Garbage Collector Tuning

in Pathological Allocation Pattern Applications

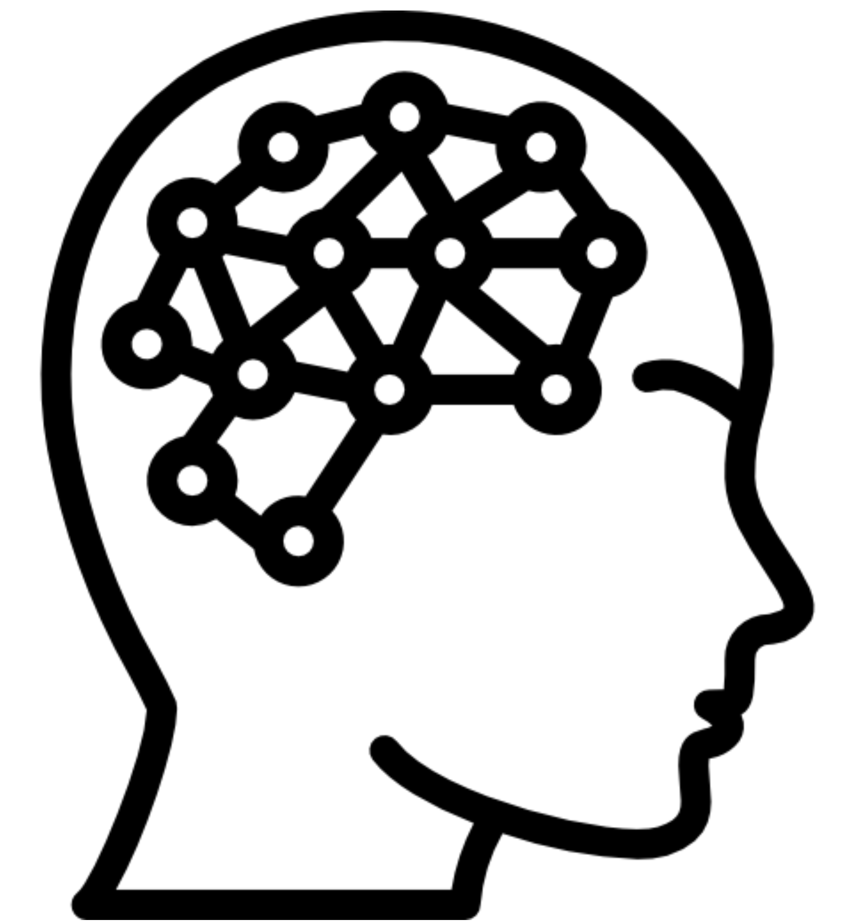
Nahuel Palumbo - Sebastian Jordan Montaño - Guillermo Polito - Pablo Tesone - Stéphane Ducasse

nahuel.palumbo@inria.fr

IWST ’23

Motivation

"Pharo is slow"



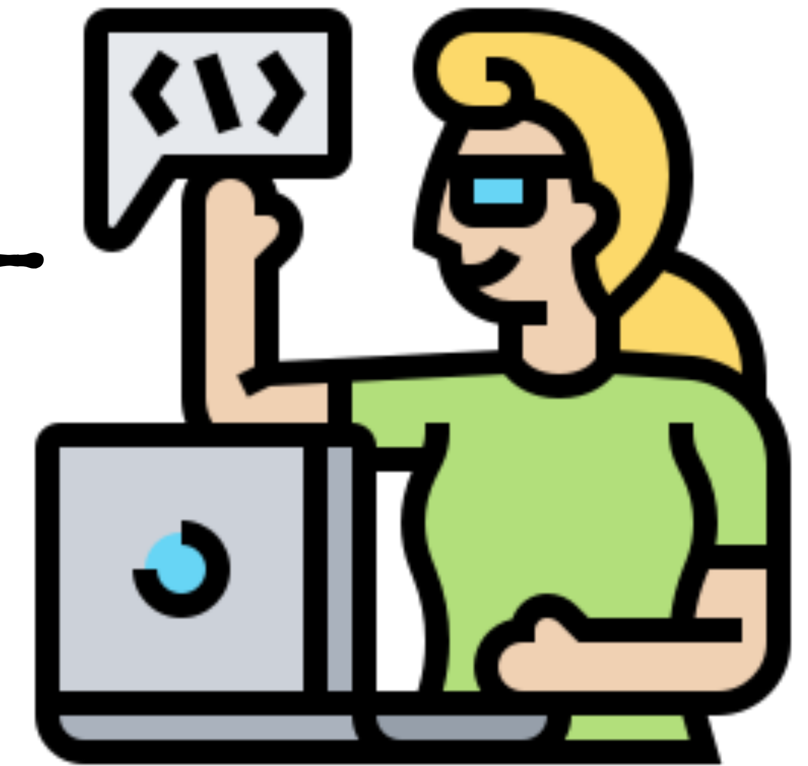
Pharo-AI Developers

My application takes >1h30m

What are you doing?

Loading a 3GB dataset

Ok, let's see the memory management



VM Developers

Motivation

“~~Pharo~~ The GC is slow”

Data size	Total time (sec)	GC time (sec)	GC overhead
529 MB	43	7	16%
1.6 GB	150	38	25%
3.1 GB	5599	5158	**92%**

DataFrame

Playground

Do it Publish Bindings Versions Pages

```
1 DataFrame readFromCsv: pathToFile asFileReference.
```

Line: 1:1 +L

Motivation

“~~Pharo~~ The GC is slow”

Data size	Total time (sec)	GC time (sec)	GC overhead
529 MB	43 ~1min	7	16%
1.6 GB	150 =2.5min	38	25%
3.1 GB	5599 >1h30m	5158	92%

DataFrame

Playground

Do it Publish Bindings Versions Pages

```
1 DataFrame readFromCsv: pathToFile asFileReference.
```

Line: 1:1 +L

Motivation

"~~Pharo~~ The GC is slow"

Data size	Total time (sec)	GC time (sec)	GC overhead
529 MB	43	7	16%
1.6 GB	150	38	25%
3.1 GB	5599	5158	**92%**

3x, 2x (data size increase); 1.5x, 3.6x (GC overhead increase)

Playground

Do it | Publish | Bindings | Versions | Pages

```
DataFrame readFromCsv: pathToFile asFileReference.
```

Line: 1:1 +L

Motivation

"~~Pharo~~ The GC is slow"

Data size	Total time (sec)	GC time (sec)	GC overhead
529 MB	43	7	16%
1.6 GB	150	38	25%
3.1 GB	**5599** >1h30m	**5158**	**92%** !!!!

~1h25m

DataFrame

Playground

Do it | Publish | Bindings | Versions | Pages

```
1 DataFrame readFromCsv: pathToFile asFileReference.
```

Line: 1:1 +L

Memory Management

Garbage Collection

Manually Memory Management

A work for devs?

```
data = malloc(size)
...
... use data ...
...
free(data)
```

**Manually
Memory Management**

Automatic Memory Management

Garbage Collectors

```
data = malloc(size)
...
... use data ...
...
free(data)
```

Manually Memory Management

Compute the size
Allocate in the memory

```
data = Data new
...
... use data ...
...
??????????
```

Maybe move the data for better use of the memory

Free the space when data is not used anymore

Automatic Memory Management

Garbage Collectors

Compute the size
Allocate in the memory

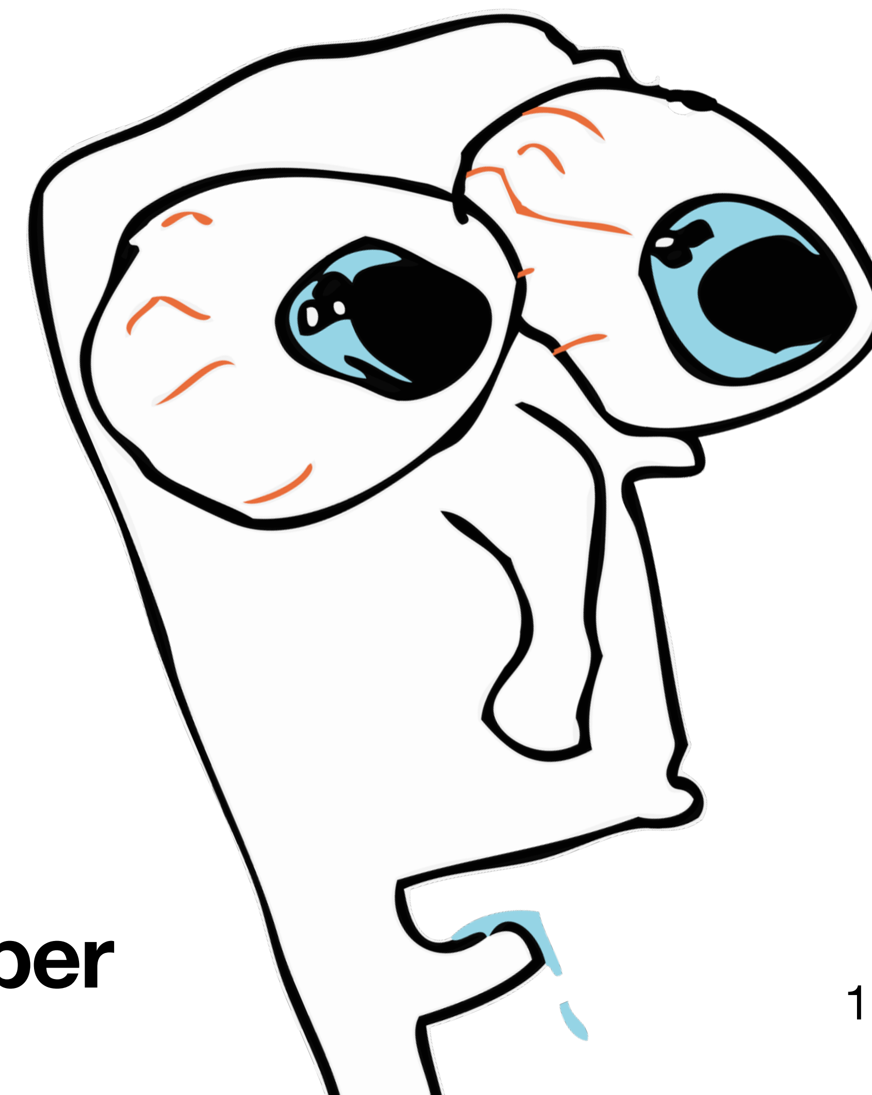
```
data = malloc(size)
...
... use data ...
...
free(data)
```

Manually Memory Management

```
data = Data new
...
... use data ...
...
??????????
```

Maybe move the data for better use of the memory

Free the space when data is not used anymore



Developer

Automatic Memory Management

Garbage Collectors

```
data = malloc(size)
...
... use data ...
...
free(data)
```

Manually
Memory Management

Developer

Compute the size
Allocate in the memory

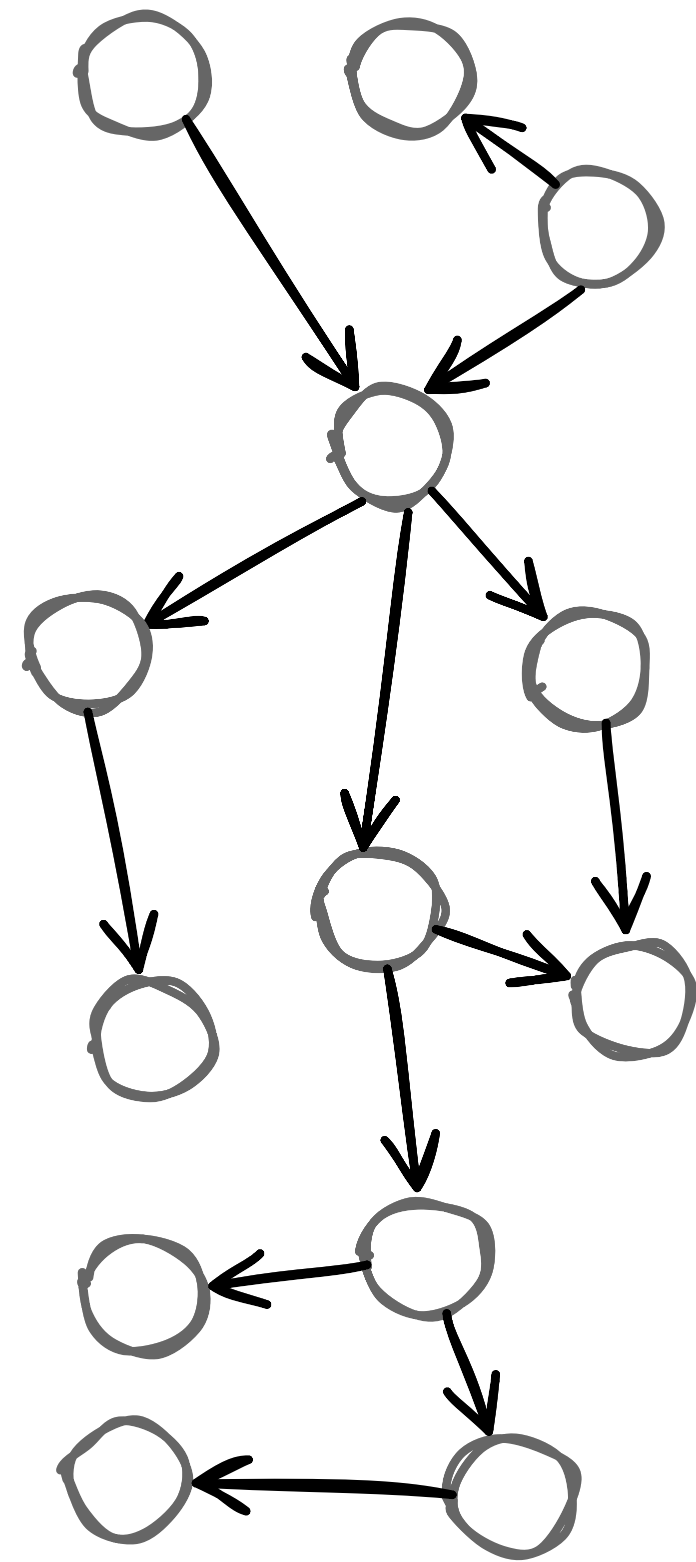
```
data = Data new
...
... use data ...
...
??????????
```

Maybe move the data for
better use of the memory

Free the space when data
is not used anymore

Application's Allocation Patterns

How do the applications use the memory?

