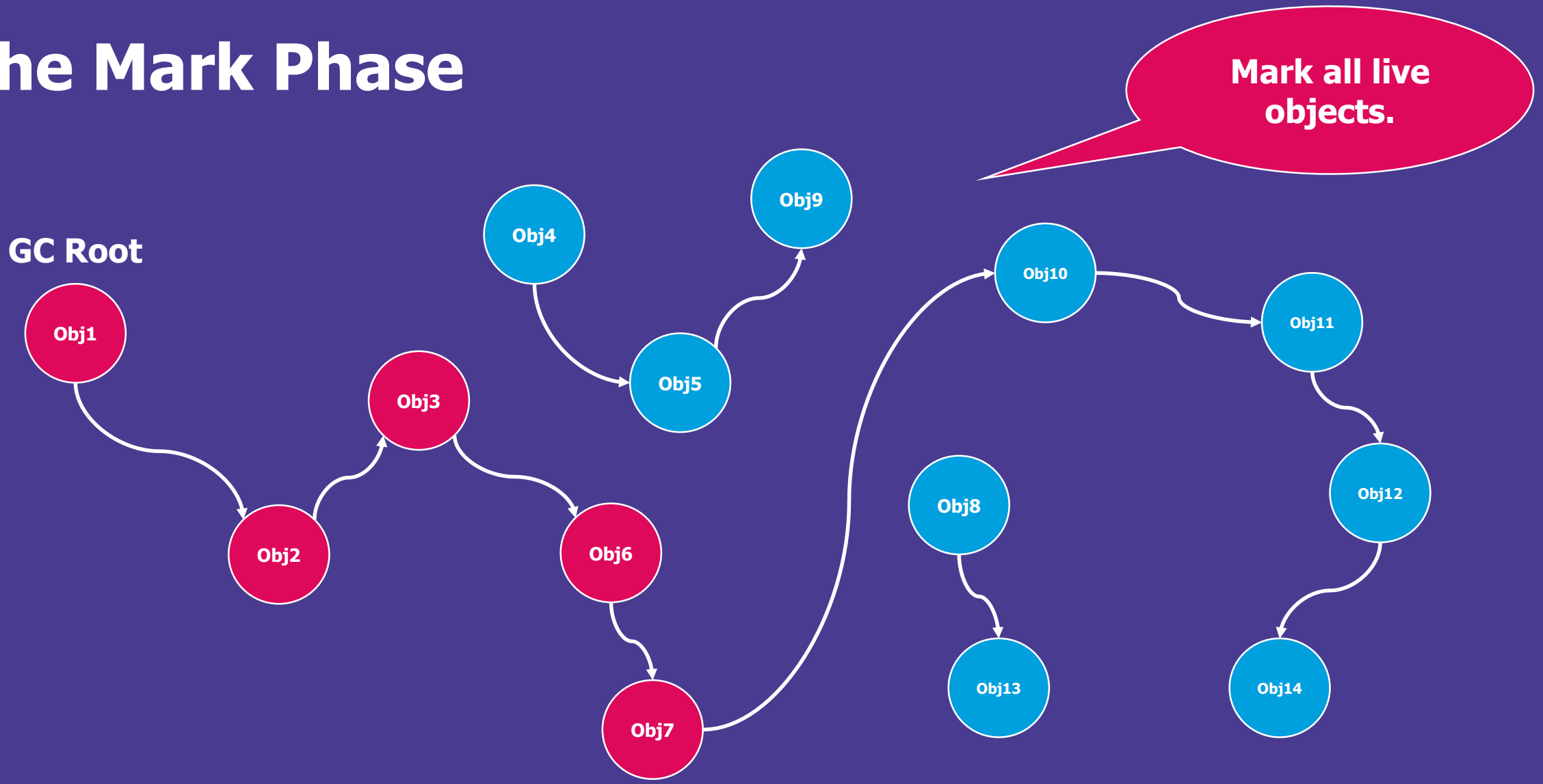
# The Mark Phase



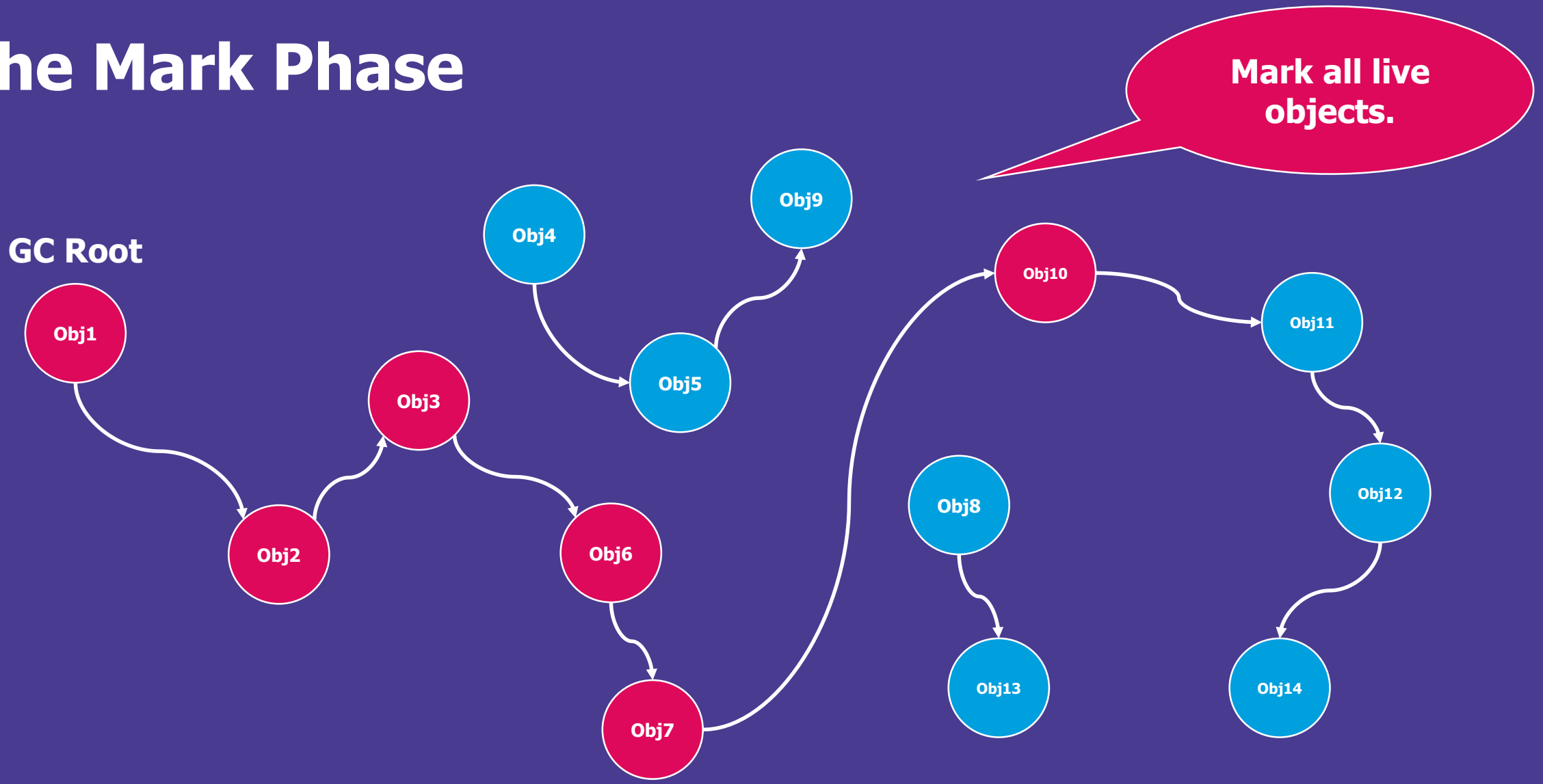
# The Mark Phase



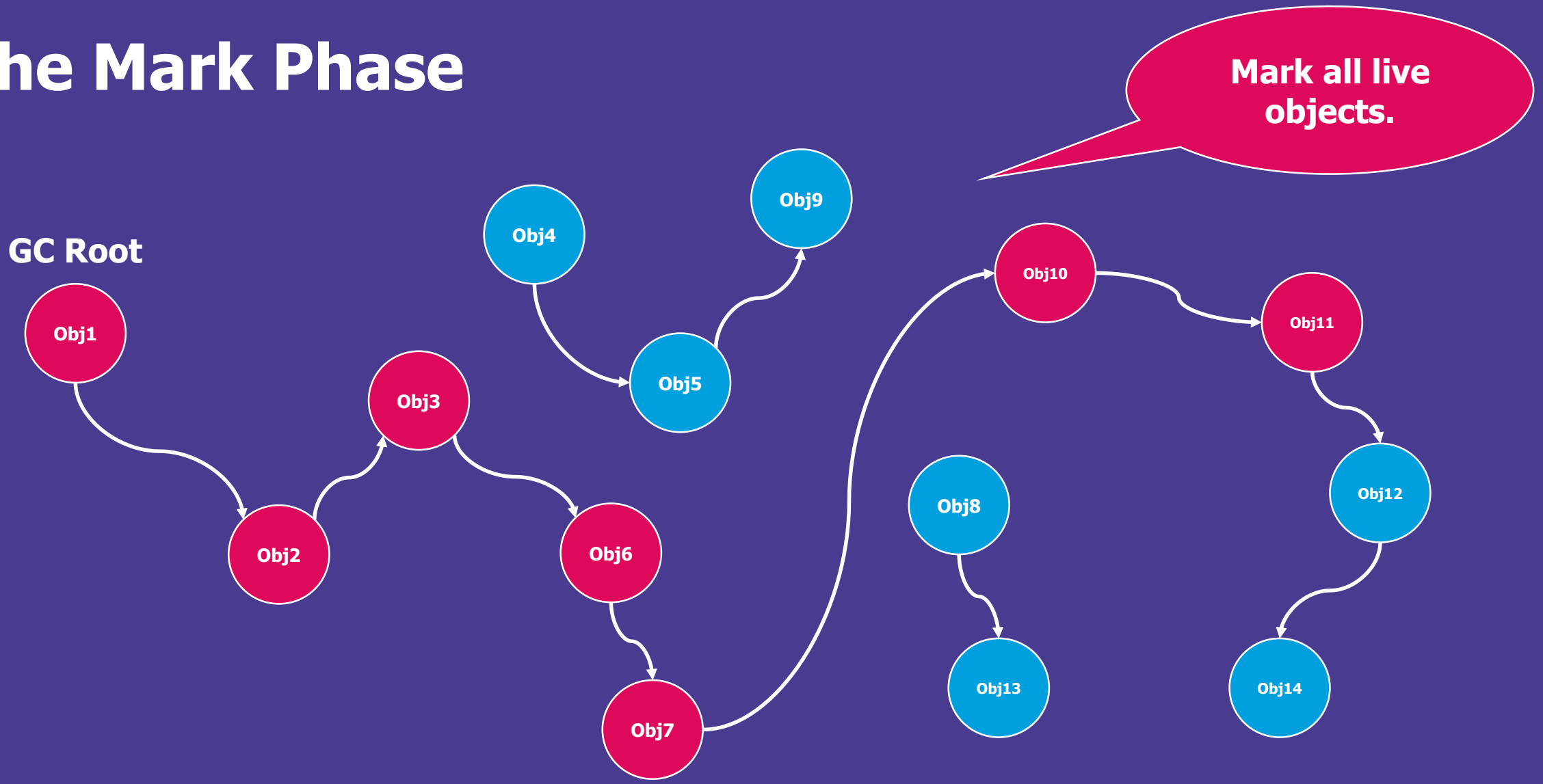
# The Mark Phase



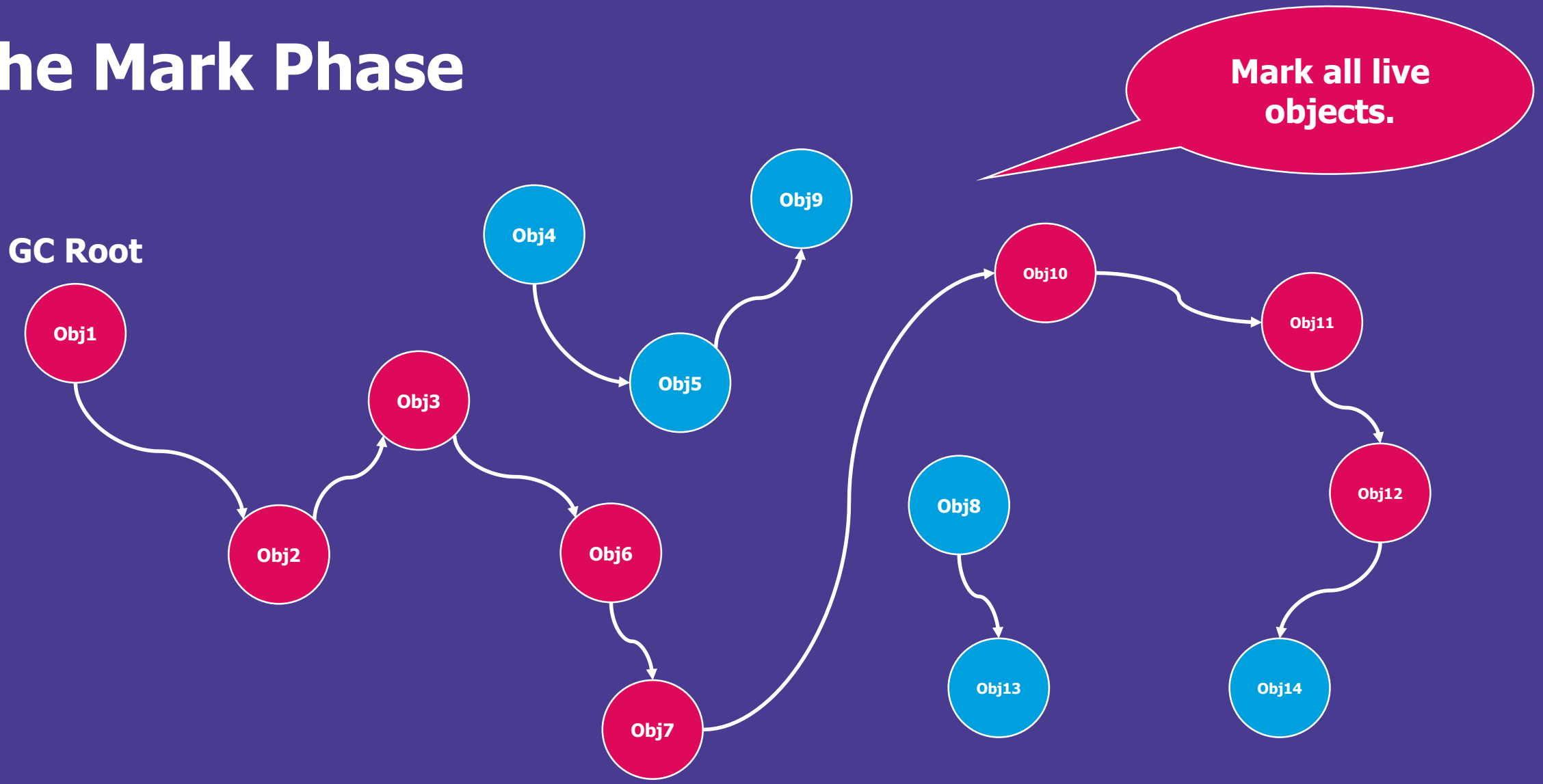
# The Mark Phase



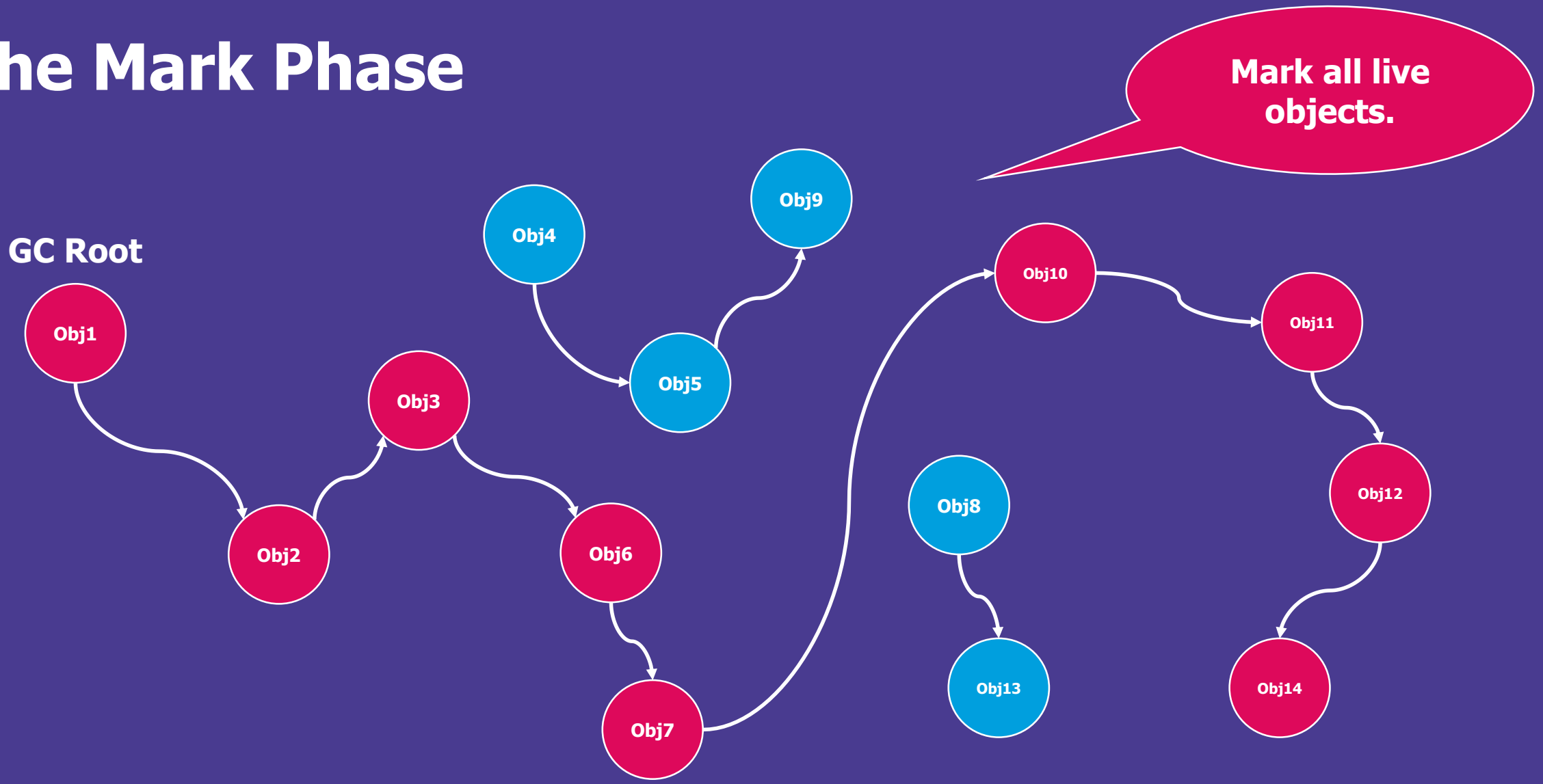
# The Mark Phase



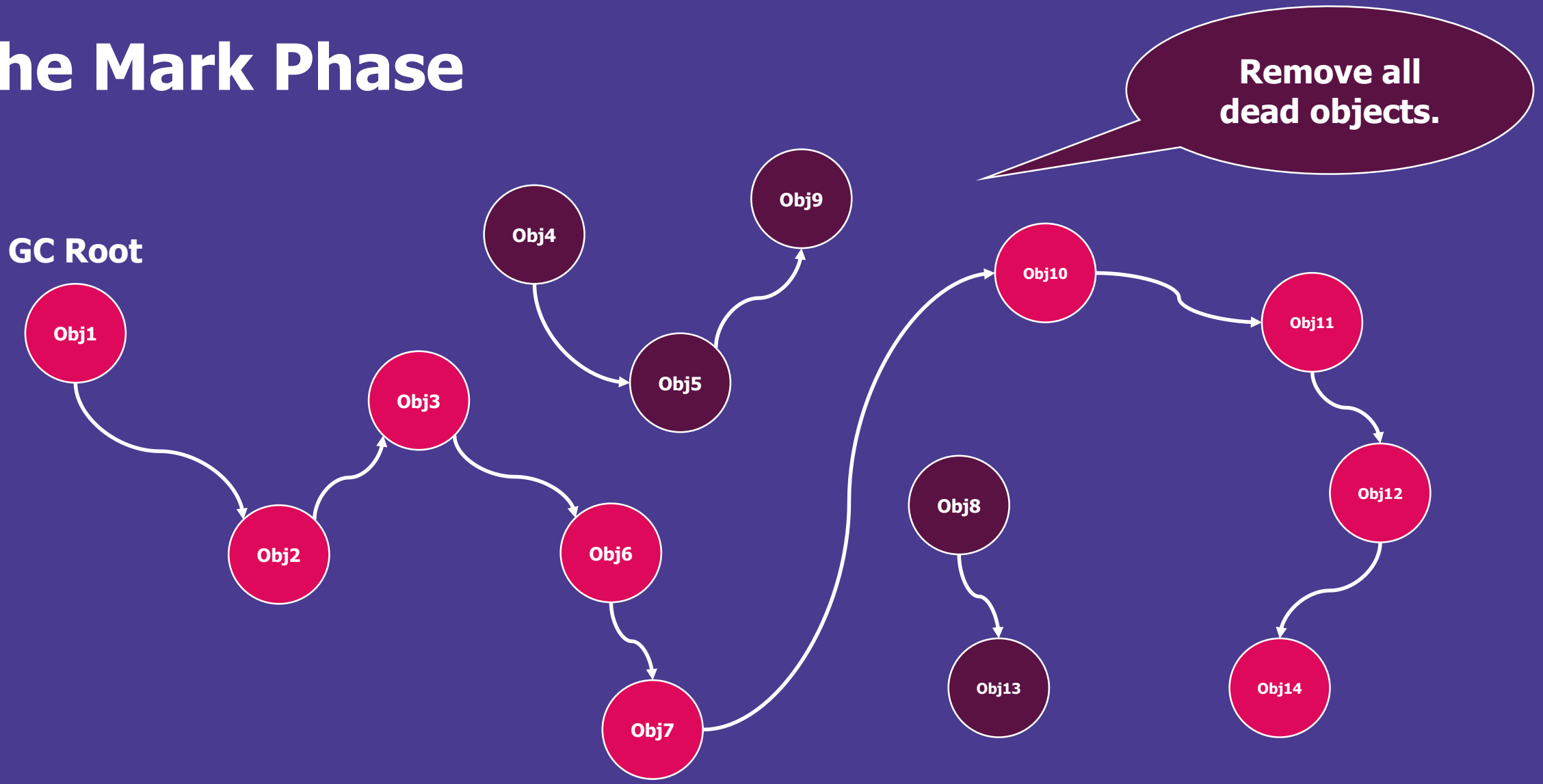
