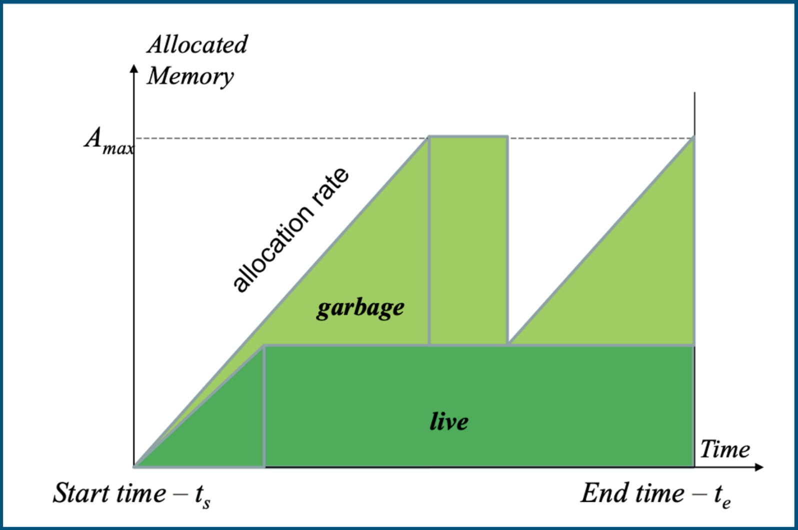
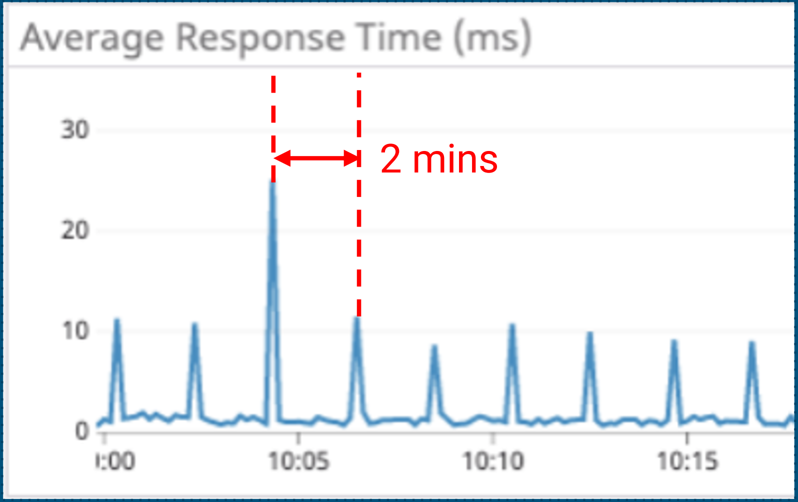
Memory Management:
Go vs Rust

Garbage Collection

- Program allocation
- Mark-and-Sweep
    - Mark live objects, deallocate rest
    - Iterates over entire heap including long-living objects
- Discord
    - Huge cache with tens of millions of objects, takes long time to iterate over
    - Go won't collect garbage more often than every 2 mins



[CS 3214 - Sp 2020 AMM/GC presentation by Dr. Back]



[Discord Blog]

Go Garbage Collection

- Tricolor Mark-and-Sweep
- Runs concurrently with program
- Low latency at the cost of lower throughput
- Non-generational, non-compacting

Rust Memory Management

- Has no Garbage Collector
- Programs have “ownership” over memory via smart pointers
- Immediately frees memory once no longer used or needed

Rust vs Go

Go

- Optimized for low average latency
- Forced 2-minute garbage collection
- Heavy-cost garbage collection = latency spikes
- Size of latency spikes determined by size of the LRU cache
- Lower cache sizes means more misses but smaller latency spikes
- Higher cache sizes means less misses but higher latency spikes

Rust

- No garbage collection = no latency spikes caused by GC
- Constant memory tracking and immediate frees
- Enforcement of memory rules at compile time
- All performance metrics (Latency, memory, CPU) better than Go

Discord - From Go to Rust

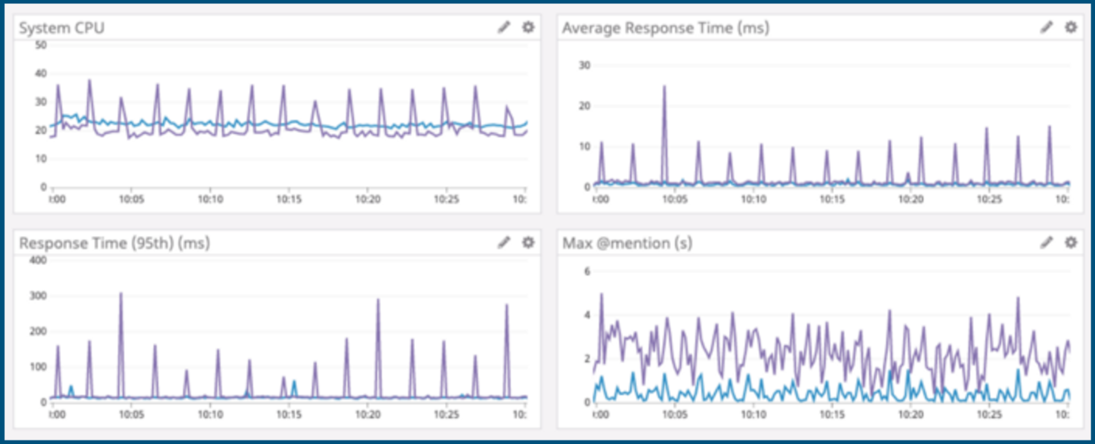
What is Discord?

- IM and VoIP program

Why switch to Rust?

- Optimizes “read states” services
- No overhead
- Higher stability

Purple = Go Blue = Rust



source: Discord's blog

Discussion

References

https://stackoverflow.blog/2020/06/05/why-the-developers-who-use-rust-love-it-so-much/

https://blog.discord.com/why-discord-is-switching-from-go-to-rust-a190bbca2b1f

https://blog.plan99.net/modern-garbage-collection-911ef4f8bd8e