What’s So Hard About Pinning?

Maoni Stephens

dotnetos 2020

Basics of pinning

- What's pinning?
  - Make the GC not move an object
- How to pin?
  - Pinned GC handle
  - fixed keyword

Fundamentals

- How does the GC discover an object is pinned?
  - Mark phase –
    - Start from user roots - stack, GC handles and finalize queue
    - These roots can report a pinned object
      - GC then sets a bit on the object header
  - Plan phase –
    - Checks the bit and…

But wait, what is the plan phase?

- Plan phase is a simulation of the compact phase
  - Meaning we calculate where to move objects to, if we compact
- Plan phase goes through the condemned portions of the heap
  - “Condemned portion” is very important!
  - “What does being a generation GC imply”

The generational aspect

- Applies to all phases of the GC
- Some root reporting is generational
  - Handles are generational, stack is not.
  - Will be filtered as soon as it comes into the GC

Plan phase – cont.

- Checks 2 bits – the mark bit and the pinned bit
  - When it encounters a pinned object it will record info on a side data structure (pinned queue)
- When calculating relocate address it knows to not move pinned object
  - However, plan phase operates on plugs, not objects

Plugs

- What’s a plug?
  - A plug is an adjacent group of live objects
  - Space in between is called a gap
  - Explained in this short video; more implementation details in the Pro .NET Memory book.
- Plugs were used for performance reasons
  - We can calculate reloc address on a plug instead of an object
  - We can take advantage of the dead object space inbetween to store the reloc address

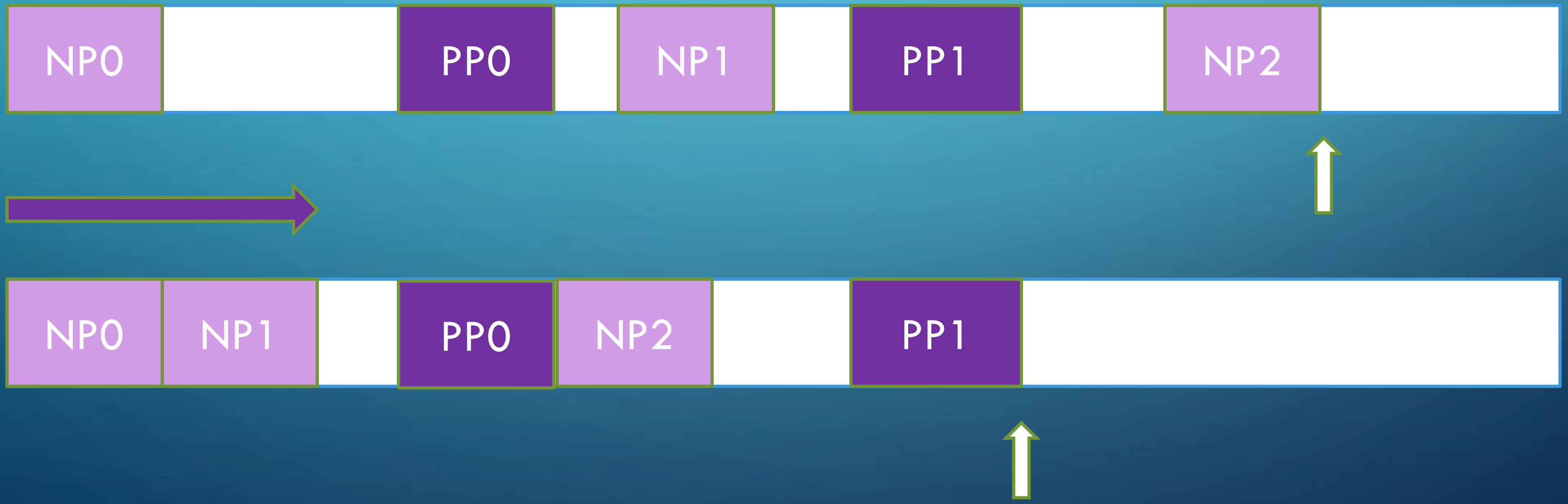
Plugs – cont.

- Great for perf but with a limitation
  - We had to pin the whole plug if at least one obj is pinned
- What’s the performance consequence of this?