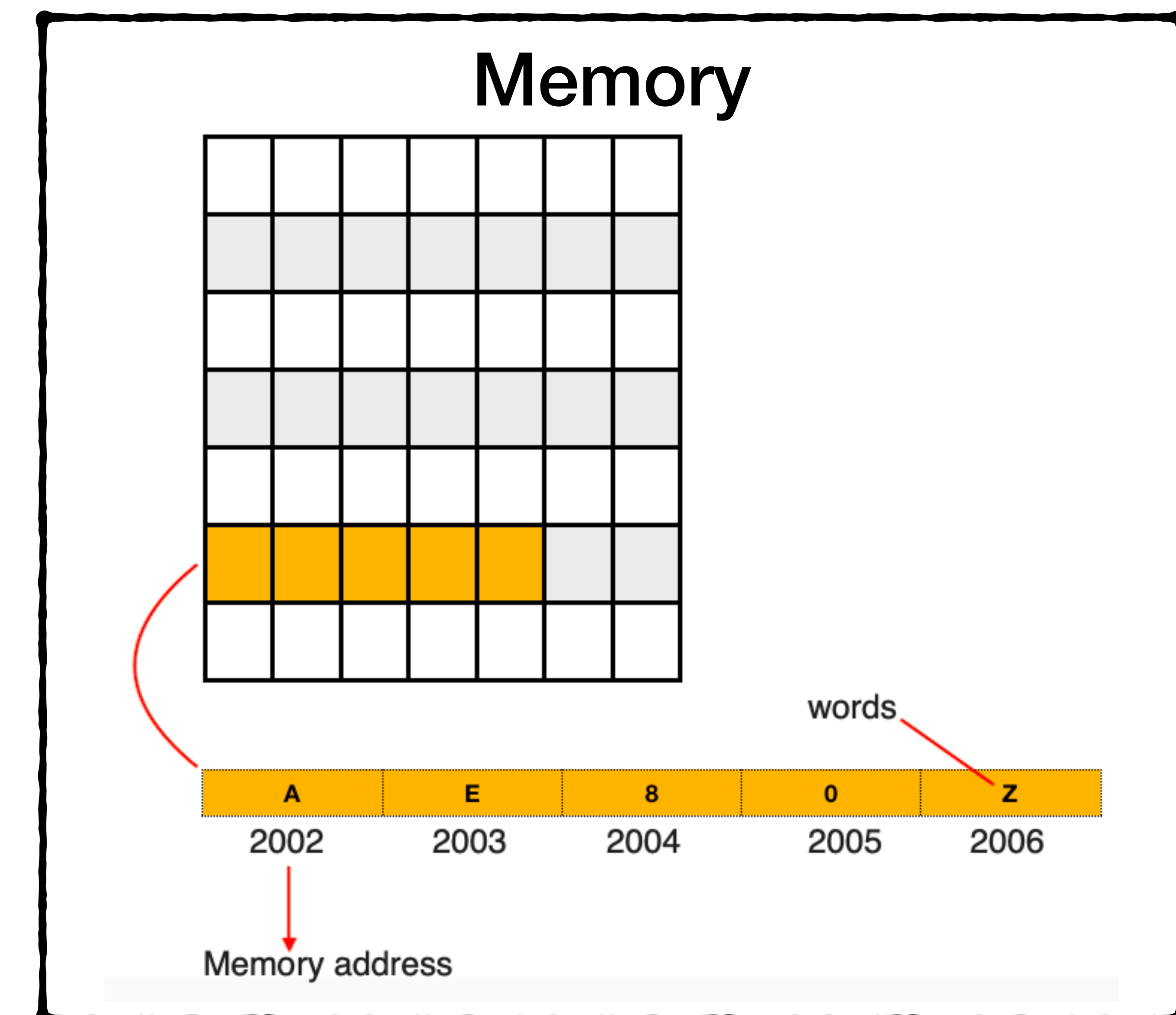


Allocations are particular for each application

→ **Hard to predict**

There are some general heuristics

→ **Easy to predict**



Application's Allocation Patterns

Weak generational hypothesis

```
data = Data new
... first use of data ...
??????????
```

Will you use it again?

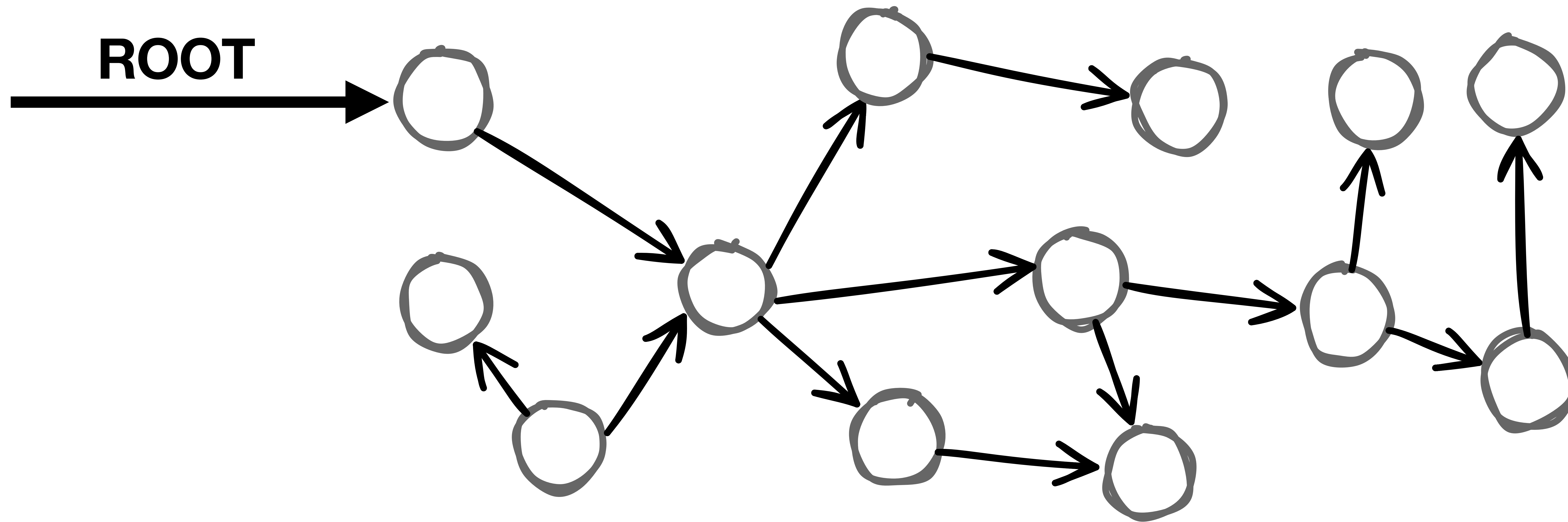
>90% No, you can free the memory

<10% Yes, keep it!

"Most of the objects die young"

Application's Allocation Patterns

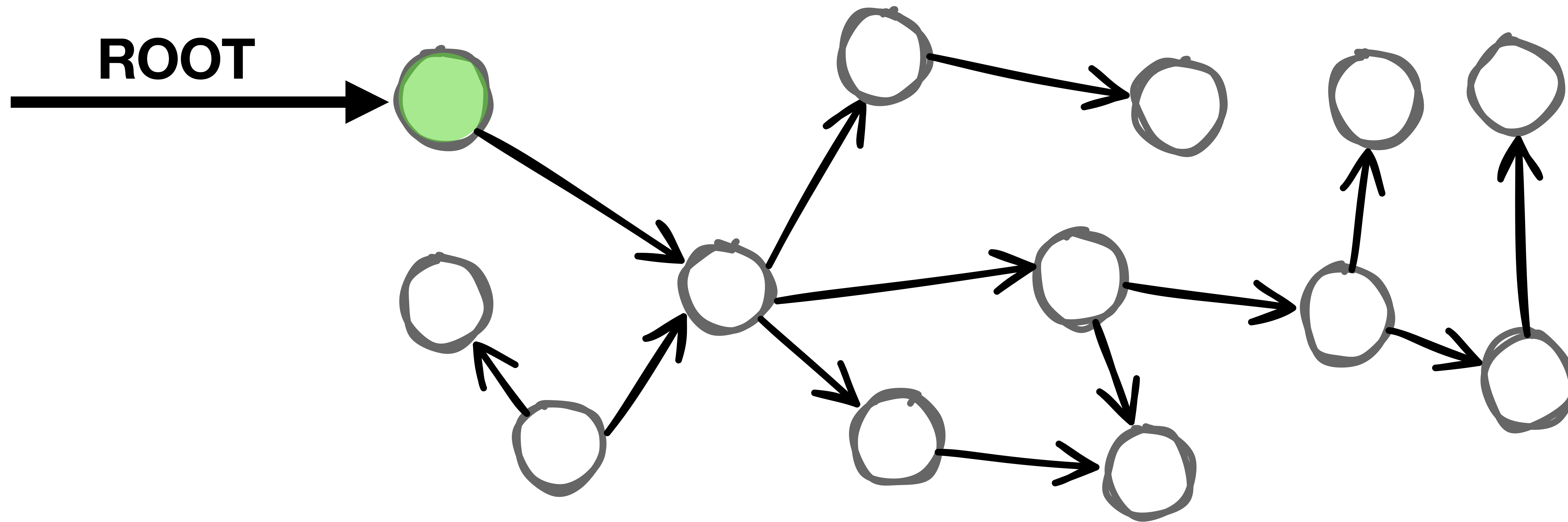
When an object dies?



"Must be accessible from the roots"

Application’s Allocation Patterns

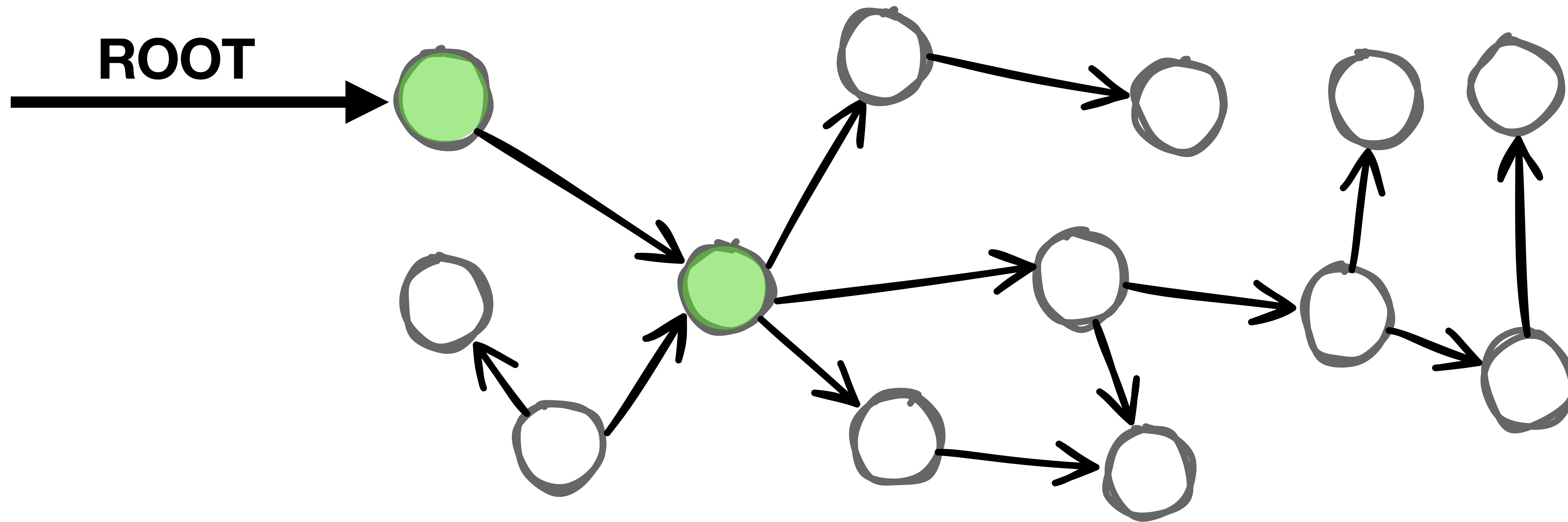
When an object dies?



“Must be accessible from the roots”

Application's Allocation Patterns

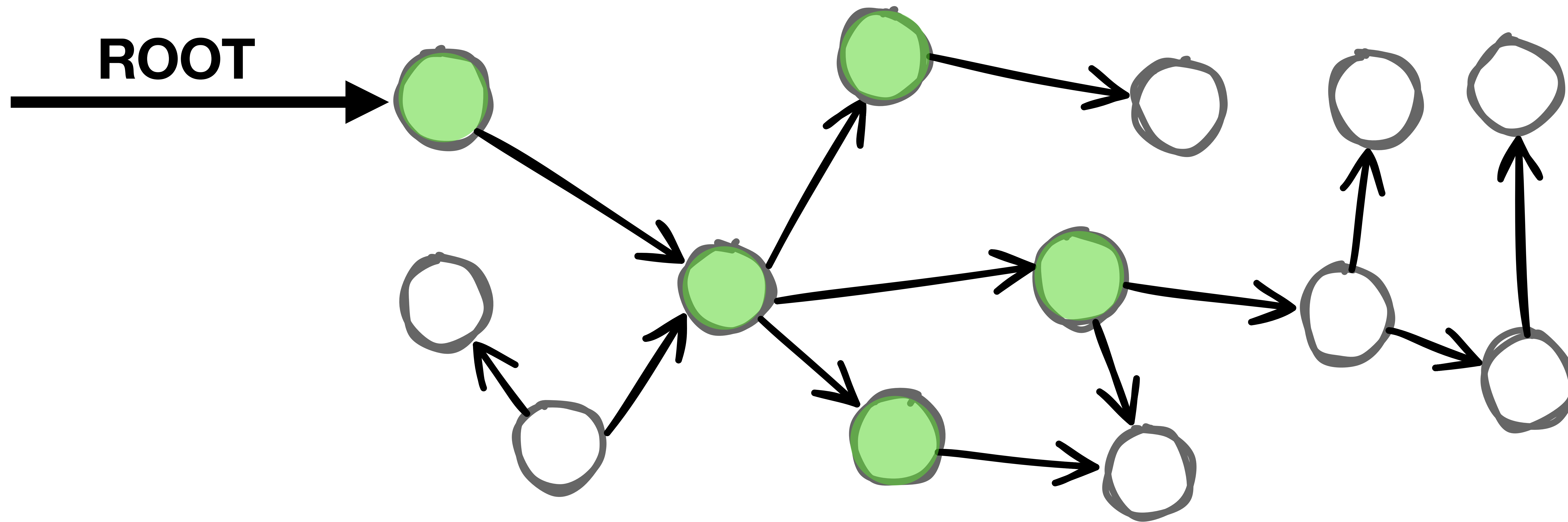
When an object dies?



"Must be accessible from the roots"

Application's Allocation Patterns

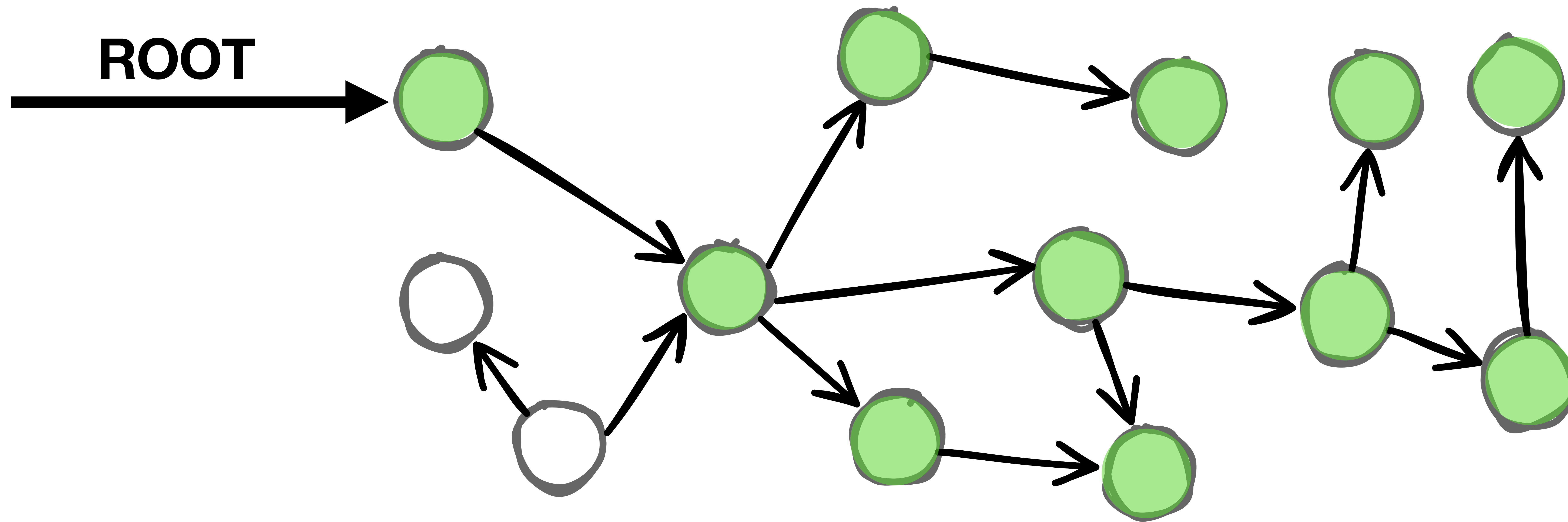
When an object dies?



"Must be accessible from the roots"

Application's Allocation Patterns

When an object dies?



