Garbage collection

CS110L
January 26, 2022

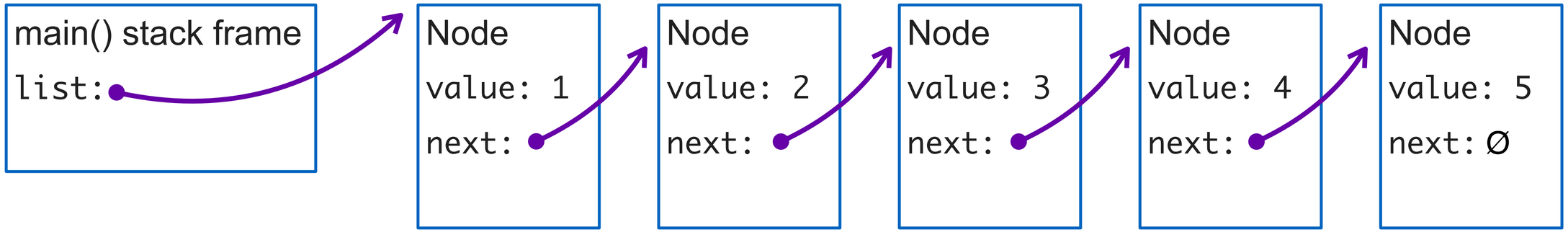
But wait… Why is this *so* much harder than it is in Python?

Garbage collection

Garbage collection

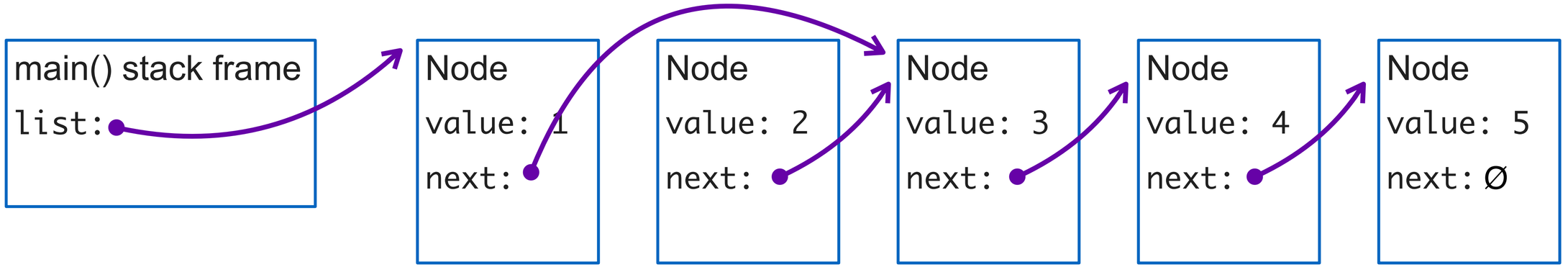
- C/C++ have a problem: When should you free your memory?
  - In complicated codebases, it's very easy to have memory leaks, double frees, or use-after-frees
- Rust: Use a fancy type system to denote who is responsible for freeing memory, and let the compiler check that everything looks right
  - Still difficult to program.
  - You're constantly thinking about who has ownership of what
- Much older approach: Garbage collection
  - When writing your program, don't worry about freeing memory
  - When running your program, the runtime will observe when memory is no longer being used, and will free it for you
  - *Ex: you never have to manually manage your memory in (e.g.) Python!*