# The Mark Phase



# The Mark Phase

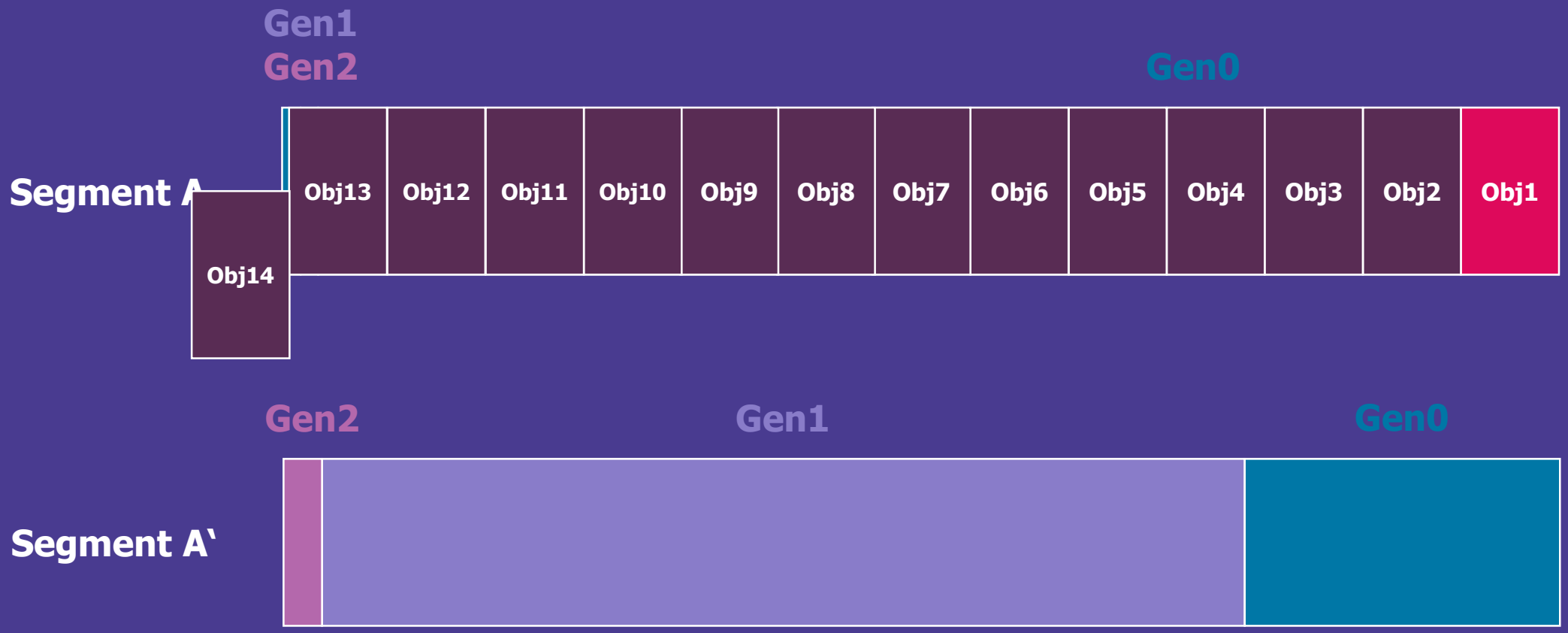


# The Mark Phase



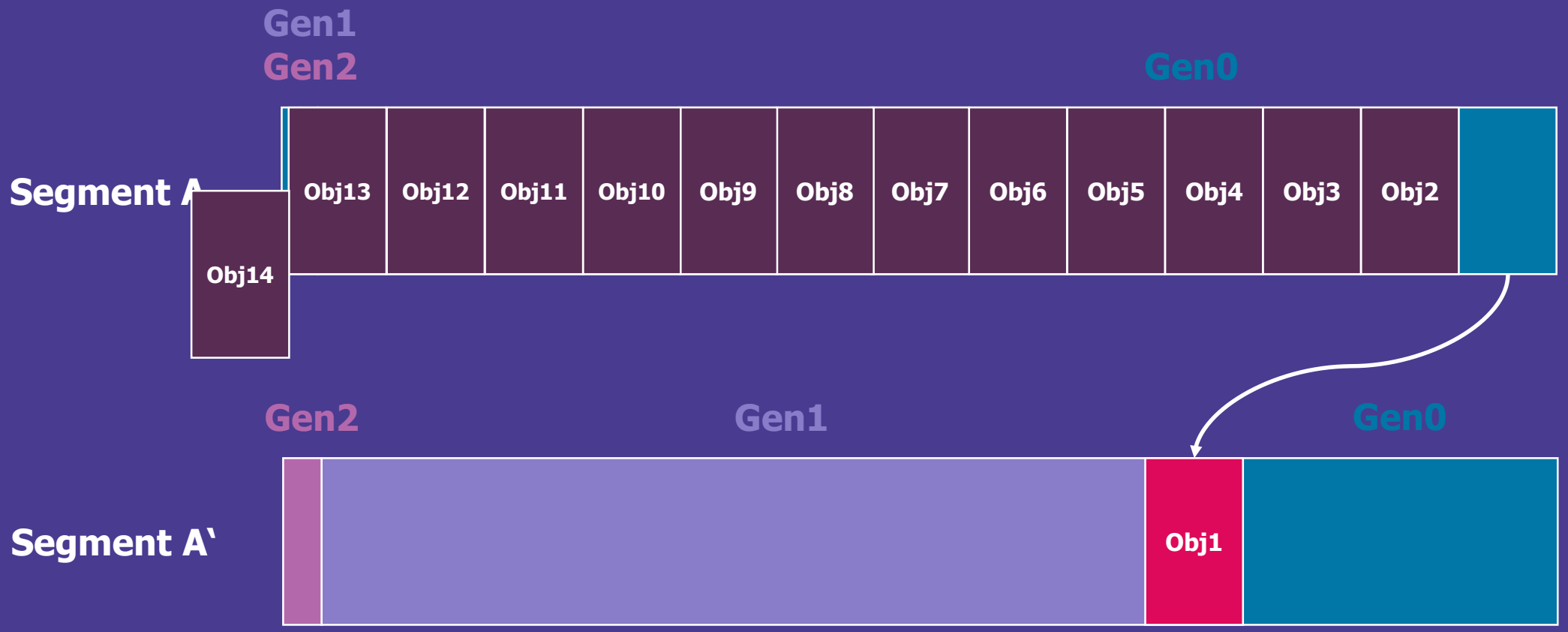
# The Compact Phase

Relocate and promote live objects.



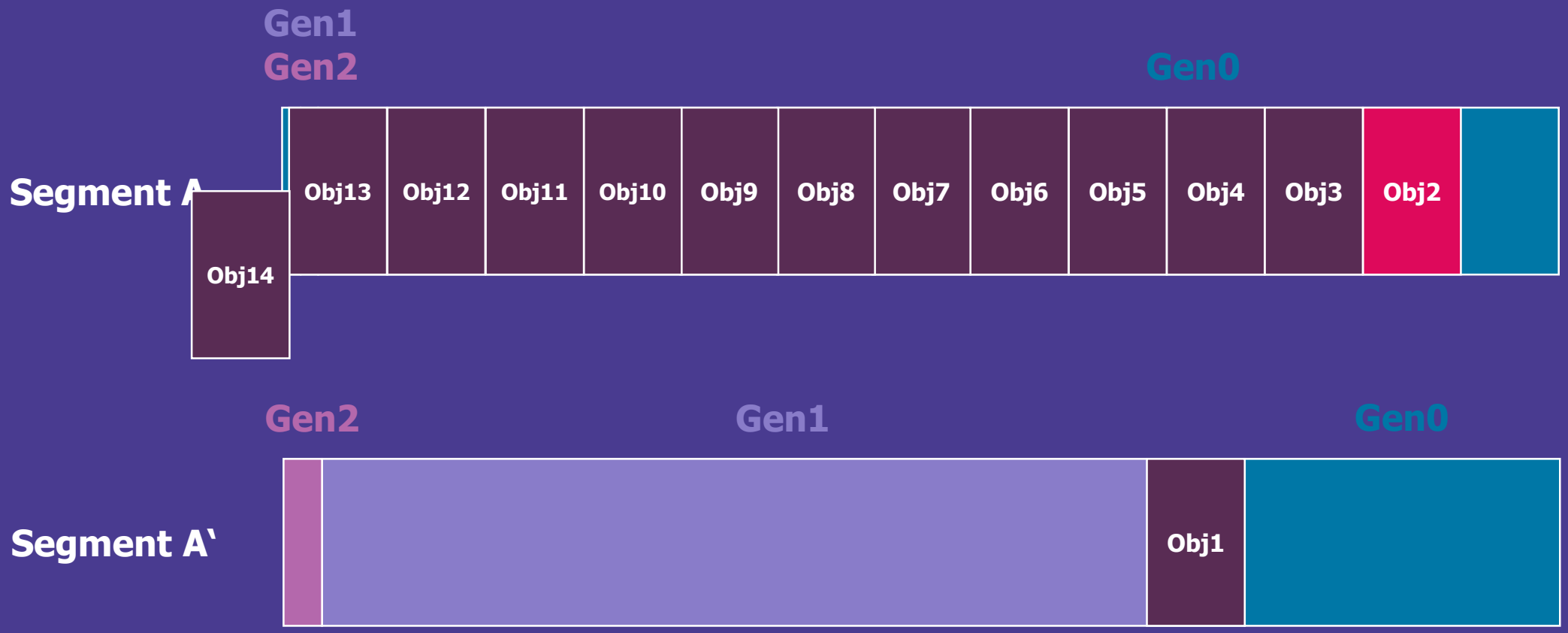
# The Compact Phase

Relocate and promote live objects.