"Must be accessible from the roots"

Application's Allocation Patterns

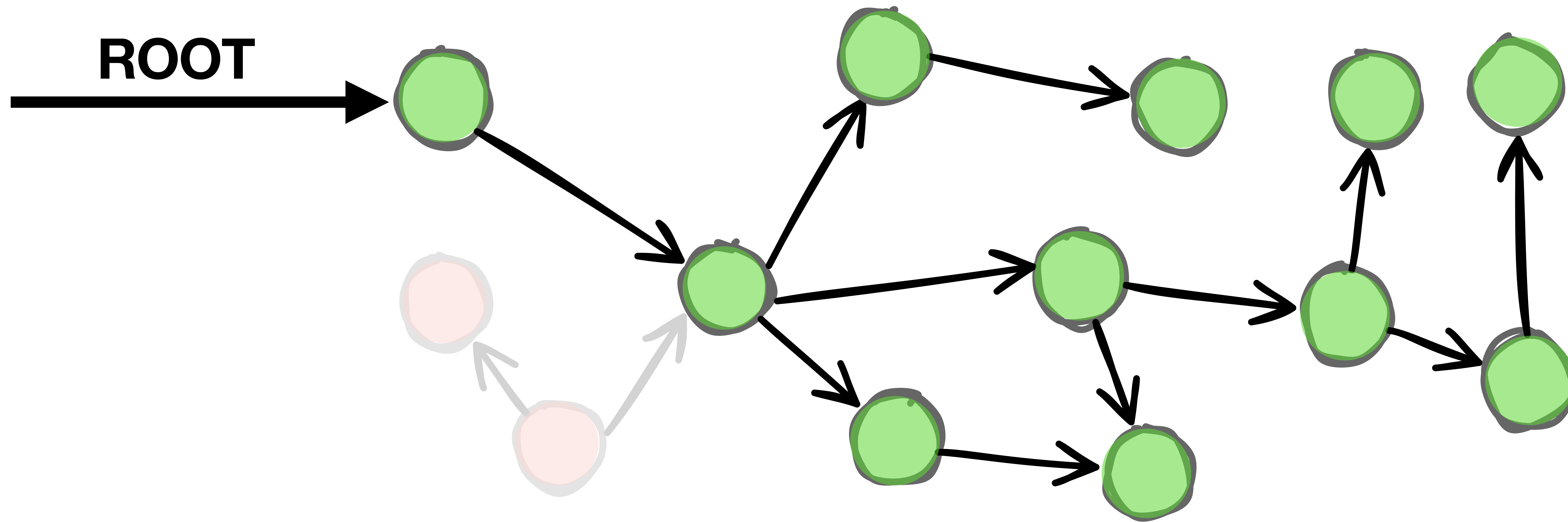
When an object dies?

ROOT

"Must be accessible from the roots"

Application's Allocation Patterns

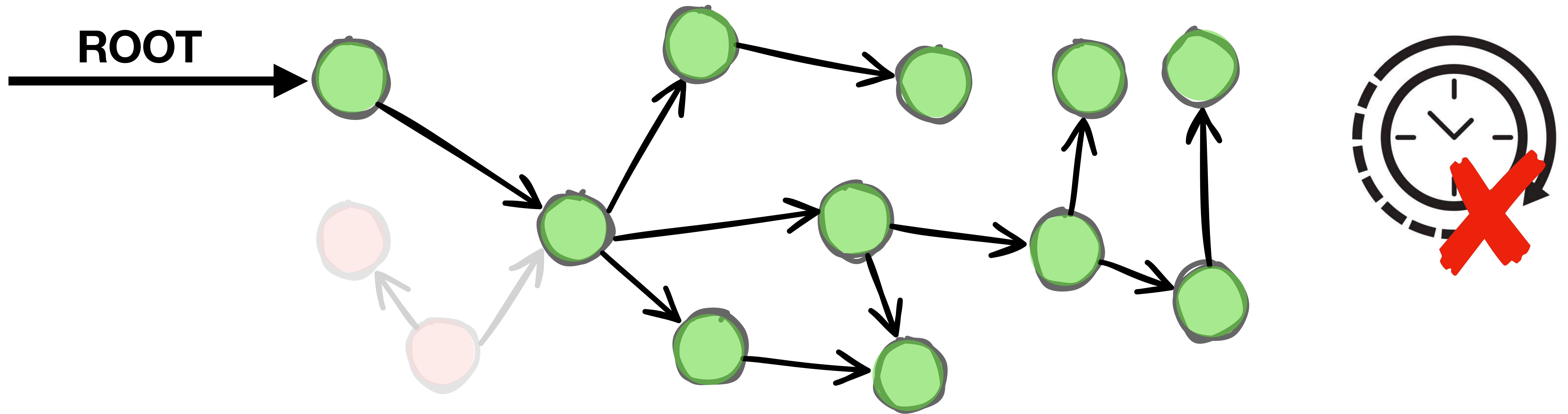
When an object dies?



"Must be accessible from the roots"

Application's Allocation Patterns

When an object dies?

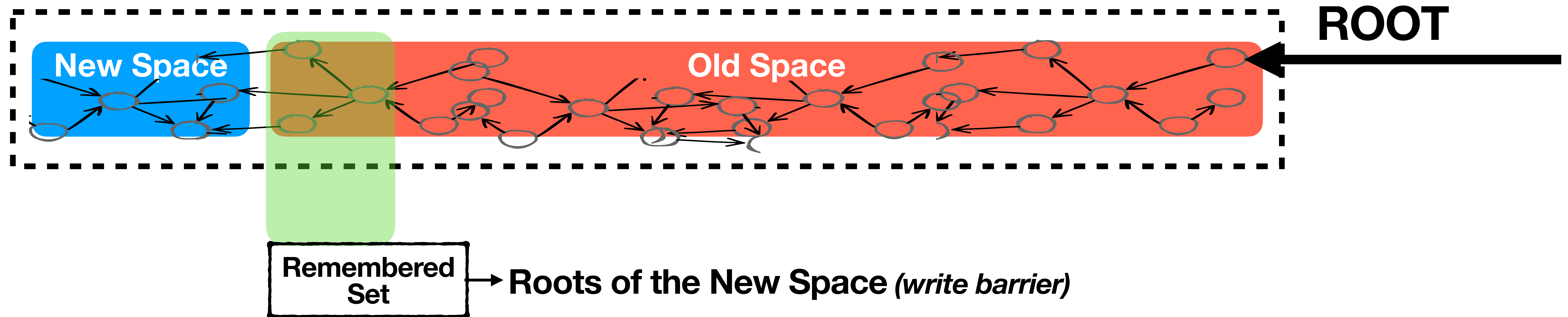


"Must be accessible from the roots"

Generational Garbage Collector

High-Performance Automatic Memory Management for OOP

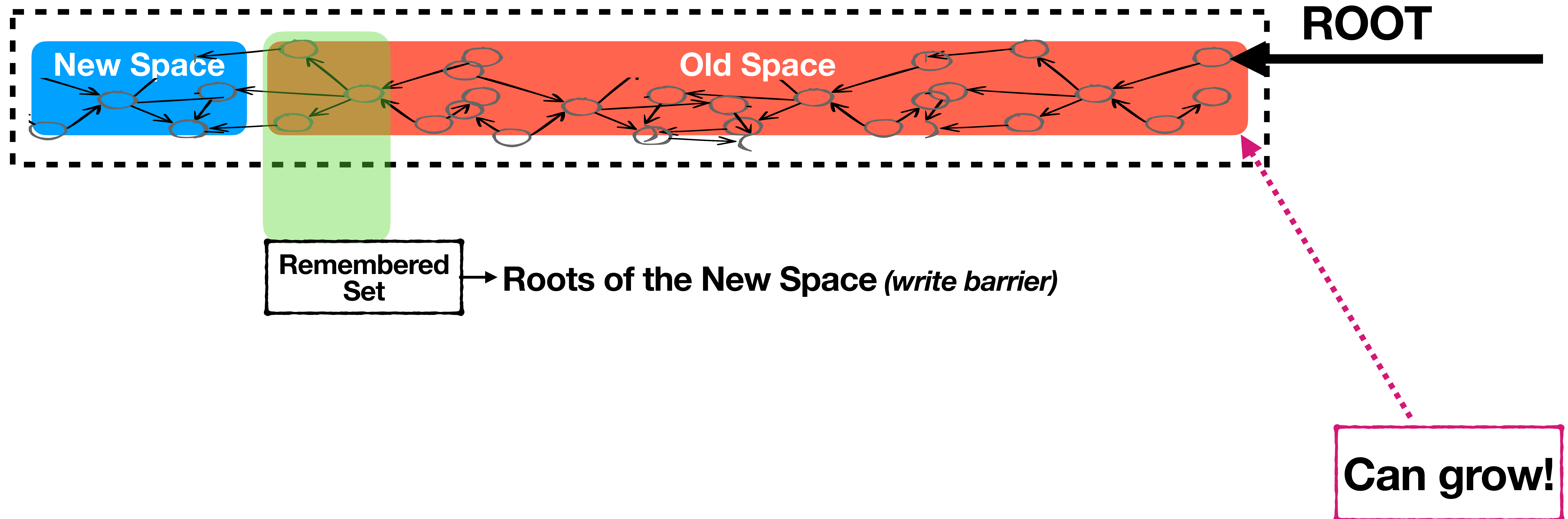
HEAP



D. Ungar, *Generation scavenging: A non-disruptive high performance storage reclamation algorithm*

Generational Garbage Collector

High-Performance Automatic Memory Management for OOP

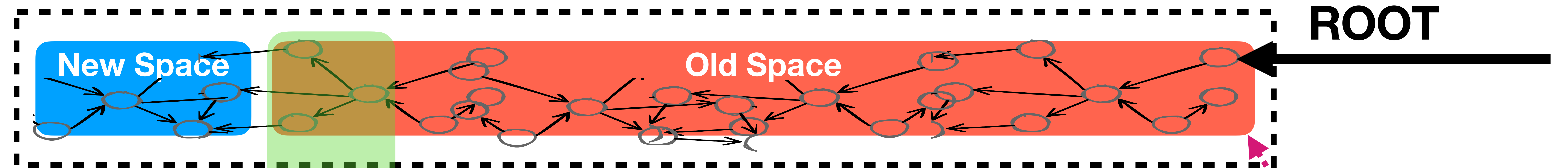


D. Ungar, *Generation scavenging: A non-disruptive high performance storage reclamation algorithm*

Generational Garbage Collector

High-Performance Automatic Memory Management for OOP

HEAP



FAST (often) → **Scavenge**

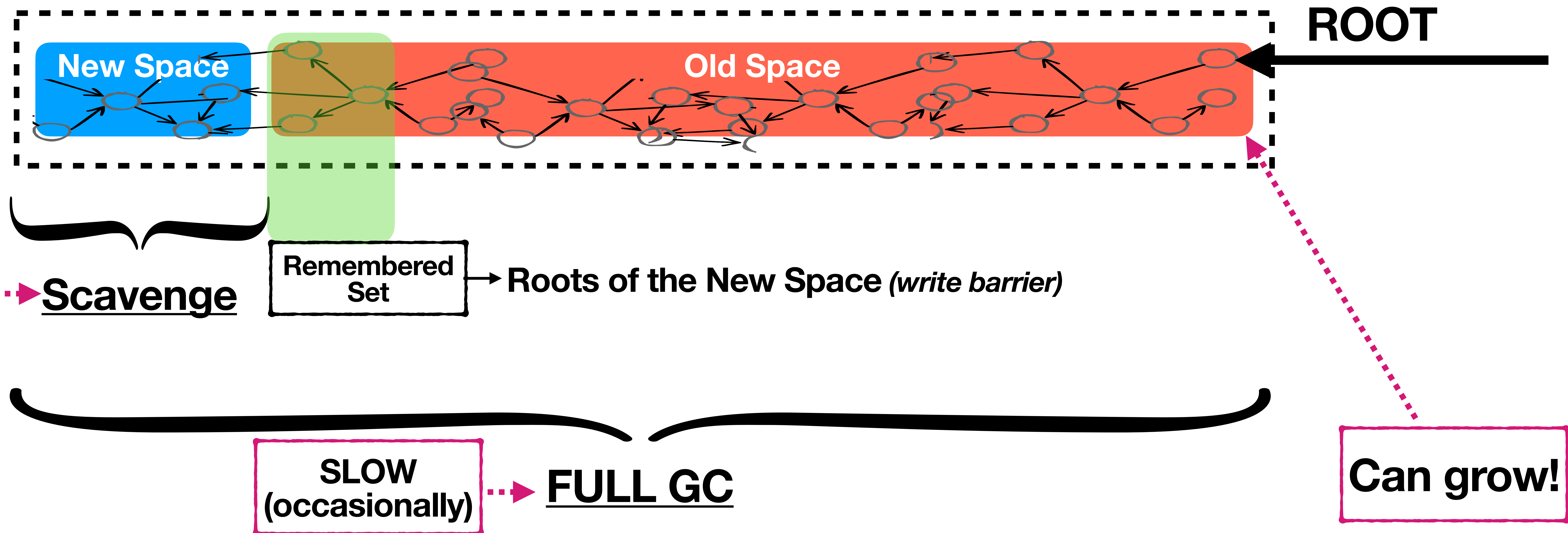
Remembered Set → **Roots of the New Space** ***(write barrier)***

Can grow!

D. Ungar, *Generation scavenging: A non-disruptive high performance storage reclamation algorithm*

Generational Garbage Collector

High-Performance Automatic Memory Management for OOP



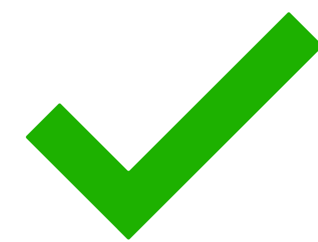
D. Ungar, *Generation scavenging: A non-disruptive high performance storage reclamation algorithm*