Tracing garbage collection



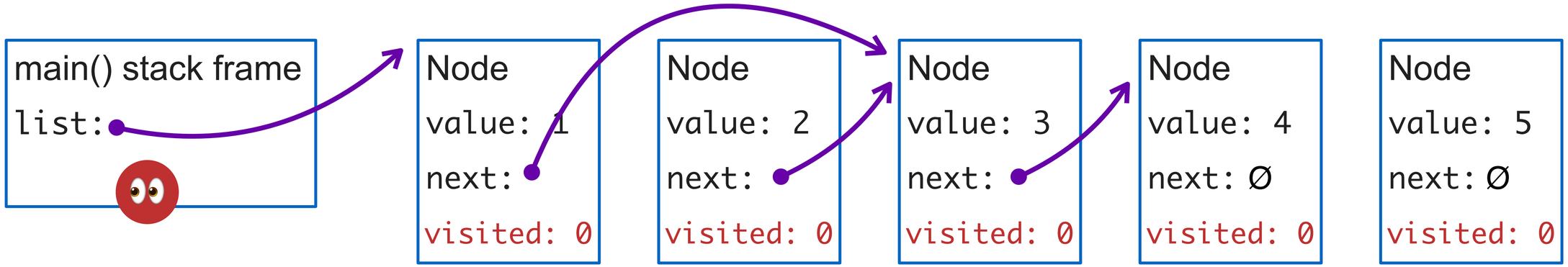
Remove 2nd node

Tracing garbage collection



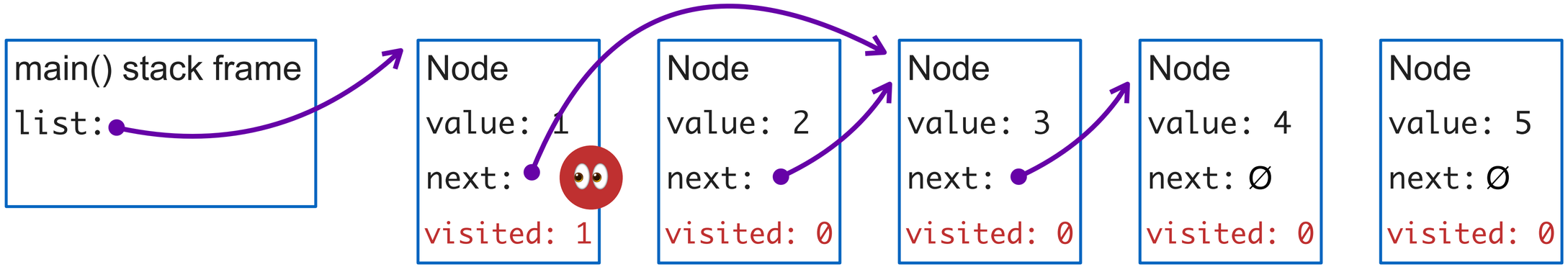
Remove last node

Tracing garbage collection



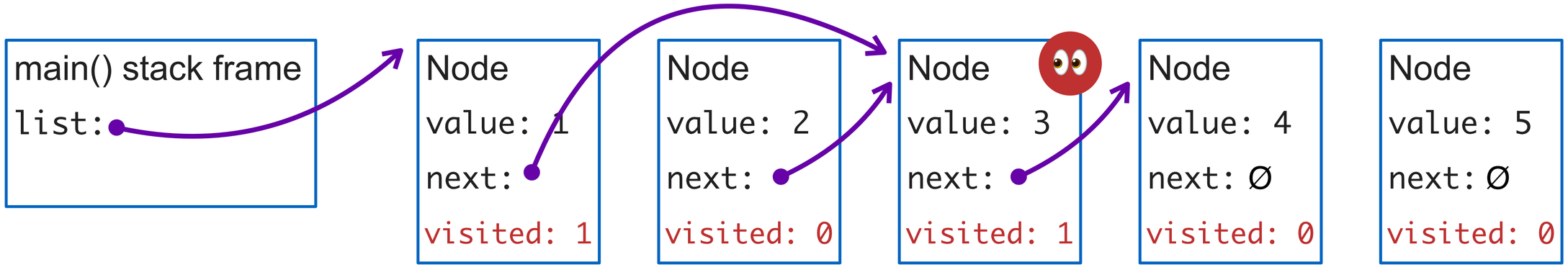
!! Pause execution, begin GC

Tracing garbage collection



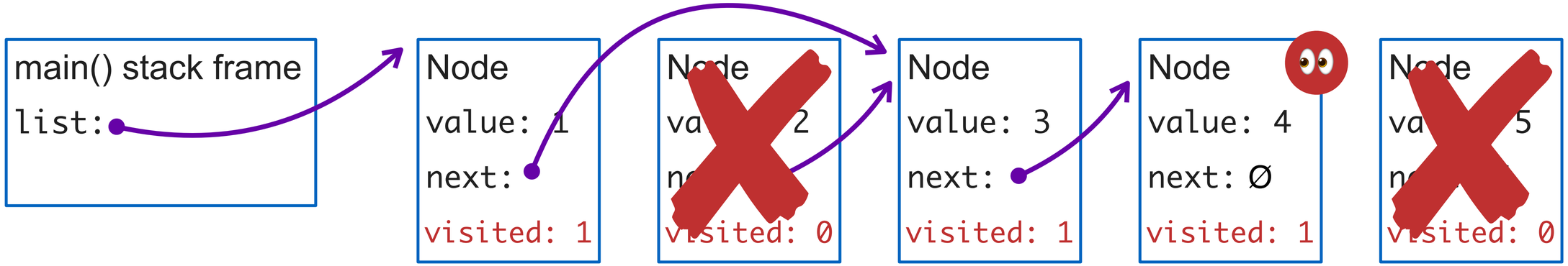
!! Pause execution, begin GC

Tracing garbage collection



!! Pause execution, begin GC