Performance implications of pinning

- Heap size
  - GC cannot move pinning plugs
  - GC can compact plugs inbetween pinned plugs

NP – NonPinned Plug; PP – Pinned Plug

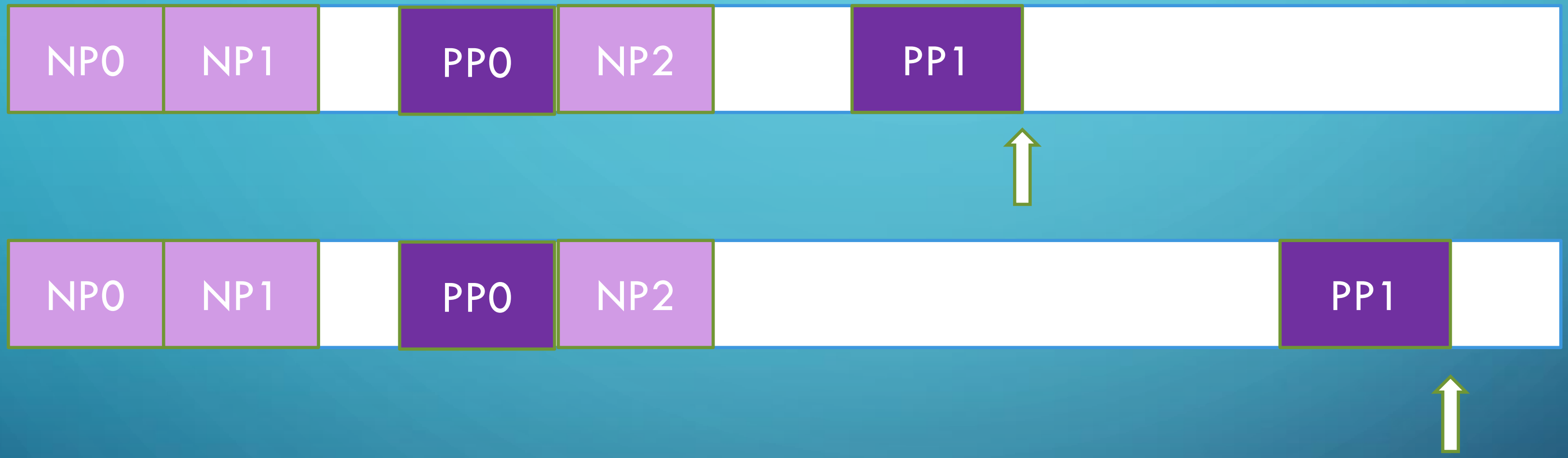


Perf implications – cont.

- Heap size
    - Heap can become fragmented
    - However pinned objects are limited to user roots
- So do we not have a problem if there aren't too many pins?

Perf implications – cont.

- Even if we don't have too many they can still be scattered



This is why we suggest to pin a batch of objects and pin early, if you can

The generational aspect of pinning

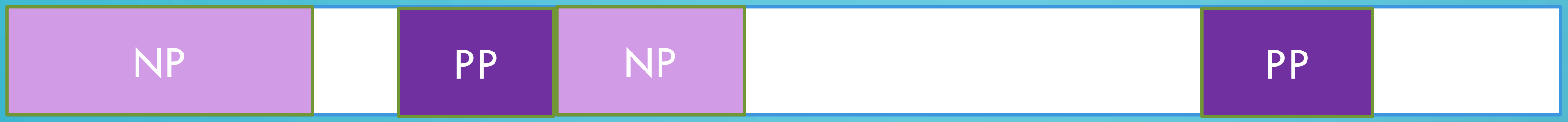
- We compact the whole heap very infrequently
- What matter is when GC sees pins in the condemned generations
- If GC does not observe it, it doesn't matter if it's pinned
  - This implies what GC observes during ephemeral collections is very important
  - But if you do run in high memory load situation, full compacting GCs may occur