Pathological Allocation Pattern

Garbage Collectors' problem

Weak generational

Stable memory use



Few Full GCs

Fast Scavengers

Long lifetimes

Memory-starved

Many Full GCs

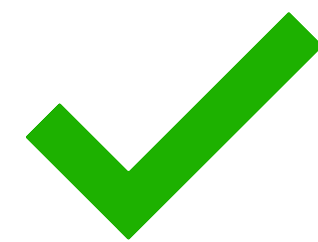
Scavenger overhead

Pathological Allocation Pattern

Garbage Collectors' problem

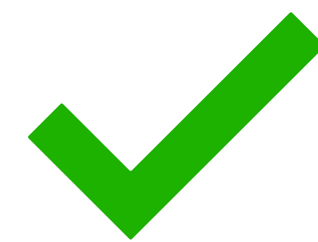
Weak generational

Stable memory use



Few Full GCs

Fast Scavengers



Long lifetimes

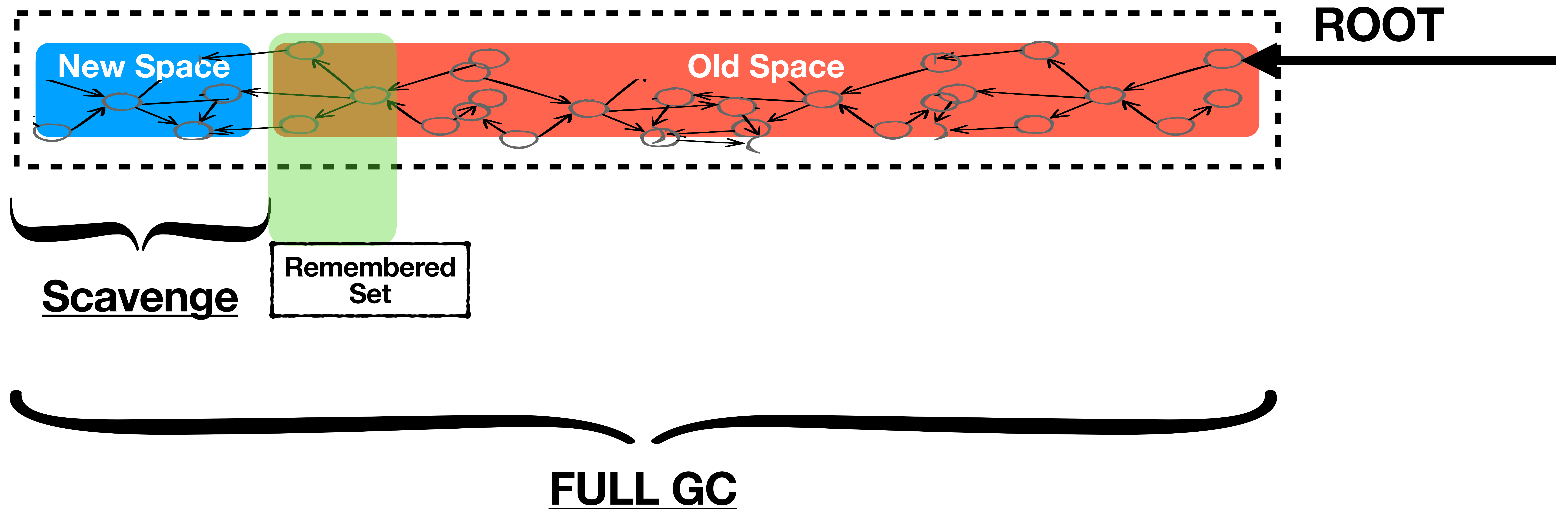
Memory-starved

Many Full GCs

Scavenger overhead

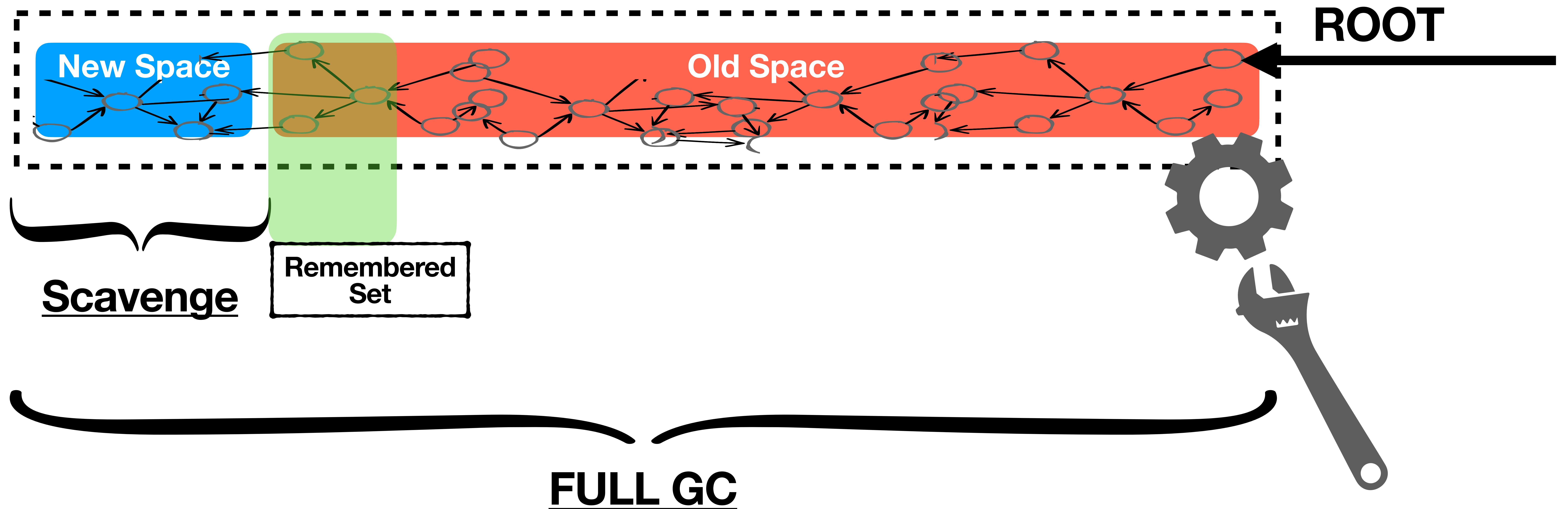
Pathological Allocation Pattern

Tuning the Garbage Collector



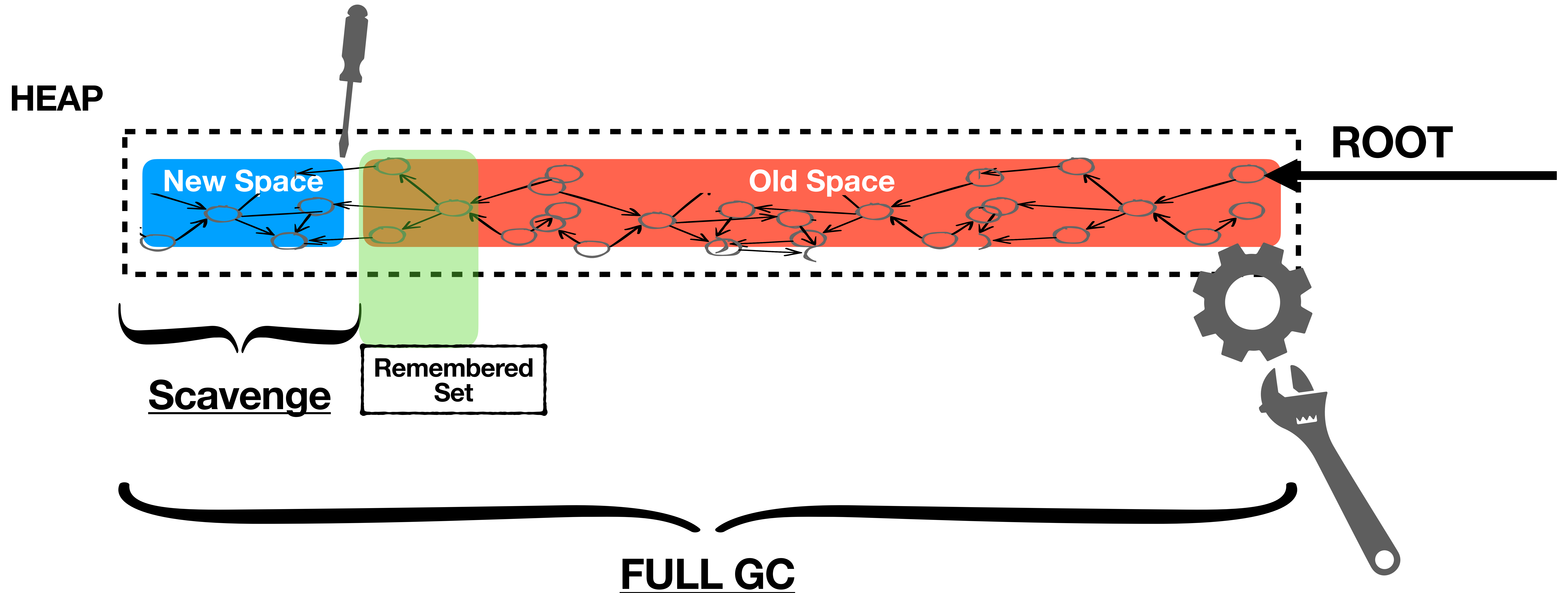
Pathological Allocation Pattern

Tuning the Garbage Collector



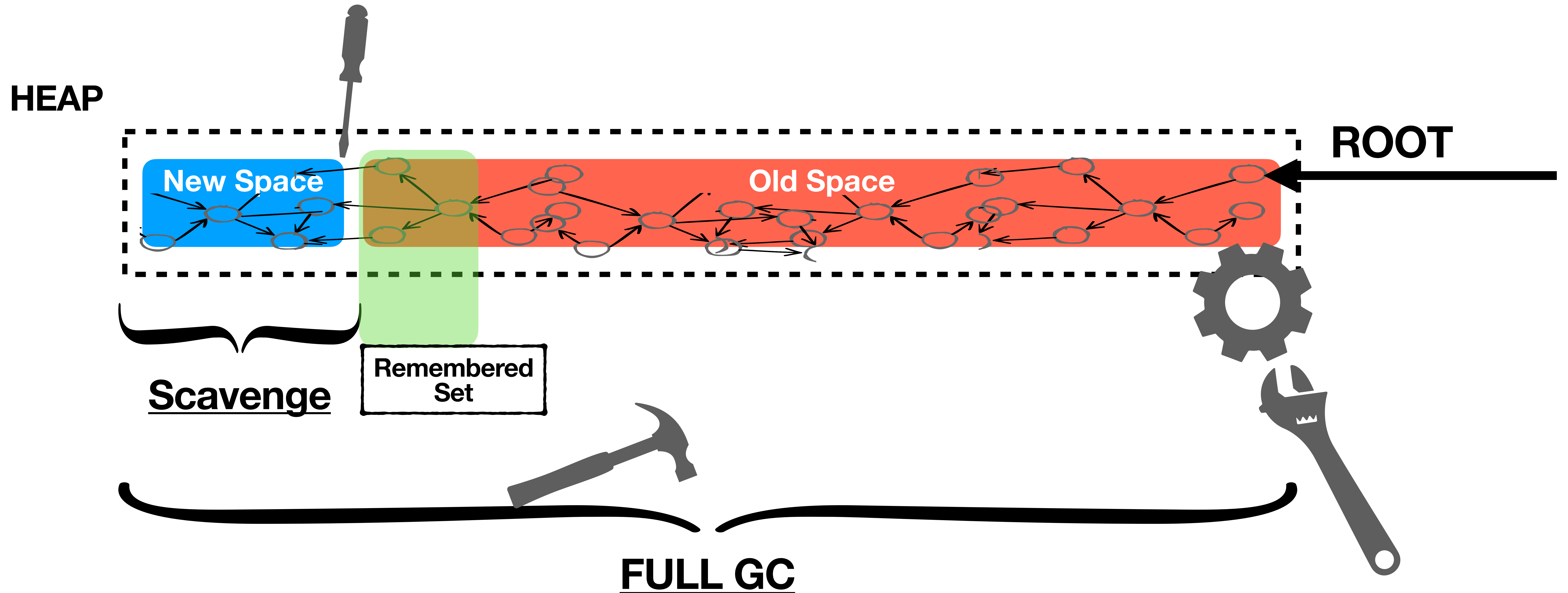
Pathological Allocation Pattern

Tuning the Garbage Collector



Pathological Allocation Pattern

Tuning the Garbage Collector



Pathological Allocation Pattern

Tuning the Garbage Collector

HEAP

New Space

Old Space

ROOT

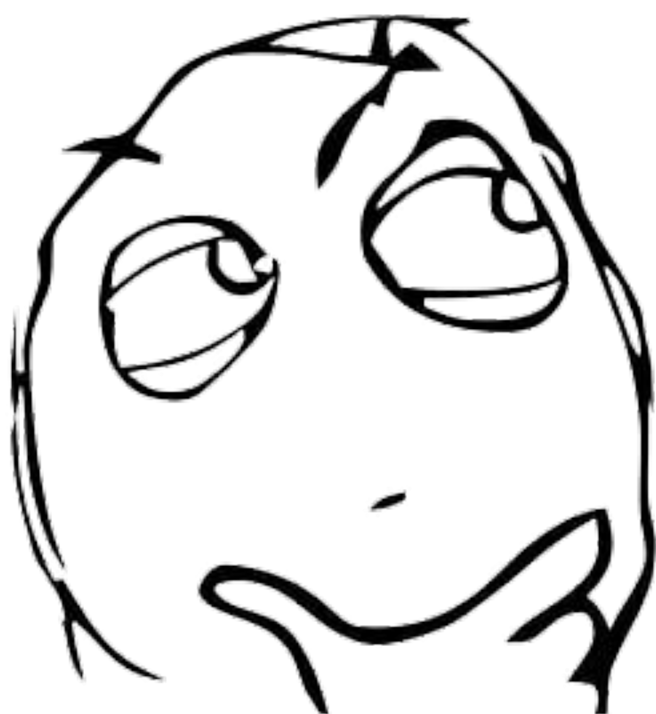
Scavenge

Remembered Set

FULL GC

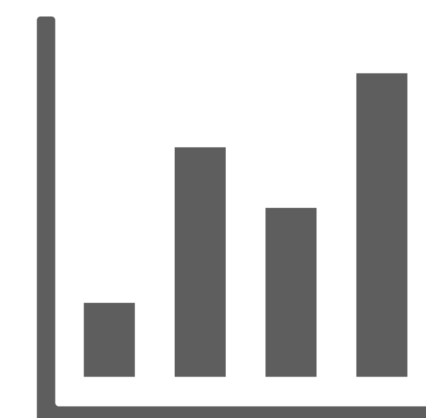
Developer

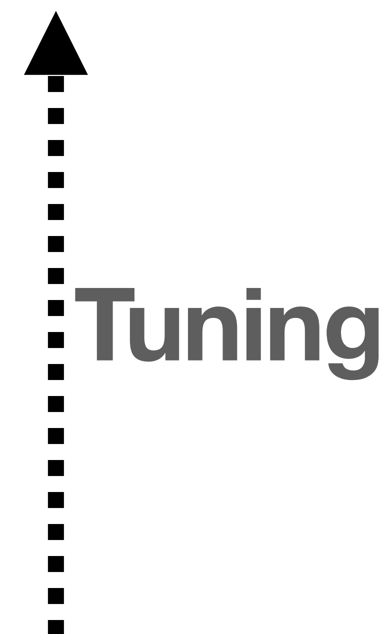
How should I tune the GC parameters for my application?

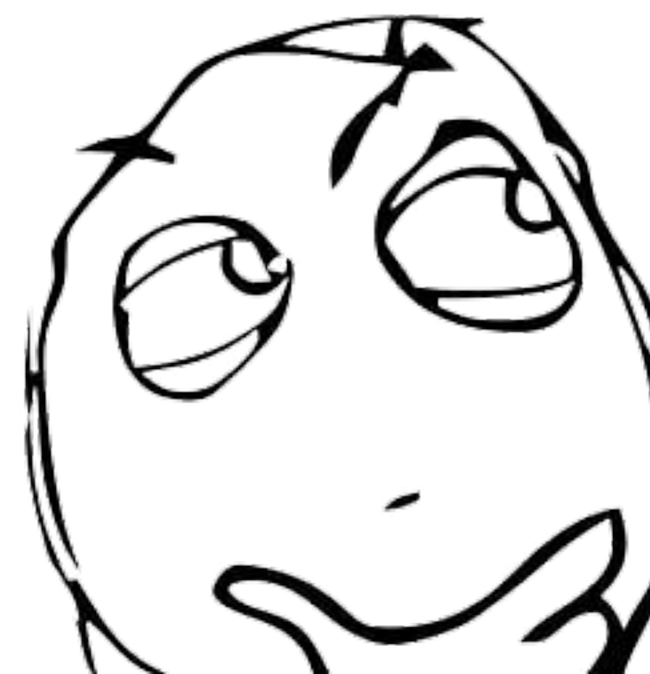


Our methodology for GC tuning

Profile GC events







From Scavenges:

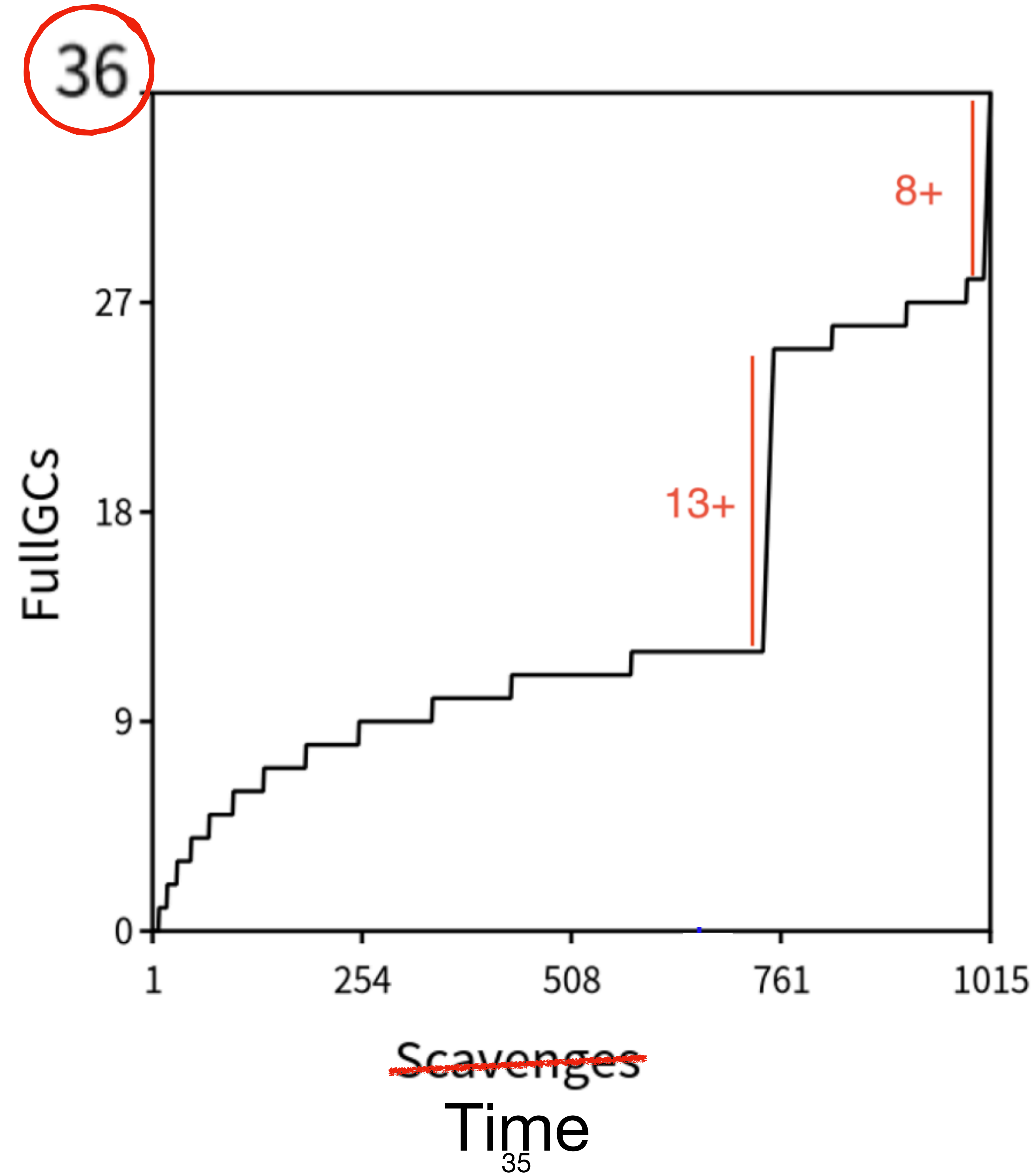
- Amount of memory used (before and after).
- Size of the Remembered Set (before and after).
- Tenuring info (amount of data - threshold).
- Executed time.

From FullGC:

- Time spent marking/sweeping/compacting.
- Executed time.

How Memory Grows

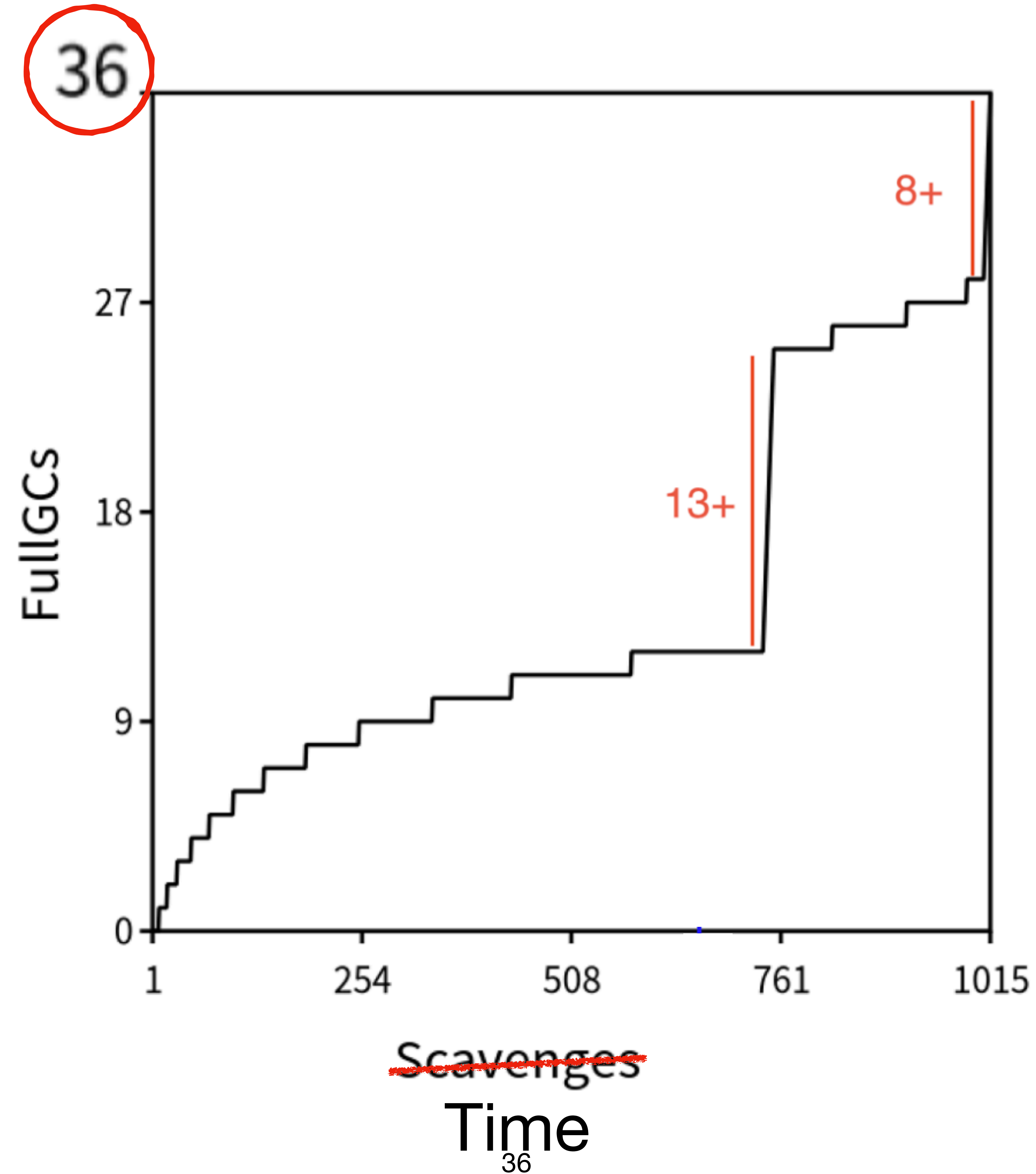
The overhead



How Memory Grows

The overhead

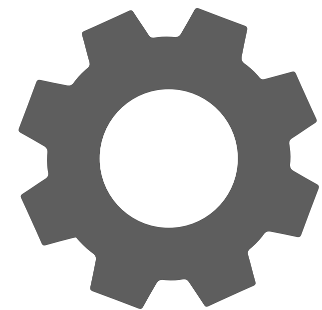
I run some FullGCs when memory grows so much



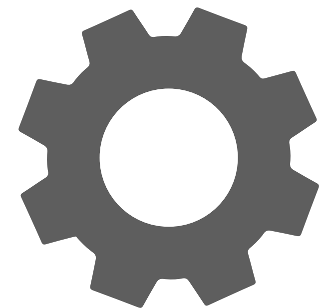
Don't do that

How Memory Grows

The tuning solution



FullGC Ratio - Threshold for triggering a FullGC when the old space grows more than expected



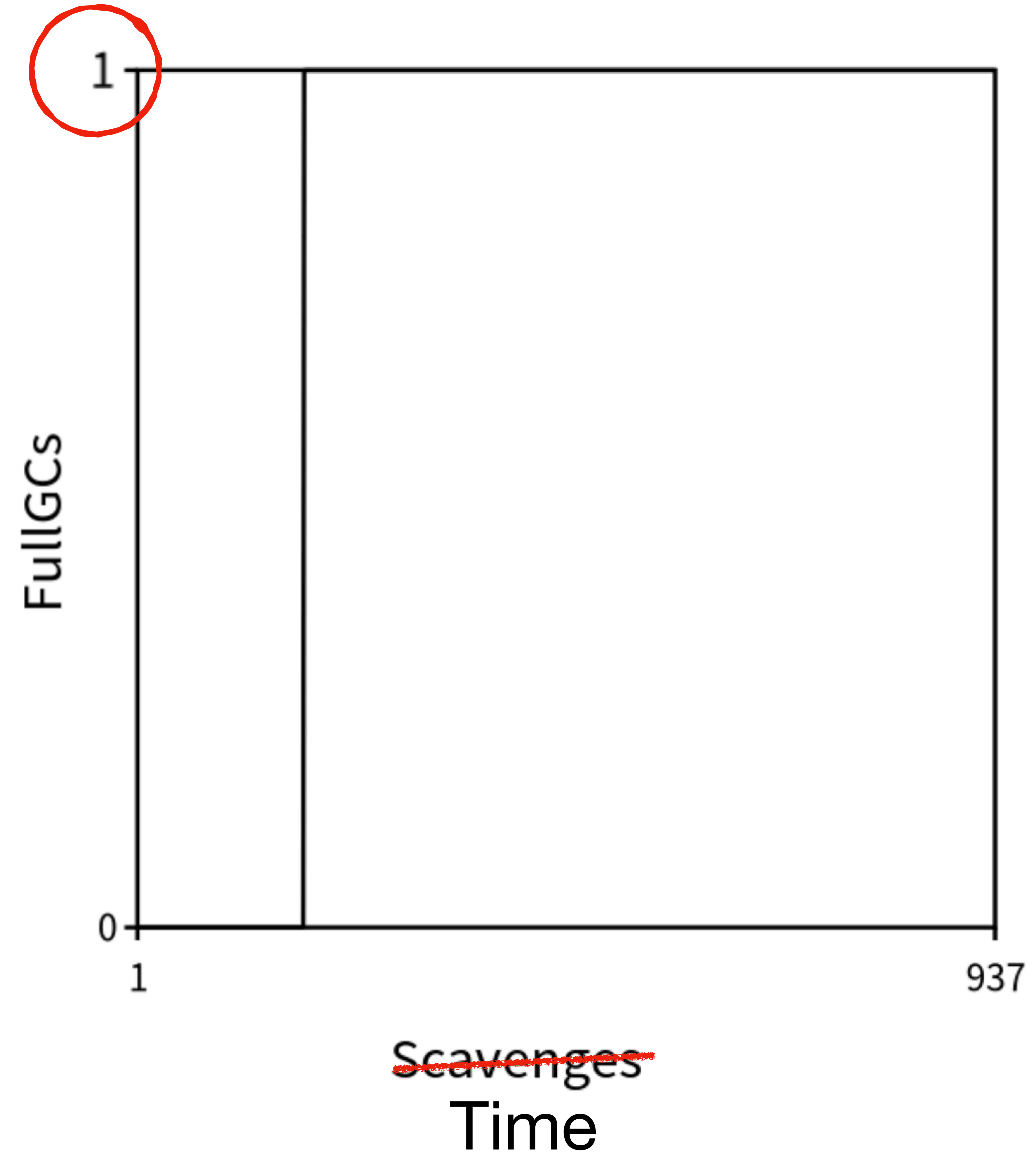
Grow Headroom - Minimum amount of memory that the GC will order from the OS

DataFrame

I will load 3GB of data

How Memory Grows

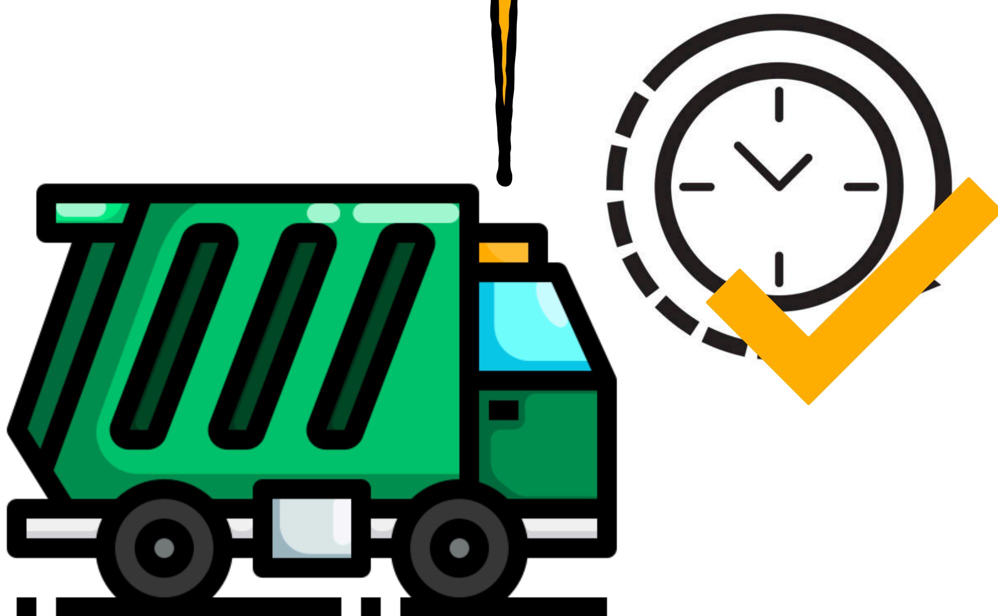
The tuning solution



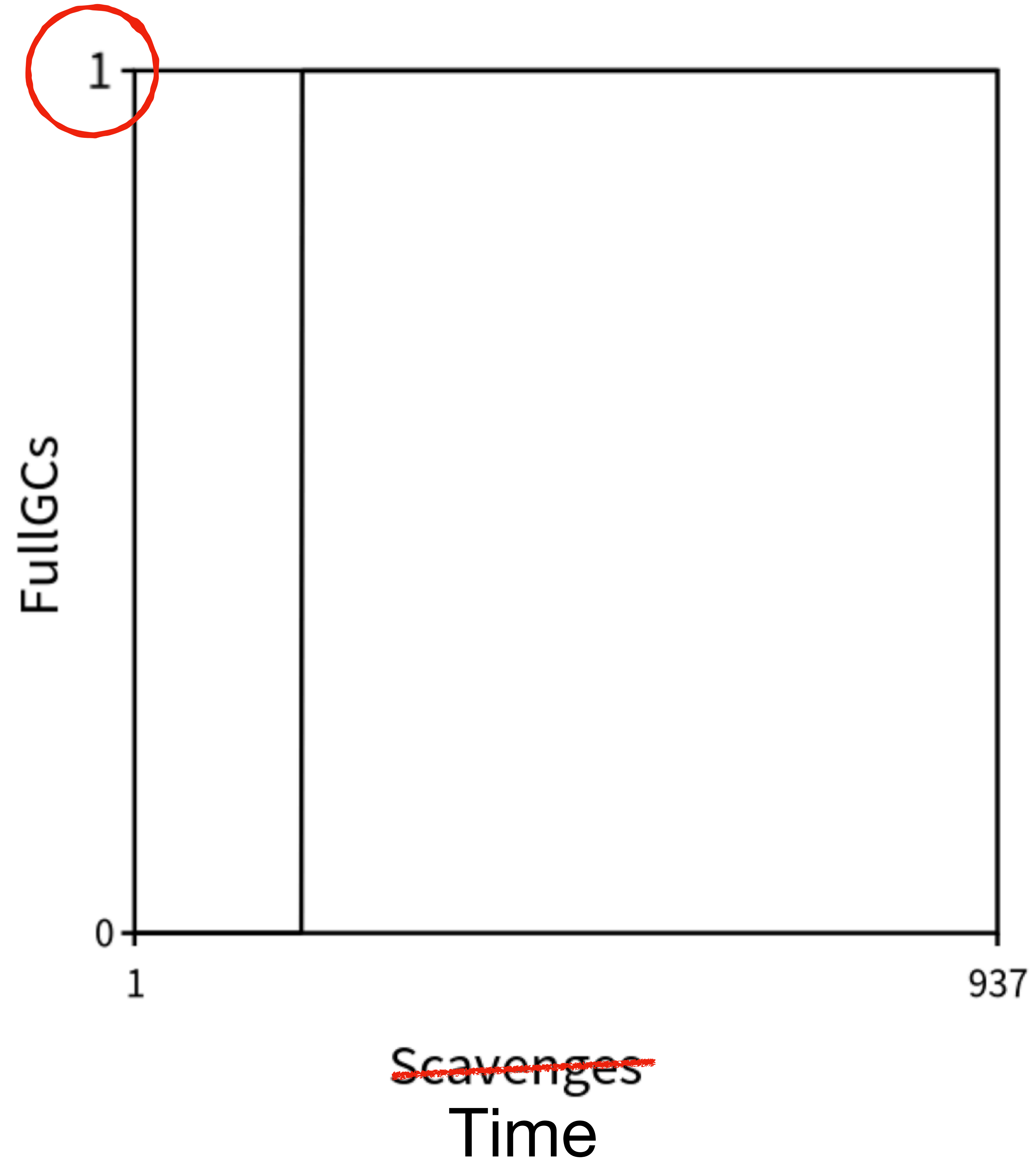
How Memory Grows

The tuning solution

1% - 5% faster!