Tracing garbage collection



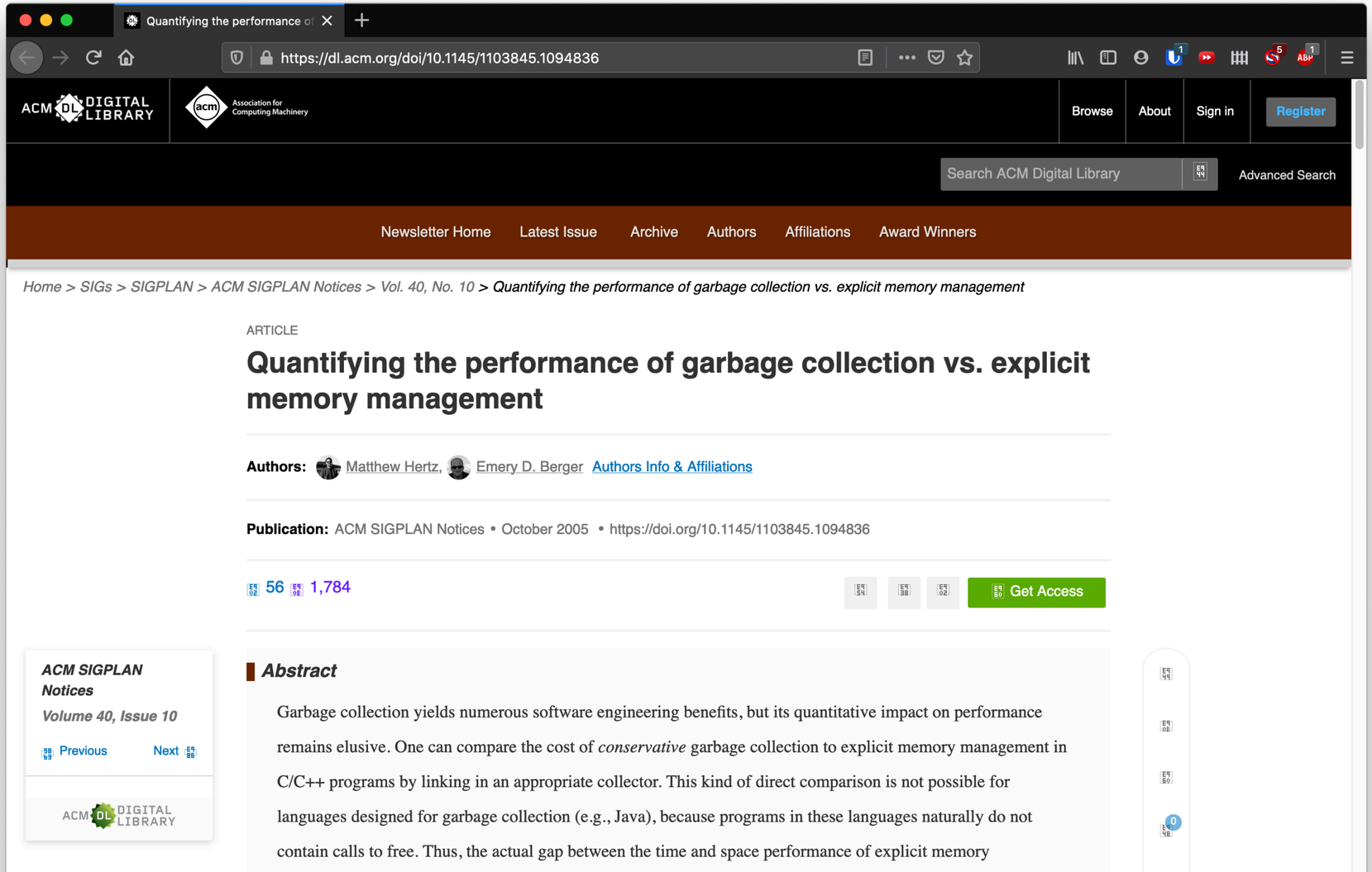
‼ Pause execution, begin GC

Downsides of garbage collection

- Expensive
  - No matter what type of garbage collection is used, there will always be nontrivial memory overhead
- Disruptive
  - Drop what you're doing — it's time for GC!
- Non-deterministic
  - When will the next GC pause be? Who knows! Depends on how much memory is being used