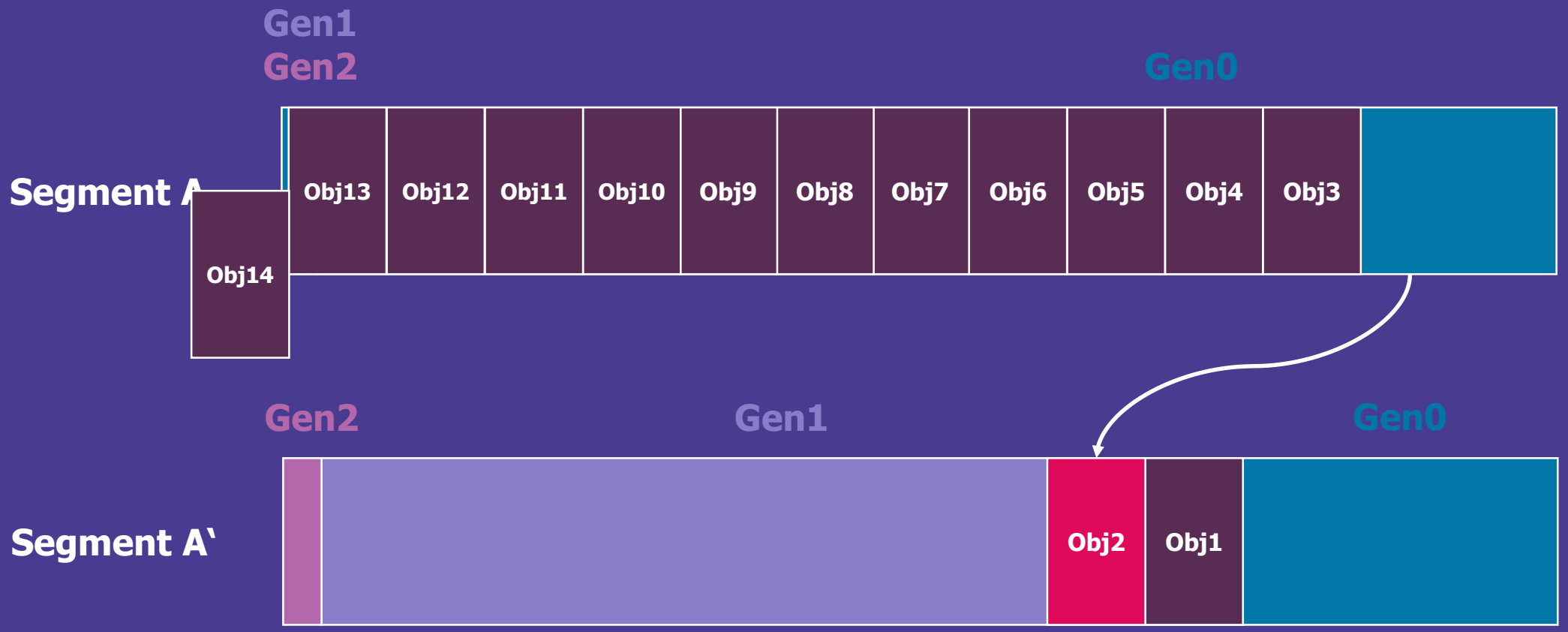
# The Compact Phase

Relocate and promote live objects.



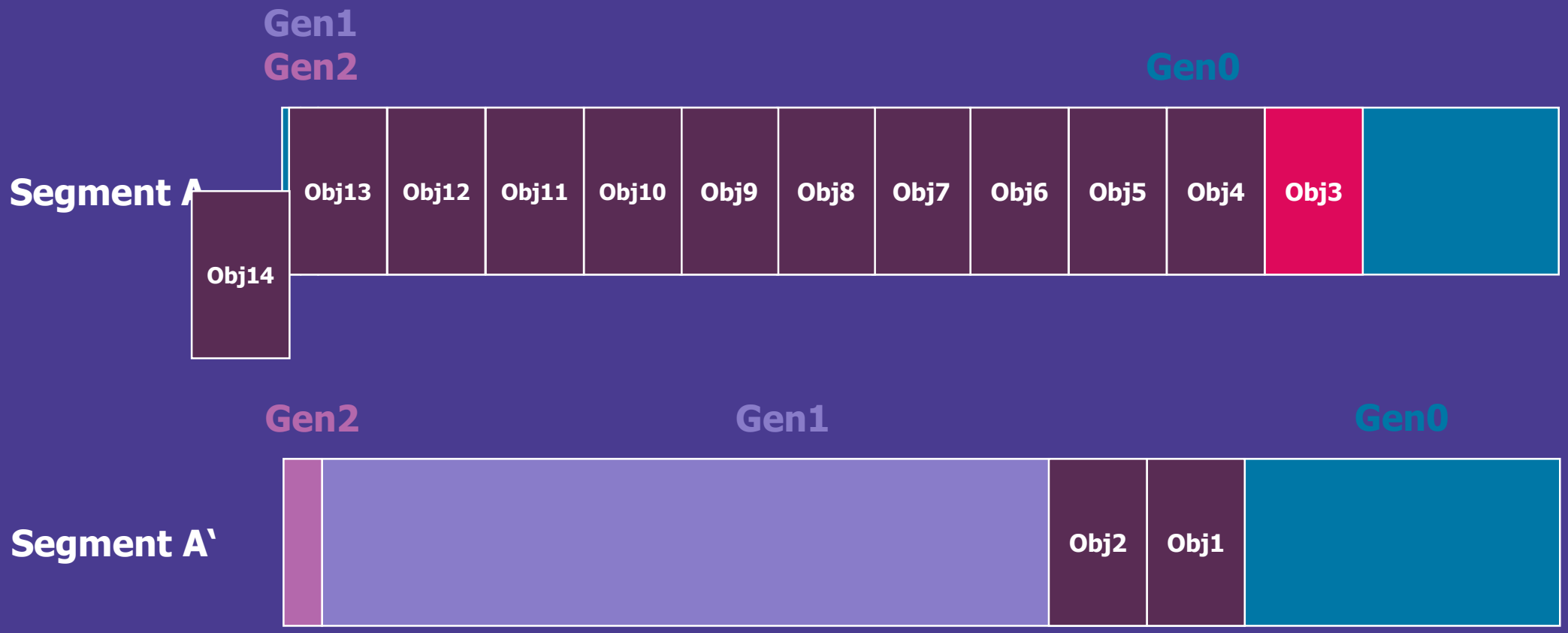
# The Compact Phase

Relocate and promote live objects.



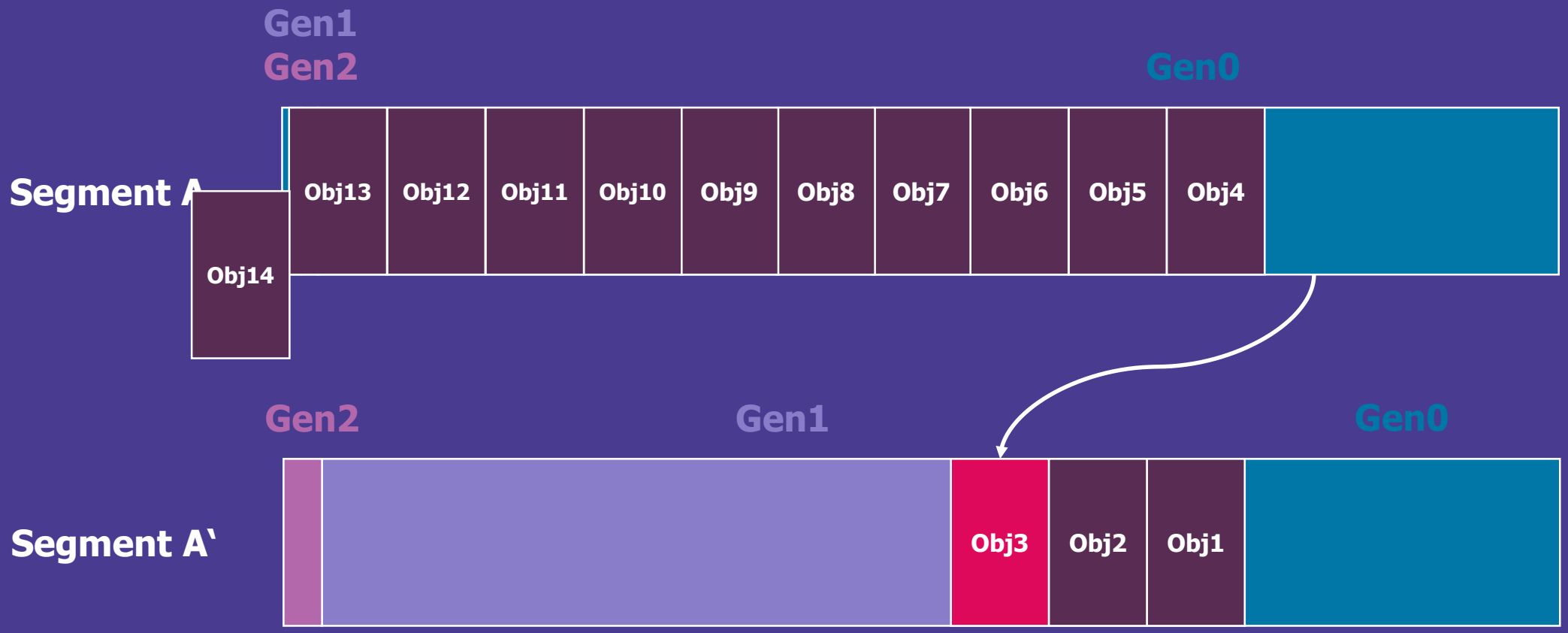
# The Compact Phase

Relocate and promote live objects.



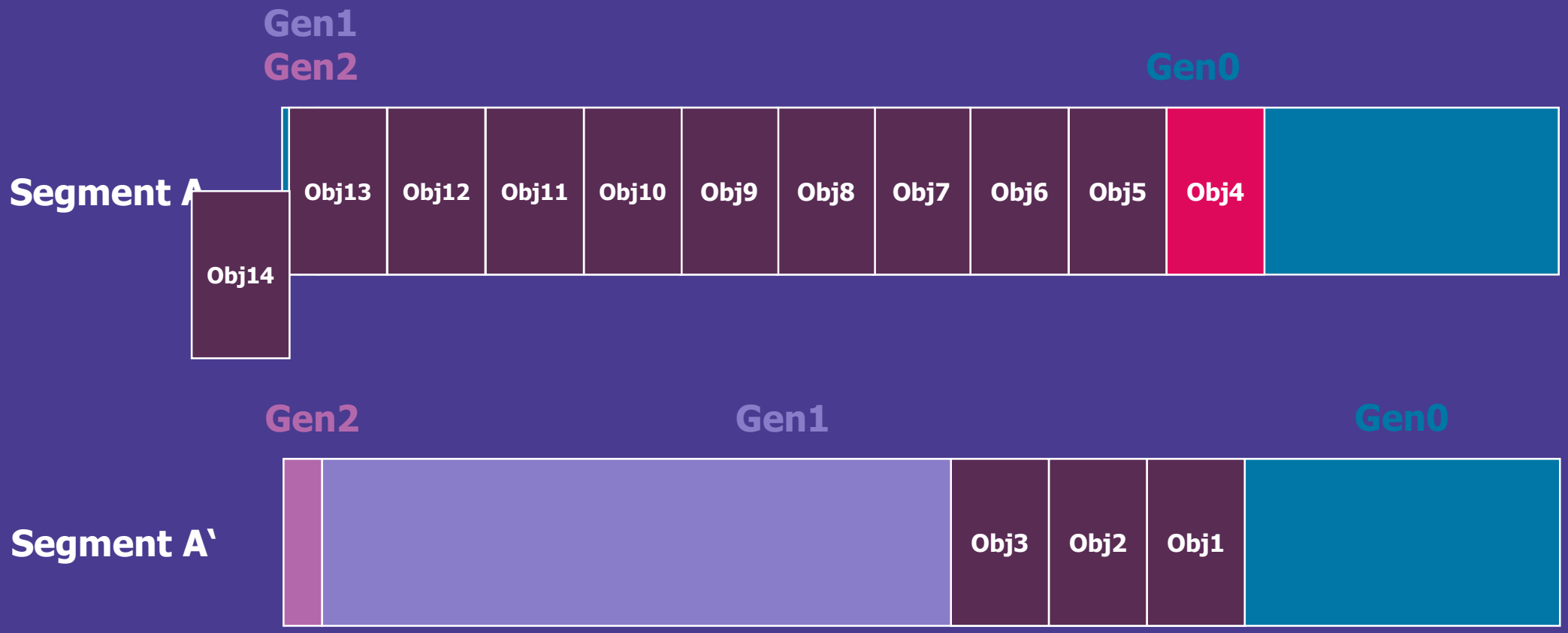
# The Compact Phase

Relocate and promote live objects.



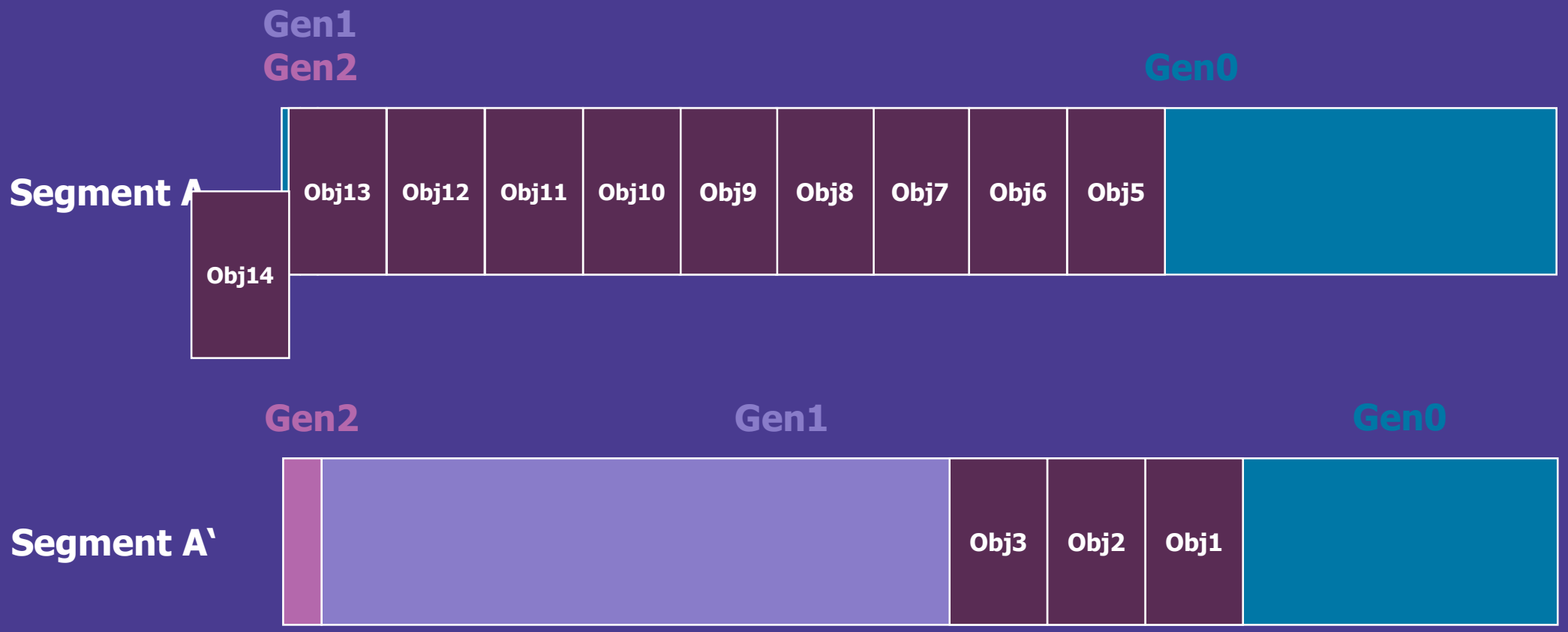
# The Compact Phase

Relocate and promote live objects.