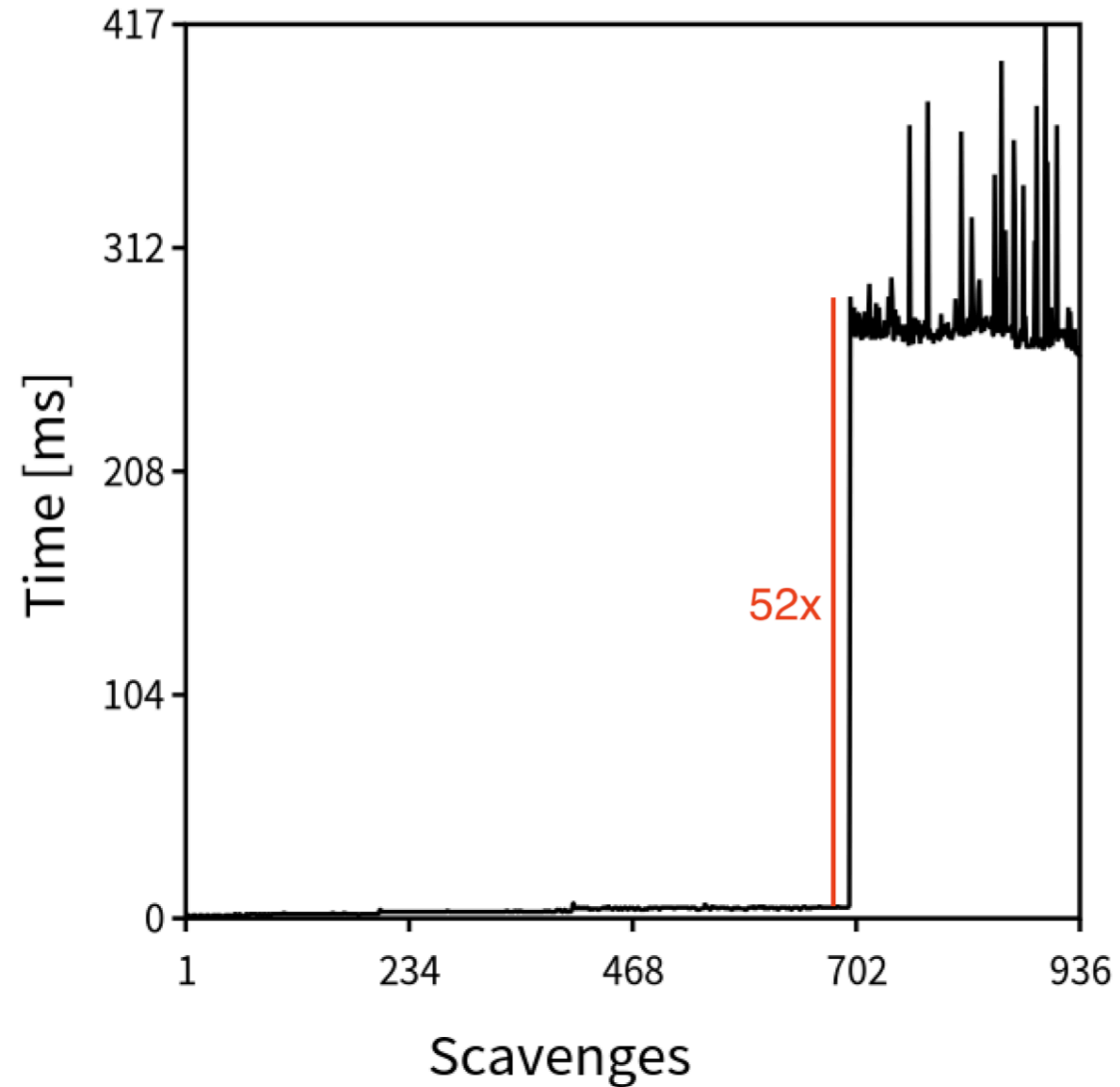


Just that?