- Precludes manual optimization
  - In some situations, you may want to structure your data in memory in a specific way in order to achieve high cache performance
  - GC can't know how you will use memory, so it optimizes for the average use case

GC is expensive

https://dl.acm.org/doi/10.1145/1103845.1094836



GC is expensive

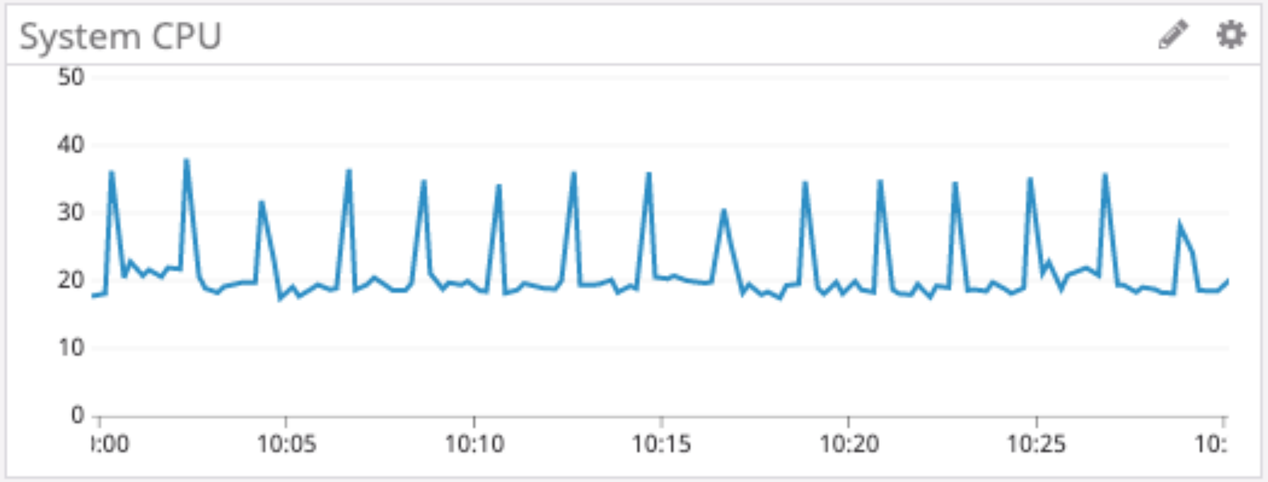
https://dl.acm.org/doi/10.1145/1103845.1094836