Full blocking GCs in high mem situation

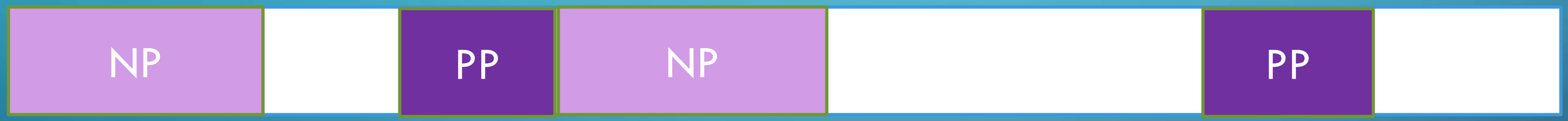
- Pinning makes estimating how much we can shrink the heap harder
- We can’t really predict the pinning situation
- We could end up triggering frequent full blocking GCs

Previously

after gen2



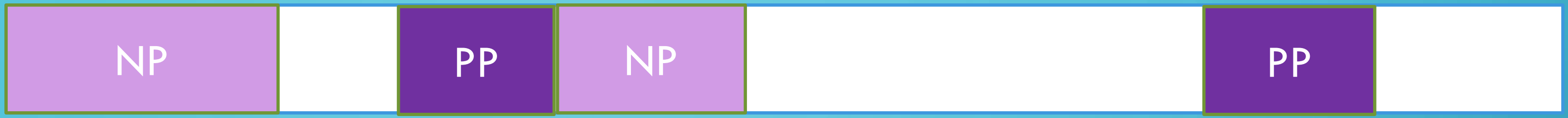
Soon after GC detects “enough fragmentation in gen2” so does another gen2, on entry of this gen2



This means we didn’t use the free space efficiently

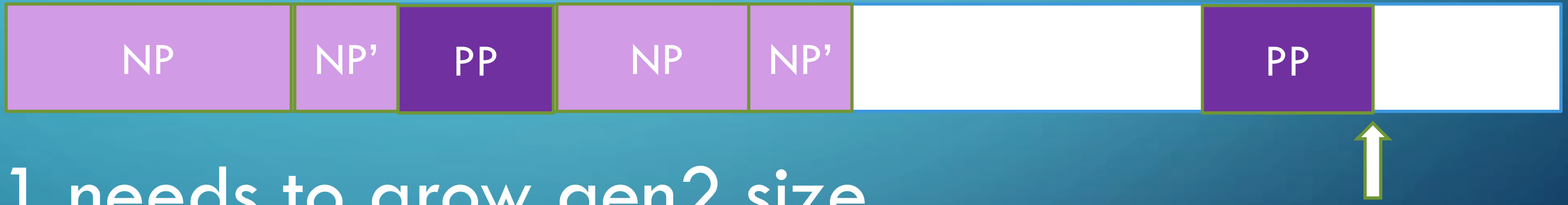
With provisional mode

after gen2

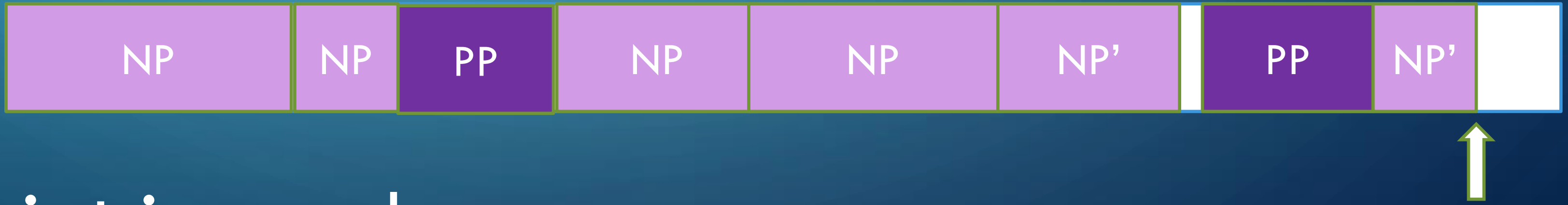


GC detects "after a compacting gen2 we still have high fragmentation in high mem situation" → provisional mode on