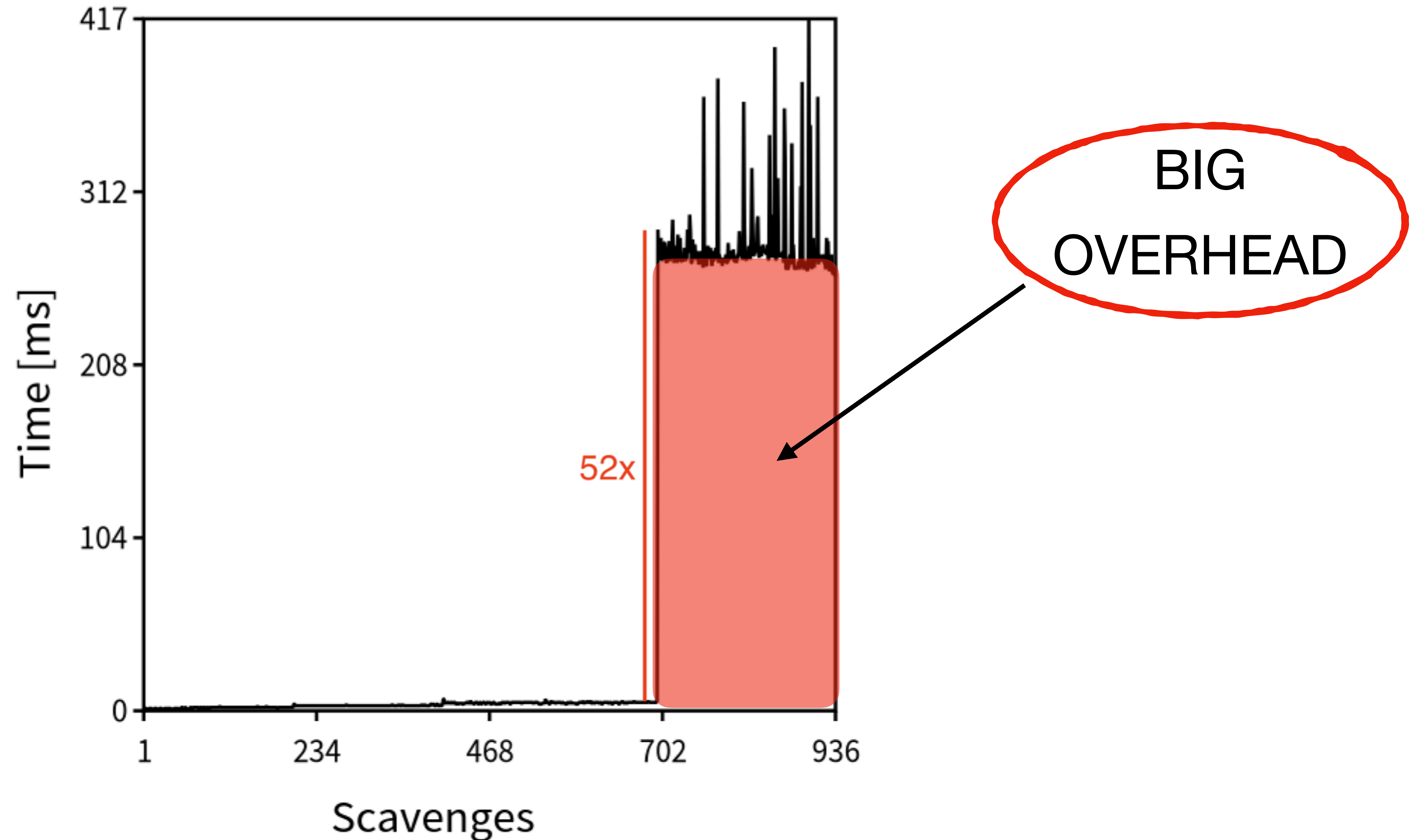
Deeper in the allocation pattern

Generational clash



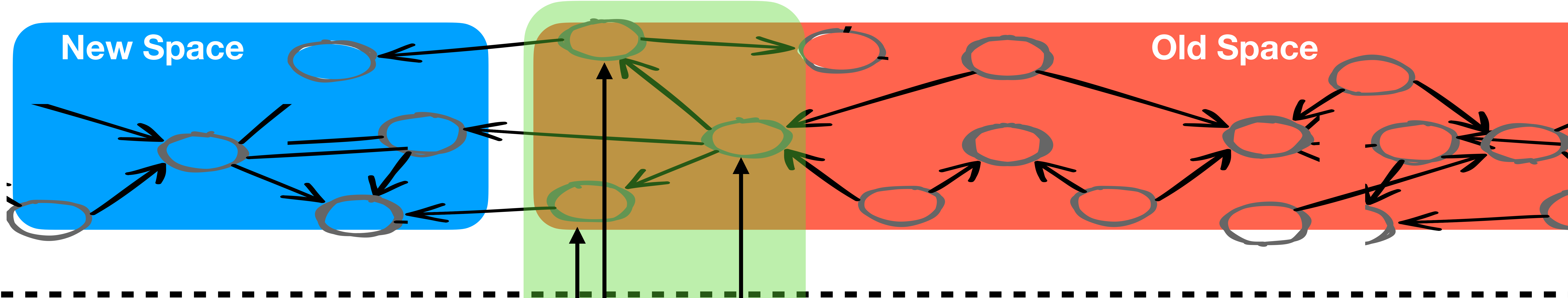
Deeper in the allocation pattern

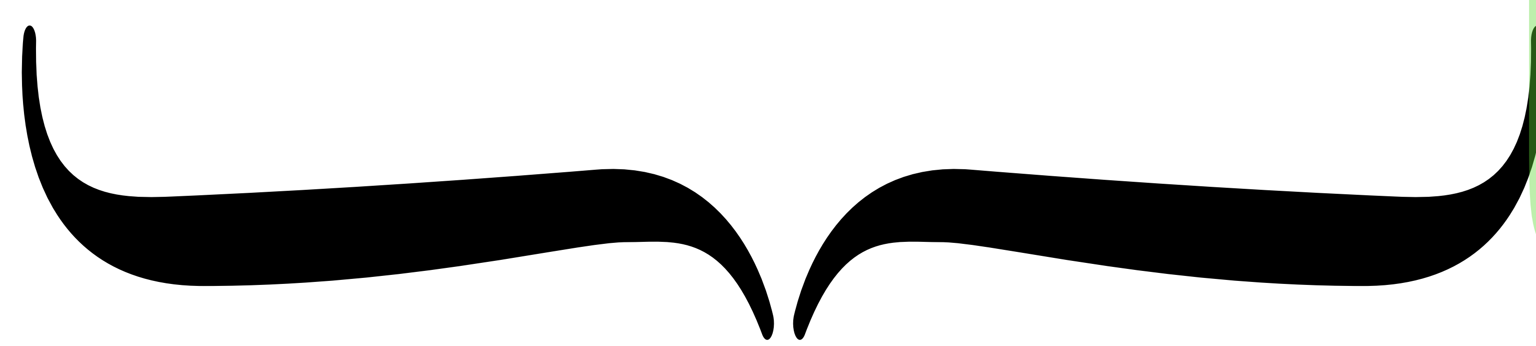
Generational clash



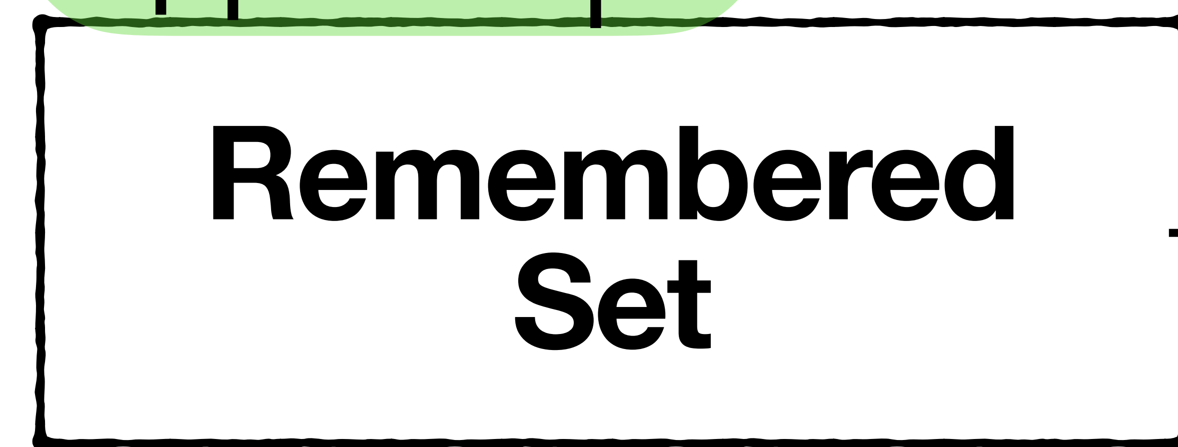
Deeper in the allocation pattern

Generational clash





Scavenge

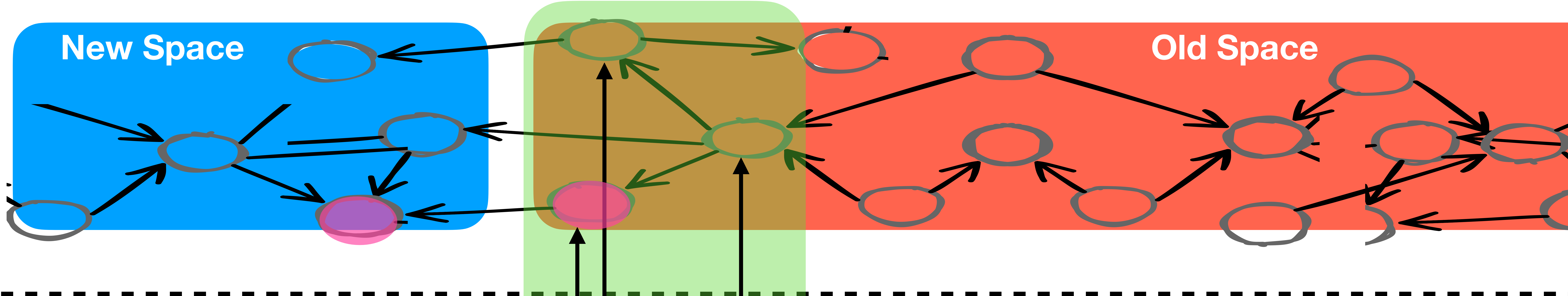


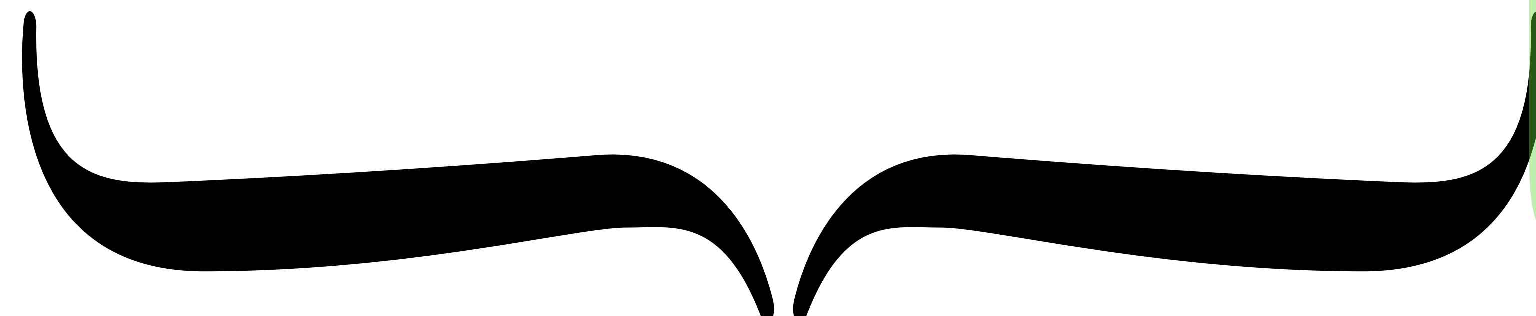
Roots of the New Space

Deeper in the allocation pattern

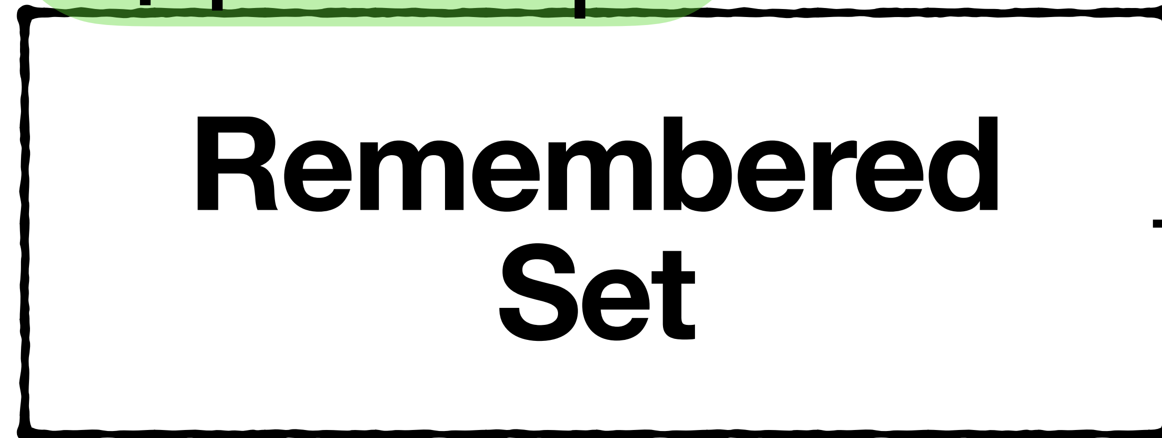
Generational clash

HEAP





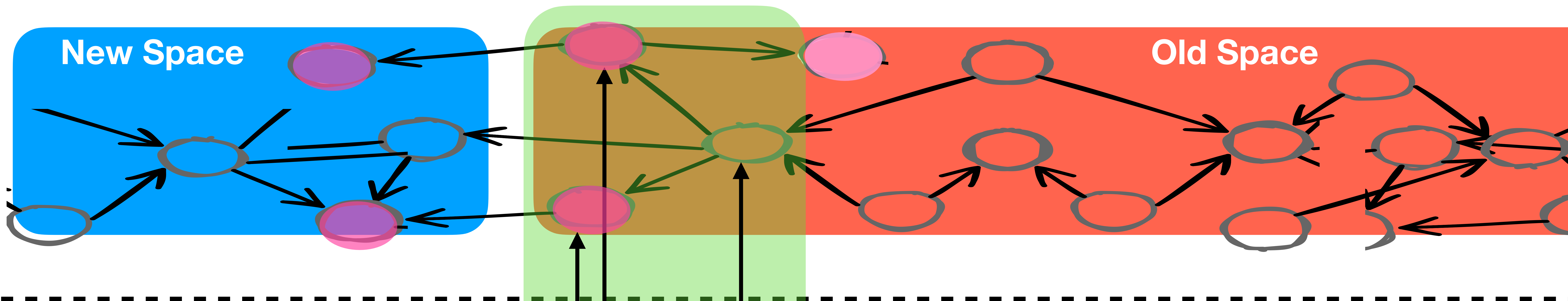
Scavenge



→ Roots of the New Space

Deeper in the allocation pattern

Generational clash



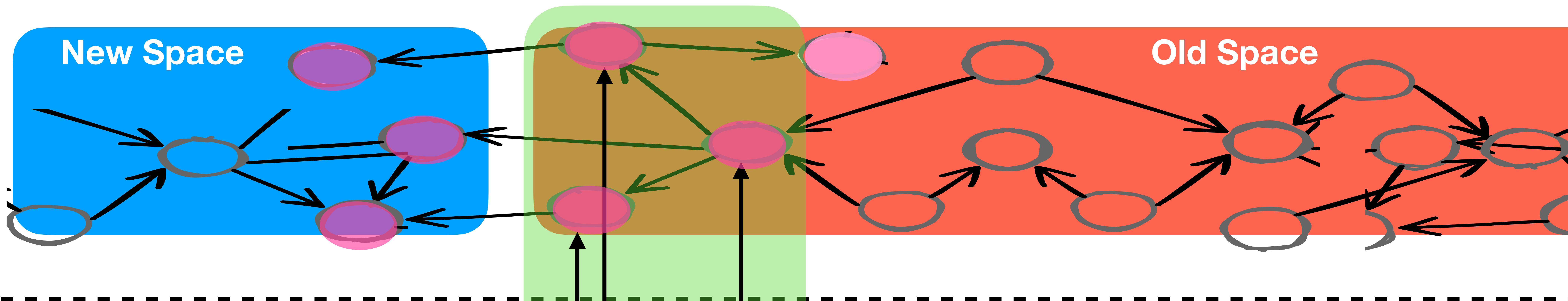
Scavenge

Remembered Set → Roots of the New Space

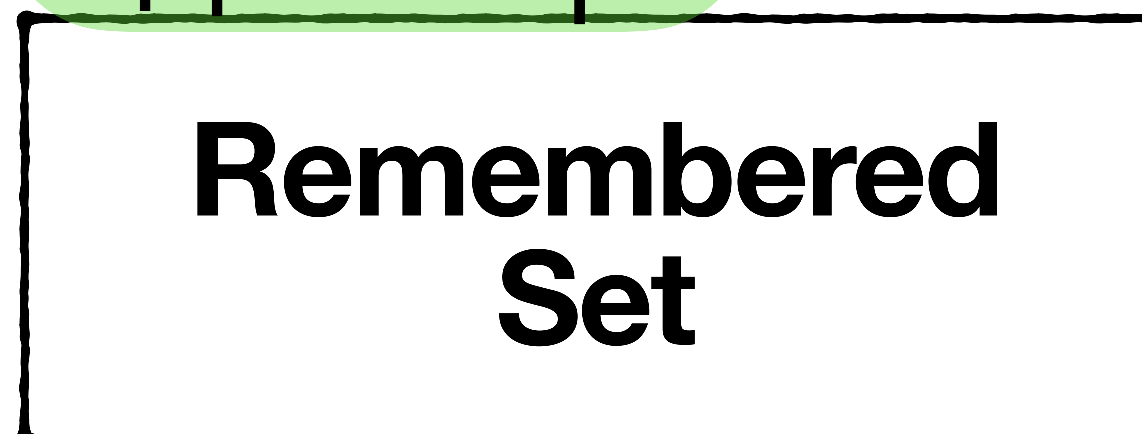
Deeper in the allocation pattern

Generational clash

HEAP



Scavenge



Roots of the New Space

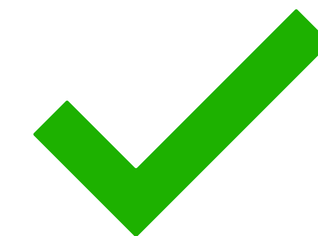
Deeper in the allocation pattern

Remembered Set overhead

The Remembered Set is large (lot of objects)

`* No chart :( *`

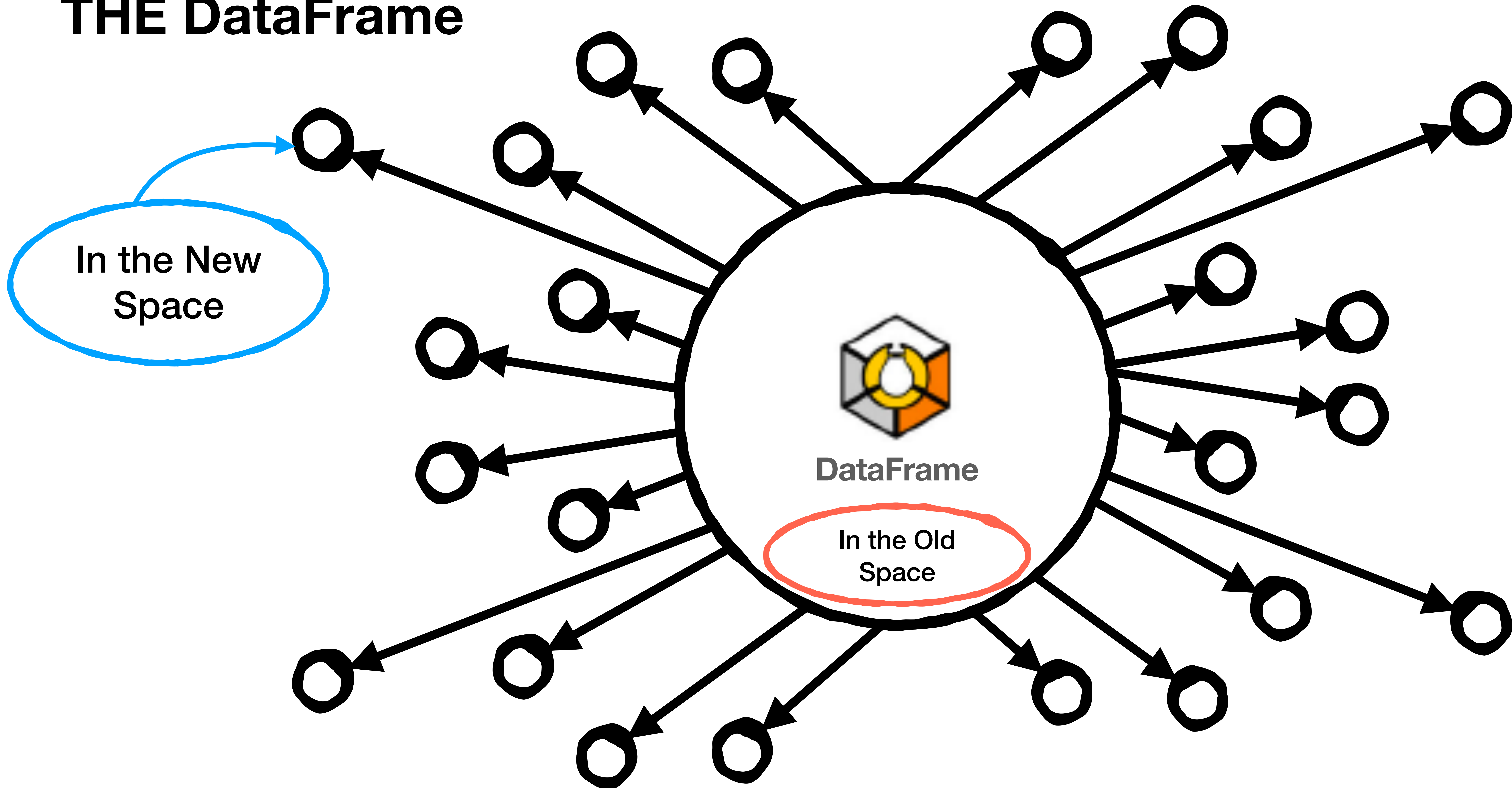
The objects in the Remembered Set are large



DataFrame

Deeper in the allocation pattern

THE DataFrame



Long Scavenges

The tuning solution

We need to avoid having large objects in the Remembered Set

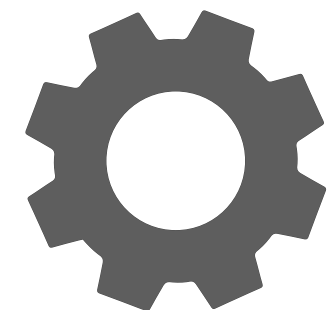
How? I don't have any algorithm for that