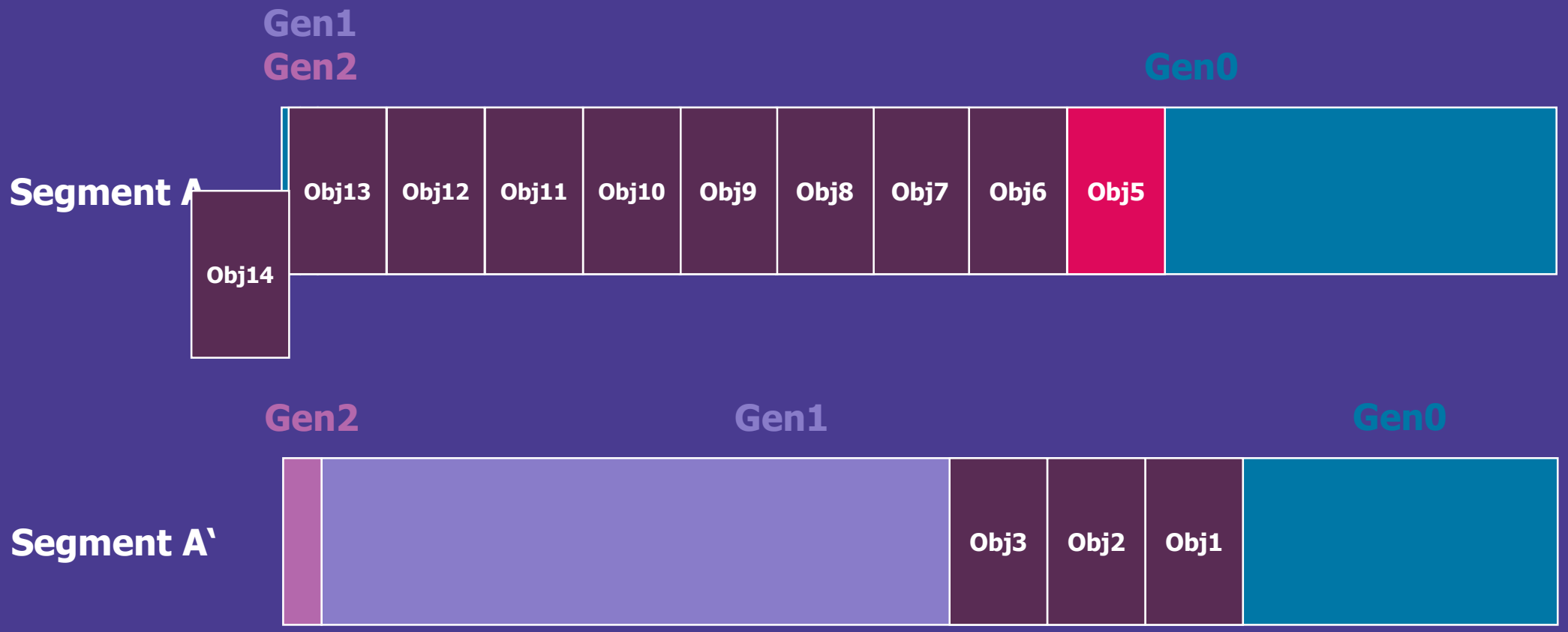
# The Compact Phase

Relocate and promote live objects.



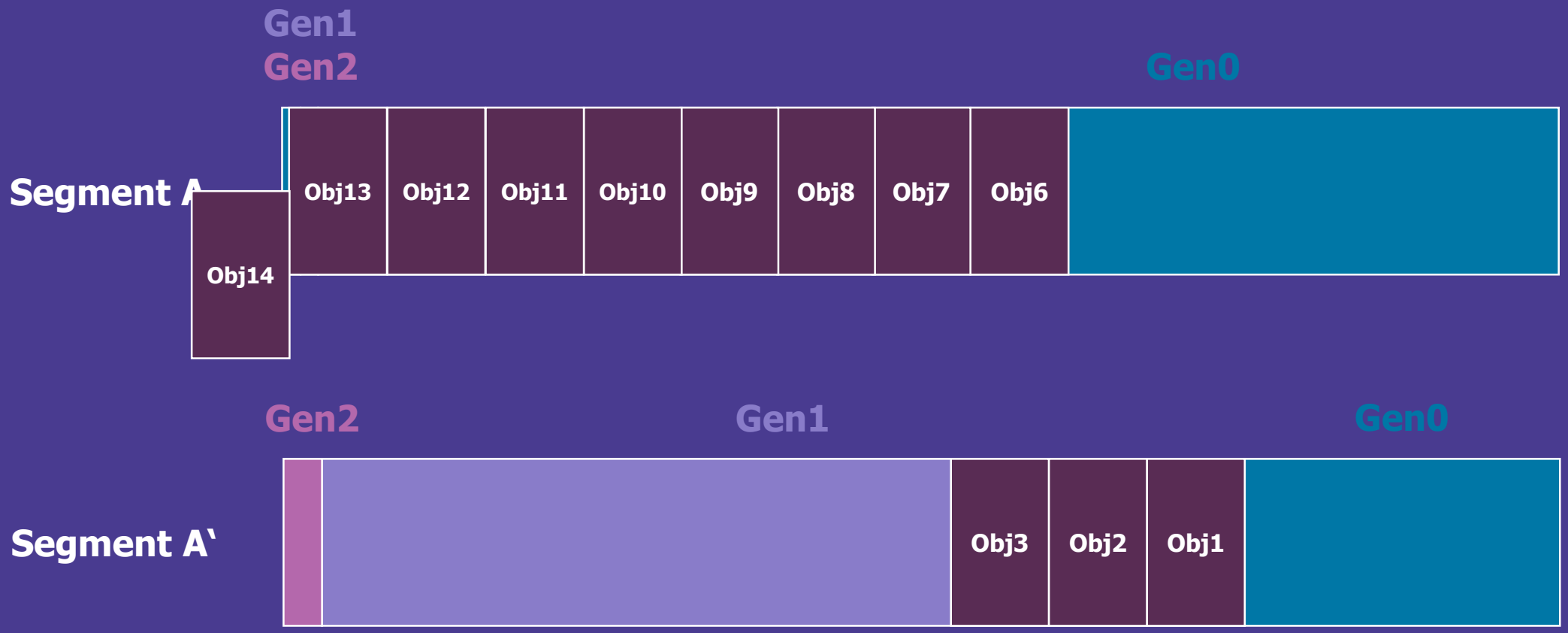
# The Compact Phase

Relocate and promote live objects.



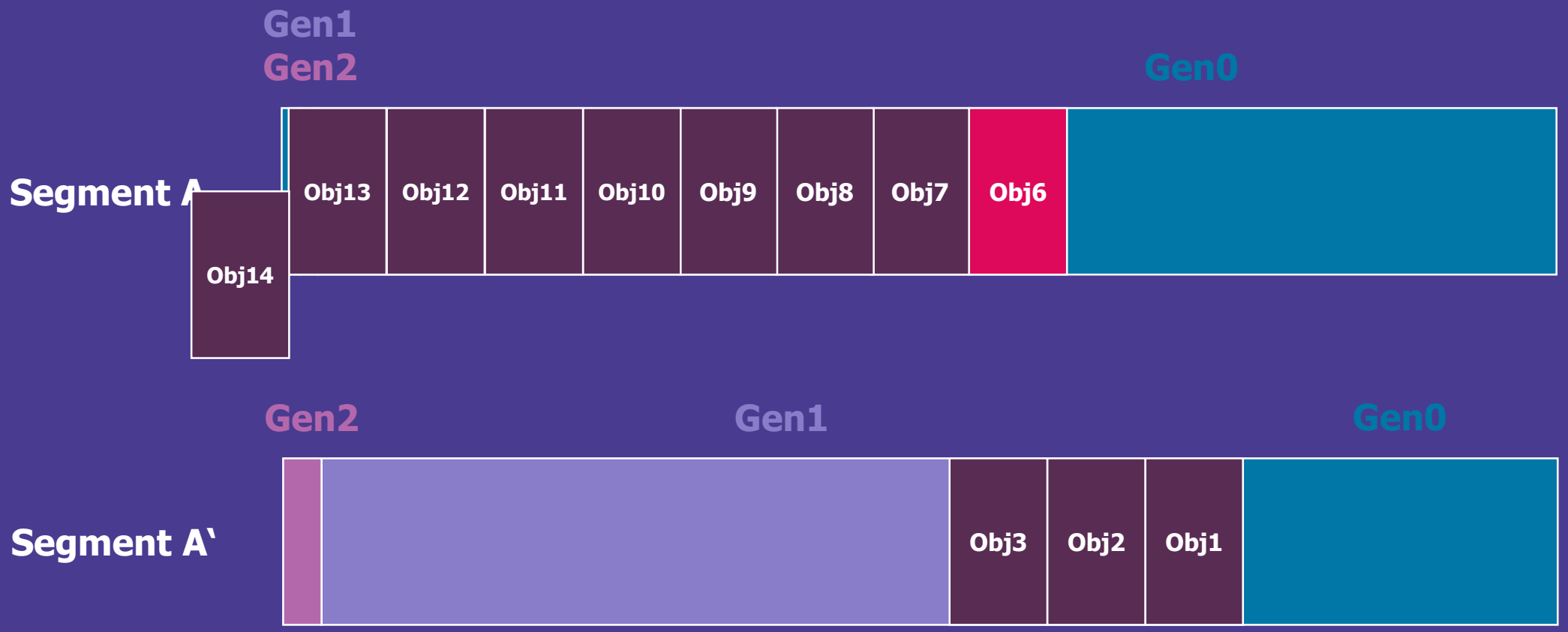
# The Compact Phase

Relocate and promote live objects.



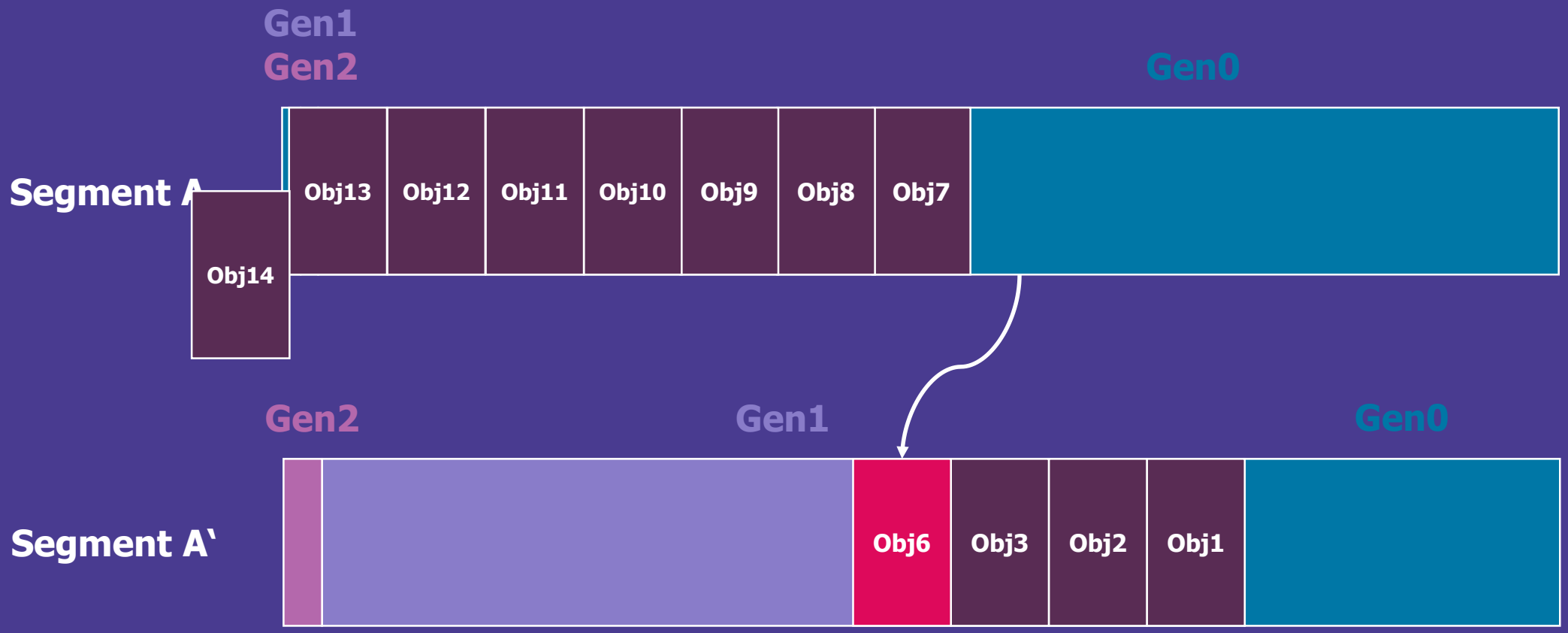
# The Compact Phase

Relocate and promote live objects.



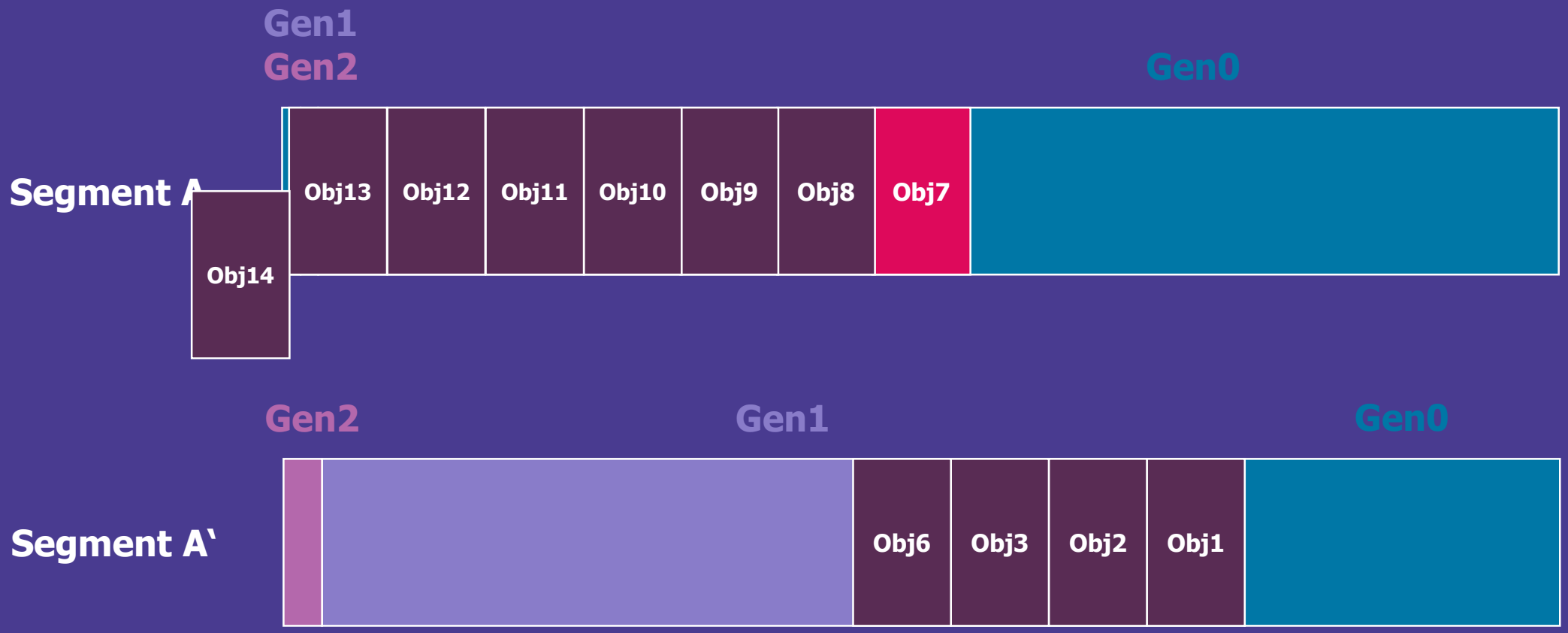
# The Compact Phase

Relocate and promote live objects.