after gen1 – NP' is what this gen1 promoted



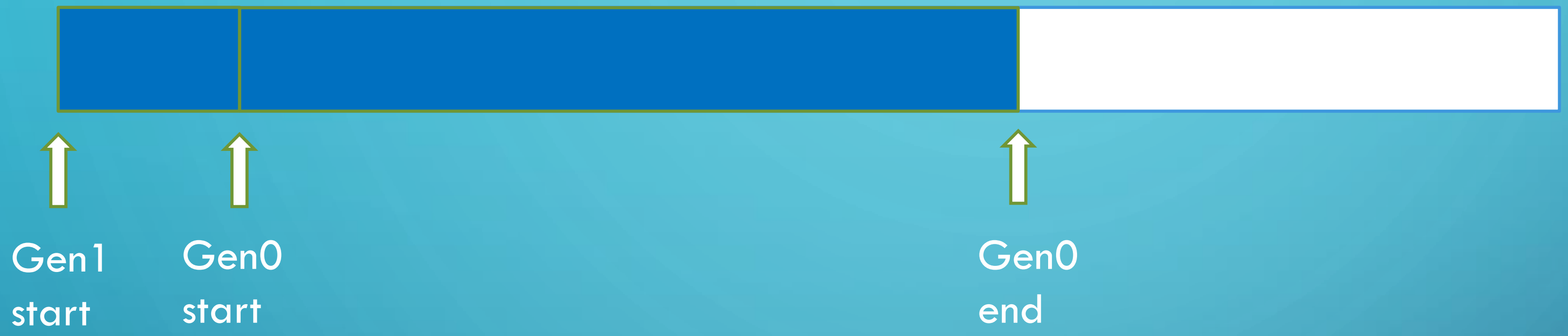
… till a gen1 needs to grow gen2 size



then a gen2 is triggered

Pinning during ephemeral GCs

Before a gen0 GC

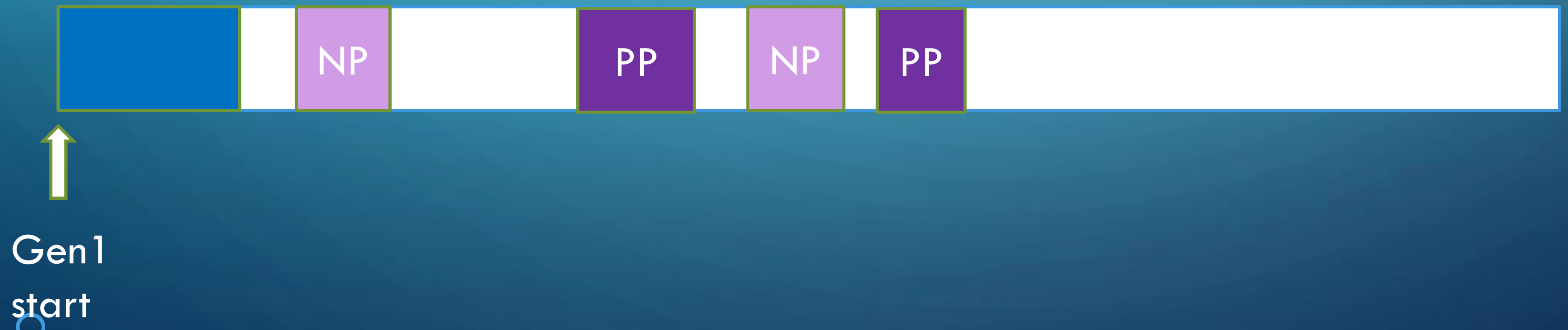


Gen1 start