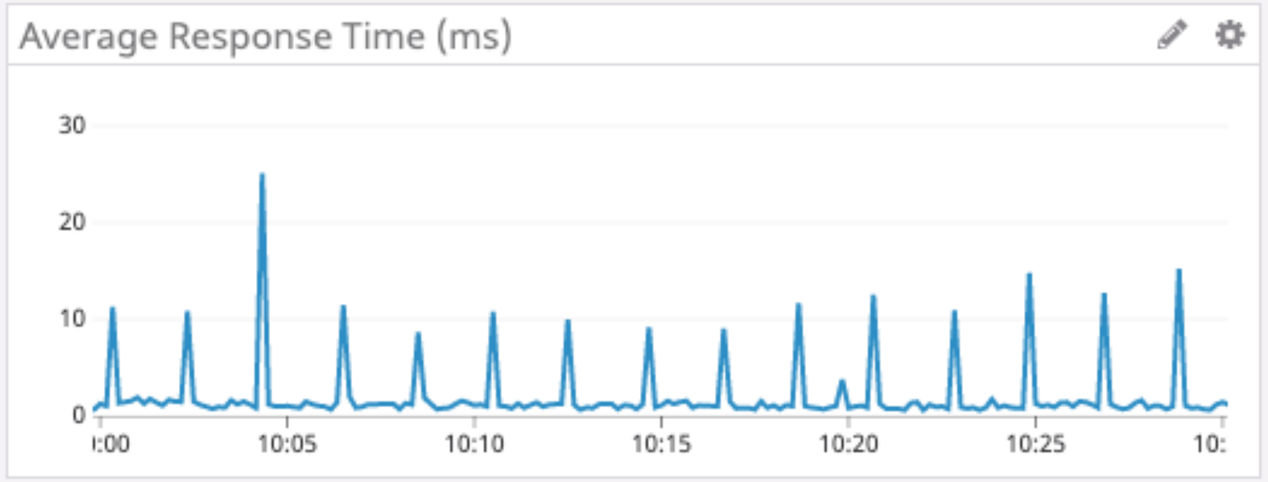
> *With five times as much memory, an Appel-style generational collector with a non-copying mature space matches the performance of reachability-based explicit memory management. With only three times as much memory, the collector runs on average 17% slower than explicit memory management. However, with only twice as much memory, garbage collection degrades performance by nearly 70%. When physical memory is scarce, paging causes garbage collection to run an order of magnitude slower than explicit memory management.*

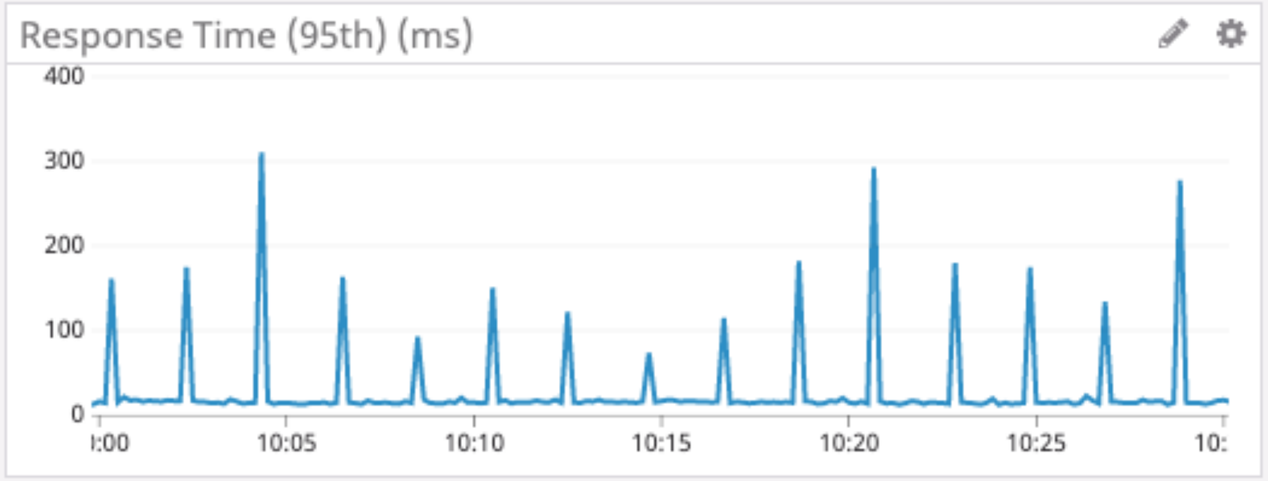
“Quantifying the performance of garbage collection vs. explicit memory management,” Hertz and Berger