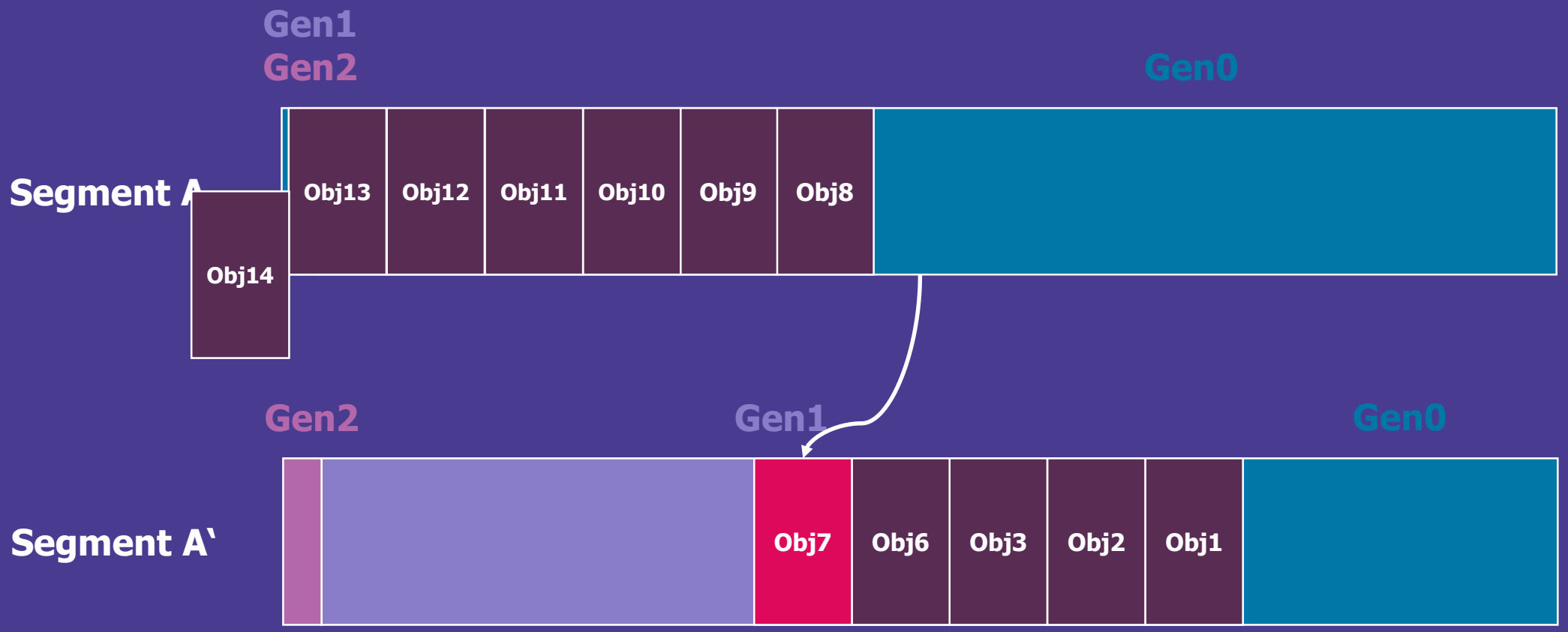
# The Compact Phase

Relocate and promote live objects.



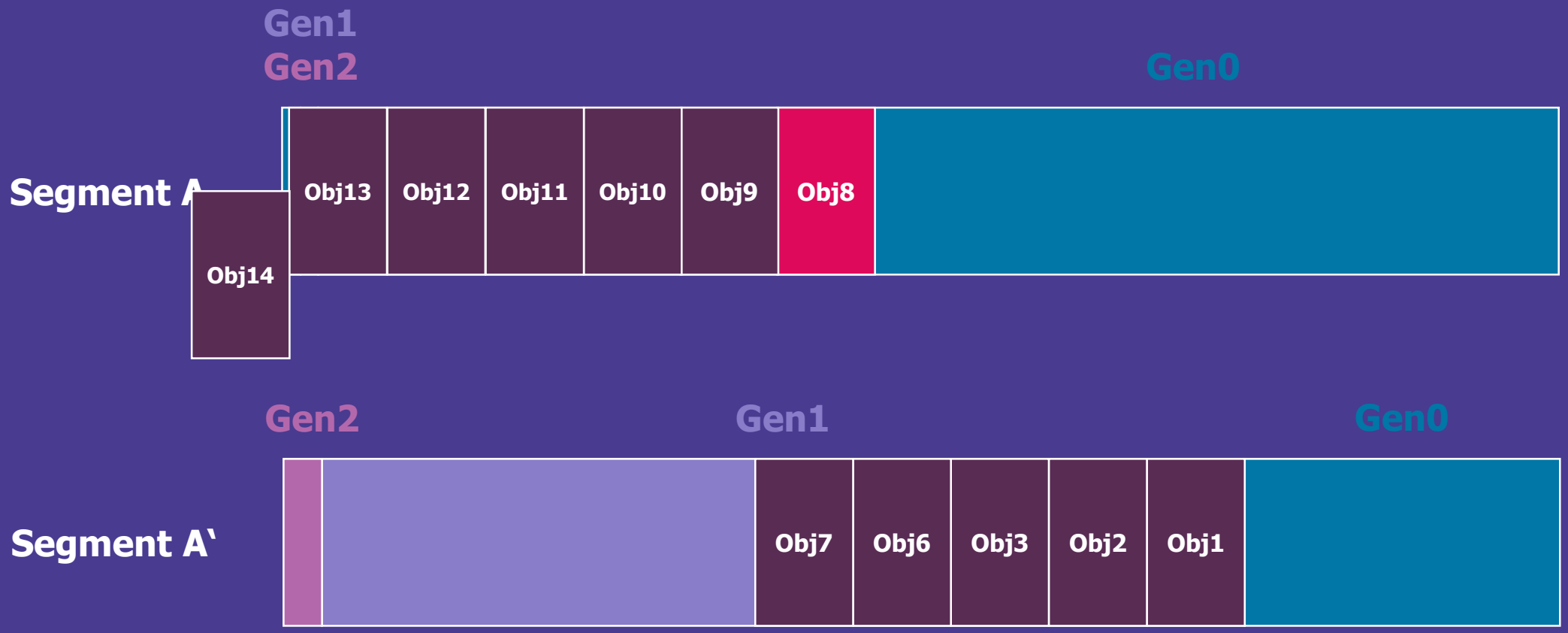
# The Compact Phase

Relocate and promote live objects.



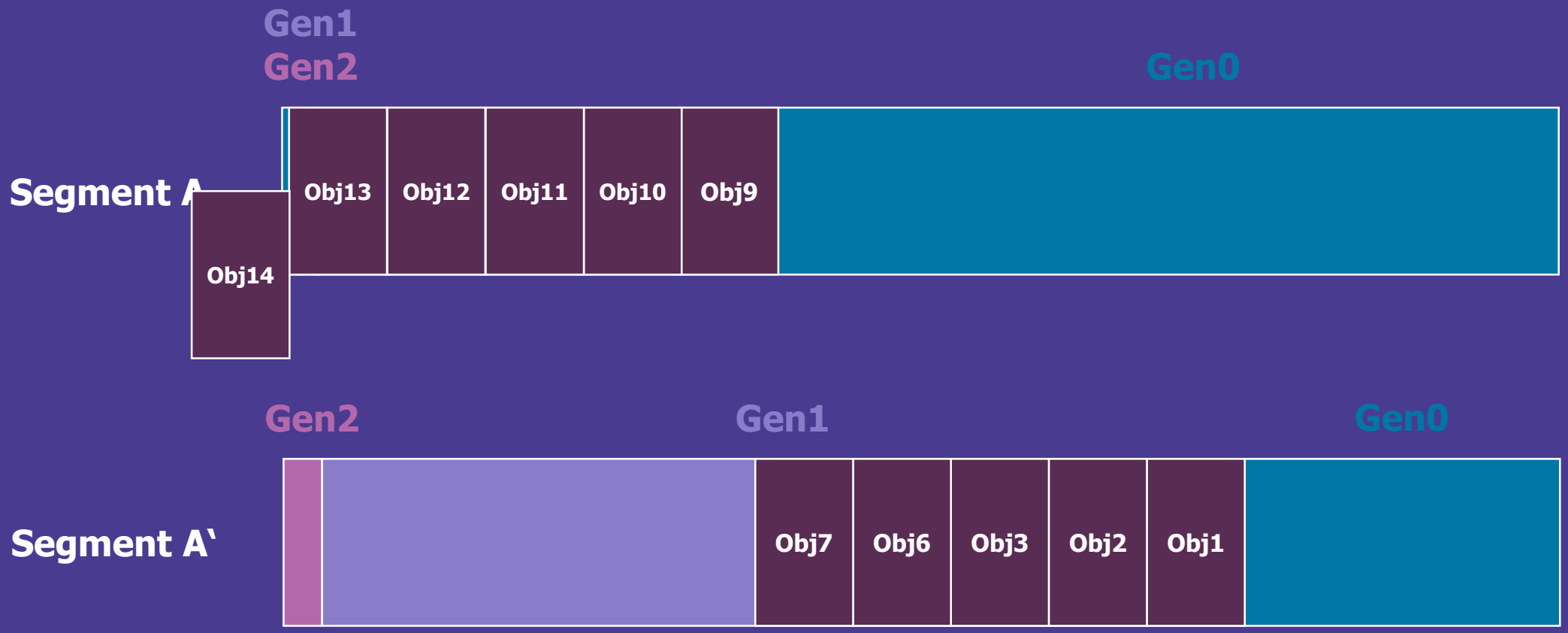
# The Compact Phase

Relocate and promote live objects.



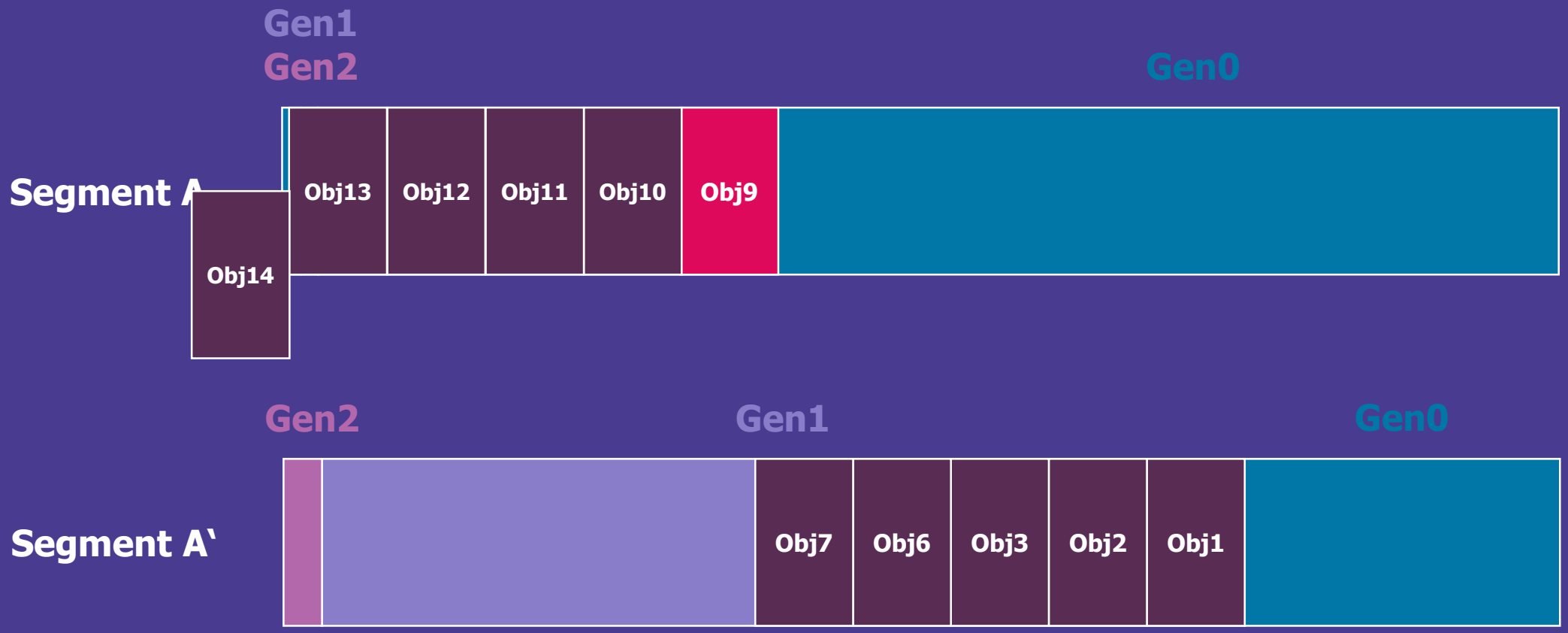
# The Compact Phase

Relocate and promote live objects.



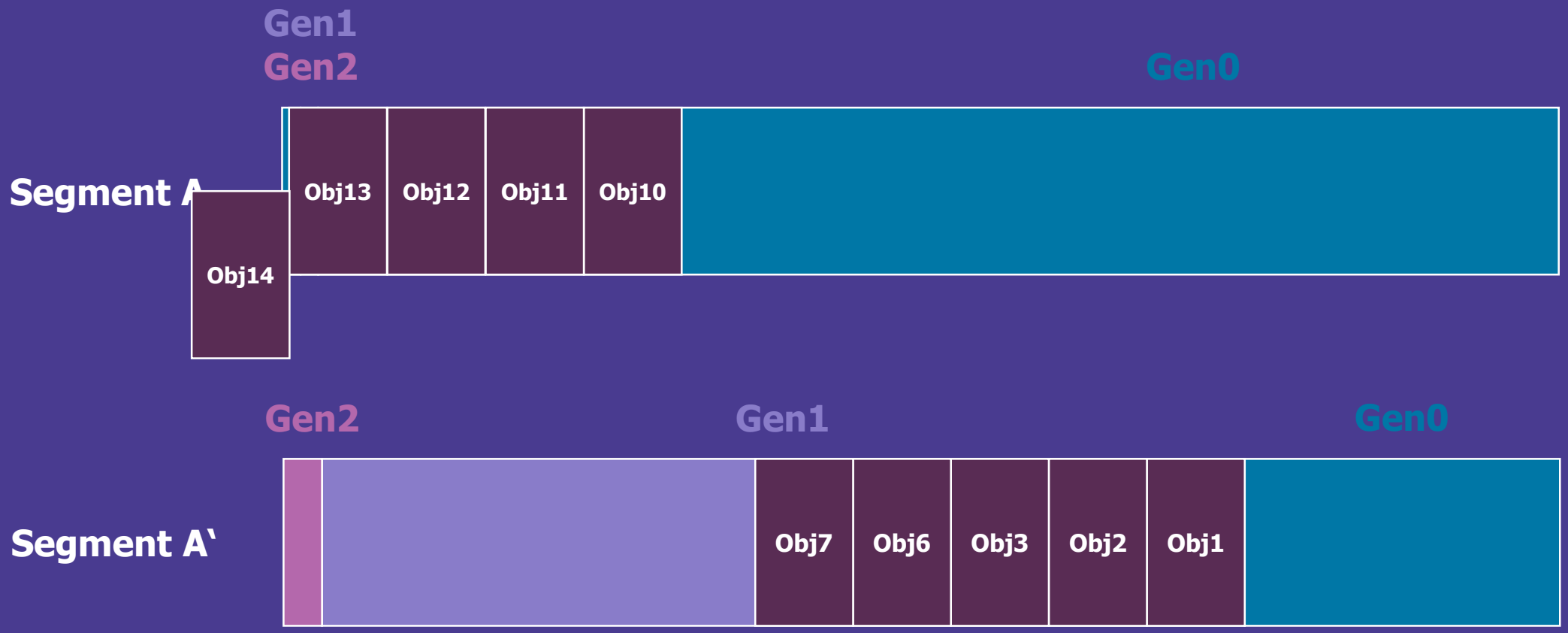
# The Compact Phase

Relocate and promote live objects.