Gen0 start

Gen0 end

Plugs formed during the plan phase



NP

PP

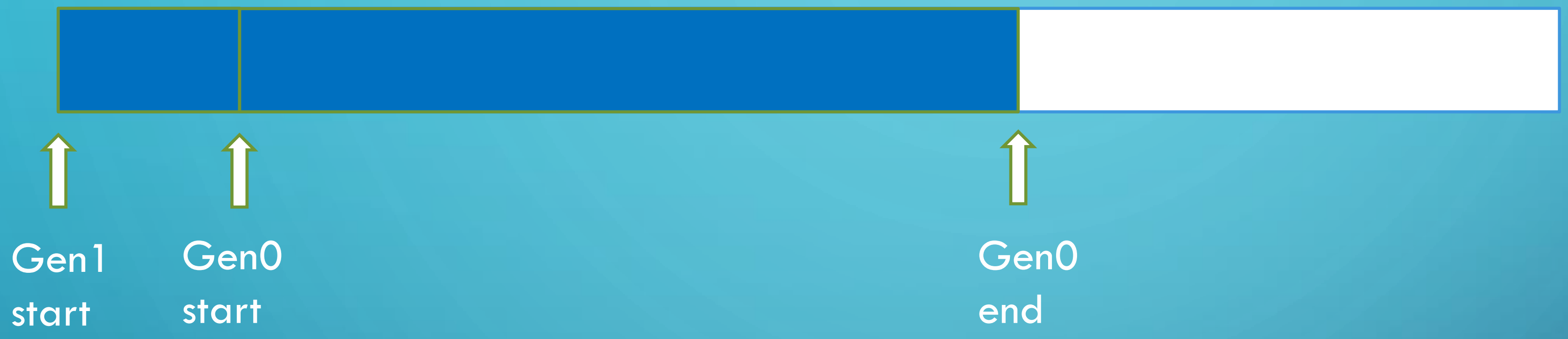
NP

PP

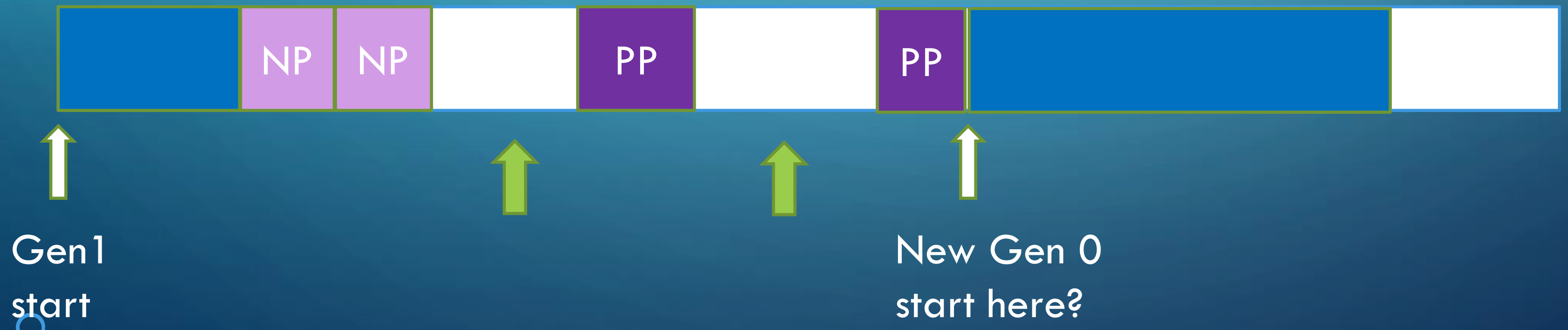
Gen1 start

Pinning during ephemeral GCs

Before a gen0 GC

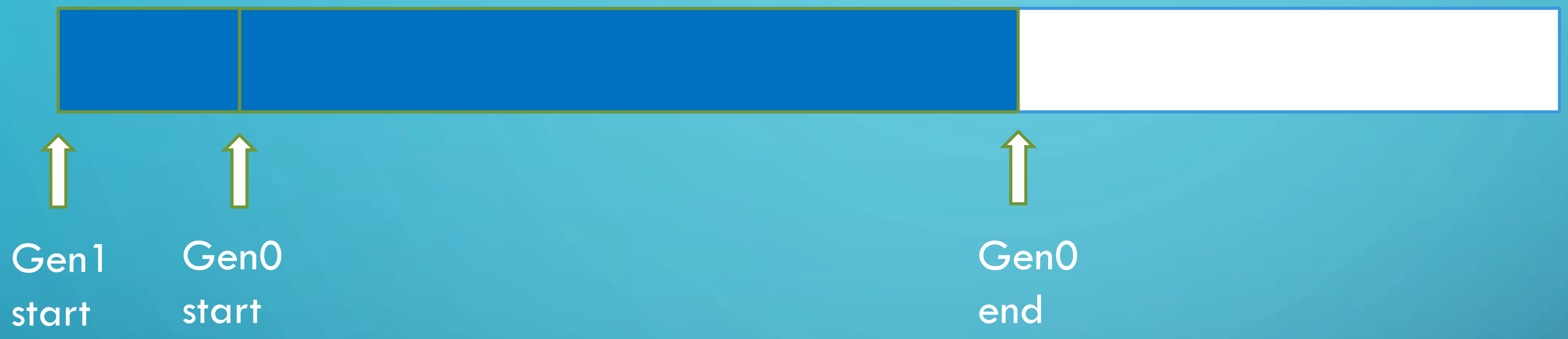


