Channels and goroutines in Go are designed to be lightweight abstractions that consume minimal memory compared to other concurrency mechanisms. However, they do have some memory impact. Channels:

  - The memory impact of a channel depends on whether it is buffered or unbuffered.
  - An unbuffered channel typically requires a small fixed amount of memory for bookkeeping purposes, regardless of the data it holds. This memory overhead is usually negligible.
  - Buffered channels, on the other hand, have additional memory overhead proportional to their capacity. Each buffered channel element consumes memory to store the value, so a larger buffer size will consume more memory.

- Goroutines:
  - Goroutines are very lightweight in terms of memory consumption. Creating a new goroutine typically requires only a few kilobytes of memory.
  - The initial memory overhead of a goroutine is small, but it can grow slightly as the goroutine uses stack space during execution. However, this growth is usually minimal and controlled.
  - Goroutines share the same address space and heap, so they can be created and destroyed efficiently without incurring significant memory overhead.

Overall, channels and goroutines are considered lightweight constructs in Go. They are designed to support concurrent programming with efficiency and scalability in mind. However, the actual memory impact of channels and goroutines in a program depends on factors such as the number of goroutines, the size of buffered channels, and the amount of data being processed. It's always a good practice to profile your

application's memory usage to ensure it meets your specific requirements and constraints.

What if you are getting multiple requests? What if you are using a combination of go routine + channel + next go routine + channel + go routine + channel .... and all of this is dedicated to one request? Maybe just maybe you never have to create an unbuffered channel. Maybe a channel with capacity of zero is a good idea.

Maybe if one request requires to go through a pipeline of multiple tasks ... T1, T2, T3 then the entire pipeline can be constructed using go routines and channels (a series) dedicated to one request. So if there are 500 requests, maybe you have 500 chains.