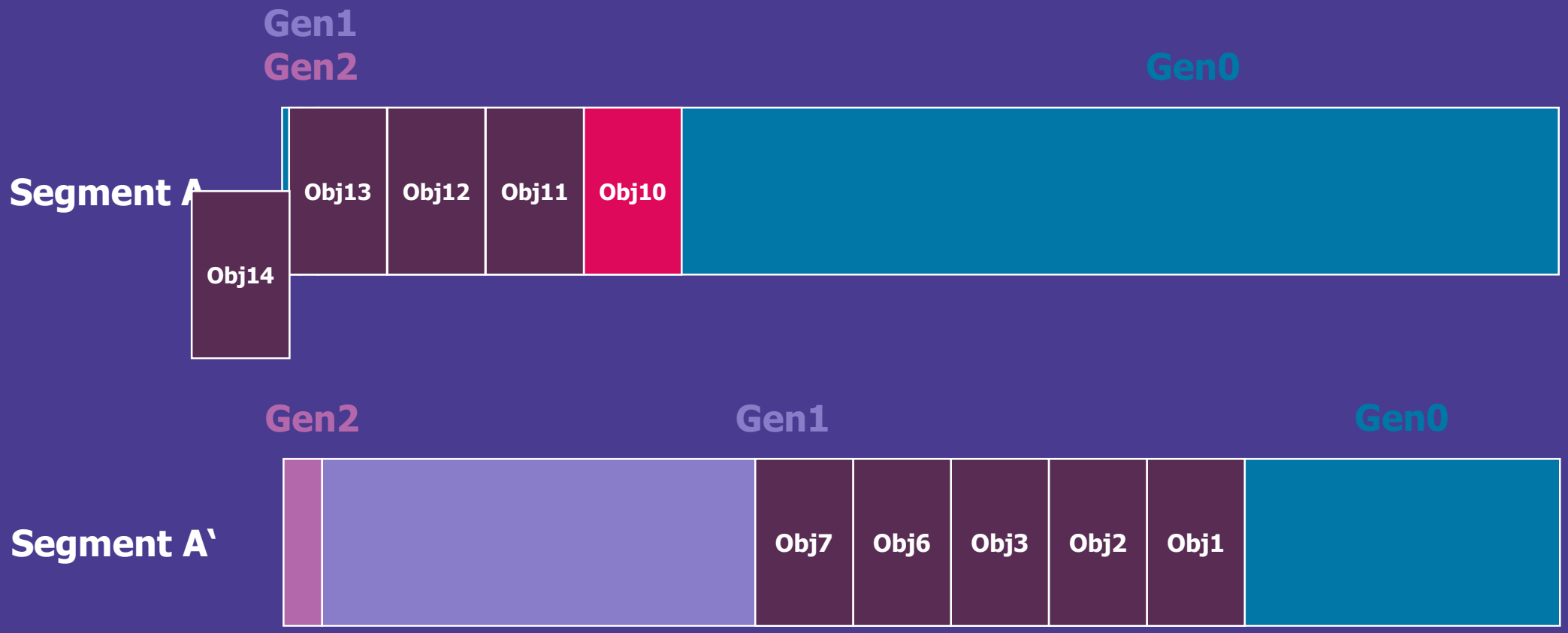
# The Compact Phase

Relocate and promote live objects.



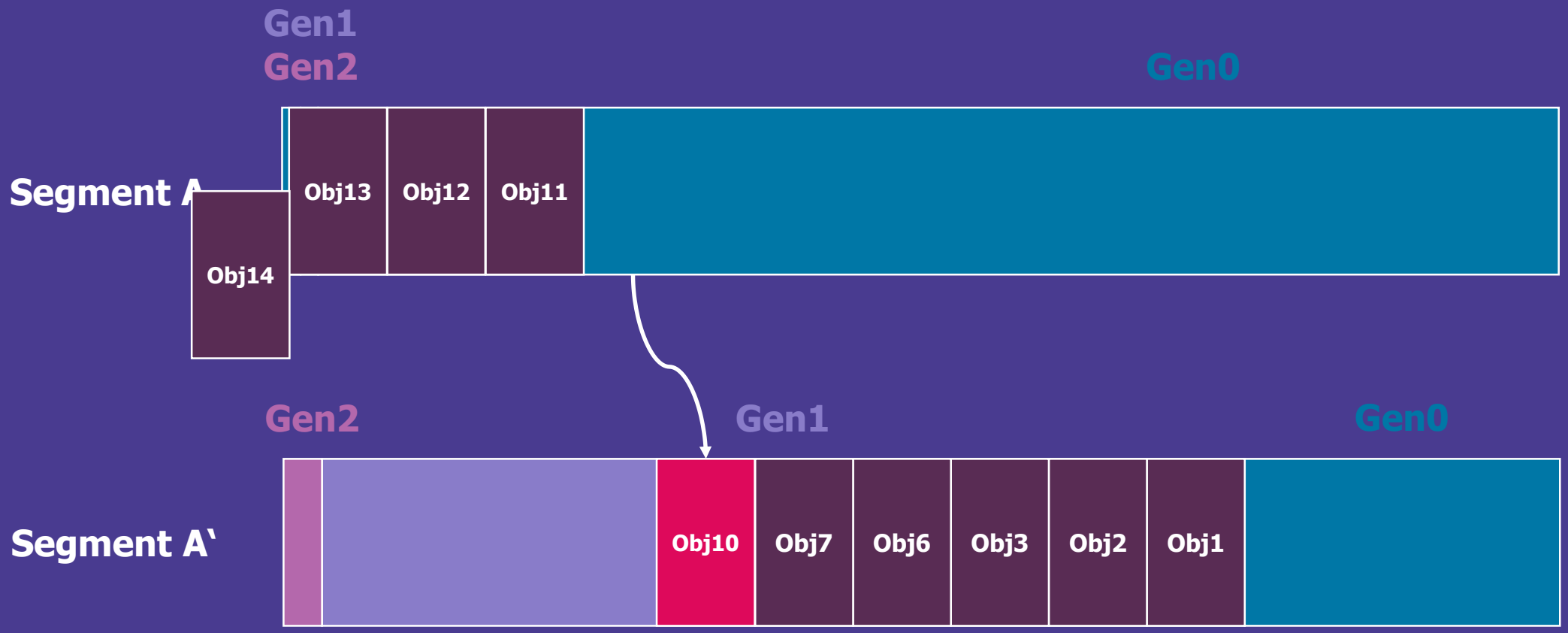
# The Compact Phase

Relocate and promote live objects.



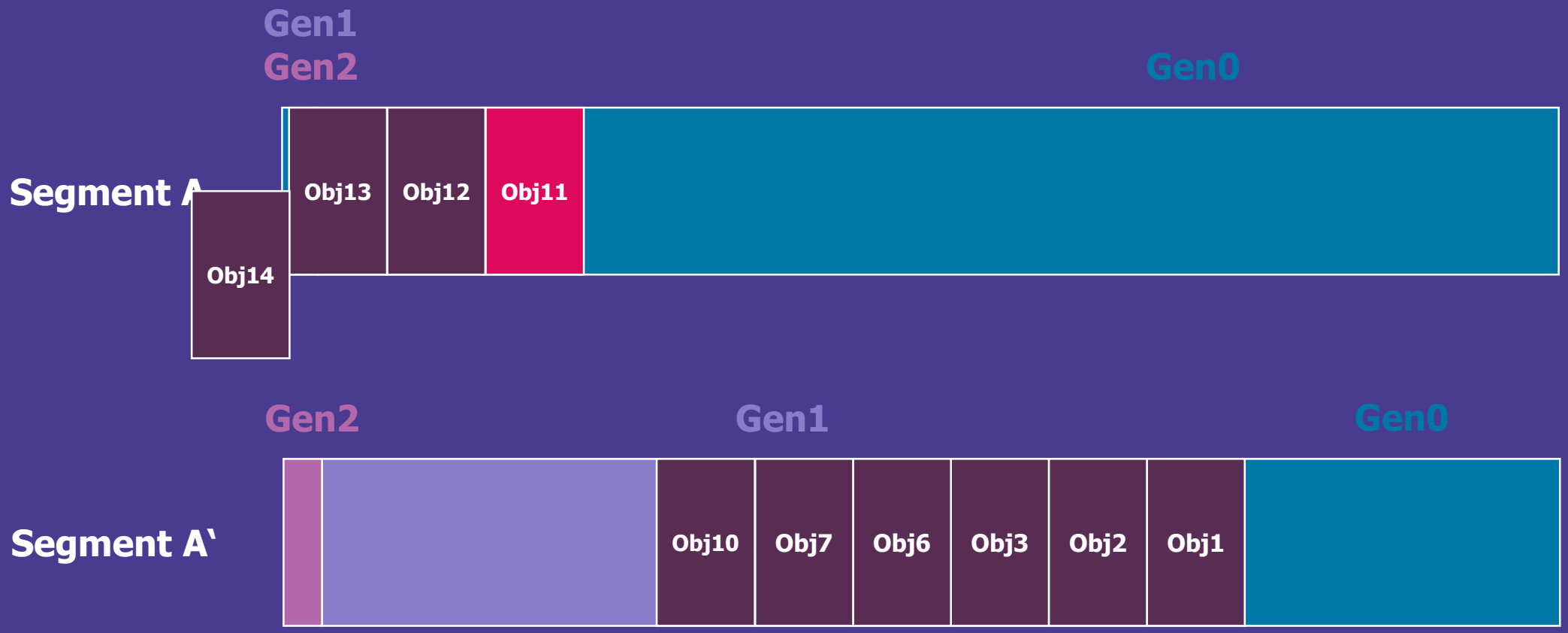
# The Compact Phase

Relocate and promote live objects.



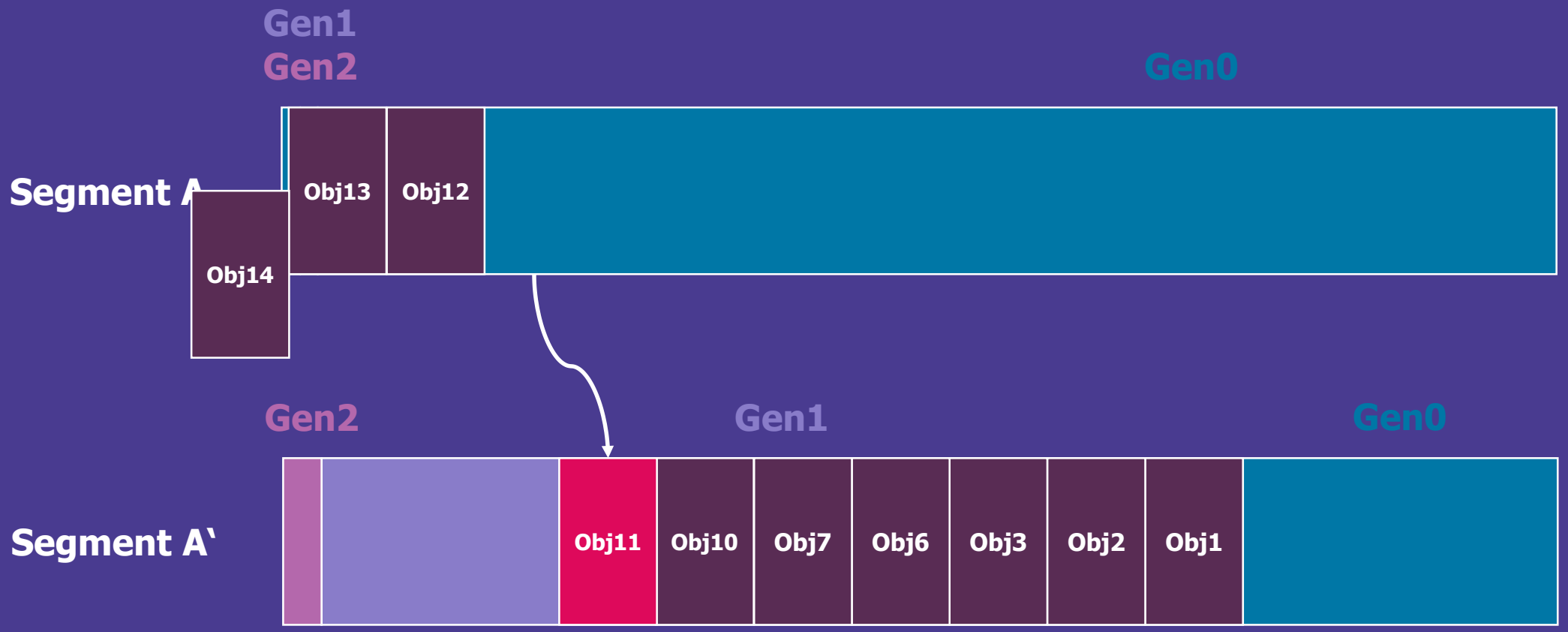
# The Compact Phase

Relocate and promote live objects.



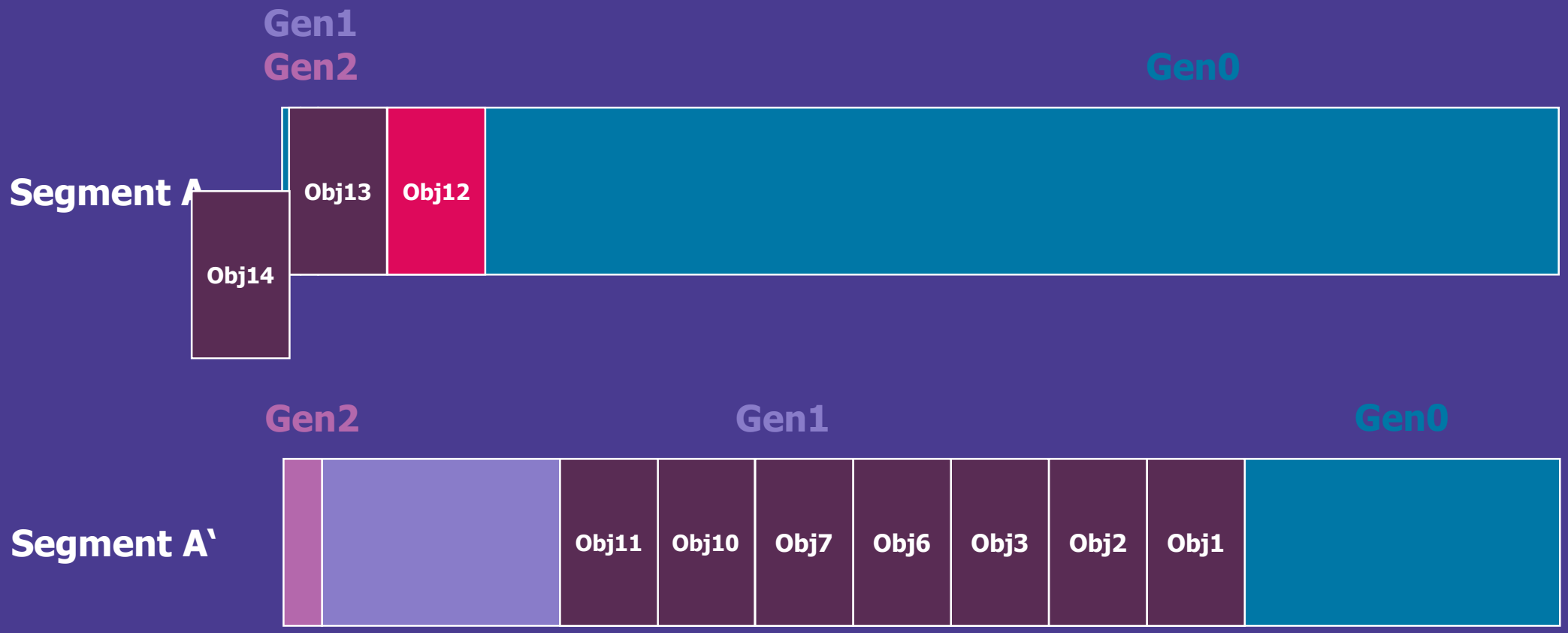
# The Compact Phase

Relocate and promote live objects.