Plugs formed during the plan phase

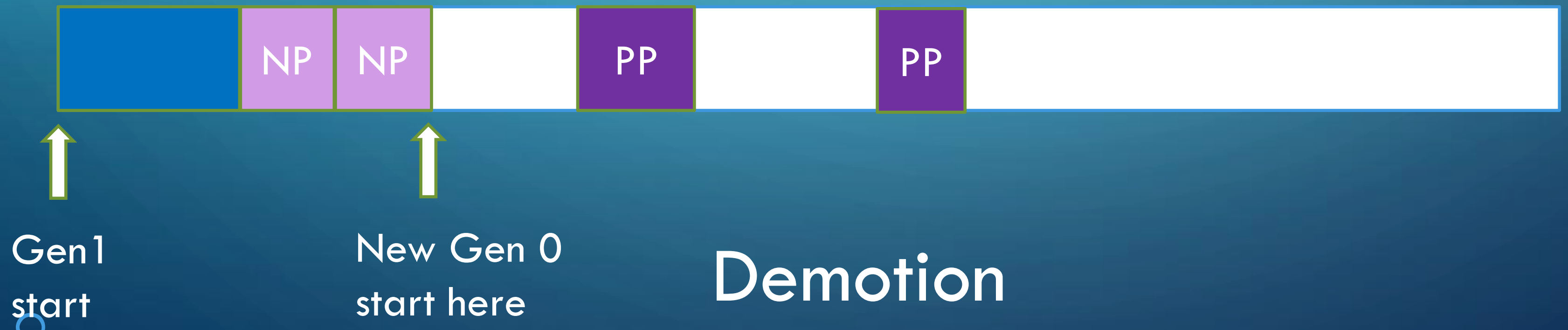


Pinning during ephemeral GCs

Before a gen0 GC



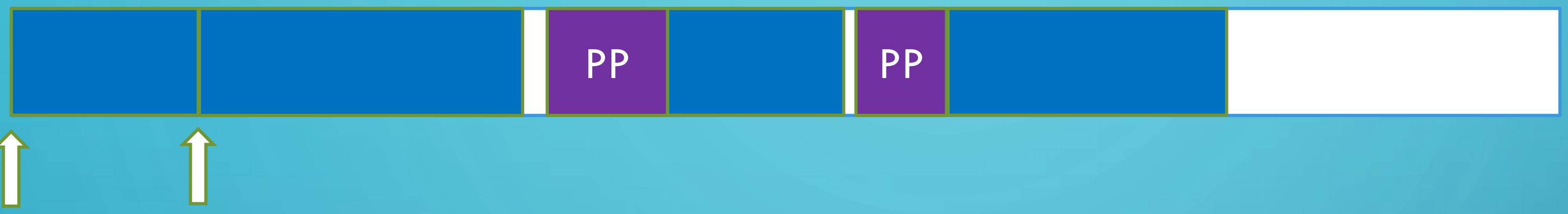
Plugs formed during the plan phase



Demotion

Is our problem solved?