Then, close the New Space

Long Scavenges

The tuning solution

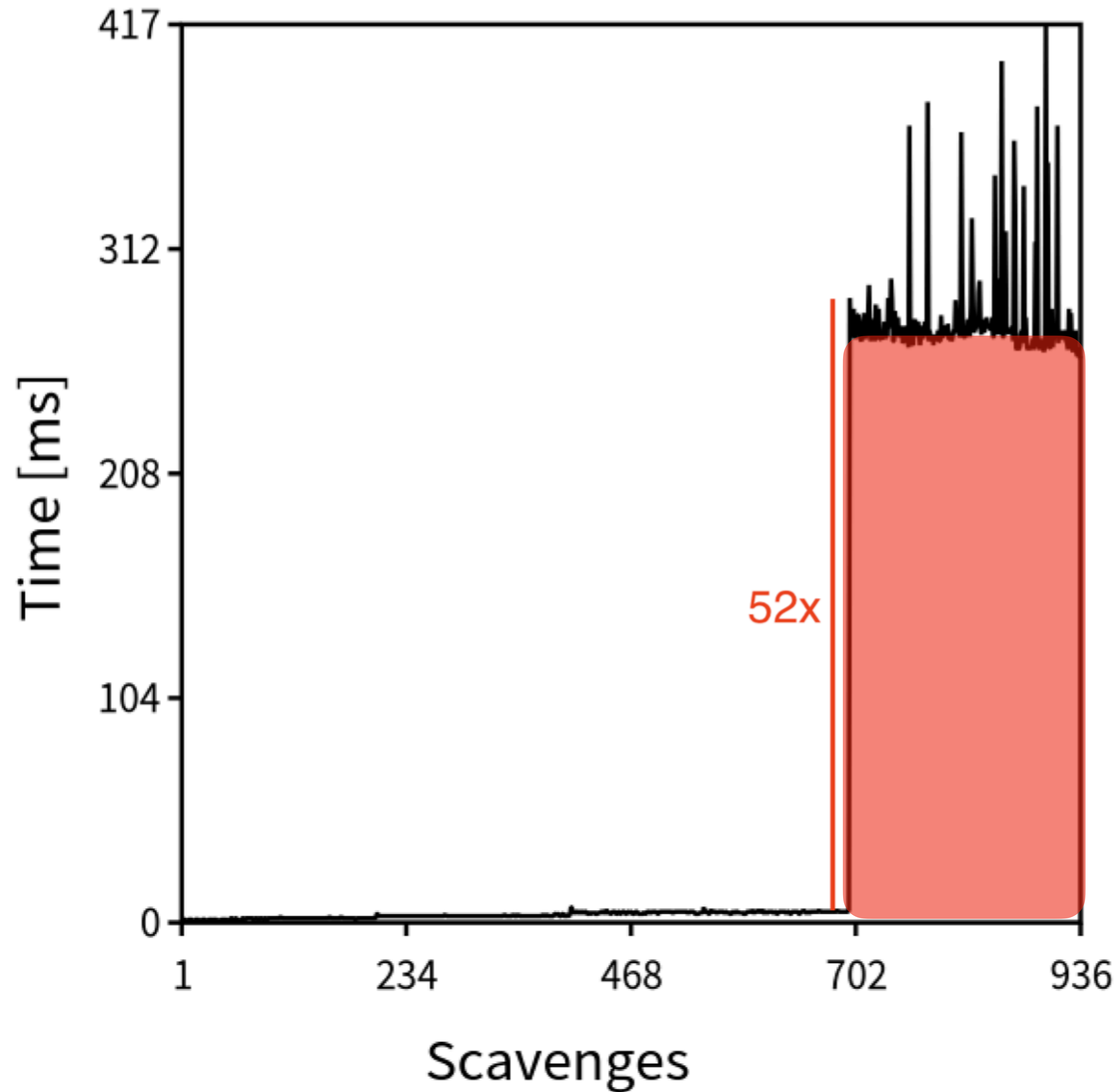
Close the door

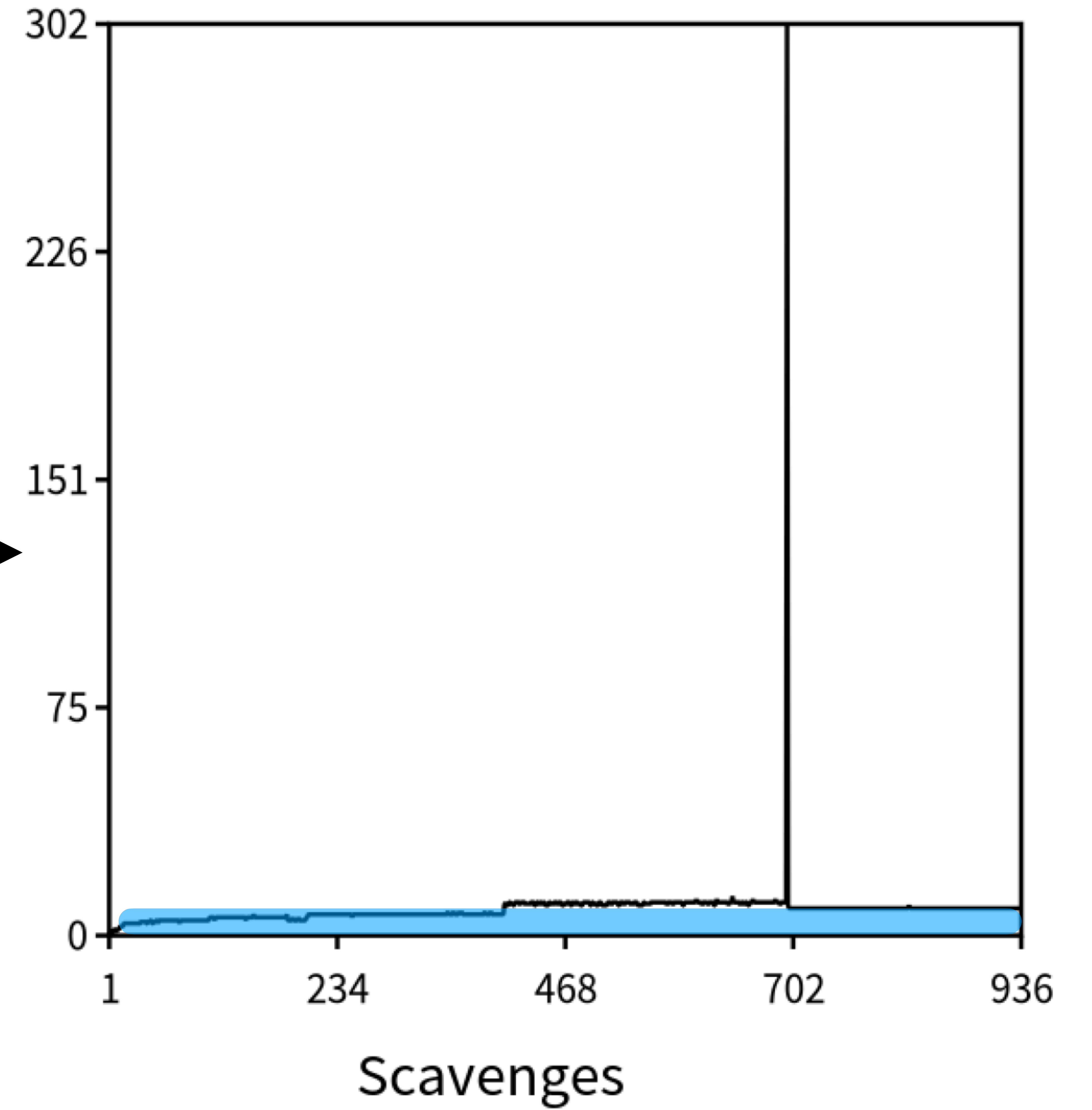


Tenuring threshold - Desired number of objects already in the New Space for tenuring to the Old Space

Long Scavenges

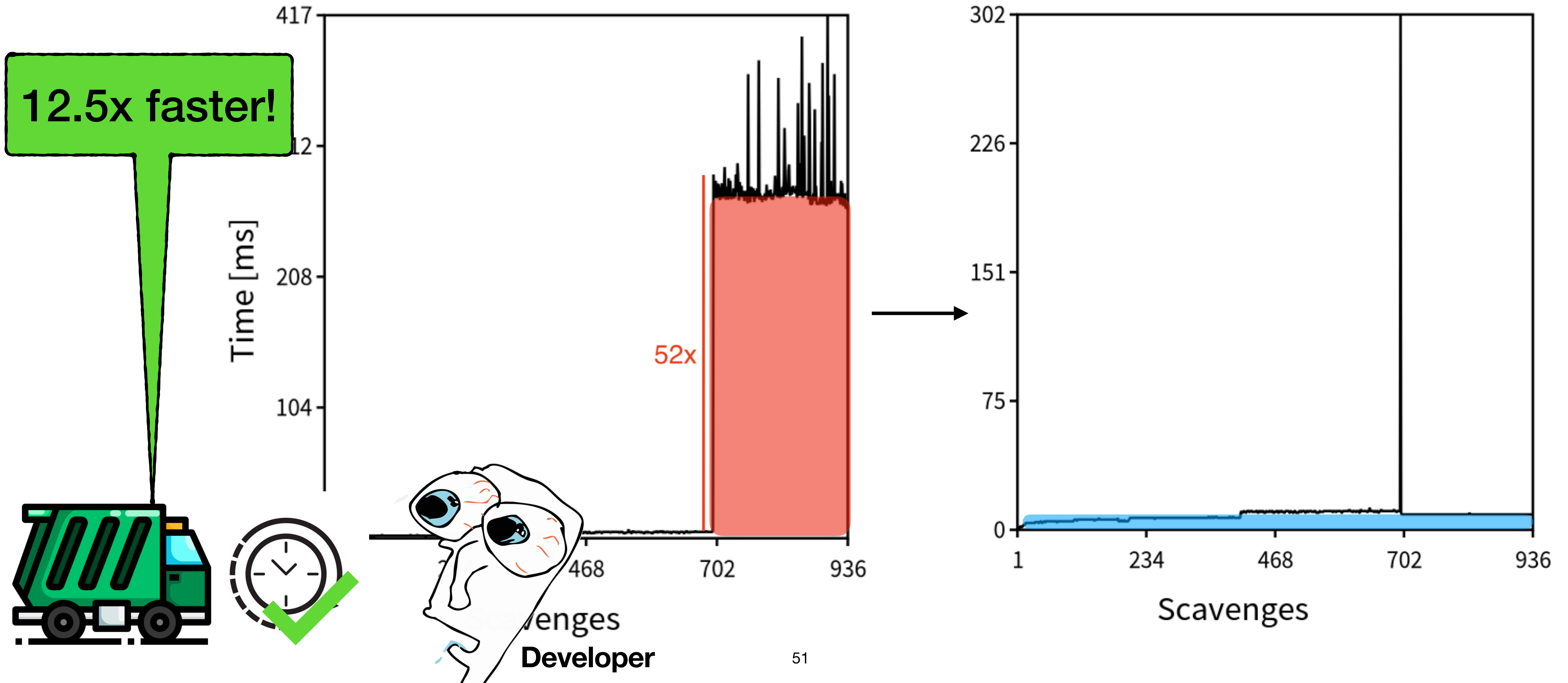
The tuning solution



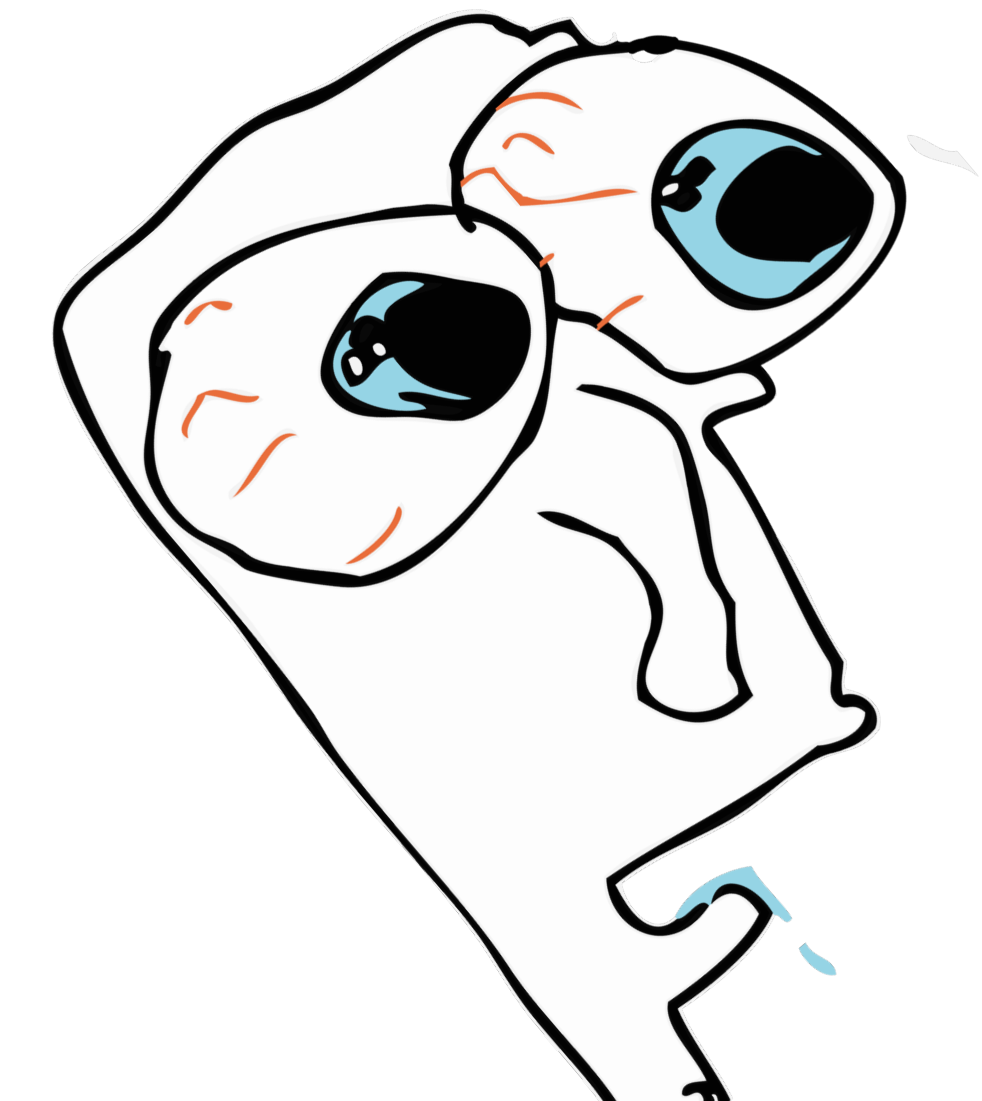


Long Scavenges

The tuning solution



Conclusions



Final result

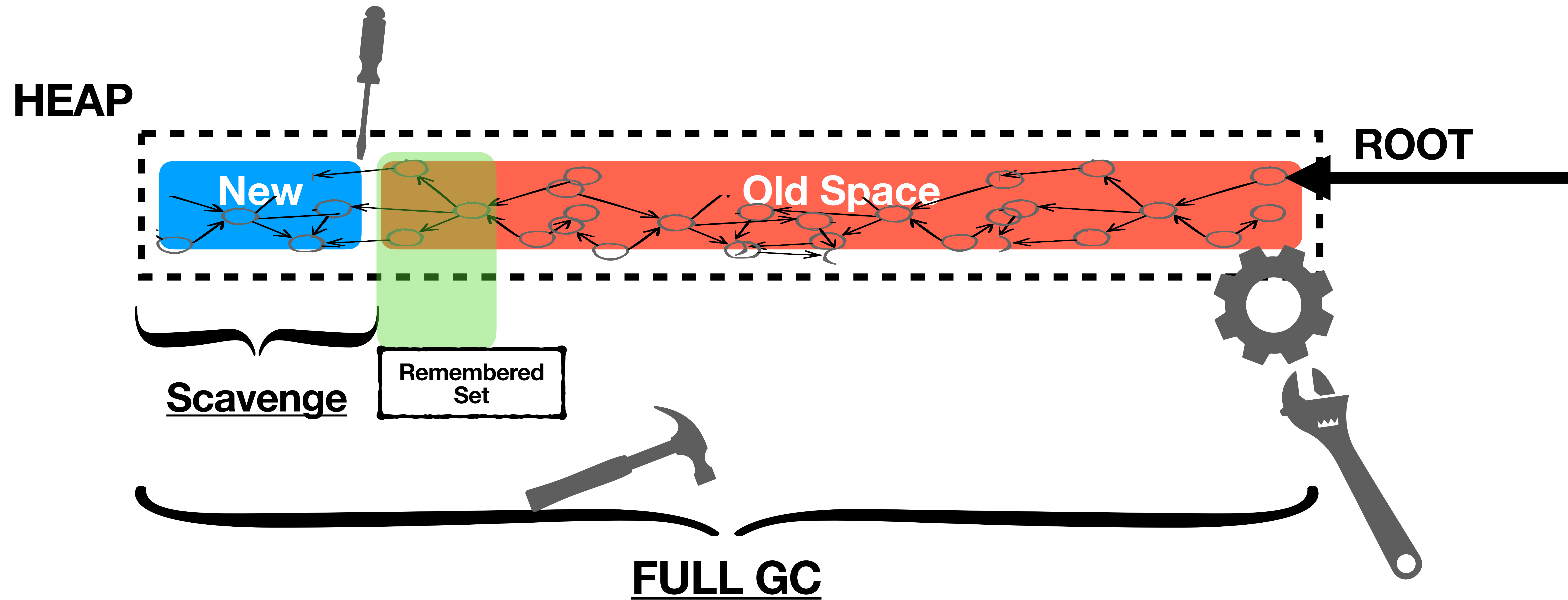
1) Have an infinite FullGC ratio To reduce the number of FullGC when the Old Space grows.

2) Have a grow headroom equal to the loaded file To avoid many FullGC together.

3) Keep all survivors in the semi-space To tenure new objects to the Old Space quickly.

Data size	Total secs before	GC overhead before	Total secs after	GC overhead after
529 MB	43	16%	37 (1.1x)	5% (3.2x)
1.6 GB	150	25%	122 (1.2x)	7% (3.6x)
3.1 GB	5599 >1h30m	**92%**	440 (12.5x) ~7mins	**24%** (3.8x)

Conclusions

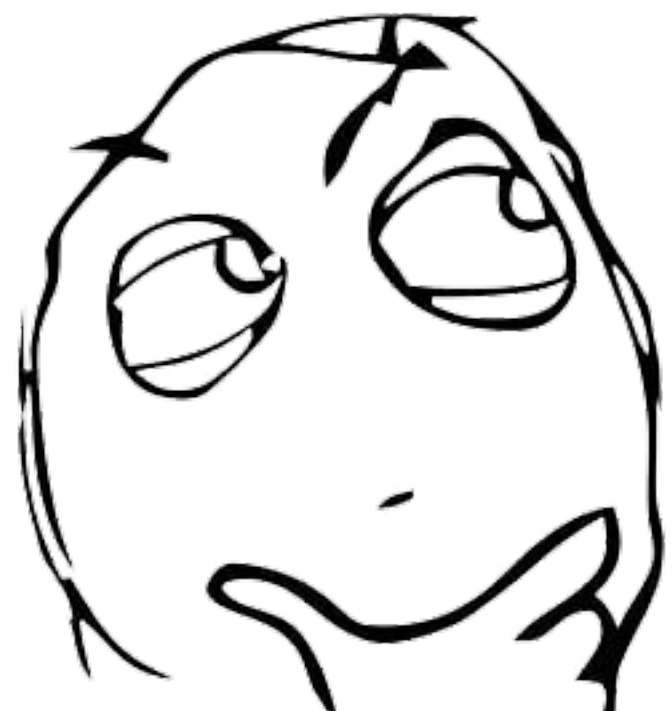


Conclusions

Conclusions

My questions

- How much should devs know about their applications?
- How much should devs know about Garbage Collection algorithms?
- How much should devs know about the running VM?



Conclusions

Conclusions