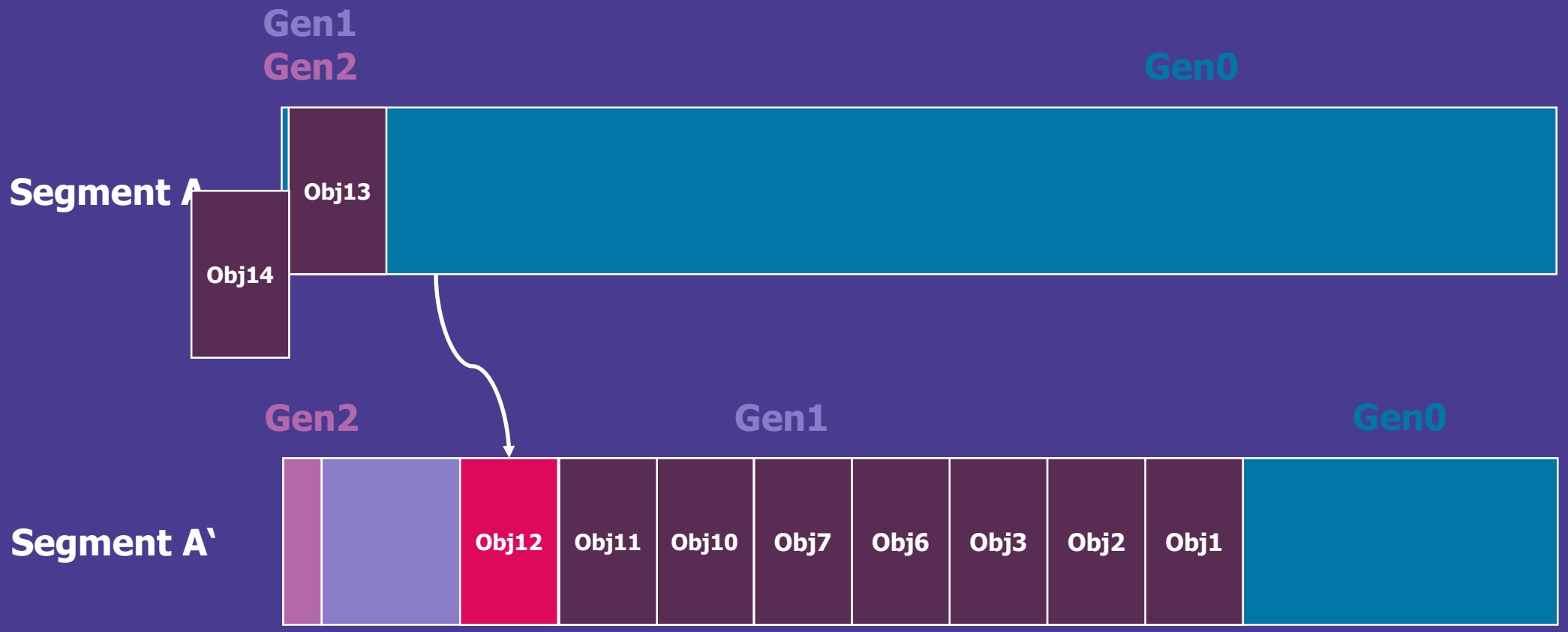
# The Compact Phase

Relocate and promote live objects.



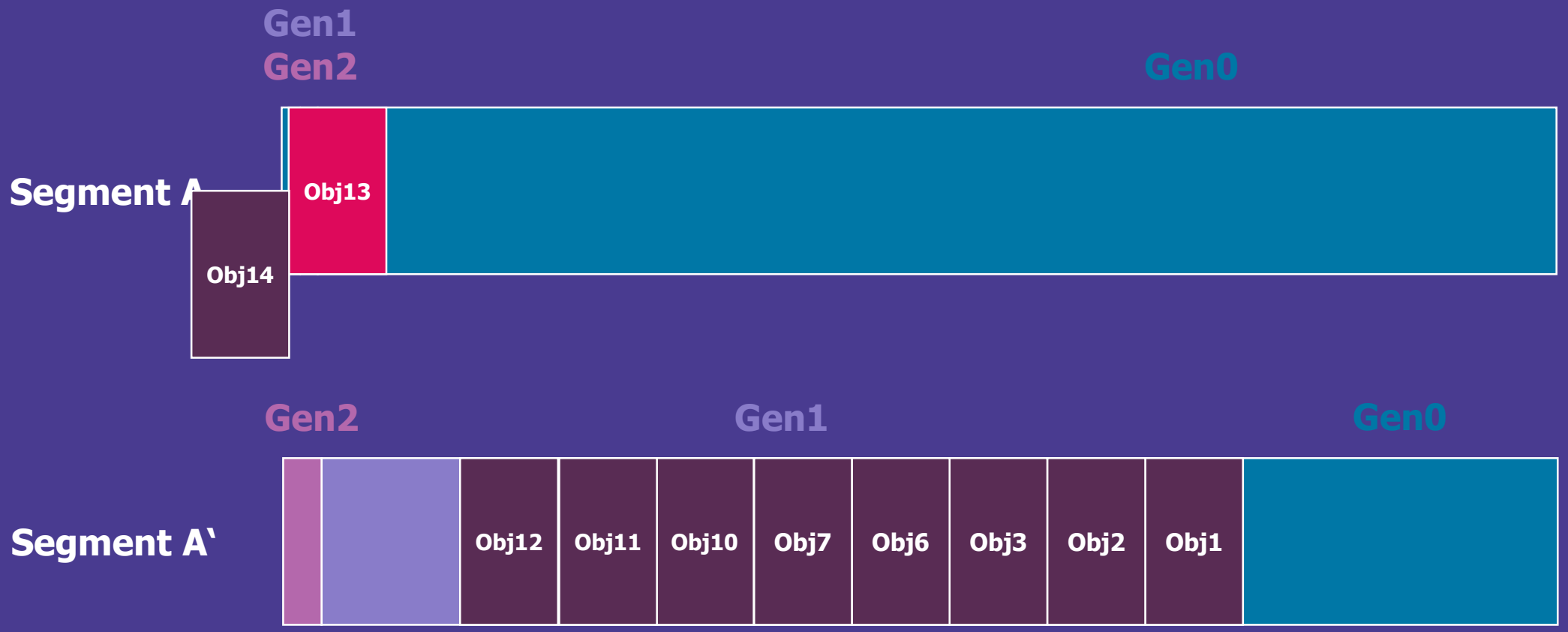
# The Compact Phase

Relocate and promote live objects.



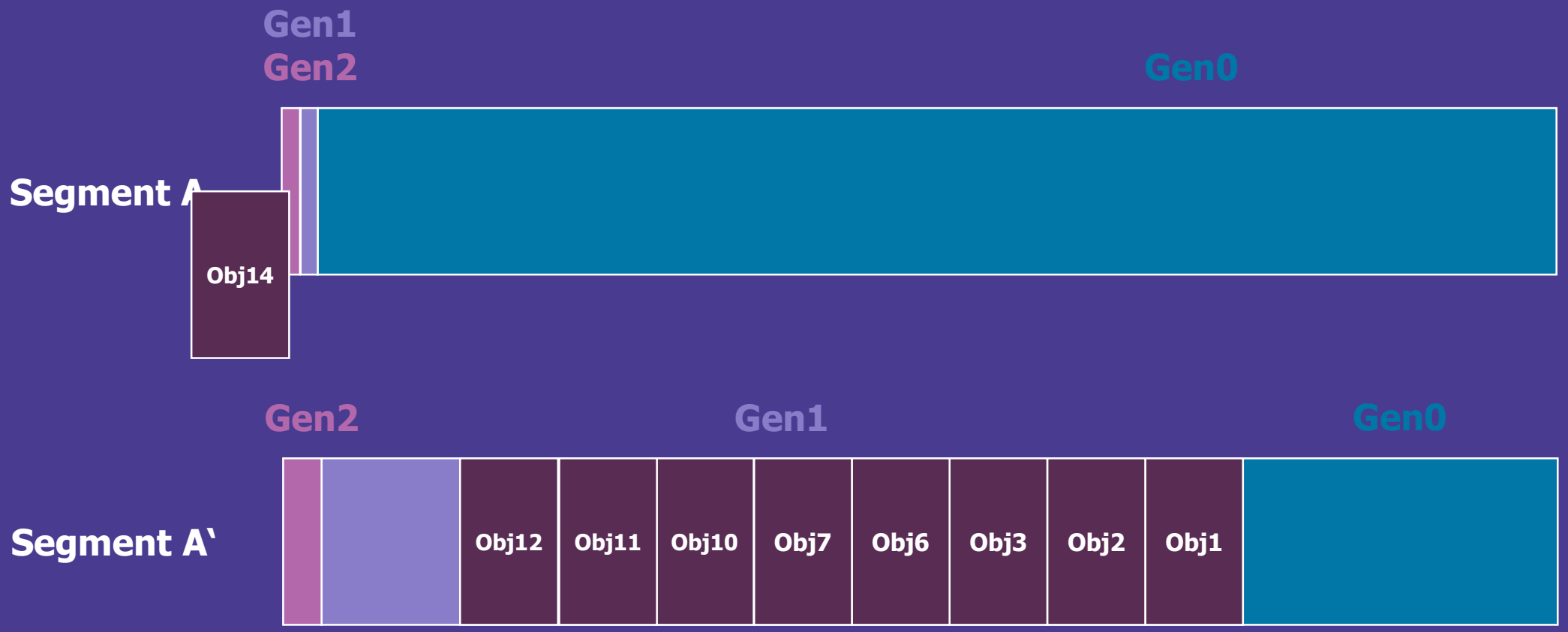
# The Compact Phase

Relocate and promote live objects.



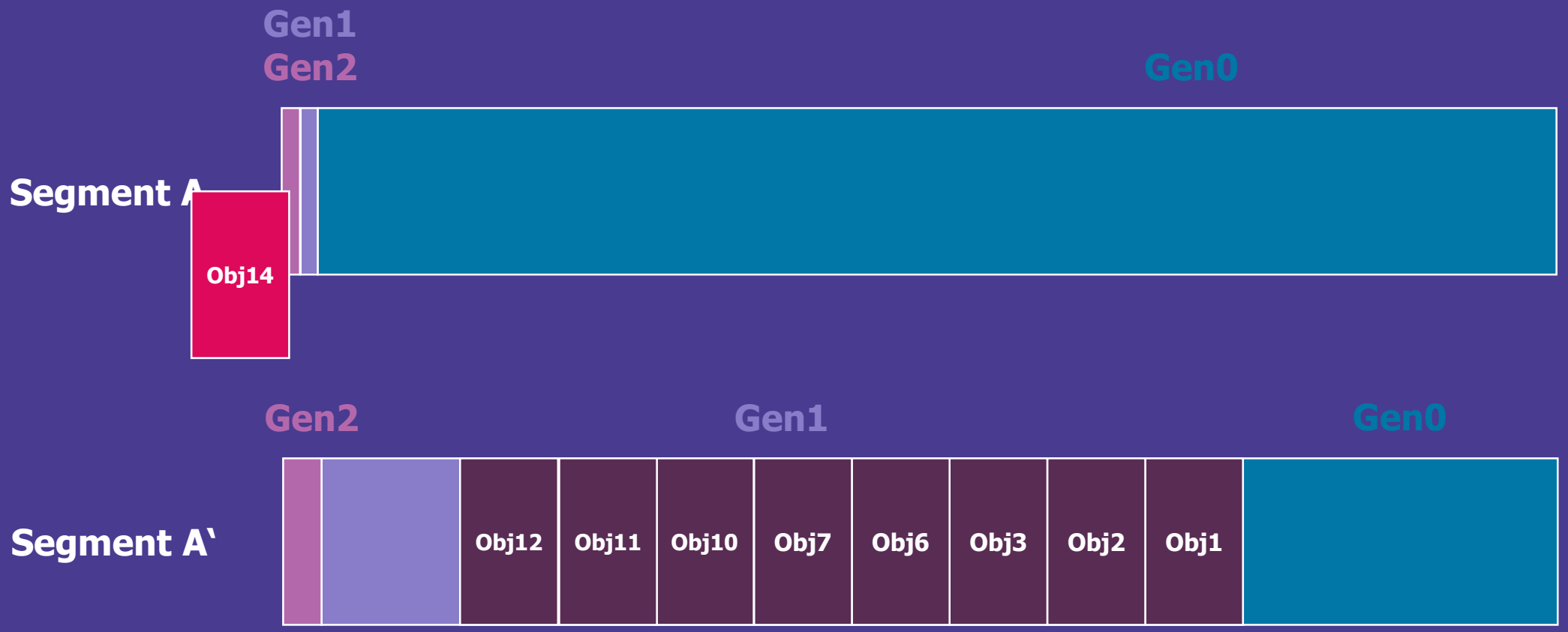
# The Compact Phase

Relocate and promote live objects.



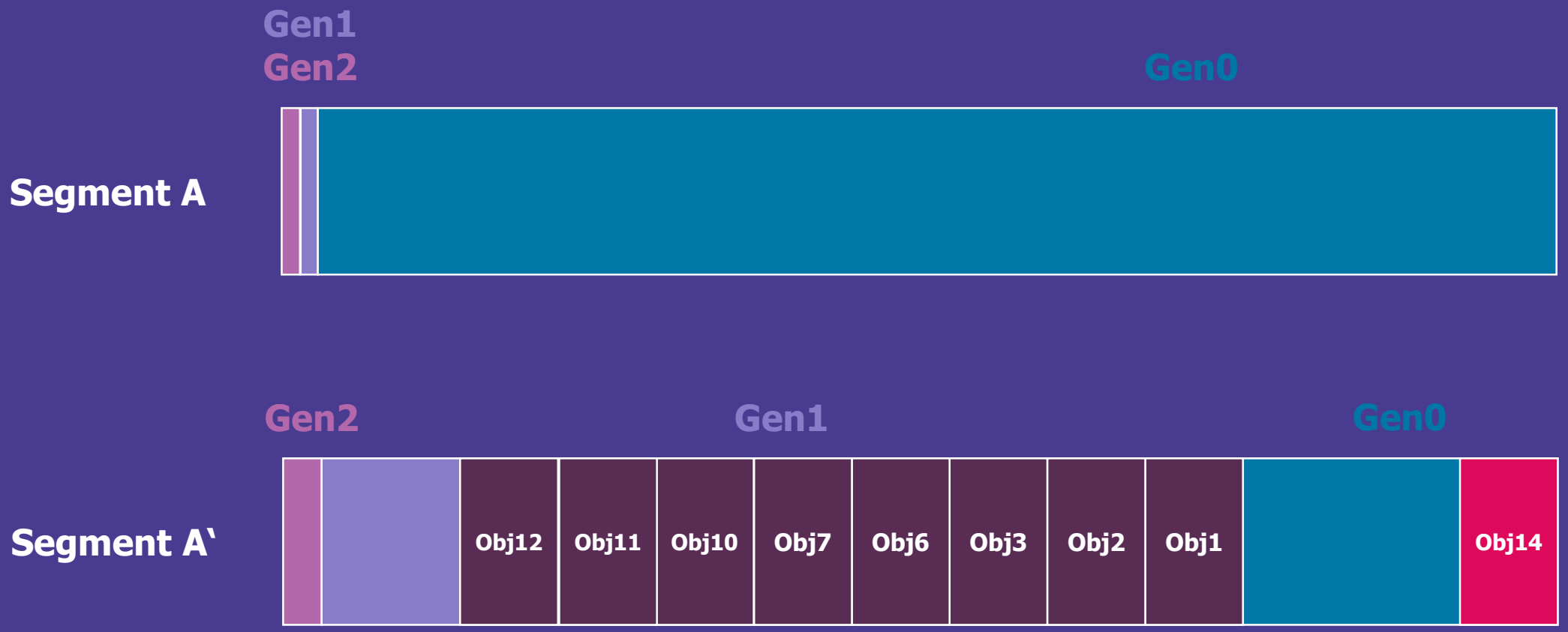
# The Compact Phase

Relocate and promote live objects.