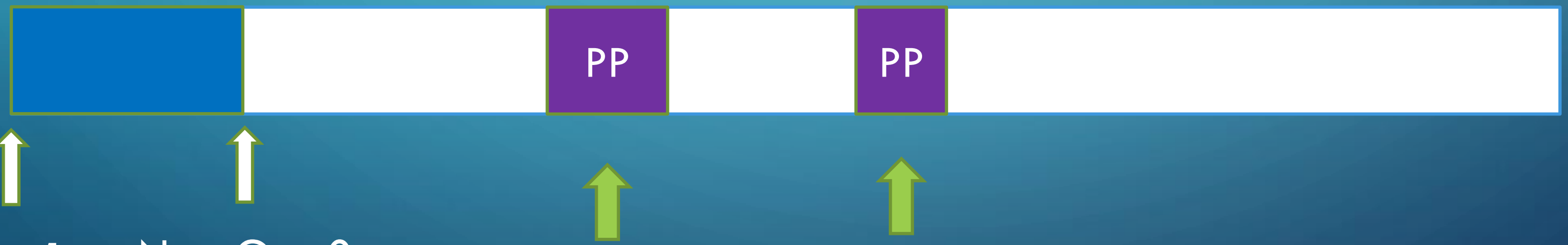
After a gen0



Gen1
start

New Gen 0
start here

After another gen0



Gen1
start

New Gen 0
start here

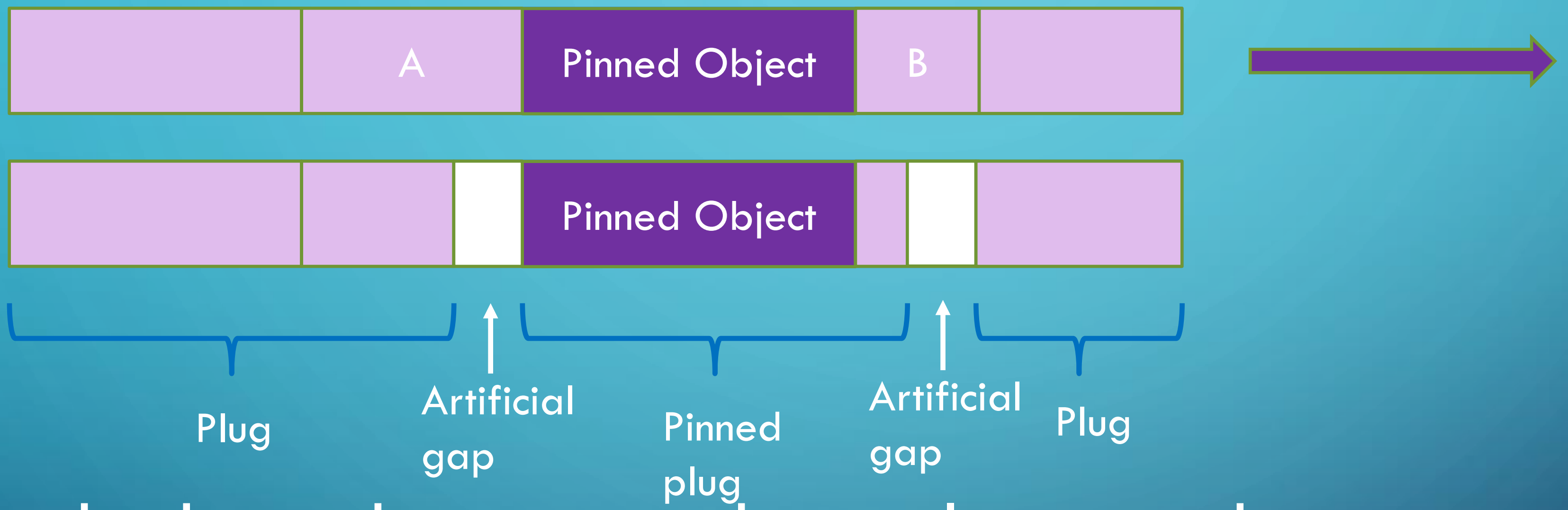
What if these pinned plugs are big
and stay for many GCs?

This was a real problem

- Introduced the POPO (Promote Only Pinned Objects) feature
- For the first time in the .NET GC history we broke plugs apart
- Was very difficult to do but yielded significant perf benefits

Some of the reasons why it was difficult

- The idea is to break the plug up into 2 or 3 plugs but still make each one look like normal plugs



- Each phase that cares about plugs need to recognize this
- We need to record the info we overwrite with artificial gaps
  - What if A or B is too small

Root cause for all these perf issues

- We have no control over which objects get pinned
- This was a very conscious decision at the beginning of the runtime – to make interop fast
- We cannot break this contract
- However, we can provide a mechanism to help users organize these objects if they have control

POH (Pinned Object Heap)

- Introduced in .NET 5.0
- For the scenarios when you know you will pin objects when you allocate them
- They will stay on their own segments
- They are swept in gen2 GCs

Connect with me for GC resources

- .NET Memory Performance Analysis doc (new)
- My blog
- https://www.youtube.com/MaoniStephens (new, short videos)
- https://twitter.com/maoni0 (I don't look at twitter constantly)
- File an issue at the runtime repo