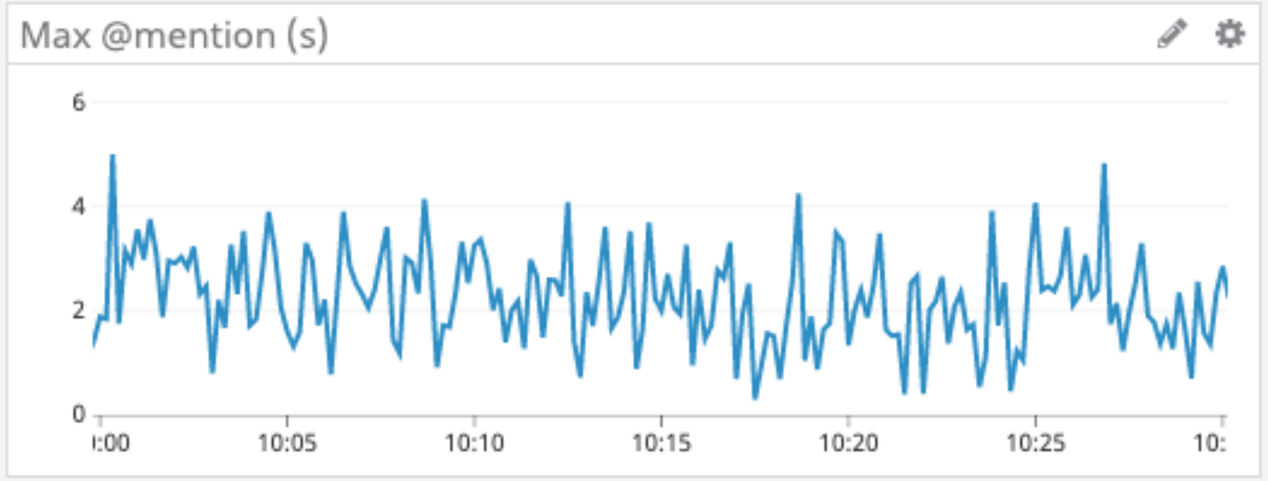
Why Discord is switching from Go to Rust

Jesse Howarth Follow
Feb 4 · 10 min read









Note latency spikes every 2 minutes

LinkedIn Engineering:

“In our production environments, we have seen unexplainable large STW pauses (> 5 seconds) in our mission-critical Java applications.”

https://engineering.linkedin.com/blog/2016/02/eliminating-large-jvm-gc-pauses-caused-by-background-io-traffic

Latency matters

- User interfaces
- Games
- Self-driving cars
- Payment processing
- High frequency trading

Takeaways

- Use GC languages when it makes sense, but know their limits
  - It doesn’t matter how much memory you save if it takes you so long to develop your app that no one uses it
  - You can always rewrite certain components in other languages if efficiency becomes a problem
  - *Side note: GC languages can still lead to resource leaks — file descriptors, database handles, race conditions in multithreaded code, etc.*
- In resource-constrained or latency-sensitive environments, GC may not be a viable option

Where is Rust used? One [not the only!] perspective…

- Memory safety: C and C++ generally suck when it comes to manual memory management.
- GC generally sucks for resource (power, memory) consumption & latency.
- Rust is still doing manual memory management
  - The compiler does a lot of it for you
  - Rust's type system is designed to help us communicate our expectations so that the compiler can validate them
- For applications that need both memory safety AND resource efficiency/low latency, you'll notice people switching to Rust
  - Mozilla, Cloudflare, embedded operating systems, etc.
  - https://en.wikipedia.org/wiki/Rust_(programming_language)#Adoption