# The Compact Phase

Relocate and promote live objects.



# The Compact Phase


# Garbage Collection

**Garbage Collection occurs only for a specific generation and all lower generations:**

*Gen0 Collection collects Gen0 only.*

*Gen1 Collection collects Gen1 and Gen0.*

*Gen2 Collection collects all generations (Full Collection).*

*(However, sub-graphs of higher generations may also be traversed.)*

# A Thought on Non-Determinism

**The Garbage Collector itself *is not* non-deterministic.**

**We just cannot predict its exact behavior from our code alone.**

**It monitors the memory usage on the target machine.**

**So we can think of it as non-deterministic.**

**However, the Garbage Collector actually follows strict rules.**

# A Thought on Non-Determinis

The Garbage Co

We just ca ione.

It moni

So we can think of it as nor rminis

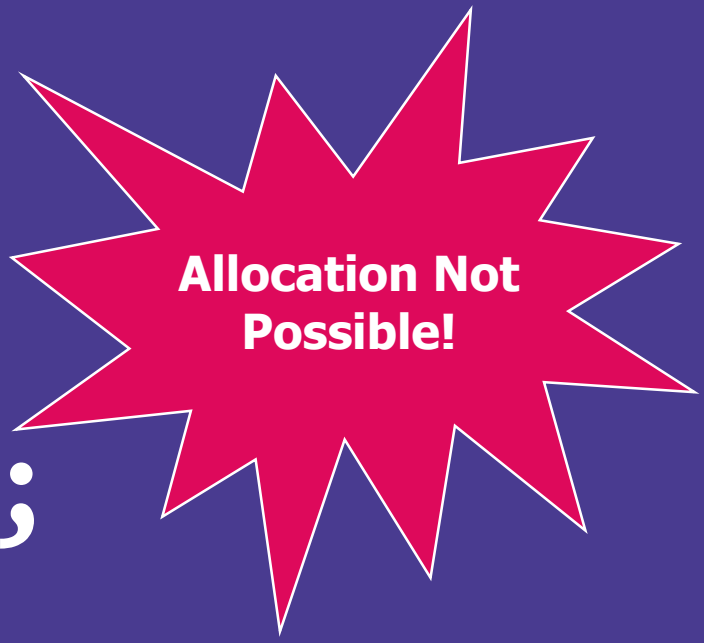
However, the Garbage Colle or actually follows strict rules.

Another
*Garbage Collection*
was triggered!

# Gen1 Caching

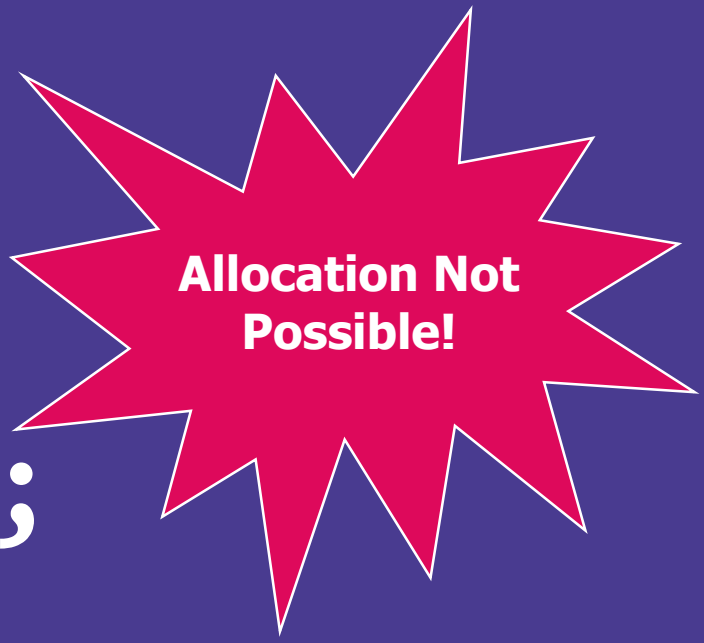
```
Obj15 = new B();
```

# Gen1 Caching




```
Obj15 = new B();
```

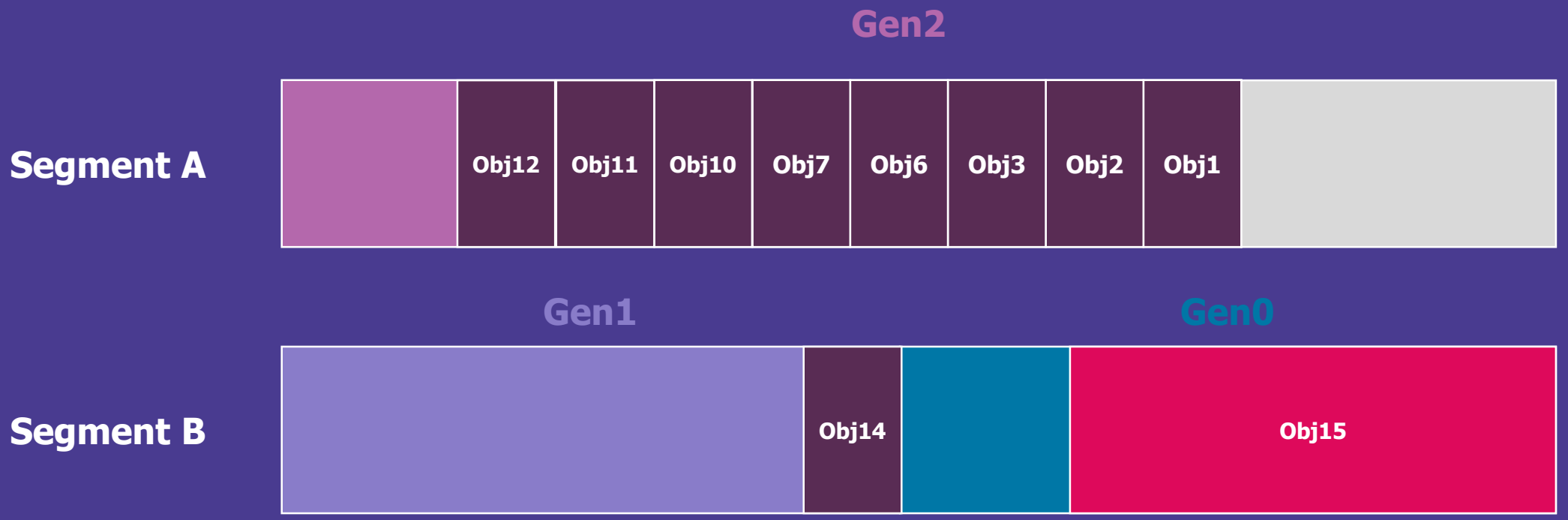
# Gen1 Caching




```
Obj15 = new B();
```

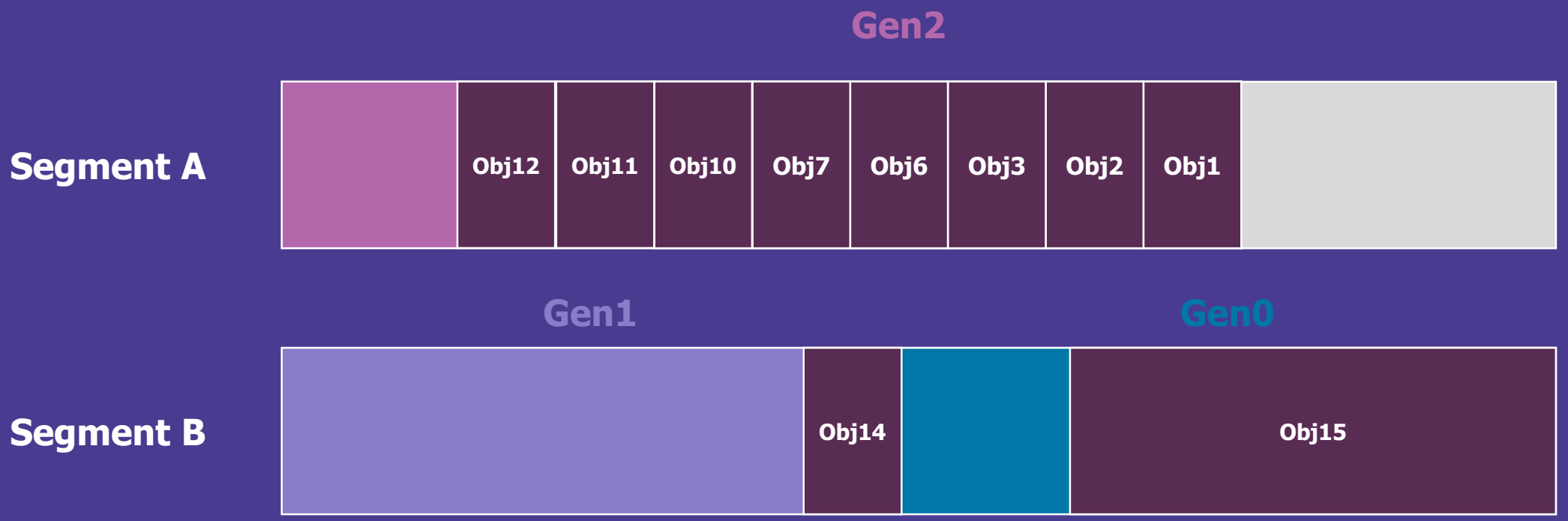
# Gen1 Caching

```
Obj15 = new B();
```



# Gen1 Caching

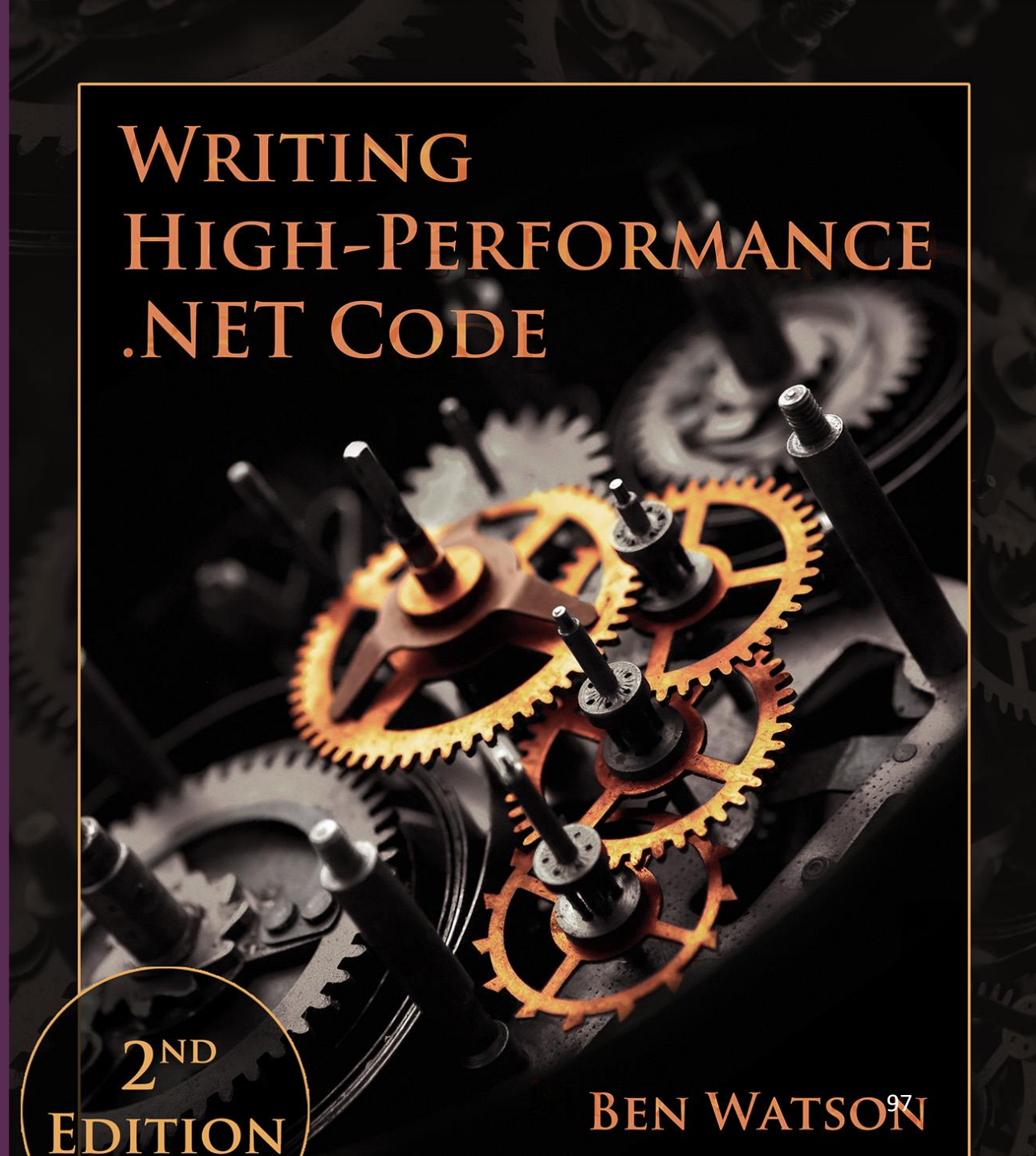
**Gen1 objects do not necessarily need to be relocated during GC runs.**



# The Most Important Rule

"[…] The garbage collector was explicitly designed with this idea in mind: **Collect objects in Gen0 or not at all.**"

**Ben Watson**
***Writing High-Performance .NET Code***



# The Most Important Rule

**„Collect objects in Gen0 or not at all.“**

**Because:**

*Collections for higher generations are more expensive.*
*Objects which don‘t die in Gen0 need to be relocated **physically**.*
*This requires*
*(a) time*
*(b) all managed threads to be suspended.*

***So, during GC runs, the CLR will not execute your program.***

# The Most Important Rule

**„Collect objects in Gen0 or not at all.”**

*Like any other program the Garbage Collector is also designed with specific assumptions in mind.*

***So, if our code breaks these assumptions we cannot complain when it doesn‘t work how we expect it to!***

# Don’t work against the Garbage Collector!

# Time spend in Garbage Collection is time not spend in the rest of the program!

# References


- Ben Watson and Leticia Watson. 2018. *Writing High-Performance .NET Code* (2nd. ed.).

  writinghighperf.net

- Konrad Kokosa. 2018. *Pro .NET Memory Management: For Better Code, Performance, and Scalability* (1st. ed.). Apress, USA.

  prodotnetmemory.com

BRICKMAKERS
cutting edge apps

# Danke für die Aufmerksamkeit!

**BRICKMAKERS GmbH**

**Am Plan 14-16**
**56068 Koblenz**

**info@brickmakers.de**

**brickmakers.de**